Onboarding DWH

Базовые определения

База данных

Список ссылок на основные понятия:

1. База данных
2. Система управления базами данных
3. Реляционная модель данных

Хранилище данных и витрина данных

> **Хранилище данных** (*Data Warehouse*) — предметно-ориентированная информационная база данных, специально разработанная и предназначенная для подготовки отчётов и бизнес-анализа с целью поддержки принятия решений в организации.

> **Витрина данных** (*Data Mart*) — подмножество хранилища данных, которое поддерживает требования отдельного подразделения/некоторой группы пользователей

Подробнее про DWH и DM в специальном разделе

Качество данных

> **Качество данных** (*Data Quality*) — процесс управления данными, необходимый для гарантии того, что данные могут быть использованы для принятия решений в организации. Данные, которые считаются пригодными для использованию по назначению, являются данными высокого качества.

Примеры проблем с качеством данных — дублирование данных, неполные данные, противоречивые данные, неверные данные, плохо определенные данные, плохо организованные данные.

Существует шесть основных параметров качества данных:

1. **Точность**: данные отражают реальные бизнес-процессы; их точность может быть подтверждена проверенным источником.
2. **Полнота**: данные содержат все необходимые значения
3. **Согласованность**: одни и те же значения данных, хранящиеся в разных местах, не противоречат друг другу.
4. **Валидность**: данные собираются в соответствии с определенными бизнес-правилами и параметрами, соответствуют правильному формату и находятся в правильном диапазоне.
5. **Уникальность**: в данных отсутствует дублирование или пересечений значений во всех наборах данных.
6. **Своевременность**: данные доступны с такой задержкой, с которой они требуются.

Модель данных

Модель данных корпоративного хранилища представляет собой ER-модель (Entity-relationship model — модель «сущность-связь»), описывающую на нескольких уровнях набор взаимосвязанных сущностей, которые сгруппированы по функциональным областям и отражают потребности бизнеса в аналитическом анализе и отчетности.

Общая модель данных корпоративного хранилища разрабатывается последовательно и состоит из:

- концептуальной модели данных
- логической модели данных
- физической модели данных

Концептуальная модель

Концептуальная модель хранилища данных представляет собой описание основных сущностей и отношений между ними. Концептуальная модель является отражением предметных областей, в рамках которых планируется построение хранилища данных.

Логическая модель

Логическая модель расширяет концептуальную путем определения для сущностей их атрибутов, описаний и ограничений, уточняет состав сущностей и взаимосвязи между ними.

Физическая модель

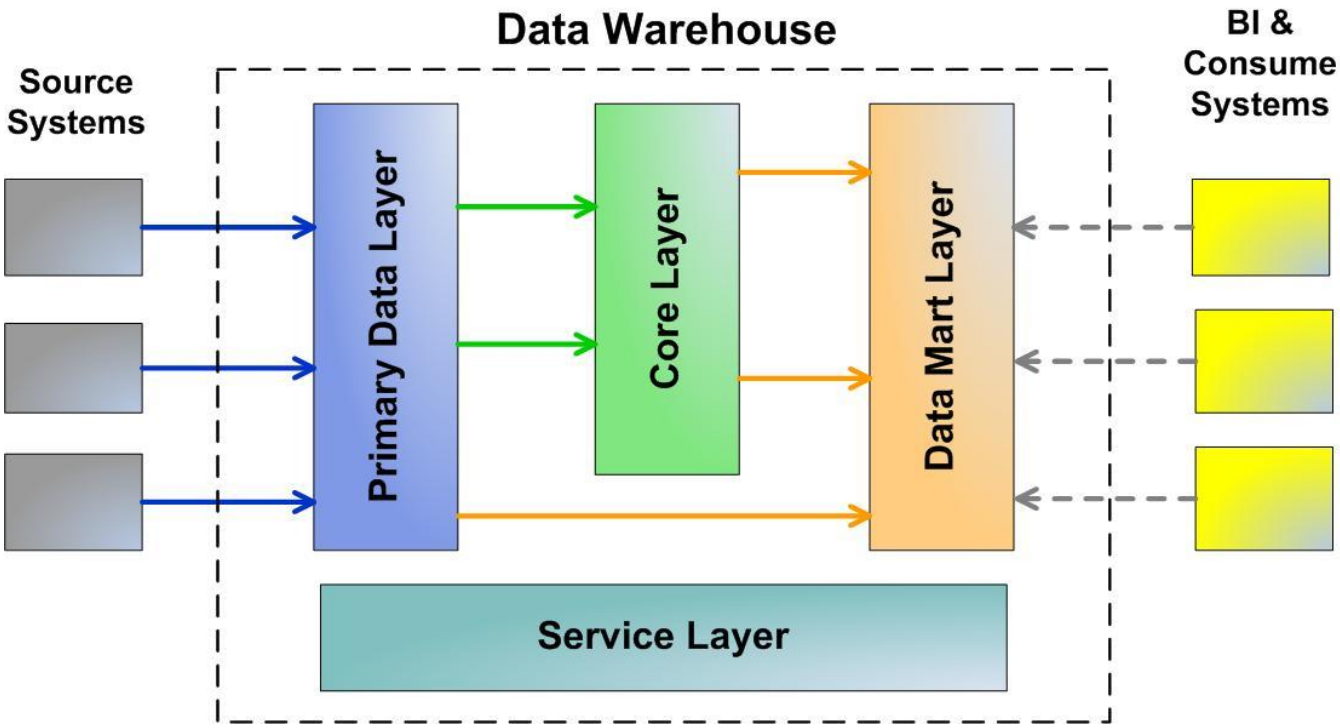
Физическая модель данных описывает реализацию объектов логической модели на уровне объектов конкретной базы данных.

Что такое DWH

Что это, зачем это и из чего состоит

Слои данных

Для того, чтобы получить пригодные для анализа данные, в DWH используется принцип "слоеного пирога" (LSA – Layered Scalable Architecture). Данные копируются с уровня на уровень и по ходу трансформируются определенным образом.



Обычно выделяют следующие слои данных:

1. **Сырой слой**, или **слой первичных данных (ODS)**
   На этом этапе выполняется загрузка информации из систем-источников в исходном качестве и сохранением полной истории изменений.

