MySQL

Урок 3

SQL-команды DML

Научимся, пользуясь SQL, манипулировать данными из нашей БД.

Типы команд DML

Data Manipulation Language (DML) (язык управления или манипулирования данными) – это семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных.

Функции языков DML определяются первым словом в предложении (часто называемом запросом), которое почти всегда является глаголом. В случае с SQL эти глаголы – «select» («выбрать»), «insert» («вставить»), «update» («обновить») и «delete» («удалить»). Это превращает код на языке в последовательность обязательных утверждений (команд) относительно базы данных.

Рассмотрим каждую из команд подробно.

SELECT

Одна из самых сложных по структуре команд в MySQL. Позволяет выбрать любой срез данных из БД. Таблицы можно объединять, данные можно сортировать, объединять, группировать.

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [MAX_STATEMENT_TIME = N]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Рассмотрим теперь отдельно все возможности выборки.

Мы можем проводить вычисления, не используя вообще никаких таблиц.

```
mysql> SELECT 1 + 1;
        -> 2
```

Также можно сделать запрос в несуществующую таблицу:

```
mysql> SELECT 1 + 1 FROM DUAL;
        -> 2
```

Простые запросы

Рассмотрим типовые запросы при выборке данных:

```
SELECT * FROM tbl_name;
```

В данном контексте мы выбираем все поля (*) из таблицы tbl_name. Если мы хотим сделать выборку определенных полей, нам нужно указать их после оператора SELECT.

```
SELECT field1, field2 FROM tbl_name;
```

Поля можно объединять:

```
SELECT field1, field2, field3 + field4 FROM tbl_name;
```

Пользуясь оператором CONCAT, можно сцепить значения колонок в одну. Используя оператор AS, можно задать результирующей колонке специфическое имя full_name.

```
SELECT CONCAT(last_name, ', ', first_name) AS full_name FROM mytable;
```

DISTINCT

Если нам нужно выбрать только уникальные записи в поле, нужно использовать DISTINCT.

```
SELECT DISTINCT field1 AS distinct_field1, field2 FROM tbl_name;
```

WHERE

Теперь рассмотрим условия выборки, для этого у нас есть клауза WHERE:

```
SELECT field1, field2 FROM tbl_name WHERE field1 = 20;
```

Мы выбираем из таблицы tbl_name поля field1, field2 и указываем дополнительно условие, что поле field1 должно быть равно 20.

В WHERE можно указывать любые условия из булевой алгебры, пользуясь следующим синтаксисом выражений.

Операторы сравнения

```
comparison_operator: = | >= | > | <= | < | <> | !=
```

Равенство: =

Неравенство: <>, !=

Больше, меньше, больше или равно, меньше или равно: >, <, >=, <=

Приведем примеры:

```
SELECT field1, field2 FROM tbl_name WHERE field1 > 20;
SELECT field1, field2 FROM tbl_name WHERE field1 != 20;
SELECT field1, field2 FROM tbl_name WHERE field1 <= 20;
```

Булевы операторы

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary
boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
  | boolean_primary comparison_operator {ALL | ANY} (subquery)
  | predicate
```

Приведем несколько примеров:

```
SELECT field1, field2 FROM tbl_name WHERE field1 > 20 OR field2 !=30;
SELECT field1, field2 FROM tbl_name WHERE field1 != 20 AND field2 < 10;
SELECT field1, field2 FROM tbl_name WHERE field1 IS NOT 20;
```

IN, BETWEEN, LIKE, REGEXP

```
predicate:
    bit_expr [NOT] IN (subquery)
  | bit_expr [NOT] IN (expr [, expr] ...)
  | bit_expr [NOT] BETWEEN bit_expr AND predicate
  | bit_expr SOUNDS LIKE bit_expr
  | bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
  | bit_expr [NOT] REGEXP bit_expr
  | bit_expr
```

IN

Оператор IN можно использовать для сравнения с определенным множеством:

```
SELECT field1, field2 FROM tbl_name WHERE field1 IN (118,17,113,23,72);
```

Также можно проводить сравнение с результатом другого запроса:

```
SELECT * FROM table1 WHERE id IN (SELECT id FROM table2);
```

BETWEEN

Выбирать из определенного интервала значений можно с помощью оператора BETWEEN:

```
SELECT field1, field2 FROM tbl_name WHERE field1 BETWEEN 117 AND 220;
```

Также во всех этих выражениях можно использовать оператор NOT, который служит логическим отрицанием:

```
SELECT field1, field2 FROM tbl_name WHERE field1 NOT IN (118,17,113,23,72);

SELECT field1, field2 FROM tbl_name WHERE field1 NOT BETWEEN 117 AND 220;
```

LIKE

Оператор LIKE интересен тем, что позволяет осуществлять поиск по строкам, используя примитивный синтаксис регулярных выражений SQL:

Символ	Описание
%	Соответствует любому количеству символов, даже нулевых
_	Соответствует ровно одному символу

Давайте приведем пример оператора LIKE в действии.

Имена питомцев, которые начинаются с «b»:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

Имена питомцев, которые заканчиваются «fy»:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

Имена питомцев, которые содержат «w»:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Ищем имена, которые состоят из пяти символов:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

REGEXP

Также поддерживается синтаксис полноценных регулярных выражений.

Если нам нужен реально регистрозависимый поиск в строке, нужно использовать ключевое слово BINARY. Следующий пример будет искать строки, начинающиеся с маленькой буквы:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

В базе нет записей, начинающихся с маленькой буквы.

Поищем питомца, имя которого заканчивается на «fy»:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

Подробнее о синтаксисе регулярных выражений можно прочитать в памятке.

Регулярные выражения

Якоря

^	Начало строки +
\A	Начало текста +
$	Конец строки +
\Z	Конец текста +
\b	Граница слова +
\B	Не граница слова +
\<	Начало слова
\>	Конец слова

Символьные классы

\c	Управляющий символ
\s	Пробел
\S	Не пробел
\d	Цифра
\D	Не цифра
\w	Слово
\W	Не слово
\xhh	Шестнадцатиричный символ hh
\Oxxx	Восьмиричный символ xxx

Символьные классы POSIX

[:upper:]	Буквы в верхнем регистре
[:lower:]	Буквы в нижнем регистре
[:alpha:]	Все буквы
[:alnum:]	Буквы и цифры
[:digit:]	Цифры
[:xdigit:]	Шестнадцатиричные цифры
[:punct:]	Пунктуация
[:blank:]	Пробел и табуляция
[:space:]	Пустые символы
[:cntrl:]	Управляющие символы
[:graph:]	Печатные символы
[:print:]	Печатные символы и пробелы
[:word:]	Буквы, цифры и подчеркивание

Утверждения

?=	Вперед смотрящее +
?!	Отрицательное вперед смотрящее +
?<=	Назад смотрящее +
?!= или ?	Отрицательное назад смотрящее +
?>	Однократное подвыражение
?()	Условие [если, то]
?()|	Условие [если, то, а иначе]
?#	Комментарий

Примечание *Отмеченное + работает в большинстве языков программирования.*

Образцы шаблонов

([A-Za-z0-9-]+)	Буквы, числа и знаки переноса
