13. Масштабирование БД

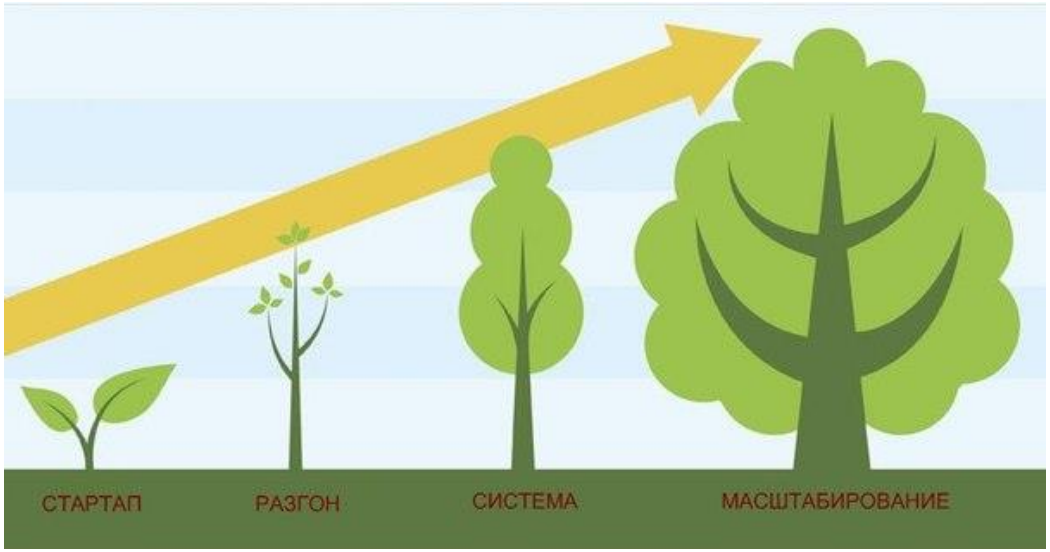
к.т.н., доцент кафедры ИиСП

Лучинин
Захар Сергеевич

Масштабирование

Масштабирование — это процесс изменения размера системы

Масштабируемость — это свойство системы, сохраняя пропускную способность, справляться с увеличением нагрузки при увеличении определенных ресурсов системы



Необходимость в масштабировании

- Ограничение пропускной способности чтения данных
- Ограничение пропускной способности записи данных

Виды масштабирования систем

- **Вертикальное масштабирование** – увеличение производительности узла

- **Горизонтальное масштабирование** – увеличение количества узлов

Вертикальное масштабирование

- Увеличение объема оперативной памяти
- Наращивание мощности процессора
- Усиление дисковой подсистемы

Масштабируемость в этом контексте означает **возможность заменять** в существующей вычислительной системе компоненты более мощными и быстрыми

Самый простой способ масштабирования, так как не требует никаких изменений в прикладных программах.

Горизонтальное масштабирование

- Горизонтальное масштабирование — разбиение системы на более мелкие структурные компоненты и разнесение их по отдельным физическим машинам

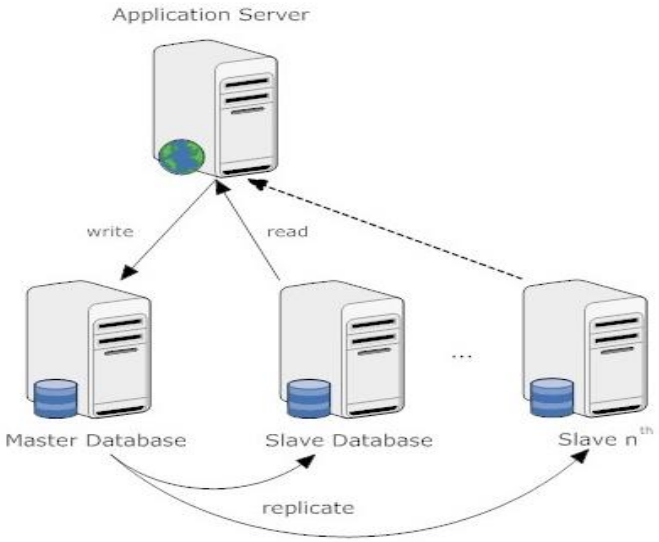
Масштабируемость в этом контексте означает **возможность добавлять к системе новые узлы**, серверы, процессоры для увеличения общей производительности.