2. **Детальный слой (DDS)**
   На данном слое объединяются данные из разных источников, проводится их очистка и обогащение, приведение к единым структурам. Здесь мы обеспечиваем целостность и качество наших данных.
3. Иногда выделяют **агрегаты**
   На этом слое данные группируются, например по времени (чаще всего происходит суммирование).
4. **Слой витрин (DM)**
   Здесь мы создаем “тематические” наборы данных, хранящиеся в виде пригодном для их анализа. Каждая витрина обычно нужна для анализа или поддержки конкретного бизнес-процесса.

Методы моделирования данных

При построении хранилища данных/витрины данных необходимо спроектировать логическую модель, т.е. описать все необходимые сущности, их атрибуты и отношения между сущностями.

blocked URL

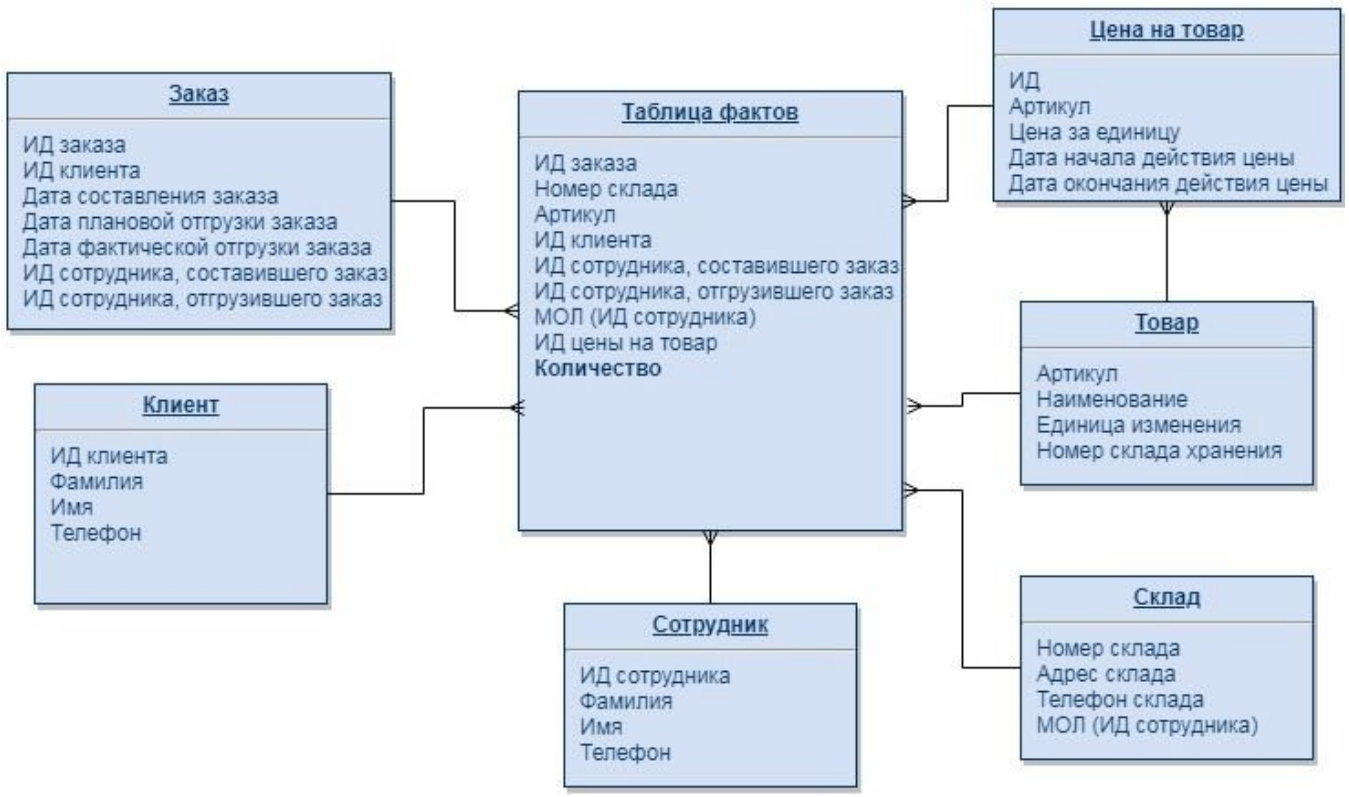
Звезда-снежинка VS 3НФ

Схема звезда = подход Кимбалла = OLAP

Модель данных состоит из двух типов таблиц: одной таблицы фактов (*fact table*) — центр «звезды» — и нескольких таблиц измерений (*dimension table*).

1. Таблица фактов обычно содержит одну или несколько колонок, дающих числовую характеристику какому-то аспекту предметной области (например, количество позиций в заказе), и несколько колонок-ключей, являющихся ссылками на таблицы измерений.
2. Таблицы измерений расшифровывают ключи, на которые ссылается таблица фактов; например, таблица измерения «товар» может содержать сведения о названии товара, его производителе, типе товара.

Обычно данные в таблицах-измерениях денормализованы: ценой несколько неэффективного использования дискового пространства удается уменьшить число джойнов в запросах, что обычно приводит к сильному уменьшению времени выполнения запроса. Иногда, тем не менее, требуется произвести нормализацию таблиц-измерений. Такая схема носит название «снежинка» (snowflake schema).

