

# **Автоматизация развертывания и эксплуатации программного обеспечения**

## **Раздел «Базы данных»**

Лекция №12

Виноградова Мария Валерьевна

МГТУ им. Н.Э. Баумана, 2022 г.

# Темы лекции

- Основные виды баз данных:
  - реляционные и объектно-реляционные,
  - NoSql и NewSql,
  - комбинированные и многомодельные.
- Индексы.
- Репликация и масштабирование.
- Вопросы администрирования баз данных.
- Работа с облачными базами данных.

# База данных (БД)

- Массив структурированных данных.
- Длительное (постоянное) хранение.
- Работает под управлением СУБД.

## **Система управления базами данных (СУБД)**

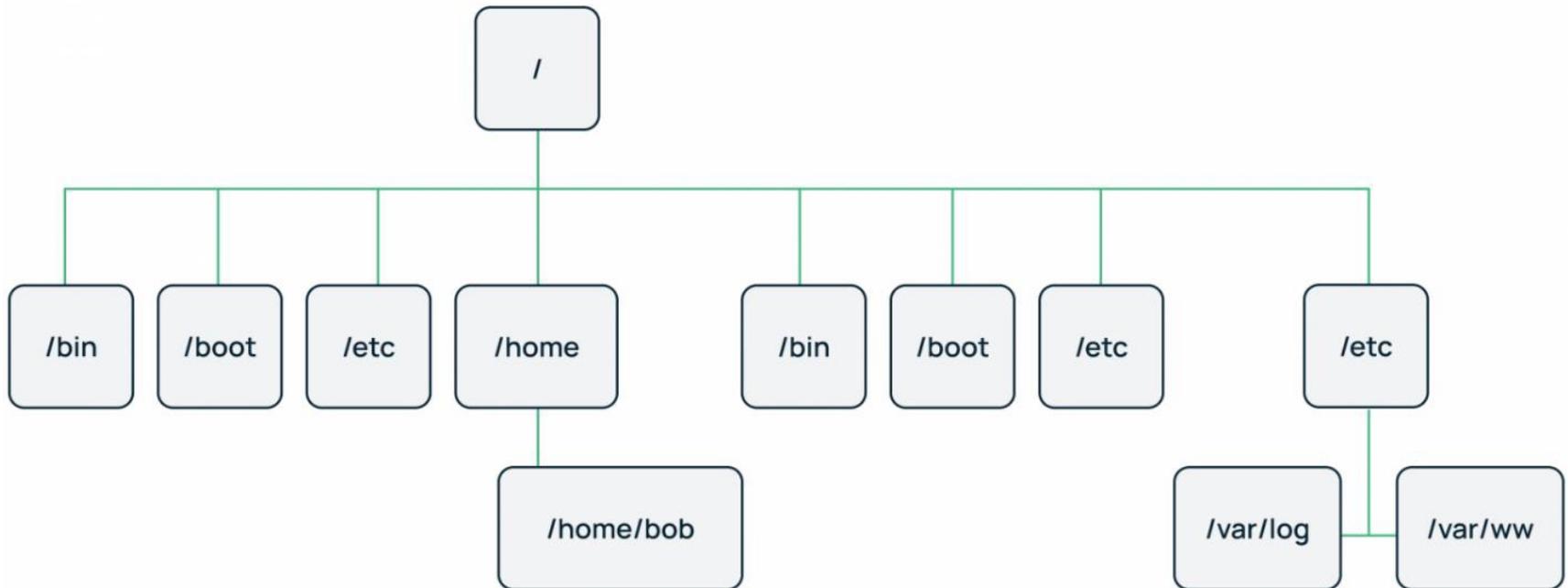
- Длительное хранение больших объемов структурированных данных.
- Поддержка многопользовательской среды.
- Поддержка языка DDL (Data Definition Language) – определение новых баз данных и их схем.
- Поддержка языка DML (Data Manipulation Language) – формирование запросов к данным (CRUD).

# Первое поколение БД и СУБД

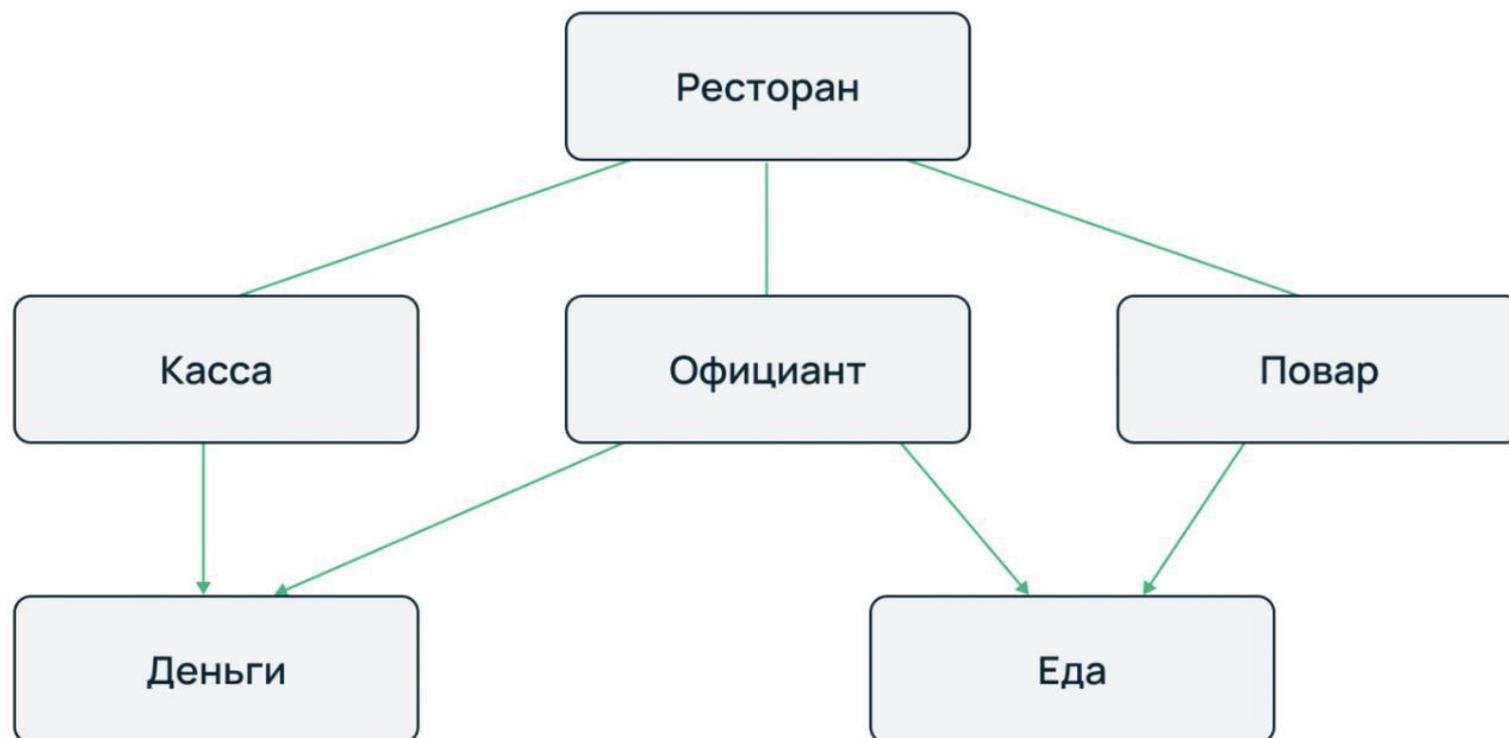
## Первое поколение (конец 60-х годов):

- Коммерческие приложения:
  - Бронирование авиабилетов;
  - Банковская система;
  - Корпоративные системы.
- Файловая система + модель описания структуры.
- Модели данных:
  - «иерархическая» (дерево);
  - «сетевая» (граф) - стандарт CODASYL.
- Недостатки:
  - Отсутствие поддержки высокоуровневых запросов.
  - Навигация по структурам данных.

