

< Teach
Me
Skills />

Занятие 13.

Базы данных

Вспоминаем теорию

- Что такое БД
- Какие подмножества языка SQL вы знаете
- Какие виды баз данных существуют и чем отличаются
- Что такое СУБД и какими свойствами она должна обладать
- Из чего состоит БД
- Какие типы данных поддерживаются БД
- Что такое первичный ключ и каким он может быть
- Как создать первичный ключ
- Как можно отфильтровать данные
- При помощи какой команды можно получить данные из БД
- При помощи какой команды можно удалить данные из БД
- Какие функции вы знаете
- Как обновить сразу несколько записей в БД

1

Определение связей

Foreign key

Что это такое?

- внешний ключ позволяет установить связь между таблицами
- как правило внешний ключ ссылается на значение первичного ключа другой таблицы

Какие связи бывают:

- один к одному (one-to-one)
- один ко многим (one-to-many)
- многие ко многим (many-to-many)

```
CREATE TABLE orders (
    id int NOT NULL,
    order_number int NOT NULL,
    customer_id int,
    PRIMARY KEY (id),
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

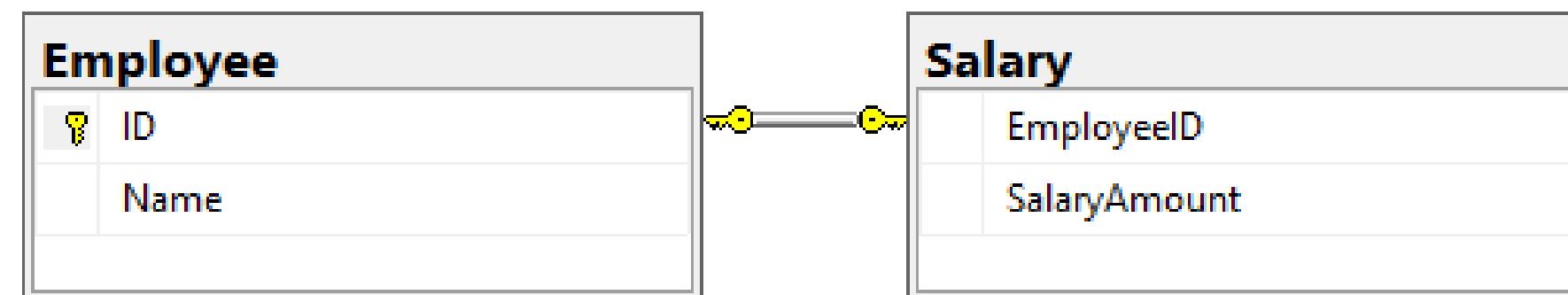
Один к одному (one-to-one)

- такие отношения нужны для того чтобы ограничить количество связанных записей, например, у нас есть таблица с пользователями и мы хотим создать таблицу с зарплатами, каждому пользователю будет соответствовать только 1 запись

```
CREATE TABLE [Employee] (
    [ID]      INT PRIMARY KEY
,   [Name]    VARCHAR(50)
);

