# MySQL

Таблицы: создание и типы данных

# Создание таблиц

# Подготовка базы данных к созданию таблиц

Основная сущность реляционной данных - таблица - не может существовать сама по себе, она должна находиться в какой-либо базе данных. Это значит, что сначала нам надо создать базу данных и указать, что именно в контексте этой базы будет происходить работа с таблицами.

```
-- создание базы данных shop (для тренировки укажем кодировку явно)

CREATE DATABASE shop DEFAULT CHARSET utf8mb4;
```

```
-- выбор базы данных для работы

USE shop;
```

# Пример создания простейшей таблицы

```
-- создание таблицы products
CREATE TABLE products (
      name TEXT,
      code TEXT
);
```

Возможное содержание таблицы **products**:

| name | code |
|---|---|
| iPhone 14 Plus | S8DF7S87DF87S8 |
| GeForce RTX 4080 | DF987GD8F7G87F |
| Карбюратор трактора "Беларусь" МТЗ-50 | F3K4J5L3J4KLJE |

# Пример взаимодействия с таблицей (добавление и получение данных)

```
-- добавление данных в таблицу products
INSERT INTO products (name, code) VALUES
('iPhone 14 Plus', 'S8DF7S87DF87S8'),
('GeForce RTX 4080', 'DF987GD8F7G87F'),
('Карбюратор трактора "Беларусь" МТЗ-50', 'F3K4J5L3J4KLJE');
```

```
-- получение всех данных из таблицу products
SELECT name, code FROM products;
```

# Получение списка существующих таблиц

Команда для получения списка таблиц может быть выполнена только после того, как мы выбрали какую-либо базу данных для работы (при помощи **USE ...**).

```
SHOW TABLES;
```

Таблицы для базы данных shop:

| Tables_in_shop |
| --- |
| products |

# Безопасное создание таблиц (с проверкой на существование имени)

Чтобы избежать ошибки при создании таблицы (т.е., если таблица с таким именем уже существует), мы можем применять конструкцию **… IF NOT EXISTS …** . Если таблица уже существует, то команда будет проигнорирована, если не существует - таблица будет создана.

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

# Создание таблицы на основе уже существующей таблицы

Представленная ниже команда позволяет скопировать структуру (но не данные) одной таблицы в другую таблицу:

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

# Создание временных таблиц

Таблицы, созданные при помощи модификатора **TEMPORARY**, существуют только на время подключения к базе данных. Затем временные таблицы и все данные, сохраненные в эти таблицы, автоматически удаляются.

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

# Создание таблицы с указанием движка

По умолчанию все таблицы в MySQL 8 создаются на основе InnoDB - самого мощного и универсального движка. Однако есть два способа изменить это поведение.

1. В конфигурационном файле my.ini в разделе **[mysqld]** создать переменную **default-storage-engine** и вписать в нее название нужного нам движка, например **MEMORY** (хранит данные таблицы в оперативной памяти компьютера):

```
default-storage-engine=MEMORY
```

2. Указать движок во время создания таблицы:

```
CREATE TABLE clients (
    firstname VARCHAR(255),
    lastname VARCHAR(255)
) ENGINE = MEMORY;
```

# Создание таблицы с указанием кодировки

Все таблицы в MySQL 8 создаются с кодировкой своей базы данных, которая по умолчанию равняется **utf8mb4**. Однако у нас опять целых три способа это изменить:

1. Изначально создать базу данных с другой кодировкой:

```
CREATE DATABASE some_name DEFAULT CHARSET latin5;
```

2. В конфигурационном файле my.ini в разделе **[mysqld]** создать переменную **character-set-server** и вписать в нее название нужной нам кодировки, например, **big5.** Отныне все новые базы и их таблицы по умолчанию будут создаваться именно в **big5**:

```
character-set-server=big5
```

3. Указать кодировку во время создания таблицы:

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

# Комбинирование настроек при создании таблиц

При создании таблиц можно указывать несколько настроек (кодировка, движок, проверка на существование), если эти эти настройки не противоречат друг другу:

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

```
-- комбинирование TEMPORARY, IF NOT EXISTS И LIKE
CREATE TEMPORARY TABLE IF NOT EXISTS visitors LIKE clients;
```

# Модификация настроек таблиц после создания

```
-- изменение движка
ALTER TABLE clients ENGINE = InnoDB;
```

```
-- изменение кодировки
ALTER TABLE languages DEFAULT CHARSET = utf8mb4;
```

