# **Основы проектирования баз данных**

## **Создание базы данных и таблиц**

Качество проектирования базы данных может влиять на работу с ней. С хорошо спроектированной базой данных легче работать, легче писать к ней запросы. И в данном руководстве мы рассмотрим основные принципы проектирования баз данных.

Для качественного проектирования базы данных существуют различные методики, различные последовательности шагов или этапов, которые во многом похожи. И в целом мы можем выделить следующие этапы:

1. Выделение сущностей и их атрибутов, которые будут храниться в базе данных, и формирование по ним таблиц. Атомизация сложных атрибутов на более простые.
2. Определение уникальных идентификаторов (первичных ключей) объектов, которые хранятся в строках таблицы
3. Определение отношений между таблицами с помощью внешних ключей
4. Нормализация базы данных

На первом этапе происходит выделение сущностей. **Сущность** (entity) представляет тип объектов, которые должны храниться в базе данных. Каждая таблица в базе данных должна представлять одну сущность. Как правило, сущности соответствуют объектам из реального мира.

У каждой сущности определяют набор атрибутов. **Атрибут** представляет свойство, которое описывает некоторую характеристику объекта.

Каждый столбец должен хранить один атрибут сущности. А каждая строка представляет отдельный объект или экземпляр сущности.

### **Восходящий и нисходящий подходы**

При проектировании базы данных на этапе выделения сущностей и их атрибутов мы можем использовать два подхода: восходящий и нисходящий.

Восходящий подход предусматривает выделение необходимых атрибутов, которые надо сохранить в бд. Затем выделенные атрибуты группируются в сущности, для которых впоследствии создается таблицы. Такой подход больше подходит для проектирования небольших баз данных с небольшим количеством атрибутов.

Например, нам дана следующая информация:

Том посещает курс по математике, который преподает профессор Смит.

Сэм посещает курс по математике, которые преподает профессор Смит.

Том посещает курс по языку JavaScript, который преподает ассистент Адамс.

Боб посещает курс по алгоритмам, который преподает ассистент Адамс.

Сэм имеет следующие электронный адрес sam@gmail.com и телефон +1235768789.

Какие данные из этой информации мы можем сохранить: имя студента, название курса, учебная должность преподавателя, имя преподавателя, электронный адрес студента.

Затем мы можем выполнить группировку по сущностям, к которым относятся эти данные:

|  |  |  |
| --- | --- | --- |
| Студент | Преподаватель | Курс |
| Имя студента  Название курса  Дата рождения студента  Электронный адрес студента  Телефон студента | Имя преподавателя  Должность преподавателя  Название курса | Имя студента  Имя преподавателя  Название курса |

Так, те данные, которые имеются позволяют выделить три сущности: студент, преподаватель и курс. При этом мы вполне можем добавлять какие-то недостающие данные. Также следует отметить, что какие-то данные могут иметь отношение к разным сущностям. Например, курс хранит информацию о студенте, которые его посещает. А студент хранит информацию о посещаемом курсе. Подобная избыточность данных решается на последующих шагах проектирования в процессе нормализации базы данных.

Но подобных атрибутов может оказаться очень много: сотни и даже тысячи. И в этом случае более оптимальным будет нисходящий подход. Данный подход подразумевает выявление сущностей. Затем происходит анализ сущностей, выявляются связи между ними, а потом и атрибуты сущностей.

То есть в данном случае мы могли бы сразу определить, что нам надо хранить данные по студентам, курсам и преподавателям. Затем в рамках каждой сущности выявить атрибуты

Например, у сущности "Студент" мы могли бы выделить такие атрибуты, как имя студента, его адрес, телефон, рост, вес, год его рождения. В тоже время нам надо учитывать не вообще все свойства, которые в принципе могут быть у сущности "Студент", а только те, которые имеют значение в рамках описываемой системы. Вряд ли в данном случае играют роль такие свойства как рост или вес студента, поэтому мы можем их вычеркнуть из списка атрибутов при проектировании таблицы.

Иногда подходы комбинируются. Для описания разных частей системы могут использоваться разные подходы. А затем их результаты объединяются.

### **Атомизация атрибутов**

При определении атрибутов происходит разделение сложных комплексных элементов на более простые. Так, в случае с именем студента мы можем его разбить на собственно имя и фамилию. Это позволит впоследствии выполнять операции с эти подэлементами отдельно, например, сортировать студентов только по фамилии.

То же самое касается адреса - мы можем сохранить весь адрес целиком, а можем разбить его на части - дом, улицу, город и т.д.

В то же время возможность разделения одного элемента на подэлементы не всегда может быть востребованной. В ряде задач это может быть просто не нужно. Выделять необходимо только те элементы, которые действительно нужны.