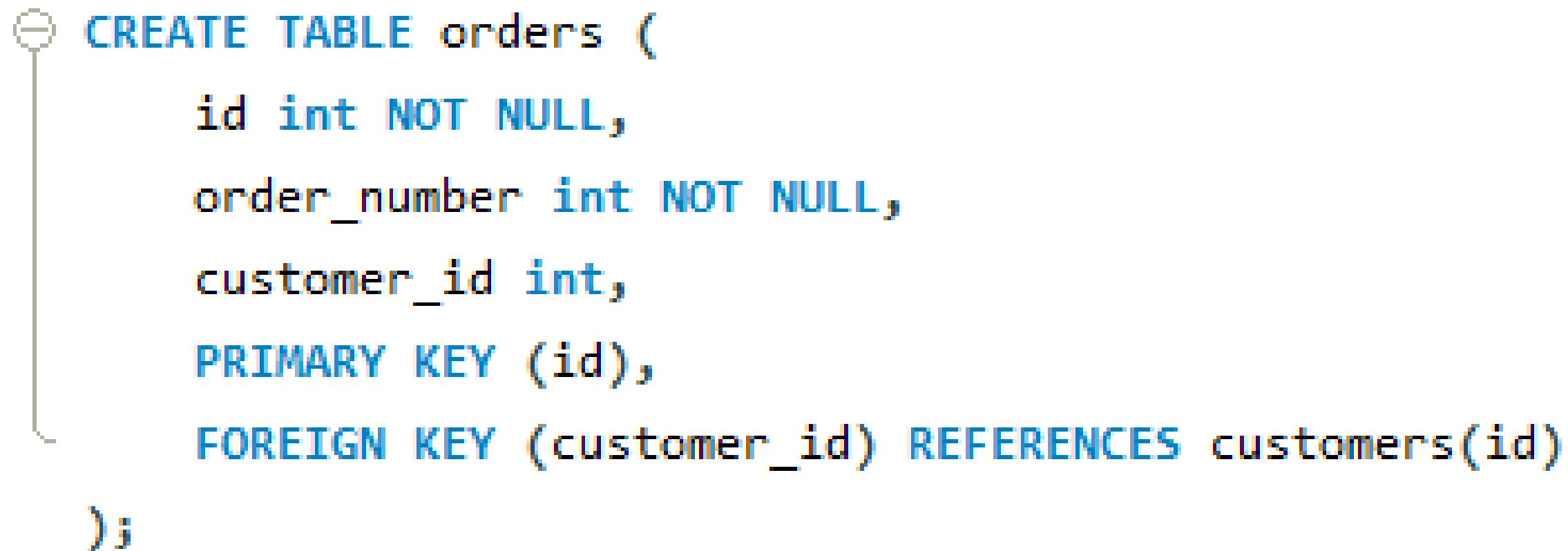
CREATE TABLE [Salary] (
    [EmployeeID]    INT UNIQUE NOT NULL
,   [SalaryAmount] INT
);

