

Лекция 5.

Язык SQL для работы с реляционными базами данных

Structured Query Language – декларативный язык для описания, изменения и извлечения данных, хранимых в реляционных базах данных.

- Стандартизирован организациями ANSI, ISO (SQL-86, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016, SQL:2019, SQL:2023)
- Кроссплатформенный
- Простой в изучении и использовании
- Интерактивный и программный режим запросов
- Обеспечивает различные представления данных
- Хорошо подходит для архитектуры «клиент-сервер»

Возникновение SQL в IBM

- **1970 год:** Эдгар Кодд (Edgar F. Codd), сотрудник IBM, публикует статью «A Relational Model of Data for Large Shared Data Banks», в которой впервые формализует **реляционную модель данных**. Это стало теоретической основой для будущих реляционных СУБД.
- **1974–1975 годы:** В исследовательском центре IBM в Сан-Хосе (ныне IBM Almaden Research Center) группа под руководством Дональда Чемберлина (Donald D. Chamberlin) и Рэя Бояра (Raymond F. Boyce) разрабатывает язык запросов под названием **SEQUEL** (Structured English Query Language), позже переименованный в **SQL** из-за торговой марки.
- **1978–1979 годы:** IBM выпускает первую экспериментальную СУБД **System R**, в которой реализован язык SEQUEL/SQL. System R доказала практическую применимость реляционной модели и SQL.

Коммерциализация и конкуренция

- **1979 год:** Компания **Relational Software, Inc.** (позже переименованная в **Oracle Corporation**) выпускает первую коммерческую реляционную СУБД — **Oracle V2**, поддерживающую SQL. Это был первый коммерческий продукт с SQL, опередивший даже IBM.
- **1980-е годы:** IBM выпускает свои коммерческие СУБД с SQL: **SQL/DS** (1981) и **DB2** (1983). В это же время появляются и другие СУБД с поддержкой SQL, например, **Ingres** от университета Калифорнии в Беркли.

Альтернативы SQL

- **QUEL** — язык запросов, разработанный в рамках проекта **Ingres** (Беркли). QUEL считался более строгим и последовательным по сравнению с ранними версиями SQL. Однако из-за агрессивного маркетинга Oracle и поддержки SQL со стороны IBM, QUEL проиграл в «войне стандартов».
- **Alpha** — язык, предложенный самим Эдгаром Коддом как более теоретически обоснованная альтернатива SQL. Однако он так и не получил широкого распространения.
- **ISBL** (Information System Base Language) — использовался в ранних реляционных системах, таких как **Peterlee Relational Test Vehicle (PRTV)** в Великобритании.

Почему SQL победил?

- Поддержка со стороны IBM и Oracle.
- Простота синтаксиса по сравнению с более формальными языками.
- Коммерческая агрессия Oracle, которая сделала SQL де-факто стандартом задолго до официальной стандартизации.
- Гибкость: SQL оказался достаточно выразительным для большинства задач, несмотря на критику со стороны теоретиков (например, за отклонения от реляционной алгебры).

Стандарты SQL

- **1986 год:** Американский национальный институт стандартов (**ANSI**) публикует первый официальный стандарт SQL — **SQL-86** (официально ANSI X3.135-1986).
- **1987 год:** Международная организация по стандартизации (**ISO**) принимает SQL как международный стандарт (**ISO/IEC 9075:1987**).

Стандарты SQL

- **SQL-89** — незначительные уточнения.
- **SQL-92 (SQL2)** — крупное расширение: подзапросы, внешние соединения, скалярные операции и т.д.
- **SQL:1999 (SQL3)** — добавлены объектно-реляционные возможности, регулярные выражения, рекурсивные запросы (WITH).
- **SQL:2003** — поддержка XML, оконные функции.
- **SQL:2008, SQL:2011, SQL:2016, SQL:2019, SQL:2023** — дальнейшие уточнения, аналитические функции, JSON-поддержка, полиморфные табличные функции и др.

Несмотря на наличие стандарта, большинство коммерческих СУБД (Oracle, Microsoft SQL Server, PostgreSQL, MySQL и др.) реализуют **свои диалекты SQL**, часто с отклонениями от стандарта.

