[⭐⭐] Нормализация

Files & media

Нормализация — это процесс организации данных в базе данных таким образом, чтобы:

1. Уменьшить избыточность данных (дублирование информации).
2. Обеспечить целостность данных (уменьшить вероятность ошибок при изменении данных).
3. Сделать базу данных более гибкой и легко расширяемой.

Нормализация достигается путём деления больших таблиц на более мелкие и создания связей между ними. Это помогает избежать дублирования данных и минимизировать аномалии при вставке, обновлении или удалении данных.

▼ Аномалии

Аномалия — эффект, возникающий при недостаточной нормализации БД или сложных зависимостях между данными, влекущий за собой проблемы

- возможной логической некорректности данных
- невозможности представления некоторых данных в данной форме
- технической/алгоритмической сложность внесения или изменения данных

CourseId	Lecturer	Phone
1	Корнеев Г. А.	111-11-11
2	Киракозов А. Х.	222-22-22
2	Киракозов А. Х.	333-33-33
3	Кудряшов Б. Д.	444-44-44
3	Кудряшов Б. Д.	555-55-55
4	Сегаль А. С.	666-66-66

Таблица не находися в 1NF, что может привести к аномалиям.

Аномалия вставки — зависимость возможности записать обладающие собственным независимым смыслом данные от наличия другой связанной информации.

В рассмотренном выше примере невозможно записать информацию о телефоне конкретного преподавателя, если он не читает ни один курс (таким образом, возможность записать **PhonePhone** для конкретного **LecturerLecturer** зависит от наличия соответствующего **CourseIdCourseId**, хотя напрямую они не зависят друг от друга).

Аномалия удаления — невозможность удалить часть данных, не удалив никакую связанную с ней информацию.

В рассмотренном выше, опять же, примере невозможно удалить информацию о том, что конкретный преподаватель читает конкретный курс, не потеряв его номер телефона (как и в случае с аномалией вставки, возможность хранить **PhonePhone** зависит от существования соответствующего **CourseIdCourseId**).

Аномалия изменения — ситуация, в которой частичное изменение данных нарушает целостность базы данных.

В рассмотренном примере если один преподаватель ведет один курс и имеет два телефона, при изменении CourseIdCourseId в одной из соответствующих ему записей будет невозможно восстановить какой курс на самом деле ведет преподаватель (записи с разными PhonePhone, но одинаковыми LecturerLecturer и CourseIdCourseId, должны всегда поддерживаться в таком же состоянии).

CourseId	Year	Lecturer	Phone
1	2020	Корнеев Г. А.	111-11-11
2	2019	Киракозов А. Х.	222-22-22
2	2020	Киракозов А. Х.	222-22-22
3	2019	Левина А. Б.	333-33-33
3	2020	Чепурной А. И.	444-44-44

Аномалия, свойственная 2НФ, возникает, когда какой-то атрибут зависит от ключа транзитивно через множество неключевых атрибутов. Рассмотрим следующий пример (см. слева таблицу).

В нем есть две базовые функциональные зависимости:

CourseId, Year→Lecturer и Lecturer→Phone. Несмотря на то, что данное отношение находится во 2НФ, в нем все еще имеют место все три аномалии 1НФ – аномалии вставки, удаления и изменения (информация о телефонах и о преподавании никак не разделена). Для исправления аномалий 2НФ отношение переводят в третью нормальную форму и выше.

▼ 0FN

Таблица находится в **нулевой нормальной форме (0NF)**, если:

- В таблице **нет ограничений на структуру данных**.

- Данные могут быть **неатомарными** (вложенные таблицы, списки, массивы в одном поле).
- Не соблюдаются никакие правила нормализации.

Это самый простой и изначальный вид организации данных. Таблица в 0NF может содержать дублирующиеся и несогласованные данные, и нет никаких гарантий целостности.

Проблемы с 0NF:

1. **Неатомарные данные**: Несколько значений в одной ячейке могут создавать трудности при поиске или обновлении данных.
2. **Отсутствие целостности**: Данные могут легко дублироваться и быть несогласованными.
3. **Трудности с обработкой**: Извлечение и изменение таких данных сложнее, так как они находятся в нелогичном формате.

