Лекция 9

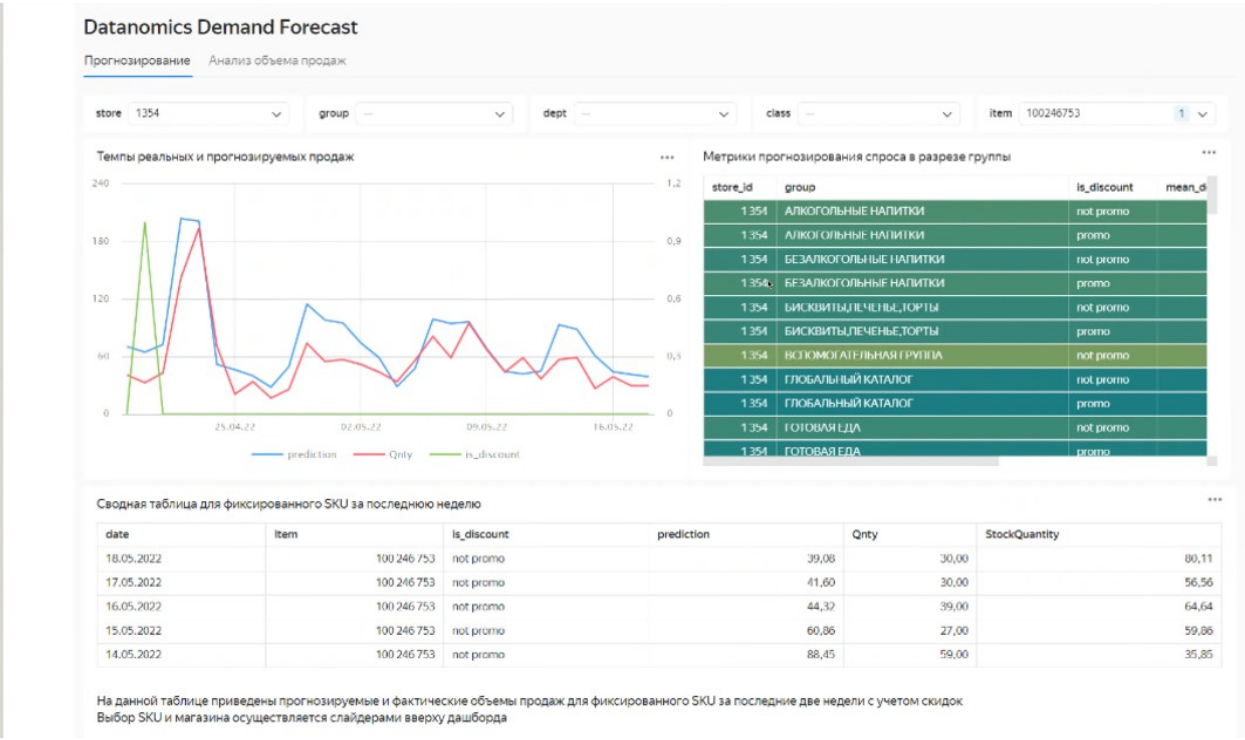
Преподаватель: Иван Ремень

Уровень данных и СУБД (Высокие нагрузки)

Содержание

Для чего бизнесу данные

Оперативные данные — данные которые видит пользователь сырые данные которые им видны. Пример: интернет блок. (Хабр: статьи — данные — метки — теги ← данные которые хранятся в оперативе, данные которые нужны)



ЧТО-то там за вопрос был (где-то 8 минута)

когда делаем бизнес, когда принимаем бизнес решения на основе данных, глубокие аналитика прошлого и предсказывание будущего. Не знаем правильные ли данные, дублирующие или нет … Часто эти данные не структурированые. Эти данные парсим, как-то обрабатываем.