В соответствии с этим аспектом мы можем выделить у сущности "Студент" следующие атрибуты: имя студента, фамилия студента, год рождения, город, улица, дом, телефон.

### **Домен**

Каждый атрибут имеет **домен** (domain). Домен представляет набор допустимых значений для одного или нескольких атрибутов. По сути домен определяет смысл и источник значений, которые могут иметь атрибуты.

Домены могут отличаться для разных атрибутов, но также несколько атрибутов могут иметь один домен.

Например, выше были определены атрибуты сущности Студент. Определим используемые домены:

* **Имя**. Домен представляет все возможные имена, которые могут использоваться. Каждое имя представляет строку длиной максимум 20 символов (маловероятно, что нам могут встретиться имена свыше 20 символов).
* **Фамилия**. Домен представляет все возможные фамилии, которые могут использоваться. Каждая фамилия представляет строку длиной максимум 20 символов.
* **Год рождения**. Домен представляет все года рождения. Каждый год является числовым значением от 1950 до 2017.
* **Город**. Домен представляет все города текущей страны. Каждый город представляет строку длиной максимум 50 символов.
* **Улица**. Домен представляет все улицы текущей страны. Каждая улица представляет строку длиной максимум 50 символов.
* **Дом**. Домен представляет все возможные номера домов. Каждый номер дома является числом от 1 до, скажем, 10000.
* **Телефон**. Домен представляет все возможные телефонные номера. Каждый номер является строкой длиной в 11 символов.

Определяя домен, мы сразу видим, какие данные и каких типов будут хранить атрибуты. Какое-то другое значение, которое не соответствует домену, атрибут иметь не может.

В примере выше каждый атрибут имеет свой домен. Но, домены могут совпадать. Например, если бы сущность содержала бы следующие два атрибута: город рождения и город проживания, то домен бы совпадал и был бы одним и тем же для обоих атрибутов.

### **Определитель NULL**

При определении атрибутов и их домена необходимо проанализировать, а может ли у атрибута отсутствовать значение. Определитель NULL позволяет задать отсутствие значения. Например, в примере выше у студента обязательно должно быть какое-либо имя, поэтому недопустима ситуация, когда у атрибута, который представляет имя, отсутствует значение.

В то же время студент может не иметь телефонного номера или в рамках системы телефон не обязателен. Поэтому на этапе проектирования таблицы можно указать, что данный атрибут позволяет значение NULL.

Как правило, большинство современных реляционных СУБД поддерживают определитель NULL и позволяют задать его допустимость для столбца таблицы.

## **Ключи**

Ключи представляют способ идентификации строк в таблице. С помощью ключей мы также можем связывать строки между различными таблицами в отношения.

### **Суперключ**

Superkey (суперключ) - комбинация атрибутов (столбцов), которые уникально идентифицируют каждую строку таблицы. Это могут быть и все столбцы, и несколько и и один. При этом строки, которые содержат значения этих атрибутов, не должны повторяться.

Например, у нас есть сущность Student, которая представляет данные о пользователях и которая имеет следующие атрибуты:

* FirstName (имя)
* LastName (фамилия)
* Year (год рождения)
* Phone (номер телефона)

Какие атрибуты в данном случае могут составлять суперключ:

* {FirstName, LastName, Year, Phone}
* {FirstName, Year, Phone}
* {LastName, Year, Phone}
* {FirstName, Phone}
* {LastName, Phone}
* {Year, Phone}
* {Phone}

Каждого студента уникально может идентифицировать телефонный номер, поэтому любые наборы, в которых встречается атрибут Phone, представляют суперключ.

А вот, к примеру, набор {FirstName, LastName, Year} не является суперключом, так как у нас теоретически могут быть как минимум два студента с одинаковыми именем, фамилией и годом рождения.

### **Потенциальный ключ**

Candidate key (потенциальный ключ) - представляет собой минимальный суперключ отношения (таблицы), то есть набор атрибутов, который удовлетворяет ряду условий:

* **Неприводимость**: он не может быть сокращен, он содержит минимально возможный набор атрибутов
* **Уникальность**: он должен иметь уникальные значения вне зависимости от изменения строки
* **Наличие значения**: он не должен иметь значения NULL, то есть он обязательно должен иметь значение.

Возьмем ранее выделенные суперключи и найдем среди них candidate key. Первый пять суперключей не соответствуют первому условию, так как все их можно сократить до суперключа {Phone}:

* {FirstName, LastName, Year, Phone}
* {FirstName, Year, Phone}
* {LastName, Year, Phone}
* {FirstName, Phone}
* {LastName, Phone}
* {Year, Phone}

Суперключ {Phone} соответствует первому и второму условию, так как он имеет уникальное значение (в данном случае все пользователи могут иметь только уникальные телефонные номера). Но соответствует ли он третьему условию? В целом нет, так как теоретически студент может и не иметь телефона. В этом случае атрибут Phone будет иметь значение NULL, то есть значение будет отсутствовать.

