НЕТОЛОГИЯ
групп

ЗАНЯТИЕ 1.2

# Работа с базами данных


## Алексей Кузьмин

Директор разработки; Data Scientist

**ДомКлик.ру**

aleksej.kyzmin@gmail.com

# О ЧЁМ ПОГОВОРИМ И ЧТО СДЕЛАЕМ

- Понятие простой БД
- Где располагаются таблицы
- Устройство реляционной СУБД
- Какие БД по умолчанию есть в Postgress
- Понятие схемы, отношения, атрибута, домена
- Основные типы данных PostgreSQL
- Запрос данных и пересечения
- Фильтрация результатов запросов

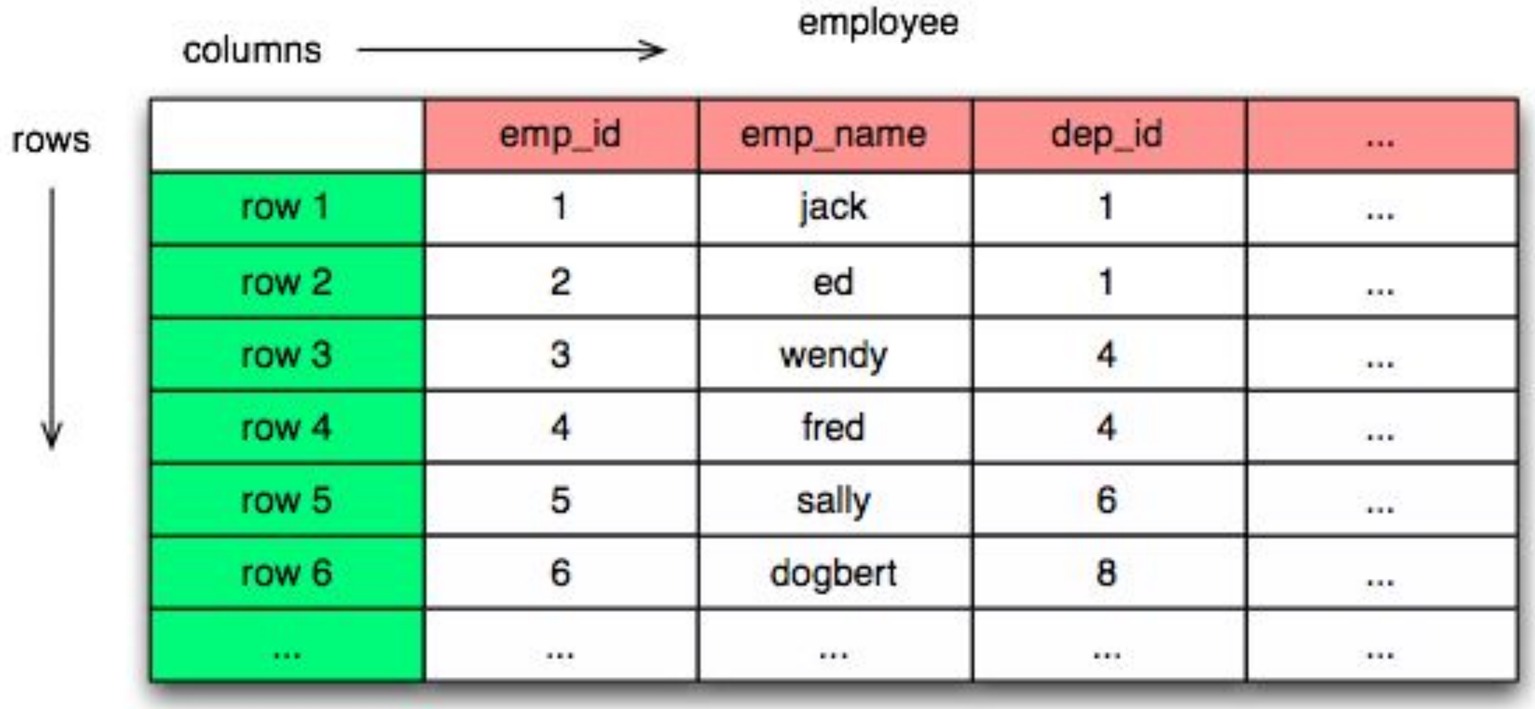
# РЕЛЯЦИОННАЯ МОДЕЛЬ

## Реляционная модель

Модель представляет собой фиксированную структуру математических понятий, которая описывает то, как будут представлены данные. Базовой единицей данных в пределах реляционной модели является таблица.

