MySQL

ДОБАВЛЕНИЕ, ОБНОВЛЕНИЕ И УДАЛЕНИЕ ДАННЫХ

СОДЕРЖАНИЕ

ДОБАВЛЕНИЕ ДАННЫХ

ВВЕДЕНИЕ

- Добавление данных является одним из самых простых методов взаимодействия с СУБД MySQL. В отличие от получения данных, вставка может быть описана гораздо меньшим количеством ключевых слов и конструкций (если кратно, то надо помнить только 2 оператора - **INSERT INTO** и **REPLACE INTO**, а также ключевое слово **VALUES**). Но и в данном случае есть много “подводных камней” и особенностей, знание которых с одной стороны позволит избежать потенциальных проблем, а с другой - создавать более эффективные и элегантные решения (приложения, сайты и т.д.) на базе MySQL.

- Чтобы начать работать с добавлением данных создадим две таблицы - простые гости (guests) и клиенты (clients). Код для создания обеих таблиц приведен ниже:

```
CREATE TABLE guests (
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    visit_date DATE,
    total_paid DECIMAL(20,2) DEFAULT 0
);

CREATE TABLE clients (
    id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    verified BOOLEAN NOT NULL,
    created_at DATE
);
```

ПРОСТЕЙШИЕ СПОСОБЫ ДОБАВЛЕНИЯ ДАННЫХ

- Основным оператором для вставки является **INSERT INTO**. Именно с него начинается самый распространенный запрос на добавление данных, затем следует название таблицы (куда данные будут вставлены), после этого в скобках через запятую перечисляются столбцы (можно перечислять не все, а лишь необходимые), для которых будут вставляться значения. Потом идет ключевое слово **VALUES**, а уже после него в скобках через запятую перечисляются значений столбцов (должны идти в том

же порядке, в котором в этом же запросе столбцы были перечислены после названия таблицы). Естественно, что словесное описание - не лучший вариант для таких случаев, поэтому приведем пример - вставим гостя в таблицу guests:

```
INSERT INTO guests (
    id, name, visit_date, total_paid
) VALUES (
    1, 'John Smith', '2023-01-01', 0
);
```

Обратим внимание, что имя гостя и дата его визита указаны в кавычках. Кавычки обязательны для всех текстовых типов данных (**VARCHAR**, **CHAR**, **TEXT**, **BLOB** и т.д.), типов даты и времени (**DATE** и **DATETIME**), а также для столбцов типа **JSON**. Притом совершенно неважно, используем ли мы одинарные или двойные кавычки - MySQL понимает оба варианта. После вставки гостя таблица будет выглядеть следующим образом:

id	name	visit_date	total_paid
1	John Smith	2023-01-01	0.00

- Зададим вопрос - что делать, если запрос на добавление данных сам по себе содержит кавычки? Если оставить все как есть, то при вставке произойдет ошибка. Однако есть очень простой способ этого избежать - в текстовых значения надо добавить обратный слэш \ перед каждой кавычкой (т.е. произвести экранирование) - тогда запрос пройдет без нарушений и данные будут вставлены нормально (притом сами экранирующие символы будут отброшены и сохранены не будут):

```
INSERT INTO guests (
    id, name, visit_date, total_paid
) VALUES (
    2, 'Martin O\'Brien', '2023-01-02', 0
);
```

id	name	visit_date	total_paid
1	John Smith	2023-01-01	0.00
2	Martin O'Brien	2023-01-02	0.00

- Два предыдущий раза мы использовали при запросу именно тот порядок столбцов, который определен при создании базы данных. Важно знать что в этом случае (когда мы хотим вставить все столбцы именно в этом порядке), можно использовать упрощенную версию INSERT INTO, в которой вообще нет перечисления столбцов:

```
INSERT INTO guests VALUES (
    3, 'Mary Sue', '2023-01-03', 20
);
```

id	name	visit_date	total_paid
1	John Smith	2023-01-01	0.00
2	Martin O'Brien	2023-01-02	0.00
3	Mary Sue	2023-01-03	20.00

- Теперь приведем пример, когда столбцы для добавления указаны в произвольном порядке. Хотя и в этом случае мы должны четко понимать, что значение для слобцов быть именно в том порядке в котором указаны столбцы в запросе (не может столбец id быть на втором месте, а значение для него на четвертом - в этом случае оно обязано быть перечислено вторым):

```
INSERT INTO guests (
    visit_date, id, total_paid, name
) VALUES (
    '2023-01-04', 4, 35.99, 'Joe Maldini'
);
```

id	name	visit_date	total_paid
1	John Smith	2023-01-01	0.00
2	Martin O'Brien	2023-01-02	0.00
3	Mary Sue	2023-01-03	20.00
4	Joe Maldini	2023-01-04	35.99

- Как уже упоминалось, иногда нам не надо перечислять все столбцы и все значения для них. Например, для таблицы guests значение id может генерироваться само, т.к. при создании

этого столбца мы добавили атрибут **AUTO_INCREMENT**, столбец visit_date вообще не является обязательным - если не предоставить для него значение, то в базу данных сохранится NULL. В свою очередь total_paid имеет значение по умолчанию 0 - оно и попадет в таблицу, если мы не укажем значение явно. Приведем пример всего вышесказанного (т.е. вставим в таблицу только значение столбца name) и убедимся, что запрос выполнится успешно:

```
INSERT INTO guests (name) VALUES ('Rolf Meier');
```

id	name	visit_date	total_paid
1	John Smith	2023-01-01	0.00
2	Martin O'Brien	2023-01-02	0.00
3	Mary Sue	2023-01-03	20.00
4	Joe Maldini	2023-01-04	35.99
5	Rolf Meier	NULL	0.00

