**Глоссарий**

|  |  |
| --- | --- |
| **Термин** | **Определение** |
| **База данных (БД)** | Хранилище структурированной информации. |
| **Реляционная база данных (РБД)** | БД, в которой данные организованы в виде связанных между собой таблиц. |
| **СУБД (DBMS)** | Система управления базами данных — ПО для создания, изменения и использования БД. |
| **PostgreSQL** | Мощная и гибкая реляционная СУБД с открытым исходным кодом. |
| **SQL** | Structured Query Language — язык структурированных запросов для работы с данными в РБД. |
| **Запрос** | Инструкция на SQL для получения, изменения или удаления данных. |
| **Таблица** | Основная структура хранения данных в РБД. |
| **Запись (строка)** | Информация об одном объекте в таблице. |
| **Поле (столбец)** | Характеристика объекта, определённого типа. |
| **Ячейка** | Пересечение строки и столбца, содержащее значение. |
| **Первичный ключ** | Поле или группа полей для однозначной идентификации записей. Все значения первичного ключа уникальны. Первичный ключ из нескольких полей называется составным. |
| **Схема** | Логическое объединение объектов БД: таблиц, представлений, функций и др. |
| **ER-диаграмма** | Entity-Relationship Diagram — графическое отображение связей между сущностями. |

**Базы данных. Теория**

База данных — это репозиторий (хранилище) данных, которые упорядочены и связаны между собой.

Базы данных бывают разных видов в зависимости от способа хранения данных. Если данные в базе представлены в виде связанных таблиц, такую базу данных называют реляционной (англ. relation, «связь»).

В таблице реляционной базы данных не должно быть дубликатов. Если все записи в таблице уникальны, вы будете уверены, что работаете с нужной строкой. Для этого к каждой записи добавляется уникальный признак — первичный ключ. Такой ключ не позволит создать абсолютно одинаковые записи. Первичный ключ может быть не только номером, но и набором любых символов. Главное, чтобы ключ не повторялся.

**ER-диаграммы**

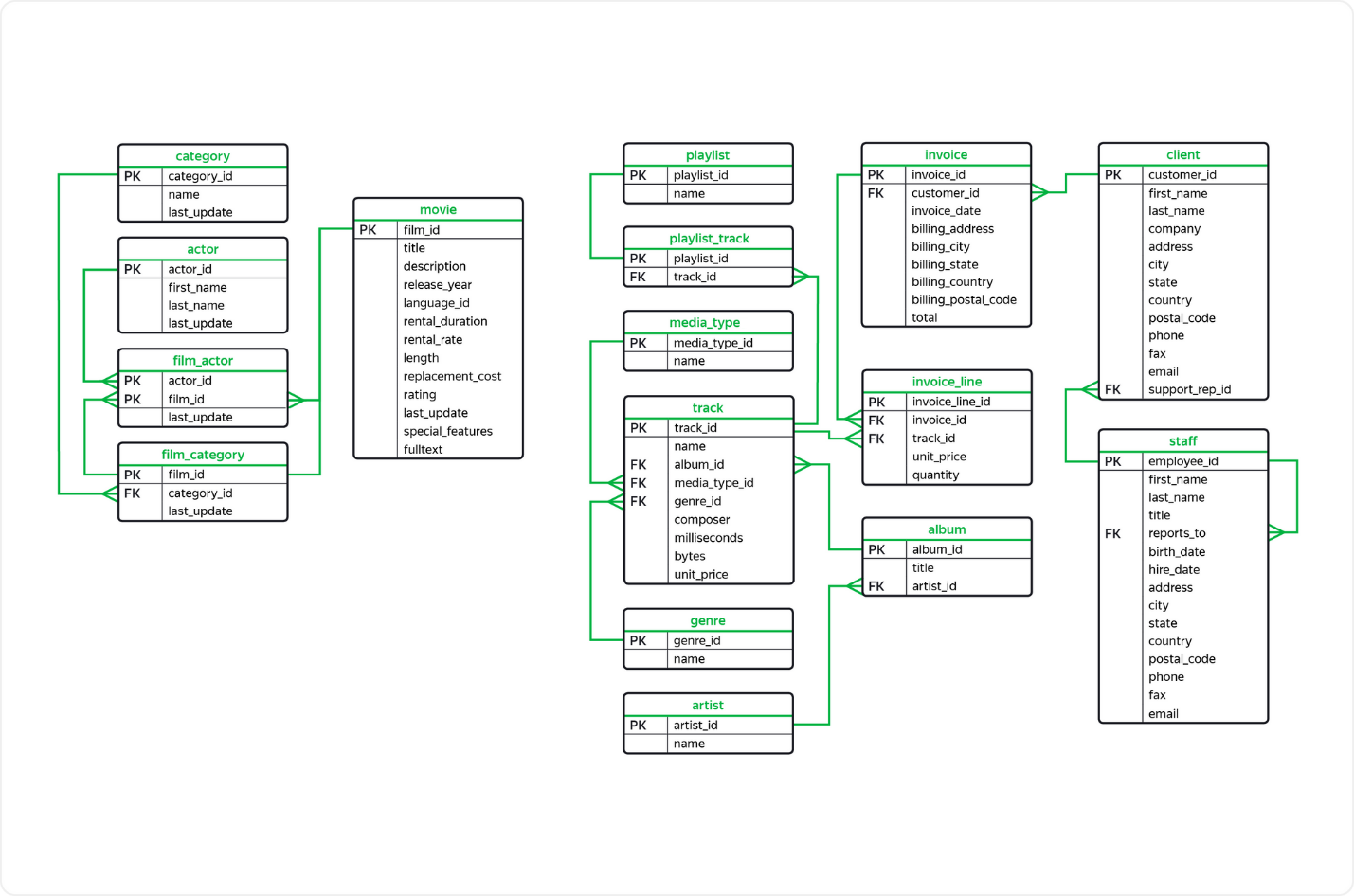
Разобраться в структуре базы поможет ER-диаграмма (от англ. entity-relationship diagram, диаграмма «сущность — связь»).