ALTER TABLE [Salary]
ADD CONSTRAINT FK_Salary_Employee FOREIGN KEY([EmployeeID])
REFERENCES [Employee]([ID]);
```



Один ко многим (one-to-many)

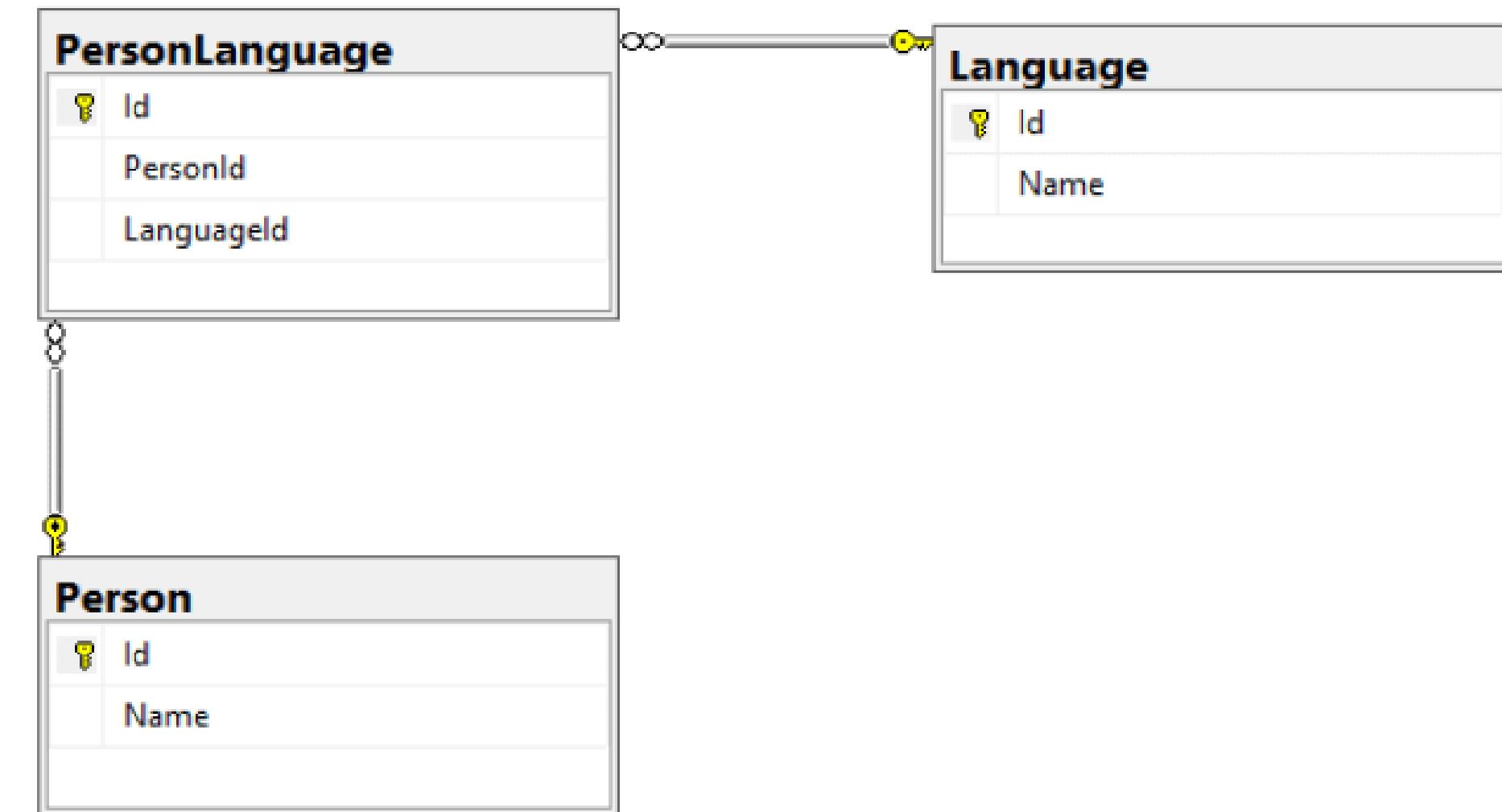
- такие отношения нужны для того чтобы иметь возможность хранить более 1 записи для каждого пользователя, например, покупатель может иметь много заказов
- самый распространенный вид связи



```
CREATE TABLE orders (
    id int NOT NULL,
    order_number int NOT NULL,
    customer_id int,
    PRIMARY KEY (id),
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

Многие ко многим (many-to-many)

- такие отношения нужны для того чтобы иметь возможность ссылаться из множества записей ко множеству, например, пользователи соц сети и подписчики, у каждого члена сети может быть много подписчиков, а также он сам может быть подписан на множество людей
- для этого создается отдельная таблица



Поведение при удалении/изменении

Мы можем задавать поведение при удалении или изменении связанной строки из главной таблицы

- **CASCADE** - удаляет или изменяет строки в зависимой таблице при удалении
- **SET NULL** - устанавливает значение внешнего ключа в NULL
- **RESTRICT** - запрещает удаление или изменение при наличии зависимостей
- **NO ACTION** - то же самое что и RESTRICT
- **SET DEFAULT** - задает дефолтное значение, которое задачи при помощи атрибута DEFAULT

```
1 CREATE TABLE Orders
2 (
3     Id INT PRIMARY KEY AUTO_INCREMENT,
4     CustomerId INT,
5     CreatedAt Date,
6     FOREIGN KEY (CustomerId) REFERENCES Customers (Id) ON DELETE SET NULL
7 );
```