- Можно заметить, что в предыдущем запросе, как и ожидалось, значение столбца id было сгенерировано автоматически (т.к. это столбец с автоинкрементом). В таких случая (при автоматическом создании значения через автоинкремент, если прописывать значение вручную, то этот функционал недоступен) у нас появляется возможность узнать, какое именно последнее значение было сгенерировано. Для этого используется специальная функция **LAST_INSERT_ID()**. Если мы запросим у этой функции данные сразу последнего запроса (когда был вставлен Rolf Meier), то получим в ответ пятерку (5):

```
SELECT LAST_INSERT_ID();
```

LAST_INSERT_ID()
5

- Если запросов с автоматической вставкой автоинкремента в контексте данного подключения к базе данных не было, MySQL будет всегда возвращать нуль (0).

ДОБАВЛЕНИЕ НЕСКОЛЬКИХ СТРОК ОДНОВРЕМЕННО

- До этого мы рассматривали добавление данных в контексте вставки только одной строки. Однако MySQL позволяет вставлять несколько строк одновременно - для этого надо просто перечислить скобки со значениями столько раз, сколько нам необходимо строк, разделив эти скобки запятыми. Количество строк теоретически неограниченно, но если MySQL серверу не хватит памяти, запрос на добавление данных закончится неудачей.

- На количество разрешенной памяти влияет файл **my.ini** и его настройка **max_allowed_packet**. Эта настройка устанавливает то количество разрешенное памяти, которое может занимает каждый конкретный запрос. Она может установлена как в разделе клиента ([mysql] - т.е. максимальный размер отправляемого запроса), так и в разделе сервера ([mysqld] - максимальный размер принимаемого запроса). По умолчанию клиенту выделено 16MB, а серверу - 64MB[1].

- Приведем пример добавления нескольких строк за один раз. В этом случае вставим данные в таблицу clients:

```
INSERT INTO clients (
    name, verified, created_at
) VALUES
('Jack Hawkins', 0, '2023-02-01'),
('Sheila Elliott', 1, '2023-02-02'),
('Rosa Fernandez', 1, '2023-02-03');
```

Обратим внимание, что значения столбца id были сгенерированы автоматически с учетом того, что в таблицу были вставлены несколько строк - каждая строка получила свой уникальный id, и не было попытки вставить всем одинаковое значение.

id	name	verified	created_at
1	**Jack Hawkins**	0	**2023-02-01**
2	**Sheila Elliott**	1	**2023-02-02**
3	**Rosa Fernandez**	1	**2023-02-03**

[1] https://dev.mysql.com/doc/refman/8.0/en/packet-too-large.html

- Существует еще один способ вставки нескольких строк одновременно. Сам по себе он не гарантирует, что данные вообще будут вставлены, но сейчас мы поймем, что вероятность этого весьма высока. Дело в том, что в этом случае в качестве источника данных выступает не данные, предоставляемые пользователей, а данные, получаемые из другой таблицы. Мы делаем запрос в некую таблицу при помощи **SELECT**, выбирая необходимый набор столбцов, и сразу же после этого вставляем их в таблицу при помощи **INSERT INTO**. Попробуем преобразовать всех гостей (таблица guests) в клиентов (таблица clients) в рамках одного запроса:

```
INSERT INTO clients (
    name, verified, created_at
) SELECT
    name,
    IF(total_paid > 0, 1, 0),
    '2023-02-10'
FROM guests;
```

Заметим, что названия столбцов в **SELECT** не имеют значения, важен их порядок - он должен по смыслу совпадать с тем порядком столбцов, который определен в **INSERT**. Кроме того, не все данные могут выбираться из таблицы напрямую. Например, значение для verified мы устанавливаем на лету для каждой строки (проверенными станут только те гости, которые потратили хоть какое-то количество денег), а значение created_at мы просто прописали вручную - '2023-02-10':

id	name	verified	created_at
1	**Jack Hawkins**	0	**2023-02-01**
2	**Sheila Elliott**	1	**2023-02-02**
3	**Rosa Fernandez**	1	**2023-02-03**
4	**John Smith**	0	**2023-02-10**
5	**Martin O'Brien**	0	**2023-02-10**
6	**Mary Sue**	1	**2023-02-10**
7	**Joe Maldini**	1	**2023-02-10**
8	**Rolf Meier**	0	**2023-02-10**

- **INSERT … SELECT** - довольно мощный инструмент, но данная конструкция имеет один недостаток. Чтобы его увидеть, внесем в таблицу еще одну строку:

```
INSERT INTO clients (
    name, verified, created_at
) VALUES ('Ava Harding', 0, '2023-04-01');
```

id	name	verified	created_at
1	**Jack Hawkins**	0	**2023-02-01**
2	**Sheila Elliott**	1	**2023-02-02**
3	**Rosa Fernandez**	1	**2023-02-03**
4	**John Smith**	0	**2023-02-10**
5	**Martin O'Brien**	0	**2023-02-10**
6	**Mary Sue**	1	**2023-02-10**
7	**Joe Maldini**	1	**2023-02-10**
8	**Rolf Meier**	0	**2023-02-10**
11	**Ava Harding**	0	**2023-04-01**

Очевидно, что произошла странность - значение автоинкремента возросло скачкообразно (оно должно было равняться 9, а стало 11). Это не вина последней вставки, главный виновник - **INSERT … SELECT**. Дело в том, при использовании такой конструкции не гарантируется, что значение автоинкремента будет соразмерно вставленным данным, но гарантируется, что это значение будет уникальным. Это можно предотвратить, если поставить в **my.ini** настройку **innodb_autoinc_lock_mode=0**, которая блокирует таблицу во время вставки и вставляет строки строго одну за другой. Но лучше этого не делать, т.к. блокировка может снизить скорость работы с этой таблицей.