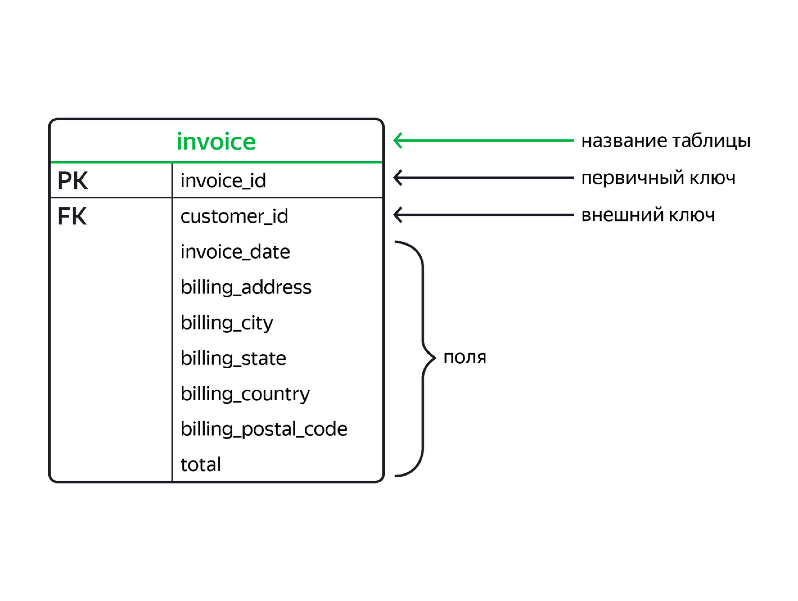
ER-диаграмма графически показывает, как таблицы связаны между собой.

Каждая сущность = таблица.

Связи (lines) показывают, по каким полям таблицы соединяются (например, user\_id).

Используется при проектировании и понимании структуры базы.





**Виды связей между таблицами**

1. Один к одному (One-to-One)

Каждой записи в одной таблице соответствует ровно одна запись в другой.

Пример: staff и таблица с разработчиками.

Используется редко, так как проще добавить нужные поля в одну таблицу.

2. Один ко многим (One-to-Many)

Одна запись в первой таблице может соответствовать нескольким во второй.

Пример: client и invoice. Один клиент может иметь много заказов.

Самый распространённый тип связи.

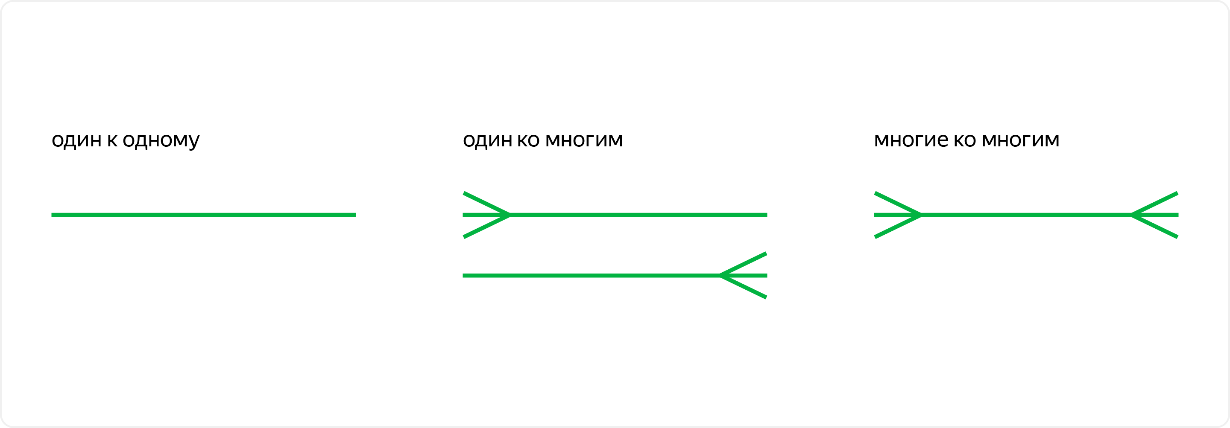
3. Многие ко многим (Many-to-Many)

Несколько записей в одной таблице соответствуют нескольким во второй.

