MySQL

ТАБЛИЦЫ: СОЗДАНИЕ И ТИПЫ ДАННЫХ

СОДЕРЖАНИЕ

СОЗДАНИЕ ТАБЛИЦ

- Как мы уже знаем, основной сущностью, которую использует MySQL, является таблица - в этом разделе мы познакомимся с основными аспектами их создания. Самым первым аспектом является тот факт, что любая таблица не принадлежит сама себе - она может быть создана только в контексте базы данных. Поэтому сначала создадим базу данных с названием **shop** (кодировку укажем явно, чтобы на зависеть от глобальных настроек):

```
CREATE DATABASE shop DEFAULT CHARSET utf8mb4;
```

- После того, как база данных была создана (если она уже не была создана ранее), необходимо явно указать, что она будет использоватся - это делается при помощи следующем команды:

```
USE shop;
```

- Далее уже можно создавать таблицы, однако следует помнить, что таблица не может быть пустой - в ней должен быть по крайней мере один столбец. Пока мы еще не проходили типы столбцов в таблицах, но на данный момент можем условиться, что в качестве примера будет использован тип **TEXT** (не сложно догадаться, что он предназначен для хранения текстовых данных). Далее пример запроса на создания таблицы **products**:

```
CREATE TABLE products (
    name TEXT,
    code TEXT
);
```

- Предыдущий запрос создал таблицу с двумя столбцами - name и code. Приведем пример возможного содержания этой таблицы:

name	code
iPhone 14 Plus	S8DF7S87DF87S8
GeForce RTX 4080	DF987GD8F7G87F
Intel Core i9-10900K	F3K4J5L3J4KLJE

- Чтобы посмотреть все таблицы, уже созданные в контексте выбранной базы данных, следует выполнить представленную ниже команду - на данный момент в качестве ответа мы должны получить таблицу из одной строки - таблицы **products**:

```
SHOW tables;
```

- Иногда при разработке какого либо приложения у нас нет возможности (или времени) каждый раз проверять, была ли таблица уже создана или нет - в этом случае можно попробовать создать таблицу без риска получить ошибку. Для этого применяется конструкция **IF NOT EXISTS** - если таблица с таким именем уже существует, СУБД просто проигнорирует запрос на создание такой же таблицы:

```
CREATE TABLE IF NOT EXISTS products (
    name TEXT,
    code TEXT
);
```

- Однако бывают случаи, когда нам необходимо создать таблицу, которая полностью повторяет структуру уже существующей таблицы (хотя и с другим именем) - например, нам нужен архив всех продуктов, которые продавались у нас в магазине, но могли быть удалены или обновлены. Можно предположить, что самый простой способ - вручную создать таблицу с идентичными столбцами, одна есть еще более простой вариант - использовать оператор **LIKE**, представленный в коде снизу:

```
CREATE TABLE products_archive LIKE products;
```

- Во всех предыдущих случаях создавались постоянные таблицы, которые будут существовать при всех последующий подключениях к базе данных и даже в том случае, если компьютер, на котором установлена MySQL, будет отключен от питания. Но иногда существует потребность создать временную таблицу, которая будет существовать только в то время, когда пользователь будет подключен к СУБД (это время также называют сессией), а затем автоматически удалится. Это бывает необходимо для хранения каких-либо сложных вычислений, который происходят только во время сессии, и не имеют смысла в будущем. Для создания таких временных таблиц

используется оператор **TEMPORARY**, а весь остальной синтаксис создания таблицы остается неизменным:

```
CREATE TEMPORARY TABLE calculations (
    operation_name TEXT,
    operation_result TEXT
);
```

- Временные таблицы не видны при просмотре общего списка таблиц, т.е. мы не увидим строки с таблицей **calculations** после выполнения запроса **SHOW TABLES**.

- В данном случае следует отметить еще важный момент - таблицы, созданные при помощи оператора **TEMPORARY** не обязательно хранятся во временной памяти компьютера. Они хранятся в том виде, который определяется движком таблицы. Движок по умолчанию прописан в файле **my.ini**, а конкренто в секции **[mysqld]** - за это отвечает настройка **default-storage-engine**. Если администратор не трогал базовые конфигурации, то движком по умолчанию является **InnoDB** - это означает, что все таблицы (как временные, так и постоянные) будут хранится на диске компьютера.

- Если при создании таблицы нам необходимо использовать какой-либо другой движок, то это определяется при помощи ключевого слова **ENGINE** (далее мы применим движок MyISAM):

```
CREATE TABLE clients (
    firstname TEXT,
    lastname TEXT
) ENGINE = MyISAM;
```

- Проясним один момент - даже если при создании таблицы был применен движок, который определяет хранение таблицы в памяти (например, **MEMORY**), то это не означает, что таблица будет автоматически удалена после окончания сессии. Это происходит только тогда, когда при создании таблицы был применен оператор **TEMPORARY**. Одновременно стоит помнить, что при перезагрузке СУБД или при выключении питания компьютера, все данные (но не структура), хранящиеся в таблицах типа **MEMORY**, пропадут безвозвратно.