Горизонтальное масштабирование БД

- Репликация
- Секционирование
- Шардинг

Репликация

- **Репликация** — механизм синхронизации содержимого нескольких копий объекта (например, содержимого базы данных).
- Наращиваемое решение. Если одного сервера не хватает — ставится второй, третий и т.д.



Когда использовать?

Распространение данных

Репликация позволяет создать копию базы данных у географически удаленном пункте, например в другом центре обработки данных.

Балансировка нагрузки

С помощью репликации можно распределить запросы на чтение между несколькими серверами. В приложениях с интенсивным чтением эта тактика работает очень хорошо.

Реализовать несложное балансирование нагрузки можно, внеся совсем немного изменений в код.

Почему использовать?

- Распространение данных
- Балансировка нагрузки
- Резервное копирование
- Аварийное переключение на резервный сервер (failover)
- Тестирование новых версий СУБД

Роли серверов в репликации

- **Мастер сервер** - основной сервер базы данных. На нем происходят все изменения в данных (любые запросы MySQL INSERT/UPDATE/DELETE).

- **Слейв сервер** – сервер, который постоянно копирует все изменения с Мастера.

Виды репликации

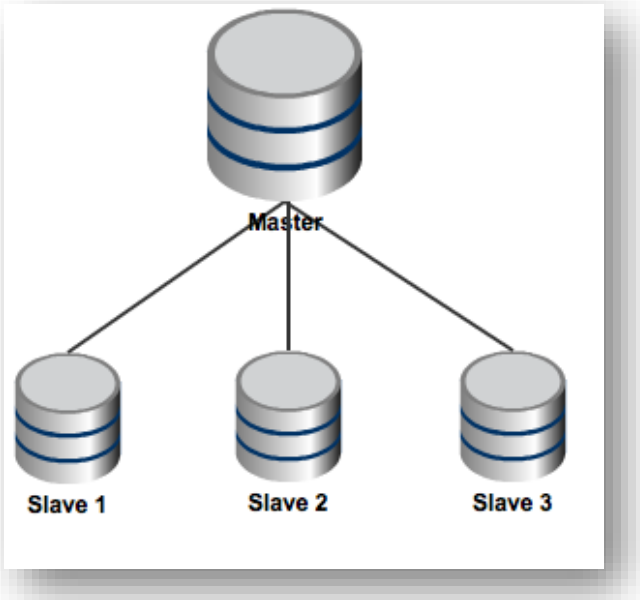
- **Master-slave репликация**: основной сервер БД доступен на чтение и запись, реплика доступна для чтения либо недоступна вообще. Данные на мастере и реплике идентичны.

- **Master-master репликация**: все сервера являются доступными для записи, а измененные данные распространяются на весь кластер

Репликация Мастер-Слейв

1 мастер, много слейвов

Схема обычно используется в приложениях с доминирующими запросами на чтение - все запросы на изменение базы направляются мастер серверу, тогда как запросы на чтение распределяются между слейвами.



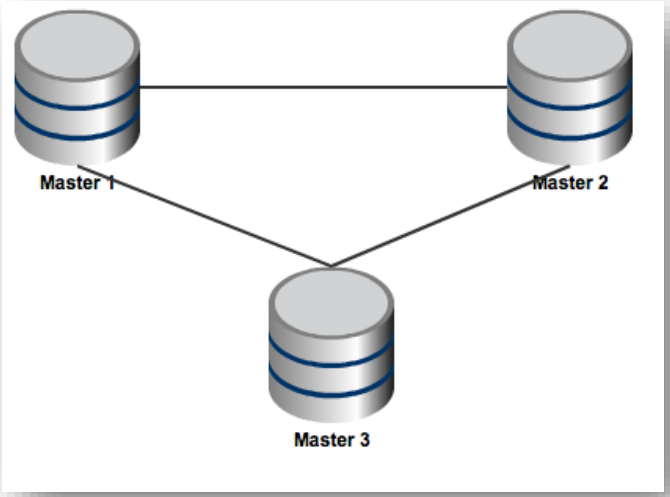
Недостаток

При выходе из строя мастер сервера, все запросы на модификацию и добавление данных не будут выполняться.