Пример: преподаватели и предметы. Один преподаватель ведёт несколько предметов, и один предмет может вестись разными преподавателями.

Для реализации создаётся промежуточная таблица (связывающая).

Типы связей изображают разными условными обозначениями. На данной ER-диаграмме, типы связей отображены таким образом:



Помимо диаграммы, специалист часто работает **с описанием базы данных.**

**Схемы в базе данных**

Схема — это «папка» внутри базы данных.

В одной базе может быть несколько схем, каждая из которых содержит таблицы, функции и другие объекты.

В PostgreSQL по умолчанию используется схема public.

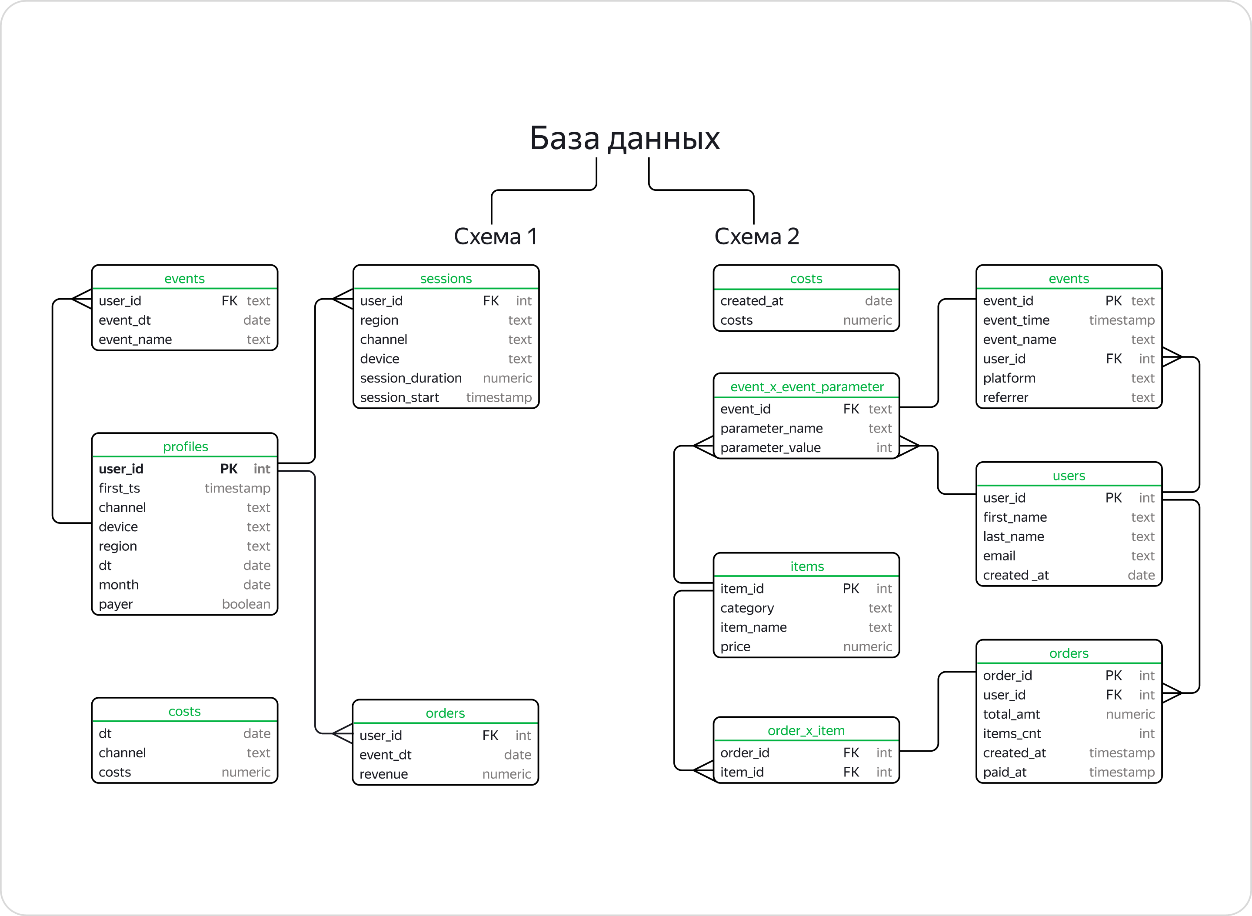
В реляционной базе данных схемы не обязаны быть напрямую связаны между собой, но они могут быть логически связаны, если это нужно архитектуре системы.