Стандарты SQL

Официальный стандарт SQL публикуется Международной организацией по стандартизации (ISO) совместно с Международной электротехнической комиссией (IEC) под обозначением:

“ISO/IEC 9075: *Information technology — Database languages — SQL*”

Этот стандарт не является одним монолитным документом, а состоит из нескольких частей (parts), каждая из которых охватывает отдельный аспект языка:

🔒 **Важно:** Полный официальный текст стандарта недоступен бесплатно. Это часто вызывает критику со стороны open-source сообщества.”

Стандарты SQL

- Стандарт очень объёмный, и никто не реализует его полностью.
- Производители добавляют собственные расширения (например, `LIMIT` в MySQL/PostgreSQL vs `TOP` в SQL Server).
- Некоторые части стандарта (например, PSM — хранимые процедуры) реализованы по-разному или вообще отсутствуют.
- Стандарт отстаёт от практики: часто сначала появляется фича в СУБД (например, оконные функции в Oracle), а потом её включают в стандарт.

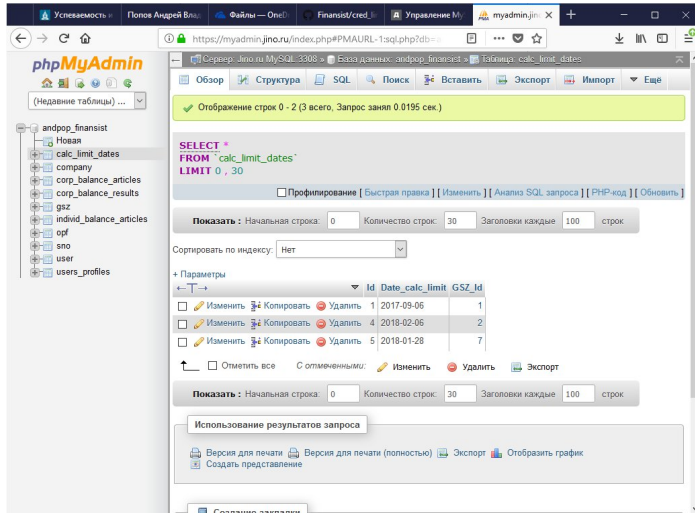
Таким образом, SQL — это де-юре стандарт, но де-факто — семейство совместимых диалектов, объединённых общей идеей и синтаксисом.

Категории команд

- **DDL** (Data Definition Language). **CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.**
- **DML** (Data Manipulation Language). **INSERT, UPDATE, DELETE**
- **DQL** (Data Query Language). **SELECT**
- **DCL** (Data Control Language). **GRANT, REVOKE.**
- Команды управления транзакциями.
BEGIN, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

Интерактивный режим

Web



CLI

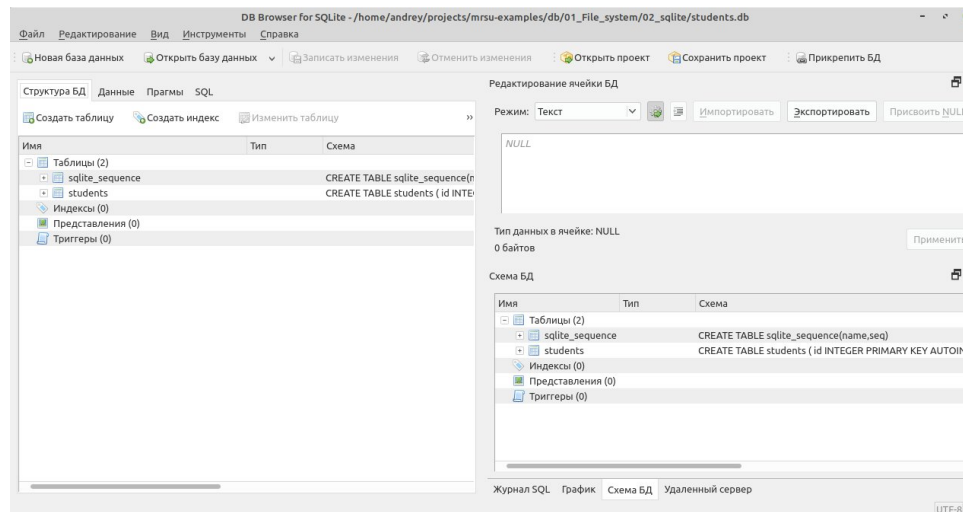
```
andrey@home:~$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 10.3.22-MariaDB-1:10.3.22+maria-bionic-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
MariaDB [(none)]>
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.001 sec)
```

Desktop



Создание базы данных

Создание базы данных

Команда `CREATE DATABASE` используется в некоторых СУБД (например, MySQL, PostgreSQL, SQL Server, Oracle) для создания новой базы данных — то есть отдельного логического или физического контейнера для хранения таблиц, индексов, пользователей и т.д.