# Иерархическая модель данных



# Сетевая модель данных



# Реляционные БД и СУБД

## Второе поколение БД (с 70-х годов)

- Реляционная модель данных.
- Отношения (таблицы – англ.**relation**).
- Язык SQL ( Structured Query Language).
- Реляционная алгебра (автор Э.Ф.Кодд).
- Оптимизация запросов.
- Нормализация схемы БД.
- Основные базы данных до 90-х гг.

# Реляционная модель данных

- Особенности:
  - Атомарные атрибуты;
  - Отношения (таблицы);
  - Схема отношения (описание структуры таблицы);
  - Кортеж (строка таблицы);
  - Домен;
  - Схема БД.
- Преимущества:
  - Нормализация / SQL / Оптимизация.

# Отношение - Relation

**Отношение** – основная структура в реляционной модели данных.  
Представляется двумерной таблицей.

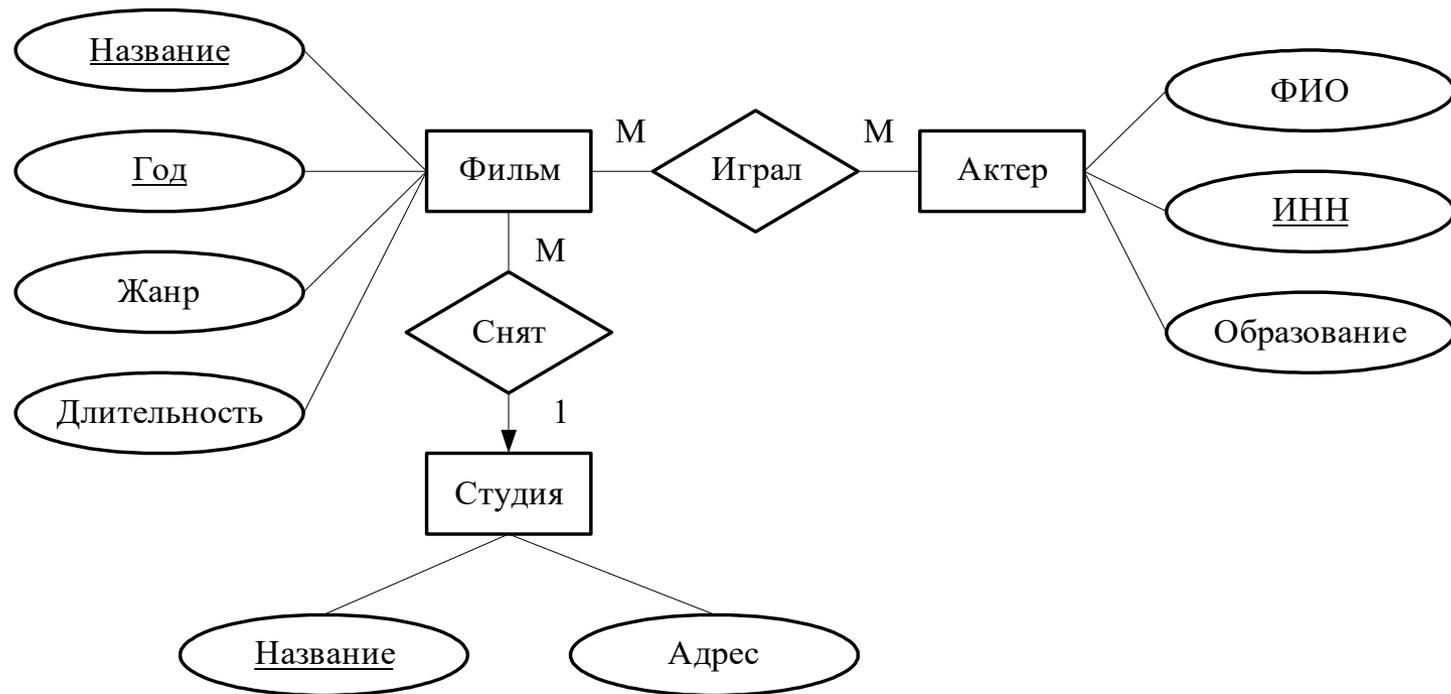
**Отношение** состоит из **кортежей** (строк) и **атрибутов** (столбцов, колонок).  
Пересечение строки и столбца - **поле**.



**Домен** (тип данных) – область допустимых значений, которые могут принимать атрибуты. **Поле** – для хранения значения данных.

# Атомарные типы

- **integer** — целое поле;
- **char** (количество\_символов) — строка из фиксированного количества символов;
- **varchar** (число\_символов) — строка символов, размером не более указанного;
- **smallint** — целое число;
- **float** — действительное число;
- **date** — дата;
- **time** — время.



- Актёр (ИНН, Ф.И.О, Образование)
- Фильм (Название, Год, Длительность, Жанр, Студия)
- Студия (Название, Адрес)
- Актёр–Фильм (Название, Год, ИНН)

# SQL – Structured Query Language

- 1986 – ANSI – SQL/86 (ISO в 1987)
- 1989 - SQL/86 – запросы и схемы
- 1992 - SQL/92 – схемы, транзакции, соединения, авторизация
- 1995 - SQL/CLI – динамический, ODBC
- 1996 - SQL/PSM – хранимые процедуры
- 1999 – SQL:1999 (SQL3) – объектное расширение, UDT
- 2003 – SQL:2003 – OLAP, XML
- 2006 – XQuery

# Основные разделы SQL

- Создание/изменение схемы БД - **DDL**
- Ограничения целостности и триггеры
- Представления БД
- Структуры физического уровня
- Управление доступом
- Транзакции
- Запросы к данным (CRUD) - **DML**

# Операторы языка описания данных SQL

- CREATE TABLE - создание структуры таблицы;
- ALTER TABLE - изменение структуры таблицы;
- DROP TABLE - удаление структуры таблицы;
- CREATE VIEW - создание представления;
- DROP VIEW - удаление представления;
- CREATE INDEX - создание индекса (не входит в стандарт);
- DROP INDEX - удаление индекса;
- GRANT - предоставление привилегий (прав) пользователям;
- REVOKE - удаление привилегий;
- CREATE SCHEMA - задание последовательности операторов
- DROP SCHEMA - удаление схемы.

# **DML - язык манипулирования данными**

- **SELECT** - оператор выборки данных;
- **INSERT** - оператор добавления данных;
- **UPDATE** - оператор обновления данных;
- **DELETE** - оператор удаления данных

# SQL операторы транзакции

- Набор операций над БД, который выполняется атомарно (ACID). Выполняются либо все операции транзакции, либо ни одна из них.

