
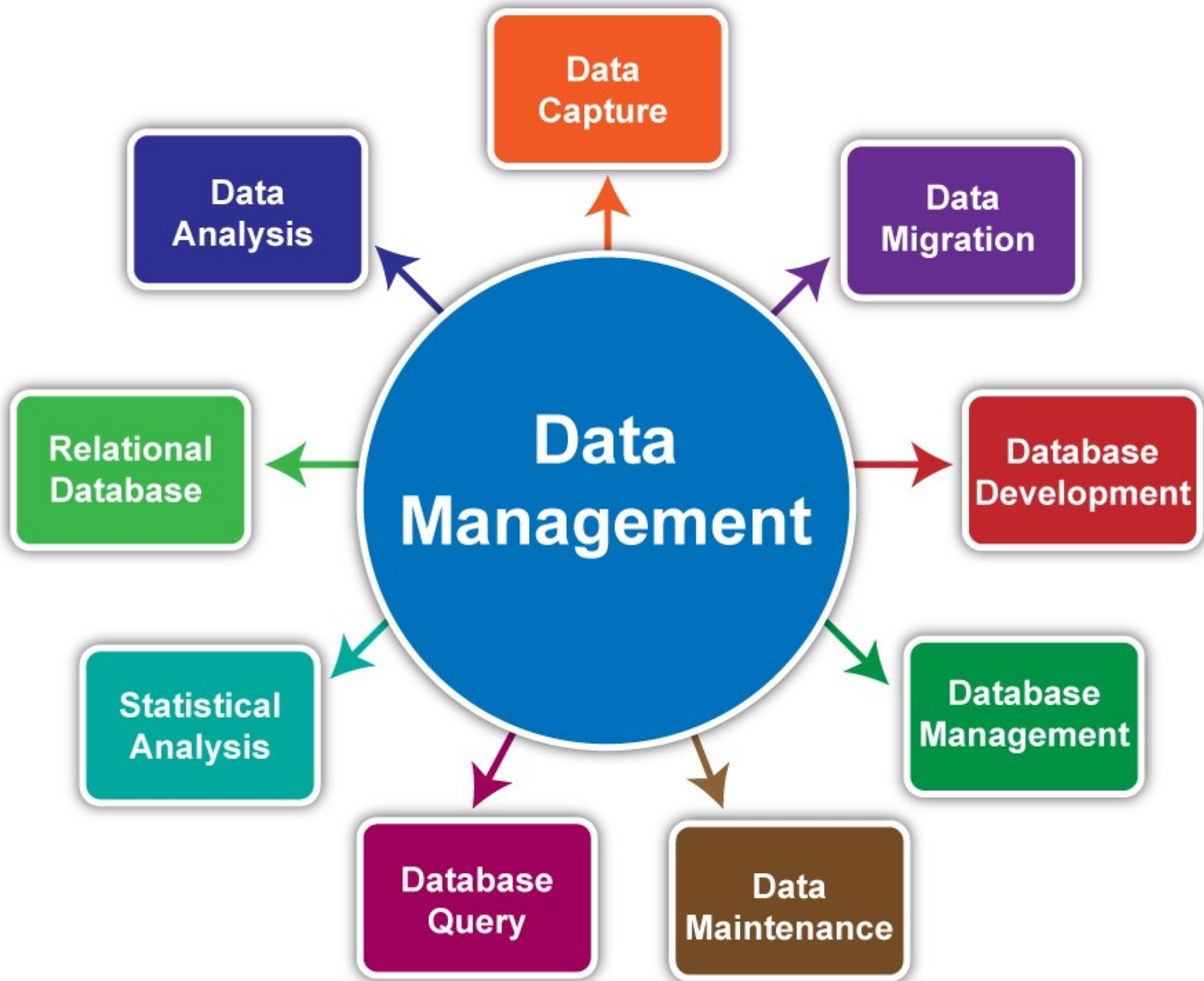


An abstract graphic in the top-left corner consisting of several overlapping squares and circles in various shades of blue and white, creating a layered, geometric effect.

# DB Design



**Данные** - это информация, представленная в определенном виде, позволяющем автоматизировать ее сбор, хранение и дальнейшую обработку человеком или информационным средством.



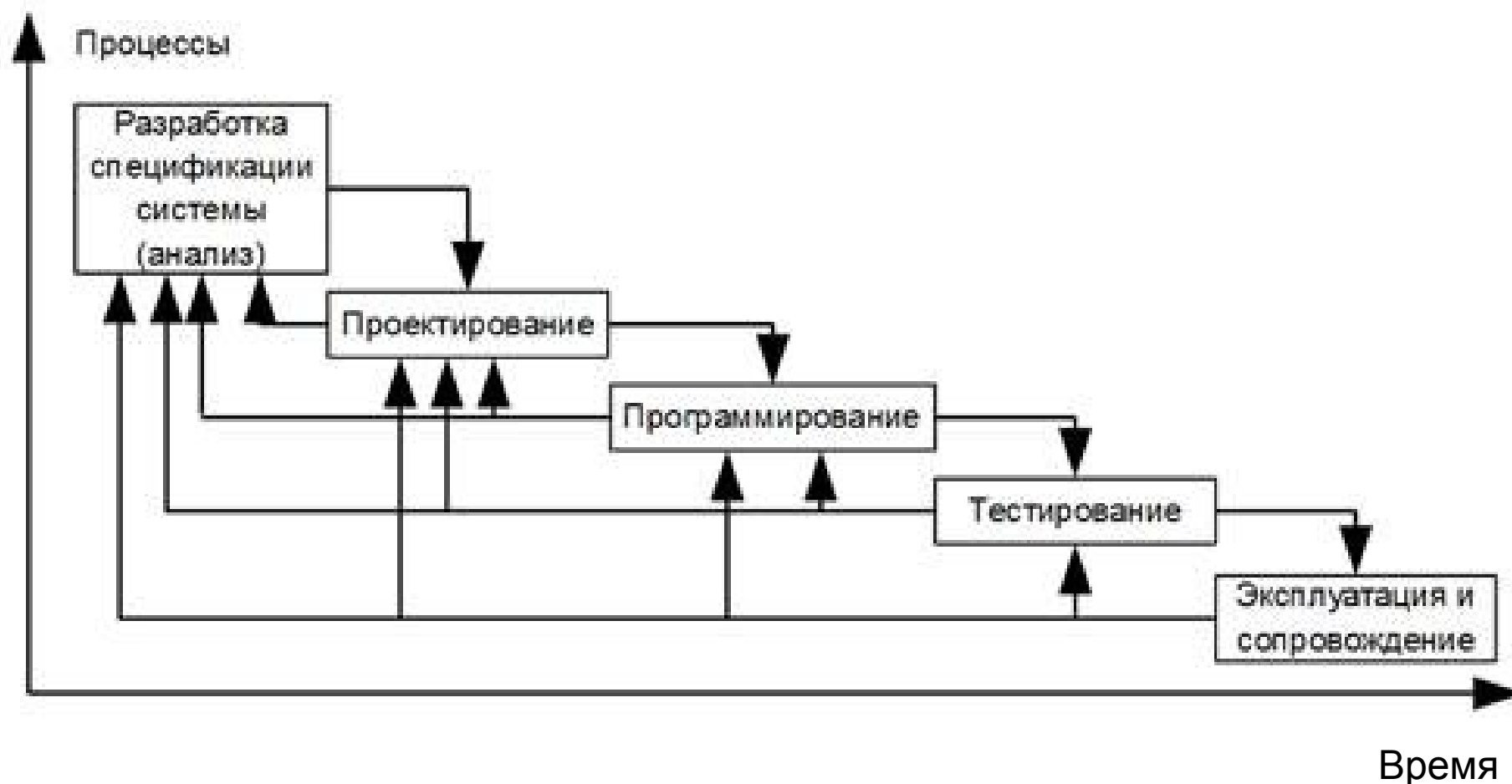


# Автоматизированная информационная система (АИС)

АИС - это совокупность программных и аппаратных средств, предназначенных для хранения и/или управления данными и информацией, а также для производства вычислений.

