Я решила написать эту статью, потому что именно такой статьи мне очень не хватало несколько лет назад, когда я только начала карьеру в аналитике данных. Тогда я часто слышала слова «база данных», «реляционная база», «primary key», примерно понимала, что они означают, но единую картину в голове у меня сложить не получалось.

\_\_\_\_\_\_\_\_\_\_\_

Что такое база данных и зачем она?

Компании часто собирают информацию о своих клиентах, сотрудниках, операциях, финансах и т. д. Потом эту информацию можно выгодно использовать. Например, можно ее проанализировать и понять, какими способами можно увеличить прибыль. Можно на ее основе построить хитрые MLмодели, которые помогут улучшить продукт. Или, в конце концов, эту информацию можно просто перепродать другим компаниям.

Чтоб собирать и анализировать информацию, надо уметь ее сохранять. Конечно, можно сохранять информацию в печатном виде в обычных папках или в Excel-файлах. И многие компании до сих пор так сохраняют информацию. Однако, такое подойдет только для маленьких компаний с небольшим количеством данных. Когда компания вырастает, то и данных становится много, такие варианты сохранения информации становятся непригодны. Тогда на помощь приходят базы данных.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Базы данных помогают справиться с большим количеством проблем, решить которые папкам и Excel-файлам не под силу:

* В базе данных можно хранить очень огромное количество данных – миллиарды и триллионы записей;
* Базы помогают защищать данные - они позволяют давать доступ к данным только определенному кругу лиц. При этом можно ставить ограничения, кому к каким данным можно давать доступ и какого типа доступ, только чтение или редактирование тоже;
* Базы данных могут помогать следить за правильностью данных с помощью различного вида проверок;
* Также, базы данных могут позволять большому количеству людей одновременно взаимодействовать с данными.
* **Надежность**:
  + Файлы можно случайно удалить, повредить, потерять историю изменений. В базах данных есть резервное копирование, транзакции и механизмы восстановления.
* **Безопасность**:
  + В СУБД есть ролевой доступ (например, бухгалтер не увидит данные HR), шифрование, аудит действий.
* **Совместная работа**:
  + В Excel конфликтуют версии при редактировании, а базы данных позволяют множеству пользователей работать параллельно.
* Возможность настройки.

*"Представьте интернет-магазин во время Black Friday: тысячи заказов в минуту. Excel «упадёт», а БД автоматически распределит нагрузку."*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Так что же такое база данных? Если говорить коротко, то это **определенная** структура, в которой хранится информация. Я понимаю, что из этого определения пока мало что понятно. Однако, более конкретное определение дать сложно, потому что существует много типов баз данных, и все они совершенно разные.

*"Представьте, что база данных — это библиотека. Excel/папки — это стопка бумаг на столе, а БД — это каталогизированная система с полками, индексами и библиотекарем (СУБД), которая быстро находит нужную книгу."*

Упомянуть **СУБД** (системы управления базами данных) как «движок» для работы с БД.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Я думаю, это определение станет понятнее, когда я далее опишу наиболее популярные типы баз данных на конкретных примерах.

Типы баз данных

Существует много разных типов баз данных. Наиболее популярные типы:

* Реляционные базы данных
* Key-value базы данных
* Документно-ориентированные базы данных
* Графовые базы данных
* Колоночные базы данных

Далее я расскажу о каждом из этих типов. Однако, начну я реляционных баз данных и больше всего буду рассказывать о них, потому что именно этим типом баз данных чаще всего пользуются аналитики данных и data scientist-ы.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