(\d{1,2}\/\d{1,2}\/\d{4})	Дата (напр., 21/3/2006)
([^\s]+(?=\.(jpg|gif|png))\.\2)	Имя файла jpg, gif или png
(^[1-9]{1}$\|^[1-4]{1}[0-9]{1}$|^50$)	Любое число от 1 до 50 включительно
(#?([A-Fa-f0-9]){3}(([A-Fa-f0-9]){3})?)	Шестнадцатиричный код цвета
((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,15})	От 8 до 15 символов с минимум одной цифрой, одной заглавной и одной строчной буквой (полезно для паролей).
(\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,6})	Адрес email
(\<(/?[^\>]+)\>)	HTML теги

Примечание *Эти шаблоны предназначены для ознакомительных целей и основательно не проверялись. Используйте их с осторожностью и предварительно тестируйте.*

Кванторы

*	0 или больше +
*?	0 или больше, нежадный +
+	1 или больше +
+?	1 или больше, нежадный +
?	0 или 1 +
??	0 или 1, нежадный +
{3}	Ровно 3 +
{3,}	3 или больше +
{3,5}	3, 4 или 5 +
{3,5}?	3, 4 или 5, нежадный +

Специальные символы

\	Экранирующий символ +
\n	Новая строка +
\r	Возврат каретки +
\t	Табуляция +
\v	Вертикальная табуляция +
\f	Новая страница +
\a	Звуковой сигнал
[\b]	Возврат на один символ
\e	Escape-символ
\N{name}	Именованый символ

Подстановка строк

$n	n-ая непассивная группа
$2	«xyz» в /^(abc(xyz))$/
$1	«xyz» в /^(?:abc)(xyz)$/
$`	Перед найденной строкой
$'	После найденной строки
$+	Последняя найденная строка
$&	Найденная строка целиком
$_	Исходный текст целиком
$$	Символ «$»

Диапазоны

.	Любой символ, кроме переноса строки (\n) +
(a|b)	a или b +
(...)	Группа +
(?:...)	Пассивная группа +
[abc]	Диапазон (a или b или c) +
[^abc]	Не a, не b и не c +
[a-q]	Буква между a и q +
[A-Q]	Буква в верхнем регистре между A и Q +
[0-7]	Цифра между 0 и 7 +
\n	n-ая группа/подшаблон +

Примечание *Диапазоны включают граничные значения.*

Модификаторы шаблонов

g	Глобальный поиск
i	Регистронезависимый шаблон
m	Многострочный текст
s	Считать текст одной строкой
x	Разрешить комментарии и пробелы в шаблоне
e	Выполнение подстановки
U	Нежадный шаблон

Мета-символы (экранируются)

^	[	.
$	{	*
(	\	+
)	|	?
<	>	

Эта таблица доступна на www.exlab.net
Англоязычный оригинал на AddedBytes.com

GROUP BY и агрегирующие функции

Очень часто (например, при формировании отчетности) перед нами встает задача группировки и агрегации данных. Для этого у нас есть GROUP BY и функции-агрегаторы.

Имя	Описание
AVG()	Среднее значение
COUNT()	Количество записей
MAX()	Максимальное значение
MIN()	Минимальное значение
SUM()	Сумма значений
STD(expr) STDDEV(expr)	Возвращает среднеквадратичное отклонение значения в аргументе expr. Эта функция является расширением ANSI SQL. Форма STDDEV() обеспечивает совместимость с Oracle.

Самый большой практический интерес вызывают первые пять функций.

Функции-агрегаторы используются только совместно с оператором GROUP BY.

Примеры запросов

Вычислить максимальную зарплату у сотрудников:

```
SELECT MAX(salary), CONCAT(last_name, ', ', first_name) AS full_name
FROM tbl_employee
GROUP BY full_name;
```

Вычислить среднюю зарплату у сотрудников:

```
SELECT AVG(salary), CONCAT(last_name, ', ', first_name) AS full_name
FROM tbl_employee
GROUP BY full_name;
```

Вычислить максимальную зарплату у сотрудников из отдела IT:

```
SELECT MAX(salary), CONCAT(last_name, ', ', first_name) AS full_name
FROM tbl_employee
WHERE department = "IT"
GROUP BY full_name;
```

HAVING

Данный оператор очень часто путают с оператором WHERE, ведь если его использовать без GROUP BY, то работают они одинаково. Но смысл этого оператора в том, что он позволяет фильтровать записи после того, как у нас выполнилась выборка и группировка.

Давайте вернемся к спецификации запроса SELECT:

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [MAX_STATEMENT_TIME = N]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
```

Как вы видите, оператор HAVING обладает такой же спецификой, что и оператор WHERE (where_condition), другими словами, эти операторы имеют абсолютно одинаковую семантику, просто выполняются в разные моменты времени.

Не нужно использовать оператор HAVING вместо оператора WHERE, это противоречит стандарту SQL.

Примеры запросов

Выбрать работников из отдела IT с максимальной зарплатой от 10000.

```
SELECT MAX(salary) AS max_salary, CONCAT(last_name, ' ', first_name) AS
full_name
FROM tbl_employee
WHERE department = "IT"
GROUP BY full_name
HAVING max_salary > 10000;
```

Выбрать всех сотрудников, посчитать среднюю зарплату, найти сотрудников у которых зарплата ниже 3000.

```
SELECT AVG(salary) AS avg_salary, CONCAT(last_name, ' ', first_name) AS
full_name
FROM tbl_employee
GROUP BY full_name
HAVING avg_salary < 3000;
```

ORDER BY

Сортировка полученных данных по определенной колонке. Есть два вида сортировки – ASC и DESC (ascending, descending order – сортировка по возрастанию / убыванию). Если явно не указывать вид сортировки, используется ASC.

Примеры запросов

Вычислить среднюю зарплату сотрудников, найти зарплаты более 10000 и отсортировать по возрастанию.

```
SELECT AVG(salary) AS avg_salary, CONCAT(last_name, ' ', first_name) AS
full_name
FROM tbl_employee
GROUP BY full_name
HAVING avg_salary > 10000
ORDER BY avg_salary;
```

Вычислить максимальную зарплату у сотрудников из отдела IT, отобрать зарплаты больше 10000, отсортировать по убыванию.

```
SELECT MAX(salary) AS max_salary, CONCAT(last_name, ' ', first_name) AS
full_name
FROM tbl_employee
WHERE department = "IT"
GROUP BY full_name
HAVING max_salary > 10000
ORDER BY max_salary DESC;
```

LIMIT

Оператор используется для ограничения количества выбираемых данных или выборки определенного среза данных со смещением.
LIMIT примечателен тем, что ограничение выборки происходит в самом конце выполнения запроса. MySQL выполняет сам запрос, вернет его полный результат, и только после этого будут отброшены лишние данные. По этим причинам нужно максимально ограничить диапазон выбираемых данных, чтобы не было падения производительности СУБД.

Примеры запросов

Выбрать сотрудника из отдела IT с максимальной зарплатой.

```
SELECT MAX(salary) AS max_salary, CONCAT(last_name, ' ', first_name) AS
full_name
FROM tbl_employee
WHERE department = "IT"
GROUP BY full_name
ORDER BY max_salary DESC
```

```
LIMIT 1;
```

Выбрать сотрудников с 5 по 10 из отдела IT с максимальной зарплатой:

```
SELECT MAX(salary) AS max_salary, CONCAT(last_name, ', ', first_name) AS
full_name
FROM tbl_employee
WHERE department = "IT"
GROUP BY full_name
ORDER BY max_salary DESC
LIMIT 4, 5;
```

JOIN

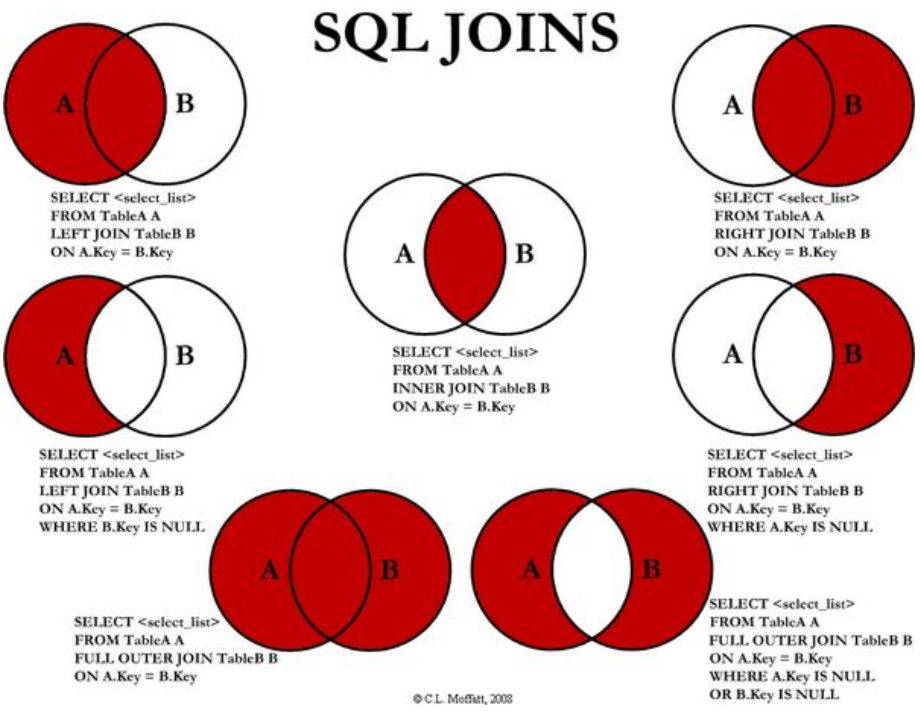
JOIN реализует в себе возможности реляционной алгебры: таблицы можно объединять.