- После рассмотрения особенности определения движков таблицы нельзя не упомянуть про кодировки таблиц. В разделе о базах данных уже упоминалось, что сами базы создаются в кодировке по умолчанию (которая прописана в конфигурационном **файле my.ini** при помощи настройки **character-set-server**, если же кодировка не прописана явно, то всегда в utf8mb4) или же в той кодировке, которая явно прописана при создании базы. В таблицах механизм несколько иной - таблицы создаются в кодировке той базы данных, которой таблица принадлежит, либо в той кодировке, которая явно указана при создании таблицы. Первый случай понятен - его подробно рассматривать не будем, но второй случай - явное указание кодировки приведем в следующем примере (как и ранее с базами данных, это осуществляется при помощи конструкции **DEFAULT CHARSET**):

```
CREATE TABLE IF NOT EXISTS currencies (
    title TEXT,
    sign TEXT
) DEFAULT CHARSET = latin1;
```

- При разъяснении всех предыдущих особенностей создания баз может возникнуть закономерный вопрос - а возможно ли комбинировать упомянутые выше конструкции (**IF NOT EXISTS**, **TEMPORARY**, **ENGINE**, **LIKE**, **DEFAULT CHARSET**) одновременно? Ответ - да, возможно, если это не противоречит здравому смыслу (например, нельзя использовать **LIKE** и **DEFAULT CHARSET** одновременно, т.к. **LIKE** уже определяет структуру таблицы и кодировки). Ниже приведены два примера таких случаев:

```
CREATE TEMPORARY TABLE IF NOT EXISTS languages (
    name TEXT,
    iso TEXT
) ENGINE = InnoDB DEFAULT CHARSET = latin1;
```

```
CREATE TEMPORARY TABLE IF NOT EXISTS visitors LIKE clients;
```

- Часто бывает, что структура таблицы должна быть изменена. Мы не будем сейчас рассматривать модификацию столбцов таблицы и их типов - это тема для дальнейших разделов, однако обратим внимание на то, как поменять движок и кодировку таблицы. Движок таблицы **currencies** при создании был определен как **MyISAM**, мы же хотим поменять его на **InnoDB**. Это может быть осуществлено при помощи следующей команды:

```
ALTER TABLE clients ENGINE = InnoDB;
```

- Кодировка меняется при помощи схожей команды:

```
ALTER TABLE languages DEFAULT CHARSET = utf8mb4;
```

- Важное примечание по поводу кодировки - при смене кодировки всей таблицы кодировки существующих текстовых столбцов изменены не будут. Измененная кодировка окажет влияние только на те столбцы, которые будут добавляться в будущем. Однако существует возможность поменять кодировку каждого столбца таблицы в отдельности (это возможно только в том случае, если новая кодировка позволяет хранить уже записанные в таблицу данные) - это мы сделаем, когда будем рассматривать символьные типы данных.

- Если таблица нам не нужна, и мы не хотим, чтобы она занимала место на диске (или в памяти) компьютера, то мы можем ее удалить. Это осуществляется командой **DROP TABLE**, например:

```
DROP TABLE languages;
```

- В том случае, если мы не уверены, что на момент удаления таблица существует, мы можем обезопасить себя от ошибки при помощи конструкции **IF EXISTS**:

```
DROP TABLE IF EXISTS languages;
```

- Часто при создании таблиц существует соблазн назвать столбец (или даже всю таблицу) уже зарезервированным в MySQL словом, например, **TABLE**. Очевидно, что при таком поведении мы немедленно столкнемся с ошибками и наше приложение не будет работать:

```
-- ERROR 1064 (42000): You have an error in your SQL ...
CREATE TABLE table (
    content TEXT
);
```

  Однако, если такое наименование является вопросом жизни и смерти, то MySQL может позволить это сделать, но с одним

условием - наименование должно быть обрамлено в обратные кавычки - `table`. Часто для безопасности обратные кавычки используются для именования всех сущностей в базе данных, но мы ограничимся только наименованием таблицы:

```
-- в данном случае ошибки не будет
CREATE TABLE `table` (
    content TEXT
);
```

- Перед переходом к конкретным типам столбцов таблицы стоит сказать несколько слов о том, как эти столбцы должны быть определены при создании таблицы. Как мы уже видели из предыдущих примеров, сначала необходимо указать название столбца, а затем его тип - это минимальный набор. Например, создадим таблицу **articles** с одним столбцом - **content**. Тогда при определении этого столбца сначала следует указать слово **content** (имя столбца), а затем **TEXT** (тип столбца):

```
CREATE TABLE articles (
    content TEXT
);
```

