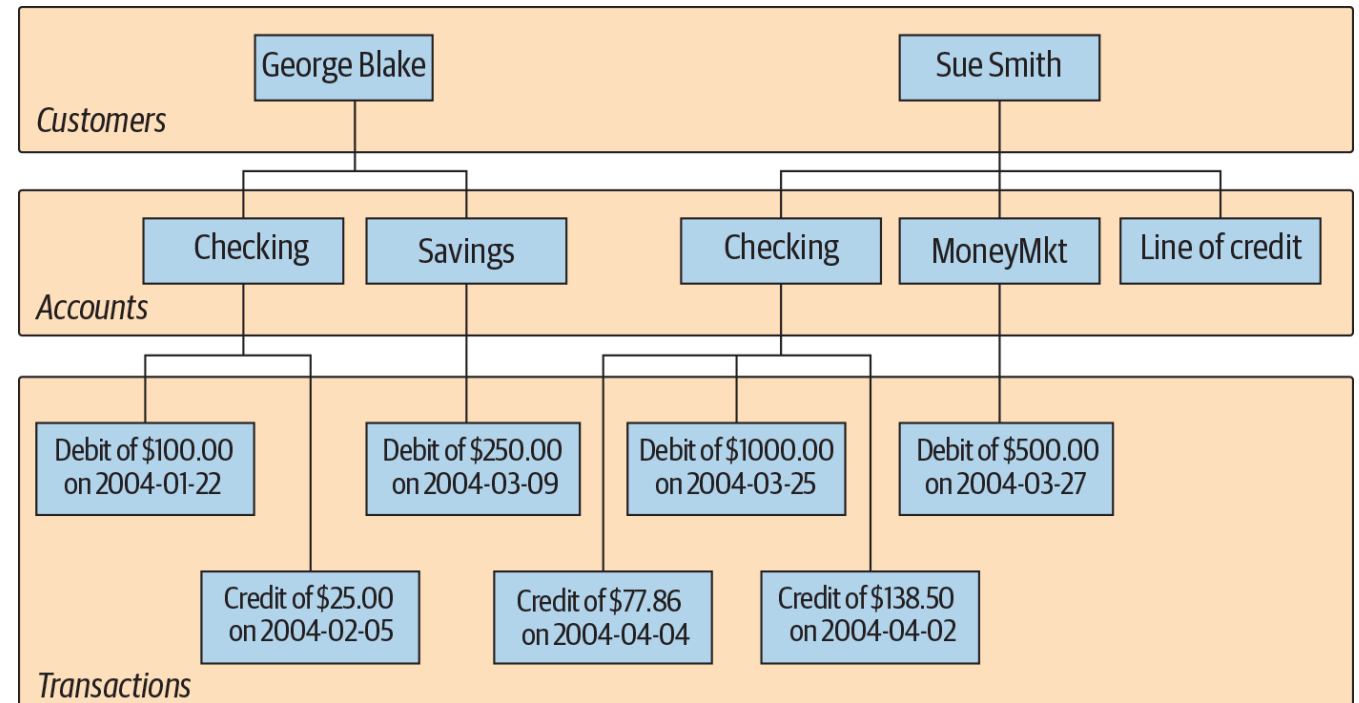


Сервисы с состоянием

Системы управления базами данных

- Хранят данных на долговременных носителях
- Предоставляют интерфейс по созданию данных, поиску и получению данных, обновлению данных и удалению данных
- Различаются по способу хранения и работы с данными
 - noSql
 - SQL

Пример noSQL иерархической базы данных



Пример SQL: реляционных данных

Customer

cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account

account_id	product_cd	cust_id	balance
103	CHK	1	\$75.00
104	SAV	1	\$250.00
105	CHK	2	\$783.64
106	MM	2	\$500.00
107	LOC	2	0

Product

product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	\$100.00	2004-01-22
979	CDT	103	\$25.00	2004-02-05
980	DBT	104	\$250.00	2004-03-09
981	DBT	105	\$1000.00	2004-03-25
982	CDT	105	\$138.50	2004-04-02
983	CDT	105	\$77.86	2004-04-04
984	DBT	106	\$500.00	2004-03-27

Основные термины

- **Сущность**
объект реального мира, который интересно хранить в СУБД
- **Колонка**
Атрибут сущности, который сохраняется в таблице
- **Строка**
Набор атрибутов сущности, хранящихся в колонках и представляющие все данные сущности
- **Таблица**
набор строк, которые хранятся на долговременном носителе
- **Основной ключ (primary key)**
Одна или несколько колонок, с помощью которой можно идентифицировать сущность
- **Внешний ключ (foreign key)**
Одна или несколько колонок, с помощью которой можно идентифицировать сущность связанную с текущей, но находящейся в другой таблице

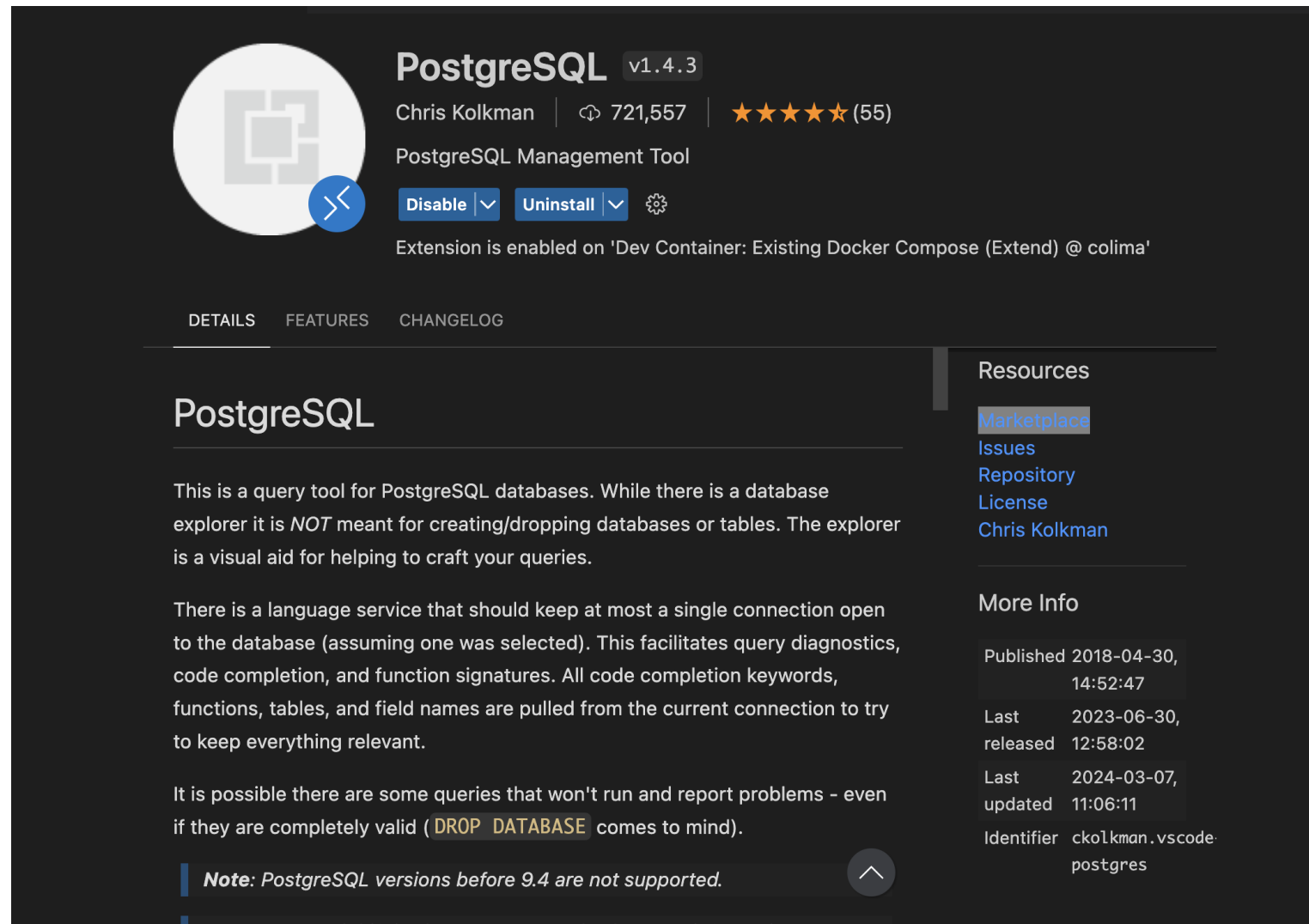
PostgreSQL

запускаем через docker-compose

```
db:
  container_name: postgres
  image: postgres:15
  environment:
    - POSTGRES_USER=stud
    - POSTGRES_PASSWORD=stud
    - PGDATA=/data/postgres
    - POSTGRES_DB=archdb
  volumes:
    - db:/data/postgres
  ports:
    - "5432:5432"
```

<https://www.postgresql.org/>

Используем расширение PostgreSQL Explorer



The screenshot shows the 'PostgreSQL' extension page in the Visual Studio Code Marketplace. At the top, there's a circular icon with a stylized 'P' and a blue circle with '<<' indicating it's installed. The extension is by 'Chris Kolkman', has 721,557 downloads, and a 5-star rating from 55 reviews. It's labeled as 'PostgreSQL Management Tool'. Below this, there are buttons for 'Disable' and 'Uninstall', and a gear icon for settings. A status message says 'Extension is enabled on 'Dev Container: Existing Docker Compose (Extend) @ colima''. The main content area has tabs for 'DETAILS', 'FEATURES', and 'CHANGELOG'. The 'DETAILS' tab is active, showing the extension's description: 'This is a query tool for PostgreSQL databases. While there is a database explorer it is NOT meant for creating/dropping databases or tables. The explorer is a visual aid for helping to craft your queries.' It also mentions a language service that keeps a single connection open for diagnostics and code completion. A note at the bottom states: 'Note: PostgreSQL versions before 9.4 are not supported.' On the right side, there's a 'Resources' section with links to 'Marketplace', 'Issues', 'Repository', 'License', and 'Chris Kolkman'. Below that is a 'More Info' section with a table of metadata.