**begin transaction** [ Имя ]

- начать транзакцию

**Commit transaction**

- Зафиксировать транзакцию

**Rollback transaction**

- откатить транзакцию

**Save transaction** Имя\_точки

- Создать точку сохранения

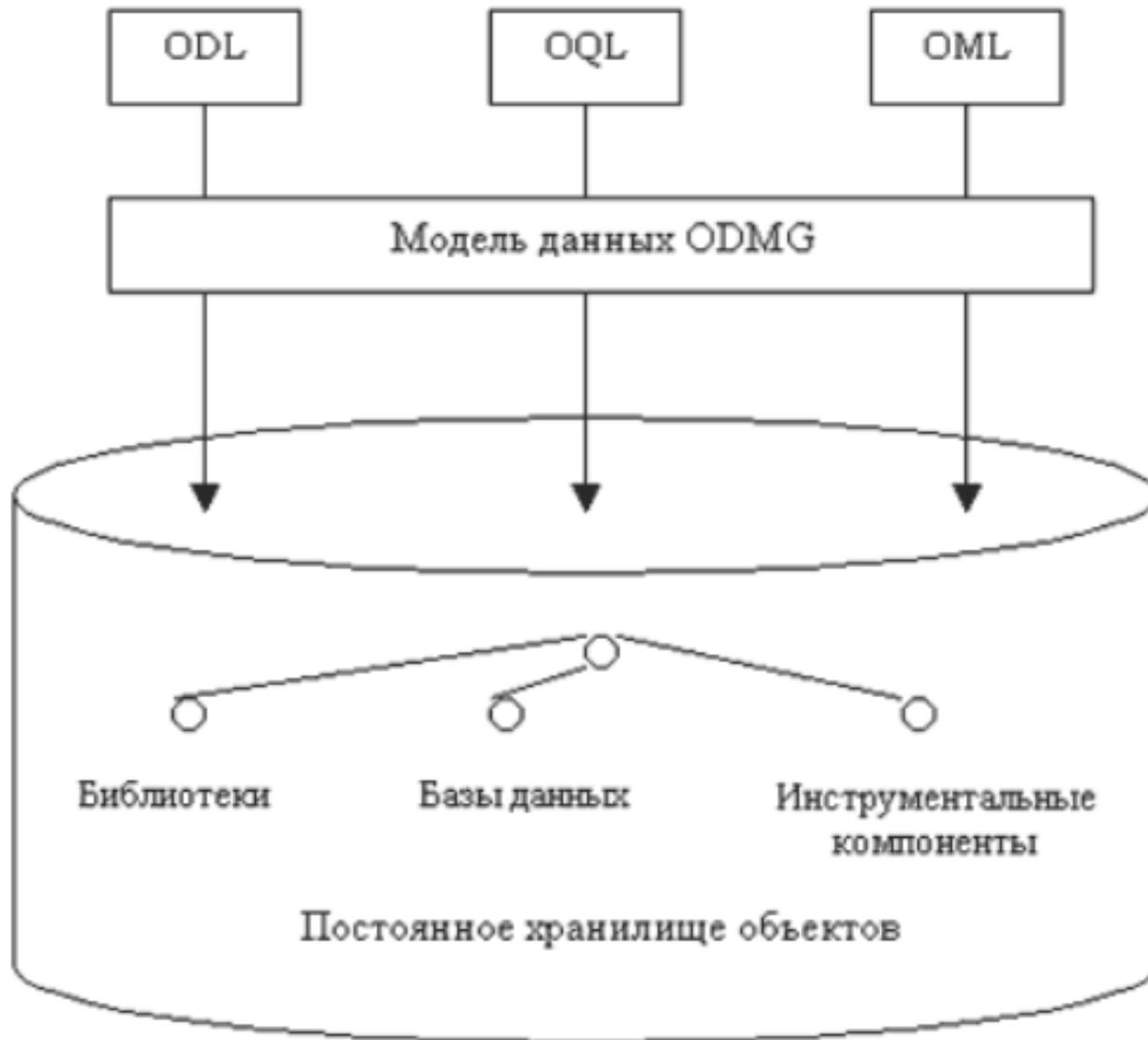
# Объектные и объектно-реляционные БД

- Причина появления
  - Расширение структур данных
  - Сложность сочетания программы и РБД
- Объектные БД
  - ОО язык программирования высокого уровня
  - Хранение объектов в БД
- Объектно-реляционные БД
  - Надстройка на реляционной БД
  - Сложные типы данных
  - Программирование на стороне сервера

# Стандарты объектных баз данных ODMG

- 1989 - консорциумом OMG (*Object Management Group*)
- 1991 - консорциум ODMG (*Object Database Management Group*, позднее *Object Data Management Group*)
- Выботка стандартов для ООБД на основе объектной модели OMG COM (*Core Object Model*) .
- ODMG-93 и ODMG-2003 – наиболее известные стандарты.
- После 2001 стандартом занимается OMG.

# Архитектура ODMG



# Языки стандарта ODMG

- ODL - язык определения данных
  - Виртуальный
  - Задаёт возможности
  - Используются ООЯП для СУБД
- OQL - язык объектных запросов
  - Основан на SQL
  - Встраивается в ОО ЯПВУ
- OML - язык манипулирования объектами
  - Интегрирован или расширяет ОО ЯПВУ

# Особенности объектно-реляционных БД

- Объектно-реляционная модель данных
- Расширение языка SQL
- Обратная совместимость с РБД

# Объектно-реляционная модель позволяет

- хранить и обрабатывать сложные типы данных (структуры и коллекции);
- использовать пользовательские типы данных (UDT);
- включать в пользовательские типы данных атрибуты и методы;
- поддерживать уникальные идентификаторы кортежей и ссылки на них;
- определять иерархию (наследование) типов данных.

# Типы данных ORM

- атомарные;
- структуры, содержащие набор именованных полей;
- массивы, хранящие упорядоченный набор элементов;
- множества и мультимножества, хранящие неупорядоченный набор элементов;
- пользовательские типы данных;
- ссылки на объекты (кортежи) пользовательских типов данных.

Элементами структур и коллекций могут быть любые типы данных, в том числе составные.

# Пример встроенных типов в PostgreSQL

- Числовые, Денежные, Символьные, Двоичные, Типы даты/времени, Логический
- Типы перечислений
- Геометрические типы (Точки, Прямые, Отрезки, Прямоугольники, Пути, Многоугольники, Окружности )
- Типы, описывающие сетевые адреса
- Битовые строки
- Типы, предназначенные для текстового поиска
- Тип UUID
- Тип XML
- Типы JSON
- Массивы
- Составные типы (структуры)
- Диапазонные типы
- Типы доменов
- Идентификаторы объектов (Oid)

# Расширение языка SQL для ОРБД

- Поддержка сложных типов данных
- Встроенные (системные) функции
- Хранимые процедуры и функции (PSM)
- Общие табличные выражения
- Рекурсия
- Ранжирование
- Транспонирование
- Триггеры DDL/DML
- Секционирование

# Хранимые процедуры и функции

- PERSISTENT STORED MODULES
- SQL/PSM, PSM-96
- Объекты схемы БД
- CREATE – ALTER – DROP
- Хранятся (в откомпилированном виде) и исполняются на сервере

# Пример процедуры

```
CREATE PROCEDURE triple(INOUT x int)
LANGUAGE plpgsql
AS $$
    BEGIN
        x := x * 3;
    END;
$$;
```

```
DECLARE myvar int := 5;
CALL triple(myvar);
```

# Возможности PSM

- Объявление и использование переменных
- Возврат значений
- Условия
- Циклы
- Работа с курсором
- Перехват, обработка и вызов исключений
- Вызов системных функций
- DML и DDL команды языка SQL
- Динамические запросы

```
DECLARE MovieCursor CURSOR FOR SELECT length
FROM Movie WHERE studioName = s;
DECLARE newLen INTEGER;
DECLARE movieCount INTEGER;
BEGIN
    SET movieCount = 0;
    OPEN MovieCursor;
ml: LOOP FETCH MovieCursor INTO newLen;
    IF Not_Found THEN LEAVE ml
    END IF;
    SET movieCount = movieCount + 1;
    SET mean = mean + newLen;
    SET variance = variance + newLen * newLen;
END LOOP;
SET mean = mean / movieCount;
SET variance = variance / movieCount - mean * mean;
```

# Типы пользовательских функций

- **Скалярные** – возвращает одно значение
- **Табличные** – возвращают набор записей
- **Встроенные (inline)** – как представление с параметром
- **Агрегатные** – применяют к набору значений. Функции инициализации, итерации, и итогового вычисления.