ДОБАВЛЕНИЕ ДАННЫХ С ОБНОВЛЕНИЕМ

- Иногда мы хотим вставить данные, которые должны частично обновить уже существующие строки в таблице, а частично создать новые строки. Допустим у нас в административной панели есть список пользователей и есть возможность

добавлять новых. Приходят данные о пользователях все сразу - следовательно часть мы должны обновить, а часть - создать как новых. MySQL позволяет это сделать при помощи конструкции **INSERT INTO … ON DUPLICATE KEY UPDATE … .**

- Данная конструкция работает следующим образом - если в данных нет столбцов (или столбцы есть, но они имеют значение **NULL**), которые являются ключом (первичным или уникальным), то произойдет вставка данных. Если же значения таких столбцов есть - для этой строки будет выполняться условие, которое указано после ключевого слова **UPDATE.** Приведем пример такого запроса в контексте таблицы clients. Попытаемся обновить имя и верифицированность для клиента с id = 4, и одновременно добавить нового клиента:

```
INSERT INTO clients (id, name, verified, created_at)
VALUES
(4, 'Homer Simpson', 1, '2023-05-01'),
(NULL, 'Josh Dolton', 0, '2023-05-01') AS new
ON DUPLICATE KEY UPDATE
    name = new.name, verified = new.verified;
```

Заметим, что для вставляемых данных мы использовали псевдоним new и обращались к нему, когда хотели выполнить обновление.

id	**name**	**verified**	**created_at**
1	Jack Hawkins	0	2023-02-01
2	Sheila Elliott	1	2023-02-02
3	Rosa Fernandez	1	2023-02-03
4	Homer Simpson	1	2023-02-10
5	Martin O'Brien	0	2023-02-10
6	Mary Sue	1	2023-02-10
7	Joe Maldini	1	2023-02-10
8	Rolf Meier	0	2023-02-10
11	Ava Harding	0	2023-04-01
12	Josh Dolton	0	2023-05-01

- У конструкции **INSERT INTO … ON DUPLICATE KEY UPDATE …** есть альтернатива - это директива **REPLACE INTO.** Какое же между ними различие? Если **INSERT … UPDATE** вставляет строку или обновляет уже существующую, то **REPLACE** с одной стороны тоже вставляет строку, но с другой - при попытке вставить данные с уже существующим ключом (первичным или уникальным) сначала удалит существующую строку, а потом вставит те данные, которые пришли в запросе. Это значит, что при использовании **REPLACE** не будет доступно обновление отдельных столбцов - мы заменяем всю строку. Приведем пример с новой конструкцией:

```
REPLACE INTO clients (id, name, verified, created_at) VALUES
(4, 'John Smith', 1, '2023-06-01'),
(NULL, 'Emily Carter', 0, '2023-06-01');
```

Посмотрев на результат, мы увидим, что автоинкрементное значение id совершило скачок на 1 позицию (ожидалось, что оно будет равно 13, а оно равняется 14). Это связано с “проделками” **ON DUPLICATE KEY UPDATE** из предыдущего шага при работе со строкой с id = 4. Эта конструкция попыталась сначала вставить данные, что увеличило значение автоинкремента на единицу, но ей это не удалось, и она просто обновила строку. **REPLACE INTO** ни в чем не виновата.

id	name	verified	created_at
1	Jack Hawkins	0	2023-02-01
2	Sheila Elliott	1	2023-02-02
3	Rosa Fernandez	1	2023-02-03
4	John Smith	1	2023-06-01
5	Martin O'Brien	0	2023-02-10
6	Mary Sue	1	2023-02-10
7	Joe Maldini	1	2023-02-10
8	Rolf Meier	0	2023-02-10
11	Ava Harding	0	2023-04-01
12	Josh Dolton	0	2023-05-01
14	Emily Carter	0	2023-06-01

ОБРАБОТКА ОШИБОК ПРИ МНОЖЕСТВЕННОМ ДОБАВЛЕНИИ

- Зададимся вопросом - что случиться, если мы вставляем много строк, но некоторые из них имеют неправильные значения - вставляются ли правильные строки или запрос будет отменен полностью? Если не применять никаких особенных конструкций, то произойдет второй вариант. Например, вот такой запрос не пройдет и данные первой (правильной) строки не будут вставлены в таблицу:

```
INSERT INTO clients(id, name, verified, created_at)
VALUES
(NULL, 'Joseph Parker', 1, '2023-05-02'),
(1, 'Some Name', 1, '2023-05-02'),
(NULL, 'Elen Walters', 1, 'wrong-date');
```

- Заметим, что предыдущий запрос все же окажет некоторое влияние на таблицу. Была попытка (хотя и неудачная) вставить 3 строки - значит значение автоинкремента (которое после манипуляций с **REPLACE INTO** равнялось 16 - ведь тогда было вставлено 2 строки после того, как автоинкремент равнялся 14) сместилось на 3, и если мы вставим новую строку, то она получит id = 19 (если мы не укажем id явно).