More Info	
Published	2018-04-30, 14:52:47
Last released	2023-06-30, 12:58:02
Last updated	2024-03-07, 11:06:11
Identifier	ckolkman.vscode-postgres

Подключаемся к PostgreSQL

```
> psql -d archdb -U stud -h db
```


SQL

- Язык для работы с реляционными данными
- Позволяет работать с таблицами (создавать, менять, удалять) и данными

Пример SQL создание таблицы

```
CREATE TABLE corporation (corp_id SMALLINT,  
name VARCHAR(30) );
```

<https://www.w3schools.com/sql/>

Пример SQL добавление строки в таблицу

```
INSERT INTO corporation (corp_id, name)  
VALUES (27, 'Acme Paper Corporation');
```

<https://www.w3schools.com/sql/>

Пример SQL поиск данных в таблице

```
SELECT name FROM corporation WHERE corp_id =  
27;
```

<https://www.w3schools.com/sql/>

Пример SQL

ПОИСК

СВЯЗАННЫХ

данных в

таблицах

```
SELECT t.txn_id,  
t.txn_type_cd,  
t.txn_date, t.amount
```

```
FROM individual i
```

```
INNER JOIN account a ON  
i.cust_id = a.cust_id
```

```
INNER JOIN product p ON  
p.product_cd =  
a.product_cd
```

```
INNER JOIN transaction  
t ON t.account_id =  
a.account_id
```

```
WHERE i.fname =  
'George' AND i.lname =  
'Blake' AND p.name =  
'checking account';
```

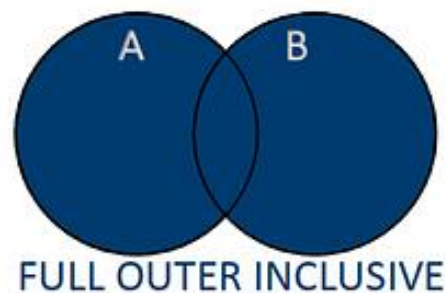
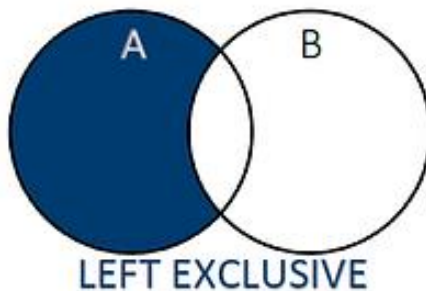
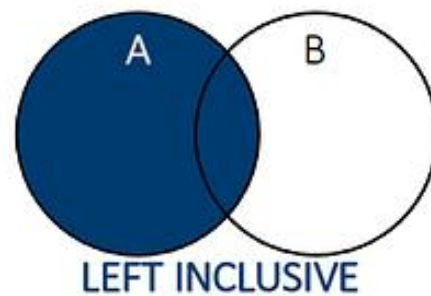
Customer cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account account_id	product_cd	cust_id	balance
103	CHK	1	\$75.00
104	SAV	1	\$250.00
105	CHK	2	\$783.64
106	MM	2	\$500.00
107	LOC	2	0

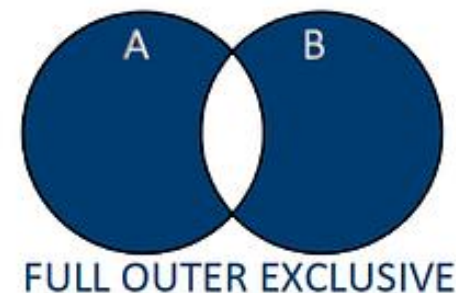
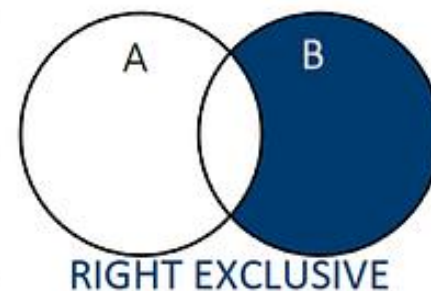
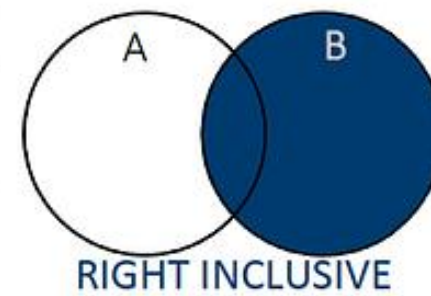
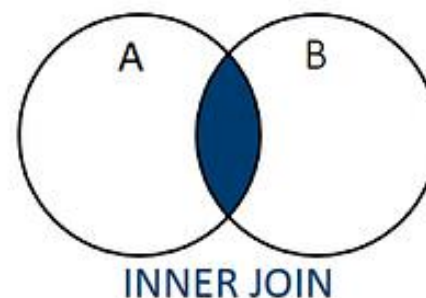
Product product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	\$100.00	2004-01-22
979	CDT	103	\$25.00	2004-02-05
980	DBT	104	\$250.00	2004-03-09
981	DBT	105	\$1000.00	2004-03-25
982	CDT	105	\$138.50	2004-04-02
983	CDT	105	\$77.86	2004-04-04
984	DBT	106	\$500.00	2004-03-27

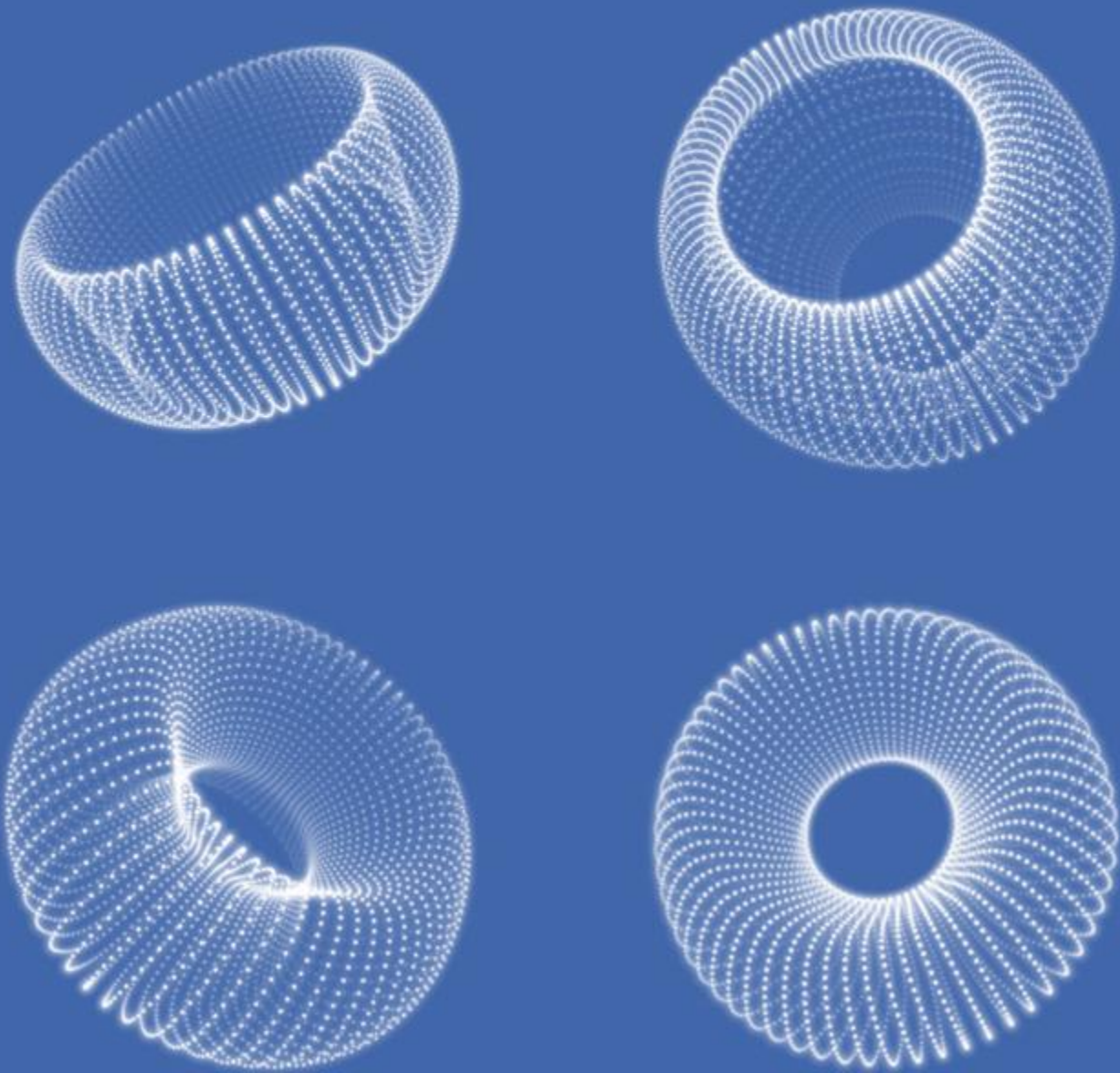
Виды join



SQL JOINS	
LEFT INCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key	RIGHT INCLUSIVE SELECT [Select List] FROM TableA A RIGHT OUTER JOIN TableB B ON A.Key = B.Key
LEFT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key WHERE B.Key IS NULL	RIGHT EXCLUSIVE SELECT [Select List] FROM TableA A LEFT OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL
FULL OUTER INCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key	FULL OUTER EXCLUSIVE SELECT [Select List] FROM TableA A FULL OUTER JOIN TableB B ON A.Key = B.Key WHERE A.Key IS NULL OR B.Key IS NULL
INNER JOIN SELECT [Select List] FROM TableA A INNER JOIN TableB B ON A.Key = B.Key	



Нормальные формы



Нормализация

Цель нормализации: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей(строк таблицы).

Аномалией называется такая ситуация в таблице БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.

Первая нормальная форма

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Производитель	Автомобиль
Автоваз	Vesta, Granta
Volkswagen	Golf, Passat, Taureg



Производитель	Автомобиль
Автоваз	Vesta
Автоваз	Granta
Volkswagen	Golf
Volkswagen	Passat
Volkswagen	Taureg

Вторая
нормальная
форма

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Производитель	Автомобиль	Цена	Скидка
Автоваз	Vesta	5500000	5%
Автоваз	Granta	2300000	5%
Volkswagen	Golf	2000000	80%
Volkswagen	Passat	2400000	80%

Производитель	Автомобиль	Цена
Автоваз	Vesta	5500000
Автоваз	Granta	2300000
Volkswagen	Golf	2000000
Volkswagen	Passat	2400000

Производитель	Скидка
Автоваз	5%
Volkswagen	80%

Третья нормальная форма

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Производитель	Магазин	Адрес
Автоваз	Реал-авто	Краснопролетарская, 4
УАЗ	Реал-авто	Краснопролетарская, 4
Volkswagen	Даз-авто	Леснорядский, 1
Камаз	Даз-авто	Леснорядский, 1

Производитель	Магазин
Автоваз	Реал-авто
Уаз	Реал-авто
Volkswagen	Даз-авто
Камаз	Даз-авто

Магазин	Адрес
Реал-авто	Краснопролетарская, 4
Даз-авто	Леснорядский, 1

Масштабирование данных

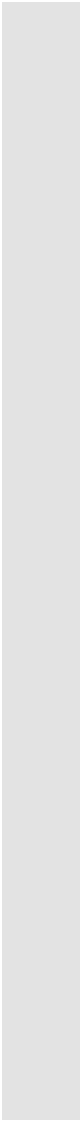



Проблемы

- **Когда думать о масштабировании**
 - На этапе проектирования
 - На этапе эксплуатации
- **Основные причины**
 - Рост трафика на проекте
 - Увеличение вычислительной сложности
 - Постоянный рост объема данных
 - Рост нагрузки на БД в определенных частях приложения
 - Проблема с бэкапами и миграциями данных
 - Вы можете спрогнозировать время когда кончится место на диске, или не будет хватать процессора