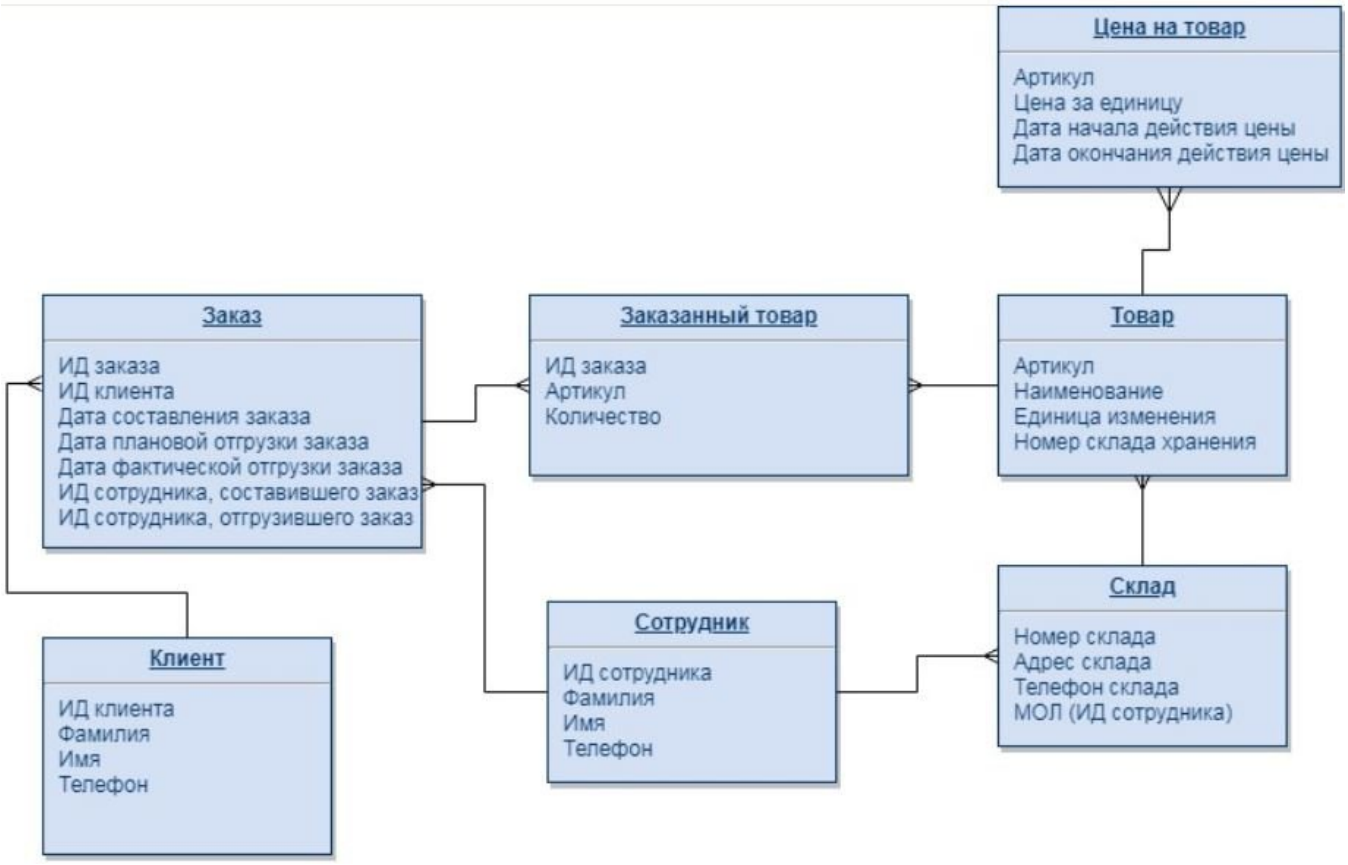


3НФ = подход Инмона = OLTP

Данный подход предполагает, что данные лежат в хранилище в третьей нормальной форме, когда каждый неключевой атрибут таблицы должен предоставлять информацию о лишь о полном ключе и ни о чём более. Это помогает избежать избыточности, которая может привести к логически ошибочным результатам выборки или изменения данных. 3НФ избавляет от транзитивных зависимостей: любой столбец таблицы зависеть только от ключевого столбца, что позволяет обеспечивать целостность данных в СУБД.

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

Подробное описание всех НФ



Data Vault VS Ancor Model

Но современные задачи требует современных решений и классические подходы уже не работают.

Классическое DWH

Красивые витрины по схеме звезда или снежинка, все лежит аккуратненько в 3НФ

По waterfall сперва проанализируем требования, потом подумаем, потом согласуем, обмозгуем еще хорошенько, в общем, DWH – это не быстро, это основательно

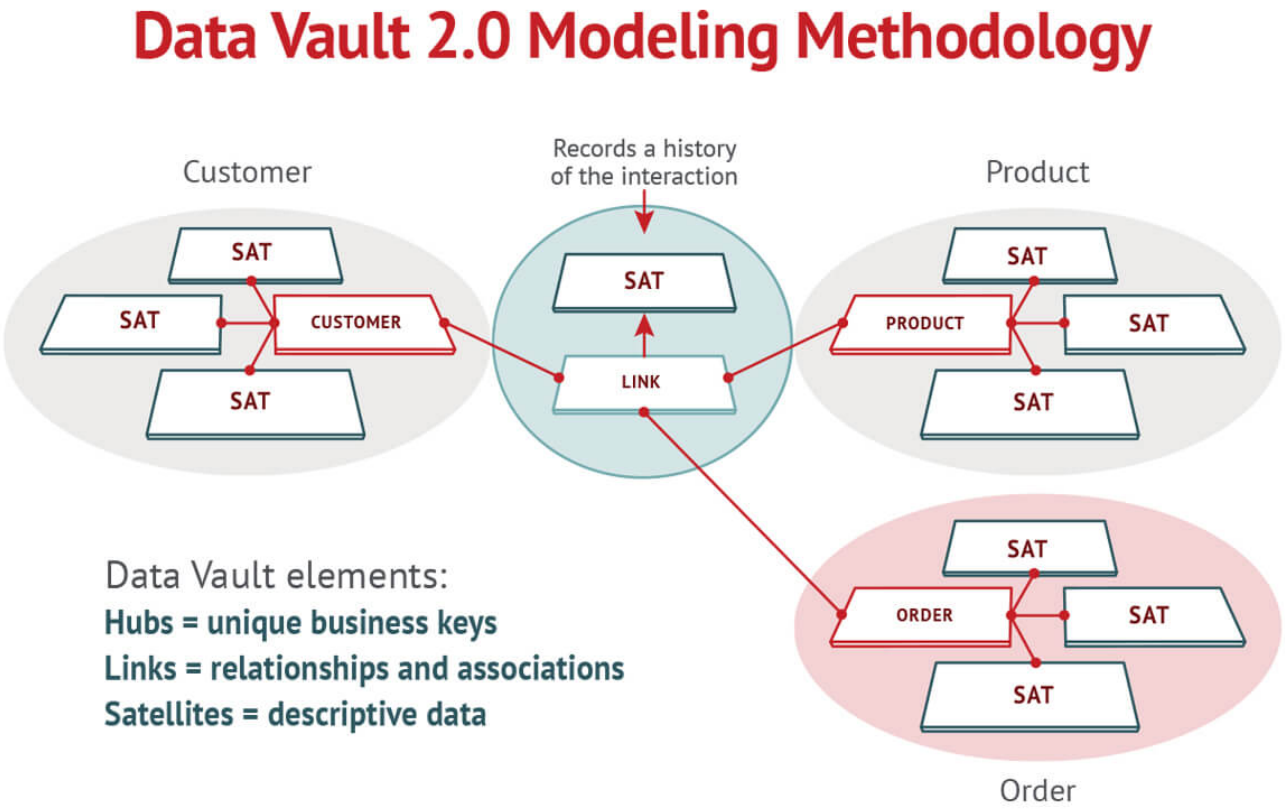
Но заказчик почему-то не знает, что хочет, и вообще готов смотреть на то, что получается, а потом уточнять, что ему хочется.