Но если нужно, через JOIN-ы можно делать запросы между таблицами разных схем:

SELECT a.name, b.salary

FROM hr.employees a

JOIN finance.salaries b ON a.emp\_id = b.emp\_id;



ER-диаграмма БД с двумя схемами

Чтобы обратиться к схеме базы данных, её нужно указать перед названием таблицы через точку:

SELECT \*

FROM schema\_name.table\_name;

В PostgreSQL по умолчанию используется схема public Если схема не указана, запрос обратится к схеме public.

SELECT \*

FROM table\_name;

и

SELECT \*

FROM public.table\_name;

равнозначны.

Различайте **схему** и **ER-диаграмму** базы данных.

* Схема — подбаза внутри основной базы.
* ER-диаграмма — графическое отображение сущностей базы.

**Преимущества баз данных**

**1. Масштабируемость и гибкость**

**Что это значит:**

* **Масштабируемость** — способность работать с большими объёмами данных. Если таблицы начинают тормозить при миллионах строк, то СУБД (например, PostgreSQL, MySQL) легко справляются с такими объёмами.
* **Гибкость** — возможность хранить разные типы данных:
  + текст, числа, даты;
  + изображения, документы (в формате BLOB);
  + связи между объектами (через внешние ключи).

**Почему это важно:**

* Позволяет создавать сложные структуры хранения, где каждая таблица отвечает за свою сущность (например, пользователи, заказы, товары), и все они связаны.
* При добавлении новых данных или изменении структуры таблиц не нужно переписывать всё с нуля, как это часто бывает в Excel.

**2. Целостность и непротиворечивость данных**

**Что это значит:**

* База данных может:
  + запрещать дублирующиеся значения (UNIQUE);
  + требовать наличие обязательных значений (NOT NULL);
  + обеспечивать правильные связи между таблицами (FOREIGN KEY);
  + поддерживать транзакции — гарантии, что все изменения произойдут или не произойдут вовсе.

**Пример:**

* Нельзя удалить заказ, если на него есть отгрузка — база просто не даст это сделать, пока не уберёшь зависимость.

**Почему это важно:**

* Гарантируется корректность данных.
* Уменьшается риск ошибок, которые в обычных таблицах могут пройти незамеченными.

**3. Надёжность и безопасность**

**Что это значит:**

* Возможность:
  + настроить доступ: один пользователь может только читать, другой — редактировать;
  + включить шифрование: защита от утечек;
  + вести журнал изменений (audit log).

**Почему это важно:**

* Можно безопасно хранить чувствительные данные (например, персональные данные пользователей).
* Позволяет контролировать, кто и что изменил в базе — это невозможно в простом Excel-документе.

**4. Улучшенная производительность**

**Что это значит:**

* За счёт **индексов** запросы работают быстро, даже если таблицы содержат миллионы строк.
* СУБД может кэшировать часто запрашиваемые данные.
* Можно выполнять **сложные запросы (JOIN, GROUP BY, подзапросы)** быстро и эффективно.

**Пример:**

* Найти всех пользователей, которые за последние 30 дней сделали покупки более чем на 10 000 рублей — в таблице это потребует много ручных действий, в SQL — одна строка.

**Почему это важно:**

* Оптимизация работы с большими объёмами информации.
* Автоматизация рутинных операций.

**5. Обмен данными и сотрудничество**

**Что это значит:**

* СУБД позволяет:
  + работать многим пользователям одновременно (без конфликтов и потерь данных);
  + подключать другие приложения (через API, ETL, BI-инструменты).

**Пример:**

* Один пользователь может загружать данные, другой — строить отчёты, третий — анализировать их в Power BI или Jupyter.

**Почему это важно:**

* Упрощает совместную работу внутри команды или между отделами.
* База данных может быть единственным источником истины (Single Source of Truth).

**СУБД и PostgreSQL**

В базах данных роль библиотекаря выполняет специальная система управления базами данных, или СУБД.

СУБД — это программное обеспечение для управления БД.

По-английски: DBMS - database management system.

RDBMS (Relational DBMS) — СУБД для реляционных баз.

СУБД способна на многое, например, с её помощью можно:

* создать базу или таблицу в базе;
* внести новые данные или удалить устаревшие;
* выгрузить нужную информацию, задав условие;
* обеспечить безопасный доступ к данным.

СУБД существует очень много. PostgreSQL это мощная СУБД с открытым исходным кодом, у которой почти нет ограничений по объёмам. С PostgreSQL можно работать на Windows, Linux и Mac.

Перечислим несколько преимуществ PostgreSQL:

* бесплатна и с открытым исходным кодом;
* нет ограничений по размеру базы данных;
* большой выбор встроенных языков программирования;
* возможность создавать собственные типы данных.

Помимо всего, PostgreSQL — популярная СУБД. В компаниях, которым требуются специалисты по SQL, часто используют PostgreSQL в качестве основного инструмента.

**Типы данных в PostgreSQL**

В PostgreSQL выделяют несколько групп типов данных:

* числовые типы,
* символьные типы: character
* логические типы,
* типы для работы с датой и временем,
* специализированные типы для конкретных задач.