Репликация Мастер-Мастер

Цепочка мастер серверов

Очень удобна для географического распределения данных. Например, ставим сервера в Европе, Азии и Америке, настраиваем их в цепочку и учим приложение выбирать сервер в зависимости от региона. Таким образом получаем выигрыш за счет уменьшения пути путешествия запроса к серверу.



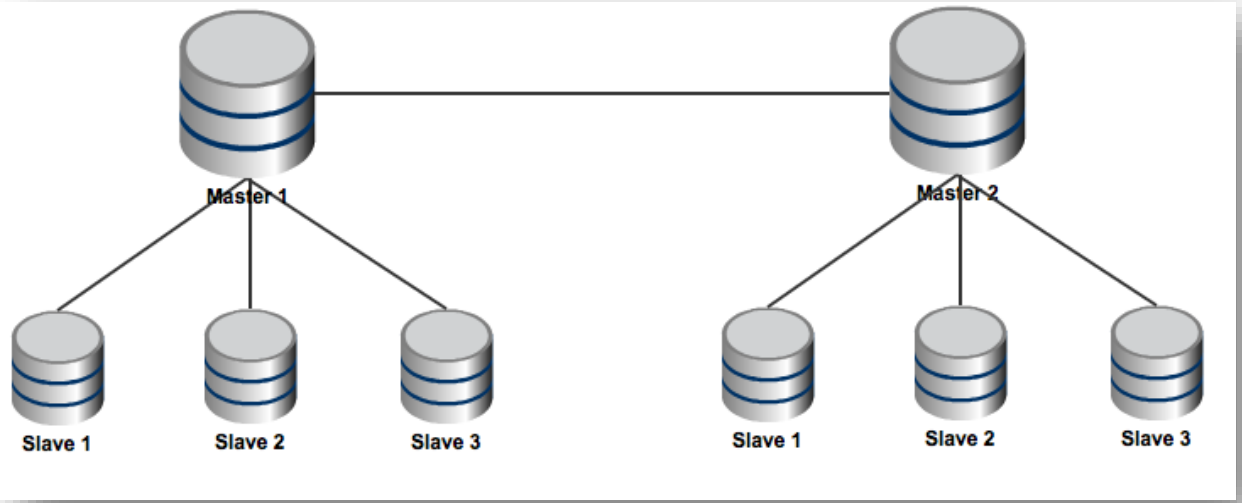
Недостаток

- Сложность реализации.
- Не многие СУБД поддерживают такую схему.
- Нужно продумывать обработку конфликтов

Репликация Мастер-Мастер

2 мастера, много слейвов

Схема является цепочкой из двух мастер серверов. Оба мастера выполняют запросы на модификацию данных и имеют равное количество слейвов. Таким образом при выходе из строя одного мастера, приложение продолжит работать со вторым и его слейвами.



Гарантии репликации

Синхронная

Мастер-сервер не подтверждает транзакцию до того, как реплика не подтвердит получение данных.

Асинхронная

Мастер-сервер не ждёт подтвержения получения данных от реплики.

Majority

Мастер-сервер ждёт подтвержения получения данных от N-реплик.

Семисинхронная (MySQL)

Мастер-сервер не подтверждает транзакцию до того, как "живые" реплики не подтвердят получение данных.

Синхронная репликация

Синхронная репликация – при изменении состояния одной реплики все остальные должны быть обновлены в одной и той же транзакции.

Все операции чтения можно всегда отправлять на слейв. Однако это может значительно уменьшить скорость работы СУБД.

Асинхронная репликация

Асинхронная репликация – при изменении состояния одной реплики все остальные обновляются спустя некоторое время, т.е. асинхронная репликация не гарантирует, что на всех репликах в одно и то же время содержатся одинаковые данные, доступные для чтения

Так как мастер не ждет обновления слейва, то запросы на изменения выполняются **быстрее**

Варианты реализации

Физическая

Передаётся информация о физическом изменении страниц базы данных.

Логическая

Передаётся информация об изменении записей базы данных.

Физическая репликация

Общий принцип:

- Главный сервер записывает изменения данных в журнал транзакций;
- Подчиненный сервер копирует события журнала транзакций;
- Подчиненный сервер воспроизводит изменения из журнала транзакций.