▼ *Пример*

В таблице присуствуют неатомарные значения в `adress` и `orders`

customer_id	customer_name	address	orders
1	Alice	"Street A, City 1, Country 1"	"Apple, Banana"
2	Bob	"Street B, City 2, Country 2"	"Orange, Mango"
3	Carol	"Street C, City 3, Country 3"	"Grapes, Watermelon"

▼ Первая нормальная форма (1NF)

Таблица находится в первой нормальной форме, если:

1. Все значения в таблице являются **атомарными** (неделимыми).
2. Каждая строка таблицы уникальна (обычно это обеспечивается с помощью **первичного ключа**).
3. Нет дублирующих строчек (что очевидно из 2го пункта)
4. Все столбцы содержат однотипные данные (например, все значения в одном столбце — это числа, а не смесь чисел и текста).

▼ *Пример:*

Таблица, которая нарушает 1NF:

customer_id	customer_name	address	orders
1	Alice	"Street A, City 1, Country 1"	"Apple, Banana"
2	Bob	"Street B, City 2, Country 2"	"Orange, Mango"
3	Carol	"Street C, City 3, Country 3"	"Grapes, Watermelon"

Приведение к **первой нормальной форме (1NF)**

В первой нормальной форме мы избавляемся от **неатомарных значений**. Это означает, что каждое значение в ячейке должно быть неделимым (атомарным). Разделим составные данные (например, адреса и заказы) на отдельные строки и столбцы.

1. **Разделение адреса** на отдельные компоненты (улица, город, страна).
2. **Создание отдельной строки для каждого заказа**.

customer_id	customer_name	street	city	country	order
1	Alice	Street A	City 1	Country 1	Apple
1	Alice	Street A	City 1	Country 1	Banana
2	Bob	Street B	City 2	Country 2	Orange
2	Bob	Street B	City 2	Country 2	Mango
3	Carol	Street C	City 3	Country 3	Grapes
3	Carol	Street C	City 3	Country 3	Watermelon

Теперь данные атомарны, и таблица соответствует **1NF**.

▼ Вторая нормальная форма (2NF)

Таблица находится во второй нормальной форме, если:

1. Она уже находится в **1NF**.
2. Все неключевые атрибуты **полностью** зависят от цельного **первичного ключа**, а не от его части (если первичный ключ составной).

То есть, 2NF устраняет частичные зависимости, когда неключевые столбцы зависят только от части составного ключа.

▼ *Пример*

Чтобы привести таблицу ко **второй нормальной форме**, нужно устранить **частичные зависимости**. Частичная зависимость — это когда неключевой атрибут зависит только от части составного ключа. В нашем случае, составной ключ — это `customer_id` и `order` (если мы рассматриваем таблицу как таковую).

Однако данные, такие как **адрес клиента**, зависят только от `customer_id`, а не от заказа. Это значит, что нужно разделить данные, которые зависят от **клиента**, и данные, которые зависят от **заказа**.

Создадим две таблицы:

1. **Таблица клиентов** (`customers`), где храним данные о клиентах и их адресах.
2. **Таблица заказов** (`orders`), где храним заказы клиентов.

Таблица customers:

customer_id	customer_name	street	city	country
1	Alice	Street A	City 1	Country 1
2	Bob	Street B	City 2	Country 2
3	Carol	Street C	City 3	Country 3

Таблица orders:

customer_id	order

1	Apple
1	Banana
2	Orange
2	Mango
3	Grapes
3	Watermelon

Теперь данные находятся во **2NF**, так как каждая таблица содержит атрибуты, которые зависят от **полного первичного ключа**.

▼ Третья нормальная форма (3NF)

Таблица находится в третьей нормальной форме, если:

1. Она уже находится в **2NF**.
2. Все неключевые атрибуты **независимы друг от друга** и зависят только от **первичного ключа**.

