

Базы данных и язык SQL

Егор Суворов

Курс «Парадигмы и языки программирования», подгруппа 3

Понедельник, 4 декабря 2017 года

План занятия

- 1 Основы основ
 - Что такое СУБД
 - ДЗ и простой SQL
 - Использование СУБД
 - Бонус: доступ из кода
- 2 Запросы посложнее
 - Группировка строк
 - Подзапросы
 - Соединения
 - Объединения результатов запросов

1 Основы основ

- Что такое СУБД
- ДЗ и простой SQL
- Использование СУБД
- Бонус: доступ из кода

2 Запросы посложнее

- Группировка строк
- Подзапросы
- Соединения
- Объединения результатов запросов

Постановка задачи

- Пусть мы пишем приложения для учёта товаров в магазине.
- Надо знать:
 - 1 Какой товар есть на складе и витринах.
 - 2 Где он лежит.
 - 3 По какой цене товар закуплен (могут быть разные партии).
 - 4 По какой цене товар сейчас продаётся.
 - 5 Какие покупки были сделаны (что куплено вместе, на какую сумму, в какое время).
- Возможные события:
 - 1 Приехала поставка со склада.
 - 2 Касса пробила чек — совершена покупка.
- Надо, чтобы приложение сохраняло состояние между перезапусками.
- Вопрос: как это сделать?

Усложнения

- Запуск должен быть быстрый
- Данные могут не помещаться в память
- Может случайно отключаться электричество
- Десять касс и два компьютера в разных концах склада.
- Хочется получать обновления «как только так сразу»
- Иногда может теряться связь между кассами и складом
- Может меняться формат хранения (например, добавили бонусы за товар)
- Кассы не должны уметь как угодно менять данные

СУБД

- Система управления базами данных (СУБД) — это сервис, который умеет хранить данные произвольной структуры (в определённых рамках, конечно, не совсем бессистемные).
- База данных — это описание данных и их структуры, которые хранятся в СУБД.
- Реляционные БД — это классика (существуют с 80-х годов), их и будем изучать.
- Обычно запросы к реляционным СУБД формулируются на декларативном языке SQL (Structured Query Language).
- Примеры реляционных: MySQL, MariaDB, Oracle, MS SQL, Sqlite.
- Примеры нереляционных (нынче модно): MongoDB, Redis, Memcached, Cassandra.

В чём плюсы

- Можно выделить один (или несколько) больших серверов под хранение обработку данных; для всех приложений в организации
- Автоматически получаем единый контроль доступа, бэкапы, масштабирование и прочие плюшки
- SQL все знают и могут делать запросы к БД напрямую (для отладки/отчётов)
- Сложнее посадить баг в коде и уронить приложение

В чём минусы

- Мы отдаём контроль за скоростью выполнения и потреблением памяти в руки СУБД (как и при любой абстракции). Это обычно приемлемый компромисс.
- Приложение сложнее запустить: нужно настроить СУБД, что обычно занимает несколько шагов. В нестандартных ситуациях — больше.
- Иногда приложение требует слишком хитрую настройку СУБД (например, для корректной работы с не-латиницей и датами).
- Многие инструменты заточены под промышленные решения и имеют слишком много рычажков и кнопок для простых целей.

Встраиваемые СУБД

- Самая известная встраиваемая СУБД — `sqlite`.
- Предназначена не для сетевого доступа, а для использования в рамках одной конкретной программы.
- Её можно просто вкомпилировать в своё приложение, не требуется никакой настройки.
- `sqlite` хранит каждую БД в отдельном файле определённого формата (последний — `sqlite3`).
- Формат `sqlite3` один на все приложения, можно даже залезть в чужие БД и посмотреть.
- Занимает мало места в скомпилированном приложении.
- Используется **во многих приложениях**: под Android, в Firefox, в Chrome, в клиенте Dropbox¹...
- Задание: скачайте файл `example.sqlite3`

¹ищите файлы `.db`, `.sqlite`, `.sqlite3`

Реляционные СУБД на практике

- СУБД хранит одну или несколько независимых БД (баз данных).
- Каждая БД — это набор *таблиц* («отношений»), которые содержат данные.
- Таблица имеет фиксированный набор столбцов с названиями и типами.
- Фиксированный в каждый момент времени; вообще столбцы можно добавлять, менять, удалять, хоть это и сложные для СУБД операции.
- В таблице лежит неупорядоченный набор строк с данными.
- На столбцы (или их группы) могут накладываться дополнительные ограничения (например, «все значения в столбце различны»).