Содержание занятия

1. Индексы
2. Партиционирование
3. Шардирование

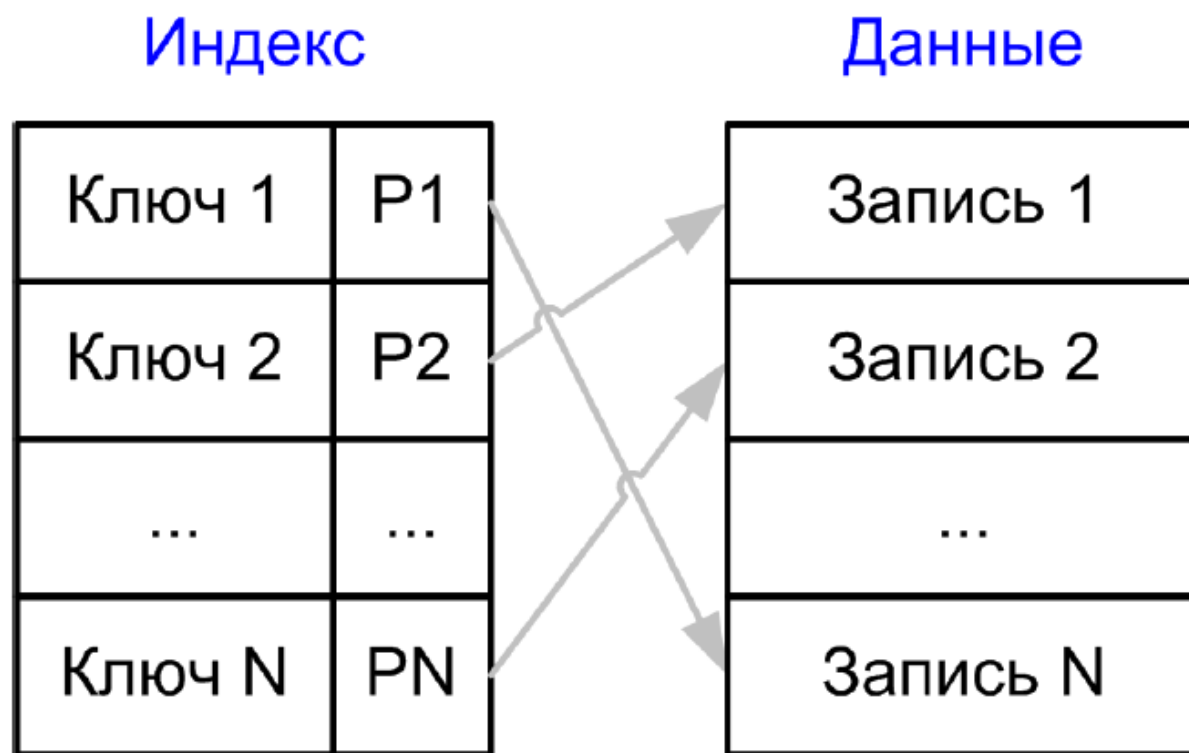


Индексы в реляционных базах данных

Индекс

- служебная структура данных
- некий способ отображения ключа в данные
- ускоряют поиск, сортировку
- требуют доп. место для хранения
- обновляются при вставке (модификации данных)

Пример

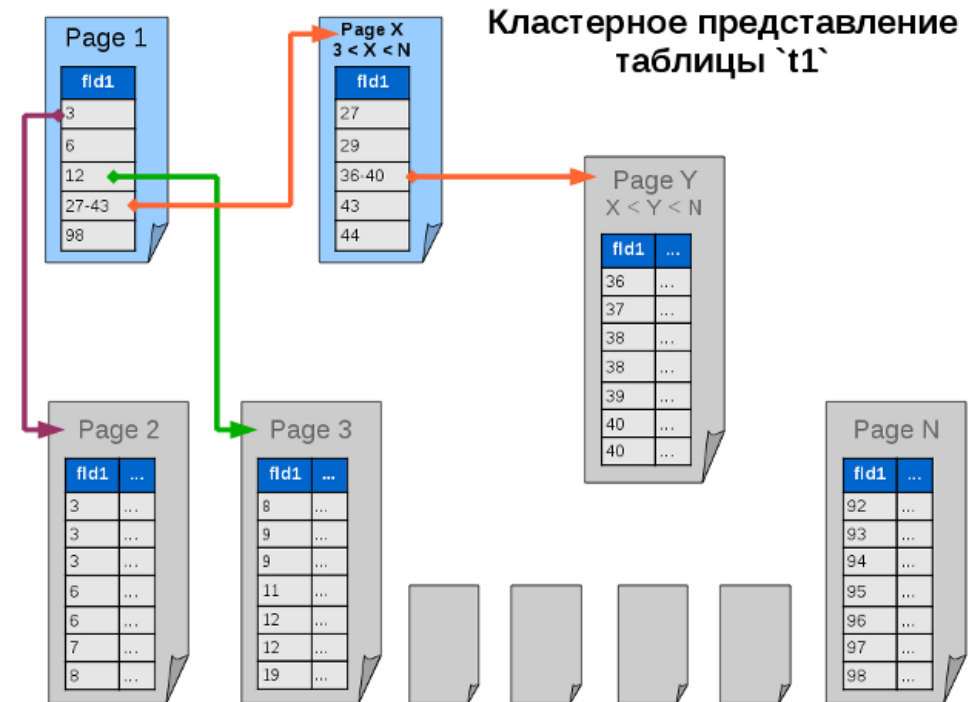
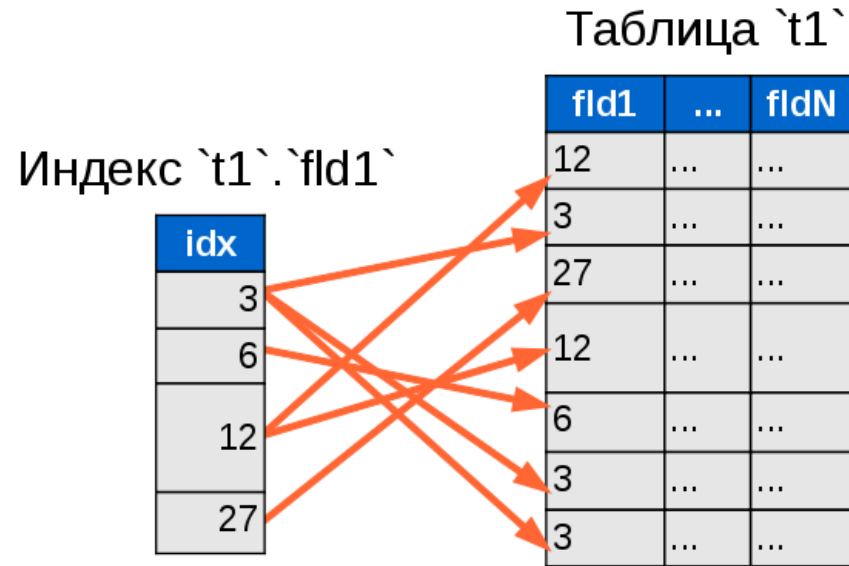


Индексы

- Индексная таблица весит много меньше таблицы с данными.
- СУБД чаще всего стараются кешировать индексы в оперативную память.
- В индексах отсутствуют дублирующиеся строки. А значит, как только мы нашли первое значение, поиск можно прекращать — оно же и последнее.
- Данные в индексе отсортированы (в общем случае)

Виды

- уникальные и нет
- простые и составные
- кластерные и некластерные



Кластерные и не кластерные индексы

<https://habr.com/ru/post/141767/>

Кластерные индексы

- Кластерный индекс — это древовидная структура данных, при которой значения индекса хранятся вместе с данными, им соответствующими.
- И индексы, и данные при такой организации упорядочены.
- При добавлении новой строки в таблицу, она дописывается не в конец файла, не в конец плоского списка, а в нужную ветку древовидной структуры, соответствующую ей по сортировке.

Вопрос

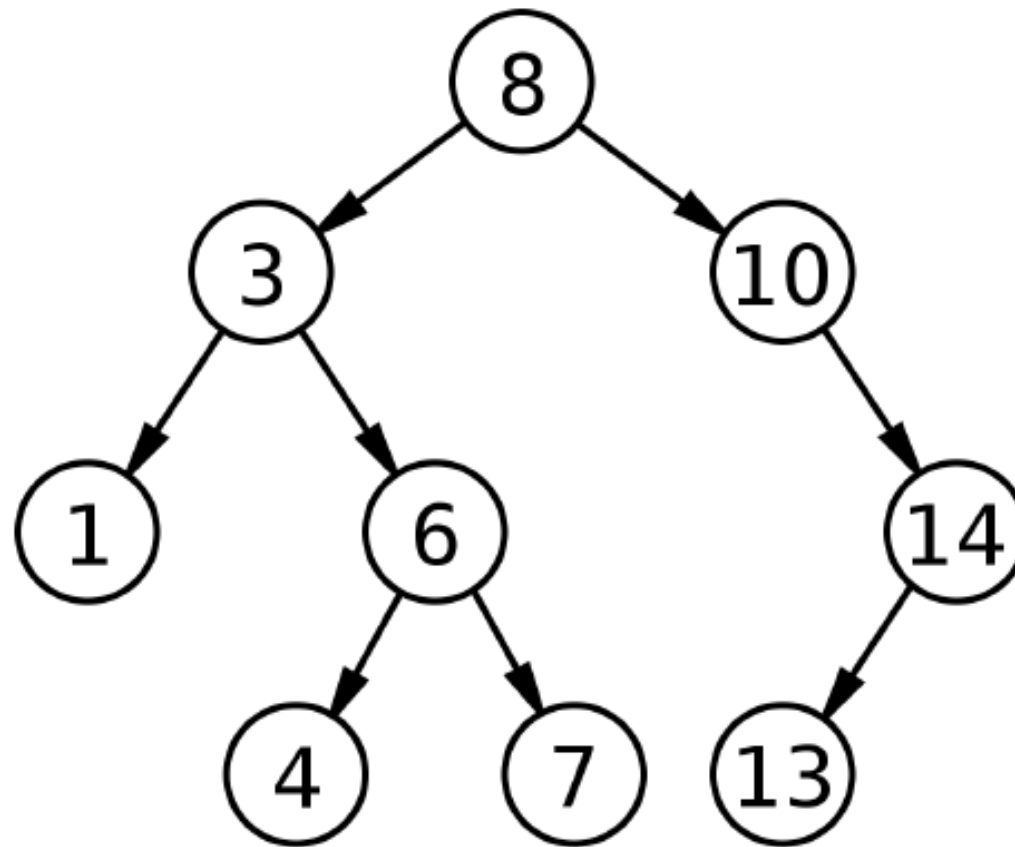
Много индексов - хорошо
или не очень?



Устройство индекса

деревья

Простое бинарное дерево



Свойства бинарного дерева

- гарантированный порядок элементов (+)
- понятный поиск (+)
- в вырожденном случае придется посетить все элементы, т.е. $O(n)$ (-)

Оценка сложности

Определение $O(n)$:

Пусть $f(n)$ и $g(n)$ — две функции, отображающие натуральные числа в действительные числа.