 **Общий синтаксис (варьируется в зависимости от СУБД):**

```
sql
```

```
1 CREATE DATABASE имя_базы_данных;
```

Создание базы данных

MySQL

```
sql
1 CREATE DATABASE company_db;
2 -- С опциями:
3 CREATE DATABASE company_db
4     CHARACTER SET utf8mb4
5     COLLATE utf8mb4_unicode_ci;
```

PostgreSQL

```
sql
1 CREATE DATABASE company_db;
2 -- В PostgreSQL база создаётся на основе шаблона:
3 CREATE DATABASE company_db WITH OWNER = admin ENCODING = 'UTF8';
```

В SQLite база данных — это просто **файл на диске**, и она создаётся автоматически при первом подключении (через `sqlite3 имя.db` или программно). Поэтому команда `CREATE DATABASE` в SQLite не существует и вызовет ошибку.

База данных и схема

В MySQL понятие «схема» синонимично базе данных. Команда `CREATE SCHEMA имя;` в MySQL — это то же самое, что `CREATE DATABASE имя;`.

```
sql
1  -- В MySQL это одно и то же:
2  CREATE DATABASE company;
3  CREATE SCHEMA company;
```

В SQLite есть только одна схема — `main` (и временная `temp`). Нельзя создать несколько схем в одной базе. Поэтому в SQLite «база данных» и «схема» почти совпадают.

В PostgreSQL:

- База данных — это изолированный контейнер (физически отдельный).
- Схема — логическое подразделение внутри одной базы.
- Пример: одна БД `enterprise` может содержать схемы `finance`, `hr`, `inventory`.

Создание таблиц в БД

Создание таблиц

Команда `CREATE TABLE` — одна из ключевых в SQL: она позволяет определить структуру таблицы: имена столбцов, их типы данных и **ограничения (constraints)**, которые обеспечивают целостность данных.

♦ **Общий синтаксис** `CREATE TABLE`

sql



```
1 CREATE TABLE имя_таблицы (  
2     столбец1 тип_данных [ограничения],  
3     столбец2 тип_данных [ограничения],  
4     ...  
5     [ограничения на уровне таблицы]  
6 );
```

Ограничения (constraints)

ОГРАНИЧЕНИЕ	НАЗНАЧЕНИЕ
NOT NULL	Значение не может быть NULL
UNIQUE	Все значения в столбце должны быть уникальными
PRIMARY KEY	Уникальный идентификатор строки (не NULL + UNIQUE)
FOREIGN KEY	Ссылка на первичный ключ другой таблицы
CHECK	Пользовательское условие (например, возраст ≥ 0)
DEFAULT	Значение по умолчанию, если не указано явно

Создание таблиц

1. PostgreSQL (полная поддержка всех ограничений)

```
sql
1 CREATE TABLE users (
2     id SERIAL PRIMARY KEY,
3     username VARCHAR(50) NOT NULL UNIQUE,
4     email VARCHAR(100) NOT NULL UNIQUE,
5     age INTEGER CHECK (age >= 0 AND age <= 150),
6     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
7     country CHAR(2) DEFAULT 'US',
8     manager_id INTEGER REFERENCES users(id) ON DELETE SET NULL
9 );
```

Особенности:

- **SERIAL** — автоинкрементное целое число.
- Полная поддержка **CHECK**, **FOREIGN KEY**, **UNIQUE**, **DEFAULT**.
- **REFERENCES** можно указывать inline или отдельно.

2. MySQL (InnoDB — поддерживает все ограничения)

```
sql
1 CREATE TABLE users (
2     id INT AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(50) NOT NULL UNIQUE,
4     email VARCHAR(100) NOT NULL,
5     age TINYINT UNSIGNED CHECK (age BETWEEN 0 AND 150),
6     created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
7     country CHAR(2) DEFAULT 'US',
8     manager_id INT,
9     FOREIGN KEY (manager_id) REFERENCES users(id) ON DELETE SET NULL,
10    CONSTRAINT unique_email UNIQUE (email)
11 );
```

Особенности:

- `AUTO_INCREMENT` вместо `SERIAL`.
- До MySQL 8.0.16 `CHECK` игнорировался (только парсился), но с 8.0.16+ работает.
- `FOREIGN KEY` работает только в движке InnoDB (MyISAM их игнорирует).
- Можно задавать именованные ограничения (`CONSTRAINT имя ...`).