Типы данных

Смотрим на таблицу Country:

- `INTEGER` — целое число (размер варьируется от СУБД к СУБД, как и название).
- `REAL` — вещественное число с плавающей запятой.
- `VARCHAR(45)` — строка произвольной длины, но не длиннее 45 символов². Если не влезает — поведение зависит от СУБД.

Вообще говоря, конкретно в SQLite значения в столбце могут иметь тип, не совпадающий со столбцом, но об этом лучше не думать.

За кадром остались типы:

- `BLOB` — бинарные данные любой длины.
- `TEXT` — строка произвольной длины без ограничений.
- `CHAR(10)` — строка фиксированной длины (может работать быстрее).

²А сколько символов занимает буква «ш»? А в какой кодировке?

Прочие типы данных

В каждой СУБД свои типы, они могут отличаться по поведению даже просто при разных настройках внутри одной базы данных. Но обычно они называются приблизительно так:

- `DECIMAL(10, 5)` — вещественное число с фиксированной запятой.
- Вариации на тему целых чисел: `SMALLINT`, `MEDIUMINT`, ...
- `FLOAT` — альтернатива `DOUBLE`.
- `DATE`, `TIME`, `DATETIME`, `TIMESTAMP` и вариации для хранения дат ³.

Мораль двух слайдов: сразу сказать, какой тип «правильный» в конкретной ситуации нельзя, надо хорошо понимать предметную область и СУБД, с которой вы работаете. Но для своих проектов по умолчанию можно ограничиться теми типам, что проще называются (предыдущий слайд).

³Правильная работа с датами — тема отдельной лекции: [1](#), [2](#) 

Демонстрация SQL-запросов

- Запросы отделяются между собой точкой с запятой.
- Иногда интерфейс к СУБД не умеет делать несколько запросов одновременно и тогда точка с запятой не нужна.
- Комментарии — либо два дефиса в начале строки, либо `/* ... */`
- Результат запроса `SELECT` — тоже таблица, полученная из исходной.
- Наборы строк и столбцов в результате `SELECT` могут разительно отличаться от исходной.
- Можно фильтровать строки по условиям.
- Можно попросить не строки, а какую-то статистику.
- Если вы в SQL что-то написали, оно практически всегда либо упадёт при компиляции, либо как-то отработает на любых значениях.

LIMIT и OFFSET

- Порядок строк в таблицах и результате работы `SELECT` не определён.
- Но его можно явно задать при помощи `ORDER BY`, который сравнивает поля лексикографически в указанном порядке.
- Важно задавать `ORDER BY` так, чтобы он всегда отличал две строки. Этого можно добиться, только если мы знаем, какие наборы столбцов всегда отличаются.
- Другая стандартная проблема: пусть мы подгружаем бесконечную ленту новостей запросом `LIMIT 10 OFFSET already_loaded`.

LIMIT и OFFSET

- Порядок строк в таблицах и результате работы `SELECT` не определён.
- Но его можно явно задать при помощи `ORDER BY`, который сравнивает поля лексикографически в указанном порядке.
- Важно задавать `ORDER BY` так, чтобы он всегда отличал две строки. Этого можно добиться, только если мы знаем, какие наборы столбцов всегда отличаются.
- Другая стандартная проблема: пусть мы подгружаем бесконечную ленту новостей запросом `LIMIT 10 OFFSET already_loaded`.
- Новости в ленте могут добавляться и удаляться, и номера строк даже в идеально отсортированной таблице постоянно меняются.

Мораль: не стоит надеяться на номера строк, `LIMIT` и `OFFSET` обычно возникают только при выборке «топ-10».

