MySQL

Ограничения, нормальные формы и связи

Проверка значения CHECK

Определение

Ограничение CHECK позволяет применить условие (его мы определяем сами, но оно должно включать проверки на `>`, `>=`, `<`, `<=`, `=`, `<>`, `!=`) для проверки значения, которое мы пытаемся вставить в столбец. Если проверка не будет пройдена, то данные не будут вставлены в таблицу, а MySQL вернет ошибку.

Пример создания ограничения CHECK

В столбец **experience** можно вставить только те числа, которые больше или равны 5:

```
CREATE TABLE developers (
    firstname VARCHAR(255) NOT NULL,
    lastname VARCHAR(255) NOT NULL,
    position VARCHAR(255) NOT NULL,
    age INT UNSIGNED NOT NULL,
    experience INT UNSIGNED NOT NULL CHECK(experience >= 5)
);
```

Применение нескольких условий в ограничении CHECK

В столбец **position** могут быть вставлены только строки 'programmer' или 'designer':

```
CREATE TABLE developers (
	firstname VARCHAR(255) NOT NULL,
	lastname VARCHAR(255) NOT NULL,
	position VARCHAR(255) NOT NULL CHECK(position = 'administrator' OR position = 'designer'),
	age INT UNSIGNED NOT NULL,
	experience INT UNSIGNED NOT NULL CHECK(experience >= 5)
);
```

Альтернативный пример создания ограничения CHECK

Ограничение **CHECK** можно вставлять не только рядом с тем столбцом, где оно используется, можно вставлять его так же, как вставляется сам столбец:

```
CREATE TABLE developers (
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        position VARCHAR(255) NOT NULL,
        age INT UNSIGNED NOT NULL,
        experience INT UNSIGNED NOT NULL,
        CHECK(position = 'administrator' OR position = 'designer'),
        CHECK(experience >= 5)
);
```

Перечисление ограничений CHECK всех столбцов в рамках одной конструкции

```
CREATE TABLE developers (
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        position VARCHAR(255) NOT NULL,
        age INT UNSIGNED NOT NULL,
        experience INT UNSIGNED NOT NULL,
        CHECK((position = 'administrator' OR position = 'designer') AND (experience >= 5))
);
```

Назначение имени ограничениям CHECK

Любому ограничению **CHECK** при его создании назначается имя. Если мы не ставим имя сами, то MySQL присваивает это имя автоматически. Однако лучше всего это делать явно, т.к. СУБД присваивает не слишком читаемые названия - для этого надо использовать ключевое слово **CONSTRAINT**:

```
CREATE TABLE developers (
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        position VARCHAR(255) NOT NULL,
        age INT UNSIGNED NOT NULL,
        experience INT UNSIGNED NOT NULL,
        CONSTRAINT position_limit CHECK(position = 'administrator' OR position =
'designer'),
        CONSTRAINT experience_limit CHECK(experience >= 5)
);
```

Добавление ограничения CHECK после создания таблицы

Ограничения CHECK можно создавать не только в момент создания таблицы, но и после того, как таблица уже существует. Как и ранее, можно создавать CHECK и с названием, которое назначит сама MySQL:

```
-- создание ограничения с явным назначением имени
ALTER TABLE developers ADD CONSTRAINT age_limit CHECK(age >= 18);

-- создание ограничения, которому название даст сама MySQL
ALTER TABLE developers ADD CHECK(age >= 18);
```

Удаление ограничения CHECK

Удаление ограничения делается при помощи его имени:

```
-- 1-й способ
ALTER TABLE developers DROP CHECK age_limit;

-- 2-й способ
ALTER TABLE developers DROP CONSTRAINT age_limit;
```

Проверка на уникальность UNIQUE

Определение

Ограничение **UNIQUE** следит за тем, чтобы в столбце таблицы не было повторяющихся значений - только уникальные. При попытке вставить значение, которое уже существует, MySQL будет возвращать ошибку и ничего вставлять не будет. Мы можем создать столько столбцов с ограничением **UNIQUE**, сколько захотим.

