Прикладной Python

Лекция №6

Владимир
Денисов

Базы данных

- Базовые понятия реляционных БД
- Проектирование БД
- SQL. Основные операции: SELECT, INSERT, UPDATE, DELETE, JOIN
- Индексы
- EXPLAIN
- NoSQL

Где хранить данные?

- На клиенте
  - Cookie (4кб)
  - Web Storage
- На сервере
  - В памяти
  - На диске
  - На диске и в памяти

Терминология

- **БД** - Взаимосвязанные данные специальным образом хранящиеся на каком-либо носителе
- **СУБД** – Программный комплекс обеспечивающий работу с данными в БД

Предназначение СУБД

- Управление данными на дисках и в оперативной памяти
- Журнализация, резервное копирование
- Предоставление интерфейсов взаимодействия с БД
- Предоставление механизма транзакций

Реляционная модель данных

- Таблица - отношение, relation
- Строка - кортеж, tuple
- Столбец - атрибут, column

Таблица пользователей

id	first_name	last_name	username
26479	Дмитрий	Васильев	dmitry.vasilyev
26477	Кирилл	Занадворов	k.zanadvorov
26475	Аникеев	Алексей	a.anikeev
26473	Руслан	Сабитов	r.sabitov
26471	Полина	Ольхова	p.olkhova

Терминология

- **Первиичный ключ** (*primary key*) — в реляционной модели данных один из потенциальных ключей отношения, выбранный в качестве основного ключа (или ключа по умолчанию).

  Если в отношении имеется единственный потенциальный ключ, он является и первичным ключом. Если потенциальных ключей несколько, один из них выбирается в качестве первичного, а другие называют «альтернативными».

Терминология

- **Внешний ключ** — это столбец или комбинация столбцов, значения которых соответствуют Первичному ключу в другой таблице. Связь между двумя таблицами задается через соответствие первичного **ключа** в одной из таблиц **внешнему** ключу во второй.

Виды связей в реляционной БД

Связь **один к одному** образуется, когда ключевой столбец (идентификатор) присутствует в другой таблице, в которой тоже является ключом либо свойствами столбца задана его уникальность (одно и тоже значение не может повторяться в разных строках).

Связи **один ко многим** одной записи первой таблицы соответствует несколько записей в другой таблице.

Если нескольким записям из одной таблицы соответствует несколько записей из другой. таблицы, то такая связь называется «**многие ко**

Примеры

Color	
Key	Name
1	Black
2	Pink

Car			
Key	Model	Color	Year
1	Toyota Camry	1	2010
2	Audi A4	2	2013
3	Honda Civic	2	2013

Примеры

Cousine		
Key	Name	Type
1	Baked fish	Main Course
2	Pork Chops	Main Course

Ingredient	
Key	Name
1	Pork
2	Lamb
3	Salmon

Пример

Cousine		
Key	Name	Type
1	Baked fish	Main Course
2	Pork Chops	Main Course

Ingredient	
Key	Name
1	Pork
2	Lamb
3	Salmon

CousineIngredient	
CousineKey	IngredientKey
1	3
2	1

Структура SQL запроса SELECT


```
1.  SELECT
2.     [DISTINCT | DISTINCTROW | ALL]
3.     select_expression,...
4.  FROM table_references
5.  [WHERE where_definition]
6.  [GROUP BY {unsigned_integer | col_name | formula}]
7.  [HAVING where_definition]
8.  [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC],
    ...]
```

Операции SQL: SELECT


```
1.  SELECT * FROM users WHERE age > 10;
2.  SELECT * FROM users WHERE name = 'masha';
3.  SELECT MAX(age) FROM users;

4.  SELECT id, name, LENGTH(name) AS len
5.  FROM users
6.  WHERE email LIKE '%@mail.ru' AND age > 10
7.  ORDER BY name DESC
8.  LIMIT 10 OFFSET 15;
```

Дока по встроенным методам в MySQL
https://dev.mysql.com/doc/refman/8.0/en/string-functions.html

Агрегация


```
1.  SELECT first_name, count(id) as cnt
2.  FROM users_user
3.  WHERE first_name
4.  LIKE "%дим%"
5.  GROUP BY first_name
6.  HAVING cnt > 100
7.  ORDER BY cnt;
```