# Пример скалярной функции

```
CREATE FUNCTION add_xy(x integer, y integer)
  RETURNS integer AS
  $$
    SELECT x + y;
  $$
LANGUAGE SQL;
```

Вызов функции:

```
SELECT add_xy(1, 2);
```

# Задачи на построение рекурсивных запросов

- Обход дерева или иерархии,
- Нахождение путей и их параметров
- Проверка достижимости

<b>cfrom</b> character v	<b>cto</b> character v	<b>rtype</b> character v	<b>rcost</b> integer	<b>rtime</b> integer
MOS	SPb	RGD	4000	4
MOS	ANAPA	avia	3000	3
ANAPA	Stambul	avia	5000	1
SPb	Riga	avia	1000	1
Riga	London	avia	7000	3
London	NewYork	sea	15000	30
London	Paris	avia	2000	2

Таблица дорог

Roads(cfrom, cto,  
rtype, rcost, rlen)

# Рекурсивные запросы

```
WITH Recursive Path as      -- вычисляемая таб.  
(  
  select cfrom,cto          -- база вычислений  
  from Roads  
Union  
  select Path.cfrom, Roads.cto  -- индукция  
  from Path, Roads  
  where Path.cto = Roads.cfrom  
)  
select * from Path          -- итоговый запрос
```

# Пример рекурсивного запроса

```
WITH Recursive
  Path(cfrom,cto,rcost,rnum) as
(
  select cfrom,cto,rcost,
    1 as rnum
    from Roads
union
  select Path.cfrom,
    Roads.cto,
    Path.rcost + Roads.rcost,
    Path.rnum + 1
    from Path, Roads
  where Path.cto = Roads.cfrom
)
select * from Path
where rcost<20000
```

<b>cfrom</b> character	<b>cto</b> character var	<b>rcost</b> integer	<b>rnum</b> integer
MOS	SPb	4000	1
MOS	ANAPA	3000	1
ANAPA	Stambul	5000	1
SPb	Riga	1000	1
Riga	London	7000	1
London	NewYork	15000	1
London	Paris	2000	1
MOS	Riga	5000	2
MOS	Stambul	8000	2
SPb	London	8000	2
Riga	Paris	9000	2
MOS	London	12000	3
SPb	Paris	10000	3
MOS	Paris	14000	4

# Пример ранжирования (оконные функции)

```
| select ROW_NUMBER() over (order by src),  
       rank() over (order by src),  
       dense_rank() over (order by src),  
       ntile(3) over (order by src),  
       src  
- from roads
```

1	1	1	1	анапа
2	2	2	1	лондон
3	3	3	1	москва
4	3	3	2	москва
5	5	4	2	париж
6	6	5	3	СПб
7	6	5	3	СПб

# Триггеры DML

**CREATE TRIGGER** Название  
**AFTER UPDATE OF** Таблица **ON** Поле  
**REFERENCING**

**OLD ROW AS** oldTuple,  
    **NEW ROW AS** newTuple

**FOR EACH ROW**

**WHEN** (условие)

**BEGIN**

    код

**END**

# DDL триггеры

- Уровня БД / уровня сервера
- Изменения структуры объектов БД
- Системные переменные (структуры) для получения информации о событии
- Возможность отката транзакции

# Секционирование

- Разбиение таблицы на разделы (секции) по некоторому условию.
- Вертикальное – разбиение по столбцам.
- Горизонтальное – разбиение по строкам.

# Особенности баз данных NoSql

- Семинар 2009 г., Сан-Франциско
- Нет определенного термина
- Не используют SQL
- Открытый исходный код (не всегда)
- Работа на кластерах (есть графовые)
- БД после 2000 г.
- Учитывают объемы веб
- Без схемы данных (гибкая структура)

# Категории баз данных NoSql

- **БД «ключ-значение»** — содержит ключи и значения.  
Примеры: Riak, Amazon DynamoDB;
- **Документно-ориентированная БД** — хранит документы, состоящие из тегированных элементов (JSON).  
Пример: CouchDB;
- **Колоночная БД** — хранение данных по столбцам, произвольные наборы столбцов для разных строк  
Примеры: HBase, Cassandra;
- **Графовая БД** — сетевая база данных, которая использует узлы и рёбра для отображения и хранения данных.  
Пример: Neo4J.

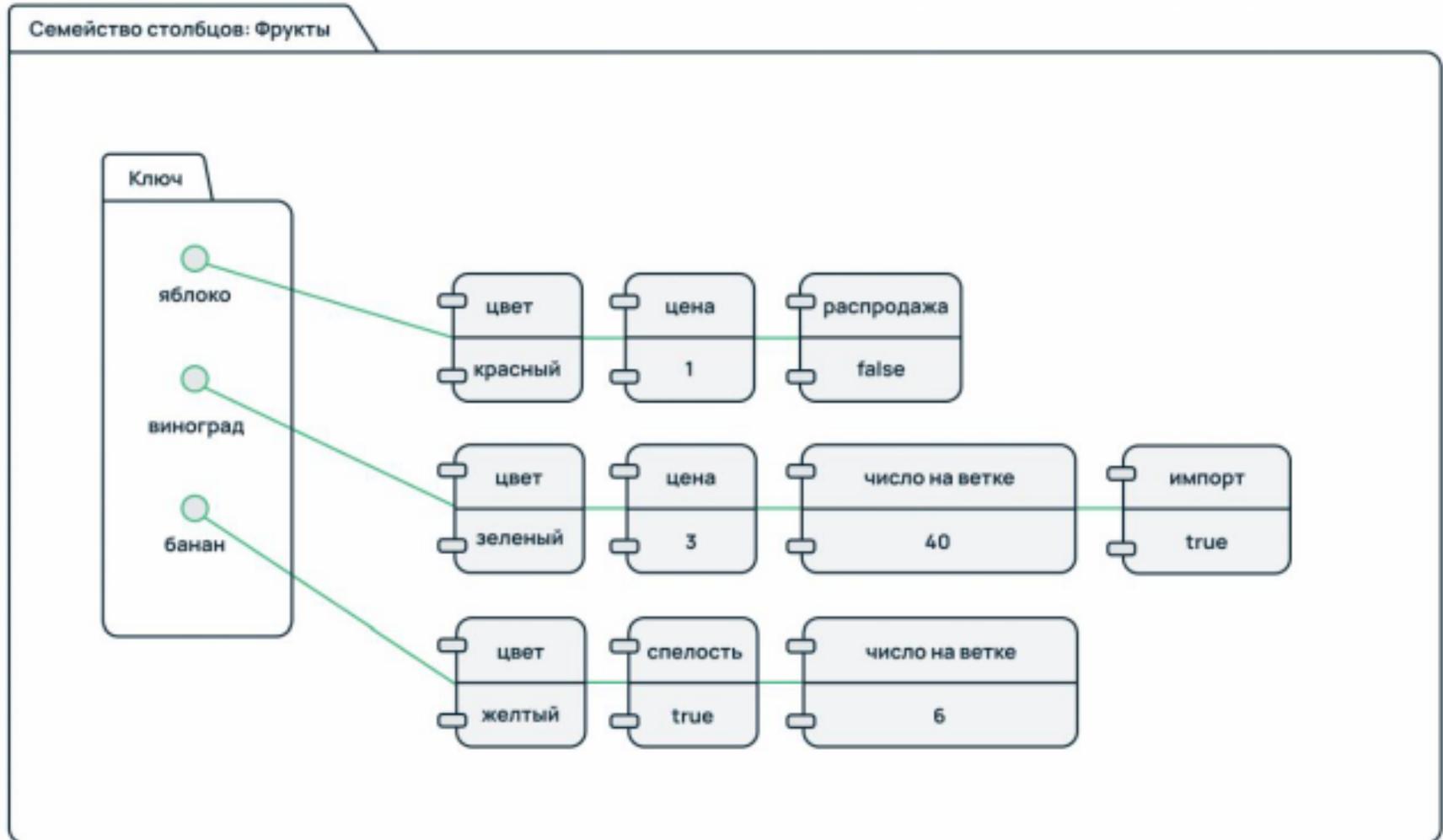
# База данных «ключ-значение»