Пример создания ограничения UNIQUE

В столбец **personal_code** можно вставлять только уникальные значения:

```
CREATE TABLE citizens(
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        personal_code VARCHAR(255) NOT NULL UNIQUE
);
```

Назначение имени ограничениям UNIQUE

```
-- 1-й способ
CREATE TABLE citizens(
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        personal_code VARCHAR(255) NOT NULL,
        CONSTRAINT personal_code_unq UNIQUE(personal_code)
);

-- 2-й способ
CREATE TABLE citizens(
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        personal_code VARCHAR(255) NOT NULL,
        UNIQUE KEY personal_code_unq (personal_code)
);
```

UNIQUE и NULL

Если ограничение **UNIQUE** было присвоено столбцу, который разрешает вставлять в себя **NULL**, то в этот столбец значение **NULL** можно вставлять больше одного раза - т.е. в данном контексте **UNIQUE** не будет применять свою проверку. Дело в том, что **NULL** трактуется как “отсутствие значения”, т.е. оно просто игнорируется.

Пример использования UNIQUE и NULL в одной таблице

```
CREATE TABLE citizens(
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        personal_code VARCHAR(255) NULL UNIQUE
);
```

Ограничение UNIQUE для нескольких столбцов

UNIQUE может быть применено не только для одного столбца, а сразу для нескольких столбцов. Это значит, что комбинация значений этих столбцов должна быть уникальна, но значение какого-либо столбца из одной комбинации вполне может совпадать со значением такого же столбца из другой комбинации:

```
CREATE TABLE staff(
    employee VARCHAR(255) NOT NULL,
    position VARCHAR(255) NOT NULL,
    CONSTRAINT emp_pos_unq UNIQUE(employee, position)
);
```

Пример использования ограничения UNIQUE для нескольких столбцов

Таблица **staff** может быть заполнена следующим образом:

employee	position
John Smith	programmer
John Smith	designer
Mary Evans	programmer

Однако вот так заполнена таблица **staff** быть не может:

employee	position
John Smith	programmer
John Smith	programmer
Mary Evans	programmer

UNIQUE для нескольких столбцов и NULL

Если хотя бы один из столбцов, который используется для ограничения UNIQUE может содержать в себе NULL и содержит его, то в таблице могут быть одинаковые строки:

```
CREATE TABLE staff(
      employee VARCHAR(255) NULL,
      position VARCHAR(255) NULL,
      CONSTRAINT emp_pos_unq UNIQUE(employee, position)
);
```

В этом случае мы можем заполнить таблицу **staff** так:

employee	position
John Smith	NULL
John Smith	NULL
NULL	programmer
NULL	programmer

Добавление ограничения UNIQUE после создания таблицы

Пример для создания ограничения UNIQUE с явным присвоением имени:

```
CREATE TABLE staff(
      employee VARCHAR(255) NOT NULL,
      position VARCHAR(255) NOT NULL
);

-- создание ограничения для таблицы (явное указание имени)
ALTER TABLE staff ADD CONSTRAINT emp_pos_unq UNIQUE(employee, position);
```

Пример для создания ограничения UNIQUE без явного присвоения имени:

```
CREATE TABLE citizens(
   firstname VARCHAR(255) NOT NULL,
   lastname VARCHAR(255) NOT NULL,
   personal_code VARCHAR(255) NOT NULL
);

-- создание ограничения для таблицы (без указания имени)
ALTER TABLE citizens ADD UNIQUE(personal_code);
```

Удаление ограничения UNIQUE

```
-- 1-й способ
ALTER TABLE staff DROP KEY emp_pos_unq;

-- 2-й способ
ALTER TABLE staff DROP CONSTRAINT emp_pos_unq;
```

Ограничения внешнего ключа FOREIGN KEY

Определение

При применении ограничения **FOREIGN KEY** (поддерживается движком **InnoDB**) в столбец какой-либо таблице (или нескольких таблицах) можно вставить только то значение, какое уже вставлено в определенный столбец какой-то другой конкретной таблицы. Важный момент - на столбец таблицы с изначальными значениями обязательно должен быть поставлен так называемый индекс (**INDEX**). Подробнее об индексам мы поговорим в дальнейших разделах, но пока запомним, что индекс способствует ускорению поиска по таблице в контексте столбца, которому установлен этот индекс.

Столбец таблицы с изначальными значениями и столбец таблицы, которая ссылается на изначальную таблицу, должны иметь одинаковый тип - в противном случае MySQL не позволит создать ограничение внешнего ключа.