Агрегатные функции MySQL

AVG: вычисляет среднее значение

SUM: вычисляет сумму значений

MIN: вычисляет наименьшее значение

MAX: вычисляет наибольшее значение

COUNT: вычисляет количество строк в запросе

JOIN


```
1.  SELECT h.name, a.name
2.  FROM heroes h, abilities a
3.  WHERE h.id = a.hero_id;

4.  SELECT h.name, a.name
5.  FROM heroes h
6.  INNER JOIN abilities a ON h.id = a.hero_id;

7.  SELECT h.name, a.name
8.  FROM heroes h
9.  LEFT JOIN abilities a ON h.id = a.hero_id;
```

Вложенные запросы


```
1.  SELECT title
2.  FROM article t1
3.  JOIN (
4.      SELECT rubric_id, MAX(id) max_id
5.      FROM article
6.      GROUP BY rubric_id LIMIT 5
7.  ) t2
8.  ON t1.id = t2.max_id;
```

Операции SQL: INSERT, UPDATE, DELETE


```
1.  INSERT INTO users (name, age) VALUES ('Petr', 10);

2.  UPDATE users SET rating = rating + 1;
3.  UPDATE users SET age = 20 WHERE name = 'Petr';

4.  DELETE FROM users WHERE name = 'Masha';
5.  DELETE FROM users WHERE age > 150;
```

Терминология

- **Индекс** — объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированного дерева.

- **Кластерный индекс** – индекс, хранящий не только значение колонки, но и данные всей строки. Может быть только 1 для таблицы.

По каким полям надо делать индексы

- Индексы для полей, по которым происходит JOIN
- Индексы для полей, по которым фильтруются записи
- Индексы для полей, по которым идет сортировка

Задачи проектирования

- Обеспечение хранения всей необходимой информации
- Обеспечение возможности получения данных по всем запросам
- Сокращение избыточности и дублирования данных
- Обеспечение целостности данных

Типы данных в MySQL

INT - Целое число нормального размера. Диапазон со знаком от -2147483648 до 2147483647. Диапазон без знака от 0 до 4294967295.

DOUBLE - Число с плавающей точкой удвоенной точности нормального размера. Допустимые значения:
от -1,7976931348623157E+308 до -2,2250738585072014E-308, 0, и
от 2,2250738585072014E-308 до 1,7976931348623157E+308. Если указан атрибут UNSIGNED, отрицательные значения недопустимы.

DATE - Дата. Поддерживается интервал от '1000-01-01' до '9999-12-31'. MySQL выводит значения DATE в формате 'YYYY-MM-DD', но можно установить значения в столбец DATE, используя как строки, так и числа. See section 6.2.2.2 Типы данных DATETIME, DATE и TIMESTAMP.

DATETIME - Комбинация даты и времени. Поддерживается интервал от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'.

TIMESTAMP - Временная метка.

TIMEB - Время. Интервал от '-838:59:59' до '838:59:59’.

YEAR - Год в двухзначном или четырехзначном форматах (по умолчанию формат четырехзначный).

Больше типов данных

CHAR(M) [BINARY] - Строка фиксированной длины, при хранении всегда дополняется пробелами в конце строки до заданного размера. Диапазон аргумента M составляет от 0 до 255 символов. Если не задан атрибут чувствительности к регистру BINARY, то величины CHAR сортируются и сравниваются как независимые от регистра в соответствии с установленным по умолчанию алфавитом.

CHAR - Это синоним для CHAR(1).

VARCHAR(M) [BINARY] - Строка переменной длины.

TINYBLOB, TINYTEXT - Столбец типа BLOB или TEXT с максимальной длиной

255 (2^8 - 1) символов.

BLOB, TEXT - Столбец типа BLOB или TEXT с максимальной длиной 65535 (2^16 - 1) символов.

Больше типов и более подробная дока:

http://www.mysql.ru/docs/man/Column_types.html

Проектируем БД

Спроектировать базу данных для магазина

Анализ запросов: EXPLAIN

- Ничего не говорит о том как влияют на запросы триггеры.
- Не работает с хранимыми процедурами (хотя можно разложить процедуру на запросы и выполнить каждый из них)
- Ничего не говорит об оптимизациях на этапе выполнения запроса
- Часть отображаемой информации оценочная.