# Удаление таблиц

```
-- стандартное удаление таблицы
DROP TABLE languages;
```

```
-- удаление таблицы с проверкой (даже если таблицы уже нет - ошибки не будет)
DROP TABLE IF EXISTS languages;
```

# Безопасные названия таблиц

Если в качестве названий таблиц использовать зарезервированные слова MySQL, то мы немедленно столкнемся с ошибкой:

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

Однако в случае необходимости мы может избежать ошибки - для этого надо обернуть название в обратные кавычки:

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

# Ограничение на создание столбцов с одинаковыми названиями

В MySQL запрещено в одной и той же таблице создавать столбцы с одинаковыми именами:

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

# Дополнительные атрибуты столбцов таблиц

Во время создания таблицы мы можем задавать не только параметры таблицы, но и параметры каждого конкретного столбца. Например, атрибут **NOT NULL** означает, что столбец должен всегда быть заполнен, а атрибут **CHARACTER SET ...** устанавливает кодировку для конкретного столбца (разные столбцы в таблице могут иметь разные кодировки).

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

# Информация о столбцах таблицы

Получение данных о структуре таблицы products:

```
DESCRIBE products;
```

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

# Получение примера запроса на создание уже существующей таблицы

```
SHOW CREATE TABLE products;
```

| Table | Create Table |
| --- | --- |
| products | CREATE TABLE `products` (<br>  `name` text,<br>  `code` text<br>) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4<br>COLLATE=utf8mb4_0900_ai_ci |

# Типы данных 1

Символьный тип данных

# Жизненный цикл кодировок при работе с символьными данными

При подключении пользователя MySQL несколько раз автоматически меняет кодировку, что может отразиться на работе с таблицами.

1. СУБД считывает запрос с помощью клиентской кодировки (эту кодировку можно узнать через переменную **@@character_set_client**).

2. Потом MySQL преобразовывает запрос в кодировку соединения (переменная **@@character_set_connection**), в которой и хранит данные запроса во время всего подключения, а также именно из нее происходит преобразование в кодировку столбцов таблицы - при вставке, выборке и т.д.

3. Если некий запрос возвращает данные, то он делает это в результирующей кодировке (переменная **@@character_set_results**).

```
-- получение информации о кодировках
SELECT @@character_set_client, @@character_set_connection, @@character_set_results;
```

# Изменение кодировок после подключения к базе данных

Изменение клиентской и результирующей кодировок, а также кодировки соединения для клиента командной строки mysql (в примере мы устанавливаем кодировку **utf8mb4**):

```
CHARSET utf8mb4;
```

---

Изменение кодировок для программ, написанных на языках программирования Python, Java, PHP и т.д. (для примера используется кодировка **latin1**):

1 вариант:

```
SET NAMES latin1;
```

2 вариант:

```
SET @@character_set_client = latin1;
SET @@character_set_connection = latin1;
SET @@character_set_results = latin1;
```

# CHAR и VARCHAR: основы

- **VARCHAR** - максимальный размер хранимых данных ограничен 65535 байтами
- **CHAR** - не может хранить больше 255 символов
- При создании столбца для обоих типов необходимо явно указывать максимальный размер хранимых символов (даже для **VARCHAR**)

Пример создания таблицы со столбцами типа **CHAR** и **VARCHAR:**

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

# CHAR и VARCHAR: особенности хранения

**CHAR** всегда занимает максимально возможный размер данных своего столбца, в то время как **VARCHAR** пытается подстроиться и хранит ровно столько, сколько занимают данные, а также длину сохраненной строки:

| 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 |

# BINARY и VARBINARY

**BINARY** и **VARBINARY** схожии с парой **CHAR** и **VARCHAR**, но их основная задача - хранить бинарные данные (файлы).

- Максимальный размер BINARY - 255 байт
- Максимальный размер VARBINARY - 65535 байт
- Всегда хранятся в кодировке **binary**

Пример создания таблицы со столбцами типа **BINARY** и **VARBINARY:**

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

# Прочие символьные типы данных

| 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);** |

* Представленным выше типам не надо прописывать максимально возможный размер данных.