Физическая репликация

Плюсы:

- Простота и надёжность;
- Подчиненный сервер в точности соответствует мастер-серверу;
- Практически отсутствуют накладные расходы.

Минусы:

- Если данные на мастере были испорчены из-за сбоев RAM, то на подчинённом сервере так же будут испорченные данные;
- На реплике не может быть локальных изменений схемы данных;
- Не возможна мастер-мастер репликация;

Логическая репликация

Общий принцип:

- Мастер сервер записывает изменения данных в журнал транзакций;
- На базе журнала транзакций мастер сервер восстанавливает информация об изменении записей;
- Данные об изменении записей передаются на подчиненный сервер.

Логическая репликация

Плюсы:

- Более компактный обмен данными;
- Если данные на мастере были испорчены из-за сбоев RAM, то репликация остановится;
- На мастере и подчинённом сервере можно использовать разную схему данных;

Минусы:

- Более высокая нагрузка на подчинённый сервер;
- Нет хорошего решения проблемы репликации DDL-запросов.

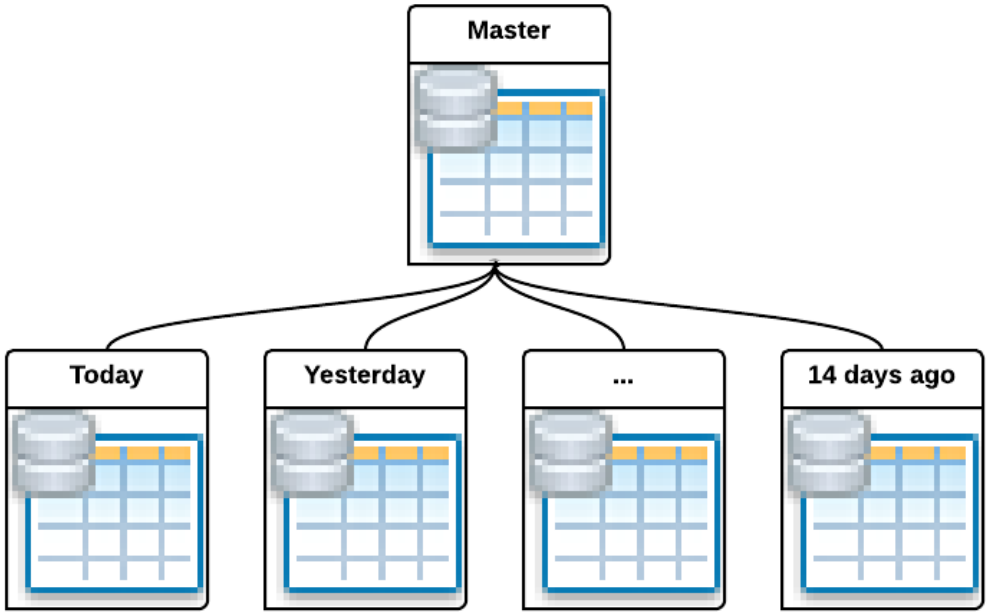
Преимущества и недостатки

+ Решает проблему нагрузки.
+ Имеем копию нашей БД

- При большом количестве слейвов усложняется схема распространения изменений, которая в дальнейшем становится узким местом.
- Усложнение программной архитектуры – например, чтение данных с слейва, до которого не докатились изменения.

Секционирование

- Секционирование (partitioning) - это разбиение больших таблиц на логические части по выбранным критериям.



Секционирование

Позволяет:

- Отделить статические данные от изменяющихся
- Воспользоваться физической близостью взаимосвязанных данных для оптимизации запросов
- Проектировать таблицы так, чтобы запрос обращался к возможно меньшему объему данных
- Упростить обслуживание очень больших наборов данных
- Размещать партиции на разных физических дисках
- Редко используемые архивные данные можно разместить на медленном хранилище
- Уменьшить размер индексов

Варианты секционирования

- По диапазону значений
- По точному списку значений
- По HASH функции
- По ключу