В то же время это может зависеть от ситуации. Если в какой-то систему номер телефона является неотъемлемым атрибутом, например, используется для регистрации и входа в систему, то его можно считать потенциальным ключом. Но в данном случае мы рассматриваем общую ситуацию. И для понимания потенциального ключа необходимо отталкиваться от конкретной системы, которую описывает база данных.

И в таком случае суперключи таблицы не содержат потенциального ключа.

### **Первичный ключ**

Первичный ключ (primary key) непосредственно применяется для идентификации строк в таблице. Он должен соответствовать следующим ограничениям:

* Первичный ключ должен быть уникальным все время
* Он должен постоянно присутствовать в таблице и иметь значение
* Он не должен часто менять свое значение. В идеале он вообще не должен изменять значение.

Как правило, первичный ключ представляет один столбец таблицы, но также может быть составным и состоять из нескольких столбцов.

Если для таблицы можно выделить потенциальный ключ, то его можно использовать в качестве первичного ключа.

Если же потенциальные ключи отсутствуют, то для первичного ключа можно добавить к сущности специальный атрибут, который, как правило, называется, Id или имеет форму *[Имя\_сущности]Id* (например, StudentId), либо может иметь другое название. И обычно данный атрибут принимает целочисленное значение, начиная с 1.

Если же у нас есть несколько потенциальных ключей, то те потенциальные ключи, которые не составляют первичный ключ, являются **альтернативными ключами** (alternative key).

Например, возьмем представление пользователей на сайтах с двухфакторной авторизацией, где нам обязательно иметь электронный адрес, который нередко выступает в качестве логина, и какой-нибудь номер телефона. В этом случае таблицу пользователей мы можем задать с помощью следующих атрибутов:

* Name (имя пользователя)
* Email (электронный адрес)
* Password (пароль)
* Phone (телефонный номер)

В данном случае атрибуты Email и Phone являются потенциальными ключами, они обязательны в рамках рассматриваемой системы и в принципе уникальны. И теоретически, мы можем использовать один из этих атрибутов в качестве первичного ключа, тогда второй будет альтернативным ключом. Однако опять же поскольку теоретически значения обоих атрибутов могут меняться, то лучше все таки определить дополнительный атрибут специально под первичный ключ.

## **Внешние ключи и связи**

Базы данных могут содержать таблицы, которые связаны между собой различными связями. Связь (relationship) представляет ассоциацию между сущностями разных типов.

При выделении связи выделяют главную или родительскую таблицу (primary key table / master table) и зависимую, дочернюю таблицу (foreign key table / child table). Дочерняя таблица зависит от родительской.

Для организации связи используются внешние ключи. Внешний ключ представляет один или несколько столбцов из одной таблицы, который одновременно является потенциальным ключом из другой таблицы. Внешний ключ необязательно должен соответствовать первичному ключу из главной таблицы. Хотя, как правило, внешний ключ из зависимой таблицы указывает на первичный ключ из главной таблицы.

Связи между таблицами бывают следующих типов:

* **Один к одному** (One to one)
* **Один к многим** (One to many)
* **Многие ко многим** (Many to many)

### **Связь один к одному**

Данный тип связей встречает не часто. В этом случае объекту одной сущности можно сопоставить только один объект другой сущности. Например, на некоторых сайтах пользователь может иметь только один блог. То есть возникает отношение один пользователь - один блог.

Нередко этот тип связей предполагает разбиение одной большой таблицы на несколько маленьких. Основная родительская таблица в этом случае продолжает содержать часто используемые данные, а дочерняя зависимая таблица обычно хранит данные, которые используются реже.

В этом отношении первичный ключ зависимой таблицы в то же время является внешним ключом, который ссылается на первичный ключ из главной таблицы.

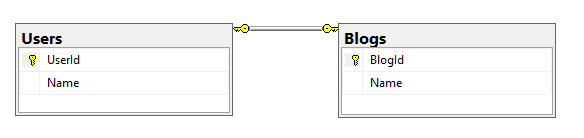
Например, таблица Users представляет пользователей и имеет следующие столбцы:

* UserId (идентификатор, первичный ключ)
* Name (имя пользователя)

И таблица Blogs представляет блоги пользователей и имеет следующие столбцы:

* BlogId (идентификатор, первичный и внешний ключ)
* Name (название блога)

В этом случае столбец BlogId будет хранить значение из столбца UserId из таблицы пользователей. То есть столбец BlogId будет выступать одновременно первичным и внешним ключом.



