СУБД ЛИНТЕР СОКОЛ

РУКОВОДСТВО

ПОЛЬЗОВАТЕЛЯ

Версия СУБД — 3.3

АО НПП «Реляционные экспертные системы»

Воронеж, 2026 г.

СОДЕРЖАНИЕ

1. ПРЕДИСЛОВИЕ ... 10
1.1. Термины, акронимы, сокращения ... 10
1.2. Что такое СУБД СОКОЛ (общие сведения) ... 14
1.3. Что нового в СУБД ЛИНТЕР СОКОЛ ... 15
1.4. Почему выбирают СУБД ЛИНТЕР СОКОЛ ... 16
2. ВВЕДЕНИЕ ... 18
2.1. Информация о разработчике ... 18
2.2. Сертификаты и лицензирование деятельности ... 19
2.3. Технические требования к оборудованию ... 19
2.4. Технические характеристики СУБД ... 20
2.5. Версии и лицензии СУБД ... 21
2.6. Требования к знаниям пользователя ... 22
3. АРХИТЕКТУРНЫЕ АСПЕКТЫ ... 23
3.1. Влияние неблокирующих подходов на производительность ... 23
3.2. Технологическая платформа ... 24
3.3. Основное хранилище данных ... 26
3.4. Журналирование ... 27
3.5. Конфликты при доступе к данным на странице хранилища ... 28
4. УСТАНОВКА СУБД ЛИНТЕР СОКОЛ ... 29
4.1. Перед установкой ... 29
4.2. Установка СУБД в среде ОС UNIX/Windows ... 29
4.2.1. Изменение максимального числа открытых файловых дескрипторов в ОС Linux ... 30
4.2.2. Изменение максимального числа открытых файловых дескрипторов в ОС Windows ... 31
4.3. Назначение файлов дистрибутива ... 31
4.4. Сведения о сторонних компонентах ... 33
4.4.1. LLVM ... 33
4.4.2. ICU (International Components for Unicode) ... 33
4.4.3. libyaml ... 34
4.4.4. zlib ... 34
4.4.5. unixODBC ... 34
5. ЗАПУСК СЕРВЕРА И НАЧАЛО РАБОТЫ С СУБД ЛИНТЕР СОКОЛ ... 35
5.1. Первоначальный запуск сервера СУБД ... 35
5.2. Создание базы данных ... 36
5.3. Подключение к базе данных из утилиты vsql_console ... 37
5.4. Выполнение первого запроса к базе данных ... 38
5.5. Создание таблицы в базе данных ... 38
5.6. Вставка данных в таблицу базы данных ... 38

5.7. Выборка вставленных данных ... 39
5.8. Изменение данных в таблице БД ... 40
5.9. Удаление данных из таблицы БД ... 40
5.10. Удаление таблицы из базы данных ... 41
5.11. Удаление базы данных ... 41
5.12. Останов сервера ... 42
6. О СТРОКЕ ПОДКЛЮЧЕНИЯ ДЛЯ УТИЛИТ ... 43
7. КОНФИГУРАЦИОННЫЙ ФАЙЛ SOQOL_CONFIG.YML ... 45
8. КАТАЛОГ СЕРВИСА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ AUXDB ... 49
9. ПРОГРАММНЫЕ ИНТЕРФЕЙСЫ ... 50
9.1. ODBC-драйвер ... 50
9.1.1. Установка ODBC-драйвера под ОС Windows ... 50
9.1.2. Установка ODBC-драйвера для ОС UNIX ... 52
9.1.3. Поддерживаемые версии, список реализованных операций в ODBC ... 57
9.2. JDBC-драйвер ... 57
9.2.1. Подключение JDBC-драйвера ... 57
9.2.2. Строка подключения ... 57
9.2.3. Проверка работоспособности ... 59
9.2.4. Логирование в JDBC-драйвере ... 60
9.2.5. Поддерживаемые версии Java и список реализованных функций в JDBC ... 61
9.2.6. При возникновении проблем с подключением JDBC-драйвера ... 61
9.3. PHP-драйвер ... 62
9.3.1. Установка ... 62
9.3.2. Использование ... 63
9.3.2.1. Класс PDO ... 63
9.3.2.2. Формат строки подключения (DSN) ... 63
10. ВИЗУАЛЬНЫЕ ИНСТРУМЕНТЫ ДЛЯ РАБОТЫ С СУБД СОКОЛ ... 65
10.1. DBeaver ... 65
10.2. Другие инструменты ... 71
11. РАБОТА С СУЩНОСТЬЮ БД ... 72
11.1. CREATE DATABASE ... 72
11.2. STARTUP DATABASE ... 78
11.3. ALTER DATABASE ... 79
11.3.1. ALTER DATABASE KILL SESSION ... 80
11.4. ALTER SERVICE ... 81
11.5. ALTER SESSION ... 84
11.6. SHUTDOWN DATABASE ... 86
11.7. DETACH DATABASE ... 87

11.8. ATTACH DATABASE ... 88
11.9. DROP DATABASE ... 89
11.10. SHUTDOWN SERVICE ... 90
11.11. RESTART SERVICE[CV] ... 91
12. РУКОВОДСТВО ПО SQL ... 93
12.1. Типы данных ... 93
12.1.1. Часовые пояса[CV] ... 96
12.2. Литералы ... 97
12.2.1. Строковый литерал ... 97
12.2.2. Числовой литерал ... 98
12.2.3. Литерал даты и времени ... 98
12.2.4. Логический литерал ... 101
12.3. Операции ... 102
12.3.1. Арифметические операции ... 103
12.3.2. Строковые операции ... 103
12.3.3. Логические операции ... 104
12.3.4. Операции сравнения ... 104
12.3.5. Свод допустимости преобразований типов данных ... 106
12.4. Выражения ... 107
12.5. Объекты базы данных в СУБД СОКОЛ и правила их именования ... 108
12.5.1. Таблица (TABLE) ... 108
12.5.2. Представление (VIEW) ... 110
12.5.3. Генератор последовательности (SEQUENCE) ... 111
12.5.4. Индекс (INDEX) ... 112
12.5.5. Схема (SCHEMA) ... 115
12.5.6. Пользователь (USER) ... 116
12.5.7. Привилегии (PRIV) и роли (ROLE) ... 117
12.5.8. Хранимая процедура (PROCEDURE) ... 120
12.5.9. Псевдостолбец ROWID ... 120
12.5.10. Правила именования объектов БД и обращения к ним ... 121
12.6. Работа с транзакциями ... 123
12.7. План выполнения запроса (EXPLAIN) ... 127
12.8. Справочник команд SQL ... 131
12.8.1. CREATE ... 132
12.8.1.1. CREATE TABLE ... 132
12.8.1.2. CREATE VIEW ... 150
12.8.1.3. CREATE INDEX ... 153
12.8.1.4. CREATE SEQUENCE ... 157

12.8.1.5. CREATE SCHEMA 163
12.8.1.6. CREATE USER 164
12.8.1.7. CREATE ROLE 167
12.8.1.8. CREATE PROCEDURE 168
12.8.2. DROP 169
12.8.2.1. DROP TABLE 169
12.8.2.2. DROP INDEX 171
12.8.2.3. DROP SEQUENCE 172
12.8.2.4. DROP PROCEDURE 173
12.8.2.5. DROP VIEW 173
12.8.2.6. DROP USER 174
12.8.2.7. DROP ROLE 177
12.8.2.8. DROP SCHEMA 178
12.8.3. ALTER 180
12.8.3.1. ALTER SCHEMA 180
12.8.3.2. ALTER USER SCHEMA 181
12.8.3.3. ALTER USER 182
12.8.3.4. ALTER TABLE[CV] 183
12.8.3.5. ALTER INDEX[CV] 187
12.8.4. TRUNCATE TABLE 188
12.8.5. SELECT 189
12.8.5.1. Общий синтаксис команды SELECT 190
12.8.5.2. <конструкция_FROM> 192
12.8.5.3. <конструкция_ORDER_BY> 195
12.8.5.4. <ограничение_числа_строк_выборки> 196
12.8.5.5. <блокировка_строк> 196
12.8.5.6. Примеры применения команды SELECT 197
12.8.6. INSERT INTO TABLE 212
12.8.6.1. INSERT INTO TABLE с добавлением в таблицу заданных значений 213
12.8.6.2. INSERT INTO TABLE с возможностью добавления в таблицу результата SELECT-запроса 215
12.8.7. UPDATE TABLE 217
12.8.8. DELETE 219
12.8.9. Конструкция RETURNING 221
12.8.10. EXPLAIN 225
12.8.11. CALL 228
12.8.12. Команды работы с транзакциями 229
12.8.12.1. SET TRANSACTION ISOLATION LEVEL 229
12.8.12.2. START TRANSACTION 230

12.8.12.3. Точка сохранения ... 233
12.8.12.4. ROLLBACK ... 238
12.8.12.5. COMMIT ... 241
12.8.13. GRANT ... 243
12.8.13.1. GRANT PRIV ... 243
12.8.13.2. GRANT ROLE ... 246
12.8.14. REVOKE ... 248
12.8.14.1. REVOKE PRIV ... 248
12.8.14.2. REVOKE ROLE ... 251
12.8.15. SET SCHEMA ... 253
12.8.16. PRESS[CV] ... 255
12.8.16.1. PRESS TABLE ... 256
12.8.16.2. PRESS ALL TABLES ... 256
12.8.17. ANALYZE TABLE[CV] ... 256
12.8.18. WITH (Common Table Expressions)[CV] ... 257
12.9. Функции SQL ... 258
12.9.1. Агрегатные функции ... 258
12.9.1.1. AVG ... 259
12.9.1.2. COUNT ... 260
12.9.1.3. MAX ... 262
12.9.1.4. MIN ... 264
12.9.1.5. STRING_AGG ... 264
12.9.1.6. SUM ... 266
12.9.2. Скалярные функции ... 267
12.9.2.1. ABS ... 267
12.9.2.2. BITAND ... 268
12.9.2.3. CAST ... 269
12.9.2.4. CASE[CV] ... 270
12.9.2.5. COALESCE ... 273
12.9.2.6. COLLATION ... 274
12.9.2.7. CONCAT ... 274
12.9.2.8. CONCAT_BLOB[CV] ... 276
12.9.2.9. CURRENT_DATE ... 277
12.9.2.10. CURRENT_DBNAME ... 277
12.9.2.11. CURRENT_ISOLATION_LEVEL ... 277
12.9.2.12. CURRENT_SCHEMA ... 278
12.9.2.13. CURRENT_TIMESTAMP ... 278
12.9.2.14. CURRENT_USER ... 279

12.9.2.15. DBTIMEZONE[CV] 279
12.9.2.16. DECODE 280
12.9.2.17. EXTRACT[CV] 281
12.9.2.18. HEX 283
12.9.2.19. INSTR[CV] 283
12.9.2.20. LENGTH 286
12.9.2.21. LENGTHB 287
12.9.2.22. LOCALTIMESTAMP[CV] 287
12.9.2.23. LOWER 287
12.9.2.24. LTRIM 288
12.9.2.25. MOD 289
12.9.2.26. NULLIF 289
12.9.2.27. NVL 290
12.9.2.28. RANDOM_VALUE 291
12.9.2.29. RAND_INT 292
12.9.2.30. REPLACE 293
12.9.2.31. ROUND 294
12.9.2.32. ROWNUM[CV] 296
12.9.2.33. RTRIM 298
12.9.2.34. SESSION_ID[CV] 299
12.9.2.35. SESSIONTIMEZONE[CV] 299
12.9.2.36. SUBSTR 300
12.9.2.37. TIMESTAMP_ADD[CV] 302
12.9.2.38. TIMESTAMP_FROM_PARTS 304
12.9.2.39. TIMESTAMP_DIFF[CV] 305
12.9.2.40. TIMESTAMP_TRUNC[CV] 307
12.9.2.41. TO_CHAR 308
12.9.2.42. TO_DATE[CV] 310
12.9.2.43. TO_TIMESTAMP[CV] 313
12.9.2.44. TRANSACTION_ID[CV] 316
12.9.2.45. TRIM[CV] 316
12.9.2.46. TRUNC 317
12.9.2.47. UPPER 318
12.9.2.48. VERSION 319
13. ИДЕНТИФИКАЦИЯ ВЕРСИИ СЕРВЕРА 320
14. ОБЪЕКТЫ, ФОРМИРУЕМЫЕ СИСТЕМОЙ ПРИ СОЗДАНИИ БД В СУБД СОКОЛ 321
14.1. Предустановленные пользователи 322
14.2. Предустановленные роли 322

14.3. Предустановленные схемы 322
14.3.1. Схема INFO 322
14.3.2. Схема SYS 323
14.3.3. Схема PUBLIC 323
15. АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ В СУБД СОКОЛ 324
15.1. Копирование / архивирование базы данных и её последующее восстановление 324
15.2. Состав каталога БД 325
16. ПРОЦЕДУРНЫЙ ЯЗЫК PL/SOQOL 327
16.1. Лексические единицы 327
16.1.1. Идентификаторы 327
16.1.2. Зарезервированные слова 329
16.1.3. Литералы 329
16.1.4. Комментарии 329
16.2. Структура процедурного языка 330
16.2.1. Анонимный блок 331
16.2.1.1. Операторы 334
16.2.2. Именованный блок 338
16.3. Выражения 339
16.3.1. Операция конкатенации 339
16.3.2. Логические операции 339
16.3.3. Операции сравнения 339
16.3.4. Приоритет операций 339
16.3.5. Функции SQL в выражениях 340
16.4. Типы данных процедурного языка 340
16.5. Пример использования процедурного языка в СУБД СОКОЛ 340
17. УТИЛИТЫ[CV] 345
17.1. vsql_server_wait 345
17.2. vsql_console 346
17.3. vsql_copy 347
18. КОМАНДЫ КОНСОЛЬНОЙ УТИЛИТЫ VSQL_CONSOLE 350
18.1. Команды подключения 350
18.1.1. Подключение к базе данных 350
18.1.2. Подключение к сервису управления БД 351
18.1.3. Подключение через URL к БД и к сервису управления БД 351
18.2. Выход из утилиты 352
18.3. Объявление переменной 352
18.4. Вывод в консоль значения из переменной 354
19. КОДЫ ОШИБОК СУБД[CV] 355

ПРИЛОЖЕНИЕ 1 356
ПРИЛОЖЕНИЕ 2 357
ПРИЛОЖЕНИЕ 3 366
ПРИЛОЖЕНИЕ 4 367
ПРИЛОЖЕНИЕ 5 372

1. ПРЕДИСЛОВИЕ

Данное руководство обеспечит пользователя необходимой информацией для самостоятельной работы с СУБД ЛИНТЕР СОКОЛ. Из руководства вы узнаете:

– об особенностях СУБД, её преимуществах и возможностях;

– об уникальной архитектуре СУБД, позволяющей максимально быстро выполнять обработку данных;

– об интерфейсах системы;

– об используемом в системе процедурном языке;

– о поддерживаемом диалекте языка SQL;

– о дополнительных инструментах обработки данных;

– ответы на частые вопросы и другие важные моменты для эффективной работы с СУБД.

При возникновении вопросов, которые не отражены в данном документе, обращайтесь в службу технической поддержки в телеграм-бот https://t.me/soqol_sd_bot. Сотрудники службы оперативно помогут справиться с проблемой или подобрать оптимальные пути решения ваших задач.

Если вам необходима доработка СУБД ЛИНТЕР СОКОЛ по индивидуальному запросу, оставьте заявку в службу технической поддержки, мы постараемся вам помочь.

Вернуться в оглавление

1.1. Термины, акронимы, сокращения

1) CV — метка, обозначающая, что указанный элемент документации появился в текущей версии.

2) DCL (data control language) — группа операторов SQL для определения доступа к данным: разрешение или запрещение выполнения определенных операций над объектами базы данных.

3) DDL (data description language) — группа операторов SQL для создания и изменения объектов базы данных, например, таблиц, индексов, пользователей.

4) DML (data manipulation language) — группа операторов SQL для манипулирования данными: выборка данных, добавление новых данных, обновление и удаление данных.

5) JDBC (Java Database Connectivity) — прикладной программный интерфейс (API) доступа к базам данных из приложений на языке программирования Java на основе SQL. JDBC обеспечивает унификацию работы с различными СУБД для Java-приложений.

6) JDBC-драйвер — программа, в которой реализован JDBC API с учётом особенностей конкретной СУБД. Java-приложение посылает запросы к СУБД через ее JDBC-драйвер.

7) MVCC (Multiversion Concurrency Control запись) - концепция, используемая в базах данных для управления параллельным доступом к данным и обеспечения согласованности в транзакционных системах. При параллельно выполняемых транзакциях, MVCC-запись позволяет каждой транзакции видеть свою "версию" данных и избегать блокировки других транзакций. Вместо блокировки записи, MVCC-запись создает несколько версий записей для каждой транзакции. Каждая версия MVCC-записи содержит уникальный идентификатор, временную метку и значения полей данных.

8) PV — метка, обозначающая, что указанный элемент документации не актуален для текущей версии, однако был актуален для предыдущей версии.

9) ODBC (Open Database Connectivity) — прикладной программный интерфейс (API) доступа к базам данных из приложений на основе SQL. ODBC обеспечивает унификацию работы с различными СУБД. Поддерживается для различных языков программирования.

10) ODBC-драйвер — программа, в которой реализован ODBC API с учётом особенностей конкретной СУБД. Приложение посылает запросы к СУБД через ее ODBC-драйвер.

11) OLAP (online analytical processing, интерактивная аналитическая обработка) — технология обработки данных, заключающаяся в подготовке

суммарной (агрегированной) информации на основе больших массивов данных, структурированных по многомерному принципу.

12) OLTP (Online Transaction Processing), транзакционная система — обработка транзакций в реальном времени. Способ организации БД, при котором система работает с небольшими по размерам транзакциями, но идущими большим потоком, и при этом клиенту требуется от системы минимальное время отклика.

13) PATH (в командном интерпретаторе операционной системы) — переменная окружения Unix-подобных операционных систем, DOS, OS/2 и Microsoft Windows, представляющая собой список каталогов, в которых расположены исполняемые файлы.

14) SMO (structure management operation) — операции в B-Tree, которые не ассоциируется напрямую с операциями над данными пользователя, но необходимы для поддержания эффективной структуры.

15) SQL — язык структурированных запросов, декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

16) URL — система унифицированных адресов электронных ресурсов. Каждый корректный URL ведёт на уникальный ресурс.

17) UTC — стандарт всемирного координированного времени.

18) Абсолютный путь (в файловой системе) — полный путь от корневого каталога сервера (для Windows - от корня диска), вне зависимости от текущего (рабочего) каталога.

19) БД — база данных.

20) Бенчмарк TPC-C (Transaction Processing Performance Council Benchmark C) — тест, используемый для сравнения производительности систем онлайн-обработки транзакций (OLTP). Для его прохождения в базе данных генерируется набор данных, характерный для бизнес-систем.

21) Владелец объекта базы данных — пользователь, владеющий схемой, в которой находится объект, и обладающий возможностью совершать действия с этим объектом в рамках доступных владельцу привилегий.

22) Диспетчер драйверов (для Windows) — библиотека динамической компоновки (DLL), которая управляет взаимодействием между приложениями и драйверами.

23) АО НПП «РЕЛЭКС» — научно-производственное предприятие «Реляционные экспертные системы» (далее РЕЛЭКС).

24) ИС — информационная система.

25) Консоль (интерфейс командной строки) — окно операционной системы, в котором пользователь взаимодействует с операционной системой при помощи команд. Выходные сообщения от операционной системы отображаются в этом же окне.

26) Литерал — способ записи конкретного значения одного из типов данных.

27) ОП — оперативная память.

28) ОС — операционная система.

29) Относительный путь (в файловой системе) — путь по отношению к текущему (рабочему) каталогу пользователя.

30) ПО — программное обеспечение.

31) Первичный ключ (primary key) — в реляционной модели данных это основной ключ отношений (или ключ по умолчанию), обладающий свойствами уникальности (сохраняя её со временем) и минимальности (имеет наименьший размер).

32) Привилегия — право пользователя совершать определённые действия по отношению к БД и её объектам.

33) РБД — реляционная база данных.

34) Рабочая станция — персональный компьютер, включенный в состав сети, на котором запускается сервер и/или клиент (по документу см. в зависимости от контекста).

35) СУБД — система управления базами данных.

36) СУБД ЛИНТЕР СОКОЛ — российская система управления базами данных нового поколения ЛИНТЕР СОКОЛ (далее СУБД ЛИНТЕР СОКОЛ, СУБД СОКОЛ).

37) Сервис управления базами данных (далее — сервис, сервис управления) — сервис приложения-сервера, доступный по специальному URL и обеспечивающий работу с базами данных, находящимися под его контролем (создание, удаление и управление).

38) Текущая база данных — для пользователя это база данных, к которой он в настоящий момент подключён.

39) Текущая версия СУБД СОКОЛ — версия СУБД, для которой написана данная документация.

40) Транзакция — это логически неделимая последовательность изменений базы данных. Транзакция может быть выполнена либо целиком и успешно, либо не выполнена вообще.

41) ЦПУ (CPU) — центральное процессорное устройство.

42) Ядро — ядро СУБД.

Вернуться в оглавление

1.2. Что такое СУБД СОКОЛ (общие сведения)

Это стопроцентно российская инновационная система, построенная на современных алгоритмах обработки больших объёмов данных. Архитектура реляционной СУБД ЛИНТЕР СОКОЛ нового поколения отличается от ранее известных. Она основана на синтезе современных неблокирующих подходов обработки данных в памяти и работе с данными на внешних носителях, эффективном использовании кэша и адаптивности системы исполнения запросов.

Несмотря на свою современность, СУБД СОКОЛ поддерживает хорошо знакомый пользователям стандарт ANSI SQL. Первые версии СУБД СОКОЛ создаются с опорой на стандарт ANSI SQL:2016.

В новой СУБД заложена кроссплатформенность и поддержка российских процессоров. При разработке СУБД СОКОЛ использовались собственные запатентованные методы обработки данных.

В системах управления базами данных компании РЕЛЭКС всегда большое внимание уделяется защите информации. СУБД СОКОЛ не исключение:

– реализована компанией с «нуля»; ядро СУБД и основные алгоритмы обработки данных разработаны самостоятельно и не основаны на существующих системах управления базами данных с открытым исходным кодом (продукт не является производным от PostgreSQL или иных СУБД);

– для выполнения отдельных функций (JIT-компиляция запросов, интернационализация, работа с конфигурационными файлами) в составе СУБД используются статически компонуемые библиотеки с открытым исходным кодом; при этом работа СУБД возможна без использования JIT-компиляции;

– является системой с закрытым кодом, что повышает её защиту от внешнего исследования с целью взлома;

– реализована авторизация пользователей и дискреционная защита данных.

Вернуться в оглавление

1.3. Что нового в СУБД ЛИНТЕР СОКОЛ

СУБД ЛИНТЕР СОКОЛ представляет собой концентрат технологий, которые значительно повышают эффективность обработки данных:

– дата-центричный подход формирования исполняемого кода вместо операторо-центричного. Дата-центричный подход представляет собой систему, в которой данные составляют основной и неотъемлемый ресурс, а способы их использования меняются;

– неблокирующие алгоритмы (Distibuted Epoch Based Reclamation, Multiple CAS, multiple producers & single consumer queue);

– методика комбинирования неблокирующих подходов с классической синхронизацией;

– нативная компиляция кода исполнения процедур и SQL-запросов;

– адаптивность системы исполнения запросов, заключающаяся в переключении между интерпретируемым и нативным кодом;

– обеспечивающая легковесное переключение с одного процесса на другой кооперативная многозадачность, управляемая собственным менеджером процессов, позволяющим оптимально распределять ресурсы;

– масштабирование на современном высокопроизводительном оборудовании и управление потоками ввода-вывода. Внедрение в архитектуру СУБД собственного планировщика ввода-вывода со своим оригинальным алгоритмом позволяет достигать лучшую суммарную производительность всей системы.

Применение в СУБД СОКОЛ лучших технологий позволяет решать задачи класса High Load и High Available. СУБД перерабатывает с высокой скоростью OLTP запросы, что позволяет ей не только обрабатывать большое число бизнес-транзакций за минимальное время, но и эффективно работать с профилем нагрузки OLAP.

Вернуться в оглавление

1.4. Почему выбирают СУБД ЛИНТЕР СОКОЛ

Высокопроизводительная

Это реляционная СУБД нового поколения, использующая быстродействие in-memory решений. Сравнительное тестирование по методике TPC-C доказывает многократное превосходство в скорости обработки данных перед аналогичными системами.

Удобная

СУБД СОКОЛ — это:

– реляционная модель организации данных, с доступом к базе данных посредством знакомого SQL и привычными интерфейсами;

– возможность прямого взаимодействия с российской компанией-разработчиком СУБД;

– техническая поддержка на родном языке;

– возможность физического присутствия технического специалиста из команды разработки СУБД в регионе;

– возможность влияния на функционал СУБД в момент её разработки;

– высокая скорость обработки больших массивов данных обеспечивает пользователю наибольший комфорт.

Экономичная

Архитектура СУБД, позволяющая работать эффективнее конкурентов, даёт экономию средств, затрачиваемых на приобретение и поддержание оборудования.

Быстрый отклик системы, при обработке данных ежедневно, даёт значительную экономию рабочего времени.

Надёжная

– полностью отечественная разработка без заимствованного или модифицированного программного кода из какой-либо существующей системы;

– имеет закрытый код, что максимально снижает риски внешнего исследования системы с целью взлома;

– разрабатывается с учётом требований защиты персональных данных при их обработке в ИС;

– СУБД СОКОЛ имеет встроенные средства защиты информации от несанкционированного доступа;

– в ближайших релизах планируется сертификация СУБД на соответствие требованиям ФСТЭК и Министерства Обороны России.

Вернуться в оглавление

2. ВВЕДЕНИЕ

2.1. Информация о разработчике

Научно-производственное предприятие «Реляционные экспертные системы» (РЕЛЭКС) — российская IT-компания, занимающаяся разработкой программного обеспечения. Компания основана в 1990 году и является в России независимым разработчиком собственных систем управления базами данных.

РЕЛЭКС развивает собственную школу СУБД-строения. Здесь молодые специалисты получают теоретические и практические знания по основам разработки систем управления базами данных. Лучшие приглашаются для дальнейшего сотрудничества.

РЕЛЭКС сотрудничает с вузами России и бесплатно предоставляет свои СУБД в учебных целях.

РЕЛЭКС — это сотни успешно реализованных проектов, профессиональная команда разработчиков, индивидуальный подход к каждому заказчику и каждому проекту, открытость и прозрачность процесса разработки.

РЕЛЭКС — это десятки партнерских соглашений с компаниями по всему миру, работающими в разных отраслях.

Среди решений РЕЛЭКС:

– системы хранения данных и управления данными;

– низкоуровневое программное обеспечение и драйверы;

– информационно-аналитические системы;

– веб-ориентированные портальные решения;

– системы управления проектами;

– мобильные приложения;

– системы дистанционного обучения и тестирования;

– специализированные наукоёмкие решения.

Продукты РЕЛЭКС используются в тысячах программно-аппаратных комплексов, управляющих добычей нефти и полезных ископаемых, безопасностью воздушного движения, атомными реакторами и деятельностью предприятий.

Контактные данные АО НПП «РЕЛЭКС»:

– 394006, Россия, г. Воронеж, ул. Бахметьева, 2Б;

– тел./факс: (473) 2-711-711, 2-778-333, +7 (495) 660-24-50;

– сайт https://relex.ru.

Вернуться в оглавление

2.2. Сертификаты и лицензирование деятельности

Право на разработку и выпуск защищённого программного обеспечения подтверждено действующей лицензией Министерства обороны РФ №1689 от 25.10.2018 года на деятельность в области создания средств защиты информации.

Более подробную информацию можно получить на сайтах:

– компании РЕЛЭКС: https:/www.relex.ru;

– СУБД СОКОЛ: https://www.soqol.ru/.

Вернуться в оглавление

2.3. Технические требования к оборудованию

СУБД ЛИНТЕР СОКОЛ является кросс-платформенной системой, эффективно использует возможности многоядерных систем.

Текущая версия СУБД СОКОЛ работает в среде следующих ОС:

– Windows 10;

– Linux x86-64 (дистрибутивы, бинарно совместимые с Red Hat8: Ubuntu 18.04 LTS, RHEL 8.0, Debian 10, Alt linux 10.0, Astralinux 1.7).

Минимальные требования:

– 512 МБ оперативной памяти для ядра СУБД;

– 64-битный процессор с поддержкой SSE4.1.

Есть возможность разработки специальной версии СУБД СОКОЛ под необходимые параметры оборудования. Для этого необходимо оставить заявку в техническую службу компании РЕЛЭКС в телеграмм-бот https://t.me/soqol_sd_bot.

Вернуться в оглавление

2.4. Технические характеристики СУБД

Для текущей версии СУБД СОКОЛ технические характеристики указаны далее в таблице 1.

В следующих версиях СУБД технические характеристики могут быть изменены в рамках её развития или по требованию пользователя.

Таблица 1 — Технические характеристики СУБД ЛИНТЕР СОКОЛ

Наименование характеристики	**Значение**
Максимальная длина поля (кроме BLOB/CLOB полей)	до 8000 байт
Размер BLOB/CLOB-значения	2^64 байт
Максимальная длина ключа	до 2000 байт *P.S. В будущих версиях СУБД значение будет меняться в большую сторону*
Максимальная длина имени таблицы и поля	до 64 байт
Число столбцов в таблице	до 2000
Максимальное число строк в таблице	ограничено общим размером таблицы (2^64)
Максимальное число индексов в таблице	явных ограничений нет
Максимальное число таблиц в запросе (на одном уровне)	ограничено памятью, доступной при обработки запроса
Максимальное число всех объектов БД	до 2^48
Максимальное число одновременно открытых операторов в сессии	128 *P.S. В будущих версиях СУБД значение может иметь большее значение и настраиваться*
Минимальный объем памяти, занимаемой ядром СУБД	512 Мб
Типы данных	CHAR (size) VARCHAR (size) VARCHAR2 (size) (синоним типа VARCHAR (size))

Наименование характеристики	Значение
	BOOLEAN NUMBER [(precision, scale)] NUMERIC [(precision, scale)] (синоним типа NUMBER [(precision, scale)]) DECIMAL [(precision, scale)] (синоним типа NUMBER [(precision, scale)]) SMALLINT (синоним типа NUMBER (38, 0)) INTEGER (синоним типа NUMBER (38, 0)) INT (синоним типа NUMBER (38, 0)) BIGINT (синоним типа NUMBER (38, 0)) FLOAT [(p)] DOUBLE PRECISION REAL DATE DATETIME (синоним типа DATE) TIMESTAMP BLOB CLOB BINARY (size) VARBINARY (size) RAW (size) (синоним типа VARBINARY (size)) ROWID
Администрирование	Утилиты для Windows и UNIX: - для исполнения запросов интерактивно из консоли или пакетом из файла; - для ожидания готовности сервиса после запуска; - для импорта и экспорта данных из/в CSV
Поддержка средств интернационализации	Поддержка объектов CHARACTER SET и TRANSLATION стандарта SQL. Поддержка кириллических кодировок (CP1251), многобайтовых кодировок (UTF-8). Поддержка UNICODE (для всех версий). Кодировка данных объектов БД задаётся на этапе её создания и не доступна для изменения.
Целевой стандарт языка SQL	Система разработана с опорой на стандарт ANSI SQL:2016
Процедурные расширения языка SQL	Собственный синтаксис языка хранимых процедур, близкий к PL/SQL

Вернуться в оглавление

2.5. Версии и лицензии СУБД

В настоящий момент доступны следующие лицензии СУБД СОКОЛ:

– демонстрационная — не предполагает какого-либо коммерческого использования системы и предназначена исключительно для ознакомления с возможностями системы и её функциональностью. Основные ограничения демонстрационной версии изложены в соответствующем Лицензионном соглашении, которое включено в дистрибутив;

– учебная — не предполагает коммерческого использования и предназначена для учебных целей, предоставляется учебным заведениям;

– коммерческая — предназначена для коммерческого использования.

Дополнительно ознакомиться с условиями использования версий СУБД СОКОЛ и задать вопросы можно:

– на официальном сайте https://www.soqol.ru/;

– по телефону +7 (495) 660-24-50.

Вернуться в оглавление

2.6. Требования к знаниям пользователя

Текущая версия СУБД ЛИНТЕР СОКОЛ предполагает её установку и запуск в консольном режиме ОС. Поэтому важно обладать знаниями:

– о том, как запускать консоль ОС;

– о стандартных консольных командах в ОС;

– что такое абсолютные и относительные пути в файловой системе;

– как запускать программы (с указанием пути к ним или настройки переменной окружения PATH).

Далее в документации положение запускаемых программ будет указано относительно корневого каталога дистрибутива. В случае необходимости указания абсолютного пути пользователь должен внести коррективы самостоятельно.

Вернуться в оглавление

3. АРХИТЕКТУРНЫЕ АСПЕКТЫ

СУБД ЛИНТЕР СОКОЛ является классической дисковой СУБД с точки зрения пользователя — система основана на дисковом хранилище вида B-Tree и WAL-журнале (Write-Ahead Logging).

Архитектура СУБД СОКОЛ поддерживает от самого нижнего до самого верхнего уровня неблокирующий подход (использование неблокирующих алгоритмов) в работе с данными. Это обеспечивает высокую эффективность работы системы при конкурентной обработке данных. «Неблокируемость» в архитектуре не стоит путать с блокировками на пользовательском, логическом уровне работы с данными — здесь работают привычные пользователю подходы.

Архитектура СУБД СОКОЛ ориентирована на генерацию и исполнение как IR (Intermediate Representation) кода, так и нативного кода. Генерация кода выполняется в датацентричном подходе, что повышает эффективность исполнения кода и аппаратного кеширования данных.

Вернуться в оглавление

3.1. Влияние неблокирующих подходов на производительность

В высококонкурентной обработке данных на передний план выходит проблема разрешения конфликтов.

Классический подход в разрешении конфликтов — использование концепции взаимного исключения: конкурентная часть кода размещается в критических секциях, которые обеспечивают эксклюзивное исполнение только в одном потоке, остальные конкурирующие потоки будут в этот момент заблокированы. Такой простой и удобный подход неэффективен на многоядерном оборудовании.

Неблокирующий подход исключает использование критических секций с концепцией взаимного исключения. Такой подход не отрицает саму возможность конфликта, но уровень конфликта переносится с уровня целых структур на отдельные действия внутри этих структур, где разрешается на основе атомарных операций ЦПУ,

т.е. значительно минимизируется. Как следствие, параллельность исполнения значительно возрастает относительно блокирующих подходов.

Сложность внедрения неблокирующего подхода заключается в том, что такие алгоритмы работы с данными имеют свои требования к «среде» своего исполнения и классические наработки алгоритмов, созданные для блокирующих подходов, нельзя тривиальным способом трансформировать в неблокирующие. Возникает потребность в целой технологической платформе для их исполнения.

Для преодоления формировавшейся десятилетиями программной практики в области классических баз данных, основанной на блокирующих подходах, архитектура СУБД СОКОЛ и весь код проекта написан с чистого листа. В проекте нет прямых заимствований от прежних проектов компании или каких-либо других проектов.

Вернуться в оглавление

3.2. Технологическая платформа

Технологическая платформа включает в себя следующие компоненты:

– компонент управления динамической памятью;

– компонент сборщика мусора;

– компонент многозадачности, включающий планировщик задач и объекты синхронизации;

– компонент ввода-вывод в синхронном и асинхронном вариантах работы, замещающий системные вызовы работы с файлами и сетью;

– планировщик потоков ввода-вывода;

– набор базовых контейнеров.

Технологическая платформа построена в предположении, что задачи могут исполняться в двух режимах: «оптимистичный», когда отсутствует ожидание каких-либо событий или данных, необходимых для обработки, и «пессимистичный», когда возникают такие ожидания. При этом задача самостоятельно может переходить между этими режимами при определенных условиях.

В технологической платформе созданы необходимые примитивы для поддержания исполнения задач в двух режимах.

Оптимистичный режим работы обслуживается исключительно неблокирующими алгоритмами. Например, в нем работают операции, выполняющие обработку в режиме чтения находящихся в кеше страниц. Поскольку операция чтения в архитектуре СУБД СОКОЛ никогда не конфликтует с другими операциями и между собой, то она всегда выполняется в оптимистичном состоянии.

Пессимистичный режим используется при необходимости ожидания события и при необходимости изоляции сложной конфликтной ситуации. Например, ожидания загрузки страницы или ожидание подтверждения сброса WAL-журнала на операции COMMIT — это всегда выполнение в пессимистичном режиме.

Многие неблокирующие алгоритмы используют динамическую память и требуют сборщика мусора, который тоже должен быть неблокирующим. В СУБД СОКОЛ используется собственная вариация DEBRA-алгоритма (Distributed Epoch-Based memory Reclamation Algorithm).

Для эффективной реализации DEBRA-алгоритма количество исполняемых потоков должно быть согласовано с количеством ядер в системе. Это привело к потребности строить многозадачность в СУБД СОКОЛ на основе кооперативного варианта и созданию собственного планировщика задач.

В итоге, на ядре ЦПУ располагается не больше одного потока и каждый поток выполняет роль исполняющей очереди. Планировщик в каждой исполняющей очереди работает изолированно, за исключением моментов балансировки нагрузки.

Кооперативная многозадачность дала дополнительные преимущества:

– контекст кооперативной задачи стал значительно меньше по размеру против системной задачи или потока;

– появилась возможность неактивную задачу переводить в бесстековое состояние. Например, задача соединения, ожидающая команду от клиента или подтверждение COMMIT, обычно не требует сложного контекста и может быть переведена в бесстековое состояние;

– синхронизация между кооперативными задачами стала дешевле, практически не требует вовлечения ядра ОС;

– собственный набор примитивов синхронизации позволил переключаться при необходимости из неблокирующего режима в режим ожидания событий.

В реализации компонента ввода-вывода дисковые операции исполняются, минуя файловый кеш ОС в предположении, что сервис, реализуемый на технологической платформе, лучше знает, что должно быть закешировано. Все дисковые операции имеют ассоциацию с определенным потоком ввода-вывода:

– приоритетная запись;

– приоритетное чтение;

– запись вытеснения;

– неприоритетная пакетная запись;

– неприоритетное пакетное чтение.

Такое разделение выполнено для достижения планировщиком потоков ввода-вывода наилучшей производительности системы в целом.

Вернуться в оглавление

3.3. Основное хранилище данных

В качестве основного хранилища данных на диске используется представление вида B-Tree. В СУБД СОКОЛ B-Tree хранилище используется для таблиц и вторичных индексов (первичные индексы хранятся вместе с данными в кластеризованных ключах таблиц).

Для широких записей и BLOB/CLOB полей используются дополнительные хранилища.

Страница дерева, прочитанная с диска в оперативную память, может содержать также цепочку изменений страницы в виде дельт. Дельта ссылается на существующий ключ и предоставляет новое значение, соответствующее ключу. Перед записью на диск, а также при наступлении определенных событий или условий цепочка дельт

консолидируется с данными страницы на новой странице данных. Данные никогда не меняются по месту расположения.

Использование дельт обусловлено прежде всего применением неблокирующего подхода.

Вернуться в оглавление

3.4. Журналирование

В СУБД Сокол используется два вида журналов: WAL-журнал и исторический.

Для восстановления данных после аварийного завершения системы достаточно WAL-журнала, который содержит только REDO-записи, и данных основного хранилища (на странице хранилища всегда размещается последняя зафиксированная и текущую редактируемая, но еще не зафиксированная версии записи).

Исторический журнал используются только для отката внутри одной транзакции до точек сохранения и для MVCC. Он содержит только UNDO-записи. Данные из журнала исторических данных не требуют синхронизации на COMMIT-операции, доступны только во время сессии базы данных и не сохраняются между сессиями.

Исторические данные для обеспечения MVCC формируются в виде цепочки исторических значений для каждого ключа и размещаются непосредственно на странице хранилища в момент исполнения операции и далее при консолидации перемещаются в журнал исторических данных, если есть заинтересованные в таких данных транзакции. Такой подход позволяет практически исключить запись исторических данных на диск при коротких транзакциях. При наличии долгих транзакций рост размера исторического журнала будет примерно соответствовать росту WAL-журнала.

Вернуться в оглавление

3.5. Конфликты при доступе к данным на странице хранилища

Неблокирующий подход в реализации доступа к записям на странице исключает некоторые конфликты:

– операция чтения записи не конфликтует с какой-либо операцией;

– операции модификации разных записей на странице исполняются параллельно.

Неблокирующий подход в реализации доступа к записям на странице не исключает следующие конфликты, которые возникают по объективным причинам:

– конфликт транзакций на отдельной записи. Он неизбежен и разрешается классически за счет выноса явной блокировки на запись в таблице блокировок. Взаимная блокировка также разрешается классическим способом;

– конфликт операций SMO между собой и операциями модификации. В реализации B-Tree в СУБД СОКОЛ выделяются следующие SMO-операции: split, merge, borrow, consolidate (формирование новой версии страницы после применения дельт изменений). Операции split, merge, borrow являются типовыми SMO-операциями B-Tree.

Вернуться в оглавление

4. УСТАНОВКА СУБД ЛИНТЕР СОКОЛ

4.1. Перед установкой

Перед установкой СУБД СОКОЛ обязательно ознакомьтесь с лицензионным соглашением, которое поставляется в одном архиве с дистрибутивом (файл license.txt). Установка системы и работа с ней подразумевают автоматическое согласие с условиями лицензионного соглашения.

Также перед установкой СУБД СОКОЛ ознакомьтесь с минимальными техническими требованиями к оборудованию (см. п. 2.3. Технические требования к оборудованию), техническими характеристиками системы (см. п. 2.4. Технические характеристики СУБД) и убедитесь, что Ваша система соответствует им.

Дистрибутив СУБД ЛИНТЕР СОКОЛ включает компоненты с открытым исходным кодом. Полный перечень используемых компонентов, их версии, сведения о лицензиях и условиях использования приведены в разделе 4.4 «Сведения о сторонних компонентах».

Функция JIT-компиляции запросов (с использованием библиотеки LLVM) не является обязательной для работы СУБД и может быть отключена.

Вернуться в оглавление

4.2. Установка СУБД в среде ОС UNIX/Windows

Установка СУБД СОКОЛ в средах UNIX и Windows выполняется путём распаковки архива в выбранный каталог или с помощью инсталлятора (для соответствующей ОС).

ВАЖНО перед началом работы с СУБД: в каждой операционной системе имеется свое ограничение на количество файловых дескрипторов. Для 64-х битной системы в Linux максимальное число файловых дескрипторов по умолчанию равно 1024, в Windows — 2^{32}. Учитывая организацию хранения данных в СУБД СОКОЛ и то, что помимо сервера БД файловые дескрипторы используют и другие приложения ОС, в Linux есть бОльшая вероятность достичь этого предела. В таком случае любой новый процесс и рабочие потоки будут заблокированы, система вернёт ошибку.

Для стабильной работы сервера базы данных проверьте, чтобы число открытых файлов (ориентировочно это суммарное число таблиц и индексов) не превышало допустимого числа файловых дескрипторов в операционной системе.

Вернуться в оглавление

4.2.1. Изменение максимального числа открытых файловых дескрипторов в ОС Linux

При необходимости в ОС Linux можно увеличить значение максимального числа открытых файловых дескрипторов при помощи настроек операционной системы.

Для получения информации об общем числе файлов, открытых всеми процессами в ОС Linux введите команду:

```
# cat /proc/sys/fs/file-nr
```

В выведенном сообщении будет представлено:

– первое число — общее количество используемых на данный момент времени файловых дескрипторов;

– второе число — количество выделенных процессам, но не используемых в данный момент дескрипторов;

– третье число — максимальное количество открытых дескрипторов.

Для увеличения лимита файловых дескрипторов в Linux используйте утилиту sysctl и поставьте нужное число, например, 200000:

```
# sysctl -w fs.file-max=200000
```

Внесённые изменения останутся активными до следующей перезагрузки.

Для изменений на постоянной основе, отредактируйте конфигурационный файл:

```
# vi /etc/sysctl.conf
```

В файле нужно установить необходимый параметр, например 200000:

```
fs.file-max=200000
```

и перезагрузить ОС.

После перезагрузки системы можно проверить текущее значение максимального числа файловых дескрипторов с помощью команды:

```
# cat /proc/sys/fs/file-max
```

Вернуться в оглавление

4.2.2. Изменение максимального числа открытых файловых дескрипторов в ОС Windows

В ОС Windows достаточно большое максимальное число дескрипторов, что позволяет работать без дополнительных корректировок данного параметра.

Если же у вас возникли проблемы, то возможно проблема кроется в утечке дескрипторов ОС и лежит в слое прикладного программного обеспечения.

Вернуться в оглавление

4.3. Назначение файлов дистрибутива

Состав дистрибутива для ОС UNIX и ОС Windows идентичный. Отличие заключается только в наименовании расширений для исполняемых файлов и динамических библиотек (далее по тексту раздела обобщенное расширение указано как <ext>). В текущем описании будем ссылаться на имена файлов в ОС Windows.

В состав дистрибутива входит:

1) bin/ — каталог, содержащий исполняемые файлы и динамические библиотеки:

– bin/vsql_server.<ext> — исполняемый файл сервера СУБД;

– bin/vsql_server_wait.<ext> — утилита ожидания запуска сервера. На протяжении таймаута (300 с) она проверяет, отвечает ли сервер на запросы. Если да, то утилита завершается с успешным кодом завершения. Иначе, по истечении

установленного таймаута, утилита возвращает неуспешный код завершения. Утилита полезна в скриптах, где перед посылом серверу команд требуется удостовериться, что сервер запустился и отвечает на запросы;

– bin/vsql_console.<ext> — клиентская программа для взаимодействия с сервером. Она позволяет исполнять запросы к СУБД в интерактивном и пакетном режимах;

– bin/vsql_copy.<ext> — утилита, позволяющая выгружать (экспортировать) из БД и загружать (импортировать) в БД из CSV-формата (Comma Separated Values). В качестве параметра можно указать не только таблицу БД, но и SQL-запрос (select в случае выгрузки, insert в случае загрузки);

– bin/soqol-jdbc-<версия>-SNAPSHOT.jar — файлы JDBC-драйвера, где в качестве <версии> указана версия драйвера;

– bin/vsql-odbc.<ext> — файл 64-битной сборки ODBC-драйвера;

– bin/odbc32/vsql-odbc.<ext> — файл 32-битной сборки ODBC-драйвера;

– bin/php_pdo_soqol.<ext> - файл PHP-драйвера;

– bin/trace_msg.py — утилита чтения файлов трассировки (предварительно необходимо установить Python). Утилита может принимать маску имён файлов в unix стиле, что позволяет считывать сообщения из нескольких последовательных файлов трассировки (например, для чтения единственного файла указывается debugger/trace/trace_msg.py trace_0000000000000000.dat, а для чтения файлов по маске указывается debugger/trace/trace_msg.py trace_*.dat). Имеющиеся возможности фильтрации и форматирования вывода утилиты можно посмотреть вызвав её с опцией --help или -h.

2) doc/ — каталог, содержащий файлы документации.

3) examples/ — каталог, содержащий примеры работы с языком запросов и процедурным языком в текстовых файлах.

4) license.txt — текстовый файл, содержащий лицензионное соглашение на использование устанавливаемой версии СУБД.

5) NOTICE.txt — текстовый файл, содержащий уведомления о сторонних компонентах с открытым исходным кодом, включённых в состав СУБД, с указанием наименований компонентов, их лицензий и сведений об авторских правах.

6) licenses/ — каталог, содержащий полные тексты лицензий каждого из сторонних компонентов, включённых в состав СУБД.

7) changelog.txt — текстовый файл, содержащий перечень изменений для конкретной версии СУБД.

8) readme.txt — текстовый файл, содержащий базовую информацию об устанавливаемой версии СУБД, данные об установке, краткую информацию о каталогах дистрибутива СУБД ЛИНТЕР СОКОЛ, контактную информацию службы поддержки компании РЕЛЭКС.

Вернуться в оглавление

4.4. Сведения о сторонних компонентах

В составе СУБД ЛИНТЕР СОКОЛ используются следующие программные компоненты, разработанные третьими лицами и распространяемые на условиях открытых лицензий.

4.4.1. LLVM

Версия: 11.0.1

Назначение: обеспечение JIT-компиляции и нативной компиляции кода запросов и хранимых процедур в ядре СУБД (исполняемый файл vsql_server).

Компонент не является обязательным. Работа СУБД возможна с отключённой JIT-компиляцией.

4.4.2. ICU (International Components for Unicode)

Версия: 66.1

Назначение: перекодирование строковых данных, поддержка правил сравнения (collation) и функций интернационализации в клиентских библиотеках и утилитах.

4.4.3. libyaml

Версия: 0.2.5

Назначение: чтение и запись конфигурационного файла soqol_config.yml в ядре СУБД.

4.4.4. zlib

Версия: 1.2.11

Назначение: обеспечение функций сжатия данных (используется как зависимость компонента LLVM).

4.4.5. unixODBC

Версия: 2.3.7

Назначение: предоставление интерфейсных заголовочных файлов для сборки ODBC-драйвера СУБД. Сама библиотека unixODBC не включается в состав дистрибутива СУБД и должна быть установлена отдельно в операционной системе при необходимости использования ODBC-интерфейса.

Полные тексты лицензий и уведомления об авторских правах содержатся в файле NOTICE.txt и в каталоге licenses/ в корневом каталоге дистрибутива СУБД.

Вернуться в оглавление

5. ЗАПУСК СЕРВЕРА И НАЧАЛО РАБОТЫ С СУБД ЛИНТЕР СОКОЛ

Запуск сервера осуществляется исполнением бинарного файла vsql_server.

Рекомендуется включить исполняемые файлы СУБД СОКОЛ в путь поиска исполняемых файлов ОС (переменная окружения PATH) или использовать абсолютный или относительный путь для их запуска.

Далее в документации при указании команд будут опускаться местоположения исполняемых файлов. При необходимости префикс полного или относительного пути необходимо дописать к исполняемому файлу согласно правилам данной операционной системы.

Вернуться в оглавление

5.1. Первоначальный запуск сервера СУБД

При запуске сервер СУБД работает с базами данных и их настройками, расположенными в текущем каталоге ОС. Поэтому сделайте текущим ваш рабочий каталог, например:

```
cd C:\opt\soqol
```

В ОС Windows перед командой cd необходимо сделать текущим диск, на котором размещается нужный каталог, в данном примере C. Для этого введите команду:

```
C:
```

Если используемый диск является текущим, этот шаг можно пропустить.

При необходимости предварительно создайте каталог:

```
md C:\opt\soqol
```

Затем сделайте его текущим командой cd.

Запуск сервера выполняется из файла vsql_server командой:

```
vsql_server
```

При первом запуске сервера СУБД в текущем каталоге создаётся подкаталог AUXDB со служебными данными. В этом подкаталоге хранится информация о создаваемых в дальнейшем базах, а также другая служебная информация. Если остановить работу сервера и повторно запустить, то сервер будет использовать ранее созданный подкаталог AUXDB.

Сервер при старте проверяет в текущем каталоге наличие конфигурационного файла soqol_config.yml. Если он есть, то сервер использует его. Если конфигурационного файла нет, то он создается с настройками по умолчанию. Подробнее о настройках конфигурационного файла см. п. 7. Конфигурационный файл soqol_config.yml.

Настройки конфигурационного файла в дальнейшем можно изменить. После сохранения внесённых изменений сервер необходимо перезапустить.

Сервер готов к приёму команд после появления в консоли сообщения vsql_server is ready to receive commands.

Если данного сообщения нет, то:

1) Внимательно проверьте верность совершённых действий и введённых команд для запуска сервера СУБД.

2) Если п. 1 не помог в решении вопроса, обратитесь в техническую поддержку компании РЕЛЭКС.

Запущенный сервер предназначен для одновременного обслуживания множества баз. Запуск нескольких экземпляров vsql_server возможен, но не имеет практического смысла.

При длительном простое в консоли выводится информационное сообщение.

Вернуться в оглавление

5.2. Создание базы данных

После запуска сервера СУБД можно создать базу данных. Для начала утилитой vsql_console подключитесь к сервису управления базами данных сервера (далее «сервис управления»):

```
vsql_console -h SOQOL:SOQOL@localhost
```

При успешном запуске утилиты будет выведено сообщение о готовности к приёму команд:

```
vsql>
```

Введите команду для создания базы данных DB:

```
create database DB on './DB';
```

В данной команде DB — это имя базы данных, которая создаётся в каталоге './DB' (подкаталог DB в текущем каталоге). Созданная база данных готова к работе.

Подробнее о правилах именования БД смотри таблицу 4 в п. 11.1 CREATE DATABASE.

В новой базе данных присутствует пользователь с именем SOQOL с правами администратора и паролем SOQOL. Ознакомиться с другими объектами и субъектами, которые создаются в базе по умолчанию, можно в п. 15 Объекты, формируемые системой при создании БД в СУБД СОКОЛ.

После создания базы данных можно завершить работу с vsql_console командой:

```
quit
```

Вернуться в оглавление

5.3. Подключение к базе данных из утилиты vsql_console

Для работы с базой данных необходимо выполнить подключение к ней (а не к сервису управления, как это было сделано в предыдущем разделе).

Для этого дополните строку подключения именем базы данных:

```
vsql_console -h SOQOL:SOQOL@localhost/DB
```

После подключения к базе данных она становится текущей, и дальнейшие запросы будут обращены к её объектам.

Вернуться в оглавление

5.4. Выполнение первого запроса к базе данных

Для начала работы можно выполнить первый запрос к базе данных:

```
select * from INFO.SCHEMAS;
```

Этот запрос извлекает и отображает в консоли перечень схем базы данных.

Вернуться в оглавление

5.5. Создание таблицы в базе данных

В качестве демонстрации работы с СУБД возьмём пример, близкий к реальному — хранение информации об абонентах.

Для систематизации данных об абонентах создайте таблицу PHONE с информацией о номерах телефонов NUMBER и их владельцах NAME:

```
create table PHONE (NUMBER int primary key, NAME varchar(200));
```

У каждого номера может быть только один владелец, поэтому идентификатором (первичным ключом) будет служить номер телефона.

Вернуться в оглавление

5.6. Вставка данных в таблицу базы данных

В созданную таблицу PHONE добавьте запись о первом абоненте:

```
insert into PHONE values (89876543210, 'Иванов');
```

Добавьте в таблицу еще несколько записей для дальнейшей демонстрации различных возможностей работы с данными:

```
insert into PHONE values (89876543212, 'Попов');
insert into PHONE values (89876543213, 'Петров');
insert into PHONE values (89876543214, 'Сидоров');
```

Обратите внимание, что в утилите vsql_console каждая команда вводится отдельно.

Предположим, что у номера 89876543212 сменился владелец. Если попробовать создать нового абонента с уже существующим в БД номером телефона:

```
insert into PHONE values (89876543212, 'Васечкин');
```

В этом случае наличие первичного ключа не позволит создать нового абонента с существующим в базе идентификатором, и введение такого запроса приведёт к ошибке:

```
SQLExecDirect failed
code -20014: HY000=runtime error
```

Как изменить данные, если у номера сменился владелец, рассмотрим далее.

Вернуться в оглавление

5.7. Выборка вставленных данных

Посмотреть все записи таблицы PHONE можно с помощью команды:

```
select * from PHONE;
```

Будет извлечена информация:

```
NUMBER     | NAME
-----------+-----------
89876543210 | Иванов
89876543212 | Попов
89876543213 | Петров
89876543214 | Сидоров
```

Посмотреть владельца телефонного номера 89876543213 позволит выборка записей с заданным условием:

```
select * from PHONE where NUMBER = 89876543213;
```

В консоли будет выведена соответствующая информация:

```
NUMBER     | NAME
-----------+-----------
89876543213 | Петров
```

Вернуться в оглавление

5.8. Изменение данных в таблице БД

Вернёмся к случаю, когда у номера телефона 89876543212 сменился владелец. Необходимо внести изменения в таблицу БД. Для этого введите:

```
update PHONE set NAME = 'Васечкин' where NUMBER = 89876543212;
```

Проверить верность внесённых изменений можно запросом:

```
select * from PHONE;
```

В окне консоли будет выведена запрашиваемая информация:

```
NUMBER     | NAME
-----------+-----------
89876543210 | Иванов
89876543212 | Васечкин
89876543213 | Петров
89876543214 | Сидоров
```

Вернуться в оглавление

5.9. Удаление данных из таблицы БД

Допустим, абонент Сидоров переехал в другой регион и написал заявление о деактивации номера телефона. Удалите данные из БД командой DELETE, указав в условии его номер телефона:

```
delete from PHONE where NUMBER = 89876543214;
```

Извлеките данные из таблицы для проверки произведённых действий:

```
select * from PHONE;
```

В полученной информации видно, что данные абонента удалены:

```
NUMBER     | NAME
-----------+-----------
89876543210 | Иванов
89876543212 | Васечкин
89876543213 | Петров
```

Вернуться в оглавление

5.10. Удаление таблицы из базы данных

Предположим, что компанией было принято решение вести данные абонентов в новом приложении, которое работает с таблицами другого формата. Таблица PHONE стала не нужна. Для удаления таблицы PHONE из базы данных введите запрос:

```
drop table PHONE;
```

Если после этого оператором SELECT сделать выборку из таблицы PHONE, то в консоли будет выведено сообщение, что система не находит эту таблицу:

```
SQLExecDirect failed
code -25016: HY000=(0,13)ERROR: unknown symbol PHONE
```

Перед работой с примерами следующих разделов рекомендуем выйти из vsql_console командой:

```
quit
```

Вернуться в оглавление

5.11. Удаление базы данных

Допустим, после решения вести данные абонентов в новом приложении, была создана новая база данных. Вся информация скопирована, и база DB не нужна, её нужно удалить. Для этого выполните следующие шаги:

1) Подключитесь к сервису управления базами данных:

```
vsql_console -h SOQOL:SOQOL@localhost
```

2) Выполните останов базы данных:

```
shutdown database "DB";
```

3) Удалите базу данных:

```
drop database "DB";
```

После заключительного шага можно открыть текущий каталог и убедиться в успешном удалении БД: в каталоге будет отсутствовать подкаталог удалённой базы данных.

Вернуться в оглавление

5.12. Останов сервера

Для корректного завершения работы сервера из подключения к сервису управления базами данных необходимо выполнить команду:

```
SHUTDOWN SERVICE;
```

Выполнить останов сервера может пользователь сервиса с системной привилегией DATABASE ADMIN.

Подробнее о командах внешнего управления БД см. п. 11. Работа с сущностью БД.

Вернуться в оглавление

6. О СТРОКЕ ПОДКЛЮЧЕНИЯ ДЛЯ УТИЛИТ

Строка подключения содержит информацию, необходимую для подключения к источнику данных. В качестве источника данных могут выступать базы данных и сервис по управлению ими.

Строка подключения может использоваться как при работе с утилитами, входящими в состав дистрибутива СУБД СОКОЛ, так и при подключении через поддерживаемые системой интерфейсы, например, JDBC.

В клиентские утилиты строка подключения передается параметром -h <строка подключения>. Она имеет следующий вид:

```
[<protocol>://]<user>:[<password>][@<host>[:<port>]][/[<dbname>]][?<параметры>]
```

Где <параметры> представлены в виде пар:

```
<параметр>=<значение>[&<параметр>[=<значение>]...]
```

В строку подключения входит:

– <protocol> — протокол (значение по умолчанию — soqol);

– <user> — имя пользователя;

– <password> — пароль пользователя (значение по умолчанию — пустая строка);

– <host> — хост (значение по умолчанию — localhost);

– <port> — порт (значение по умолчанию — 2060);

– <dbname> — имя базы данных, к которой необходимо подключиться. При отсутствии имени базы данных выполняется подключение к сервису управления базами данных;

– <параметр> — возможный параметр для указания на текущий момент один — charset, который используется для указания имени кодировки строковых данных, передаваемых клиентским приложением на сервер (допустимые имена см. ПРИЛОЖЕНИЕ 3).

Во избежание ошибок неоднозначности задания свойств позиционные параметры запрещено определять через раздел <параметры>.

Учитывая значения по умолчанию, для подключения к сервису управления с заданным именем пользователя и паролем можно использовать короткую командную строку:

```
vsql_console -h <user>:<password>
```

Пользователь, заданный в строке подключения, должен существовать в БД (подробнее о создании пользователя см. п. 12.8.1.6 CREATE USER).

ВАЖНО: в строке подключения имя пользователя и пароль вводите с учётом регистрозависимости. Т.е. обратите внимание, что:

– если при создании пользователя (create user) его имя было указано без кавычек (пример, create user cLiEnT …), то символы имени приведены системой к верхнему регистру (т.е. в системе пользователь сохранен как CLIENT). Поэтому в строке подключения необходимо указывать имя пользователя в верхнем регистре — CLIENT;

– если при создании пользователя (create user) его имя было указано в кавычках (например, create user "cLiEnT" …), то оно сохранено в системе без изменений (т.е. как cLiEnT) . В этом случае в строке подключения имя пользователя нужно указывать в том же виде (без кавычек) — cLiEnT.

Вернуться в оглавление

7. КОНФИГУРАЦИОННЫЙ ФАЙЛ SOQOL_CONFIG.YML

Конфигурационный файл использует формат YAML 1.2 (см. https://yaml.org/spec/1.2/spec.html).

Далее будут использоваться следующие единицы измерения и значения для параметров:

1) Размер памяти:

– KB = килобайты, 2^10;

– MB = мегабайты, 2^20;

– GB = гигабайты, 2^30;

– TB = терабайты, 2^40.

2) Количественные множители:

– K = кило, 10^3;

– M = мега, 10^6;

– G = гига, 10^9;

– T = тера, 10^12.

3) Логическое значение:

– yes, on, true = значение true;

– no, off, false = значение false.

4) При отсутствии единицы измерения у значения какого-либо параметра подразумевается значение в штуках.

Описание параметров конфигурации:

1) service — группа настроек сервиса СУБД:

– service.listen: адреса сервера для приема соединений в виде списка URL (комбинация протокола, адреса и порта), элементы которого разделяются символом ";". Значение по умолчанию для UNIX — "/tmp/soqol:2060;0.0.0.0:2060". Значение по умолчанию для Windows — "0.0.0.0:2060". Адрес «0.0.0.0» позволяет принимать соединения с любых сетевых интерфейсов по протоколу TCP/IP.

Для управления доступом пользователей к серверу, вместо указанных параметров по умолчанию можно указать необходимые;

– service.max_connections: максимальное число подсоединений к сервису. Значение по умолчанию 100.

Допустимо указание любого иного значения в рамках диапазона от 1 до 65536.

Каждый клиент может открыть больше одного соединения и при превышении установленного максимума попытки новых подсоединений будут заканчиваться ошибкой до тех пор, пока хотя бы одно из соединений не будет завершено.

Важно: для обеспечения управления базой данных и возможности подсоединения к ней пользователя с системной привилегией DATABASE ADMIN зарезервировано дополнительно одно соединение, сверх указанного значения параметра service.max_connections, не предоставляемое пользователям без системной привилегии DATABASE ADMIN;

– service.memory: общий объём используемой сервером оперативной памяти (далее – ОП), округленный в большую сторону, кратному размеру гранулы (16Mb). Значение по умолчанию 0, что обозначает автоматическое вычисление используемой сервером ОП в соответствии с исходными данными: 2 GB (если общий объём ОП < = 16 GB и объем свободной ОП > = 6 GB) или 1 GB (если общий объём ОП < = 16 GB и объем свободной ОП < 6 GB).

При общей ОП > 16GB система выполняет расчёты и выбирает наименьшее значение:

– если свободной ОП больше 8Гб, то выбирается наименьшее значение из 25% всей ОП или (свободная ОП - 8Гб), но не менее 1Гб;

– если свободной ОП меньше или равно 8Гб, то значение по умолчанию для service.memory = 1Гб.

Недостаток используемой оперативной памяти может привести к ощутимому замедлению и сбоям в работе СУБД. В этом случае можно скорректировать параметр в большую сторону. Однако стоит помнить, что оперативная память каждой машины ограничена и для работы других приложений также необходима часть оперативной памяти.

Для автоматической конфигурации при запуске сервера необходим минимум 1Gb оперативной памяти. В противном случае, необходимо явно указать количество свободной оперативной памяти;

– service.cores: число используемых ядер (потоков исполнения). По умолчанию 0 — используются все ядра всех процессоров рабочей станции.

При необходимости распределения ядер для работы других приложений на машине, можно установить значение, меньше числа всех ядер всех процессоров рабочей станции. Уменьшение параметра service.cores приведет к снижению скорости обслуживания запросов сервером БД;

– service.scheduler_timeslice: интервал времени работы сопрограммы без переключения, отведённый планировщиком в миллисекундах. Значение по умолчанию 10 мс;

– service.data_dir: текущий рабочий каталог сервиса. По умолчанию используется каталог файла конфигурации.

2) trace — группа настроек трассировки:

– trace.components: список активных компонентов трассировки. По умолчанию — пустой, что означает отключенную трассировку;

– trace.buffer_size: размер буфера памяти, в котором накапливаются сообщения перед записью на диск. Значение по умолчанию — 16MB;

– trace.file_size: максимальный размер одного файла трассировки. Значение по умолчанию 128MB.

3) storage — параметры пакетной записи страниц хранилища:

– storage.press_table.task_count: количество задач фоновой очистки страниц хранилища. Значение по умолчанию — 0, означает автоконфигурацию, максимальное значение зависит от количества ядер процессоров;

– storage.press_table.page_count: количество страниц хранилища, обслуживаемых фоновой очисткой за один запуск. Значение по умолчанию — 128, min 16, max 1024.

Все параметры, указанные в конфигурационном файле, можно корректировать для оптимизации работы СУБД и при понимании последствий. После внесения корректировок необходимо перезапустить сервер.

При необходимости обратиться в службу технической поддержки.

Вернуться в оглавление

8. КАТАЛОГ СЕРВИСА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ AUXDB

Каталог AUXDB создаётся при первом запуске сервера и представляет собой каталог базы данных сервиса управления базами данных.

AUXDB используется для:

– хранения сведений о пользователях сервиса;

– создания/удаления баз данных. Для этих целей в БД AUXDB существует служебная таблица баз данных;

– хранения настроек каждой базы данных.

Вернуться в оглавление

9. ПРОГРАММНЫЕ ИНТЕРФЕЙСЫ

Прикладные программные интерфейсы доступа к базам данных обеспечивают унификацию работы с различными СУБД, помогают приложениям на стороне клиента получать доступ к базе данных на стороне сервера.

В следующих пунктах рассмотрим установку реализованных в СУБД СОКОЛ драйверов программных интерфейсов.

Вернуться в оглавление

9.1. ODBC-драйвер

Напомним, что в составе дистрибутива СУБД СОКОЛ поставляются следующие файлы (поддерживающие ODBC версии 3.8):

– для Windows: bin/vsql-odbc.dll — динамическая библиотека ODBC-драйвера СУБД;

– для Linux: bin/vsql-odbc.so — ODBC-драйвер СУБД.

Вернуться в оглавление

9.1.1. Установка ODBC-драйвера под ОС Windows

Установка ODBC-драйвера СУБД СОКОЛ в среде ОС Windows выполняется из консоли ОС, запущенной под пользователем с правами администратора, следующими командами:

1) Команда регистрации драйвера:

```
odbcconf INSTALLDRIVER "Soqol ODBC Driver|driver=c:\soqol\bin\vsql-odbc.dll"
```

Где указаны:

– Soqol ODBC Driver — имя драйвера. Имя драйвера может быть задано произвольным, но должно согласовываться с описанной ниже командой регистрации источника данных;

– c:\soqol\bin\ — путь к файлу vsql-odbc.dll, содержащему реализацию ODBC-драйвера СУБД ЛИНТЕР СОКОЛ. Укажите абсолютный путь к файлу исходя из его расположения на вашей рабочей станции.

2) Команда регистрации источника данных (DSN) привязывает DSN к зарегистрированному в предыдущей команде имени драйвера:

```
odbcconf CONFIGDSN "Soqol ODBC Driver" "DSN=soqoldsn|SERVER=localhost|UID=username|PWD=password"
```

Где указаны:

– soqoldsn — имя DSN. Имя источника данных может быть задано произвольным. Это имя будет использовано приложением для подключения к БД;

– localhost — адрес сервера в качестве значения параметра <SERVER>;

– username — имя пользователя в качестве значения параметра <UID>;

– password — пароль пользователя в качестве значения параметра <PWD>. Пароль может быть пустой, тогда его не нужно указывать;

Помимо указанных параметров через символ "|" можно задать и другие параметры соединения (<URL>, <DATABASE>, <PORT>, <CHARSET>). Например:

```
odbcconf CONFIGDSN "Soqol ODBC Driver"
    "DSN=soqoldsn|SERVER=localhost|UID=username|PWD=password|URL=soqol://SOQOL:@0.0.0.0:2060/DB".
```

ВАЖНО: если задан параметр <URL> и дополнительно отдельные параметры через символ "|", то отдельные параметры перекрывают значения параметров строки <URL>.

ВАЖНО: подключиться можно или к базе данных, или к сервису. При отсутствии параметра DATABASE подключение выполняется к сервису.

Проверьте наличие зарегистрированного источника данных через 64-битную версию графической утилиты odbcad32. Для этого запустите odbcad32 в консоли ОС или через меню запуска Win+R. В открывшемся окне утилиты перейдите на вкладку:

– «Драйверы» и убедитесь в наличии драйвера "Soqol ODBC Driver" (рис. 1);

– «Пользовательский DSN» и убедитесь в наличии источника "soqoldsn" (рис. 2).

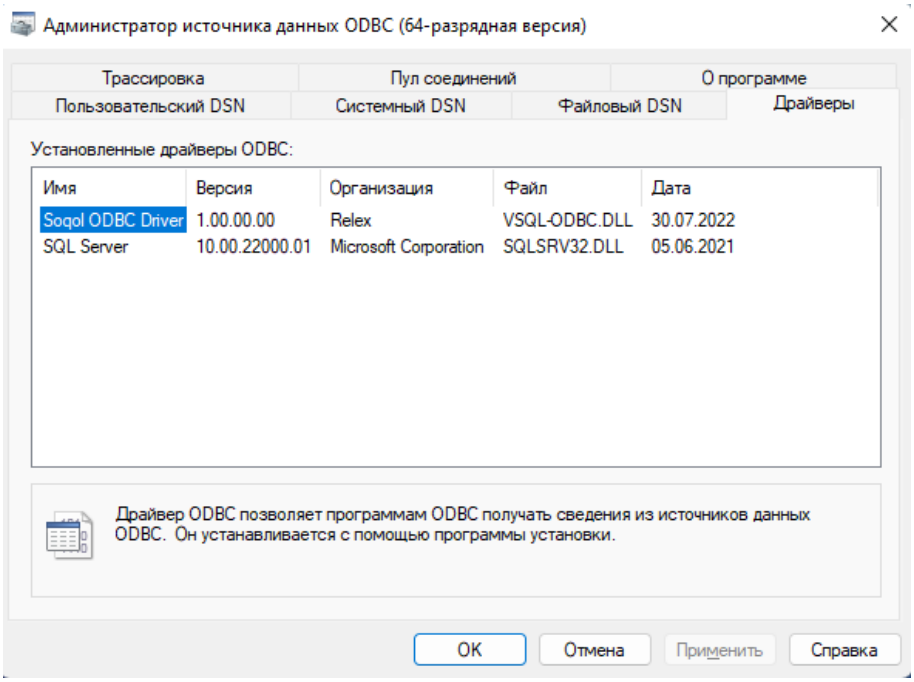


Рисунок 1 — Вкладка «Драйверы» в окне администратора источника данных

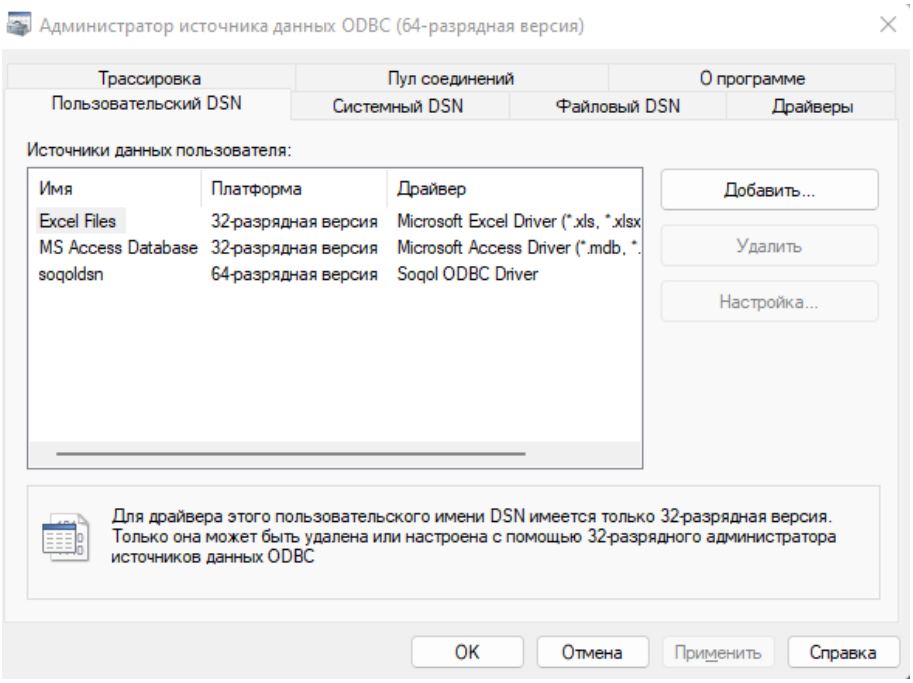


Рисунок 2 — Вкладка «Пользовательский DSN» в окне администратора источника данных

Вернуться в оглавление

9.1.2. Установка ODBC-драйвера для ОС UNIX

Данное руководство ориентировано на работу с менеджером драйверов unixODBC.