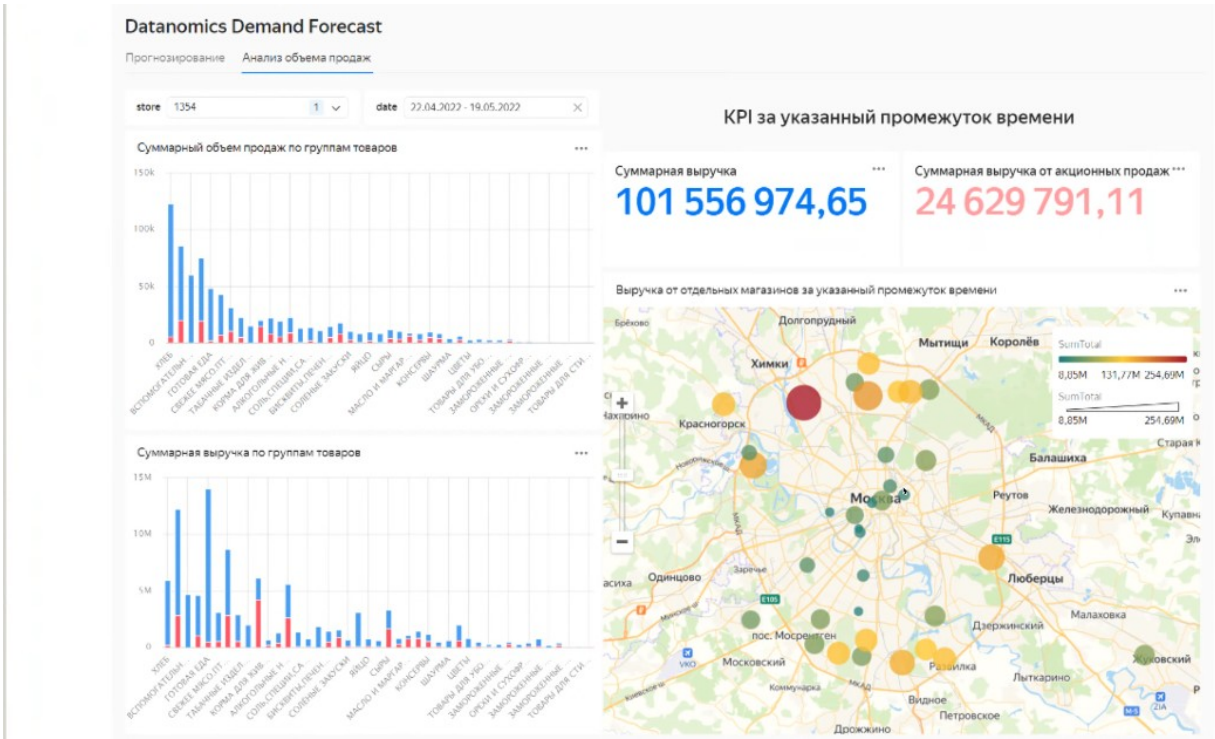
Проблема: данных много, для этого надо делать предагрегаты. на выходе получаем огромные пайплайн linage — династия(перевод). У нас может накапливается ошибка в данных… ОШИБКА <= это баги, где-то неправильно обработали что-то…

На основе этих данных можно дать сложный бизнес процесс. ЭТО ПРОИСХОДИТ в ДАТАОФИСЕ.

Все что сверху — это аналитика больших данных.

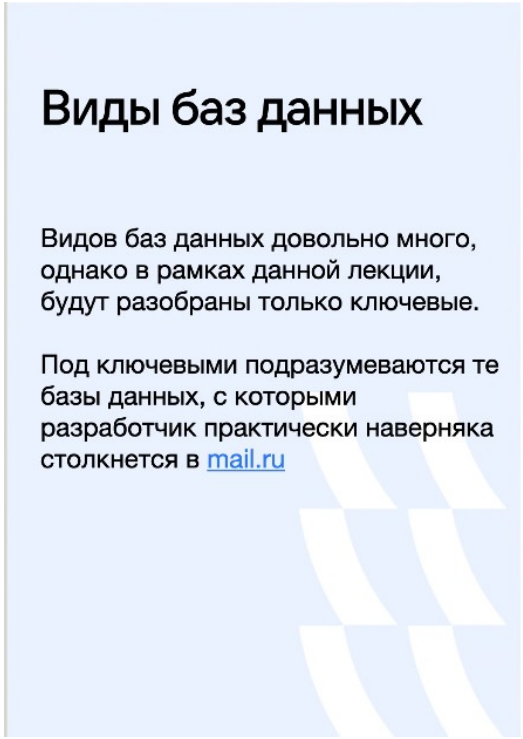
ВИДЫ ВАЖНЫХ ДАННЫХ:
OLTP -online transactuin process
OLAP — огрегатная статистика в онлайне
- **Не онлайн данные**, агрегаты, вокруг них обучаются модельки, строиться аналитика