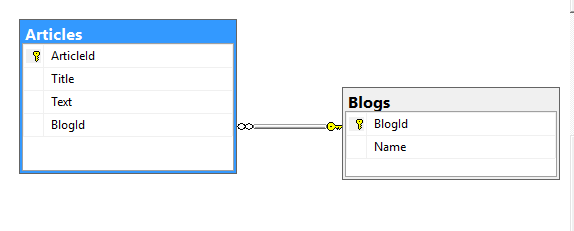
### **Связь один ко многим**

Это наиболее часто встречаемый тип связей. В этом типе связей несколько строк из дочерний таблицы зависят от одной строки в родительской таблице. Например, в одном блоге может быть несколько статей. В этом случае таблица блогов является родительской, а таблица статей - дочерней. То есть один блог - много статей. Или другой пример, в футбольной команде может играть несколько футболистов. И в то же время один футболист одновременно может играть только в одной команде. То есть одна команда - много футболистов.

К примеру, пусть будет таблица Articles, которая представляет статьи блога и которая имеет следующие столбцы:

* ArticleId (идентификатор, первичный ключ)
* BlogId (внешний ключ)
* Title (название статьи)
* Text (текст статьи)

В этом случае столбец BlogId из таблицы статей будет хранить значение из столбца BlogId из таблицы блогов.



### **Связь многие ко многим**

При этом типе связей одна строка из таблицы А может быть связана с множеством строк из таблицы В. В свою очередь одна строка из таблицы В может быть связана с множеством строк из таблицы А. Типичный пример - студенты и курсы: один студент может посещать несколько курсов, и соответственно на один курс могут записаться несколько студентов.

Другой пример - статьи и теги: для одной статьи можно определить несколько тегов, а один тег может быть определен для нескольких статей.

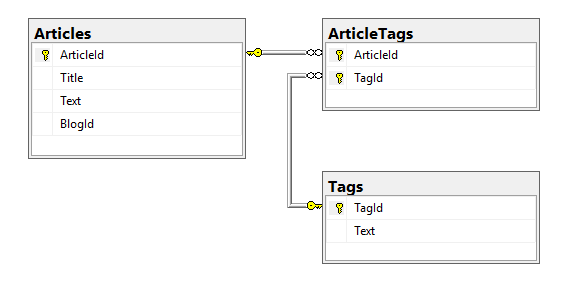
Но в SQL Server на уровне базы данных мы не можем установить прямую связь многие ко многим между двумя таблицами. Это делается посредством вспомогательной промежуточной таблицы. Иногда данные из этой промежуточной таблицы представляют отдельную сущность.

Например, в случае со статьями и тегами пусть будет таблица Tags, которая имеет два столбца:

* TagId (идентификатор, первичный ключ)
* Text (текст тега)

Также пусть будет промежуточная таблица ArticleTags со следующими полями:

* TagId (идентификатор, первичный и внешний ключ)
* ArticleIdId (идентификатор, первичный и внешний ключ)



Технически мы получим две связи один-ко-многим. Столбец TagId из таблицы ArticleTags будет ссылаться на столбец TagId из таблицы Tags. А столбец ArticleId из таблицы ArticleTags будет ссылаться на столбец ArticleId из таблицы Articles. То есть столбцы TagId и ArticleId в таблице ArticleTags представляют составной первичный ключ и одновременно являются внешними ключами для связи с таблицами Articles и Tags.

### **Ссылочная целостность данных**

При изменении первичных и внешних ключей следует соблюдать такой аспект как **ссылочная целостность данных** (referential integrity). Ее основная идея состоит в том, чтобы две таблице в базе данных, которые хранят одни и те же данные, поддерживали их согласованность. Целостность данных представляет правильно выстроенные отношения между таблицами с корректной установкой ссылок между ними. В каких случаях целостность данных может нарушаться:

* **Аномалия удаления** (deletion anomaly). Возникает при удалении строки из главной таблицы. В этом случае внешний ключ из зависимой таблицы продолжает ссылаться на удаленную строку из главной таблицы
* **Аномалия вставки** (insertion anomaly). Возникает при вставке строки в зависимую таблицу. В этом случае внешний ключ из зависимой таблицы не соответствует первичному ключу ни одной из строк из главной таблицы.
* **Аномалии обновления** (update anomaly). При подобной аномалии несколько строк одной таблицы могут содержать данные, которые принадлежат одному и тому же объекту. При изменении данных в одной строке они могу прийти в противоречие с данными из другой строки.

#### **Аномалия удаления**

Для решения аномалии удаления для внешнего ключа следует устанавливать одно из двух ограничений:

* Если строка из зависимой таблицы обязательно требует наличия строки из главной таблицы, то для внешнего ключа устанавливается каскадное удаление. То есть при удалении строки из главной таблицы происходит удаление связанной строки (строк) из зависимой таблицы.
* Если строка из зависимой таблицы допускает отсутствие связи со строкой из главной таблицы (то есть такая связь необязательна), то для внешнего ключа при удалении связанной строки из главной таблицы задается установка значения NULL. При этом столбец внешнего ключа должен допускать значение NULL.