Многие разработчики считают эту тему сложной для понимания, но на самом деле, прежде чем рассматривать синтаксис JOIN, нужно хорошо разобраться, как объединяются таблицы, на диаграммах.

Существуют следующие виды JOIN:

```
join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor
```

Нагляднее всего можно пояснить это при помощи диаграмм.



Для хорошего понимания JOIN нужно представлять себе два и более множеств и четко понимать, какие таблицы объединяются и в каком порядке.

Примеры запросов

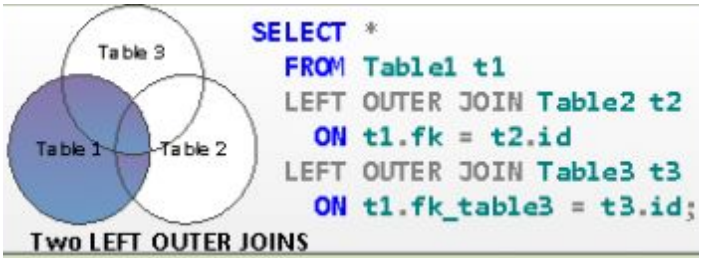
Объединение трех таблиц с помощью LEFT JOIN:

```
SELECT * FROM tbl_table1 LEFT JOIN (tbl_table2, tbl_table3, tbl_table4)
ON     (tbl_table2.a=tbl_table1.a     AND     tbl_table3.b=tbl_table1.b     AND
tbl_table4.c=tbl_table1.c)
```

Можно объединять множества между собой, а не с первой таблицей. При этом объединение происходит с результирующим множеством:

```
SELECT * FROM Table1 t1 LEFT OUTER JOIN Table2 t2
ON t1.fk = t2.id
LEFT OUTER JOIN Table3 t3
ON t1.fk_table3 = t3.id;
```

Чтобы понимать, что при этом происходит, давайте снова представим это в виде диаграммы множеств:



При объединении любого количества множеств в любом порядке нужно всегда представлять диаграмму пересечения множеств.

UNION

В языке SQL ключевое слово UNION применяется для объединения результатов двух SQL-запросов в единую таблицу, состоящую из схожих строк.

Существуют два основных правила, регламентирующие порядок использования оператора UNION:

- Число и порядок извлекаемых столбцов должны совпадать во всех объединяемых запросах.
- Типы данных в соответствующих столбцах должны быть совместимы.

```
SELECT …
UNION [ALL | DISTINCT] SELECT …
[UNION [ALL | DISTINCT] SELECT ...]
```

Примеры запросов

Даны две таблицы:

person	amount
Иван	1000
Алексей	2000
Сергей	5000

person	amount
Иван	2000
Алексей	2000
Петр	35000

При выполнении следующего запроса

```
(SELECT * FROM sales2005)
UNION
(SELECT * FROM sales2006);
```

получается результирующий набор, однако порядок строк может произвольно меняться, поскольку ключевое выражение ORDER BY не было использовано:

person	amount
Иван	1000
Алексей	2000
Иван	2000
Сергей	5000
Петр	35000

В результате отобразятся две строки с Иваном, так как они различаются значениями в столбцах. Но при этом в результате присутствует лишь одна строка с Алексеем, поскольку значения в столбцах полностью совпадают.