При настройке доступа к ODBC-драйверу СУБД СОКОЛ используются две утилиты из пакета unixODBC: odbcinst и isql. Перед началом работы проверьте их наличие командами which odbcinst и which isql.

При наличии менеджера драйверов unixODBC, а также утилит odbcinst и isql переходите к установке ODBC-драйвера.

Установка ODBC-драйвера СУБД СОКОЛ выполняется в следующем порядке:

1) Создайте файл odbcinst.ini с настройками установки драйвера в удобном для Вас каталоге следующего содержания:

```
[soqoldrv]
Description=Soqol ODBC Driver
Driver=/opt/soqol/bin/vsql-odbc.so
```

Где указаны:

– [soqoldrv] — имя драйвера, которое должно согласовываться с информацией, указанной в файле odbc.ini (см. далее);

– Soqol ODBC Driver — описание драйвера базы данных, которое может быть произвольным;

– /opt/soqol/bin/vsql-odbc.so — полный путь к библиотеке ODBC-драйвера из дистрибутива СУБД ЛИНТЕР СОКОЛ (путь указан в качестве примера). Укажите абсолютный путь к файлу из дистрибутива СУБД ЛИНТЕР СОКОЛ, исходя из его расположения на вашей рабочей станции.

2) В консоли, запущенной с правами суперпользователя, выполните команду для установки драйвера:

```
sudo odbcinst -i -d -f ./odbcinst.ini
```

Где указаны:

– ключ -i — сообщает, что действием команды является установка объекта;

– ключ -d — сообщает, что объектом действия является драйвер;

– ключ -f — сообщает, что следующим параметром будет указан файл настроек установки драйвера;

– файл настроек ./odbcinst.ini — полное имя созданного вами в п. 1 файла настроек.

Из указанного в команде файла утилита odbcinst копирует параметры драйвера в /etc/odbcinst.ini (системный файл настроек драйверов unixODBC).

Проверить успешность установки драйвера можно командой:

```
sudo odbcinst -q -d -n "soqoldrv"
```

Где указаны:

– ключ -q — запрашивает список объектов, присутствующих в системе;

– ключ -d — сообщает, что объектом действия является драйвер;

– ключ -n — сообщает, что следующим параметром будет указано имя драйвера;

– soqoldrv — имя драйвера.

Результатом выполнения команды будет информация об установленном в ОС драйвере.

3) Создайте в удобном для Вас каталоге файл odbc.ini с настройками регистрации источника данных (DSN) следующего содержания:

```
[soqoldsn]
Driver = soqoldrv
UID = username
PWD = password
SERVER = localhost
DATABASE = DB
```

Где указаны:

– [soqoldsn] — имя источника данных (DSN), которое может быть задано произвольным. Используйте это имя в приложениях для подключения к БД;

– soqoldrv — имя драйвера, обрабатывающего запросы к источнику данных. В качестве значения <Driver> должно быть указано имя драйвера, заданное в п. 1 при регистрации драйвера;

– username — имя пользователя в качестве значения параметра <UID>;

– password — пароль пользователя в качестве значения параметра <PWD>. Пароль может быть пустой, тогда его не нужно указывать;

– localhost — адрес сервера в качестве значения параметра <SERVER>;

– DB - имя базы данных в качестве значения параметра <DATABASE>, с которой далее будет осуществлено соединение. Необязательный параметр.

Также могут быть указаны и другие параметры, которые позволяют уточнить параметры соединения и перекрыть значения по умолчанию:

– <PORT> — порт подключения;

– <CHARSET> — параметр используется для указания имени кодировки строковых данных, передаваемых клиентским приложением на сервер (см. ПРИЛОЖЕНИЕ 3);

– <URL> — комбинация параметров для настройки соединения между клиентским компьютером и базой данных в одной строке (например, soqol://SOQOL:SOQOL@localhost).

ВАЖНО: если задан параметр <URL> и дополнительно отдельные параметры соединения, то их значения перекрывают значения параметров в строке <URL>.

ВАЖНО: подключиться можно или к базе данных или к сервису. При отсутствии параметра <DATABASE> подключение выполняется к сервису.

4) Введите команду для регистрации DSN:

```
sudo odbcinst -i -s -f ./odbc.ini
```

Где указаны:

– ключ -i — сообщает, что действием команды является регистрация объекта;

– ключ -s — сообщает, что объектом действия является источник данных;

– ключ -f — сообщает, что следующим параметром будет указан файл настроек для регистрации источника данных;

– файл настроек ./odbc.ini — полное имя созданного вами в п. 3 файла настроек.

Проверить регистрацию DSN можно командой:

```
sudo odbcinst -q -s -n soqoldsn
```

Где указаны:

– ключ -q — запрашивает список объектов, присутствующих в системе;

– ключ -d — сообщает, что объектом действия является источник данных;

– ключ -n — сообщает, что следующим параметром будет указано имя источника данных;

– soqoldsn — имя источника данных.

Результатом выполнения команды будет отображение на экране содержимого файла ./odbc.ini.

Результатом выполнения команды будет информация об установленном в ОС источнике данных.

После завершения установки драйвера и регистрации DSN можно проверить работоспособность ODBC-драйвера. Для этого:

– запустите сервер СУБД СОКОЛ (см. п. 5.1. Первоначальный запуск сервера СУБД);

– присоединитесь к сервису и создайте базу данных с именем DB (см. п. 5.2. Создание базы данных). Имя этой базы ранее было указано в файле odbc.ini;

– выполните команду:

```
isql soqoldsn SOQOL SOQOL
```

Где указаны:

– soqoldsn — имя источника данных;

– SOQOL — имя пользователя;

– SOQOL — пароль пользователя.

После соединения появится сообщение Connected! и приглашение для выполнения запросов SQL>.

Для выхода из утилиты введите команду:

```
quit
```

Вернуться в оглавление

9.1.3. Поддерживаемые версии, список реализованных операций в ODBC

В текущей версии СУБД ЛИНТЕР СОКОЛ реализовано подмножество функций ODBC, в том числе покрывающих исполнение теста TPC-C.

Список реализованных функций ODBC-драйвера в текущей версии СУБД СОКОЛ указан в Приложении 1.

Вернуться в оглавление

9.2. JDBC-драйвер

Напомним, что в составе дистрибутива текущей версии СУБД СОКОЛ JDBC-драйвер поставляется в виде файла soqol-jdbc-<номер_версии>.jar (поддерживает JDBC версии 4.2).

Вернуться в оглавление

9.2.1. Подключение JDBC-драйвера

Подключения JDBC-драйвера к пользовательскому Java-приложению выполняется путём указания пути к JDBC-драйверу в параметре -cp (classpath) при запуске приложения (подробнее в п. 9.2.3 Проверка работоспособности).

Вернуться в оглавление

9.2.2. Строка подключения

Строка подключения содержит информацию, необходимую для подключения к источнику данных. В качестве источника данных могут выступать базы данных и сервис по управлению ими.

Строка подключения через JDBC-драйвер имеет следующий формат:

```
jdbc:soqol://<user>:[<password>][@<host>[:<port>]][/[<dbname>]][?<параметры>]
```

Где параметры представлены в виде пар и можно указать несколько параметров через «&»:

```
<параметр>=<значение>[&<параметр>=<значение>...]
```

В строку подключения входят:

– jdbc:soqol:// — префикс, обязательный для JDBC-драйвера СУБД СОКОЛ, содержащий значение протокола по умолчанию — soqol;

– <user> — имя пользователя;

– <password> — пароль пользователя (значение по умолчанию — пустая строка);

– <host> — имя хоста или IP хоста для соединения (значение по умолчанию - localhost);

– <port> — порт подключения (значение по умолчанию — 2060);

– <dbname> — имя базы данных, к которой необходимо выполнить подключение.

В текущей версии в качестве дополнительного <параметра> в строке подключения можно указать только параметр <charset> — используется для указания имени кодировки строковых данных, передаваемых клиентским приложением на сервер (см. Приложение 3).

ВАЖНО: подключиться можно или к базе данных или к сервису. При отсутствии <dbname> подключение выполняется к сервису.

Например, строка для подключения к сервису:

```
jdbc:soqol://SOQOL:SOQOL@localhost:2060
```

Строка для подключения к базе данных:

```
jdbc:soqol://SOQOL:SOQOL@localhost:2060/DB
```

Вернуться в оглавление

9.2.3. Проверка работоспособности

Для проверки работоспособности JDBC-драйвера выполните следующие действия:

1) Запустите сервер СУБД СОКОЛ (см. п. 5.1. Первоначальный запуск сервера СУБД).

2) Подключитесь к сервису и создайте базу данных DB (см. п. 5.2. Создание базы данных).

3) Создайте в текущем каталоге файл JDBCTest.java следующего содержания:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBCTest {
   public static void main(String[] args) {
      try {
         String connUrl = "jdbc:soqol://SOQOL:SOQOL@localhost:2060/DB";
         Connection connection = DriverManager.getConnection(connUrl, "SOQOL", "SOQOL");
         System.out.println("Connection to \"" + connUrl + "\" - Ok");
         connection.close();
      } catch (SQLException e) {
         e.printStackTrace();
      }
      System.exit(0);
   }
}
```

4) Скомпилируйте файл JDBCTest. java командой:

```
javac JDBCTest.java
```

5) После запустите полученное приложение командой:

```
java -cp .;<путь_до_jar>/soqol-jdbc-<версия>-SNAPSHOT.jar JDBCTest
```

В случае успешного соединения будет выведено сообщение:

```
Connection to "jdbc:soqol://SOQOL:SOQOL@localhost:2060/DB" - Ok
```

При возникновении проблем с подключением JDBC-драйвера см. п. 9.2.6. При возникновении проблем с подключением JDBC-драйвера.

Вернуться в оглавление

9.2.4. Логирование в JDBC-драйвере

Для анализа поступающих в JDBC-драйвер СУБД СОКОЛ команд можно использовать его возможности по логированию.

В текущей версии JDBC-драйвер СУБД СОКОЛ содержит логирование текстов SQL-запросов и вызовов отдельных API-методов (создание/закрытие соединения, оператора). Логирование использует пакет java.util.logging.

Для управления логированием нужно при запуске Java-приложения задать путь к файлу конфигурации логирования через системную переменную:

```
-Djava.util.logging.config.file=<имя-файла-конфигурации-логирования>.
```

Простейший пример файла конфигурации логирования:

```
# Global logger settings
handlers=java.util.logging.FileHandler,java.util.logging.ConsoleHandler
.level=INFO

# Filehandler settings
java.util.logging.FileHandler.level=INFO
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
java.util.logging.FileHandler.encoding=UTF-8
java.util.logging.FileHandler.count=10
java.util.logging.FileHandler.append=true
java.util.logging.FileHandler.limit=10000000
java.util.logging.FileHandler.pattern=./jdbc-log.%u.%g.log

java.util.logging.ConsoleHandler.level=OFF
```

Данная конфигурация направляет вывод всех сообщений уровня INFO, WARN и SEVERE в файлы лога с именами jdbc-log.x.x.log. Данная конфигурация выключает вывод в консоль:

(java.util.logging.ConsoleHandler.level=OFF).

ВАЖНО: включение логирования негативно влияет на производительность.

Вернуться в оглавление

9.2.5. Поддерживаемые версии Java и список реализованных функций в JDBC

Поддерживаются версии Java не ниже 8 и 64-разрядные JVM.

Список реализованных функций в текущей версии JDBC-драйвера СУБД СОКОЛ указан в ПРИЛОЖЕНИЕ 2.

Вернуться в оглавление

9.2.6. При возникновении проблем с подключением JDBC-драйвера

При возникновении проблем с подключением следует понимать алгоритм поиска драйвером jni-библиотеки для загрузки:

1) Если при запуске Java-приложения указано свойство Dsoqol.library.path=<полное-имя-jni-библиотеки>, то производится попытка загрузки библиотеки по данному полному имени.

2) При отсутствии свойства −Dsoqol.library.path или отсутствии библиотеки по данному пути, производится попытка загрузки библиотеки из jar-архива драйвера путём копирования из архива во временный каталог (в случае отсутствия данной версии jni-библиотеки в этом временном каталоге). Jni-библиотеки располагаются внутри jar-архива в каталоге jni-lib и подкаталогах для каждой ОС.

3) При отсутствии библиотеки в архиве производится попытка загрузки из путей, указанных в -Djava.library.path или в переменной окружения PATH (Windows), LD_LIBRARY_PATH (Linux). Например, если полное имя библиотеки C:\SOQOL\soqol-jdbc-jni-<версия>.dll, то в переменной PATH должен присутствовать каталог C:\SOQOL.

Вернуться в оглавление

9.3. PHP-драйвер

В текущей версии СУБД СОКОЛ PDO-драйвер собирается для версии PHP 8.1 и поставляется в составе дистрибутива как PHP-расширение в виде файла php_pdo_soqol.<ext>.

Далее описан порядок установки и использования расширения (драйвера), реализующего интерфейс PDO (PHP Data Objects) для СУБД СОКОЛ.

Вернуться в оглавление

9.3.1. Установка

Установка драйвера выполняется стандартным для расширений PHP способом (подробнее описано в документации PHP), а именно:

1) Скопировать файл расширения в каталог, который использует PHP для хранения динамически загружаемых расширений.

2) Добавить в конфигурационный файл PHP строку:

```
extenstion=pdo_soqol
```

Для проверки корректности установки драйвера PDO достаточно выполнить команду:

```
php -r 'phpinfo();'
```

При успешной установке расширения в выводе команды будет присутствовать следующая информация (с точностью до версии):

```
pdo_soqol
PDO Driver for SoQoL => enabled
SoQoL Version => 1.0
```

Вернуться в оглавление

9.3.2. Использование

Подробное руководство по использованию PDO доступно на официальном сайте языка программирования PHP.

Далее будут рассмотрены особенности реализации интерфейса PDO для СУБД Сокол.

Вернуться в оглавление

9.3.2.1. Класс PDO

Данный класс представляет соединение между PHP и сервером базы данных.

Далее в таблице 2 указаны методы, которые имеют в реализации специализацию СУБД СОКОЛ. Для методов, у которых явно не указаны особенности реализации — реализация стандартная (согласно официальной документации PDO).

Таблица 2 — Методы со специализацией СУБД СОКОЛ

Метод	Описание/особенности реализации
PDO::__construct	Конструктор класса. Формат строки подключения (DSN) описан далее (см. 9.3.2.2 Формат строки подключения (DSN))
PDO::getAttribute	Получение атрибута соединения с базой данных. Поддерживаемые атрибуты: - PDO_ATTR_EMULATE_PREPARES - всегда false; - PDO_ATTR_CLIENT_VERSION - номер версии клиентского интерфейса СУБД СОКОЛ

Порядок работы с классом PDO и вызов его методов определён в официальной документации языка программирования PHP.

Вернуться в оглавление

9.3.2.2. Формат строки подключения (DSN)

Конструктор класса PDO использует строку соединения (DSN), которая имеет специфический формат для каждого PDO драйвера.

У драйвера PDO для СУБД Сокол данная строка подключения состоит из параметров (разделенных пробелом или точкой с запятой), указанных далее в таблице 3.

Таблица 3 — Параметры строки подключения для драйвера PDO СУБД Сокол

Имя параметра	Описание	Значение по умолчанию
soqol:	Префикс DSN, который сообщает о необходимости использования драйвера СУБД СОКОЛ	Нет
host	Адрес сервера базы данных, к которому производится подключение	localhost
port	Порт, используемый для подключения к серверу базы данных	2060
dbname	Имя базы данных, к которой производится подключение	Нет
charset	Имя кодировки строковых данных, передаваемых клиентским приложением на сервер	UTF-8
user	Имя пользователя, под которым производится подключение к базе данных. Параметр используется только в том случае, если не задан аргумент $username конструктора класса PDO	Нет
password	Пароль пользователя, под которым производится подключение к базе данных. Параметр используется только в том случае, если не задан аргумент $password конструктора класса PDO.	Нет

Все элементы DSN, за исключением префикса, могут быть указаны в произвольном порядке. Значения параметров указываются в формате:

```
<имя параметра>=<значение параметра>
```

Пример строки подключения:

```
soqol:host=localhost;port=2060;dbname=DB
```

Вернуться в оглавление

10. ВИЗУАЛЬНЫЕ ИНСТРУМЕНТЫ ДЛЯ РАБОТЫ С СУБД СОКОЛ

СУБД ЛИНТЕР СОКОЛ поддерживает стандартный интерфейс JDBC. Это позволяет применять внешние инструменты для управления и разработки БД, совместимые с JDBC-интерфейсом.

Вернуться в оглавление

10.1. DBeaver

Рассмотрим процедуру подключения JDBC-драйвера на примере одного из известных инструментов — DBeaver. Это клиентское приложение для исполнения SQL и инструмент администрирования баз данных.

Для начала скачайте дистрибутив DBeaver для вашей ОС с официального сайта (https://dbeaver.io/download/), например, версию Community.

Для установки DBeaver запустите исполняемый файл установщика и следуйте инструкциям на экране.

После установки запустите DBeaver и выполните процедуру подключения JDBC-драйвера СУБД СОКОЛ к DBeaver в следующем порядке:

1) В меню главного окна откройте пункт «База данных» -> «Управление драйверами» (рис. 3).

2) В открывшемся окне «Менеджер Драйверов» нажмите кнопку «Новый» (рис. 4).

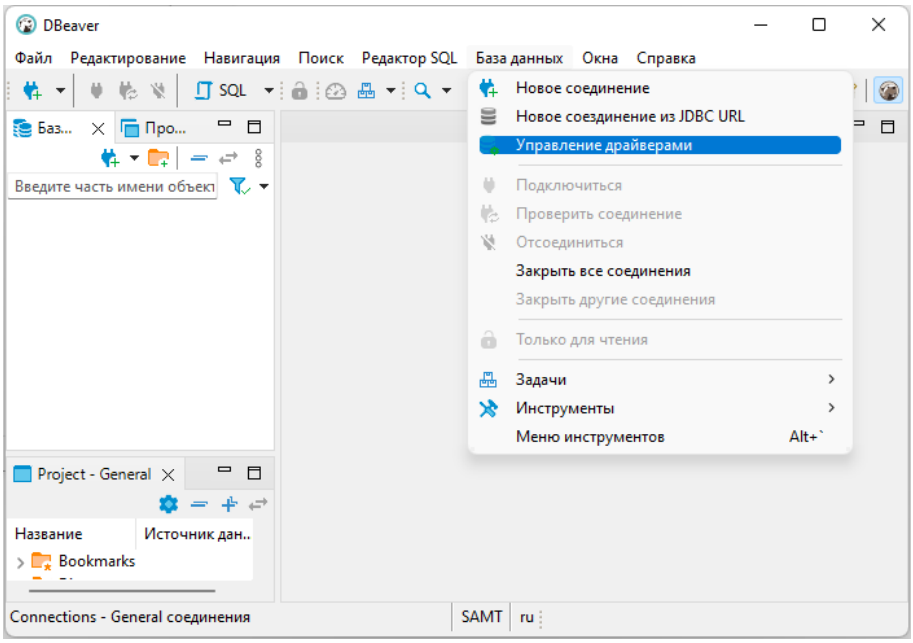


Рисунок 3 — «Управление драйверами»

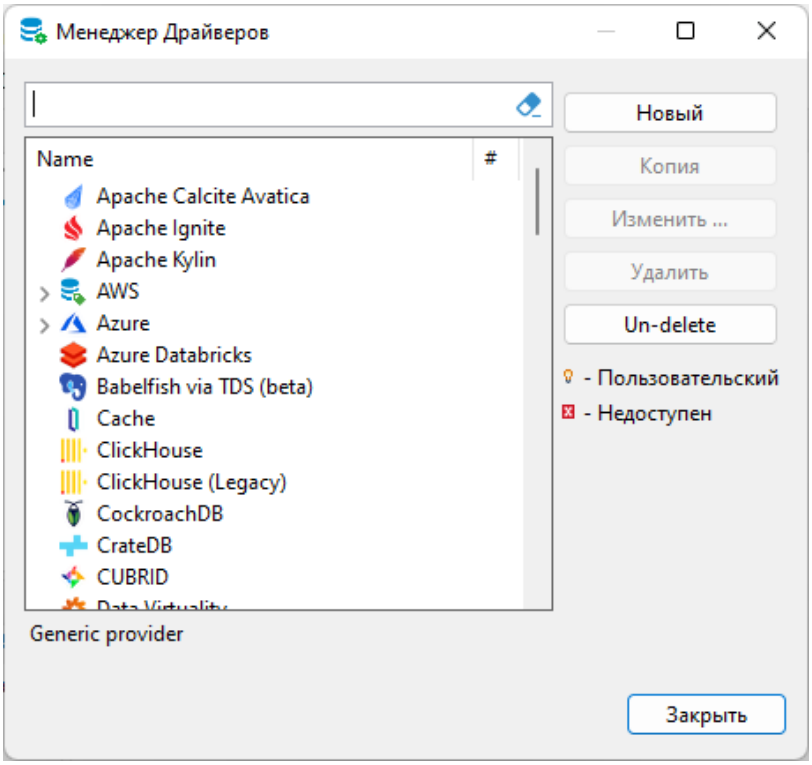


Рисунок 4 — Окно «Менеджер Драйверов»

3) В окне создания драйвера на вкладке «Настройки» (рис. 5):

– введите имя драйвера (произвольно, например, «SOQOL DB»), которое вы потом будете использовать при создании соединения с сервером;

– выберите тип драйвера «Generic»;

– введите имя класса ru.relex.soqol.jdbc.Driver.

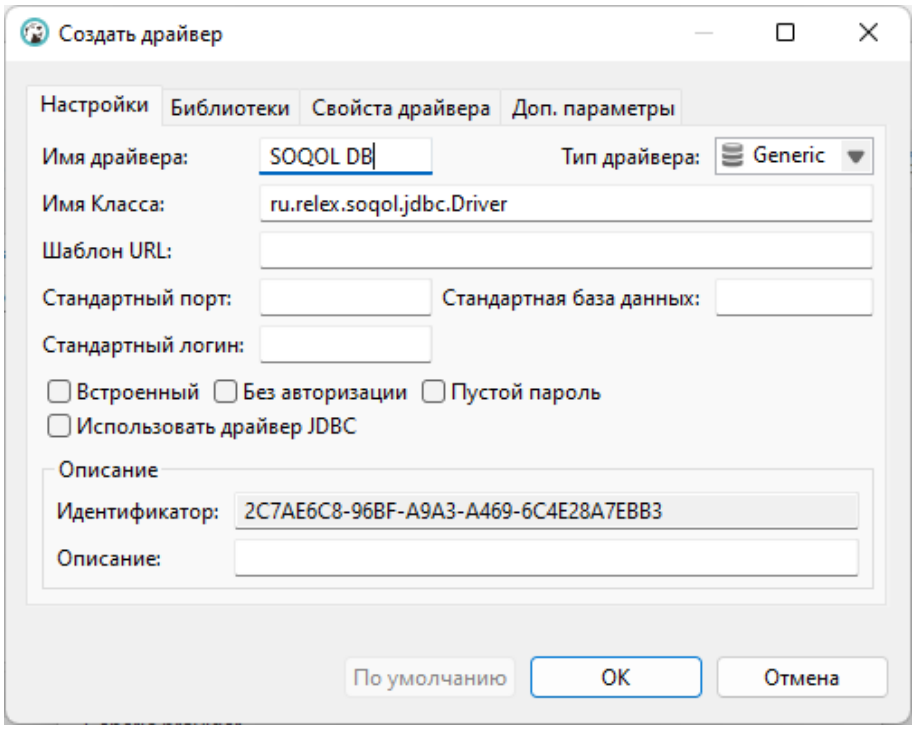


Рисунок 5 — Окно создания драйвера, настройки

4) В этом же окне на вкладке «Библиотеки» нажмите «Добавить файл» и выберите jar-файл драйвера soqol-jdbc-<номер_версии>.jar (рис. 6).

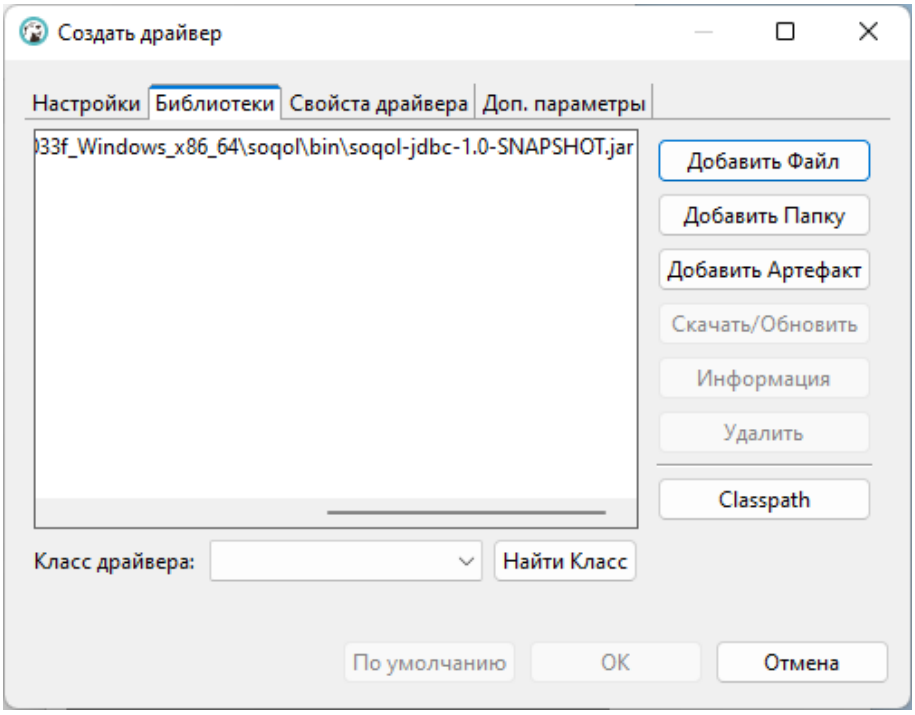


Рисунок 6 — Окно создания драйвера, «Библиотеки»

5) Сохраните введенные данные о регистрации драйвера нажатием кнопки «ОК».

В окне «Менеджер Драйверов» видно, что в списке драйверов появился элемент с именем «SOQOL DB» (рис. 7).

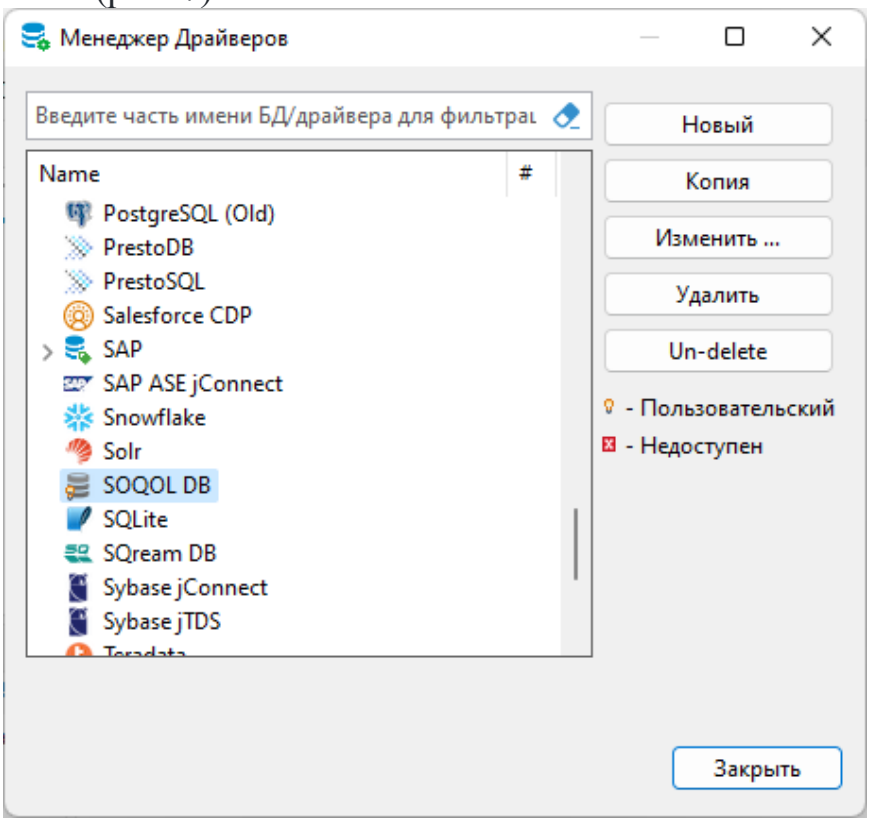


Рисунок 7 — Окно «Менеджер Драйверов»

Создание нового соединения с источником данных. Для этого выполните действия в следующем порядке:

1) В меню главного окна откройте пункт «База данных» -> «Новое соединение» (рис. 8).

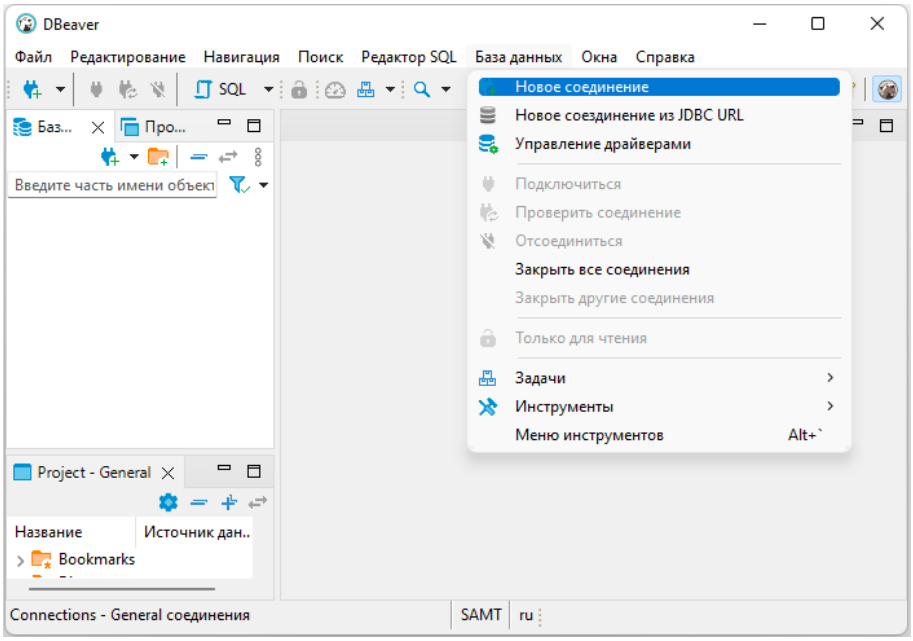


Рисунок 8 — Окно создания нового соединения

2) В открывшемся окне выставьте фильтр «All» и выберите ранее зарегистрированный драйвер «SOQOL DB» (рис. 9).

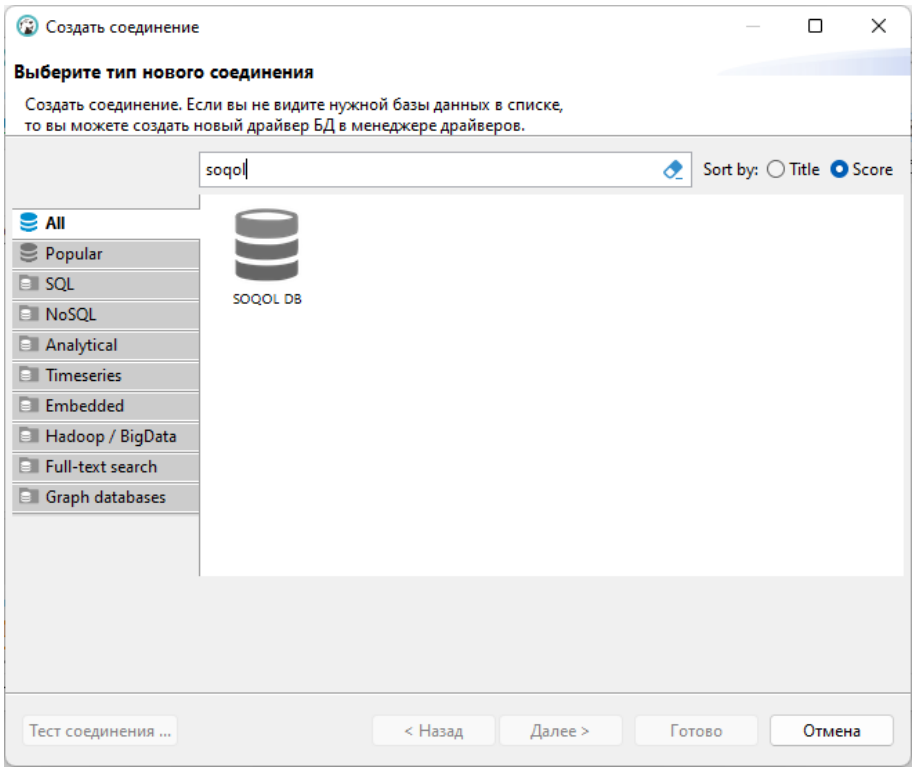


Рисунок 9 — Окно для поиска зарегистрированного драйвера

3) На вкладке «Главное» введите (рис. 10):

– в поле «JDBC URL» строку соединения, например, «jdbc:soqol://SOQOL:SOQOL@localhost:2060/<имя_базы_данных>» для подключения к **существующей** базе данных (на рис. 6 это имя «DB») или «jdbc:soqol://SOQOL:SOQOL@localhost:2060/» для подключения к **сервису** управления базами данных;

– в поля «Пользователь» и «Пароль» соответственно имя пользователя и его пароль (например, «SOQOL», «SOQOL»).

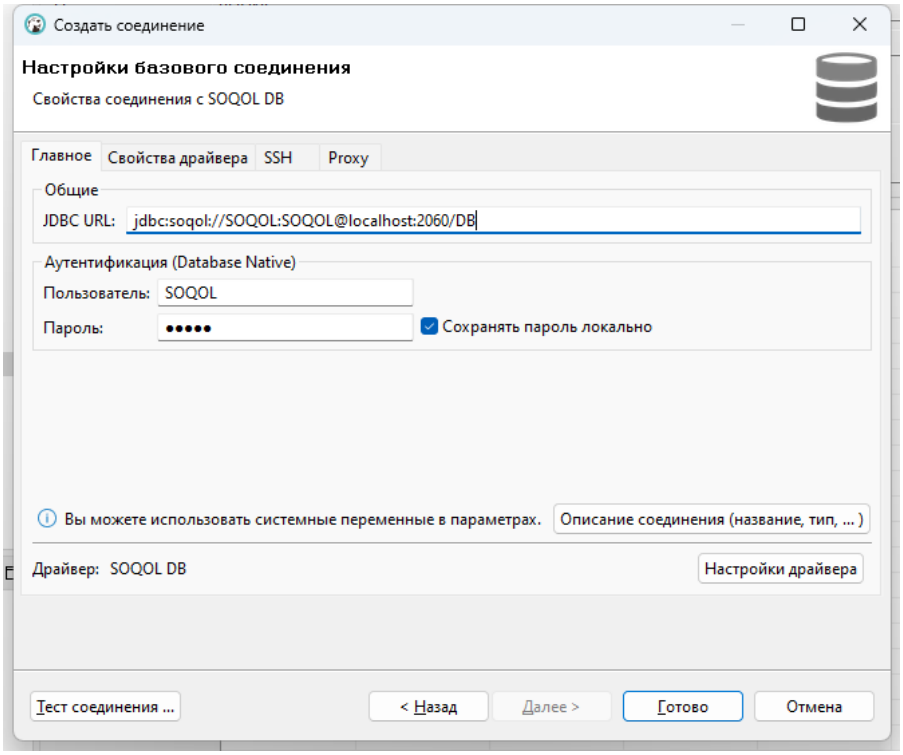


Рисунок 10 — Окно настроек базового соединения

Для сохранения введённых данных нажмите кнопку «Готово».

4) Во вкладке подключенных баз данных главного окна DBeaver, появилось соединение с базой данных DB (рис. 11).

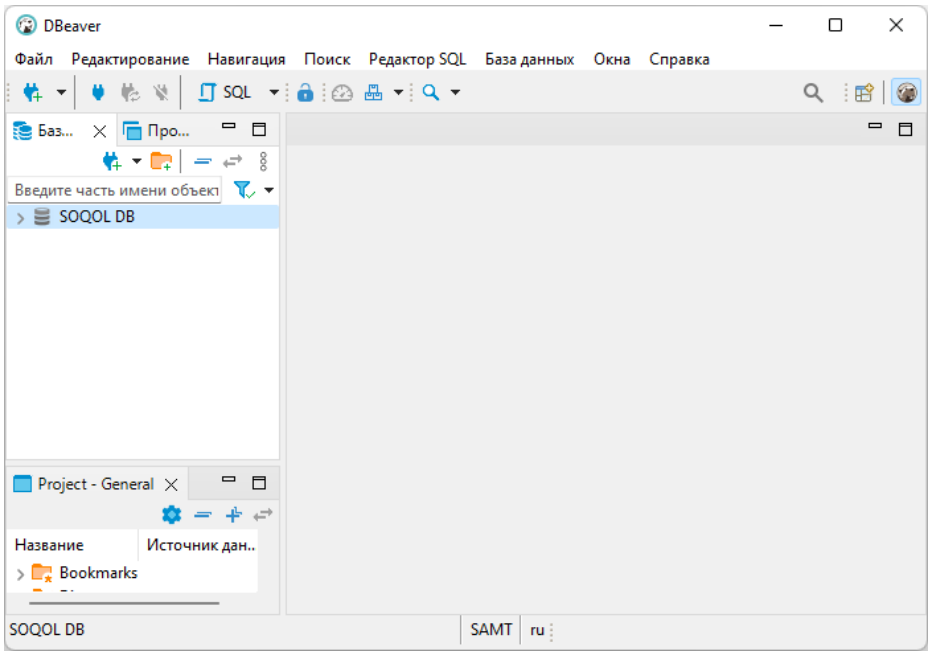


Рисунок 11 — Главное окно DBeaver с созданным соединением с БД

Вернуться в оглавление

10.2. Другие инструменты

Процедура подключения JDBC-драйвера для использования с внешними инструментами управления СУБД (например, DBeaver, DataGrip, Execute Query, IDE Eclipse и Idea и т.д.) заключается в основном в:

– выборе jar-файла драйвера;

– указании имени Java-класса драйвера СУБД СОКОЛ (там, где это требуется) ru.relex.soqol.jdbc.Driver;

– указании строки подключения.

Вернуться в оглавление

11. РАБОТА С СУЩНОСТЬЮ БД

Архитектура СУБД СОКОЛ позволяет пользователю подключаться как к сервису управления базами данных, так и к базе данных.

База данных представляет собой объект, которым можно манипулировать: создавать, запускать, модифицировать, удалять и т.д.

Далее подробнее рассмотрим реализованные в СУБД СОКОЛ команды для выполнения манипуляций с базой данных.

Вернуться в оглавление

11.1. CREATE DATABASE

Команда создаёт базу данных, регистрирует и запускает её обслуживание в сервисе управления базами данных.

Перед созданием базы данных важно убедиться, что на диске достаточно свободного пространства (не менее 1 ГБ).

Команда выполняется из подключения к сервису управления базами данных. Выполнить создание базы данных может пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды создания базы данных:

```
CREATE DATABASE <имя_базы_данных> [ON '<имя_каталога>'] [WITH <опции>];

<опции> ::= <имя_опции> = <значение_опции>[, <имя_опции> = <значение_опции>...]
```

Где:

– <имя_базы_данных> — имя создаваемой базы данных. Имя базы данных должно быть уникально среди других БД, находящихся под управлением сервиса, и задано в соответствии с правилами именования БД, указанными далее в таблице 4;

– ON '<имя_каталога>' — конструкция позволяет задать имя каталога для создаваемой базы данных. Имя каталога БД должно быть уникальных среди имён других каталогов в родительском каталоге.

Конструкция не является обязательной. Если не указать её в команде, то в текущем каталоге будет создан каталог БД, одноимённый с создаваемой базой данных.

База данных может располагаться не только в текущем каталоге, но в любом каталоге и на любом диске. Для этого при создании базы данных в качестве значения параметра '<имя_каталога>' нужно указать полный путь с именем каталога БД.

Важно: максимальная длина пути корневого каталога базы данных равна 220 байт. Ограничение одинаково для ОС Windows и Linux.

– WITH <опции> — конструкция позволяет указать одну или несколько опций для дополнительной настройки создаваемой базы данных. В конструкции имя опции указывается без кавычек и апострофов.

Конструкция WITH <опции> не является обязательной. Если её не указать в команде создания БД, то новая база данных создаётся со значениями опций по умолчанию.

В текущей версии реализованы опции, указанные далее в таблице 5.

Новые значения опций, кроме значения опции database.autostart, указанные при создании базы данных, начнут действовать сразу после успешного выполнения команды. Новое значение опции database.autostart начнёт действовать при перезапуске сервера.

Таблица 4 — Правила именования БД

Правило	Описание
Указание имени без кавычек	Имя БД вводится без кавычек и апострофов.
Регистрозависимость	Символы имени регистронезависимые — указанные в имени БД символы приводятся к верхнему регистру.
Допустимые символы	Латинские буквы, цифры, знак подчеркивания (_). Никакие другие символы не являются допустимыми.
Первый символ имени	Первым символом в имени должна быть буква.
Имена исключения для БД и её каталога	Для обеспечения совместимости и переноса БД под WINDOWS/LINUX в качестве имени БД (и её каталога) нежелательно указывать имена, зарезервированные ОС (например, CON, PRN, AUX, NUL, COM1 и т.д.) и запрещённые знаки (все, кроме указанные в «Допустимых символах»).
Максимальная длина имени базы данных	Максимальная длина имени базы данных равна 64 байтам.

Таблица 5 — Опции базы данных

Описание	Имя опции	Значения
Максимальное число страниц предварительной выборки	storage.prefetch_count	По умолчанию: 32 Минимальное: 0 Максимальное: 256
Синхронизация диска для входа в базу данных при фиксации транзакции	database.sync_commit	По умолчанию: on Также допустимо: off При отключении синхронизации не гарантируется сохранение результатов зафиксированных транзакций, гарантируется консистентность базы данных в точке последней синхронизации
Максимальное время между автоматическим созданием контрольных точек	database.incremental_checkpoint.timeout	По умолчанию: 1800 Минимальное: 0 Максимальное: не ограничено
Размер данных журнала WAL, при достижении которого автоматически должна быть создана контрольная точка	database.incremental_checkpoint.log_limit	По умолчанию: 0 (обозначает, что опция отключена) Минимальное: 0 Максимальное: не ограничено
Общий размер буфера записи основного журнала	database.dblog.buffer_size	По умолчанию: 64MB Минимальное: 2MB Максимальное: 8GB Значение опции не может быть меньше 2 * database.dblog.flush_cluster_size
Размер кэшируемой области буфера записи основного журнала	database.dblog.cache_size	По умолчанию: 32MB Минимальное: 1MB Максимальное: 4GB Значение опции не может быть меньше database.dblog.flush_cluster_size и больше половины database.dblog.buffer_size
Размер одного файла логирования (write)	database.dblog.file_size	По умолчанию: 1GB Минимальное: 16MB Значение опции может быть больше или равно 2 * database.dblog.flush_cluster_size Указанное значение должно быть кратно 8KB (размеру дисковой страницы журнала). Иначе оно должно быть округлено в большую сторону.

Описание	Имя опции	Значения
		Значение задаётся при создании БД и не доступно для корректировки
Размер кластера для записи на диск	database.dblog.flush_cluster_size	По умолчанию: 0 (обозначает значение по умолчанию 1280 KB) Минимальное: 128KB Максимальное: 32MB
Общий размер буфера записи исторического журнала	database.histlog.buffer_size	По умолчанию: 64MB Минимальное: 2MB Максимальное: 8GB Значение опции не может быть меньше 2 * database.histlog.flush_cluster_size
Размер кэшируемой области буфера записи исторического журнала	database.histlog.cache_size	По умолчанию: 32MB Минимальное: 1MB Максимальное: 4GB Значение опции не может быть меньше database.histlog.flush_cluster_size и больше половины database.dblog.buffer_size
Размер одного файла журнала истории	database.histlog.file_size	По умолчанию: 128MB Минимальное: 16MB Значение опции может быть больше или равно 2 * database.dblog.flush_cluster_size Указанное значение должно быть кратно 8KB (размеру дисковой страницы журнала). Иначе оно должно быть округлено в большую сторону. **Значение задаётся при создании БД и не доступно для корректировки**
Минимальное число файлов histlog в кольцевом журнале	database.histlog.min_file_ring	По умолчанию: 2 Минимальное: 1 Максимальное: 1024
Максимальное число файлов histlog в кольцевом журнале	database.histlog.max_file_ring	По умолчанию: 4 Минимальное: 1 Максимальное: 1024
Размер кластера буфера, по достижении которого изменения могут быть сохранены на диск	database.histlog.flush_cluster_size	По умолчанию: 0 (обозначает значение по умолчанию 1280KB) Минимальное: 128KB Максимальное: 32MB
Автоматический сбор статистики по каждой таблице БД.	database.translate.auto_statistics	По умолчанию: on Также допустимо: off

Описание	Имя опции	Значения
Собранная статистика используется для оптимизации исполняемых запросов		
Имя кодировки строковых данных, которая будет применима по умолчанию к строковым данным всех объектов БД	database.charset	По умолчанию: UTF-8 Также допустимо: CP1251 **Значение задаётся при создании БД и не доступно для корректировки.**
Автоматический запуск базы данных при запуске сервера	database.autostart	По умолчанию: on Также допустимо: off База данных со значением опции 'on' запускается автоматически при запуске сервера
Фоновая очистка страниц для данной базы	database.press_table.enable	По умолчанию: on Также допустимо: off В базе данных со значением опции 'on' включена фоновая очистка страниц. Для высоконагруженных систем администратор базы может отключить фоновую очистку для конкретной базы данных, установив значение опции 'off'
Значение часового пояса базы данных в соответствии со стандартом UTC (подробнее о часовых поясах см. п. 12.1.1. Часовые пояса)[CV]	database.time_zone	Формат значения: '[+|-]HH:MM', где: - HH — смещение по часам относительно UTC; - MM — смещение по минутам относительно UTC; - [+|-] — знак обозначает смещение относительно UTC: "+" обозначает – восточнее, "-" – западнее. По умолчанию применяется знак "+". По умолчанию опция получает значение смещения относительно UTC локального времени компьютера, на котором запущен сервер СУБД. Допустимые значения: - HH диапазон от -14 до +14; - MM диапазон от 0 до +59

Важно: кодировка данных, указанная при создании БД, будет применима к текстовым данным объектов БД. Правило сравнения строковых данных

устанавливается при создании таблиц БД в соответствии со следующим приоритетом (в порядке убывания приоритета):

1) Правило, заданное для текстового поля при создании таблицы (подробнее см. п. 12.8.1.1. CREATE TABLE).

2) Правило, указанное на период сессии (подробнее см. п. 11.5. ALTER SESSION).

3) Правило, установленное по умолчанию при создании БД - BINARY.

Допустимые правила сравнения для кодировок см. в Приложении 3.

Пример применения команды CREATE DATABASE

Рассмотрим для примера различные варианты создания базы данных: с базовыми настройками в текущем каталоге, не в текущем каталоге, с заданием нового значения опций из таблицы 5.

Для создания базы данных в текущем каталоге с настройками по умолчанию, достаточно указать в команде только имя создаваемой БД:

```
create database PHONE;
```

В результате выполнения команды в текущем каталоге создан каталог базы данных, одноимённый с ней: './PHONE'.

Можно создать базу данных не только в текущем каталоге, но и на другом диске. Это позволит распределить нагрузку на диски и повысить эффективность их работы. Для этого при создании базы данных указываем полный путь с именем каталога БД:

```
create database WORKERS on 'D:\Databases\WORKERS';
```

В результате выполнения команды база данных создана в каталоге WORKERS на диске D:.

Допустим, необходимо создать базу данных для хранения архивных данных, которая будет редко использоваться. И для экономии системных ресурсов было решено запускать её явно, по требованию. Для этого при создании БД необходимо отключить автозапуск, указав соответствующее значение опции database.autostart:

```
create database CITY with database.autostart='off';
```

После успешного выполнения команды будет создана база данных, для работы с которой при перезапуске сервера её необходимо будет запустить (подробнее см. п.11.2. STARTUP DATABASE).

Вернуться в оглавление

11.2. STARTUP DATABASE

Команда запускает существующую базу данных, делая её доступной для пользователей.

Команда выполняется из подключения к сервису управления базами данных. Выполнить запуск базы данных может пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды запуска базы данных:

```
STARTUP DATABASE <имя_базы_данных>;
```

Где <имя_базы_данных> — имя базы данных, которую необходимо запустить.

После успешно выполненной команды любой пользователь базы данных с соответствующими привилегиями может подключиться к ней и работать с объектами БД в рамках своих привилегий (подробнее о привилегиях и ролях см. п. 12.5.7 Привилегии (PRIV) и роли (ROLE)).

Пример применения команды STARTUP DATABASE

В примере раздела 11.1 CREATE DATABASE была создана база данных CITY с отключенным автозапуском.

Базу данных CITY необходимо запустить перед подключением пользователей к ней и дальнейшей работой:

```
startup database CITY;
```

После успешного выполнения команды база данных запущена и готова к работе.

Вернуться в оглавление

11.3. ALTER DATABASE

Команда изменения настроек существующей базы данных.

Команда может быть выполнена как из соединения с сервисом (по отношению к любой зарегистрированной в сервисе базе данных — запущенной или нет), так и из соединения с какой-либо базой данных (по отношению к этой базе). Выполнить изменение настроек существующей базы данных может:

– по отношению к БД пользователь этой базы данных с привилегией DATABASE ADMIN;

– по отношению к любой БД пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды изменения настроек базы данных:

```
ALTER DATABASE [<имя_базы_данных>] <настройка>;

<настройка> ::= SET <имя_опции> = <значение_опции>
        | checkpoint
```

Где:

– <имя_базы_данных> — имя базы данных, настройки которой необходимо изменить.

Имя базы данных указывать обязательно, если команда изменения настроек этой БД выполняется из подключения к сервису. Иначе выполнение команды завершится ошибкой.

При выполнении команды изменения настроек БД из подключения к этой базе данных, её имя указывать не обязательно;

– SET <имя_опции> = <значение_опции> — конструкция, позволяющая указать опцию и её новое значение. Имя опции указывается без кавычек и апострофов;

– checkpoint — указание этого ключевого слова позволяет создать контрольную точку на момент выполнения команды.

Подробнее о контрольных точках будет написано в подразделе об архитектуре (см. 3. Архитектурные аспекты).

После успешного выполнения команды:

1) В случае изменения значений опций БД — новые значения всех опций, кроме database.translate.auto_statistics, будут действительны после перезапуска базы данных. Новые значения опции database.translate.auto_statistics будут действительны сразу после успешного выполнения команды.

2) В случае явного создания контрольной точки — создана новая контрольная точка, при этом автоматическое создание контрольных точек не отключается. Следующая контрольная точка в автоматическом режиме будет создана с учётом создания (явным или неявным способом) последней контрольной точки.

Команду изменения настроек существующей базы данных допустимо выполнять сколько угодно раз.

Подробнее об опциях СУБД СОКОЛ см. п. 11.1 CREATE DATABASE.

Пример применения команды ALTER DATABASE

Ранее для примера п. 11.1 CREATE DATABASE была создана база данных CITY с отключенным автозапуском при запуске сервера.

Допустим, эта база данных стала использоваться часто и для удобства необходимо, чтобы она автоматически запускалась при каждом запуске сервера.

Для этого нужно включить автозапуск, изменив значение опции database.autostart:

```
alter database CITY set database.autostart='on';
```

В результате выполненной команды опция автозапуска БД будет включена. Теперь при всех последующих запусках сервера база данных будет запускаться автоматически.

Вернуться в оглавление

11.3.1. ALTER DATABASE KILL SESSION

Команда предназначена для прерывания существующей сессии пользователя базы данных.

Команда выполняется по отношению к любой существующей сессии базы данных пользователем этой базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды прерывания существующей сессии:

```
ALTER DATABASE KILL SESSION <номер_сессии> [IMMEDIATE];
```

Где:

– <номер_сессии> — номер сессии пользователя, который можно получить из системного представления sys._vsessions.

Если указан номер несуществующей сессии, выполнение команды будет завершено с ошибкой;

– IMMEDIATE — указывает на необходимость вернуть сообщение о старте выполнения или невозможности выполнения команды без ожидания завершения сессии в течение фиксированного лимита времени (1 минута).

При отсутствии в команде IMMEDIATE отправленная команда будет ожидать завершения сессии в течение фиксированного лимита времени и только при успешном завершении сессии вернёт сообщение об успешном выполнении команды. В ином случае, если сессия в течение 1 минуты не завершилась, выполнение команды будет завершено с ошибкой.

Указание слова IMMEDIATE обязательно в случае прерывания пользователем собственной сессии. В ином случае выполнение команды будет завершено с ошибкой.

Если в момент отправки запроса на прерывание указанной сессии у пользователя есть незавершённые транзакции, то они будут принудительно завершены системой с откатом всех выполненных в транзакциях изменений.

Вернуться в оглавление

11.4. ALTER SERVICE

Параметры сервиса СУБД СОКОЛ можно скорректировать в конфигурационном файле soqol_config.yml или командой ALTER SERVICE.

Команда выполняется из подключения к сервису управления базами данных. Выполнить изменение параметров сервиса может пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды изменения настроек сервиса:

```
ALTER SERVICE SET <имя_параметра> = <значение_параметра>;
```

Где SET <имя_параметра> = <значение_параметра> — конструкция, позволяющая указать параметр конфигурации и его новое значение.

Имя параметра конфигурации указывается без кавычек и апострофов.

Подробнее имена и описание параметров конфигурации см. в п. 7 Конфигурационный файл soqol_config.yml

После выполнения команды указанные значения параметров записываются в конфигурационный файл, а сервер продолжает работать с прежними настройками. Новые настройки будут применены после перезапуска сервера.

Значения всех текущих параметров сервиса доступны администратору сервиса баз данных в системном представлении sys._vservice_options:

```
select name, value from sys._vservice_options;
```

На запрос будут получены значения параметров сервиса управления базами данных, например:

```
 NAME                    |  VALUE
--------------------------------+------------
service.listen             | 0.0.0.0:2060
service.max_connections      | 100
service.memory               | 0
service.cores              | 0
service.coroutine_pool       | 0
service.scheduler_timeslice   | 10
service.data_dir           |
trace.components             |
trace.buffer_size          | 16MB
```

```
trace.file_size            | 128MB
trace.affinity             | 0
storage.page_write_batch   | 16
storage.read_queue_depth   | 0
storage.write_queue_depth  | 0
storage.batch_queue_depth  | 0
storage.press_table.task_count| 0
storage.press_table.page_count| 128
translate.print_plan       | off
translate.codegen          | 1
translate.print_link       | 0
```

Изменение значений параметров допустимо для оптимизации работы СУБД при понимании их воздействия на систему.

Пример применения команды ALTER SERVICE

Допустим, для работы сервиса был выделен объем оперативной памяти 4Gb, что было видно из запроса к системному представлению sys._vservice_options:

```
select name, value from sys._vservice_options;


 NAME                 |  VALUE
----------------------------+------------
service.listen          | tcp://0.0.0.0:2060
service.max_connections   | 100
service.memory          | 4194304KB
service.cores           | 0
...
```

Позже к сервису были подключены новые БД и для сохранения производительности потребовалось увеличить объем выделенной оперативной памяти для сервиса до 8Gb.

Для этого из подключения к сервису управления базами данных необходимо изменить значение параметра service.memory на необходимое (можно указать значение в байтах или в виде литерала '8Gb'):

```
alter service set service.memory = 8589934592
```

В результате выполнения команды указанное значение параметра записывается в конфигурационный файл, но сервер пока продолжает работать на прежних настройках. Для применения настроек необходимо перезапустить сервер (как приложение, предоставляющее сервис).

Вернуться в оглавление

11.5. ALTER SESSION

Команда для изменения настроек соединения пользователя с базой данных.

Команда выполняется из подключения к базе данных, настройки которого будут изменяться. Указанные параметры действуют в период сессии пользователя. Выполнять команду ALTER SESSION в рамках сессии можно множество раз. При новом подключении (в новой сессии) действительны настройки по умолчанию.

Изменить настройки соединения с базой данных может любой пользователь БД с системной привилегией CREATE SESSION.

Синтаксис команды изменения настроек соединения:

```
ALTER SESSION SET <имя_опции> = <значение_опции>;
```

Где:

– SET <имя_опции> = <значение_опции> — конструкция, позволяющая указать опцию и её новое значение в рамках сессии (имя опции указывается без кавычек и апострофов).

Допустимые опции соединения указаны далее в таблице 6.

Таблица 6 — Опции соединения

Описание	Имя опции	Значения
Режим управления транзакциями - автокоммит	autocommit	По умолчанию: on Также допустимо: off Изменение значения опции влияет на режим управления транзакциями в текущей сессии: on — включен режим автоматического управления транзакциями; off — включен режим явного управления транзакциями. Если команда выполняется в рамках незавершённой транзакции, то: - команда будет выполнена без принудительного завершения и / или отката текущей транзакции; - новый режим начнёт действовать с новой транзакции, следующей после завершения текущей.

Описание	Имя опции	Значения
		Подробнее о транзакциях см. 12.6 Работа с транзакциями.
Правило сравнения строковых данных по умолчанию. Это правило будет назначено столбцам таблиц, которые будут создаваться во время сеанса (в любой схеме), если правило явно не указано для столбца при создании таблицы. Установленное правило сравнения по умолчанию для сеанса не будет действовать на таблицы, создаваемые в других сеансах.	default_collation	По умолчанию: BINARY Также допустимо: см. ПРИЛОЖЕНИЕ 3. Указанное правило сравнения будет применимо к данным таблиц при выполнении соответствующих операций (например, при упорядочении строк).
Значение для локального часового пояса, устанавливаемого в рамках сессии и в соответствии со стандартом UTC (подробнее о часовых поясах см. п. 12.1.1. Часовые пояса)[CV]	time_zone	Формат значения: '[+|-]HH:MM', где: HH — смещение по часам относительно UTC; MM — смещение по минутам относительно UTC; [+|-] — знак обозначает смещение относительно UTC: "+" обозначает — восточнее, "-" — западнее. По умолчанию применяется знак "+". По умолчанию значение опции устанавливается в соответствии со значением опции time_zone базы данных. Допустимые значения: HH диапазон от -14 до +14; MM диапазон от 0 до +59
Гарантированный размер памяти сессии	memory	По умолчанию: гарантированная память сессии рассчитывается исходя из доступной ОП и количества потоков исполнения, но не менее 800KB на сессию. Допустимые значения: - 0 — будут использованы настройки сервиса для сессии. Используется для возврата к начальным значениям; - при указании значения < 800KB будет использовано 800KB Настройка действует до конца сессии. При старте сессии используется значение настройки

Описание	Имя опции	Значения
		сервиса гарантированной памяти сессии.

Пример применения команды ALTER SESSION

Базы данных в СУБД СОКОЛ создаются с настройками по умолчанию, если в команде CREATE DATABASE не были указаны иные значения опций.

В примере раздела 11.1 CREATE DATABASE была создана БД PHONE с настройками по умолчанию, где значением опции autocommit по умолчанию является 'on'.

Допустим, для работы в данной сессии необходимо ручное управление началом и завершением транзакций. Для этого сначала необходимо изменить значение опции autocommit:

```
alter session set autocommit = 'off';
```

После успешного выполнения команды можно управлять транзакциями в ручном режиме, и для фиксации изменений необходимо послать команду:

```
commit;
```

или для отмены всех изменений в транзакции:

```
rollback;
```

Вернуться в оглавление

11.6. SHUTDOWN DATABASE

Команда останова базы данных.

Команда SHUTDOWN DATABASE выполняется из подключения к сервису управления базами данных. Выполнить останов базы данных может только пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды останова базы данных:

```
SHUTDOWN DATABASE <имя_базы_данных>;
```

Где <имя_базы_данных> — имя базы данных, останов которой необходимо выполнить.

Если во время выполнения команды есть незаконченные транзакции, то они завершаются с откатом. При попытке пользователя выполнить какую-либо команду будет возвращена ошибка.

Пример применения команды SHUTDOWN DATABASE

Допустим, с БД CITY не планируется работать длительное время. Было принято решение отправить её в архив.

Для этого сначала необходимо выполнить останов базы данных:

```
shutdown database CITY;
```

После успешно выполненной команды можно выводить базу данных CITY из-под управления сервиса командой DETACH DATABASE (подробнее см. п. 11.7. DETACH DATABASE) и далее архивировать её каталог (подробнее см. п. 15.1. Копирование / архивирование базы данных и её последующее восстановление).

Вернуться в оглавление

11.7. DETACH DATABASE

Команда дерегистрации базы данных в сервисе управления, в результате выполнения которой сервис удаляет информацию об отсоединяемой БД в собственной базе данных.

Команда выполняется из подключения к сервису управления БД по отношению к ранее остановленной базе данных (подробнее см. п. 11.6. SHUTDOWN DATABASE).

Выполнить дерегистрацию базы данных может только пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды вывода базы данных из под управления сервиса:

```
DETACH DATABASE <имя_базы_данных>;
```

Где <имя_базы_данных> — имя базы данных, которую необходимо дерегистрировать в сервисе управления.

Если команда запущена для базы данных, останов которой не был выполнен, то будет возвращена ошибка.

Пример применения команды DETACH DATABASE

В примере раздела 11.6. SHUTDOWN DATABASE была остановлена база данных CITY, которую планируется отправить в архив.

Перед архивированием базы данных CITY необходимо вывести её из-под управления сервиса:

```
detach database CITY;
```

В результате выполнения команды из базы данных сервиса удалена информация об этой БД.

Далее можно архивировать каталог БД (подробнее см. п. 15.1. Копирование / архивирование базы данных и её последующее восстановление) или вернуть БД обратно под управление сервиса (подробнее см. п. 11.8. ATTACH DATABASE)

Вернуться в оглавление

11.8. ATTACH DATABASE

Команда регистрации базы данных в сервисе управления базами данных, в результате выполнения которой сервис прописывает информацию о присоединяемой БД в собственной базе данных. Это обеспечивает возможность дальнейшего запуска базы данных.

Команда выполняется из подключения к сервису управления базами данных. Выполнить регистрацию базы данных может пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды присоединения базы данных к сервису:

```
ATTACH DATABASE <имя_базы_данных> FROM '<каталог_базы_данных>';
```

Где:

– <имя_базы_данных> — имя базы данных, под которым её необходимо зарегистрировать в сервисе управления базами данных;

– <каталог_базы_данных> — путь к каталогу базы данных.

Важно: максимальная длина пути корневого каталога базы данных равна 220 байт. Ограничение одинаково для ОС Windows и Linux.

Пример применения команды ATTACH DATABASE

В примере раздела 11.7. DETACH DATABASE была дерегистрирована из управления сервиса база данных CITY, каталог которой далее отправили в архив (подробнее см. п. 15.1. Копирование / архивирование базы данных и её последующее восстановление).

Спустя некоторое время была выполнена распаковка архива каталога БД CITY на диск D: в каталоге 'Databases'. Для регистрации базы данных CITY в сервисе управления базами данных необходимо выполнить команду ATTACH DATABASE с указанием полного пути к каталогу БД:

```
attach database CITY from 'D:\Databases\CITY';
```

В результате выполнения команды сервис прописал информацию о БД CITY в собственной базе данных.

Далее для работы с БД CITY её необходимо запустить командой STARTUP DATABASE (подробнее см. п. 11.2. STARTUP DATABASE).

Вернуться в оглавление

11.9. DROP DATABASE

Команда удаления базы данных.

Команда DROP DATABASE выполняется из подключения к сервису управления базами данных только после останова БД, иначе будет возвращена ошибка. При выполнении команды сервис удаляет информацию об удаляемой БД, а также удаляется и сам каталог БД.

Выполнить удаление базы данных может пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления базы данных:

```
DROP DATABASE <имя_базы_данных>;
```

Где <имя_базы_данных> — имя базы данных, которую необходимо удалить.

Выполнение команды DROP DATABASE имеет тот же результат, что и последовательность действий:

```
DETACH DATABASE <имя_базы_данных>;
+
удаление каталога БД в файловой системе
```

Пример применения команды DROP DATABASE

Допустим, есть база данных DB, которая автоматически запускается при запуске сервера. В этой базе данных хранится устаревшая информация, с ней уже не работают, поэтому принято решение её удалить.

Перед удалением базу данных необходимо остановить. Если выполнить попытку удаления БД без ее останова:

```
drop database DB;
```

В этом случае команда выполнится с ошибкой. Поэтому сначала необходимо отправить запрос на останов базы данных:

```
shutdown database DB;
```

И только после успешно выполненного останова БД можно выполнить её удаление:

```
drop database DB;
```

В результате успешного выполнения запроса из сервиса управления удаляется запись о базе данных и сам каталог БД в файловой системе.

Вернуться в оглавление

11.10. SHUTDOWN SERVICE

Команда останова сервиса управления базами данных и приложения-сервера используется для корректного завершения их работы.

Команда SHUTDOWN SERVICE выполняется из подключения к сервису управления базами данных. Выполнить останов сервиса управления базами данных и приложения-сервера может пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды:

```
SHUTDOWN SERVICE;
```

Во время выполнения команды все запущенные экземпляры баз данных, находящиеся под управлением этого сервиса, будут остановлены.

Если во время выполнения команды есть незаконченные транзакции, то они завершаются с откатом. При попытке пользователя выполнить какую-либо команду будет возвращена ошибка.

Пример применения команды SHUTDOWN SERVICE

Если есть необходимость выключить сервер, то необходимо корректно завершить работу сервиса и сервера соответствующей командой:

```
shutdown service;
```

В результате выполнения команда остановит сервис и все запущенные экземпляры баз данных, находящиеся под его управлением, и приложение-сервер завершит свою работу.

Вернуться в оглавление

11.11. RESTART SERVICE[CV]

Команда перезапуска сервиса управления базами данных.

Команда RESTART SERVICE выполняется из подключения к сервису управления базами данных. Выполнить перезапуск сервиса управления базами данных может пользователь сервиса с системной привилегией DATABASE ADMIN.

Синтаксис команды:

```
restart service;
```

Во время выполнения команды все запущенные экземпляры баз данных, находящиеся под управлением этого сервиса, будут остановлены.

Если во время выполнения команды есть незаконченные транзакции, то они завершаются с откатом. При попытке пользователя выполнить какую-либо команду будет возвращена ошибка.

После успешного выполнения команды RESTART SERVICE:

– будут разорваны все текущие соединения с сервером;

– сервер продолжит работу на основе текущего состояния конфигурационного файла;

– сервис управления базами данных будет запущен;

– базы данных со значением 'on' для опции database.autostart будут запущены, а запущенные до выполнения команды перезапуска базы данных со значением 'on' - перезапущены.

Для дальнейшей работы с сервисом управления БД или управляемыми им базами данных необходимо подключиться к ним снова.

Вернуться в оглавление

12. РУКОВОДСТВО ПО SQL

12.1. Типы данных

В СУБД СОКОЛ каждый столбец, локальная переменная, выражение и параметр имеет определенный тип данных.

Тип данных — атрибут, определяющий, какого рода данные могут храниться в объекте: целые числа или дробные, символы, метки времени и даты, бинарные данные, логические и так далее. Для всех типов допускается значение NULL.

Ниже приведена таблица 7 с типами данных СУБД СОКОЛ.

Таблица 7 — Тип данных

№ п/п	Тип данных	Описание
Строковый тип данных		
1	CHAR (size)	Строковый тип переменной длины (от 1 до 8000 байт). В поле помещается только то количество символов, которое необходимо. Хранимые величины не дополняются пробелами. Здесь size — общее количество байт в строке
2	VARCHAR (size)	Строковый тип переменной длины (от 1 до 8000 байт). В поле помещается только то количество символов, которое необходимо. Хранимые величины не дополняются пробелами. Здесь size — общее количество байт
3	VARCHAR2 (size)	Реализован как синоним типа VARCHAR (size)
4	CLOB	Строковый тип большой длины до 2^64 байт. Используется для представления большого блока текстовых данных в заданном наборе символов
Тип данных для хранения даты		
5	DATE	Тип данных даты и времени для представления даты (ГОД, МЕСЯЦ, ДЕНЬ) и времени (ЧАС, МИНУТА, СЕКУНДА) без информации о долях секунды Длина до 22 байт
6	DATETIME	Реализован как синоним типа DATE
7	TIMESTAMP	Тип данных даты и времени для представления даты (ГОД, МЕСЯЦ, ДЕНЬ) и времени (ЧАС, МИНУТА, СЕКУНДА), включая доли секунды (с точностью до аттосекунд (10^-18)) Длина до 22 байт
Логический тип данных		

№ п/п	Тип данных	Описание
8	BOOLEAN	Логический тип данных для представления логических значений true или false
Числовой тип данных		
9	NUMBER [(precision [, scale])]	Числовой тип данных для представления целых чисел, дробных чисел с фиксированной запятой или плавающей запятой (число, которое может быть представлено в виде мантиссы с фиксированной запятой и экспоненты). Здесь: - precision (точность) — общее количество десятичных цифр от 1 до 38; - scale (шкала) — количество цифр справа (положительная шкала) или слева (отрицательная шкала) от десятичной запятой до наименее значащей цифры с округлением к ближайшему целому, при котором число округляется до целого, модуль разности с которым у этого числа минимален. В общем случае, когда число в десятичной системе округляют до N-го знака, правило может быть сформулировано следующим образом: - если (N+1) цифра числа < 5, то N-ю цифру числа сохраняют, а (N+1)-ю и все последующие обнуляют; - если (N+1) цифра числа ≥ 5, то N-ю цифру числа увеличивают на единицу, а (N+1)-ю и все последующие обнуляют). Шкала может варьироваться от -84 до 127. Если значения точности и шкалы не указаны явно, то это означает возможность хранения до 38 значащих цифр. Тип данных хранит ноль, а также положительные и отрицательные фиксированные числа с абсолютными значениями от 0.01E-14337 до 0.99999999999999999999999999999999999999e14846, но не включая их. Длина до 22 байт
10	SMALLINT, INTEGER, INT, BIGINT	Реализованы как синоним типа NUMBER (38,0)
11	NUMERIC [(precision [, scale])], DECIMAL [(precision [, scale])]	Реализованы как синонимы типа NUMBER [(precision [, scale])]
12	FLOAT [(p)]	Числовой тип данных с плавающей запятой и двоичной точностью, где p — двоичная точность, которая может варьироваться от 1 до 126 двоичных разрядов. По умолчанию точность составляет 126 двоичных разрядов или 38 десятичных знаков. Для преобразования двоичной точности в десятичную нужно умножить двоичную точность на 0,30103. Для преобразования

№ п/п	Тип данных	Описание
		десятичной точности в двоичную нужно умножить десятичную точность на 3.32193. Порядок допустим от 1*10^-14336 до 1*10^14846, исключая граничные значения. Так самое малое число = 0.(9) * 10^-14336, где число девяток задаётся в выражении FLOAT(p), где p="число десятичных знаков"/0.301). Реализован как тип NUMBER [(precision [, scale])]. Длина до 22 байт
13	DOUBLE PRECISION	Числовой тип данных с плавающей запятой и двоичной точностью 126 разрядов. Реализован как тип NUMBER [(precision [, scale])]. Длина до 22 байт
14	REAL	Числовой тип данных с плавающей запятой и двоичной точностью 63 разряда или 19 десятичных знаков. Реализован как тип NUMBER [(precision [, scale])]. Длина до 22 байт
Двоичный тип данных		
15	BLOB	Массив бинарных данных большой длины до 2^64 байт
16	BINARY (size)	Двоичный тип постоянной длины (от 1 до 8000 байт). Если введённые величины менее фиксированной длины, то они дополняются нулевыми байтами. Здесь size — общее количество байт
17	VARBINARY (size)	Двоичный тип переменной длины (от 1 до 8000 байт). Используется только то количество байтов, которое необходимо, величины нулевыми байтами не дополняются. Здесь size — общее количество байт.
18	RAW (size)	Реализован как синоним VARBINARY (size)
19	ROWID	Двоичный тип данных переменной длины (от 1 до 2048 байт). Используется только необходимое количество байт, величины нулевыми байтами не дополняются. ROWID используется в псевдоколонке ROWID для значений, уникально идентифицирующих каждую строку таблицы.

Указанные в таблице синонимы типов данных реализованы через основной тип данных, т.е. воспринимаются системой как основной тип данных.

Вернуться в оглавление

12.1.1. Часовые пояса[CV]

В СУБД СОКОЛ часовые пояса задаются в виде смещения относительно стандарта UTC (всемирного координированного времени).

В СУБД СОКОЛ используется часовой пояс базы данных (опция database.time_zone, см. п. 11.1. CREATE DATABASE), который задается при установке базы данных и используется по умолчанию для временных операций в базе данных. По умолчанию опция получает значение смещения относительно UTC локального времени компьютера, на котором запущен сервер СУБД.

При необходимости значение часового пояса базы данных можно изменить после её создания (подробнее см. п. 11.3. ALTER DATABASE). После выполнения команды значение часового пояса для этой БД сохраняется до момента нового изменения значения в опции time_zone. После выполнения команды значение часового пояса для этой БД сохраняется до момента нового изменения значения в опции time_zone.

Кроме этого, в СУБД СОКОЛ также можно использовать временные пояса для сессии. В каждой сессии часовой пояс наследуется из опции time_zone базы данных, к которой подключился пользователь. Данное значение можно изменить на период конкретной сессии (подробнее см. п. 11.5. ALTER SESSION).

Узнать дату с учётом установленного часового пояса можно с помощью функций CURRENT_TIMESTAMP, LOCALTIMESTAMP, CURRENT_DATE. Узнать установленный часовой пояс можно с помощью функций DBTIMEZONE, SESSIONTIMEZONE.

Подробнее о каждой функции смотри соответствующий пункт:

– 12.9.2.7. CURRENT_DATE;

– 12.9.2.13. CURRENT_TIMESTAMP;

– 12.9.2.15. DBTIMEZONE;

– 12.9.2.22. LOCALTIMESTAMP;

– 12.9.2.35. SESSIONTIMEZONE.

Вернуться в оглавление

12.2. Литералы

В следующих подпунктах будут рассмотрены представления литералов различных типов данных СУБД СОКОЛ.

В нотациях представления литералов использованы обозначения, указанные далее в таблице 8.

Таблица 8 — Обозначения

Обозначение	Значение
< >	в эти символы заключается название синтаксической единицы
::=	символы используются в качестве разделителя левой и правой части правила. В правой части правила раскрывается определение левой
|	символ используется для разделения альтернативных синтаксических единиц, буквально обозначающий "или"
[]	в эти символы заключается необязательная синтаксическая единица
…	набор символов указывает на то, что непосредственно предшествующая синтаксическая единица может повторяться один или несколько раз

Вернуться в оглавление

12.2.1. Строковый литерал

Строковый литерал представляет собой последовательность произвольных символов, заключённых в апострофы. Например: 'Basa', 'Имя 2'.

Синтаксис:

```
<Строковый литерал>::= '<символы>'

<символы>::= <пусто> | <символ>...
```

Вернуться в оглавление

12.2.2. Числовой литерал

Числовые литералы используются для представления целого числа, фиксированного числа или числа с плавающей запятой. Число может быть положительное или отрицательное и должно содержать минимум одну цифру. Если знак опущен, то значение считается положительным.

В дробных числах разделителем целой и дробной части выступает символ ".". Дробный литерал может иметь целую и дробную часть, а может только дробную. Например: 2, 0.1, -15.6, .6, 1e3 (то же, что и 1000), 53, 28430147172709553615.

Синтаксис:

```
<числовой литерал>::= <целое число> | <дробное число>


<целое число>::=[[+]|-]<цифра>[<цифра>...]
<дробное число> ::= [<целое число>.<цифра>[<цифра>...][E[[+]|-]<цифра>[<цифра>...]]
```

Вернуться в оглавление

12.2.3. Литерал даты и времени

В СУБД СОКОЛ даты представлены в соответствии с григорианским календарём.

В текущей версии СУБД СОКОЛ для представления литерала даты и времени используются символьные строки, описывающие значение даты и времени в одном из заданных форматов:

```
<литерал_даты>::= 'YYYY-MM-DD [<время>]'
          | 'DD.MM.YYYY [<время>]'
          | 'DD/MM/YYYY [<время>]'
          | 'DD MM YYYY [<время>]'
```

Для представления литерала даты с явным указанием эры (по умолчанию дата относится к нашей эре) используется формат:

```
<литерал_даты>::= 'AD|BC YYYY-MM-DD [<время>]'
          | 'AD|BC DD.MM.YYYY [<время>]'
          | 'AD|BC DD/MM/YYYY [<время>]'
          | 'AD|BC DD MM YYYY [<время>]'
```

Где:

1) YYYY — номер года, который указывается положительным целым числом. Возможно указание в формате от 1 до 4 цифр (число от 0 до 9999).

Если необходимо указать год, значение которого короче 4 цифр, то достаточно указать лидирующий ноль (например, для указания значения года '65' достаточно указать '065'). В противном случае значения, указанные в диапазоне 0-69, будут преобразованы в 2000—2069 гг., а значения, указанные в диапазоне 70-99 — в 1970—1999 гг.

2) AD и BC — обозначение нашей эры и до нашей эры соответственно. Обратите внимание, что в утилите vsql_console формат вывода дат по умолчанию не предусматривает признаки эры, поэтому значения дат с разными признаками эры будут выглядеть одинаково. Преобразование даты с указанием эры в строку возможно с помощью функции TO_CHAR.

3) MM — месяц в году, который может быть указан положительным целым числом или латиницей, в следующих форматах:

– числового обозначения от 1 до 12. При указании числа месяца, состоящего из одной цифры, допускается введение как с лидирующим нулём, так и без него;

– числового обозначения римской цифрой (от I до XII). Причём число 4 распознаётся в двух формах (IV, IIII), число 9 также распознаётся в двух формах (VIIII, IX). Регистр не важен;

– 3-х символьного обозначения наименования месяца латиницей (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec). Регистр не важен;

– полного наименования месяца латиницей (january, february, march, april, may, june, july, august, september, october, november, december). Регистр не важен.

4) DD — день месяца, который указывается положительным целым числом в формате номера дня в месяце (от 1 до 31). При указании числа дня, состоящего из одной цифры, допускается введение как с лидирующим нулём, так и без него.

В случае неверного указания числа дней в месяце будет возвращена ошибка (например, при указании '31/January/22' или '29/02/21').

5) <время> — время, которое указывается положительными целыми числами с точностью до секунды (DATE) или до аттосекунды 10^-18 (TIMESTAMP):

```
<время>::= [AM|PM] [HH:MI[:SS[.MS]]]
```

Где:

– AM/PM — время до обеда (Ante Meridiem) и после обеда (Post Meridiem) соответственно. При использовании этого параметра часы указываются в 12 часовом формате. Регистр не важен;

– HH — часы, формат представления которых может быть:

1) Двадцатичетырёхчасовой (число от 0 до 23).

2) Двенадцатичасовой (число от 0 до 12).

– MI — минуты (число от 0 до 59);

– SS — секунды (число от 0 до 59);

– MS — доли секунды (для указания точности допустимо задавать до 18 цифр).

При указании значения часа, минуты или секунды, состоящего из одной цифры, допускается введение как с лидирующим нулём, так и без него (значение аттосекунд хранится в том виде, в котором внесено).

Если указать значение даты без компонента времени, то по умолчанию используется время полночь (00:00:00 как для 24-часового формата времени, так и для 12-часового).

Числовое или символьное значение может быть преобразовано в дату с помощью, например, функции CAST.

Подробнее о функциях см п. 12.9. Функции SQL.

Важно: указание обозначения эры допускается до/после даты или после параметра <время>. Все форматы позволяют поменять местами дату и время.

Далее в таблице 9 рассмотрены примеры литералов и соответствующим им значения дат.

Таблица 9 — Примеры литералов и соответствующие им значения дат

	DATE	TIMESTAMP
Литерал	'12.05.2017'	'12.05.**0317**'
Дата	AD 2017-05-12 00:00:00	AD 317-05-12 00:00:00.0
Литерал	'2019-**july**-19'	'2019-jul-19'
Дата	AD 2019-07-19 00:00:00	AD 2019-07-19 **00:00:00.0**
Литерал	'20/january/02 **PM** 11:25:54'	'20/**jan**/02 **PM** 11:25:54.254'
Дата	AD 2002-01-20 23:25:54	AD 2002-01-20 23:25:54.254
Литерал	'21 february **70 AM** 11:25:54'	'21 feb **70 AM** 11:25:54.254'
Дата	AD 1970-02-21 11:25:54	AD 1970-02-21 11:25:54.254
Литерал	'2019-june-19'	'2019-jun-19 11:25:54.135212351562'
Дата	AD 2019-06-19 00:00:00	AD 2019-06-19 11:25:54.135212351562
Литерал	'12.**IV**.18'	'12.**IIII**.18'
Дата	AD 2018-04-12 00:00:00	AD 2018-04-12 00:00:00
Литерал	'**AD** 12.05.2017'	'**BC** 12.05.2017'
Дата	AD 2017-05-12 00:00:00	BC 2017-05-12 00:00:00.0
Литерал	'25.05.2017 11:25:54**.235**'	'25.05.2017 11:25:54.12345678901234567**91**'
Дата	ошибка	ошибка

Вернуться в оглавление

12.2.4. Логический литерал

Логические литералы имеют два возможных значения (true и false) и воспринимаются системой без учёта регистра.

Значения логических литералов зарезервированы как ключевые слова и не могут использоваться в качестве идентификаторов (это может привести к пересечению с ключевыми словами в контексте запроса).

Синтаксис:

```
<логический литерал>::= true | false
```

Вернуться в оглавление

12.3. Операции

Операции обозначаются специальными символами или ключевыми словами и манипулируют значениями конкретных типов данных.

Операции делятся на:

– арифметические;

– строковые;

– логические;

– операции сравнения.

Каждая операция имеет свой приоритет. Далее в таблице 10 указаны операции, их приоритеты (где 1 — самый высокий приоритет) и ассоциативность.

Таблица 10 — Сводная таблица приоритетов операций

Приоритет	**Символ операции**	**Комментарий**	**Ассоциативность**
1	"+", "-"	Унарные операции	Справа налево
2	"*", "/"	Умножение и деление	Слева направо
3	"+", "-", "||"	Сложение, вычитание, конкатенация	Слева направо
4	"=", "!=", "<>", "~=", ">", "<", ">=", "<="	Операции сравнения	Нет
5	"IS [NOT] NULL","LIKE", "[NOT] BETWEEN", "[NOT] IN", "EXISTS"	Операции сравнения	Нет
6	"NOT"	Логическое отрицание	Справа налево
7	"AND"	Логическое «И» — конъюнкция	Слева направо
8	"OR"	Логическое «ИЛИ» — дизъюнкция	Слева направо

Операции с одинаковым приоритетом вычисляются в соответствии с правилами ассоциативности. Неассоциативные операции требуют явного указания порядка при помощи скобок.

Изменить порядок вычислений можно с помощью группировки операций круглыми скобками — выражения с операциями в скобках имеют наивысший приоритет.

Вернуться в оглавление

12.3.1. Арифметические операции

Арифметические операции указаны в таблице 11 и применимы к числовым типам данных (например, к NUMBER).

Таблица 11 — Арифметические операции

Символ	Значение	Результат выполнения операции
"+"	унарный или бинарный плюс	стоящий перед числовым операндом рассматривается как унарный плюс, не меняет знака операнда; стоящий между числовыми операндами рассматривается как сложение значений операндов.
"-"	унарный или бинарный минус	стоящий перед числовым операндом рассматривается как унарный минус, меняет знак операнда; стоящий между числовыми операндами рассматривается как вычитание значений операндов.
"*"	умножение	умножение значений операндов.
"/"	деление	деление значений операндов.

Вернуться в оглавление

12.3.2. Строковые операции

Строковая операция описана в таблице 12 и применима к строковым типам данных (например, к CHAR).

Таблица 12 — Строковые операции

Символ	Значение	Результат выполнения операции
"||"	конкатенация	Объединение данных двух символьных строк в одну. Максимальная длина получившейся строки не должна превышать максимальную длину типа CHAR. Если длина получившейся строки превысит максимум, то будет возвращена ошибка.

Вернуться в оглавление

12.3.3. Логические операции

Логические операции применимы к логическому типу данных (BOOLEAN).

В СУБД СОКОЛ используются логических операции "OR", "AND", "NOT". Более подробная информация о логических операциях указана в таблице 13.

Таблица 13 — Логические операции

Символ	Значение	Результат выполнения операции
"AND"	Логическое "и"— конъюнкция	Значение true, если выполняются условия левого и правого операндов, иначе — false.
"OR"	Логическое "или" — дизъюнкция	Значение true, если выполняются условия хотя бы одного из операндов, иначе — false.
"NOT"	Логическое отрицание	Значение true, если не выполняются условия правого операнда, иначе — false.

Вернуться в оглавление

12.3.4. Операции сравнения

Операции сравнения в СУБД СОКОЛ обозначаются символами, указанными в таблице 14, и применимы ко всем типам данных СУБД. Операции сравнения не применимы к типу BLOB, а также к типам данных, которые не могут быть приведены друг к другу (будет возвращена ошибка). О преобразовании типов подробнее в п. 12.3.5 Свод допустимости преобразований типов данных.

Таблица 14 — Операции сравнения

Символ	Значение	Результат выполнения операции
"="	равно	Значение true, если операнды равны, иначе — false.
"!=" , "<>" , "~="	не равно	Значение true, если операнды не равны, иначе — false.
">"	больше, чем	Значение true, если левый операнд больше правого, иначе — false.
"<"	меньше, чем	Значение true, если левый операнд меньше правого, иначе — false.
">="	больше или равно	Значение true, если левый операнд больше или равен правому, иначе — false.
"<="	меньше или равно	Значение true, если левый операнд меньше или равен правому, иначе — false.
"IS NULL"	равно NULL	Значение true, если значение операнда равно NULL, иначе — false.
"IS NOT NULL"	не равно NULL	Значение true, если значение операнда не равно NULL, иначе — false.
"LIKE"	соответствие шаблону	Значение true, если значение операнда соответствует заданному шаблону, иначе — false. При указании шаблона могут быть использованы: – % (знак процента) — обозначает один или несколько символов, в т.ч. 0; – _ (нижнее подчёркивание) — обозначает один символ; – ESCAPE '<символ>' — где любой символ из ASCII может быть использован в качестве <символ>. Указание ESCAPE-символа означает, что следующий за ним символ интерпретируется как обычный символ (в т.ч. спец. символ типа % или _).
"BETWEEN"	в диапазоне	Значение true, если значение операнда входит в указанный диапазон значений, иначе — false.
"NOT BETWEEN"	вне диапазона	Значение true, если значение операнда не входит в указанный диапазон значений, иначе — false.
"IN"	в	Значение true, если значение операнда в числе указанных нескольких значений, иначе — false.
"NOT IN"	не в	Значение true, если значение операнда не в числе указанных нескольких значений, иначе - false.
"EXISTS"	существует	Значение true, если подзапрос возвращает хотя бы одну строку, иначе - false.

Вернуться в оглавление

12.3.5. Свод допустимости преобразований типов данных

В СУБД СОКОЛ поддерживается явное и неявное преобразование основных типов данных.

Далее в таблице 15 указана сводная информация по всем возможным (явным и неявным) преобразованиям типов.

В таблице используются обозначения:

1) Цифрой 1 обозначены неявные преобразование, которые применяются при выполнении любых операциях.

2) Под цифрой 2 обозначены преобразования, которые возможны в тех случаях, где указано об их допустимости (например, в функциях - в описании которых указана возможность преобразования значений этих типов данных). операциях.

3) Под цифрой 3 обозначены преобразования, допустимые только при явном преобразовании с помощью функции CAST (см. п. 12.9.2.3 CAST).

Явные преобразования допустимо выполнять и для тех типов, для которых допустимы неявные преобразования (далее в таблице 15 см. обозначение 1 и 2).

Таблица 15 — Возможность преобразования типов данных

ТИП ДАННЫХ итоговый → исходный ↓	CHAR	VARCHAR	CLOB	DATE	TIMESTAMP	BOOLEAN	NUMBER	BLOB	BINARY	VARBINARY	ROWID
CHAR	1	1	2	2	2	2	2	2	1	1	2
VARCHAR	1	1	2	2	2	2	2	2	2	2	2
CLOB	2	2	1	-	-	3	-	-	-	-	-
DATE	2	2	-	1	1	-	-	-	-	-	-
TIMESTAMP	2	2	-	1	1	-	-	-	-	-	-
BOOLEAN	2	2	2	-	-	1	3	-	-	3	-
NUMBER	2	2	-	-	-	3	1	-	-	-	-

ТИП ДАННЫХ итоговый → исходный ↓	CHAR	VARCHAR	CLOB	DATE	TIMESTAMP	BOOLEAN	NUMBER	BLOB	BINARY	VARBINARY	ROWID
BLOB	2	2	2	-	-	-	-	1	2	2	-
BINARY	2	2	2	-	-	-	-	2	1	1	2
VARBINARY	2	2	2	-	-	-	-	2	1	1	2
ROWID	2	2	2	-	-	-	-	2	2	2	1

1	Допустимо неявное преобразование
2	Допустимо неявное преобразование только в отдельных случаях
3	Допустимо только явное преобразование
-	Преобразование недопустимо

Подробнее правила и результаты неявного преобразование представлены далее в приложении 4.

Вернуться в оглавление

12.4. Выражения

Выражения представляют собой комбинацию из одного или нескольких значений, констант, переменных, полей, операторов и функций SQL, которая приводит к вычислению некоторого значения.

Действия над операндами выполняются согласно приоритету операций, указанному в таблице 10.

Порядок выполнения операций внутри выражения может быть изменен при помощи круглых скобок.

Выражение принимает тип данных результирующего значения.

При вычислении выражения можно преобразовать результат в необходимый тип данных (подробнее о функциях см. 12.9. Функции, о неявном преобразовании типов данных см. п. 12.3.5. Свод допустимости преобразований типов данных).

Вернуться в оглавление

12.5. Объекты базы данных в СУБД СОКОЛ и правила их именования

Рассмотрим каждый из объектов БД в СУБД СОКОЛ.

Вернуться в оглавление

12.5.1. Таблица (TABLE)

Основные объекты реляционной БД — таблицы, в которых хранятся все данные базы. Таблица представляет собой двумерную структуру, содержащую некоторое количество строк и столбцов.

Таблица изначально должна содержать хотя бы один столбец (column) и создаётся пустой. Тип данных каждого столбца задаётся при создании таблицы (типы данных см. в таблице 7).

В СУБД СОКОЛ реализована возможность создания базовых таблиц и временных.

Временные таблицы в СУБД СОКОЛ реализованы в двух вариантах: глобальные (GLOBAL TEMPORARY TABLE) и приватные (PRIVATE TEMPORARY TABLE).

Существование временных таблиц и их данных ограничено не только моментом явного удаления, как у базовых таблиц, но и установленным при создании таблицы правилом ON COMMIT:

– при наступлении COMMIT;

– при закрытии соединения.

Подробнее сводную информацию по характеристикам базовых и временных таблиц см. далее в таблице 16.

Таблица 16 — Характеристики таблиц в СУБД СОКОЛ

	Базовая таблица TABLE	**GLOBAL TEMPORARY TABLE**	**PRIVATE TEMPORARY TABLE**
Момент возникновения таблицы	Момент создания командой CREATE	Момент создания командой CREATE или, для ранее созданной, момент обращения к таблице в новом соединении	Момент создания командой CREATE
Момент возникновения данных	Момент добавления данных командой INSERT	Момент добавления данных командой INSERT	Момент добавления данных командой INSERT
Момент окончания жизни данных	Момент явного удаления данных командой DELETE	Момент явного удаления данных командой DELETE или момент окончания транзакции для опции ON COMMIT DELETE ROWS или момент окончания сеанса для опции ON COMMIT PRESERVE ROWS	Момент явного удаления данных командой DELETE или момент окончания транзакции для опции ON COMMIT DROP DEFINITION или момент окончания сеанса для опции ON COMMIT PRESERVE DEFINITION
Момент окончания существования структуры таблицы	Момент явного удаления таблицы командой DROP	Момент явного удаления таблицы командой DROP	Момент явного удаления командой DROP или момент окончания транзакции для опции ON COMMIT DROP DEFINITION или момент окончания сеанса для опции ON COMMIT PRESERVE DEFINITION
Доступность	Таблица доступна всем соединениям	Таблица доступна всем соединениям	Таблица доступна только создавшему её соединению
Доступность данных	Данные доступны любому пользователю с соответствующими привилегиями	Данные доступны только создавшему их соединению	Данные доступны только создавшему их соединению
Типы данных	В текущей реализации таблицы могут содержать поля со всеми типами данных, реализованными в СУБД (см. табл. 7)	В текущей реализации таблицы могут содержать поля со всеми типами данных, реализованными в СУБД (см. табл. 7), кроме типов BLOB и CLOB	В текущей реализации таблицы могут содержать поля со всеми типами данных, реализованными в СУБД (см. табл. 7), кроме типов BLOB и CLOB

Подробнее:

– о создании базовой таблицы см. п. 12.8.1.1.1. Синтаксис создания базовой таблицы;

– о создании временной таблицы см. п. 12.8.1.1.2. Синтаксис создания временной таблицы;

– об удалении таблицы см. п. 12.8.2.1. DROP TABLE.

Вернуться в оглавление

12.5.2. Представление (VIEW)

Представление в СУБД СОКОЛ — это виртуальная таблица, которая фактически не хранит данных. Владельцем представления является создавший его пользователь.

Содержимое представления определяется запросом SELECT, указанным в команде создания представления.

Представление, как результат запроса данных из таблицы, имеет ряд именованных столбцов и строк.

Представление нельзя построить на основе временных таблиц.

Примеры использования представлений:

– необходимость предоставления только части информации из большой таблицы, полный доступ к которой нежелателен;

– необходимость упрощения доступа к информации, распределенной по таблицам базы данных.

Подробнее:

– о создании представления см. п. 12.8.1.2. CREATE VIEW;

– об удалении представления см. п. 12.8.2.5. DROP VIEW;

– о построении запроса выборки данных см. п. 12.8.5. SELECT.

Вернуться в оглавление

12.5.3. Генератор последовательности (SEQUENCE)

Генератор последовательности предназначен для генерации последовательности целых чисел в соответствии с правилами, установленными при его создании. Каждое следующее обращение к генератору увеличивает текущее значение последовательности на шаг, определяемый при создании последовательности.

Допустимый диапазон генерируемых значений последовательности: от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

Работа генератора выполняется в рамках БД и без привязки к какой-либо транзакции. Все обращения к генератору последовательности, включая параллельные, упорядочиваются, что приводит только к последовательным изменениям генератора. Поэтому генератор последовательности допускает одновременное обращение конкурирующих транзакций. Каждый пользователь получит уникальное новое значение, которое зависит от времени обращения к генератору.

СУБД СОКОЛ поддерживает отказоустойчивое хранение генераторов последовательностей. Это требует дополнительных обращений к диску и замедляет процесс получения последовательности значений. Один из подходов к ускорению процесса генерации последовательности значений — так называемое кеширование значений, когда в журнал восстановления БД записывается не каждое следующее значение генератора, а только будущее значение с заданным шагом. Все промежуточные значения генерируются на основе информации из оперативной памяти без фиксации значений в журнале восстановления. Это позволяет ускорить работу с генератором, но при отказе системы все значения генератора, до сохраненного в журнале, будут утеряны.

Примеры использования генераторов последовательности:

– автоматическая генерация первичных ключей таблиц;

– сквозная нумерация данных в нескольких таблицах.

Подробнее:

– о создании генератора последовательности см. п. 12.8.1.4. CREATE SEQUENCE;

– об удалении генератора последовательности см. п. 12.8.2.3. DROP SEQUENCE.

Вернуться в оглавление

12.5.4. Индекс (INDEX)

Индекс предназначен для ускорения поиска данных в таблице. Это достигается созданием структуры (индекса), в которой искомые данные располагаются в упорядоченном виде (например, сбалансированное дерево) и каждая запись такой структуры имеет ссылку на строку исходной таблицы для получения полной информации.

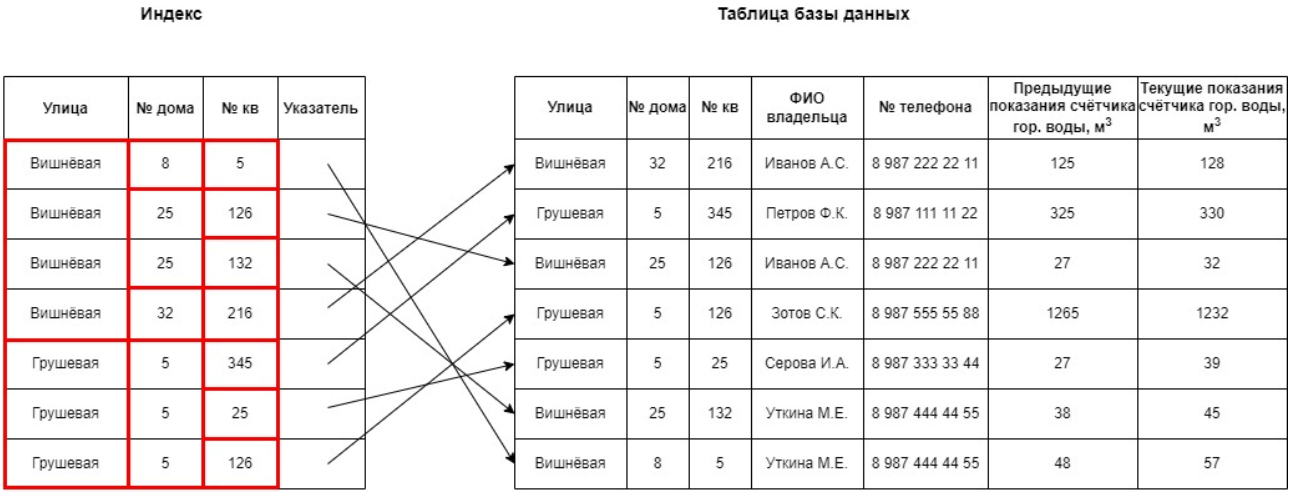
Индекс может быть:

– простой — создаётся по одному столбцу таблицы;

– составной — создаётся по нескольким столбцам таблицы.

Для оптимальной производительности запросов, индексы обычно создаются по тем столбцам таблицы, которые часто используются в запросах.

Наглядно пример индекса, сформированного по определённым столбцам таблицы, см. на рисунке 12.



Улица	№ дома	№ кв	Указатель
Вишнёвая	8	5	
Вишнёвая	25	126	
Вишнёвая	25	132	
Вишнёвая	32	216	
Грушевая	5	345	
Грушевая	5	25	
Грушевая	5	126	

Улица	№ дома	№ кв	ФИО владельца	№ телефона	Предыдущие показания счётчика гор. воды, м^3	Текущие показания счётчика гор. воды, м^3
Вишнёвая	32	216	Иванов А.С.	8 987 222 22 11	125	128
Грушевая	5	345	Петров Ф.К.	8 987 111 11 22	325	330
Вишнёвая	25	126	Иванов А.С.	8 987 222 22 11	27	32
Грушевая	5	126	Зотов С.К.	8 987 555 55 88	1265	1232
Грушевая	5	25	Серова И.А.	8 987 333 33 44	27	39
Вишнёвая	25	132	Уткина М.Е.	8 987 444 44 55	38	45
Вишнёвая	8	5	Уткина М.Е.	8 987 444 44 55	48	57

Рисунок 12 — Индекс и таблица БД

В примере показана таблица база данных водоснабжающей организации, которая ведёт учёт показаний счётчиков воды. Индекс построен по трём полям таблицы (улица, № дома, № квартиры). Такой индекс позволяет быстро искать информацию о владельце по трем полям индекса. Помимо этого, индекс может эффективно использоваться по неполному списку первых полей индекса. Например, можно быстро найти информацию о показаниях всех счетчиков, зарегистрированных на определенной улице или в определенном доме улицы.

Для одной таблицы может быть создано несколько индексов. Однако, увеличение числа индексов может снизить эффективность работы:

– индексы занимают дополнительный объем памяти (оперативной и дисковой);

– добавление, изменение, удаление записей в таблице требует перестроения её индексов, что замедляет исполнение этих операций.

Индекс может быть построен как для обычной таблицы, так и для временной. Однако, для временной глобальной таблицы индекс должен быть построен до вставки первой записи, иначе при работе с её данными может быть возвращена ошибка. При возникновении подобной ситуации необходимо очистить временную таблицу и далее начать работу с ней в обычном режиме.

Время жизни индекса равно времени жизни таблицы, если не удалить его раньше.

Длина индекса, в том числе построенного для первичного или кластеризованного ключа, не может превышать 2000 байт при расчёте по формуле:

```
S + PK + 2 * <количество_столбцов_ключа> + 4
```

где:

– S — сумма длин столбцов, указанных в индексе;

– PK — длина столбца первичного ключа таблицы (явного или неявного).

Вспомогательные сведения для вычисления каждого компонента описаны далее.

Расчёт суммы длин, указанных в индексе столбцов, производится согласно их типу и заданному ограничению длины:

– для CHAR (size) / VARCHAR (size) — (size + 2) байт;

– для любых числовых типов, типов данных для хранения даты — 22 байта;

– для целочисленных типов данных (number (38,0)) - 21 байт;

– для дробных числовых типов - 21 байт, кроме тех, у которых scale больше 56 или меньше -56. Для таких длина должна быть равна 22 байта;

– для BINARY (size) / VARBINARY (size) — (BASE128_size + 1) байт, где BASE128_size — это длина в байтах после BASE128-преобразования оригинальной последовательности байт.

Индекс (в том числе первичный ключ) не может быть построен по столбцам с данными типа ROWID, CLOB и BLOB.

Длина столбца первичного ключа таблицы (явного или неявного).

Таблица всегда имеет первичный ключ:

– если при создании таблицы указывается UNIQUE CLUSTERED KEY / PRIMARY KEY, то формируется **явный** первичный ключ;

– если при создании таблицы не указывается UNIQUE CLUSTERED KEY / PRIMARY KEY, то формируется неявный первичный ключ на основе скрытого числового столбца (22 байта);

– если при создании таблицы указывается CLUSTERED KEY, то первичный ключ неявно формируется на основе столбца CLUSTERED KEY и дополнительного скрытого числового столбца (22 байта).

Подробнее:

– о создании индекса см. п. 12.8.1.3. CREATE INDEX;

– об удалении индекса см. п. 12.8.2.2. DROP INDEX.

Вернуться в оглавление

12.5.5. Схема (SCHEMA)

Схема представляет собой средство группировки объектов базы данных и является отдельным пространством имён для своих объектов. Это позволяет хранить в разных схемах объекты с одинаковыми именами.

Схемой могут быть сгруппированы:

– таблицы;

– генераторы последовательности;

– индексы;

– хранимые процедуры;

– представления.

К схеме можно настроить доступ только определенных пользователей, что значительно облегчает управление объектами в базах данных с большим числом сущностей.

При создании пользователя может быть автоматически создана одноимённая схема, владельцем которой он является. Данная схема становится его схемой по умолчанию. Или при создании пользователя может быть указана существующая в БД схема, которая будет схемой по умолчанию для этого пользователя. В этом случае схема с именем пользователя не будет автоматически создаваться. Владелец схемы и схема по умолчанию для пользователя могут быть изменены.

При явном создании схемы обязательно указывается пользователь, который будет её владельцем.

Если в запросе к базе данных для какого-либо объекта не указана схема, то подразумевается схема по умолчанию текущего пользователя.

После соединения с БД текущей схемой пользователя становится схема по умолчанию.

Подробнее:

– о создании схемы см. п. 12.8.1.5. CREATE SCHEMA;

– о создании пользователя см. п. 12.8.1.6. CREATE USER;

– об изменении владельца схемы см. п. 12.8.3.1. ALTER SCHEMA;

– об изменении для пользователя схемы по умолчанию см. п. 12.8.3.2. ALTER USER SCHEMA;

– об удалении схемы см. п. 12.8.2.8. DROP SCHEMA.

Вернуться в оглавление

12.5.6. Пользователь (USER)

Пользователь представляет собой учётную запись, с помощью которой можно подсоединиться к БД и работать с её объектами в рамках предоставленных привилегий.

Созданному пользователю автоматически предоставляется системная роль PUBLIC, а также автоматически он наделяется всеми объектными привилегиями на создаваемые им объекты в схемах, владельцем которых он является.

Пользователю может быть назначена как отдельная привилегия, так и набор привилегий в виде роли. Привилегии и роли у пользователя можно отозвать в любой момент.

Наличие разных пользователей позволяет предотвращать несанкционированное использование БД или её компонентов.

При создании пользователя пароль задаётся сразу или может отсутствовать. При отсутствии пароля пользователь не имеет возможности аутентифицироваться и работать с объектами БД.

Подробнее:

– о видах привилегий см. п. 12.5.7. Привилегии (PRIV) и роли (ROLE);

– о создании пользователя см. п. 12.8.1.6. CREATE USER;

– о назначении привилегий см. п. 12.8.14.1. REVOKE PRIV и роли см. п. 12.8.13.2. GRANT ROLE;

– об отзыве привилегий см. п.12.8.14.1. REVOKE PRIV и роли см. п. 12.8.14.2 REVOKE ROLE;

– об удалении пользователя см. п. 12.8.2.6 DROP USER.

Вернуться в оглавление

12.5.7. Привилегии (PRIV) и роли (ROLE)

Привилегия представляет собой право выполнения определенных действий по отношению к базе данных, к объектам базы данных или к данным в базе данных.

В СУБД СОКОЛ привилегии делятся на системные и объектные:

– системные привилегии позволяют контролировать доступ к базе данных и дают право на выполнение определенного действия на уровне БД (как с самой базой данных, так и с её объектами);

– объектные привилегии позволяют контролировать доступ к конкретным объектам базы данных и выполнять с ними определенные действия.

Назначение объектных привилегий разрешено для объектов любой схемы, кроме административных схем (SYS, SOQOL, INFO, PUBLIC). В административных схемах допустима только привилегия SELECT.

Назначение объектных привилегий недопустимо для приватных временных таблиц (PRIVATE TEMPORARY TABLE).

Роль представляет собой набор привилегий и часто используется в качестве инструмента администрирования привилегий. Роль может быть назначена пользователю БД или другой роли.

В СУБД СОКОЛ реализованы предопределенные системные роли и привилегии. Все они, кроме роли PUBLIC, не подлежат изменению.

В таблице 17 представлена информация о предопределенных привилегиях и ролях, а также разрешённых в соответствии с ними операциях.

Таблица 17 — Привилегии и роли пользователей БД в СУБД СОКОЛ

Наименование роли / привилегии	**Разрешённые операции**
Системные привилегии	
CREATE SESSION	Позволяет выполнять: - подключение к базе данных/сервису пользователя с паролем; - изменения настроек соединения пользователя с базой данных командой ALTER SESSION.
RESOURCES CONTROL	Позволяет пользователю выполнять:

Наименование роли / привилегии	Разрешённые операции
	- операции, разрешённые системной привилегией CREATE SESSION; - в схеме PUBLIC и в схеме, владельцем которой он является создание объектов БД командой CREATE (TABLE, VIEW, INDEX, PROCEDURE, SEQUENCE) и удаление объектов БД командой DROP (TABLE, VIEW, INDEX, PROCEDURE, SEQUENCE); - команду ALTER TABLE[CV] по отношению к таблице, владельцем которой он является.
DATABASE ADMIN (для пользователя сервиса)	Позволяет пользователю выполнять операции, разрешённые системными привилегиями CREATE SESSION и RESOURCES CONTROL, а также позволяет выполнять команды администрирования любых баз данных (кроме таблиц словаря), находящихся под управлением сервиса: - создание БД командой CREATE DATABASE; - запуск базы данных командой STARTUP DATABASE; - останов базы данных командой SHUTDOWN DATABASE; - удаление БД командой DROP DATABASE; - присоединение базы данных к сервису управления базами данных командой ATTACH DATABASE; - отсоединение базы данных от сервиса управления базами данных командой DETACH DATABASE; - изменение настроек БД командой ALTER DATABASE; - изменение параметров сервиса командой ALTER SERVICE.
DATABASE ADMIN (для пользователя базы данных)	Позволяет пользователю выполнять операции, разрешённые системными привилегиями CREATE SESSION и RESOURCES CONTROL, а также позволяет выполнять команды администрирования своей базы данных (кроме таблиц словаря), работу с объектами БД, включая возможность предоставления, изменения, удаления привилегий и ролей другим пользователям или ролям: - изменение настроек БД командой ALTER DATABASE; - выполнение всех команд из справочника команд SQL в схеме любого пользователя (12.8 Справочник команд SQL).
Объектные привилегии	
SELECT	Позволяет выполнять выборку данных из таблицы или представления командой SELECT
INSERT	Позволяет добавлять данные в таблицу командой INSERT
DELETE	Позволяет удалять записи из таблицы командой DELETE
UPDATE	Позволяет обновлять данные в таблице командой UPDATE
EXECUTE	Позволяет запускать процедуры, созданные другим пользователем.
USAGE	Позволяет обращаться к последовательности, созданной другим пользователем.
INDEX	Позволяет создавать индексы на чужие таблицы в чужой схеме.

Наименование роли / привилегии	Разрешённые операции
Системные роли	
PUBLIC	Роль используется для удобства предоставления привилегий новым пользователям. Изначально роль пуста и автоматически назначаться каждому новому пользователю.
CONNECT	Включает в себя системную привилегию CREATE SESSION и возможность выполнять выборку данных из системной таблицы INFO.tables.
RESOURCE	Включает в себя только системную привилегию RESOURCES CONTROL.
DBA	Включает в себя только системную привилегию DATABASE ADMIN.

Пользователь базы данных с привилегией DATABASE ADMIN может создать новую роль с необходимым набором привилегий в той базе, в которой он находится. Созданная роль изначально имеет пустой список привилегий. Привилегии назначаются роли командой - GRANT PRIV, а отзываются командой REVOKE PRIV.

Автоматически созданному пользователю предоставляется системная роль PUBLIC. При создании объектов пользователем в схемах, владельцем которых он является, ему автоматически предоставляются объектные привилегии на созданные объекты.

Владелец схемы может предоставить объектные привилегии другому пользователю на объекты в своих схемах.

Пользователь с привилегией DATABASE ADMIN может предоставить любому пользователю базы данных или отозвать любую:

– системную привилегию;

– объектную привилегию на любой объект БД;

– роль (в т.ч. предустановленную).

Подробнее:

– о создании роли см. п. 12.8.1.7. CREATE ROLE;

– об удалении роли см. п. 12.8.2.7. DROP ROLE;

– о добавлении роли см. п. 12.8.13.2. GRANT ROLE;

– о добавлении привилегий см. п. 12.8.13.1. GRANT PRIV;

– об отзыве роли см. п. 12.8.14.2. REVOKE ROLE;

– об отзыве привилегий см. п. 12.8.14.1. REVOKE PRIV.

Вернуться в оглавление

12.5.8. Хранимая процедура (PROCEDURE)

Хранимая процедура представляет собой набор инструкций процедурного языка, реализующих заданный алгоритм.

Помимо инструкций процедурного языка хранимая процедура может содержать операции работы с базами данных (DDL, DML, DCL).

Хранимая процедура может сформировать и вернуть результаты в виде отдельных значений или наборов данных.

Хранимые процедуры допускают взаимный и рекурсивный вызов.

В хранимых процедурах разрешается использование временных таблиц.

Примеры использования хранимых процедур:

– повышение производительности за счет выполнения набора команд SQL на сервере без обмена промежуточным результатами по сети между клиентом и сервером;

– инкапсуляция логики предметной области в базе данных.

Подробнее:

– о создании хранимой процедуры см. п. 12.8.1.8. CREATE PROCEDURE;

– о вызове хранимой процедуры см. п. 12.8.11. CALL;

– об удалении хранимой процедуры см. п. 12.8.2.4. DROP PROCEDURE.

Вернуться в оглавление

12.5.9. Псевдостолбец ROWID

В базе данных СУБД СОКОЛ в каждой таблице присутствует псевдостолбец ROWID, которая содержит значение, уникально идентифицирующее каждую строку таблицы — адрес строки. Зная адрес строки, можно быстро получить доступ к значениям в ней, исключая поиск перебором других значений таблицы. Или, например, есть таблица без первичных ключей. В ней могут быть повторяющиеся

строки, которые нужно удалить, кроме одной. Зная адрес этой строки, это легко сделать.

Псевдостолбцы ROWID имеют одноименный тип данных ROWID (подробнее см. п. 12.1. Типы данных).

В псевдостолбец ROWID нельзя вставлять значения, изменять имеющиеся в нём или удалять их. Сам псевдостолбец нельзя удалить, но можно прочитать значения в нём.

Например, у нас есть таблица PHONE с данными абонентов:

```
 NUMBER     | NAME   | AGE | CITY
------------+--------+-----+----------
 89876543212 | Петров | 53  | Воронеж
 89876543214 | Иванов | 46  | Москва
 89876543215 | Попова | 26  | Воронеж
```

Мы можем выполнить запрос идентификаторов (адресов) строк, значения столбца AGE которых больше 40:

```
select ROWID from PHONE where AGE > 40;
```

После выполнения команды будут получены уникально идентифицирующие каждую строку значения, для которых указанное условие верно, например:

```
 ROWID
-----------
 d77900000700000040001100000000000b5530f0008000000c611c5996d411800
 d77900000700000040001100000000000b5530f0008000000c611c5996d411c00
```

Вернуться в оглавление

12.5.10. Правила именования объектов БД и обращения к ним

Каждый объект базы данных СУБД СОКОЛ имеет имя, которое назначается ему в соответствии с определёнными правилами:

1) При назначении имени любому объекту базы данных необходимо учитывать особенности использования кавычек:

– при указании имени в кавычках — регистр букв сохраняется. Например, имена "телефон", "Телефон", "ТЕЛЕФОН" в СУБД СОКОЛ интерпретируются как разные;

– при указании имени без кавычек — буквы приводятся к верхнему регистру. Например, имена phone, Phone, PHONE в СУБД СОКОЛ интерпретируются как одинаковые.

Эти и другие особенности именования объектов БД сведены в таблице 18.

Таблица 18 — Особенности именования объектов БД

	Имя, заключённое в " "	**Имя, указанное без " "**
Регистрозависимость	Регистрозависимое (регистр символов сохраняется)	Регистронезависимое (символы приводятся к верхнему регистру)
Допустимые символы имени для всех объектов	Любые (исключая символ с кодом 0), включая пробел	Латинские буквы, цифры и символы "_", "#", "$"
Первый символ имени	Для PRIVATE TEMPORARY TABLE и индекса, построенного для временной таблицы, первым в имени должен стоять символ "#". Для других объектов, кроме SCHEMA, USER, ROLE, PROCEDURE, допустим любой символ (исключая символ с кодом 0 и "#"), включая пробел. Для объектов SCHEMA, USER, ROLE, PROCEDURE допустим символ "#"	Для PRIVATE TEMPORARY TABLE и индекса, построенного для временной таблицы, первым в имени должен стоять символ "#", для остальных объектов, кроме SCHEMA, USER, ROLE, PROCEDURE — латинская буква или символ "_". Для объектов SCHEMA, USER, ROLE, PROCEDURE допустим символ "#". Символ "$" нельзя использовать первым в имени объектов БД, т.к. он зарезервирован в качестве первого символа для системных объектов

2) Имена объектов фиксированы, их нельзя изменить.

3) Имена объектов TABLE, SEQUENCE, PROCEDURE, VIEW должны быть уникальными в рамках одной схемы.

4) Имена ограничений (constraint), INDEX должны быть уникальными в рамках одной схемы.

5) Имя SCHEMA должно быть уникальным среди имён существующих в БД схем.

6) Имя USER должно быть уникальным среди имён существующих в БД пользователей.

7) Имя ROLE должно быть уникальным среди имён существующих в БД ролей.

8) Максимальная длина имени объекта БД — 64 байта. Обратите внимание, что имеется в виду размер внутреннего представлении имени в виде набора байтов, а не число символов в имени. Имя объекта в СУБД СОКОЛ всегда хранится в кодировке БД.

9) При создании любого объекта в текущей схеме или обращении к объекту допускается указание как полного имени, состоящего из имени схемы и краткого имени объекта, разделённых точкой, так и краткого имени:

```
<имя_объекта> ::= [<имя_схемы> "."]<краткое_имя_объекта>

<краткое_имя_объекта> ::= <имя_таблицы>
                |<имя_представления>
                |<имя_индекса>
                |<имя_схемы>
                |<имя_последовательности>
                |...
```

При создании объекта в схеме, отличной от текущей схемы пользователя, или при обращении к такому объекту обязательно указывать его полное имя. Если указать только краткое имя объекта, то произойдет обращение к объекту в текущей схеме.

Наряду с правилами именования объектов БД существует ограничение на имена столбцов таблиц: имя столбца в рамках одной таблицы, представления или индекса должно быть уникальным среди имён других столбцов.

Вернуться в оглавление

12.6. Работа с транзакциями

Транзакция представляет собой одну или ряд последовательных операций по работе с данными БД в рамках одной сессии, которая рассматривается системой как единое неразрывное действие по изменению состояния БД.

Каждая транзакция имеет начало и завершение.

Завершается транзакция одной из команд:

– COMMIT — попытка фиксации изменений и окончание транзакции. При успешной попытке изменения становятся неотъемлемой частью данных БД.

– ROLLBACK — отмена всех незафиксированных изменений текущей транзакции с последующим окончанием транзакции.

После соединения с БД включен режим AUTOCOMMIT. В этом режиме каждый оператор SQL «обрамляется» командами начала транзакции и командой COMMIT после успешного выполнения оператора или командой ROLLBACK после ошибочного выполнения.

Для перехода в режим явного управления началом и концом транзакций необходимо запустить транзакцию командой START TRANSACTION явно или выключить режим AUTOCOMMIT командой ALTER SESSION SET autocommit = 'off' (обратите внимание на нижний регистр ключа и значения). В первом случае транзакция начинается сразу после выполнения команды, во втором случае — с последующего оператора SQL.

В режиме явного управления транзакциями используйте команды COMMIT или ROLLBACK для завершения транзакции. После завершения текущей транзакции следующая явно управляемая транзакция начнётся либо со следующего посланного оператора SQL, либо командой START TRANSACTION.

Между транзакциями можно выполнить команду установки уровня изоляции для всех последующих транзакций SET TRANSACTION ISOLATION LEVEL (уровень изоляции можно установить и командой начала транзакции), а также переключить режим AUTOCOMMIT командой ALTER SESSION SET autocommit = 'on' | 'off'.

Действие любого оператора (в т.ч. в составе процедуры) применяется к БД в случае успеха. Если запрос завершится неуспешно, то будут отменены выполненные оператором изменения.

В рамках исполняемой транзакции можно создавать точки сохранения явно (специальные именованные отметки внутри текущей транзакции). Ссылаясь затем на такую точку в команде ROLLBACK <точка сохранения>, можно отменить изменения в транзакции, сделанные после этой точки (вместе с отменой изменений удаляются и

точки сохранения, созданные после неё). Транзакция в этом случае продолжается от данной точки сохранения.

Созданные явно в текущей транзакции точки сохранения можно удалить явно. При удалении явно созданной точки сохранения удаляются и все точки, установленные после неё.

В режиме явного управления транзакциями после каждой успешной выполненной команды DML неявно создаётся точка сохранения. В случае завершения выполнения команды DML ошибкой, автоматически выполняется ROLLBACK до последней успешно выполненной DML команды в текущей транзакции.

Обратите внимание, что отмена изменений транзакции может быть выполнена только для команд манипулирования данными языка SQL — DML. Остальные команды (например, команды типа DDL, DCL или команды управления базами данных в рамках сервиса управления БД) не могут быть отменены.

Уровни изоляции

Во избежание влияния результатов работы одних транзакций на другие используется изоляция.

Уровень изоляции — условное значение, определяющее, в какой мере в результате выполнения логически параллельных транзакций в СУБД допускается получение несогласованных данных. Шкала уровней изолированности транзакций содержит ряд значений, проранжированных от низшего к высшему. Более высокий уровень изолированности соответствует лучшей согласованности данных, но повышает вероятность отката транзакции. Более низкий уровень изолированности снижает вероятность отката транзакций, но тем больше видно влияние параллельных транзакций на данные. Так, выбором уровня изолированности транзакций в определённой мере достигается компромисс между скоростью работы и гарантированной согласованностью получаемых из системы данных.

В СУБД СОКОЛ реализованы следующие уровни изоляции:

– READ COMMITTED — самый низкий уровень изоляции в СУБД СОКОЛ. При этом уровне изоляции видны изменения от других транзакций, зафиксированные **до**

начала выполнения каждого оператора. Если транзакция с уровнем READ COMMITTED требует блокировки строк (при выполнении команд INSERT, UPDATE, SELECT FOR UPDATE, DELETE), принадлежащих другой транзакции, то она будет ждать снятия блокировки и только после этого будет работать с обновлёнными данными. READ COMMITTED является уровнем, устанавливаемым по умолчанию при соединении с БД;

– SERIALIZABLE — самый высокий уровень изоляции в СУБД СОКОЛ, при котором видны только изменения, зафиксированные **до момента старта транзакции**. Если транзакция с уровнем SERIALIZABLE требует блокировки строк, которые уже изменены и не зафиксированы другой транзакцией, то она будет ждать завершения конкурентной транзакции. Далее ожидающая транзакция с уровнем SERIALIZABLE завершается с откатом в случае фиксирования конкурентной транзакции или продолжает свое исполнение в случае отката конкурирующей транзакции.

Для уровней изоляции READ COMMITTED и SERIALIZABLE допускается указание дополнительного модификатора SNAPSHOT.

SNAPSHOT — модификатор, устанавливаемый по умолчанию, при указании которого транзакция с уровнем изоляции READ COMMITTED и SERIALIZABLE читает данные следующим образом:

– в случае READ COMMITED — зафиксированные на момент начала оператора;

– в случае SERIALIZABLE — на момент начала транзакции. В данном случае это чревато более вероятными откатами в случае конкурентной работы.

Установить уровень изоляции можно в период между транзакциями командой SET TRANSACTION ISOLATION LEVEL, а командой START TRANSACTION ISOLATION LEVEL можно начать транзакцию с заданным уровнем.

При попытке запуска команды установки уровня изоляции в рамках текущей транзакции будет возвращена ошибка.

Установленный уровень изоляции действует для всех последующих транзакций до конца сессии, либо до его изменения одним из указанных выше способов.

Подробнее:

– о создания точки сохранения см. п. 12.8.12.3.1. SAVEPOINT;

– о команде удаления точки сохранения см. п. 12.8.12.3.2. RELEASE SAVEPOINT;

– о команде отмены изменений см. п. 12.8.12.4. ROLLBACK;

– о команде фиксации изменений и завершения текущей транзакции см. п. 12.8.12.5. COMMIT;

– о команде установки уровня изоляции транзакции см. п. 12.8.12.1. SET TRANSACTION ISOLATION LEVEL;

– о команде запуска транзакции см. п. 12.8.12.2. START TRANSACTION;

– о команде включения и отключения AUTOCOMMIT см. п. 11.5. ALTER SESSION.

Вернуться в оглавление

12.7. План выполнения запроса (EXPLAIN)

Команда позволяет получить план выполнения запроса, построенный планировщиком СУБД ЛИНТЕР СОКОЛ. В текущей реализации СУБД допустимо получение плана выполнения запроса SELECT. Информация о шагах выполнения, использованных индексах полезна для последующей оптимизации запросов (например, добавления новых индексов в таблицу, изменения самого запроса).

Чтобы получить план выполнения запроса выборки данных, необходимо перед планируемым SELECT указать ключевое слово EXPLAIN. В этом случае запрос SELECT не будет выполнен, а в качестве ответа сервера будет выдан предполагаемый план исполнения указанного запроса.

Информация представляет собой описание древовидной структуры плана исполнения запроса и возвращается в табличном виде. Подробное описание столбцов в таблице плана выполнения запроса см. в таблице 19.

Таблица 19 — Описание столбцов плана исполнения запроса

Наименование столбца	Описание
OPERATION	Наименование операции
OPTIONS	Описание параметров операции, таких как выражения фильтров записей, условий соединений или списки ключей при запросах данных из таблиц.
OBJECT_TYPE	Тип объекта, над которым производится операция. Для промежуточных операций остаётся пустым.
OBJECT_NAME	Имя объекта, над которым производится операция. Для промежуточных операций остаётся пустым.
OBJECT_ALIAS	Псевдоним (алиас) объекта, над которым производится операция, если он объявлен в запросе. Для промежуточных операций псевдоним отсутствует, т.к. псевдоним является свойством источников данных и для дальнейших операций не заполняется.
DEPTH	Глубина операции в дереве плана от корня, начиная с единицы. Все источники данных для текущей операции имеют равные значения глубины, на единицу больше, чем эта операция.
COST	Оценка стоимости операции в условных временных единицах. Чем больше число в этом столбце, тем бо́льше длительность её исполнения.
CARDINALITY	Оценка количества записей после исполнения операции. При отсутствии статистики по соответствующим источникам данных информация в этом столбце условна и пригодна только для оценки относительной селективности операций.
PROJECTION	Список выражений, передаваемых операцией далее по дереву плана.

Столбцы в полученной таблице могут содержать один или несколько элементов, разделённых символом ";", а также структурные элементы более высокого уровня с разделителем "\", обычно начинающиеся с тега описания типа элемента, оканчивающегося двоеточием:

– FILTER: — выражение фильтра, которому должны удовлетворять записи, передаваемые следующему узлу плана;

– WHILE: — выражение условия остановки сканирования хранилища данных. Сканирование продолжается, пока значение этого выражения равно true;

– INDEX: — наименование индекса, используемого в операции. При отсутствии упоминания индекса подразумевается использование основного хранилища;

– KEYS: — список выражений-источников данных для ключей, которые используются операцией при выборке данных из хранилища.

В описании выражений используются следующие сокращения:

– COL#<число> — ссылка на столбец записи данных, полученная от предыдущего узла, по её номеру;

– MRS#<число> — локальный временный источник данных, выдающий несколько записей, с его уникальным (в пределах запроса) номером;

– QP#<число> — параметр запроса с его номером.

Например: KEYS:QP#1\FILTER:VAL1=QP#2. В данном примере используется параметр запроса QP#1 в качестве ключа для поиска в таблице. Записи дополнительно фильтруются по условию равенства столбца таблицы VAL1 параметру запроса QP#2.

Каждая строка выведенного плана выполнения запроса — узел плана, описывает одну операцию над данными. Подробнее об узлах плана выполнения запроса далее см. в таблице 20.

Таблица 20 — Описание узлов плана исполнения запроса

Наименование узла	Описание
AGGREGATE	Агрегирование записей, полученных из входного потока данных. В столбце OPTIONS выводится список функций агрегации с параметрами
DISTINCT	Исключение повторяющихся записей
EXPRESSIONS SOURCE	Локальный источник данных, выдающий одну запись со значениями, полученными в результате вычисления набора выражений
GROUP	Группировка записей входного потока данных по заданному набору выражений. В столбце OPTIONS выводится список выражений, по которым производится группировка
INDEX JOIN	Соединение входящего потока данных с одной записью таблицы, получаемой по полному ключу основного хранилища (LOOKUP). В столбце OPTIONS выводится список ключей, использующихся для получения записи из таблицы. Опционально может присутствовать выражение дополнительного фильтра, которому должна удовлетворять полученная запись
INTERMEDIARY FILTER	Промежуточный фильтр записей по условию. Если задано условие, то оно выводится в столбце OPTIONS
	Соединение входящего потока данных с одной записью таблицы, получаемой по полному ключу основного хранилища (LOOKUP). В столбце OPTIONS выводится список ключей, использующихся для получения записи из таблицы. Опционально может присутствовать выражение дополнительного фильтра, которому должна удовлетворять полученная запись
LEFT INDEX JOIN	Левое внешнее соединение потока данных с одной записью таблицы, получаемой по полному ключу основного хранилища (LOOKUP). В столбце OPTIONS выводится список ключей, использующихся для получения записи из таблицы. Опционально может присутствовать выражение дополнительного фильтра, которому должна удовлетворять полученная запись
LEFT JOIN	Левое внешнее соединение двух потоков данных по заданному условию.

Наименование узла	Описание
	В столбце OPTIONS выводится выражение условия соединения потоков данных
LIMIT RECORDS	Ограничение числа выводимых записей. В столбце OPTIONS выводится значение ограничения
LOOKUP	Выборка одной записи из таблицы по полному набору ключей основного хранилища данных. В столбце OPTIONS выводится список ключей, использующихся для получения записи. Опционально может присутствовать выражение дополнительного фильтра, которому должна удовлетворять запись
MULTIRECORD EXPRESSIONS SOURCE	Локальный источник данных, выдающий несколько записей со значениями, полученными в результате вычисления набора выражений
NESTED LOOP JOIN	Соединение двух потоков данных по условию (декартово произведение при отсутствии условия). В столбце OPTIONS выводится выражение условия соединения потоков данных
RECURSIVE WITH PUMP	Получение очередной записи из результатов WITH в рамках исполнения WITH с рекурсией. В столбце PROJECTION выводится список столбцов, которые запрашиваются из данного WITH
RECURSIVE WITH UNION	Операция объединения результатов запросов в рамках исполнения WITH с рекурсией
REVERSE SCAN	То же что "SCAN" (см. далее), только с реверсным сканированием
REVERSE SCAN/LOOKUP	То же что "SCAN/LOOKUP" (см. далее), только с реверсным сканированием
SCAN	Сканирование записей таблицы из заданного диапазона (вплоть до полного сканирования всей таблицы). Может производиться как по основному хранилищу, так и по одному из вторичных индексов. В столбце OPTIONS выводится: – информация об используемом вторичном индексе; – список ключей, использующихся для получения первой записи из хранилища; – выражение условия остановки сканирования хранилища данных; – выражение дополнительного фильтра, которому должна удовлетворять каждая запись
SCAN/LOOKUP	Комбинированная операция сканирования записей из таблицы в заданном диапазоне по вторичному индексу с последующим обращением к основному хранилищу для получения колонок, недостающих в этом хранилище используемого индекса. В столбце OPTIONS выводится: – информация об используемом вторичном индексе; – список ключей, использующихся для получения первой записи из хранилища; – выражение условия остановки сканирования хранилища данных; – выражение дополнительного фильтра, которому должна удовлетворять каждая запись
SELECT	Корневой узел основного запроса.

Наименование узла	Описание
	В столбце PROJECTION выводится список выражений, которые извлекает запрос
SELECT (WITH)	Корневой узел запроса, использующего результаты WITH. В столбце PROJECTION выводится список выражений, которые извлекает запрос
SEMI JOIN	Полусоединение: пропускает записи из первого потока данных далее, если получена хотя бы одна запись из второго потока данных для соответствующей записи из первого потока.
SORT	Сортировка записей входного потока данных по заданному набору выражений. В столбце OPTIONS выводится список выражений, по которым производится сортировка.
SORT DISTINCT	Сортировка записей входного потока данных по всем столбцам записей с последующим исключением повторяющихся записей.
SUBQUERIES	Производит исполнение одного или нескольких подзапросов в рамках операций EXISTS или IN, превращая их результаты в логические значения true/false, которые затем используются для вычисления выражения фильтра, пропускающего далее записи основного потока данных, а также при вычислении выражений выходных колонок записей, которые передаются следующему узлу плана. Условие фильтра, если оно есть, выводится в столбце OPTIONS.
SUBQUERY	Корневой узел подзапроса. В столбце PROJECTION выводится список выражений, которые извлекает подзапрос. Указанные выражения могут быть источниками данных для других узлов плана.
UNION	Операция объединения результатов запросов.
WITH PUMP	Получение очередной записи из результатов WITH. В столбце PROJECTION выводится список столбцов, которые запрашиваются из данного WITH.
WITH QUERY	Корневой узел запроса WITH. В столбце PROJECTION выводится список выражений, которые извлекает данный запрос, и которые могут являться источниками данных в операциях "WITH PUMP" / "RECURSIVE WITH PUMP"
WITH UNION	Операция объединения результатов запросов в рамках исполнения WITH

Подробнее:

– о получении плана запроса см. п. 12.8.10. EXPLAIN;

– о команде SELECT см. п. 12.8.5. SELECT.

Вернуться в оглавление

12.8. Справочник команд SQL

Посредством команд языка SQL можно оперировать данными в реляционной базе данных: создавать, модифицировать, управлять, получать и удалять.

Каждая команда языка SQL имеет свой синтаксис и особенности использования. В данном разделе описан поддерживаемый СУБД СОКОЛ диалект языка SQL и указаны примеры его использования.

Все команды с объектами проводятся в текущей базе данных.

Вернуться в оглавление

12.8.1. CREATE

Запрос CREATE используется для создания новой сущности (например, TABLE, ROLE, SCHEMA, USER, INDEX, SEQUENCE, VIEW, PROCEDURE).

Подробнее о правилах именования объектов БД и обращения к ним см. п. 12.5.10. Правила именования объектов БД и обращения к ним.

Вернуться в оглавление

12.8.1.1. CREATE TABLE

Команда создаёт таблицу. Таблица изначально создаётся пустая. Подробно о таблице см. п. 12.5.1. Таблица (TABLE).

Создать таблицу может:

– пользователь базы данных с системной привилегией RESOURCES CONTROL в схеме, владельцем которой он является;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Для команды CREATE TABLE используется два варианта синтаксиса: создание базовой таблицы и временной таблицы.

Вернуться в оглавление

12.8.1.1.1. Синтаксис создания базовой таблицы

```
CREATE TABLE <имя_таблицы> (
    <имя_столбца> <тип_данных> [<свойства_столбца>] [<ограничение_столбца>],
     [<имя_столбца> <тип_данных> [<свойства_столбца>] [<ограничение_столбца>],...]
     [<ограничение_таблицы>]
);
```

Где:

1) <имя_таблицы> — имя создаваемой таблицы.

2) <имя_столбца> — имя столбца, которое должно быть уникальным в рамках таблицы и удовлетворять правилам задания имён сущностей.

3) <тип_данных> — тип данных, поддерживаемый СУБД СОКОЛ (подробнее о типах см. п. 12.1. Типы данных).

4) <ограничение_столбца> — ограничение, накладываемое на данные столбца. Синтаксис конструкции <ограничение_столбца>:

```
<ограничение_столбца>::= [CONSTRAINT <имя_ограничения>] <тип_ограничения>

<тип_ограничения> ::= | PRIMARY KEY
                      | NOT NULL
                      | UNIQUE
                      | [UNIQUE] CLUSTERED KEY
                      | CHECK (<условие>)
                      | <конструкция_FOREIGN_KEY>
```

Где:

– CONSTRAINT <имя_ограничения>[CV] — конструкция для указания имени ограничения, накладываемого на данные столбца. Указание имени ограничения позволит в дальнейшем при необходимости удалить это ограничение (подробнее см. п. 12.8.3.4. ALTER TABLE). Если имя ограничения не указать, то по умолчанию СУБД автоматически сгенерирует имя, соответствующее правилам именования объектов БД;

– PRIMARY KEY — указывает на то, что по данному столбцу должен быть построен первичный ключ, обладающий характеристиками уникальности (UNIQUE) и непустых значений (NOT NULL). Длина первичного ключа не может превышать 2000 байт и рассчитывается по той же формуле, что и для индекса (см. п. 12.5.4. Индекс (INDEX));

– NOT NULL — указывает на то, что по данному столбцу должны быть введены не пустые значения (по умолчанию в полях допускается NULL);

– UNIQUE — указывает на то, что по данному столбцу должны быть введены данные, обладающие характеристикой уникальности;

– CLUSTERED KEY — указывает на то, что по данному столбцу должен быть построен кластеризованный ключ, который может не обладать характеристиками уникальности (UNIQUE) и непустых значений (NOT NULL). Кластеризованный ключ представляет собой индексную структуру, в которой строки таблицы физически упорядочиваются по значениям кластеризованного ключа. По причине упорядоченности ключей обеспечивается быстрый доступ к подмножествам строк, значения ключей которых находятся рядом друг с другом;

– UNIQUE CLUSTERED KEY — указывает на то, что по данному столбцу должен быть построен кластеризованный ключ, обладающий характеристикой уникальности (UNIQUE).

Важно: таблице может быть только один столбец с ограничением [UNIQUE] CLUSTERED KEY. В одной таблице может быть или PRIMARY KEY или [UNIQUE] CLUSTERED KEY. Одновременное указание этих ограничений в рамках одной таблицы недопустимо;

– CHECK (<условие>)[CV] — конструкция для установки проверочного ограничения. В качестве <условия> задается логическое выражение, которому должно удовлетворять каждое значение строки таблицы. Выполнение команды будет прекращено с возвратом ошибки при попытке:

– добавления данных (или их изменение), не соответствующих установленному проверочному ограничению;

– указания проверочного ограничения для виртуальных столбцов;

– указания в <условии> виртуального столбца или столбца другой таблицы.

Важно: При создании нескольких проверочных ограничений для одного столбца будьте внимательны, т.к. система не проверяет, что условия установленных ограничений не являются взаимоисключающими.

5) <конструкция_FOREIGN_KEY>[CV] — указывает на то, что по данному столбцу должен быть построен внешний ключ, который позволяет поддерживать ссылочную целостность между двумя связанными таблицами:

– дочерней — в которой создаётся внешний ключ и задаётся его ограничение;

– родительской — на строки которой будет ссылаться создаваемый внешний ключ.

Синтаксис конструкции <FOREIGN_KEY>:

```
конструкция <FOREIGN_KEY> ::= REFERENCES <имя_родительской_таблицы>
                        [(<имя_столбца>)] [<сценарий_поведения>]


[<сценарий_поведения>]  ::= [ON DELETE { CASCADE | SET NULL | RESTRICT } ]
                    [ON UPDATE { CASCADE | SET NULL | RESTRICT } ]
```

Где:

– <имя_родительской_таблицы> — имя родительской таблицы, на строки которой будет ссылаться внешний ключ, указываемый в дочерней таблице;

– <имя_столбца> — имя столбца в родительской таблице, на который будет ссылаться столбец внешнего ключа дочерней таблицы. Указанный столбец родительской таблицы должен обладать ограничением уникальности. Соответствующий столбец внешнего ключа в родительской и дочерней таблицах, указанные в команде, должны совпадать по порядку, типам данных (синонимы типы данных считаются одинаковыми типами данных). Недопустимо указание во внешнем ключе столбца с типами данных CLOB, BLOB. Если не указать столбец родительской таблицы, то по умолчанию столбец внешнего ключа дочерней таблицы будет ссылаться на одноимённый столбец родительской таблицы;

– ON DELETE и ON UPDATE — позволяют устанавливать сценарий, который будет выполняться при попытке удаления (или изменения соответственно) связанной строки из родительской таблицы.

Допустимые сценарии:

– CASCADE — автоматически будет удалена (или изменена соответственно) строка (строки) в столбце из дочерней таблицы при удалении (или изменении соответственно) связанных строк в родительской таблице;

– SET NULL — будет установлено для строки в столбце внешнего ключа дочерней таблицы значение NULL при удалении (или изменении соответственно) связанной строки из родительской таблицы. Если столбец внешнего ключа не допускает значение NULL, то будет возвращена ошибка;

– RESTRICT — попытка удаления (или изменения соответственно) из родительской таблицы строки, на которую по внешнему ключу ссылается какая-либо строка из дочерней таблицы, закончится ошибкой.

Если при установленном сценарии RESTRICT необходимо удалить (или изменить) строки из родительской таблицы, то предварительно необходимо удалить связанные строки из дочерней таблицы.

Если в команде не указывается ON DELETE и ON UPDATE, то по умолчанию должен действовать сценарий ON DELETE RESTRICT и ON UPDATE RESTRICT соответственно.

Количество внешних ключей, заданных в одной дочерней таблице, может быть больше одного, но будет возвращена ошибка если попытаться:

– указать в одной команде больше одного внешнего ключа с одинаковыми характеристиками;

– создать разными командами для одной и той же пары дочерней и родительской таблиц второй внешний ключ с теми же столбцами в составе.

6) <свойства_столбца> — дополнительные свойства столбца, которые можно указать при создании таблицы:

```
<свойства_столбца> ::= <свойство_столбца> [<свойство_столбца>...]
            | GENERATED [ALWAYS | BY DEFAULT] AS IDENTITY
                  [(<свойство_последовательности>[ <свойство_последовательности>...])]

<свойство_столбца> ::= [COLLATE <правило_сравнения>] [DEFAULT <выражение>]
            | [COLLATE <правило_сравнения>] [<вычисляемый_столбец>]

<вычисляемый_столбец>::= [GENERATED ALWAYS] AS (<выражение>) [VIRTUAL]
```

Где:

– DEFAULT <выражение> — конструкция используется для указания значения по умолчанию для столбца, которое будет добавлено в поле, если при вставке данных в таблицу командой INSERT этот столбец явно не упоминается. Конструкция DEFAULT

<выражение> может располагаться в команде как до, так и после ограничений столбца;

– COLLATE <правило_сравнения> — конструкция используется для указания имени правила сравнения, необходимого при сортировке данных столбца. Использование COLLATE применимо только к полям, которые имеют символьный тип данных (см. п. 12.1. Типы данных).

В именах правил сравнения встречаются следующие элементы, понимание которых может облегчить выбор того или иного правила:

– аббревиатура кодировки, например, UTF-8;

– применение регистрозависимого сравнения — CS (case sensitive);

– применение регистронезависимого сравнения — CI (case insensitive);

– использование лексикографического порядка слов при сравнении - LX (Lexicographic order).

Бинарный принцип сравнения BINARY, при котором символы сравниваются в своем байтовом представлении, применим ко всем кодировкам и используется по умолчанию.

Выбранное правило сравнения должно быть совместимо с кодировкой поля таблицы, иначе будет возвращена ошибка. Поэтому, перед указанием параметров COLLATE, важно знать кодировку поля таблицы.

В текущей версии СУБД СОКОЛ кодировки данных задаются на уровне БД, поэтому правила сравнения привязаны к используемым в БД кодировкам (см. Приложение 3).

Узнать кодировку данных БД можно запросом:

```
select * from sys._vdatabase_options where name = 'database.charset';
```

– <вычисляемый_столбец> — столбец таблицы, который не хранится на диске и значения которого вычисляются при запросе.