Пример создания ограничения FOREIGN KEY

```
CREATE TABLE clients(
    id INT UNSIGNED NOT NULL,
    firstname VARCHAR(255) NOT NULL,
    lastname VARCHAR(255) NOT NULL,
    INDEX(id)
);
```

```
CREATE TABLE orders(
    client_id INT UNSIGNED NOT NULL,
    reference VARCHAR(255) NOT NULL UNIQUE,
    price NUMERIC(20,2),
    FOREIGN KEY(client_id) REFERENCES clients(id)
);
```

Пример использования ограничения FOREIGN KEY

Предположим, что в таблице **clients** хранятся следующие записи:

id	firstname	lastname
1	John	Smith
2	Mary	Evans
3	Joanna	Brooks

При таких значениях в первой таблице во второй таблице в столбце **client_id** могут хранится только 1, 2, 3 - при попытке вставить в **client_id** другие значения MySQL вернет ошибку и ничего вставлять не будет. Далее пример таблицы **orders** с правильными значениями:

client_id	reference	price
1	23924384745	99.99
2	48499494432	115.75
3	93843746626	20.17
2	66884848848	65.00

Особенности взаимодействия FOREIGN KEY и NULL

Если столбцу таблицы, зависящему от столбца первоначальной таблицы, разрешить вставлять значение **NULL**, то в зависимый столбец можно будет вставлять значение, которое отлично от значений первоначального столбца - можно понять, что это значение **NULL**. Чтобы продемонстрировать наши утверждения, обновим столбец **client_id** таблицы **orders** - разрешим вставлять вышеупомянутый **NULL**:

```
ALTER TABLE orders MODIFY COLUMN client_id INT UNSIGNED NULL;
```

Теперь в таблице **orders** могут хранится следующие значения:

client_id	reference	price
1	23924384745	99.99
NULL	34563394432	46.22
NULL	16702746626	21.15

FOREIGN KEY и модификация значения первоначального столбца (1): RESTRICT

По умолчания ограничение внешнего ключа запрещает удаление и обновление значения первоначального столбца. Однако запрещение можно прописать явно при помощи слова **RESTRICT**. Данный пример запретит менять столбец **id** в таблице **clients**:

```
-- явный запрет удалять и обновлять значение id в clients
CREATE TABLE orders(
    client_id INT UNSIGNED NOT NULL,
    reference VARCHAR(255) NOT NULL UNIQUE,
    price NUMERIC(20,2),
    FOREIGN KEY(client_id) REFERENCES clients(id) ON DELETE RESTRICT ON UPDATE
RESTRICT
);
```

FOREIGN KEY и модификация значения первоначального столбца (2): CASCADE

При удалении и обновлении значения первоначального столбца существует возможность удалять всю строку из зависимой таблицы (при удалении значения первоначального столбца) или обновлять значение зависимого столбца (при обновлении значения первоначального столбца). Производится при помощи слова **CASCADE**:

```
CREATE TABLE orders(
    client_id INT UNSIGNED NOT NULL,
    reference VARCHAR(255) NOT NULL UNIQUE,
    price NUMERIC(20,2),
    FOREIGN KEY(client_id) REFERENCES clients(id) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

FOREIGN KEY и модификация значения первоначального столбца (3): SET NULL

Если в зависимом столбце разрешено значение **NULL**, то при удалении или обновлении значения первоначального столбца зависимому столбцу в качестве значения можно автоматически вставить **NULL**. Осуществляется при помощи конструкции **SET NULL**:

```
CREATE TABLE orders(
    client_id INT UNSIGNED NULL,
    reference VARCHAR(255) NOT NULL UNIQUE,
    price NUMERIC(20,2),
    FOREIGN KEY(client_id) REFERENCES clients(id) ON DELETE SET NULL ON UPDATE
SET NULL
);
```

Назначение имени ограничениям FOREIGN KEY

```
CREATE TABLE orders(
        client_id INT UNSIGNED NOT NULL,
        reference VARCHAR(255) NOT NULL UNIQUE,
        price NUMERIC(20,2),
        CONSTRAINT control_client_id FOREIGN KEY(client_id) REFERENCES clients(id)
);
```

Добавление ограничения внешнего ключа после создания таблицы

```
-- создание ограничения с явным назначением имени
ALTER TABLE orders ADD CONSTRAINT control_client_id  FOREIGN KEY(client_id)
REFERENCES clients(id);