АИС представляют совокупность функциональных подсистем сбора, ввода, обработки, хранения, поиска и распространения информации. Процессы сбора и ввода данных необязательны, поскольку вся необходимая и достаточная для функционирования АИС информация, может уже находиться в составе ее базы данных. Каждая АИС имеет дело с той или иной частью реального мира, которую принято называть предметной областью

# Жизненный цикл разработки ПО (АИС,ИС ...)





## Предметная область

Это сфера экономической деятельности, например, транспортная логистика, банковский сектор, биллинговые системы, медицина, электронная коммерция, игровое приложение.



## База данных (БД)

Именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области, которая допускает использование данных оптимальным образом для одного или нескольких приложений.

В БД информация должна быть организована таким образом, чтобы обеспечить минимальную долю ее избыточности. Частичная избыточность информации необходима, но она должна быть минимизирована, т.к. чрезмерная избыточность данных влечет за собой ряд негативных последствий, главные из которых:

- увеличение объема информации
- появление ошибок при вводе дублирующей информации, нарушающих целостность базы данных и создающих противоречивые данные.



## Для организации БД используют СУБД

**Реляционные** - совокупность таблиц и связей между ними.  
Примеры: PostgreSQL, SQLite

**Документно-ориентированные** - хранит иерархические структуры данных (документы), как правило реализована с помощью подхода NoSQL. Примеры: MongoDB, CouchDB

**Объектно-ориентированные** - хранят информацию в виде объектов, как в объектно-ориентированных языках программирования. Примеры: db4o

**Графовые** - предназначены для хранения взаимосвязей и навигации в них, данные хранятся в виде структуры данных граф, где узлы хранят сущности, а ребра связи. Примеры: OrientDB, Amazon Neptune, Neo4j





# Система управления базами данных (СУБД)

Совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями. Возможности современных СУБД:

- язык определения данных(DDL), с помощью которого можно создать базу данных, описать ее структуру, типы данных и средства для задания ограничений целостности данных;
- язык управления данными(DML), которые позволяет вставлять, обновлять, удалять и извлекать информацию из БД;



## Возможности СУБД

- кроссплатформенность - независимость от используемой архитектуры или ОС;
- подсистема разграничения доступа к данным;
- подсистема поддержки целостности БД обеспечивающая непротиворечивое состояние хранимых данных;
- Подсистема управления параллельной работой приложений контролирующая процессы их совместного доступа к БД;
- подсистема восстановления, позволяющая вернуть БД к предыдущему состоянию, нарушенному из-за аппаратного или программного сбоя;



# Реляционные базы данных как один из типов БД.

**Реляционная база данных** – это совокупность отношений, содержащих всю информацию, которая должна храниться в БД. Однако пользователи могут воспринимать такую базу данных как совокупность таблиц.

**Моделирование данных** - это средство формального сбора данных, относящихся к бизнес-процессу данной организации. Моделирование является приемом анализа, на базе которого строятся реляционные БД.



# DB Design

Процесс моделирования данных включает три уровня моделей: от концептуальной к логической, а затем к физической.

**Концептуальная модель** - модель предметной области, состоящая из перечня взаимосвязанных объектов, используемых для описания этой области, вместе со свойствами и характеристиками. Концептуальная модель, как правило, представляет сущность бизнеса и ничего больше.

**Логическая модель** - представляется диаграммой «сущность-связь» или ER (Entity-Relationship) диаграммой.

**Физическая модель** - это схема базы данных для конкретной СУБД, где сущности предметной области превращаются в таблицы, атрибуты в ее колонки (часто называют полями) с использованием конкретного типа данных, связи в ограничения целостности.

# Database Design Steps



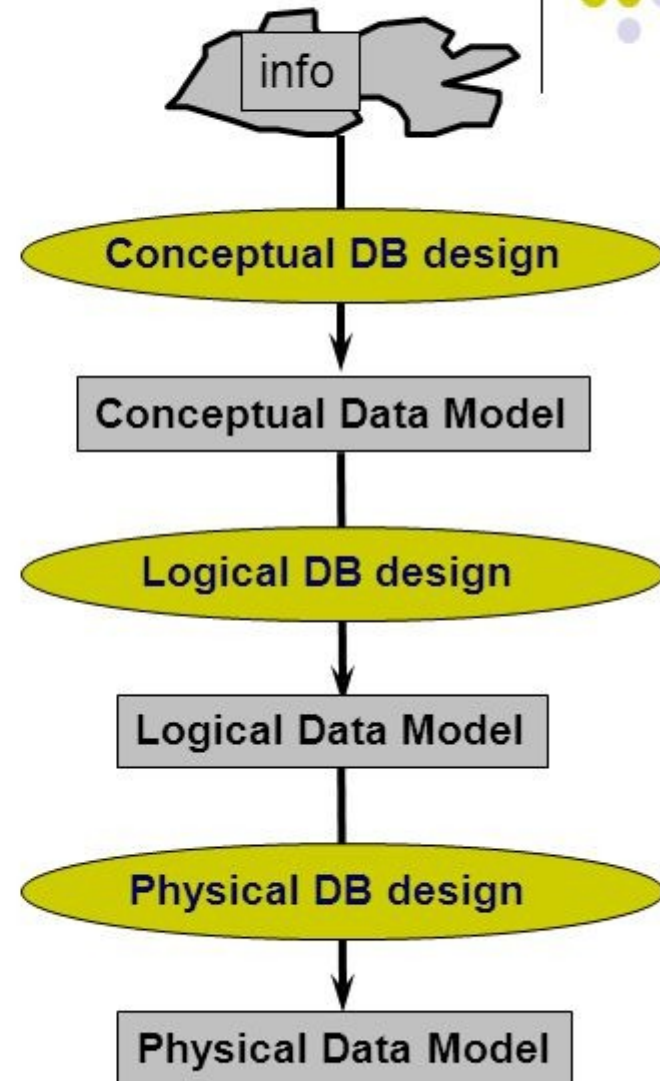
## Entity-relationship Model

Typically used for conceptual database design

## Three Levels of Modeling

## Relational Model

Typically used for logical database design





# Концептуальная модель данных

Концептуальная модель данных (КМД) – это общая информационная модель предметной области, охватывающая вопросы классификации, структуризации и семантической целостности (достоверности и согласованности данных).

Концептуальная модель данных разрабатывается независимо от ограничений, вытекающих из моделей данных, поддерживаемая той или иной СУБД.

Для описания концептуальной модели может быть использован естественный язык, но такое описание будет громоздким и неоднозначным, поэтому для описания КМ чаще всего используется формализованный язык.