```
<вычисляемый_столбец>::= [GENERATED ALWAYS] AS (<выражение>) [VIRTUAL]
```

Тип данных вычисляемого столбца можно не указывать. В этом случае тип данных столбца определяется на основе типа данных выражения, указанного для вычисления значения вычисляемого столбца.

Нельзя добавить или изменить значение в вычисляемом столбце, явно указав его в запросах INSERT или UPDATE. Однако, значение вычисляемого столбца меняется при изменении данных, используемых для вычисления его значения;

– [GENERATED ALWAYS] AS (<выражение>) [VIRTUAL] — указывает на то, что значения данного столбца вычисляются на основе заданного <выражения>. Ключевые слова GENERATED ALWAYS и VIRTUAL не обязательны для указания.

Если тип данных для вычисляемого столбца опущен, то он определяется на основе типа результата <выражения>.

Важно: указание вычисляемого столбца в выражении для вычисляемого столбца запрещено. В этом случае выполнение команды будет прекращено с возвратом ошибки.

– GENERATED AS IDENTITY — столбец таблицы с числовым типом данных (далее – столбец с автонумерацией), который автоматически заполняется значениями из генератора последовательности, в соответствии с заданными <свойство_последовательности>[<свойство_последовательности>...] генератора;

– <свойство_последовательности> — свойство генератора последовательности, в соответствии с которым должны генерироваться значения. Если свойства генератора последовательности не указаны, то должны использоваться значения по умолчанию (подробнее о свойствах генератора последовательности см. в п. 12.8.1.4 CREATE SEQUENCE).

Для столбца с автонумерацией ограничение NOT NULL устанавливается по умолчанию, а NULL устанавливать запрещено.

В таблице допускается создание не более одного столбца с автонумерацией, иначе будет возвращена ошибка. Столбец с автонумерацией может заполняться значениями по двум сценариям:

– ALWAYS — указывает, что значения в столбце формируются исключительно на основе генератора последовательности. Указание значения столбца при добавлении строки таблицы или попытка изменения существующего значения столбца будет приводить к ошибке. Сценарий заполнения ALWAYS используется по умолчанию;

– BY DEFAULTY — указывает, что значения в столбце формируются на основе генератора последовательности в том случае, если значение явно не указано при добавлении строки таблицы.

7) <ограничение_таблицы> — ограничения, накладываемые на данные таблицы:

```
<ограничение_таблицы>::= [CONSTRAINT <имя_ограничения>] <тип_ограничения>

<тип_ограничения>  ::= [PRIMARY KEY (<столбцы>)]
                | [UNIQUE (<столбцы>)]
                | [CLUSTERED KEY (<столбцы>)]
                | [UNIQUE CLUSTERED KEY (<столбцы>)]
                | CHECK (<условие>)
                | конструкция <FOREIGN_KEY>

<столбцы> ::= <имя_столбца>[, <имя_столбца>...]
```

Синтаксисы конструкций ограничений, накладываемых на таблицу, схожи с рассмотренными ранее в ограничениях столбца. Отличие имеет лишь синтаксис конструкции <FOREIGN_KEY>:

```
конструкция <FOREIGN_KEY> ::= FOREIGN KEY (<имя_столбца>[, <имя_столбца>, ...])
                        REFERENCES <имя_родительской_таблицы>
                         [(<имя_рт_столбца>)] [<сценарий_поведения>]
```

Действие ограничений таблицы аналогично указанным в <ограничение_столбца> с той разницей, что они применяются к совокупности столбцов. При этом, порядок указанных столбцов в дочерней таблице должен совпадать с порядком указанных столбцов в родительской таблице.

Длина любого ключа таблицы, по которому строится индекс, не может иметь длину больше, чем 2000 байт (формулу расчета см. в п. 12.5.4 Индекс (INDEX)).

Примеры применения команды CREATE TABLE — создание базовой таблицы

Для начала создадим самую простую таблицу PHONE с указанием наименований полей и типов данных для каждого из них:

```
create table PHONE (
NUMBER bigint,
NAME varchar(100),
CONNECTIONDATE date,
AGE int,
CITY varchar (150));
```

В данной таблице по столбцам не указано никаких ограничений, что не позволит отклонять, например, неуникальные или пустые значения. Для этого необходимо указать ограничения по необходимым столбцам.

Пример с применением параметров <ограничение_столбца>

Для примера создадим таблицу PHONE, которая хранит данные о номерах телефонов и абонентах. Для поддержания уникальности значений номеров телефонов и исключения отсутствия имени абонента установим ограничения для соответствующих столбцов:

```
create table PHONE (
NUMBER bigint primary key,
NAME varchar(200) not null,
AGE int,
CITY varchar (150)
);
```

Установленное ограничение:

– PRIMARY KEY по столбцу NUMBER позволяет отклонять не уникальные значения;

– NOT NULL по столбцу NAME позволяет отклонять пустые значения.

Внесём первые данные:

```
insert into PHONE (NUMBER, NAME, AGE, CITY) values (89876543212, 'Петров', 53, 'Воронеж');
```

Выполним запрос для просмотра всех данных таблицы:

```
select * from PHONE;
```

В результате получаем:

```
 NUMBER     | NAME   | AGE   | CITY
------------+---------+-------+-----------
 89876543212 | Петров  | 53    | Воронеж
```

Далее попробуем внести данные, дублирующие значения в столбце NUMBER:

```
insert into PHONE (NUMBER, NAME, AGE, CITY) values (89876543212, 'Сидоров', 46, 'Москва');
```

В соответствии с установленным ограничением в виде PRIMARY KEY будет возвращена ошибка, и данные таблицы остаются неизменными:

```
select * from PHONE;

 NUMBER     | NAME    | AGE   | CITY
------------+---------+-------+-----------
 89876543212 | Петров  | 53    | Воронеж
```

Следующим запросом пробуем внести данные по всем столбцам, кроме поля NAME:

```
insert into PHONE (NUMBER, AGE, CITY) values (89876543214, 46, 'Москва');
```

В соответствии с установленным ограничением в виде NOT NULL будет возвращена ошибка.

Пример с применением дополнительных свойств столбца

Допустим, при внесении информации об абонентах на начальном этапе могут отсутствовать какие-либо данные, которые впоследствии необходимо внести. Для удобства отслеживания, можно задать значение по умолчанию, по которому можно будет фильтровать незанесённые данные.

Рассмотрим пример с указанием дополнительного свойства DEFAULT. Создадим для примера таблицу PHONE, содержащую поле CITY со значением по умолчанию <не задан>:

```
create table PHONE (
NUMBER bigint,
NAME varchar(200),
AGE int,
CITY varchar (150) default '<не задан>'
);
```

Добавляем данные во все столбцы таблицы, кроме CITY:

```
insert into PHONE (NUMBER, NAME, AGE) values (89876543212, 'Петров', 53);
```

Для сравнения добавляем данные во все столбцы таблицы:

```
insert into PHONE values (89876543213, 'Иванов', 28, 'Москва');
```

Извлекаем данные из таблицы PHONE:

```
select * from PHONE;
```

В результате видим, что по столбцу CITY в первом случае значение вставлено в соответствии с указанными параметрами DEFAULT, а во втором случае — указанное при внесении значение:

```
NUMBER      | NAME    | AGE    | CITY
------------+---------+--------+-----------
 89876543212 | Петров  | 53     | <не задан>
 89876543213 | Иванов  | 28     | Москва
```

Пример с указанием дополнительного свойства столбца — COLLATE

Для демонстрации работы разных правил сравнения, создадим таблицу PHONE с данными абонентов и для двух строковых столбцов укажем свойство COLLATE с разными правилами сравнения (с учётом того, что в БД задана кодировка по умолчанию UTF8):

```
create table PHONE (
NUMBER bigint,
NAME varchar (200) collate UCA_UTF8_LX_CS,
CITY varchar (150) collate UCA_UTF8_LX_CI
);
```

Внесём в строковые столбцы значения, одинаковые по смыслу, но с разным буквенным регистром:

```
INSERT INTO PHONE values (89876543212, 'Петров', 'Воронеж');
INSERT INTO PHONE values (89876543213, 'ПЕТРОВ', 'ВОРОНЕЖ');
```

Сделаем выборку данных с условием по полю NAME, для которого было задано регистрозависимое правило сравнения:

```
select * from PHONE where NAME = 'Петров';
```

Будет получен результат, в котором регистры букв в поле NAME соответствуют заданному значению:

```
NUMBER      | NAME    | CITY
------------+---------+-----------
 89876543212 | Петров  | Воронеж
```

Теперь сделаем выборку данных с условием по полю CITY, где было задано регистронезависимое правило сравнения:

```
select * from PHONE where CITY = 'Воронеж';
```

В полученную выборку попадают все значения по полю CITY, которые соответствуют заданному условию независимо от регистра букв:

```
NUMBER      | NAME    | CITY
------------+----------+---------
89876543212  | Петров   | Воронеж
89876543213  | ПЕТРОВ   | ВОРОНЕЖ
```

Пример создания и работы с вычисляемым столбцом.

Для демонстрации работы с вычисляемым столбцом создадим таблицу PHONE с данными абонентов и для удобства доступа к полному имени абонента определим дополнительный столбец, в котором будут представлены объединённые данные имени и фамилии. Постоянного хранения такой столбец не требует, поэтому сделаем его вычисляемым и зададим способ его вычисления:

```
create table PHONE (
  NUMBER bigint,
  NAME varchar(200) not null,
  SURNAME varchar (200) not null,
  FULLNAME varchar (500) generated always as (NAME||' '||SURNAME) VIRTUAL);
```

Внесём данные в таблицу:

```
  insert into PHONE values (89876543212, 'Иван', 'Петров');
  insert into PHONE values (89876543213, 'Пётр', 'Иванов');
```

Извлечём данные из таблицы PHONE:

```
  select * from PHONE;
```

В результате видим, что значения по столбцу FULLNAME вычисляются автоматически, в соответствии с указанным при создании таблицы правилом:

```
   NUMBER     | NAME  | SURNAME  | FULLNAME
  ------------+-------+----------+--------------
   89876543212 | Иван  | Петров   | Иван Петров
   89876543213 | Пётр  | Иванов   | Пётр Иванов
```

Теперь попробуем внести данные явно во все столбцы таблицы, включая вычисляемый:

```
insert into PHONE (NUMBER, NAME, SURNAME, FULLNAME)
  values (89876543214, 'Иван', 'Иванов', 'Иван Иванов');
```

На этот запрос будет возвращена ошибка.

Пример применения параметра <ограничение_таблицы>

Рассмотрим пример, когда важно, чтобы в таблицу вносились в совокупности уникальные данные каждого сотрудника. Например: "дата приёма + имя + отдел". Для этого воспользуемся конструкцией <ограничение_таблицы>.

Для примера, создадим таблицу PHONE с указанием совокупного ограничения:

```
create table PHONE (
ADMISSION date,
NAME varchar(200),
DEPARTMENT varchar(100),
unique clustered key (ADMISSION, NAME, DEPARTMENT)
);
```

Следующим запросом вносим данные по всем столбцам:

```
insert into PHONE values ('12.05.2017', 'Пётр Иванов', 'Отдел кадров');
```

При попытке внести данные, равные по значению по всем столбцам, по которым указано ограничение:

```
insert into PHONE values ('12.05.2017', 'Пётр Иванов', 'Отдел кадров');
```

будет возвращена ошибка.

Внесём данные, имеющие одинаковое значение в столбцах NAME и DEPARTMENT, но отличные по столбцу ADMISSION:

```
insert into PHONE values ('25.05.2021', 'Пётр Иванов', 'Отдел кадров');
```

Данный запрос не нарушает установленных ограничений таблицы и будет выполнен успешно.

Извлекаем данные из таблицы PHONE:

```
select * from PHONE;
```

В полученном результате видим, что данные по всем столбцам внесены:

```
ADMISSION          | NAME      | DEPARTMENT
-----------—-----———--+----------—--+---------------
 2017-05-12 00:00:00| Пётр Иванов| Отдел кадров
 2021-05-25 00:00:00| Пётр Иванов| Отдел кадров
```

Вернуться в оглавление

12.8.1.1.2. Синтаксис создания временной таблицы

Как описано ранее, временные таблицы в СУБД СОКОЛ реализованы в двух вариантах:

– глобальные — GLOBAL TEMPORARY TABLE;

– приватные — PRIVATE TEMPORARY TABLE.

Подробнее о таблицах см. п. 12.5.1. Таблица (TABLE).

Важно: индекс для временной глобальной таблицы должен быть построен до вставки первой записи, иначе при работе с её данными может быть возвращена ошибка. При возникновении подобной ситуации необходимо очистить временную таблицу и далее начать работу с ней в обычном режиме.

Синтаксис создания приватной временной таблицы:

```
CREATE PRIVATE TEMPORARY TABLE <имя_таблицы> (
    <имя_столбца><тип_данных>[[<свойства_столбца>][<ограничение_столбца>]
    [, <имя_столбца><тип_данных>[<свойства_столбца>][<ограничение_столбца>], ...]
    [, <ограничение_таблицы>]])
    [ON COMMIT PRESERVE DEFINITION | ON COMMIT DROP DEFINITION];
```

Синтаксис создания глобальной временной таблицы:

```
CREATE GLOBAL TEMPORARY TABLE <имя_таблицы> (
    <имя_столбца><тип_данных>[[<свойства_столбца>][<ограничение_столбца>]
    [, <имя_столбца><тип_данных>[<свойства_столбца>][<ограничение_столбца>], ...]
    [, <ограничение_таблицы>]])
    [ON COMMIT PRESERVE ROWS | ON COMMIT DELETE ROWS];
```

Где:

– PRIVATE — указывает на создание приватной временной таблицы;

– GLOBAL — указывает на создание глобальной временной таблицы;

– <имя_таблицы> — имя создаваемой временной таблицы. Стоит помнить при создании приватных таблиц, что они отличаются от глобальных и обычных таблиц правилом задания имени - первым в имени приватной временной таблицы должен стоять символ "#";

– <имя_столбца> — имя столбца таблицы, которое должно быть уникальным в рамках таблицы и удовлетворять правилам задания имён сущностей;

– <тип_данных> — тип данных столбца, поддерживаемый СУБД СОКОЛ (подробнее о типах см. п. 12.1. Типы данных);

– <свойства_столбца> — дополнительные свойства столбца, которые можно указать при создании таблицы (подробнее см. п. 12.8.1.1.1. Синтаксис создания базовой таблицы);

– <ограничение_столбца> — ограничение, накладываемое на данные столбца (подробнее см. п. 12.8.1.1.1. Синтаксис создания базовой таблицы);

– ON COMMIT — конструкция используется для указания ограничения времени существования временных таблиц и (или) их данных:

1) Для PRIVATE TEMPORARY TABLE:

– PRESERVE DEFINITION — время существования структуры таблицы и её данных не ограничено завершением транзакции, а ограничено только соединением с БД. Структура таблицы вместе с данными будет удалена с завершением соединения с БД;

– DROP DEFINITION — время существования структуры таблицы и её данных ограничено существованием транзакции. Структура таблицы вместе с данными будет удалена с завершением транзакции.

Если не указать одну из опций, то для PRIVATE TEMPORARY TABLE используется по умолчанию DROP DEFINITION.

2) Для GLOBAL TEMPORARY TABLE:

– PRESERVE ROWS — время существования данных таблицы ограничено соединением с БД. Строки будут удалены с завершением соединения с БД, а структура таблицы сохраняется до момента её явного удаления;

– DELETE ROWS — время существования данных таблицы ограничено существованием транзакции, по окончании которой все строки таблицы будут удалены. Структура таблицы сохраняется до момента её явного удаления.

Если не указать одну из опций, то для GLOBAL TEMPORARY TABLE используется по умолчанию DELETE ROWS.

Примеры варианта 2 применения команды CREATE TEMPORARY TABLE — создание временной таблицы

Рассмотрим примеры создания временных таблиц с указанием разных ограничений для времени существования таблиц и (или) их данных.

И для начала рассмотрим пример с GLOBAL TEMPORARY TABLE.

Допустим, в БД компании есть таблица PHONE с данными абонентов по всем городам, где работает компания:

```
 NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
```

```
89876543212 | Петров  | 53   | Воронеж
89876543214 | Иванов  | 46   | Москва
89876543244 | Котов   | 37   | Москва
89876543255 | Попова  | 35   | Воронеж
89876543266 | Лебедев | 42   | Москва
```

Данная таблица PHONE находится в схеме их руководителя подразделения Soqolov. Двое его подчинённых (Petrov и Ivanov) периодически проводят аналитику данных абонентов в разрезе какого-то одного города.

Для удобства работы с частью данных из таблицы PHONE каждому сотруднику можно создать таблицу, куда он будет выгружать необходимую для работы информацию. Таблица с данными для постоянного хранения не нужна, поэтому создаём временную таблицу, причем глобальную, т.к. объектные привилегии на неё можно предоставить нескольким пользователям, но каждый будет иметь доступ только к тем данным, которые он добавил.

Для сравнения поведения данных в таблицах создадим в схеме руководителя подразделения Soqolov две таблицы с разными опциями, имена которых отражают время существования хранящихся в них данных:

```
create global temporary table SAVE_ON_COMMIT
    (NUMBER bigint primary key, NAME varchar(200), AGE int, CITY varchar (150))
       on commit preserve rows;

create global temporary table DELETE_ON_COMMIT
    (NUMBER int primary key, NAME varchar(200), AGE int, CITY varchar (150))
       on commit delete rows;
```

В первой таблице время существования её данных ограничено соединением с БД и данные сохраняются при завершении транзакции. Во второй таблице — ограничено транзакцией и данные удаляются при завершении транзакции.

Для работы с данными предоставим пользователям Petrov и Ivanov все объектные привилегии к данным таблицам:

```
grant all on PHONE to Petrov;
grant all on SAVE_ON_COMMIT to Petrov;
grant all on DELETE_ON_COMMIT to Petrov;

grant all on PHONE to Ivanov;
grant all on SAVE_ON_COMMIT to Ivanov;
```

```
grant all on DELETE_ON_COMMIT to Ivanov;
```

Теперь каждый сотрудник добавит в таблицу SAVE_ON_COMMIT те данные, которые ему нужны для работы из таблицы PHONE:

Petrov:	**Ivanov:**
insert into Soqolov.SAVE_ON_COMMIT select NUMBER, NAME, AGE, CITY from PHONE where CITY = 'Москва';	insert into Soqolov.SAVE_ON_COMMIT select NUMBER, NAME, AGE, CITY from Petrov.PHONE where CITY = 'Воронеж';

После успешного выполнения команды каждый делает выборку полных данных из таблицы SAVE_ON_COMMIT:

Petrov:	**Ivanov:**
select * from Soqolov.SAVE_ON_COMMIT;	select * from Soqolov.SAVE_ON_COMMIT;

И получает данные:

Petrov:	**Ivanov:**
NUMBER | NAME | AGE | CITY ----------+--------+------+-------- 89876543214 | Иванов | 46 | Москва 89876543244 | Иванов | 37 | Москва	NUMBER | NAME | AGE | CITY ----------+--------+------+-------- 89876543212 | Петров | 53 | Воронеж 89876543255 | Иванов | 35 | Воронеж

Из выполненных запросов и полученных данных видно, что в каждой сессии создается свой физический экземпляр таблицы SAVE_ON_COMMIT.

Если пользователь Ivanov отсоединится от БД и, подключившись снова к ней, выполнит попытку запроса на выборку данных из таблицы SAVE_ON_COMMIT:

```
select * from Soqolov.SAVE_ON_COMMIT;
```

Ему вернётся пустая таблица, т.к. данные хранились в рамках соединения с базой данных.

Если Petrov при включенном режиме AUTOCOMMIT выполнит вставку данных в таблицу DELETE_ON_COMMIT и отправит запрос на вывод данных из неё, а Ivanov выполнит тоже самое, но предварительно начнёт транзакцию в ручном режиме:

Petrov:	**Ivanov:**
В режиме AUTOCOMMIT insert into Soqolov.DELETE_ON_COMMIT select NUMBER, NAME, AGE, CITY from PHONE where CITY = 'Москва'; select * from Soqolov.DELETE_ON_COMMIT;	start transaction; insert into Soqolov.DELETE_ON_COMMIT select NUMBER, NAME, AGE, CITY from Petrov.PHONE where CITY = 'Воронеж'; select * from Soqolov.DELETE_ON_COMMIT;

В этом случае они получат разный ответ — у одного пустой, у второго будут отражены вставленные данные:

Petrov:	**Ivanov:**
NUMBER | NAME | AGE | CITY ----------+--------+------+--------	NUMBER | NAME | AGE | CITY ----------+--------+------+----- 89876543212 | Петров | 53 | Воронеж 89876543255 | Иванов | 35 | Воронеж

Это связано с тем, что в таблице DELETE_ON_COMMIT при создании была указана опция DELETE ROWS, поэтому время существования данных ограничено существованием транзакции. В режиме AUTOCOMMIT после завершения транзакции все строки таблицы удаляются.

Сотрудник Ivanov выполнил запрос данных в незавершённой транзакции, поэтому по его запросу данные были отражены. После завершения транзакции пользователем Ivanov данные в его таблице также удалятся.

Теперь рассмотрим пример с PRIVATE TEMPORARY TABLE.

Отличительна черта приватной временной таблицы состоит в том, что она не видна никому, кроме её владельца.

Также для сравнения, пользователь Petrov в своей схеме создаёт две таблицы PRIVATE, имена которых отражают время существования хранящихся в них данных:

```
create private temporary table #PRESERVE_ON_COMMIT
    (NUMBER bigint primary key, NAME varchar(200))
      on commit preserve definition;

create private temporary table #DROP_ON_COMMIT
    (NUMBER bigint primary key, NAME varchar(200))
      on commit drop definition;
```

В режиме AUTOCOMMIT пользователь Petrov добавляет в каждую таблицу данные из таблицы PHONE и отправляет запрос на выборку всех данных:

#PRESERVE_ON_COMMIT	#DROP_ON_COMMIT
insert into #PRESERVE_ON_COMMIT select NUMBER, NAME from PHONE where CITY = 'Москва'; select * from #PRESERVE_ON_COMMIT;	insert into #DROP_ON_COMMIT select NUMBER, NAME from PHONE where CITY = 'Москва'; select * from #DROP_ON_COMMIT

После успешного выполнения запросов будут отличные результаты:

#PRESERVE_ON_COMMIT	#DROP_ON_COMMIT
NUMBER | NAME ----------+---------- 89876543214 | Иванов 89876543244 | Иванов	Возвращена ошибка

При выполнении запроса на выборку данных из первой таблицы возвращены добавленные данные, т.к. таблица и её данные хранятся в рамках соединения с БД.

При создании таблицы #DROP_ON_COMMIT была указана опция DROP DEFINITION, поэтому время существования структуры таблицы и её данных ограничено существованием транзакции. В режиме AUTOCOMMIT после завершения транзакции структура таблицы и её данные удаляются, поэтому и была возвращена ошибка.

Для успешного выполнения запроса, по аналогии с примером для глобальной таблицы DELETE_ON_COMMIT, необходимо начать транзакцию в ручном режиме.

Попробуем предоставить объектные привилегии на одну из приватных таблиц другому пользователю:

```
grant all on #PRESERVE_ON_COMMIT to Ivanov;
```

Приватные таблицы доступны только владельцу, поэтому и объектные привилегии на них предоставить нельзя. На данный запрос будет возвращена ошибка.

Вернуться в оглавление

12.8.1.2. CREATE VIEW

Команда создаёт представление на основании запроса SELECT. Подробно о представлениях см. п. 12.5.2. Представление (VIEW).

Создать представление может:

– пользователь базы данных с системной привилегией RESOURCES CONTROL в схеме, владельцем которой он является;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды создания представления:

```
CREATE VIEW <имя_представления> [(<список_столбцов>)] AS <выборка_данных>;

<список_столбцов>::= <имя_столбца>[, <имя_столбца>...]
```

Где:

– <имя_представления> — имя создаваемого представления;

– <имя_столбца> — имя столбца создаваемого представления, которое становится новым именем для столбца выборки.

Если не указать параметр <список_столбцов>, то столбцы представления приобретут те же имена, что и столбцы таблицы, указанные в <выборка_данных> (SELECT-запрос).

При указании параметра <список_столбцов> нужно учитывать, что число указанных столбцов представления должно быть равно числу столбцов, полученных в результате выполнения SELECT. В противном случае будет возвращена ошибка и представление не будет создано;

– <выборка_данных> — представляет собой выборку данных из БД при помощи SELECT-запроса.

В выборке недопустимо использование временных таблиц.

Подробнее о команде SELECT см. п. 12.8.5. SELECT.

Пример создания представления

Для демонстрации создания представления и работы с ним создадим таблицу WORKERS, в которой будут собраны данные по сотрудникам компании (табельный номер, имя и фамилия, возраст, пол):

```
create table WORKERS (TN int PRIMARY KEY, NAME varchar (200), LASTNAME varchar (200), AGE int, GENDER varchar (3));
```

Наполним новую таблицу данными сотрудников компании:

```
insert into WORKERS values (212, 'Иван', 'Петров', 53, 'м');
insert into WORKERS values (213, 'Пётр','Иванов', 27, 'м');
insert into WORKERS values (214, 'Игорь', 'Котов', 38, 'м');
insert into WORKERS values (215, 'Анна', 'Гузеева', 70, 'ж');
insert into WORKERS values (216, 'Юлия', 'Серова', 65, 'ж');
insert into WORKERS values (217, 'Елена', 'Попова', 19, 'ж');
insert into WORKERS values (218, 'Ирина', 'Зотова', 48, 'ж');
insert into WORKERS values (219, 'Фёдор', 'Уваров', 67, 'м');
```

Допустим, что необходимо предоставить часть информации по сотрудникам, в то время как демонстрация всех данных таблицы нежелательна.

Создадим представление с выборкой данных по двум запрашиваемым столбцам из созданной таблицы WORKERS (без указания наименования столбцов):

```
create view WORKERS_1 AS SELECT TN, LASTNAME FROM WORKERS;
```

Извлечём данные из представления:

```
select * from WORKERS_1;
```

В результате видим в представлении информацию только по двум необходимым столбцам, и наименования столбцов представления приобрели те же имена, что в исходной таблице:

```
TN       | LASTNAME
-----------------+-------------
 212     | Петров
 213     | Иванов
 214     | Котов
 215     | Гузеева
 216     | Серова
 217     | Попова
 218     | Зотова
 219     | Уваров
```

Допустим, по запросу необходимо некоторые данные объединить (например, имя и фамилию сотрудника). В этом случае при создании представления можно указать и новые наименования столбцов:

```
create view WORKERS_2 ("Полное имя", "Возраст")
  as select NAME || ' ' || LASTNAME, AGE from WORKERS;
```

Извлечём данные из представления WORKERS_2 и видим, что столбцы представления имеют наименования в соответствии с указанными именами при создании:

```
 Полное имя     | Возраст
----------------+----------
 Иван Петров    | 53
 Пётр Иванов    | 27
 Игорь Котов    | 38
 Анна Гузеева   | 70
 Юлия Серова    | 65
 Елена Попова   | 19
 Ирина Зотова   | 48
 Фёдор Уваров   | 67
```

Попробуем создать представление, указав меньше наименований столбцов, чем указано для выборки:

```
create view WORKERS_3 ("Полное имя") as select NAME || ' ' || LASTNAME, AGE from WORKERS;
```

Данный запрос завершится с ошибкой.

Далее создадим представление с выборкой женской части коллектива:

```
create view WORKERS_3 ("Табельный номер", "Полное имя", "Возраст")
  as select TN, NAME || ' ' || LASTNAME, AGE
    from WORKERS where GENDER='ж';
```

Извлечём данные из представления WORKERS_3 и увидим лишь ту часть данных из таблицы WORKERS, которая удовлетворяет заданному в WHERE условию:

```
 Табельный номер  | Полное имя   | Возраст
------------------+--------------+----------
 215              | Анна Гузеева | 70
 216              | Юлия Серова  | 65
 217              | Елена Попова | 19
 218              | Ирина Зотова | 48
```

Вернуться в оглавление

12.8.1.3. CREATE INDEX

Команда создаёт индекс по заданным столбцам конкретной таблицы.

Важно: для временной глобальной таблицы индекс должен быть построен до вставки первой записи, иначе при работе с её данными может быть возвращена ошибка. При возникновении подобной ситуации необходимо очистить временную таблицу и далее начать работу с ней в обычном режиме.

Подробно об индексах см. п. 12.5.4 Индекс (INDEX).

Создать индекс может:

– пользователь базы данных с системной привилегией RESOURCES CONTROL в схеме, владельцем которой он является;

– пользователь базы данных с системной привилегией RESOURCES CONTROL и объектной привилегией INDEX в чужой схеме на ту таблицу, на которую ему предоставлена объектная привилегия INDEX;

– пользователь базы данных с системной привилегией DATABASE ADMIN в любой схеме.

Синтаксис команды создания индекса:

```
CREATE [UNIQUE] INDEX <имя_индекса> ON <имя_таблицы> (<список_столбцов>);

<список_столбцов>::= <имя_столбца>[, <имя_столбца>...]
```

Где:

– UNIQUE — указывает на то, что индекс должен сохранять внутри таблицы БД состояние уникальности значений по совокупности заданных столбцов, по которым он построен, блокируя попытки нарушения уникальности (при изменении значений или добавлении новых значений). Если изначально совокупные значения столбцов, по которым планируется построить уникальный индекс, не обладают характеристикой уникальности, то в ответ на команду создания индекса с характеристикой UNIQUE будет возвращена ошибка.

Важно: при построении уникального индекса по столбцам обычной таблицы допускается несколько полей со значением NULL. Для временных таблиц построение уникального индекса допустимо только по столбца с характеристикой NOT NULL. Иначе будет возвращена ошибка.

– <имя_индекса> — имя создаваемого индекса;

– <имя_таблицы> — имя таблицы, по столбцам которой создаётся индекс;

– <имя_столбца> — имя столбца таблицы (или одного из столбцов), по которому будет построен индекс.

При попытке создания индексов одинаковых по набору и порядку следования столбцов для одной и той же таблицы выполнение команды будет прекращено с возвращением ошибки.

Индекс не может быть построен по столбцам с данными типа CLOB и BLOB.

Пример создания индекса

Для демонстрации работы с индексом создадим таблицу GORVODA с данными водоснабжающей организации, ведущей учёт показаний счётчиков воды (см. рис. 12):

```
create table GORVODA ("Улица" varchar (200), "N дома" bigint, "N кв" bigint, "ФИО владельца" varchar (300), "№ телефона" int, "Предыдущ показ, м3" bigint, "Текущие показ, м3" bigint);
```

Наполним новую таблицу данными:

```
insert into GORVODA values ('Вишнёвая', 32, 216, 'Иванов А.С.', 89872222211, 125, 128);
insert into GORVODA values ('Грушевая', 5, 345, 'Петров Ф.К.', 89871111122, 325, 330);
```

```
insert into GORVODA values ('Вишнёвая', 25, 126, 'Иванов А.С.', 89872222211, 27, 32);
insert into GORVODA values ('Грушевая', 5, 126, 'Зотов С.К.', 89875555588, 1265, 1232);
insert into GORVODA values ('Грушевая', 5, 25, 'Серова И.А.', 89873333344, 27, 39);
insert into GORVODA values ('Вишнёвая', 25, 132, 'Уткина М.Е.', 89874444455, 38, 45);
insert into GORVODA values ('Вишнёвая', 8, 5, 'Уткина М.Е.', 89874444455, 48, 57);
```

Извлечём данные из таблицы:

```
select * from GORVODA;
```

В результате видим, что таблица заполнена в соответствии с внесённой информацией:

```
Улица    | N дома | N кв  | ФИО владельца |  телефона   | Предыдущ показ, м3 | Текущие показ, м3
---------+--------+-------+---------------+-------------+--------------------+-------------------
Вишнёвая | 32     | 216   | Иванов А.С.   | 89872222211 | 125                | 128
Грушевая | 5      | 345   | Петров Ф.К.   | 89871111122 | 325                | 330
Вишнёвая | 25     | 126   | Иванов А.С.   | 89872222211 | 27                 | 32
Грушевая | 5      | 126   | Зотов С.К.    | 89875555588 | 1265               | 1232
Грушевая | 5      | 25    | Серова И.А.   | 89873333344 | 27                 | 39
Вишнёвая | 25     | 132   | Уткина М.Е.   | 89874444455 | 38                 | 45
Вишнёвая | 8      | 5     | Уткина М.Е.   | 89874444455 | 48                 | 57
```

Далее построим простой индекс — по одному столбцу:

```
create index GORVODA_IDX1 on GORVODA ("Улица");
```

Такой индекс ускорит поиск информации по улице:

```
select * from GORVODA where "Улица"='Вишнёвая';
```

```
Улица    | N дома | N кв  | ФИО владельца |  телефона   | Предыдущ показ, м3 | Текущие показ, м3
---------+--------+-------+---------------+-------------+--------------------+-------------------
Вишнёвая | 32     | 216   | Иванов А.С.   | 89872222211 | 125                | 128
Вишнёвая | 25     | 126   | Иванов А.С.   | 89872222211 | 27                 | 32
Вишнёвая | 25     | 132   | Уткина М.Е.   | 89874444455 | 38                 | 45
Вишнёвая | 8      | 5     | Уткина М.Е.   | 89874444455 | 48                 | 57
```

Допустим, что водоснабжающая организация часто обращается к БД для поиска информации об абоненте, зарегистрированном по определённому адресу (улица, № дома, № квартиры).

Для эффективного поиска такой информации желательно построить составной индекс по столбцам, определяющим этот адрес:

```
create index GORVODA_IDX2 on GORVODA ("Улица", "N дома", "N кв");
```

Обратим внимание, что такой индекс позволит вносить в таблицу значения, не уникальные в совокупности по трём столбцам, по которым построен индекс.

Проверим это, добавив новую запись с дублирующимися в этих трёх столбцах значениями:

```
insert into GORVODA values ('Вишнёвая', 8, 5, 'Фролов А.Е.', 89871111111, 53, 56);
```

Извлечём данные из таблицы и увидим, что данные внесены:

```
Улица     | N дома| N кв| ФИО владельца |  телефона   |Предыдущ показ, м3|Текущие показ, м3
------—---+-------+-----+---------------+---------- --+------------------+------------------
Вишнёвая | 32    | 216 | Иванов А.С.   | 89872222211 | 125           | 128
Грушевая | 5     | 345 | Петров Ф.К.   | 89871111122 | 325           | 330
Вишнёвая | 25    | 126 | Иванов А.С.   | 89872222211 | 27            | 32
Грушевая | 5     | 126 | Зотов С.К.    | 89875555588 | 1265          | 1232
Грушевая | 5     | 25  | Серова И.А.   | 89873333344 | 27            | 39
Вишнёвая | 25    | 132 | Уткина М.Е.   | 89874444455 | 38            | 45
Вишнёвая | 8     | 5   | Уткина М.Е.   | 89874444455 | 48            | 57
Вишнёвая | 8     | 5   | Фролов А.Е.   | 89871111111 | 53            | 56
```

Как видим, такая структура индекса позволяет вносить ошибочные данные в таблицу (одному адресу ошибочно соответствуют разные абоненты, имеющие свои счетчики воды).

Собирая информацию о показаниях счётчиков воды, водоснабжающая организация считает, что в базе данных по отдельности значения столбцов "Улица", "N дома", "N кв" могут повторяться, но в совокупности эти данные повторяться не должны.

Внесём изменения, позволяющие блокировать внесение дублирующих данных на уровне СУБД. Для этого построим по трём столбцам данной таблицы уникальный составной индекс:

```
create unique index GORVODA_IDX3 on GORVODA ("Улица", "N дома", "N кв");
```

Но при текущем состоянии таблицы индекс создан не будет в связи с тем, что совокупность значений столбцов, на которых основан индекс, не обладает характеристикой уникальности.

Перед построением уникального индекса устраним дублирование записей путём удаления дублирующей записи:

```
delete from GORVODA where "Улица"='Вишнёвая' and "N дома"=8 and "N кв"=5 and "ФИО
владельца"='Фролов А.Е.';
```

Теперь команда построения уникального составного индекса будет выполнена успешно:

```
create unique index GORVODA_IDX3 on GORVODA ("Улица", "N дома", "N кв");
```

Покажем, что построенный индекс поддерживает совокупную уникальность значений по трём столбцам, выбранным для построения индекса.

Для этого еще раз внесём данные, дублирующие значения в совокупности по трём столбцам, выбранным для индекса:

```
insert into GORVODA values ('Вишнёвая', 8, 5, 'Фролов А.Е.', 89871111111, 53, 56);
```

Будет возвращена ошибка, и данные не будут внесены в таблицу.

Как видим, благодаря построенному уникальному индексу в таблице БД сохраняется требуемая уникальность совокупности значений, что устраняет внесение ошибочных данных.

Продемонстрируем работу индекса, выполнив запрос для поиска показаний счётчика по определённому адресу:

```
select * from GORVODA where "Улица"='Вишнёвая' and "N дома"=25 and "N кв"=126;
```

Будет получена соответствующая информация:

```
Улица    | N дома| N кв| ФИО владельца| телефона   |Предыдущ показ, м3|Текущие показ, м3
---------+-------+-----+--------------+------------+------------------+-----------------Вишнёвая | 25    | 126 | Иванов А.С.  | 89872222211| 27
| 32
```

Для выполнения этого запроса серверу базы данных не пришлось сканировать все строки таблицы, чтобы проверить каждую запись на соответствие. В данном случае, был выполнен обход дерева индекса, и запрос был выполнен значительно быстрее. Особенно это заметно при работе с большими объёмами данных.

Вернуться в оглавление

12.8.1.4. CREATE SEQUENCE

Команда создаёт генератор последовательности. Подробно о генераторе последовательности см. п. 12.5.3. Генератор последовательности (SEQUENCE).

Создать генератор последовательности может:

– пользователь базы данных с системной привилегией RESOURCES CONTROL в схеме, владельцем которой он является;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды создания генератора последовательности:

```
CREATE SEQUENCE <имя_последовательности>
    [<свойство_последовательности> [<свойство_последовательности>...]];


<свойство_последовательности>::= INCREMENT BY <значение>
                                | START WITH <значение>
                                | SPREAD
                                | [NOMAXVALUE | MAXVALUE <значение>]
                                | [MINVALUE <значение> | NOMINVALUE]
                                | [CYCLE | NOCYCLE]
                                | [CACHE <значение>]
```

Где:

1) <имя_последовательности> — имя создаваемой последовательности.

2) <свойство_последовательности> — свойство последовательности, (необязательный параметр).

3) <значение> — целое число, которое может быть как положительным, так и отрицательным.

4) INCREMENT BY <значение> — конструкция используется для указания значения шага генерируемой последовательности, на которое каждое следующее генерируемое значение изменяется относительно предыдущего после извлечения из генератора. Если указанное <значение> отрицательное, то генерируемое значение уменьшается (убывающая последовательность), если <значение> положительное — увеличивается (возрастающая последовательность). Значение шага не может быть равно 0 и должно быть в рамках диапазона от -2 305 843 009 213 693 952 до 2 305 843 009 213 693 951. По умолчанию значение INCREMENT BY равно 1, если не указано явно.

5) START WITH <значение> — конструкция используется для указания первого значения генерируемой последовательности, которое может превышать минимальное значение восходящей последовательности, или быть меньше максимального значения убывающей последовательности. По умолчанию первое значение — минимальное значение для возрастающей последовательности и максимальное — для убывающей.

6) SPREAD — экспериментальная опция генератора последовательности, которая перед возвратом очередного значения обменивает его младшие разряды, что позволяет распределить выходные значения на значительном расстоянии друг от друга. При указании в команде опции SPREAD:

– игнорируются MINVALUE, MAXVALUE, START WITH и используются их значения по умолчанию;

– генерируемая последовательность будет выдавать уникальные, но не строго возрастающие значения.

Использование значений такой последовательности в primary key существенно снижает конкуренцию при вставке записей. Имеет смысл применять опцию SPREAD при достаточном объёме оперативной памяти (2-4 мегабайт на каждый индекс, построенный по столбцу, значения для которого сгенерированы генератором с параметром SPREAD).

Не рекомендуется применять SPREAD:

– при недостаточном объёме оперативной памяти;

– при отсутствии конкуренции при вставке (однопользовательская работа).

В таких случаях применение SPREAD может привести к снижению производительности;

7) MAXVALUE <значение> — конструкция используется для указания максимального значения, создаваемое последовательностью. Дальнейшие действия генератора, при достижении указанного значения, зависят от указанного свойства: CYCLE / NOCYCLE (по умолчанию NOCYCLE). Имеет смысл указывать значение MAXVALUE при положительном значении шага, а также при отрицательном значении шага. Если MAXVALUE не указано, то применяется значение по умолчанию – NOMAXVALUE.

8) NOMAXVALUE — указывает, что максимального значения для создаваемого генератора последовательности нет. В этом случае, верхней границей будет:

– максимальное значение из допустимого диапазона — для возрастающей последовательности;

– значение -1 — для убывающей последовательности.

9) MINVALUE <значение> — конструкция используется для указания минимального значения, создаваемого последовательностью. Дальнейшие действия генератора, при достижении указанного значения, зависят от указанного свойства: CYCLE / NOCYCLE (по умолчанию NOCYCLE).

Имеет смысл указывать значение MINVALUE при отрицательном значении шага, а также при положительном значении шага. Если MINVALUE не указано, то применяется значение по умолчанию — NOMINVALUE;

10) NOMINVALUE — указывает, что минимального значения для создаваемого генератора последовательности нет. В этом случае, нижней границей будет:

– минимальное значение из допустимого диапазона — для убывающей последовательности;

– значение 1 — для возрастающей последовательности.

11) CYCLE — указывает на возможность последовательности при достижении заданной границы зациклить генерацию значений: при достижении максимального значения возрастающей последовательности (или минимального значения убывающей) следующим будет сгенерировано её минимальное значение (максимальное значение для убывающей последовательности). По умолчанию используется значение NOCYCLE.

Не рекомендуется CYCLE применять для столбцов с ограничением PRIMARY KEY, UNIQUE, т.к. при достижении верхнего порога и начале нового цикла будет возвращена ошибка.

При указании CYCLE обязательно указание MAXVALUE для возрастающей последовательности, MINVALUE — для убывающей последовательности.

Указывать CYCLE вместе с NOMAXVALUE или NOMINVALUE нельзя.

12) NOCYCLE — указывает, что последовательность не сможет продолжить генерацию значений после достижения заданных значений MAXVALUE или MINVALUE. Выполнение команды будет прервано и будет возвращена ошибка.

13) CACHE — указывает на возможность создавать заранее и поддерживать в памяти заданное число значений последовательности для быстрого доступа. Минимальное значение равно 1 и означает, что кеширования нет. Максимально допустимое значение равно 2^{31}. В случае указания значения вне допустимого диапазона будет возвращена ошибка.

Если не указать параметр CACHE явно, то по умолчанию CACHE = 32.

Значения, указываемые для START WITH, MINVALUE, MAXVALUE, должны быть в рамках максимально допустимых значений для последовательности — от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

Пример применения команды CREATE SEQUENCE

Для демонстрации работы со значениями генератора последовательности рассмотрим два варианта: уникальную нумерацию записей в одной таблице и сквозную нумерацию записей в двух таблицах.

Допустим, в таблицу базы данных будут помещаться краткие данные о сотрудниках компании, где они должны приобретать трёхзначные табельные номера. Табельные номера в нашем случае будут генерироваться генератором последовательности.

Создадим генератор SEQ1 с заданными параметрами:

```
create sequence SEQ1 start with 100 increment by 1 maxvalue 999 nocycle;
```

Теперь создадим таблицу WORKERS, для поля <Таб №> которой укажем значение по умолчанию, основанное на значениях генератора последовательности, и ограничение primary key:

```
create table WORKERS ("Таб №" bigint DEFAULT SEQ1.nextval primary key, "ФИО" varchar (150), "Подразделение" varchar (100));
```

Внесём несколько записей, указав значения всех столбцов таблицы, кроме столбца <Таб №>:

```
INSERT INTO WORKERS ("ФИО", "Подразделение") values ('Петров А.В.', 'Отдел кадров');
INSERT INTO WORKERS ("ФИО", "Подразделение") values ('Васечкин П.А.', 'Отдел кадров');
INSERT INTO WORKERS ("ФИО", "Подразделение") values ('Селезнёв Р.И.', 'ПЭО');
INSERT INTO WORKERS ("ФИО", "Подразделение") values ('Котова А.В.', 'МТО');
```

Извлечём данные из таблицы:

```
select * from WORKERS;
```

В полученной таблице увидим, что данные по столбцу <Таб №> заполнены значениями из генератора последовательности:

```
 Таб №      | ФИО          | Подразделение
-------------+----------------+---------------------
 101        | Петров А.В.    | Отдел кадров
 102        | Васечкин П.А.  | Отдел кадров
 103        | Селезнёв Р.И.  | ПЭО
 104        | Котова А.В.    | МТО
```

Допустим, что в компании несколько филиалов. Данные по сотрудникам каждого филиала ведутся в отдельной таблице, но табельные номера уникальные в рамках всей компании и заполняются значениями из одного генератора последовательности.

Для демонстрации данного примера создадим две таблицы для разных городов:

```
create table WORKERS_MOSCOW (
    "Таб №" bigint DEFAULT SEQ1.nextval primary key,
    "ФИО" varchar (150),
    "Подразделение" varchar (100),
    "Город" varchar (100) generated always as ('Москва') virtual);

create table WORKERS_VORONEZH (
    "Таб №" bigint DEFAULT SEQ1.nextval primary key,
    "ФИО" varchar (150),
    "Подразделение" varchar (100),
    "Город" varchar (100) generated always as ('Воронеж') virtual);
```

Внесём несколько записей в каждую таблицу, не указывая значение столбца <Таб №>:

```
INSERT INTO WORKERS_VORONEZH ("ФИО", "Подразделение") values ('Петров А.В.', 'МТО');

INSERT INTO WORKERS_VORONEZH ("ФИО", "Подразделение") values ('Уткин П.А.', 'ПЭО');

INSERT INTO WORKERS_MOSCOW ("ФИО", "Подразделение") values ('Селезнёв Р.И.', 'ПЭО');

INSERT INTO WORKERS_MOSCOW ("ФИО", "Подразделение") values ('Котова А.В.', 'МТО');
```

При внесении записей в таблицы, столбцы <Таб №> будут заполняться значениями со сквозной нумерацией. Это видно при извлечении данных из обеих таблиц:

```
select * from WORKERS_VORONEZH union select * from WORKERS_MOSCOW;

 Таб №      | ФИО          | Подразделение  | Город
```

```
------------+---------------+----------------+----------------
 100        | Петров А.В.   | МТО            | Воронеж
 101        | Уткин П.А.    | ПЭО            | Воронеж
 102        | Селезнёв Р.И. | ПЭО            | Москва
 103        | Котова А.В.   | МТО            | Москва
```

Таким образом, это удобно, например, при распределении данных на несколько таблиц и необходимости сквозной нумерации.

Вернуться в оглавление

12.8.1.5. CREATE SCHEMA

Команда создаёт схему. Подробно о схеме см. п. 12.5.5. Схема (SCHEMA).

Создать схему может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды создания схемы:

```
CREATE SCHEMA <имя_схемы> AUTHORIZATION <имя_пользователя>;
```

Где:

– <имя_схемы> — имя создаваемой схемы;

– <имя_пользователя> — имя зарегистрированного в БД пользователя, который будет владельцем создаваемой схемы.

Пример применения команды CREATE SCHEMA

Допустим, у нас есть две таблицы с данными сотрудников компании. С одной таблицей работают сотрудники отдела кадров, а с другой — сотрудники отдела мотивации. Таблицы одноимённые, но отличаются содержанием.

Продемонстрируем возможность создания одноимённых таблиц в разных схемах и для этого создадим две схемы:

```
create schema "Отдел кадров" authorization IVANOVPA;
create schema "Отдел мотивации" authorization IVANOVPA;
```

Теперь в одной схеме создадим таблицу с данными сотрудников компании:

```
create table "Отдел кадров"."Сотрудники компании"
```

```
("Таб №" int primary key, "ФИО" varchar (150), "Наименование отдела" varchar (350), "Оклад" int, "Возраст" int);
```

В другой схеме создадим одноимённую с первой таблицу (но с отличными характеристиками):

```
create table "Отдел мотивации"."Сотрудники компании"
  ("ФИО" varchar (150), "Наименование отдела" varchar (350), "Оклад" int);
```

Теперь с этими таблицами можно работать, указывая при обращении схему, в которой находится нужная таблица:

```
select * from "Отдел кадров"."Сотрудники компании";

select * from "Отдел мотивации"."Сотрудники компании";
```

Добавление других объектов в существующую схему см. далее в п. 12.8.3.1 ALTER SCHEMA.

Вернуться в оглавление

12.8.1.6. CREATE USER

Команда создаёт нового пользователя. Подробно о пользователе см. п. 12.5.6. Пользователь (USER).

Создать пользователя может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды создания пользователя:

```
CREATE USER <имя_пользователя> <наличие_пароля> [DEFAULT SCHEMA <имя_схемы>];

<наличие_пароля> ::= IDENTIFIED BY '<пароль_пользователя>' | NO AUTHENTICATION
```

Где:

– <имя_пользователя> — имя создаваемого пользователя;

– <пароль_пользователя> — пароль, используемый для подтверждения пользователя в процессе аутентификации.

При указании пароля в текущей версии СУБД СОКОЛ разрешены латинские буквы (в верхнем и нижнем регистре), цифры (от 0 до 9), символы:

```
_:@#$%&?,^-.=+!*)(~|
```

Буквы пароля всегда регистрозависимые. Максимальная длина пароля 64 байта (для кодировки символов пароля используется кодировка базы данных).

Для установки или смены пароля существующего пользователя используйте команду ALTER USER (см. п. 12.8.3.3 ALTER USER);

– NO AUTHENTICATION — указывается для создания пользователя без пароля и возможности аутентификации. Данный параметр можно использовать, например, для создания отдельной схемы без соответствующего активного пользователя или в случае подготовки доступных данных для пользователя, которому позже будет предоставлена возможность аутентификации посредством команды ALTER USER с указанием пароля (см п. 12.8.3.3 ALTER USER);

– <имя_схемы> — указывается имя существующей схемы в БД, которая будет схемой по умолчанию для нового пользователя. В этом случае схема с именем пользователя не будет автоматически создаваться.

Если при создании пользователя не указана схема по умолчанию, то после успешного выполнения команды автоматически будет создана одноименная с пользователем схема, владельцем которой он является. Данная схема будет его схемой по умолчанию.

Пример применения команды CREATE USER

Рассмотрим примеры создания пользователя БД (с паролем и без пароля) для сотрудника компании и влияние параметров, указанных в синтаксисе команды.

Допустим, на работу принят новый сотрудник Петров и ему необходимо предоставить возможность подсоединиться к БД.

Создадим для этого сотрудника пользователя БД с паролем:

```
create user "Петров" IDENTIFIED BY 'joInInG';
```

Попытаемся подсоединиться под новым пользователем к БД любым удобным способом.

Будет возвращена ошибка, т.к. пользователь не может подсоединиться к БД без системной роли CONNECT.

Исправим это и добавим пользователю роль CONNECT:

```
grant CONNECT to "Петров";
```

Теперь подсоединимся под пользователем "Петров" с паролем 'joIninG' к базе данных любым удобным способом. Процесс аутентификации и подсоединения к БД прошёл успешно.

Для выполнения конкретных действий над данными и объектами БД пользователю дополнительно необходимо добавить соответствующие привилегии или роли.

Рассмотрим ещё один вариант: допустим, принят новый сотрудник, который скоро выйдет на работу и ему будет необходим доступ к данным в БД. Для такого сотрудника можно заранее подготовить пользователя, а возможность подсоединения к БД и пароль для аутентификации дать при выходе на работу. Для этого создадим сначала пользователя без пароля:

```
create user "Иванов" NO AUTHENTICATION;
```

При попытке подсоединения под новым пользователем к БД любым способом будет возвращена ошибка, т.к. пользователь не может пройти аутентификацию без пароля и подключиться к БД без системной роли CONNECT.

Предположим, что сотрудник пришёл на рабочее место и его руководитель заявкой администратору БД подтвердил необходимость доступа нового сотрудника к БД. В этом случае администратор установит пользователю пароль и добавит роль CONNECT для возможности аутентификации:

```
alter user "Иванов" IDENTIFIED BY 'oPeniNg';
grant connect to "Иванов";
```

Теперь у созданного пользователя есть системная роль для соединения с базой данных и пароль для возможности аутентификации.

Подсоединимся к БД под пользователем "Иванов" с паролем 'oPeniNg'любым удобным способом.

Процесс аутентификации и подсоединения к БД прошёл успешно.

Подробнее:

– о привилегиях см. п. 12.5.7. Привилегии (PRIV) и роли (ROLE);

– о добавлении привилегий см. п. 12.8.13.1. GRANT PRIV;

– о добавлении ролей см. п. 12.8.13.2. GRANT ROLE;

– о добавлении и изменении паролей см. п. 12.8.3.3. ALTER USER.

Вернуться в оглавление

12.8.1.7. CREATE ROLE

Команда создаёт роль. Подробно о ролях см. п. 12.5.7. Привилегии (PRIV) и роли (ROLE).

Создать роль может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды создания роли:

```
CREATE ROLE <имя_роли>;
```

Где <имя_роли> — имя создаваемой роли.

Пример применения команды CREATE ROLE

Рассмотрим пример создания ролей, разграничивающих доступ сотрудников следующих должностей:

– младший администратор — может подсоединяться к БД, создавать и удалять объекты базы данных;

– менеджер — может подсоединяться к БД, просматривать и делать выборку данных из таблицы "Продажи", добавлять новые данные в таблицу.

Для начала создадим две роли:

```
create role "Младший_администратор";
create role "Менеджер";
```

Теперь в каждую из ролей добавим соответствующие роли или привилегии, необходимые для выполнения указанных ранее действий над данными или объектами БД:

```
grant RESOURCE to "Младший_администратор";
grant SELECT, INSERT on "Продажи" to "Менеджер";
```

Назначив затем роли менеджера и младшего администратора соответствующим пользователям, можно легко разграничить их доступ к данным и объектам БД.

Подробнее:

– о привилегиях см. п. 12.5.7. Привилегии (PRIV) и роди (ROLE);

– о добавлении привилегий см. п. 12.8.13.2. GRANT ROLE;

– о добавлении ролей см. п. 12.8.13.2. GRANT ROLE.

Вернуться в оглавление

12.8.1.8. CREATE PROCEDURE

Команда создаёт хранимую процедуру. Подробно о процедуре см. п. 12.5.8. Хранимая процедура (PROCEDURE).

Создать процедуру может:

– пользователь базы данных с системной привилегией RESOURCES CONTROL в схеме, владельцем которой он является;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды создания процедуры:

```
CREATE [OR REPLACE] PROCEDURE <имя_процедуры> [(<параметр>[,<параметр>…])]
  IS | AS <блок_деклараций> <блок_операторов>;
```

Где:

1) OR REPLACE — при использовании данной конструкции одноимённая процедура будет удалена (если она существует) и создана новая на основе описания, указанного в команде.

2) <имя_процедуры> — имя создаваемой процедуры.

3) <параметр> — аргумент создаваемой хранимой процедуры, представляет собой конструкцию:

```
<параметр> ::= <имя_параметра> [IN] <тип_данных> [:=<выражение> | DEFAULT <выражение>]
        | <имя_параметра> [OUT | IN OUT] <тип_данных>
```

Где:

– <имя_параметра> — имя параметра создаваемой процедуры;

– <тип_данных> — тип данных параметра, поддерживаемый СУБД СОКОЛ (подробнее о типах см. п. 12.1. Типы данных);

– :=<выражение> или DEFAULT <выражение> — используется для указания выражения, в соответствии с которым будет вычисляться значение параметра, если фактический параметр при вызове процедуры не указан. Эти две конструкции синонимичны.

– IN — указывает, что параметр передается в процедуру (используется только как входной параметр);

– OUT — указывает, что параметр возвращается процедурой (используется только как выходной параметр);

– IN OUT — указывает, что параметр передается процедуре и возвращается ею (используется как входной/выходной параметр).

4) <блок_деклараций> — блок деклараций, содержащий описание переменных, типов и т.д. Подробнее о блоке деклараций см. п. 16. ПРОЦЕДУРНЫЙ ЯЗЫК В СУБД СОКОЛ.

5) <блок_операторов> — блок операторов, который необходимо выполнить при вызове процедуры. Подробнее о блоке операторов см. п. 16. ПРОЦЕДУРНЫЙ ЯЗЫК В СУБД СОКОЛ.

Вернуться в оглавление

12.8.2. DROP

Запрос DROP используется для удаления существующей сущности (например, TABLE, ROLE, SCHEMA, USER, INDEX, SEQUENCE, VIEW, PROCEDURE).

Подробнее о правилах именования объектов БД и обращения к ним см. п. 12.5.10. Правила именования объектов БД и обращения к ним.

Вернуться в оглавление

12.8.2.1. DROP TABLE

Команда удаляет таблицу со всеми имеющимися в ней данными.

Удалить таблицу может:

– её владелец с системной привилегией RESOURCES CONTROL;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления таблицы:

```
DROP TABLE [IF EXISTS]CV <имя_таблицы> [<удаление_ограничений>]CV;

<удаление_ограничений> ::= CASCADE CONSTRAINT
                | CASCADE CONSTRAINTS
                | CASCADE
```

Где:

– <имя_таблицы> — имя удаляемой таблицы;

– CASCADE — указание этого ключевого слова позволяет удалить все ссылочные ограничения целостности, которые ссылаются на ключи в удаляемой таблице.

CASCADE CONSTRAINT и CASCADE CONSTRAINTS являются синонимами CASCADE – указание любого из них обеспечивает результат, указанный для CASCADE.

Если не указать одно из указанных синонимов, а ограничения ссылочной целостности существуют, то выполнение команды завершиться возвращением ошибки;

– IF EXISTS — указание позволяет при выполнении команды не возвращать ошибку, если для удаления указана несуществующая таблица в БД.

При удалении таблицы объекты, которые на неё ссылаются (например, представления, процедуры), становятся нерабочими до тех пор, пока не будет создан одноимённый объект. Индексы таблицы и привилегии пользователей на доступ к таблице удаляются.

Если с таблицей в момент удаления работают пользователи, то таблица будет удалена по завершении всех работающих с ней транзакций.

Пример удаления таблицы

Ранее мы создавали для примера таблицу GORVODA. Предположим, что компанией было принято решение вести данные в новом приложении, которое

работает с таблицами другого формата. Таблица GORVODA стала не нужна, и её можно удалить:

```
drop table GORVODA;
```

Попробуем извлечь данные из удалённой таблицы:

```
select * from GORVODA;
```

На этот запрос будет возвращена ошибка, поскольку система не находит указанную таблицу.

Вернуться в оглавление

12.8.2.2. DROP INDEX

Команда удаляет индекс.

Удалить индекс может:

– владелец таблицы с системной привилегией RESOURCES CONTROL, для которой построен индекс;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления индекса:

```
DROP INDEX <имя_индекса>;
```

Где <имя_индекса> — имя удаляемого индекса.

Пример удаления индекса

Ранее мы создавали индекс GORVODA_IDX1 по значениям одного столбца таблицы GORVODA (подробнее см. п. 12.8.1.3. CREATE INDEX). В процессе работы и построения запросов стало понятно, что эффективнее искать информацию по трём столбцам таблицы. Был построен новый индекс.

Таким образом, индекс GORVODA_IDX1 как малоэффективный можно удалить:

```
drop index GORVODA_IDX1;
```

Вернуться в оглавление

12.8.2.3. DROP SEQUENCE

Команда удаляет генератор последовательности.

Удалить генератор последовательности может:

– владелец генератора с системной привилегией RESOURCES CONTROL;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления генератора последовательности:

```
DROP SEQUENCE <имя_последовательности>;
```

Где <имя_последовательности> — имя удаляемого генератора последовательности.

Если с генератором последовательности в момент удаления работают пользователи, то команда будет выполнена по завершении всех работающих с этим генератором транзакций.

Пример применения команды DROP SEQUENCE

Допустим, данные по сотрудникам компании велись в таблице WORKERS:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-------------+-----------------+--------------------+--------------
101       | Зощенко П.М.| 100000        | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
104       | Иванова А.С.| 60000         | ведущий специалист | МТО
105       | Петров Г.С. | 50000         | специалист 1 кат   | МТО
106       | Попова А.П. | 40000         | специалист 2 кат   | МТО
```

Считаем, что при внесении данных нового сотрудника столбец с табельными номерами <Таб №> заполняется значениями по умолчанию — из генератора последовательности SEQ.

Было принято решение о переносе данных по сотрудникам в другую базу данных и ведении их там. Таблица была удалена и необходимости в генераторе последовательности SEQ также нет. Тогда его нужно удалить:

```
drop sequence SEQ;
```

В дальнейшем при попытке упоминания или обращения к удалённому генератору последовательности будет возвращена ошибка.

Продемонстрируем это запросом:

```
select SEQ.nextval;
```

В ответ будет возвращена ошибка, т.к. из базы данных этот генератор последовательности удалён.

Вернуться в оглавление

12.8.2.4. DROP PROCEDURE

Команда удаляет процедуру.

Удалить процедуру может:

– владелец процедуры с системной привилегией RESOURCES CONTROL в схеме, которой он владеет;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления процедуры:

```
DROP PROCEDURE <имя_процедуры>;
```

Где <имя_процедуры> — имя удаляемой процедуры.

Если с процедурой в момент её удаления работают пользователи, то команда будет выполнена по завершении всех работающих с этой процедурой транзакций.

Вернуться в оглавление

12.8.2.5. DROP VIEW

Команда удаляет представление.

Удалить представление может:

– владелец представления с системной привилегией RESOURCES CONTROL;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления представления:

```
DROP VIEW <имя_представления>;
```

Где <имя_представления> — имя удаляемого представления.

Если с представлением в момент его удаления работают пользователи, то команда будет выполнена по завершении всех работающих с этим представлением транзакций.

Пример удаления представления

Ранее для таблицы WORKERS было создано представление WORKERS_1 (см. п. 12.8.1.2 CREATE VIEW), которое позволяло предоставить часть информации по сотрудникам, чтобы ограничить видимость данных основной таблицы.

Предположим, что работа, для которой необходима была часть информации из таблицы, выполнена и данное представление больше не используется.

В таком случае его можно удалить:

```
drop view WORKERS_1;
```

В дальнейшем при попытке извлечь какие-либо данные из удалённого представления будет возвращена ошибка.

Продемонстрируем это запросом:

```
select * from WORKERS_1;
```

В ответ будет возвращена ошибка, т.к. из базы данных это представление удалено.

Вернуться в оглавление

12.8.2.6. DROP USER

Команда удаляет пользователя.

Удалить пользователя может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления пользователя:

```
DROP USER <имя_пользователя>;
```

Где <имя_пользователя> — имя пользователя, которого необходимо удалить.

До удаления пользователя необходимо разорвать его связь с объектами, которыми он владеет, одним из следующих способов:

– назначить схемам, владельцем которых он является, другого пользователя владельцем;

– удалить все объекты из схем и сами схемы, владельцем которых он является.

В противном случае будет возвращена ошибка и удаление пользователя выполнено не будет.

В связи с тем, что в команде DROP USER не реализована возможность каскадного удаления всех объектов, находящихся в схеме пользователя, необходимо их удалить в явном виде.

Получить перечень всех объектов, владельцем которых является пользователь, можно запросом:

```
select
  cast (U.NAME as varchar(64)) as USER_NAME,
  cast (S.NAME as varchar(64)) as SCHEMA_NAME,
  cast (O.NAME as varchar(64)) as OBJECT_NAME,
  cast ((case O.TYPE when 4 then 'TABLE'
                     when 5 then 'VIEW'
                     when 6 then 'PROCEDURE'
                     when 11 then 'SEQUENCE'
                     end) as varchar(20)) as OBJECT_TYPE
from
  SYS._SCHEMA U,
  SYS._SCHEMA S,
  SYS._OBJECT O
where
  O.SCHEMA_ID = S.SCHEMA_ID
  and S.TYPE = 1
  and S.OWNER_ID = U.SCHEMA_ID
  and U.TYPE = 2
  and U.NAME = '<имя_пользователя>';
```

Будет получена информация в формате: имя пользователя, имя схемы, имя объекта и тип объекта.

Если не указать конструкцию:

```
and U.NAME = '<имя_пользователя>'
```

В этом случае будут извлечены объекты всех пользователей БД.

О том, как получить список всех объектов в конкретной схеме, см. п. 12.6.2.8 DROP SCHEMA.

Если пользователь в момент его удаления находится в активной сессии, то команда будет выполнена по завершении этой сессии.

Пример применения команды DROP USER

Рассмотрим пример удаления пользователя до и после выполнения необходимых требований.

Допустим, из компании уволился сотрудник Петров. Ранее в примере п. 12.8.1.6 CREATE USER для него был создан пользователь с паролем и предоставлением системной роли CONNECT. В период работы сотрудник Петров создал в своей схеме таблицу "Статистика".

Руководитель заявкой подтверждает администратору БД необходимость удаления пользователя и изменения владельца таблицы "Статистика". В этом случае администратор БД выполняет необходимые действия.

Если администратор попытается просто удалить пользователя Петрова:

```
drop user "Петров";
```

В этом случае будет возвращена ошибка в связи с тем, что пользователь является владельцем:

– своей схемы по умолчанию;

– созданной таблицы "Статистика".

Для понимания, владельцем каких именно объектов является Петров, выполним запрос:

```
select
  cast (U.NAME as varchar(64)) as USER_NAME,
  cast (S.NAME as varchar(64)) as SCHEMA_NAME,
  cast (O.NAME as varchar(64)) as OBJECT_NAME,
  cast ((case O.TYPE when 4 then 'TABLE'
                when 5 then 'VIEW'
                when 6 then 'PROCEDURE'
                when 11 then 'SEQUENCE'
                end) as varchar(20)) as OBJECT_TYPE
from
  SYS._SCHEMA U,
  SYS._SCHEMA S,
  SYS._OBJECT O
where
  O.SCHEMA_ID = S.SCHEMA_ID
  and S.TYPE = 1
  and S.OWNER_ID = U.SCHEMA_ID
  and U.TYPE = 2
  and U.NAME = 'Петров';
```

Будет возвращена таблица с соответствующими данными:

```
 USER_NAME     | SCHEMA_NAME     | OBJECT_NAME     | OBJECT_TYPE
```

```
---------------+-----------------+-----------------+-----------------
Петров        | Петров         | Статистика      | TABLE
Петров        | Петров         | PK_138_C        | NULL
Петров        | Петров         | __HIDDEN_PK_SEQ_138_C| SEQUENCE
```

Для удаления пользователя необходимо или удалить таблицу "Статистика" и схему "Петров", или назначить схеме "Петров" другого владельца.

Назначим схеме "Петров" другого владельца — пользователя "Иванов":

```
alter schema "Петров" authorization "Иванов";
```

После успешно выполненной команды можно удалить пользователя "Петров":

```
drop user "Петров";
```

Вернуться в оглавление

12.8.2.7. DROP ROLE

Команда удаляет роль.

Удалить роль может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления роли:

```
DROP ROLE <имя_роли>;
```

Где <имя_роли> — имя удаляемой роли.

При выполнении команды, роль будет отозвана у всех пользователей и ролей, которым она была предоставлена. Если в момент удаления роли пользователи, которым она назначена, находятся в активной сессии, то роль у них будет отозвана по завершении сессии.

Пример применения команды DROP ROLE

Ранее были созданы роли с привилегиями соответственно должности: "младший_администратор", "менеджер".

Допустим, структура компании изменилась, должности и обязанности сотрудников также изменились. Для того, чтобы роли в базе данных соответствовали новым должностям по наименованию и привилегиям, было принято решение удалить существующие роли и создать новые.

Для удаления роли используем команду:

```
drop role "менеджер";
```

Теперь выполним попытку предоставить удалённую ранее роль пользователю базы данных Ivanov:

```
grant role "менеджер" to Ivanov;
```

Будет возвращена ошибка.

Вернуться в оглавление

12.8.2.8. DROP SCHEMA

Команда удаляет схему.

Удалить схему может только пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления схемы:

```
DROP SCHEMA <имя_схемы>;
```

Где <имя_схемы> — имя удаляемой схемы.

В связи с тем, что в команде DROP SCHEMA на текущий момент не реализована возможность каскадного удаления всех объектов, находящихся в схеме, необходимо до удаления схемы удалить все объекты из неё. В противном случае будет возвращена ошибка и процедура удаления схемы выполнена не будет.

Получить перечень всех объектов, созданных пользователем в конкретной схеме, можно запросом:

```
select
  cast (S.NAME as varchar(64)) as SCHEMA_NAME,
  cast (O.NAME as varchar(64)) as OBJECT_NAME,
  cast ((case O.TYPE when 4 then 'TABLE'
                when 5 then 'VIEW'
                when 6 then 'PROCEDURE'
                when 11 then 'SEQUENCE'
                end) as varchar(20)) as OBJECT_TYPE
from
  SYS._SCHEMA S,
  SYS._OBJECT O
where
  O.SCHEMA_ID = S.SCHEMA_ID
  and S.TYPE = 1 and S.NAME = '<имя_схемы>';
```

Будет извлечена информация в формате: имя схемы, имя объекта и тип объекта. Если не указать конструкцию:

```
and S.NAME = '<имя_схемы>'
```

В этом случае будут получены объекты всех схем БД.

Пример применения команды DROP SCHEMA

Ранее в примере п. 12.8.1.5. CREATE SCHEMA были созданы две схемы: "Отдел кадров" и "Отдел мотивации". В каждой схеме находились объекты, с которыми работали сотрудники соответствующих подразделений — одноимённые таблицы "Сотрудники компании" с разным содержанием.

В целях оптимизации было принято решение все объекты поместить в одной схеме «Управление персоналом», а ранее созданные схемы удалить. Предварительно схемы нужно очистить от **объектов, созданных** в них **пользователями БД**, иначе будет возвращена ошибка.

Попробуем выполнить удаление схемы без предварительного удаления из неё объектов:

```
drop schema "Отдел кадров";
```

В ответ будет возвращена ошибка.

Для понимания, какие именно объекты находятся в схеме, выполним запрос:

```
select
  cast (S.NAME as varchar(64)) as SCHEMA_NAME,
  cast (O.NAME as varchar(64)) as OBJECT_NAME,
  cast ((case O.TYPE when 4 then 'TABLE'
                     when 5 then 'VIEW'
                     when 6 then 'PROCEDURE'
                     when 11 then 'SEQUENCE'
                     end) as varchar(20)) as OBJECT_TYPE
from
  SYS._SCHEMA S,
  SYS._OBJECT O
where
  O.SCHEMA_ID = S.SCHEMA_ID
  and S.TYPE = 1 and S.NAME = 'Отдел кадров';
```

Будет возвращена таблица с соответствующими данными:

```
SCHEMA_NAME     | OBJECT_NAME             | OBJECT_TYPE
----------------+-------------------------+-----------------
 Отдел кадров   | Сотрудники компании     | TABLE
 Отдел кадров   | PK_xxx_Сотрудники компании| INDEX
```

В данной таблице:

– TABLE <Сотрудники компании> — таблица, созданная пользователем БД;

– INDEX <PK_*xxx*_Сотрудники компании> — индекс, построенный автоматически по установленному при создании таблицы ограничению PRIMARY KEY для столбца "Таб №".

Удалим таблицу (и сопутствующий ей индекс):

```
drop table "Отдел кадров"."Сотрудники компании";
```

После успешно выполненной команды можно удалить схему:

```
drop schema "Отдел кадров";
```

Аналогичным способом можно удалить объекты второй схемы и саму схему.

Вернуться в оглавление

12.8.3. ALTER

Запрос ALTER используется для изменения существующей сущности (например, SCHEMA, USER).

Подробнее о правилах именования объектов БД и обращения к ним см. п. 12.5.10 Правила именования объектов БД и обращения к ним.

12.8.3.1. ALTER SCHEMA

Команда изменения владельца схемы.

Изменить владельца схемы может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды изменения владельца схемы:

```
ALTER SCHEMA <имя_схемы> AUTHORIZATION <имя_пользователя>;
```

Где:

– <имя_схемы> — имя схемы, для которой необходимо изменить владельца;

– <имя_пользователя> — имя нового владельца схемы.

Пример изменения владельца схемы

Допустим, сотрудник Иванов уволился из компании, а в его схеме есть объекты, которые необходимы для работы другим сотрудникам компании (например, таблица «Контакты партнёров»).

