Введение в базы данных

Цель занятия

После освоения темы вы:

- узнаете, что такое базы данных и как с ними работать;
- познакомитесь с реляционными базами данных и СУБД;
- научитесь базовым навыкам проектирования баз данных.

План занятия

1. Что такое база данных
2. Типы баз данных
3. Как выбрать подходящий тип базы данных
4. Основы теории реляционных баз данных
5. Системы управления базами данных (СУБД) и SQLite
6. SQL – язык структурированных запросов
7. Основные операции с таблицами

Используемые термины

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

Реляционная база данных — совокупность данных, представленная в виде таблиц, где строки таблиц содержат информацию об объектах. База данных представляет собой набор таблиц со связями.

СУБД — комплекс программно-языковых средств, позволяющих создать базы данных и управлять данными.

Схема базы данных (от англ. Database schema) — структура БД, описанная на формальном языке, поддерживаемым СУБД.

Модель данных — формальная теория представления и обработки данных в системе управления базами данных (СУБД).

SQL [ˌes kjuː' el] англ. structured query language, «язык структурированных запросов» — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Конспект занятия

1. Что такое база данных

У базы данных есть **отличительные черты**:

- БД хранится и обрабатывается в вычислительной системе — бумажная версия записи данных не является БД.
- Данные в БД должны быть логически структурированы (систематизированы).
- БД содержит схему, которая описывает логическую структуру БД в формальном виде.

В совокупности по этим признакам можно сделать вывод, что нельзя назвать базой данных библиотеку или любую таблицу.

В базе данных удобно хранить большие объемы данных. Особенно когда файлов огромное количество, они связаны между собой и представляют табличные данные. В этом случае отлично подойдет реляционный тип базы данных.

2. Типы баз данных

Существует несколько типов баз данных:

- Иерархические.
- Сетевые.
- Документная база данных.
- Базы данных временных рядов.
- Базы данных «ключ-значение».
- Колоночные базы данных.
- NewSQL базы данных.
- Реляционные базы данных.

Рассмотрим подробнее каждый тип.

Иерархические базы данных

Отличительная черта: БД представляется в виде древовидной (иерархической) структуры, которая состоит из объектов (данных) различных уровней.

Достоинства и недостатки:

+ эффективное использование памяти;
+ быстрое выполнение операций над данными (добавление записи, изменение, удаление и извлечение);
- применима только к данным с иерархической структурой.

Где применяется: файловая система, для данных с иерархической организацией.

Примеры: Information Management System (IMS) фирмы IBM (1966—1968 гг.), System 2000 разработки SAS Institute.

Сетевые базы данных

Отличительная черта: в отличие от иерархической БД между объектами может быть несколько связей.

Достоинства и недостатки:

+ возможность реализации множественных связей между объектами;
+ низкие затраты памяти;
+ оперативность доступа к данным;
- нельзя изменить структуру БД после ввода данных.

Где применяется: для данных, представимых в виде графов.

Примеры: DBMS, IDMS, db_VISTA III, КОМПАС.

Документная база данных

Отличительная черта: нереляционная база данных, предназначенная для хранения и запроса данных в виде документов в формате, подобном JSON.

Достоинства и недостатки:

+ позволяет хранить неструктурированные данные;
- отсутствие хорошей аналитической поддержки и поддержки транзакций;
- сложности с масштабированием.

Где применяется: каталоги, пользовательские профили, издательское дело, системы управления контентом.

Примеры: CouchDB, Couchbase, MongoDB.

Базы данных временных рядов

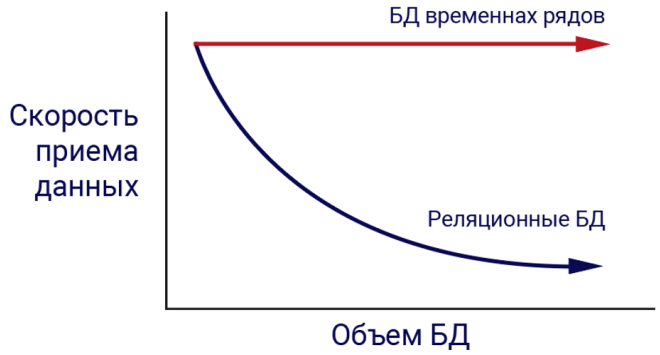
Отличительная черта: предназначены для хранения данных с временной составляющей.