Explain: id

EXPLAIN select * from users_car where id LIKE "1%"

```
+----+-------------+-----------+------------+------+---------------+
| id | select_type | table     | partitions | type | possible_keys |
+----+-------------+-----------+------------+------+---------------+
|  1 | SIMPLE      | users_car | NULL       | ALL  | PRIMARY       |
+----+-------------+-----------+------------+------+---------------+
```

EXPLAIN select *, (SELECT 1 from users_user) from users_car where id LIKE "1%";

```
+----+-------------+-----------+------------+-------+---------------+
| id | select_type | table     | partitions | type  | possible_keys |
+----+-------------+-----------+------------+-------+---------------+
|  1 | PRIMARY     | users_car | NULL       | ALL   | PRIMARY       |
|  2 | SUBQUERY    | users_user| NULL       | index | NULL          |
+----+-------------+-----------+------------+-------+---------------+
```

EXPLAIN SELECT 1 UNION ALL SELECT 1;

```
+----+-------------+-------+------------+------+
| id | select_type | table | partitions | type |
+----+-------------+-------+------------+------+
|  1 | PRIMARY     | NULL  | NULL       | NULL |
|  2 | UNION       | NULL  | NULL       | NULL |
+----+-------------+-------+------------+------+
```

Explain: select_type

id	select_type	table	partitions	type	possible_keys
1	PRIMARY	users_car	NULL	ALL	PRIMARY
2	SUBQUERY	users_user	NULL	index	NULL

SIMPLE – Простой запрос SELECT без подзапросов или UNION

PRIMARY - Самый внешний запрос

SUBQUERY - Запрос Select, который содержится в подзапросе (не в From)

DERIVED - Значение DERIVED означает, что запрос SELECT является подзапросом в фразе FROM

UNION - Второй и последуюие запросы SELECT входящие в объединение union помечаются признаком UNION

UNION RESULT - Запрос SELECT, применяемый для выборки данных из временной таблицы созданной в ходе выполнения UNION

Explain: table

EXPLAIN select * from users_car where id LIKE "1%"

```
+----+-------------+-----------+------------+------+---------------+
| id | select_type | table     | partitions | type | possible_keys |
+----+-------------+-----------+------------+------+---------------+
|  1 | SIMPLE      | users_car | NULL       | ALL  | PRIMARY       |
+----+-------------+-----------+------------+------+---------------+
```

EXPLAIN select * from users_car as uc join users_user as uu on uc.user_id=uu.id;

```
+----+-------------+-------+------------+--------+
| id | select_type | table | partitions | type   |
+----+-------------+-------+------------+--------+
|  1 | SIMPLE      | uc    | NULL       | ALL    |
|  1 | SIMPLE      | uu    | NULL       | eq_ref |
+----+-------------+-------+------------+--------+
```

Explain: type


```
+----+-------------+-------+------------+--------+
| id | select_type | table | partitions | type   |
+----+-------------+-------+------------+--------+
|  1 | SIMPLE      | uc    | NULL       | ALL    |
|  1 | SIMPLE      | uu    | NULL       | eq_ref |
+----+-------------+-------+------------+--------+
```

ALL - Этот подход обычно называют сканированием таблицы.

index - То же, что и сканирование таблицы, но MySQL просматривает записи в порядке задаваемом индексом, а не в порядке следования строк.

range - Просмотр диапазона – неполное сканирование индекса.

ref - доступ по индексу, возвращает строки соответствующие единственному заданному значению

eq_ref - поиск по индексу в случае если MySQL точно знает, что будет возвращено 1 значение.

Null - запрос на фазе оптимизации разрешен так, что не потребовалось обращаться к таблицам базы данных

Explain: possible_keys, key

EXPLAIN select id from users_user\G;