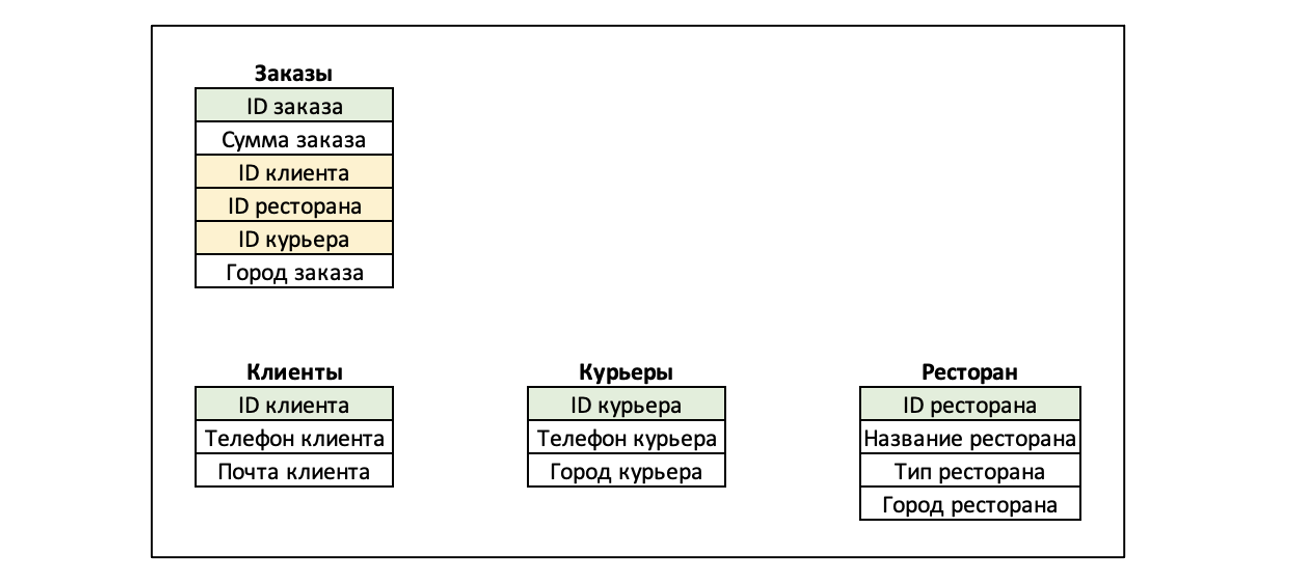
Реляционные базы данных (MySQL, PostgreSQL, Oracle DB)

**Реляционная база данных** – это база данных, которая состоит из таблиц. У реляционной базы данных 2 очень важные характеристики:

* Данные распределены по смыслу по таблицам
* Между таблицами есть отношения

Рассмотрим пример реляционной базы. Допустим, у нас есть сервис доставки еды. Тогда, если мы построим реляционную базу данных для этого сервиса, то она, скорее всего, будет содержать следующие таблицы:

* Таблица с заказами
* Таблица с клиентами
* Таблица с курьерами
* Таблица с ресторанами



На рисунке 1 я попыталась изобразить графически реляционную базу данных. Мы видим таблицы, из которых состоит база, и также видим, какие столбцы содержит каждая из таблиц.

Как я отметила выше, второй важной характеристикой реляционных баз данных является то, что между таблицами существуют отношения. Отношения между таблицами определяются с помощью primary key и foreign key.

**Primary key** – это столбец (или группа столбцов) таблицы, который содержит уникальные значения для каждой строки. На примере выше primary key каждой таблицы я выделила зеленым цветом. То есть, например, в таблице с заказами каждая строка будет описывать отдельный заказ. Не будет 2 строк, которые описывают один и тот же заказ, потому ID заказа будет разный для каждой строки.

**Foreign key** – это столбец в таблице, который содержит primary key другой таблицы. На рисунке foreign key отмечены желтым. То есть, таблица с заказами содержит ID клиента, который является primary key в таблице с клиентами, но в таблице с заказами он будет foreign key.

Primary key и foreign key помогают не только связывать между собой таблицы реляционной базы данных отношениями. Они еще помогают следить за целостностью и правильностью данных в базе. Например, если мы ошибемся в ID клиента, добавляя новый заказ в таблицу с заказами, то база выдаст ошибку, так как не найдет соответствующий ID клиента в таблице с клиентами.

Для взаимодействия с реляционными базами данных чаще всего используется **SQL** (Structured QueryLanguage). Это специальный язык программирования, на котором пишутся запросы к реляционной базе. SQL-запросами можно создавать и удалять таблицы в реляционной базе, изменять данные в существующих таблицах и доставать из таблиц необходимую информацию.

Как я уже говорила выше, реляционные базы данных удобно использовать в аналитике, так как информация в них структурирована и распределена по смыслу, что, конечно, мечта любого аналитика. Однако, аналитики часто пишут сложные и не очень эффективные SQL-запросы, потому важно придумывать способы ускорения обработки запросов к реляционной базе.