Да и бизнес-модель меняется раз в две недели, нам выживать надо, а не хранилища строить

И вообще, мы agile и гибкие, давайте все переделаем…

..и желательно завтра результат уже пощупать.

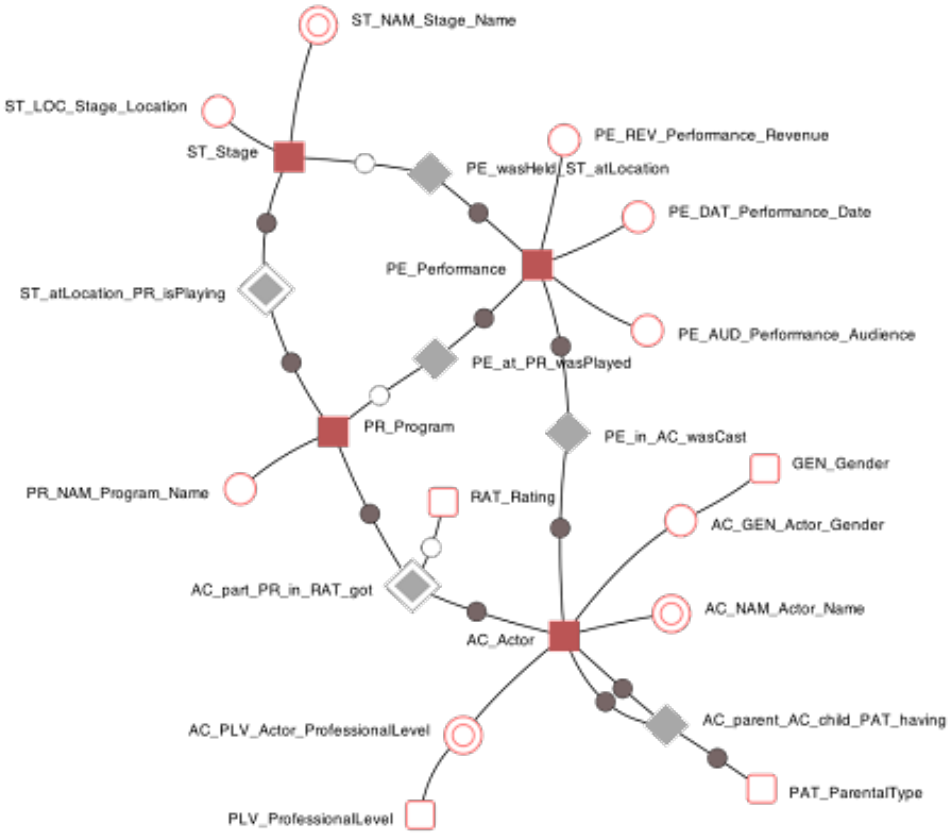
Так появились Data Vault и якорная модель.



Data Vault вводит и регламентирует основные типы таблиц: Хаб (Hub), Связь (Link) и Сателлит (Satellite).

1. Хабы - таблицы, содержащие уникальный список бизнес ключей. Атрибуты Хаба включают:
    a. Ключ бизнес-сущности из внешней системы
    b. Суррогатный ключ
    c. Временная отметка даты загрузки
    d. Код источника данных
2. Связи представляют отношения или транзакцию между двумя или более компонентами бизнеса (два или более бизнес ключа). Атрибуты линка включают:
    a. Суррогатный ключ
    b. Ключи Хабов: от 1-го Хаба до N-го Хаба
    c. Временная отметка даты загрузки
    d. Код источника данных
3. Атрибуты саттелита включают:
    a. Первичный ключ Сателлита: Первичный ключ Хаба или первичный ключ Связи
    b. Даты действия записи (SCD2)
    c. Временная отметка даты загрузки
    d. Код источника данных

Больше про Data Vault...



Якорное моделирование - это технология моделирования гибкой базы данных, подходящая для информации, которая со временем изменяется как по структуре, так и по содержанию. В методике моделирования используются четыре типа таблиц: якорь, атрибут, связь и узел. Полученные модели могут быть переведены в физические объекты баз данных с помощью автоматизации.

1. Anchor (Якорь) — это существительное, объект реального мира. Anchor таблица состоит из следующих атрибутов:
    a. Суррогатный ключ
    b. Временная отметка даты загрузки
2. Tie (Связь) — это таблица для хранения связей между объектами. У Tie не может быть атрибутов.
3. Attribute (Атрибут) — это таблица для хранения свойства, атрибута объекта, при этом на каждое свойство ровно одна таблица. Каждая Attribute-таблица содержит:
    a. Суррогатный ключ
    b. Временная отметка даты загрузки
    c. Непосредственно значение

Больше про Ancor Model...

Slowly Changing Dimension (SCD)

Как правило при построении хранилища данных есть требование хранить историю изменения данных в источнике. Есть два основных способа хранения истории - на ежедневной основе дописывать актуальный срез данных или выстраивать интервалы актуальности версий каждой строки. Первый способ понятен и легко реализуем, но зачастую занимает неоправданно много памяти. Второй способ - SCD2.

> Медленно меняющиеся измерения (Slowly Changing Dimensions, SCD) — механизм отслеживания изменений в данных измерения в терминах хранилища данных. Применяется в случае, если данные меняются не очень часто и не по расписанию.