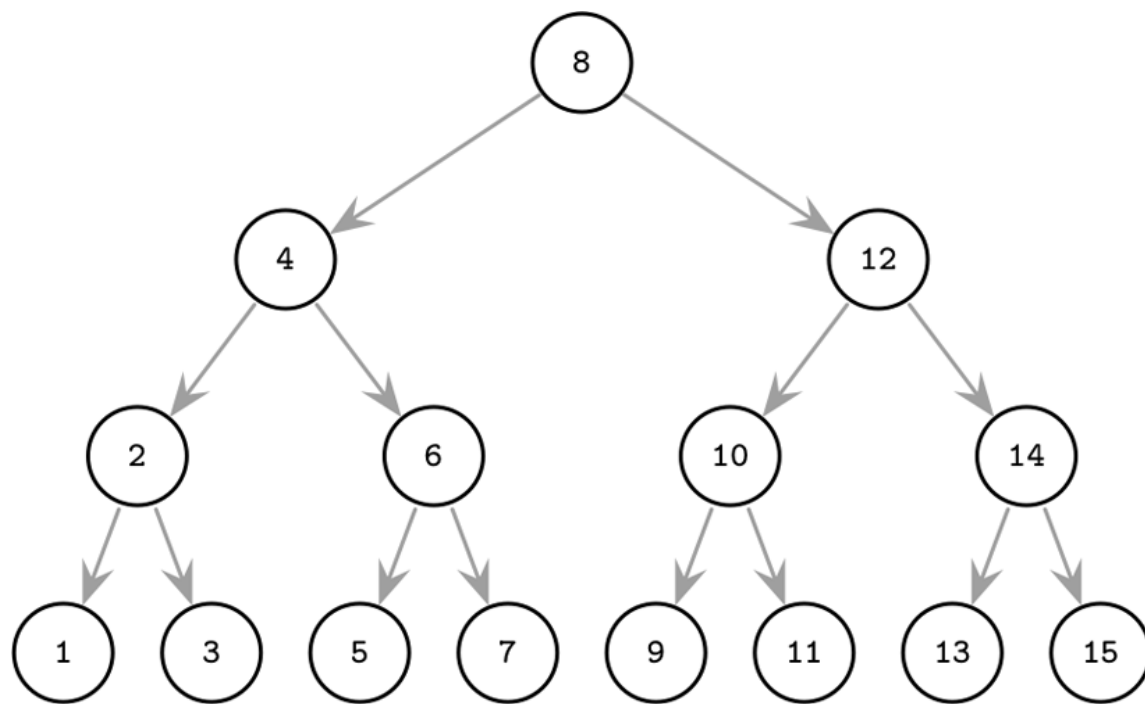
Говорят, что $f(n)$ является **$O(g(n))$** (читается как "О большое от $g(n)$ "), если существуют такие положительные константы C и n_0 , что для всех $n \geq n_0$ выполняется неравенство:

$$|f(n)| \leq C \cdot |g(n)|$$

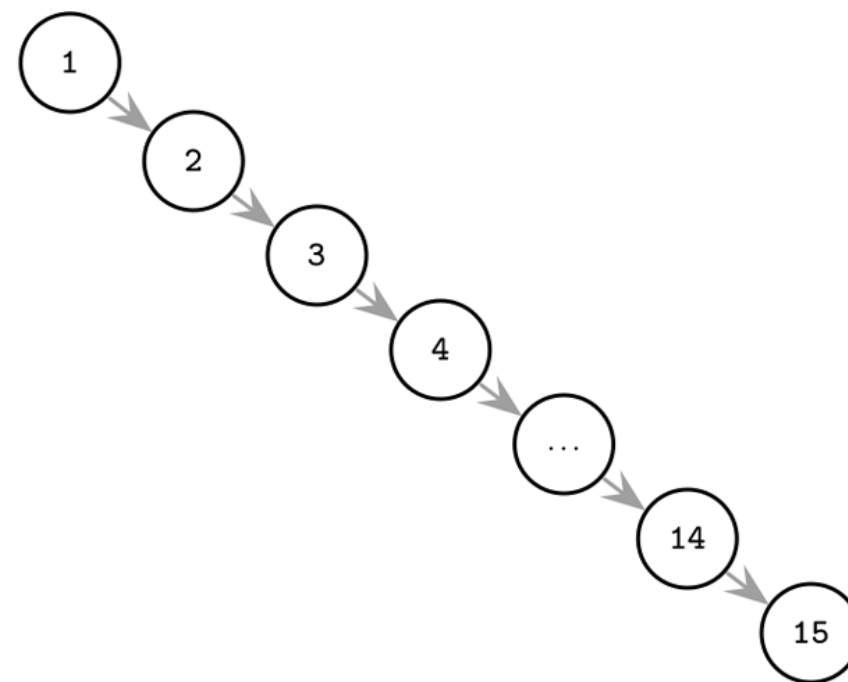
Определение для $O(n)$:

В частности, если $g(n)=n$, то говорят, что $f(n)$ является **$O(n)$** , если существуют такие положительные константы C и n_0 , что для всех $n \geq n_0$ выполняется неравенство:

$$|f(n)| \leq C \cdot n$$



a)



b)

Сбалансированные и не сбалансированные деревья

Что поменялось?

- решает проблему вырожденного случая бинарного дерева (+)
- поиск за $O(\text{высоты дерева})$ (+)
- требует дополнительных усилий на балансировку (-)

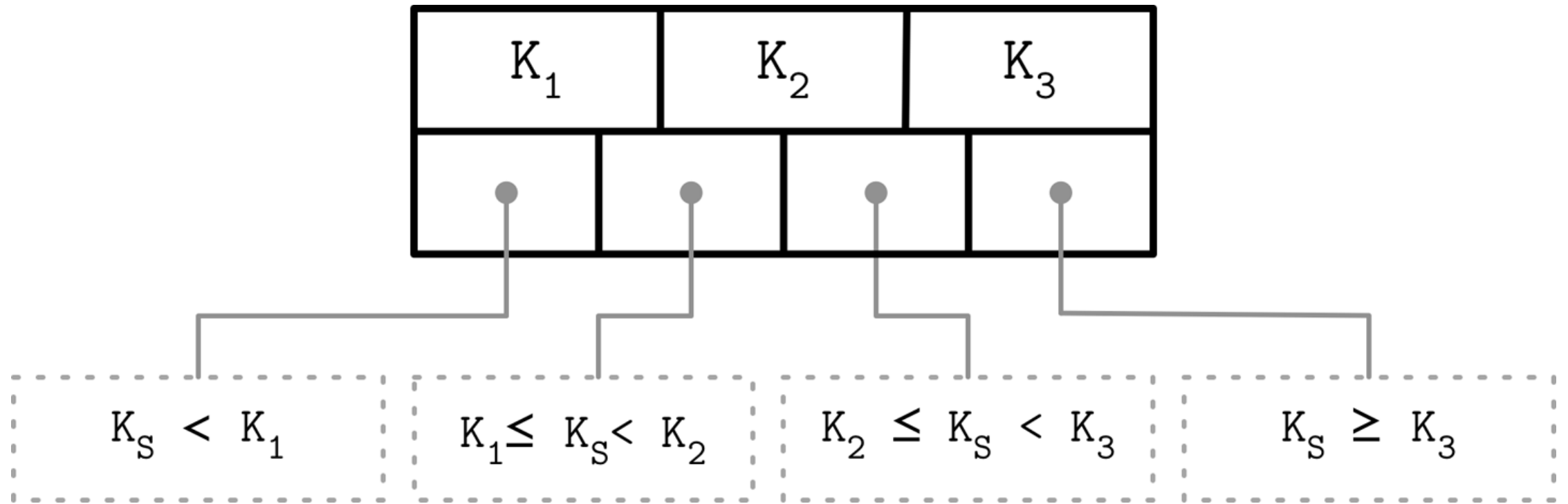
Проблемы с бинарными деревьями для хранения больших объемов данных

- оперативная память очень ограничена
- будем использовать внешний носитель (привет, медленный HDD)
- $\log N$ обращений к медленной памяти

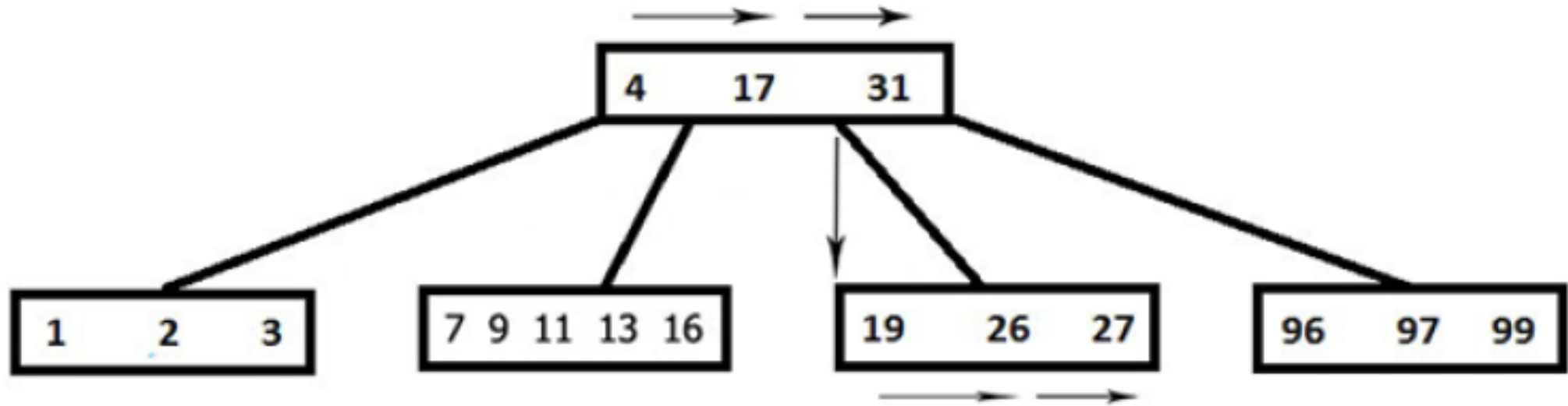
B-Tree

В-дерево — это особый вид сбалансированного дерева, который позволяет нам выполнять операции поиска, вставки и удаления записей из внешнего файла с гарантированной производительностью для самой неблагоприятной ситуации

1. Корень либо является листом, либо имеет по крайней мере двух сыновей.
2. Каждый узел, за исключением корня и листьев, имеет от $\lceil t/2 \rceil$ до t сыновей.
3. Все пути от корня до любого листа имеют одинаковую длину.