- А что, если мы хотим, чтобы в таких случая были вставлены правильные строки, а неправильные проигнорированы? Решение этой проблемы есть - MySQL предоставляет конструкцию **INSERT IGNORE** … , которая вставит все возможные строки и даже попытается исправить ошибки в неправильных строках и все равно их вставить:

```
INSERT IGNORE INTO clients(
    id, name, verified, created_at
)
VALUES
(NULL, 'Joseph Parker', 1, '2023-05-02'),
(1, 'Some Name', 1, '2023-05-02'),
(NULL, 'Elen Walters', 1, 'wrong-date');
```

Первая строка была вставлена успешно, вторая отброшена, т.к. она нарушает ограничение первичного ключа, третья строка также была вставлена, но неправильное значение даты 'wrong-date' было автоматически трансформировано в '0000-00-00':

id	name	verified	created_at
1	Jack Hawkins	0	2023-02-01
2	Sheila Elliott	1	2023-02-02
3	Rosa Fernandez	1	2023-02-03
4	John Smith	1	2023-06-01
5	Martin O'Brien	0	2023-02-10
6	Mary Sue	1	2023-02-10
7	Joe Maldini	1	2023-02-10
8	Rolf Meier	0	2023-02-10
11	Ava Harding	0	2023-04-01
12	Josh Dolton	0	2023-05-01
14	Emily Carter	0	2023-06-01
19	Joseph Parker	1	2023-05-02
20	Elen Waters	1	0000-00-00

ОБНОВЛЕНИЕ ДАННЫХ

ОСНОВЫ ОБНОВЛЕНИЯ ДАННЫХ

- Обновление значений строк осуществляется с помощью ключевого слова **UPDATE.** После этого слова необходимо указать название таблицы, затем ключевое слово **SET,** а уже после через запятую перечислить названия обновляемых столбцов и их значения в формате *столбец1 = значение*, *столбец2 = значение* и т.д. Сразу запомним, что обновлять значения всех столбцов каждой строки не надо, можно обновить только некоторые столбцы. Посмотрим обновление на примере и обновим таблицу clients из предыдущей главы - поставим всем клиентам верифицированный статус и дату создания, равную '2023-07-01':

```
UPDATE clients SET verified = 1, created_at = '2023-07-01';
```

id	name	verified	created_at
1	Jack Hawkins	1	2023-07-01
2	Sheila Elliott	1	2023-07-01
3	Rosa Fernandez	1	2023-07-01
4	John Smith	1	2023-07-01
5	Martin O'Brien	1	2023-07-01
6	Mary Sue	1	2023-07-01
7	Joe Maldini	1	2023-07-01
8	Rolf Meier	1	2023-07-01
11	Ava Harding	1	2023-07-01
12	Josh Dolton	1	2023-07-01
14	Emily Carter	1	2023-07-01
19	Joseph Parker	1	2023-07-01
20	Elen Waters	1	2023-07-01

ОБНОВЛЕНИЕ ДАННЫХ ДЛЯ КОНКРЕТНЫХ СТРОК

- С помощью прошлого запроса мы обновили всю таблицу, но эта ситуация происходит довольно редко. Гораздо чаще нам надо обновить какие либо конкретные строки. В данном случае применяется уже хорошо знакомый нам оператор **WHERE**, который ограничивает те строки которые будут обновлены. Попробуем обновить только те строки (поставим им другую дату создания - *created_at*), у которых столбце id имеет значение 4, 5, 6 или столбец *name* которых равен Josh Dolton:

```
UPDATE clients
SET created_at = '2023-01-01'
WHERE id IN (4,5,6) OR name = 'Josh Dolton';
```

id	name	verified	created_at
1	Jack Hawkins	1	2023-07-01
2	Sheila Elliott	1	2023-07-01
3	Rosa Fernandez	1	2023-07-01
4	John Smith	1	2023-01-01
5	Martin O'Brien	1	2023-01-01
6	Mary Sue	1	2023-01-01
7	Joe Maldini	1	2023-07-01
8	Rolf Meier	1	2023-07-01
11	Ava Harding	1	2023-07-01
12	Josh Dolton	1	2023-01-01
14	Emily Carter	1	2023-07-01
19	Joseph Parker	1	2023-07-01
20	Elen Waters	1	2023-07-01

- Иногда при обновлении нам надо не полностью переписать значение, а дополнить его. Например, в нашей таблице guests есть столбец total_paid, который у гостя с именем Mary Sue равен 20. Стало известно, что этот гость заплатил еще 10 евро. Чтобы обновить значение этого столбца, нам не надо

получать его из таблицы, вручную прибавлять к нему 10, а уже затем вставлять обновленное значение в таблицу - это все можно сделать при помощи следующего единого запроса:

```
UPDATE guests
SET total_paid = total_paid + 10 WHERE name = 'Mary Sue';
```

id	name	visit_date	total_paid
1	John Smith	2023-01-01	0.00
2	Martin O'Brien	2023-01-02	0.00
3	Mary Sue	2023-01-03	30.00
4	Joe Maldini	2023-01-04	35.99
5	Rolf Meier	NULL	0.00

ПРИМЕНЕНИЕ CASE … WHEN … THEN … ELSE

- Если в предыдущем случае мы обновляли значение только для одной конкретной строки, то в этот раз попробуем обновить несколько конкретных строк и указать для каждой из них свое значение. Для этого применим уже знакомую нам конструкцию **CASE … WHEN … THEN:**

```
UPDATE guests
SET total_paid = CASE
WHEN name = 'John Smith' THEN 15.99
WHEN name = 'Martin O\'Brien' THEN 22.35
WHEN name = 'Rolf Meier' THEN 85.49
END;
```