По распоряжению руководства уволившегося сотрудника администратор БД перед удалением пользователя Иванова назначает владельцем его схемы пользователя Котова:

```
alter schema "Иванов" authorization "Котов";
```

После выполнения данной команды владельцем схемы стал другой сотрудник компании — Котов, который может работать с объектами в данной схеме. Например, делать выборку данных из таблицы «Контакты партнёров»:

```
select * from "Иванов"."Контакты партнёров";
```

У пользователя Иванова теперь нет схем и объектов, владельцем которых он является и его можно удалить:

```
drop user "Иванов";
```

Вернуться в оглавление

12.8.3.2. ALTER USER SCHEMA

Команда изменения у пользователя схемы по умолчанию.

Изменить схему по умолчанию может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды изменения у пользователя схемы по умолчанию:

```
ALTER USER <имя_пользователя> DEFAULT SCHEMA <имя_схемы>;
```

Где:

– <имя_пользователя> — имя пользователя, для которого необходимо изменить схему по умолчанию;

– <имя_схемы> — имя схемы, которая должна стать схемой по умолчанию.

Пример применения команды ALTER USER SCHEMA

Допустим, для отдела кадров была создана отдельная схема HR, в которую помещены некоторые объекты с данными, необходимыми для работы сотрудника отдела кадров, в т.ч. таблица WORKERS с данными сотрудников компании. Владельцем схемы HR является руководитель подразделения Иванов.

Схема HR не является схемой по умолчанию для сотрудников отдела кадров, поэтому при любом обращении к таблице им необходимо указывать её полное имя:

```
select * from HR.WORKERS;
```

В связи с тем, что один сотрудник отдела кадров (Котов) постоянно работает только с объектами в этой схеме, то было принято решение — сделать данную схему для него схемой по умолчанию.

После согласования с руководителем подразделения администратор БД сначала изменяет владельца схемы:

```
alter schema "HR" authorization "Котов";
```

После успешно выполненной команды, администратор изменяет схему по умолчанию для сотрудника отдела кадров Котова:

```
alter user "Котов" default schema "HR";
```

Теперь пользователи при обращении к объектам этой схемы могут указывать их краткое имя, например:

```
select * from WORKERS;
```

Вернуться в оглавление

12.8.3.3. ALTER USER

Команда изменения пароля пользователя.

Изменить пароль может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды изменения пароля пользователя:

```
ALTER USER <имя_пользователя> {IDENTIFIED BY '<пароль>' | NO AUTHENTICATION};
```

Где:

– <имя_пользователя> — имя пользователя, для которого необходимо указать новый пароль;

– IDENTIFIED BY <пароль> — конструкция применяется для установки нового пароля пользователя;

– NO AUTHENTICATION — конструкция применяется для удаления пароля пользователя, что приводит к запрещению его аутентификации.

Пример применения команды ALTER USER

Рассмотрим примеры применения команды изменения пароля пользователя.

Допустим, для повышения безопасности учетной записи в компании рекомендовано периодически менять пароль. Сотрудник проработал в компании уже некоторое время и пароль пора менять.

Администратор БД вводит команду с конструкцией для указания нового пароля:

```
alter user "Иванов" IDENTIFIED BY 'MRfirsT';
```

Допустим, сотрудник уволился, но временно данные его объектов ещё нужны (например, данные таблиц, созданных в его схеме). В таком случае, чтобы пользователь не мог пройти аутентификацию, администратор БД может удалить пароль у пользователя:

```
alter user "Иванов" NO AUTHENTICATION;
```

Для проверки блокировки аутентификации попытаемся подключиться под пользователем к БД любым удобным способом. Будет возвращена ошибка — пользователь без пароля не может пройти аутентификацию в СУБД СОКОЛ.

Вернуться в оглавление

12.8.3.4. ALTER TABLE[CV]

Команда изменения структуры таблицы базы данных (любой таблицы БД, кроме временной приватной таблицы и системной таблицы).

Вносить изменения в таблицу базы данных может:

– владелец таблицы с системной привилегией RESOURCES CONTROL;

– пользователь БД с системной привилегией DATABASE ADMIN для любой существующей таблицы БД.

Синтаксис команды изменения таблицы:

```
ALTER TABLE <имя_таблицы> <изменения>

<изменения> ::= RENAME TO <новое_имя_таблицы>
        | RENAME COLUMN <имя_столбца> TO <новое_имя_столбца>
        | ADD [CONSTRAINT <имя_ограничения>] <ограничение_таблицы>
        | ADD COLUMN <столбец>
        | ADD (<столбец>[, <столбец>,...])
        | MODIFY (<имя_столбца> <характеристики>[, <имя_столбца> <характеристики>...]);
        | DROP CONSTRAINT <имя_ограничения>
        | DROP COLUMN <имя_столбца> [<удаление_ограничений>]
        | DROP (<имя_столбца>[, <имя_столбца>,...]) [<удаление_ограничений>]

<ограничение_таблицы> ::= UNIQUE (<имя_столбца>[, <имя_столбца>,...])
                | CHECK (<условие>)
                | <конструкция_FOREIGN KEY>

<столбец> ::= <имя_столбца> <тип_данных> [<свойства_столбца>] [<ограничение_столбца>]

<характеристики>::= <тип_данных> [<свойства_столбца>] [<ограничение_столбца>]

<удаление_ограничений> ::= CASCADE CONSTRAINT
                | CASCADE CONSTRAINTS
                | CASCADE
```

Где:

1) <имя_таблицы> — имя таблицы, в которую необходимо внести изменения.

2) RENAME TO <новое_имя_таблицы> — конструкция для указания нового краткого имени таблицы (без указания схемы), которое должно соответствовать правилам именования объектов БД (подробнее см. п. 12.5.10. Правила именования объектов БД и обращения к ним).

При задании нового имени таблицы запрещено для постоянной / временной глобальной таблицы задавать имя с # в качестве первого символа (что обязательно для временной приватной таблицы). Иначе будет возвращена ошибка. Попытка переименования временной приватной таблицы завершится ошибкой.

После выполнения команды переименования таблицы:

– при упоминании прежнего имени таблицы в командах / процедурах будет возвращена ошибка;

– ограничения целостности, индексы и разрешения с таблицы с прежним именем автоматически будут перенесены на таблицу с новым именем;

– ссылки автоматически будут перенесены на таблицу с новым именем в случае, если переименованная таблица является дочерней или родительской для созданного внешнего ключа;

– зависимые объекты (представления, процедуры), ссылающиеся на таблицу, станут недействительными, но останутся в БД.

3) RENAME COLUMN <имя_столбца> TO <новое_имя_столбца> — конструкция для указания нового имени столбца таблицы, которое должно соответствовать правилам именования объектов БД.

4) ADD [<имя_ограничения>] <ограничение_таблицы> — конструкция для добавления ограничения таблицы. Подробнее про указанные ограничения, накладываемые на таблицу, см. в п. 12.8.1.1. CREATE TABLE.

Попытка установить ограничение по столбцам, значения строк которого не соответствуют условию устанавливаемого ограничения, будет завершена с возвратом ошибки.

Важно: при создании нескольких проверочных ограничений для одного столбца будьте внимательны, т.к. система не проверяет, что условия установленных ограничений не являются взаимоисключающими.

5) ADD COLUMN <столбец> и ADD (<столбец>[, <имя_столбец>,...]) — конструкции для добавления нового столбца (столбцов) в таблицу. Указание параметров нового столбца аналогично их указанию при создании таблицы (подробнее см. п. 12.8.1.1. CREATE TABLE). Для добавляемых столбцов нельзя задавать следующие ограничения:

– primary key;

– [unique] clustered key.

Попытка указать данные ограничения для новых столбцов будет завершена с возвратом ошибки.

6) MODIFY (<имя_столбца> <характеристики>[, ...]) — конструкция для изменения характеристик указанного столбца (столбцов) таблицы (как обычных, так и для вычисляемых) в соответствии с заданными для каждого параметрами:

– тип данных;

– ограничения;

– свойства.

Подробнее о характеристиках столбца см. п. 12.8.1.1. CREATE TABLE.

Ограничения NULL и NOT NULL подменяются. Например, если по столбцу было NOT NULL, а устанавливается NULL, то произойдёт замена ограничения.

Если при изменении характеристик новый размер типа данных будет указан меньше текущего размера данных в столбце или существующие значения строк в указанном столбце не соответствуют новым задаваемым для него характеристикам, то выполнение команды будет остановлено с возвращением ошибки.

Нельзя выполнять:

– изменение столбцов с типами данных BLOB, CLOB;

– изменение типа столбца, если он указан в выражении вычисляемого столбца;

– изменение типа столбца если существует ограничение CHECK, FOREIGN KEY в составе с этим столбцом;

– для временной таблицы указание ограничения ссылочной целостности (FOREIGN KEY) для изменяемого столбца.

В этих случаях выполнение команды будет остановлено с возвращением ошибки.

Если по изменяемому столбцу ранее был задан DEFAULT, то изменение его типа возможно будет только с новым указанием DEFAULT. Иначе будет возвращена ошибка. При изменении характеристик столбца (столбцов) их порядок не изменяется.

7) DROP CONSTRAINT <имя_ограничения> — конструкция для удаления установленного ранее ограничения (подробнее об ограничениях см. п. 12.8.1.1. CREATE TABLE).

8) DROP COLUMN <имя_столбца> — конструкция для указания одного столбца таблицы, который необходимо удалить.

9) DROP (<имя_столбца>[, <имя_столбца>,...]) — конструкция для указания столбца (или столбцов) таблицы, которые необходимо удалить.

10) CASCADE — указывается при удалении столбца (столбцов) из таблицы. Указание этого ключевого слова позволяет удалить все ссылочные ограничения целостности, которые ссылаются на уникальные ключи с удаляемыми столбцами в составе.

CASCADE CONSTRAINT и CASCADE CONSTRAINTS являются синонимами CASCADE — указание любого из них обеспечивает результат, указанный для CASCADE.

Если до начала момента выполнения команды ALTER TABLE другая транзакция не завершила начатые изменения в этой же таблице, то команда будет выполнена после окончания этой транзакции, без её отката или разрыва соединения с БД.

Команда не может быть применена к предустановленным объектам БД.

Вернуться в оглавление

12.8.3.5. ALTER INDEX[CV]

Команда переименования существующего индекса (кроме индексов, автоматически создаваемых самой СУБД, информация о которых находится в системных таблицах).

Переименовывать индекс может:

– владелец индекса с системной привилегией RESOURCES CONTROL;

– пользователь БД с системной привилегией DATABASE ADMIN для любого существующего индекса БД.

Синтаксис команды изменения таблицы:

```
ALTER INDEX <текущее_имя> RENAME TO <новое_имя>
```

Где:

– <текущее_имя> — текущее имя индекса;

– <новое_имя> — новое имя индекса, которое должно соответствовать правилам именования объектов БД (подробнее см. п. 12.5.10. Правила именования объектов БД и обращения к ним).

После выполнения команды при упоминании прежнего имени индекса в командах будет возвращена ошибка.

При изменении имени индекса он сохраняет все свои атрибуты, указанные при его создании (подробнее см. п. 12.8.1.3. CREATE INDEX).

Вернуться в оглавление

12.8.4. TRUNCATE TABLE

Команда быстрого удаления всех записей таблицы с сохранением её структуры (столбцы, ограничения, индексы и т.д.).

Выполнить быстрое удаление всех записей таблицы с сохранением её структуры командой TRUNCATE TABLE может:

– владелец таблицы с системной привилегией RESOURCES CONTROL;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления всех записей таблицы:

```
TRUNCATE TABLE <имя_таблицы>;
```

Где <имя_таблицы> — имя таблицы, из которой необходимо удалить все записи.

Команда TRUNCATE TABLE имеет действие, аналогичное команде DELETE без условия WHERE. Однако, между ними есть отличия:

– при выполнении команды TRUNCATE блокируется вся таблица, а при DELETE — каждая удаляемая строка;

– после выполнения команды TRUNCATE TABLE, откат содержимого таблицы командой rollback в прежнее состояние невозможен;

– команда DELETE возвращает число удаленных строк, а команда TRUNCATE всегда возвращает значение 0 независимо от числа удалённых строк.

Если с таблицей в момент выполнения команды TRUNCATE TABLE работают другие пользователи, то команда будет выполнена по завершении всех работающих с ней транзакций.

Пример применения команды TRUNCATE TABLE

Допустим, в нашу базу данных была скопирована таблица из другой базы данных, но заполнить её нужно полностью новыми данными. Для этого очищаем скопированную таблицу от всех данных:

```
truncate table PHONE;
```

Проверить успешность выполненной команды можно командой выборки всех данных из таблицы:

```
select * from PHONE;
```

В ответ будет извлечена пустая таблица.

Вернуться в оглавление

12.8.5. SELECT

Команда выборки данных. Подробнее о таблицах см. п. 12.5.1. Таблица (TABLE), о представлениях см. п. 12.5.2. Представление (VIEW).

Выполнить выборку данных командой SELECT может:

– пользователь базы данных с объектной привилегией SELECT;

– владелец таблицы, представления;

– пользователь базы данных с привилегией DATABASE ADMIN.

Вернуться в оглавление

12.8.5.1. Общий синтаксис команды SELECT

```
SELECT [DISTINCT | ALL] <выборка>
   [INTO <переменная> [, <переменная>...]]
   [<конструкция_FROM>]
   [WHERE <выражение>]
   [GROUP BY [<выражение>, <выражение>...]]
   [HAVING <выражение>]
   [UNION [ALL] <SELECT-элемент>]
   [<конструкция_ORDER_BY>]
   [<ограничение_числа_строк_выборки>]
   [<блокировка_строк>]

<выборка> :: = * | <выражение> [[AS] <локальное_имя>][, <выражение> [[AS] <локальное_имя>]...]
<переменная> :: = <имя_переменной> | :<имя_параметра> | ?
```

Где:

1) <SELECT-элемент> — команда указанного выше синтаксиса, но без элементов <ограничение_числа_строк_выборки>, <конструкция_ORDER_BY>, <блокировка_строк>, INTO <переменная> [, <переменная>...].

2) DISTINCT — указывается для возврата строк, удовлетворяющих условию выборки, исключая строки-дубликаты.

Важно: DISTINCT в текущей версии СУБД не работает со столбцами, тип данных которых BLOB / CLOB или размер которых превышает 4000 байт.

3) ALL — указывается для возврата всех строк, удовлетворяющих условию выборки, включая все строки-дубликаты. ALL является условием по умолчанию.

4) <выражение> в выборке указывает выражение для вычисления значения, которое необходимо извлечь в качестве результата.

5) * — используется при необходимости выбора всего итогового множества строк всех столбцов таблицы.

6) <выражение> AS <локальное_имя> — используется для указания локального имени (псевдонима) выражения, служащего наименованием столбца результирующей таблицы. Указанный псевдоним может использоваться в <конструкции_ORDER_BY> и для ссылки на столбцы запроса/подзапроса.

7) INTO <переменная> [, <переменная>...] — используется (в зависимости от контекста выполнения команды) для указания выходных параметров (переменных), в которых будут сохраняться значения, возвращаемые запросом. Выходной параметр может выглядеть как:

– именованный параметр «:<имя_переменной>» при отправке запроса из процедурного языка или через vsql_console;

– неименованный параметр «?» при отправке запроса через внешние интерфейсы (например, ODBC, JDBC и другие).

В конструкции INTO необходимо указать соответствующую совместимую по типу переменную для каждого значения <выражения> в выборке.

8) <конструкция_FROM> — конструкция позволяет указать один или несколько источников данных, из которых необходимо выбрать данные (подробнее см. далее п.12.8.5.2. <конструкция_FROM>).

9) WHERE <выражение> — конструкция позволяет выбрать только те строки, для которых значение указанного выражения равно TRUE.

10) GROUP BY [<выражение>, <выражение>...] — позволяет в результирующем наборе группировать строки, имеющие одинаковое значение по одному или нескольким столбцам, соответствующим указанному значению заданного выражения. После группировки будет возвращена одна строка для каждой группы.

Важно: конструкция в текущей версии СУБД не работает со столбцами, тип данных которых BLOB / CLOB или размер которых превышает 4000 байт.

11) HAVING <выражение> — конструкция в сочетании с GROUP BY позволяет возвращать только те группы строк, для которых значение указанного выражения равно TRUE. Конструкция HAVING отличается от конструкции WHERE тем, что WHERE выполняется до формирования групп строк, а HAVING выполняется после формирования групп и содержит условия, применимые к группам строк. Учитывая, что конструкция HAVING <выражение> применяется к группам строк, элементы, указанные в <выражение>, должны ссылаться на значения, которые постоянны

внутри каждой группы, или ссылаться на результаты агрегатных функций, вычисляемые на группах строк.

12) UNION — позволяет объединять результаты SELECT-запросов в один, который из дублирующихся значений возвращает только одно.

13) UNION ALL — позволяет объединять результаты SELECT-запросов в один, который возвращает все значения. Количество столбцов и типы данных столбцов, выбранных каждым запросом, участвующим в UNION / UNION ALL, должны быть одинаковыми. Имена столбцов в результирующем наборе — это имена выражений в списке выборки первого запроса.

14) <конструкция_ORDER_BY> — конструкция используется для упорядочения результирующего набора в соответствии с заданным правилом (подробнее см. далее п.12.8.5.3. <конструкция_ORDER_BY>).

15) <ограничение_числа_строк_выборки> — конструкция используется для указания ограничений числа строк в результирующем наборе (подробнее см. п. 12.8.5.4. <ограничение_числа_строк_выборки>).

16) <блокировка_строк> — конструкция позволяет заблокировать все строки таблицы. Это запрещает вносить изменения и блокировать строки другими пользователями до завершения транзакции. Также конструкция <блокировка_строк> позволяет выбрать принцип поведения относительно строк, уже заблокированных параллельной транзакцией (подробнее см. п. 12.8.5.5. <блокировка_строк>).

Вернуться в оглавление

12.8.5.2. <конструкция_FROM>

Конструкция используется для указания источников данных (таблиц, подзапросов, соединений, представлений), из которых необходимо сделать выборку данных.

Синтаксис <конструкции_FROM>:

```
FROM <источник_данных> [,<источник_данных>]


<источник_данных> ::= <имя_таблицы> [[AS] <новое_имя>]
                | (<SELECT_запрос>) [[AS] <новое_имя>]
                | <соединение_источников>

<соединение_источников> ::= <элемент> [INNER] JOIN <элемент> <условие_соединения>
                    | <элемент> NATURAL [INNER] JOIN <элемент>
                    | <элемент> CROSS JOIN <элемент>
                    | <элемент> LEFT [OUTER] JOIN <элемент> [<условие_соединения>]
                    | <элемент> RIGHT [OUTER] JOIN <элемент> [<условие_соединения>]

<условие_соединения> ::= ON <выражение> | USING (<столбец>[, <столбец>...])
```

Где:

1) <элемент> — таблица, подзапрос, соединение источников, из которых необходимо сделать выборку данных.

2) <соединение_источников> — используется для извлечения данных из двух источников. При соединении источников их значения по столбцам, соответствующим условию соединения, сравниваются между собой построчно в соответствии с заданным типом соединения и (или) условием. В результирующий набор попадают строки, содержащие «соединённые» и соответствующие указанным условиям значения.

3) [INNER] JOIN — указывает на выполнение внутреннего соединения. Внутреннее соединение возвращает все строки, соединяемые значения которых удовлетворяют указанному <условию_соединения>. Ключевое слово INNER не обязательное для указания в конструкциях внутреннего соединения и используется по умолчанию.

4) <условие_соединения> — используется для указания условия внутреннего или внешнего соединения данных из двух таблиц, которое определяет, какие строки из них считаются «соответствующими» друг другу:

– USING (<столбец[, столбец...]>) — используется для указания столбцов, одноимённых для обеих таблицах. Так указанное поле одной таблицы через свои значения ссылается на значения указанного одноимённого поля другой таблицы. При этом результирующий набор содержит один одноимённый столбец из двух таблиц.

Если в одной из таблиц не будет столбца с указанным наименованием, то будет возвращена ошибка;

– ON <выражение> — используется для указания условия соединения в виде логического выражения, в соответствии с которым будут сопоставляться строки соединяемых таблиц. Т.е. соединяются строки из одной таблицы со строками из второй таблицы, значения которых соответствуют указанному выражению. При этом результирующий набор содержит все столбцы соединяемых таблиц.

5) NATURAL [INNER] JOIN — указывается для выполнения естественного внутреннего соединения, которое основано на всех столбцах в двух таблицах. При этом виде соединения выбираются строки из двух таблиц, имеющих равные значения в соответствующих столбцах.

6) CROSS JOIN — указывается для выполнения перекрёстного соединения, при котором каждая строка одной таблицы соединяется с каждой строкой второй таблицы, давая тем самым в результате все возможные сочетания строк двух таблиц.

7) OUTER — ключевое слово, указывающее на внешнее соединение. Внешнее соединение возвращает все строки одного из источников (LEFT, RIGHT), соединённые с соответствующими значениями из второго источника, если они удовлетворяют условию соединения. В противном случае, возвращает вместо не найденных соответствий NULL в каждом столбце. Главное отличие внешнего соединения от внутреннего в том, что оно обязательно возвращает все строки одного из источников (LEFT, RIGHT). Ключевое слово OUTER необязательное для указания в конструкции внешнего соединения и используется по умолчанию.

8) LEFT [OUTER] JOIN — указывается для выполнения левого внешнего соединения, при котором сначала выполняется внутреннее соединение (INNER JOIN), после чего в результирующую таблицу добавляются все строки из таблицы, указанной слева, не имеющие совпадений со значениями строк таблицы, указанной справа, а вместо значений правой таблицы вставляются значения NULL.

9) RIGHT [OUTER] JOIN — указывает на правое внешнее соединение, при котором сначала выполняется внутреннее соединение (INNER JOIN), после чего в

результирующую таблицу добавляются все строки из таблицы, указанной справа, не имеющие совпадений со значениями строк таблицы, указанной слева, а вместо значений левой таблицы вставляются значения NULL.

10) [AS] <новое_имя> — конструкция используется для указания псевдонима подзапроса таблицы в рамках выполняемого запроса. Указанный псевдоним может использоваться для ссылки на результаты.

Вернуться в оглавление

12.8.5.3. <конструкция_ORDER_BY>

Конструкция используется для упорядочения записей в результирующем наборе в соответствии с заданным правилом.

Синтаксис <конструкции_ORDER_BY>:

```
ORDER BY <объект_упорядочения> [ASC | DESC] [, <объект_упорядочения> [ASC | DESC] ... ]
```

Где:

– <объект_упорядочения> — задает номер столбца в соответствии с порядком столбцов после ключевого слова SELECT или выражение, по значению которого необходимо упорядочить результирующий набор. Столбцы имеют номера, начиная от единицы;

– ASC — используется для упорядочения записей в результирующем наборе по возрастанию значений объекта упорядочения. Сортировка ASC используется по умолчанию;

– DESC — используется для упорядочения записей в результирующем наборе по убыванию значений объекта упорядочения.

Правило упорядочивания (ASC или DESC) применяется к тому выражению, после которого оно указано.

Важно: конструкция в текущей версии СУБД не работает со столбцами, тип данных которых BLOB / CLOB или размер которых превышает 4000 байт.

Вернуться в оглавление

12.8.5.4. <ограничение_числа_строк_выборки>

Используется для указания условий, ограничивающих число строк в результирующем наборе.

Синтаксис конструкции <ограничение_числа_строк_выборки>:

```
<ограничение_числа_строк_выборки> ::= FETCH FIRST <выражение> ROW ONLY
                                    | LIMIT <выражение> [OFFSET <выражение>]
```

Где:

– FETCH FIRST <выражение> ROW ONLY — позволяет получить заданное число первых строк из результирующего набора. Число строк определяется значением выражения. В конструкции вместо слова ROW допускается употребление слова ROWS. Они синонимичны и на результат это никак не влияет;

– LIMIT <выражение> — позволяет получить заданное число первых строк из результирующего набора. Число строк определяется значением выражения;

– OFFSET <выражение> — позволяет указать, какое число строк таблицы, соответствующих запросу, необходимо пропустить прежде чем начать возвращать строки результата. Число пропускаемых строк определяется значением выражения.

Конструкции FETCH FIRST и LIMIT (за исключением фразы OFFSET) равнозначны.

Вернуться в оглавление

12.8.5.5. <блокировка_строк>

Конструкция позволяет заблокировать строки таблицы. Это запрещает вносить изменения и блокировать строки другим пользователям до завершения транзакции. Также конструкция позволяет выбрать принцип поведения относительно строк, уже заблокированных параллельной транзакцией.

Конструкции этого типа нельзя использовать в подзапросах.

Синтаксис конструкции <блокировка_строк>:

```
FOR UPDATE [ NOWAIT | SKIP LOCKED ]
```

Где:

– FOR UPDATE — указывает на необходимость заблокировать строки таблицы с возможностью указания принципа поведения относительно тех строк, которые уже заблокированы параллельной транзакцией:

1) NOWAIT — указывает, что если строки заблокированы параллельной транзакцией, то будет возвращён результат, собранный до первой найденной заблокированной строки, без учёта этой строки и сообщение об ошибке.

2) SKIP LOCKED — указывает, что если строки заблокированы параллельной транзакцией, то при выборке их нужно пропустить.

Если принцип поведения относительно строк, уже заблокированных параллельной транзакцией, не указан и строка (строки) таблицы уже заблокирована параллельной транзакцией, то результаты запроса не будут возвращены до тех пор, пока блокировка таблицы не будет снята.

Вернуться в оглавление

12.8.5.6. Примеры применения команды SELECT

Рассмотрим примеры применения команды SELECT с использованием её разных конструкций.

Начнём с самого простого: запроса всех данных из таблицы без дополнительных условий.

Допустим, у нас есть таблица PHONE с данными абонентов из всех городов, где компания оказывает услуги. Рассмотрим для начала самый простой запрос — выборку значений всех строк и столбцов таблицы, где для указания источника данных используется <конструкция_FROM>:

```
select * from PHONE;
```

В этом случае будут получены значения всех столбцов всех строк таблицы:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543212 | Петров  | 53   | Воронеж
89876543214 | Иванов  | 46   | Москва
89876543244 | Котов   | 37   | Москва
89876543215 | Попова  | 26   | Воронеж
89876543266 | Лебедева| 37   | Москва
89876543277 | Ермаков | 38   | Химки
89876543288 | Тихонова| 29   | Москва
89876543299 | Ермакова| 45   | Химки
89876543300 | Лебедев | 37   | Москва
```

Далее рассмотрим примеры запросов SELECT с использованием различных его конструкций.

12.8.5.6.1. Пример использования ключевого слова DISTINCT в запросе SELECT

В таблицу PHONE внесены данные абонентов по всем населённым пунктам, где работает компания. Для получения списка всех населённых пунктов без повторов, в которых зарегистрированы абоненты, нужно в запросе SELECT указать ключевое слово DISTINCT:

```
select distinct CITY from PHONE;
```

После выполнения запроса будет получен список:

```
CITY
-----------
Воронеж
Москва
Химки
```

Вернуться в оглавление

12.8.5.6.2. Пример использования конструкции WHERE <выражение>

Допустим, из таблицы PHONE необходимо выбрать всех абонентов из города Химки. Для этого указываем <конструкцию_FROM> и конкретизируем условия в конструкции WHERE <выражение>:

```
select * from PHONE where CITY = 'Химки';
```

После выполнения команды будут получены данные по всем абонентам из города Химки:

```
NUMBER     | NAME     | AGE     | CITY
-------------+-----------+----------+-----------
89876543277 | Ермаков   | 38      | Химки
89876543299 | Ермакова  | 45      | Химки
```

Вернуться в оглавление

12.8.5.6.3. Пример использования конструкции <выражение> AS <локальное_имя> в запросе SELECT

Допустим, для составления отчета необходимо из таблицы PHONE выбрать номера телефонов абонентов из Москвы, но так, чтобы в наименовании столбца присутствовало название города. Для этого в запросе SELECT указываем новое имя столбца:

```
select NUMBER as NUMBER_MOSCOW from PHONE where CITY = 'Москва';
```

После выполнения команды будут получены номера телефонов абонентов из города Москва с новым наименованием столбца:

```
NUMBER_MOSCOW
--------------------
89876543214
89876543244
89876543266
89876543288
89876543300
```

Вернуться в оглавление

12.8.5.6.4. Пример использования ключевых слов UNION и UNION ALL для объединения результатов SELECT-запросов в один

Допустим, в компании данные сотрудников расположены в разных таблицах в соответствии с тем, в каком городе находится филиал — Москва и Воронеж.

Необходимо выполнить сбор данных по всем сотрудникам — из двух таблиц.

Для примера возьмём две таблицы, WORKERS_VORONEZH и WORKERS_MOSCOW:

```
select * from WORKERS_VORONEZH;

 Таб №  | Имя   | Фамилия   | Возраст  | Пол   | Подразделение
--------+-------+-----------+----------+-------+----------------
 212    | Иван  | Петров    | 53       | м     | МТО
 213    | Пётр  | Иванов    | 27       | м     | МТО
 214    | Игорь | Котов     | 38       | м     | Отдел кадров
 215    | Алёна | Котова    | 70       | ж     | ПТО
 216    | Юлия  | Серова    | 65       | ж     | ФЭУ
 217    | Елена | Попова    | 19       | ж     | ФЭУ

select * from WORKERS_MOSCOW;

 Таб №  | Имя   | Фамилия   | Возраст  | Пол   | Подразделение
--------+-------+-----------+----------+-------+----------------
 218    | Ирина | Зотова    | 48       | ж     | Отдел кадров
 219    | Фёдор | Уваров    | 67       | м     | МТО
 220    | Алёна | Котова    | 67       | ж     | МТО
 221    | Андрей| Быков     | 68       | м     | ПТО
 222    | Фёдор | Быков     | 53       | м     | ФЭУ
 223    | Юлия  | Гузеева   | 56       | ж     | ФЭУ
 224    | Иван  | Петров    | 70       | м     | ПТО
```

Допустим, для статистического отчёта необходимо сделать выборку данных по сотрудникам из двух филиалов — какие сотрудники пожилого возраста старше 65 лет работают в компании.

Выполним объединение соответствующих запросов с помощью ключевых слов UNION ALL:

```
select "Таб №", "Пол", "Возраст" from
   (select "Таб №", "Пол", "Возраст" from WORKERS_VORONEZH
      union all
         select "Таб №", "Пол", "Возраст" from WORKERS_MOSCOW) where "Возраст" > 65;
```

В этом случае будет возвращён объединённый результирующий набор всех значений, соответствующих заданным условиям в запросах:

```
Таб №   | Пол       | Возраст
---------+-------------+----------
215     | ж         | 70
219     | м         | 67
220     | ж         | 67
221     | м         | 68
224     | м         | 70
```

Допустим, для статистического отчёта необходима информация о том, в каких подразделениях работают сотрудники старше 65 лет.

Для получения такой информации выполним объединение соответствующих запросов с помощью ключевого слова UNION:

```
select "Возраст", "Подразделение" from
    (select "Возраст", "Подразделение" from WORKERS_VORONEZH
       union
     select "Возраст", "Подразделение" from WORKERS_MOSCOW)
        where "Возраст" > 65;
```

В этом случае будет возвращён объединённый результирующий набор без дубликатов:

```
Возраст  | Подразделение
---------+---------------------
67      | МТО
68      | ПТО
70      | ПТО
```

Для сравнения: если для выборки подразделений, в которых работают сотрудники старше 65 лет, использовать объединение запросов с помощью ключевых слов UNION ALL:

```
select "Возраст", "Подразделение" from
    (select "Возраст", "Подразделение" from WORKERS_VORONEZH
       union all
     select "Возраст", "Подразделение" from WORKERS_MOSCOW)
        where "Возраст" > 65;
```

В этом случае будет возвращён набор со всеми строками, значения которых удовлетворяют заданным условиям, включая дубликаты:

```
Возраст  | Подразделение
---------+---------------------
70      | ПТО
67      | МТО
```

```
67    | МТО
68    | ПТО
70    | ПТО
```

Допустим, для статистического отчёта необходимо сделать выборку данных по сотрудникам из двух филиалов — сколько пожилых сотрудников (старше 65 лет) и какого возраста работают в компании.

Для этого используем функцию COUNT, объединение запросов с помощью ключевого слова UNION ALL и группировку значений по возрасту:

```
select "Возраст", count (*) as count from
    (select ALL "Возраст" from WORKERS_VORONEZH
        union ALL
     select ALL "Возраст" from WORKERS_MOSCOW)
where "Возраст" > 65 group by "Возраст";
```

В этом случае будет возвращён объединённый результирующий набор с подсчётом дублирующих значений по заданному столбцу:

```
AGE     | COUNT
----------+------------
67      | 2
68      | 1
70      | 2
```

Вернуться в оглавление

12.8.5.6.5. Пример использования конструкции GROUP BY [<выражение>, <выражение>...] HAVING <выражение>

Для примера возьмём ту же таблицу PHONE с данными абонентов из всех городов:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543212 | Петров  | 53   | Воронеж
89876543214 | Иванов  | 46   | Москва
89876543244 | Котов   | 37   | Москва
89876543215 | Попова  | 26   | Воронеж
89876543266 | Лебедева| 37   | Москва
89876543277 | Ермаков | 38   | Химки
89876543288 | Тихонова| 29   | Москва
89876543299 | Ермакова| 45   | Химки
89876543300 | Лебедев | 37   | Москва
```

Допустим, отдел аналитики проводит исследование на тему распределения абонентов по городам.

Для понимания, в каких городах сколько находится абонентов компании, можно выполнить запрос с использование конструкции GROUP BY [<выражение>, <выражение>...] и функции COUNT:

```
select CITY, count (*) as COUNT from PHONE group by CITY;
```

После выполнения запроса будет получена соответствующая информация:

```
CITY     | COUNT
----------+---------
 Воронеж  | 2
 Москва   | 5
 Химки    | 2
```

Допустим, нужно из таблицы PHONE извлечь названия городов, число абонентов в которых больше (меньше или равно) определённого количества, например, больше или равно 5.

В этом случае к конструкции GROUP BY [<выражение>, <выражение>...] добавим конструкцию HAVING <выражение>:

```
select CITY, count(*) as COUNT from PHONE group by CITY having count(*) >= 5;
```

После выполнения запроса будет получена соответствующая информация:

```
CITY     | COUNT
----------+------------
 Москва   | 5
```

Конструкция HAVING <выражение> позволила выбрать только те группы, которые соответствовали указанному условию, а именно, где число строк в группе было больше или равно 5.

Вернуться в оглавление

12.8.5.6.6. Пример использования <конструкции_FROM> с использованием разных характеристик соединения данных разных источников

Для демонстрации разных типов соединений возьмём простые таблицы TABLE1, TABLE2, в которых одноимённый столбец будет связующим.

Для понимания структуры и содержания обеих таблиц выполним к каждой из них запрос всех данных:

```
select * from TABLE1;

TABLE1:
NUMBER      | NAME1
------------+-----------
1           | имя1
2           | имя2
5           | имя5
```

```
select * from TABLE2;

TABLE2:
NUMBER      | NAME2
------------+-----------
1           | имя1
2           | имя2
3           | имя3
4           | имя4
```

Рассмотрим запросы SELECT с указанием внутреннего типа соединения (INNER JOIN) и разными условиями соединения значений двух таблиц.

Выполним запрос SELECT с указанием фразы INNER JOIN и связующего столбца, одноимённого для двух таблиц (NUMBER):

```
select * from TABLE1 inner join TABLE2 using (NUMBER);
```

При этом виде соединения возвращаются значения всех строк обеих таблиц, которые имеют равные значения в указанном столбце NUMBER каждой таблицы:

```
NUMBER      | NAME1    | NAME2
------------+----------+-----------
1           | имя1     | имя1
2           | имя2     | имя2
```

Теперь выполним запрос SELECT с указанием INNER JOIN, но в другом варианте — с условием соединения таблиц:

```
select * from TABLE1 inner join TABLE2 on TABLE1.NUMBER = TABLE2.NUMBER;
```

После выполнения команды будет возвращена результирующая таблица, где каждой строке из TABLE1 сопоставлена строка из TABLE2, удовлетворяющая указанному условию соединения:

```
NUMBER      | NAME1    | NUMBER      | NAME2
------------+----------+-------------+-----------
1           | имя1     | 1           | имя1
2           | имя2     | 2           | имя2
```

Рассмотрим запрос SELECT с указанием NATURAL INNER JOIN для естественного внутреннего соединения данных двух таблиц:

```
select * from TABLE1 natural inner join TABLE2;
```

После выполнения команды будет возвращена результирующая таблица, в которую выбраны строки из двух таблиц, имеющих равные значения в одноимённых столбцах:

```
NUMBER      | NAME1    | NAME2
------------+----------+-----------
1           | имя1     | имя1
2           | имя2     | имя2
```

Теперь выполним запрос с указанием CROSS JOIN для перекрёстного соединения данных двух таблиц:

```
select * from TABLE1 cross join TABLE2;
```

После выполнения команды будет возвращена результирующая таблица, в которой указаны все возможные сочетания строк двух таблиц:

```
 NUMBER     | NAME1    | NUMBER      | NAME2
------------+----------+-------------+-----------
1           | имя1     | 1           | имя1
1           | имя1     | 2           | имя2
1           | имя1     | 3           | имя3
1           | имя1     | 4           | имя4
2           | имя2     | 1           | имя1
2           | имя2     | 2           | имя2
2           | имя2     | 3           | имя3
2           | имя2     | 4           | имя4
5           | имя5     | 1           | имя1
5           | имя5     | 2           | имя2
5           | имя5     | 3           | имя3
5           | имя5     | 4           | имя4
```

Аналогичный результат можно получить, указав в конструкции FROM источники данных через запятую:

```
select * from TABLE1, TABLE2;
```

Далее рассмотрим запросы SELECT с указанием внешних соединений.

Выполним запрос с указанием внешнего соединения LEFT и связующего столбца NUMBER, одноимённого для двух таблиц:

```
select * from TABLE1 left join TABLE2 using (NUMBER);
```

После выполнения команды будет возвращена таблица, состоящая из всех строк обеих таблиц, имеющих равные значения в указанном столбце NUMBER каждой таблицы. Дополнительно к этому будут выбраны все строки из таблицы, указанной слева (TABLE1), не имеющие совпадений по столбцу NUMBER со значениями строк

таблицы, указанной справа (TABLE2), и вместо значений правой таблицы указаны значения NULL:

```
NUMBER1     | NAME1    | NAME2
------------+----------+-----------
1           | имя1     | имя1
2           | имя2     | имя2
5           | имя5     | NULL
```

Теперь выполним запрос SELECT с указанием внешнего соединения RIGHT и в качестве условия соединения укажем выражение:

```
select * from TABLE1 right outer join TABLE2 on TABLE1.NUMBER = TABLE2.NUMBER;
```

После выполнения команды будет возвращена результирующая таблица, в которую выбраны все строки обеих таблиц, значения которых соответствуют указанному выражению. Дополнительно к этому будут выбраны все строки из таблицы, указанной справа (TABLE2), не имеющие соответствующих заданному выражению значений строк таблицы, указанной слева (TABLE1), и вместо значений левой таблицы указаны значения NULL:

```
NUMBER     | NAME1    | NUMBER     | NAME2
-----------+----------+------------+-----------
1          | имя1     | 1          | имя1
2          | имя2     | 2          | имя2
NULL       | NULL     | 3          | имя3
NULL       | NULL     | 4          | имя4
```

Для выполнения полного (двустороннего) внешнего соединения в запросе SELECT в следующих версиях Сокола будет реализовано ключевое слово FULL. В текущей реализации для получения результирующего набора без дублирования значений нужно воспользоваться объединением результатов двух запросов:

```
select * from TABLE1 right outer join TABLE2 on TABLE1.NUMBER = TABLE2.NUMBER
 union
select * from  TABLE1 left join TABLE2 on TABLE1.NUMBER = TABLE2.NUMBER;
```

После выполнения команды будет возвращена результирующая таблица, в которую попали строки из обеих таблиц. В случае, когда для строки одной таблицы не нашлось соответствующих строк другой таблицы по указанному выражению соединения, в качестве значений другой таблицы будут выбраны значения NULL:

```
 NUMBER     | NAME1     | NUMBER      | NAME2
--------------+------------+--------------+------------
NULL        | NULL      | 3          |  имя3
NULL        | NULL      | 4          |  имя4
1          | имя1      | 1          |  имя1
2          | имя2      | 2          |  имя2
5          | имя5      | NULL       |  NULL
```

Вернуться в оглавление

12.8.5.6.7. Пример использования <конструкции_ORDER_BY>

Допустим, у нас есть таблица с данными сотрудников. Необходимо сделать выборку данных и для удобства восприятия упорядочить строки по значениям конкретных столбцов.

Для примера была создана таблица WORKERS:

```
TN     | NAME   | LASTNAME| AGE      | GENDER
---------+----- --+---------+----------+--------
212    | Иван   | Петров  | 53       | м
213    | Пётр   | Иванов  | 27       | м
214    | Игорь  | Котов   | 53       | м
215    | Анна   | Гузеева | 70       | ж
216    | Юлия   | Серова  | 65       | ж
217    | Елена  | Попова  | 19       | ж
218    | Ирина  | Зотова  | 48       | ж
219    | Фёдор  | Уваров  | 67       | м
220    | Алёна  | Котова  | 65       | ж
221    | Андрей | Быков   | 65       | м
```

Необходимо сделать выборку сотрудников, возраст которых больше 50 лет, и упорядочить полученный список в порядке уменьшения возраста:

```
select * from WORKERS where AGE>50 order by AGE desc;
```

После выполнения команды будет получен результат:

```
TN     | NAME   | LASTNAME| AGE      | GENDER
---------+--------+---------+----------+-------------
215    | Анна   | Гузеева | 70       | ж
219    | Фёдор  | Уваров  | 67       | м
216    | Юлия   | Серова  | 65       | ж
221    | Андрей | Быков   | 65       | м
220    | Алёна  | Котова  | 65       | ж
212    | Иван   | Петров  | 53       | м
214    | Игорь  | Котов   | 53       | м
```

Обратите внимание, что порядок строк в группах с одинаковым возрастом может получиться произвольным.

Для задания более строгого порядка строк внутри групп с одинаковым значением AGE (например, положим, что внутри групп строки должны быть упорядочены по полю LASTNAME в лексикографическом порядке) добавим ещё один объект упорядочения в конструкцию ORDER BY:

```
select * from WORKERS where AGE>50 order by AGE desc, LASTNAME asc;
```

Будет возвращён следующий результат:

```
 TN     | NAME   | LASTNAME| AGE     | GENDER
---------+--------+---------+---------+-------------
 215    | Анна   | Гузеева | 70      | ж
 219    | Фёдор  | Уваров  | 67      | м
 221    | Андрей | Быков   | 65      | м
 220    | Алёна  | Котова  | 65      | ж
 216    | Юлия   | Серова  | 65      | ж
 214    | Игорь  | Котов   | 53      | м
 212    | Иван   | Петров  | 53      | м
```

Приведём пример упорядочения выборки по номеру столбца. Положим, что из таблицы будут извлекаться следующие значения:

```
TN, LASTNAME || ' ' || NAME as FULLNAME, AGE, GENDER
```

При этом пусть поставлена задача упорядочить результат выборки по значениям второго столбца (то есть по выражению LASTNAME || ' ' || NAME). Тогда запрос приобретет вид:

```
select TN, LASTNAME || ' ' || NAME as FULLNAME, AGE, GENDER from WORKERS order by 2;
```

В данном примере не указано обозначение порядка сортировки. Это значит, что подразумевается порядок по умолчанию ASC. В результате будет получена выборка:

```
 TN     | FULLNAME         | AGE     | GENDER
---------+------------------+---------+--------
 221    | Быков Андрей     | 65      | м
 213    | Иванов Пётр      | 27      | м
 215    | Гузеева  Анна    | 70      | ж
 218    | Зотова Ирина     | 48      | ж
 214    | Котов Игорь      | 53      | м
 220    | Котова Алёна     | 65      | ж
 212    | Петров Иван      | 53      | м
 217    | Попова Елена     | 19      | ж
 216    | Серова Юлия      | 65      | ж
 219    | Уваров Фёдор     | 67      | м
```

Обратите внимание, что результатом применения конструкции ORDER BY 2 стало упорядочение значений во втором столбце FULLNAME.

Вернуться в оглавление

12.8.5.6.8. Пример использования конструкции <ограничение_числа_строк_выборки>

Допустим, у нас есть таблица с данными сотрудников. Необходимо сделать выборку данных по сотрудникам старше 50 лет. Для сравнения работы разных конструкций выполним выборку:

– всех сотрудников, попадающих под данное условие по возрасту;

– только первых трёх, соответствующих заданному условию;

– первых трёх, пропустив сначала первых пять, соответствующих заданному условию.

Для примера была создана и заполнена таблица WORKERS с данными сотрудников компании:

```
TN    | NAME   | LASTNAME| AGE     | GENDER
---------+--------+---------+----------+-------------
212    | Иван   | Петров  | 53      | м
213    | Пётр   | Иванов  | 27      | м
214    | Игорь  | Котов   | 38      | м
215    | Анна   | Гузеева | 70      | ж
216    | Юлия   | Серова  | 65      | ж
217    | Елена  | Попова  | 35      | ж
218    | Ирина  | Андреева| 48      | ж
219    | Фёдор  | Жмуркин | 63      | м
220    | Алёна  | Мишина  | 65      | ж
221    | Андрей | Быков   | 65      | м
222    | Фёдор  | Быков   | 53      | м
223    | Елена  | Попова  | 35      | ж
224    | Юлия   | Гузеева | 53      | ж
225    | Игорь  | Иванов  | 53      | м
226    | Павел  | Иванов  | 37      | м
```

Для выборки данных всех сотрудников старше 50 лет выполним запрос:

```
select * from WORKERS where AGE>50;
```

Будут получены данные:

```
TN    | NAME   | LASTNAME| AGE     | GENDER
---------+--------+---------+----------+-------------
212    | Иван   | Петров  | 53      | м
215    | Анна   | Гузеева | 70      | ж
216    | Юлия   | Серова  | 65      | ж
219    | Фёдор  | Жмуркин | 63      | м
220    | Алёна  | Мишина  | 65      | ж
221    | Андрей | Быков   | 65      | м
222    | Фёдор  | Быков   | 53      | м
224    | Юлия   | Гузеева | 53      | ж
```

```
225    | Игорь  | Иванов  | 53      | м
```

Теперь выполним запрос только первых трёх сотрудников, соответствующих заданному условию:

```
select * from WORKERS where AGE>50 fetch first 3 rows only;
```

Будут получены данные:

```
TN     | NAME   | LASTNAME| AGE     | GENDER
---------+--------+---------+---------+-------------
212    | Иван   | Петров  | 53      | м
215    | Анна   | Гузеева | 70      | ж
216    | Юлия   | Серова  | 65      | ж
```

И выполним запрос первых трёх сотрудников, пропустив сначала первые пять записей, соответствующие заданному условию:

```
select * from WORKERS where AGE>50 limit 3 offset 5;
```

Будут получены данные:

```
TN     | NAME   | LASTNAME| AGE     | GENDER
---------+--------+---------+---------+-------------
221    | Андрей | Быков   | 65      | м
222    | Фёдор  | Быков   | 53      | м
224    | Юлия   | Гузеева | 53      | ж
```

Вернуться в оглавление

12.8.5.6.9. Пример применения конструкции <блокировка_строк>

Допустим, у нас есть таблица с данными абонентов. Одновременно с данной таблицей работают двое сотрудников: один из них вносит изменения в данные, а второй выполняет запросы на выборку данных.

При выполнении операций с данными каждый сотрудник должен понимать, какие данные он получит: полные или нет, актуальные (с учётом изменений другими пользователями) или нет.

Для примера возьмём таблицу PHONE с данными абонентов, которая находится в схеме CLIENTS:

```
NUMBER     | NAME     | AGE     | CITY
-------------+----------+---------+----------
89876543212 | Петров   | 53      | Воронеж
89876543214 | Иванов   | 46      | Москва
89876543215 | Попова   | 26      | Воронеж
```

```
89876543244 | Котов    | 37     | Москва
89876543255 | Уваров   | 25     | Воронеж
89876543266 | Лебедева | 37     | Москва
89876543277 | Ермаков  | 38     | Химки
89876543288 | Тихонова | 29     | Москва
89876543299 | Ермакова | 45     | Химки
89876543300 | Лебедев  | 37     | Воронеж
```

Сотрудник Ильин собирается вносить изменения в таблицу. Он явно начинает транзакцию в ручном режиме для того, чтобы при необходимости отменить внесённые изменения:

```
start transaction;
```

Далее он вносит изменения в данные по абоненту:

```
update CLIENTS.PHONE set NAME = 'Яшин' where NUMBER = 89876543255;
```

Команда выполнена успешно, и Ильин продолжает работу, не заканчивая транзакцию. Пока Ильин не завершил транзакцию, изменяемые им строки заблокированы.

В это же время сотруднику Наумову необходимо также выполнить изменения данных таблицы, и он тоже начинает транзакцию явно:

```
start transaction;
```

Для начала Наумов отправляет запрос с указанием блокировки строк и принципом поведения относительно уже заблокированных строк параллельной транзакцией NOWAIT:

```
select NUMBER, CITY from CLIENTS.PHONE where CITY='Воронеж' for update NOWAIT;
```

Возвращается ошибка и результат, собранный до первой заблокированной строки:

```
 NUMBER     | CITY
------------+-----------
 89876543212 | Воронеж
 89876543215 | Воронеж
```

Раз вернулась ошибка, значит какие-то строки изменяются другим пользователем.

Допустим, пользователь знает, что его коллега вносит минимальные правки, и готов получить результат без учёта изменяемых строк.

Для этого сотрудник Наумов отправляет запрос с указанием ключевых слов SKIP LOCKED, что позволит выполнить выборку, пропуская изменяемые другим пользователем строки:

```
select NUMBER, CITY from PHONE where CITY='Воронеж' for update SKIP LOCKED;
```

Возвращается результат, в котором учтены все строки таблицы, а заблокированные параллельной транзакцией — пропущены:

```
 NUMBER     | CITY
-------------+-----------
 89876543212 | Воронеж
 89876543215 | Воронеж
 89876543300 | Воронеж
```

Если Наумов знает, что коллега вносит серьёзные правки в данные таблицы, и ему необходим результат выборки данных с учётом всех выполненных и сохранённых изменений, то в запросе ему необходимо указать просто FOR UPDATE:

```
 select NUMBER, CITY from PHONE where CITY='Воронеж' for update;
```

В этом случае результат будет возвращён тогда, когда сотрудник Ильин завершит транзакцию явно:

```
commit;
```

Только после этого Наумов получит результат на свой запрос:

```
 NUMBER     | CITY
-------------+-----------
 89876543212 | Воронеж
 89876543215 | Воронеж
 89876543255 | Воронеж
 89876543300 | Воронеж
```

Вернуться в оглавление

12.8.6. INSERT INTO TABLE

Команда добавления данных в таблицу. Подробнее о таблицах см. п. 12.5.1.

Выполнить добавление данных в таблицу командой INSERT INTO TABLE может:

– владелец таблицы;

– пользователь базы данных с объектной привилегией INSERT;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Для команды INSERT INTO TABLE используется два варианта синтаксиса: с вариантом добавления заданных значений и результатов SELECT-запроса.

Вернуться в оглавление

12.8.6.1. INSERT INTO TABLE с добавлением в таблицу заданных значений

Синтаксис команды:

```
INSERT INTO <имя_таблицы> [(<имя_столбца>[, имя_столбца...])]
  VALUES (<значение>[, <значение>...]) [<конструкция_RETURNING>];

<значение> ::= <выражение> | NULL | DEFAULT
```

Где:

1) <имя_таблицы> — имя таблицы, в которую необходимо добавить данные.

2) <имя_столбца> — имя столбца, в строку которого необходимо добавить <значение>, указанное после ключевого слова VALUES. Указание параметра <имя_столбца> не обязательно. При его отсутствии подразумевается список имён столбцов, указанный при создании таблицы. При указании имени одного и того же столбца больше одного раза будет возвращена ошибка.

3) <значение> — значение данных, которое необходимо добавить в таблицу. Значение может быть:

– <выражение> — значение параметра (значение заданного выражения);

– NULL — означает, что в случае отсутствия значения при вставке данных и явного упоминания столбца будет добавлено NULL-значение;

– DEFAULT — означает, что в случае отсутствия значения при вставке данных и явного упоминания столбца будет добавлено значение по умолчанию, указанное при создании таблицы.

Список значений должен соответствовать числу имён столбцов команды INSERT. При отсутствии списка имён столбцов необходимо внести значения в строки всех столбцов, указанных при создании таблицы.

4) <конструкция_RETURNING> — конструкция, позволяющая получить добавленные данные (подробнее про конструкцию RETURNING см. п. 12.8.9 Конструкция RETURNING). Конструкцию RETURNING удобно применять в команде INSERT с целью получения значений, вычисленных в процессе вставки данных. Конструкция RETURNING не обязательна.

Пример применения команды INSERT INTO TABLE с добавлением в таблицу заданных значений

Допустим, в компании есть таблица PHONE с данными абонентов:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543212 | Петров  | 53   | Воронеж
89876543214 | Иванов  | 46   | Москва
89876543244 | Котов   | 37   | Москва
89876543255 | Попова  | 35   | Воронеж
89876543266 | Лебедев | 42   | Москва
```

Номера телефонов абонентов должны быть уникальны, поэтому для соблюдения данного требования по столбцу NUMBER установлено ограничение PRIMARY KEY.

Допустим, новый абонент Егоров, возрастом 38 лет, из Воронежа, был подключен с номером 89876543277. Вся информация об абоненте известна и её можно внести в таблицу.

Для этого используем команду **INSERT INTO TABLE**, без указания списка имён столбцов, поэтому значения столбцов указываются в порядке, определённом при создании таблицы:

```
insert into PHONE values (89876543277, 'Егоров', 38, 'Воронеж');
```

После успешно выполненной команды можно сделать проверку, сформировав запрос данных из таблицы.

```
select * from PHONE where NUMBER = 89876543277;
```

Будут получены данные:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543277  | Егоров  | 38   | Воронеж
```

Теперь допустим, что новый абонент при подключении предоставил не все данные.

В этом случае, при внесении данных указываем только те столбцы, по которым будут внесены значения:

```
insert into PHONE (NUMBER, NAME, AGE) values (89876543288, 'Беляев', 38);
```

По тому столбцу, по которому данные указаны не были, будет внесено значение NULL:

```
 NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
 89876543288 | Беляев  | 38   | NULL
```

Если при создании таблицы PHONE для столбца CITY было указано значение по умолчанию (например, DEFAULT '<заполнить>') и при вставке данных в таблицу командой INSERT этот столбец явно не упоминается, то система автоматически его заполнит:

```
 NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
 89876543300 | Антонов | 43   | <заполнить>
```

Вернуться в оглавление

12.8.6.2. INSERT INTO TABLE с возможностью добавления в таблицу результата SELECT-запроса

Синтаксис команды:

```
INSERT INTO <имя_таблицы> [(<имя_столбца>[, имя_столбца...])]
 <SELECT-запрос> [<конструкция_RETURNING>]
```

Где:

– <имя_таблицы> — имя таблицы, в которую необходимо добавить данные;

– <имя_столбца> — имя столбца, в который будут добавляться значения из соответствующего по номеру столбца результатов <SELECT-запроса>. Указание

параметра <имя_столбца> не обязательно. При его отсутствии подразумевается список имён столбцов, указанный при создании таблицы. При указании имени одного и того же столбца больше одного раза будет возвращена ошибка;

– <SELECT-запрос> — используется для выборки значений из строк другой таблицы для их последующей вставки. Количество столбцов результата <SELECT-запроса> должно соответствовать количеству имён столбцов команды INSERT. Подробнее описание синтаксиса команды SELECT см. п. 12.8.5. SELECT;

– <конструкция_RETURNING> — конструкция, позволяющая оперативно получить добавленные данные (подробнее про конструкцию RETURNING см. п. 12.8.9. Конструкция RETURNING). Конструкцию RETURNING удобно применять в команде INSERT с целью получения значений, вычисленных в процессе вставки данных. Конструкция RETURNING не обязательна.

Пример применения команды INSERT INTO TABLE с добавлением в таблицу результата SELECT-запроса

Допустим, у нас есть таблица CALLER с данными абонентов:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543212 | Петров  | 53   | Воронеж
89876543214 | Иванов  | 46   | Москва
89876543244 | Котов   | 37   | Москва
89876543255 | Попова  | 35   | Воронеж
89876543266 | Лебедев | 42   | Москва
```

И есть вторая таблица PHONE_MOSCOW:

```
NUMBER     | NAME    | AGE
-------------+---------+------
89876543301 | Шубин   | 53
89876543302 | Лукин   | 46
89876543303 | Шилов   | 37
```

В таблицу PHONE_MOSCOW необходимо перенести номера и фамилии абонентов Москвы из таблицы CALLER.

Для выполнения данной вставки указывается INSERT с SELECT:

```
insert into PHONE_MOSCOW select NUMBER, NAME, AGE from CALLER where CITY = 'Москва';
```

Выполним запрос данных из таблицы PHONE_MOSCOW и получим:

```
NUMBER     | NAME    | AGE
-------------+---------+------
89876543301 | Шубин   | 53
89876543302 | Лукин   | 46
89876543303 | Шилов   | 37
89876543214 | Иванов  | 46
89876543244 | Котов   | 37
89876543266 | Лебедев | 42
```

В полученных данных видно, что были вставлены значения только указанных столбов.

Вернуться в оглавление

12.8.7. UPDATE TABLE

Команда изменения значений в таблице. Подробнее о таблицах см. п. 12.5.1 Таблица (TABLE).

Выполнить изменение значений в таблице может:

– владелец таблицы;

– пользователь базы данных с объектной привилегией UPDATE;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды изменения строки таблицы:

```
UPDATE <имя_таблицы> SET <имя_столбца> = <выражение> [, <имя_столбца> = <выражение> ...]
   [WHERE <выражение>]
   [<конструкция_RETURNING>];
```

Где:

– <имя_таблицы> — имя таблицы, в строки которой необходимо внести изменения;

– <имя_столбца> = <выражение> — конструкция позволяет указать имя столбца, значения в котором необходимо заменить на новое значение (значение заданного <выражения>). Может быть указан вложенный SELECT-запрос как элемент <выражения>, который возвращает одно значение для каждой обновляемой строки. Если вложенный SELECT-запрос не возвращает значение, то его результатом будет полагаться значение NULL;

– WHERE <выражение> — конструкция позволяет внести изменения только в те строки, для которых указанное значение (значение заданного выражения) равно TRUE. Конструкция необязательная. В случае её отсутствия изменения вносятся во все строки таблицы;

– <конструкция_RETURNING> — конструкция, позволяющая получить обновлённые данные (подробнее про конструкцию RETURNING см. п. 12.8.9 Конструкция RETURNING). Конструкция RETURNING не обязательна.

Пример применения команды UPDATE TABLE

Рассмотрим примеры применения команды изменения строк таблицы.

Допустим, в компании есть таблица PHONE с данными абонентов:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543212 | Петров  | 53   | Воронеж
89876543214 | Иванов  | 46   | Москва
89876543244 | Котов   | 37   | Москва
89876543255 | Попова  | 35   | Воронеж
89876543266 | Лебедев | 42   | Москва
```

Номера телефонов абонентов должны быть уникальны, поэтому по столбцу NUMBER установлено ограничение PRIMARY KEY.

У одного из абонентов изменилась фамилия и необходимо изменить в данные таблицы.

Для этого используем команду UPDATE, указывая все необходимые данные:

```
update PHONE set NAME = 'Блинова' where NUMBER = 89876543255;
```

После успешно выполненной команды, для проверки верности внесённых данных, можно выполнить запрос:

```
select * from PHONE where NUMBER = 89876543255;
```

Результат выполнения запроса будет таким:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543255 | Блинова | 35   | Воронеж
```

Допустим, к номеру переподключили нового абонента. Это потребует обновления данных строки, в поле NUMBER которой находится нужный номер.

Для этого вводим команду UPDATE с новыми данными:

```
update PHONE set NAME = 'Сафонов', AGE = 47, CITY = Воронеж where NUMBER = 89876543244;
```

Для проверки выполним запрос данных для номера, по которому вносили изменения. Будут возвращены новые данные:

```
NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
89876543244 | Сафонов | 47   | Воронеж
```

Вернуться в оглавление

12.8.8. DELETE

Команда удаления одной, нескольких или всех строк таблицы. Подробнее о таблицах см. п. 12.5.1. Таблица (TABLE).

Выполнить удаление одной, нескольких или всех строк таблицы может:

– владелец таблицы;

– пользователь базы данных с объектной привилегией DELETE;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды удаления одной, нескольких или всех строк таблицы:

```
DELETE FROM <имя_таблицы> [WHERE <выражение>] [<конструкция_RETURNING>];
```

Где:

– <имя_таблицы> — имя таблицы, в которой необходимо удалить одну, несколько или все строки;

– WHERE <выражение> — конструкция позволяет удалить только те строки, для которых указанное значение (значение заданного выражения) равно TRUE. Конструкция WHERE <выражение> не обязательна. Если конструкция WHERE <выражение> отсутствует, то команда DELETE удаляет из таблицы все строки. В результате будет получена пустая таблица;

– <конструкция_RETURNING> — конструкция, позволяющая получить данные из удалённых строк (подробнее про конструкцию RETURNING см. п. 12.8.9. Конструкция RETURNING). Конструкция RETURNING не обязательна.

Пример применения команды DELETE

Рассмотрим примеры применения команды удаления строк из таблицы.

Допустим, в компании есть таблица PHONE с данными абонентов:

```
NUMBER     | NAME    | AGE  | CITY
-----------+----------+------+-----------
89876543212 | Петров   | 53   | Воронеж
89876543222 | Гурьева  | 35   | Ульяновск
89876543233 | Хохлов   | 43   | Ульяновск
89876543214 | Иванов   | 46   | Москва
89876543244 | Котов    | 37   | Москва
89876543255 | Попова   | 35   | Воронеж
89876543266 | Лебедев  | 42   | Москва
89876543277 | Дроздова | 43   | Ульяновск
```

Считаем, что уникальность номеров телефонов абонентов реализована при помощи ограничения PRIMARY KEY для столбца NUMBER.

Положим, один из абонентов с номером 89876543255 перешёл в другую компанию, и его данные больше хранить не нужно. Новый абонент на данный номер пока не подключен. Поэтому строку с данными по этому номеру можно удалить.

Для удаления строки вводим команду:

```
delete from PHONE where NUMBER = 89876543255;
```

После успешно выполненной команды для проверки можно выполнить запрос:

```
select * from PHONE where NUMBER = 89876543255;
```

В ответ на данный запрос сервер возвращает пустой результат. Это значит, что такой записи в таблице больше нет.

Допустим, абонентов компании в разных городах стало много и для удобства работы было принято решение вести данные об абонентах каждого города раздельно. Данные предварительно были перенесены в соответствующие таблицы и из первоначальной таблицы необходимо удалить всех абонентов, кроме абонентов из Воронежа.

Для этого указываем команду DELETE с необходимыми параметрами:

```
delete from PHONE where CITY = 'Москва' or CITY = 'Ульяновск';
```

Для проверки извлечём все данные из таблицы:

```
select * from PHONE;
```

В ответ на данный запрос сервер возвращает следующий результат:

```
 NUMBER     | NAME    | AGE  | CITY
------------+---------+------+----------
 89876543212 | Петров   | 53   | Воронеж
 89876543255 | Попова   | 35   | Воронеж
```

В продолжении примера допустим, что филиал компании прекратил обслуживание абонентов в городе Воронеж и можно удалить все записи из таблицы, сохранив саму таблицу для возможности дальнейшей работы с ней.

Для этого вводим команду DELETE без указания условия WHERE:

```
delete from PHONE;
```

После успешно выполненной команды можно сделать проверку, сформировав запрос всех данных из таблицы.

В ответ сервер вернёт пустую таблицу.

Таким образом, при верно указанных условиях команда DELETE позволяет удалить одну, несколько или все строки таблицы.

Вернуться в оглавление

12.8.9. Конструкция RETURNING

Конструкция RETURNING позволяет получать данные из строки в процессе её изменения/добавления/удаления, что даёт возможность обойтись без дополнительного SELECT-запроса к БД для выборки изменённых данных.

Синтаксис конструкции RETURNING:

```
RETURNING <выражение>[, <выражение>...] [INTO <переменная> [, <переменная>...] ]

<переменная> = <имя_переменной> | :<имя_параметра> | ?
```

Где:

1) <выражение> — значение параметра или выражение, вычисленное над строкой: при изменении/добавлении строки — после операции, при удалении строки — до операции.

2) INTO <переменная>[, <переменная>...] — используется (в зависимости от контекста выполнения команды) для указания выходных параметров (переменных), в которых будут сохраняться соответствующие по порядку значения, вычисленные в каждом параметре <выражение>. Выходной параметр может выглядеть как:

– именованный параметр <имя_переменной> — при отправке запроса из процедурного языка;

– именованный параметр «:<имя_параметра>» при отправке запроса через vsql_console;

– неименованный параметр «?» при отправке запроса через интерфейсы (напр., ODBC, JDBC и др.).

Для каждого значения параметра <выражение> необходимо указать соответствующий совместимый по типу параметр в конструкции INTO.

Запрос, в котором указана конструкция INTO, должен возвращать однострочный результат, иначе команда завершится с ошибкой. Если команда возвращает табличный результат, то в текущей реализации СУБД СОКОЛ нужно удалить фразу INTO и обработать табличный результат как результат запроса.

Конструкция INTO не обязательна.

Конструкцию RETURNING можно использовать в командах INSERT, UPDATE и DELETE.

Пример применения конструкции RETURNING

Для примера была создана и заполнена таблица WORKERS с данными сотрудников компании, в которой по столбцу TN указано ограничение PRIMARY KEY:

```
 TN     | NAME  | LASTNAME| GENDER
---------+-------+---------+---------
 212    | Иван  | Петров  | м
 213    | Пётр  | Иванов  | м
 214    | Игорь | Котов   | м
 215    | Анна  | Гузеева | ж
```

```
216   | Юлия  | Серова  | ж
217   | Елена | Попова  | ж
```

При приёме нового сотрудника в таблицу добавляются данные о нём, а столбец TN с табельными номерами сотрудников заполняется уникальными значениями из генератора последовательности SEQ1 (см. пример для п. 12.8.1.4 CREATE SEQUENCE).

Табельный номер сотрудника можно получить отдельным запросом SELECT, но для этого необходимо знать уникальный идентификатор этой строки. Т.е. в таблице должно быть поле (или набор полей) с уникальными значениями. В противном случае в результирующий набор может попасть множество строк, значения которых будут совпадать с условиями.

В таблице WORKERS только одно поле с уникальными значениями — TN. Поэтому воспользоваться запросом SELECT для получения табельного номера сотрудника из этой таблицы не получится.

В этом случае можно воспользоваться конструкцией RETURNING и получить табельный номер сотрудника одновременно с добавлением данных в таблицу.

Допустим, был принят новый сотрудник — Игорь Иванов, возрастом 53 года. Данные о новом сотруднике внесём в таблицу и в конструкции RETURNING укажем имя столбца TN:

```
INSERT INTO WORKER (NAME, LASTNAME, AGE, GENDER) values ('Игорь', 'Иванов', 53, 'м') returning TN;
```

Будет получена строка с табельным номером нового сотрудника:

```
TN
-------------
 225
```

Попробуем при внесении новых значений в конструкции RETURNING указать имена всех столбцов:

```
INSERT INTO WORKER (NAME, LASTNAME, AGE, GENDER) values ('Антон', 'Мухин', 38, 'м') returning TN, NAME,
LASTNAME, AGE, GENDER;
```

Будут возвращены значения соответствующей строки по всем указанным столбцам:

```
TN     | NAME  | LASTNAME| AGE     | GENDER
---------+-------+---------+----------+--------
226    | Антон | Мухин   | 38      | м
```

Рассмотрим ещё один пример с использованием в конструкции RETURNING параметра консоли.

Для примера также возьмём таблицу WORKERS с данными сотрудников компании:

```
TN     | NAME  | LASTNAME| AGE     | GENDER
---------+-------+---------+----------+--------
212    | Иван  | Петров  | 53      | м
213    | Пётр  | Иванов  | 27      | м
214    | Игорь | Котов   | 38      | м
215    | Анна  | Гузеева | 70      | ж
216    | Юлия  | Серова  | 65      | ж
217    | Елена | Попова  | 19      | ж
218    | Ирина | Зотова  | 48      | ж
219    | Фёдор | Уваров  | 67      | м
220    | Алёна | Котова  | 65      | ж
221    | Андрей| Быков   | 65      | м
222    | Фёдор | Быков   | 53      | м
223    | Елена | Попова  | 23      | ж
224    | Юлия  | Гузеева | 53      | ж
```

При приёме нового сотрудника в таблицу добавляются данные о нём, а столбец TN с табельными номерами сотрудников заполняется значениями генератора последовательности SEQ1 (см. пример для п. 12.8.1.4 CREATE SEQUENCE).

Табельный номер сотрудника можно получить одновременно с добавлением данных в таблицу. Для этого в команде INSERT указывается конструкция RETURNING.

При работе с СУБД в консольном режиме и выполнении нескольких команд подряд одно из полученных значений можно сохранить в именованном параметре консоли, а затем использовать его в последующих командах.

Рассмотрим это на примере и для начала объявим в консольном режиме переменную TAB:

```
variable TAB number;
```

При внесении данных о новом сотруднике в конструкции RETURNING указываем имя столбца, значение которого необходимо сохранить в переменную, и имя самой переменной TAB:

```
INSERT INTO WORKER (NAME, LASTNAME, AGE, GENDER) values ('Игорь', 'Иванов', 53, 'м') returning TN into :TAB;
```

Далее сохранённое в переменной консоли значение может быть использовано при необходимости: просто выведено значение или вставлено в другую таблицу.

При необходимости вывода в консоли значения, сохранённого в переменной консоли, вводим:

```
print TAB;
```

Будет получена таблица с сохранённым значением:

```
TAB
----------
 4
```

Если значение, сохранённое в переменной консоли, необходимо вставить в другую таблицу (например, FORREPORTCARD), то команда будет выглядеть так:

```
insert into FORREPORTCARD (TN) values (:TAB);
```

Значение в переменной консоли может храниться только одно и до тех пор, пока не будет заменено другим или до завершения работы консольной утилиты.

Вернуться в оглавление

12.8.10. EXPLAIN

Команда позволяет получить план выполнения запроса, построенный планировщиком СУБД ЛИНТЕР СОКОЛ. Используется для оптимизации запросов, получения информации об использованных и возможных индексах.

В текущей реализации СУБД СОКОЛ команда работает с командой SELECT.

Получить план выполнения запроса может:

– владелец таблицы, представления;

– пользователь базы данных с объектными привилегиями SELECT;

– пользователь базы данных с привилегией DATABASE ADMIN.

Синтаксис команды для получения плана выполнения запроса:

```
EXPLAIN <SELECT-запрос>;
```

Где <SELECT-запрос> — запрос, план выполнения которого необходимо получить.

Результат команды — описание плана в формате таблицы для указанного запроса, где каждая строка описывает одну операцию над данными.

Пример применения команды EXPLAIN

Допустим, у нас есть таблица WORKERS с данными сотрудников компании:

```
 TN     | NAME  | LASTNAME| AGE     | GENDER
---------+-------+---------+----------+--------
 212    | Иван  | Петров  | 53      | м
 213    | Пётр  | Иванов  | 27      | м
 214    | Игорь | Котов   | 38      | м
 215    | Анна  | Гузеева | 70      | ж
 216    | Юлия  | Серова  | 65      | ж
 217    | Елена | Попова  | 19      | ж
 218    | Ирина | Зотова  | 48      | ж
 219    | Фёдор | Уваров  | 67      | м
 220    | Алёна | Котова  | 65      | ж
 221    | Андрей| Быков   | 65      | м
 222    | Фёдор | Быков   | 53      | м
 223    | Елена | Попова  | 23      | ж
 224    | Юлия  | Гузеева | 53      | ж
 225    | Игорь | Иванов  | 53      | м
```

Необходимо сделать выборку данных всех сотрудниц с фамилией Попова, которых может быть несколько.

Для понимания, как система выполнит запрос и необходима ли будет оптимизация запроса, укажем EXPLAIN перед конкретным запросом:

```
 explain select * from WORKERS where LASTNAME = 'Попова';
```

Будет возвращён план выполнения запроса, с информацией о каждом шаге:

```
 OPERATION| OPTIONS          | OBJECT_TYPE| OBJECT_NAME| OBJECT_ALIAS| DEPTH| COST| CARDINALITY| PROJECTION
----------+------------------------+------------+------------+-------------+------+-----+------------+---------------
 SELECT   |                |        |         |         | 1    | 0   | 1        | WORKERS.TN;
                                                                                    WORKERS.NAME;
                                                                                   WORKERS.LASTNAME;
                                                                                    WORKERS.AGE;
                                                                                    WORKERS.GENDER
 SCAN     | FILTER:LASTNAME=QP#1| TABLE     | WORKERS    |           | 2    | 1000| 1        | WORKERS.TN;
                                                                                    WORKERS.NAME;
```

```
                                                    WORKERS.LASTNAME;
                                                WORKERS.AGE; WORKERS.GENDER
```

Здесь указано, что при выполнении запроса проводится сканирование всех записей таблицы (SCAN) и оценка стоимости выполнения запроса в условных временных единицах равна 1000 (COST).

Сравним полученный результат поиска со случаем, когда будет построен индекс по столбцу таблицы, среди значений которого производится поиск.

Построим простой индекс по столбцу LASTNAME таблицы WORKERS:

```
create index WORKERS_IDX1 on WORKERS ("LASTNAME");
```

Такой индекс должен ускорить поиск информации по значениям столбца LASTNAME.

Проверим предположение, запустив тот же запрос с указанием EXPLAIN. Будет возвращён следующий план выполнения запроса:

```
OPERATION | OPTIONS          | OBJECT_TYPE| OBJECT_NAME| OBJECT_ALIAS| DEPTH | COST | CARDINALITY | PROJECTION
----------+------------------+------------+------------+-------------+-------+------+-------------+-----------
SELECT    |                  |            |            |             | 1     | 0    | 1           | WORKERS.TN;
                                                                                                    WORKERS.NAME;
                                                                                                   ORKERS.LASTNAME;
                                                                                                    WORKERS.AGE;
                                                                                                   WORKERS.GENDER
SCAN/LOOKUP| INDEX:WORKERS_IDX1 | TABLE      | WORKERS    |             | 2     | 900  | 3           | WORKERS.TN;
                                                                                                    WORKERS.NAME;
                                                                                                \KEYS:QP#1 WORKERS.LASTNAME;
                                                                                                    WORKERS.AGE;
                                                                                               WORKERS.GENDER \WHILE:LASTNAME=QP#1
```

При сравнении двух запросов видно, что во втором случае:

– проводится сканирование не всех записей таблицы (SCAN), а ограниченное сканирование с использованием индекса WORKERS_IDX1 (OPTIONS);

– сократилась стоимость выполнения запроса в условных временных единицах: 900 против 1000 (COST).

Вернуться в оглавление

12.8.11. CALL

Команда вызова хранимой процедуры. Подробнее о хранимых процедурах см. п. 12.5.8 Хранимая процедура (PROCEDURE).

Выполнить вызов хранимой процедуры может:

– владелец схемы, которой принадлежит процедура;

– пользователь базы данных с системной привилегией RESOURCES CONTROL и объектной привилегией EXECUTE на хранимую процедуру;

– пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды вызова хранимой процедуры:

```
CALL <имя_процедуры> [(<аргумент>[, <аргумент>…])];

<аргумент> = <выражение>[, <выражение>…] | <параметр> [, <параметр>…]

<параметр> = :<имя_параметра> | ?
```

Где:

– <имя_процедуры> — имя хранимой процедуры, которую необходимо вызвать;

– <выражение> — значение параметра или выражение для вычисления значения.

Указывать параметр <выражение> не обязательно;

– <параметр> — используется для указания выходных параметров запроса.

Выходной параметр может выглядеть как:

– именованный параметр <имя_переменной> — при отправке запроса из процедурного языка;

– именованный параметр «:<имя_параметра>» при отправке запроса через vsql_console;

– неименованный параметр «?» при отправке запроса через интерфейсы (напр., ODBC, JDBC и др.).

На применение аргументов к процедурам есть ограничение: число аргументов, включая любые возвращаемые аргументы, ограничено 1000.

Вернуться в оглавление

12.8.12. Команды работы с транзакциями

Команды используются для работы с транзакциями. Подробнее о работе с транзакциями см. п. 12.6. Работа с транзакциями.

Для справки: перед выполнением примеров для некоторых команд работы с транзакциями необходимо понимать, включен ли режим AUTOCOMMIT. Определить активность режима AUTOCOMMIT в рамках сессии можно запросом:

```
select value from sys._vsession_options where name = 'autocommit';
```

Если режим AUTOCOMMIT включен, запрос вернёт значение, равное on (off - при отключенном режиме AUTOCOMMIT).

Изменение статуса AUTOCOMMIT будет отражено только в случае его изменения с помощью ALTER SESSION. Неявное отключение AUTOCOMMIT при выполнении START TRANSACTION не учитывается в запросе SYS._VSESSION_OPTIONS.

Вернуться в оглавление

12.8.12.1. SET TRANSACTION ISOLATION LEVEL

Команда установки уровня изоляции для всех последующих транзакций в рамках текущей сессии.

Команда выполняется в период между транзакциями.

Выполнить установку уровня изоляции для всех последующих транзакций в рамках текущей сессии может любой пользователь, подсоединившийся к БД.

Синтаксис команды установки уровня изоляции:

```
SET TRANSACTION ISOLATION LEVEL <уровень_изоляции>;

<уровень_изоляции> = READ COMMITTED [SNAPSHOT]
                   | REPEATABLE READ
                   | SERIALIZABLE [SNAPSHOT]
```

Где:

– <уровень_изоляции> — устанавливаемый уровень изоляции для текущей транзакции;

– READ COMMITTED — самый низкий уровень изоляции, реализованный в СУБД СОКОЛ. READ COMMITTED является уровнем изоляции по умолчанию;

– REPEATABLE READ — в СУБД СОКОЛ поддерживается для совместимости ПО, но реализован подобно уровню SERIALIZABLE с параметром SNAPSHOT;

– SERIALIZABLE — самый высокий уровень изоляции в СУБД СОКОЛ;

– SNAPSHOT — модификатор, устанавливаемый по умолчанию, при котором транзакция с уровнем изоляции READ COMMITTED и SERIALIZABLE читает данные следующим образом:

1) В случае READ COMMITED — зафиксированные на момент начала оператора.

2) В случае SERIALIZABLE — на момент начала транзакции. Это повышает вероятность откатов при конкурентной работе.

Вернуться в оглавление

12.8.12.2. START TRANSACTION

Команда запуска новой транзакции с возможностью установки уровня изоляции всех последующих транзакций в рамках текущей сессии.

Команда выполняется в период между транзакциями.

Команда отключает режим AUTOCOMMIT, и транзакция будет выполняться до тех пор, пока не будет задан явный COMMIT или ROLLBACK.

Выполнить запуск новой транзакции с возможностью установки уровня изоляции всех последующих транзакций в рамках текущей сессии может любой пользователь, подсоединившийся к БД.

Синтаксис команды запуска новой транзакции:

```
START TRANSACTION [ISOLATION LEVEL <уровень_изоляции>];

<уровень_изоляции> = READ COMMITTED [SNAPSHOT]
                   | REPEATABLE READ
                   | SERIALIZABLE [SNAPSHOT]
```

Где:

– ISOLATION LEVEL <уровень_изоляции> — конструкция, позволяющая указать устанавливаемый уровень изоляции для текущей транзакции. При отсутствии данной конструкции команда START TRANSACTION запускает новую транзакцию с отключенным AUTOCOMMIT;

– READ COMMITTED — самый низкий уровень изоляции, реализованный в СУБД СОКОЛ. READ COMMITTED является уровнем изоляции по умолчанию;

– REPEATABLE READ — в СУБД СОКОЛ поддерживается для совместимости ПО, но реализован подобно уровню SERIALIZABLE с параметром SNAPSHOT;

– SERIALIZABLE — самый высокий уровень изоляции в СУБД СОКОЛ;

– SNAPSHOT — модификатор, устанавливаемый по умолчанию, при котором транзакция с уровнем изоляции READ COMMITTED и SERIALIZABLE читает данные следующим образом:

1) в случае READ COMMITED — зафиксированные на момент начала оператора;

2) в случае SERIALIZABLE — на момент начала транзакции. Это повышает вероятность откатов при конкурентной работе.

Подробнее про SET TRANSACTION ISOLATION LEVEL см. п. 12.8.12.1 SET TRANSACTION ISOLATION LEVEL.

Пример применения команды START TRANSACTION

Допустим, имеется таблица WORKERSZP с данными по оплате труда сотрудников компании:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-------------+----------------+-------------------+-------------
101       | Зощенко П.М.| 100000         | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000          | главный специалист | МТО
103       | Иванов Ф.Е. | 70000          | главный специалист | МТО
104       | Иванова А.С.| 50000          | специалист 1 кат   | МТО
105       | Петров Г.С. | 50000          | специалист 1 кат   | МТО
106       | Попова А.П. | 40000          | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000          | специалист 2 кат   | МТО
```

Считаем, что уникальность табельных номеров сотрудников реализована при помощи ограничения PRIMARY KEY для столбца Таб №.