Key	Value
"India"	{"B-25, Sector-58, Noida, India – 201301}"
"Romania"	{"IMPS Moara Business Center, Buftea No. 1, Cluj-Napoca, 400606", City Business Center, Coriolan Brediceanu No. 10, Building B, Timisoara, 300011}"
"US"	{"3975 Fair Ridge Drive. Suite 200 South, Fairfax, VA 22033}"

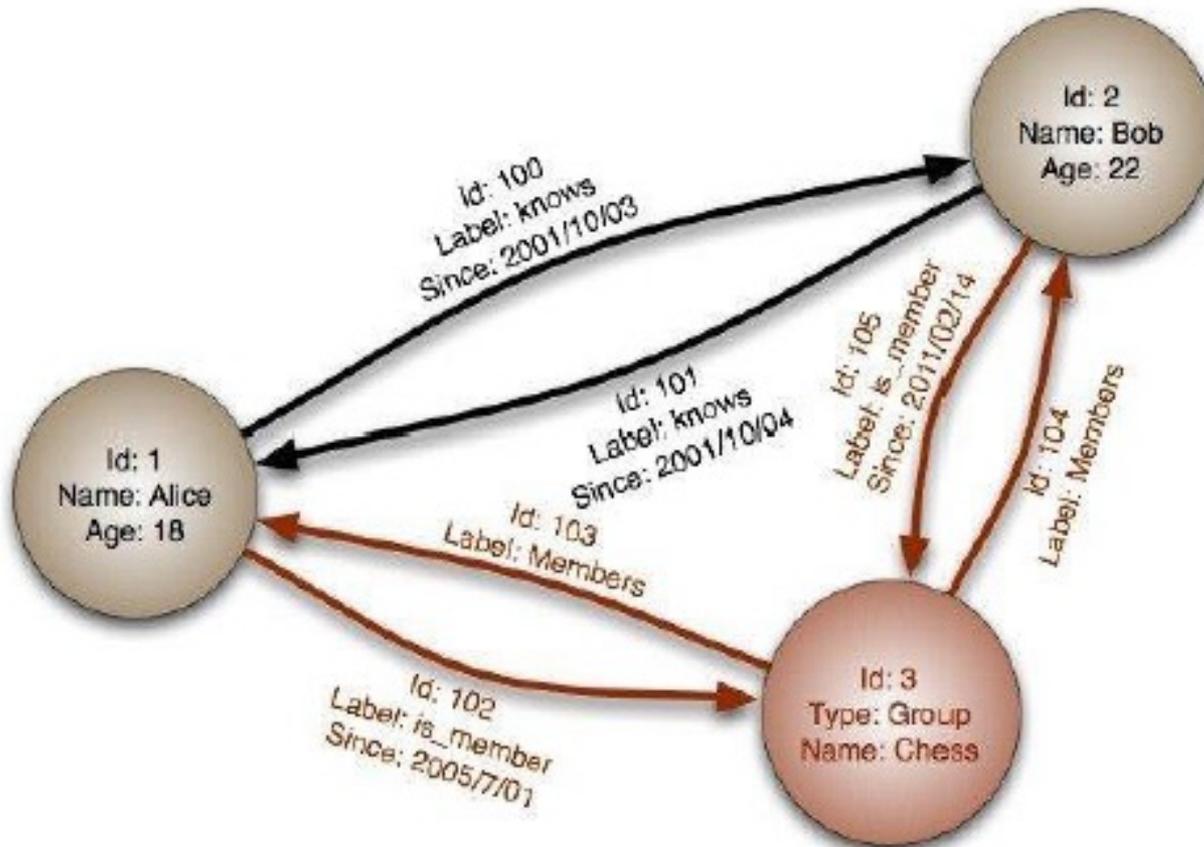
# Документно-ориентированная БД

```
{officeName: "3Pillar Noida",  
  {Street: "B-25", City: "Noida",  
    State: "UP", Pincode: "201301"}  
}  
{officeName: "3Pillar Timisoara",  
  {Boulevard: "Coriolan Brediceanu No. 10",  
    Block: "B, 1st Floor",  
    City: "Timisoara", Pincode: "300011"}  
}  
{officeName: "3Pillar Cluj",  
  {Latitude: "40.748328",  
    Longitude: "-73.985560"}  
}
```

# Колоночная БД



# Графовая БД



# Комбинированные базы

- Эта разновидность баз совмещает в себе SQL- и NoSQL-подходы к организации хранения и обработки данных.
- Этот класс баз включает в себя NewSQL и многомодельные решения.

# Базы данных NewSQL

- Особенности:
  - Возможности масштабирования,
  - Поддержка ACID-транзакций и согласованности данных.
- Термин предложил в 2011 году аналитик компании 451 Group Мэтью Аслет
  - Он отмечал высокую потребность в таких системах для сфер, работающих с критическими данными — здравоохранение, финансы, обработка заказов и т.д.

# Особенности БД NewSQL

- Примеры БД NewSQL
  - **MemSQL, VoltDB, NuoDB** – реляционная модель.
  - **FoundationalDB** – ключ-значение и поверх SQL.
- Архитектура и возможности
  - In-memory (БД хранится в ОП).
  - Кластеры из множества узлов.
  - Журнал операций (запись снимков данных на диск).
  - Реляционная модель и поддержка SQL.
  - Одноранговая и ведущий-ведомый репликации.
  - Транзакции и блокировки записей.
  - Фрагментация таблиц.

# Многомодельные базы данных

- Такие БД сочетают в себе несколько подходов к организации данных одновременно.
- Это обеспечивает функциональное разнообразие при разработке систем с их использованием.
- **Примеры:** ArangoDB.
- **Особенности:**
- возможность в одном запросе работать с данными, хранящимися в разных типах баз, не нарушая при этом согласованности;
- обширные возможности масштабирования за счет легкой интеграции новых моделей баз данных в существующую инфраструктуру проекта.