- Важно заметить, что в таблице не может быть столбцов с двумя одинаковыми названиями - если попробовать создать такую таблицу, то MySQL отменит создание и вернет ошибку (это случиться даже в том случае, если типы столбцов были бы разными):

```
-- ERROR 1060 (42S21): Duplicate column name 'content'
CREATE TABLE articles (
    content TEXT,
    content TEXT
);
```

- Кроме названия и типа, при создании столбца существует возможность указать множество дополнительных атрибутов, которые будут влиять на поведение данных в этом столбце. Атрибуты перечисляются после типа через пробел. В качестве примера установим атрибут кодировки столбца, а также укажем, что этот столбец всегда должен быть заполнен (по умолчанию при вставке данных разрешается не заполнять все столбцы - в таком случае незаполненные столбцы получают специальное

значение **NULL** - мы еще обсудим его подробнее в следующих разделах):

```
CREATE TABLE tags (
    title TEXT CHARACTER SET latin1 NOT NULL
);
```

- Напоследок рассмотрим два случая - получение информации о структуре таблицы (какие столбцы типы данных она содержит), кроме того получение полного SQL запроса на создание таблицы, которая идентична уже существующей таблице (полезно, если мы хотим создать такую-же таблицу в каком-то другом месте)

- Получение данных о структуре таблицы **products**:

```
DESCRIBE products;
```

В качестве ответа получим таблицу, схожую со следующим примером:

Field	Type	Null	Key	Default	Extra
name	text	YES		NULL	
code	text	YES		NULL	

- Получение запроса на создание уже существующей таблицы:

```
SHOW CREATE TABLE products;
```

В качестве ответа получим следующее:

Table	Create Table

products	**CREATE TABLE `products` (
	  `name` text,
	  `code` text
	) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci**

СИМВОЛЬНЫЕ ТИПЫ ДАННЫХ

- Символьные или текстовые типы данных предназначены для хранения текстовой информации, но в отдельных случаях они также используются для хранения содержимого файлов.

- Как уже упоминалось, при хранении текстов важно учитывать их кодировку. В общем случае (но не в случае бинарных типов) при сохранении данных в таблицу MySQL считывает ту кодировку, в которой клиент прислал данные (ее можно посмотреть при помощи переменной **@@character_set_client**), потом пытается преобразовывает ее в кодировку соединения (**@@character_set_connection**), и в конце концов преобразовывает данные в ту кодировку, которая требуется столбцу таблицы и производит сохранение (или сравнение, если это не вставка, а выборка с условием). Если пришедшие символы не разрешены в контексте кодировки таблицы столбца, MySQL не даст сохранить данные и вернёт ошибку.

- Перед отправкой данных клиенту MySQL преобразует их в кодировку, которая установлена при помощи переменной **@@character_set_results**.

- Каждая из этих переменных либо устанавливается явно в момент соединения с базой данных (как мы помним, стандартный клиент **mysql** делал это при помощи опции **default-character-set**), либо эти переменные берутся из настроек файла **my.ini**. Однако даже после установки соединения эти переменные можно установить вручную двумя способами:

```
/*
    Команда устанавливает одинаковое значение utf8mb4 сразу
    для трех переменных - @@character_set_client,
    @@character_set_connection и @@character_set_results
*/

SET NAMES utf8mb4;
```

```
-- установка вручную каждой переменной

SET @@character_set_client = utf8mb4;
SET @@character_set_connection = utf8mb4;
SET @@character_set_results = utf8mb4;
```

- Самые простые типы данных для хранения текстовой информации - тип **VARCHAR** и тип **CHAR**. **VARCHAR** может хранить в себе данные с максимальным объемом 65535 байт, а **CHAR** вмещает в себя максимум 255 символов. При создании таблицы для обоих типов следует указывать максимальное количество символов, которое мы разрешаем для конкретного столбца (очевидно, что этот максимум не должен превышать стандартные ограничения этих типов). Далее приведем пример создания таблицы, в которой используется **VARCHAR** и **CHAR**.

```
CREATE TABLE posts (
    title CHAR(30),
    body VARCHAR(600)
);
```

- В контексте этих двух типов может возникнуть вопрос - если нам необходимо хранить текст до 255 символов, какой тип нам надо выбрать, какое между ними различие? Различие есть, и оно существенное. Дело в том, что **CHAR** при хранении всегда занимает на диске (или в памяти) компьютера тот максимальный размер символов, который для него был установлен при создании таблицы. **VARCHAR** пытается приспособиться и хранит ровно столько, сколько занимают сами данные. Далее представлена наглядная таблица для лучшего понимания:[1]

Значение	CHAR(4)	Размер	VARCHAR(4)	Размер
''	' '	4 байта	''	1 байт
'ab'	'ab '	4 байта	'ab'	3 байта
'abcd'	'abcd'	4 байта	'abcd'	5 байт

- Из приведенной таблицы можно понять, что **VARCHAR** более экономно расходует память, однако **CHAR** позволяет (иногда) чуть быстрее вставлять и получать данные из таблицы.