#### **Аномалия вставки**

Для решения аномалии вставки при добавлении в зависимую таблицу данных столбец, который представляет внешний ключ, должен допускать значение NULL. И таким образом, если добавляемый объект не имеет связи с главной таблицей, то в столбце внешнего ключа будет стоять значение NULL.

#### **Аномалии обновления**

Для решения проблемы аномалии обновления применяется нормализация, которая будет рассмотрена далее.

# **Нормализация**

Нормализация представляет процесс разделения данных по отдельным связанным таблицам. Нормализация устраняет избыточность данных (data redundancy) и тем самым избежать нарушения целостности данных при их изменении, то есть избежать аномалий изменения (update anomaly).

Как правило, нормализация преимущественно применяется при восходящем подходе проектировании базы данных, то есть когда мы все атрибуты, которые надо сохранить в бд, группируем по сущностям, для которых затем создаются таблицы. Однако при нисходящем подходе, когда вначале выявляются сущности, а затем их атрибуты и связи между ними, нормализация также может применяться, например, для проверки корректности спроектированных таблиц.

В ненормализованной форме таблица может хранить информацию о двух и более сущностях. Также она может содержать повторяющиеся столбцы. Также столбцы могут хранить повторяющиеся значения. В нормализованной же форме каждая таблица хранит информацию только об одной сущности.

Нормализация предполагает применение нормальных форм к структуре данных. Существуют 7 нормальных форм. Каждая нормальная форма (за исключением первой) подразумевает, что к данным уже была применена предыдущая нормальная форма. Например, прежде чем применить третью нормальную форму к данным должна быть применена вторая нормальная форма. И строго говоря, база данных считается нормализованной, если к ней применяется третья нормальная форма и выше.

Первая нормальная форма (1NF) предполагает, что сохраняемые данные на пересечении строк и столбцов должны представлять скалярное значение, а таблицы не должны содержать повторяющихся строк.

Вторая нормальная форма (2NF) предполагает, что каждый столбец, не являющийся ключом, должен зависеть от первичного ключа.

Третья нормальная форма (3NF) предполагает, что каждый столбец, не являющийся ключом, должен зависеть только от первичного ключа.

Нормальная форма Бойса-Кодда (BCNF) является немного более строгой версией третьей нормальной формы.

Четвертая нормальная форма (4NF) применяется для устранения многозначных зависимостей (multivalued dependencies) - таких зависимостей, где столбец с первичным ключом имеет связь один-ко-многим со столбцом, который не является ключом. Эта нормальная форма устраняет некорректные отношения многие-ко-многим.

Пятая нормальная форма (5NF) разделяет таблицы на более малые таблицы для устранения избыточности данных. Разбиение идет до тех пор, пока нельзя будет воссоздать оригинальную таблицу путем объединения малых таблиц.

Шестая нормальная форма (domain key normal form / 6NF). Каждое ограничение в связях между таблицами должно зависеть только от ограничений ключа и ограничений домена, где домен представляет набор допустимых значений для столбца. Эта форма предотвращает добавление недопустимых данных путем установки ограничения на уровне отношений между таблицами, но не на уровне таблиц или столбцов. Данная форма, как правило, не применима на уровне СУБД, в том числе и в SQL Server.

## **Функциональная зависимость**

Ключевым понятием нормализации является **функциональная зависимость**. Функциональная зависимость описывает связь между атрибутами отношения. Например, если атрибут В функционально зависит от атрибута А (А → В), то каждое значение атрибута А связано только с одним значением атрибута В. Причем атрибуты А и В могут состоять из одного или нескольких атрибутов. То есть, если две строки имеют одно и то же значение атрибута А, то они обязательно имеют одно и то же значение атрибута В. При этом для одного значения атрибута В могут существовать несколько различных значений атрибута А. Атрибут А в этой зависимости еще называется детерминантом.

Например, возьмем следующую таблицу, которая представляет университетские курсы:

|  |  |  |
| --- | --- | --- |
| Course | Teacher | Position |
| Математика | Смит | Профессор |
| Алгориты | Адамс | Ассистент |
| JavaScript | Адамс | Ассистент |

Здесь атрибут Teacher функционально **зависит** от атрибута Course (Course → Teacher). То есть зная название курса, мы можем определить его преподавателя. И в этом случае можно говорить, что между атрибутами Course и Teacher есть связь 1:1, а между Teacher и Course связь 1:N, так как есть несколько курсов, которые может вести один преподаватель. При этом атрибут Course функционально **не зависит** от атрибута Teacher.