Принцип разделения элементов



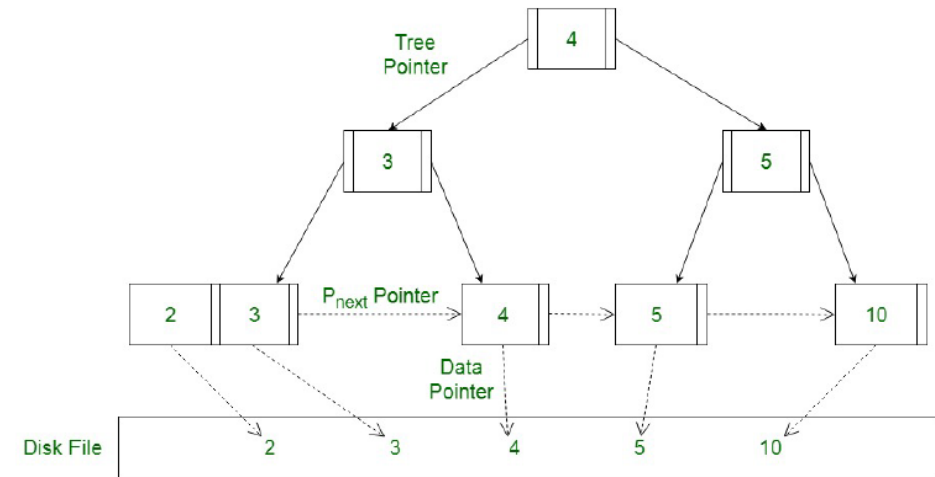
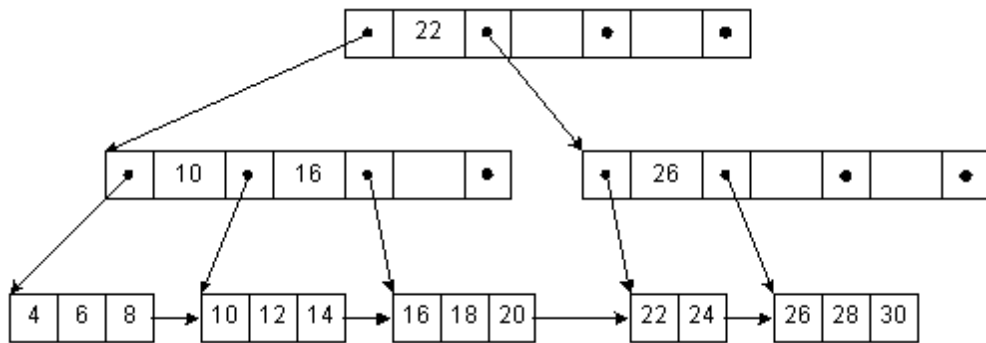
Поиск в В-дереве

Поиск в b-tree

- алгоритм аналогичен бинарному, но дальнейший выбор не из 2х, а из нескольких
- t – размер блока
- поиск за $O(t \log(n))$
- НО обращений к диску $O(\log(n))$

размер t

- больше $t \Rightarrow$ меньше высота дерева
- зависит от размера блока на диске
- зависит от объема оперативной памяти
- обычно от 50 до 2000
- $t=1000$ и 1млрд записей \Rightarrow 3 операции для любого ключа



B+tree

все ключи хранятся в листьях

Посмотрим интерактив

<https://planetscale.com/blog/btrees-and-database-indexes>



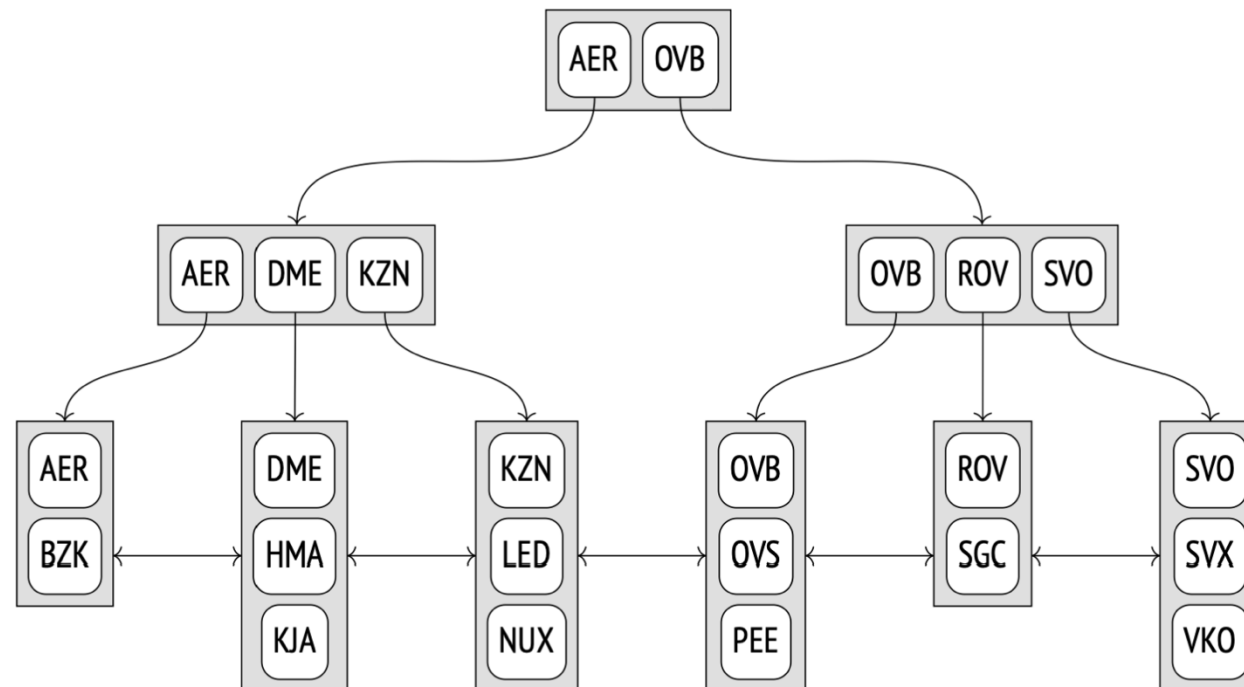
Индексы в PostgreSQL

Виды индексов

- Хэш индекс
- **B – дерево**
- GIST (Generalized Search Tree)
- SP-GIST (Space Partitioning)
- GIN (Generalized Inverted Index)
- BRIN (Block Range Index)

B-Tree

Организованы с
помощью B+ tree



B-Tree

- Может помочь с поиском по:
 - Равенству ($a = 5$)
 - Открытому диапазону ($a > 5$ or $a < 3$)
 - Закрытому диапазону ($3 < a < 5$)
- Не может помочь с поиском:
 - Четных чисел
 - Суффикса (но может помочь с префиксом)

создадим индекс

- `CREATE UNIQUE INDEX author_id ON Author (id);`
- `CREATE INDEX fn ON Author(first_name);`
- `CREATE INDEX fn_ln ON Author(first_name,last_name);`

Online data generator

Contact More tools

Welcome to
Online test data generator
Here you can generate up to 100 combinations of data formats and information and export up to 100,000 records. Build up your test datatable and export your data in CSV, Excel, Json, or even Sql script to create your table. You can use weights, nullable fields and more options to generate test data.

1 Define columns

Column Name	Field Type		
first_name	First Name		
last_name	Last Name		
email	Email Address		
title	Job Title		
Add column		Reset columns	

2 Export data

Rows

100000

Choose how many rows do you want to generate

Export Type

Json

Choose how your data will look like

EXPORT

3 Preview data live

Our website use cookies to ensure you get the best experience on our website!
Learn more

Got it!

Генерируем данные

<https://www.onlinedatagenerator.com/>

Зальем данные в СУБД

```
import pandas as pd
from sqlalchemy import create_engine
engine = create_engine("postgresql+psycopg2://stud:stud@db/archdb", echo = False)

df = pd.read_json("ExportJson.json")
df.to_sql("users", con=engine, if_exists = 'replace', index=False)
```

Анализируем запросы

- `explain(analyze) select * from users where id=10;`
- анализирует то как будет выполняться запрос в СУБД

Индексы при поиске по строкам

Для строк работают все те же правила

- Правила сравнения строк называются collation
- LIKE по префиксу может работать с индексами (LIKE 'a%', т. к. эквивалентен диапазону 'a[lowest]' < key < 'a[largest]')

Индексы по нескольким полям

```
create index ln_fn on users(last_name,first_name);
```

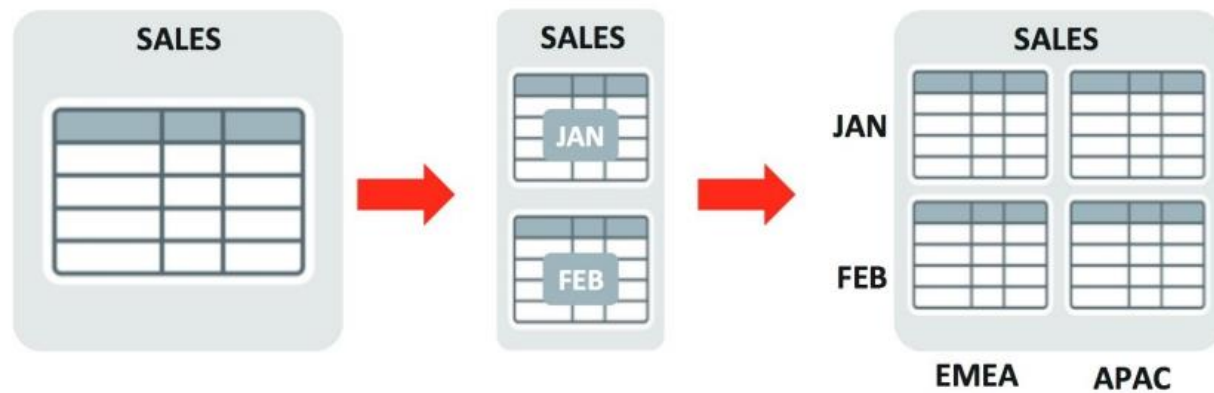
```
explain (analyze) select * from users where first_name  
like 'Elle%' and last_name like 'A%';
```



Partitioning

Партиционирование partitioning

- Подход к масштабированию БД
- Разделение данных по определенным признакам
- Разделенные данные лежат отдельно
- Можно делить как вертикально, так и горизонтально
- Можно эффективно утилизировать дисковую подсистему
- Все partitions в пределах одного инстанса БД (сервера)



Партиционирование partitioning

Основные типы

- Key
- List
- Range
- Hash

Partitioning PostgreSQL

```
CREATE TABLE orders (  
  order_id SERIAL,  
  order_date DATE NOT NULL,  
  customer_name VARCHAR(255),  
  product_name VARCHAR(255),  
  quantity INT  
) PARTITION BY RANGE(EXTRACT(YEAR FROM order_date));
```

```
CREATE TABLE orders_2019 PARTITION OF orders FOR VALUES FROM (2019) TO  
(2020);  
CREATE TABLE orders_2020 PARTITION OF orders FOR VALUES FROM (2020) TO  
(2021);  
CREATE TABLE orders_2021 PARTITION OF orders FOR VALUES FROM (2021) TO  
(2022);
```

Partitioning - итого

- может ускорить запросы (если запросы попадают в одну партицию)
- может замедлить запросы (если запросы всегда попадают в несколько партиций)
- не решается проблема ограниченности сервера
- не используем если строк меньше 1 млн

Где используется

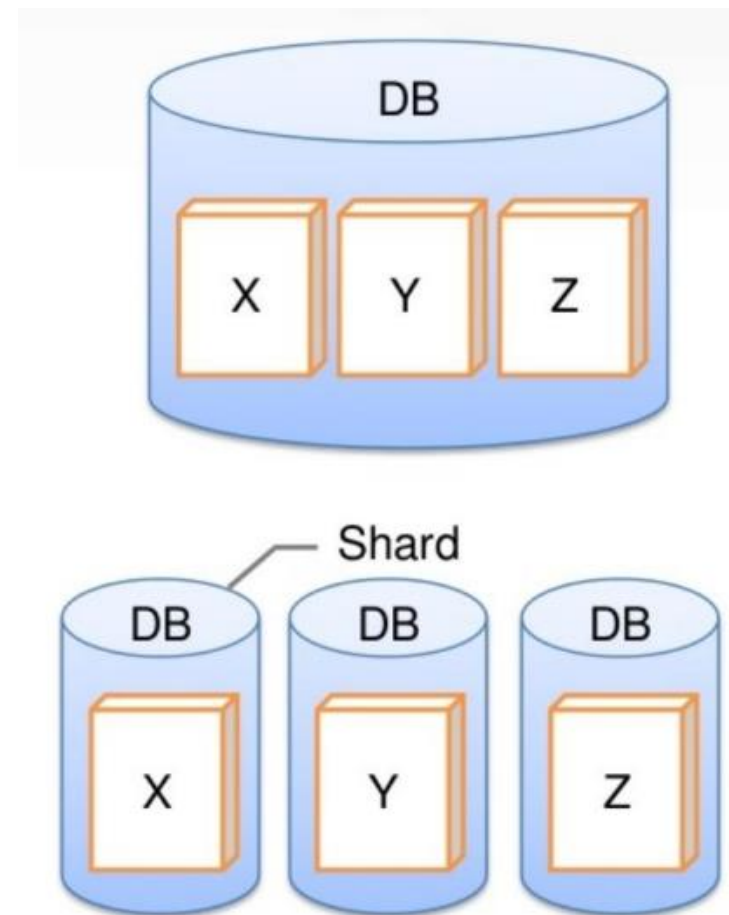
- Для исторических данных
- Для аналитики
- Для группировки данных (география, категории, языки)