3. SQLite (ограниченная поддержка)

```
sql
1 CREATE TABLE users (
2     id INTEGER PRIMARY KEY, -- автоинкремент в SQLite
3     username TEXT NOT NULL UNIQUE,
4     email TEXT NOT NULL,
5     age INTEGER CHECK (age >= 0 AND age <= 150),
6     created_at TEXT DEFAULT (datetime('now')),
7     country TEXT DEFAULT 'US',
8     manager_id INTEGER REFERENCES users(id) ON DELETE SET NULL
9 );
```

Особенности:

- `INTEGER PRIMARY KEY` — автоматически становится автоинкрементным.
- Типы данных динамические («type affinity»), но лучше использовать `TEXT`, `INTEGER`, `REAL`, `BLOB`.
- `CHECK` поддерживается и работает.
- `FOREIGN KEY` по умолчанию отключены! Нужно включить:

```
sql
1 PRAGMA foreign_keys = ON;
```

Создание таблиц

Команда `CREATE TABLE` позволяет не просто задать структуру, но и гарантировать корректность данных с помощью ограничений. Хотя синтаксис в целом стандартизирован, важно учитывать особенности конкретной СУБД, особенно в части поддержки `FOREIGN KEY` и `CHECK`. Это критично при переносе схем между системами или разработке кроссплатформенных приложений.

Метаданные

В SQLite метаданные хранятся в таблице `sqlite_master`

Команды `.table` и `.schema`

Создание таблиц

```
v example customer
  id : int(6)
  name : varchar(30)
  city : varchar(30)
  # rating : int(3)
  # vendor_id : int(6)
```

```
v example vendor
  id : int(6)
  name : varchar(30)
  city : varchar(30)
  # percent : int(4)
```

```
v example order
  id : int(6)
  # summa : float
  order_date : date
  # customer_id : int(6)
  # vendor_id : int(6)
```

Вставка, изменение и удаление данных в таблицах

Добавление строк

INSERT INTO имя_таблицы [(имя_столбца, ...)] **VALUES** (значение, ...), ...

INSERT INTO имя_таблицы [(имя_столбца, ...)] **SELECT** (...)

```
INSERT INTO `vendor` (`id`, `name`, `city`, `percent`) VALUES
(1001, 'Иванов', 'Саранск', 12),
(1002, 'Петров', 'Москва', 13),
(1003, 'Андреев', 'Кострома', 10),
(1004, 'Сидоров', 'Саранск', 10);
```

Добавление строк

Vendor (Продавцы)

id	name	city	percent
1001	Иванов	Саранск	12
1002	Петров	Москва	13
1003	Андреев	Кострома	10
1004	Сидоров	Саранск	10

Customer (Покупатели)

id	name	city	rating	vendor_id
2001	Потапов	Саранск	100	1001
2002	Гарин	Владимир	200	1003
2003	Ли	Москва	200	1002
2004	Глухов	Самара	300	1002
2005	Клюев	Саранск	100	1001

Order (Заказы)

id	summa	Date	Pok_Nom	Prod_Nom
3001	18.69	10.12.2016	2005	1001
3002	767.19	10.12.2016	2001	1001
3003	1900.10	10.12.2016	2002	1003
3004	123.45	15.12.2016	2003	1002
3005	100.00	16.12.2016	2004	1002
3006	95.13	15.12.2016	2001	1001

INSERT, UPDATE, DELETE

Вставка строк в таблицу

```
INSERT INTO vendor(name, city, percent) VALUES ("Иванов", "Тверь", 10)
```

```
INSERT INTO ... SELECT ...
```

Изменение строк

```
UPDATE vendor SET name = "Шахов" WHERE name = "Иванов"
```

Удаление строк

```
DELETE FROM vendor WHERE name = "Шахов"
```

Выборка данных из таблиц

Структура оператора SELECT

SELECT	Определяет, какие столбцы включить в результирующий набор
FROM	Определяет таблицы, из которых выбираются данные, и которые нужно соединить друг с другом
WHERE	Отсеивает ненужные данные
GROUP BY	Используется для группировки строк по общим значениям столбцов
HAVING	Отсеивает ненужные группированные данные
ORDER BY	Сортирует строки результирующего набора по одному или нескольким столбцам

SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ORDER BY ...