Одним из наиболее популярных методов ускорения работы запросов к реляционным базам данных является **индексирование таблиц**. Индекс – это определенный столбец в таблице, по которому осуществляется поиск.

Приведу пример работы индекса. Например, мы хотим найти все заказы клиента 007 из ресторана 1. Тогда, если у нас в таблице с заказами нет индекса, то мы будем перебирать все заказы пока не найдем нужные. Если же у нас есть индекс в таблице с заказами, то ситуация будет иной. Допустим, что индексом является столбец ID ресторана. Тогда наши данные в таблице с заказами будут сгруппированы по ID ресторана. И тогда при поиске заказов клиента 007 из ресторана 1, мы не будем перебирать всю таблицу с заказами, а найдем группу заказов из ресторана 1 и будем искать необходимые данные внутри этой группы.

Из примера выше с индексом выше понятно, что индексом удобно выбирать такой столбец, в разрезе которого часто ищутся данные.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Также, одним из важных свойств реляционных баз данных является **соответствие требованиям ACID**. Я не буду углубляться в детали этих требований, только отмечу, что эти требования гарантируют целостность и корректность данных, несмотря на ошибки, системные сбои, перебои в питании, изменение данных несколькими пользователями одновременно и прочие необычные ситуации.

*"ACID — это как правила банковских транзакций: даже если свет выключится во время перевода, деньги не исчезнут."*

Выглядит так, что реляционная база данных идеальная база, и непонятно, почему бы постоянно ее не использовать. Однако, у реляционной базы данных есть и недостатки, и потому данный тип не всегда подходит для нужд бизнеса. Например, реляционная база данных не подходит для данных без четкой структуры, потому что мы не сможем разложить эти данные в отдельные таблицы по смыслу. А данных без четкой структуры гораздо больше, чем данных с четкой структурой.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Какие еще есть типы баз данных?

Прочие типы баз данных, которые не реляционные, часто называются noSQL базы данных. Обсудим наиболее популярные типы нереляционных баз данных.

**Key-value базы данных** (пример - Redis)

Название говорит о том, какие данные удобно хранить в Key-value базе – в такой базе хранят данные, которые удобно представить в виде пары ключ-значение. Основное преимущество таких баз – это очень быстрый поиск значения по ключу. При этом значение может содержать какие угодно типы данных.

Такие базы данных удобно применять в проектах, где необходимо выдавать быстрый результат по ключу, например, для онлайн торгов или сделок.

🔍 **Как это работает?**

Представьте словарь:

* **Ключ** — это слово (уникальный идентификатор, например user\_123).
* **Значение** — это определение (любые данные: число, строка, JSON, даже изображение).

**Пример хранения**:

| **Ключ** | **Значение** |
| --- | --- |
| user\_123 | {"name": "Алиса", "age": 25} |
| config:dark\_mode | true |
| last\_orders:2024 | [100, 203, 505] |

💡 **Зачем это нужно?**

Key-Value БД идеальны для:

1. **Кэширования** (ускорение сайтов и приложений):
   * Например, Redis хранит кэш страниц по ключу page:/home.
2. **Сезонных данных**:
   * Корзины покупок, сессии пользователей.
3. **Настроек и конфигов**:
   * Флаги функций (feature:dark\_mode = true).
4. **Очередей задач**:
   * RabbitMQ и Kafka используют похожий принцип.

🚀 **Преимущества**

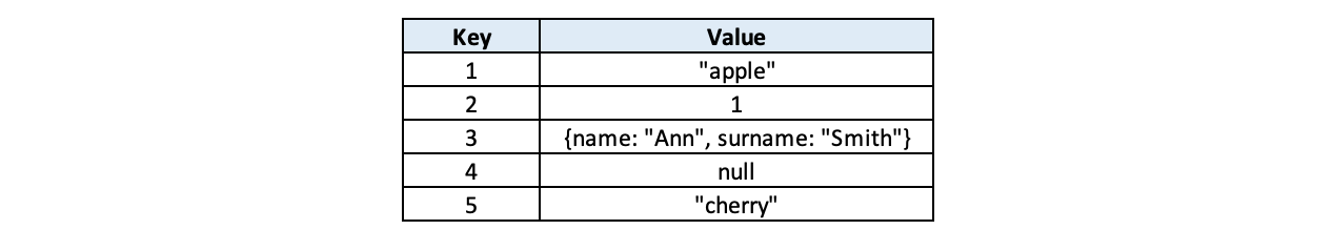
✅ **Скорость**:

* Запросы занимают **менее 1 мс** (ключ ищется по хэшу, как в словаре).  
  ✅ **Масштабируемость**:
* Легко распределять данные по серверам (шардирование).  
  ✅ **Простота**:
* Никаких схем, индексов, JOIN-ов — только get/set.