Достоинства и недостатки:

+ оптимизированы для быстрого приема данных в отличие от реляционных БД;
- работают только с временными рядами.

Примеры: InfluxDB.

Построим график зависимости скорости приема данных от объема данных.



Разница между реляционными базой данных и базой данных временных рядов

- Для реляционных БД с ростом объема базы данных скорость приема данных уменьшается.
- Для БД временных рядов скорость остается неизменной. Это преимущество БД временных рядов, особенно когда данных много.

Базы данных «ключ-значение»

Отличительная черта: относятся к нереляционным БД, и для хранения данных применяется простой метод «ключ-значение».

Достоинства и недостатки:

+ горизонтальное масштабирование;
- низкая производительность;
- отсутствие стандарта.

Где применяется: интернет-магазины, хранилище сессий.

Примеры: Redis, Amazon DynamoDB.

Key	Value
K1	AAA, BBB, CCC
K2	AAA, BBB
K3	AAA, DDD
K4	AAA, 2,01/01/2015
K5	3, ZZZ, 5623

Пример БД на основе пар «ключ-значение»

Колоночные базы данных

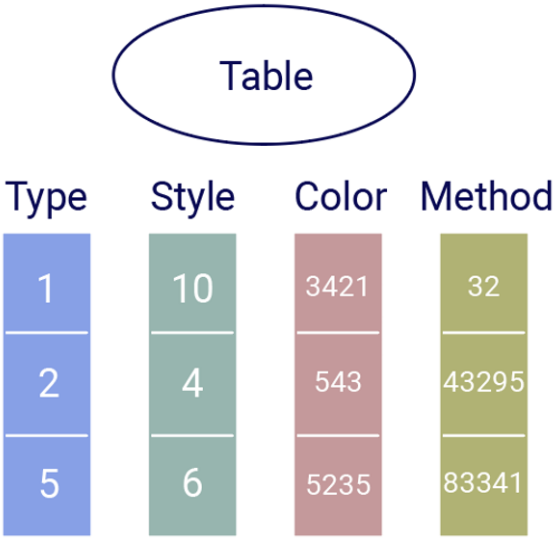
Отличительная черта: данные хранятся в столбцах.

Достоинства и недостатки:

+ возможность быстрого изменения структуры больших таблиц;
+ сжатие данных;
+ эффективность сложных выборок на больших таблицах;
+ масштабирование;
+ поддержка аналитики;
- не стоит использовать с выборками по ключу и известными структурами запросов.

Где применяется: событийные таблицы (добавление товаров в корзину в интернет-магазине), таблицы со статистикой добавления товаров в корзину по дням и т. д.

Примеры: Clickhouse, Vertica.



Пример колоночной базы данных

Обращение к данным производится по столбцам, а не по строкам.

NewSQL базы данных

Отличительная черта: сочетают в себе преимущества NoSQL и транзакционные требования классических баз данных.

Достоинства и недостатки:

+ отсутствие схемы (структура данных не фиксирована);
+ высокая производительность;
+ линейное масштабирование;
+ высокая скорость разработки.

Где применяется: высоконагруженные системы с поддержкой распределенных транзакций.

Примеры: H-Store, VoltDB, NuoDB, MariaDB.

Реляционные базы данных

Отличительная черта: данные хранятся в связанных между собой таблицах. В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке — значение атрибута.

Достоинства и недостатки:

+ представление данных удобно для пользователя;
+ позволяют создавать языки манипулирования данными не процедурного типа;
- медленный доступ к данным;
- трудоемкость разработки.

Где применяется: электронный документооборот, биллинговые системы, хранение данных о пользователях, адресные книги, справочники и т. д.

Примеры: SQLite, PostgresSQL, Oracle.

Row oriented (Relational)

Students		
ID	**First name**	**Last name**
1	Luna	Lovegood
2	Hermione	Granger
3	Ron	Weasley

Column oriented

Students		
ID	**First name**	**Last name**
1	Luna	Lovegood
2	Hermione	Granger
3	Ron	Weasley

Примеры реляционной базы данных (слева) и колоночной базы данных (справа)

3. Как выбрать подходящий тип базы данных

Чтобы выбрать подходящий тип базы данных, нужно:

1. Посмотреть на тип данных — структурированные или неструктурированные, данные в виде графов или таблицы.
2. Выбрать требования к системе — быстродействие, какие запросы планируется выполнять, важна ли возможность аналитики и т. д.