Это означает, что **нет транзитивных зависимостей**. Транзитивная зависимость возникает, когда один неключевой столбец зависит от другого неключевого столбца, который, в свою очередь, зависит от первичного ключа.

▼ *Пример*

Теперь нужно устранить **транзитивные зависимости**. Это означает, что неключевые атрибуты не должны зависеть друг от друга. Например, если в таблице клиенты у нас есть зависимость между полями `city` и `country`, это нарушает 3NF.

Для приведения таблицы к **3NF**, разделим данные об адресах в отдельную таблицу, так как город и страна связаны между собой и зависят от адреса, а не от `customer_id`.

Создадим три таблицы:

1. **customers** для хранения информации о клиентах.
2. **addresses** для хранения информации о адресах.
3. **orders** для хранения заказов.

Таблица customers:

customer_id	customer_name	address_id
1	Alice	1
2	Bob	2
3	Carol	3

Таблица addresses:

address_id	street	city	country
1	Street A	City 1	Country 1
2	Street B	City 2	Country 2

3	Street C	City 3	Country 3

Таблица orders:

customer_id	order
1	Apple
1	Banana
2	Orange
2	Mango
3	Grapes
3	Watermelon

Теперь у нас нет транзитивных зависимостей, и таблицы находятся в **третьей нормальной форме (3NF)**.

▼ Другие формы нормализации (4NF-5NF)

Формы выше **третьей нормальной** формы применяются в сложных базах данных, где:

- Много взаимосвязанных атрибутов и данных, которые часто дублируются в одном ключе.
- Требуется максимально уменьшить дублирование данных и повысить целостность, даже если это приводит к большему количеству таблиц.
- Часто встречаются в больших корпоративных системах с множеством сложных бизнес-правил.

В большинстве случаев для повседневных приложений достаточно нормализации до **3NF**, но в системах с очень сложными зависимостями, где важна максимальная оптимизация и чистота данных, могут быть использованы **4NF** и **5NF**.

Четвёртая нормальная форма (4NF)

Таблица находится в **четвёртой нормальной форме (4NF)**, если:

1. Она уже находится в **третьей нормальной форме (3NF)**.
2. В таблице **нет многозначных зависимостей**.

Что такое многозначная зависимость?

Многозначная зависимость возникает, когда один атрибут зависит от другого атрибута, но эти зависимости не связаны между собой напрямую. (например, студент учит несколько языков и ходит в несколько секций)

4NF полезна для устранения избыточных данных в случаях, когда есть несколько независимых атрибутов, зависящих от одного ключа. Применяется реже, так как для многих систем достаточно нормализации до 3NF, а многозначные зависимости встречаются не так часто.

Пятая нормальная форма (5NF)

Таблица находится в **пятой нормальной форме (5NF)**, если:

1. Она уже находится в **четвёртой нормальной форме (4NF)**.

2. **Каждая зависимость в таблице является зависимостью по всему ключу**, и данные не могут быть разделены без потери информации (эта форма также известна как устранение зависимостей от соединения).

Что такое зависимость от соединения?

Зависимость от соединения возникает, когда информация, разделённая на несколько таблиц, может быть правильно восстановлена только с помощью **соединения (JOIN)**.

5NF используется для устранения более сложных зависимостей и предотвращения избыточности в системах, где данные распределены между несколькими сущностями, и важны сложные связи. В реальных приложениях эта форма используется редко, поскольку большинству систем хватает нормализации до 3NF или 4NF.

Заключение

3NF: подойдёт для большинства систем, так как решает основные проблемы с избыточностью и обеспечивает целостность данных.

4NF: необходима, когда в данных присутствуют многозначные зависимости, которые могут привести к дублированию информации.

5NF: используется в очень сложных системах, где важно устранить зависимости от соединения.

▼ Денормализация

Денормализация — это процесс обратный нормализации, когда данные намеренно дублируются или объединяются в одну таблицу для улучшения производительности системы.