# Логическое моделирование

На этапе логического моделирования выявляются сущности, их атрибуты и связи между ними.

**Сущность** - это объект реального мира, который может быть как материальным, так и нет. Пример материальных объектов: покупатель, продукт, автомобиль. Пример нематериальных объектов: заказ, формула, расписание.

С логической точки зрения сущность представляет собой совокупность однотипных объектов называемых экземплярами этой сущности. Экземпляры сущности должны быть уникальными, то есть полный набор значений их атрибутов не должен дублироваться.



**Атрибуты сущности** - это фактически свойства объекта. Например, у покупателя есть фамилия, имя и отчество - это его свойства или атрибуты.

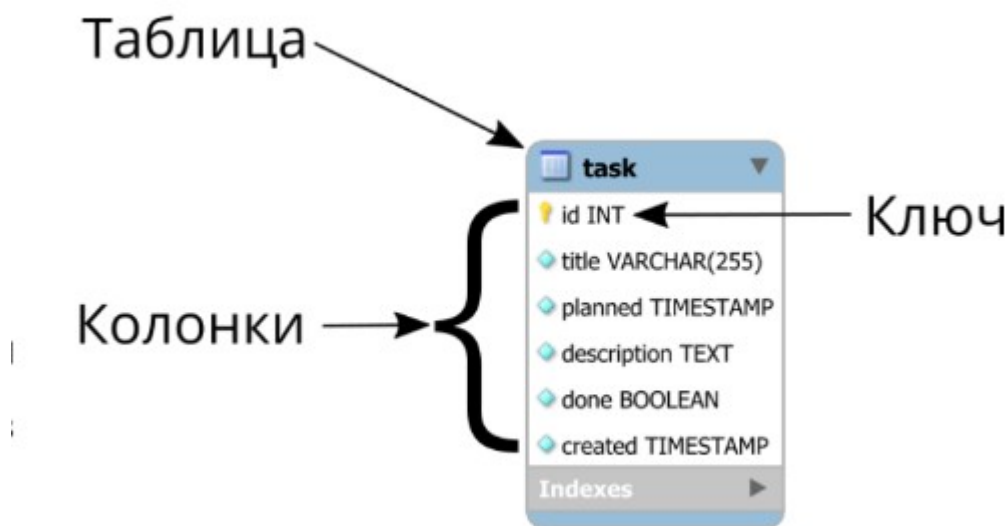
**Атрибуты** могут быть ключевыми и не ключевыми. Например, у покупателя может быть **уникальный идентификатор**, значение которого можно использовать в атрибуте “покупатель” сущности заказ.

На этапе логического проектирования для каждого атрибута обычно определяется примерный тип данных (строковый, числовой, логический, двоичные данные и др.).



# Физическая модель

Физическая модель - это схема базы данных для конкретной СУБД, где сущности предметной области превращаются в таблицы, атрибуты в ее колонки (часто называют столбцами) с использованием конкретного типа данных, связи в ограничения целостности. Строка таблицы - экземпляр одной сущности предметной области.





## Физическая модель

Специфика конкретной СУБД при физическом проектировании включает выбор решений, связанных с физической средой хранения данных. Выбор методов управления дисковой памятью, разделение БД по файлам и устройствам (шардирование), методов доступа к данным, создание индексов, и т.д.



# Зачем нужен первичный ключ?

**Первичным ключом (Primary key)** - называется атрибут или набор атрибутов, который уникальным образом идентифицирует сущность. Если первичный ключ состоит более чем из одного атрибута, то его называют составным первичным ключом.

**Каждая сущность должна иметь первичный ключ**, в противном случае вы никогда не сможете однозначно ее идентифицировать. Требуются очень веские причины, почему ваша сущность не имеет первичного ключа, такое встречается, но крайне редко.

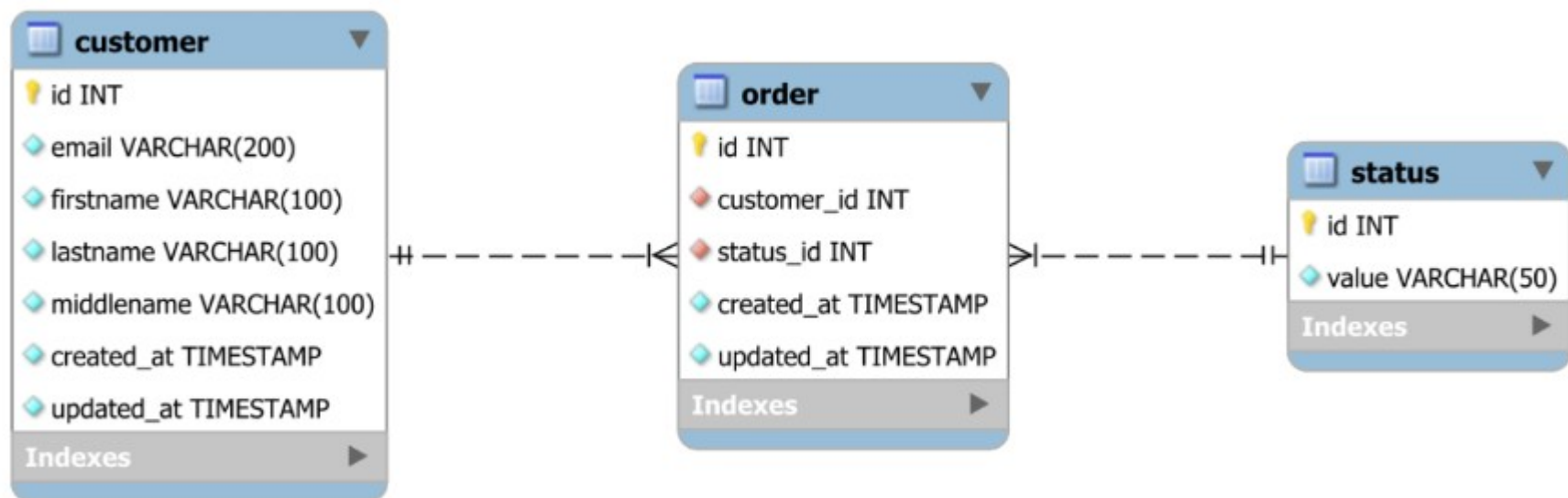


## Зачем нужен внешний ключ?

**Внешние ключи** (Foreign key) - используются для организации связей между таблицами базы данных (родительскими и дочерними) и для поддержания ограничений ссылочной целостности данных.