Строки с соответствующими именами действительно обновились, но мы с удивлением обнаружим, что обновилась вообще вся таблица, т.к. у всех строк, которые не попали в конструкцию фильтрации, столбец *total_paid* получил значение **NULL.** В связи с этим запомним, что **CASE … WHEN … THEN** без дополнительных условий обновляет всю таблицу (там где условия не подходят, генерируется значение **NULL**) и в будущем будем использовать эту конструкцию только в совокупности с ключевым словом **ELSE.**

id	name	visit_date	total_paid
1	John Smith	2023-01-01	15.99
2	Martin O'Brien	2023-01-02	22.35
3	Mary Sue	2023-01-03	NULL
4	Joe Maldini	2023-01-04	NULL
5	Rolf Meier	NULL	85.49

- Восстановим неправильно обновленные значения таблицы при помощи той же самой конструкции **CASE** … **WHEN** … **THEN**, но в этом случае применим уже упомянутое выше ключевое слово **ELSE**:

```
UPDATE guests
SET total_paid = CASE
WHEN name = 'Mary Sue' THEN 30.00
WHEN name = 'Joe Maldini' THEN 35.99
ELSE total_paid
END;
```

Теперь таблица была обновлена абсолютно правильно, но все же заметим, что запрос проверил абсолютно все строки таблицы, что может негативно сказаться на скорости выполнения запроса, если таблица будет очень большая. Для оптимизации в конце запроса можно указать значение **WHERE** и отфильтровать только те строки, значение которых должно обновиться. Результат запроса не изменится, но выполняться запрос в некоторых случаях будет быстрее:

```
UPDATE guests
SET total_paid = CASE
WHEN name = 'Mary Sue' THEN 30.00
WHEN name = 'Joe Maldini' THEN 35.99
ELSE total_paid
END
WHERE name IN('Mary Sue', 'Joe Maldini');
```

id	name	visit_date	total_paid
1	John Smith	2023-01-01	15.99
2	Martin O'Brien	2023-01-02	22.35

3	Mary Sue	2023-01-03	30.00
4	Joe Maldini	2023-01-04	35.99
5	Rolf Meier	NULL	85.49

- **CASE** … **WHEN** … **THEN** также позволяет обновлять не только один столбец, но и несколько столбцов одновременно - для этого надо чуть-чуть переписать запрос (не будем применять ключевое слово **ELSE**, т.к. **WHERE** оставит для обновления только те строки, которые нам нужны - других там не будет):

```
UPDATE guests
SET
visit_date = CASE
    WHEN name = 'Martin O\'Brien' THEN '2023-01-03'
    WHEN name = 'Rolf Meier' THEN '2023-01-03'
END,
total_paid = CASE
    WHEN name = 'Martin O\'Brien' THEN 48.56
    WHEN name = 'Rolf Meier' THEN 78.33
END
WHERE name IN('Martin O\'Brien', 'Rolf Meier');
```

id	name	visit_date	total_paid
1	John Smith	2023-01-01	15.99
2	Martin O'Brien	2023-01-03	48.56
3	Mary Sue	2023-01-03	30.00
4	Joe Maldini	2023-01-04	35.99
5	Rolf Meier	2023-01-03	78.33

ИСПОЛЬЗОВАНИЕ ДАННЫХ ИЗ ДРУГОЙ ТАБЛИЦЫ

- Как и в случае с добавления данных, для обновления можно использовать не только прописанные вручную значения, но и данные из других таблиц. Например, обновим значение столбца created_at для таблицы clients на основе таблицы guests и ее столбца visit_date (для связки двух таблиц, будем использовать значения столбца name в clients и guests):

```
UPDATE clients c
SET c.created_at = IFNULL((
    SELECT g.visit_date
    FROM guests g
    WHERE g.name = c.name
), c.created_at);
```

Обратим особое внимание, что подзапрос на получение значения обернут в функцию IFNULL, которая вернет нам второй аргумент, если первый аргумент будет равен значению NULL. Не для всех людей в таблице clients существуют соответствующие им строки с такими же именами в таблице guests - следовательно для них в подзапросе вернется NULL. Мы не хотим вставлять NULL, поэтому оставляем оригинальное значение created_at из таблицы clients.

id	name	verified	created_at
1	Jack Hawkins	1	2023-07-01
2	Sheila Elliott	1	2023-07-01
3	Rosa Fernandez	1	2023-07-01
4	John Smith	1	2023-01-01
5	Martin O'Brien	1	2023-01-03
6	Mary Sue	1	2023-01-03
7	Joe Maldini	1	2023-01-04
8	Rolf Meier	1	2023-01-03
11	Ava Harding	1	2023-07-01
12	Josh Dolton	1	2023-01-01
14	Emily Carter	1	2023-07-01
19	Joseph Parker	1	2023-07-01
20	Elen Waters	1	2023-07-01

- Интересно заметить, что предыдущий запрос можно переписать с помощью обычного JOIN, который мы применяли в конструкции SELECT - такой способ является даже более предпочтительным, т.к. он позволяет обновить несколько столбцов и не использовать функцию **IFNULL**. Чтобы продемонстрировать

вышесказанное, перепишем предыдущий запрос уже с конструкцией **JOIN** - результат его исполнения будет такой же:

```
UPDATE clients c
JOIN guests g ON c.name = g.name
SET c.created_at = g.visit_date;
```

- Далее зададимся примерно тем же самым вопросом, какой мы уже задавали в конце главы о добавлении данных. Мы можем обновить множество строк в рамках одного запроса - что же будет, если для каких-то строк (например в контексте конструкции **CASE** … **WHEN** … **THEN**) будут предоставлены правильные данные, а для других - данные с ошибками? Не будем ничего говорить заранее, просто попробуем осуществить такой ошибочный запрос на практике:

```
UPDATE clients
SET created_at = CASE
WHEN id = 1 THEN '2023-08-01'
WHEN id = 2 THEN 'wrong-date'
ELSE created_at
END;
```

Запрос закономерно завершится с ошибкой и данные первой строки обновлены не будут.