Sharding



Шардирование

- Хранение данных в различных инстансах БД
- Различные стратегии разделения данных
- Можно хранить на разных серверах



Мотивация

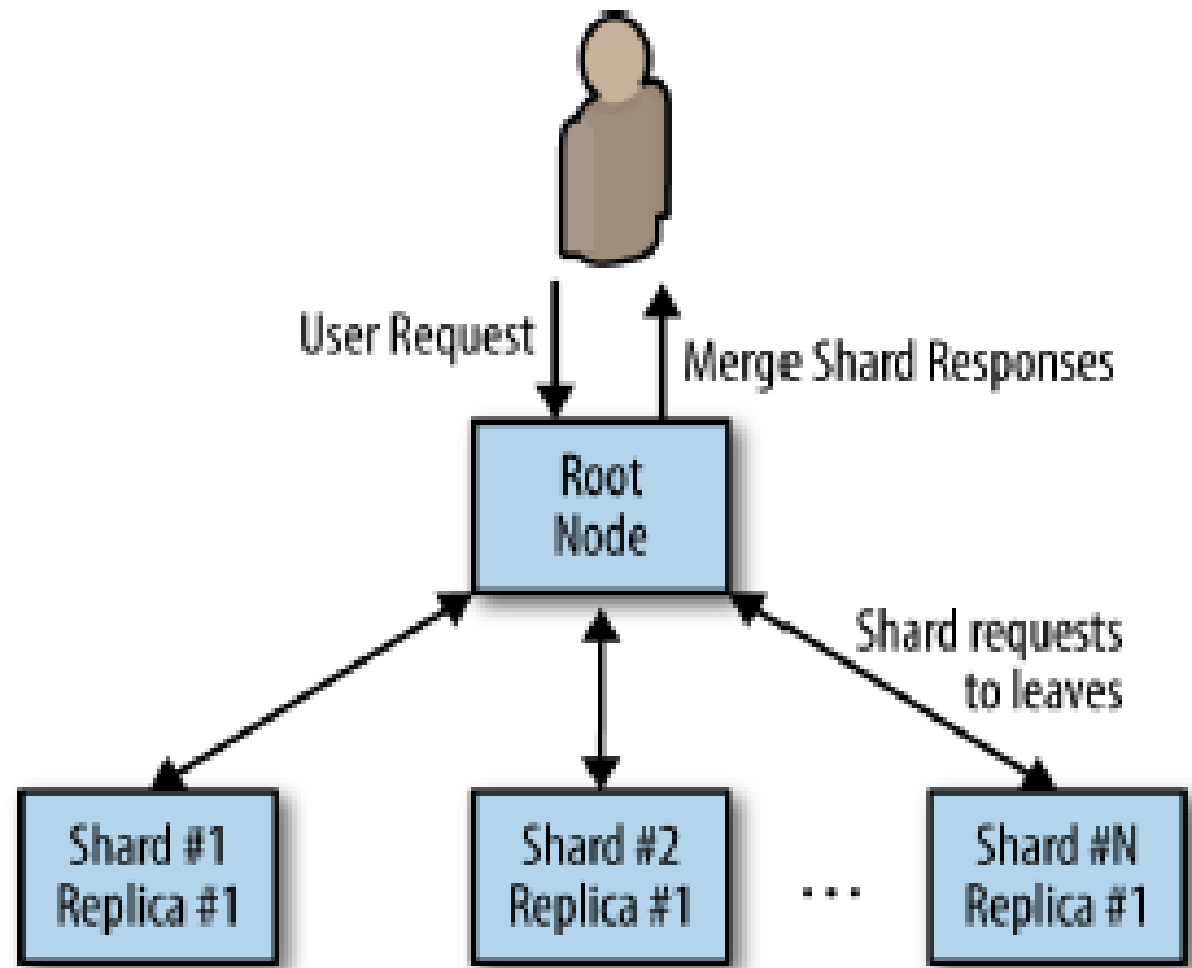
Зачем

- позволяет горизонтально масштабировать данные
- ускоряет обработку запросов (особенно на запись)
- повышает отказоустойчивость
- экономит деньги на дорогих серверах

Где используется

- Соц. сети
- Системы сообщений, хостинги данных
- Сервисы сбора данных
- Гео-распределенные сервисы

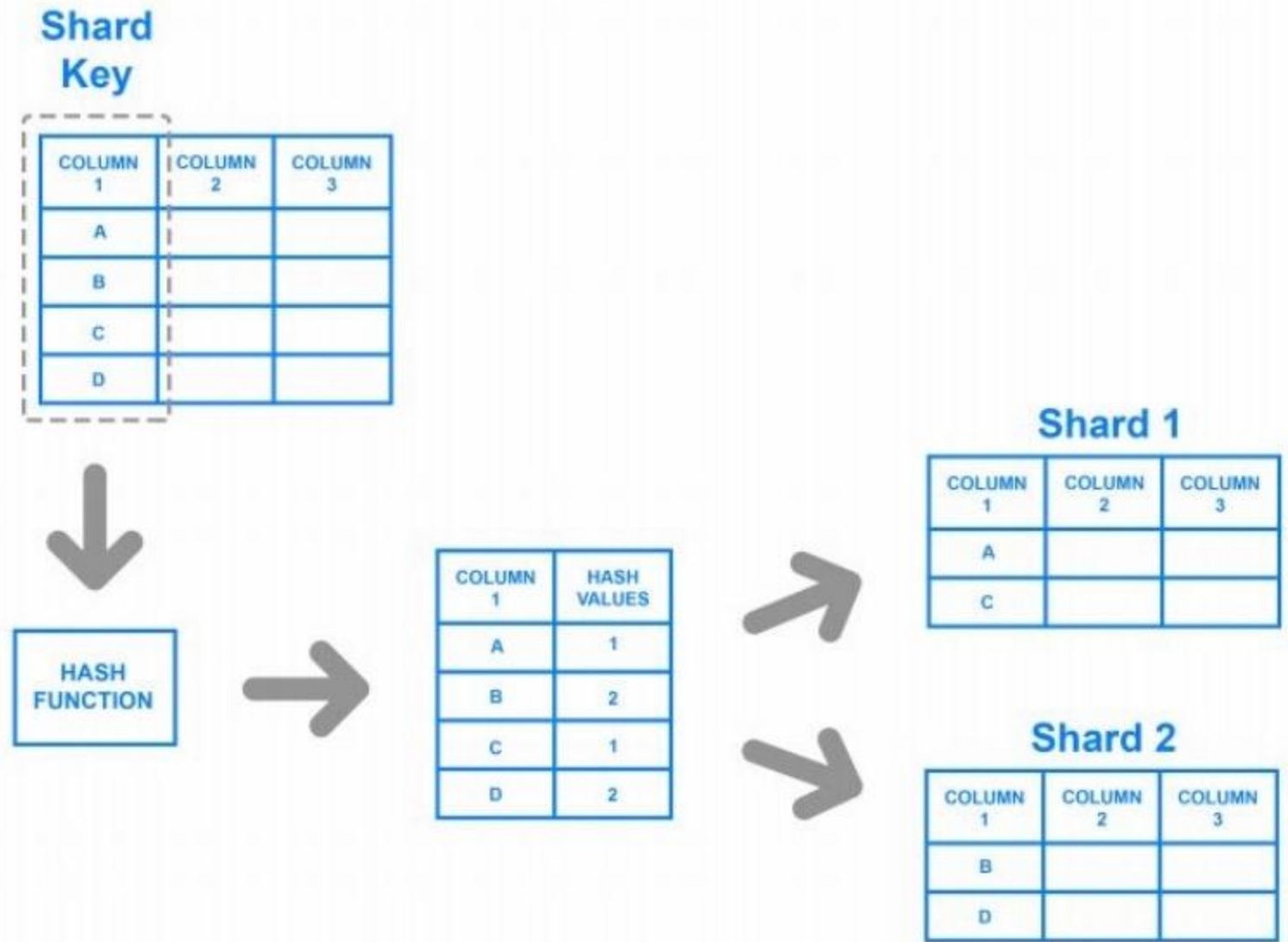
Scatter and gatherer



Критерии хорошей стратегии шардирования

- Запросы должны попадать в один шард (идеально)
- Шарды должны быть сравнимы по размерам
- Запросы должны разделяться по шардам равномерно

Key Based Sharding



Key Based Sharding

Характеристики

- hash based
- формула примерно такая: $F(\text{key}) \rightarrow \text{shard_id}$
- F и key очень важны
- наиболее распространенный способ