Кроме того, здесь можно проследить еще две функциональных зависимости. В частности, атрибут Position зависит от атрибута Teacher (Teacher → Position). Зная имя преподавателя, мы можем определить его должность.

И также атрибут Position функционально зависит от атрибута Course - зная название курса, мы можем сказать должность преподавателя.

В таблице в нормализованной базе данных единственным детерминантом должен быть атрибут, который является первичным ключом. А все остальные атрибуты должны функционально зависеть от первичного ключа.

Например, в данном случае мы можем взять в качестве первичного ключа название курса с учетом, что курсы могут иметь только уникальные названия. Однако должность преподавателя в данном случае будет зависеть сразу от двух атрибутов - от Course и Teacher. И подобные зависимости могут свидетельствовать о том, что база данных и конкретно таблица курсов имеет недостатки в проектировании и может быть источником аномалий обновления.

## **Первая нормальная форма**

Первая нормальная форма предполагает, что таблица не должна содержать повторяющихся столбцов или таких столбцов, которые содержат наборы значений. Ненормализованная таблица в этом случае может содержать одну или несколько повторяющихся групп данных. Повторяющаяся группа - это группа из одного или нескольких атрибутов таблицы, в которой возможно наличие нескольких значений для ключевого атрибута таблицы.

Итогом применения первой формы должно стать наличие для одного атрибута сущности только одного столбца в таблице, который при этом должен содержать скалярное значение.

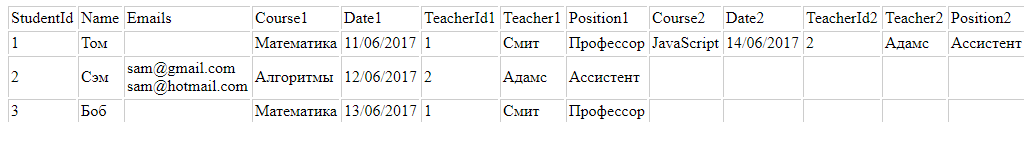
Есть два похода к переходу к ненормализованной таблицы к первой нормальной форме. Первый способ называется выравниванием или flattaning. Он предполагает декомпозицию строки с повторяющимися группами данных, при котором для каждой повторяющейся группы создается своя строка. Полученная в результате таблица будет содержать атомарные значения для каждого из атрибутов. Хотя в то же время этот подход увеличит избыточность данных.

Второй подход предполагает, что один атрибут или группа атрибутов назначаются ключом ненормализованной таблицы, а затем повторяющиеся группы удаляются из таблицы и помещаются в отдельную таблицу вместе с копиями ключа из исходной таблицы.

Рассмотрим применение нормализации на примере. Пусть у нас есть система, которая описывается следующей информацией:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Том посещает курс по математике, который преподает профессор Смит. Дата записи 11/06/2017.  Сэм посещает курс по алгоритмам, которые преподает ассистент Адамс. Дата записи 12/06/2017.  Боб посещает курс по математике, который преподает профессор Смит. Дата записи 13/06/2017.  Том посещает курс по языку JavaScript, который преподает ассистент Адамс. Дата записи 14/06/2017.  Сэм имеет два электронных адреса: sam@gmail.com и sam@hotmail.com.  В университете может быть только один курс с определенным именем. Один преподаватель может преподавать несколько курсов. |

Вначале определим ненормализованную таблицу StudentCourses, которая содержит всю эту информацию:



Для каждого студента определен уникальный идентификатор StudentId, а также атрибут Name (имя), Emails (все электронные адреса), Course1 /Course2(курс), Date1/Date2 (дата поступления), Teacher1/Teacher2 (преподаватель), Position1/Position2 (должность преподавателя). Также чтобы различать преподавателей (так как теоретически могут быть преподаватели с одной и той же фамилией), добавлен атрибут TeacherId1/TeacherId2. Для курсов такой идентификатор не требуется, так как в нашем случае название курса уникально.

Поскольку Том записан сразу на два курса, то несколько атрибутов пришлось дублировать. Но что будет, когда Том в стремлении получить никому не нужные сертификаты запишется еще на десяток курсов?

Эта таблица представляет прекрасный пример отклонения от первой нормальной формы. В первую очередь мы видим группу повторяющихся атрибутов, которые представляют данные по одному курсу: Course, Date, TeacherId, Teacher, Position. Эти атрибуты представляют повторяющуюся группу, которую можно условно назвать StudentCourse.

StudentCourse = (Course, Date, TeacherId, Teacher, Position)

Вторая проблема - атрибут Emails содержит набор электронных адресов. Фактически этот атрибут также образует повторяющуюся группу.