# Отношение один-ко-многим

Один экземпляр одной сущности может быть связан с множеством экземпляров другой сущности. Например, покупатель может совершить множество заказов в интернет магазине, но один конкретный заказ может принадлежать только одному покупателю. В таблицу order требуется добавить внешний ключ customer\_id. Значением внешнего ключа – будет значение первичного ключа из таблицы customer:





## Примеры

- Один статус заказа можно использовать во множестве заказов, но в один момент времени заказ может иметь только один статус;
- У товара может быть множество отзывов, но конкретный отзыв может быть оставлен только для одного товара;
- В одном отделе может работать множество сотрудников, но конкретный сотрудник работает только в одном отделе;



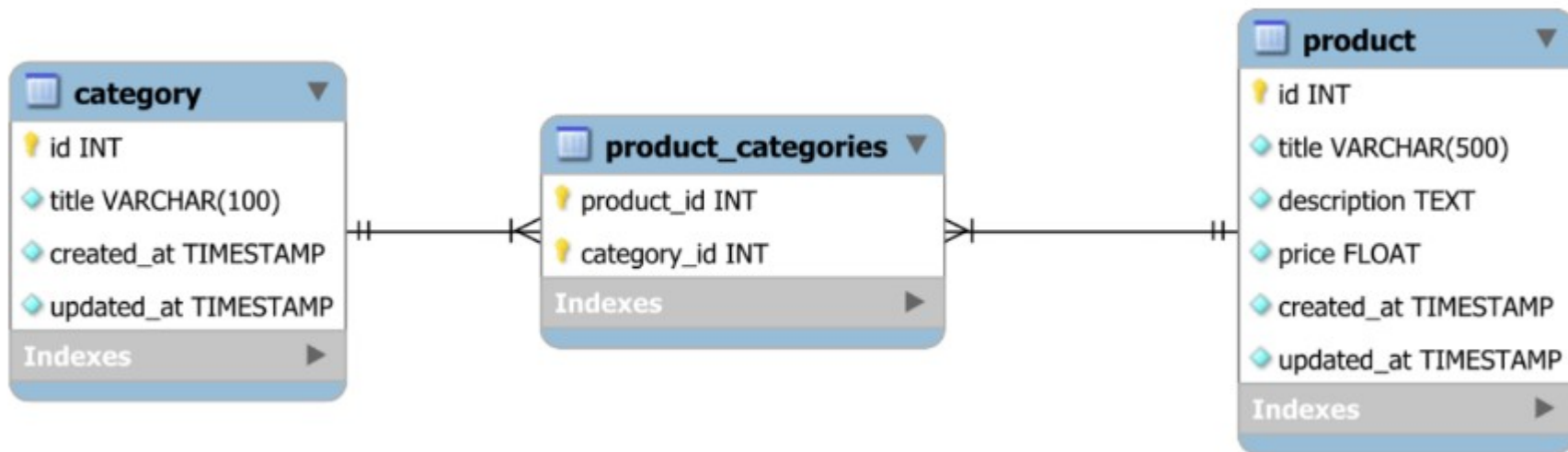
## МНОГИЕ-КО-МНОГИМ

Множество экземпляров одной сущности может быть связано с множеством экземпляров другой сущности. Например, в одну категорию можно добавить множество товаров, однако для удобства пользователя товару можно назначить множество категорий.

В реляционной БД связь многие-ко-многим можно разрешить только через вспомогательную. Таким образом связь многие-ко-многим превращается в две связи один-ко-многим.

## МНОГИЕ-КО-МНОГИМ

В таблицу ассоциации `product_categories` требуется добавить два внешних ключа `product_id` и `category_id`. Значением внешних ключей будут значения первичных ключей из таблиц `product` и `category` соответственно. Чтобы избежать случайного добавления товара в одну категорию дважды, либо присвоения категории одному товару дважды, внешние ключи вместе образуют первичный ключ:





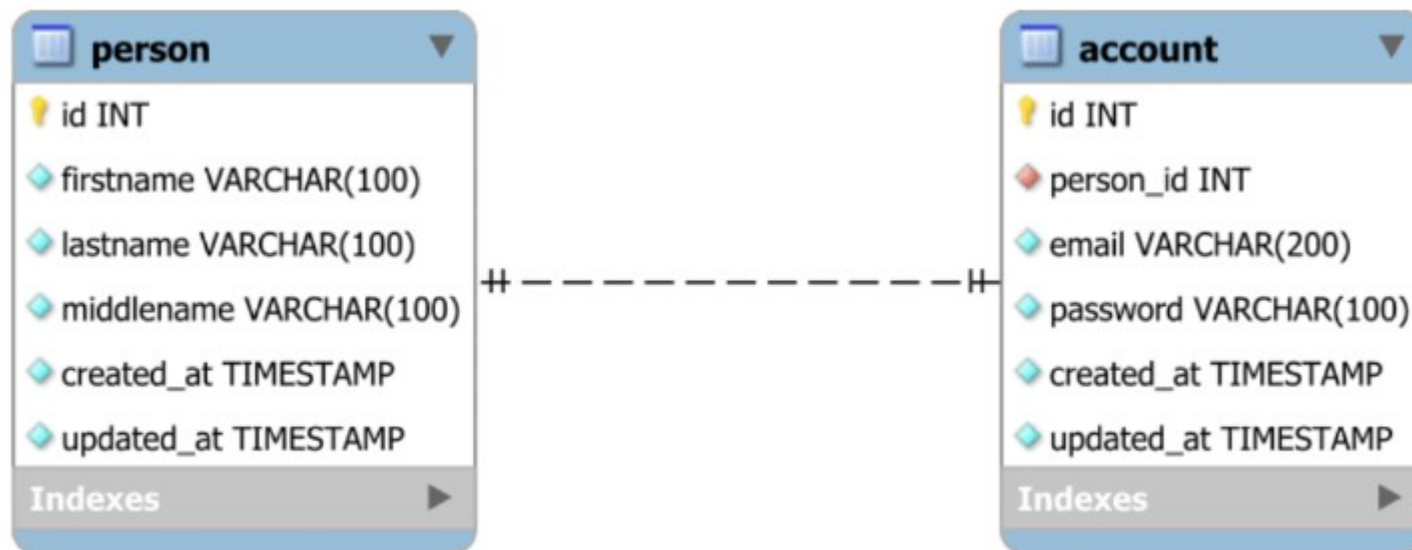


# Примеры многие-ко-многим

- Автор может написать множество книг, но одну книгу может написать множество авторов;
- На курсах дополнительного образования, студент может учиться на множестве курсов, но на одном курсе может учиться множество студентов;
- Посту в инстаграмме можно добавить множество хеш-тегов, но один хеш-тег можно использовать для множества постов;

# Отношения один-к-одному

Один экземпляр одной сущности может быть связан не более чем с одним экземпляром другой сущности. Используется, если необходимо отделить некоторый набор сведений, однозначно связанный с конкретным экземпляром. Связь один-к-одному в БД описывается также, как связь один-ко-многим. В таблицу `account` требуется добавить внешний ключ `person_id`. Значением внешнего ключа - будет значение первичного ключа из таблицы `person`. Если вы хотите избежать случайного добавления второго аккаунта, то значение внешнего ключа `person_id` можно сделать уникальным индексом:





# SQL - Structured Query Language

SQL - язык структурированных запросов, обычно используется в реляционных базах данных для выполнения запросов и состоит из двух подмножеств:

- DDL (Data Definition Language) - язык определения данных;
- DML (Data Manipulation Language) – язык манипулирования данными. Условно, можно сказать, что язык стандартизированный (одинаковый для любой СУБД), однако стандартизированный заканчивается в тот момент, когда в запросе используются специфичные для выбранной СУБД типы данных, функции или иной синтаксис.



# Python Database API

PEP-249 описывает программный интерфейс взаимодействия с БД. Таким образом любой модуль, который работает с БД, должен реализовывать (поддерживать) этот интерфейс. Пример наиболее часто используемых СУБД и модулей, поддерживающих DB API:

- MySQL (MariaDB) - PyMySQL, MySQLdb
- PostgreSQL - psycopg2
- SQLite3 - встроенный модуль sqlite3

Благодаря этому, на примере модуля psycopg2, вы сможете изучить и опробовать все методы. Если в будущем вам понадобится работать с другой СУБД, то достаточно установить сторонний модуль, все методы вам уже знакомы.