- Наш предыдущий запрос окончился неудачно, но у нас есть возможность его выполнить при помощи конструкции **UPDATE IGNORE** - как и в случае добавления данных, все возможные строки будут обновлены, строки с ошибками - по возможности исправлены и тоже обновлены, а те строки, которые невозможно исправить - попросту проигнорированы:

```
UPDATE IGNORE clients
SET created_at = CASE
WHEN id = 1 THEN '2023-08-01'
WHEN id = 2 THEN 'wrong-date'
ELSE created_at
END;
```

Ошибки не будет, первая строка обновится согласно предоставленному значению, а значение для второй строки будет преобразовано в '0000-00-00':

id	name	verified	created_at
1	Jack Hawkins	1	2023-08-01
2	Sheila Elliott	1	0000-00-00
3	Rosa Fernandez	1	2023-07-01
4	John Smith	1	2023-07-01
5	Martin O'Brien	1	2023-01-03
6	Mary Sue	1	2023-01-03
7	Joe Maldini	1	2023-01-04
8	Rolf Meier	1	2023-01-03
11	Ava Harding	1	2023-07-01
12	Josh Dolton	1	2023-01-01
14	Emily Carter	1	2023-07-01
19	Joseph Parker	1	2023-07-01
20	Elen Waters	1	2023-07-01

Не будем смиряться с ошибочным значением и обновим эту строку правильно (например, значением 2023-08-01):

```
UPDATE clients SET created_at = '2023-08-01' WHERE id = 2;
```

- Довольно редким, но все же не особо уникальным случаем при обновлении является подзапрос за данными в таблицу, которая в данный момент обновляется. Например, представим, что нам надо добавить в таблицу clients столбец position и присвоить ему значение, которое соответствует его позиции на основе столбца created_at (те клиенты, которые были созданы раньше - получат первые номера, те, которые позже - последние). Сначала добавим столбец position:

```
ALTER TABLE clients ADD COLUMN position INT UNSIGNED;
```

Затем попробуем обновить столбец таблицы clients на основе столбца created_at:

```
UPDATE clients a
SET a.position = (
    SELECT ROW_NUMBER() OVER(ORDER BY b.created_at)
    FROM clients b WHERE a.id = b.id
    ORDER BY CASE WHEN a.id = b.id THEN 1 ELSE 2 END LIMIT 1
);
```

Мы получим ошибку, смысл которой состоит в том, что нельзя обращаться к той таблице, которую мы обновляем. Но мы должны знать, что это ограничение не техническое, а логическое. Если нельзя обратиться к таблице, то можно обратиться к выборке из таблицы (т.е. получается “двойной” SELECT):

```
UPDATE clients a
SET a.position = ( SELECT c.number FROM (
    SELECT ROW_NUMBER() OVER(ORDER BY b.created_at) AS number
    FROM clients b
    ORDER BY CASE WHEN a.id = b.id THEN 1 ELSE 2 END LIMIT 1
) c );
```

id	name	verified	created_at	position
1	Jack Hawkins	1	2023-08-01	12
2	Sheila Elliott	1	2023-08-01	13
3	Rosa Fernandez	1	2023-07-01	7
4	John Smith	1	2023-01-01	1
5	Martin O'Brien	1	2023-01-03	3
6	Mary Sue	1	2023-01-03	4
7	Joe Maldini	1	2023-01-04	6
8	Rolf Meier	1	2023-01-03	5
11	Ava Harding	1	2023-07-01	8
12	Josh Dolton	1	2023-01-01	2
14	Emily Carter	1	2023-07-01	9
19	Joseph Parker	1	2023-07-01	10
20	Elen Waters	1	2023-07-01	11

- На самом деле, мы немного слукавили в предыдущем пункте, т.к. главной задачей было показать, что нельзя напрямую обращаться к той таблице, которую мы сейчас меняем, но можно обращаться к выборке из нее. Есть гораздо более простой способ осуществить поставленную задачу - при помощи JOIN, который сам по себе создаст нужную нам выборку и предоставит такой же результат как и в предыдущем запросе:

```
UPDATE clients c
JOIN (
    SELECT
        id,
        ROW_NUMBER() OVER(ORDER BY created_at ASC) AS number
    FROM clients
) c2 ON c.id = c2.id
SET c.position = c2.number;
```

СОРТИРОВКА И ОГРАНИЧЕНИЕ ПО КОЛИЧЕСТВУ ПРИ ОБНОВЛЕНИИ

- Как бы это удивительно не звучало, но обновление поддерживает сортировку (**ORDER BY**) и ограничение по количеству при обновлении (**LIMIT**). Можно сразу спросить - а зачем это вообще надо? Признаем, что это может понадобиться довольно редко и в основном необходимо, когда будут исправляться некие ошибки. Представим, что у нас в таблицу clients занесли нового пользователя, но дали ему такое же имя, какое существует в таблице (например, Rolf Meier):

```
INSERT INTO clients (name, verified, created_at, position)
VALUES ('Rolf Meier', 0, '2024-01-01', 14);
```