**Таблица**

Таблица - это базовая единица данных. В реляционной алгебре она называется «отношение» (relation). Состоит из столбцов (columns), которые определяют конкретные типы данных. Данные в таблице организованы в строки (rows), которые содержат множества значений столбцов.




| | emp_id | emp_name | dep_id | ... |
|---|---|---|---|---|
| row 1 | 1 | jack | 1 | ... |
| row 2 | 2 | ed | 1 | ... |
| row 3 | 3 | wendy | 4 | ... |
| row 4 | 4 | fred | 4 | ... |
| row 5 | 5 | sally | 6 | ... |
| row 6 | 6 | dogbert | 8 | ... |
| ... | ... | ... | ... | ... |

## Первичные ключи

При создании таблицы могут быть использованы различные **«ограничения»** (constraints), которые содержат правила, указывающие, какие данные представлены в ней. Одним из самых используемых ограничений является **первичный ключ** (primary key constraint), который гарантирует, что каждая строка таблицы должна содержать уникальное значение. Первичный ключ может состоять из одного или нескольких столбцов. Первичные ключи, состоящие из нескольких столбцов, также называются **«составными»** (composite).

Правильным считается наличие первичного ключа во всех таблицах базы данных. При этом существует два варианта первичных ключей: **искусственный (суррогатный)** (surrogate primary key) и **естественный (натуральный)** (natural primary key).

Первый вариант обычно представляет собой целочисленный идентификатор. Применяется там где нет возможности использовать натуральный первичный ключ. Позволяют решать те же практические задачи, что и естественные: улучшение производительности памяти и индексов при операциях обновления.

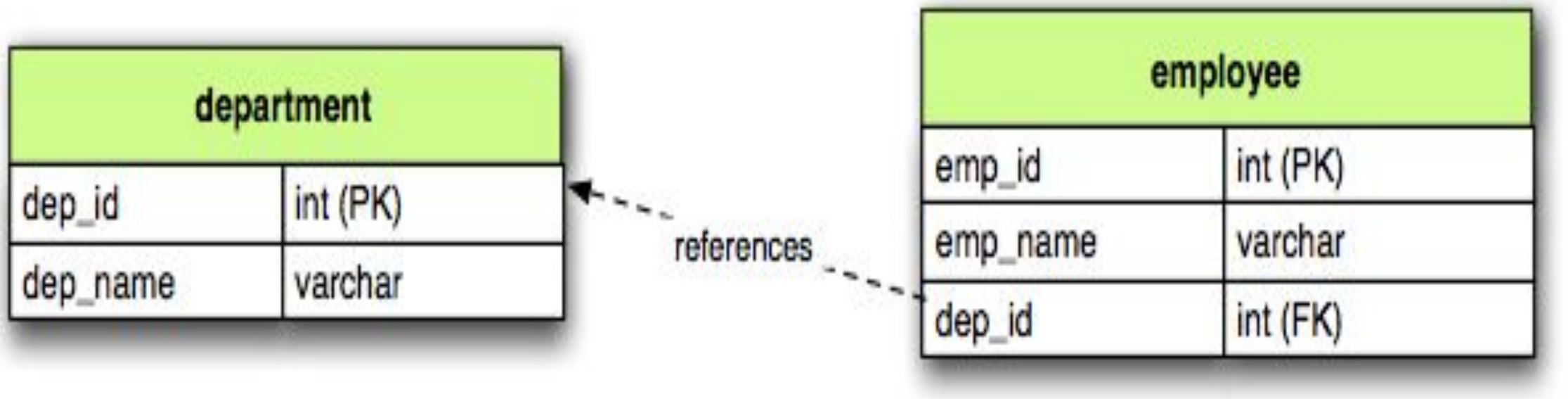
Второй же вариант представляет собой данные, которые уже присутствуют в описываемой предметной области. Например, почтовые индексы могут быть использованы как естественные первичные ключи без дополнительной обработки. Их использование, если, конечно, оно возможно, считается более правильным, чем искусственных.