2

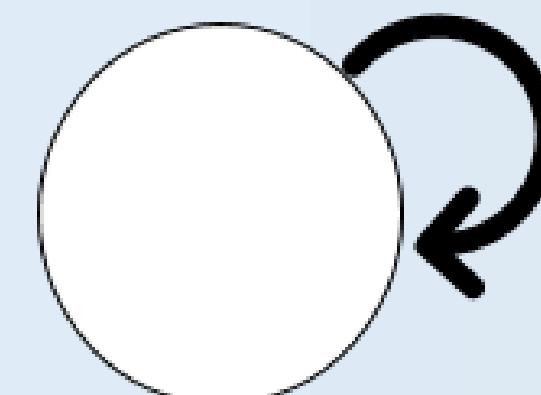
Join

JOIN

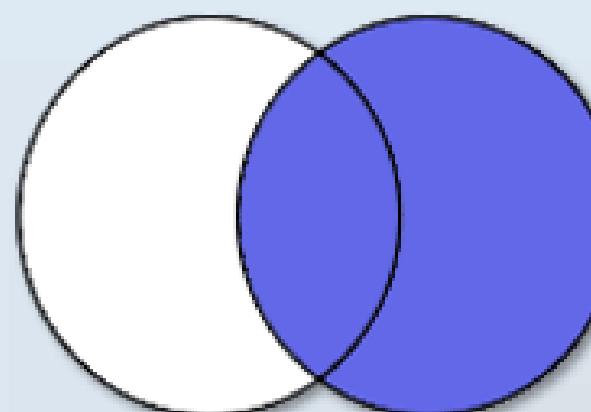
Способ объединить информацию из нескольких таблиц, например, информацию о пользователе (его имя и телефон и информацию о его заказах)