Для примера применения команды START TRANSACTION и работы в транзакции с отключенным AUTOCOMMIT выполним несколько различных команд DML, установим одну точку сохранения и отменим изменения, выполненные после точки сохранения.

Для сохранения изменений, выполненных до установки точки сохранения, завершим транзакцию командой фиксации изменений.

Перед началом выполнения примера убедитесь, что или включен режим AUTOCOMMIT (по умолчанию для сессии), или текущая транзакция завершена. В противном случае команда начала транзакции из примера ниже будет завершена с ошибкой. Подробнее о способе определения активности режима AUTOCOMMIT см. п. 12.8.12 Команды работы с транзакциями.

Запустим новую транзакцию:

```
start transaction;
```

Изменим фамилию сотрудника с табельным номером 106 на Жукова:

```
update WORKERSZP set "ФИО" = 'Жукова' WHERE "Таб №" = 106;
```

Установим точку сохранения:

```
savepoint FIO;
```

Далее выполним ещё одно изменение — добавим в таблицу нового сотрудника:

```
insert into WORKERSZP values (108, 'Беляев Е.П.', 70000, 'главный специалист', 'МТО');
```

Извлечём текущее содержимое таблицы командой:

```
select * from WORKERSZP;
```

Получим следующее состояние:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-----------+---------------+-------------------+-------------
101       | Зощенко П.М.| 100000        | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
104       | Иванова А.С.| 50000         | специалист 1 кат   | МТО
105       | Петров Г.С. | 50000         | специалист 1 кат   | МТО
106       | Жукова А.П. | 40000         | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000         | специалист 2 кат   | МТО
108       | Беляев Е.П. | 70000         | главный специалист | МТО
```

Допустим, стало известно о преждевременности внесения информации по новому сотруднику. Пока транзакция не завершена, отменим изменения до точки сохранения FIO:

```
rollback to savepoint FIO;
```

Извлечём данные по сотрудникам с табельными номерами 106 и 108:

```
select * from WORKERSZP where "Таб №" = 106 or "Таб №" = 108;
```

В связи с тем, что был выполнен откат добавленной строки с табельным номером 108, в результат попадет только строка с табельным номером 106:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-----------+---------------+-------------------+-------------
106       | Жукова А.П. | 40000         | специалист 2 кат   | МТО
```

Для сохранения текущего состояния таблицы (на текущий момент таблица содержит только изменения, совершённые командой update) выполним команду завершения транзакции с фиксацией изменений:

```
commit;
```

Вернуться в оглавление

12.8.12.3. Точка сохранения

В рамках текущей транзакции можно установить одну или несколько точек сохранения, которые позволят отменить изменения в транзакции, сделанные после

этой точки (вместе с отменой изменений удаляются и точки сохранения, созданные после неё).

Подробнее о транзакциях см. п. 12.6. Работа с транзакциями.

Вернуться в оглавление

12.8.12.3.1. SAVEPOINT

Команда создания точки сохранения.

Выполнить создание точки сохранения может любой пользователь, подсоединившийся к БД.

Синтаксис команды создания точки сохранения:

```
SAVEPOINT <имя_точки_сохранения>;
```

Где <имя_точки_сохранения> — имя точки сохранения, после которой в дальнейшем при необходимости можно отменить все изменения.

Если имя создаваемой точки сохранения совпадает с именем уже существующей точки, то более ранняя точка сохранения удаляется. На точки сохранения с другими именами это действие не влияет.

После завершения транзакции все точки сохранения удаляются.

Пример применения команды SAVEPOINT

Допустим, имеется таблица WORKERS с данными сотрудников компании:

```
Таб №     | ФИО        | Оклад         | Должность         | Отдел
------------+-------------+-----------------+--------------------+--------------
101       | Зощенко П.М.| 100000        | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
104       | Иванова А.С.| 50000         | специалист 1 кат   | МТО
105       | Петров Г.С. | 50000         | специалист 1 кат   | МТО
106       | Попова А.П. | 40000         | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000         | специалист 2 кат   | МТО
```

Считаем, что уникальность табельных номеров сотрудников реализована при помощи ограничения PRIMARY KEY для столбца Таб №.

Для примера применения команды SAVEPOINT запустим новую транзакцию, выполним несколько различных команд DML и установим точку сохранения. После этого отменим часть изменений после точки сохранения.

Перед началом выполнения примера убедитесь, что либо включен режим AUTOCOMMIT (по умолчанию для сессии), либо текущая транзакция завершена (подробнее о способе определения активности режима AUTOCOMMIT см. п. 12.8.12. Команды работы с транзакциями). В противном случае команда начала транзакции из примера ниже завершится с ошибкой.

Запустим новую транзакцию:

```
start transaction;
```

Допустим, сотрудница Иванова А.С. была переведена на новую должность. Выполним соответствующее изменение в таблице:

```
update WORKERS set "Должность" = 'ведущий специалист', "Оклад" = 60000  where "Таб №" = 104;
```

Далее будут выполнены изменения другого характера. Установим точку сохранения перед внесением следующих изменений:

```
savepoint "ПЕРЕВОД";
```

Так установленной точкой мы обозначили, что можно будет применить все изменения, выполненные после перевода сотрудника.

Далее продолжим вносить изменения — добавим данные о вновь принятых сотрудниках:

```
insert into WORKERS values (108, 'Белов И.И.', 55000, 'специалист 1 кат', 'Отдел кадров');
insert into WORKERS values (109, 'Буров Т.А.', 55000, 'специалист 1 кат', 'Отдел кадров');
```

Информация обо всех новых сотрудниках внесена и после этих изменений также нужно установить ещё одну точку сохранения для «отсечки» и возможности отмены только тех изменений, которые будут выполнены:

```
savepoint "ПРИЁМ";
```

Выполним запрос данных из таблицы на текущий момент:

```
select * from WORKERS;
```

Будет получена таблицы в следующем виде:

```
Таб №     | ФИО       | Оклад        | Должность        | Отдел
------------+-------------+----------------+--------------------+--------------
101       | Зощенко П.М.| 100000       | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000        | главный специалист | МТО
103       | Иванов Ф.Е. | 70000        | главный специалист | МТО
104       | Иванова А.С.| 60000        | ведущий специалист | МТО
105       | Петров Г.С. | 50000        | специалист 1 кат   | МТО
106       | Попова А.П. | 40000        | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000        | специалист 2 кат   | МТО
108       | Белов И.И.  | 55000        | специалист 1 кат   | Отдел кадров
109       | Буров Т.А.  | 55000        | специалист 1 кат   | Отдел кадров
```

После успешно выполненной команды можно продолжить выполнять изменения. Установленные точки сохранения позволят при необходимости отменить не все изменения, выполненные в транзакции, а только часть из них.

Допустим, внесение данных о новых сотрудниках было преждевременным и необходимо отменить выполненное добавление. Для этого достаточно выполнить отмену изменений до точки <ПЕРЕВОД>:

```
rollback to savepoint "ПЕРЕВОД";
```

При запросе данных на текущий момент будет получена таблица следующего вида:

```
Таб №     | ФИО       | Оклад        | Должность        | Отдел
------------+-------------+----------------+--------------------+--------------
101       | Зощенко П.М.| 100000       | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000        | главный специалист | МТО
103       | Иванов Ф.Е. | 70000        | главный специалист | МТО
104       | Иванова А.С.| 60000        | ведущий специалист | МТО
105       | Петров Г.С. | 50000        | специалист 1 кат   | МТО
106       | Попова А.П. | 40000        | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000        | специалист 2 кат   | МТО
```

Подробнее о применении команды ROLLBACK см. п. 12.8.12.4. ROLLBACK.

Далее можно продолжить вносить изменения или сохранить изменения, закончив транзакцию командой COMMIT (подробнее о команде см. п. 12.8.12.5. COMMIT).

Вернуться в оглавление

12.8.12.3.2. RELEASE SAVEPOINT

Команда удаления точки сохранения в текущей транзакции.

Выполнить удаление точки сохранения в текущей транзакции может любой пользователь, подсоединившийся к БД.

Синтаксис команды удаления точки сохранения:

```
RELEASE SAVEPOINT <имя_точки>;
```

Где <имя_точки> — имя точки сохранения, которую необходимо удалить. Напомним, что при удалении точки сохранения удаляются и все точки, установленные после неё.

Если в команде указано <имя_точки>, которой не существует, то будет возвращена ошибка.

Пример применения команды RELEASE SAVEPOINT

Ранее в п. 12.8.12.4.1. SAVEPOINT был рассмотрен пример с таблицей WORKERS с данными по оплате труда сотрудников компании.

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-----------+---------------+-------------------+-------------
101       | Зощенко П.М.| 100000        | Начальник отдела  | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
104       | Иванова А.С.| 50000         | специалист 1 кат  | МТО
105       | Петров Г.С. | 50000         | специалист 1 кат  | МТО
106       | Попова А.П. | 40000         | специалист 2 кат  | МТО
107       | Котов И.Р.  | 40000         | специалист 2 кат  | МТО
```

В данную таблицу были внесены изменения и в рамках текущей транзакции установлены точки сохранения <ПРИЁМ> и <ПЕРЕВОД>.

Допустим, далее были внесены ещё изменения — удаление данных уволенного сотрудника:

```
delete from WORKERS where "Таб №" = 107;
```

Установим ещё одну точку сохранения для возможности отмены выполненных далее изменений в рамках незаконченной транзакции:

```
savepoint "УДАЛЕНИЕ";
```

Допустим, что точка "УДАЛЕНИЕ" была поставлена преждевременно. В этом случае точку сохранения нужно удалить:

```
release savepoint "УДАЛЕНИЕ";
```

После успешно выполненной команды транзакция продолжается.

Вернуться в оглавление

12.8.12.4. ROLLBACK

Команда отмены изменений до начала транзакции или до точки сохранения, выполненных в текущей транзакции.

Выполнить отмену изменений до начала транзакции или до точки сохранения, выполненных в текущей транзакции, может любой пользователь, подсоединившийся к БД.

Синтаксис команды отмены изменений в текущей транзакции:

```
ROLLBACK [TO SAVEPOINT <имя_точки>];
```

Где:

– TO SAVEPOINT — необязательный параметр, используется для указания точки сохранения, до которой необходимо отменить все изменения;

– <имя_точки> — имя точки сохранения, до которой необходимо отменить все изменения в текущей транзакции. При этом точка сохранения, до которой производится отмена изменений, сохраняется и уничтожаются все точки сохранения, установленные после неё.

В рамках данной транзакции можно снова выполнить изменения и отменить их до данной точки сохранения.

При запуске команды ROLLBACK с указанием имени точки сохранения, которой не существует в текущей транзакции, будет возвращена ошибка и ROLLBACK не будет выполнен.

При запуске команды ROLLBACK без указания TO SAVEPOINT:

– будет выполнена отмена всех действий до начала транзакции;

– снимаются блокировки, установленные транзакцией;

– текущая транзакция завершается.

Напомним, что команду ROLLBACK нельзя выполнить для:

– команд DDL;

– команд DCL;

– команд работы с БД.

Пример применения команды ROLLBACK

Ранее в п.12.8.12.3.1. SAVEPOINT была рассмотрена таблица WORKERS с данными сотрудников компании:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-------------+----------------+-------------------+-------------
101       | Зощенко П.М.| 100000        | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
104       | Иванова А.С.| 50000         | специалист 1 кат   | МТО
105       | Петров Г.С. | 50000         | специалист 1 кат   | МТО
106       | Попова А.П. | 40000         | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000         | специалист 2 кат   | МТО
```

В данную таблицу были внесены изменения и в рамках текущей транзакции была установлены точки сохранения <ПРИЁМ> и <ПЕРЕВОД>.

После внесения изменений данные в таблице стали выглядеть следующим образом:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-------------+----------------+-------------------+-------------
101       | Зощенко П.М.| 100000        | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
104       | Иванова А.С.| 60000         | ведущий специалист | МТО
105       | Петров Г.С. | 50000         | специалист 1 кат   | МТО
106       | Попова А.П. | 40000         | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000         | специалист 2 кат   | МТО
108       | Белов И.И.  | 55000         | специалист 1 кат   | Отдел кадров
109       | Буров Т.А.  | 55000         | специалист 1 кат   | Отдел кадров
```

Считаем, что транзакция ещё не завершена и нужно выполнить новые изменения. Незавершённая транзакция также позволит при необходимости отменить часть или все изменения, выполненные командами DML в её рамках.

Удалим из таблицы данные об уволенном сотруднике:

```
delete from WORKERS where "Таб №" = 107;
```

Допустим, стало понятно, что изменения внесены неверно и данные сотрудника Котова И.Р. должны остаться в таблице.

Для восстановления данных выполним отмену изменений до установленной ранее точки сохранения <ПРИЁМ>:

```
rollback to savepoint "ПРИЁМ";
```

Выполним запрос данных по табельному номеру 107:

```
select * from WORKERS where "Таб №" = 107;
```

Будут получены восстановленные данные:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
------------+-------------+-----------------+--------------------+--------------
107       | Котов И.Р.  | 40000         | специалист 2 кат   | МТО
```

Если вместо предыдущего запроса выполнить ROLLBACK до точки сохранения PEREVOD, то кроме отмены удаления данных, будут отменены и добавления данных по новым сотрудникам. Тогда в транзакции останется выполненное изменение, связанное с переводом сотрудника, и таблица примет следующий вид:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
------------+-------------+-----------------+--------------------+--------------
101       | Зощенко П.М.| 100000        | Начальник отдела   | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
104       | Иванова А.С.| 60000         | ведущий специалист | МТО
105       | Петров Г.С. | 50000         | специалист 1 кат   | МТО
106       | Попова А.П. | 40000         | специалист 2 кат   | МТО
107       | Котов И.Р.  | 40000         | специалист 2 кат   | МТО
```

Если необходимо отменить все изменения, выполненные с начала транзакции, то указываем ROLLBACK без аргументов:

```
rollback;
```

После успешно выполненной команды данные в таблице приводятся к состоянию, в котором они были до начала транзакции.

Вернуться в оглавление

12.8.12.5. COMMIT

Команда фиксации изменений и завершения текущей транзакции.

Выполнить фиксацию изменений и завершение текущей транзакции может любой пользователь, подсоединившийся к БД.

Синтаксис команды фиксации изменений в текущей транзакции:

```
COMMIT;
```

При выполнении команды COMMIT фиксируются все изменения от начала текущей транзакции.

После выполнении команды COMMIT:

– выполненные изменения становятся постоянными без возможности отмены командой ROLLBACK;

– уничтожаются все точки сохранения в транзакции;

– снимаются блокировки, установленные транзакцией.

После успешного выполнения команды COMMIT можно выполнить команду установки уровня изоляции транзакции или с первым оператором SQL начать новую транзакцию.

Пример применения команды COMMIT

Ранее в п. 12.8.12.4.1. SAVEPOINT была рассмотрена таблица WORKERS с данными по оплате труда сотрудников компании:

```
Таб №     | ФИО       | Оклад         | Должность         | Отдел
----------+-----------+---------------+-------------------+-------------
101       | Зощенко П.М.| 100000        | Начальник отдела  | МТО
102       | Серов Ю.С.  | 70000         | главный специалист | МТО
103       | Иванов Ф.Е. | 70000         | главный специалист | МТО
```

```
104     | Иванова А.С.| 50000        | специалист 1 кат   | МТО
105     | Петров Г.С. | 50000        | специалист 1 кат   | МТО
106     | Попова А.П. | 40000        | специалист 2 кат   | МТО
107     | Котов И.Р.  | 40000        | специалист 2 кат   | МТО
```

В данную таблицу были внесены изменения, и в рамках текущей транзакции были установлены точки сохранения <ПРИЁМ> и <ПЕРЕВОД>.

После внесения изменений данные в таблице стали выглядеть следующим образом:

```
Таб №    | ФИО       | Оклад         | Должность        | Отдел
-----------+------------+----------------+-------------------+-------------
101     | Зощенко П.М.| 100000       | Начальник отдела   | МТО
102     | Серов Ю.С.  | 70000        | главный специалист | МТО
103     | Иванов Ф.Е. | 70000        | главный специалист | МТО
104     | Иванова А.С.| 60000        | ведущий специалист | МТО
105     | Петров Г.С. | 50000        | специалист 1 кат   | МТО
106     | Попова А.П. | 40000        | специалист 2 кат   | МТО
107     | Котов И.Р.  | 40000        | специалист 2 кат   | МТО
108     | Белов И.И.  | 55000        | специалист 1 кат   | Отдел кадров
109     | Буров Т.А.  | 55000        | специалист 1 кат   | Отдел кадров
```

Было принято решение, что все изменения внесены верно и больше изменений вносить не планируется. Для сохранения внесённых изменений необходимо завершить транзакцию:

```
commit;
```

После успешно выполненной команды невозможно воспользоваться точками сохранения, созданными внутри завершенной транзакции.

Покажем это попыткой выполнить команду ROLLBACK до точки сохранения PEREVOD:

```
rollback to savepoint "ПЕРЕВОД";
```

Будет возвращена ошибка, т.к. при завершении транзакции удаляются все точки сохранения.

Вернуться в оглавление

12.8.13. GRANT

Команда используется для предоставления привилегий и ролей пользователям и другим ролям.

Подробнее о ролях и привилегиях см. п. 12.5.7. Привилегии (PRIV) и роли (ROLE).

12.8.13.1. GRANT PRIV

Команда предоставления привилегий пользователю или роли.

Для команды GRANT PRIV используется два варианта синтаксиса: для предоставления системных привилегий и объектных привилегий.

Вернуться в оглавление

12.8.13.1.1. GRANT PRIV для предоставления системных привилегий

Выполнить предоставление системных привилегий может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды предоставления системных привилегий:

```
GRANT <системная_привилегия>[,<системная_привилегия>...]
                    TO <получатель>[,<получатель>...];


<получатель> ::= <имя_пользователя> | <имя_роли>
```

Где:

– <системная_привилегия> — наименование предоставляемой системной привилегии;

– <имя_пользователя> — имя пользователя, которому предоставляются системные привилегии;

– <имя_роли> — имя роли, которой предоставляются системные привилегии.

Выполнение действий, обозначенных предоставленными системными привилегиями, станет доступно пользователю в новой сессии (после его переподключения к базе данных).

Пример использования команды GRANT PRIV для предоставления системных привилегий

Для контроля доступа к базе данных и возможности выполнения определенного действия на уровне БД (как с самой базой данных, так и с её объектами) пользователю нужно предоставить системные привилегии.

Например, на должность ведущего специалиста отдела кадров принят новый сотрудник Буров А.В. Его руководитель подтвердил администратору БД заявкой, что на период стажировки сотруднику необходимо предоставить только возможность подсоединения к базе данных.

В таком случае администратор БД сначала создаёт нового пользователя с паролем:

```
create user "Буров А.В." IDENTIFIED BY 'freSHdock';
```

И добавляет пользователю системную привилегию CREATE SESSION:

```
grant CREATE SESSION to "Буров А.В.";
```

Наличие пароля и привилегии CREATE SESSION позволят пользователю пройти аутентификацию и подсоединиться к базе данных, но без возможности работать с объектами БД. Далее пользователю могут быть назначены права, например, на учебные материалы в отдельных учебных схемах.

После окончания периода стажировки руководителем будет принято дополнительное решение о необходимом наборе привилегий для работы сотрудника.

Для осуществления контроля доступа к БД, объектам БД и администрирования привилегий в компании принято решение для каждой должности создавать отдельную роль.

Сотруднику Бурову А.В. для работы необходимы как системные привилегии, так и объектные. В таком случае, в соответствии с решением компании, необходимо

создать роль с последующим предоставлением ей необходимых системных и объектных привилегий и назначить её Бурову А.В.

Создадим новую роль:

```
create role "Ведущий специалист ОК";
```

Добавим в новую роль системные привилегии, позволяющие пользователю выполнять подключение к БД, создавать и удалять объекты в собственной схеме:

```
grant CREATE SESSION, RESOURCES CONTROL to "Ведущий специалист ОК";
```

Далее сотруднику Бурову А.В. предоставляем роль:

```
grant "Ведущий специалист ОК" to "Буров А.В.";
```

Вернуться в оглавление

12.8.13.1.2. GRANT PRIV для предоставление объектных привилегий

Выполнить предоставление объектных привилегий может:

– пользователь базы данных с системной привилегией DATABASE ADMIN;

– владелец объекта.

Синтаксис команды предоставления объектных привилегий

```
GRANT <привилегии> ON <имя_объекта> TO <получатель>[, <получатель>...];

<привилегии>::= <объектная_привилегия>[, <объектная_привилегия>...] | ALL [PRIVILEGES]
<получатель> ::= <имя_пользователя> | <имя_роли>
```

Где:

– <объектная_привилегия> — наименование предоставляемой пользователю объектной привилегии;

– ALL [PRIVILEGES] — указывает на предоставление всех объектных привилегий, где ALL — краткий вариант, ALL PRIVILEGES — полный вариант конструкции;

– <имя_объекта> — имя объекта, на который предоставляются объектные привилегии;

– <имя_пользователя> — имя пользователя, которому предоставляются объектные привилегии;

– <имя_роли> — имя роли, которой предоставляются объектные привилегии.

Пример использования команды GRANT PRIV для предоставления объектных привилегий

Предоставление объектных привилегий возможно как администратором БД на любой объект базы данных, так и владельцем объекта.

Допустим, руководитель отдела создал в своей схеме таблицу "Оперативные данные". Для совместной работы он может предоставить объектные привилегии на данную таблицу другим сотрудникам, например, Иванову А.Р.:

```
grant SELECT, INDEX on "Оперативные данные" to "Иванов А.Р.";
```

После предоставленных привилегий Иванов А.Р. может выполнять выборку данных из таблицы "Оперативные данные" или строить индексы.

Вернуться в оглавление

12.8.13.2. GRANT ROLE

Команда предоставления роли пользователю или другой роли.

Выполнить предоставление роли может пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды предоставления роли:

```
GRANT <роль> TO <получатель>[, <получатель>...];

<получатель> ::= <имя_пользователя> | <имя_роли>
```

Где:

– <роль> - имя предоставляемой роли;

– <имя_пользователя> — имя пользователя, которому предоставляется роль;

– <имя_роли> — имя роли, которой предоставляется другая роль.

Пример применения команды GRANT ROLE

Для работы с объектами и данными базы данных сотрудникам кроме возможности подключения к базе данных нужны привилегии, соответствующие специфике работы.

Ранее в примере п. 12.8.1.7. CREATE ROLE были созданы две роли для разных должностей (младший администратор, менеджер). Каждая роль со своим набором привилегий.

Допустим, в компанию был принят новый сотрудник Дроздов А.П. на должность менеджера.

По окончании стажировки его руководитель заявкой администратору БД подтвердил необходимость предоставления сотруднику соответствующей роли. Администратор БД предоставляет сотруднику роль:

```
grant "Менеджер" to "Дроздов А.П.";
```

Допустим, был принят новый сотрудник Гришин С.Е. на должность старшего менеджера.

Руководителем определены привилегии для данной должности, но отдельной роли нет. Для удобства администрирования привилегий и контроля доступа к объектам БД (в данном случае к таблице "Продажи") нужно создать роль "Старший менеджер".

Роль "Старший менеджер" должна включать в себя привилегии роли "Менеджер" и дополнительно объектные привилегии DELETE, UPDATE для таблицы "Продажи". В таком случае новой роли нужно предоставить роль "Менеджер" и две дополнительные привилегии.

Создадим новую роль:

```
create role "Старший менеджер";
```

Предоставим новой роли роль "Менеджер":

```
grant "Менеджер" to "Старший менеджер";
```

Предоставим новой роли дополнительные объектные привилегии:

```
grant INSERT, UPDATE on "Продажи" to "Старший менеджер";
```

После подтверждения руководителем необходимости предоставления сотруднику роли администратор БД предоставляет сотруднику сформированную роль:

```
grant "Старший менеджер" to "Гришин С.Е.";
```

Вернуться в оглавление

12.8.14. REVOKE

Команда REVOKE используется для отзыва:

– системных ролей у пользователей и других ролей;

– системных привилегий у пользователей и ролей;

– объектных привилегий у пользователей и ролей.

Подробнее о ролях и привилегиях см. п. 12.5.7. Привилегии (PRIV) и роли (ROLE).

Вернуться в оглавление

12.8.14.1. REVOKE PRIV

Команда отзыва предоставленных системных и объектных привилегий у пользователей и ролей.

Для команды REVOKE PRIV используется два варианта синтаксиса: для отзыва системных привилегий и отзыва объектных привилегий.

Вернуться в оглавление

12.8.14.1.1. REVOKE PRIV для отзыва системных привилегий

Выполнить отзыв системных привилегий может только пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды отзыва системной привилегии:

```
REVOKE <системная_привилегия>[, <системная_привилегия>...] FROM <получатель>[, <получатель>...];

<получатель> ::= <имя_пользователя> | <имя_роли>
```

Где:

– <системная_привилегия> — наименование системной привилегии, которую необходимо отозвать;

– <имя_пользователя> — имя пользователя, у которого должна быть отозвана системная привилегия;

– <имя_роли> — имя роли, у которой должна быть отозвана системная привилегия.

Выполнение действий, обозначенных отозванными системными привилегиями, станет недоступно пользователю в новой сессии (после его переподключения к базе данных).

Пример применения команды REVOKE PRIV - для отзыва системных привилегий

Рассмотрим вариант отзыва системной привилегии у пользователя.

Для подсоединения к БД сотрудника Бурова А.В. был создан пользователь "Буров А.В." с паролем и ему была предоставлена системная привилегия CREATE SESSION (см. пример для п. 12.8.13.1.1 GRANT PRIV для предоставления системных привилегий).

Сотрудник ушёл в длительный отпуск и, по регламенту компании, на этот период должен быть лишён возможности подсоединяться к БД. Для этого администратор выполняет отзыв привилегии CREATE SESSION:

```
revoke CREATE SESSION from "Буров А.В.";
```

При попытке подсоединения под пользователем "Буров А.В." к БД будет возвращена ошибка, т.к. пользователь не может подключиться к БД без системной привилегии CREATE SESSION.

Теперь рассмотрим вариант отзыва системной привилегии у роли.

Допустим, для отдела кадров была создана роль "Ведущий специалист ОК". Данной роли были предоставлены системные привилегии, позволяющие выполнять подключение к БД, создавать и удалять объекты в собственной схеме:

```
create role "Ведущий специалист ОК";

grant CREATE SESSION, RESOURCES CONTROL to "Ведущий специалист ОК";
```

При реорганизации в отдел кадров была введена должность "Главный специалист ОК" и создана одноимённая роль. При перераспределении обязанностей было решено, что системная привилегия RESOURCES CONTROL должна быть предоставлена только роли "Главный специалист ОК".

В таком случае, у роли "Ведущий специалист ОК" необходимо данную роль отозвать:

```
revoke RESOURCES CONTROL from "Ведущий специалист ОК";
```

Теперь при попытке пользователя, которому предоставлена роль "Ведущий специалист ОК", выполнить создание или удаление объектов БД будет возвращена ошибка.

Вернуться в оглавление

12.8.14.1.2. REVOKE PRIV для отзыва объектных привилегий

Выполнить отзыв объектных привилегий может:

– пользователь базы данных с системной привилегией DATABASE ADMIN;

– владелец объекта.

Синтаксис команды отзыва объектной привилегии:

```
REVOKE <объектные_привилегии> ON <имя_объекта> FROM <получатель>[, <получатель>...];

<объектные_привилегии> ::= <объектная_привилегия>[, <объектная_привилегия>...]
```

```
<получатель> ::= <имя_пользователя> | <имя_роли>
```

Где:

– <объектная_привилегия> — наименование объектной привилегии, которую необходимо отозвать;

– <имя_объекта> — имя объекта, для которого должна быть отозвана объектная привилегия;

– <имя_пользователя> — имя пользователя, у которого должна быть отозвана объектная привилегия;

– <имя_роли> — имя роли, у которой должна быть отозвана объектная привилегия.

Пример применения команды REVOKE PRIV - для отзыва объектных привилегий

Допустим, руководитель отдела создал в своей схеме таблицу "Оперативные данные". Для выполнения задания он предоставил объектные привилегии на данную таблицу сотруднику подразделения Иванову А.Р.:

```
grant SELECT, INDEX on "Оперативные данные" to "Иванов А.Р.";
```

После выполненной работы у сотрудника Иванова А.Р. нет необходимости в доступе к таблице "Оперативные данные". Отзыв объектной привилегии на данную таблицу у пользователя "Иванов А.Р." выполняется командой:

```
revoke SELECT, INDEX on "Оперативные данные" from "Иванов А.Р.";
```

Теперь при попытке пользователя "Иванов А.Р." выполнить запрос SELECT к таблице "Оперативные данные" будет возвращена ошибка.

Вернуться в оглавление

12.8.14.2. REVOKE ROLE

Команда отзыва предоставленной роли у пользователей и других ролей.

Выполнить отзыв предоставленной роли у пользователей и других ролей может только пользователь базы данных с системной привилегией DATABASE ADMIN.

Синтаксис команды отзыва предоставленной роли:

```
REVOKE <роль> FROM <получатель>[,<получатель>...];

<получатель> ::= <имя_пользователя> | <имя_роли>
```

Где:

– <роль> — имя роли, которую необходимо отозвать;

– <имя_пользователя> — имя пользователя, у которого необходимо отозвать роль;

– <имя_роли> — имя роли, у которой необходимо отозвать роль.

Пример применения команды REVOKE ROLE

Рассмотрим сначала пример отзыва роли у пользователя.

Ранее сотруднику Гришину С.Е. была предоставлена роль "Старший менеджер" (подробнее см. в примере п. 12.8.13.2 GRANT ROLE), которая в том числе включает в себя объектные привилегии на таблицу "Продажи". Принято решение о переводе Гришина С.Е. на другую должность со сменой обязанностей.

В компании для удобства администрирования привилегий для каждой должности создана своя роль. После подтверждения руководителем перевода Гришина С.Е. администратор БД выполняет отмену ранее предоставленной роли "Старший менеджер":

```
revoke "Старший менеджер" from "Гришин С.Е.";
```

После успешного выполнения команды сотрудник больше не имеет объектных привилегий к таблице "Продажи".

При попытке сотрудника выполнить какой-либо запрос к таблице "Продажи", например:

```
select * from "Отдел_продаж"."Продажи" where DATE = '15.02.2022';
```

будет возвращена ошибка, т.к. у сотрудника нет привилегий на доступ к данной таблице.

Далее администратор предоставляет сотруднику другую, необходимую для работы на новой должности, роль:

```
grant "Ведущий специалист АХО" to "Гришин С.Е.";
```

Теперь рассмотрим пример отзыва роли у роли.

Роли "Старший менеджер" была предоставлена роль "Менеджер" и дополнительно объектные привилегии, не включённые в роль "Менеджер" (подробнее см. в примере п. 12.8.13.2 GRANT ROLE).

Было принято решение об изменении обязанностей должности старшего менеджера и набора предоставленных привилегий соответствующей роли. Предоставленные ранее привилегии роли "Старший менеджер", в виде роли "Менеджер", не нужны.

После подтверждения руководителем необходимости изменения привилегий администратор БД выполняет отмену ранее предоставленной роли:

```
revoke "Менеджер" from "Старший менеджер";
```

Далее администратор БД назначает роли "Старший менеджер" необходимые привилегии или другую роль.

Вернуться в оглавление

12.8.15. SET SCHEMA

Команда установки текущей схемы пользователя в рамках текущего соединения с базой данных.

Текущая схема устанавливается на период соединения, и в следующем соединении текущей будет схема по умолчанию.

Выполнить установку текущей схемы пользователя в рамках текущего соединения с базой данных может любой пользователь, но только для тех схем, владельцем которых он является.

Синтаксис команды установки текущей схемы:

```
SET SCHEMA <имя_схемы>;
```

Где:

– <имя_схемы> — наименование схемы, которую необходимо установить.

После установки текущей схемы к её объектам можно обращаться без указания имени схемы, а к объектам других схем — обязательно с указанием имени схемы (подробнее о правилах именования объектов БД и обращения к ним см. п. 12.5.10. Правила именования объектов БД и обращения к ним).

Пример установки текущей схемы

Рассмотрим примеры установки текущей схемы.

Допустим, руководитель управления персоналом является владельцем двух схем "Отдел кадров" и "Отдел мотивации", в каждой из которых есть таблица "Сотрудники компании" (подробнее о создании схем и таблиц в каждой из них см. пример п. 12.8.1.5. CREATE SCHEMA).

В данный момент он много работает с объектами, находящимися не в его текущей схеме — "Отдел кадров". И, например, при обращении к таблице "Сотрудники компании" каждый раз необходимо указывать полное имя объекта:

```
select * from "Отдел кадров"."Сотрудники компании";
```

Для того, чтобы при обращении к объектам в другой схеме не указывать каждый раз имя схемы, достаточно сделать эту схему текущей:

```
set schema "Отдел кадров";
```

Теперь схема "Отдел кадров" на весь период соединения с базой данных будет текущей.

Для примера создадим в этой схеме новую таблицу и выполним к ней запрос, указав краткое имя объекта:

```
create table "Ежегодный отпуск"
  ("Таб №" int primary key, "ФИО" varchar (150), "Подразделение" varchar (350), "Дата" date);
```

Выполним запрос к таблице без указания имени схемы:

```
select * from "Ежегодный отпуск";
```

В ответ будет получена таблица с данными.

Теперь отсоединимся от базы данных и подсоединимся вновь. После подсоединения попробуем выполнить тот же запрос выборки данных из таблицы "Ежегодный отпуск".

На данный запрос будет возвращена ошибка, т.к. после соединения текущей схемой пользователя стала его схема по умолчанию, а таблица "Ежегодный отпуск" находится в другой схеме — "Отдел кадров". И в текущей ситуации, в запросе к таблице "Ежегодный отпуск" нужно указывать её полное имя:

```
select * from "Отдел кадров"."Ежегодный отпуск";
```

Вернуться в оглавление

12.8.16. PRESS[CV]

Команды из группы PRESS используются для выполнения ручной очистки одной или всех таблиц базы данных (кроме временных таблиц и системных объектов) от устаревших MVCC-записей (подробнее о MVCC-записях см. п. 1.1. Термины, акронимы, сокращения).

Очистка физического хранилища табличных данных от устаревших MVCC-записей позволяет уменьшить объем используемого дискового пространства и повысить эффективность доступа к таблице.

Выполняемые команды накладывают на указанные таблицы и схемы блокировки, предотвращающие их удаления. Данные запросы выполняют очистку таблиц от устаревших MVCC-записей, поэтому команды рекомендуется выполнять в часы наименьшей нагрузки на базу данных.

Команды из группы PRESS рекомендуется выполнять при отключенной опции фоновой очистки страниц для данной базы (подробнее об опции см. п. 11.1. CREATE DATABASE).

Вернуться в оглавление

12.8.16.1. PRESS TABLE

Команда для выполнения ручной очистки таблицы от устаревших MVCC-записей.

Выполнить команду может:

– владелец таблицы с привилегией RESOURCE;

– любой пользователь БД с привилегией RESOURCE для таблицы в схеме PUBLIC;

– пользователь с системной привилегией DATABASE ADMIN для любой существующей таблицы БД.

Синтаксис команды ручной очистки таблицы:

```
PRESS TABLE <имя_таблицы>;
```

Где <имя_таблицы> — имя таблицы, очистку от устаревших MVCC-записей которой необходимо выполнить.

Вернуться в оглавление

12.8.16.2. PRESS ALL TABLES

Команда для выполнения ручной очистки всех таблиц базы данных от устаревших MVCC-записей.

Выполнить очистку всех таблиц БД может только пользователь с системной привилегией DATABASE ADMIN.

Синтаксис команды ручной очистки всех таблиц БД:

```
PRESS ALL TABLES;
```

Вернуться в оглавление

12.8.17. ANALYZE TABLE[CV]

Команда выполняет обновление статистики таблицы базы данных, результаты которой сохраняются в системной таблице. Впоследствии планировщик запросов использует эту статистику для определения наиболее эффективных планов

выполнения запросов (подробнее о получении плана выполнения запросов, построенного планировщиком см. п. 12.7. План выполнения запроса (EXPLAIN)).

Выполнить команду ANALYZE TABLE может:

– владелец таблицы с привилегией RESOURCE;

– пользователь с системной привилегией DATABASE ADMIN для любой существующей таблицы БД.

Синтаксис команды обновления статистики:

```
ANALYZE TABLE <имя_таблицы>;
```

Где <имя_таблицы> — имя таблицы, обновление статистики которой необходимо выполнить.

Выполнение команды запрещено:

– для временных таблиц;

– для системных таблиц.

При попытке обновить статистику для указанных объектов выполнение команды будет завершено с ошибкой.

В СУБД предусмотрено автоматическое обновление статистики по каждой таблице БД, которое происходит с периодичностью каждые 10 минут.

В ручном режиме (командой ANALYZE TABLE) полезно обновлять статистику в случае отключенного автоматического сбора статистики (подробнее см. опции БД в п. 11.1. CREATE DATABASE).

Вернуться в оглавление

12.8.18. WITH (Common Table Expressions)[CV]

Common Table Expressions (CTE) — это именованный временный результирующий набор, который существует в рамках одного SQL-оператора и на который можно ссылаться в рамках этого SQL-оператора, возможно, несколько раз.

Для создания временных результирующих наборов используется оператор WITH, за которым следуют характеристики создаваемого CTE.

Оператор WITH позволяет создавать обычные CTE и рекурсивные.

Вернуться в оглавление

12.9. Функции SQL

Функции SQL встроены в СУБД СОКОЛ и доступны для использования в различных инструкциях SQL.

Встроенные функции SQL делятся на две группы:

– агрегатные;

– скалярные.

Далее рассмотрим более подробно каждую функцию.

Вернуться в оглавление

12.9.1. Агрегатные функции

Агрегатные функции оперируют набором значений, соответствующих группе строк выборки, а не одним значением. На основании набора значений агрегатные функции вычисляют единственный результат.

Каждое из значений набора вычисляется как значение заданного выражения, посчитанного для отдельной строки. Указание скалярного выражения в качестве аргумента агрегатной функции допустимо, а указание агрегатных функций — не допускается.

Формирование групп строк:

1) При выборке значений без конструкции GROUP BY формируется одна группа строк, для которой агрегатная функция вычислит одно значение.

2) При выборке значений с использованием конструкции GROUP BY будет сформировано несколько групп, для каждой из которых агрегатная функция вычислит отдельное значение.

При вычислении агрегатные функции (за исключением COUNT (*)) не учитывают значения NULL.

Вернуться в оглавление

12.9.1.1. AVG

Функция подсчитывает среднее арифметическое заданных значений аргумента, выполняя деление суммы его значений на число значений, не равных NULL.

Синтаксис функции

```
AVG ([ ALL | DISTINCT ] <выражение>)
```

Аргументы:

– <выражение> — выражение, которое необходимо вычислить для каждой строки в группе, с последующим вычислением среднего арифметического значения группы. Значения <выражения> должны быть любого числового типа данных, поддерживаемого СУБД СОКОЛ.

Допустимы также значения типов CHAR, VARCHAR, которые функция попытается неявно преобразовать в значение типа NUMBER максимальной точности (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция AVG всегда возвращает тип данных NUMBER максимальной точности.

При использовании ключевого слова ALL будет возвращено среднее значение с учётом дубликатов, но без учёта строк со значением NULL. ALL используется по умолчанию.

При использовании ключевого слова DISTINCT будет возвращено среднее значение без учёта строк с дублирующими значениями и строк со значениями NULL.

Пример применения функции AVG

Допустим, необходимо вычислить средний оклад сотрудников в каждом подразделении и в компании в целом.

Для примера возьмём таблицу WORKERS_VORONEZH с данными всех работников компании:

```
Таб №  | Имя  | Фамилия  | Оклад  | Подразделение
---------+-------+-----------+----------+----------------
212    | Иван | Петров   | 53000  | МТО
213    | Пётр | Иванов   | 27000  | МТО
214    | Игорь | Котов    | 38000  | Отдел кадров
215    | Алёна | Котова   | 70000  | ПТО
216    | Юлия | Серова   | 65000  | ФЭУ
217    | Елена | Попова   | 49000  | ФЭУ
```

Выполним запрос с указанием столбца, среднее значение по которому необходимо высчитать, и группировкой значений по подразделениям:

```
select "Подразделение", avg ("Оклад") as "Средний оклад"
   FROM WORKERS_VORONEZH group by "Подразделение";
```

После успешно выполненного запроса возвращается результат:

```
Подразделение | Средний оклад
--------------+----------------
МТО          | 40000
Отдел кадров | 38000
ПТО          | 70000
ФЭУ          | 57000
```

Теперь выполним запрос для вычисления среднего значения оклада по сотрудникам всей компании:

```
select avg ("Оклад") as "Средний оклад" FROM WORKERS_VORONEZH;
```

После успешно выполненного запроса возвращается соответствующий результат:

```
Средний оклад
----------------------
50333.33333333333333333333333333333333
```

Вернуться в оглавление

12.9.1.2. COUNT

Функция выполняет подсчёт числа строк, возвращаемых запросом, или количества не NULL-значений заданного выражения для этих строк.

Синтаксис функции

```
COUNT ([ * | ALL | DISTINCT ] <выражение> )
```

Аргументы:

– <выражение> — выражение, которое необходимо вычислить для каждой строки в группе. Далее функция подсчитывает количество значений выражения, не равных NULL.

Значения <выражения> могут быть любого типа данных, реализованного в СУБД СОКОЛ.

Возвращаемое значение

Функция возвращает число строк в группе. Тип данных результата — NUMBER.

Если указать COUNT(*), то функция вернёт общее число строк таблицы, с учётом строк-дубликатов.

При использовании ключевых слов ALL и DISTINCT не подсчитываются строки, для которых значение <выражения> равно NULL.

При использовании ключевого слова ALL подсчитывается число строк с учётом дублирующихся значений, полученных при вычислении <выражения>.

При использовании ключевого слова DISTINCT подсчитывается число строк с уникальными значениями, полученными при вычислении <выражения>.

Пример применения функции COUNT

Допустим есть таблица с данными всех абонентов, зарегистрированных в разных городах. Необходимо получить информацию о том, сколько городов охватывает компания и сколько зарегистрировано абонентов в каждом городе.

Для примера была создана и заполнена таблица PHONE:

```
 NUMBER     | NAME    | AGE  | CITY
-------------+---------+------+-----------
 89876543212 | Петров  | 53   | Воронеж
 89876543214 | Иванов  | 46   | Москва
 89876543244 | Котов   | 37   | Москва
 89876543215 | Попова  | 26   | Воронеж
 89876543266 | Лебедева| 37   | Москва
 89876543277 | Ермаков | 38   | Химки
 89876543288 | Тихонова| 29   | Москва
 89876543299 | Ермакова| 45   | Химки
 89876543300 | Лебедев | 37   | Москва
```

Выполним запрос для получения информации о том, сколько городов охватывает компания. Для исключения дублирующихся значений укажем ключевое слово DISTINCT:

```
select count(DISTINCT CITY) from PHONE;
```

После успешно выполненного запроса возвращается результат:

```
COUNT(CITY)
-----------
3
```

Теперь выполним запрос числа зарегистрированных абонентов в каждом городе. Для этого воспользуемся группировкой значений по городам:

```
select CITY, count(*) as SUM_PHONE from PHONE group by CITY;
```

После успешно выполненного запроса возвращается результат с разбивкой по городам:

```
CITY     | SUM_PHONE
----------+-----------
Воронеж  | 2
Москва   | 5
Химки    | 2
```

Вернуться в оглавление

12.9.1.3. MAX

Функция выполняет поиск максимального значения среди значений аргумента.

Синтаксис функции

```
MAX ([ ALL | DISTINCT ] <выражение>)
```

Аргументы:

– <выражение> — выражение, которое необходимо вычислить для каждой строки в группе. Далее функция находит среди полученных значений максимальное.

Значения <выражения> могут быть любого типа данных (кроме BLOB и CLOB), реализованного в СУБД СОКОЛ.

Возвращаемое значение

Функция возвращает максимальное значение в группе. Тип данных результата совпадает с типом данных аргумента.

Если все значения аргументов равны NULL, то функция вернёт значение NULL.

Использование ключевых слов ALL и DISTINCT в функции MAX не влияет на конечный результат.

Пример применения функции MAX

Допустим, необходимо получить максимальный оклад сотрудников компании в целом и в каждом подразделении по отдельности.

Для примера возьмём таблицу WORKERS_VORONEZH с данными сотрудников компании:

```
 Таб №   | Имя   | Фамилия   | Оклад    | Подразделение
---------+-------+-----------+----------+----------------
 212     | Иван  | Петров    | 53000    | МТО
 213     | Пётр  | Иванов    | 27000    | МТО
 214     | Игорь | Котов     | 38000    | Отдел кадров
 215     | Алёна | Котова    | 70000    | ПТО
 216     | Юлия  | Серова    | 65000    | ФЭУ
 217     | Елена | Попова    | 49000    | ФЭУ
```

Выполним запрос с указанием столбца, максимальное значение по которому необходимо найти:

```
select max("Оклад") as "Максимальный оклад" FROM WORKERS_VORONEZH;
```

После успешно выполненного запроса возвращается максимальный оклад сотрудников компании:

```
 Максимальный оклад
---------------------
 70000
```

Для получения максимального оклада сотрудника в каждом подразделении выполним запрос с группировкой значений по подразделениям:

```
select "Подразделение", max("Оклад") as "Максимальный оклад"
    FROM WORKERS_VORONEZH group by "Подразделение";
```

Команда вернет следующий результат:

```
Подразделение | Максимальный оклад
--------------+-------------------
 МТО          | 53000
 Отдел кадров | 38000
 ПТО          | 70000
 ФЭУ          | 65000
```

Вернуться в оглавление

12.9.1.4. MIN

Функция выполняет поиск минимального значения среди значений аргумента.

Синтаксис функции

```
MIN ([ ALL | DISTINCT ] <выражение>)
```

Аргументы:

– <выражение> — выражение, которое необходимо вычислить для каждой строки в группе. Далее функция находит среди полученных значений минимальное.

Значения аргумента могут быть любого типа данных (кроме BLOB и CLOB), реализованного в СУБД СОКОЛ.

Возвращаемое значение

Функция возвращает минимальное значение в группе. Тип данных результата совпадает с типом данных аргумента.

Если все значения аргументов равны NULL, то функция вернёт значение NULL.

Использование ключевых слов ALL и DISTINCT в функции MIN не влияет на конечный результат.

Пример применения функции MIN

См. примеры для функции MAX 12.9.1.3 MAX.

Вернуться в оглавление

12.9.1.5. STRING_AGG

Функция выполняет конкатенацию значений аргумента с заданным разделителем между ними.

Синтаксис функции

```
STRING_AGG ([ ALL | DISTINCT ] <выражение>, <разделитель>)
```

Аргументы:

– <выражение> — выражение, которое необходимо вычислить для каждой строки в группе. Далее функция выполнит конкатенацию полученных значений, не равных NULL;

– <разделитель> — строковое выражение для разделения объединяемых значений аргумента <выражение>. Если в качестве значения <разделителя> будет значение NULL, то значения аргумента <выражение> будут конкатенированы без разделителя.

Значения аргументов могут быть только типов CHAR, VARCHAR, BINARY, VARBINARY. При этом значения аргументов <выражение> и <разделитель> должны оба иметь или строковый тип, или бинарный тип. Иначе работа функции будет завершена с ошибкой.

Возвращаемое значение

Функция возвращает строку с заданным разделителем между значениями <выражения>. Тип данных результата совпадает с типом данных первого аргумента, кроме случаев:

– если тип данных аргумента <выражение> CHAR, то тип результата — VARCHAR;

– если тип данных аргумента <выражение> BINARY, то тип результата — VARBINARY.

При использовании ключевого слова ALL при конкатенации будут задействованы все значения <выражения>, включая дублирующиеся. ALL используется по умолчанию.

При использовании ключевого слова DISTINCT при конкатенации будут задействованы только уникальные значения <выражения>.

Максимально допустимая длина результирующей строки равна максимальному размеру строкового значения. При превышении этого значения работа функции будет завершена с ошибкой.

Пример применения функции STRING_AGG

Допустим, есть таблица WORKERS с данными о сотрудниках:

```
Таб №      | Сотрудник     | Подразделение
-----------+---------------+------------------
 212       | Иван Петров   | МТО
 213       | Пётр Иванов   | МТО
 214       | Игорь Котов   | Отдел кадров
 215       | Алёна Котова  | ПТО
 216       | Юлия Серова   | ФЭУ
 217       | Елена Попова  | ФЭУ
```

Необходимо по каждому подразделению получить одной строкой список сотрудников, разделенных запятыми.

Выполним запрос с группированием значений по подразделениям:

```
select "Подразделение", string_agg (ALL "Сотрудник", ', ') as "Список сотрудников"
    from WORKERS group by "Подразделение";
```

После успешно выполненной команды возвращается результат:

```
Подразделение         | Список сотрудников
----------------------+----------------------------
 МТО                  | Иван Петров, Пётр Иванов
 Отдел кадров         | Игорь Котов
 ПТО                  | Алёна Котова
 ФЭУ                  | Юлия Серова, Елена Попова
```

Вернуться в оглавление

12.9.1.6. SUM

Функция выполняет суммирование числовых значений аргумента.

Синтаксис функции

```
SUM ([ ALL | DISTINCT ] <выражение>)
```

Аргументы:

– <выражение> — выражение, которое необходимо вычислить для каждой строки в группе. Далее функция выполнит суммирование полученных значений.

Значения аргумента должны быть любого числового типа данных, реализованного в СУБД СОКОЛ.

Допустимы также также значения типов CHAR, VARCHAR, которые функция попытается неявно преобразовать в значение типа NUMBER максимальной точности (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция возвращает сумму числовых значений аргумента. Тип данных результата — NUMBER максимальной точности.

При использовании ключевого слова ALL суммируются все вычисленные значения <выражения> с учетом дублирующихся значений. ALL используется по умолчанию.

При использовании ключевого слова DISTINCT суммируются все вычисленные уникальные значения <выражения>.

Вернуться в оглавление

12.9.2. Скалярные функции

Скалярные функции работают только с одним значением и возвращают для него один результат.

Аргументы скалярных функций могут быть разных типов (подробнее о реализованных в СУБД СОКОЛ типах данных см. 12.1. Типы данных).

Вернуться в оглавление

12.9.2.1. ABS

Функция вычисляет абсолютное значение аргумента.

Синтаксис функции

```
ABS (<x>)
```

Аргумент:

– <x> — величина любого числового типа данных, реализованного в СУБД СОКОЛ, для которой необходимо вычислить абсолютное значение.

Допустимы также значения типов CHAR, VARCHAR, которые функция попытается неявно преобразовать в значение типа NUMBER максимальной точности

(подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция ABS возвращает абсолютное значение заданного числа. Тип результата — NUMBER максимальной точности.

Вернуться в оглавление

12.9.2.2. BITAND

Функция выполняет операцию побитового умножения (аналог операции логического AND) целочисленных значений двух аргументов.

Синтаксис функции

```
BITAND (<x>, <y>)
```

Аргументы:

– <x>, <y> — значения числового типа данных NUMBER.

Допустимы также значения типов CHAR, VARCHAR, которые функция попытается неявно преобразовать в значение типа NUMBER максимальной точности (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция BITAND всегда возвращает целочисленное значение. Тип результата — NUMBER максимальной точности.

Если тип значения какого-либо аргумента отличен от NUMBER, то выполняется неявное преобразование к типу NUMBER. Далее усекается дробная часть и значение приводится к целочисленному виду в диапазоне от -2^{63} до $2^{63}-1$ (64 бита) в дополнительном коде. В случае переполнения происходит потеря старших разрядов без сообщения об ошибке. Над полученными 64-битными значениями выполняется побитовое умножение. Результат побитового умножения является результатом выполнения функции.

Если значение хотя бы одного из аргументов равно NULL, то возвращается NULL.

Вернуться в оглавление

12.9.2.3. CAST

Функция выполняет преобразование заданного значения к заданному типу данных.

Синтаксис функции

```
CAST (<x> AS <тип_данных>)
```

Аргументы:

– <x> — значение, которое необходимо преобразовать к указанному <типу_данных>;

– <тип_данных> — тип данных, к которому необходимо преобразовать значение <x>.

Допустимые преобразования типов данных с помощью функции указаны далее в таблице 21.

Таблица 21 — Преобразования типов данных с помощью функции CAST

ТИП ДАННЫХ итоговый → исходный ↓	CHAR	VARCHAR	CLOB	DATE	TIMESTAMP	BOOLEAN	NUMBER (38,0)	NUMBER	BLOB	BINARY	VARBINARY	ROWID
CHAR	V	V	V	V	V	V	V	V	V	V	V	V
VARCHAR	V	V	V	V	V	V	V	V	V	V	V	V
CLOB	V	V	V	-	-	V	-	-	-	-	-	-
DATE	V	V		V	V	-	-	-	-	-	-	-
TIMESTAMP	V	V		V	V	-	-	-	-	-	-	-
BOOLEAN	V	V	V	-	-	V	V	-	-	-	V	-
NUMBER (38,0)	V	V		-	-	V	V	V	-	-	-	-
NUMBER	V	V		-	-	V	V	V	-	-	-	-

ТИП ДАННЫХ итоговый → исходный ↓	CHAR	VARCHAR	CLOB	DATE	TIMESTAMP	BOOLEAN	NUMBER (38,0)	NUMBER	BLOB	BINARY	VARBINARY	ROWID
BLOB	V	V	V	-	-	-	-	-	V	-	-	-
BINARY	V	V	V	-	-	-	-	-	V	V	V	V
VARBINARY	V	V	V	-	-	-	-	-	V	V	V	V
ROWID	V	V	V	-	-	-	-	-	V	V	V	V
V	Явное преобразование допустимо											
-	Явное преобразование недопустимо											

Возвращаемое значение

Функция CAST возвращает преобразованное значение аргумента. Тип результата — указанный для преобразования.

Если значение аргумента <x> равно NULL, то функция вернёт NULL.

Вернуться в оглавление

12.9.2.4. CASE[cv]

CASE выполняет сравнение значения исходного выражения с каждым из значений последующих выражений в порядке их указания слева направо или, при отсутствии исходного выражения, проверяет заданные логические выражения на соответствие TRUE и возвращает результат в зависимости от полученного итога (подобно операторам IF-THEN-ELSE).

Синтаксис варианта 1:

```
CASE <выражение>
  WHEN <выражение1> THEN <результат1>
  [...WHEN <выражениеN> THEN <результатN>]
  [ELSE <значение_по_умолчанию>]
END
```

Аргументы:

1) <выражение> — выражение, значение которого сравнивается со значением каждого из указанных далее выражений (<выражение1>... <выражениеN>).

2) <выражение1>…<выражениеN> — выражения, со значениями которых сравнивается значение исходного <выражения>.

3) <результат1>…<результатN> — значения, одно из которых (<результатi>) будет возвращено в качестве результата оператора CASE при совпадении <выражения> с <выражениеmi>.

4) <значение_по_умолчанию> — значение, которое будет возвращено в качестве результата оператора CASE в случае, когда не найдено ни одного совпадения значения <выражения> со значениями <выражение1>…<выражениеN>.

В качестве значений аргументов <выражение> и <выражениеi> допустимы значения:

– строкового типа данных (кроме CLOB). Если значения аргументов разного строкового типа данных, то будет выполнена попытка неявного преобразования в тип данных VARCHAR;

– любого числового типа данных. Если значения аргументов разного числового типа данных, то будет выполнена попытка неявного преобразования в тип данных NUMBER;

– логического типа данных BOOLEAN.

Подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

При этом значения аргументов <выражение> и <выражениеi> должны быть одного типа данных, иначе должна быть возвращена ошибка.

При неудачной попытке преобразования работа функции будет завершена с возвращением ошибки.

В качестве значений аргументов <результатi> и <значение_по_умолчанию> допустимы значения любого типа данных, реализованного в СУБД.

Значения аргументов <результатi> и <значение_по_умолчанию> должны быть одного типа данных, иначе будет возвращена ошибка.

Для любого аргумента допустимо значение NULL.

Синтаксис варианта 2:

```
CASE
   WHEN <условие1> THEN <результат1>
   [...WHEN <условиеN> THEN <результатN>...]
  [ELSE <значение_по_умолчанию>]
END
```

Аргументы:

1) <условие1>…<условиеN> — логические выражения, результаты которых проверяются на равенство TRUE.

2) <результат1>…<результатN> — значения, одно из которых (<результатi>) будет возвращено в качестве результата оператора CASE, если <условиеi> истинно.

3) <значение_по_умолчанию> — значение, которое будет возвращено в качестве результата оператора CASE в случае, когда все <условияi> ложны.

Значения аргументов <условиеi> должны иметь тип данных BOOLEAN.

В качестве значений аргументов <результатi> и <значение_по_умолчанию> допустимы значения любого типа данных, но при этом все аргументы должны быть одинакового типа.

Для любого аргумента допустимо значение NULL.

Возвращаемое значение

CASE возвращает значение аргумента <результатi> или <значение_по_умолчанию>. Тип данных результата должен соответствовать типу данных значения возвращаемого аргумента.

Если конструкция ELSE <значение_по_умолчанию> опущена, то будет возвращено значение NULL.

Для варианта 1:

Когда CASE находит первое <выражениеi>, равное <выражению>, то возвращается значение <результатаi>.

Если исходное <выражение> имеет значение NULL, то оно не будет соответствовать ни одному из <выражениiі>, даже тем, которые имеют значение NULL.

Для варианта 2:

Когда CASE находит первое <условиеі>, результат которого соответствует TRUE, то возвращается значение <результатаі>.

Вернуться в оглавление

12.9.2.5. COALESCE

Функция возвращает значение первого аргумента в списке, отличного от NULL.

Синтаксис функции

```
COALESCE (<x1>, <x2> [, <x3>...])
```

Аргументы <x1>, <x2> [, <x3>...] — значения, среди которых нужно найти первое значение, отличное от NULL.

Необходимо указать минимум два аргумента.

Значения аргументов могут быть любого типа данных, поддерживаемого в СУБД СОКОЛ, но тип данных всех аргументов должен совпадать.

При этом значения следующих типов данных функция будет воспринимать как совпадающие:

– BINARY и VARBINARY;

– CHAR и VARCHAR;

– DATE и TIMESTAMP.

Возвращаемое значение

Функция COALESCE возвращает первое значение из списка, отличное от NULL. Тип данных результата совпадает с типом данных аргумента, значение которого возвращает функция.

Если типы данных аргументов не совпадают, то функция вернёт ошибку.

Если все выражения имеют значение NULL, то функция вернёт значение NULL.

Вернуться в оглавление

12.9.2.6. COLLATION

Функция определяет правило сравнения для значения аргумента.

Синтаксис функции

```
COLLATION (<x>)
```

Аргумент <x> — значение, для которого функция определит правило сравнения.

Тип данных аргумента — строковый тип.

Возвращаемое значение

Функция COLLATION возвращает наименование правила сравнения, применимое для строкового значения аргумента.

Если значение аргумента нестроковое, то функция возвращает наименование правила, установленное по умолчанию (для сессии, если не установлено для сессии, то для базы данных).

Тип результата — VARCHAR.

Вернуться в оглавление

12.9.2.7. CONCAT

Функция выполняет операцию конкатенации значений двух аргументов.

Синтаксис функции

```
CONCAT (<s1>, <s2>)
```

Аргументы <s1>, <s2> — строковые значения, конкатенацию в порядке следования которых выполнит функция.

Допустимы также также значения типов NUMBER, BOOLEAN, DATE, TIMESTAMP, BINARY, VARBINARY, ROWID, которые функция попытается неявно преобразовать в значение типа VARCHAR (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Для одной функции CONCAT допустимо указание только двух аргументов.

Для конкатенации более двух значений необходимо использовать вложенный вызов функции CONCAT.

Возвращаемое значение

Функция CONCAT возвращает конкатенацию значения двух аргументов. Тип данных результата — VARCHAR.

Если значение одного из аргументов равно NULL, то функция вернёт значение NULL.

Пример применения функции CONCAT

Допустим, необходимо получить список полных имён (имя+фамилия) всех сотрудников компании. Для примера возьмём таблицу WORKERS_VORONEZH с данными всех работников компании:

```
Таб №  | Имя   | Фамилия   | Оклад   | Подразделение
---------+-------+-----------+----------+---------------
212    | Иван  | Петров    | 53000   | МТО
213    | Пётр  | Иванов    | 27000   | МТО
214    | Игорь | Котов     | 38000   | Отдел кадров
215    | Алёна | Котова    | 70000   | ПТО
216    | Юлия  | Серова    | 65000   | ФЭУ
217    | Елена | Попова    | 49000   | ФЭУ
```

Выполним запрос с указанием столбцов, значения строк которых необходимо объединить:

```
select concat ("Имя", "Фамилия") as "Полное имя" from WORKERS_VORONEZH;
```

После успешно выполненного запроса возвращается соответствующий результат:

```
  Полное имя
---------------
ИванПетров
ПётрИванов
ИгорьКотов
АлёнаКотова
ЮлияСерова
ЕленаПопова
```

В результирующей таблице объединённые значения, но это неудобно для прочтения и использования.

Для добавления пробела между объединяемыми значениями используем вложение функции CONCAT:

```
select concat ("Имя", concat (' ', "Фамилия")) as "Полное имя" from WORKERS_VORONEZH;
```

После успешно выполненного запроса получаем читаемый список полных имён сотрудников компании:

```
  Полное имя
---------------
 Иван Петров
 Пётр Иванов
 Игорь Котов
 Алёна Котова
 Юлия Серова
 Елена Попова
```

Вернуться в оглавление

12.9.2.8. CONCAT_BLOB[CV]

Функция выполняет конкатенацию двух значений типа BLOB.

Синтаксис функции

```
CONCAT_BLOB (<s1>, <s2>)
```

Аргументы:

– <s1>, <s2> — значения типа BLOB, конкатенацию в порядке следования которых выполнит функция.

Для одной функции CONCAT_BLOB допустимо указание только двух аргументов.

Для конкатенации более двух значений необходимо использовать вложенный вызов функции CONCAT_BLOB.

Возвращаемое значение

Функция CONCAT_BLOB возвращает результат конкатенации значений двух аргументов. Тип данных результата — BLOB.

Если значение одного из аргументов равно NULL, то функция вернёт значение NULL.

Вернуться в оглавление

12.9.2.9. CURRENT_DATE

Функция определяет текущую локальную дату (скорректированную с учётом значения опции time_zone сессии) в соответствии со стандартом UTC с точностью до секунд.

Подробнее о корректировке значения опции time_zone для сессии см. п. 11.5. ALTER SESSION), о часовых поясах — п. 12.1.1. Часовые пояса.

Синтаксис функции

```
CURRENT_DATE [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция CURRENT_DATE возвращает текущую дату в соответствии со стандартом UTC с точностью до секунд. Тип данных результата — DATE.

Вернуться в оглавление

12.9.2.10. CURRENT_DBNAME

Функция определяет имя текущей базы данных (подробнее термин «Текущая база данных» см. в 1.1. Термины, акронимы, сокращения).