⚠️ **Ограничения**

❌ **Нет сложных запросов**:

* Нельзя сделать SELECT ... WHERE — только поиск по точному ключу.  
  ❌ **Минимальная аналитика**:
* Подсчёт суммы заказов потребует отдельного ключа (например, total\_orders:2024).

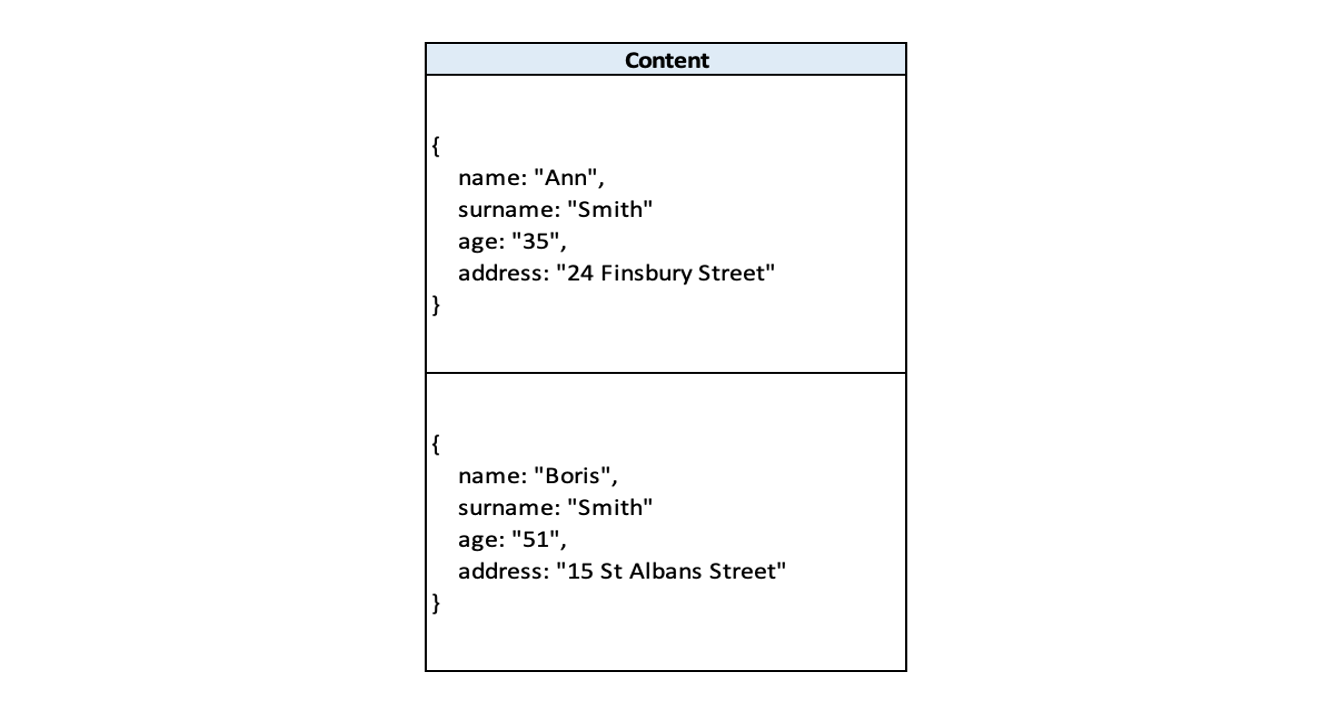


\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Документно-ориентированные** (пример - Mongo DB)

В документно-ориентированной базе данных единицей хранения является документ (который может быть в формате json, или xml, или в каком-нибудь еще формате). Удобство таких баз в том, что в них быстро и легко записывать любые типы данных, при этом эти данные не обязаны обладать четкой структурой. Минус таких баз в том, что данные в них неудобно анализировать.