-- создание ограничения, которому название даст сама MySQL
ALTER TABLE orders ADD FOREIGN KEY(client_id) REFERENCES clients(id);
```

Удаление ограничения внешнего ключа

```
-- 1-й способ
ALTER TABLE orders DROP FOREIGN KEY control_client_id;

-- 2-й способ
ALTER TABLE orders DROP CONSTRAINT control_client_id;
```

Ограничения первичного ключа PRIMARY KEY

Определение

“Каким удобным способом мы можем передать куда-то информацию о записи в таблице, но не целиком, а только частично, т.к. нам не нужная обработка этой записи в данный момент, но может потребоваться в дальнейшем?”. Для решения этой проблемы в таблицах принято использовать называемый первичный ключ (**PRIMARY KEY**) - столбец в таблице, который однозначно и **навсегда** (по крайней мере - теоретически) идентифицирует каждую запись. Важно знать, что первичный ключ может включать в себя несколько столбцов, но опять же - комбинация их значений должна однозначно идентифицировать каждую запись в таблице.

PRIMARY KEY похож на ограничение **UNIQUE** (первичный ключ также должен содержать уникальное в контексте таблицы значение), но в отличие от **UNIQUE** он, во-первых, никогда не может быть равен **NULL** и, во-вторых, в контексте таблицы может только один первичный ключ.

Пример создания ограничения PRIMARY KEY

В данной таблице столбец **id** должен получать только уникальные значение - в противном случае MySQL не будет вставлять данные и вернёт ошибку:

```
CREATE TABLE products(
    id INT UNSIGNED PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(20,2) NOT NULL,
    description TEXT NOT NULL
);
```

Назначение имени ограничениям PRIMARY KEY

```
CREATE TABLE products(
        id INT UNSIGNED,
        name VARCHAR(255) NOT NULL,
        price DECIMAL(20,2) NOT NULL,
        description TEXT NOT NULL,
        CONSTRAINT id_pk PRIMARY KEY(id)
);
```

PRIMARY KEY с использованием нескольких столбцов

В таблице есть два столбца (**version** и **type**), комбинация значений которых должна быть уникальна. Несмотря на то, что у этих двух столбцов не указана конструкция **NOT NULL**, **PRIMARY KEY** автоматически запрещает вставлять **NULL** в качестве значения любого из этих столбцов:

```
CREATE TABLE documents (
        version INT UNSIGNED,
        type VARCHAR(255),
        name VARCHAR(255),
        content TEXT NOT NULL,
        CONSTRAINT ver_type_pk PRIMARY KEY(version, type)
);
```

PRIMARY KEY и AUTO_INCREMENT (теория)

- В контексте первичного ключа (кстати, это же относится и к ограничению **UNIQUE**) есть одна возможность явно не указывать значение при вставке записи в таблицу. Если столбец с ограничением **PRIMARY KEY** или **UNIQUE** имеет тип целого числа (**TINYINT, INT, BIGINT** и т.д.), то ему можно присвоить атрибут **AUTO_INCREMENT**. Автоинкремент автоматически генерирует для своего столбца последовательное численное значение в том случае, когда в его таблицу вставляется запись, и в этой записи явно не предоставляется значение для столбца, содержащего автоинкремент. В рамках одной таблицы разрешен только один столбец с автоинкрементом.

- По умолчанию автоинкремент имеет изначальное значение 1 - таким образом, при первой вставке в столбце будет вставлено число 1, при второй вставке - 2, при третьей - 3 и т.д.

- Если в столбец с автоинкрементом будет вручную вставлено значение, которое превышает значение последнего автоматически созданного числа, то автоинкремент возьмет это значение, увеличенное на единицу, в качестве новой точки отсчета для генерации следующего значения.

PRIMARY KEY и AUTO_INCREMENT (пример создания)

```
CREATE TABLE products(
        id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
        name VARCHAR(255) NOT NULL,
        price DECIMAL(20,2) NOT NULL,
        description TEXT NOT NULL
);
```

Установка начального значения автоинкремента при создании таблицы

В данной таблице в столбце **id** автоматические значения будут стартовать со 100:

```
CREATE TABLE products(
    id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(20,2) NOT NULL,
    description TEXT NOT NULL
) AUTO_INCREMENT = 100;
```

Добавление ограничения первичного ключа после создания таблицы