ВИЗУАЛИЗАЦИЯ ДАННЫХ — это очень важная и сложная задача. Тепловые карты…



ВЫВОДЫ:
1. данные нужны для онлайн процесса
2. данные нужны для аналитики
3. принимаем решения основанные на данных
4. строим гипотезы и проверяем

ВИДЫ БАЗЫ ДАННЫХ



Множество видов

- Реляционные
- Ключ-значение
- Документо-ориентированные
- Базы данных временных рядов
- Графовые базы данных
- Поисковые базы данных (Search Engines)
- Объектно-ориентированные базы данных
- RDF (Resource Description Framework)
- Wide Column Stores
- Мультимодальные СУБД
- Native XML СУБД
- GEO/GIS (пространственные) и специализированные СУБД
- Event СУБД (баз данных переходов состояний)
- Контентные СУБД
- Навигационные (Navigational) СУБД
- Векторные базы данных

одно другого может не исключать…
- это набор характеристик. Одна БД может содержать несколько характеристик

РЕЛЯЦИОННЫЕ БД -
- построены на основе реляционной алгебре — она применялась непрерывно
- в середине NoSQL появляется
- NewSQL — сейчас появляется, учитываем минусы SQL

НЕДОСТАТКИ:
- это масштабируемость
- отсутствие класстерности

ДАЮТ:
- это возможность обеспечить ACID транзакции
- позволяют сделать JOIN (все беды из-за JOIN … JOIN отслеживают целостность реляционной БД)

KEY-VALUE — это кэширование Radis/ Tarantul / AreSpake / ManCash
В Python — dict
В них нет сложных индексов…

БАЗА данных временных рядов — БД работающая на основе времени. Показатель зависит от времени…

Примеры:
- это графики… Мониторинги
- бизнес показатели
- датчикиа

Ищим по перцентилю…
Скользящее окно…

Графовые БД

- очень старые и часто не использовались…
- она связывает некоторые узлы через набор факты

вершины — это факты
ребра — связи фактов

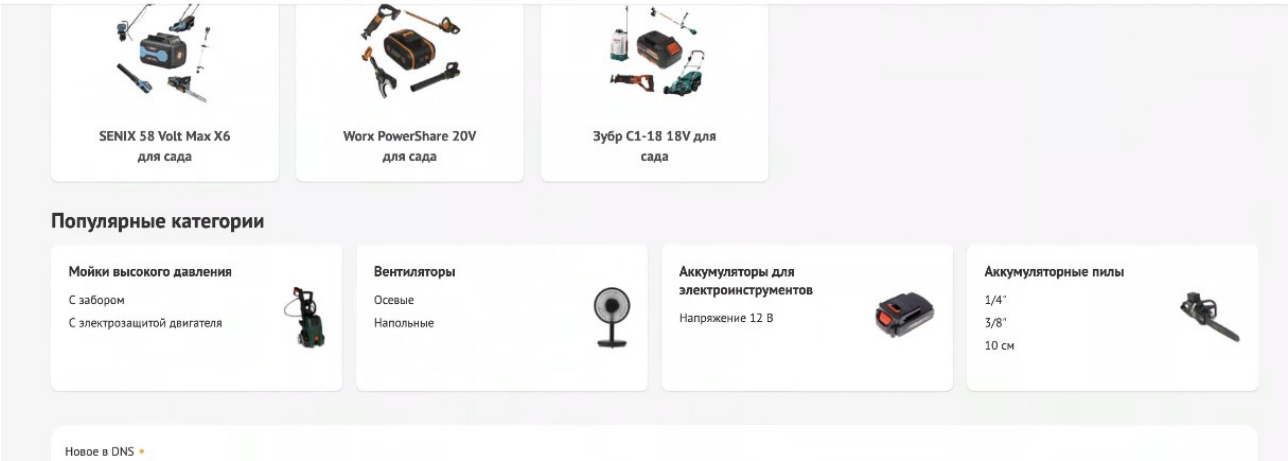
Почему она стала более популярной в наше время?
- В ИИ появилась: в рекомендательных системах — набор фактов удобно хранить … В **RAC** используется.