Joins in MySQL

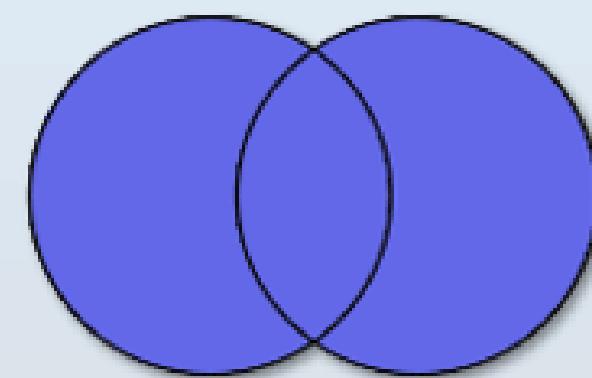
Self Join



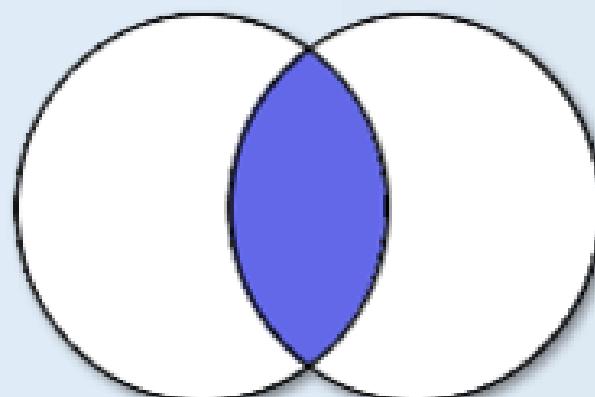
Right Join



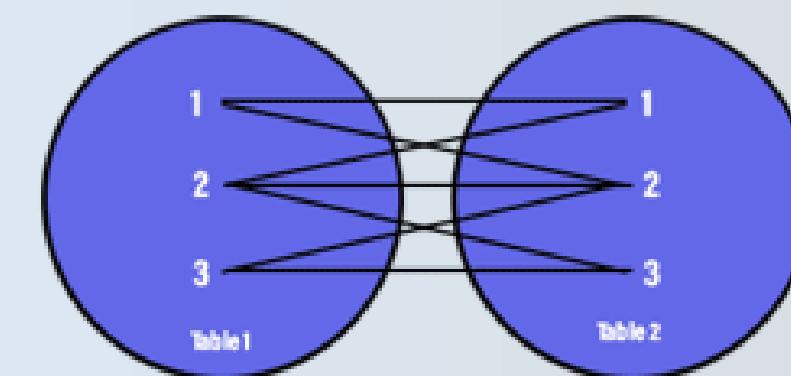
Full Join



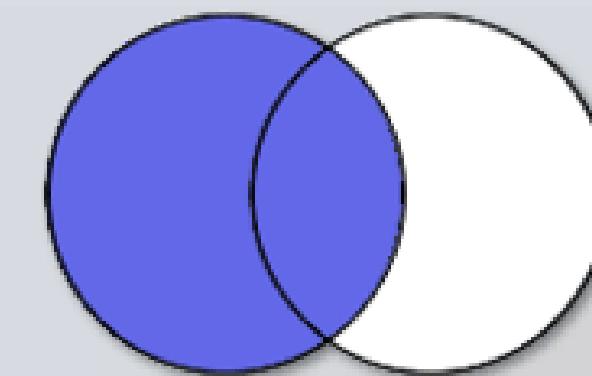
Inner Join



Cross Join

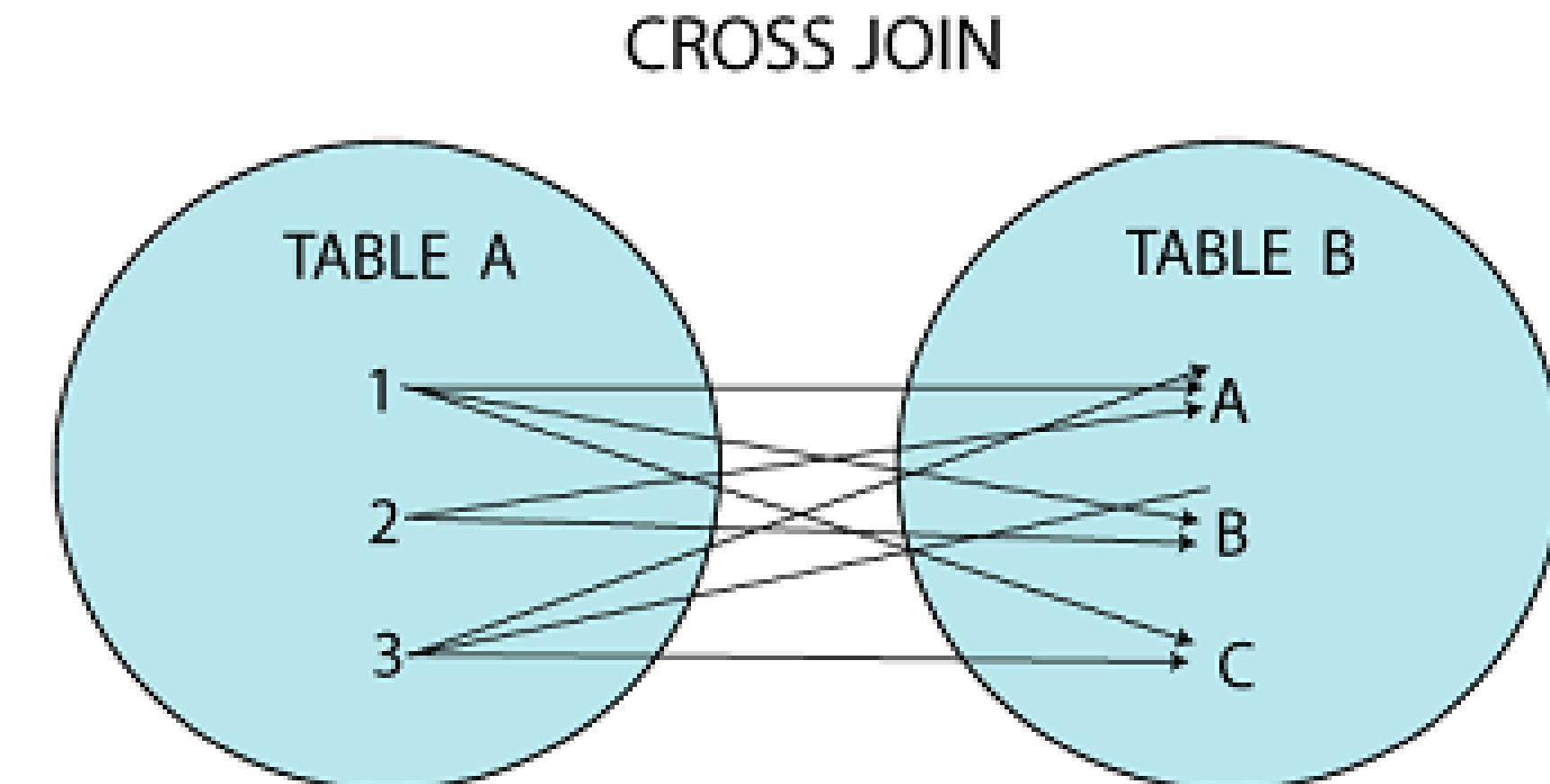


Left Join



JOIN

1. **INNER JOIN** - (по умолчанию) - формирует результат если и справа, и слева есть значения
2. **LEFT JOIN, RIGHT JOIN** - отличие в том что строки из левой (или правой) таблицы попадают в конечный результат в любом случае
3. **FULL JOIN** - сюда попадают все строки, вне зависимости от того, выполняется ли условие
4. **CROSS JOIN** - все со всеми
5. **SELF JOIN** - с собой



JOIN - примеры запросов

```
SELECT *
FROM table1
LEFT JOIN table2 ON table1.parameter=table2.parameter
WHERE table2.parameter IS NULL
```

```
SELECT *
FROM products JOIN products ON table.product=table.brand
```

```
SELECT *
FROM table1
FULL OUTER JOIN table2 ON table1.parameter=table2.parameter
```

3

Нормализация

Нормализация

Нормализация отношений в SQL призвана организовать информацию в базе данных таким образом, чтобы она не занимала много места и с ней было удобно работать. Это удаление избыточных данных, устранение дублей, идентификация наборов связанных данных через PRIMARY KEY, etc.

Соответственно, денормализация является обратным процессом, который вносит в нормализованную таблицу избыточные данные.