Правильное решение задачи с лентой:

LIMIT и OFFSET

- Порядок строк в таблицах и результате работы `SELECT` не определён.
- Но его можно явно задать при помощи `ORDER BY`, который сравнивает поля лексикографически в указанном порядке.
- Важно задавать `ORDER BY` так, чтобы он всегда отличал две строки. Этого можно добиться, только если мы знаем, какие наборы столбцов всегда отличаются.
- Другая стандартная проблема: пусть мы подгружаем бесконечную ленту новостей запросом `LIMIT 10 OFFSET already_loaded`.
- Новости в ленте могут добавляться и удаляться, и номера строк даже в идеально отсортированной таблице постоянно меняются.

Мораль: не стоит надеяться на номера строк, `LIMIT` и `OFFSET` обычно возникают только при выборке «топ-10».

Правильное решение задачи с лентой: «верни топ-10 записей после такой-то новости из ленты».

DELETE и INSERT

Удаление значений:

- `DELETE FROM Country` удалит **все** строки и не почешется.
- Надо писать `DELETE FROM Country WHERE ...`
- По-хорошему перед `DELETE` стоит сделать `SELECT` и посмотреть.

Добавление значений:

- После слова `VALUES` можно написать несколько кортежей со значениями, но надо знать точный порядок столбцов в таблице.
- На практике столбцы иногда (не часто, но иногда) меняются, добавляются и удаляются, поэтому всегда следует писать явно, каким столбцам что соответствует.
- При вставке может возникнуть ошибка, если на каком-то столбце были ограничения

Остальные запросы

- Также бывают запросы `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE` для работы с таблицами.
- Несмотря на наличие стандартов языка SQL, каждая база дополняет его по-своему, из-за чего получается множество несовместимых диалектов.
- Самые базовые команды (только что были) везде работают *примерно* одинаково (за исключением неоднозначных ситуаций).

NULL

Также есть специальное значение NULL, которое может лежать в любой колонке, если только на ней нет ограничения NOT NULL. Может обозначать:

- Отсутствие каких-либо данных (неизвестно население страны).
- Неопределённый результат вычисления (среднее значение пустого множества, деление на ноль).
- Что угодно ещё по желанию программиста.

Возникающая проблема: нет одного объяснения, как NULL себя ведёт в разных запросах.

- Если считать, что NULL распространяется как NaN (Not a Number), т.е. любое вычисление с NULL даёт NULL, то сложно писать запросы в ситуации, где наличие NULL — норма.
- Если NULL просто игнорировать, то про него легко забыть (так часто и делают); а он может где-то требовать специальной обработки.

Демонстрация NULL

- Разные функции обрабатывает NULL по-разному, общая цель — наиболее консистентное и адекватное поведение.
- Обычно в агрегирующих функциях игнорируется.
- Если вы пишете чуть-чуть несимметричный код (вроде `SUM(a) / COUNT(*)`), могут быть последствия.
- Очень легко забыть и получить какой-то правдоподобный, но неверный результат (особенно в соединениях — будут дальше).

1 Основы основ

- Что такое СУБД
- ДЗ и простой SQL
- **Использование СУБД**
- Бонус: доступ из кода

2 Запросы посложнее

- Группировка строк
- Подзапросы
- Соединения
- Объединения результатов запросов

Анонс домашнего задания

- Вам будет выдан файл с SQL-запросами, которые создают таблицы со странами (структуру разберём) и заполняют их данными.
- Вам нужно написать несколько SQL-запросов SELECT, которые что-то вычисляют.
- Тестировать можно на созданных тестовых данных.
- Как именно тестировать — сейчас покажу.

Консольная утилита

- Называется sqlite3. Это просто программа, которая умеет выполнять SQL-запросы на БД sqlite.
- По умолчанию создаёт пустую БД в памяти.
- Можно попросить открыть существующую БД в файле (или создать новый файл).
- При помощи перенаправления может выполнять SQL из файла.
- SQL-запрос должен заканчиваться точкой с запятой.