# Базы данных временных рядов

- Данный тип БД можно использовать при необходимости отслеживания исторической динамики по ряду показателей.
- Данные группируются по временным меткам.
- Базы временных рядов чаще ориентированы на запись, чем на построение сложных аналитических запросов.
- **Примеры:** OpenTSDB, Prometheus, InfluxDB, TimescaleDB.
- **Особенности:** можно обрабатывать постоянный поток входных данных.
- **Ограничения:** производительность зависит от объема поступающей информации, количества отслеживаемых метрик, а также временного лага между записью новых данных и запросами на чтение.

# Индексы

Индексы - это особые таблицы, используемые поисковыми системами для поиска данных.

Их активное использование играет важнейшую роль в повышении производительности sql серверов.

```
CREATE INDEX Назв. ON Табл(поля);
```

```
CREATE INDEX YearIndex1 ON Movie(year);
```

```
CREATE UNIQUE INDEX YearIndex2 ON Movie(year);
```

```
CREATE INDEX KeyIndex3 ON Movie(title, year);
```

```
DROP INDEX YearIndex1;
```

# Кластерный индекс

- **Кластерным** называется особый индекс, который использует первичный ключ для структуризации данных в таблице.
- Он не требует явного объявления и создается по умолчанию при определении ключа.
- Отсортированный в порядке возрастания первичный ключ по умолчанию применяется в качестве кластеризованного индекса.

product\_pkey

product_id
1
2
3
4



product

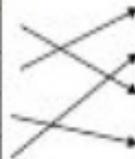
product_id	product_name	product_subcategory	brand	category	price
1	A	headphone	Sony	electronics	\$280
2	B	sneaker	Nike	shoes	\$70
3	C	shirt	Levi's	clothing	\$50
4	D	baseball bat	Louisville Slugger	sports	\$100

# Некластерный индекс

- **Некластерный индекс** хранится в одном месте, а физические данные таблицы — в другом.
- Благодаря этой особенности для каждой таблицы можно создавать более одного некластерного индекса.

product\_category\_index

category	product_id
clothing	3
electronics	1
sports	4
shoes	2



product

product_id	product_name	product_subcategory	brand	category	price
1	A	headphone	Sony	electronics	\$280
2	B	sneaker	Nike	shoes	\$70
3	C	shirt	Levi's	clothing	\$50
4	D	baseball bat	Louisville Slugger	sports	\$100

# Масштабируемость БД

- **Масштабируемость базы данных** —это способность базы данных обрабатывать изменяющиеся требования путем добавления и удаления ресурсов.
- Уменьшение компьютеров:
  - Майнфрейм/ миникомпьютер/ ПК/ мобильные устройства.
- Увеличение объемов данных и нагрузки.
- Снижение уровня блокировки:
  - Таблицы/ дисковые блоки/ записи.
- Увеличение возможностей оборудования:
  - Однопроцессорные/ многопроцессорные/ многоядерные.

# Аспекты масштабирования

- Объем данных.
- Объем запросов.
- Размер запросов.

OLAP – мало запросов к объемным данным,  
OLTP – много мелких запросов.

**Эластичность** - способность системы прозрачно добавлять и уменьшать емкость для удовлетворения меняющихся рабочих нагрузок.

# Типы масштабирования БД

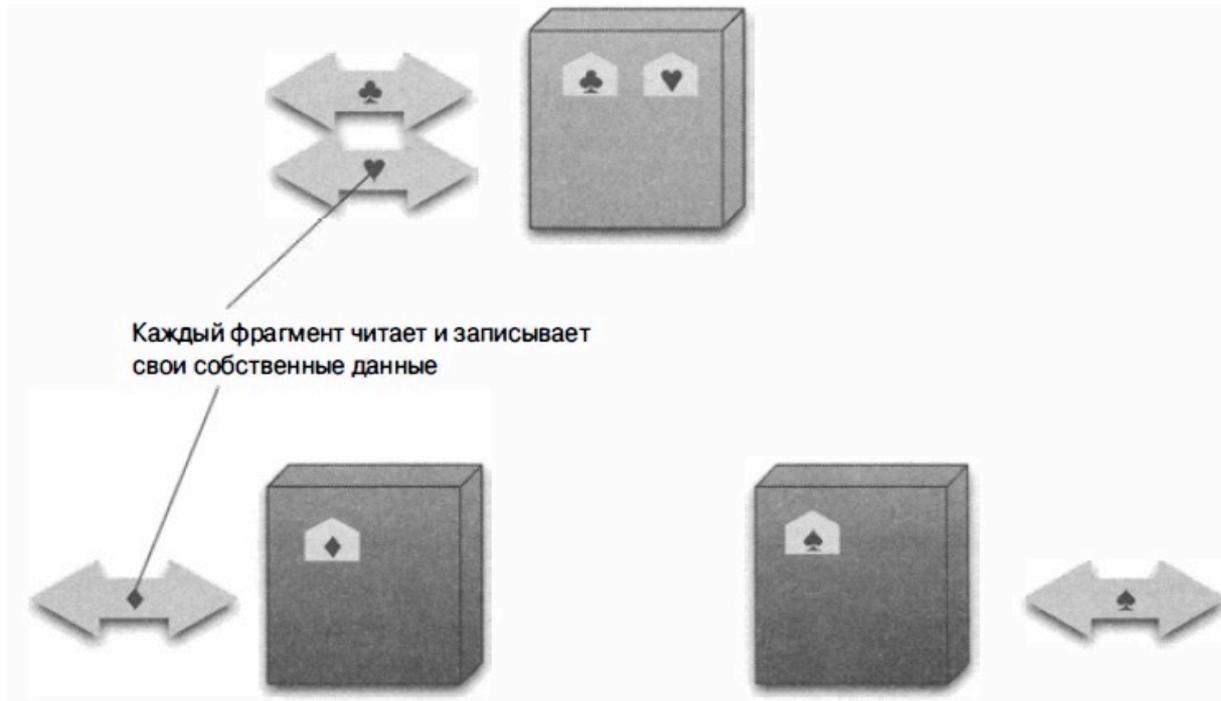
- **Вертикальное масштабирование**
  - Увеличение возможностей аппаратного обеспечения сервера.
  - Дорого.
  - Физически ограничено.
- **Горизонтальное масштабирование**
  - Создание кластеров из обычных компьютеров.
  - Дешевле.
  - Усложняется процесс управления данными и выполнение запросов.

# Тиражирование данных

- Двухфазная фиксация транзакций (синхронная репликация).
- Репликация (кластеры NoSQL):
  - Одноранговая,
  - Ведущий-ведомый.
- Отказ от строгих (ACID) транзакций.

# Фрагментация

- Фрагментация – хранение разных наборов данных на разных узлах.