Нормализация – это процесс организации, структуризации данных в базе, который обеспечивает большую гибкость базы данных за счет исключения избыточности и несогласованности зависимостей.

Нормализация

Нормальная форма – свойство таблицы, рассматриваемое в контексте нормализации, которое характеризует таблицу с точки зрения простоты и правильности построения структуры. Нормальная форма определяется как совокупность требований, которым должна удовлетворять таблица.

Всего существует шесть нормальных форм, но на практике применяются не более первых трех

Нормальные формы

1. Первая нормальная форма.

- a. Все атрибуты простые (то есть атомарные и неделимые);
- b. Все данные скалярные;
- c. Нет повторяющихся строк (для этого для каждой строки создается первичный ключ).

Грубо говоря, в каждой ячейке таблицы только одно значение, не должно быть повторяющихся строк (имеется ввиду что с ячейке не может находиться список и тд)

Нормальные формы

1. Вторая нормальная форма.

- a. Соблюдены условия первой нормальной формы;
- b. Каждый неключевой атрибут ссылается на первичный ключ и не зависит от неключевого атрибута.

Есть первичный ключ, все атрибуты зависят от первичного ключа, а не от какой-то его части, то есть `worker_id`, `user_id`, тут нужна декомпозиция

Нормальные формы

1. Третья нормальная форма.