- **Документо-ориентированные и поисковые БД**

MONGO ← плохо структурированные данные хорошо обрабатывает и легко масштабируется … / сфинкс или ELASTIC-SEARCH

- семантика
- индексы

КАТАЛОГИ: - лучше делать на ELASTICe < - идеально подходит...:



RDF — ормат описания мета данных
В таблице находятся что-то и проще ищутся

Wide Column Stores — google big table и прочее. BigTableDB/Касандра

Click house супер популярная БД от яндекса

Мультимодальные СУБД - это просто СУБД хранящая разные типы данных, аудио, видео

NATIVE XML СУБД- хранит просто XML

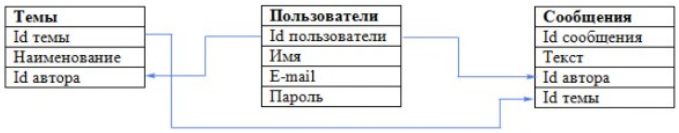
GEO/GIS — хранит точки и поддерживает индексы на геоданных и может вернуть точки которые лежат возле другой точки в каком-то радиусе

EVENT СУБД
и т. д. Он не рассказал про них...

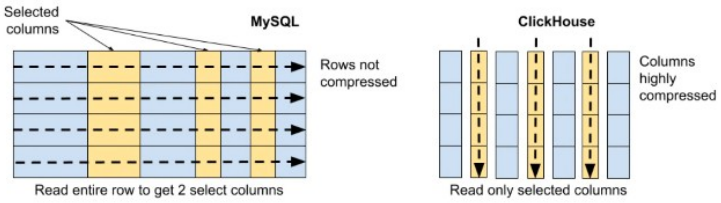
ВЕКТОРНЫЕ БД — там где ключ это вектор и ввозвращает топ N соседей по расстоянию косинуса. Для ИИ

Реляционные базы данных

Наиболее известными реляционными базами данных являются Open Source проекты PostgreSQL, MySQL и SQLite, а также проприетарные решения Oracle, Microsoft SQL Server и IBM Db2Relational.



Стоит заметить, что реляционные базы бывают с хранением данных по строкам (PostgreSQL) и по столбцам/колонкам (ClickHouse, Vertica). Колоночные/столбцовые базы лучше подходят для аналитики, в то время как ориентация на строки лучше подходит для транзакционных нагрузок.



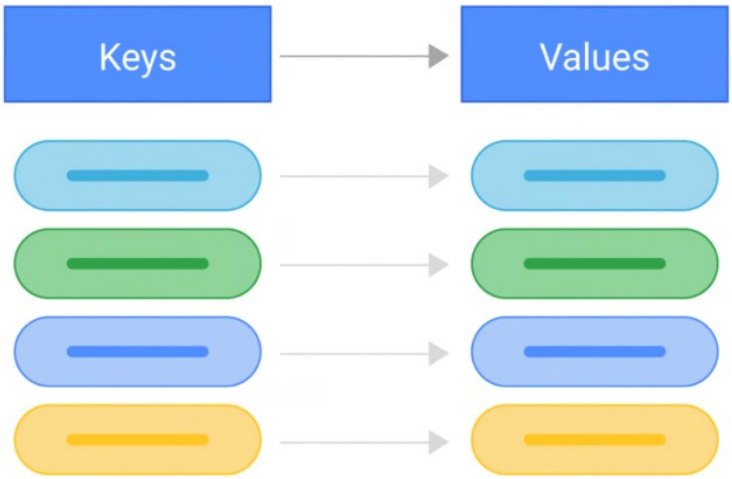
Представьте есть файл, в котором идут строчка за строчкой всё (**MYSQL**)— **это строчная** - данные подряд на диске храняться

А в **колоночной (CLICKHOUSE)**— сначала пишется один столбец, а потом другой…

Key-Value базы данных

Тип баз данных Key-value предназначен для осуществления быстрых, почти мгновенных запросов для таких задач как кэш, отображение баланса и т.д.. Высокая скорость осуществляется за счет хранения данных по принципу ключ-значение, и в большинстве случаев благодаря работе в оперативной памяти.