Применение UNION ALL дает другой результат, так как дубликаты не скрываются. Выполнение запроса

```
(SELECT * FROM sales2005)
UNION ALL
(SELECT * FROM sales2006);
```

даст следующий результат, выводимый без упорядочивания ввиду отсутствия выражения ORDER BY:

person	amount
Иван	1000
Иван	2000
Алексей	2000
Алексей	2000
Сергей	5000
Петр	35000

UPDATE

Позволяет обновить данные в таблице:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET col_name1={expr1|DEFAULT} [, col_name2={expr2|DEFAULT}] ...
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Примеры запросов

Увеличить зарплату сотрудникам отдела IT на 100:

```
UPDATE tbl_employee emp
SET emp.salary = emp.salary + 100
WHERE department = "IT";
```

Обновить данные из нескольких таблиц:

```
UPDATE items, month SET items.price = month.price
WHERE items.id = month.id;
```

DELETE

Данный оператор используется для удаления записей из таблицы. Синтаксис оператора следующий:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Работу WHERE, ORDER BY и LIMIT мы подробно рассмотрели в разделе, посвященном SELECT. В этой команде данные операторы работают абсолютно аналогично.

Примеры запросов

Удаляем всех сотрудников из отдела IT:

```
DELETE FROM tbl_employee
WHERE department = "IT";
```

INSERT

Позволяет добавить данные в таблицу. Можно добавить одну или несколько записей.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [(col_name,...)]
    {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
```

Или:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
```

[LOW_PRIORITY | DELAYED | HIGH_PRIORITY] приоритет, с которым запись будет добавлена. Конкурентные записи поддерживаются только движком MyISAM (подробнее о движках на уроке 7).

Если вы используете LOW_PRIORITY, запись может не добавляться значительный промежуток времени, а точнее до тех пор, пока другие клиенты осуществляют операции чтения.

По умолчанию записи добавляются в режиме LOW_PRIORITY.

[INTO] tbl_name – в какую таблицу (tbl_name) будет осуществляться добавление записи.

{VALUES | VALUE} – какое значение или значения будут добавляться в таблицу.

Примеры использования

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

```
INSERT INTO tbl_name (a,b,c) VALUE(1,2,3);
```

INSERT ... SELECT

INSERT можно делать на основе SELECT – добавить данные в таблицу из выборки.

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name,...)]
    [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col_name=expr
        [, col_name=expr] ... ]
```

Пример запроса

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Кавычки в MySQL

Давайте сначала поговорим о том, что такое строки в MySQL. Строка представляет собой последовательность символов, заключенных либо в одинарные кавычки (‘ ‘) - апострофы, либо в двойные кавычки (“ “). При использовании диалекта ANSI SQL допустимы только одинарные кавычки. Например:

'a string'
"another string"

Внутри строки некоторые последовательности символов имеют специальное назначение. Каждая из этих последовательностей начинается обратным слешем ('\'), известным как escape-символ или символ перехода. MySQL распознает следующие escape-последовательности:

\0	Символ 0 (NUL) в ASCII-коде.
\'	Символ одиночной кавычки (').
\"	Символ двойной кавычки (").
\b	Возврат на один символ.
\n	Символ новой строки (перевода строки).
\r	Символ перевода каретки.
\t	Символ табуляции.
\z	Символ (Control-Z) таблицы ASCII(26). Данный символ можно закодировать, чтобы обойти проблему, заключающуюся в том, что под Windows ASCII(26) означает конец файла (проблемы возникают при использовании ASCII(26) в выражении mysql database < filename).

\\	Символ обратного слеша.
\%	Символ процентов '%'. Используется для поиска копий литерала '%' в контекстах, где выражение '%' в противном случае интерпретировалось бы как групповой символ.
_	Символ подчеркивания '_'. Используется для поиска копий литерала '_' в контекстах, где выражение '_' в противном случае интерпретировалось бы как групповой символ.

Обратите внимание, что при использовании '\%' или '_' в контекстах некоторых строк будут возвращаться значения строк '\%' и '_', а не '%' и '_'.

Существует несколько способов включить кавычки в строку:

- Одиночная кавычка (апостроф) ' внутри строки, заключенной в кавычки '", может быть записана как "".
- Двойная кавычка "" внутри строки, заключенной в двойные кавычки "", может быть записана как """".
- Можно предварить символ кавычки символом экранирования ('\').
- Для символа " внутри строки, заключенной в двойные кавычки, не требуется специальной обработки; его также не требуется дублировать или предварять обратным слешем. Точно так же не требует специальной обработки двойная кавычка "" внутри строки, заключенной в одинарные кавычки '".

Ниже показаны возможные варианты применения кавычек и escape-символа на примерах выполнения команды SELECT:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+
mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+
mysql> SELECT "This\nIs\nFour\nlines";
+--------------------+
| This
Is
Four
lines |
+--------------------+
```