Единственное отличие при использовании этих модулей, это аргументы функции connect() - фактически это аргументы конструктора, а сама функция возвращает экземпляр объекта соединения.

# Алгоритм взаимодействия с БД

1. Установка соединения с сервером БД функцией `connect()`
2. Выполнение запроса:
  - 2.1. Получить объект курсора методом `cursor()`
  - 2.2. Выполнить запрос методом `execute()`
  - 2.3. Если запрос на изменение данных или структуры БД:
    - 2.3.1. Нужно зафиксировать изменения методом `commit()`
  - 2.4. Если запрос на получение данных (SELECT):
    - 2.4.1. Фактические данные нужно раз-fetch-ить:
      - `fetchall()` - получить все строки таблицы в список
      - `fetchone()` - получить одну строку из таблицы
      - `fetchmany(N)` - получить нужное кол-во строк (N) из таблицы
3. Закрыть соединение с сервером БД методом `close()`

```
# Note: the module name is psycopg, not psycopg3
import psycopg

# Connect to an existing database
with psycopg.connect("dbname=test user=postgres") as conn:

    # Open a cursor to perform database operations
    with conn.cursor() as cur:

        # Execute a command: this creates a new table
        cur.execute("""
            CREATE TABLE test (
                id serial PRIMARY KEY,
                num integer,
                data text)
            """)

        # Pass data to fill a query placeholders and let Psycopg perform
        # the correct conversion (no SQL injections!)
        cur.execute(
            "INSERT INTO test (num, data) VALUES (%s, %s)",
            (100, "abc'def"))

        # Query the database and obtain data as Python objects.
        cur.execute("SELECT * FROM test")
        cur.fetchone()
        # will return (1, 100, "abc'def")

        # You can use `cur.fetchmany()`, `cur.fetchall()` to return a list
        # of several records, or even iterate on the cursor
        for record in cur:
            print(record)

        # Make the changes to the database persistent
        conn.commit()
```



# Документация

<https://www.psycopg.org/psycopg3/docs/basic/usage.html>

ER – моделирование

<https://editor.ponyorm.com/>

An abstract graphic in the top-left corner consisting of several overlapping squares and circles in various shades of blue and white, creating a layered, geometric effect.

# Физическая модель





# Метаморфоза

Переход из одной формы в другую с приобретением нового внешнего вида и функций.

Переход из логической структуры к физической. Сущность становится таблицей, атрибуты — столбцами, уникальный идентификаторы — ключами, связи — ограничением целостности (констрейтами). Для того чтобы переход состоялся нужно осуществить описание объектов на языке SQL и выполнить скрипт создания объектов в СУБД.

Stop following me, you fucking freaks!



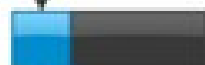
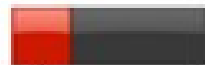
Key-Value



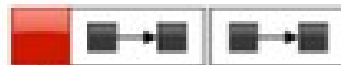
Key Value



Ordered Key-Value



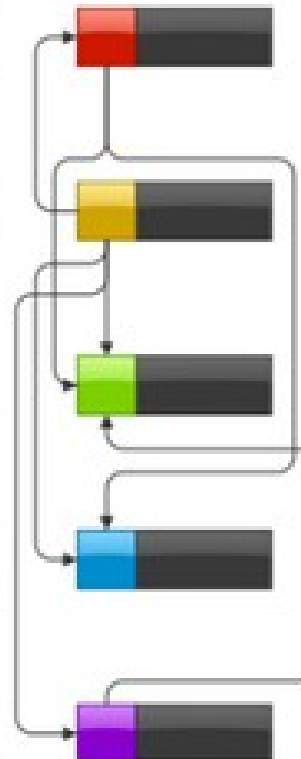
Big Table



Document,  
Full-Text Search



Graph



SQL

```
graph TD; SQL[SQL] --> DDL[DDL]; SQL --> DML[DML];
```

**SQL**

**DDL**

язык определения данных  
(Data Definition Language)

**DML**

язык манипулирования данными  
(Data Manipulation Language)

# DDL- язык для создания объектов базы данных

Комманды:

CREATE DATABASE	— создать базу данных
CREATE USER	— создать пользователя
CREATE TABLE	— создать таблицу
ALTER TABLE	— модифицировать таблицу
RENAME TO	— переименовать таблицу
CHANGE COLUMN	— изменить имя и тип данных столбца
MODIFY COLUMN	— изменить тип данных или позицию столбца.
ADD COLUMN	— добавить столбец в таблицу
DROP COLUMN	— удалить столбец из таблицы
DROP TABLE	— удалить таблицу

# DML - язык манипулирования данными

Комманды:

SELECT	—	извлечение данных
UPDATE	—	модификация данных
DELETE	—	удаление данных
INSERT INTO	—	вставка новых данных в таблицу

# Объекты базы данных

- Таблицы
- Ключи
- Индексы
- Констрейнты (связи)
- Представления
- Процедуры
- Триггера
- Функции

# Создание пользователя

```
CREATE USER test  
WITH PASSWORD 'jw8s0F4';
```

<https://www.postgresql.org/docs/current/sql-createuser.html>

# Создание DB

## Синтаксис

```
CREATE DATABASE name
  [ [ WITH ] [ OWNER [=] user_name ]
    [ TEMPLATE [=] template ]
    [ ENCODING [=] encoding ]
    [ LOCALE [=] locale ]
    [ LC_COLLATE [=] lc_collate ]
    [ LC_CTYPE [=] lc_ctype ]
    [ TABLESPACE [=] tablespace_name ]
    [ ALLOW_CONNECTIONS [=] allowconn ]
    [ CONNECTION LIMIT [=] connlimit ]
    [ IS_TEMPLATE [=] istemplate ] ]
```

<https://www.postgresql.org/docs/current/sql-createdatabase.html>



CREATE DATABASE

```
CREATE DATABASE music  
WITH OWNER 'test'  
LOCALE 'ru_RU.utf8'  
TEMPLATE template0;
```

# Типы данных

SERIAL — автоинкремент  
INTEGER — целое число  
NUMERIC — число с плавающей точкой  
VARCHAR() — текстовые данные длиной до 255  
TEXT — набор с максимальной длиной 65535  
DATE — дата.  
TIMESTAMP — дата и время.  
BOOLEAN — логический тип  
JSON — текстовый json  
BLOB — массив двоичных данных.