В mail.ru вы столкнетесь с Redis и Tarantool

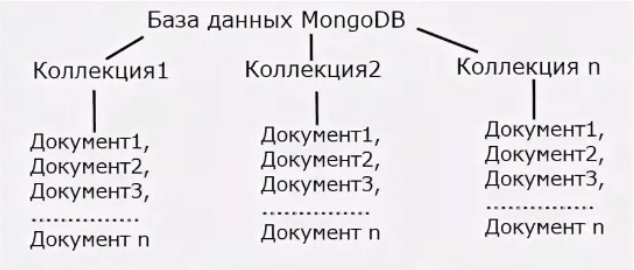


Документы ориентированные базы данных

Если вам нужно хранить много файлов/документов и вы не хотите задумываться (до разумных пределов, разумеется) о структуре хранения, иерархии, связях, вам может подойти одна из документо-ориентированных баз данных. Дополнительным преимуществом являются широкие возможности для масштабирования.

В некоторых бизнес юнитах вы столкнетесь с MongoDB

Документо-ориентированные базы данных созданы для хранения иерархических структур данных (документов). Основой документоориентированных СУБД являются документные хранилища, имеющие структуру дерева или леса. Деревья начинаются с корневого узла и может содержать несколько внутренних и листовых узлов. Листовые узлы содержат данные, которые при добавлении документа заносятся в индексы, это дает возможность даже при достаточно сложной структуре находить путь к искомых данных.



MONGODB — все меньше и меньше начало использоваться

НОРМАЛИЗАЦИЯ ДАННЫХ

Нормализация данных

Нормализация — это способ организации данных. В нормализованной базе нет повторяющихся данных, с ней проще работать и можно менять её структуру для разных задач. В процессе нормализации данные преобразуют, чтобы они занимали меньше места, а поиск по элементам был быстрым и результативным.

Код сотрудника	ФИО	Должность	Номер отдела	Наименование отдела	Квалификация
7513	Иванов И.И.	Программист	128	Отдел проектирования	C, Java
9842	Сергеева С.С.	Администратор БД	42	Финансовый отдел	DB2
6651	Петров П.П.	Программист	128	Отдел проектирования	VB, Java
9006	Николаев Н.Н.	Системный администратор	128	Отдел проектирования	Windows, Linux

Необработанные данные

Нет **аномалий** в данных

Важность нормализации

Правильно применяя принципы нормализации, вы можете создать схему базы данных, которую будет легче поддерживать, обновлять и запрашивать.

Хорошо нормализованная схема базы данных дает большие преимущества

1. **Целостность данных.** Обеспечение согласованности данных в базе данных является основной целью нормализации. Вы можете обеспечить целостность данных во всей системе базы данных, устранив избыточность и несогласованность данных.
2. **Повышение эффективности обслуживания и обновления.** Ненормализованную базу данных может быть сложно обновлять и поддерживать, что приводит к увеличению вероятности ошибок при изменении или удалении данных. Нормализация упрощает процесс обновления и снижает риск ошибок.
3. **Оптимизация места для хранения.** Нормализация уменьшает избыточность данных за счет устранения дублирующихся данных, тем самым уменьшая необходимое пространство для хранения и повышая эффективность базы данных.
4. **Повышение производительности запросов.** Хорошо структурированная база данных часто приводит к повышению производительности запросов, поскольку более простая схема позволяет более эффективно обрабатывать и оптимизировать запросы.
5. **Легче понимать и управлять.** Нормализованные базы данных легче понимать и управлять ими благодаря их согласованной структуре и уменьшенной избыточности данных. Это упрощает эффективную работу с системой разработчикам и администраторам баз данных.

Уровни нормализации

Существует пять основных уровней нормализации, известных как нормальные формы (NF), каждый из которых решает различные проблемы при проектировании базы данных и накладывает дополнительные ограничения на схему.