Пример секционирования

```
-- создание функции секционирования
CREATE PARTITION FUNCTION part_func (int)
AS RANGE LEFT FOR VALUES (100, 200, 300);

-- мапинг файловых групп к функции
CREATE PARTITION SCHEME part_sch_test
AS PARTITION part_func
TO (fg1, fg2, fg3, fg_default);

-- создание секционированной таблицы
CREATE TABLE [dbo].[issuance](
   [id_issuance] [int] IDENTITY(1,1) NOT NULL,
   [id_copy] [int] NOT NULL, [id_reader] [int] NOT NULL,
   [issue_date] [date] NOT NULL, [release_date] [date] NOT NULL,
   [deadline_date] [date] NOT NULL,
 CONSTRAINT [PK_issuance] PRIMARY KEY CLUSTERED
(
[id_issuance] ASC
)
) ON part_func ([id_reader])
```

Секционирование по дате

Достоинства:

- легко понять;
- количество строк в данной таблице будет достаточно стабильным;

Недостатки:

- требует поддержки – время от времени нам придётся добавлять новые партиции;
- поиск по имени пользователя или id потребует сканирования всех партиций;

Секционирование по id

Достоинства:

- легко понять;
- количество строк в партиции будет на 100% стабильным;

Недостатки:

- требует поддержки – время от времени нам придётся добавлять новые партиции;
- поиск по имени пользователя потребует сканирования всех партиций;

По первой букве имени

Достоинства:

- легко понять;
- никакой поддержки – есть строго определенный набор партиций и нам никогда не придется добавлять новые;

Недостатки:

- количество строк в партициях будет стабильно расти;
- в некоторых партициях будет существенно больше строк, чем в других (больше людей с никами, начинающимися на “t*", чем на “y*");
- поиск по id потребует сканирования всех партиций;

По хэшу имени пользователя

Достоинства:

- никакой поддержки – есть строго определенный набор партиций и нам никогда не придется добавлять новые;
- строки будут равно распределяться между партициями;

Недостатки:

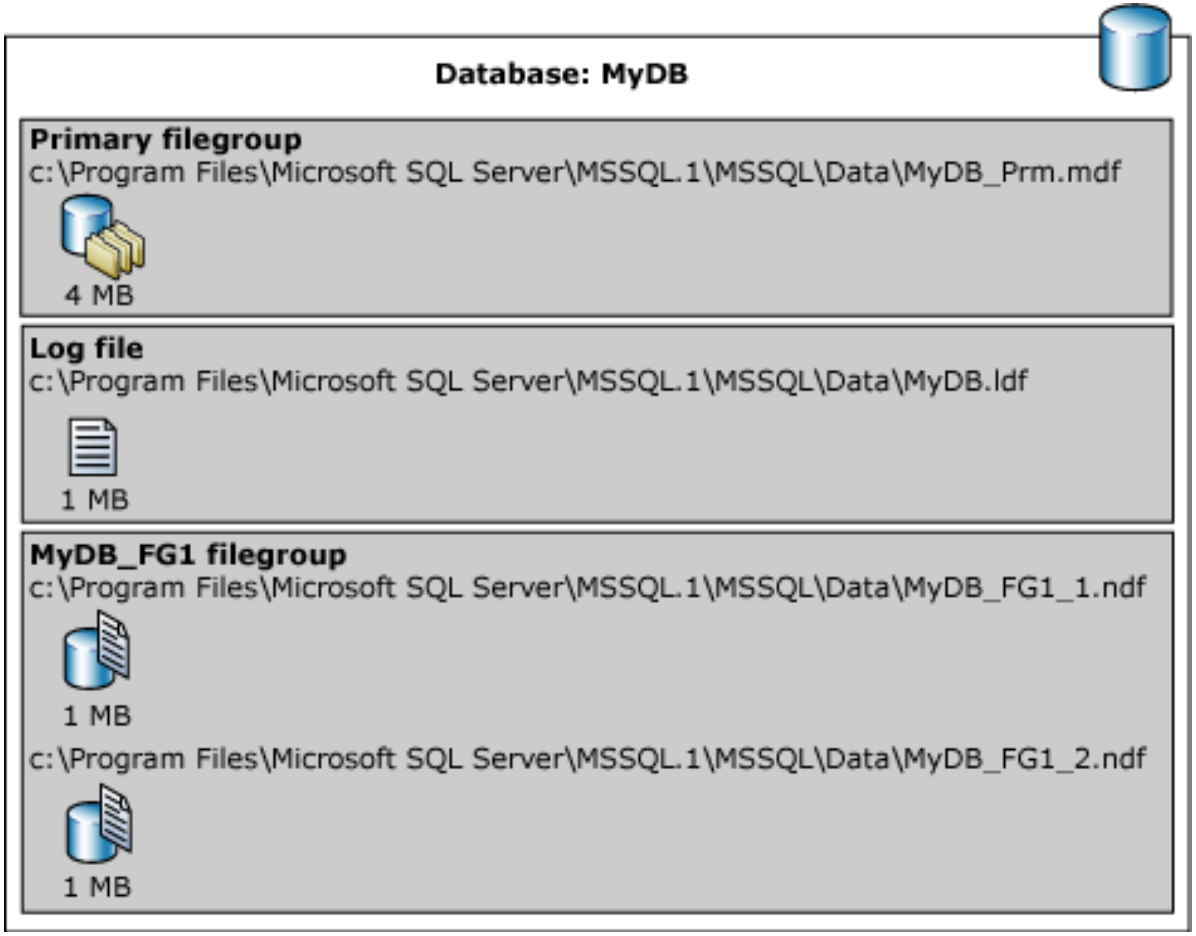
- количество строк в партициях будет стабильно расти;
- поиск по id потребует сканирования всех партиций;
- поиск по имени пользователя будет сканировать только одну партицию, но только при использовании дополнительных условий.