Что еще поможет в выборе:

- БД — это удобный способ хранения больших объемов данных с возможностью использования несколькими пользователями.
- Один из наиболее популярных типов — реляционные БД, в которых данные представимы в виде связанных таблиц.
- Для неструктурированных данных (например, документов) лучше использовать документную (документоориентированная) БД.
- Если запросы нужно делать к столбцам, то лучше выбрать колоночную базу данных.
- Для временных рядов есть отдельный тип баз данных, который рекомендуется использовать для хранения временных рядов.

4. Основы теории реляционных баз данных

В реляционных БД данные представляются в таблицах, где строки таблиц содержат информацию об объектах. БД представляет собой набор таблиц со связями.

Основные элементы реляционных БД

Объект:

id	date	customer	quantity	price	country
1	12.05.2021	1501	25	1	UK
2	10.05.2021	1231	2	4	Germany
3	14.05.2020	1045	5	6	UK
4	05.05.2021	1510	10	4	Russia
5	16.05.2020	2567	15	8	USA

Атрибут:

id	date	customer	quantity	price	country
1	12.05.2021	1501	25	1	UK
2	10.05.2021	1231	2	4	Germany
3	14.05.2020	1045	5	6	UK
4	05.05.2021	1510	10	4	Russia
5	16.05.2020	2567	15	8	USA

Запись данных:

id	date	customer	quantity	price	country
1	12.05.2021	1501	25	1	UK
2	10.05.2021	1231	2	4	Germany
3	14.05.2020	1045	5	6	UK
4	05.05.2021	1510	10	4	Russia
5	16.05.2020	2567	15	8	USA

Первичный ключ:

id	date	customer	quantity	price	country
1	12.05.2021	1501	25	1	UK
2	10.05.2021	1231	2	4	Germany
3	14.05.2020	1045	5	6	UK
4	05.05.2021	1510	10	4	Russia
5	16.05.2020	2567	15	8	USA

Вторичный ключ:

id	date	customer	quantity	price	country
1	12.05.2021	1501	25	1	UK
2	10.05.2021	1231	2	4	Germany
3	14.05.2020	1045	5	6	UK
4	05.05.2021	1510	10	4	Russia
5	16.05.2020	2567	15	8	USA

5. Системы управления базами данных (СУБД)

СУБД помогает управлять базами данных. Ее можно представить как посредника между базой данных и пользователем.

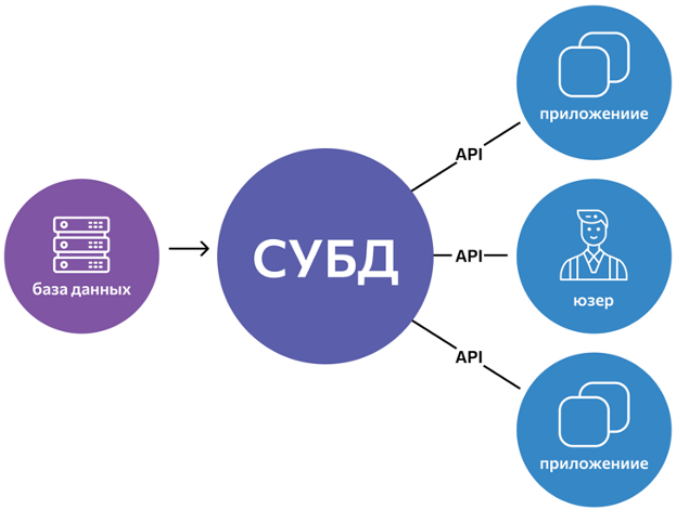


Схема представления СУБД

Примеры СУБД:

- SQLite.
- MySQL.
- Oracle DB.
- PostgreSQL.

Важно! СУБД не равно БД — БД хранит данные, а СУБД управляет данными.