1. **Первая нормальная форма (1НФ):** таблица находится в 1НФ, если она не содержит повторяющихся групп или повторяющихся столбцов для одного значения ключа. Каждое значение столбца должно быть атомарным, то есть его нельзя разложить дальше. Такая форма упрощает структуру таблицы и облегчает хранение и поиск данных.
2. **Вторая нормальная форма (2НФ):** таблица находится в 2НФ, если она находится в 1НФ и все ее неключевые столбцы полностью функционально зависят от первичного ключа. Это означает, что значение первичного ключа определяет значение каждого неключевого столбца. 2NF гарантирует отсутствие частичной зависимости внутри структуры таблицы и дополнительно снижает избыточность данных.
3. **Третья нормальная форма (3НФ):** таблица находится в 3НФ, если она находится в 2НФ; все его неключевые столбцы не зависят транзитивно от первичного ключа. Другими словами, ни один неключевой столбец не должен зависеть от других неключевых столбцов, определяемых первичным ключом. 3NF устраняет транзитивные зависимости, повышая эффективность и согласованность данных.
4. **Нормальная форма Бойса-Кодда (BCNF):** таблица находится в BCNF, если она находится в 3NF, и каждый определитель (набор столбцов, который однозначно определяет другие столбцы) является потенциальным ключом. BCNF — это более сильная форма 3NF, которая устраняет аномалии в некоторых таблицах 3NF. Это устраняет избыточность и потенциальные несоответствия из-за перекрытия потенциальных ключей.
5. **Четвертая нормальная форма (4НФ):** таблица находится в 4НФ, если она находится в BCNF и нет многозначных зависимостей. Это означает, что таблицу с более чем одним независимым многозначным атрибутом следует разложить на отдельные таблицы. 4NF решает проблемы избыточности и несогласованности данных, связанные с многозначными зависимостями.

1НФ - когда нет массивов и JSON-ов
2НФ — все зависит от первичного ключа
3НФ — все вторичные столбцы не зависят транзитивно от первичного включа

ACID введение

Акроним ACID описывает требования к транзакционной системе, обеспечивающие наиболее надежную и предсказуемую её работу.

Atomicity — Атомарность

Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной.

Isolation — Изолированность

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат. Изолированность — требование дорогое, поэтому в реальных БД существуют режимы, не полностью изолирующие транзакцию (уровни изолированности Repeatable Read и ниже).

Consistency — Согласованность

Транзакция, достигающая своего нормального завершения (EOT — end of transaction, завершение транзакции) и, тем самым, фиксирующая свои результаты, сохраняет согласованность базы данных. Другими словами, каждая успешная транзакция по определению фиксирует только допустимые результаты.

Durability — Устойчивость

Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу.

Атамарнность — либо пройдет либо не пройдет тарнзакция
Согласованность — после commit -а все данные придут в правильную сторону
Устойчивость — если зафиксировали транзакцию, то нельзя откатиться назад

Изолированность — типо строим транзакции параллельно… Освобождают от разного вида аномалий.

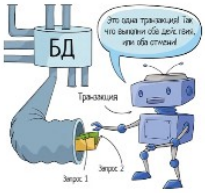
Атомарность. Наглядно

Отправка денег со счета на счет. [1]

- Если деньги есть на счету и счет не блокирован то транзакция завершиться успешно

- Если нет денег, транзакция неуспешна

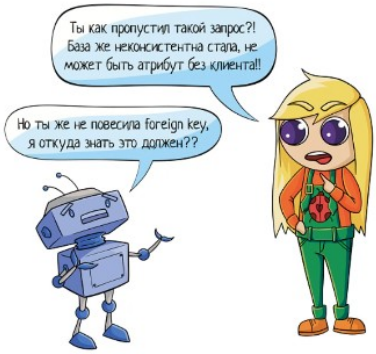
- Счет второго участника, заблокирован. Критично, нужно вернуть деньги на счет



Консистентность. Наглядно

Это свойство вытекает из предыдущего. Благодаря тому, что транзакция не допускает промежуточных результатов, база остается консистентной.