Файлы и файловые группы

- Файловые группы позволяют распределить хранение таблиц/баз данных по разным физическим дискам.

- Применяется с целью избежать ограничений скорости хранилища данных.

Файлы и файловые группы



Шардинг

Шардинг – это разделение данных на уровне ресурсов, в результате которого данные размещаются на различных серверах исходя из требований к нагрузке

Вертикальный шардинг

Вертикальный шардинг — это выделение таблицы или группы таблиц на отдельный сервер. Например, в приложении есть такие таблицы:

- user — данные пользователей
- photo — фотографии пользователей
- album — альбомы пользователей

Таблицу user Вы оставляете на одном сервере, а таблицы photo и album переносите на другой. В таком случае в приложении Вам необходимо будет использовать соответствующее соединение для работы с каждой таблицей

Горизонтальный шардинг

Горизонтальный шардинг — это разделение одной таблицы на разные сервера. Это необходимо использовать для огромных таблиц, которые не умещаются на одном сервере. Разделение таблицы на куски делается по такому принципу:

- На нескольких серверах создается одна и та же таблица (только структура, без данных).
- В приложении выбирается условие, по которому будет определяться нужное соединение (например, четные на один сервер, а нечетные — на другой).
- Перед каждым обращением к таблице происходит выбор нужного соединения.

Критерии шардинга

Какой-то параметр, который позволит определять, на каком именно сервере лежат те или иные данные.

ID поля таблицы

Хеш-таблица с соответствиями «пользователь=сервер»

(Тогда, при определении сервера, нужно будет выбрать сервер из этой таблицы. В этом случае узкое место — это большая таблица соответсвия, которую *нужно* хранить в одном месте.)

Определять имя сервера с помощью числового (буквенного) преобразования.

(Например, можно вычислять номер сервера, как остаток от деления на определенное число (количество серверов, между которыми Вы делите таблицу). В этом случае узкое место — это проблема добавления новых серверов — Вам придется делать перераспределение данных между новым количеством серверов.)

Преимущества шардинга

- Единственный способ добиться увеличения пропускной способности записи
- Масштабирование используя десятки, сотни и тысячи узлов баз данных, построенных на недорогом оборудовании вместо использования дорогих и высокопроизводительных серверов
- Достижение нужной производительности путем увеличения количества узлов
- Построение относительно дешевых решений с большим объемом данных
- При полном выходе из строя одного шарда безвозвратно потеряется только часть данных

Проблемы шардинга

- Поддержка ссылочной целостности
- Объединения таблиц между шардами
- Генерация эффективного идентификатора для шардинга
- Определение, в какой шард попадают данные
- Перераспределение шардов
- Сортировка и агрегация данных между шардами
- Управление соединениями