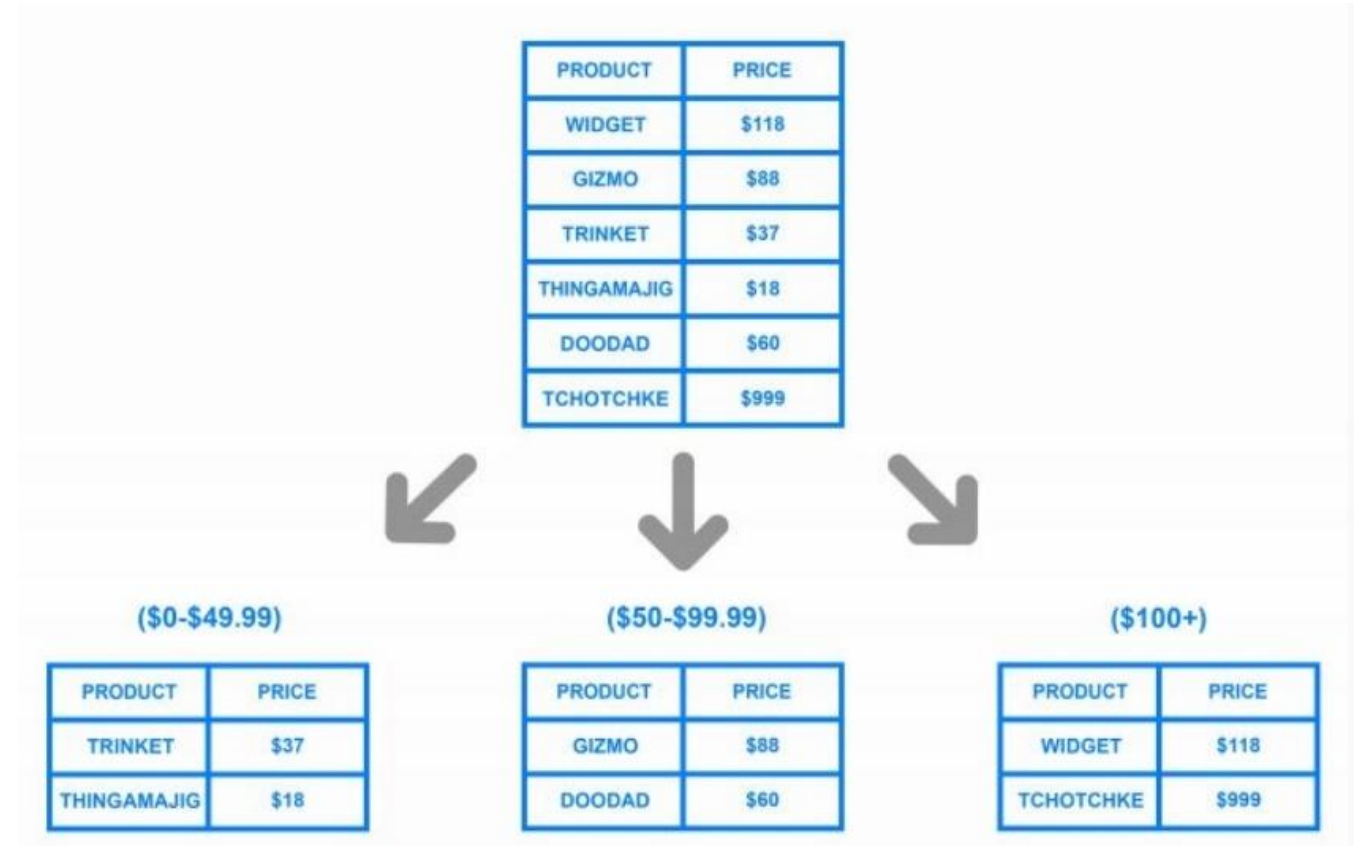
Плюсы

- Просто и понятно
- Равномерное распределение

Минусы

- Добавление и удаление шарда - может быть сложно
- Особенности распределения нужно учитывать при разработке приложения

Range Based Sharding



Range Based Sharding

Характеристики

- еще называют table function / virtual bucket
- статический конфиг range -> shard_id
- формула примерно такая: $\text{func}(\text{key}) \rightarrow \text{virtual_bucket} \rightarrow \text{shard_id}$

Плюсы

- Просто и понятно
- Легко тестировать и анализировать

Минусы

- Сложный процесс обновления

Directory Based Sharding

Pre-Sharded Table

DELIVERY ZONE	FIRST NAME	LAST NAME
3	DARCY	CLAY
1	DENISE	LASALLE
2	HIROSHI	YOSHIMURA
4	KIRSTY	MACCOLL



DELIVERY ZONE	SHARD ID
1	S1
2	S2
3	S3
4	S4



S1		
1	DENISE	LASALLE

S2		
2	HIROSHI	YOSHIMURA

S3		
3	DARCY	CLAY

S4		
4	KIRSTY	MACCOLL

Directory Based Sharding

Характеристики

- требует определенной структуры данных
- похож на range based
- статический конфиг key -> shard_id

Плюсы

- Не нужно думать о хеш функции
- Совпадает с вашей бизнес-логикой
- Можно реализовать гибкую логику

Минусы

- Часто можно встретить low cardinality key
- Потенциально можно упереться в аппаратные мощности по определенному ключу
- Боль с обновлением
- Single Point Of Failure

Как управлять доступом к данным

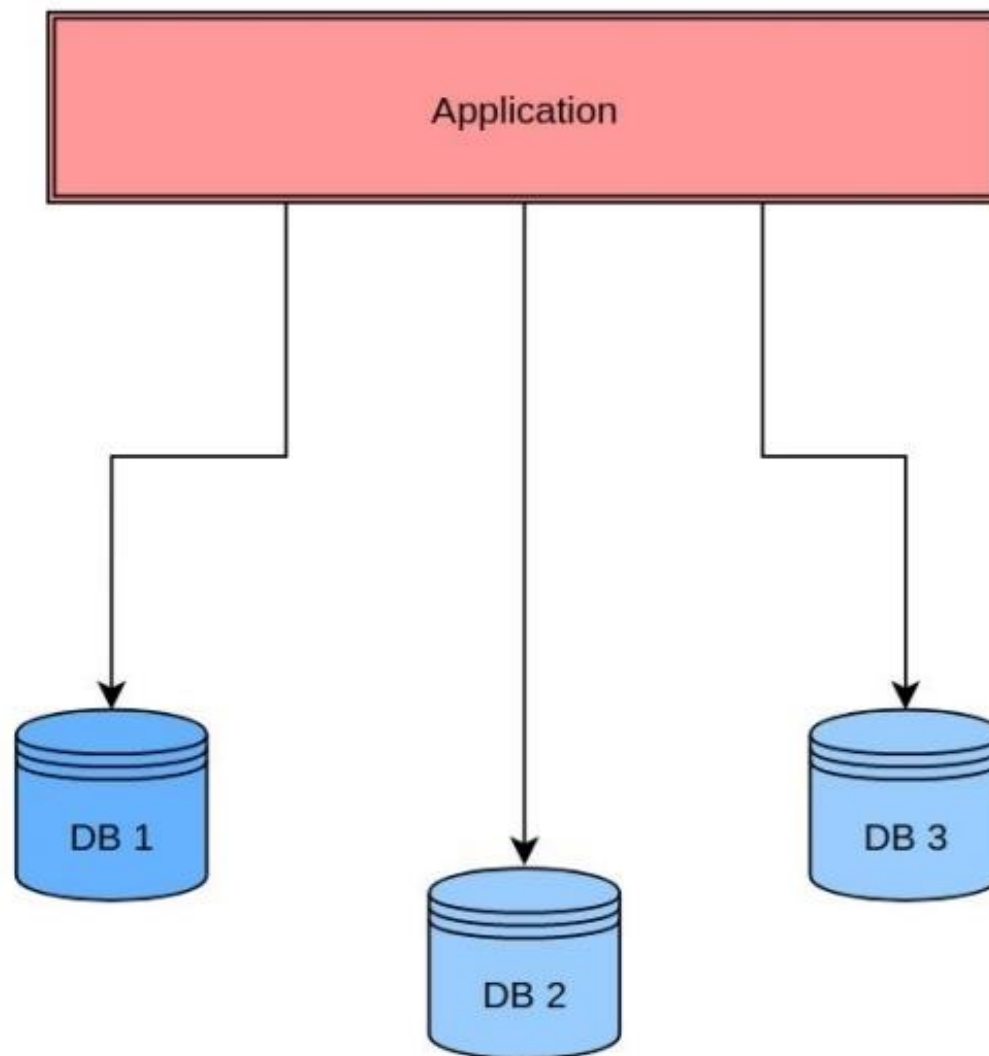
Умный клиент

Плюсы

- Простой метод
- Нет лишних хопов по сети

Минусы

- Нужно учитывать при разработке
- Приложение должно знать довольно много про инфраструктуру (hosts, credentials)
- Сложность с обновлением
- Сложность с тестированием
- Как делать решардинг?



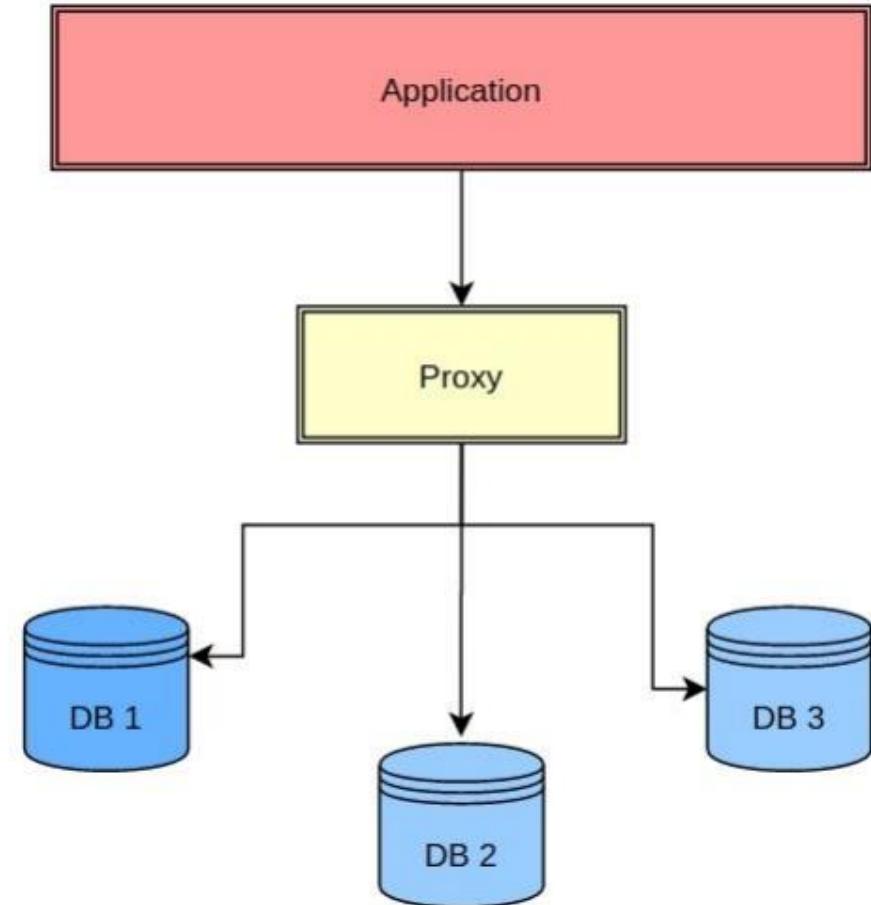
Routing-proxy

Плюсы

- Приложение ничего не знает о шардинге
- Код не меняется

Минусы

- Лишний хоп
- Потеря Latency
- SPOF (single point of failure)
- Потеря в функциональности



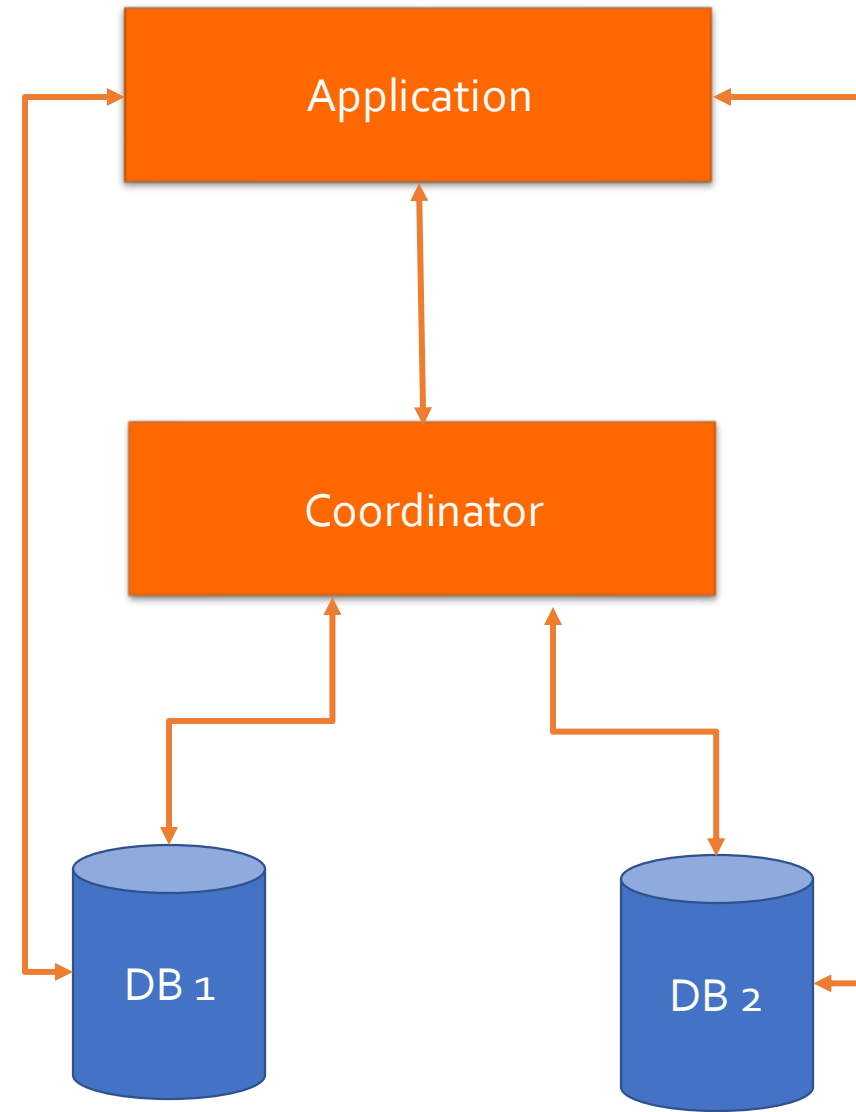
Routing координатор

Плюсы

- Приложение ничего не знает о шардинге
- Код не меняется
- Есть возможности по оптимизации и кэширования