- У **VARCHAR** и **CHAR** существуют родственные типы данных **VARBINARY** и **BINARY**, которые ведут себя схожим образом (в смысле особенностей хранения и максимального размера), но имеют некоторые отличия, связанные с кодировкой. Дело в том, что эти два типа данных предназначены для хранения не

[1] https://dev.mysql.com/doc/refman/8.0/en/integer-types.html

текстовых, а бинарных данных (обычно такие данные используются для хранения содержимого файлов).

- Это значит, что MySQL вообще не учитывает стандартные кодировки при работе с колонками данного типа - не происходит никаких преобразований при сохранении и отдаче данных - все происходит в “сыром виде”. Если быть совсем точным, то в данном случае используется кодировка **binary**.

- Еще раз повторим про ограничения, но уже в более явном виде - максимальный размер **VARBINARY** - 65535 байт, а **BINARY** - 255 байт.

- Далее приведен пример создания таблицы, в которой используются **VARBINARY** и **BINARY**:

```
CREATE TABLE files (
    small_image BINARY(255),
    large_image VARBINARY(65535)
);
```

- Практически все последующие типы данных отличаются от **VARCHAR** и **VARBINARY** только размером (еще одним отличием является отсутствие необходимости указывать максимальный размер данных в колонке), поэтому покажем их особенности в виде удобной таблицы:

Тип данных	Бинарный Тип	Максимальный размер	Пример создания таблицы
TINYBLOB	+	255 байт	**CREATE TABLE files (image TINYBLOB);**
TINYTEXT		255 байт	**CREATE TABLE posts(content TINYTEXT);**
BLOB	+	65535 байт	**CREATE TABLE files (image BLOB);**
TEXT		65535 байт	**CREATE TABLE posts(content TEXT);**

MEDIUMBLOB	+	16777215 байт	**CREATE TABLE files (image MEDIUMBLOB);**
MEDIUMTEXT		16777215 байт	**CREATE TABLE posts(content MEDIUMTEXT);**
LONGBLOB	+	до 4 Гб	**CREATE TABLE files (image LONGBLOB);**
LONGTEXT		до 4 Гб	**CREATE TABLE posts(content LONGTEXT);**

- Кроме конкретно текстовых (или бинарных) данных, MySQL позволяет хранить так называемые перечисления - **ENUM.** Перечисление это столбец, который может принимать значение из списка допустимых значений, явно перечисленных в спецификации столбца в момент создания таблицы. Пример создания перечисления представлен ниже:

```
CREATE TABLE graphics (
    y_month ENUM('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct', 'Nov','Dec')
);
```

- Максимальное количество вариантов для каждого конкретного перечисления - 65535, при хранении занимает не больше 2 байт.

- В подразделе о создании таблиц уже упоминалось, что по умолчанию при вставке данных любой столбец можно оставлять пустым - в этом случае столбец автоматически получит значение **NULL**. Такая же логика присуща и перечислениями. Однако в том случае, если столбец явно требует присвоения значения (т.е. указан атрибут **NOT NULL**), то перечисление начинает вести себя очень необычно. При попытке не указывать данные при вставке обычное поле вернет ошибку, а **ENUM** успешно вставит некое значение - этим значением будет первый вариант из списка разрешенных вариантов. Например, предположим, что нам доступна следующая таблица:

```
CREATE TABLE humans (
```

```
    name VARCHAR(255),
    sex ENUM('female', 'male') NOT NULL
);
```

Если мы вставим строку, в которой будет указано только имя (name), то в поле пол (sex) автоматически будет установлено значение female.

- В качестве последнего типа данных в этом разделе рассмотрим множества - **SET**. Множества имеют схожие черты с перечислениями, но они устроены несколько сложнее - в данном случае при создании также необходимо прописать список разрешенных значений, однако при вставке нам разрешено вставлять не одно, а несколько значений, разделенных запятой. Далее пример таблицы с типом данных **SET**:

```
CREATE TABLE employees (
    name VARCHAR(255),
    position SET('programmer', 'administrator', 'manager')
);
```

- В приведенной выше таблице в столбце position может хранится любое из следющих значений:

```
'programmer', 'administrator', 'manager',
'programmer,administrator', 'programmer,manager',
'administrator,manager', 'programmer,administrator,manager'
```

- Максимально количество отдельных значений для множества - 64, при хранении столбец занимает не больше 8 байт.