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: users_user
   partitions: NULL
         type: index
possible_keys: NULL
          key: users_user_c9e9a848
      key_len: 4
          ref: NULL
         rows: 12024
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0,01 sec)
```

NoSQL Rising

Общие характеристики NoSQL БД

- Не используют реляционную модель
- Хорошо подходят для развертывания на кластере
- Open-source
- Schemaless

NoSQL: key-value СУБД

Кейсы применения БД хранилищ ключ-значение:

- Кеширование - быстрое и частое сохранение данных для будущего использования
- Очередь - некоторые БД типа ключ-значение поддерживают списки, наборы и очереди
- Распределение информации/задач - используется для реализации паттерна Pub/Sub
- Живое обновление информации - приложения использующие состояния

Популярные решения:

Memcached / MemcacheDB - распределённая БД в оперативной памяти

Redis - БД в оперативной памяти с поддержкой структур данных и возможностью выполнять операции на данных

…

NoSQL: распределенные СУБД

Кейсы применения распределенных СУБД:

Хранение неструктурированных, не разрушаемых данных - если вам необходимо хранить большие объемы данных в течение долгого времени, то такие БД очень хорошо справятся с задачей

Масштабирование - по задумке такие базы данных легко масштабируются.

Популярные СУБД:

Cassandra - структура данных основана на BigTable и DynamoDB

HBase - хранилище для Apache Hadoop основанное на принципах BigTable

NoSQL: документоориентированные СУБД

Кейсы применения документоориентированные СУБД:

Популярные СУБД

MongoDB - очень популярное и функциональное хранилище