Минусы

- Лишний хоп (hop*)
- Потеря Latency
- Инфраструктурная сложность
- Потеря в функциональности
- Нагрузка
- SPOF





Решардинг

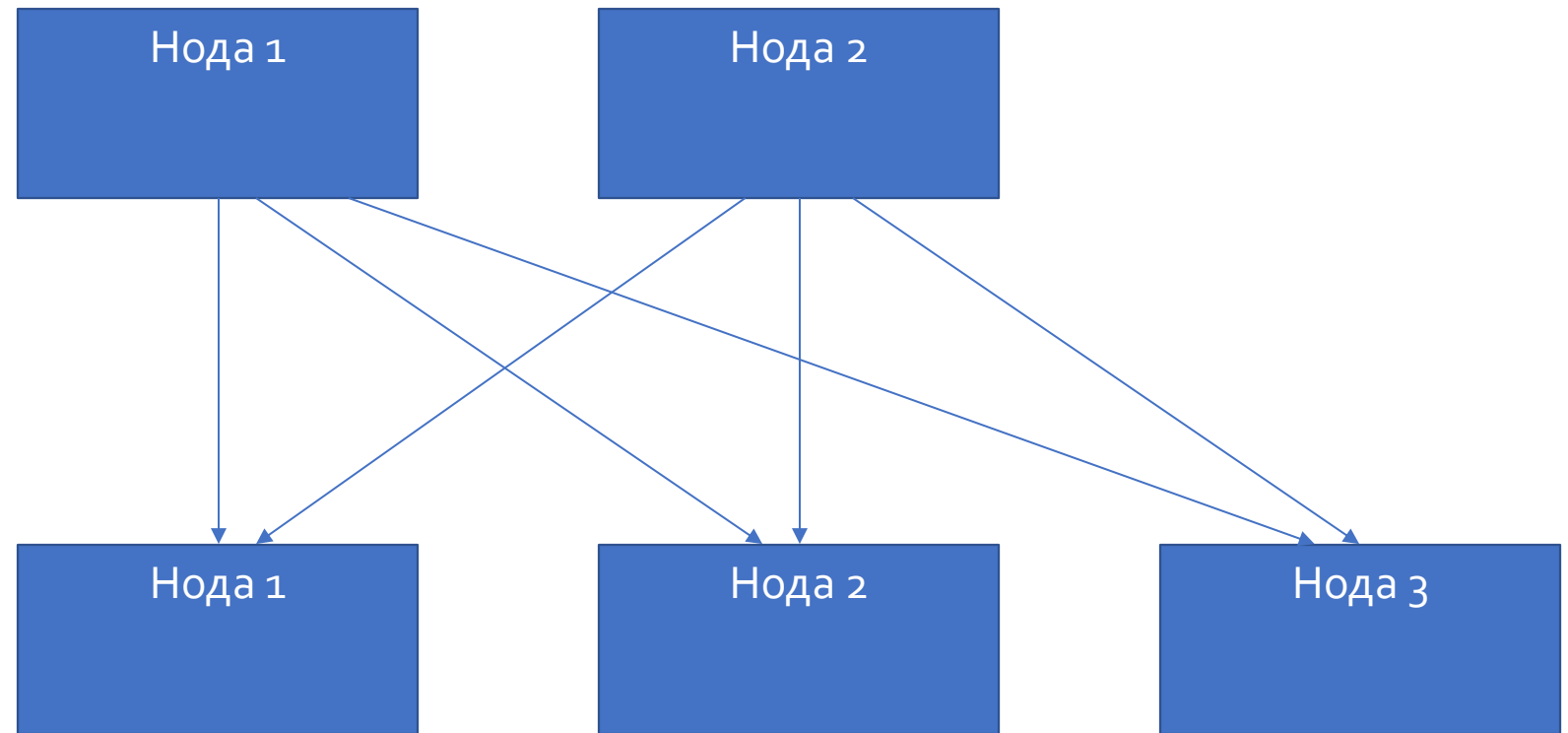
Resharding

Добавление / удаление нод

- Потребовалось “передвинуть данные”
- Исправление ошибок при выборе hash-функции
- Устранение отказа оборудования
- Когда-то придется решардить так или иначе

Добавляем
ноду

$\text{node_index} = \text{hash}(\text{key}) \% \text{max_node}$



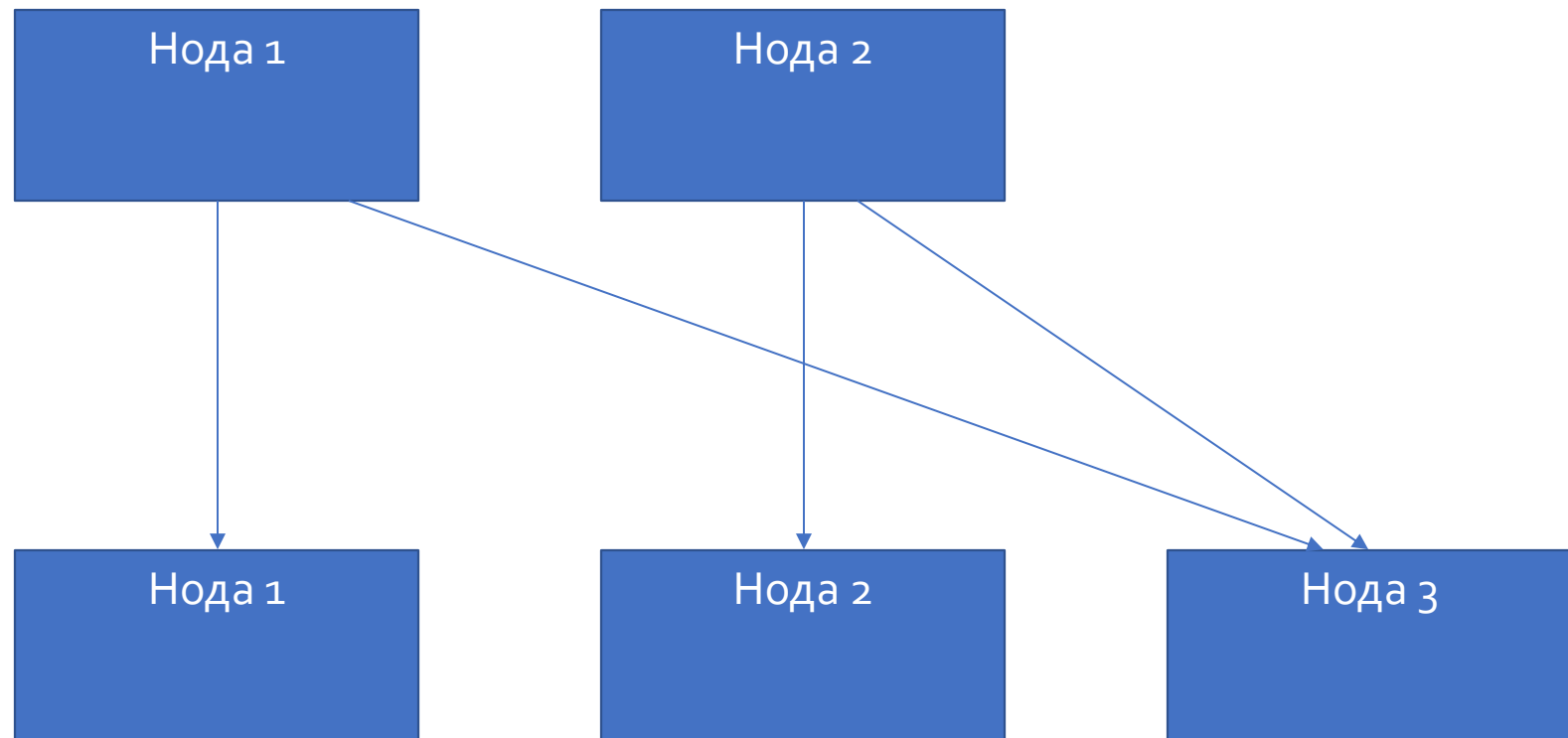
Рандеву хэширование

Rendezvous hash

1. Считаем несколько хэшей по числу нод
 1. $H_1 = \text{hash}(\text{key}, \text{node}_1)$
 2. $H_2 = \text{hash}(\text{key}, \text{node}_2)$
 3. $H_3 = \text{hash}(\text{key}, \text{node}_3)$
 4. $H = \max(H_1, H_2, H_3)$
2. Выбираем максимальный хэш и соответственно ноду
3. При добавлении новой ноды
 1. Если новый хэш больше – то данные переходят на новую ноду
 2. Если новый хэш меньше – то данные остаются на старой ноду

Таким образом удастся избежать лишних перемещений

Рандеву хэширование



Когда нужен шардинг?

- Много запросов на вставку данных
- БД не помещается на одном сервере
- Данные одной таблицы растут в X раз быстрее

Решения для шардинга

- PostgreSQL
 - pg_shard - расширение
 - Citus Extension - расширение
 - Pgouncer
- MySQL
 - Cluster NDB
 - ProxySQL
 - Vitess

Эффект «Леди Гаги»

<https://www.rbc.ru/society/17/07/2019/5d2f52af9a794718c7ace938>



Решение

- Нам надо чтобы все запросы к посту не попали на один shard.
- Мы сделаем составной ключ, который включает идентификаторы «писатель поста» и «читатель поста» и сделаем шардирование от hash от составного ключа.
- Минус решения – придётся размножить данные

Домашнее задание

1. Данные должны храниться в СУБД PostgreSQL;
2. Должны быть созданы таблицы для каждой сущности из вашего задания;
3. Должен быть создан скрипт по созданию базы данных и таблиц, а также наполнению СУБД тестовыми значениями;
4. Для сущности, должны быть созданы запросы к БД (CRUD) согласно ранее разработанной архитектуре
5. Данные о пользователе должны включать логин и пароль. Пароль должен храниться в закрытом виде (хэширован) – в этом задании опционально
6. Должно применяться индексирования по полям, по которым будет производиться поиск

На сегодня все

ddzuba@yandex.ru