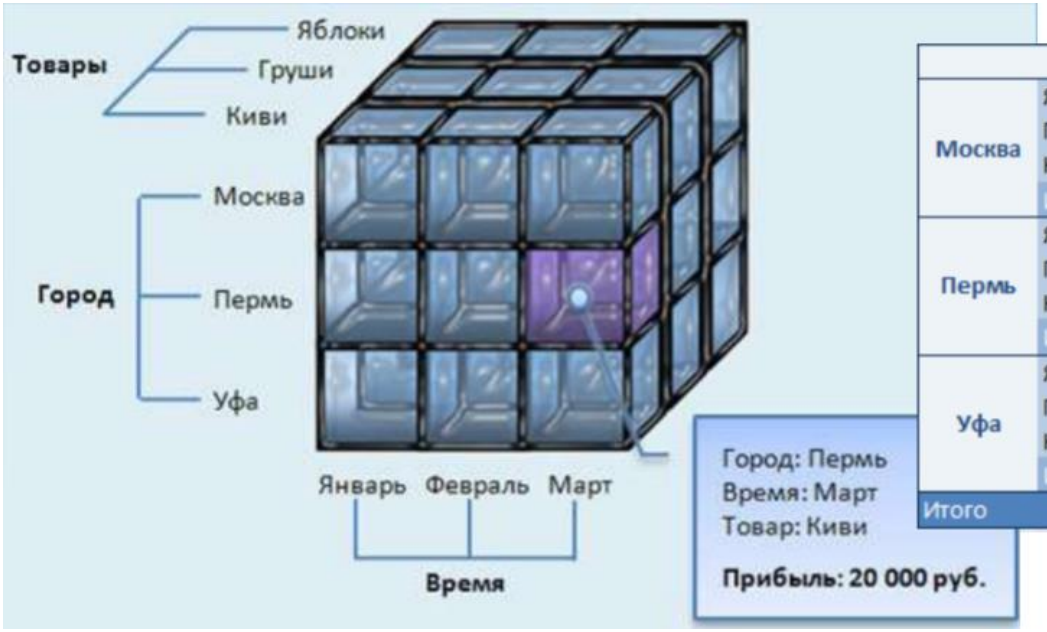
Системы обработки данных

- OLTP(Online Transaction Processing) – оперативная обработка транзакций.

- OLAP(Online Analytical Processing) – оперативная аналитическая обработка.

Многомерная схема хранения данных

- Идея в том, что можно сделать специализированный сервер и пусть данные в нем будут храниться не в виде таблиц, связанных между собой, а в виде **кубов** и измерений.
- **Измерения** - основные атрибуты анализируемого бизнес-процесса
- На пересечениях осей - измерений (Dimensions) - находятся данные, количественно характеризующие процесс - **меры** (Measures).



		Январь	Февраль	Март	
Москва	Яблоки	20 000	21 000	22 000	63 000
	Груши	25 000	27 000	29 000	81 000
	Киви	30 000	33 000	36 000	99 000
	Итого	75 000	81 000	87 000	243 000
Пермь	Яблоки	18 000	19 000	20 000	57 000
	Груши	20 000	22 000	24 000	66 000
	Киви	14 000	17 000	20 000	51 000
	Итого	52 000	58 000	64 000	174 000
Уфа	Яблоки	12 000	13 000	14 000	39 000
	Груши	18 000	20 000	22 000	60 000
	Киви	22 000	25 000	28 000	75 000
	Итого	52 000	58 000	64 000	174 000
Итого		179 000	197 000	215 000	591 000

Как это работает?

Предпосылки появления OLAP

Проблемы реляционных СУБД:

- Относительная сложность написания SQL запросов и их большое количество
- Агрегированные выборки
- Необходимость привлечения IT специалистов

Характеристики OLAP и OLTP

Характеристики OLTP системы

- Часто различные БД для разных подразделений
- Нормализованная схема, отсутствие дублирования информации
- Интенсивное изменение данных
- Транзакционный режим работы
- Транзакции затрагивают небольшой объем данных
- Обработка текущих данных – мгновенный снимок
- Очень много клиентов
- Малое время отклика – несколько мс

Характеристики OLAP системы

- Синхронизированная информация из различных БД с использованием общих классификаторов
- Ненормализованная схема БД с дубликатами
- Данные меняются редко, изменение происходит через пакетную загрузку
- Выполняются сложные нерегламентированные запросы над большим объемом данных с широким применением группировок и агрегатных функций.
- Анализ временных зависимостей
- Небольшое количество работающих пользователей – аналитики и менеджеры
- Большее время отклика (но все равно приемлемое) – несколько минут