**Числовые типы**

| **Тип** | **Описание** |
| --- | --- |
| integer / int / int4 | Целые числа от -2 147 483 648 до +2 147 483 647 |
| real / float4 | Вещественные числа (с плавающей точкой) |
| numeric / decimal | Вещественные числа с точностью, без потери значений |

Целые и дробные числа относятся к разным типам данных. Дробные числа иначе называют вещественными.

Тип integer хранит целые числа от -2147483648 до +2147483647. Для простых задач этого хватает. Тип данных integer обозначают либо int, либо int4. Такое обозначение называется псевдонимом. Псевдонимы есть не у всех типов данных, их часто используют по историческим причинам. Да, так действительно написано в документации PostgreSQL.

Тип real обозначает вещественные числа. У этого типа тоже есть псевдоним — float4.

*Типы данных для вещественных чисел*

real (4 байта, ~6 цифр точности)

double precision (8 байт, ~15 цифр)

numeric и decimal (переменный размер, до 131072 цифр) — предпочтительны для точных расчётов

numeric используется по умолчанию, корректно округляет (в отличие от real и double precision, которые округляют к ближайшему чётному)

*Приведение типов данных*

В PostgreSQL деление integer / integer возвращает integer с усечением (9 / 10 → 0).

Чтобы получить дробные значения, нужно:

CAST(column AS numeric)

column \* 1.0

column::numeric

Предпочтительно использовать CAST, т.к. он совместим со стандартом SQL.

**Символьные типы**

| **Тип** | **Псевдоним** | **Описание** |
| --- | --- | --- |
| character(n) | char(n) | Фиксированная длина строки, заполняется пробелами |
| character varying(n) | varchar(n) | Строка переменной длины, максимум n |
| text | — | Строка произвольной длины (спец. тип PostgreSQL) |

Тип character содержит текст фиксированной длины. Его обозначают псевдонимом char(n). На месте символа n может стоять любое число. Например, если вместо n подставить 4, строка будет состоять только из четырёх символов. Больше нельзя, но меньше можно: недостающие символы будут заполнены пробелами. Если не указывать n, его значение по умолчанию будет 1.

В тип character varying входят тексты нефиксированной длины. Его псевдоним — varchar(n). Значение n управляет лимитом строки, но его можно не указывать.

Тип text содержит строку любой длины. Такого типа в стандарте SQL нет, но в некоторых СУБД, помимо PostgreSQL, его тоже используют.

**Логические типы**

В PostgreSQL к типу boolean относят два значения: TRUE и FALSE. Но эти значения можно указывать по-разному.

Для TRUE список аналогий такой: 'true', 't', 'yes', 'y', 'on', '1'.

Вместо FALSE можно написать: 'false', 'f', 'no', 'n', 'off', '0'.

**Типы для работы с датой и временем**

| **Тип** | **Описание** |
| --- | --- |
| date | Только дата |
| time | Только время (от 00:00:00 до 24:00:00) |
| timestamp | Дата и время без часового пояса |
| timestamptz | Дата и время с часовым поясом |
| interval | Интервал между двумя датами |

Тип timestamp содержит дату и время. Чаще всего дату и время указывают в формате ISO: 'YYYY-MM-DD'. Например, 30 июня 2010 года записывают так: '2010-06-30'.

В PostgreSQL типов timestamp два: timestamp with time zone, включающий данные о часовом поясе, и timestamp without time zone, который данных о часовом поясе не содержит.

Часовые пояса указывают по стандарту UTC: '2004-10-19 10:23:54+02'. В часовом поясе UTC+2 находится, например, Калининград. Сравните со значением типа timestamp without time zone: '2004-10-19 10:23:54'.

По стандарту SQL обозначение timestamp по умолчанию подразумевает тип timestamp without time zone. Для обозначения типа timestamp with time zone в PostgreSQL используют короткую форму timestamptz.

В тип date входит только дата. Дату можно задавать в любом формате, например, October 19, 2004, 2004-Oct-19 или даже 20041019.

Тип time хранит только время и принимает значения от 00:00:00 до 24:00:00.

Тип interval используют для обозначения интервала между датами. Задать нужный интервал можно с помощью условных обозначений или в свободной форме. Например, запись '1 12:59:10' обозначает 1 день 12 часов 59 минут 10 секунд.

**PostgreSQL через Python**

Для работы с PostgreSQL через Python, нужно использовать библиотеку psycopg2. Она реализует Python DB API 2.0 и является самым популярным драйвером.

# **Установка psycopg2**

pip install psycopg2-binary

-binary — это версия, не требующая сборки, удобна для большинства случаев.