Функции СУБД:

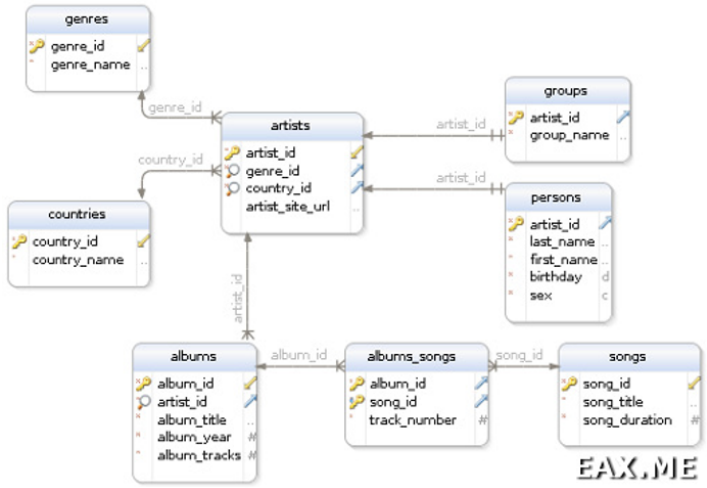
- Управление данными в оперативной памяти с использованием дискового кэша.
- Управление данными во внешней памяти (на дисках).
- Логирование изменений, создание резервных копий и восстановление базы данных после сбоев.
- Поддержка языков БД (язык определения данных, язык манипулирования данными).

Схема и модель базы данных

Модель БД показывает:

- какие таблицы есть в базе данных;
- из каких полей они состоят;
- какого типа эти поля;
- какие связи есть между полями.

На схеме связи между таблицами показаны стрелками:



Модель связей таблиц внутри СУБД

Виды отношений между таблицами:

- Один к одному.
- Один ко многим.
- Много к одному.
- Много ко многим.

6. SQL — язык структурированных запросов

Какие задачи можно решить с помощью SQL:

- создание в БД новой таблицы;
- добавление в БД новых записей;
- изменение записей;
- удаление записей;
- формирование выборок записей из одной или нескольких таблиц (в соответствии с условием).

Для практики мы будем использовать программу SQLite для работы с реляционными базами данных. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Это его отличительная черта.

Для SQLite характерна динамическая типизация данных, т. е. мы можем менять типы столбцов в процессе работы с таблицей.

Реализованы типы данных:

- INTEGER — целочисленные.
- REAL — численные.
- TEXT — текстовые.
- BLOB — двоичные.
- Специальное значение NULL — пропуски.

Работать с SQLite можно через командную строку / консоль SQLite, DB Browser for SQLite (DB4S) или Юпитер-ноутбук.

SQLite — это удобный вариант для знакомства с реляционными БД.

Практика

Работа с командной строкой:

1. Скачайте программу на свою операционную систему с официального сайта https://www.sqlite.org/download.html. Извлеките файл и запустите sqlite3. Никакие дополнительные установки не потребуются.
2. После запуска откроется командная строка для работы с базой данных.
3. Начнем с метакоманд для формирования таблиц и управления базами данных. Метакоманды начинаются с точки.
4. Чтобы открыть базу данных, используйте метакоманду **.databases**. Программа не выдаст результат, если у вас нет в директории ни одной базы данных.

```
sqlite> .databases
main: "" r/w
sqlite>
```

5. Чтобы создать новую базу данных, используйте **.open <Название БД>.<расширение>**. Если в директории нет базы данных с таким названием, команда создаст новую БД.

```
sqlite> .open Python_23.db
sqlite>
```

6. Чтобы отобразить таблицу, которая есть в базе данных, используйте метакоманду **.tables**. Программа не выдаст результат, если в БД отсутствуют таблицы.

7. Чтобы создать таблицу, нужно ввести команду **CREATE TABLE <Название> ()**. В скобках указываем поля таблицы и их тип.

```
sqlite> .tables

sqlite> CREATE TABLE Users (user_id integer, Name text, age
integer, Country text)

sqlite> .tables

Users

sqlite>
```

8. Чтобы заполнить таблицу, нужно воспользоваться оператором **INSERT INTO <Название таблицы> (<названия полей>) VALUES (<значения полей>)**.

```
sqlite> INSERT INTO Users (user_id, Name, age, Country) VALUES
(1, 'Alexander', 27, 'UK')

sqlite>
```

9. Для проверки ввода данных используем команду **SELECT * FROM <Название таблицы>;** — она выведет все строки в таблице.

```
sqlite> SELECT * FROM Users;
```

```
1|Alexander|27|UK
sqlite>
```