Синтаксис функции

```
CURRENT_DBNAME [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция CURRENT_DBNAME возвращает имя текущей базы данных. Тип данных результата — VARCHAR.

Вернуться в оглавление

12.9.2.11. CURRENT_ISOLATION_LEVEL

Функция определяет текущий уровня изоляции транзакции.

Синтаксис функции

```
CURRENT_ISOLATION_LEVEL [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция CURRENT_ISOLATION_LEVEL возвращает наименование установленного уровня изоляции транзакции:

– «READ COMMITTED SNAPSHOT» для уровней READ COMMITTED и READ COMMITTED SNAPSHOT;

– «SERIALIZABLE SNAPSHOT» для уровней SERIALIZABLE, SERIALIZABLE SNAPSHOT и REPEATABLE READ.

Тип данных результата — VARCHAR.

Вернуться в оглавление

12.9.2.12. CURRENT_SCHEMA

Функция определяет имя схемы по умолчанию для текущего пользователя.

Синтаксис функции

```
CURRENT_SCHEMA [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция CURRENT_SCHEMA возвращает имя схемы по умолчанию для текущего пользователя. Тип данных результата — VARCHAR.

Вернуться в оглавление

12.9.2.13. CURRENT_TIMESTAMP

Функция определяет текущую локальную дату и время с точностью до тысячной доли секунды (скорректированную с учётом значения опции time_zone сессии) в соответствии со стандартом UTC.

Подробнее о корректировке значения опции time_zone для сессии см. п. 11.5. ALTER SESSION, о часовых поясах — п. 12.1.1. Часовые пояса.

Синтаксис функции

```
CURRENT_TIMESTAMP [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция CURRENT_TIMESTAMP возвращает текущую дату и время в соответствии со стандартом UTC с точностью до тысячной доли секунды. Тип данных результата — TIMESTAMP.

Вернуться в оглавление

12.9.2.14. CURRENT_USER

Функция определяет имя текущего пользователя.

Синтаксис функции

```
CURRENT_USER [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция CURRENT_USER возвращает имя текущего пользователя базы данных. Тип данных результата — VARCHAR.

Вернуться в оглавление

12.9.2.15. DBTIMEZONE[CV]

Функция определяет часовой пояс в соответствии со стандартом UTC, установленный для базы данных.

Подробнее о часовых поясах см. п. 12.1.1. Часовые пояса.

Синтаксис функции

```
DBTIMEZONE [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция возвращает часовой пояс БД в формате: '[+|-]HH:MM', где:

– HH — смещение по часам относительно UTC;

– MM — смещение по минутам относительно UTC;

– [+|-] — знак обозначает смещение относительно UTC: "+" — восточнее, "-" — западнее.

Тип данных результата — VARCHAR (6).

Вернуться в оглавление

12.9.2.16. DECODE

Функция преобразует исходное значение в результирующее на основе заданных условий.

Синтаксис функции

```
DECODE (<исходное_значение>,
      <значение1>, <результат1>
   [, <значение2>, <результат2>...], <значение_по_умолчанию>)
```

Аргументы:

– <исходное_значение> — значение, с которым функция будет сравнивать заданные для сравнения значения <значение1>, <значение2>...;

– <значение1>, <значение2>... — значения, которые функция сравнивает одно за другим со значением аргумента <исходное_значение>;

– <результат1>, <результат2>... — значение, возвращаемое при условии совпадения <исходного_значения> со значением соответствующего аргумента: <значение1>, <значение2>...;

– <значение_по_умолчанию> — значение, используемое по умолчанию, если функция не найдет совпадений.

Значения аргументов могут быть любого типа данных, поддерживаемого в СУБД СОКОЛ, но тип данных сравниваемых аргументов должен быть одинаковым.

В ином случае тип данных аргументов <значение1>, <значение2>... функция попытается неявно привести к типу аргумента <исходное_значение>, а тип данных результата — к типу данных аргумента <результат1> (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция возвращает значение в зависимости от результата сравнения <исходного_значения> с заданными значениями:

– если <исходное_значение> совпадёт со значением аргумента <значение1>, то функция вернёт <результат1>, если со значением аргумента <значение2>, то функция вернёт <результат2> и т.д.;

– если совпадений не найдено, то функция вернёт <значение_по_умолчанию>;

– если совпадений не найдено и не задано <значение_по_умолчанию>, то функция вернёт NULL.

Тип данных результата совпадает с типом аргумента <результат1>.

Вернуться в оглавление

12.9.2.17. EXTRACT[CV]

Функция извлекает указанный элемент из заданного значения даты.

Синтаксис функции

```
EXTRACT (<элемент_даты> FROM <дата>)

<элемент_даты> : YEAR
               | MONTH
               | DAY
               | HOUR
               | MINUTE
               | SECOND
               | DAYOFYEAR
               | WEEKDAY
               | QUARTER
```

Аргументы:

1) <элемент_даты> — обозначение той части даты, которую функция должна извлечь:

– YEAR — год. Возвращаемое значение в диапазонах от -9999 до -1 (для дат до н.э. - с идентификатором BC) или от 1 до 9999 (для дат н.э. - с идентификатором AD) в зависимости от исходной даты;

– MONTH — месяц. Возвращаемое значение в диапазоне от 1 до 12;

– DAY — день месяца. Возвращаемое значение в диапазоне от 1 до 31;

– HOUR — час. Возвращаемое значение в диапазоне от 0 до 23;

– MINUTE — минута. Возвращаемое значение в диапазоне от 0 до 59;

– SECOND — секунда. Возвращаемое значение в диапазоне от 0 до 59;

– DAYOFYEAR — день года. Возвращаемое значение в диапазоне от 1 до 366;

– WEEKDAY — день недели. Возвращаемое значение в диапазоне от 1 до 7, учитывая, что 1 – понедельник;

– QUARTER — квартал. Возвращаемое значение в диапазоне от 1 до 4.

2) <дата> – значение даты, из которого необходимо извлечь значение одного элемента.

Тип данных аргумента DATE, TIMESTAMP.

Если тип данных аргументов CHAR, VARCHAR, то функция попытается неявно преобразовать значение к типу данных TIMESTAMP (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция возвращает значение указанного элемента в заданной дате. Тип данных результата — NUMBER.

Если для извлечения указан элемент даты, отсутствующий в исходном значении даты, то функция вернёт 0.

Если в качестве значения даты указано NULL, то функция вернёт NULL.

Вернуться в оглавление

12.9.2.18. HEX

Функция формирует бинарное значение по его шестнадцатеричному представлению, заданному аргументом в строковом виде.

Синтаксис функции

```
HEX (<x>)
```

Аргумент <x> — значение строкового типа данных (кроме CLOB), которое функция преобразует в тип данных VARBINARY.

В качестве значения аргумента должны использоваться символы шестнадцатеричной системы попарно, где левое значение — старшие четыре разряда, а правое значение — младшие четыре разряда. Допустимые символы: цифры 0-9, латинские буквы A, B, C, D, E, F в верхнем или нижнем регистре.

Возвращаемое значение

Функция возвращает преобразованное значение аргумента. Тип данных результата — VARBINARY.

Если параметр имеет значение NULL, то функция вернёт NULL.

Вернуться в оглавление

12.9.2.19. INSTR[CV]

Функция возвращает номер символа в исходной строке определённого по счёту вхождения указанной подстроки.

Синтаксис функции

```
INSTR (<строка>, <подстрока> [, <номер_символа> [, <номер_вхождения>]])
```

Аргументы:

1) <строка> — исходная строка, в которой функция будет искать вхождение подстроки. Первый символ <строки> всегда имеет позицию 1.

2) <подстрока> — подстрока для поиска в <строке>.

3) <номер_символа> — ненулевое целое число, указывающее номер символа <строки>, с которой функция начнёт поиск <подстроки>. Значение аргумента <номер_символа> можно задавать как положительным числом, так и отрицательным:

– при положительном значении аргумента имеется в виду номер символа <строки> (от 1-го и далее вправо 2-й, 3-й и т.д. вправо), с которым сопоставляется первый символ подстроки;

– при отрицательном значении имеется в виду номер символа <строки> (от последнего (-1), предпоследнего (-2) и т.д. влево), с которыми сопоставляется последний символ подстроки.

Прочтение строк при сопоставлении <строки> и <подстроки> всегда производится слева направо, и функция возвращает номер того символа заданного вхождения в <строке>, который соответствует первому символу <подстроки> при прочтении.

Если значение <номера_символа> не задано, то функция начинает поиск с первого символа <строки>.

4) <номер_вхождения> — целое положительное число, указывающее, какое по счёту вхождение подстроки в строке должна искать функция. Если значение <номер_вхождения> не задано, то по умолчанию равно 1. Если значение аргумента <номер_вхождения> больше 1, то функция после первого совпадения продолжит сравнивать последовательные подстроки в строке до тех пор, пока не будет найдено вхождение подстроки с заданным номером повторения. При этом, каждый новый поиск следующего вхождения функция начинает с символа, следующего за первым символом предыдущего вхождения.

Если в качестве значения аргумента <номер_символа> или <номер_вхождения> указано дробное число, то функция усечёт его до целого числа.

Для аргументов <строка> и <подстрока> допустимы значения типов данных:

– CHAR, VARCHAR;

– NUMBER, BOOLEAN, DATE, TIMESTAMP, BINARY, VARBINARY, ROWID. В этом случае функция попытается значение аргумента неявно привести к значению типа

VARCHAR (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных);

– NULL.

Для аргументов <номер_символа> и <номер_вхождения> допустимы значения типов данных:

– NUMBER;

– CHAR, VARCHAR. В этом случае функция попытается значение аргумента привести к значению типа NUMBER (38, 0);

– NULL.

Подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных.

Возвращаемое значение

Функция вернёт номер символа в исходной строке в соответствии с заданными значениями аргументов.

Тип данных результата — NUMBER (38, 0).

Если значение какого-либо аргумента NULL, то функция вернёт NULL.

Если вхождение не найдено, то функция вернёт 0.

Функция вернёт ошибку если:

– значение аргумента <номер_символа> задаётся числом, равным 0;

– значение аргумента <номер_вхождения> задаётся отрицательным числом или значением, равным 0.

Примеры работы функции

Пример 1:

```
select instr ('AaaaAA, кричал сисадмин', 'a');

INSTR('AaaaAA, кричал сисадмин''a')
-------------------------------------------
 2
```

Пример 2:

```
select instr ('AaaaAA, кричал сисадмин', 'aa', 1, 2);

INSTR('AaaaAA, кричал сисадмин''aa'12)
--------------------------------------------------------------
 3
```

Пример 3:

```
select instr ('aaaAAaa, кричал сисадмин, AA', 'AA', -3, 1);

INSTR('aaaA, кричал сисадмин''AA'-31)
--------------------------------------------------------------
 4
```

Пример 4:

```
select instr ('aaaAA, кричал сисадмин', 'дас', -3, 1);

 INSTR('aaaAA, кричал сисадмин''дас'-31)
-----------------------------------------------------------------
 0
```

Вернуться в оглавление

12.9.2.20. LENGTH

Функция определяет длину значения аргумента в символах.

Синтаксис функции

```
LENGTH (<x>)
```

Аргументы:

– <x> — значение строкового типа данных (кроме CLOB), длину которого вычислит функция.

Возвращаемое значение

Функция LENGTH возвращает количество символов в значении аргумента. Тип данных результата — NUMBER (38, 0).

Если в качестве значения аргумента указано NULL, то функция вернёт значение NULL.

Вернуться в оглавление

12.9.2.21. LENGTHB

Функция определяет длину значения аргумента в байтах.

Синтаксис функции

```
LENGTHB (<x>)
```

Аргумент <x> — значение любого типа данных, длину которого в байтах вычислит функция.

Возвращаемое значение

Функция LENGTHB возвращает количество байт в заданном значении аргумента. Тип данных результата — NUMBER (38, 0).

Если значение аргумента NULL, то функция вернет значение NULL.

Вернуться в оглавление

12.9.2.22. LOCALTIMESTAMP[CV]

Функция имеет реализацию, совпадающую с реализацией функции CURRENT_TIMESTAMP (подробнее см. п. 12.9.2.11. CURRENT_TIMESTAMP).

Вернуться в оглавление

12.9.2.23. LOWER

Функция преобразует все прописные буквы заданного значения аргумента в строчные.

Синтаксис функции

```
LOWER (<строка>)
```

Аргумент <строка> — значение строкового типа данных (кроме CLOB), символы которого функция преобразует в строчные.

Допустимы также значения типов данных NUMBER, BOOLEAN, DATE, TIMESTAMP, BINARY, VARBINARY, ROWID. В этом случае функция попытается неявно

преобразовать значение к типу данных VARCHAR (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция LOWER возвращает значение с преобразованными буквами, при этом остальные символы остаются без изменений. Тип данных результата совпадает с типом данных аргумента, если значение аргумента строкового типа данных. В остальных случаях тип данных результата — VARCHAR.

Если значение аргумента NULL, то функция вернет значение NULL.

Вернуться в оглавление

12.9.2.24. LTRIM

Функция выполняет левостороннее удаление множества символов из заданной строки.

Синтаксис функции

```
LTRIM (<x>[, <символы>])
```

Аргументы:

– <x> — значение строкового типа данных (CHAR, VARCHAR), слева от которого необходимо удалить указанные <символы>;

– <символы> — значение строкового типа данных (CHAR, VARCHAR), множество символов которого функция удалит с левой стороны от значения <x>.

Возвращаемое значение

Функция возвращает строку, оставшуюся от значения <x> после удаления слева всех символов из множества указанных <символов>. Тип результата — VARCHAR.

Если <символы> для удаления не указаны, то по умолчанию слева будут удалены пробелы.

Вернуться в оглавление

12.9.2.25. MOD

Функция возвращает остаток от деления заданного значения на заданный делитель.

Синтаксис функции

```
MOD (<x>, <делитель>)
```

Аргументы:

– <x> — значение любого числового типа данных, которое необходимо поделить. При этом значение аргумента может быть как целым числом, так и дробным (с фиксированной или плавающей запятой);

– <делитель> — значение любого числового типа данных, на которое необходимо поделить указанное значение аргумента <x>.

Возвращаемое значение

Функция возвращает в качестве значения остаток от деления значения аргумента <x> на значение <делителя>.

Знак результата совпадает со знаком аргумента <x>.

Если значение <делителя> равно 0, то будет возвращена ошибка.

Если значение аргумента <x> равно 0, то будет возвращено значение 0.

Если значение хотя бы одного из аргументов равно NULL, то возвращается NULL.

Если абсолютное значение аргумента <x> меньше абсолютного значения <делителя>, то будет возвращено значение аргумента <x>.

Тип данных результата — NUMBER максимальной точности.

Вернуться в оглавление

12.9.2.26. NULLIF

Функция возвращает значение NULL, если аргументы равны.

Синтаксис функции

```
NULLIF (<x>, <y>)
```

Аргументы <x> , <y> — значения, которые функция будет сравнивать между собой. При сравнении функция учитывает регистр символов.

Тип данных аргументов может быть любого типа (кроме BLOB, CLOB), и он должен совпадать (типы CHAR / VARCHAR функция воспринимает как совпадающие).

Возвращаемое значение

Функция возвращает значение, в зависимости от полученного результата сравнения:

– если указанные значения равны, то функция NULLIF возвращает значение NULL;

– если указанные значения не равны, то функция возвращает значение первого выражения.

Недопустимо явное указание NULL в качестве значения аргумента <x>. В этом случае работа функции будет завершена с ошибкой.

Тип данных результата соответствует типу данных первого аргумента.

Вернуться в оглавление

12.9.2.27. NVL

Функция NVL выполняет замену NULL-значений — возвращает значение второго аргумента, если первый аргумент равен NULL, иначе возвращает значение первого аргумента.

Синтаксис функции

```
NVL (<x>, <y>)
```

Аргументы:

– <x> — выражение, значение которого проверяется на NULL;

– <y> — выражение, значение которого будет возвращено, если значение <x> равно NULL.

Тип данных аргументов может быть любым (кроме BLOB, CLOB). Если аргументы имеют разные типы данных, то функция неявно попытается преобразовать

значение второго аргумента в тип данных первого аргумента (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Если значение <x> равно NULL, то функция NVL возвращает значение <y>.

Если значение <x> не равно NULL, то функция NVL возвращает значение <x>.

Тип данных результата соответствует типу данных первого аргумента.

Вернуться в оглавление

12.9.2.28. RANDOM_VALUE

Функция генерирует псевдослучайные дробные числа в пределах обозначенного диапазона (в равномерном распределении).

Синтаксис функции

```
RANDOM_VALUE (<min>, <max>)
```

Аргументы:

– <min> — нижняя граница диапазона, включительно;

– <max> — верхняя граница диапазона, включительно.

Значения аргументов задаются в диапазоне от -0.99999999999999999999999999999999999999e14846 до 0.99999999999999999999999999999999999999e14846 (исключая границы диапазона).

Значения аргументов могут быть:

– любого числового типа данных, поддерживаемого в СУБД;

– типа данных CHAR, VARCHAR, которые функция попытается неявно преобразовать в тип NUMBER максимальной точности (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция RANDOM_VALUE возвращает случайное дробное число из заданного диапазона.

Если значение аргумента <min> больше значения аргумента <max>, то будет возвращена ошибка.

Если значение какого-либо аргумента задано числом вне диапазона, то будет возвращена ошибка.

Если заданное значение аргумента <min> равно значению аргумента <max>, то функция будет возвращать одно и то же число.

Если значение хотя бы одного из аргументов равно NULL, то функция вернёт значение NULL.

Тип данных результата — NUMBER максимальной точности.

Вернуться в оглавление

12.9.2.29. RAND_INT

Функция генерирует псевдослучайные целые числа в пределах обозначенного диапазона (в равномерном распределении).

Синтаксис функции

```
RAND_INT (<min>, <max>)
```

Аргументы:

– <min> — нижняя граница диапазона, включительно;

– <max> — верхняя граница диапазона, включительно.

Значения аргументов задаются в диапазоне от -9223372036854775807 до +9223372036854775806.

Значения аргументов могут быть:

– любого числового типа данных, поддерживаемого в СУБД. При этом, если значение аргумента задано дробным числом, то оно предварительно усекается до целого;

– типа данных CHAR, VARCHAR, которые функция попытается неявно преобразовать в тип NUMBER максимальной точности (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция RAND_INT возвращает случайное целое число. Тип данных результата — NUMBER (38, 0).

Если значение аргумента <min> больше значения аргумента <max>, то будет возвращена ошибка.

Если значение какого-либо аргумента задано числом вне диапазона, то будет возвращена ошибка.

Если заданное значение аргумента <min> равно значению аргумента <max>, то функция будет возвращать одно и то же число.

Если значение хотя бы одного из аргументов равно NULL, то функция вернёт значение NULL.

Вернуться в оглавление

12.9.2.30. REPLACE

Функция выполняет замену указанной с учётом регистра подстроки в заданной строке другой подстрокой.

Синтаксис функции

```
REPLACE (<строка>, <подстрока_для_поиска>, [<подстрока_для_замены>])
```

Аргументы:

– <строка> — значение строкового типа данных, в котором функция ищет для замены подстроку, указанную в <подстрока_для_замены>;

– <подстрока_для_поиска> — подстрока, которую функция ищет в <строке> для замены;

– <подстрока_для_замены> — подстрока, на которую функция заменит найденную <подстроку_для_поиска> в <строке>.

Значения аргументов могут быть:

– строковых типов данных (кроме CLOB);

– типов данных NUMBER, BOOLEAN, DATE, TIMESTAMP, BINARY, VARBINARY, ROWID, которые функция попытается неявно привести к типу VARCHAR (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция REPLACE возвращает новую строку, где все появления <подстроки_для_поиска> заменены на <подстроку_для_замены>.

Тип данных результата совпадает с типом данных аргумента <исходная_строка>.

Если указанная <подстрока_для_поиска> не найдена в <строке> или значение <подстроки_для_поиска> равно NULL, то функция вернёт исходное значение <строки> без изменений.

Если значение <подстроки_для_замены> не указано или равно NULL, то найденная в <строке> <подстрока_для_поиска> будет функцией удалена.

Если значение исходной <строки> равно NULL, то функция вернёт NULL.

Вернуться в оглавление

12.9.2.31. ROUND

Функция округляет заданное значение аргумента до указанного количества десятичных знаков.

Синтаксис функции

```
ROUND (<x> [, <точность>])
```

Аргументы:

– <x> — значение любого числового типа данных, округление которого выполнит функция;

– <точность> — целое число, указывающее точность округления в виде количества десятичных знаков до или после десятичной запятой.

Для аргументов допустимы также значения типов данных CHAR, VARCHAR, которые функция попытается неявно преобразовать в тип NUMBER максимальной точности (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Положительное значение параметра <точность> используется для указания количества цифр после десятичной запятой, до которых необходимо выполнить округление.

Отрицательное значение параметра <точность> используется для указания количества цифр перед запятой, которые необходимо округлить. Например, при <точности> = -1 — округление до ближайшего целого десятка, при <точности> = -2 — до сотни, и т.д.

Функция округляет число в большую сторону, если первая округляемая цифра больше или равна 5. В противном случае число округляется в меньшую сторону. Например, round (45.1368, 2) = 45.14, a round (245.8468, -1) = 250.

Возвращаемое значение

Функция ROUND возвращает всегда значение, округлённое до указанной точности. Тип данных результата — NUMBER максимальной точности.

Если отрицательное значение <точности> превышает количество цифр перед десятичной запятой, то функция вернёт ноль.

Если положительное значение <точности> превышает количество цифр после десятичной запятой, то функция вернёт значение без изменений.

Если значение параметра <точность> опущено, то указанное число <x> функция округлит до ближайшего целого числа.

Если значение одного из аргументов NULL, то функция вернёт значение NULL.

Пример применения функции ROUND

В примере для функции AVG был рассмотрен случай, когда нужно вычислить средний оклад сотрудников подразделения / компании. В рассмотренном примере все оклады указаны в целых числах.

Допустим, необходимо узнать среднюю заработную плату сотрудников одного подразделения, получив целое значение без копеек.

Для примера возьмём таблицу WORKERS_VORONEZH:

```
 Таб №   | Имя   | Фамилия   | Оклад    | Подразделение
---------+-------+-----------+----------+----------------
 212     | Иван  | Петров    | 53525    | МТО
 213     | Пётр  | Иванов    | 27000    | МТО
 214     | Игорь | Котов     | 38000    | Отдел кадров
 215     | Алёна | Котова    | 70000    | ПТО
 216     | Юлия  | Серова    | 65000    | ФЭУ
 217     | Елена | Попова    | 49000    | ФЭУ
```

Выполним соответствующие запрос, с указанием столбца, среднее значение по которому необходимо высчитать и округлить:

```
select round (avg ("Оклад")) as "Средний оклад"
    FROM WORKERS_VORONEZH where "Подразделение" = 'МТО';
```

После успешно выполненного запроса получим результат:

```
 Средний оклад
----------------------
 40263
```

Без округления среднее значение равно 40262,50.

Вернуться в оглавление

12.9.2.32. ROWNUM[CV]

Функция позволяет получить номер строки в результирующем наборе SELECT-запроса.

Синтаксис функции

```
ROWNUM
```

Функцию допустимо применять:

– в списке SELECT-запроса для получения номера строки в результирующем наборе;

– в качестве замены LIMIT при использовании в WHERE.

Возвращаемое значение

Функция возвращает число, представляющее порядковый номер строки в выборке, где значение 1 соответствует первой строке, 2 – второй строке и т.д. Тип результата — NUMBER.

Пример применения функции ROWNUM

Допустим, у нас есть таблица с данными сотрудников. Необходимо сделать выборку данных и для удобства восприятия упорядочить строки по значениям конкретных столбцов, пронумеровав каждую строку.

Для примера возьмём ранее созданную таблицу WORKERS:

```
 TN     | NAME   | LASTNAME| AGE     | GENDER
---------+----- --+---------+----------+--------
 212    | Иван   | Петров  | 53      | м
 213    | Пётр   | Иванов  | 27      | м
 214    | Игорь  | Котов   | 53      | м
 215    | Анна   | Гузеева | 70      | ж
 216    | Юлия   | Серова  | 65      | ж
 217    | Елена  | Попова  | 19      | ж
 218    | Ирина  | Зотова  | 48      | ж
 219    | Фёдор  | Уваров  | 67      | м
 220    | Алёна  | Котова  | 65      | ж
 221    | Андрей | Быков   | 65      | м
```

Необходимо сделать выборку сотрудников, возраст которых больше 50 лет, и упорядочить полученный список в порядке уменьшения возраста. Чтобы добавить столбец с номерами строк перед, добавим столбец с помощью функции ROWNUM:

```
select ROWNUM, TN, NAME || ' ' || LASTNAME as fullname, AGE
   from workers where AGE>50 order by AGE desc;
```

После выполнения команды будет получен результат:

```
 ROWNUM  | TN     | FULLNAME      | AGE
---------+---------+---------------+----------
 1      | 215    | Анна Гузеева  | 70
 2      | 219    | Фёдор Уваров  | 67
 3      | 216    | Юлия Серова   | 65
 4      | 221    | Андрей Быков  | 65
 5      | 220    | Алёна Котова  | 65
 6      | 212    | Иван Петров   | 53
```

```
7    | 214    | Игорь Котов   | 53
```

Таким образом получили пронумерованный перечень сотрудников, соответствующих запросу.

Вернуться в оглавление

12.9.2.33. RTRIM

Функция выполняет правостороннее удаление множества символов из заданной строки.

Синтаксис функции

```
RTRIM (<x>[, <символы>])
```

Аргументы:

– <x> — значение строкового типа данных (CHAR, VARCHAR), справа от которого необходимо удалить указанные <символы>;

– <символы> — значение строкового типа данных (CHAR, VARCHAR), множество символов которого функция удалит с правой стороны от значения <x>.

Кроме строковых типов данных для аргументов допустимы значения типов данных NUMBER, BOOLEAN, DATE, TIMESTAMP, BINARY, VARBINARY, ROWID, которые функция попытается неявно преобразовать в тип VARCHAR (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция возвращает строку, оставшуюся от значения <x> после удаления справа всех символов из множества указанных <символов>. Тип данных результата совпадает с типом данных первого аргумента, если значение аргумента строкового типа данных. В остальных случаях тип данных результата — VARCHAR.

Если <символы> для удаления не указаны, то по умолчанию справа будут удалены пробелы.

Если в качестве значения аргумента указано NULL, то функция вернёт значение NULL.

Вернуться в оглавление

12.9.2.34. SESSION_ID[CV]

Функция определяет номер текущей сессии пользователя, уникальный в рамках работы базы данных от запуска до её останова.

Синтаксис

```
SESSION_ID[()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция возвращает уникальный номер текущей сессии пользователя. Тип данных результата — BIGINT.

Вернуться в оглавление

12.9.2.35. SESSIONTIMEZONE[CV]

Функция определяет часовой пояс в соответствии со стандартом UTC, установленный для текущей сессии.

Подробнее о часовых поясах см. п. 12.1.1. Часовые пояса.

Синтаксис функции

```
SESSIONTIMEZONE [()]
```

Функция не имеет аргументов.

Возвращаемое значение

Функция возвращает часовой пояс сессии в формате: '[+|-]HH:MM', где:

– HH — смещение по часам относительно UTC;

– MM — смещение по минутам относительно UTC;

– [+|-] — знак обозначает смещение относительно UTC: "+" — восточнее, "-" — западнее.

Тип данных результата — VARCHAR (6).

Вернуться в оглавление

12.9.2.36. SUBSTR

Функция извлекает подстроку из заданной строки.

Синтаксис функции

```
SUBSTR (<x>, <начальный_символ> [, <количество_символов>])
```

Аргументы:

– <x> — значение строкового типа данных (кроме CLOB), из которого необходимо извлечь подстроку;

– <начальный_символ> — номер символа в строке <x>, с которого начинается извлекаемая подстрока. Номера символов в строке отсчитываются, начиная с 1;

– <количество_символов> — положительное значение, обозначающее количество символов извлекаемой подстроки. Данный аргумент указывать не обязательно.

Дробное число, указанное в качестве значения аргумента <начальный_символ> или <количество_символов>, усекается функцией до целого числа.

Возвращаемое значение

Функция SUBSTR возвращает извлечённую из строки <x> подстроку в соответствии с заданными условиями:

– если значение аргумента <начальный_символ> равно 0, то функция SUBSTR принимает значение этого параметра равным 1.

Например:

```
select substr ('Наш сисадмин', 0, 3);
select substr ('Наш сисадмин', 1, 3);

Результат одинаковый:

 SUBSTR('Наш сисадмин'03)
--------------------------------------------
 Наш

SUBSTR('Наш сисадмин'13)
--------------------------------------------
 Наш
```

– если значение аргумента <начальный_символ> задано положительным числом, то оно означает позицию, считая от начала строки слева направо.

Например:

```
select substr ('Наш сисадмин', 7, 3);


 SUBSTR('Наш сисадмин'73)
-----------------------------------------
 сад
```

– если значение аргумента <начальный_символ> задано отрицательным числом, то оно означает позицию, считая от конца строки справа налево.

Например:

```
select substr ('Наш сисадмин', -6, 3);

 SUBSTR('Наш сисадмин'-63)
-------------------------------------------
 сад
```

– если аргумент <количество_символов> не задан, то функция SUBSTR вернёт строку от указанного <начального_символа> до конца строки <x>.

Например:

```
select substr ('Наш сисадмин', -5);

 SUBSTR('Наш сисадмин'-5)
-----------------------------------------
 админ
```

– если значение аргумента <количество_символов> задано числом меньше 1, то функция SUBSTR возвращает значение NULL.

– если какой-либо из аргументов имеет значение NULL, то функция вернёт NULL.

Тип данных результата совпадает с типом данных аргумента <x>.

Вернуться в оглавление

12.9.2.37. TIMESTAMP_ADD[CV]

Функция вычисляет новое значение даты с учётом заданной дельты (промежутка времени) для указанного элемента даты. Правила исчисления дат соответствуют григорианскому календарю.

Синтаксис функции

```
TIMESTAMP_ADD (<элемент_даты>, <дельта>, <дата>)
```

Аргументы:

1) <элемент_даты> — обозначение той части даты даты, к которой функция должна применить заданное значение дельты:

– YEAR — год;

– MONTH — месяц;

– DAY — дни месяца;

– HOUR — час;

– MINUTE — минута;

– SECOND — секунда;

– WEEK — неделя;

– QUARTER — квартал.

2) <дельта> — целочисленное значение промежутка времени, определяющее величину изменения заданного элемента даты. Допустимо задавать значение как положительным числом, так и отрицательным. Если в качестве значения <дельта> указано дробное значение, то функция усечёт его до целого. Для аргумента <дельта> допустимы значения типа данных:

– любого числового типа данных;

– любого другого, в случае допустимости их неявного преобразования к типу NUMBER (подробнее см. 12.3.5. Свод допустимости преобразований типов данных).

3) <дата> — значение исходной даты, относительно которой нужно вычислить новое значение на основе значения <дельты>. Для аргумента <дата> допустимы значения типа данных:

– DATE или TIMESTAMP;

– строкового типа данных (CHAR, VARCHAR), которые функция попытается неявно привести к типу TIMESTAMP (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных). В случае невозможности преобразования работа функции завершится с ошибкой.

Возвращаемое значение

Функция TIMESTAMP_ADD возвращает значение даты с учётом заданной дельты (промежутка времени) для указанного элемента даты.

Тип данных результата — TIMESTAMP.

Если результирующая дата будет относиться к периоду до н.э., то функция вернёт значение с соответствующим идентификатором эры — BC.

Если при приращении указанного значения дельты к заданному элементу даты произошло его переполнение, то функция должна вычислить результирующую дату с учётом всех вышестоящих элементов даты и переходов между ними (годами, месяцами, днями и т.д.) в соответствии с григорианским календарём. Например:

– результатом выполнения команды:

```
select timestamp_add (HOUR, 45, '2022-01-01 11:25:00');
```

будет дата '2022-01-03 08:25:00.000';

– результатом выполнения команды:

```
select timestamp_add (DAY, 45, '2022-01-01');
```

будет дата '2022-02-15 00:00:00.000';

– результатом выполнения команды:

```
select timestamp_add (MONTH, 7, '2022-09-05');
```

будет дата '2023-04-05 00:00:00.000'.

При приращении заданной дельты к месяцу, когда в исходной дате указан последний день месяца или день исходного месяца не существует в результирующем месяце, функция всегда вернёт значение последнего дня результирующего месяца. Например:

– результатом выполнения команды:

```
select timestamp_add (MONTH , 1, '2022-01-30');
```

будет дата '2022-02-28';

– результатом выполнения команды:

```
select timestamp_add (MONTH , -1, '2022-02-28');
```

будет дата '2022-01-31';

Если аргумент <дата> имеет значение NULL, то функция должна вернуть значение NULL.

Если аргумент <дельта> имеет значение NULL, то работа функции завершится с ошибкой.

Вернуться в оглавление

12.9.2.38. TIMESTAMP_FROM_PARTS

Функция преобразует заданные числовые значения аргументов в дату.

Синтаксис функции

```
TIMESTAMP_FROM_PARTS ( <год>, <месяц>, <день>, <часы>, <минуты>, <секунды>,
    <разряды_секунды>, <порядок_разрядов_секунды>)
```

Аргументы:

– <год> — целочисленное значение года;

– <месяц> — целочисленное значение месяца в диапазоне 1 — 12, где январь представлен как 1, а декабрь — как 12;

– <день> — целочисленное значение дня месяца. Значение соответствует числу месяца в диапазоне 1 — 28..31;

– <час> — целочисленное значение часа в диапазоне 0 — 23;

– <минута> — целочисленное значение минуты в диапазоне 0 — 59;

– <секунда> — целочисленное значение секунды в диапазоне 0 — 59;

– <разряды_секунды>, <порядок_разрядов_секунды> — целочисленные значения, указание которых позволяет задать значение долей секунды в возвращаемой дате.

При построении даты значение доли секунды вычисляется как:

```
<разряды_секунды> / 10^<порядок_разрядов_секунды>
```

Значение аргумента <разряды_секунды> задаётся в диапазоне от 0 до 9999999999999999999 (18 разрядов), значение аргумента <порядок_разрядов_секунды> задаётся в диапазоне от 1 до 18.

Пример преобразования заданных числовые значений аргументов в дату:

```
select TIMESTAMP_FROM_PARTS (2023, 5, 5, 12, 2, 2, 12, 3);

 TIMESTAMP_FROM_PARTS(2023551222123)
-------------------------------------------
 2023-05-05 12:02:02.012
```

Возвращаемое значение

Функция TIMESTAMP_FROM_PARTS возвращает значение даты, собранное из заданных значений.

Тип данных результата — TIMESTAMP.

Вернуться в оглавление

12.9.2.39. TIMESTAMP_DIFF[CV]

Функция должна вычисляет разницу между заданными датами в выбранном элементе даты в соответствии с григорианским календарем.

Синтаксис функции

```
TIMESTAMP_DIFF (<элемент_даты>, <дата1>, <дата2>)
```

Аргументы:

1) <дата1>, <дата2> — значения начальной и конечной дат соответственно.

Для аргументов <дата1> и <дата2> допустимо значение типа:

– DATE или TIMESTAMP;

– значения строкового типа данных (CHAR, VARCHAR), которые функция должна неявно попытаться привести к типу TIMESTAMP (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных). В случае невозможности преобразования должна быть возвращена ошибка.

Если значения начальной и конечной дат имеют разные типы данных даты, то функция неявно попытается преобразовать значения к типу TIMESTAMP и присвоит значение 0 недостающим частям другого аргумента.

2) <элемент_даты> — обозначение той части даты, разницу в котором должна вернуть функция:

– DAY — день;

– HOUR — час;

– MINUTE — минута;

– SECOND — секунда.

Возвращаемое значение

Функция TIMESTAMP_DIFF возвращает разницу между заданными датами в выбранном элементе даты. Тип данных результата — NUMBER.

При вычислении разницы в выбранном элементе между двумя датами, функция возвращает целое число значений между этими датами ("целое число дней", "целое число часов" и т.д.), с отсечением дробной части и учитывая все элементы даты. Например:

При вычислении разницы в днях (элемент DAY) между датами:

– '2022-12-31 15:12' и '2023-01-02 14:00' вернётся значение 1, что соответствует одному целому дню (реальная разница составляет 1 день 22 часа 48 минут);

– '2022-12-31 15:12' и '2023-01-02 16:00' вернётся значение 2, что соответствует двум целым дням (реальная разница составляет 2 дня 48 минут);

При вычислении разницы в часах (элемент HOUR) между датами:

– '2022-12-31 15:12' и '2023-01-02 14:00' вернётся значение 46, что соответствует 46 целым часам (реальная разница составляет 46 часов 48 минут);

– '2022-12-31 15:12' и '2023-01-02 16:00' вернётся значение 48, что соответствует 48 целым часам (реальная разница составляет 48 часов 48 минут).

Если в качестве значения хотя бы одного из аргумента с датой указано NULL, то функция должна вернуть NULL.

Если конечная дата больше, чем начальная, функция вернёт положительное значение разницы.

Если конечная дата меньше, чем начальная, функция вернёт отрицательное значение разницы.

Вернуться в оглавление

12.9.2.40. TIMESTAMP_TRUNC[CV]

Функция усекает заданное значение даты до указанного элемента даты.

Синтаксис функции

```
TIMESTAMP_TRUNC ([<элемент_даты>], <дата>)
```

Аргументы:

1) <элемент_даты> — обозначение той части даты даты, до которой функция усечёт исходное значение:

– YEAR — год, усекается до первого дня года;

– MONTH — месяц, усекается до первого дня месяца;

– DAY — день месяца, усекается время. Является значением по умолчанию;

– HOUR — час, усекаются минуты, секунды и доли секунд;

– MINUTE — минута, усекаются секунды и доли секунд;

– SECOND — секунда, усекаются доли секунд;

– QUARTER — квартал, усекается до первого дня квартала.

2) <дата> — значение исходной даты, которую необходимо усечь.

Для аргумента <дата> допустимо значение типа:

– DATE или TIMESTAMP;

– значения строкового типа данных (CHAR, VARCHAR), которые функция попытается неявно привести к типу TIMESTAMP (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных). В случае невозможности преобразования должна быть возвращена ошибка.

Возвращаемое значение

Функция TIMESTAMP_TRUNC возвращает значение даты, усечённое до указанного элемента. Тип данных результата — TIMESTAMP.

Если в качестве значения аргумента <дата> указано NULL, то функция вернёт NULL.

Если значение аргумента <элемент_даты> не задано, то функция вернёт значение даты с усечённым временем.

Примеры работы функции при усечении даты '2022-12-31 15:12:25.325' до:

– года (YEAR) вернётся значение '2022-01-01 00:00:00.000';

– месяца (MONTH) вернётся значение '2022-12-01 00:00:00.000';

– дня месяца (DAY) вернётся значение '2022-12-31 00:00:00.000';

– часа (HOUR) вернётся значение '2022-12-31 15:00:00.000';

– минут (MINUTE) вернётся значение '2022-12-31 15:12:00.000';

– секунд (SECOND) вернётся значение '2022-12-31 15:12:25.000';

– квартала (QUARTER) вернётся значение '2022-10-01 00:00:00.000'.

Вернуться в оглавление

12.9.2.41. TO_CHAR

Функция преобразует заданное значение аргумента в символьное представление.

Синтаксис функции

```
TO_CHAR (<x>[, <формат>])
```

Аргументы:

1) <x> — значение, которое функция преобразует в символьное представление.

Для аргумента <x> допустимо значение типа:

– CHAR, VARCHAR, BINARY, VARCHAR, ROWID, BOOLEAN, NUMBER — если для функции задано значение только одного аргумента <x>;

– DATE, TIMESTAMP — если для функции задано значение двух аргументов (<x> и <формат>).

2) <формат> — строковый литерал, задающий формат символьного представления значения аргумента <x>.

Для преобразования дат в строковый тип в таблице 22 приведен список допустимых элементов, которые могут быть использованы в различных комбинациях.

Таблица 22 — Список допустимых элементов формата функции TO_CHAR, используемых для преобразования даты и времени

Элемент	Описание
BC или AD B.C. или A.D.	Индикатор эры с точками или без них. Независимо от того, какие из указанных букв индикатора будут использованы в значении аргумента, результат вывода будет соответствовать форматируемому значению: для периода нашей эры будет выводиться значение «A.D.» или «AD», для периода до нашей эры — «B.C.» или «BC»
CC	Цифры века: - если последние две цифры года находятся между 01 и 99 (включительно), то век на единицу больше, чем первые две цифры этого года; - если последние две цифры четырёхзначного года равны 00, то век совпадает с первыми двумя цифрами этого года. Двадцать первый век начался 2001-01-01. Например, в 2001 году возвращается 21; 2000 возвращает 20
SYYYY	Значение года с префиксом, указывающим эру, где: - S - префикс эры (н.э - «+0», до н.э. – «-0»); - YYYY - обозначение года.
YYYY	Значение года, содержащее больше двух цифр
YY	Последние две цифры года
Q	Квартал года (1, 2, 3, 4)
MONTH	Полное английское наименование месяца
MON	Сокращенное английское наименование месяца
MM	Месяц (01-12; январю соответствует значение 01 и т.д.)
RM	Римская цифра обозначения месяца (I-XII, январю соответствует значение I и т.д.)
A.M. или P.M. AM или PM	Индикатор периода дня с точками или без них. Независимо от того, какие из указанных букв индикатора будут использованы в значении аргумента, результат вывода будет соответствовать форматируемому значению: для времени до обеда будет выводиться значение «A.M.» или «AM», для времени после обеда — «P.M.» или «PM»
DDD	День года (001-366)
DD	День месяца (01-31)
DAY	Полное английское наименование дня недели
DY	Сокращенное английское наименование дня недели
D	Номер дня недели (1-7, с учётом того, что первый день недели — воскресенье
HH или HH24	Час дня (00-23)

Элемент	Описание
HH12	Час дня (01-12)
MI	Минуты (00-59)
SS	Секунды (00-59)
SSSSS	Секунды после полуночи (00000-86399)
MS	Миллисекунды (000-999)
FF	Дробные секунды с точностью до 6 разрядов дробных долей секунды
FF14	Дробные секунды с точностью до 14 разрядов дробных долей секунды

Например, выведем текущую дату в формате с указанием века, года с префиксом, дополнительно указанием эры, времени до обеда или после, месяц арабской цифрой и римской, дня месяца, часа в 24 часовом формате, минуты и дробной секунды:

```
select to_char (current_timestamp, 'CC SYYYY BC AM MM RM DD HH MI FF');


 TO_CHAR(CURRENT_TIMESTAMP'CC SYYYY BC AM MM RM DD HH MI FF')
-------------------------------------------------------------------
 21 +02023 AD PM 04 IV 26 23 06 078490
```

Возвращаемое значение

Функция TO_CHAR возвращает преобразованное значение аргумента. Тип данных результата — VARCHAR.

Вернуться в оглавление

12.9.2.42. TO_DATE[CV]

Функция преобразует заданное символьное значение аргумента в дату.

Синтаксис функции

```
TO_DATE (<x>[, <формат>])
```

Аргументы:

– <x> — значение строкового типа данных, которое функция преобразует в дату;

– <формат> — формат даты, с помощью которого функция распознаёт части даты в исходном значении <x> для дальнейшего преобразования в дату. Если <формат> не задан, то по умолчанию применяется формат: 'YYYY-MM-DD'.

Для аргументов допустимы значения строкового типа данных — CHAR, VARCHAR.

Для преобразования строки в дату далее в таблице 23 приведён список допустимых элементов, которые могут быть использованы в различных комбинациях.

Для верного преобразования указанные элементы даты в <формате> должны порядково точно совпадать с указанными элементами даты в значении аргумента <x>.

Между элементами в <формате> и в <x> допустимы только разделители.

Символы "/" (слэш), " " (пробел), "." (точка), "," (запятая) ":" (двоеточие) воспринимаются функцией как разделители элементов и указание любого из них в <формате> соответствует любому из указанных разделителей в значении <x>. Если присутствуют иные символы, то работа функции будет завершена с ошибкой.

Между элементами в <формате> и в <x> допустимо указание множества пробелов, количество которых может и не совпадать.

Таблица 23 — Список допустимых элементов в значении аргумента <формат>

Элемент	**Описание**	**Учитывается в результирующей дате (V)**
BC или AD B.C. или A.D.	Обозначение эры. Разрешено использовать только при наличии обозначения года без префикса эры (см. параметры далее). Иначе выполнение функции завершится с ошибкой	V
SYYYY	Обозначение года с префиксом, указывающим эру, где: - S - префикс эры (н.э - «+0», до н.э. – «-0»); - YYYY - обозначение года. Такое обозначение разрешено использовать только при отсутствии обозначения эры (см. параметр выше). Иначе выполнение функции завершится ошибкой	V
YYYY или YY	Обозначение года (значение года, содержащее больше двух цифр или последние 2 цифры года)	V
MONTH	Полное английское наименование месяца (january, february, march, april, may, june, july, august, september, october, november, december) Регистр букв не важен	V
MON	Первые три буквы английского наименования месяца (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec) Регистр букв не важен	V
MM	Месяц (1-12; январь - это 1). При указании значения, состоящего из одной цифры, допускается введение как с лидирующим нулём, так и без него	V
RM	Римская цифра обозначения месяца (I-XII; январь - это I и т.д.)	V
A.M. или P.M. AM или PM	Обозначение времени до обеда или после обеда. При использовании этого параметра часы должны быть указаны в	V

Элемент	Описание	Учитывается в результирующей дате (V)
	12 часовом формате, иначе работа функции будет завершена с ошибкой. Регистр букв не важен	
DDD	День года (1-366). При указании значения, состоящего из одной или двух цифр, допускается введение как с лидирующим нулём, так и без него. При этом день года не должен противоречить указанной дате по юлианскому календарю, иначе работа функции будет завершена с ошибкой	-
DD	День месяца (1-31) При указании значения, состоящего из одной цифры, допускается введение как с лидирующим нулём, так и без него	V
DAY	Полное английское наименование дня недели (monday, tuesday, wednesday, thursday, friday, saturday, sunday). Регистр букв не важен. При этом день недели не должен противоречить указанной дате по юлианскому календарю, иначе работа функции будет завершена с ошибкой	-
DY	Первые три буквы английского наименования дня недели (mon, tue, wed, thu, fri, sat, sun). Регистр букв не важен. При этом день недели не должен противоречить указанной дате по юлианскому календарю, иначе работа функции будет завершена с ошибкой	-
D	Номер дня недели (1-7, с учётом того, что первый день недели – воскресенье). При этом номер дня недели не должен противоречить указанной дате по юлианскому календарю, иначе работа функции будет завершена с ошибкой	-
HH24	Значение часа в 24-х часовом формате (0-23)	V
HH или HH12	Значение часа в 12-ти часовом формате (1-12)	V
MI	Минуты (0-59)	V
SS	Секунды (0-59)	V
SSSSS	Секунды после полуночи (0-86399). Если значение в исходной строке превышает допустимый максимум, то работа функции будет завершена с ошибкой	-
MS	Миллисекунды (0-999). Если значение в исходной строке превышает допустимый максимум, то значение должно быть округлено до трёх цифр.	-

Возвращаемое значение

Функция TO_DATE возвращает преобразованное значение аргумента. Тип данных результата — DATE.

Вернуться в оглавление

12.9.2.43. TO_TIMESTAMP[CV]

Функция преобразует заданное символьное значение аргумента в дату со временем.

Синтаксис функции

```
TO_TIMESTAMP (<x>[, <формат>])
```

Аргументы:

– <x> — значение строкового типа данных, которое функция преобразует в дату со временем;

– <формат> — формат даты, с помощью которого функция распознаёт части даты в исходном значении <x> для дальнейшего преобразования в дату. Если <формат> не задан, то по умолчанию применяется формат: 'YYYY-MM-DD'.

Для аргументов допустимы значения строкового типа данных — CHAR, VARCHAR.

Для преобразования строки в дату далее в таблице 24 приведён список допустимых элементов, которые могут быть использованы в различных комбинациях.

Для верного преобразования указанные элементы даты в <формате> должны порядково точно совпадать с указанными элементами даты в значении аргумента <x>.

Между элементами в <формате> и в <x> допустимы только разделители.

Символы "/" (слэш), " " (пробел), "." (точка), "," (запятая) ":" (двоеточие) воспринимаются функцией как разделители элементов и указание любого из них в <формате> соответствует любому из указанных разделителей в значении <x>. Если присутствуют иные символы, то работа функции будет завершена с ошибкой.

Между элементами в <формате> и в <x> допустимо указание множества пробелов, количество которых может и не совпадать.

Таблица 24 — Список допустимых элементов в значении аргумента <формат>

Элемент	Обозначение	Учитывается в результирующей дате (V)
BC или AD B.C. или A.D.	Обозначение эры. Разрешено использовать только при наличии обозначения года без префикса эры (см. параметры далее). Иначе выполнение функции завершится с ошибкой	V
SYYYY	Обозначение года с префиксом, указывающим эру, где: – S - префикс эры (н.э - «+0», до н.э. – «-0»); – YYYY - обозначение года. Такое обозначение разрешено использовать только при отсутствии обозначения эры (см. параметр выше). Иначе выполнение функции завершится с ошибкой	V
YYYY или YY	Обозначение года (значение года, содержащее больше двух цифр или последние 2 цифры года)	V
MONTH	Полное английское наименование месяца (january, february, march, april, may, june, july, august, september, october, november, december). Регистр букв не важен	V
MON	Первые три буквы английского наименования месяца (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec). Регистр букв не важен	V
MM	Месяц (1-12; январь - это 1). При указании значения, состоящего из одной цифры, допускается введение как с лидирующим нулём, так и без него	V
RM	Римская цифра обозначения месяца (I-XII; январь - это I и т.д.)	V
A.M. или P.M. AM или PM	Обозначение времени до обеда или после обеда. При использовании этого параметра часы должны быть указаны в 12 часовом формате, иначе выполнение функции завершится с ошибкой. Регистр букв не важен	V
DDD	День года (1-366). При указании значения, состоящего из одной или двух цифр, допускается введение как с лидирующим нулём, так и без него. При этом день года не должен противоречить указанной дате по юлианскому календарю, иначе выполнение функции завершится с ошибкой	-
DD	День месяца (1-31). При указании значения, состоящего из одной цифры, допускается введение как с лидирующим нулём, так и без него	V
DAY	Полное английское наименование дня недели (monday, tuesday, wednesday, thursday, friday, saturday, sunday). Регистр букв не важен. При этом день недели не должен противоречить указанной дате по юлианскому календарю, иначе выполнение функции завершится с ошибкой	-
DY	Первые три буквы английского наименования дня недели (mon, tue, wed, thu, fri, sat, sun). Регистр букв не важен. При этом день недели не должен противоречить указанной дате по юлианскому календарю, иначе выполнение функции завершится с ошибкой	-

Элемент	Обозначение	Учитывается в результирующей дате (V)
D	Номер дня недели (1-7, с учётом того, что первый день недели – воскресенье). При этом номер дня недели не должен противоречить указанной дате по юлианскому календарю, иначе выполнение функции завершится с ошибкой	-
HH24	Значение часа в 24-х часовом формате (0-23)	V
HH или HH12	Значение часа в 12-ти часовом формате (1-12)	V
MI	Минуты (0-59)	V
SS	Секунды (0-59)	V
SSSSS	Секунды после полуночи (0-86399). Если значение в исходной строке превышает допустимый максимум, то выполнение функции завершится с ошибкой	-
MS	Миллисекунды (0-999). Если значение в исходной строке превышает допустимый максимум, то значение будет округлено до трёх цифр	V
FF[<точность>]	Значение доли секунды, которую необходимо округлить до указанной точности (например, при указании FF3 значение долей секунд будут округлены до тысячной доли). Пример: select to_timestamp ('2012-08-02 11:10:10.132456', 'yyyy-mm-dd HH24:MI:SS.FF5'); Должно сохранить значение даты: 2012-08-02 11:10:10.13246 Допустимо для указания точности использовать цифры от 1 до 9. Начиная со значения точности > 2, применимы альтернативные написания: fff = ff3, ffff = ff4 и т.д. Если после FF точность не указана, то значение точности по умолчанию = 3. Если после FF указано значение, превышающее допустимое, то выполнение функции завершится с ошибкой. Если значение долей секунд в исходной строке меньше указанной точности для округления, то значение долей будет дополнено нулями до указанной точности для округления	V

Возвращаемое значение

Функция TO_TIMESTAMP возвращает преобразованное значение аргумента. Тип данных результата — TIMESTAMP.

Вернуться в оглавление

12.9.2.44. TRANSACTION_ID[CV]

Функция определяет уникальный номер транзакции в рамках существования конкретной БД.

Синтаксис функции

```
TRANSACTION_ID[()]
```

Возвращаемое значение

Функция возвращает уникальный номер транзакции. Тип данных результата — NUMBER (38,0).

Функции вернёт NULL если транзакция не запущена.

Вернуться в оглавление

12.9.2.45. TRIM[CV]

Функция выполняет удаление множества символов или пробелов из заданной строки.

Синтаксис функции

```
TRIM ([[<правило>] <символы> FROM] <x>)

<правило> = LEADING | TRAILING | BOTH
```

Аргументы:

1) <правило> — правило удаления символов означает, в какой части строки <x> будут символы из множества символов <символы>:

– LEADING — в начале строки;

– TRAILING — в конце строки;

– BOTH — в начале и конце строки.

2) <x> — значение строкового типа данных (CHAR, VARCHAR), в котором выполняется удаление символов.

3) <символы> — значение строкового типа данных (CHAR, VARCHAR), содержащее множество символов для удаления из значения <x>.

Возвращаемое значение

Функция возвращает строку, оставшуюся от значения <x> после удаления символов из множества указанных <символов> или пробелов. Тип результата — VARCHAR.

Если <правило> удаления символов не указано, то будут удалено заданное множество символов с начала и с конца строки.

Если <символы> для удаления не указаны, то по умолчанию будут удалены пробелы.

Вернуться в оглавление

12.9.2.46. TRUNC

Функция усекает заданное значение до указанного количества десятичных знаков.

Синтаксис функции

```
TRUNC (<x> , [<точность>])
```

Аргументы:

– <x> — значение любого числового типа данных, усечение которого выполнит функция;

– <точность> — целое число, указывающее точность усечения в виде количества десятичных знаков до или после десятичной запятой.

Для аргументов также допустимы значения типов данных CHAR, VARCHAR, которые функция неявно попытается преобразовать в тип NUMBER максимальной точности (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Положительное значение параметра <точность> используется для указания количества цифр после десятичной запятой, до которых необходимо выполнить усечение.

Например:

```
select trunc(2145.1368, 2);
```

```
 TRUNC(2145.13682)
------------------------
 2145.13
```

Отрицательное значение параметра <точность> используется для указания количества цифр перед запятой, которые необходимо усечь. Например, при <точности> = -1 — усечение до ближайшего целого десятка, при <точности> = -2 — до сотни, и т.д.

Например:

```
select trunc(2175.1368, -2);

 TRUNC(2175.1368-2)
-------------------------
 2100
```

Возвращаемое значение

Функция возвращает значение, усечённое до указанной точности. Тип данных результата — NUMBER максимальной точности.

Если отрицательное значение <точности> превышает количество цифр перед десятичной запятой, то функция вернёт ноль.

Если положительное значение <точности> превышает количество цифр после десятичной запятой, то функция вернёт значение без изменений.

Если значение параметра <точность> опущено, то указанное число <x> усекается до ближайшего целого числа.

Вернуться в оглавление

12.9.2.47. UPPER

Функция преобразует все строчные буквы заданного значения аргумента в прописные.

Синтаксис функции

```
UPPER (<строка>)
```

Аргумент <строка> — значение строкового типа данных (кроме CLOB), символы которого функция преобразует в прописные.

Для аргументов также допустимы значения типов данных NUMBER, BOOLEAN, DATE, TIMESTAMP, BINARY, VARBINARY, ROWID, которые функция неявно попытается преобразовать в тип VARCHAR (подробнее о неявном преобразовании см. п. 12.3.5. Свод допустимости преобразований типов данных).

Возвращаемое значение

Функция UPPER возвращает значение с преобразованными буквами, при этом остальные символы остаются без изменений. Тип данных результата совпадает с типом данных аргумента, если значение аргумента строкового типа данных. В ином случае тип данных результата — VARCHAR.

Если в качестве значения аргумента указано NULL, то функция вернёт значение NULL.

Вернуться в оглавление

12.9.2.48. VERSION

Функция возвращает текущую версию сервера.

Синтаксис функции

```
VERSION [()]
```

Возвращаемое значение

Функция возвращает текущую версию сервера. Тип данных результата — VARCHAR.

Вернуться в оглавление

13. ИДЕНТИФИКАЦИЯ ВЕРСИИ СЕРВЕРА

В случае возникновения ошибки или нештатной ситуации при работе с продуктом необходимо понимать, с какой версией продукта идёт работа.

Для удобства при старте сервера или утилит выводится номер версии в формате:

```
<наименование утилиты> version = <№ версии>
```

Наименования утилит, при старте которых выводится номер версии:

– vsql_console;

– vsql_copy;

– vsql_server_wait.

Узнать версию сервера СУБД СОКОЛ можно также запросом:

```
select version();
```

После успешно выполненной команды будет возвращена строка с версией сервера СУБД:

```
VERSION()
----------------
 SOQOL 1
```

Выполнить данный запрос может любой пользователь с правом подключения к БД.

Вернуться в оглавление

14. ОБЪЕКТЫ, ФОРМИРУЕМЫЕ СИСТЕМОЙ ПРИ СОЗДАНИИ БД В СУБД СОКОЛ

При создании каждой базы данных в СУБД СОКОЛ (включая БД сервиса управления) система автоматически формирует набор базовых объектов.

Автоматически формируемые объекты БД отражены на рисунке 13.

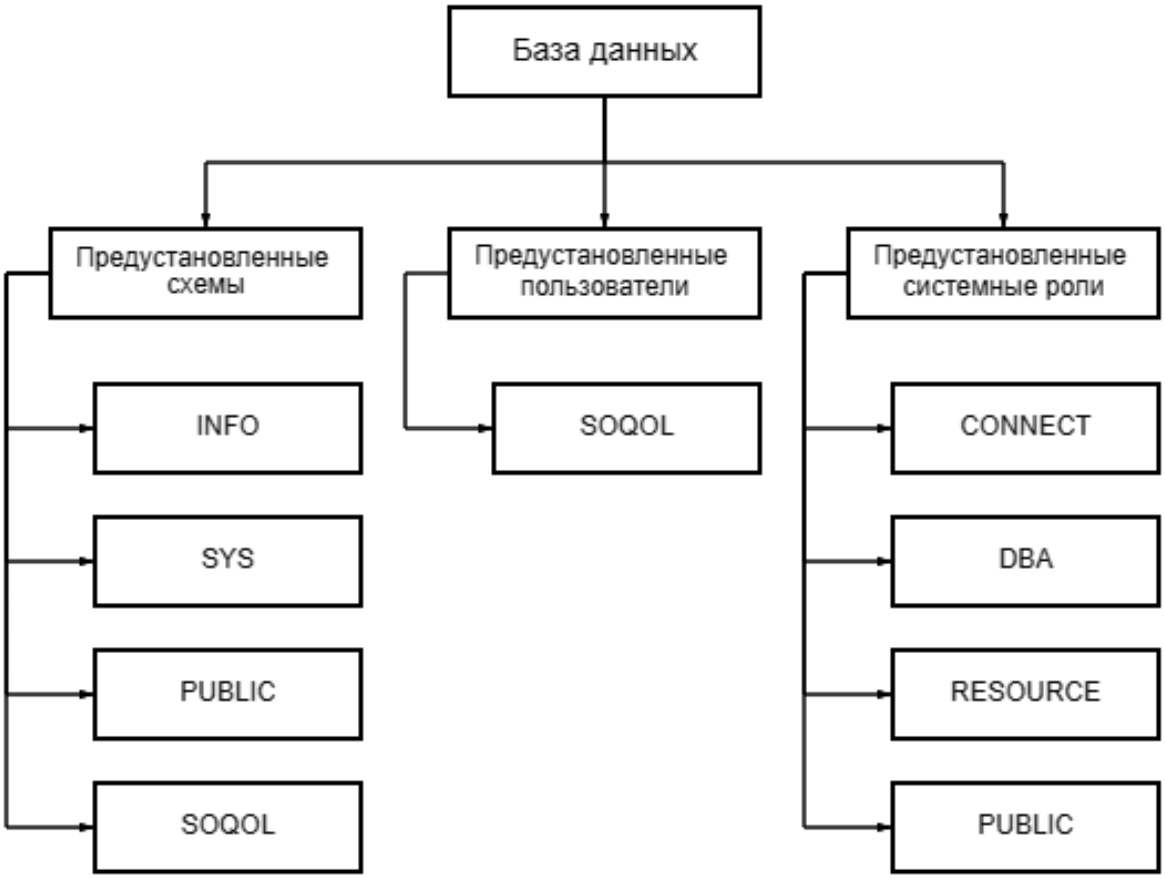


Рисунок 13 — Объекты, формируемые системой при создании каждой базы данных

Далее будут кратко рассмотрены данные объекты БД.

С командами управления базами данных и сервиса управления БД подробнее можно ознакомиться в п. 11. Работа с сущностью БД, администрированием БД — в п. 15. Администрирование баз данных в СУБД СОКОЛ.

Вернуться в оглавление

14.1. Предустановленные пользователи

В каждой базе данных при её создании система автоматически создаёт пользователя SOQOL с паролем SOQOL. Данный пользователь является предустановленным администратором базы данных и обладает системными привилегиями CREATE SESSION и DATABASE ADMIN.

Вернуться в оглавление

14.2. Предустановленные роли

В каждой базе данных при её создании система автоматически создаёт системные роли: CONNECT, DBA, RESOURCE, PUBLIC. Подробнее о каждой из них см. в п. 12.5.7. Привилегии (PRIV) и роли (ROLE).

Вернуться в оглавление

14.3. Предустановленные схемы

В каждой базе данных при её создании формируется несколько схем: INFO, SYS, PUBLIC, SOQOL.

Вернуться в оглавление

14.3.1. Схема INFO

Схема INFO — информационная схема, которая состоит из набора представлений, содержащих сведения об объектах базы данных. Входящие в эту схему системные представления предназначены для упрощения выборки метаданных о различных объектах (группах объектов) БД. В частности, эти представления используются для поддержки стандартов ODBC и JDBC.

В текущей версии СУБД СОКОЛ схема INFO включает таблицу TYPEINFO и представления SCHEMAS, TABLES, COLUMNS, PSEUDOCOLUMNS, PRIMARY_KEYS,

TABLESTATISTICS, TABLEPRIVILEGES, COLUMNPRIVILEGES, PROCEDURES, PROCEDURE_COLUMNS, SEQUENCES, JDBC_TYPEINFO, OBJ_DEFINITIONS, TABLE_TYPES, FOREIGN_KEYS, VSESSION_OPTIONS.

В дальнейшем в СУБД ЛИНТЕР СОКОЛ могут быть созданы дополнительные системные представления.

Вернуться в оглавление

14.3.2. Схема SYS

Схема SYS представляет собой схему словаря базы данных. Схема SYS включает в себя информацию, которая помогает администраторам управлять базами данных, проводить их анализ и настройки.

Вернуться в оглавление

14.3.3. Схема PUBLIC

Схема PUBLIC представляет собой схему, в которой может работать любой пользователь с системной привилегией RESOURCES CONTROL, подключившийся к базе данных.

Вернуться в оглавление

15. АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ В СУБД СОКОЛ

15.1. Копирование / архивирование базы данных и её последующее восстановление

Перед началом копирования / архивирования базы данных:

1) Выполните останов нужной базы данных командой SHUTDOWN DATABASE (подробнее см. п. 11.6. SHUTDOWN DATABASE).

2) Выполните дерегистрацию базы данных в сервисе управления базами данных командой DETACH DATABASE (подробнее см. п. 11.7. DETACH DATABASE).

3) После успешного завершения вышеуказанных шагов выполните резервное копирование или архивирование каталога базы данных (каждой базе данных соответствует свой каталог). Информации в этом каталоге достаточно для последующего полного восстановления БД.

Перед возвращением в работу базы данных из архива:

1) Определите каталог, в которую будет распакован каталог БД (это может быть любой каталог на любом диске).

2) Проверьте, что в каталоге, куда планируется произвести распаковку каталога БД, отсутствует одноимённый каталог с какой-либо базой данных. Иначе это может привести к ошибкам в дальнейшей работе.

3) Проведите распаковку архива каталога БД в выбранный каталог.

Возвращение базы данных в работу:

1) Запустите сервер (см. п. 5.1. Первоначальный запуск сервера СУБД), если он не запущен.

2) Под пользователем сервиса с системной привилегией DATABASE ADMIN подключитесь к сервису управления базами данных (см. п. 5.1. Первоначальный запуск сервера СУБД).

3) Проверьте, с какими именами уже зарегистрированы базы данных в сервисе. Это можно сделать запросом из подключения к сервису:

```
select * from SYS._DATABASES;
```

Запрос вернет информацию обо всех БД, находящихся под управлением сервиса, например:

```
DB_ID     | NAME     | PATH     | VTAG     | STATUS   | CREATION_DATE
----------+----------+----------+----------+----------+-------------------
1         | DB       | DB       | 0        | 1        | 2023-03-02 06:26:50
```

4) Выполните в сервисе управления регистрацию базы данных с уникальным именем относительно уже зарегистрированных баз данных (см. п. 11.8. ATTACH DATABASE).

Проверить успешность выполненного шага можно по обновлённому содержимому таблицы SYS._DATABASES (см. выше в п.3).

5) Выполните запуск базы данных (см. п. 11.2. STARTUP DATABASE)

После успешного выполнения вышеуказанных шагов база данных готова к работе.

6) При необходимости можно выполнить настройки опций базы данных с помощью команды ALTER DATABASE (подробнее см. п. 11.3. ALTER DATABASE).

Вернуться в оглавление

15.2. Состав каталога БД

База данных СУБД СОКОЛ (в т.ч. БД сервиса — AUXDB) представляет собой каталог с файлами. Все файлы сгруппированы в подкаталоги в соответствии с их назначением:

1) **log** — для хранения файлов основного WAL-журнала с redo-записями, который используется для:

– ведения опережающей записи всех изменений, выполненных в транзакции: все изменения, выполняемые в базе данных, сначала записываются в WAL-журнал, а затем применяются к фактическим объектам;

– отката всех действий, совершённых в транзакции при ROLLBACK;

– корректного восстановления базы данных после нештатного завершения системы и отката незафиксированных транзакций;

– логирования redo-информации по операциям СУБД при выполнении модифицирующих операций.

2) **history** — история изменения табличных данных БД. После отключения БД данные этого каталога не восстанавливаются. При восстановлении БД данные каталога history не используются. Данные каталога history используются для:

– логирования значений при выполнении модифицирующих операций над таблицами, выполненных в рамках сессии БД;

– логирования обновляемых значений при изменении существующих записей таблицы в undo-формате;

– чтения снепшота в транзакциях;

– чтения undo-информации и отката изменений до точки сохранения (savepoint) в рамках незавершённой транзакции.

3) **data** — для хранения файлов данных основных таблиц, индексов и других постоянных объектов СУБД, в т.ч. таблиц словаря данных.

4) **tmp** — для хранения временных объектов (например: временных таблиц, файлов сортировки).

Вернуться в оглавление

16. ПРОЦЕДУРНЫЙ ЯЗЫК PL/SOQOL

Процедурный язык PL/SOQOL создан для возможности выполнения сложных вычислений, требующих добавления управляющих структур к языку SQL.

Исполнение SQL-команд в PL/SOQOL происходит «бесшовно» (т.е. контекст исполнения операторов PL/SOQOL и SQL-команд один и тот же). Процедурный язык наследует все типы данных, функции и операторы, которые реализованы в SQL СУБД СОКОЛ, описанные в п. 12.5. Объекты базы данных СУБД СОКОЛ и правила их именования.

Вернуться в оглавление

16.1. Лексические единицы

Лексической единицей процедурного языка является его мельчайшая отдельная компонента:

– идентификаторы;

– зарезервированные слова;

– литерал (разных типов);

– комментарий;

– ограничители.

Все лексические элементы в тексте процедурного языка могут разделяться любым количеством пробелов, знаков табуляции и переводов строк. Такие разделители системой игнорируются.

Символ пробела не является разделителем в комментарии, строковом литерале.

Рассмотрим далее лексические элементы подробнее.

Вернуться в оглавление

16.1.1. Идентификаторы

Идентификаторы представляют собой имена следующих объектов процедурного языка PL/SOQOL:

– константы;

– курсоры;

– исключения;

– процедуры;

– функции;

– объекты БД;

– переменные.

Основные свойства идентификаторов процедурного языка:

1) Максимальный размер - 64 байта (в кодировке БД).

2) Должны быть уникальными в пределах своего контекста (например, в пределах одной процедуры или блока кода).

3) Могут содержать латинские буквы, цифры, знаки доллара («$»), подчеркивания («_») и решетки («#»).

4) Могут указываться без кавычек. В этом случае идентификатор:

– не может содержать пробельные символы;

– должен начинаться с буквы;

– символы не чувствительны к регистру (например, идентификаторы INTEGER, integeR или Integer совпадают).

5) Могут указываться в кавычках. В этом случае идентификатор:

– может содержать пробельные символы;

– может содержать русские буквы;

– начинаться может с любого допустимого символа, кроме решетки («#»);

– символы чувствительны к регистру (подразумевается в точности то, что написано, например: идентификаторы "INTEGER", "integeR" или "Integer" не совпадают).

Вернуться в оглавление

16.1.2. Зарезервированные слова

Зарезервированные слова имеют особое значение в SQL и процедурном языке. Их нежелательно использовать качестве обычных пользовательских идентификаторов, т.к. это может привести к непредсказуемому результату.

Все зарезервированные слова в процедурном языке не чувствительны к регистру. Например, любое из слов SELECT, select или Select воспринимается системой однозначно как одно ключевое слово.

Вернуться в оглавление

16.1.3. Литералы

Литералы процедурного языка включают в себя все литералы SQL, описанные в разделе 12.2. Литералы.

Вернуться в оглавление

16.1.4. Комментарии

В процедурном языке допускается указание однострочного и многострочного комментария.

Однострочный комментарий начинается с -- (двойное тире) и продолжается до конца строки.

Многострочный комментарий начинается с /* (слэш и звездочка), заканчивается на */ (звездочка и слэш), и может занимать несколько строк.

Вложенные многострочные комментарии запрещены, но один многострочный комментарий может содержать однострочные комментарии.

Количество комментариев не ограничено и они системой игнорируются.

Пример с однострочными комментариями:

```
DECLARE
 howmany    NUMBER;
 num_tables  NUMBER;
BEGIN
 -- Begin processing
 SELECT COUNT(*) INTO howmany
 FROM USER_OBJECTS
 WHERE OBJECT_TYPE = 'TABLE'; -- Check number of tables
 num_tables := howmany;      -- Compute another value
END;
/
```

Пример с многострочными комментариями:

```
DECLARE
 some_condition  BOOLEAN;
 pi            NUMBER := 3.1415926;
 radius         NUMBER := 15;
 area          NUMBER;
BEGIN
 /* Perform some simple tests and assignments */

 IF 2 + 2 = 4 THEN
   some_condition := TRUE;
 /* We expect this THEN to always be performed */
 END IF;

 /* This line computes the area of a circle using pi,
 which is the ratio between the circumference and diameter.
 After the area is computed, the result is displayed. */

 area := pi * radius*2;
END;
/
```

Вернуться в оглавление

16.2. Структура процедурного языка

Основной программной единицей процедурного языка является блок (<блок_PL_SOQOL>), в котором группируются связанные объявления и операторы.

Блок процедурного языка <блок_PL_SOQOL> может быть:

– анонимным;

– именованным.

Вернуться в оглавление

16.2.1. Анонимный блок

Анонимный блок не имеет имени и выполняется непосредственно после определения.

Анонимный блок целостно, аналогично любой SQL-команде, описанной в разделе 12.8. Справочник команд SQL, отправляется на сервер для исполнения.

Анонимный блок определяется ключевыми словами DECLARE, BEGIN, EXCEPTION и END. Указанные ключевые слова разделяют блок процедурного языка на:

– декларативную часть;

– исполняемую часть;

– блок обработки исключений.

Синтаксис анонимного блока PL/SOQOL выглядит следующим образом:

```
[DECLARE [<декларация>[, <декларация>…]]]
      BEGIN <исполняемая_часть>[<обработка_исключений>] END;

<декларация> ::= <переменная>;
           | <курсор>;
           | <имя_исключения> EXCEPTION;
           | <прагма>

<исполняемая_часть> ::= [«<<» <имя_метки> «>>»] <оператор>;[[«<<» <имя_метки> «>>»] <оператор>;…]

<оператор> ::= <имя_процедуры>
         | <оператор_присваивания>
         | <оператор_цикла>
         | <OPEN>
         | <CLOSE>
         | <FETCH_INTO>
         | <IF_THEN_ELSE>
         | <EXIT>
         | <RAISE>
         | <CONTINUE>
         | <EXECUTE_IMMEDIATE>
         | <NULL>
         | <SQL_команда>
         | <блок_PL_SOQOL>

<обработка_исключений> ::= EXCEPTION <исключения>
```

Где:

1) **<имя_метки>** — идентификатор оператора, к которому можно перейти из другого места программы. **<имя_метки>** заключается в двойные угловые скобки и располагается в начале блока.

2) **DECLARE** — указывает на начало декларативной части блока процедурного языка, содержащего объявление переменных, констант, курсоров, исключений и прагм. Перед тем, как ссылаться на эти объекты в исполняемой части блока процедурного языка и в разделе обработки исключений, необходимо их объявить. Объявления являются локальными для блока и прекращают свое существование после завершения блока. Элементы декларативной части:

– <переменная> — объявление переменной:

```
<переменная>::= <имя_переменной> <тип_данных> [<начальное_значение>];

<начальное_значение>::= <:=> <выражение>
                | DEFAULT <выражение>
```

Где:

– <имя_переменной> — имя объявляемой переменной, которое должно быть допустимым идентификатором;

– <тип_данных> — тип данных объявляемой переменной. К типам данных процедурного языка относятся типы данных SQL (16.4. Типы данных процедурного языка).

Если <начальное_значение> не задано, то переменная получит начальное значение NULL.

Для указания <начального_значения> используется:

– <:=> — оператор присваивания;

– DEFAULT — ключевое слово.

Допустимо в <выражении> указывать ранее объявленные в блоке процедурного языка переменные.

Если объявление переменной находится в блоке, то начальное значение присваивается ей каждый раз, когда управление передается этому блоку.

После объявления переменной ей также можно присвоить значение оператором присваивания (см. далее).

– <курсор> — указатель на результаты конкретного SELECT-запроса:

```
<курсор> ::= CURSOR <имя_курсора> IS <select-запрос>;
```

Курсор позволяет выполнить указанный SELECT-запрос и обрабатывать результаты этого запроса внутри блока процедурного языка.

– <имя_исключения> — позволяет присвоить имя исключению;

– <прагма> — инструкция компилятору, которую он обрабатывает во время компиляции:

```
<прагма> ::= PRAGMA EXCEPTION_INIT (<имя_исключения>, <код_ошибки>) ;
        | PRAGMA AUTONOMOUS_TRANSACTION;
```

Где:

– PRAGMA EXCEPTION_INIT — связывает указанное пользователем имя исключения с номером ошибки СУБД;

– <имя_исключения> — имя объявляемого пользователем исключения;

– <код_ошибки> — код ошибки, связываемый с указанным исключением;

– PRAGMA AUTONOMOUS_TRANSACTION — помечает процедуру как автономную, то есть независимую от основной транзакции.

Если двум PRAGMA назначить разные коды ошибок для одного и того же исключения, то более поздняя PRAGMA будет иметь приоритет (более поздняя переопределяет предыдущую PRAGMA).

3) **BEGIN** и **END** — ключевые слова, которые указывают на начало и конец (соответственно) <исполняемой_части> анонимного блока процедурного языка.

Исполняемая часть процедурного языка должна содержать хотя бы один исполняемый оператор.

4) <имя_процедуры> — имя существующей процедуры для её вызова.

Вернуться в оглавление

16.2.1.1. Операторы

16.2.1.1.1. <оператор_присваивания>

```
<оператор_присваивания> ::= <переменная> := <выражение>

<переменная> :: = <имя_переменной> | :<имя_параметра> | ?
```

Вернуться в оглавление

16.2.1.1.2. <оператор_цикла>

Блок операторов, выполняющий заданное действие до тех пор, пока условие истинно.