# **Подключение и основные действия**

import psycopg2

# Подключение к базе

conn = psycopg2.connect(

dbname="mydatabase",

user="postgres",

password="secret",

host="localhost",

port=5432

)

# Курсор для выполнения SQL

cur = conn.cursor()

# Создание таблицы

cur.execute("""

CREATE TABLE IF NOT EXISTS users (

id SERIAL PRIMARY KEY,

name TEXT NOT NULL,

age INTEGER

)

""")

# Вставка данных

cur.execute("INSERT INTO users (name, age) VALUES (%s, %s)", ('Egor', 30))

# Сохранение изменений

conn.commit()

# Чтение данных

cur.execute("SELECT \* FROM users")

rows = cur.fetchall()

for row in rows:

print(row)

# Закрытие

cur.close()

conn.close()

**Язык запросов SQL**

Для работы с базой данных существует свой язык программирования. Реляционными базами данных управляют с помощью языка SQL, или Structured Query Language. По-английски это означает «язык структурированных запросов».

Важная особенность: SQL — это декларативный язык, который описывает, что делать, а не как делать. В SQL описывают не алгоритм действий, а данные, которые хотят получить. Как выполнить такую команду, за вас решит СУБД.

Язык SQL — это стандарт, который используют почти во всех реляционных базах данных.

У СУБД PostgreSQL есть набор дополнительных команд, расширяющих стандартные возможности языка SQL.

**Стиль запроса в SQL**

**1. Регистр символов**

Ключевые слова SQL (например, SELECT, FROM, WHERE, JOIN) пишутся **в верхнем регистре**, а имена таблиц и столбцов — **в нижнем**. Это делает структуру запроса визуально читаемой и позволяет быстрее различать команды и данные.

SELECT first\_name, last\_name

FROM employees

WHERE department = 'HR';

**2. Форматирование списка столбцов**

Каждый столбец, особенно если их несколько, переносится на новую строку. Это улучшает читаемость и облегчает поддержку кода:

SELECT

first\_name,

last\_name,

email,

hire\_date

FROM employees;

**3. Выравнивание и отступы**

Отступы используются для обозначения вложенности, например, в подзапросах или при использовании JOIN. Обычно используют отступ в 2 или 4 пробела.

SELECT

e.first\_name,

e.last\_name,

d.department\_name

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;

**4. Алиасы**

Алиасы (AS) задаются явно, особенно для столбцов с вычислениями или при объединении таблиц. Это облегчает понимание результата запроса.

SELECT

e.salary \* 12 AS annual\_salary

FROM employees e;

**5. Фильтрация и логика**

Условия в WHERE, ON, HAVING пишутся с каждого логического выражения на новой строке, особенно если их несколько. Это повышает читаемость и снижает риск ошибок:

WHERE

department = 'HR'

AND salary > 50000

**6. Порядок операторов**

Соблюдается стандартный порядок ключевых слов:

* SELECT
* FROM
* JOIN
* WHERE
* GROUP BY
* HAVING
* ORDER BY
* LIMIT / OFFSET

**7. Комментарии**

Используйте комментарии для пояснения сложных частей запроса:

-- Выбираем сотрудников, нанятых после 2020 года

SELECT

first\_name,

hire\_date

FROM employees

WHERE hire\_date > '2020-01-01';

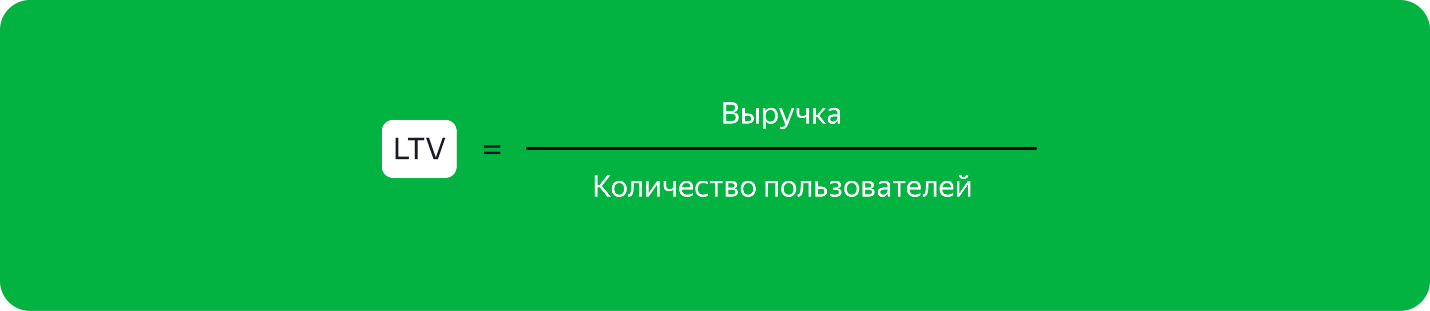
**Бизнес метрики**

**Конверсия** — это доля людей, перешедших из одного состояния в другое. Например, доля посетителей, ставших покупателями.