Couchbase - основанное на JSON, совместимое с Memcached хранилище

CouchDB - передовое документо-ориентированное хранилище

NoSQL: СУБД типа граф

Кейсы применения распределенных СУБД:

работа со сложно связанной информацией. Например граф знакомств в соц сети.

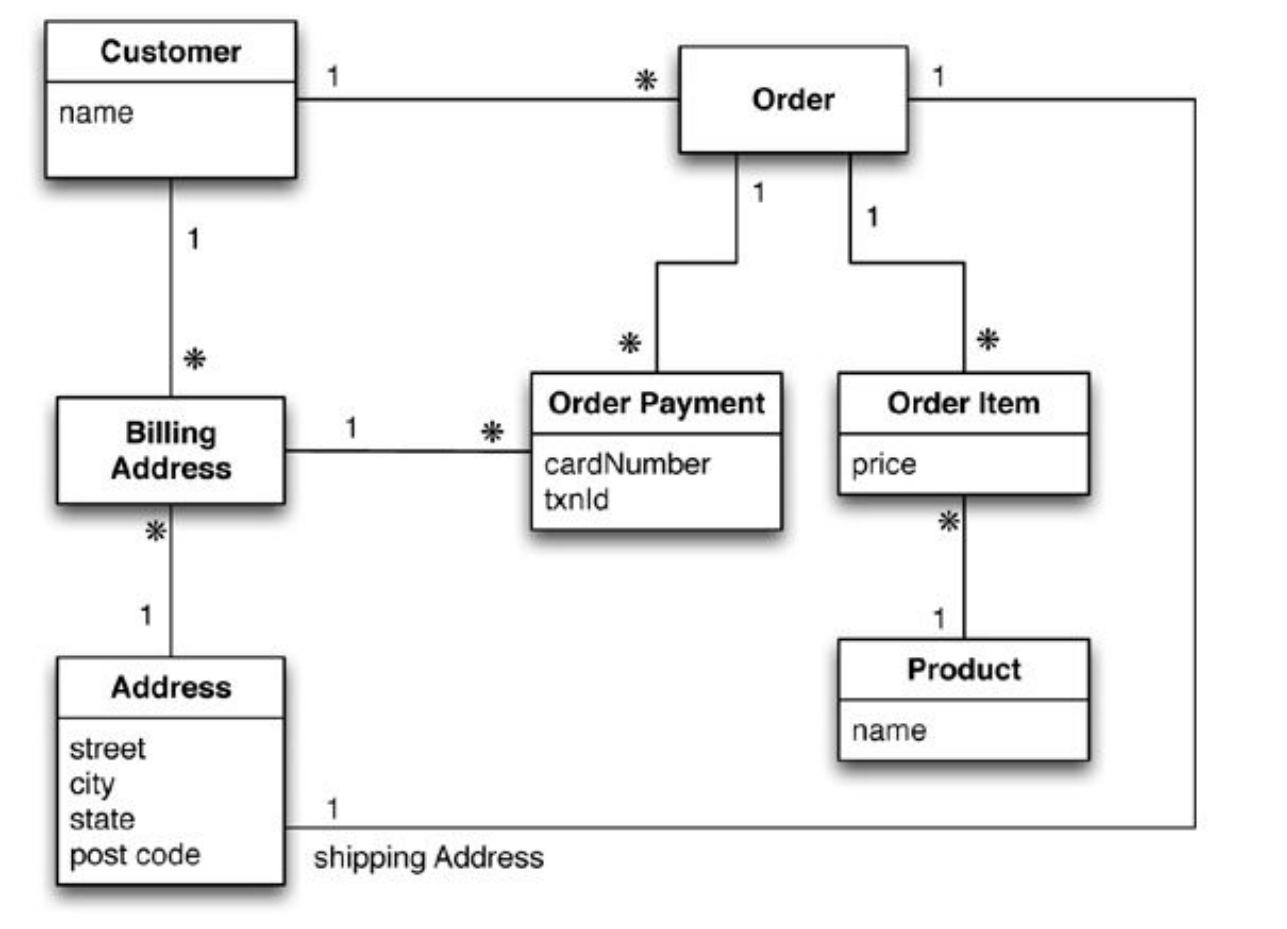
Моделирование и поддержка классификаций - такие БД преуспели везде где есть связи. Моделирование данных и классификация различной информации по связям можно с легкостью представить используя эти БД.

Популярные СУБД

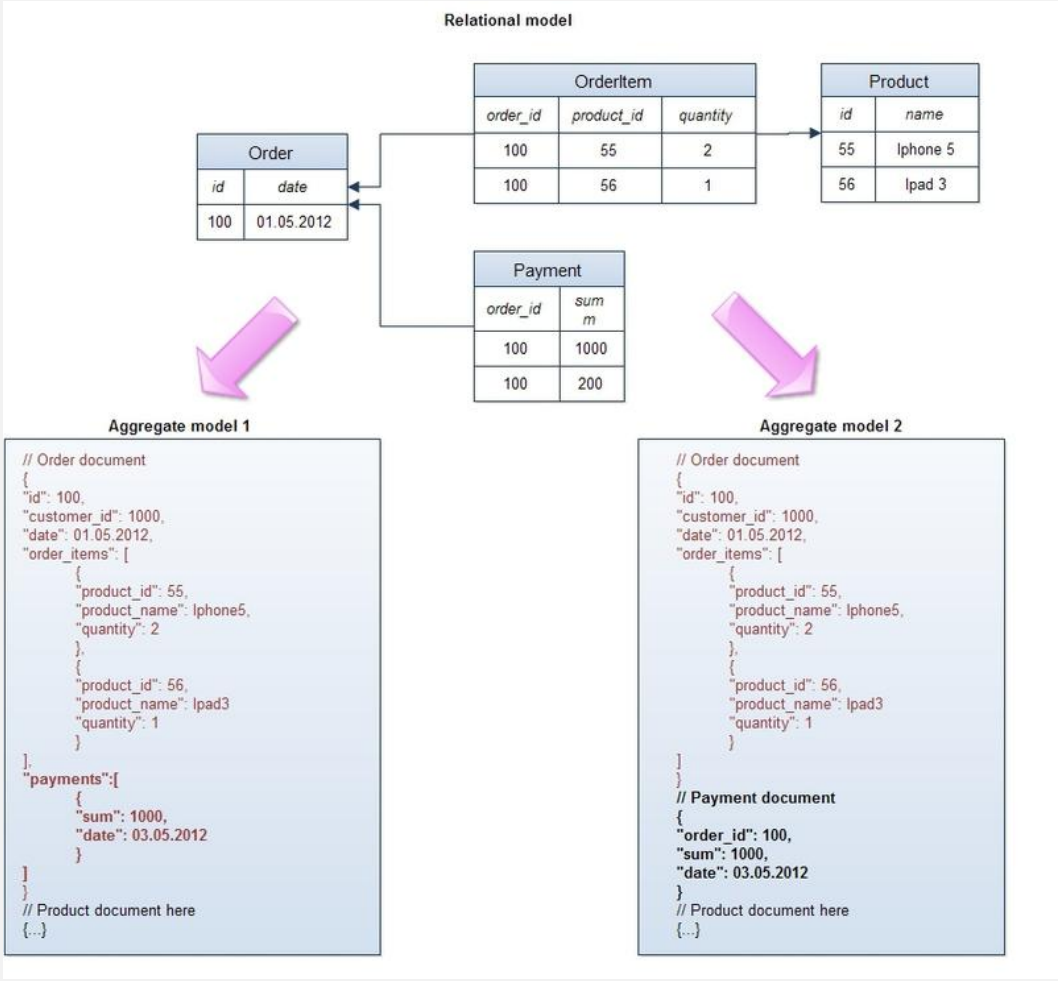
OrientDB - очень быстрое документо-ориентированное хранилище гибрид типа граф написанное на Java. Включает в себя разные режимы работы

Neo4J - безсхемное, очень мощное и популярное хранилище написанное на Java

SQL подход к проектированию БД



NoSQL подход



Разница подходов

Нормализация данных	Данные в виде агрегатов
• Целостность информации при обновлении (меняем запись в одной таблице, а не в нескольких) • Ориентированность на широкий спектр запросов к данных	• Оптимизация только под определенный вид запросов • Сложности при обновлении денормализованных данных
• Неэффективна в распределенной среде • Низкая скорость чтения при использовании объединений (joins) • Несоответствие объектной модели приложения физической структуре данных (impedance mismatch, решается с помощью Hibernate etc.)	• Лучший способ добиться большой скорости на чтение в распределенной среде • Возможность хранить физически объекты в том виде, в каком с ними работает приложение (легче кодировать и меньше ошибок при преобразовании) • Родная (native) поддержка атомарности на уровне записей