Для избавления от первой повторяющейся группы атрибутов применим первый подход: создадим для каждой повторяющейся группы отдельную строку.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| StudentId | Name | Emails | CourseId | Course | Date | TeacherId | Teacher | Position |
| 1 | Том |  | 1 | Математика | 11/06/2017 | 1 | Смит | Профессор |
| 1 | Том |  | 2 | JavaScript | 14/06/2017 | 2 | Адамс | Ассистент |
| 2 | Сэм | sam@gmail.com  sam@hotmail.com | 3 | Алгоритмы | 12/06/2017 | 2 | Адамс | Ассистент |
| 3 | Боб |  | 1 | Математика | 13/06/2017 | 1 | Смит | Профессор |

В данном случае увеличилась избыточность данных, но тем не менее мы избавились от повторяющейся группы. Также следует отметить, что теперь атрибут StudentId не может использоваться в качестве первичного ключа. И в данном случае просматривается только один потенциальный ключ, который и будет использоваться в качестве первичного - это сразу два столбца StudentId и Course. Но название курса - не лучший ключ, если учитывать, что это название может редактироваться и изменяться. Поэтому для каждого курса добавлен еще один атрибут - CourseId - уникальной номер курса, который вместе с StudentId составляет первичный ключ. Хотя в принципе может было бы и оставить в качестве части первичного ключа имя курса с учетом, что оно уникально.

Для избавления от второй повторяющейся группы - атрибута Emails применим второй подход: вынесение этой группы с копией ключа в отдельную таблицу. Для этого определим таблицу Emails:

|  |  |
| --- | --- |
| Email | StudentId |
| sam@gmail.com | 2 |
| sam@hotmail.com | 2 |

Так как электронный адрес в принципе уникален, то его можно сделать первичным ключом.

Таким образом, таблицы Emails с таблицей StudentCourses будет связана связью один ко многим (один студент - много электронных адресов). И в этом случае таблица StudentCourses сократится следующим образом:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| StudentId | Name | CourseId | Course | Date | TeacherId | Teacher | Position |
| 1 | Том | 1 | Математика | 11/06/2017 | 1 | Смит | Профессор |
| 1 | Том | 2 | JavaScript | 14/06/2017 | 2 | Адамс | Ассистент |
| 2 | Сэм | 3 | Алгоритмы | 12/06/2017 | 2 | Адамс | Ассистент |
| 3 | Боб | 1 | Математика | 13/06/2017 | 1 | Смит | Профессор |

Теперь у нас нет повторяющихся столбцов, но увеличилась избыточность данных, так как для студента Том определено уже две строки в таблице, и соответственно Id повторяется. Но тем не менее 1-я нормальная форма применена.

В принципе можно отметить, что если повторяющиеся группы содержат уникальные значения для каждой строки таблицы (как в случае с электронными адресами), то мы имеем дело с потенциальной связью один ко многим. Если же повторяющиеся группы содержат неуникальные значения, которые могут иметь разные строки таблицы (как в случае с атрибутами курсов), то это скрывается потенциальная связь многие ко многим.

## **Вторая нормальная форма**

Во второй нормальной форме каждый столбец в таблице, который не является ключом, **должен зависеть от ключа**.

Ключевой момент второй нормальной формы - полная функциональная зависимость. Она предполагает, что атрибут В полностью функционально зависим от атрибута А, если атрибут В функционально зависит от полного значения атрибута А, а не от какого-либо подмножества значений из атрибута А. То есть, если атрибут А составляют несколько значений, скажем, А1 и А2, то атрибут В полностью функционально зависит от А, если он зависит и от А1 и от А2 (А1, А2 → В).

Если атрибут В зависит только от какого-либо подмножества из атрибута А, например, только от А1, то имеет место частичная функциональная зависимость.

Эта форма применяется к тем таблицам, которые имеют составной первичный ключ, то есть где первичный ключ состоит из нескольких атрибутов. Если в таблице **несоставной** первичный ключ, то в этом случае считается, что все остальные атрибуты автоматически находятся в полной функциональной зависимости от первичного ключа.

Вторая нормальная форма применяется только к тем таблицам, которые находятся в первой нормальной форме. После применения второй формы все столбцы таблицы зависят от первичного ключа.

Возьмем сформированную в прошлой теме таблицу StudentCourses после применения первой нормальной формы:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| StudentId | Name | CourseId | Course | Date | TeacherId | Teacher | Position |
| 1 | Том | 1 | Математика | 11/06/2017 | 1 | Смит | Профессор |
| 1 | Том | 2 | JavaScript | 14/06/2017 | 2 | Адамс | Ассистент |
| 2 | Сэм | 3 | Алгоритмы | 12/06/2017 | 2 | Адамс | Ассистент |
| 3 | Боб | 1 | Математика | 13/06/2017 | 1 | Смит | Профессор |

На данный момент эта таблица имеет составной первичный ключ StudentId+CourseId. Какие функциональные зависимости от ключевых атрибутов здесь можно выделить:

StudentId, CourseId → Date

StudentId → Name

CourseId → Course, TeacherId, Teacher, Position

От обоих частей составного ключа StudentId+CourseId зависит только арибут Date - дата, в которую студент с идентификатором StudentId поступил на курс с идентифкатором CourseId.