Графическая утилита

- Я выбрал [DB Browser for SQLite](#).
- Иногда проще смотреть на таблице в графической оболочке, чем в консоли.
- Может открывать файлы с БД, все изменения идут в памяти.
- Можно откатывать изменения кнопкой «Revert Changes» до последнего сохранения.
- Можно сохранять изменения в файл кнопкой «Write Changes».
- Показывает таблицы, их структура, позволяет выполнять произвольные запросы.

Упражнения

- 1 Скачайте файл [literacy.sql](#).
- 2 Выберите имя для файла, где у вас будет лежать БД для тестов (например, `literacy.sqlite3`).
- 3 Выполните запросы из `literacy.sql`:

```
sqlite3 literacy.sqlite3 < literacy.sql
```

Каждый раз, когда вы будете выполнять команды из `literacy.sql`, таблицы будут полностью пересозданы. Не бойтесь что-то сломать.

- 4 Выполните SQL-запросы (либо в командной строке, либо в GUI):
 - 1 Всю информацию по всем странам.
 - 2 Первые десять стран (если сортировать по названию).
 - 3 Средние население и площадь страны.
 - 4 Площадь и население Франции.
 - 5 Количество французских территорий и их суммарную площадь и население.

Python

```
with sqlite3.Connection("literacy.sqlite3") as db:
    cursor = db.execute("SELECT * FROM Country LIMIT 3")
    print(cursor.description)
    print(list(cursor))
    print(list(cursor))  # Что-нибудь выведет?
```

- Терминология очень похожа во всех языках и СУБД.
- Обычно в языке есть стандартный интерфейс общения с любыми СУБД. А *драйвер* СУБД реализует этот интерфейс в языке.
- Сначала мы устанавливаем *соединение* с СУБД.
- Результатом запроса является *курсор* — это такой итератор по строчкам запроса.
- Что возвращают запросы, кроме SELECT — зависит от СУБД.
- Иногда считается, что не запрос возвращает курсор, а надо сначала создать курсор, а потом в нём выполнить запрос.

SQL-инъекции

Пусть есть таблица с полями: владелец текста, его название, содержимое. Код для доступа к базе, выполняется на сервере:

```
with sqlite3.Connection("sql-injection.sqlite3") as db:
    key = input('Text key: ')
    cursor = db.execute("""SELECT * FROM Text
                          WHERE owner='user' AND key='{}'""".
                          .format(key))

    print(list(cursor))
```

В чём проблема?

SQL-инъекции

Пусть есть таблица с полями: владелец текста, его название, содержимое. Код для доступа к базе, выполняется на сервере:

```
with sqlite3.Connection("sql-injection.sqlite3") as db:
    key = input('Text key: ')
    cursor = db.execute("""SELECT * FROM Text
                          WHERE owner='user' AND key='{}'""".
                          .format(key))

print(list(cursor))
```

В чём проблема?

Значение: key1

Было: SELECT ... WHERE owner='user' AND key='{'

Стало: SELECT ... WHERE owner='user' AND key='key1'

SQL-инъекции

Пусть есть таблица с полями: владелец текста, его название, содержимое. Код для доступа к базе, выполняется на сервере:

```
with sqlite3.Connection("sql-injection.sqlite3") as db:
    key = input('Text key: ')
    cursor = db.execute("""SELECT * FROM Text
                          WHERE owner='user' AND key='{}'""".
                          .format(key))

    print(list(cursor))
```

К коллайдеру!

SQL-инъекции

Пусть есть таблица с полями: владелец текста, его название, содержимое. Код для доступа к базе, выполняется на сервере:

```
with sqlite3.Connection("sql-injection.sqlite3") as db:
    key = input('Text key: ')
    cursor = db.execute("""SELECT * FROM Text
                          WHERE owner='user' AND key='{}'""".
                          .format(key))

    print(list(cursor))
```

К коллайдеру!

Значение: ' OR ''='

Было: SELECT ... WHERE owner='user' AND key='{}'

Стало: SELECT ... WHERE owner='user' AND key='' OR ''='

Упс.