# Создание таблицы

## Синтаксис

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name
    [ ( column_name [, ...] ) ]
    [ USING method ]
    [ WITH ( storage_parameter [= value] [, ...] ) | WITHOUT OIDS ]
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ TABLESPACE tablespace_name ]
AS query
[ WITH [ NO ] DATA ]
```

## Примеры

<https://www.postgresql.org/docs/current/sql-createtableas.html>

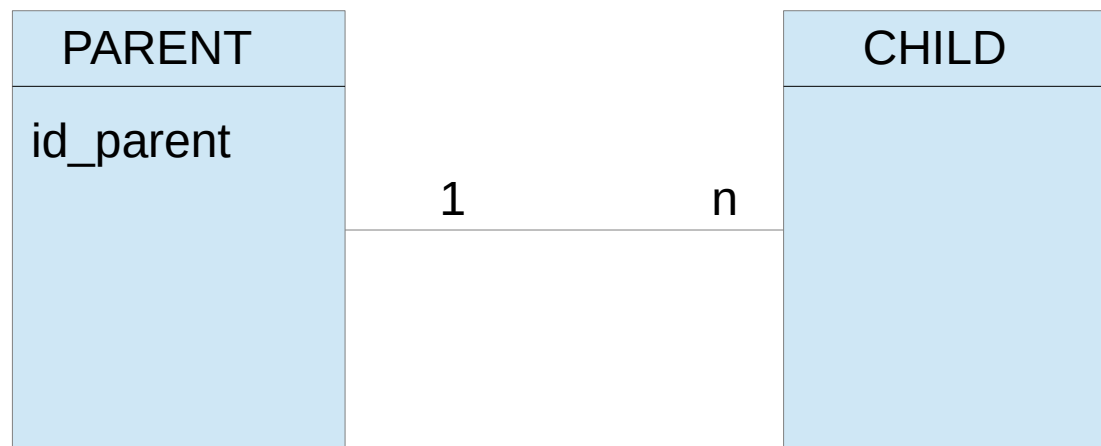
# Как создать таблицу?

```
CREATE TABLE "project" (  
  "id_project" SERIAL PRIMARY KEY,  
  "title" VARCHAR(250) NOT NULL,  
  "dt_start" DATE,  
  "dt_end" DATE,  
  "desc_full" TEXT NOT NULL,  
  "desc_short" TEXT NOT NULL,  
  "status" INTEGER,  
  "href_avatar" VARCHAR(100) NOT NULL,  
  "tag" JSONB NOT NULL,  
  "id_parent_project" INTEGER NOT NULL,  
  "name_rev" TEXT NOT NULL  
);
```

# Как добавить связь между таблицами

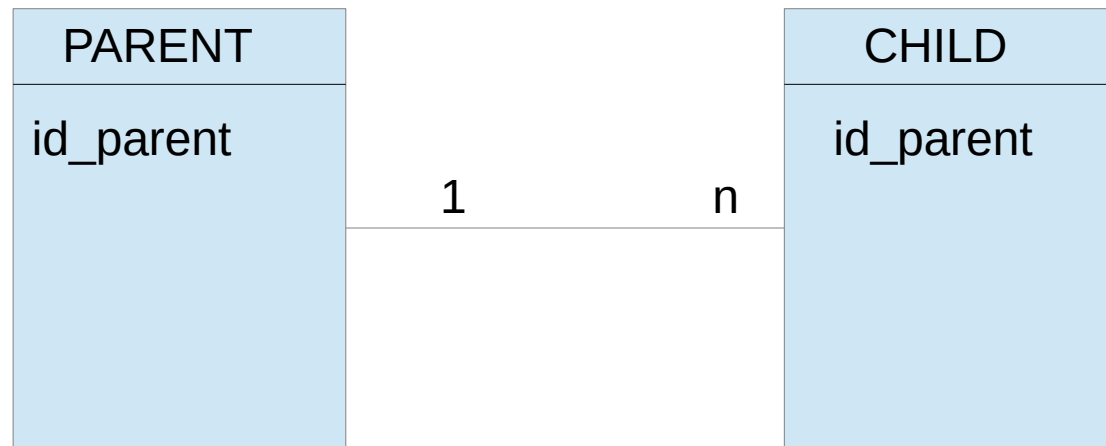
```
ALTER TABLE child_name  
ADD CONSTRAINT name_constraint  
FOREIGN KEY(id_parent)  
REFERENCES parent_name(id_parent)  
ON DELETE RESTRICT
```

# Пример



```
ALTER TABLE child ADD CONSTRAINT  
cnst_child_ref_parent  
FOREIGN KEY (id_parent)  
REFERENCES parent(id_parent)  
ON DELETE RESTRICT
```

# Результат



# Как удалить таблицу ?

```
DROP TABLE table_name;
```



# Вставка записей

## Синтаксис

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]  
    [ OVERRIDING { SYSTEM | USER } VALUE ]  
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }  
    [ ON CONFLICT [ conflict_target ] conflict_action ]  
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

where *conflict\_target* can be one of:

```
( { index_column_name | ( index_expression ) } [ COLLATE collation ] [ opclass ] [, ...] ) [ WHERE index_predicate ]  
ON CONSTRAINT constraint_name
```

and *conflict\_action* is one of:

```
DO NOTHING  
DO UPDATE SET { column_name = { expression | DEFAULT } |  
                ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |  
                ( column_name [, ...] ) = ( sub-SELECT )  
                } [, ...]  
[ WHERE condition ]
```

# INSERT — вставка записей

- 1) `INSERT INTO table_name (id_table_name, login, passwd) VALUES (NULL, 'admin', '123456');`
- 2) `INSERT INTO table_name (id_table_name, login, passwd) VALUES (1, 'admin', '123456');`
- 3) `INSERT INTO table_name  
VALUES (2, 'admin', '123456');`
- 4) `INSERT INTO table_name (login, passwd)  
VALUES ('admin', '123456');`
- 5) `INSERT INTO table_name (id_table_name, login, password) VALUES (99, 'admin', '123456');`

table\_name

id_table_name	login	passwd
1	admin	123456
2	admin	123456
3	admin	123456
99	admin	123456

# удаление записей

## Синтаксис

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
DELETE FROM [ ONLY ] table_name [ * ] [ [ AS ] alias ]  
    [ USING from_item [, ...] ]  
    [ WHERE condition | WHERE CURRENT OF cursor_name ]  
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

<https://www.postgresql.org/docs/current/sql-delete.html>

# DELETE - удаление записей

```
DELETE FROM table_name  
           WHERE id_table_name = 99
```

# Изменение записей

## Синтаксис

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
UPDATE [ ONLY ] table_name [ * ] [ [ AS ] alias ]  
    SET { column_name = { expression | DEFAULT } |  
        ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |  
        ( column_name [, ...] ) = ( sub-SELECT )  
    } [, ...]  
[ FROM from_item [, ...] ]  
[ WHERE condition | WHERE CURRENT OF cursor_name ]  
[ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

<https://www.postgresql.org/docs/current/sql-update.html>

# Update

```
UPDATE    films
           SET    kind = 'Dramatic'
           WHERE  kind = 'Drama'
```

# Выборка записей

## Синтаксис

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]  
    [ * | expression [ [ AS ] output_name ] [, ...] ]  
    [ FROM from_item [, ...] ]  
    [ WHERE condition ]  
    [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]  
    [ HAVING condition ]  
    [ WINDOW window_name AS ( window_definition ) [, ...] ]  
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]  
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]  
    [ LIMIT { count | ALL } ]  
    [ OFFSET start [ ROW | ROWS ] ]  
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES } ]  
    [ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [ NOWAIT | SKIP LOCKED ] [...] ]
```

<https://www.postgresql.org/docs/14/sql-select.html>



# SELECT

- `select * from account`
- `select * from account where id = 1`
- `select id, name from account  
where name like "P%"`