# Перечисление (ENUM)

- Тип **ENUM** позволяет хранить только заранее определенные значение.
- Максимальное количество вариантов значений для **ENUM** - 65535.
- При хранении в столбце таблицы значение **ENUM** занимает два байта.
- Без применения атрибута **NOT NULL**, если при вставке не упоминать **ENUM** столбец, то в него автоматически будет вставлено первое из разрешенных значений

Пример создания таблицы со столбцом типа **ENUM**:

```
CREATE TABLE graphics (
      name VARCHAR(255),
      y_month ENUM('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov','Dec')
);
```

# Множество (SET)

- Тип **SET** позволяет хранить любую комбинацию из заранее определенных значений.
- Максимальное количество вариантов значений для **SET**- восемь.
- При хранении в столбце таблицы значение **SET** занимает не больше 8 байт.

Пример создания таблицы со столбцом типа **SET**:

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

Все возможные варианты значения столбца position таблицы employees: **'programmer', 'administrator', 'manager', 'programmer,administrator', 'programmer,manager', 'administrator,manager', 'programmer,administrator,manager'**

# Атрибуты для символьных типов данных

Установка кодировки (**CHARACTER SET**):

```
CREATE TABLE tags(
    tag_name VARCHAR(255) CHARACTER SET utf8mb4
);
```

Разрешение и запрещение отсутствия значения (**NULL** и **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
);
```

# Значение по умолчанию для CHAR и VARCHAR

В случае отсутствия значения в столбец таблицы с типом **CHAR** или **VARCHAR** 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.

# Модификация и удаление столбцов символьных типов

Модификация столбца (**ALTER TABLE … MODIFY COLUMN …**):

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

Удаление столбца таблицы (**ALTER TABLE … DROP COLUMN …**):

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

# Типы данных 2

Целые числа

# Целочисленные столбцы и их создание

В целочисленных столбцах можно хранить положительные целые числа (10, 127, 231), отрицательные целые числа (-7, -20, -55), а также ноль (0).

Пример создания таблицы с целочисленным столбцом **number**:

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

# Дополнительные разрешенные значения

- При попытке вставить дробные числа(например 1.6, 99.99 или 115.13) в столбцы с целочисленным типом, MySQL совершит округление согласно правилам математики и сохранит данные в целом виде (2, 100, 115).

- Если в целочисленный столбец попытаться вставить текст, то в том случае, если текст можно представить в виде целого числа ('-99', '33', '92349'), текст сохранится именно в виде целого числа (-99, 33, 92349).

- При вставке в целочисленный столбец текста, который может быть представлен в виде дробных чисел ('45.45', '11144.734234' или '456.79'), то сначала текст действительно будет представлен как дробное число, а потом округлен согласно правилам математики и сохранен в целом виде (45, 11145, 457).

- Также в целочисленные столбцы разрешена вставка целых чисел в шестнадцатеричном формате (0x3E8) и двоичном формате (0b111). Однако при выборке мы будем получать их в обычном десятичном формате (1000 и 7 соответственно).

# Атрибуты для целочисленных типов данных

- Как и символьные типы данных, целочисленные столбцы могут разрешать и запрещать отсутствие значения при помощи атрибутов **NULL** и **NOT NULL**.

- Целочисленные столбцы поддерживают вставку значения по умолчанию при помощи атрибута **DEFAULT**:

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

- Также целочисленным столбцам можно устанавливать запрет на отрицательные числа при помощи атрибута **UNSIGNED**. В этом случае диапазон разрешенных положительных чисел увеличится вдвое:

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

# Список целочисленных типов

| 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);** |
| 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);** |

# Логический тип данных

Кроме целочисленных типов, в MySQL существует еще тип **BOOLEAN**. Как видно из названия, теоретически он должен хранить логические сущности **TRUE** и **FALSE**, однако на самом деле это не так. MySQL пытается экономить типы и при создании столбца подменяет этот тип на **TINYINT(1)**. Подразумевается, что клиент должен хранить в нем 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
);
```

# Модификация и удаление целочисленных столбцов

Модификация столбца (**ALTER TABLE … MODIFY COLUMN …**):

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

Удаление столбца таблицы (**ALTER TABLE … DROP COLUMN …**):

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

# Типы данных 3

Дробные (вещественные) числа

# Типы вещественных чисел

- В MySQL вещественные числа подразделяются на две группы - т.н. числа с фиксированной точкой (имеют большую точность) и числа с плавающей точкой (имеют приблизительную точность).
- Тип числа с фиксированной точкой - **DECIMAL** (также у него есть псевдоним **NUMERIC**).
- Типы числа с плавающей точкой - **FLOAT** и **DOUBLE**.
- В старых версиях MySQL все типы вещественных чисел поддерживали атрибут **UNSIGNED**, однако вел он себя немного странно - он запрещал вставку в столбец отрицательных чисел, но не расширял диапазон положительных чисел. В современных версиях MySQL применение **UNSIGNED** в контексте вещественных чисел объявлено устаревшим подходом.