Для отслеживания истории изменений SCD2 использует добавление новой строки и дополнительных столбцов. В качестве дополнительных столбцов можно добавить следующие:

- Версия строки
- Статус (актуальная строка или нет)
- Временной интервал, в течение которого данные строки можно считать актуальными
- Дата вступления строки в силу + статус

Пример:

ID записи	Табельный номер	ФИО	Должность	Отдел	Дата начала	Дата окончания
1026	ИБ-69420	Иванов Сергей Петрович	Младший специалист	Отдел оптовых закупок	2000-01-01T00:00:00	2008-08-08T00:00:00
1027	ИБ-69420	Иванов Сергей Петрович	***Главный специалист***	***Отдел продаж***	2008-08-08T00:00:00	NULL

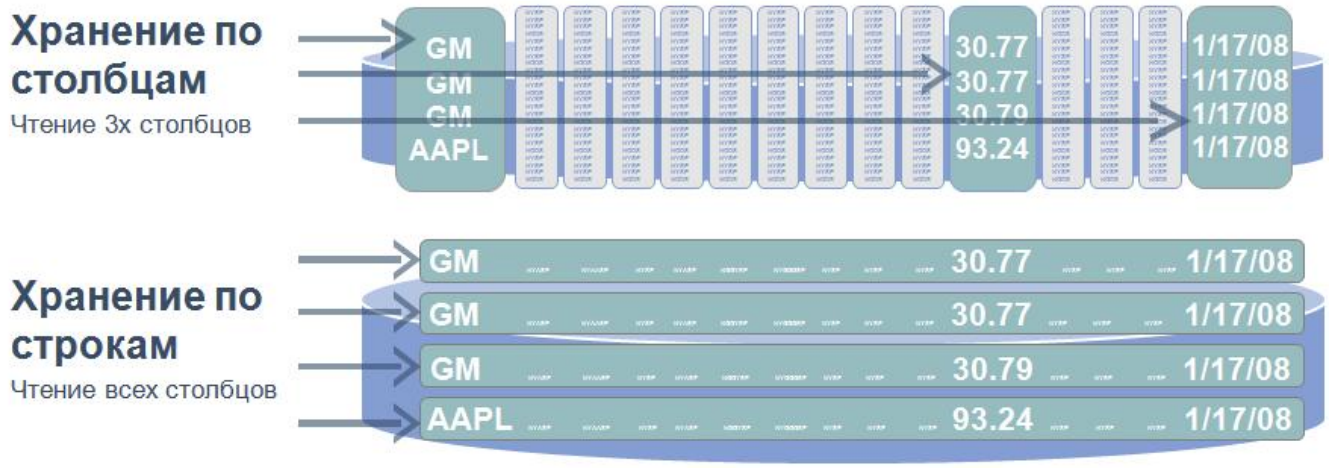
Про все типы SCD тут.

Теория БД

Какие они бывают, из чего состоят и как оптимально использовать

Колоночные и строковые СУБД

Логически можно организовать хранение данных в любой таблице двумя способами: по строкам или по столбцам (колонкам). Оба подхода имеют свои плюсы и минусы, а также определяют особенности обращения к данным, добавления и удаления новых данных и другие операции.



Строчные базы данных

В **строчной** базе данных физически хранится вся строка как единое целое, в которой последовательно записаны одно поля за другим. После последнего поля записи идет первое поле следующей записи:

[A1, B1, C1], [A2, B2, C2], [A3, B3, C3]…

где A, B и C — это поля (столбцы), а 1,2 и 3 — номер записи (строки).

Такое хранение удобно для частых операций добавления новых строк в базу данных. Например, в транзакциях, когда нужно часто и быстро вставлять и удалять новые строки.

Типичные строковые базы данных:

- Postgres
- MySQL

Колоночные базы данных

Когда вы занимаетесь, например, какой-то аналитикой, чаще всего из всех данных нужно выбирать только значения определенных атрибутов из определенных колонок. В строчной базе данных такое действие на самом деле оказывается неоптимальным: если вам нужно выбрать значение одной колонки, строчная база данных все равно должна пройтись по каждой строке, найти в ней нужную колонку и вытащить значение из нее.

Эту проблему решают **колоночные** базы данных, в которых физически хранятся не строки, а, как предполагает название, колонки:

[A1, A2, A3], [B1, B2, B3], [C1, C2, C3]…

Недостатки колоночных баз данных ровно противоположны недостаткам строчных: в них сложнее проводить запись новых данных и они не подходят для транзакционных задач.

Типичные колоночные базы данных:

- Redshift
- BigQuery
- Snowflake

Объекты в базах данных

Таблица

Как подсказывает логика, таблица — самый базовый объект в базах данных, который представляет собой совокупность строк и столбцов. Ячейка — место, где строка и столбец пересекаются. Таблица содержит определенное число столбцов, но может иметь любое количество строк. Каждая строка однозначно определяется одним или несколькими уникальными значениями, которые принимают её ячейки из определенного подмножества столбцов.

Представление (view)

Это результат выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению. Представления иногда называют «виртуальными таблицами». Такое название связано с тем, что представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему.

```
CREATE VIEW v AS SELECT subject, num_views/num_replies AS param FROM topics WHERE num_replies>0;
```

Материализованное представление (materialized view)

Материализованное представление сохраняет где-то в памяти результат запроса, который создает представление. Зачем нам это делать? Чтобы уменьшать время отклика системы на запрос и экономить ресурсы.

```
CREATE TEMPORARY TABLE tmp_table SELECT forum_id, count(*) AS num FROM topics GROUP BY forum_id;
SELECT MAX(num) FROM tmp_table;
DROP TABLE tpm_table;
```

Функция

Функция — это команда, которую можно сохранить в памяти и вызывать в необходимый момент. Она может производить действия над объектами БД и выдавать какие-то значения.

```
       CREATE FUNCTION CalcIncome ( starting_value INT )
RETURNS INT

BEGIN

   DECLARE income INT;

   SET income = 0;

   label1: WHILE income <= 3000 DO
     SET income = income + starting_value;
   END WHILE label1;

   RETURN income;

END;
```

И затем вызываем эту функцию:

```
       SELECT CalcIncome (1000);
```

Индексы

Индексы — специфическая для строчных баз данных штука, которая позволяет быстрее и удобнее находить необходимые данные в таблицах.

Если вам нужно, например, найти строки с определенной датой, самое простое, что вы можете сделать — пройтись по каждой строке, посмотреть, какая в ней записана дата и выбрать необходимые записи. Логично, что при росте объема таблицы такая операция будет занимать все больше времени, и очень скоро станет неэффективной.

Простыми словами, индекс — это аналог библиотечного каталога, который позволяет быстрее найти, где лежит необходимые данные (необходимая строка). Индекс строится по определенному полю или набору полей в таблице.