- В отличие от перечисления, множество ведет себя стандартно при отсутствии значения этого столбца при вставке - если разрешен **NULL**, то будет вставлен **NULL**, если же значение обязательно, то при его отсутствии MySQL выдаст ошибку.

- Далее рассмотрим основные атрибуты, используемые в контексте символьных полей. Самым очевидным атрибутом является установка кодировки - осуществляется при помощи конструкции **CHARACTER SET**:

```
CREATE TABLE tags(
```

```
    tag_name VARCHAR(255) CHARACTER SET utf8mb4
);
```

- Далее рассмотрим уже упоминавшееся специальное значение **NULL** - оно подразумевается по умолчанию, однако можно явно указать, что оно будет вставляться при отсутствии значения - это делается при помощи ключевого слова **NULL**, если же значение обязательно, то применяется конструкция **NOT NULL**:

```
CREATE TABLE articles (
    -- логика по умолчанию - разрешено отсутствие значения
    title VARCHAR(255),

    -- явно разрешено отсутствие значения
    content TEXT NULL,

    -- отсутствие значения запрещено
    category VARCHAR(255) NOT NULL
);
```

- Важно знать, что в случае отсутствия значения в столбец таблицы может вставляться не только **NULL**, но любое подходящее по типу значение - это осуществляется при помощи ключевого слова **DEFAULT**:

```
CREATE TABLE subscribers (
    name varchar(255),

    /*
        При отсутствии значения в столбец будет вставляться
        строка 'customer'
    */
    role varchar(255) DEFAULT 'customer'
);
```

- Типы **TINYBLOB**, **TINYTEXT**, **BLOB**, **TEXT**, **MEDIUMBLOB**, **MEDIUMTEXT**, **LONGBLOB**, **LONGTEXT** могут иметь только одно значение по умолчанию - **NULL**, остальные запрещены.

- Напоследок обратим внимание на модификацию и удаление символьных столбцов. Удаление происходит стандартно для всех столбцов всех типов при помощи конструкции **ALTER TABLE** … **DROP COLUMN** …:

```
ALTER TABLE subscribers DROP COLUMN role;
```

- Если мы хотим модифицировать столбец, то используется команда **ALTER TABLE** … **MODIFY COLUMN** … (важно помнить, что модификация может завершиться с ошибкой, если, например, мы пытаемся уменьшить максимальный размер текста, но в столбце уже хранится текст, размер которого превышает новый максимум; кроме того, ошибка может возникнуть, если предпринята попытка назначить несовместимую с уже существующими данными новую кодировку):

```
ALTER TABLE subscribers MODIFY COLUMN name TEXT CHARACTER SET utf8mb4 NOT NULL;
```

ЦЕЛЫЕ ЧИСЛА

- Как можно понять из названия, данный тип данных предназначен для хранения целых чисел без дробной части. Перед тем, как приступить к подробному описанию, рассмотрим ситуацию, при которой в целочисленный столбец пытаются вставить некоторый другой тип данных.

- При попытке заполнить целочисленный тип данных дробным числом (например 1.6, 99.99, 115.13) MySQL не будет возвращать ошибку, а успешно произведет сохранение. Однако в этом случае произойдет преобразование дробного типа числа в целый тип, притом СУБД не просто отбросит дробную часть у значения, а совершит округление согласно законам математики. Таким образом 1.6 станет 2, 999.99 станет 100, а 115.13 станет 115.

- Схожим образом MySQL ведет себя в том случае, если в целочисленный столбец попытаться вставить текстовые данные. Такие значения как '99', '33', '92349' будут вставлены как простые целые числа 99, 33, 92349. В свою очередь текстовые значения, идентичные дробным цифрам, такие как '45.45', '11144.734234' и '456.79' вначале будут представлены в виде дробных чисел, а потом будут автоматически округлены до целых чисел согласно законам математики. В результате мы получим сохраненные 45, 11145 и 457 числа соответственно. Однако важно знать, что если текстовое значение не может быть представлено в виде цифры (целой или дробной), то MySQL не будет производить никаких преобразований, вернет ошибку и данные не будут сохранены.

- Теперь, наконец, приведем пример создания таблицы с одним из целочисленных типов (**INT**):

```
CREATE TABLE numbers (
    number INT
);
```

- Интересно отметить тот факт, что MySQL принимает не только числа в десятичном формате (т.е. по основанию 10). Также в целочисленные столбцы можно вставлять шестнадцатеричные цифры (например, 0x3E8) и двоичные цифры (например, 0b111).

Однако при выборке мы все равно будет получать обычные цифры по основанию 10 - 0x3E8 вернется как 1000, а 0b111 - как 7.