## Внешние ключи

В то время как одна таблица имеет первичный ключ, другая таблица может иметь ограничение, описывающее, что её строки ссылаются на гарантированно существующие строки в первой таблице. Это реализуется через создание в **«удалённой»** таблице («потомке») столбца (может быть и несколько), значениями которого являются значения первичного ключа из **«локальной»** таблицы («родителя»). Вместе наборы этих столбцов составляют **внешний ключ** (foreign key constraint), который является механизмом базы данных, гарантирующим, что значения в «удалённых» столбцах присутствуют как первичные ключи в «локальных». Это ограничение контролирует все операции на этих таблицах: добавление / изменение данных в «удалённой» таблице; удаление / изменение данных в «родительской» таблице. Внешний ключ проверяет, чтобы данные корректно присутствовали в обеих таблицах. Иначе операции будут отменены.

Внешние ключи могут быть составными, если входящие в них первичные ключи являются таковыми.

В примере представлена таблица «department», которая связана с таблицей «employee» через отношение столбцов «employee.dep_id» и «department.dep_id»:



## Нормализация

Реляционная модель базируется на реляционной алгебре, одним из ключевых понятий которой является нормализация.

Основная идея нормализации в исключении повторяющихся данных так, чтобы конкретная часть данных была представлена только в одном месте. Этот подход позволяет упростить данные до максимально атомарного вида, с которым намного проще работать: искать, производить какие-либо операции.

Классический пример денормализованных данных:

| name | language | department |
|---|---|---|
| Dilbert | C++ | Systems |
| Dilbert | Java | Systems |
| Wally | Python | Engineering |
| Wendy | Scala | Engineering |
| Wendy | Java | Engineering |

Строки в этой таблице могут быть уникально идентифицированы по столбцам "name" и "language", которые являются потенциальным ключом. По теории нормализации таблица из примера нарушает вторую нормальную форму. Потому как не основной атрибут "department" логически связан только со столбцом "name". Правильная нормализация в данном случае выглядит следующим образом:

| name | department |
|---|---|
| Dilbert | Systems |
| Wally | Engineering |
| Wendy | Engineering |

| name | language |
|---|---|
| Dilbert | C++ |
| Dilbert | Java |
| Wally | Python |
| Wendy | Scala |
| Wendy | Java |

Теперь наглядно видно, как вторая форма улучшила структуру данных. Изначально пример содержал повторы связок полей "name" и "department" так часто, как часто встречался уникальный для данного имени "язык". Улучшенный же вариант сделал связки "name/department" и "name/language" независимыми друг от друга.

Ограничения данных, такие как первичные и внешние ключи, предназначены как раз для достижения состояния нормализации. Для примера выше это будет выглядеть так:

- "Employee Department -> name" - первичный ключ;
- "Employee Language -> name, language" - составной первичный ключ;
- "Employee Language -> name", в свою очередь, - внешний ключ, на поле "Employee Department -> name".

Если таблицу удаётся сходу свернуть в отношения ключей, то это, зачастую, значит, что она не нормализована.

# ЗАПРОСЫ

Для извлечения данных из БД применяется команда SELECT. В упрощенном виде она имеет следующий синтаксис:

**SELECT** список_столбцов **FROM** имя_таблицы;

Например, пусть ранее была создана таблица Products, и в нее добавлены некоторые начальные данные:

| ProductName | Manufacturer | ProductCount | Price |
|---|---|---|---|
| 'iPhone 8' | 'Apple' | 2 | 41000 |
| 'iPhone X' | 'Apple' | 3 | 36000 |
| 'Galaxy S9' | 'Samsung' | 2 | 46000 |
| 'Galaxy S8 Plus' | 'Samsung' | 1 | 56000 |
| 'Desire 12' | 'HTC' | 5 | 28000 |

получить все объекты из этой таблицы можно командой:

```
SELECT * FROM Products;
```

Символ ***** указывает, что нам надо получить все столбцы.

