БАЗЫ ДАННЫХ

Работа с данными

Вставка данных в таблицу

Синтаксис:

```
INSERT INTO `table_name` (список_полей)
VALUES (список_значений);
```

```
INSERT INTO `table_name` VALUES (список_значений);
```

```
INSERT INTO `table_name`
SET поле = значение, поле = значение, ...;
```

Вставка данных в таблицу

Вставка нескольких строк за одну команду:

```
INSERT INTO `table_name` (список_полей) VALUES
(список_значений_для_строки_1),
(список_значений_для_строки_2),
...;
```

```
INSERT INTO `table_name` VALUES
(список_значений_для_строки_1),
(список_значений_для_строки_2),
...;
```

Вставка данных в таблицу

Вставка данных в таблицу **без указания** списка полей означает, что список значений полей для каждой строки **должен** быть именно в том **количестве** и **порядке**, в котором указаны поля заданы **при создании** таблицы.

На практике, лучше **не** полагаться на количество и порядок, а указывать их в явном виде. Кроме того, в таком случае мы можем опустить некоторые поля.

Вставка данных в таблицу

```
INSERT INTO `users` (`login`, `created_at`)
VALUES ('vasya', NOW());
```

```
INSERT INTO `users` (`login`, `type`, `created_at`)
VALUES ('admin', 'ADMIN', NOW());
```

Обратите внимание: мы намеренно не указываем поле `id`, чтобы его значение назначалось автоматически инкремента.

Вставка данных в таблицу

```
INSERT INTO `users` (`login`, `balance`, `created_at`)
VALUES
('petya', 100.00, NOW()),
('kolya', 50.00, NOW()),
('masha', 150.00, NOW());
```

```
INSERT INTO `users` SET
    `login` = 'petya',
    `type` = 'USER',
    `balance` = 100.00,
    `created_at` = NOW();
```

Обновление данных

Синтаксис:

```
  UPDATE таблица
  SET
    поле1 = значение,
    поле2 = значение,
    ...
[ WHERE условие ]
[ LIMIT количество ];
```

Обновление данных

Правила обновления данных:

Перед обновлением данных нужно всегда проверять выборку через `SELECT`.

При обновлении данных нужно ограничивать выборку с помощью `LIMIT`.

Как правило, запуск запросов на обновление без `WHERE` (а тем более без `LIMIT`) может привести к побочным эффектам в виде обновления лишних строк, отменить которое будет невозможно.

Обновление данных

```
UPDATE `users`
SET
    `balance` = 200.00,
    `updated_at` = NOW()
WHERE
    `id` = 1
LIMIT 1;
```

Здесь мы обновляем баланс пользователя, у которого id равен 1.

В поле updated_at помещаем время обновления строки.

Обновление данных

```
UPDATE `users`
SET
    `balance` = 200.00,
    `updated_at` = NOW()
WHERE
    `email` = 'user@mail.com'
LIMIT 1;
```

Такой запрос таит в себе опасность: после первой выборки поле email могло быть изменено, а значит мы можем либо не обновить ничего (в лучшем случае), либо обновить другую строку (а это уже совсем плохо).

Лучшим решением является выборка по первичному ключу (id).

Обновление данных

```
UPDATE `users`
SET
    `balance` = `balance` - 50.00,
    `updated_at` = NOW()
WHERE
    `id` = 1
LIMIT 1;
```

В данном примере мы уменьшаем баланс пользователя на 50 единиц, причем не задаем его фиксировано, а используем текущее значение поля.

Удаление данных

Синтаксис:

```
  DELETE FROM таблица
[ WHERE условие ]
[ LIMIT количество ];
```

Удаление данных

Правила удаления аналогичны правилам обновления:

Перед удалением данных нужно всегда проверять выборку через `SELECT`.

При удалении данных нужно ограничивать выборку с помощью `LIMIT`.

Как правило, запуск запросов на удаление без `WHERE` (а тем более без `LIMIT`) может привести к побочным эффектам в виде удаления лишних строк, отменить которое будет невозможно.

Помните: удаленные данные восстановить невозможно!

Удаление данных

```
DELETE FROM `users`
WHERE `id` = 1
LIMIT 1;
```

Здесь мы удаляем пользователя, у которого id равен 1.

Удаление данных

Удаление всех данных из таблицы:

```
DELETE FROM `users`;
-- эта операция «долгая» и вероятно ошибочная

TRUNCATE `users`;
-- очистка всех данных из таблицы, происходит мгновенно, автоинкремент сбрасывается в начальное положение
```

Удаление таблицы

Синтаксис:

```
DROP TABLE [ IF EXISTS ] таблица;
```

Например:

```
DROP TABLE `users`;
```

Выборка данных

Выборка данных происходит на основе условий, заданных в `WHERE`:

```
WHERE условие
```

Условие может использовать операторы сравнения и быть поделено на логические группы (`AND`, `OR` , `NOT`).

Операторы сравнения

Оператор	Значение
`=`	Равенство
`<>`, `!=`	Неравенство
`>`, `<`	Больше, меньше
`>=`, `<=`	Больше или равно, меньше или равно
`<=>`	NULL-безопасное равенство
`BEETWEEN` *`min`* `AND` *`max`*	Диапазон значений
`IN (`*`a`*`, `*`b`*`, `*`c`*`, …, `*`n`*`)`	Сравнение со списком
`LIKE`	Соответствие (чаще шаблону)
`IS NULL, IS NOT NULL`	Сравнение с NULL

Сравнение с NULL

Значение NULL имеет особый статус. Он означает отсутствие значения для какого-либо поля. Обычное равенство через **=** с ним не работает.

```
NULL = NULL -- вернет 0 (false)
```

Операторы IS NULL и IS NOT NULL:

```
NULL IS NULL -- вернет 1 (true)
```

Оператор «spaceship» (**<=>**):

```
NULL <=> NULL -- вернет 1 (true)
```

Сравнение с NULL

Оператор ISNULL(...):

```
ISNULL(NULL) -- вернет 1 (true)
```

Оператор IFNULL(...):

```
IFNULL(NULL, 2) -- вернет 2
IFNULL(5, 2) -- вернет 5
```

Оператор NULLIF(...):

```
NULLIF(2, 2) -- вернет NULL
NULLIF(5, 2) -- вернет 5
```

Оператор LIKE

Сравнение фиксированных строк или по шаблону:

```
'hello' LIKE 'hello' -- вернет 1 (true)
```

Это эквивалентно `'hello' = 'hello'`.

```
'hello world' LIKE 'hello%' -- вернет 1 (true)
'hello world' LIKE '%llo%' -- вернет 1 (true)

'hello' LIKE 'h_llo' -- вернет 1 (true)
```