Позже мы узнали, что это было ошибочное действие и мы должны были дать имя Jonathan Evans. Нам надо исправить ошибку, но как это сделать, если мы уже забыли, какой был id у этой ошибочной строки, и все что мы помним, что ошибочная строка имела позднее время создания (**created_at**). Если мы обновимся при помощи фильтрации по имени, то обновятся сразу две строки, т.к. имя у них одинаковое! Решение этой задачи состоит в том, чтобы перед обновлением надо действительно отфильтровать строки по имени (**WHERE**), затем отсортировать оставшиеся строки (**ORDER BY**) по времени создания от позднего к раннему (чтобы более поздняя строка оказалась первой),

затем взять только первую строку (**LIMIT 1**) и обновить лишь ее. Продемонстрируем это на практике:

```
UPDATE clients c
SET name = 'Jonathan Evans'
WHERE name = 'Rolf Meier'
ORDER BY created_at DESC
LIMIT 1;
```

id	name	verified	created_at	position
1	Jack Hawkins	1	2023-08-01	12
2	Sheila Elliott	1	2023-08-01	13
3	Rosa Fernandez	1	2023-07-01	7
4	John Smith	1	2023-01-01	1
5	Martin O'Brien	1	2023-01-03	3
6	Mary Sue	1	2023-01-03	4
7	Joe Maldini	1	2023-01-04	6
8	Rolf Meier	1	2023-01-03	5
11	Ava Harding	1	2023-07-01	8
12	Josh Dolton	1	2023-01-01	2
14	Emily Carter	1	2023-07-01	9
19	Joseph Parker	1	2023-07-01	10
20	Elen Waters	1	2023-07-01	11
22	Jonathan Evans	0	2024-01-01	14

УДАЛЕНИЕ ДАННЫХ

БАЗОВЫЕ СПОСОБЫ УДАЛЕНИЯ ДАННЫХ

- За удаление данных отвечает оператор **DELETE**, который всегда ставится в самом начале запроса. После него идет ключевое слово **FROM**, а уже после него название той таблицы, данные из которой должны быть удалены. Попробуем произвести удаление из таблицы guests:

```
DELETE FROM guests;
```

Если мы произведем удаление в таком виде (без использования каких либо дополнительных конструкций), то из таблицы guests будут удалены **абсолютно все строки**, без прямой или косвенной возможности их восстановления. Но остаточный эффект присутствия строк в таблице все же может сохраниться. Если в таблице существовал столбец с автоинкрементом, и были вставлены столбцы, то максимальное значение автоинкремента сохранится. Например, у нас в таблице guests было пять записей и последняя запись имела id, равный пяти. Если же после удаления всех записей мы вставим новую строку, то она получит 6 в качестве значения столба этого столбца id. Вставим же эту строку и проверим наше утверждение:

```
INSERT INTO guests (
    name, visit_date, total_paid
) VALUES (
    'John Smith', '2024-01-01', 1
);
```

id	name	visit_date	total_paid
6	John Smith	2024-01-01	1.00

- Существует еще один способ удаления **всех строк** из таблицы - это использование конструкции **TRUNCATE**. Ее применение несколько отличается от того, что мы виде в случае с **DELETE** - сначала пишется само слово **TRUNCATE**, потом ключевое слово **TABLE**, а уже затем название той таблицы, которую мы хотим очистить.

```
TRUNCATE TABLE guests;
```

TRUNCATE производит еще более жестокое удаление, чем **DELETE** - кроме уничтожения всех строк, она также стирает значение автоинкремента. Это значит, что первая вставленная после такой очистки строка получит id, равный 1:

```
INSERT INTO guests (name, visit_date, total_paid) VALUES
('Mary Sue', '2024-01-02', 22.58);
```

id	name	visit_date	total_paid
1	Mary Sue	2024-01-02	22.58

ВОЗМОЖНЫЕ ПРОБЛЕМЫ ПРИ УДАЛЕНИИ ДАННЫХ

- Иногда мы не сможем удалить данные из таблицы как с помощью **DELETE**, как и с помощью **TRUNCATE**. Мы уже знаем, что в таблицах могут применяться так называемые внешние ключи (**FOREIGN KEY**) - значения в неком столбца одной таблицы могут быть равны только тем значениями, которые вписаны в конкретном столбце другой таблицы. Так вот, вспомним, что если зависимая таблица имеет внешний ключ, в котором установлено ограничение **ON DELETE RESTRICT**, и в нее вставлено значение, которое уже вписано в таблицу-источник, то строку с этим значением из таблицы источника удалять нельзя. Попробуем реализовать вышесказанное наглядно - создадим две таблицы - авторы (**authors**) и книги (**books**), где столбец книг author_id ссылается на столбец id авторов:

```
CREATE TABLE authors (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255)
);

CREATE TABLE books (
    id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    author_id INT UNSIGNED NOT NULL,
    title VARCHAR(255) NOT NULL,
    FOREIGN KEY(author_id) REFERENCES authors (id) ON DELETE
RESTRICT ON UPDATE RESTRICT
);
```

Затем занесем в каждую таблицу по одной записи:

```
INSERT INTO authors VALUES (1, 'Mark Twain');

INSERT INTO books VALUES (1, 1, 'Tom Sawyer');
```

А теперь попытаемся удалить все из таблицы authors (сначала при помощи **DELETE**, а потом при помощи **TRUNCATE**):

```
DELETE FROM authors;

TRUNCATE TABLE authors;
```