# Числа с фиксированной точкой (DECIMAL или NUMERIC)

- Для чисел с фиксированной точкой можно заранее указывать ту часть числа, которая отводится под целые цифры, и ту часть, которая отводится под дробные цифры.

- Явное объявление целой и дробной части не является обязательным условием при объявлении столбца типа **DECIMAL**, однако без этого объявления **DECIMAL** начнет вести себя как целочисленный тип - т.е. округлять числа до целых значений:

```
-- если вставить в столбец salary число 7.5, то оно будет округлено до 8
CREATE TABLE salaries (
        salary DECIMAL
);
```

- Чтобы заставить **DECIMAL** хранить дробные числа, надо в скобках сначала указать общее количество разрешенных цифр, а потом количество цифр, отведенных под дробную часть.

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

# Ограничения чисел с фиксированной запятой

- 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.

# Числа с плавающей точкой (FLOAT и DOUBLE)

- Основная задача чисел с плавающей точкой - использование в таких научных или инженерных задачах, где допускается приблизительность. **FLOAT** и **DOUBLE** идеального подходят для того, что отобразить эту неопределенность. Если же мы хотим хранить точные данные, то лучше использовать только **DECIMAL** (а еще лучше использовать целые числа). Единственное преимущество **FLOAT** и **DOUBLE** перед **DECIMAL** - это более оптимальное хранение, что 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.

```
-- правильное создание таблицы со столбцами типа FLOAT и DOUBLE
CREATE TABLE scientific_calculations (
        simple_calculations_result FLOAT,
        difficult_calculations_result DOUBLE
);
```

# Тип данных FLOAT (с плавающей точкой)

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 точность этой дробной части будет соблюдена при хранении.

```
-- пример создания таблицы со столбцом типа FLOAT
CREATE TABLE calculations_values (
        value FLOAT
);
```

# Тип данных DOUBLE (с плавающей точкой)

Тип данных **DOUBLE** поддерживает дробные числа, которые помещаются в 8 байт - т.е. можно хранить числа от -1.7976931348623157E+308 и до 1.7976931348623157E+308. Каждое такое число будет занимать 8 байт.

```
-- пример создания таблицы со столбцом типа DOUBLE
CREATE TABLE big_calculations_values (
    value DOUBLE
);
```

# Модификация и удаление вещественных столбцов

Модификация столбца (**ALTER TABLE … MODIFY COLUMN …**):

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

Удаление столбца таблицы (**ALTER TABLE … DROP COLUMN …**):

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

# Типы данных 4

Дата и время

# Список типов даты и времени

Типы даты и времени являются по своей сути строками в строго заданном формате.

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

# TIMESTAMP и особенности временных зон

- Тип **TIMESTAMP** не просто хранит присланное ему значение, а 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.

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

```
-- для сесии
SET @@SESSION.time_zone = '+03:00';

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

# Значения по умолчанию для типов TIMESTAMP и DATETIME

Важной особенностью **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:

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

Кроме того, **DATETIME** и **TIMESTAMP** могут не только автоматически вставлять время момента добавления, но также умеют менять свое значение при обновлении какого либо другого столбца из их строки:

```
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
);
```

# Модификация и удаление столбцов даты и времени

Модификация столбца (**ALTER TABLE … MODIFY COLUMN …**):

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

Удаление столбца таблицы (**ALTER TABLE … DROP COLUMN …**):

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

# Типы данных 5

JSON

# Взаимодействие с JSON

- В столбцах типа **JSON** 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 данных в формате JSON
-- {“name”: “John Smith”, “age”: 45, “salary”: 999.99, “married”: true, “citizenship”: [“USA”, “Germany”]}

-- пример создания таблицы со столбцом JSON
CREATE TABLE staff (
        name VARCHAR(255),
        roles JSON
);
```

- **JSON** может занимать такой же размер, как и **LONGTEXT**, т.е. до 4 гигабайт.
- **JSON** может иметь только одно значение по умолчанию - **NULL**, остальные запрещены.

# Модификация и удаление столбцов типа JSON

Модификация столбца (**ALTER TABLE … MODIFY COLUMN …**):

```
ALTER TABLE staff MODIFY COLUMN roles JSON NOT NULL;
```

Удаление столбца таблицы (**ALTER TABLE … DROP COLUMN …**):

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