Если необходимо вставить в строку двоичные данные (такие как BLOB), следующие символы должны быть представлены как escape-последовательности:

NUL	ASCII 0. Необходимо представлять в виде '\0' (обратный слеш и символ ASCII '0').

\	ASCII 92, обратный слеш. Представляется как '\\'.
'	ASCII 39, единичная кавычка. Представляется как '\''.
"	ASCII 34, двойная кавычка. Представляется как '\"'.

Также нужно отдельно упомянуть, для чего используется символ `.

Можно встретить следующий запрос:

```
CREATE TABLE `pracownik`
(
  `id` TINYINT auto_increment,
  `name` VARCHAR,
) ENGINE=InnoDB;
```

В данной ситуации, если не использовать символ обратной кавычки, нельзя будет брать ключевые слова mysql в качестве имен (например date, min, max, order и др.).

Практическая работа

1. Создаем и наполняем базу данных «Сотрудники».
2. Разбираемся с тем, как устроен DUMP и команды на создание таблиц.
3. Практикуем запросы – выбираем данные:
    - добавляем сотрудника в отдел;

```
INSERT INTO `employees`
VALUES
(10001,'1953-09-02','Georgi','Facello','M','1986-06-26'),
(10002,'1964-06-02','Bezalel','Simmel','F','1985-11-21');
```

    - удаляем сотрудника из отдела;

```
DELETE FROM `employees`
WHERE id = 10001;
```

- сколько денег было выплачено за любой год;

```
SELECT * FROM `salaries` WHERE `from_date` = '1986-06-26'
```

- находим менеджеров каждого отдела (SELECT, JOIN);

```
SELECT d.*, e.first_name, e.last_name
FROM departments d
  LEFT JOIN dept_manager m
    ON d.dept_no = m.dept_no
  LEFT JOIN employees e
    ON m.emp_no = e.emp_no
```

- смотрим максимальную зарплату менеджеров;

```
SELECT e.first_name, e.last_name, MAX(s.salary) max_salary
FROM dept_manager m
  LEFT JOIN employees e ON m.emp_no = e.emp_no
  LEFT JOIN salaries s on m.emp_no = s.emp_no
  GROUP BY e.first_name, e.last_name
  ORDER BY max_salary DESC;
```

- на усмотрение преподавателя можно сделать еще несколько выборок в любом интересном разрезе.

Практическое задание

База данных «Страны и города мира»:

1. Сделать запрос, в котором мы выберем все данные о городе – регион, страна.
2. Выбрать все города из Московской области.

База данных «Сотрудники»:

1. Выбрать среднюю зарплату по отделам.
2. Выбрать максимальную зарплату у сотрудника.
3. Удалить одного сотрудника, у которого максимальная зарплата.
4. Посчитать количество сотрудников во всех отделах.
5. Найти количество сотрудников в отделах и посмотреть, сколько всего денег получает отдел.

Дополнительные материалы

1. https://www.w3schools.com/sql/

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. http://dev.mysql.com/doc/refman/5.7/en/insert.html
2. http://dev.mysql.com/doc/refman/5.7/en/select.html
3. http://dev.mysql.com/doc/refman/5.7/en/expressions.html
4. http://dev.mysql.com/doc/refman/5.7/en/pattern-matching.html
5. http://www.exlab.net/tools/sheets/regexp.html
6. http://dev.mysql.com/doc/refman/5.7/en/union.html
7. https://ru.wikipedia.org/wiki/Union_(SQL)
8. http://dev.mysql.com/doc/refman/5.7/en/delete.html
9. http://dev.mysql.com/doc/refman/5.7/en/update.html
10. https://launchpad.net/test-db/
11. http://www.exlab.net/tools/sheets/regexp.html