Однако использование символа звездочки *считается не очень хорошей практикой, так как, как правило, не все столбцы бывают нужны. И более оптимальный подход заключается в указании всех необходимых столбцов после слова SELECT. Исключение составляет тот случай, когда надо получить данные по абсолютно всем столбцам таблицы. Также использование символа* может быть предпочтительно в таких ситуациях, когда в точности не известны названия столбцов.

Если нам надо получить данные не по всем, а по каким-то конкретным столбцам, то тогда все эти спецификации столбцов перечисляются через запятую после SELECT:

```
SELECT ProductName, Price FROM Products;
```

Спецификация столбца не обязательно должна представлять его название. Это может быть любое выражение, например, результат арифметической операции. Рассмотрим следующий запрос:

```
SELECT ProductCount, Manufacturer, Price * ProductCount
FROM Products;
```

Здесь при выборке будут создаваться три столбца. Причем третий столбец представляет значение столбца Price, умноженное на значение столбца ProductCount, то есть совокупную стоимость товара.

С помощью оператора AS можно изменить название выходного столбца или определить его псевдоним:

```
SELECT ProductCount AS Title,
Manufacturer,
Price * ProductCount  AS TotalSum
FROM Products;
```

# ВРЕМЯ ПРАКТИКИ

**Практика 1. Работаем с базой dvdrental**

- Получите атрибуты id фильма, название, описание, год релиза из таблицы фильмы. Переименуйте поля так, чтобы все они начинались со слова Film (FilmTitle вместо title и тп)
- В таблице dvdrental есть два атрибута: **rental_duration** - длина периода аренды в днях и **rental_rate** - стоимость аренды фильма на этот промежуток времени. Для каждого фильма из таблицы film получите стоимость его аренды в день

# ОСНОВНЫЕ ТИПЫ ДАННЫХ

**Числовые типы данных**

- integer: хранит числа от -2147483648 до +2147483647. Занимает 4 байта. Имеет псевдонимы int и int4.
- bigint: хранит числа от -9223372036854775808 до +9223372036854775807. Занимает 8 байт. Имеет псевдоним int8.
- real: хранит числа с плавающей точкой из диапазона от 1E-37 до 1E+37. Занимает 4 байта. Имеет псевдоним float4.
- double precision: хранит числа с плавающей точкой из диапазона от 1E-307 до 1E+308. Занимает 8 байт. Имеет псевдоним float8.

**Символьные типы**

- character(n): представляет строку из фиксированного количества символов. С помощью параметра задается количество символов в строке. Имеет псевдоним char(n).
- character varying(n): представляет строку из не фиксированного количества символов. С помощью параметра задается максимальное количество символов в строке. Имеет псевдоним varchar(n).
- text: представляет текст произвольной длины.

**Типы для работы с датами и временем**

- timestamp: хранит дату и время. Занимает 8 байт. Для дат самое нижнее значение - 4713 г до н.э., самое верхнее значение - 294276 г н.э.
- timestamp with time zone: то же самое, что и timestamp, только добавляет данные о часовом поясе.
- date: представляет дату от 4713 г. до н.э. до 5874897 г н.э. Занимает 4 байта.
- time: хранит время с точностью до 1 микросекунды без указания часового пояса. Принимает значения от 00:00:00 до 24:00:00. Занимает 8 байт.
- time with time zone: хранит время с точностью до 1 микросекунды с указанием часового пояса. Принимает значения от 00:00:00+1459 до 24:00:00-1459. Занимает 12 байт.
- interval: представляет временной интервал. Занимает 16 байт.

**Логический тип**

- Тип boolean может хранить одно из двух значений: true или false.
- Вместо true можно указывать следующие значения: TRUE, 't', 'true', 'y', 'yes', 'on', '1'.
- Вместо false можно указывать следующие значения: FALSE, 'f', 'false', 'n', 'no', 'off', '0'.