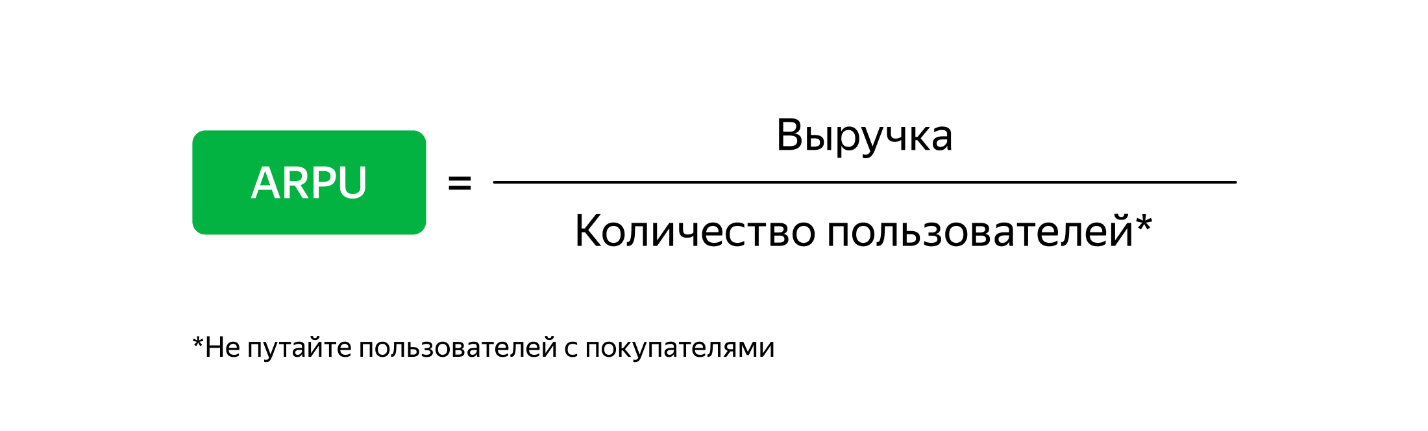
Сокращённо конверсию называют CR от англ. conversion rate, «показатель конверсии». Если в магазин зашли 100 человек и 30 из них что-нибудь купили, то CR составит 30 / 100 = 0.3, или 30%.

**LTV,** или Lifetime Value, показывает «пожизненную ценность» пользователей, то есть общую сумму денег, которую один пользователь в среднем приносит компании со всех своих покупок.

LTV считают так:



**ARPU**, или Average Revenue Per User, — это метрика, которая показывает выручку на всех пользователей, включая клиентов. Часто ARPU — это полный аналог LTV.

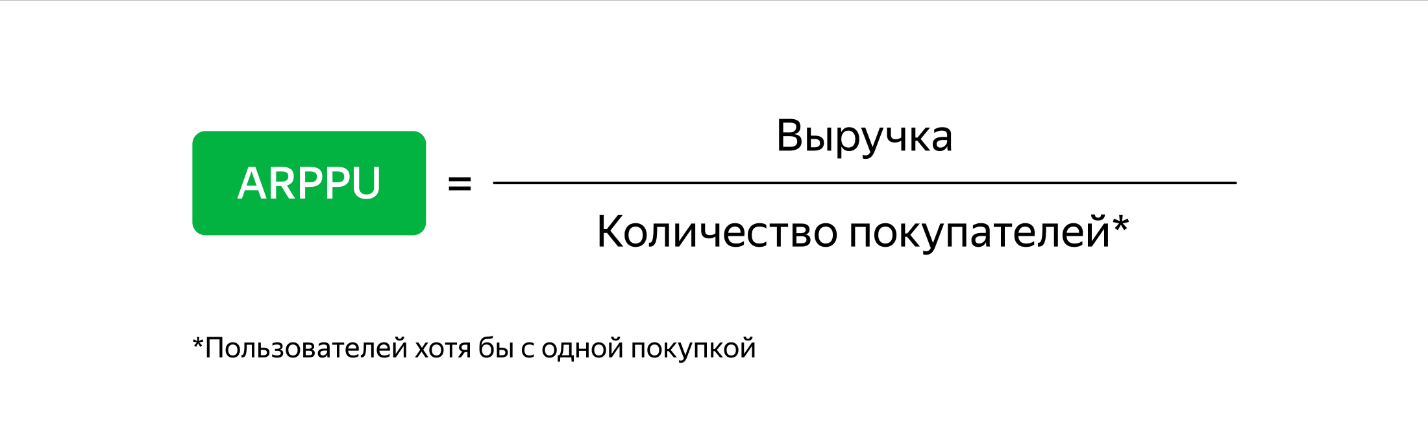


SELECT SUM(o.revenue) / COUNT(DISTINCT p.user\_id) AS arpu

FROM online\_store.profiles p

LEFT JOIN online\_store.orders o ON o.user\_id = p.user\_id;

**ARPPU**, или Average Revenue Per Paying User, наоборот, учитывает только тех пользователей, которые совершили хотя бы одну покупку.