В моей предыдущей компании такой тип баз данных служил базой для реляционных баз. То есть сначала все данные попадали и сохранялись в документно-ориентированной базе. Потом команда дата инженеров обрабатывала эти огромные полотна информации, структурировала и складывала в реляционную базу данных, которую уже могла использовать команда аналитиков и Data Science.



**Зачем они нужны?**

Идеальны для случаев, когда:

1. **Данные сложные и неструктурированные**:
   * Например, анкеты пользователей, где у кого-то есть поле "домашний питомец", а у кого-то нет.
2. **Нужна быстрая запись**:
   * Не требуется проверять связи (как в реляционных БД).
3. **Масштабируемость важнее строгой целостности**:
   * Подходят для логов, каталогов товаров, контент-платформ.

🚀 **Преимущества**

✅ **Гибкость схемы**:

* Можно добавлять поля "на лету" (например, новый атрибут товара в интернет-магазине).  
  ✅ **Простота разработки**:
* Данные хранятся так же, как в коде (JSON → легко работать с фронтендом).  
  ✅ **Горизонтальное масштабирование**:
* Легко распределять данные по серверам (например, MongoDB Sharding).

⚠️ **Ограничения**

❌ **Нет JOIN-ов**:

* Связи между документами сложнее, чем в реляционных БД.  
  ❌ **Избыточность данных**:
* Повторяющаяся информация (например, имя автора в каждом посте блога).  
  ❌ **Сложные транзакции**:
* Не все поддерживают ACID (кроме MongoDB с версии 4.0+).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Графовые базы данных** (пример - Orient DB)

Как следует из названия, в графовой базе данных данные хранятся в виде графов. Данный тип баз удобен, когда надо находить информацию не только о каком-то объекте, но и доставать информации о связах этого объекта с другими.

Например, данный тип баз используется для нахождения куки конкретного юзера и всех взаимосвязанных с этой кукой идентификаторов. Также, такой тип данных часто используется социальными сетями для сохранения информации не только о пользователях, но и о связях каждого пользователя с другими.

**Зачем это нужно?**

Графовые БД решают задачи, где **связи важнее самих данных**:

1. **Соцсети**:
   * «Кого может знать Алиса?» (поиск друзей друзей).
   * «Какие посты популярны в её круге общения?»
2. **Рекомендации**:
   * Amazon: «Люди, купившие этот товар, также брали вот это».
3. **Фрод-детекция**:
   * Выявление мошеннических схем (например, цепочки транзакций).
4. **Знаниевые графы**:
   * Google использует графы для поиска (например, связь «ИИ → Нейросети → Deep Learning»).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Колоночные (столбцовые) базы данных** (примеры - Cassandra, Clickhouse)

В реляционных базах данных данные записаны в виде строк. Что же касается колоночных баз данных, то тут данные записываются в виде столбцов. Потому поиск данных в колоночной базе данных осуществляется не перебором всех строк, как это происходит в реляционной базе данных, а поиском необходимого значения в тех столбцах таблицы, которые нас интересуют.

Преимущество колоночных баз данных в том, что они могут быстро находить определенные значения в столбцах, которые нас интересуют.

Каждый **столбец** хранится **отдельно** как независимый массив. Данные организованы *вертикально*:

Колоночные базы данных (Columnar DB) — это особый тип СУБД, где данные хранятся не построчно (как в классических реляционных базах), а **по столбцам**. Это кардинально меняет подход к хранению, чтению и анализу данных, делая такие БД идеальными для аналитики и Big Data.

**1. Как работают колоночные базы?**

**Отличие от строковых (реляционных) БД**

* **Строковые БД** (MySQL, PostgreSQL):
  + Данные хранятся **по строкам** (все поля записи лежат рядом).
  + Пример:

| order\_id | customer\_id | product\_id | price | date |
| --- | --- | --- | --- | --- |
| 1 | 101 | 500 | 29.99 | 2024-01-01 |
| 2 | 102 | 501 | 59.99 | 2024-01-02 |

* **Колоночные БД** (ClickHouse, Cassandra):
  + Данные хранятся **отдельными столбцами**, каждый в своём файле.
  + Пример:
    - Столбец order\_id: [1, 2]
    - Столбец price: [29.99, 59.99]

**Почему это эффективно?**

* **Сжатие данных**:  
  Значения в одном столбце часто повторяются (например, country=USA), что позволяет сильно сжимать данные (алгоритмы Run-Length Encoding, Delta Encoding).
* **Быстрые агрегации**:  
  Запросы вроде SUM(price) обрабатываются мгновенно, так как система читает только столбец price, а не всю строку.