В SELECT можно включать:

- Столбцы таблиц
- Литералы (числа строки)
- Вычисляемые выражения
- Вызов встроенных и пользовательских функций

```
SELECT 10 'Просто число', name 'Имя', Length(city) 'Длина имени города',  
percent || '%' 'Процент' FROM Vendor
```

Предложение FROM может отсутствовать

SELECT ... **FROM** ... WHERE ... GROUP BY ... HAVING ... ORDER BY ...

FROM определяет таблицы, используемые запросом, и средства связывания таблиц вместе.

Таблица = набор связанных строк:

- **Постоянные.** CREATE TABLE ...
- **Производные.** Возвращаются подзапросом и хранятся в памяти. (SELECT ...)
- **Временные.** Изменяемые данные, хранящиеся в памяти. CREATE TEMPORARY TABLE...
- **Виртуальные.** Представления (views). CREATE VIEW ...

SELECT: выбор из одной таблицы

Выбор полей

```
SELECT id, name, city FROM vendor  
SELECT * FROM vendor
```

Условие поиска

```
SELECT id, name, city FROM vendor  
WHERE city="Саранск"
```

Логические операции в условии поиска

```
SELECT * FROM customer WHERE rating>=200 AND city="Москва"
```

Значение поля принадлежит множеству

```
SELECT * FROM vendor WHERE city IN ("Москва", "Саранск")
```

Поиск по шаблону

```
SELECT * FROM customer WHERE name LIKE "ИВ%"
```

("_" заменяет один символ)


```
SELECT * FROM customer WHERE name RLIKE "ов$"; (MySQL)
```

```
SELECT * FROM customer WHERE LEFT(name, 2) = "ИВ"; (MySQL)
```

```
SELECT * FROM customer WHERE substr(name, 1, 2) = "ИВ"; (SQLite)
```

SELECT: выбор из одной таблицы

Построение вычисляемых полей

 `SELECT CONCAT(LEFT(name,2),".-", city) AS "Фамилия-Город" FROM`

 `endor`

`SELECT substr(name,1,2) || ".-" || city) AS "Фамилия-
Город" FROM vendor`

Сортировка выводимых данных

`SELECT * FROM vendor ORDER BY name`

Функции агрегирования (COUNT, SUM, AVG, MAX, MIN)

Общая сумма заказов

`SELECT SUM(summa) FROM `order``

Общее число покупателей

`SELECT COUNT(*) FROM customer`

Группировка данных при выборке

SELECT ... FROM ... WHERE ... **GROUP BY ... HAVING ...** ORDER BY ...

Данные можно агрегировать и группировать, тогда мы будем работать с ними на более высоком уровне (с меньшей детализацией), чем они хранятся в базе.

```
SELECT vendor_id FROM `order`;
```

```
SELECT vendor_id FROM `order` GROUP BY vendor_id;
```

```
SELECT vendor_id, count(*) FROM `order` GROUP BY vendor_id;
```

```
SELECT vendor_id, count(*) FROM `order` GROUP BY vendor_id  
HAVING count(*) > 1;
```

Промежуточные итоги (группировка)

Для каждого продавца определить максимальную сумму заказа

```
SELECT vendor_id, MAX(summa)
FROM `order`
GROUP BY vendor_id
```

Найти максимальные продажи для каждого продавца на каждый день

```
SELECT vendor_id, order_date, MAX(summa)
FROM `order`
GROUP BY vendor_id, order_date
```

Условия на группы записей - HAVING

```
SELECT vendor_id, order_date, MAX(summa)
FROM `order`
GROUP BY vendor_id, order_date
HAVING MAX(summa)>100
```

Вложенные (некоррелированные) запросы

Обычно внутренний запрос генерирует значения, которые тестируются на предмет истинности предиката.

По фамилии продавца найти все его заказы

```
SELECT * FROM `order`  
WHERE vendor_id = (SELECT id FROM vendor WHERE name="Иванов")
```

```
SELECT * FROM `order`  
WHERE vendor_id IN (SELECT id FROM vendor WHERE name="Иванов")
```