Классический комикс



А как правильно?

```
with sqlite3.Connection("sql-injection.sqlite3") as db:
    key = input('Text key: ')
    cursor = db.execute("""SELECT * FROM Text
                          WHERE owner='user'
                          AND key=?""", [key])
    print(list(cursor))
```

Теперь драйвер базы данных знает, что `key` — это значение от пользователя, которое надо *заэкранировать*:

Значение: `' OR ''='`

Было: `... WHERE owner='user' AND key=?`

Стало: `... WHERE owner='user' AND key='\ ' OR '\ '\ '='`

Независимо от того, какой код мы напишем, SQL-инъекции не случится.

Мораль: никогда не собирайте SQL-запрос руками из переменных.

- 1 Основы основ
 - Что такое СУБД
 - ДЗ и простой SQL
 - Использование СУБД
 - Бонус: доступ из кода
- 2 Запросы посложнее
 - Группировка строк
 - Подзапросы
 - Соединения
 - Объединения результатов запросов

- 1 Основы основ
 - Что такое СУБД
 - ДЗ и простой SQL
 - Использование СУБД
 - Бонус: доступ из кода
- 2 Запросы посложнее
 - Группировка строк
 - Подзапросы
 - Соединения
 - Объединения результатов запросов

GROUP BY — картинка

```

SELECT SUM(Pop), GovForm
FROM Country
WHERE ...
GROUP BY GovForm
HAVING ...

```

	Pop	GovForm	SUM(Pop)	GovForm
	618	Republic	} → 1134	Republic
	455	Republic		
WHERE →	61	Republic	} → 17425	Monarchy
	650	Monarchy		
	17364	Monarch		
	⋮	⋮	⋮	⋮

HAVING →

GROUP BY

- Полезно, когда мы хотим посчитать какую-то статистику по подмножествам строк.
- Каждая агрегирующая функция работает только внутри группы.
- Например, суммарное население стран с разным политическим строем.
- Можно группировать по нескольким полям.
- Условие WHERE применяется до группировки.
- ORDER BY применяется после (так как группировка от порядка не зависит).
- В SELECT можно использовать только агрегирующие функции и колонки, по которым сделана группировка (иначе неясно, из какой строки выбирать). Некоторые СУБД ругаются, некоторые делают что-то.
- Можно дополнительно отфильтровать результаты после группировки при помощи HAVING.

Упражнения

Посчитайте:

- 1 Среднюю площадь страны в зависимости от формы правления.
- 2 Средний квадрат площади страны в зависимости от формы правления.
- 3 Количество стран с площадью порядка миллиона, порядка двух миллионов, и так далее.

- 1 Основы основ
 - Что такое СУБД
 - ДЗ и простой SQL
 - Использование СУБД
 - Бонус: доступ из кода
- 2 Запросы посложнее
 - Группировка строк
 - **Подзапросы**
 - Соединения
 - Объединения результатов запросов

Подзапросы-1

- 1
 - Задача: хотим для города найти население страны, в которой он расположен.
 - В таблице городов есть только код страны, без населения.
 - Можно сделать подзапрос в условии `WHERE`.
- 2
 - Задача: хотим найти средний уровень самой грамотной страны по годам.
 - Надо две агрегатных функции: сначала группируем по годам (чтобы найти победителя), а потом берём среднее.
 - Можно сделать подзапрос в `FROM` (ведь результат `SELECT` — тоже таблица, которую можно назвать).

Подзапросы-2

- Задача: хотим вывести информацию по странам плюс население самого большого города.
- Эта информация лежит в двух разных таблицах.
- SQL позволяет делать `SELECT FROM` сразу из нескольких таблиц (получается декартово произведение).
- Можно взять декартово произведение городов и стран, оставить только соответствующие, а по оставшимся взять агрегирующую функцию.
- Если имена колонок в разных таблицах совпадают, надо явно указывать, к какой мы обращаемся.
- Вообще лучше всегда явно указывать, из какой таблицы мы берём колонку, если таблиц несколько.

Такого сорта выборки происходят очень часто, в реляционной алгебре они зовутся «соединениями» (join).