# Литература

SQL. Полное руководство 3 изд [2019] Джеймс  
Грофф, Пол Вайнберг, Эндрю Оппель



# Выборка данных

# Команда SELECT – выборка данных

Общий синтаксис:

```
SELECT [{ ALL | DISTINCT }] { список_вывода | * }  
  FROM имя_таблицы1 [ алиас1 ] [, имя_таблицы2 [ алиас2 ],...]  
  [ WHERE      условие_отбора_записей ]  
  [ GROUP BY { имя_поля | выражение },... ]  
  [ HAVING      условие_отбора_групп ]  
  [ UNION [ALL] SELECT ...]  
  [ ORDER BY имя_поля1 | целое [ ASC | DESC ]  
    [, имя_поля2 | целое [ ASC | DESC ],...]];
```

Примеры:

```
select * from departs;
```

```
select name, post from emp;
```

# Однотабличные запросы

```
SELECT col1, col2, col3  
    FROM table_name  
    WHERE expression
```

# Выборка из таблицы

```
SELECT ID, name, countrycode, pulation  
FROM city;
```

# Арифметика в столбцах

```
SELECT id*200, name, countrycode+10  
FROM city;
```

# Синонимы для столбцов

```
SELECT    2+3 sign,  countrycode cc  
FROM      city;
```



# Предикат условия where

```
SELECT * FROM city WHERE id = 123;
```

#конструкция для набора IN

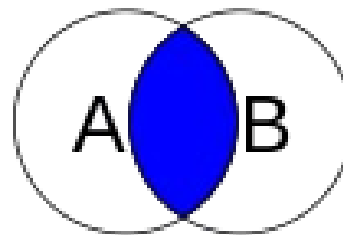
```
SELECT *  
  FROM city  
 WHERE id in (123, 7, 5);
```

#выбор по шаблону LIKE

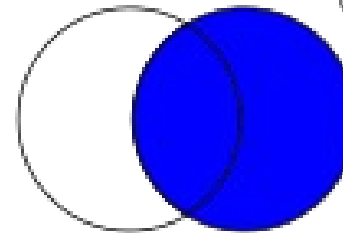
```
SELECT * FROM city  
 WHERE name like '%Petersburg%';
```

# Многотабличные запросы

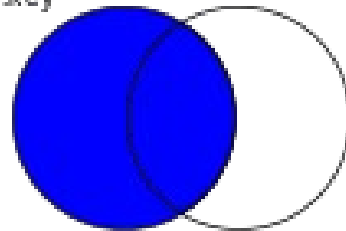
SELECT <fields>  
FROM TableA A  
INNER JOIN TableB B  
ON A.key = B.key



SELECT <fields>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.key = B.key

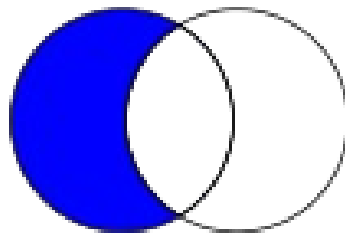


SELECT <fields>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.key = B.key

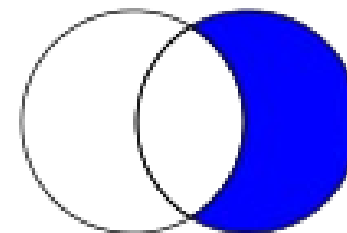


# SQL JOINS

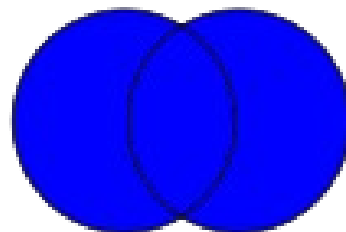
SELECT <fields>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.key = B.key  
WHERE B.key IS NULL



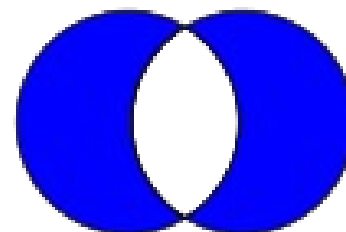
SELECT <fields>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.key = B.key  
WHERE A.key IS NULL



SELECT <fields>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.key = B.key



SELECT <fields>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.key = B.key  
WHERE A.key IS NULL  
OR B.key IS NULL



# Многотабличные запросы

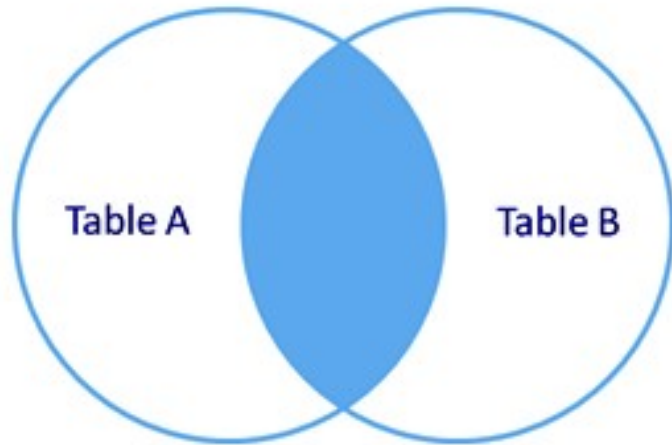
**CROSS JOIN:** Перекрестное соединение, возвращает комбинации каждой записи первой таблицы с каждой записью второй таблицы.

**INNER JOIN:** Внутренним соединением называется перекрестное соединение, из результатов которого часть записей исключена по условию запроса.

**LEFT JOIN:** В левом внешнем соединении для КАЖДОЙ ЗАПИСИ ЛЕВОЙ таблицы ищется соответствие среди записей правой таблицы.

**RIGHT JOIN:** Правое внешнее соединение ищет в левой таблице соответствия для правой таблицы.

# INNER JOIN



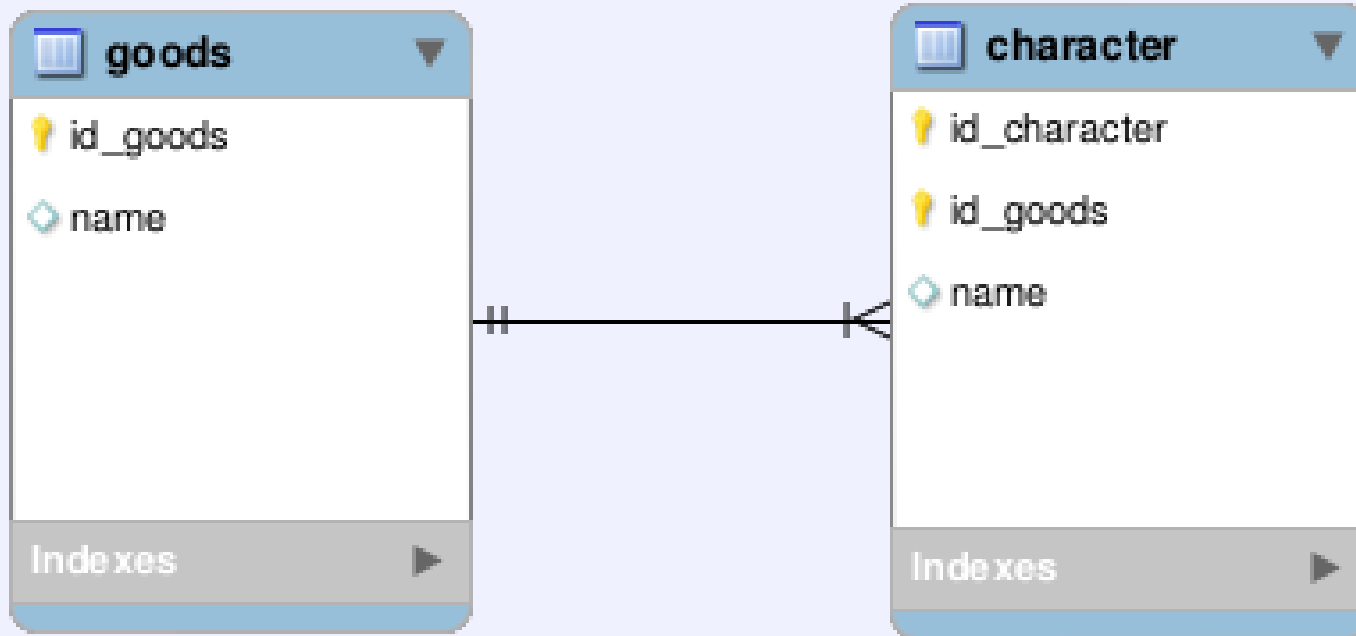
*pic. Inner join*

Шаблон запроса:

```
SELECT    a.name , b.value
FROM      table1 a, table2 b
WHERE     a.id = b.id
```