```
-- создаем таблицу без первичного ключа
CREATE TABLE products(
    id INT UNSIGNED,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(20,2) NOT NULL,
    description TEXT NOT NULL
);

-- 1-й способ создания ограничения для столбца id
ALTER TABLE products ADD PRIMARY KEY(id);

-- 2-й способ создания ограничения для столбца id (с именем)
ALTER TABLE products ADD CONSTRAINT id_pk PRIMARY KEY(id);
```

Удаление ограничения первичного ключа

```
ALTER TABLE products DROP PRIMARY KEY;
```

Нормализация и нормальные формы

Аномалия вставки данных

Проблематично добавить преподавателя, курс которого еще неизвестен:

Name	Course	Room
John Smith	MySQL	21
Juan Lopez	Java	18
Ashley Johns	Python	15

Аномалия редактирования данных

При редактировании одной сущности необходимо произвести изменения в нескольких местах:

Name	Course	Room
John Smith	MySQL	21
Ashley ~~Johns~~ Monroe	Java	18
Ashley ~~Johns~~ Monroe	Python	15

Аномалия удаления данных

При удалении одной сущности удаляется информация о другой сущности:

Name	Course	Room
John Smith	MySQL	21
~~Ashley Monroe~~	~~Java~~	~~18~~
~~Ashley Monroe~~	~~Python~~	~~15~~

Избыточность данных

Во всех предыдущих случаем самым главным врагом нормальной работы таблицы выступала избыточность данных. **Избыточность данных** - явление, когда одни и те же данные хранятся в базе в нескольких местах. С этим можно справляться на небольших проектах (подставлять **NULL** и т.д.), однако в таблицах с миллионами строк это может привести к чудовищным последствиям.

Борьба с избыточностью данных

Нормализация - постепенное преобразование базы данных путем декомпозиции отношений (таблиц), то есть разбиения одной таблицы на несколько. Существует несколько (6 основных и 2 дополнительных) этапов нормализации, которые позволяют устранить избыточность. Эти этапы называются **нормальными формами**:

1. **Первая нормальная форма (1NF)**
2. **Вторая нормальная форма (2NF)**
3. **Третья нормальная форма (3NF)**
4. Нормальная форма Бойса-Кодда (BCNF)
5. Четвертая нормальная форма (4NF)
6. Пятая нормальная форма (5NF)
7. Доменно-ключевая нормальная форма (DKNF)
8. Шестая нормальная форма (6NF)

В рамках курса мы рассмотрим только первые три формы, т.к. формы, начиная с формы Бойса-Кодда и четвертой формы, уже перестают приносить пользу при реальной разработке.

Первая нормальная форма

Таблица находится в первой нормальной форме, если соблюдены два принципа:

1. если ни одна из строк таблицы не содержит в любом своем столбце более одного значения (т.н. атомарность)
2. в таблице не должно быть дублирующихся строк

Нарушение принципов первой нормальной формы:

Name	Position	PhoneNumber
John Smith	Programmer	8274827347, 234878754481, 647363456784
Nicole Adams	Designer	982748273478
Harry Thomas	DB Administrator	28274827347
Anna Devine	Principal	82748273479
Anna Devine	Principal	82748273479

Приведение к первой нормальной форме

Удаление дубликатов, столбец с телефоном содержит только атомарные значения:

Name	Position	PhoneNumber
John Smith	Programmer	8274827347
John Smith	Programmer	234878754481
John Smith	Programmer	647363456784
Nicole Adams	Designer	982748273478
Harry Thomas	DB Administrator	28274827347
Anna Devine	Principal	82748273479

Вторая нормальная форма

Таблица находится во второй нормальной форме, если соблюдены три принципа:

1. таблица уже находится в первой нормальной форме
2. таблица имеет ключ (простой или составной), уникально идентифицирующий каждую строку таблицы
3. все неключевые столбцы таблицы должны зависеть сразу от всех столбцов, входящих в ключ

Пример таблицы во второй нормальной форме (первичный ключ - столбец **id**):

ID	Name	State
1	Jānis Bērziņš	Student
2	Иван Иванов	Student
3	John Smith	Free Listener

Нарушение принципов второй нормальной формы

Первичный ключ таблицы **projects** - столбцы **project** и **curator**. Значение столбца **position** можно узнать только используя столбец **curator**, а не весь первичный ключ. Значение столбца **months** можно узнать с помощью столбца **project**.

project	**curator**	**position**	**months**