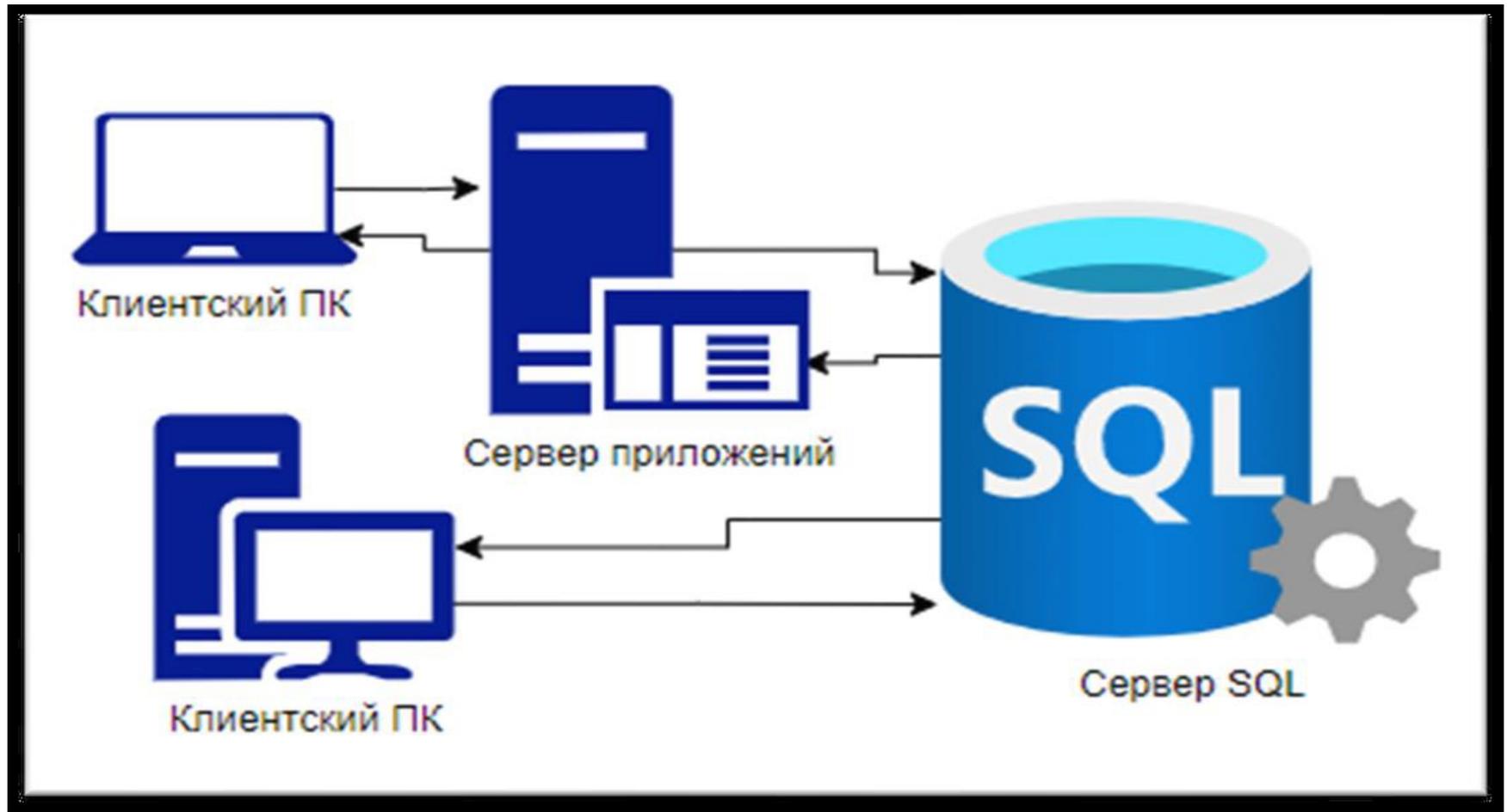
# Цели фрагментации

- Балансировка рабочей нагрузки
- Совместное использование данных
- Соответствие логике приложения
- Территориальная близость
- Автоматизация
- Повышает эффективность записи агрегатов (горизонтальное масштабирование)

# Администрирование баз данных

- Установка сервера
  - из двоичных пакетов,
  - из исходного кода.
- Настройка сервера.
- Подготовка к работе и запуск.
- Сопровождение и обслуживание.

# Клиент серверная работа PostgreSQL



# Алгоритм работы с сервером

- клиент подключается к серверу, а сервер выполняет его аутентификацию;
- клиент формирует запросы, а сервер их выполняет и возвращает результаты;
- дополнительно клиент управляет транзакциями, а сервер обеспечивает их поддержку.

# Выполнение запроса

- Разбор запроса
  - синтаксис
  - права доступа
  - системные каталоги
- Трансформация запроса
  - Подстановка: представление в запрос
  - более низкоуровневый запрос.
- Планирование
  - Выполняет планировщик, основываясь на статистике.
  - Статистика: сколько у нас таблиц, сколько в них строчек, сколько они занимают страниц, как распределены данные в столбцах и тому подобное.
  - Планировщик решает с каких таблиц начать выполнение запроса, какие условия и в каком порядке применять
  - Планировщик подготавливает план выполнения.
- Выполнение запроса (и возврат ответа клиенту)

# Процессы и память PostgreSQL

- **Основной процесс.** Он слушает назначенный порт и по необходимости запускает другие процессы: фоновые или обслуживающие. Когда клиентское приложение хочет подключиться, то основной процесс запускает обслуживающий процесс и подключает приложение к нему.
- **Обслуживающие процессы.** Работает с клиентом (отвечает за обработку его запросов). У этого процесса есть локальная память для хранения подготовленных операторов (разобранных запросов), курсоров и другого. Сколько клиентов подключаются к серверу, столько и обслуживающих процессов создается.
- **Фоновые процессы.** Выполняют различные служебные операции, например, очищают базу от уже неактуальных данных.
- **Память**, выделяемая для работы тоже бывает разных типов:
- **Локальная память процесса.** Необходима обслуживающему процессу для хранения подготовленных операторов, курсоров, различных переменных окружения и тому подобное.
- **Общая память.** В ней хранятся фактические данные, то есть таблицы базы данных. Все процессы взаимодействуют с базой данных и друг с другом через эту память.

# Параметры сервера БД

- Расположения файлов
- Подключения и аутентификация
- Потребление ресурсов
- Журнал предзаписи
- Репликация
- Планирование запросов
- Регистрация ошибок и протоколирование работы сервера
- Статистика времени выполнения
- Автоматическая очистка
- Параметры клиентских сеансов по умолчанию
- Управление блокировками
- Обработка ошибок
- Предопределённые параметры (для чтения)

# Настройка параметров БД

- в файле конфигурации (postgresql.conf)  
log\_connections = yes  
shared\_buffers = 128MB
- через SQL  
set\_config(setting\_name, new\_value, is\_local)
- в командной строке  
postgres -c log\_connections=yes -c  
log\_destination='syslog'

# Методы аутентификации PostgreSQL

- **trust**, при которой сервер доверяет пользователям, не проверяя их.
- **password**, требующая ввода пароля пользователем.
- **GSSAPI**, использующая библиотеку безопасности, совместимую с GSSAPI. (Kerberos или Microsoft Active Directory).
- **SSPI**, использующая протокол, подобный GSSAPI, но для Windows.
- **ident**, для которой используется служба, реализующая «Identification Protocol» (RFC 1413) на клиентском компьютере.
- **peer**, которая полагается на средства операционной системы, позволяющие узнать пользователя процесса на другой стороне локального подключения.
- **LDAP**, работающая с сервером аутентификации LDAP.
- **по сертификату**, требующая использования клиентами SSL-подключения и построенная на проверке передаваемых ими сертификатов SSL.
- **PAM**, реализуемая с использованием библиотеки PAM.

# Подготовка к работе

- Создание кластера.
- Настройка параметров репликации.
- Создание и конфигурация баз данных.
- Создание учетных записей пользователей.
- Определение ролей.
- Назначение прав доступа к объектам БД.
- Локализация (языки, кодировки, сортировки).