Зачем нужна денормализация:

1. **Повышение производительности**: Нормализованные данные могут замедлять работу системы, так как часто требуется выполнение множества операций `JOIN` между таблицами, чтобы собрать данные. Денормализация уменьшает количество `JOIN`, ускоряя выборку.
2. **Упрощение запросов**: Когда данные денормализованы, запросы становятся проще, так как информация уже хранится в одной таблице, а не в нескольких.
3. **Оптимизация чтения**: В высоконагруженных системах, где много операций чтения и меньше операций записи, денормализация может существенно улучшить производительность за счёт того, что данные можно прочитать быстрее из одной таблицы.

Когда денормализация полезна:

- **Системы аналитики (OLAP)**: Когда нужно быстро агрегировать большие объемы данных для отчётов, лучше работать с денормализованной таблицей, чем собирать данные из множества нормализованных таблиц.
- **Высоконагруженные системы (чтение)**: Когда система в основном выполняет запросы чтения и меньше операций записи, денормализация помогает уменьшить сложность запросов и ускорить их

выполнение.

Пример денормализации:

В нормализованной базе данных мы могли бы хранить данные о заказах в двух таблицах:

Таблица orders:

order_id	customer_id	order_date
1	101	2023-09-10
2	102	2023-09-11

Таблица customers:

customer_id	customer_name
101	Alice
102	Bob

Для того чтобы получить информацию о заказе, нужно сделать `JOIN` между двумя таблицами. Денормализация означает, что мы можем объединить данные о заказах и клиентах в одну таблицу:

Таблица denormalized_orders:

order_id	customer_id	customer_name	order_date
1	101	Alice	2023-09-10
2	102	Bob	2023-09-11

Теперь нам не нужно выполнять `JOIN` для получения полной информации о заказе.

Преимущества денормализации:

1. **Более быстрые запросы**: Меньше операций `JOIN`, что ускоряет выборку данных.
2. **Упрощение запросов**: Запросы становятся проще и понятнее.
3. **Улучшение производительности в системах с интенсивными операциями чтения**.

Недостатки денормализации:

1. **Дублирование данных**: При денормализации данные могут дублироваться в нескольких таблицах, что увеличивает размер базы данных.
2. **Сложность управления данными**: При изменении данных нужно обновлять дублирующиеся данные в разных местах, что может привести к несогласованности данных.
3. **Затраты на запись**: Денормализация может замедлить операции записи, так как одна и та же информация должна обновляться в нескольких местах.

▼ Плюсы и Минусы

Плюсы нормализации для производительности

+ Уменьшение избыточности данных:

- Нормализация данных устраняет дублирование, что значительно снижает общий объём хранимых данных. Меньший объём данных приводит к уменьшению объёма хранимых таблиц и индексов, что помогает увеличить скорость выполнения запросов, так как меньше данных нужно обрабатывать.
- Это также ускоряет операции записи, потому что обновления касаются только одной таблицы, а не нескольких.

+ Упрощённое поддержание данных:

- Когда данные нормализованы, изменение одной записи в таблице автоматически корректирует все связанные данные. Например, обновление имени клиента в одной таблице будет сразу видимо при выполнении запроса в связанных таблицах.
- Это уменьшает вероятность появления ошибок и несогласованных данных, что особенно важно в высоконагруженных системах, где ведётся большое количество параллельных операций.

+ Гибкость и масштабируемость:

- Нормализация делает структуру базы данных более гибкой и масштабируемой. Когда таблицы разделены и связи между ними чётко определены, добавление новых атрибутов или сущностей в систему не приводит к значительным изменениям существующей структуры.
- Это упрощает сопровождение базы данных в долгосрочной перспективе.

Минусы нормализации для производительности

- Увеличение количества `JOIN`ов:

- Основной недостаток нормализации — это необходимость частого использования `JOIN`**ов** для объединения данных из разных таблиц.
  - Например, если данные клиента разделены на три таблицы (данные о клиентах, адреса, заказы), каждый запрос на получение полной информации о клиенте потребует объединения всех этих таблиц.
  - Чем больше нормализована база данных, тем больше приходится объединять таблицы, что замедляет выполнение запросов, особенно если они работают с большими объёмами данных.
- В высоконагруженных системах с большим количеством запросов чтения это может быть серьёзной проблемой.