Есть такое определение транзакции: «Упорядоченное множество операций, **переводящих базу данных из одного согласованного состояния в другое**». То есть до выполнения операции и после база остается согласованной



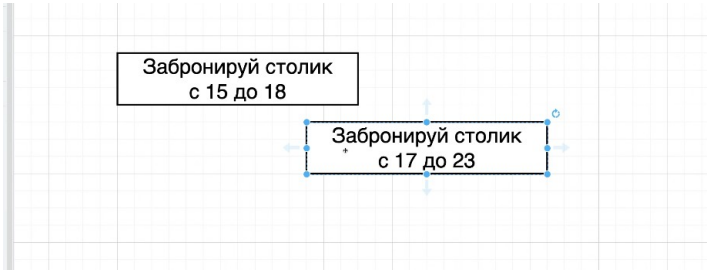
Изолированность. Наглядно

Во время выполнения транзакции параллельные транзакции не должны оказывать влияния на её результат. Если у нас система строго для одного человека, проблем не будет.

А если пользователей несколько?

Тогда транзакции запускают в параллель — для ускорения работы системы. А иначе представьте себе, что вы делаете заказ в интернет-магазине и система вам говорит: «Вы в очереди, перед вами еще 100 человек хотят заказ оформить, подождите».

Фантомы — бронь ресторана: чтобы не было пересечений…



Решение проблемы — бронь окнами…
Должно быть четкое понимание, что нельзя допустить дыры, не буден забронирован столик с пересечением.

Фантом- это когда читаем данные, а потом чел добавил что-то и не корректные данные будут у того кто читал эти данные…

Уровни изолирования транзакций

Выбирая уровень транзакции, мы пытаемся прийти к консенсусу в выборе между высокой согласованностью данных между транзакциями и скоростью выполнения этих самых транзакций. Стоит отметить, что самую высокую скорость выполнения и самую низкую согласованность имеет уровень **read uncommitted**. Самую низкую скорость выполнения и самую высокую согласованность — **serializable**.

1. **Read uncommitted**

   Уровень, имеющий самую плохую согласованность данных, но самую высокую скорость выполнения транзакций. Название уровня говорит само за себя — каждая транзакция видит незафиксированные изменения другой транзакции (феномен **грязного чтения**). Посмотрим какое влияние оказывают друг на друга такие транзакции.

2. **Read committed**

   Для этого уровня параллельно исполняющиеся транзакции видят только зафиксированные изменения из других транзакций. Таким образом, данный уровень обеспечивает защиту от **грязного чтения**.

3. **Repeatable read**

   Уровень, позволяющий предотвратить феномен **неповторяющегося чтения**. Т.е. мы не видим в исполняющейся транзакции измененные и удаленные записи другой транзакцией. Но все еще видим вставленные записи из другой транзакции. **Чтение фантомов** никуда не уходит.

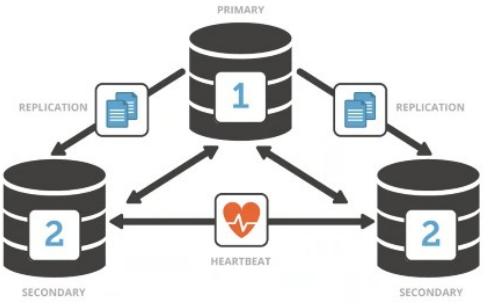
4. **Serializable**

   Уровень, при котором транзакции ведут себя как будто ничего более не существует, никакого влияния друг на друга нет. В классическом представлении этот уровень избавляет от эффекта **чтения фантомов**.

Репликация данных

Среди задач, решаемых репликацией, можно назвать как минимум

- поддержку резервной базы данных на случай потери основной;
- снижение нагрузки на базу за счёт переноса части запросов на реплики;
- перенос данных в архивные или аналитические системы.



1. Блочная репликация на уровне системы хранения данных
2. Физическая репликация на уровне СУБД
3. Логическая репликация на уровне СУБД