- 1 Основы основ
 - Что такое СУБД
 - ДЗ и простой SQL
 - Использование СУБД
 - Бонус: доступ из кода
- 2 Запросы посложнее
 - Группировка строк
 - Подзапросы
 - **Соединения**
 - Объединения результатов запросов

Соединения

- Можно считать синтаксическим сахаром для взятия подмножества декартова произведения.
- Лучше отражает суть происходящего и проще читается.
- После `ON` может быть произвольное условие (это круче, чем в реляционной алгебре).
- Обычно там ставят условие «номер страны в первой таблице равен номеру страны во второй таблице».

Соединения — картинка

Cou.Code	Cou.Name		City.CouCode	City.Name
RUS	Russia		RUS	Kaluga
ESP	Spain		RUS	Moscow
HRV	Croatia		HRV	Zagreb
ATA	Antarctica		RUS	Omsk
			ESP	Madrid

Cou.Name	City.Name
Russia	Kaluga
Russia	Moscow
Croatia	Zagreb
Russia	Omsk
Madrid	Madrid

Ключи и соединения-1

- Обычно сущностей в базе много и они как-то связаны отношениями («каждый город лежит ровно в одной стране»).
- Не хочется дублировать информацию в разных таблицах (место занимает, изменять сложно).
- Поэтому информация о стране/городе отдельно.
- А запрос «получи объект по вот этому отношению» возникает.
- Так как свойства объектов часто меняются, то обычно каждому объекту выдают *первичный ключ*, по которому его можно опознать. Обычно это просто какое-то число (возможно, с автоинкрементом).

Ключи и соединения-2

- Тогда связи вроде «в какой стране лежит город» — это просто столбец «номер страны» в таблице с городом.
- Такой столбец называют *внешним ключом*.
- Это даже можно отразить в структуре таблицы (REFERENCES).
- И ещё можно указать, что делать при удалении того объекта, куда мы ссылаемся (ON DELETE CASCADE).

Упражнения

- 1 Вывести для каждой страны максимальный уровень её грамотности за все года.
- 2 Вывести для каждой страны номер города-столицы (см. таблицу Capital).
- 3 Вывести для каждой страны название города-столицы (потребуется два JOIN).

Соединения, NULL, отсутствие значений

- 1 Посчитаем количество стран (239).
 - 2 Теперь для каждой страны посчитаем количество городов.
 - 3 Теперь посмотрим, сколько строк получили в результате — 232.
- А всё потому что есть страны, в которых городов нет — про них в таблице City просто нет информации.
 - И соединение не поможет — страна без городов отфильтруется.
 - Но есть LEFT (OUTER) JOIN — он обязуется добавить в соединение все строчки из «левой» таблицы (а если не нашлось соответствующих строк, то поля второй в строчке соединения будут NULL).
 - С ним надо быть осторожным — потому что теперь в строчках соединения могут оказаться NULL, которые, скорее всего, не надо учитывать (COUNT(*), например, их учтёт).
 - Ещё бывают аналогичные RIGHT JOIN и FULL JOIN (SQLite их не поддерживает).

Соединения — картинка

Cou.Code	Cou.Name	City.CouCode	City.Name
RUS	Russia	RUS	Kaluga
ESP	Spain	RUS	Moscow
HRV	Croatia	HRV	Zagreb
ATA	Antarctica	RUS	Omsk
		ESP	Madrid

Cou.Name	City.Name
Russia	Kaluga
Russia	Moscow
Croatia	Zagreb
Russia	Omsk
Madrid	Madrid
Antarctica	NULL

Классическая шутка



- 1 Основы основ
 - Что такое СУБД
 - ДЗ и простой SQL
 - Использование СУБД
 - Бонус: доступ из кода
- 2 Запросы посложнее
 - Группировка строк
 - Подзапросы
 - Соединения
 - **Объединения результатов запросов**

UNION

- Хотим вывести названия всех географических объектов из БД.
- Делаем два (или больше) `SELECT` с одинаковым количеством столбцов в результате.
- Если объединяем их через `UNION` — то в результате не будет одинаковых строк вообще (даже если они были внутри одного `SELECT`).
- Если объединяем их через `UNION ALL` — то просто все результаты объединятся вместе.

На практике:

- Я ни разу не встречал.
- Может потребоваться, если в БД есть похожие таблицы (например, `UsersUS` и `UsersEU`).