# Модель

Конструктор товаров



# Данные

## GOODS

ID_GOODS	NAME
1	Книга
2	Жесткий диск
3	Системный блок
4	Монитор

## CHARACTER

ID_CHARACTER	ID_GOODS	NAME
1	1	Автор
2	1	Кол-во страниц
3	1	Издательство
4	1	Год выпуска
5	2	Объем
6	2	<b>Скорость вращения шпинделя</b>
7	2	Интерфейс
8	3	Жесткий диск
9	4	Процессор

# Запрос

(связь по ключу PK = FK)

```
SELECT g.name, ch.name  
  FROM goods g, character ch  
 WHERE g.id_goods = ch.id_goods  
       AND g.id_goods = 1
```

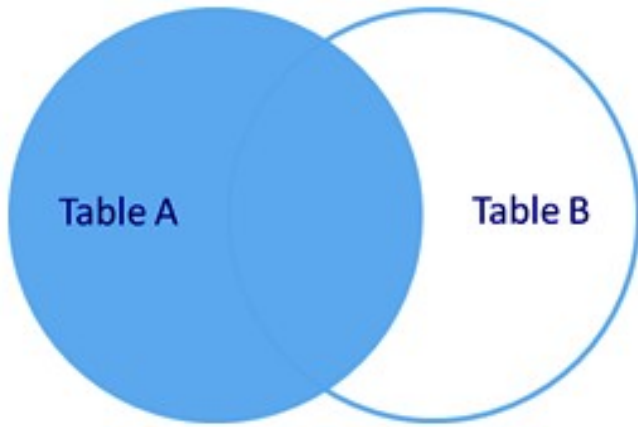
Какой результат получим ?



## CROSS JOIN

```
SELECT g.name, ch.name  
      FROM goods g, character ch
```

## LEFT JOIN

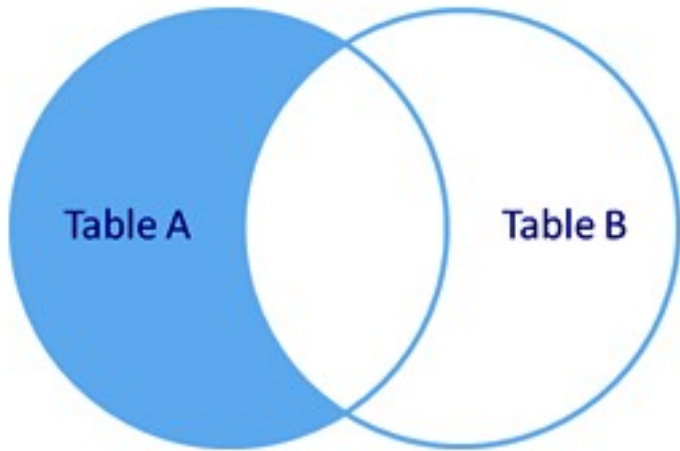


*pic. Left join*

получим все товары у  
которых заданы и не заданы  
характеристики.

```
SELECT a.name, b.name  
FROM goods a left join characters b  
on a.id_goods = b.id_goods;
```

# фильтр is not null

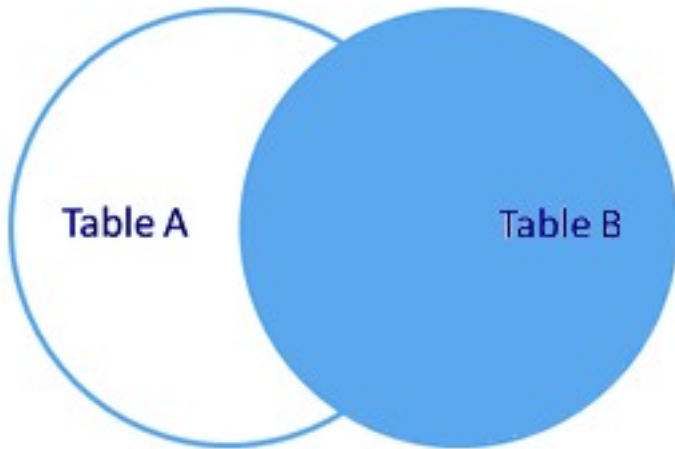


*рис. Left outer join с фильтрацией по полю*

Получим только те товары у которых не задана характеристика.

```
SELECT a.name, b.name
FROM goods a
      left join characters b
      on a.id_goods = b.id_goods
WHERE b.name IS NULL ;
```

## RIGHT JOIN



получим все характеристики  
у которых заданы и не  
заданы товары.

```
SELECT a.name, b.name
FROM goods a
      right join characters b
on a.id_goods = b.id_goods;
```

# Полное объединение

Союзы

```
SELECT n from numders1;  
UNION  
SELECT n from numbers2;
```

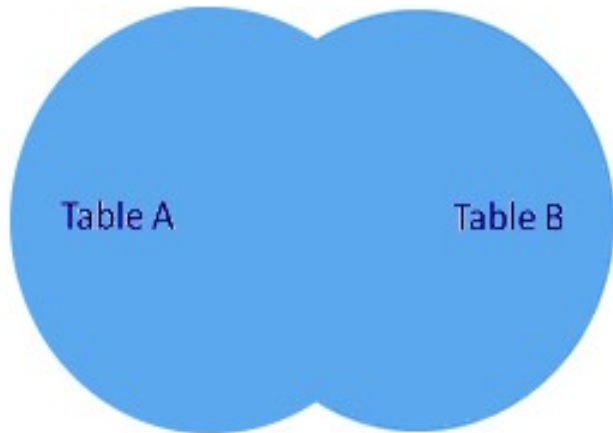
**UNION** — Объединяет в одну таблицу результаты 2-х и более запросов.

**UNION ALL** — Для получения списка со всеми дубликатами.

**INTERSECT** — Возвращает пересечение результатов нескольких запросов.

**EXCEPT** — Возвращает исключение результатов второго запроса из первого.

# FULL OUTER JOIN



получим полное  
пересечение  
соединений

```
SELECT a.name, b.name
FROM goods a
left join characters b
ON a.id_goods=b.id_goods
UNION
SELECT a.name, b.name
FROM goods a
right join characters b
ON a.id_goods=b.id_goods;
```

# Задание.

Для разрабатываемой модели построить  
многотабличные запросы исходя из  
бизнес-логики.