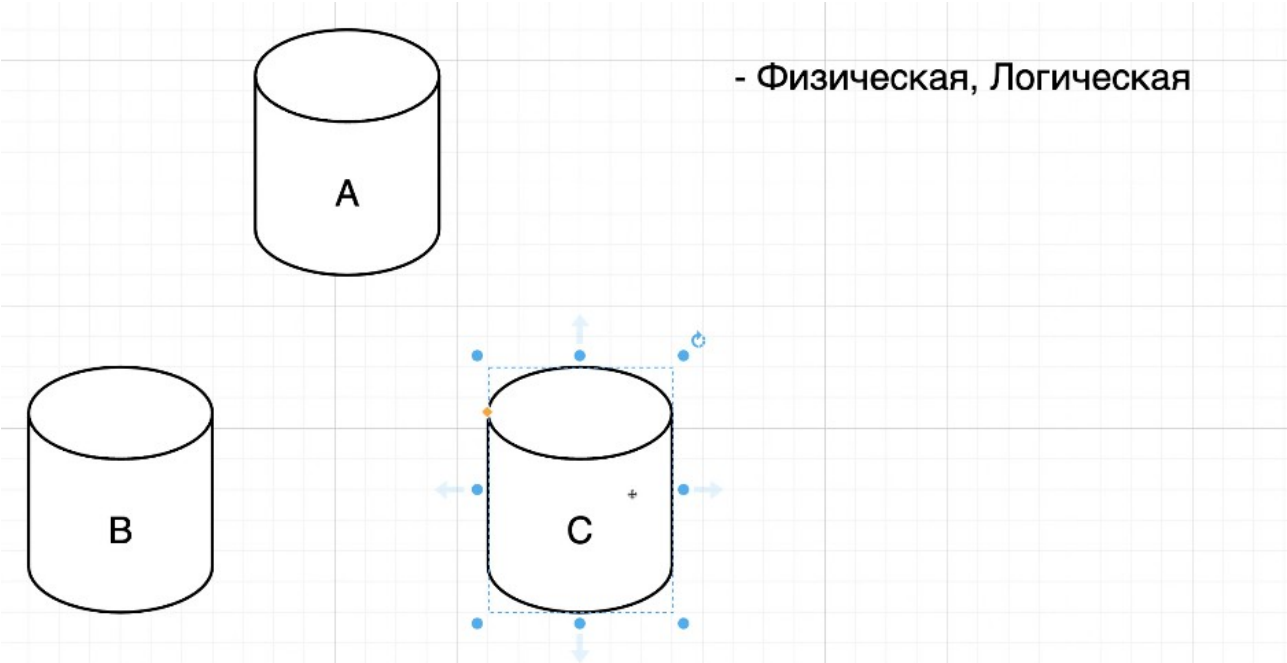
МАСШТАБИРУЕТ ЛИ МАСТЕР МАСТЕР ЗАПИСЬ — это репликация … Нет. ТУДА И ТУДА НУЖНО ЗАПИСАТЬ

Отказаустойчивость дает репликация

ШАРДИРОВАНИЕ — несколько серверов в БД, четные лежат на одном , а не четные на другом

ПРАКТИКА:

Журнал репликации есть — может лежать то как поменялась страница… (НА ЖЕСТКОМ ДИСКЕ), либо **логическая**, когда колонка была такой, а стала такой
ФИЩИЧЕСКАЯ из бита в бит ...



Виды репликации:

Асинхронная — есть пользователь написал на мастер что-то и получает потом комит, сразу после комита нет гарантии, что данные на B и C дошли данные

Синхронная - есть пользователь написал на мастер что-то и получает потом комит, только после того, как в некоторое количество реплик пришла запись. (Есть настройка кол-ва реплик)

Полусинхронная - это когда журнал транзакций был скопирован, хотя бы на одну реплику, это означает что современем записи востановятся в этой реплике. (Допустим наш Master нода вылетела… И этот журнал транзакции востоновит все современем) Нет потерь записи.
ВНИМАНИЕ: именно журнал доехал, нет гарантии, что мы это прочитаем. Сразу, есть гарантия, что мы ее когда-нибудь прочитаем.

Борьба: после записи, данные читаются только с мастера некоторое время
Реплики имеют свойство отставать

Master - сервер, который принимает запись, master может быть больше, чем один. Сам master потом записывает в B и C записи.

Продолжаем виды репликации:

master/master репликация: записи в несколько узлов. SRDT — алгоритм. Т.е. у нас два сервака, которые принимают данные и доступны для чтения. Они современем потом синхранизируются.

Пример:

1. применяется git, т. к. у каждого локальные правки… И потом пушим на центральный сервер, где могут разъехаться данные. Т.е. происходит синхранизация между разными данными
2. Google dock

master/slave репликация:
- это когда есть один мастер сервер и друние реплики берут с него данные .

masterless репликация:
- это когда нет какого мастер сервера