- Целочисленные типы (как и текстовые) по умолчанию могут принимать тип **NULL** (если столбец не имеет атрибута **NOT NULL**), а также значение по умолчанию (если столбец имеет атрибут **DEFAULT** и собственно само значение):

```
CREATE TABLE quantities (
    name VARCHAR(255),
    quantity INT DEFAULT 0
);
```

- Как мы знаем из школьного курса, во множество целых чисел также входят отрицательные числа, а также 0. MySQL полностью подчиняется данным правилам, однако есть одна маленькая особенность, которую обязательно следует знать. Если при создании целочисленного столбца после наименования типа целого числа (конкретные типы мы рассмотрим позднее) прописан атрибут **UNSIGNED**, то данные столбец может хранить только ноль и числа больше нуля. MySQL не будет разрешать сохранять в этом столбце отрицательные числа и при таких попытках будет всегда возвращать ошибку. Далее приведем пример атрибута **UNSIGNED** для типа **INT**:

```
CREATE TABLE classes (
    name VARCHAR(255),
    number_of_participants INT UNSIGNED
);
```

- Атрибут **UNSIGNED** не просто ограничивает разрешенный диапазон хранимых значений, он просто сдвигает его вперед (т.е. он попросту увеличивает в два раза разрешенный диапазон положительных чисел). Например если целочисленный тип **TINYINT** без всяких атрибутов может хранить целые числа начиная с -128 и заканчивая 127, то после применения атрибута **UNSIGNED** для данного типа данных можно хранить числа начиная с 0 и заканчивая 255.

- В MySQL позволяет неявно ограничивать размер количества цифр хранимого числа - при создании столбца можно написать, например, **INT(2)** или **INT(10)**. Однако это не означает, что в этом случае СУБД будет ограничивать диапазон хранимых

значений или как-то препятствовать вставке больших чисел - более того, использование такой логики считается устаревшим подходом.[2]

- Абсолютно все целочисленные типы имеют одинаковую структуру - они отличаются только диапазоном хранимых значений и именем. Поэтому представим их в виде удобной таблицы:

Тип	Размер байт	Мин. значение	Макс. значение	Пример создания таблицы
TINYINT	1	-128	127	**CREATE TABLE prices (price TINYINT);**
TINYINT UNSIGNED	1	0	255	**CREATE TABLE prices (price TINYINT UNSIGNED);**
SMALLINT	2	-32768	32767	**CREATE TABLE prices (price SMALLINT);**
SMALLINT UNSIGNED	2	0	65535	**CREATE TABLE prices (price SMALLINT UNSIGNED);**
MEDIUMINT	3	-8388608	8388607	**CREATE TABLE prices (price MEDIUMINT);**
MEDIUMINT UNSIGNED	3	0	16777215	**CREATE TABLE prices (price MEDIUMINT UNSIGNED);**
INT	4	-2147483648	2147483647	**CREATE TABLE prices (price INT);**

[2] https://dev.mysql.com/doc/refman/8.0/en/numeric-type-attributes.html

INT UNSIGNED	4	0	4294967295	**CREATE TABLE prices (price INT UNSIGNED);**
BIGINT	8	-2^{63}	$2^{63} - 1$	**CREATE TABLE prices (price BIGINT);**
BIGINT UNSIGNED	8	0	$2^{64} - 1$	**CREATE TABLE prices (price BIGINT UNSIGNED);**

- Кроме вышеперечисленных типов, существует еще тип **BOOLEAN**. Как видно из названия, теоретически он должен хранить логические сущности **TRUE** и **FALSE**, однако на самом деле это не так. В данном случае MySQL пытается экономить типы и при создании столбца подменяет этот тип на **TINYINT(1)** (хотя сама СУБД считает этот синтаксис устаревшим). Подразумевается, что клиент должен хранить в нем числа 1 (**TRUE**) или 0 (**FALSE**), однако никто не мешает записывать туда числа в стандартном диапазоне **TINYINT** от -128 до 127:

```
CREATE TABLE subscriptions (
     code INT,
     valid BOOLEAN
);
```

- Напоследок по традиции обратим внимание на удаление целочисленных столбцов. Удаление происходит стандартно - при помощи конструкции **ALTER TABLE … DROP COLUMN …**:

```
ALTER TABLE numbers DROP COLUMN number;
```

- Для модификации снова используется стандартная конструкция **ALTER TABLE … MODIFY COLUMN …**:

```
ALTER TABLE numbers MODIFY COLUMN number BIGINT DEFAULT 1;
```

ДРОБНЫЕ ЧИСЛА

- Закономерно, что кроме целых чисел, MySQL позволяет хранить и дробные числа. Всего есть два основных типа дробных чисел - числа с фиксированной точкой (имеют большую точность) и числа с плавающей точкой (имеют приблизительную точность).