- a. Соблюдены условия второй нормальной группы;
- b. Неключевые поля не зависят от других неключевых полей: они могут быть связаны лишь с первичным ключом

От идентификаторы шины номер телефона заказчика никак не зависит

Нормальные формы

1. Бойса-Кодда, усиленная ЗНФ

а. Ключевые атрибуты не зависят от неключевых

2. **4НФ.** Устраняются многозначные зависимости. То есть 2 атрибута, которые зависят от pk, но друг с другом не связаны (сотрудники, его хобби и проект)

3. **5НФ.** Устраняются нетривиальные зависимости и производится декомпозиция без потерь. То есть чтобы при джойне декомпозированных таблиц мы получили начальную. Нужно хорошо шарить за предметную область

Преимущества нормализации

- 1.Лучшая организация базы данных
- 2.Больше таблиц с небольшими строками
- 3.Эффективный доступ к данным
- 4.Большая гибкость для запросов
- 5.Быстрый поиск информации
- 6.Проще реализовать безопасность данных
- 7.Позволяет легко модифицировать
- 8.Сокращение избыточных и дублирующихся данных
- 9.Более компактная база данных
- 10.Обеспечивает согласованность данных после внесения изменений

Денормализация

Денормализация – намеренное снижение или нарушение форм нормализации базы данных, обычно – чтобы ускорить чтение из базы за счет добавления избыточных данных. В общем, это процесс, обратный к нормализации.

Так происходит потому, что теория нормальных форм не всегда применима на практике.

К примеру, не атомарные значения – не всегда «зло»: иногда даже наоборот. В некоторых случаях необходимо дополнительное объединение при выполнении запросов, особенно при обработке большого массива информации. В итоге это может улучшить производительность. Для баз данных, предназначенных для аналитики, часто выполняют денормализацию, чтобы ускорить выполнение запросов.

4

Работа с БД

Работа с БД из python

Зачастую при написании кода взаимодействие производится при помощи **ORM** (**object relational mapping**):

```
Record.objects.filter(is_deleted=False).\\  
| values_list('name')  
  
9  
10  SELECT name  
11  FROM records  
12 WHERE NOT is_deleted;
```

Миграции

Миграции это sql скрипты, которые помогают нам обновить схему БД и обладают следующими свойствами:

- атомарность
- обратимость
- упорядоченность

Подзапросы

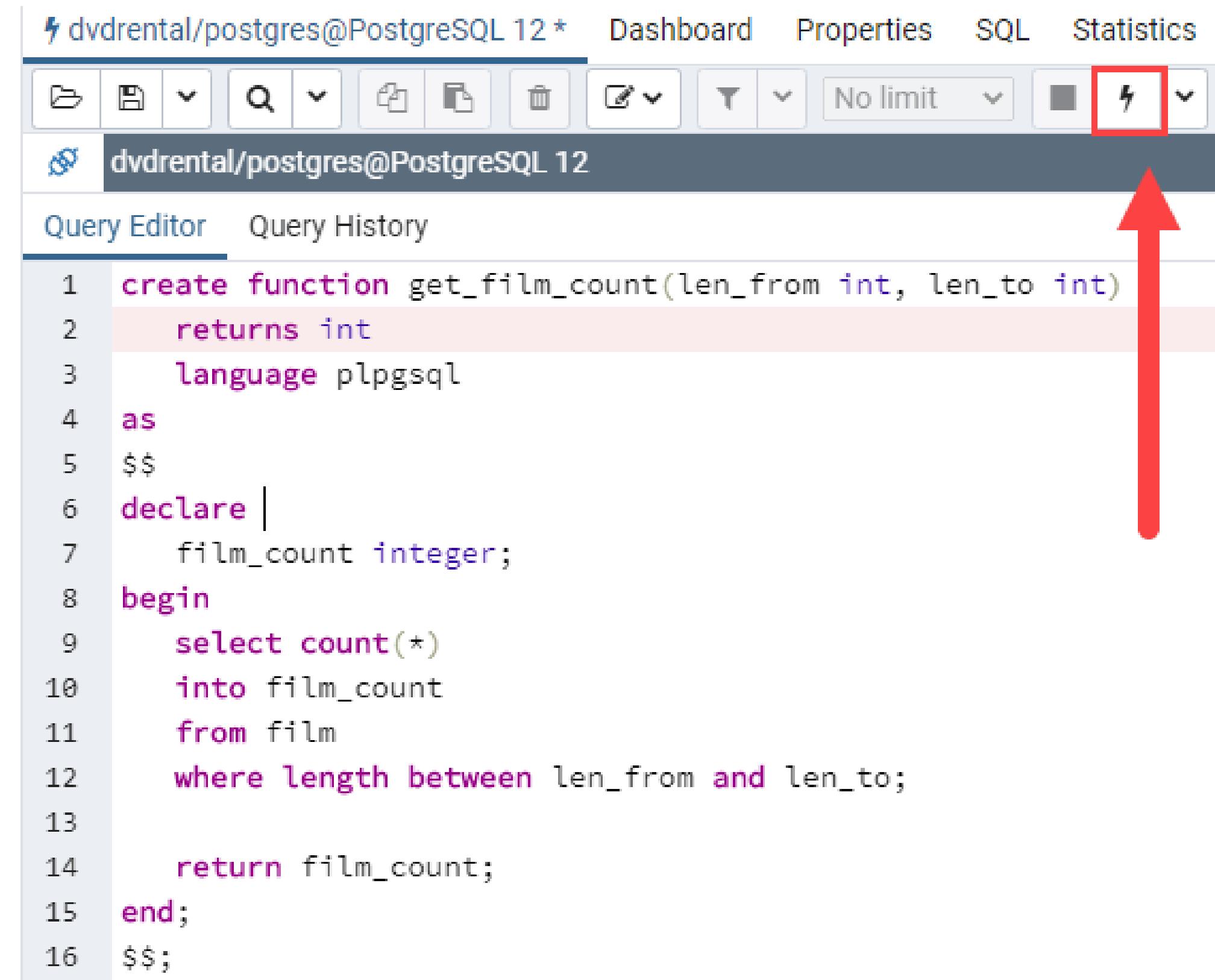
- получить все товары, цена которых ниже средней

```
1 | SELECT * FROM Products
2 | WHERE Price < (SELECT AVG(Price) FROM Products)
```

- получить все товары, цена которых ниже цены товара фирмы Apple

```
1 | SELECT * FROM Products
2 | WHERE Price < ALL(SELECT Price FROM Products WHERE Manufacturer='Apple')
```

ФУНКЦИИ



The screenshot shows the pgAdmin 4 interface for PostgreSQL 12. The title bar indicates the connection is to the 'dvrental/postgres@PostgreSQL 12' database. The toolbar at the top includes various icons for file operations (New, Open, Save, Print, Find, Copy, Paste, Delete, Undo, Redo) and a dropdown menu for 'No limit'. A red box highlights the execute icon (a lightning bolt symbol) in the toolbar. Below the toolbar, the status bar shows the same connection information. The main area is a 'Query Editor' tab where a SQL function is being defined:

```
1 create function get_film_count(len_from int, len_to int)
2     returns int
3     language plpgsql
4 as
5 $$
6 declare |
7     film_count integer;
8 begin
9     select count(*)
10    into film_count
11   from film
12  where length between len_from and len_to;
13
14  return film_count;
15 end;
16 $$;
```

Практика

5

Практика

- Спроектировать базу данных для соц сети, в ней должны быть:
 - пользователи (информация для логина)
 - информация о пользователях (личные данные)
 - подписки
 - сообщения
- Наполнить базу тестовыми данными
- Выполнить следующие запросы:
 - вывести наиболее популярного по количеству подписчиков человека
 - вывести человека, который ни на кого не подписан
 - показать информацию пользователя по id из таблицы профиля
 - вывести непрочитанные сообщения для пользователя по id, "прочитать" эти сообщения
 - получить историю переписки между двумя пользователями
 - вывести наиболее активного пользователя (с большим количеством сообщений)
 - узнать среднее количество сообщений по всем пользователям

Спасибо за внимание!