**2. Плюсы колоночных БД**

✅ **Скорость аналитических запросов** (OLAP):

* Агрегации (SUM, AVG, GROUP BY) работают в **10-100x быстрее**, чем в строковых БД.
* Пример: Расчёт выручки за год требует чтения только столбцов price и date.

✅ **Экономия места**:

* Сжатие данных сокращает объём хранилища в **5-10 раз** по сравнению с PostgreSQL.

✅ **Масштабируемость**:

* Поддерживают petabytes данных (используются в Facebook, Яндекс.Метрике).

✅ **Оптимизация для колонок**:

* Можно загружать/удалять отдельные столбцы, не трогая всю таблицу.

**3. Минусы**

❌ **Медленные операции записи**:

* Добавление новой строки требует обновления всех столбцов.
* Решение: Пакетная запись (например, раз в минуту).

❌ **Неэффективность для точечных запросов**:

* Запросы типа SELECT \* FROM orders WHERE order\_id = 123 работают медленнее, чем в строковых БД.

❌ **Сложность транзакций**:

* Часто не поддерживают ACID на уровне строк (только для больших блоков данных).

**4. Где применяются?**

* **Аналитика и BI**:
  + ClickHouse в Яндекс.Метрике, Amazon Redshift.
* **Логи и временные ряды**:
  + InfluxDB для хранения метрик серверов.
* **Big Data**:
  + Apache Parquet (колоночный формат для Hadoop).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1. Облачные базы данных (Cloud Databases)**

**Что это?**  
Базы данных, которые работают не на вашем сервере, а в облаке (AWS, Google Cloud, Microsoft Azure и др.). Управляются провайдером — вам не нужно настраивать железо или думать о резервных копиях.

**Примеры**:

* **Amazon RDS** (PostgreSQL, MySQL, Oracle),
* **Google Cloud SQL**,
* **Firebase Realtime Database** (для мобильных приложений),
* **Serverless-решения** (Aurora Serverless, PlanetScale).

**Зачем они нужны?**

* **Масштабируемость**: Автоматически расширяются при росте нагрузки (например, во время распродажи).
* **Геораспределённость**: Данные реплицируются в разные регионы — пользователи из любой точки мира получают быстрый доступ.
* **Экономия**: Платите только за ресурсы, которые используете (не нужно покупать серверы «на вырост»).

**Кейсы**:

* Стартап может развернуть PostgreSQL в облаке за 5 минут и не нанимать админа.
* Игра с миллионами игроков использует Cloud Spanner для глобальной синхронизации данных.

**Метафора**: Облачная БД — как аренда квартиры с обслуживанием. Вам не нужно чинить трубы — этим занимается арендодатель (провайдер).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Векторные базы данных (Vector Databases) — для AI**

**Что это?**  
Специальные БД для хранения и поиска **векторных данных** (числовых массивов). Они нужны для работы с нейросетями, семантическим поиском и генеративным ИИ.

**Как работают?**

* Нейросеть преобразует текст/изображение/звук в **вектор** (например, массив из 512 чисел).
* Векторная БД ищет похожие векторы (например, картинки с котиками среди миллиардов других).

**Примеры**:

* **Pinecone**, **Milvus**, **Weaviate**, **Qdrant**.
* Встроенные решения: **Redis с модулем Vector**, **PostgreSQL с pgvector**.

**Зачем они нужны?**

* **Генеративный ИИ**: Поиск похожего контента для чат-ботов (например, ChatGPT ищет ответы в своей векторной БД).
* **Рекомендации**: «Похожие товары» в интернет-магазинах.
* **Компьютерное зрение**: Поиск лиц на фото или похожих медицинских снимков.

**Метафора**: Векторная БД — как библиотека, где книги ищут не по названию, а «по смыслу». Даже если вы спросите «книгу про грустных космонавтов», она найдёт «Солярис» Лема.