```
<оператор_цикла> ::= <CURSOR_FOR_LOOP>
                   | <FOR_LOOP>
                   | <WHILE_LOOP>

<CURSOR_FOR_LOOP> ::= FOR complex_ident IN (select_подзапрос)
                        LOOP <операторы> END LOOP [<имя_метки>]
                      |FOR complex_ident IN <имя_курсора>
                        LOOP <операторы> END LOOP [<имя_метки>]

<WHILE_LOOP> ::= WHILE <логическое_выражение> LOOP <операторы> END LOOP [<имя_метки>]

<FOR_LOOP> ::= FOR <имя_счетчика> IN <нижняя_граница> RANGE_OP <верхняя_граница>
                        LOOP <операторы> END LOOP [<имя_метки>]
```

Где:

– CURSOR FOR LOOP, WHILE LOOP — обозначает цикл, выполняющий блок операторов, пока условие истинно;

– FOR LOOP — обозначает цикл, выполняющий блок операторов для каждого значения из заданного диапазона или набора.

Вернуться в оглавление

16.2.1.1.3. <OPEN> и <CLOSE>

Операторы для открытия (<OPEN>) и закрытия (<CLOSE>) ранее объявленного курсора:

```
<OPEN> ::= OPEN <имя_курсора> [FOR <select_подзапрос>]

<CLOSE> ::= CLOSE <имя_курсора>
```

При указании в курсоре SELECT-подзапроса оператор OPEN связывает указанную переменную курсора с запросом. При этом указанная переменная курсора не должна иметь возвращаемого типа.

Вернуться в оглавление

16.2.1.1.4. <FETCH_INTO>

Оператор для выборки строк из ранее объявленного курсора с возможностью сохранения результирующего значения (набора значений) в указанную переменную (список переменных):

```
<FETCH_INTO> ::= FETCH <имя_курсора> INTO <имя_параметра>[, <имя_параметра>...]
```

Вернуться в оглавление

16.2.1.1.5. <IF_THEN_ELSE>

Оператор, выполняющий определенные действия в зависимости от указанных в нём условий условия:

```
<IF_THEN_ELSE> ::= IF <логическое_выражение>
            THEN statement [ statement... ]
              [ELSIF <логическое_выражение> THEN statement [ statement... ]]
                  [ELSE statement [ statement... ]] END IF
```

Вернуться в оглавление

16.2.1.1.6. <EXIT>

Оператор для выхода из цикла или процедуры, отмеченного указанной меткой:

```
<EXIT> EXIT [<имя_метки>] [WHEN <выражение>]
```

Вернуться в оглавление

16.2.1.1.7. <RAISE>

Оператор для явного вызова конкретного исключения, ранее указанного в PRAGMA:

```
<RAISE> ::= RAISE [<имя_исключения>]
```

Вернуться в оглавление

16.2.1.1.8. <CONTINUE>

Оператор для перехода к следующей (указанной) метке:

```
<CONTINUE> ::= CONTINUE [<имя_метки>] [WHEN <выражение>]
```

Вернуться в оглавление

16.2.1.1.9. <EXECUTE_IMMEDIATE>

Оператор, который выполняет указанный динамический SQL-запрос:

```
<EXECUTE_IMMEDIATE> ::= EXECUTE IMMEDIATE <SQL-запрос>
                        [USING [IN | OUT | IN OUT] <параметр>
                          [, [IN | OUT | IN OUT] <параметр>...]]
```

В операторе могут быть использованы параметры:

– IN — указывает, что параметр передается в процедуру (используется только как входной параметр). Этот параметр используется по умолчанию;

– OUT — указывает, что параметр возвращается процедурой (используется только как выходной параметр);

– IN OUT — указывает, что параметр передается процедуре и возвращается ею (используется как входной/выходной параметр).

Вернуться в оглавление

16.2.1.1.10. <NULL>

Указание NULL в исполняемой части блока процедурного языка обозначает отсутствие операции. Данный оператор передает управление следующему оператору.

Вернуться в оглавление

16.2.1.1.11. <SQL_команды>

В процедурном языке в качестве оператора можно использовать часть команд SQL, описанных в том числе в п. 12.8. Справочник команд SQL:

```
<SQL_команды> ::= SELECT
                | DELETE
                | COMMIT
                | ROLLBACK
                | SAVEPOINT
                | RELEASE SAVEPOINT
                | INSERT INTO TABLE
                | UPDATE TABLE
                | SET TRANSACTION
                | START TRANSACTION
```

Команды, изменяющие структуру объектов БД (DDL), описанные также в п. 12.8. Справочник команд SQL могут быть также использованы в процедурном языке, но только с помощью оператора EXECUTE IMMEDIATE. Это связано с тем, что код блока процедурного языка работает со структурой объекта на начало исполнения этого блока PL/SOQOL.

Вернуться в оглавление

16.2.1.1.12. <обработка_исключений>

Часть основного блока процедурного языка для обработки ранее объявленных исключений всегда начинается с ключевого слова EXCEPTION:

```
EXCEPTION <исключение>[, <исключение>…] [<новое_исключение>]

<новое_исключение> ::= WHEN OTHERS THEN <блок_заявлений>
```

Где <исключение> – ранее объявленное исключение.

В этой части определяются действия, связанные с обработкой ошибок сервера и пользовательских исключений, возникших в секции выполняемого кода.

При возникновении исключения нормальное выполнение блока останавливается, и управление передается соответствующему обработчику исключений. После завершения обработчика исключений выполнение продолжается с инструкцией, следующей за блоком.

Если в текущем блоке нет обработчика для вызванного исключения, управление переходит к блоку, содержащему исключение. Этот процесс повторяется до тех пор, пока не будет найден обработчик исключений или не закончатся окружающие блоки. Если блоках процедурного языка не найден обработчик исключения, то выполнение останавливается с возвращением ошибки.

Вернуться в оглавление

16.2.2. Именованный блок

Именованный блок имеет имя и может быть вызван из других частей программы (например, других блоков процедурного языка).

Именованный блок процедурного языка, также как и анонимный, содержит:

– декларативную часть;

– исполняемую часть;

– часть, обрабатывающую исключения.

Верхнеуровнево синтаксис анонимного блока PL/SOQOL выглядит следующим образом:

```
CREATE [OR REPLACE] PROCEDURE <параметры_процедуры>
    [[<декларация>[, <декларация>…]]]
     BEGIN <исполняемая_часть>[<обработка_исключений>]END;
```

Где CREATE [OR REPLACE] PROCEDURE <параметры_процедуры> — создание процедуры (подробнее см. п. 12.8.1.8. CREATE PROCEDURE).

Реализация остальных частей именованного блока, за исключением декларативной части, совпадает с реализацией анонимного блока (см. п. 16.2.1. Анонимный блок). Декларативная часть отличается тем, что начинается сразу с объявления переменной, исключения и т.д., без указания ключевого слова DECLARE.

Вернуться в оглавление

16.3. Выражения

Далее будут рассмотрены операторы и функции, допустимые для указания в выражениях процедурного языка.

Вернуться в оглавление

16.3.1. Операция конкатенации

Строковый оператор конкатенации в процедурном языке поддерживается тот же, что и в языке SQL (см. п. 12.3.2. Строковые операции).

Вернуться в оглавление

16.3.2. Логические операции

Логические операции в процедурном языке поддерживаются те же, что и в языке SQL (см. п. 12.3.3. Логические операции).

Вернуться в оглавление

16.3.3. Операции сравнения

Операции сравнения и их символы в процедурном языке поддерживаются те же, что и в языке SQL (см. п. 12.3.4. Операции сравнения).

Вернуться в оглавление

16.3.4. Приоритет операций

Сводную информацию о приоритетах операций см. в п. 12.3. Операции.

Вернуться в оглавление

16.3.5. Функции SQL в выражениях

В выражениях процедурного языка можно использовать все функции SQL, кроме:

– агрегатных функций;

– функции кодирования DECODE;

– функции преобразования BIN_TO_NUM;

– функции сортировки COLLATION;

– функции NVL.

Вернуться в оглавление

16.4. Типы данных процедурного языка

К типам данных процедурного языка относятся типы данных SQL. И вся информация о типах и подтипах данных, правилах сравнения типов данных, преобразовании данных, литералах и моделях форматов относится как к SQL, так и к процедурного языка (подробнее см. п. 12. РУКОВОДСТВО ПО SQL).

Если объявленная переменная имеет тип данных BOOLEAN, то ей можно присвоить только значения TRUE, FALSE, NULL. В других случаях будет возвращена ошибка.

Вернуться в оглавление

16.5. Пример использования процедурного языка в СУБД СОКОЛ

Допустим, есть две таблицы:

– popular_names — с рейтингом ста популярных мужских и женских имён;

– persons — с данными реальных людей.

Необходимо проанализировать имена реальных людей и сформировать:

– список фамилий людей с популярными именами;

– таблицу-отчёт results с двумя столбцами: рейтинг имени и информация о каждом человеке с популярным именем в заданном формате: First name: '<имя>', Last

name: '<фамилия>', Age: '<возраст>'. Предусмотреть возможность формирования отчета по женщинам и мужчинам отдельно.

Данные в таблицу popular_names вносятся единожды, а в таблице persons периодически обновляются. Отчёт по носителям популярного имени нужно выполнять регулярно, допустим, раз в месяц.

Для решения данной задачи воспользуемся процедурным языком.

Создадим и заполним вышеозвученные таблицы (полный пример в каталоге с дистрибутивом: файлы examples/popular_names/1_popular_names.sql и examples/popular_names/2_persons.sql). После успешно выполненных команд получаем таблицу popular_names следующего вида:

```
 NUM     | NAME     | GENDER
----------+-----------+-------------
 1       | James    | M
 2       | Robert   | M
 3       | John     | M
 4       | Michael  | M
 5       | David    | M
 6       | William  | M
 7       | Richard  | M
 8       | Joseph   | M
...
 98      | Mason    | M
 99      | Philip   | M
 100     | Louis    | M
 1       | Mary     | F
 2       | Patricia | F
 3       | Jennifer | F
...
 98      | Kayla    | F
 99      | Alexis   | F
 100     | Lori     | F
```

В столбце NUM таблицы popular_names указан рейтинг имени среди 100 самых популярных имён (где 1 — самое популярное имя).

И получаем таблицу с данными реальных людей persons следующего вида:

```
FIRST_NAME      | LAST_NAME     | AGE     | GENDER
-----------------+----------------+----------+-------------
 Thomas         | Allen         | 25      | M
 Ronald         | Mason         | 51      | M
 Maria          | Lloyd         | 24      | F
 Martha         | Kelly         | 19      | F
 Susan          | James         | 44      | F
 Mark           | Harris        | 48      | M
 John           | Gray          | 35      | M
```

```
...
```

Далее создадим таблицу-отчет results:

```
create table results(num integer, info varchar(4000));
```

Теперь для поиска носителей популярных имён необходимо сравнить данные таблицы persons с именами в таблице popular_names. Для этого создадим процедуру, которая при вызове будет:

1) Перед заполнением таблицы results очищать её — это позволит в таблице иметь всегда актуальные данные.

2) Проходить по таблице persons и для каждой записи определять, есть ли такое имя среди популярных имён в таблице popular_names.

При нахождении имени в таблице popular_names:

– добавлять фамилию носителя популярного имени к списку, хранящемуся в ранее объявленной переменной;

– добавлять в таблицу results рейтинг и информацию о каждом носителе имени в заданном формате.

Создадим процедуру:

```
create or replace procedure get_popular_persons(pers_gender varchar(1), up boolean, result_str out varchar(4000))
as
   pers_fname varchar(100);
   pers_lname varchar(100);
   pers_age   integer;

   nrank integer;

   str varchar(4000);
   next boolean;

   cursor curs_pers is select first_name, last_name, age from persons order by age;
   cursor curs_names is select distinct num from popular_names where name = pers_fname and gender = pers_gender;

   no_data  EXCEPTION;
   PRAGMA   EXCEPTION_INIT(no_data, -1403);
begin
   delete from results;
   result_str := 'Popular persons names: ';
   next := false;
   open curs_pers;
   fetch curs_pers into pers_fname, pers_lname, pers_age;
   while (curs_pers%found)
   loop
      open curs_names;
      fetch curs_names into nrank;
      while (curs_names%found)
      loop
         if next then
           result_str := result_str || ', ';
         else
           next := true;
         end if;
         result_str := result_str ||  pers_lname;
         str := 'First name: '|| pers_fname || ', Last name: ' || pers_lname || ', Age: ' || pers_age;
         if up then
            str := upper(str);
         end if;
         insert into results values (nrank, str);
         fetch curs_names into nrank;
      end loop;
      close curs_names;
      fetch curs_pers into pers_fname, pers_lname, pers_age;
   end loop;
   close curs_pers;
   exception
      when no_data then
        result_str := 'No data found';
      when others then
       result_str := 'Something went wrong';
end;
```

```
/
```

Объявим переменную, в которую можно будет сохранить фамилии носителей мужских популярных имён:

```
variable male_popular varchar (4000);
```

Теперь вызовем процедуру get_popular_persons с поиском популярных имён среди мужчин и выведем их фамилии в объявленную переменную male_popular:

```
call get_popular_persons('M', false, :male_popular);

print male_popular;


MALE_POPULAR
--------------------
 Popular persons names: Gibson, Allen , Gray, Harris, Mason, Armstrong, Foster
```

При вызове процедуры get_popular_persons в таблицу results были добавлены данные носителей мужских популярных имён. Выполним запрос данных таблицы results и посмотрим данные мужчин, чьи имена из таблицы persons были найдены среди популярных в таблице popular_names:

```
select * from results order by num;


 NUM     | INFO
----------+-----------
 3       | First name: John, Last name: Gray, Age: 35
 9       | First name: Thomas, Last name: Allen, Age: 25
 15      | First name: Mark, Last name: Harris, Age: 48
 26      | First name: Ronald, Last name: Mason, Age: 51
 32      | First name: Gary, Last name: Gibson, Age: 22
 92      | First name: Roy, Last name: Armstrong, Age: 55
 93      | First name: Vincent, Last name: Foster, Age: 62
```

Для женских имён выполняются аналогичные действия.

В полученной результирующей таблице теперь находятся данные только женщин, т.к. вызванная процедура в начале своей работы сначала очистила таблицу, а потом добавила данные в соответствии с новыми условиями (полный пример создания и работы с процедурой get_popular_persons в каталоге с дистрибутивом: файл examples/popular_names/3_get_pop_pers.sql).

Вернуться в оглавление

17. УТИЛИТЫ[CV]

В СУБД СОКОЛ реализовано несколько утилит, позволяющих выполнять такие задачи как:

– проверять сервер на готовность к работе;

– исполнять запросы к СУБД в интерактивном и пакетном режиме;

– загружать данные в таблицы БД из файлов операционной системы;

– экспортировать данные из БД в файл операционной системы с помощью SQL-запроса и т.д.

Далее подробнее будет рассмотрена каждая утилита и её опции.

Вернуться в оглавление

17.1. vsql_server_wait

Утилита ожидания запуска сервера. На протяжении заданного времени утилита проверяет, запущен ли сервер. Если сервер был запущен или запустился на протяжении заданного времени, то утилита завершается с успешным кодом завершения (значение 0). Иначе, по истечении установленного таймаута, утилита возвращает неуспешный код завершения.

Рекомендуется использовать утилиту в скриптах, где перед отправкой команд серверу требуется удостовериться, что сервер запущен и отвечает на запросы.

В утилите реализованы следующие опции:

1) -h <connect-string>, --connect-string=<connect-string> — указывает строку подключения (формат указан ниже) к серверу, который проходит проверку запуска.

Формат строки подключения к серверу:

```
[<protocol>://]<user>:[<password>][@<host>[:<port>]][/]
```

Подробнее о каждой части строки подключения см. в п. 6. О строке подключения для утилит).

2) -t <timeout>, --timeout=<timeout> — указывает время ожидания запуска сервера в секундах. Значение по умолчанию — 300 секунд.

3) -v, --version — выводит версию утилиты.

4) -?, --help — выводит текст помощи.

Примеры:

Команда ожидания запуска сервера на хосте localhost и порте 2060 в течение 300 секунд:

```
vsql_server_wait -h SOQOL:SOQOL@
```

Команда для ожидания ответа сервера на хосте localhost и порте 2060 в течение явно заданных 20 секунд:

```
vsql_server_wait -h SOQOL:SOQOL@ -t 20
```

Вывод версии утилиты:

```
vsql_server_wait -v
```

Вывод текста помощи утилиты:

```
vsql_server_wait -?
```

Вернуться в оглавление

17.2. vsql_console

Утилита для исполнения запросов к СУБД в интерактивном или пакетном режиме.

В утилите реализованы следующие опции:

1) -h <connect-string>, --connect-string=<connect-string> — указывает строку подключения (формат указан ниже) к серверу, с которым будет выполняться взаимодействие.

Формат строки подключения к серверу:

```
[<protocol>://]<user>:[<password>][@<host>[:<port>]][/][<dbname>][?<parameter>=<value>]
```

Подробнее о каждой части строки подключения см. в п. 6. О строке подключения для утилит).

2) --codepage=<cp> — указывает кодировку данных во входных и выходных файлах, где <cp> может принимать значение UTF-8 или CP866, CP1251). Значение по

умолчанию для параметра зависит от локали консоли (русская версия Windows — CP866, Linux — UTF-8).

3) -i <sql-file>, --in-sql-file=<sql-file> — указывает файл с SQL-командами для исполнения в пакетном режиме.

4) -o <yaml-file>, --out-file=<yaml-file> — указывает путь к файлу, в который сохраняется структурированная информация о выполненных запросах и ответов на них в формате YAML. При каждом новом запуске vsql_console с данной опцией содержимое указанного YAML-файла замещается новым данными.

5) -v, --version — выводит версию утилиты.

6) -?, --help — выводит текст помощи.

Примеры:

Исполнение заданного пакетного файла requestSQL.txt с указанием его кодировки CP866 и последующим выводом данных на экран:

```
vsql_console
    -h soqol://SOQOL:SOQOL@localhost:2060/DB
       --codepage=CP866
         -i requestSQL.txt
```

Исполнение заданного пакетного файла SQL.txt с указанием его кодировки CP1251 и настройка структурированного вывода в YAML-файл out.yml:

```
vsql_console
  -h soqol://SOQOL:SOQOL@localhost:2060/DB
     --codepage=CP1251
       -i SQL.txt
           -o out.yml
```

Вывод версии утилиты и вывод текста помощи такой же, как в примере vsql_server_wait (см. п. 17.1 vsql_server_wait).

Вернуться в оглавление

17.3. vsql_copy

Утилита импорта / экспорта данных:

– в таблицу БД из файла CSV-формата;

– в файл CSV-формата из таблицы БД.

Для выгрузки из БД можно указать как таблицу, так и SQL-запрос для выборки данных.

В утилите реализованы следующие опции:

1) -h <connect-string>, --connect-string=<connect-string> — указывает строку подключения (формат указан ниже) к серверу, с которым будет выполняться взаимодействие.

Формат строки подключения к серверу:

```
[<protocol>://]<user>:[<password>][@<host>[:<port>]][/][<dbname>][?<parameter>=<value>]
```

Подробнее о каждой части строки подключения см. в п. 6. О строке подключения для утилит).

2) --export=<export-file> — указывает файл для записи данных из БД. Если значение файла равно «-» (дефис без кавычек, т.е. опция будет выглядеть как --export=-), то данные записываются в стандартный вывод.

3) --import=<file-path> — указывает файл с данными для импорта в БД. Если значение файла равно «-» (дефис без кавычек, т.е. опция будет выглядеть как --import=-), то данные считываются из стандартного ввода. Без указания опции -e (--headers) число столбцов значений в импортируемом файле должно быть равно числу столбцов таблицы, в которую импортируются данные. С опцией -e (--headers) значения будут вставляться только в указанные в первой строке CSV-файла столбцы таблицы (обратите внимание, что имена столбцов должны быть указаны в верхнем регистре, если при создании таблицы они были определены без кавычек).

4) -s <sql-query>, --sql=<sql-query> — указывает SQL-запрос.

Здесь <sql-query> — строка SQL-запроса (указывается в кавычках, например: "select * from PHONE where NUMBER = 89876543216;"), извлекающим экспортируемый табличный результат. Если в запросе необходимо указать кавычки, то их нужно экранировать символом «\» (например: "select * from \"Номера\";").

5) --sql-file=<SQL-file> — указывает файл с SQL-запросом, извлекающим экспортируемый табличный результат.

6) -e, --headers — указывает, что первая строка входного CSV-файла содержит заголовки столбцов.

7) -t <table-name>, --table=<table-name> — указывает имя таблицы базы данных для вставки данных или выборки для операций импорта и экспорта. Запрос для получения данных формируется автоматически. Доступно только для CSV-файла.

8) -u <delimiter>, --field-delimiter=<delimiter> — устанавливает разделитель полей, значение <delimiter> задается в кавычках (например, для разделителя "," опция будет выглядеть как -u ",").

9) -v, --version — вывод информации о версии утилиты.

10) -?, --help — вывод текста помощи утилиты.

Примеры:

Экспорт данных из базы данных DB полученных SQL-запросом с указанием файла out.csv для сохранения данных:

```
vsql_copy
  -h soqol://SOQOL:SOQOL@localhost:2060/DB?charset=cp1251
    --export=out.csv
      -s "select * from PHONE where NUMBER=89876543216"
```

Экспорт данных из таблицы WORKS базы данных DB с указанием файла works.txt для сохранения данных:

```
vsql_copy
  -h soqol://SOQOL:SOQOL@localhost:2060/DB?charset=cp1251
    -t WORKS
      --export=works.txt
```

Вывод версии утилиты и вывод текста помощи такой же, как в примере vsql_server_wait (см. п. 17.1. vsql_server_wait).

Вернуться в оглавление

18. КОМАНДЫ КОНСОЛЬНОЙ УТИЛИТЫ VSQL_CONSOLE

Все команды из разделов 11. Работа с сущностью БД и 12.8. Справочник команд SQL поддерживаются консольной утилитой vsql_console. Кроме них утилита поддерживает ещё ряд дополнительных команд, описание которых изложено далее в этом разделе.

Вернуться в оглавление

18.1. Команды подключения

Группа команд, используемая для подключения к сервису управления базами данных или к базе данных.

Все команды этой группы могут быть выполнены как из подключения к сервису, так и из подключения к базе данных.

Вернуться в оглавление

18.1.1. Подключение к базе данных

Команда выполняет подключение пользователя к запущенной базе данных.

Выполнить подключение к запущенной базе данных может любой пользователь базы данных с системной привилегией CREATE SESSION (подробнее см. п. 12.5.7. Привилегии (PRIV) и роли (ROLE)).

Синтаксис:

```
CONNECT [TO <имя_базы_данных>] AS <имя_пользователя>/<пароль_пользователя>[;]
```

Где:

– <имя_базы_данных> — имя базы данных, к которой необходимо подключиться. При отсутствии конструкции TO <имя_базы_данных> будет выполнено подключение к текущей базе данных указанного пользователя;

– <имя_пользователя> — имя пользователя базы данных, под которым необходимо подключиться к базе данных. Символы имени пользователя, указанные

в кавычках, рассматриваются системой как регистрозависимые. Если имя пользователя указывается без кавычек, то его символы приводятся системой к верхнему регистру;

– <пароль_пользователя> — пароль подключаемого пользователя. Допустимо указание пароля как в апострофах, так и без них, при этом символы пароля рассматриваются системой как регистрозависимые.

Вернуться в оглавление

18.1.2. Подключение к сервису управления БД

Команда выполняет подключение к сервису управления базами данных.

Выполнить подключение к сервису управления базами данных может любой пользователь сервиса с системной привилегией CREATE SESSION (подробнее см. п.12.5.7. Привилегии (PRIV) и роли (ROLE)).

Синтаксис:

```
CONNECT SERVICE AS <имя_пользователя>/<пароль_пользователя>[;]
```

Где:

– <имя_пользователя> — имя пользователя сервиса, под которым необходимо подключиться к сервису;

– <пароль_пользователя> — пароль подключаемого пользователя. Допустимо указание пароля как в апострофах, так и без них.

Вернуться в оглавление

18.1.3. Подключение через URL к БД и к сервису управления БД

Команда выполняет подключение к сервису или базе данных с помощью строки подключения.

Выполнить подключение к сервису или базе данных с помощью строки подключения может:

– для подключения к сервису — любой пользователь сервиса с системной привилегией CREATE SESSION;

– для подключения к базе данных — любой пользователь базы данных с системной привилегией CREATE SESSION.

Подробнее о привилегиях и ролях см. п.12.5.7. Привилегии (PRIV) и роли (ROLE).

Синтаксис:

```
CONNECT URL '<строка_подключения>'[;]
```

Где:

– <строка_подключения> — строка подключения в нужном формате (подробнее о форматах см. п. 9.2.2. Строка подключения).

Вернуться в оглавление

18.2. Выход из утилиты

Команда выхода из утилиты vsql_console.

Синтаксис:

```
QUIT[;]
```

Вернуться в оглавление

18.3. Объявление переменной

Команда объявляет переменную, которая представляет собой именованную ячейку памяти, предназначенную для сохранения некоторого значения.

Синтаксис:

```
VARIABLE <имя_переменной> <тип_переменной> [= <литерал>] [;]

<тип_параметра> = NUMBER
            | CHAR
            | BOOLEAN
            | VARCHAR
            | BINARY
            | VARBINARY
            | DATE
            | TIMESTAMP

<литерал> = строковый литерал
        | BINARY литерал
        | числовой литерал
        | логический литерал
        | NULL
```

При попытке объявления переменной с именем уже существующей ранее объявленная переменная будет удалена и создана новая на основе описания, указанного в команде.

При попытке объявления переменной без указания <типа_параметра> будет возвращена ошибка.

При объявлении переменной можно указать значение параметра <литерал>. Это значение будет изначально присвоено переменной. Если значение <литерал> не задано при объявлении переменной, то она приобретет значение NULL.

Переменная может использоваться в процедурном языке и в SQL-командах там, где может быть использовано значение соответствующего типа (кроме конструкций LIMIT <выражение>, FETCH FIRST <выражение> в команде SELECT). При упоминании в процедурном языке и в SQL-командах переменная указывается следующим образом:

```
:<имя_переменной>
```

В переменную можно сохранить полученное значение. Такой пример был рассмотрен в примере для подраздела 12.8.9. Конструкция RETURNING:

```
variable TAB number;

INSERT INTO WORKER (NAME, LASTNAME, AGE, GENDER) values ('Игорь', 'Иванов', 53, 'м') returning TN into :TAB;
```

Чтобы определить значение переменной, воспользуйтесь командой:

```
print TAB;
```

Команда распечатает значение переменной:

```
TAB
----------
 4
```

Полученное значение можно использовать в других командах, например, при вставке данных:

```
insert into FORREPORTCARD (TN) values (:TAB);
```

Значение в именованном параметре хранится до тех пор, пока не будет заменено другим значением. По завершении работы консольной утилиты все определенные переменные удаляются.

Вернуться в оглавление

18.4. Вывод в консоль значения из переменной

Команда вывода в консоль значения, сохранённого в переменной.

Синтаксис:

```
PRINT <имя_параметра>[;]
```

При попытке вывода на печать значения несуществующей переменной будет возвращена ошибка.

Вернуться в оглавление

19. КОДЫ ОШИБОК СУБД[CV]

Вызов любой операции завершается успешно или приводит к ошибке. Всем сообщениям об ошибках присвоены коды ошибок.

В приложении 5 перечислены коды ошибок, определённые в текущей версии СУБД, с которыми может столкнуться пользователь. В некоторых случаях сервер сообщает имя объекта базы данных (таблица, столбец таблицы, тип данных или ограничение), связанного с ошибкой. Например, имя уникального ограничения, которое не позволяет добавить в таблицу несоответствующее значение.

Коды ошибок и их описания в разных версиях СУБД могут отличаться.

Вернуться в оглавление

ПРИЛОЖЕНИЕ 1

СПИСОК РЕАЛИЗОВАННЫХ ФУНКЦИЙ ODBC-ДРАЙВЕРА

Метод	**Реализация**
SQLAllocHandle	да
SQLConnect	да
SQLDriverConnect	да
SQLPrepare	да
SQLExecute	да
SQLExecDirect	да
SQLParamData	да
SQLPutData	да
SQLNumResultCols	да
SQLFetch	да
SQLGetData	да
SQLMoreResults	да
SQLFreeStmt	да
SQLEndTran	да
SQLDisconnect	да
SQLAllocEnv	да
SQLFreeEnv	да
SQLAllocConnect	да
SQLFreeConnect	да
SQLAllocStmt	да
SQLGetConnectOption	да
SQLSetConnectOption	да
SQLError	да

ПРИЛОЖЕНИЕ 2

СПИСОК РЕАЛИЗОВАННЫХ ФУНКЦИЙ JDBC-ДРАЙВЕРА

Класс интерфейса java.sql	**Метод**	**Реализация**
CallableStatement		
	BigDecimal getBigDecimal(int parameterIndex)	да
	BigDecimal getBigDecimal(int parameterIndex, int scale)	да
	boolean getBoolean(int parameterIndex)	да
	byte getByte(int parameterIndex)	да
	byte[] getBytes(int parameterIndex)	да
	Clob getClob(String parameterName)	да
	double getDouble(int parameterIndex)	да
	float getFloat(int parameterIndex)	да
	int getInt(int parameterIndex)	да
	long getLong(int parameterIndex)	да
	short getShort(int parameterIndex)	да
	String getString(int parameterIndex)	да
	Time getTime(int parameterIndex)	да
	Timestamp getTimestamp(int parameterIndex)	да
	void registerOutParameter(int parameterIndex, int sqlType)	да
	void registerOutParameter(int parameterIndex, int sqlType, int scale)	да
	boolean wasNull()	да
Connection		
	void close()	да
	void commit()	да
	Statement createStatement()	да
	Statement createStatement(int resultSetType, int resultSetConcurrency)	да/нет

Класс интерфейса java.sql	Метод	Реализация
	Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	да/нет
	boolean getAutoCommit()	да
	int getHoldability()	да/нет
	DatabaseMetaData getMetaData()	да/нет
	int getTransactionIsolation()	да/нет
	boolean isClosed()	да
	boolean isReadOnly()	да/нет
	String nativeSQL(String sql)	да/нет
	CallableStatement prepareCall(String sql)	да
	CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)	да/нет
	CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	да/нет
	PreparedStatement prepareStatement(String sql)	да
	PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	да/нет
	PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	да/нет
	void releaseSavepoint(Savepoint savepoint)	да/нет
	void rollback()	да
	void setAutoCommit(boolean autoCommit)	да
DatabaseMetaData		
	boolean allProceduresAreCallable()	да
	boolean allTablesAreSelectable()	да
	boolean autoCommitFailureClosesAllResultSets()	да
	boolean dataDefinitionCausesTransactionCommit()	да
	boolean dataDefinitionIgnoredInTransactions()	да

Класс интерфейса java.sql	Метод	Реализация
	boolean deletesAreDetected(int type)	да
	boolean doesMaxRowSizeIncludeBlobs()	да
	boolean generatedKeyAlwaysReturned()	да
	Connection getConnection()	да
	int getDatabaseMajorVersion()	да
	int getDatabaseMinorVersion()	да
	String getDatabaseProductName()	да
	String getDatabaseProductVersion()	да
	int getDefaultTransactionIsolation()	да
	int getDriverMajorVersion()	да
	int getDriverMinorVersion()	да
	String getDriverName()	да
	String getDriverVersion()	да
	String getIdentifierQuoteString()	да
	int getJDBCMajorVersion()	да
	int getJDBCMinorVersion()	да
	int getMaxBinaryLiteralLength()	да
	int getMaxCharLiteralLength()	да
	int getMaxColumnNameLength()	да
	int getMaxTableNameLength()	да
	String getUserName()	да
	boolean nullPlusNonNullIsNull()	да
	boolean nullsAreSortedAtEnd()	да
	boolean nullsAreSortedAtStart()	да
	boolean nullsAreSortedHigh()	да
	boolean nullsAreSortedLow()	да
	boolean storesLowerCaseIdentifiers()	да

Класс интерфейса java.sql	Метод	Реализация
	boolean storesLowerCaseQuotedIdentifiers()	да
	boolean storesMixedCaseIdentifiers()	да
	boolean storesMixedCaseQuotedIdentifiers()	да
	boolean storesUpperCaseIdentifiers()	да
	boolean storesUpperCaseQuotedIdentifiers()	да
	boolean supportsANSI92EntryLevelSQL()	да
	boolean supportsANSI92FullSQL()	да
	boolean supportsANSI92IntermediateSQL()	да
	boolean supportsColumnAliasing()	да
	boolean supportsCorrelatedSubqueries()	да
	boolean supportsDataManipulationTransactionsOnly()	да
	boolean supportsFullOuterJoins()	да
	boolean supportsGroupBy()	да
	boolean supportsGroupByBeyondSelect()	да
	boolean supportsMinimumSQLGrammar()	да
	boolean supportsMixedCaseIdentifiers()	да
	boolean supportsMixedCaseQuotedIdentifiers()	да
	boolean supportsOuterJoins()	да
	boolean supportsStoredProcedures()	да
	boolean supportsSubqueriesInExists()	да
	boolean supportsTransactionIsolationLevel(int level)	да
	boolean supportsTransactions()	да
	boolean supportsUnion()	да
	boolean supportsUnionAll()	да
Driver		
	boolean acceptsURL(String url)	да

Класс интерфейса java.sql	Метод	Реализация
	Connection connect(String url, Properties info)	да
	int getMajorVersion()	да
	int getMinorVersion()	да
	Logger getParentLogger()	да
	DriverPropertyInfo[] getPropertyInfo(String url, Properties info)	да
	boolean jdbcCompliant()	да
ParameterMetaData		
	int getParameterCount()	да
	int getParameterMode(int param)	да
	int getParameterType(int param)	да
	int getPrecision(int param)	да
	int getScale(int param)	да
PreparedStatement		
	void clearParameters()	да
	boolean execute()	да
	ResultSet executeQuery()	да
	int executeUpdate()	да
	ParameterMetaData getParameterMetaData()	да
	void setBigDecimal(int parameterIndex, BigDecimal x)	да
	void setBinaryStream(int parameterIndex, InputStream x)	да
	void setBinaryStream(int parameterIndex, InputStream x, int length)	да
	void setBinaryStream(int parameterIndex, InputStream x, long length)	да
	void setBlob(int parameterIndex, Blob x)	да

Класс интерфейса java.sql	Метод	Реализация
	void setBlob(int parameterIndex, InputStream inputStream)	да
	void setBlob(int parameterIndex, InputStream inputStream, long length)	да
	void setBoolean(int parameterIndex, boolean x)	да
	void setByte(int parameterIndex, byte x)	да
	void setBytes(int parameterIndex, byte[] x)	да
	void setDate(int parameterIndex, Date x)	да
	void setDate(int parameterIndex, Date x, Calendar cal)	да
	void setDouble(int parameterIndex, double x)	да
	void setFloat(int parameterIndex, float x)	да
	void setInt(int parameterIndex, int x)	да
	void setLong(int parameterIndex, long x)	да
	void setNull(int parameterIndex, int sqlType)	да
	void setShort(int parameterIndex, short x)	да
	void setString(int parameterIndex, String x)	да
	void setTime(int parameterIndex, Time x)	да
	void setTimestamp(int parameterIndex, Timestamp x)	да
ResultSet		
	void close()	да
	int findColumn(String columnLabel)	да
	BigDecimal getBigDecimal(int columnIndex)	да
	BigDecimal getBigDecimal(int columnIndex, int scale)	да
	BigDecimal getBigDecimal(String columnLabel)	да
	BigDecimal getBigDecimal(String columnLabel, int scale)	да

Класс интерфейса java.sql	Метод	Реализация
	InputStream getBinaryStream(int columnIndex)	да
	InputStream getBinaryStream(String columnLabel)	да
	boolean getBoolean(int columnIndex)	да
	boolean getBoolean(String columnLabel)	да
	byte getByte(int columnIndex)	да
	byte getByte(String columnLabel)	да
	byte[] getBytes(int columnIndex)	да
	byte[] getBytes(String columnLabel)	да
	Date getDate(int columnIndex)	да
	Date getDate(String columnLabel)	да
	double getDouble(int columnIndex)	да
	double getDouble(String columnLabel)	да
	int getFetchDirection()	да
	int getFetchSize()	да
	float getFloat(int columnIndex)	да
	float getFloat(String columnLabel)	да
	int getInt(int columnIndex)	да
	int getInt(String columnLabel)	да
	long getLong(int columnIndex)	да
	long getLong(String columnLabel)	да
	ResultSetMetaData getMetaData()	да
	Object getObject(int columnIndex)	да
	Object getObject(String columnLabel)	да
	short getShort(int columnIndex)	да
	short getShort(String columnLabel)	да
	Statement getStatement()	да
	String getString(int columnIndex)	да

Класс интерфейса java.sql	Метод	Реализация
	String getString(String columnLabel)	да
	Time getTime(int columnIndex)	да
	Time getTime(String columnLabel)	да
	Timestamp getTimestamp(int columnIndex)	да
	Timestamp getTimestamp(String columnLabel)	да
	int getType()	да
	boolean isClosed()	да
	boolean next()	да
	void setFetchDirection(int direction)	да
	void setFetchSize(int rows)	да
	boolean wasNull()	да
ResultSetMetaData		
	int getColumnCount()	да
	String getColumnLabel(int column)	да
	String getColumnName(int column)	да
	int getColumnType(int column)	да
	int getPrecision(int column)	да
	int getScale(int column)	да
	boolean isCaseSensitive(int column)	да
Statement		
	void close()	да
	boolean execute(String sql)	да
	ResultSet executeQuery(String sql)	да
	int executeUpdate(String sql)	да
	Connection getConnection()	да
	int getFetchDirection()	да
	int getFetchSize()	да

Класс интерфейса java.sql	Метод	Реализация
	int getMaxRows()	да
	ResultSet getResultSet()	да
	int getUpdateCount()	да
	boolean isClosed()	да
	void setMaxRows(int max)	да

ПРИЛОЖЕНИЕ 3

КОДИРОВКИ И ПРАВИЛА СРАВНЕНИЯ, ИСПОЛЬЗУЕМЫЕ В СУБД СОКОЛ

Наименование кодировки	**Имя правила сравнения, применимое к кодировке**	**Комментарий**
UTF-8		
	BINARY, UCA_UTF8_LX_CS, UCA_UTF8_LX_CI	BINARY — по умолчанию
CP1251		
	BINARY, WIN1251_CS, WIN1251_CI	BINARY — по умолчанию

ПРИЛОЖЕНИЕ 4

ПРАВИЛА НЕЯВНОГО ПРЕОБРАЗОВАНИЯ

Целевой тип данных	Правила преобразования
Из типа CHAR (size), VARCHAR (size) в целевой тип:	
CHAR (size), VARCHAR (size)	Каждый символ исходного значения будет преобразован в соответствии с кодировкой целевого типа. Если длина в байтах полученного значения меньше или равна size в байтах целевого типа, то значение будет преобразовано без изменений. В ином случае преобразование завершится ошибкой
CLOB	Исходное значение будет преобразовано в аналогичное значение в кодировке целевого типа
DATE	Если исходное значение соответствует одному из форматов даты, указанному далее, то оно будет преобразовано без изменения значения в целевой тип. Допустимые форматы даты: 1) "[AD|BC] DD/MM/YYYY[[AM|PM] HH:MI[:SS]]" 2) "[AD|BC] DD.MM.YYYY[[AM|PM] HH:MI[:SS]]" 3) "[AD|BC] DD MM YYYY[[AM|PM] HH:MI[:SS]]" 4) "[AD|BC] YYYY-MM-DD[[AM|PM] HH:MI[:SS]]" Где: – DD — значение дня (от 1 до 31); – MM — значение месяца (арабскими цифрами от 1 до 12, римской цифрой от I до XII или символьное обозначения наименования месяца латиницей (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec). Регистр при этом не важен; – YYYY — значение года (от одной цифры до четырёх); – AD|BC — обозначение нашей эры и до нашей эры соответственно/ Регистр должен быть не важен; – AM / PM — время до обеда (Ante Meridiem) и после обеда (Post Meridiem) соответственно. При использовании этого параметра часы должны быть указаны в 12 часовом формате (где AM 12:00 - полночь, и далее AM 01:00, а PM 12:00 — полдень), иначе должна быть ошибка. Регистр не важен; – HH — значение часа, формат представления которых может быть двадцатичетырёхчасовой (число от 0 до 23) или двенадцатичасовой (число от 1 до 12); – MI — значение минут (число от 0 до 59); – SS — значение секунд (число от 0 до 59). Для всех частей даты, при условии указания значения арабскими цифрами, лидирующий ноль допустим, но не обязателен. В качестве разделителя частей даты и времени допускается любое ненулевое число пробелов. В формате №3 в качестве разделителя между частями даты допускается любое ненулевое число пробелов. Если исходное значение не соответствует указанным форматам, то преобразование завершится ошибкой.

Целевой тип данных	Правила преобразования
TIMESTAMP	Если исходное значение соответствует одному из форматов даты, указанному далее, то оно будет преобразовано без изменения значения в целевой тип. Допустимые форматы даты: 1) "[AD|BC] DD/MM/YY[[AM|PM] HH:MI[:SS[.MS]]]" 2) "[AD|BC] DD.MM.YY[[AM|PM] HH:MI[:SS[.MS]]]" 3) "[AD|BC] DD MM YY[[AM|PM] HH:MI[:SS[.MS]]]" 4) "[AD|BC] YYYY-MM-DD[[AM|PM] HH:MI[:SS[.MS]]]" Где: – DD — значение дня (от 1 до 31); – MM — значение месяца (арабскими цифрами от 1 до 12, римской цифрой от I до XII, или символьное обозначения наименования месяца латиницей (jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec). Регистр при этом должен быть не важен; – YY[YY] - значение года (от двух цифр до четырёх); – AM / PM - время до обеда (Ante Meridiem) и после обеда (Post Meridiem) соответственно. При использовании этого параметра часы должны быть указаны в 12 часовом формате (где AM 12:00 - полночь, и далее AM 01:00, а PM 12:00 - полдень), иначе должна быть ошибка. Регистр должен быть не важен; – HH - значение часа, формат представления которых может быть двадцатичетырёхчасовой (число от 0 до 23) или двенадцатичасовой (число от 1 до 12); – MI — значение минут (число от 0 до 59); – SS — значение секунд (число от 0 до 59); – MS — значение доли секунды (для указания точности), которое должно быть допустимо указывать от одной до 18 цифр арабскими цифрами). Для всех частей даты, при условии указания значения арабскими цифрами, лидирующий ноль должен быть допустим, но не обязателен. В качестве разделителя частей даты и времени должно допускаться любое ненулевое число пробелов. В формате №3 в качестве разделителя между частями даты должно допускаться любое ненулевое число пробелов. Если значение не соответствует указанным форматам, то преобразование должно завершиться ошибкой.
BOOLEAN	Если исходное значение является одним из допустимых литералов (“TRUE” или “FALSE” в любом регистре), используемых для определения логического значения, то результатом преобразования будет соответствующее логическое значение. Если исходное значение символьной строки содержит только символ “0” или символ “1”, то результатом преобразования будет значение false в первом случае или true во втором. Если исходное значение содержит начальные и / или конечные пробелы, то при преобразовании они будут игнорироваться.
NUMBER [(precision [, scale])]	Если исходное значение соответствует формату [+/-]цифра[цифра]...["."цифра[цифра]...][[e/E][+/-]цифра[цифра]...]

Целевой тип данных	Правила преобразования
	и соответствует размерности целевого типа, то исходное значение будет преобразовано. В ином случае преобразование завершится ошибкой (о соответствии размерности см. в преобразовании типа NUMBER). Если исходное значение содержит начальные и / или конечные пробелы, то при преобразовании они будут игнорироваться.
BLOB	Если в исходном значении содержаться только символы шестнадцатеричного числа (без учета регистра букв) в чётном количестве, то исходное значение будет преобразовано в числовое представление в соответствии с шестнадцатеричным представлением исходного значения, где две шестнадцатеричные цифры = 1 байт. В ином случае преобразование завершится ошибкой
BINARY (size), VARBINARY (size)	Результат будет тот же, что описан в случае с BLOB, но с дополнением: если длина в байтах полученного значения меньше или равна size целевого типа, то преобразование будет выполнено успешно. В ином случае преобразование завершится ошибкой
Из типа CLOB в целевой тип:	
CHAR (size), VARCHAR (size)	Каждый символ исходного значения будет преобразовано в соответствии с кодировкой целевого типа. Если длина в байтах полученного значения меньше или равно size в байтах целевого типа, то значение будет преобразовано. В ином случае преобразование завершится ошибкой
CLOB	Значение будет преобразовано без изменений
Из типа DATE в целевой тип:	
CHAR (size), VARCHAR (size)	Исходное значение будет преобразовано в строку в соответствии с форматом: YYYY-MM-DD HH:MI:SS Каждый символ исходного значения по шаблону будет заменяться на один символ в итоговом значении целевого типа. Если длина полученного значения меньше или равна size целевого типа, то значение должно быть преобразовано. В ином случае преобразование завершится ошибкой
DATE	Значение будет преобразовано без изменений
TIMESTAMP	Преобразованное значение будет равно исходному, а доли секунд, присутствующие в целевом типе, будут установлены равными нулю
Из типа TIMESTAMP в целевой тип:	
CHAR (size), VARCHAR (size)	Исходное значение будет преобразовано в строку в соответствии с форматом: YYYY-MM-DD HH:MI:SS.MS (где MS - доли секунд, которые занимают 3 символа в результирующей строке). Каждый символ исходного значения по шаблону будет заменяться на один символ в итоговом значении целевого типа (за исключением шаблона MS). Если длина полученного значения меньше или равна size целевого типа, то значение будет преобразовано. В ином случае преобразование завершится ошибкой
DATE	Преобразованное значение будет равно исходному, но без долей секунд (т.е. доли секунд будут усечены)
TIMESTAMP	Значение будет преобразовано без изменений
Из типа BOOLEAN в целевой тип:	
CHAR (size), VARCHAR (size)	Исходное значение будет преобразовано в строку в соответствии с правилом:

Целевой тип данных	Правила преобразования
	– если логическое значение равно TRUE, то результирующее значение будет равно 'TRUE'; – если логическое значение равно FALSE, то результирующее значение будет равно 'FALSE'. Учитывая, что длина в байтах результирующего значения 'TRUE' равна 4, а значения 'FALSE' равна 5 и если длина в байтах полученного значения меньше или равно size в байтах целевого типа, то значение будет преобразовано. В ином случае преобразование завершится ошибкой
BOOLEAN	Значение будет преобразовано без изменений
Из типа NUMBER [(precision [, scale])] в целевой тип:	
CHAR (size), VARCHAR (size)	Значение типа NUMBER будет преобразовано к формату с фиксированной точкой. Далее без учета лидирующих нулей в целой части, а также без завершающих нулей в дробной части полученное значение будет преобразовано в десятичной системе счисления в строковое представление в кодировке ASCII. Для дробных величин в качестве разделителя целой и ненулевой дробной части будет использован символ “.”. Нулевая целая часть будет представлена одним нулём. Преобразование будет выполнено, если длина в байтах полученного значения меньше или равна size целевого типа. В ином случае преобразование завершится ошибкой
NUMBER [(precision [, scale])]	Если (precision, scale) преобразуемого значения отличается от (precision, scale) целевого типа, то поведение преобразования значений из типа number в number с разными значениями precision (далее p) и scale (далее s), где p1, s1 - параметры исходного типа, p2, s2 - параметры целевого типа будет следующим: 1) Если p2 < p1, то при любом соотношении s2 и s1 преобразование завершится ошибкой. 2) Если p2 = p1 и: – s2 > s1, то преобразование завершится ошибкой; – s2 < s1, то значение будет преобразовано с округлением до scale целевого типа; – s2 = s1, то значение будет преобразовано без изменений. 3) Если p2>p1 и: – s2 < s1, то значение будет преобразовано с округлением до scale целевого типа; – s2 = s1, то значение будет преобразовано без изменений 4) Если s2 > s1 – (p2 - s2) < (p1 - s1), то преобразование завершится ошибкой (например, из number (5, 2) в number (6, 4)); – (p2 - s2) >= (p1 - s1), то значение будет преобразовано с округлением до scale целевого типа (например, из number (5, 2) в number (8, 1)).
Из типа BLOB в целевой тип:	
BLOB	Значение будет преобразовано без изменений
BINARY (size), VARBINARY (size)	Результат будет тот же, что описан в случае с BLOB, но с дополнением: если длина в байтах полученного значения меньше или равна size целевого

Целевой тип данных	Правила преобразования
	типа, то преобразование будет выполнено успешно. В ином случае преобразование завершится ошибкой
Из типа BINARY (size), VARBINARY (size), ROWID	
CHAR (size), VARCHAR (size)	Байты исходного значения будут преобразованы в символы в соответствии с кодировкой целевого типа. Если длина в байтах полученного значения меньше или равна size в байтах целевого типа, то значение будет преобразовано. В ином случае преобразование завершится ошибкой
CLOB	Байты исходного значения будут преобразованы в символы в соответствии с кодировкой целевого типа.
BLOB	Значение будет преобразовано без изменений
BINARY (size), VARBINARY (size), ROWID	Если длина в байтах исходного значения меньше или равна size в байтах целевого типа, то значение будет преобразовано без изменений. В ином случае преобразование завершится ошибкой

ПРИЛОЖЕНИЕ 5

КОДЫ ОШИБОК

Номер ошибки	**Описание**
-1403	Не найдена указанная запись
-1476	Деление на ноль запрещено
-1489	Результат объединения строк превышает максимально допустимую длину строки
-2112	Переданный номер строки больше количества строк, хранимых в таблице БД
-6502	Ошибка арифметики / преобразования / усечения или же ограничения по размеру
-20006	Указанный параметр недопустим / неверный
-20011	Неправильное состояние БД при смене опции
-20014	Значение уже существует и не может иметь дубликата
-20018	Не найдена опция БД, возможно, допущена ошибка в её написании в запросе
-21001	Конфликт транзакции - несколько транзакций пытаются взаимодействовать с одной и той же области данных
-21007	Указанная точка сохранения не существует
-21009	Транзакция уже запущена
-22002	Получена опция с неизвестным типом
-22004	База данных уже запущена
-22011	Ошибка прочтения значения из истории, необходимо произвести отмену изменений в БД
-22014	Поиск по ключу не доступен в указанном хранилище
-22017	Указан недопустимый идентификатор объекта в ROWID
-22018	Устаревшая версия БД
-22500	База данных с указанным именем уже существует
-22506	База данных с указанным именем не существует Необходимо проверить верность написания имени БД
-22509	Неудачная попытка авторизации: имя пользователя или пароль не действительны
-22510	Неудачная попытка авторизации: недостаточно прав доступа для подключения к БД
-22512	Указанная база данных не запущена
-22516	В команде допущена синтаксическая ошибка – не указано имя базы данных
-23003	Ошибка соединения с базой данных
-23010	Соединение с базой данных разорвано
-23028	Ошибка получения информации о хосте, возможно в строке подключения отсутствует один или несколько обязательных параметров (имя БД, имя пользователя, пароль, и т. п.)
-23038	Ошибка открытия сетевого соединения. Необходимо перезапустить клиента
-23203	Ошибка открытия файла конфигурации
-23301	Указанная опция не существует или указана неверно
-23302	Формат введённого значения не соответствует формату опции
-23305	Для одной и той же опции несколько раз в запросе указано значение
-23412	Неудачное создание каталога, возможно недостаточно прав или недостаточно места на машине, на которой развёрнута СУБД

Номер ошибки	Описание
-23501	Передана пустая строка со временем
-23502	Переданы недопустимые данные для даты / времени
-23503	Формат переданной строки некорректный
-23510	Ошибка ввода долей секунды
-23512	Ошибка выделения буфера под хранение даты/времени
-23513	Ошибка ввода даты
-23514	Ошибка ввода времени
-23518	Вызвана операция, недопустимая в отношении даты/времени
-24002	Количество параметров, передаваемых в базу данных, не соответствует с количеству параметров, которое ожидает сама база данных
-24107	В выборке конструкции типа «select ... from t where t.i = select ...» пришло более одной записи
-24108	Запрашиваемый функционал в данной версии СУБД не реализован, возможно допущена орфографическая ошибка в команде.
-24112	Неправильное использование LIKE
-24113	Слишком длинный шаблон был передан в конструкцию типа LIKE (шаблон не должен превышать длину в 32 символа)
-24116	Запрос не исполнился или пришел пустой запрос Необходимо проверить синтаксис SQL-команды
-24117	В запросе было указано количество параметров отличное от ожидаемых сервером
-24118	Произошло несовпадение переданных в запросе флагов с флагами, ожидаемых сервером
-24119	Произошло несовпадение переданных в запросе типов параметров с ожидаемыми сервером
-24120	Произошло несовпадение переданных в запросе длин параметров с ожидаемыми сервером
-24122	Автономная транзакция не завершена Необходимо убедиться, что все автономные операции завершены и/или не возникает из-за них наложений транзакций
-24123	Передана последовательность, которое меньше минимально допустимого, или больше максимально допустимого значений Необходимо проверить длину последовательности и убедиться, что она находится в пределах между минимальной и максимальной
-24125	Была произведена попытка управления базой данных при помощи сервис — специфицированных команд Данная команда может быть исполнена только из подключения к сервису
-24127	Неверно задана точность даты Необходимо проверить, правильное значение передано в качестве точности даты
-24128	Команде типа TRIM передано больше одного символа на отсечение
-24133	При приведении значения из binary в char буфер, выделенный на char-последовательность, меньше необходимого
-24134	Поле или переменная не были пусты, когда произошла попытка записи данных в неё В NOT NULL колонку передавать значения нельзя

Номер ошибки	Описание
-24137	В качестве номера вхождения подстроки в строку в функции INSTR было передано отрицательное значение
-24144	В функцию RAND передан интервал, в котором минимальное значение больше или равно максимальному
-24145	Был выполнен INSERT до команды CREATE INDEX Необходимо проверить очерёдность команд. Попытка вставки данных после создания индекса запрещена
-24146	Обращение через FOREIGN KEY к связанной таблице в условиях отсутствия ключа, с которым связан FOREIGN KEY
-25001	В команде допущена синтаксическая ошибка
-25011	В качестве операнда было передано значение недопустимого типа
-25012	СУБД не может обработать указанную схему
-25013	СУБД не может обработать указанную схему
-25014	Указан столбец, который нельзя однозначно идентифицировать
-25015	В конструкции WITH образовалась циклическая ссылка
-25016	В качестве переменной (таблицы, колонки и т. п.) был передан аргумент, который не найден в базе данных
-25017	В CALL была передана неизвестная процедура
-25018	Попытка обращения к неизвестной таблице
-25019	Указано неверное имя правила сравнения
-25020	Ошибка ссылки в SQL-запросе на существующий объект
-25021	В запросе содержится недопустимый символ
-25022	Попытка выполнить операцию с объектом, который имеет несоответствующий тип данных
-25023	В запросе передается неверное количество аргументов для функции или процедуры
-25024	Используется неверное имя столбца, таблицы или другого объекта в запросе
-25025	Переданное значение не соответствует типу DECIMAL
-25026	Переданное значение даты имеет некорректный формат
-25028	Ошибка преобразования типов данных
-25032	Имя объекта уже существует в базе данных
-25033	Указанная процедура уже существует
-25036	Используется недопустимая конструкция SQL-запроса
-25041	Указанное имя объекта не действительно
-25042	Операция над значениями несовместимого типа данных
-25043	В подзапросе указано слишком большое число столбцов
-25044	В USING указано несуществующее имя столбца
-25047	В функцию передано недостаточно аргументов
-25048	Невозможно осуществить сравнение двух строк
-25049	Результатом выполнения оператора CASE является не boolean значение
-25050	В запросе используется некорректное выражение для группировки
-25051	В запросе используется неправильный формат шестнадцатеричной строки
-25053	В запросе используется несуществующий индекс
-25054	Невозможно удалить таблицу и/или её элементы
-25055	Невозможно удалить индекс

Номер ошибки	Описание
-25056	В запросе INSERT не совпадает количество добавляемых значений с количеством столбцов в таблице
-25057	Система не может правильно интерпретировать SQL-запрос
-25058	Не известный тип данных
-25059	Попытка вставить или обновить значение, которое превышает максимально допустимый размер для типа данных столбца
-25060	Попытка использовать дробную часть в значении даты
-25061	Слишком большая строка, полученная в результате выполнения конкатенации строк
-25062	При создании объекта используется зарезервированное имя
-25063	Попытка сортировки по несуществующему столбцу или указание неверной позиции столбца в выражении ORDER BY
-25064	Слишком большое строковое значение
-25065	Слишком большое бинарное значение
-25067	Попытка недопустимого преобразования типов данных
-25068	Некорректный тип операнда для вызванной операции
-25069	Недействительное ограничение
-25070	Недопустимое использование виртуального столбца
-25071	Попытка вставки в таблицу или запрос слишком большого числа значений, чем ожидается
-25072	Указан неожиданный символ или ключевое слово, которое не соответствует указанному синтаксису запроса
-25073	Ожидался тип CHAR
-25075	Передано недостаточное количество элементов для вставки
-25076	Попытка использовать недопустимого выражения для генерации столбца при создании таблицы
-25077	Попытка использовать недопустимый элемент в качестве значения по умолчанию для столбца таблицы
-25078	Текущий пользователь не имеет прав на DATABASE ADMIN
-25079	Текущий пользователь не имеет прав на RESOURCE CONTROL
-25080	Текущий пользователь не имеет прав на выполнение SELECT
-25081	Попытка выполнить операцию, которую не поддерживает используемая версия базы данных
-25083	Текущий пользователь не имеет прав на выполнение INSERT
-25084	Текущий пользователь не имеет прав на выполнение UPDATE
-25085	Текущий пользователь не имеет прав на выполнение DELETE
-25087	Попытка удаления пользователя, который является владельцем существующего объекта
-25090	Попытка удаления схемы базы данных, которая не является пустой
-25091	Попытка удаления схемы, которая назначена схемой по умолчанию для пользователя Необходимо сменить у пользователя схему по умолчанию через ALTER USER SCHEMA
-25092	Ошибка запроса выражения виртуальной колонки

Номер ошибки	Описание
-25093	Количество столбцов, переданных для генерации представления не совпадает с количеством столбцов в указанном представлении
-25094	Указание представления в не подходящем месте SQL-запроса
-25095	В запросе указан недопустимый тип данных для столбца в таблице
-25096	Ошибка запроса, когда в одном SELECT-списке смешаны агрегированная (группированная) и неагрегированная (негруппированная) информация
-25097	В запросе выбирается слишком большое количество столбцов (из таблицы или в результате соединения нескольких таблиц) или происходит попытка создания БД со слишком большим числом столбцов
-25099	Неверное обращение к представлению или указано несуществующее представление
-25100	Неправильное выражение в столбце
-25101	У пользователя недостаточно прав на выполнение операции
-25103	Попытка взаимодействия с не существующим объектом базы данных
-25105	Попытка обращения к несуществующей схеме
-25106	Попытка использовать псевдоколонку в месте, где это запрещено
-25107	Попытка создать более одного кластеризованного индекса для таблицы
-25108	В SQL-команде не указаны столбцы
-25109	Попытка удалить кластеризованный индекс из таблицы, которая содержит данные
-25110	Неверно задано ограничение числа строк Необходимо проверить значение LIMIT
-25111	Некорректное использование выражения (к примеру, «select max(*) from t», где использование max(*) некорректно)
-25112	Некорректное использование аргумента
-25113	Некорректное использование объекта
-25114	Необходимо явное приведение значения к нужному типу данных с помощью CAST
-25115	Попытка использования агрегатной функции внутри другой агрегатной функции
-25116	Попытка обращения к индексу и таблице из разных схем
-25117	Некорректное имя временной приватной таблицы
-25118	Некорректное имя глобальной таблицы
-25120	Попытка выполнить преобразование значений типов данных, которое не поддерживается
-25121	В запросе используется оператор цикла, но не указана область действия цикла
-25122	В блоке TRY ... CATCH используется не существующее имя исключения
-25123	Попытка использования RETURNING с неправильным количеством значений
-25124	Попытка использовать зарезервированное имя в качестве псевдонима таблицы
-25125	Попытка использования неподходящего выражения ON COMMIT в команде ALTER TABLE
-25126	Попытка указания недопустимого значения точности (precision) для числового типа данных
-25127	Попытка указания недопустимого значения масштаба (scale) для числового типа данных

Номер ошибки	Описание
-25128	Попытка выполнить команду ANALYZE TABLE для временной таблицы
-25129	Попытка создания роли с неверным указанием её имени или у пользователя недостаточно прав доступа
-25130	Попытка удаления несуществующей роли или у пользователя недостаточно прав доступа
-25131	Попытка отзыва роли у пользователя, который ею не обладает
-25132	Попытка создания дубликата существующего столбца таблицы
-25133	Попытка выполнить операцию с несоответствующим типом данных столбца
-25134	Попытка создать слишком длинный индекс
-25135	Недостаточно прав у пользователя для изменения своего набора системных ролей и привилегий
-25136	Неверное значения начала последовательности, минимального или максимального значения её шага
-25137	Попытка обращения к несуществующему пользователю
-25138	Не удалось создать пользователя. Проверьте корректность указания имени создаваемого пользователя и наличие прав у пользователя на создание пользователя
-25139	Не удалось создать пользователя, т.к. пользователь с таким именем уже существует
-25140	Не удалось создать роль, т.к. роль с таким именем уже существует
-25141	Не удалось создать желаемую процедуру Проверьте корректность имени процедуры, синтаксиса и наличие прав у пользователя на создание процедуры
-25142	Не удалось создать таблицу, т.к. объект с таким именем уже существует
-25143	Не удалось создать желаемую таблицу Проверьте корректность имени таблицы, синтаксиса и наличие прав у пользователя на создание таблицы
-25144	Не удалось создать ограничение Проверьте корректность имени ограничения, синтаксиса и наличие прав у пользователя на создание ограничения
-25145	Попытка создать таблицу с одноимёнными столбцами
-25146	Попытка удалить хранимую процедуру, которая не может быть удалена (из-за наличия зависимостей или других причин)
-25147	Попытка удалить представление из базы данных, которое не может быть удалена (из-за наличия зависимостей или других причин)
-25148	Попытка создания последовательности с неверными параметрами или при отсутствии у пользователя необходимых прав доступа
-25149	Ошибка создания индекса
-25150	Ошибка очистки таблицы
-25151	Ошибка создания схемы по умолчанию
-25152	Ошибка создания представления Может произойти, если вставка данных заканчивается неудачей
-25153	Ошибка удаления последовательности
-25154	Ошибка удаления пользователя
-25155	Схема с указанным именем уже существует
-25156	Ошибка создания схемы

Номер ошибки	Описание
-25157	Ошибка удаления схемы
-25158	Ошибка изменения схемы.
-25159	Ошибка выполнения команды ALTER USER
-25160	Ошибка установки текущей схемы пользователя в рамках текущего соединения с базой данных
-25161	Ошибка предоставления привилегий
-25162	Ошибка отзыва привилегий
-25163	Ошибка отзыва привилегий
-25164	Ошибка предоставления роли Причины: - у пользователя недостаточно прав для выполнения команды; - указанная роль не существует
-25165	Ошибка отзыва роли у пользователя
-25166	Ошибка отзыва роли у другой роли
-25167	Возникает, когда правило сравнения базы данных не соответствует ее набору символов
-25168	Несоответствие версии БД версии сервиса
-25169	Ошибка исполнения JOIN USING, если в конструкцию JOIN USING был передан не существующий столбец
-25170	Пользователь не имеет достаточных привилегий для исполнения команды
-25171	Не подходящая привилегия для объекта
-25172	Попытка текущего пользователя без привилегии USAGE обратиться к последовательности, созданной другим пользователем
-25173	Появление циклической зависимости у представления
-25174	Попытка создания глобальной таблицы без указания хотя бы одного не виртуального столбца
-25175	Попытка создания объекта (например, таблицы, столбца, индекса и т.д.) с недопустимым именем
-25176	Пропущено указание типа данных
-25177	При создании последовательности указано некорректное значение кэша
-25178	Попытка пользователь удалить самого себя
-25179	Попытка удаления текущей схемы
-25180	Попытка использования столбца с типом LOB во временной таблице
-25181	Недопустимое использование псевдоколонок в качестве процедур
-25182	Несовместимые параметры последовательности
-25183	Необходимо указать MAXVALUE для последовательности
-25184	Попытка для одного и того же столбца установить одинаковые по смыслу ключи (PK/CLUSTERED/UNIQUE)
-25185	Попытка создать индекс на столбцах, которые уже имеют индекс
-25186	Ошибка выполнения ALTER TABLE
-25187	Ошибка удаления PRIMARY KEY
-25188	Указание запрещённого имени в последовательности
-25190	Указание имени столбца там, где это запрещено
-25191	Попытка создания последовательности со значением MINVALUE больше, чем значение MAXVALUE

Номер ошибки	Описание
-25192	Попытка удалить ограничение из таблицы, которое зависит от индекса. Сначала необходимо удалить связанный с ограничением индекс
-25193	В качестве условия в CHECK передано недопустимое выражение
-25194	Нарушено условие проверки (check constraint) для определенного столбца таблицы
-25195	Ошибка переименования таблицы
-25196	В запросе не указано ожидаемое логическое выражение
-25197	Ошибка создания уникального индекса по столбцу временной таблицы, который может содержать значения NULL
-25198	Ошибка переименования столбца
-25199	Попытка указать неверное число столбцов в FOREIGN KEY
-25200	Попытка указать недопустимые столбцы для FOREIGN KEY и PRIMARY KEY
-25201	Типы данных столбцов родительской и дочерней таблиц, указанных FOREIGN KEY, отличны
-25202	Не удалось создать FOREIGN KEY Возможно неверно указано имя родительской таблицы или указано имя несуществующей таблицы
-25203	Попытка выполнения команды PRESS TABLE или PRESS ALL TABLE по отношению к несуществующей таблице
-25205	Попытка удаления таблицы, на которую ссылаются внешние ключи из других таблиц. Необходимо использовать опцию CASCADE
-25206	Была произведена попытка попытка взаимодействия с циклическими ссылками
-25207	Попытка создания ограничения, которое уже существует
-25208	Попытка удаления или модифицирования столбца, которого не существует в таблице
-25209	Попытка удаления столбца, используемого в ограничении (constraint) Предварительно удалите ограничение или используйте CASCADE
-25210	Попытка удаления столбца, который является частью кластеризованного ключа в таблице Предварительно удалите кластеризованный ключ
-25211	Ошибка удаления столбца
-25212	Попытка удаления последней колонки из таблицы
-25213	Попытка удаления столбца, который используется в качестве выражения для виртуального столбца Предварительно удалите виртуальный столбца
-25214	Попытка изменения свойств столбца и значений, который используется в качестве foreign key, constraint или по каким-либо причинам сервер не смог выполнить данный запрос
-25215	Попытка смены типа данных столбца на несовместимый с предыдущим типом
-25217	Попытка изменения столбца, содержащего данные
-25218	Ошибка изменения типа столбца, который указан во внешнем ключе
-25219	Ошибка изменения типа столбца, который указан в проверяющем ограничении

Номер ошибки	Описание
-25220	Попытка добавить выражение, которое генерируется автоматически, к столбцу с типом данных REAL
-25221	Попытка добавления выражения по умолчанию к виртуальной колонке
-25222	Попытка изменения типа виртуальной колонки без генерируемого always выражение
-25223	Попытка использовать в выражении, которое генерируется автоматически, ссылку на виртуальный столбец
-25224	Ошибка переименования индекса. Возможно у текущего пользователя недостаточно прав для выполнения данной команды
-25225	Ошибка изменения индекса. Возможно у текущего пользователя недостаточно прав для выполнения данной команды
-25237	Указано недопустимое значение параметра time_zone
-24147	Попытка передать данные, превышающие по объёму, максимальное число байт, позволенных в БД
-24151	Попытка завершения собственной сессии без указания ключевого слова IMMEDIATE
-24152	Запрос KILL SESSION ожидал завершения указанной сессии более минуты
-24153	Указан номер несуществующей сессии