Site Development	Barbara Schmidt	System Analytic	2
Mobile Application Development	Michael Doe	Swift Programmer	1
CRM Development	Barbara Schmidt	System Analytic	24

Преобразование во вторую нормальную форму

Декомпозиция - разделение одной большой таблицы на меньшие таблицы.

Таблица **employees**

id	project	months
1	Site Development	2
2	Mobile Application Development	1
3	CRM Development	24

Таблица **projects**

id	name	position
1	Barbara Schmidt	System Analytic
2	Michael Doe	Swift Programmer

Таблица **projects_curators**

project_id	employee_id
1	1
2	2
3	1

Третья нормальная форма

Таблица находится в третьей нормальной форме, если соблюдены два принципа:

1. таблица уже находится во второй нормальной форме
2. все неключевые столбцы таблицы должны зависеть сразу от всех столбцов, входящих в ключ, но не должны зависеть от неключевых столбцов

Нарушение принципов третьей нормальной формы - столбец описания подразделения зависит от столбца подразделения, который не является первичным ключом:

id	**name**	**position**	**division**	**division_description**
1	Barbara Schmidt	System Analytic	Research and Development	Analytics and Innovations
2	Joe D'Amato	Android Programmer	Mobile Development	Develops Mobile Apps
3	John Smith	Web Programmer	Web Development	Develops Sites

Преобразование в третью нормальную форму

Таблица **divisions**

id	**division**	**division_description**
1	Research and Development	Analytics and Innovations
2	Mobile Development	Develops Mobile Apps
3	Web Development	Develops Sites

Таблица **employees**

id	**name**	**position**	**division_id**
1	Barbara Schmidt	System Analytic	1
2	Joe D’Amato	Android Programmer	2
3	John Smith	Web Programmer	3

Связи

Связь один к одному

В рамках связи один к одному какая-либо конкретная строка в одной таблице не может иметь более одной связанной строки в другой таблице. Например, человек может иметь только один паспорт (в большинстве случаев). Реализуется с помощью ограничения внешнего ключа (**FOREIGN KEY**) и ограничения по уникальности (**UNIQUE**):

Таблица **employees**

id	**name**	**position**	**division**
1	Barbara Schmidt	System Analytic	Research and Development
2	Joe D'Amato	Android Programmer	Mobile Development
3	John Smith	Web Programmer	Web Development

Таблица **passports**

employee_id	**code**	**nationality**	**age**
1	s8f8798sdf	German	35
2	98dfg87dg8	Italian	22
3	gd5f6df6gf	American	30

Связь один ко многим

В связях один ко многим одна строка в первой таблице может иметь много связанных строк во второй таблице. Но ни одна из этих совпадающих строк во второй таблице не должна иметь какую-либо другую связанную строку в первой таблице. Например, один клиент может совершить сколько угодно заказов, но каждый конкретный заказ может принадлежать только одному клиенту. Реализуется с помощью ограничения внешнего ключа (**FOREIGN KEY**):

Таблица **clients**

id	name	status
1	Alexander Kovalenko	Client
2	John Smith	Business Client

Таблица **orders**

id	client_id	reference	price
22	1	8df7d87fg8d7	99.99
23	2	df87gd8f7g88	55.55
24	1	d9f7g78fg7d8	22.22
25	2	98gh98fhg98f	11.11

Связь многие ко многим

В случае связи многие ко многим любая строка из первой таблицы может иметь сколь угодно много связанных строк во второй таблице, и наоборот - любая строка из второй таблицы может иметь сколь угодно много связанных строк в первой таблице. Как хороший пример можно упомянуть статьи и теги. Каждой статье можно поставить сколь угодно тегов, а любой тег может появиться у любой статьи. Для осуществления данной связи необходимо создать третью таблицу, где будут храниться комбинации идентификаторов статей и тегов в разных столбцах, которые будут иметь внешние ключи (**FOREIGN KEY**) на статьи и теги соответственно.

Таблица **articles**

id	**title**	**content**
1	Basic Principles Every Programmer Must Know	…
2	Queries in MySQL	…
3	DDoS attacks and other incidents	…

Таблица **tags**

id	**tag**
1	development
2	databases
3	artificial intelligence

Таблица **articles_tags**

article_id	**tag_id**
1	1
1	2
2	1
2	2