Кажется, что чем больше индексов, тем быстрее будут отрабатывать запросы к базе данных. Однако при излишнем увеличении количества индексов падает производительность операций изменения данных (вставка/изменение/удаление) и увеличивается размер БД, поэтому к добавлению индексов следует относиться осторожно (Вова Федорков не даст соврать).

Виды алгоритмов JOIN

Команда JOIN одна из базовых для языка SQL, но как она именно делает это внутри не совсем очевидно и может быть полезно разобраться чуть подробнее про алгоритмы.

Nested loops join

Тут всё просто, разберем на примере двух таблиц. Одна из них будет ведущая, другая ведомая. Берём для каждой строчки ведущей и ищем в ведомой строки, соответствующие условию соединения.

Hash join

Здесь мы используем дополнительную структуру - хэш-таблица в которую помещаем меньшую таблицу. Затем для каждой строки из большей таблицы выполняется поиск значений, соответствующих условию соединения.

Хэш-таблица (hash table) *— это специальная структура данных для хранения пар ключей и их значений. По сути это массив, в котором ключ представлен в виде хэш-функции (алгоритм, который позволяет устраивать быстрый поиск).*

Структура используется если хотят быстро выполнять операции вставки/удаления/нахождения.

Merge Join

В этом алгоритме перед тем как соединять таблицы мы сортируем их по ключу, по которому будем объединять. После этого бежим по ключу в одной из таблиц и берём соответствующие значения из второй таблицы.

Круто! А какой выбрать-то?

По умолчанию исполнитель запроса в БД автоматически выбирает самый оптимальный алгоритм JOIN по его мнению. Но иногда бывает полезно залезть в план запроса и посмотреть какой именно был выбран и явно прописать его.

Hash/Merge - используют дополнительную оперативную память (для хэш-таблицы и сортировки соответсвенно), у **Nested Loops** таких требований нет.

Hash - отлично подходит когда одна из таблиц маленькая и её можно безболезненно поместить в оперативную память.

Если перед этим в ходе предобработки делается сортировка, то можно явно использовать **Merge**.

По умолчанию же чаще всего используется **Nested Loops**.

Системные таблицы

Что это такое

Это таблицы, в которых хранятся **метаданные** определенной базы данных. В них описывается структура объектов базы данных, таких как таблицы, представления, индексы и т.д.

Зачем нужны аналитику

Для поиска нужных ему таблиц/атрибутов в таблицах по всей БД, проверки типа данных атрибутов, наличия доступа к таблице и т.д.

Какие бывают

Oracle

Несколько примеров:

- all_tables - таблицы
- all_views - представления
- all_tab_columns - колонки в таблицах и представлениях

Полный список тут

```
SELECT table_name FROM all_tables WHERE owner=' '
```

Clickhouse

Несколько примеров:

- system.tables - таблицы
- system.columns - колонки в таблицах

Полный список тут

```
SELECT name FROM system.tables t WHERE database = 'analytics'
```

Greenplum

Несколько примеров:

- information_schema.tables - таблицы
- information_schema.columns - колонки в таблицах

Полный список тут

```
SELECT table_name FROM information_schema.tables WHERE table_schema = 'DM'
```

Транзакции

Транзакция в SQL — это какое-либо действие (набор оных), которое изменяет что-то в нашей базе данных.

Разберемся на примере: операция вставки в таблицу - типичный пример транзакции.
Мы вставляем запись, а в процессе у нас может что-то сломаться со стороны БД, мы будем вставлять неверный тип данных, вступим в конфликт с кем-нибудь и в итоге застрянем посередине процесса.
Поэтому важно контролировать разные типы транзакций, следить за правильностью их выполнения и корректно обрабатывать ошибки базы данных.

Что может входить в транзакцию?

Обычно это операции по прямому изменению данных в нашей базе данных:

- вставка строк в таблицу (командой *INSERT*);
- удаление строк из таблицы (команда *DELETE*);
- обновление строк по определенному условию (команда *UPDATE*);
- создание таблиц;
- изменение типов данных;
- чтение и формирование запроса, который затем будет записан в какую-нибудь таблицу;

Управление транзакциями

После того как мы что-то поменяли в нашей базе данных связанное с командами *INSERT/DELETE/UPDATE* нужно либо подтвердить это изменение, либо, если изменение прошло неуспешно, отменить выполненную транзакцию.

Делается это с помощью двух команд:

- *COMMIT* - сохраняет все изменения, которые были внесены в базу данных с момента выполнения последней команды *COMMIT/ROLLBACK*;
- *ROLLBACK* - отменяет все изменения, которые были внесены в базу данных с момента выполнения последней команды *COMMIT/ROLLBACK*;
    - Зачем тогда *ROLLBACK*, если можно не делать *COMMIT* и ничего не применится?
      -> Если нам не понравился результат первой транзакции и мы не сделали *ROLLBACK*, то сделав следующий *COMMIT* после второй транзакции мы применим все изменения и первой и второй транзакции.
      -> Начав исполнять первую транзакцию и не закончив её для базы данных мы оставляем её в подвешенном состоянии. Тем самым иногда блокируя другие транзакции, которые должны были провзаимодействовать с нашими данными.

Еще полезная команда при написании транзакций:

- *SAVEPOINT* - позволяет создавать точку в транзакции, к которой вы можно вернуться, не откатывая полностью.

Уровни изоляции

При выполнении транзакций часто бывает так, что две транзакции изменяют одну и ту же строчку в таблице (например баланс клиента). И тогда произойдет конфликт, в результате которого значение перезапишется неправильно или одна из транзакций сломается.

Чтобы избежать таких ситуаций используется уровни изоляции, которые задают приоритет в выполнении транзакций, но при этом позволяют не терять в скорости выполнения операций.

Со звездочкой*

- потерянное обновление (*lost update*) — при одновременном изменении одного блока данных разными транзакциями теряются все изменения, кроме последнего;
- «грязное» чтение (*dirty read*) — чтение данных, добавленных или изменённых транзакцией, которая впоследствии не подтвердится (откатится);
- неповторяющееся чтение (*non-repeatable read*) — при повторном чтении в рамках одной транзакции ранее прочитанные данные оказываются изменёнными;
- фантомное чтение (*phantom reads*) — одна транзакция в ходе своего выполнения несколько раз выбирает множество строк по одним и тем же критериям. Другая транзакция в интервалах между этими выборками добавляет строки или изменяет столбцы некоторых строк, используемых в критериях выборки первой транзакции, и успешно заканчивается. В результате получится, что одни и те же выборки в первой транзакции дают разные множества строк.