10. После добавления еще пары строк таблицу можно вызвать с названиями столбцов. Используйте **.headers ON**.

```
sqlite> .headers ON
sqlite> SELECT * FROM Users;
user_id|Name|age|Country
1|Alexander|27|UK
2|Anna|25|US
3|Mike|35|France
sqlite>
```

11. Чтобы перейти в другой режим вывода данных, используйте метакоманду **.mode column**.

```
sqlite> .mode column
sqlite> SELECT * FROM Users;
user_id Name       age Country
------- ---------- --- -------
1       Alexander  27  UK
```

```
2        Anna       25   US
3        Mike       35   France
sqlite>
```

12. Посмотреть схему таблицы можно с помощью команды **.schema <Название таблицы>**.

```
sqlite> .schema Users
CREATE TABLE Users (user_id integer, Name text, age integer,
Country text)
sqlite>
```

13. Метод **.show** выводит информацию о настройках.

```
sqlite> .show
        echo: off
         eqp: off
     explain: auto
     headers: on
        mode: column --wrap 60 --wordwrap off --noquote
   nullvalue: “”
      output: stdout
```

```
colseparator: "|"
rowseparator: "\n"
       stats: off
       width: 0 0 0 0
    filename: Python_23.db
sqlite>
```

14. Чтобы загрузить готовые данные, можно использовать команду **.import <Название файла>.<расширение> <Название таблицы>**.

```
sqlite> .mode csv
sqlite> .import supermarket_sales.csv Sales
sqlite>
```

15. С помощью **.tables** проверьте, загрузилась ли таблица, а **SELECT * FROM** выведет все строки в загруженных данных.

Работа в DB Browser:

1. Скачайте программу с официального сайта https://sqlitebrowser.org/dl/. Извлеките файл и запустите DB Browser for SQLite.
2. Чтобы создать новую базу данных нажмите «Новая база данных» и дайте ей название. Затем откроется новое окно для редактирования таблицы.

3. Создайте таблицу. Введите ее название, и в окне «Поля» добавьте поля: user_id integer, Name text, age integer, Country text.
4. Выберите тип и поставьте галочки (если необходимо):

   Тип — INTEGER, TEXT и др.
   НП (Не пустое, не может быть пустым).
   ПК (Первичный ключ, если не должно быть одинаковых записей).
   АИ (Автоинкремент, для этого поля в таблице автоматически для каждой строки будут создаваться значения, которые увеличиваются на единицу).
5. Вкладка SQL позволяет писать запрос вручную.
6. Заполнить таблицу можно с помощью вкладки «Данные».
7. Для открытия существующей БД используйте команды Файл — Импорт — Таблицы из файла CSV.

7. Основные операции с таблицами

Создание и изменение таблиц

CREATE создает объект базы данных — саму базу, таблицу, представление, пользователя и так далее.

ALTER изменяет объект.

DROP удаляет объект.

Структура запроса	Пример
`CREATE TABLE table_name (`	`CREATE TABLE Persons (`

```
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

```
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
);
```

Практика

Обновление значений в таблице, удаление строк, удаление всей таблицы:

1. Открываем базу данных, с которой продолжаем работать.

```
sqlite> .open Python_23.db
```

2. Посмотрим таблицы в этой базе данных.

```
sqlite> .open Python_23.db
sqlite> .tables
```

3. Посмотрим значения в таблице Users_1 перед тем, как их модифицировать.

```
sqlite> .open Python_23.db
sqlite> .tables
sqlite> SELECT * FROM Users_1;
```

4. Обновление данных возраста для одной строки.

```
sqlite> UPDATE Users_1 SET age = 26 WHERE Name = 'Anna';
```

5. Если не будем использовать WHERE, то возраст всех строк установится 26. Восстановим возраст остальных строк.

```
sqlite> UPDATE Users_1 SET age = 26;
sqlite> UPDATE Users_1 SET age = 27 WHERE user_id = 1;
sqlite> UPDATE Users_1 SET age = 35 WHERE user_id = 3;
```

6. Для удаления одной из записей есть оператор DELETE.

```
sqlite> DELETE FROM Users_1 WHERE user_id = 1;
```

7. Добавим строчку и посмотрим, в каком порядке она добавится

```
sqlite> INSERT INTO Users_1 (user_id, Name, age, Country) VALUES
(1, 'Alexander', 27, 'UK');
```

8. Можно посмотреть, что поле user_id — первичный ключ.