- В старых версиях MySQL для дробных чисел был разрешен атрибут **UNSIGNED**, однако он работал несколько странно - он действительно запрещал вставку отрицательных чисел, однако одновременно он никак не увеличивал диапазон положительных. В современных версиях MySQL этот атрибут объявлен устаревшим для дробных чисел и его лучше не использовать.

- Тип с фиксированной точкой называется **DECIMAL** (однако у него есть псевдоним **NUMERIC** - если встречаем в СУБД два этих названия, то помним, что это одно и то же). Если объявить такой столбец следующим образом, то внезапно мы увидим, что **DECIMAL** ведет себя как целочисленный тип (т.е. будет округлять числа до целых значений):

```
CREATE TABLE salaries (
     salary DECIMAL
);
```

- Чтобы **DECIMAL** стал хранить и дробные числа, то ему надо обязательно указать разрешенное максимальное общее количество хранимых цифр в одном числе, а также количество цифр числа, которые отведены под дробную часть числа (притом разделяющая точка и минус в эти цифры не входят) - **DECIMAL(максимальное количество цифр, количество цифр для дробной части)**:

```
-- в данном случае разрешенный диапазон от -99.99 до 99.99
CREATE TABLE salaries (
     salary DECIMAL(4,2)
);
```

- Если в поле типа **DECIMAL** (например, **DECIMAL(5,2)**) попытаться вставить число, целая часть которого не входит в диапазон разрешенных значений (для нашего примера это максимально 3 цифры, т.к. 5 - 2 = 3), например, 9999, то MySQL запретит делать это и вернет ошибку. Если же попытаться вставить

дробное число, количество цифр дробной части которого также превышает допустимое значение, то MySQL попытается его округлить - например, при попытке вставить 9.899 сохранено будет 9.90. Однако если после округления целая часть достигнет запрещенных значений (например так будет при попытке вставить 999.999 - после округление это равняется 1000) - MySQL не будет ничего сохранять и вернет ошибку.

- Максимально количество цифр в типе **DECIMAL** равняется 65, а количество дробной части не должно превышать 30 цифр. Таким образом следующее определение столбца поддерживается - **DECIMAL(65, 30)**, а вот такие - **DECIMAL(66, 30)**, **DECIMAL(65, 31)** и **DECIMAL(66, 31)** - не разрешены для использования в MySQL.

- Далее рассмотрим дробный тип с плавающей запятой. Представителей такого типа всего два - это **FLOAT** и **DOUBLE**. Самое важное, что стоит помнить о них - в отличие от **DECIMAL** они не хранят точные данные. Их основная задача - использование в таких научных или инженерных задачах, где допускается некоторая приблизительность. В данном случае **FLOAT** и **DOUBLE** идеального подходят для того, что отобразить эту неопределенность. Если мы хотим хранить данные о финансовых операциях, то мы должны использовать только **DECIMAL**. Единственное преимущество **FLOAT** и **DOUBLE** перед **DECIMAL** - это более оптимальное их хранение, что несколько ускоряет операции с такими неточными типами данных.

- В предыдущих версиях MySQL **FLOAT** и **DOUBLE** поддерживали синтаксис, схожий с **DECIMAL - FLOAT (максимальное количество цифр, количество цифр для дробной части)**, например **FLOAT(10,2)** и **DOUBLE (максимальное количество цифр, количество цифр для дробной части)**, например **DOUBLE(20,4)**. Сейчас такой синтаксис объявлен устаревшим и использовать его не стоит.[3] В общем случае надо использовать только **FLOAT** и только **DOUBLE** без всяких дополнений:

```
CREATE TABLE scientific_calculations (
     simple_calculations_result FLOAT,
     difficult_calculations_result DOUBLE
);
```

[3] https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html

- Тип данных **FLOAT** поддерживает дробные числа, которые помещаются в 4 байта - учитывая особенности хранения таких чисел, можно сказать, что предоставляется возможность использовать значения начиная от -3.402823466E+38 и до 3.402823466E+38. Дробная часть может быть любой (например, 99.99, -112.123989823 и т.д.), но не гарантируется, что эта точность будет соблюдена при сохранении. Также тип данных **FLOAT** поддерживает следующий синтаксис создания:

```
CREATE TABLE calculations_values (
     value FLOAT(1)
);
```

  Данный синтаксис (число в скобках после **FLOAT**) позволяет устанавливать значение хранимого числа. Практика показывает, что это практически ни на что не влияет, просто при числах примерно начиная с 25 (**FLOAT(25)**) MySQL по сути автоматически преобразует данный тип данных с более крупными числами - **DOUBLE**.