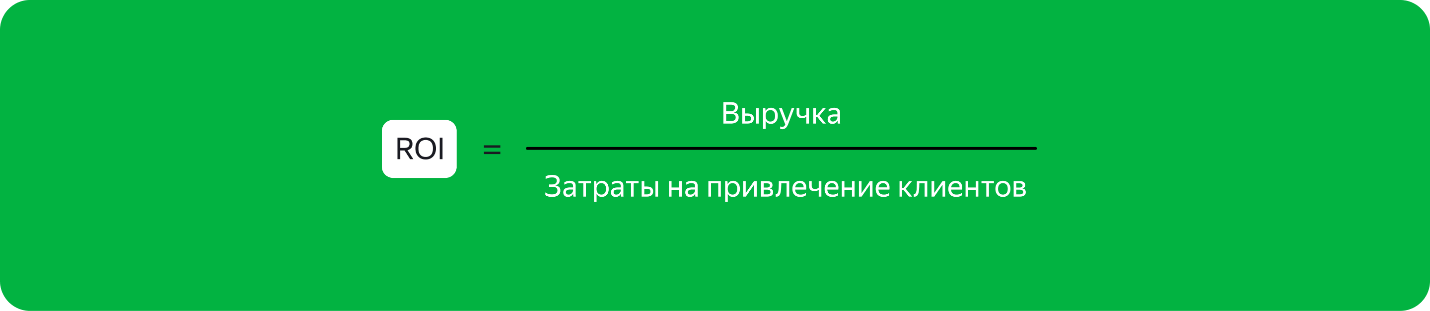


SELECT SUM(o.revenue) / COUNT(DISTINCT p.user\_id) AS arppu

FROM online\_store.orders o

JOIN online\_store.profiles p ON o.user\_id = p.user\_id;

**ROI**, или Return On Investment, используют для оценки окупаемости инвестиций. Эта метрика показывает, на сколько процентов выручка, которую приносит клиент компании, превышает стоимость привлечения клиента.



SELECT dc.dt,

revenue \* 100 / costs AS ROI

FROM

(SELECT dt,

SUM(costs) AS costs

FROM online\_store.costs

GROUP BY dt) dc

JOIN

(SELECT event\_dt AS dt,

SUM(revenue) AS revenue

FROM online\_store.orders

GROUP BY event\_dt) dr ON dc.dt = dr.dt;

**Когортный анализ**

Аналитики делят клиентов компании на группы — когорты.

Участников всех когорт изначально объединяет какое-нибудь **общее событие**. Зачастую это первое посещение сайта, регистрация или скачивание мобильного приложения. Так в анализ попадает определённая категория людей — впервые посетившие сайт, зарегистрированные в программе лояльности, пользователи приложения. После этого их делят на когорты по выбранным признакам.

Чем различаются участники разных когорт:

* Время, в которое произошло общее событие.
* Дополнительные признаки. Например, возраст и пол, профессия, география, поведенческие особенности.

В цифровых сервисах когортный анализ применяют, чтобы сравнить «качество» пользователей.

**Retention Rate** (англ. «коэффициент удержания») показывает, сколько пользователей из когорты относительно их изначального числа вернулись, то есть воспользовались продуктом или услугой в последующие периоды.





**Churn Rate** - метрика, обратная Retention Rate



**Подключение к БД через Python**

Для подключения к базе данных часто используют библиотеку SQLAlchemy — она одна из самых распространённых. Есть и другие библиотеки для подключения, например psycopg2. Задачи и принципы работы библиотек похожи.

Прежде чем использовать библиотеку, нужно её установить. Запустите команду для установки SQLAlchemy в вашей тетрадке Jupyter Notebook:

!pip install sqlalchemy

Импортируйте остальные библиотеки. Вам понадобится pandas для работы с датафреймами и функция create\_engine() из библиотеки SQLAlchemy для подключения к базе данных и выполнения SQL-запроса:

import pandas as pd

from sqlalchemy import create\_engine

Для подключения к базе данных вам понадобится:

* имя пользователя базы данных и пароль;
* адрес сервера базы данных, включая порт;
* название базы данных.

Эти данные удобно записать в объект, чтобы потом их было легко поправить. Для подключения к базе данных Практикума используйте следующие данные:

db\_config = {

'user': 'praktikum\_student', # имя пользователя

'pwd': 'Sdf4$2;d-d30pp', # пароль

'host': 'rc1b-wcoijxj3yxfsf3fs.mdb.yandexcloud.net',

'port': 6432, # порт подключения

'db': 'data-analyst-advanced-sql' # название базы данных

}

Строку для подключения можно сформировать с помощью метода .format():

connection\_string = 'postgresql://{}:{}@{}:{}/{}'.format(

db\_config['user'],

db\_config['pwd'],

db\_config['host'],

db\_config['port'],

db\_config['db'],

)

Чтобы создать соединение, строку с данными для подключения передают функции create\_engine():

engine = create\_engine(connection\_string)

Запрос можно записать в переменную query — так его удобнее редактировать.

query = '''

SELECT ...

FROM ...

'''

Результат выдачи SQL-запроса можно записать в датафрейм с помощью метода pd.read\_sql\_query():

df = pd.read\_sql\_query(query, con = engine)

Теперь с данными SQL-запроса можно работать как с датафреймом.