```
sqlite> .schema Users_1
```

9. В таблице Users user_id не является первичным ключом по схеме, поэтому при добавлении строки она добавляется в конец таблицы. Строчки не сортируются, если мы не проведем дополнительно это действие.

10. Посмотрим, как можно добавить новый столбец в таблицу.

```
sqlite> ALTER TABLE Users_1 ADD column email text;
```

11. Присвоим значения по умолчанию в этом столбце.

```
sqlite> UPDATE Usres_1 SET email = 'not defined';

sqlite> SELECT * FROM Users;
```

Посмотрим, как внести те же изменения в таблицу базы данных, используя DB Browser.

Можно пользоваться интерфейсом и менять значения столбцов. При добавлении столбца выбираем «Модифицировать таблицу», откроется окно для редактирования столбцов.

Простые выборки

Посмотрим, как можно сделать выборки заданных строк из таблиц.

SELECT выбирает данные, удовлетворяющие заданным условиям.

INSERT добавляет новые данные.

UPDATE изменяет существующие данные.

DELETE удаляет данные.

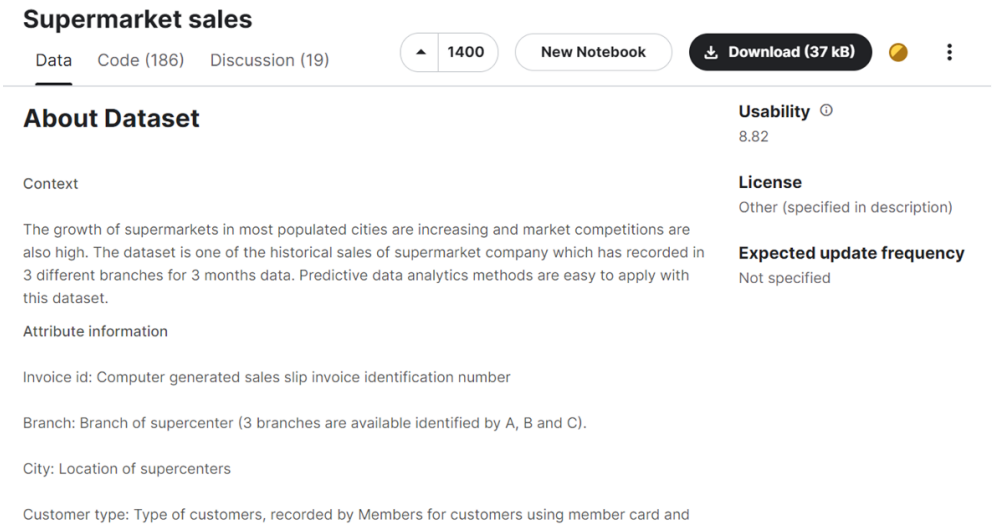
Структура запроса	**Пример**

`SELECT column1, column2, ...`
`FROM table_name`
`WHERE condition;`
	`SELECT CustomerID, Country`
	`FROM Customers`
	`WHERE Country = “United Kingdom”;`

Практика

В практической части будем работать с датасетом по продажам товаров.

Пример датасета по продажам товаров:



Скриншот с сайта kaggle.com с данными Supermarket sales

Как создать таблицу и как формировать выборки по условиям с использованием оператора SELECT:

1. Открываем открытый датасет https://www.kaggle.com/datasets/aungpyaeap/supermarket-sales.
2. Пополнять запросы мы будем при помощи DB Browser.
3. Напишем выбор всех строк из таблицы по продажам, для которых значение поля Branch равно A.

```
SELECT * FROM supermarket_sales
WHERE Branch = "A";
```

4. Добавим условие.

```
SELECT * FROM supermarket_sales
WHERE Branch = "A" AND Productline = "Electronic accessories";
```

5. При замене оператора AND на OR количество возвращенных строк не уменьшится.
6. Мы можем применять логические операторы, если поле, по которому делаем выборку, числовое. Например, наложить ограничение на выборку по цене единицы товара.

```
SELECT * FROM supermarket_sales
WHERE ((Unitprice <= 50) AND (Unitprice >=10)) OR Customertype =
"Normal";
```

7. Можно объединять неограниченное количество условий. Получим список уникальных городов.

```
SELECT DISTINCT(City) FROM supermarket_sales
WHERE Customertype = "Normal";
```