- **DOUBLE** поддерживает дробные числа, которые помещаются в 8 байт - можно хранить числа от -1.7976931348623157E+308 и до 1.7976931348623157E+308. Каждое такое число будет занимать 8 байт. Также **DOUBLE** не поддерживает никаких указаний размера или дробной части (это или объявлено устаревшей логикой или вообще не поддерживается):

```
CREATE TABLE big_calculations_values (
     value DOUBLE
);
```

- Удаляются и модифицируются дробные типы стандартно:

```
-- удаление
ALTER TABLE scientific_calculations DROP COLUMN
simple_calculations_result;
```

```
-- модификация
ALTER TABLE scientific_calculations MODIFY COLUMN
difficult_calculations_result NUMERIC(20,3);
```

ТИПЫ ДАННЫХ ДАТЫ И ВРЕМЕНИ

- Абсолютно все типы даты и времени являются по своей сути строками в строго заданном формате, поэтому удобнее всего их представить в качестве таблицы:

Тип	Диапазон	Размер байт	Пример создания таблицы
DATE	'1000-01-01' - '9999-12-31'	3	**CREATE TABLE clock (value DATE);**
DATETIME	'1000-01-01 00:00:00' - '9999-12-31 23:59:59'	8	**CREATE TABLE clock (value DATETIME);**
TIME	'-838:59:59' - '838:59:59'	3	**CREATE TABLE clock (value TIME);**
TIMESTAMP	'1970-01-01 00:00:01' - '2038-01-19 03:14:07'	4	**CREATE TABLE clock (value TIMESTAMP);**
YEAR	1900 - 2155 или 1970 - 2069[4]	1	**CREATE TABLE clock (value YEAR);**

- Особый интерес вызывает тип **TIMESTAMP**. Данный тип не просто хранит присланное ему значение. Он трактует это значение как дату во временной зоне MySQL сервера, потом конвертирует эту дату во временную зону UTC и только потом сохраняет. При выборке данных происходит обратный процесс - MySQL преобразует хранимое значение во временную зону сервера, а только потом возвращает данные клиенту. Можно понять, что если при записи и выборки сервер имел разные настройки временной зоны, то данные могут быть искажены.

- Для разных клиентских приложений существуют разные способы настройки временной зоны в MySQL (по умолчанию, если в файле **my.ini** есть настройка **default-time-zone**, то используется

[4] Если попытаться записать в поле типа **YEAR** двузначное или однозначное число, то оно будет интерпретировано как число начиная с 1970 по 2069, например, 99 - 1999, 5 - 2005 и т.д.

она, в противном случае берется системная временная зона), но самый простой способ явной установки временного пояса приведен в следующем примере:

```
SET @@SESSION.time_zone = '+03:00'; -- для сессии
SET @@GLOBAL.time_zone = '+03:00'; -- для всего сервера
```

- Важной особенностью двух типов - **DATETIME** и **TIMESTAMP** - является возможность задать им в качестве значения по умолчанию не конкретное значение, а время того момента, когда в таблицу будет вставляться их строка - далее пример того, как это можно реализовать (в данном случае при вставке значения example_name, dt_value и ts_value получат время, соответствующие моменту вставки):

```
CREATE TABLE default_time_examples (
     example_name VARCHAR(255),
     dt_value DATETIME DEFAULT CURRENT_TIMESTAMP,
     ts_value TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- Более того, **DATETIME** и **TIMESTAMP** могут не только могут автоматически добавляться при вставке их ряда. Также они могут обновлять свое значение при обновлении какого либо другого столбца их ряда:

```
CREATE TABLE default_time_dynamic_examples (
     example_name VARCHAR(255),
     dt_value DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
     ts_value TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

- Удаляются и модифицируются временные типы стандартно:

```
-- удаление
ALTER TABLE default_time_examples DROP COLUMN dt_value;
```

```
-- модификация
ALTER TABLE default_time_examples MODIFY COLUMN ts_value
TIMESTAMP DEFAULT NULL;
```

JSON

- Начиная с версии 5.7.8 MySQL поддерживает тип данных **JSON** (JavaScript Object Notation) - в столбцах этого типа можно хранить текстовые данные, но при вставке они всегда будут проверены на соответствие формату JSON. Если формат данных не пройдет проверку, MySQL не даст сохранить предоставленное значение. Пример создания поля с типом **JSON:**

```
CREATE TABLE staff (
     name VARCHAR(255),
     roles JSON
);
```

- **JSON** может занимать такой же максимальный размер, как и **LONGTEXT**, т.е. до 4 гигабайт. Однако в процессе обработки таких данных MySQL может использовать большее количество памяти - это может привести к некоторым проблемам, если на сервере не хватает памяти.

- **JSON** может иметь только одно значение по умолчанию - **NULL**, остальные запрещены.

- Удаляется и модифицируется JSON тип стандартно:

```
-- удаление
ALTER TABLE staff DROP COLUMN roles;
```

```
-- модификация
ALTER TABLE staff MODIFY COLUMN roles JSON;
```