Как для первого, так и для второго запроса, MySQL вернет нам ошибку - в первом случае (**DELETE**) будет сказано, что нарушаем ограничение внешнего ключа - т.е. есть строки, которые нельзя удалить, а во втором случае будет отмечено, что при наличии внешнего ключа команду **TRUNCATE** нельзя исполнять в любых контекстах. Несмотря на грозное предупреждение, мы все равно можем попытаться удалить данные из таблицы authors - для этого стоит использовать уже знакомое нам ключевое слово **IGNORE**. В этом случае ошибки не будет, но и данные из таблицы authors никуда не денутся.

```
DELETE IGNORE FROM authors;
```

- Чтобы получить право удалить данные из таблицы с авторами, мы сначала должны удалить все ссылающиеся на них строки из таблицы книг - только тогда запрос DELETE будет выполнен успешно (TRUNCATE будет ругаться в любом случае):

```
DELETE FROM books;

DELETE FROM authors;
```

На всякий случай напомним, что такое поведение внешнего ключа будет происходить только в том случае, если на него наложено ограничение **ON DELETE RESTRICT**. Если ограничение будет носить характер **ON DELETE CASCADE**, то запрос на удаление в любом случае пройдет успешно, но из таблицы books исчезнут все строки. А в том случае, если мы применим ограничение **ON DELETE SET NULL**, то в таблице books все

значения столбцов author_id автоматически преобразуются в NULL.

ПРИМЕНЕНИЕ ФИЛЬТРАЦИИ ПРИ УДАЛЕНИИ ДАННЫХ

- Во всех предыдущих случаях (в рамках этой главы) мы удаляли (или пытались удалить) полностью все строки из таблицы. Но что если нам надо удалить не все строки, а только какие-то конкретные - например, конкретного клиента из таблицы clients? Решение элементарное - надо в очередной раз применить оператор **WHERE**, после которого можно написать все необходимые нам условия. Для примера удалим из таблицы clients все строки, у которых время создания (created_at) больше или равно 2023-07-01:

```
DELETE FROM clients
WHERE created_at >= '2023-07-01';
```

id	name	verified	created_at	position
4	John Smith	1	2023-01-01	1
5	Martin O'Brien	1	2023-01-03	3
6	Mary Sue	1	2023-01-03	4
7	Joe Maldini	1	2023-01-04	6
8	Rolf Meier	1	2023-01-03	5
12	Josh Dolton	1	2023-01-01	2

- Как и в случае с обновлением, при удалении нельзя напрямую обращаться к той таблице, из которой мы сейчас удаляем. Например, просто для эксперимента попытаемся удалить из таблицы все строки, кроме тех, у кого id не является четным числом. Это можно сделать и без подзапроса, но для наглядности сделаем это именно так:

```
DELETE FROM clients
WHERE id NOT IN (SELECT id FROM clients WHERE id % 2 = 0);
```

В ответ мы получим закономерную ошибку, что во время удаления из таблицы нельзя зачитывать данные. Решение в этом случае такое же, как и при обновлении - зачитать данные не

из таблицы, а из выборки из таблицы. Далее приведен пример такого успешного запроса:

```
DELETE FROM clients
WHERE id NOT IN (
    SELECT a.id FROM (
        SELECT id FROM clients WHERE id % 2 = 0
    ) a
);
```

id	name	verified	created_at	position
4	John Smith	1	2023-01-01	1
6	Mary Sue	1	2023-01-03	4
8	Rolf Meier	1	2023-01-03	5
12	Josh Dolton	1	2023-01-01	2

- Фильтрации по значениям из какой-либо внешней таблицы не требуют никаких ухищрений и проходят абсолютно стандартно. Например, удалим все строки из таблицы clients, имена (name) которых отсутствуют в таблице guests:

```
DELETE FROM clients
WHERE name NOT IN (
    SELECT name FROM guests
);
```

id	name	verified	created_at	position
6	Mary Sue	1	2023-01-03	4

СОРТИРОВКА И ОГРАНИЧЕНИЕ ПО КОЛИЧЕСТВУ ПРИ УДАЛЕНИИ

- Еще одна общая черта между осуществлением обновления и удаления состоит в том, что удаление поддерживает сортировку (**ORDER BY**) и ограничение по количеству (**LIMIT**). Воспроизведем случай со вставкой в таблицу clients строки, у которой будет имя (name), уже присутствующее в другой, уже ранее сохраненной строке. В данный момент у нас только одна строка с именем Mary Sue, поэтому и продублируем это имя, но дату поставим более позднюю - 2023-01-13:

```
INSERT INTO clients (name, verified, created_at, position)
VALUES ('Mary Sue', 1, '2023-01-13', 5);
```

id	name	verified	created_at	position
6	Mary Sue	1	2023-01-03	4
23	Mary Sue	1	2023-01-13	5

Теперь по аналогии с обновлением осознаем, что совершили ошибку - такую строку вообще не надо было вставлять. Мы уже не помним ничего об этой вставке кроме того, что дата вставки created_at была позднее, чем у первой строки с таким именем. Значит нам надо получить все строки с таким именем (**WHERE name = 'Mary Sue'**), затем отсортировать (**ORDER BY created_at**) по времени создания от позднего к раннему (чтобы более поздняя строка оказалась первой), затем взять только первую строку (**LIMIT 1**) и удалить только ее:

```
DELETE FROM clients
WHERE name = 'Mary Sue'
ORDER BY created_at DESC
LIMIT 1;
```

id	name	verified	created_at	position
6	Mary Sue	1	2023-01-03	4