## Специальные типы данных

- json: хранит данные json в текстовом виде.
- jsonb: хранит данные json в бинарном формате.
- xml: хранит данные в формате XML
- массивы

**Операции над типами данных**

- Математические - https://www.postgresql.org/docs/9.2/functions-math.html
- Строковые - https://www.postgresql.org/docs/9.2/functions-string.html
- Логические - https://www.postgresql.org/docs/9.2/functions-logical.html

# УСЛОЖНЯЕМ ВЫБОРКУ ДАННЫХ

## Сортировка

Оператор **ORDER BY** позволяет отсортировать значения по определенному столбцу. Например, упорядочим выборку из таблицы Products по столбцу ProductCount:

```
SELECT * FROM Products
ORDER BY ProductCount;
```

По умолчанию данные сортируются по возрастанию, однако с помощью оператора **DESC** можно задать сортировку по убыванию. Явно задать сортировку по возрастанию можно указав оператор **ASC**.

```
SELECT * FROM Products
ORDER BY ProductCount DESC;
```

## Выборка уникальных значений

Оператор **DISTINCT** позволяет выбрать уникальные данные по определенным столбцам.

```
SELECT DISTINCT Manufacturer FROM Products;
```

## Фильтрация

Для фильтрации данных применяется оператор **WHERE**, после которого указывается условие, на основании которого производится фильтрация.

Если условие истинно, то строка попадает в результирующую выборку. В качестве можно использовать операции сравнения. Эти операции сравнивают два выражения. В PostgreSQL можно применять следующие операции сравнения:

- = сравнение на равенство
- <> сравнение на неравенство
- < меньше чем
- > больше чем
- !< не меньше чем
- !> не больше чем
- <= меньше чем или равно
- >= больше чем или равно

Например, найдем всех товары, производителем которых является компания Apple:

```
SELECT * FROM Products
WHERE Manufacturer = 'Apple';
```

Чтобы объединить нескольких условий в одно, в PostgreSQL можно использовать логические операторы:

- **AND** операция логического И. Она объединяет два выражения. Только если оба этих выражения одновременно истинны, то и общее условие оператора AND также будет истинно. То есть если и первое условие истинно, и второе.
- **OR** операция логического ИЛИ. Она также объединяет два выражения. Если хотя бы одно из этих выражений истинно, то общее условие оператора OR также будет истинно. То есть если или первое условие истинно, или второе.
- **NOT** операция логического отрицания. Если выражение в этой операции ложно, то общее условие истинно.

Например, выберем все товары, у которых производитель Samsung и одновременно цена больше 50000:

```
SELECT * FROM Products
WHERE Manufacturer = 'Samsung' AND Price > 50000;
```

# ВРЕМЯ ПРАКТИКИ

**Практика 2**

- Отсортировать список фильмов по убыванию стоимости за день аренды (второе задание первой практики)
- Вывести все уникальные годы выпуска фильмов
- Вывести весь список фильмов имеющий рейтинг 'PG-13'

# ВОПРОСЫ

# Домашнее задание

**Домашняя работа (все задания на базе dvdrental)**

- Перечислить все таблицы и первичные ключи в базе данных dvdrental (проще всего делать по ER-диаграмме)
- Вывести всех неактивных покупателей
- Самостоятельно прочитать про limit и offset в postgresql (например тут - http://www.postgresqltutorial.com/postgresql-limit/). Вывести 10 последних платежей за прокат фильмов (таблица payment)

# Полезные материалы

## ПОЛЕЗНЫЕ МАТЕРИАЛЫ

- https://github.com/ustu/lectures.db/blob/master/docs/databases/rdb.rst
- https://www.postgresql.org/docs/9.2/datatype.html
- https://www.postgresql.org/docs/9.2/functions.html
- Описание базы данных - https://dev.mysql.com/doc/sakila/en/sakila-structure-tables.html

НЕТОЛОГИЯ
групп

# Спасибо за внимание!

Алексей Кузьмин

aleksej.kyzmin@gmail.com