# Задачи регулярного обслуживания БД

- Регламентная очистка
- Регулярная переиндексация (REINDEX)
- Обслуживание журнала
  - Сохранение копий
  - Ротация журнальных файлов
- Резервное копирование и восстановление
- Мониторинг работы СУБД

# Статистика работы PostgreSQL

- Собирает статистику фоновый процесс “stats collector” (коллектор статистики).
- Статистика включается конфигурационными параметрами в файле **postgresql.conf**.
- Каждый **backend** процесс собирает статистику. Затем эта статистика отправляется процессу **stats collector**, который собирает статистику со всех **backend** процессов.
- Раз в полсекунды, статистика сбрасывается в каталог **\$PGDATA/pg\_stat\_tmp**.
- При остановке сервера PostgreSQL, статистика сбрасывается в другой каталог – **\$PGDATA/pg\_stat**.
- Статистика ведется с момента первого запуска сервера, а с помощью функции **pg\_stat\_reset()** её можно сбросить. Это обнулит не все счетчики, а только в текущей базе данных.
- На уровне всего кластера обнулить счетчики можно с помощью функции **pg\_stat\_reset\_shared ()**.

# Просмотр статистики через системные представления

- **pg\_stat\_all\_tables** – в разрезе строк и страниц для определённой базы данных;
- **pg\_statio\_all\_tables** – в разрезе 8 KB страниц для определённой базы данных;
- **pg\_stat\_all\_indexes** – по индексам для определённой базы в разрезе строк;
- **pg\_statio\_all\_indexes** – по индексам для определённой базы в разрезе страниц;
- **pg\_stat\_database** – глобальная статистика по определённой базе данных;
- **pg\_stat\_bgwriter** – статистика для анализа фоновой записи.

# Регламентная очистка

- Высвобождение дискового пространства
  - Команда **VACUUM** удаляет старые версии строк в таблицах и индексах и помечает пространство свободным.
  - Команда **VACUUM FULL**, сжимает таблицы, записывая новую версию файла таблицы.
- Обновление статистики планировщика
  - статистика о содержимом таблиц собирается командой **ANALYZE**
- Обновление карты видимости
  - Страницы с кортежами для активных транзакций
- Предотвращение ошибок из-за зацикливания счётчика транзакций
- Демон автоочистки
  - автоматическое выполнение **VACUUM** и **ANALYZE**

# Резервное копирование

- **Резервное копирование и восстановление**
  - это создание и хранение копий данных, которые можно использовать для восстановления служб организации в случае сбоя первичных данных из-за отключения электричества, атаки программ-вымогателей и прочих нештатных ситуаций.
  - Резервные копии позволяют восстановить состояние систем на более ранний момент, предшествующий потере или повреждению данных, для восстановления служб.
- **Резервные копии и снимки**
  - Разница между этими двумя терминами заключается в деталях.
  - Под резервной копией обычно понимается полная копия всех данных и файлов в системе.
  - Снимок копирует состояние системы в определенной точке во времени.

# Способы резервного копирования

- **Полное** (создается резервная копия всех данных в качестве точки восстановления, к которой можно легко откатить систему)
- **Инкрементное** (выполняется резервное копирование данных «по возрастанию». Процесс начинается с одного полного резервного копирования, а затем создаются резервные копии только тех данных, которые были изменены с момента последнего резервного копирования)
- **Дифференциальное** (начинается с полного резервного копирования, а затем выполняется резервное копирование только измененных данных. При использовании этого метода восстановление обычно происходит быстрее. Требуется два файла: последняя резервная копия полного образа и последняя дифференциальная резервная копия).

# Оптимизация производительности

- Инструмент профилирования запросов - EXPLAIN
- Планирует и выполняет запрос при подсчете строк и измерении времени в различных точках плана выполнения.

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 7000;
```

---

```
Seq Scan on tenk1 (cost=0.00..483.00 rows=7001  
width=244) Filter: (unique1 < 7000)
```

# Облачные базы данных

## Облачная база данных (CloudDatabase)

- это служба базы данных, созданная и доступная через облачную платформу;
- выполняет многие из тех же функций, что и традиционная база данных;
- с дополнительной гибкостью облачных вычислений.

## Ключевые особенности облачных баз данных:

- Создается и доступна через облачную платформу.
- Позволяет размещать БД без покупки специального оборудования.
- Может управляться пользователем или предлагаться как услуга и управляться поставщиком.
- Может поддерживать реляционные базы данных (включая MySQL и PostgreSQL) и базы данных NoSQL (включая MongoDB и CouchDB).
- Доступ через веб-интерфейс или API, предоставляемый поставщиком.

# Модели развертывания облачной базы

- **Традиционная модель**
  - Похожа на локальную базу данных, управляемую внутри компании, за исключением предоставления инфраструктуры.
  - Организация приобретает пространство виртуальной машины у поставщика облачных услуг, а база данных развертывается в облаке.
  - Разработчики организации используют модель DevOps или традиционный ИТ-персонал для управления базой данных.
  - Организация отвечает за управление базой данных.
- **База данных как услуга (DBaaS)**
  - Организация заключает договор с поставщиком облачных услуг через платную услугу подписки.
  - Поставщик услуг предлагает конечному пользователю ряд задач по обслуживанию, администрированию и управлению БД в реальном времени.
  - База данных работает в инфраструктуре поставщика услуг.
  - Включает автоматизацию в областях предоставления доступа, резервного копирования, масштабирования, высокой доступности, безопасности, установки исправлений и мониторинга работоспособности.

# Варианты управления облачной базой данных

- **Самоуправляемые облачные базы данных (Self-managed)**
  - Компания разворачивает базу данных в облачной инфраструктуре, однако управляет ею силами собственных специалистов и собственными средствами автоматизации.
- **Автоматизированные облачные базы данных**
  - Компании используют API-интерфейсы облачного сервиса БД, чтобы выполнять задачи, связанные с жизненным циклом обслуживания.
  - У компаний остается доступ к серверам базы данных и они могут контролировать ее конфигурацию.
  - Доступ ограничен, например, отсутствует установка исправлений и обслуживание.

# Варианты управления облачной базой данных

- **Управляемые облачные базы данных**
  - Похожа на автоматизированные облачные БД, однако облачный провайдер не предоставляет потребителю доступ к серверам БД.
  - Конфигурация определяется поставщиком облака.
  - Конечным пользователям не разрешается устанавливать собственное программное обеспечение.
- **Автономные облачные базы данных**
  - Полностью автономная операционная модель, в которой средства автоматизации и машинное обучение устраняют ручное управление базами данных и настройку производительности.

# Преимущества облачной БД

- **Повышенная маневренность и инновации.** Можно очень быстро настроить и так же быстро списать и перейти к следующему проекту.
- **Более быстрое время выхода на рынок.** Нет необходимости заказывать оборудование или тратить время на ожидание доставки, установки и настройки сети. Доступ к базе данных может быть получен в течение нескольких минут.
- **Сниженные риски.** Поставщики облачных услуг предоставляют автоматизацию для обеспечения безопасности и высокой доступности, а также набор сервисов по обслуживанию БД.
- **Снижение затрат.**
  - Модели подписки с оплатой по мере использования и динамическое масштабирование позволяют конечным пользователям обеспечивать стабильное состояние, затем увеличивать масштаб для пиковых нагрузок в периоды занятости и затем уменьшать масштаб при уменьшении спроса.
  - Отключение услуг, когда они не нужны.
  - Сокращение операционные расходы при автоматизации процессов обслуживания БД.

# Спасибо за внимание!

Виноградова Мария Валерьевна

[Vinogradova.m@bmstu.ru](mailto:Vinogradova.m@bmstu.ru)

МГТУ им. Н.Э. Баумана  
кафедра Систем обработки информации и  
управления (ИУ5)