- Задержки при выполнении операций чтения:

- В системах с высокой нагрузкой на чтение (например, аналитические системы или системы отчётности) **нормализация может снижать производительность** из-за необходимости частого соединения данных.
- Для таких систем лучше работает **денормализация**, которая минимизирует количество соединений и позволяет быстрее получать необходимые данные из одной таблицы.

- Трудности с кэшированием:

- В нормализованных базах данных данные находятся в разных таблицах, и это может усложнить эффективное кэширование запросов. Когда запросы используют несколько таблиц, кэшированная информация теряет свою актуальность быстрее, так как изменение одной таблицы может потребовать обновления кэша для всех связанных данных.
- В денормализованных системах, где информация хранится в одной таблице, кэширование становится более эффективным, так как данные изменяются реже.

Баланс между нормализацией и производительностью

3.1. Денормализация как компромисс:

- В некоторых случаях, чтобы повысить производительность запросов, может потребоваться **денормализация** данных. Это процесс, при котором данные, которые в нормализованной форме должны быть разделены на несколько таблиц, намеренно дублируются в одной таблице для ускорения запросов чтения.
- Денормализация полезна, когда в системе преобладают операции чтения и требуется минимизировать количество операций `JOIN`. Например, в системах отчетности или бизнес-аналитики, где скорость получения данных критична.

Пример денормализации:

Если у нас есть нормализованные таблицы `orders`, `customers` и `products`, чтобы получить информацию о заказе (например, имя клиента и продукт), нам нужно выполнить `JOIN`:

```
SELECT o.order_id, c.customer_name, p.product_name
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN products p ON o.product_id = p.product_id;
```

Но если мы денормализуем данные и храним всю информацию в одной таблице `order_details`, запрос станет проще:

```
SELECT order_id, customer_name, product_name
FROM order_details;
```

Это уменьшает нагрузку на процессор и ускоряет выполнение запроса, так как мы избегаем выполнения `JOIN`-ов.

3.2. Индексация как способ повысить производительность нормализованной базы:

- Один из способов повысить производительность в нормализованной базе данных без денормализации — это **эффективное использование индексов**. Индексы позволяют ускорить операции поиска и фильтрации данных.
- Важно создавать индексы на столбцы, которые часто участвуют в фильтрации (`WHERE`), соединениях (`JOIN`) и сортировках (`ORDER BY`), что поможет значительно ускорить выполнение запросов даже в

нормализованной базе данных.

3.3. Материализованные представления:

- В некоторых системах можно использовать **материализованные представления (materialized views)** для оптимизации запросов. Это физически сохранённый результат сложного запроса (например, `JOIN` нескольких таблиц), который периодически обновляется.
- Это помогает избегать повторных `JOIN` операций на лету при каждом запросе, улучшая производительность.

---

4. Когда использовать нормализацию, а когда денормализацию?

4.1. Когда нормализация предпочтительнее:

- **Операции с высокой нагрузкой на запись**: Нормализация помогает сократить объём данных, который нужно изменять при записи, что ускоряет операции записи. Например, при обновлении данных клиента вам нужно изменить только одну запись в таблице клиентов, а не несколько строк в разных таблицах.
- **Гарантия целостности данных**: В системах, где важна строгая целостность данных, нормализация помогает уменьшить дублирование и вероятность расхождения данных.

4.2. Когда денормализация предпочтительнее:

- **Операции с высокой нагрузкой на чтение**: Денормализация эффективна в системах, где основная нагрузка приходится на чтение данных (например, системы аналитики, отчётности или системы с интенсивным поиском данных).
- **Минимизация сложных `JOIN`ов**: Если данные часто запрашиваются в одном и том же виде, а выполнение `JOIN` замедляет работу системы, денормализация может ускорить запросы за счёт уменьшения количества таблиц.