Существует 4 проблемы с которыми можно столкнуться при работе работе нескольких транзакций.

Уровней изоляции 4 по повышению изолированности выполнения транзакций. Каждый из последующих включает в себя предыдущий.

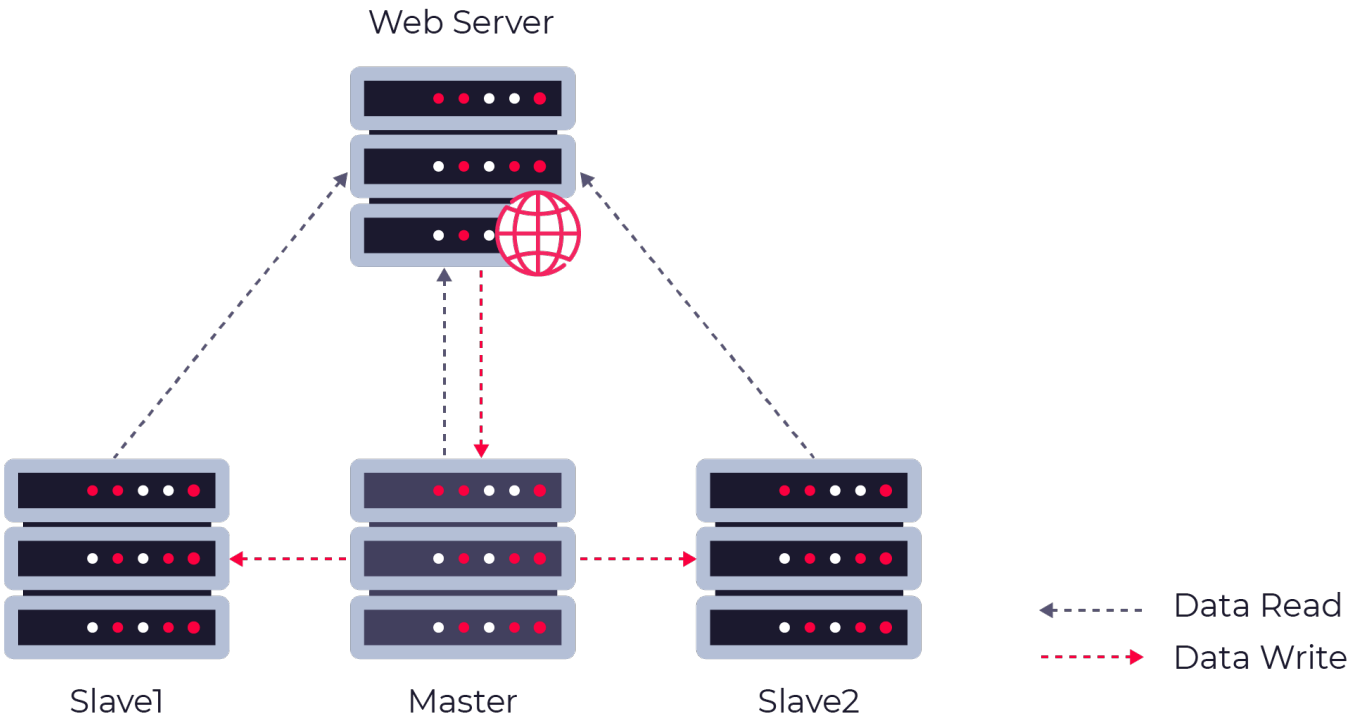
Уровень изоляции	Фантомное чтение	Неповторяющееся чтение	«Грязное» чтение	Потерянное обновление
SERIALIZABLE	+	+	+	+
REPEATABLE READ	-	+	+	+
READ COMMITTED	-	-	+	+
READ UNCOMMITTED	-	-	-	+

Масштабирование баз данных

Когда данных становится очень_очень_очень много, таблицы становятся бесконечными и место заканчивается, с этим нужно что-то делать. Помимо того, что можно просто втыкать новые жесткие диски и увеличивать место для хранения, есть более умные подходы к этой проблеме, которая в общем называются масштабированием баз данных.

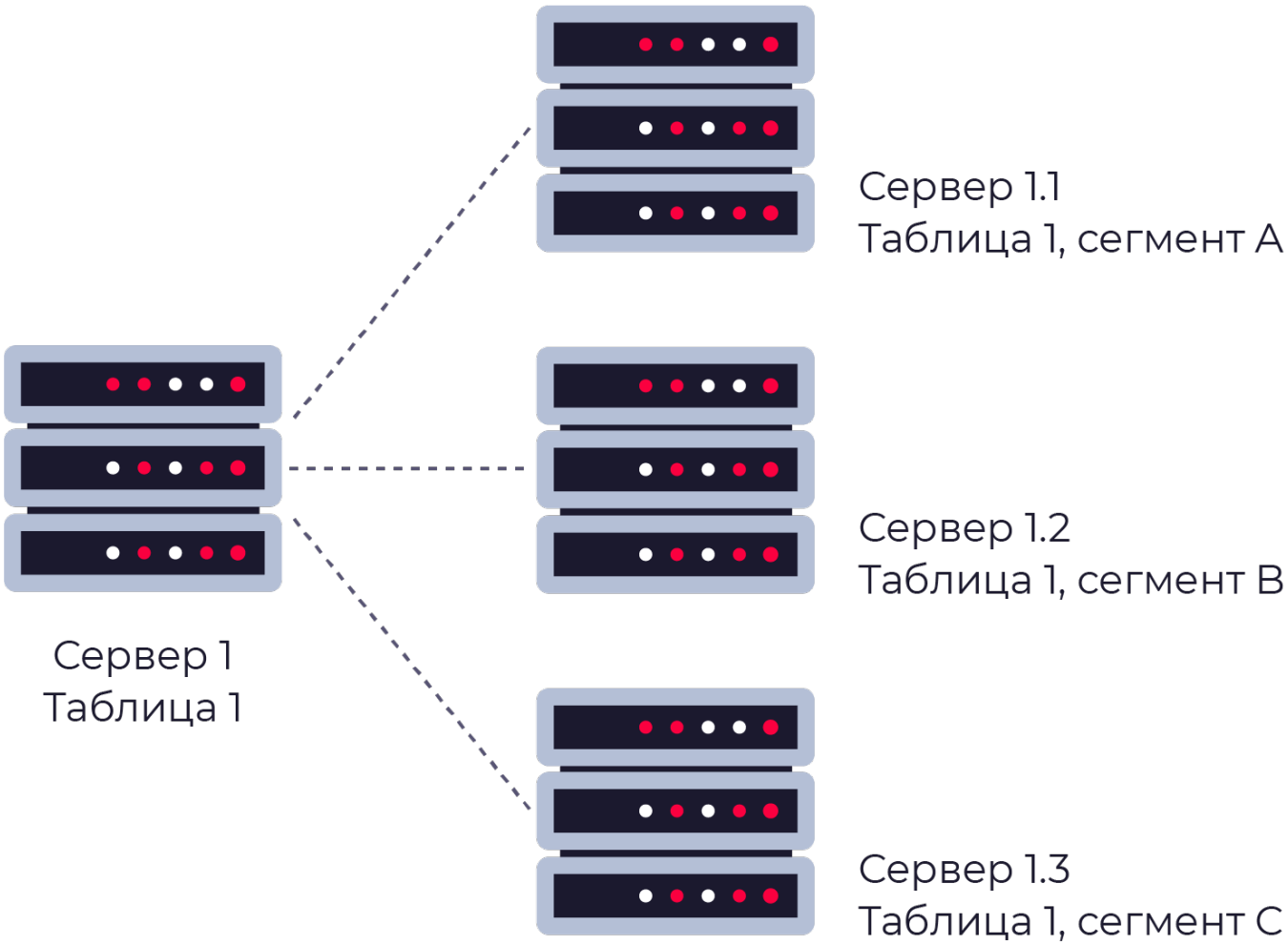
В масштабировании данных выделяют следующие основные подходы:

Репликация



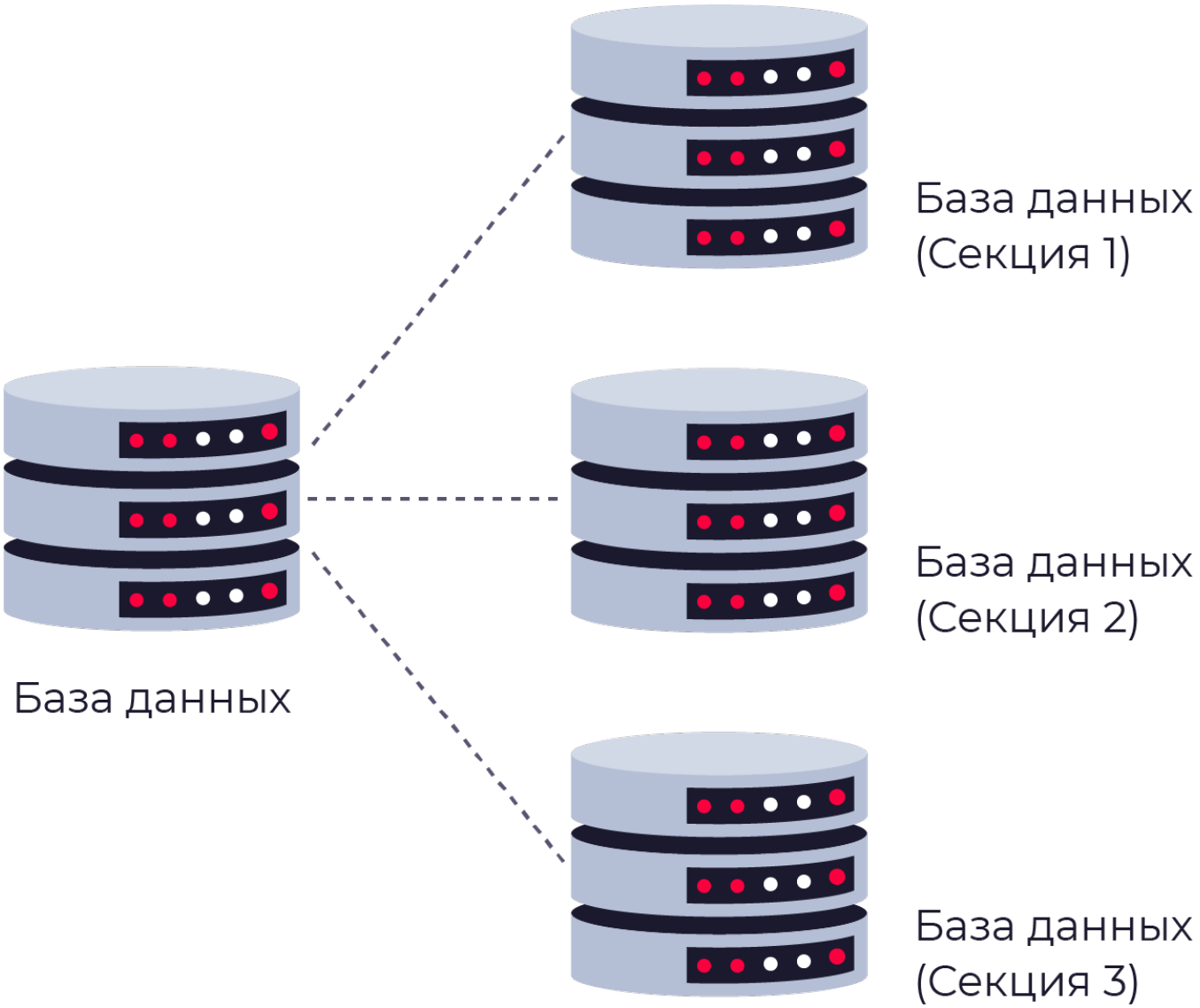
Репликация — это про дублирование данных на разных серверах (читай "компьютерах", "местах для хранения данных"). Если даже головной сервер выходит из строя, любой другой сможет его заменить и ваши расчеты не пострадают. Чаще всего репликация используется как средство для обеспечения отказоустойчивости вместе с другими методами масштабирования.

Партиционирование



Возьмем и разобьем наши данные по какому-то признаку (имена пользователей по первой букве фамилии, заказы по месяцам в году и так далее). Тогда, если мы что-то ищем, не нужно пробегать по всей таблице, а достаточно понять, в какой части находится необходимая нам информация и уже искать в конкретном сегменте.

Шардирование



Шардирование — оно же распределенное хранение таблиц — по своей сути очень похоже на партиционирование. Здесь мы также делим таблицы на части согласно какой-то логике, но части таблицы хранятся раздельно, на разных физических серверах.