Атрибут Name зависит только от части составного ключа - от атрибута StudentId, так как зная идентификатор студента, можно сказать, какое у него имя. В данном случае имеет факт частичной зависимости.

Атрибуты Course, TeacherId, Teacher, Position зависит от другой части ключа - от атрибута CourseId. Зная значение CourseId, можно сказать, как называется курс, какой у курса преподаватель, какую должность он занимает. Опять же здесь частичная зависимость.

Наличие частичных зависимостей говорит о том, что таблица не находится во второй нормальной форме. И для перехода к этой форме необходимо переместить атрибуты, которые не входят в первичный ключ, в новую таблицу вместе с копией части первичного ключа, от которой они функционально зависят.

В нашем случае из одной таблицы получатся три. Таблица Students:

|  |  |
| --- | --- |
| StudentId | Name |
| 1 | Том |
| 2 | Сэм |
| 3 | Боб |

Таблица Courses:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CourseId | Course | TeacherId | Teacher | Position |
| 1 | Математика | 1 | Смит | Профессор |
| 2 | JavaScript | 2 | Адамс | Ассистент |
| 3 | Алгоритмы | 2 | Адамс | Ассистент |

И таблица StudentCourses:

|  |  |  |
| --- | --- | --- |
| StudentId | CourseId | Date |
| 1 | 1 | 11/06/2017 |
| 1 | 2 | 14/06/2017 |
| 2 | 3 | 12/06/2017 |
| 3 | 1 | 13/06/2017 |

Итогом стало образование связи многие ко многим (много студентов - много курсов) между таблицами Students и Courses через таблицу StudentCourses .

Таким образом, база данных перешла во вторую нормальную форму.

## **Третья нормальная форма**

Третья нормальная форма предполагает, что каждый столбец, не являющийся ключом, должен зависеть **только** от столбца, который является ключом, то есть должна отсутствовать **транзитивная функциональная зависимость** (transitive functional dependency)

Транзитивная функциональная зависимость выражается следующим образом: А → В и В → С. То есть атрибут С транзитивно зависит от атрибута А, если атрибут С зависит от атрибута В, а атрибут В зависит от атрибута А (при условии, что атрибут А функционально не зависит ни от атрибута В, ни от атрибута С).

Если столбец зависит не только от первичного ключа, то данный столбец находится не в той таблице, в которой он должен находиться, либо же является производным от других столбцов.

Для нормализации из исходной таблицы те атрибуты, которые находятся в транзитивной зависимости от ключа, выносятся в отдельную таблицу с копией того атрибута, от которого они непосредственно зависят.

При применении третьей нормальной формы таблица должна находиться во второй нормальной форме. 3NF позволяет значительно снизить избыточность данных.

Для примера возьмем сформированную в прошлой теме таблицу Courses, которая содержит информацию о курсах и которая находится во второй нормальной форме:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CourseId | Course | TeacherId | Teacher | Position |
| 1 | Математика | 1 | Смит | Профессор |
| 2 | JavaScript | 2 | Адамс | Ассистент |
| 3 | Алгоритмы | 2 | Адамс | Ассистент |

Какие функциональные зависимости здесь можно выделить:

CourseId → Course, TeacherId, Teacher, Position

Course → CourseId, TeacherId, Teacher, Position

TeacherId → Teacher, Position

Вторая зависимость фактически аналогична первой и говорит о том, что атрибут Course является потенциальным ключом.

Третья зависимость говорит о том, что, зная идентификатор преподавателя, мы можем узнать его фамилию и должность. То есть через атрибут TeacherId атрибуты Teacher и Position зависят от CourseId (CourseId→ TeacherId и TeacherId → Teacher, Position). И в данном случае мы можем говорить о транзитивной зависимости Teacher, Position от CourseId.

Для нормализации необходимо вынести в отдельную таблицу атрибуты TeacherId, Teacher, Position. Для этого пусть будет отдельная таблица Teachers:

|  |  |  |
| --- | --- | --- |
| TeacherId | Teacher | Position |
| 1 | Смит | Профессор |
| 2 | Адамс | Ассистент |

А таблица Courses сократится следующим образом:

|  |  |  |
| --- | --- | --- |
| CourseId | Course | TeacherId |
| 1 | Математика | 1 |
| 2 | JavaScript | 2 |
| 3 | Алгоритмы | 2 |