Лекция №5
SELECT (продолжение), INSERT, UPDATE, DELETE, UNION , JOIN

План лекции:

*1. Оператор **SELECT** (продолжение):*
*– **GROUP BY** ;*
*– **ORDER BY** ;*
*– **LIMIT**.*
*2. Оператор **INSERT**.*
*3. Оператор **UPDATE**.*
*4. Оператор **DELETE.***
*5. Объединение выборок (**UNION**).*
*6. Объединение «таблиц» (**JOIN**).*

SELECT (продолжение)
Группировка записей запросов
GROUP BY (1)

1. Для группировки записей запроса используется инструкция GROUP BY совместно с агрегирующими функциями.

SELECT fk_client, sum(price_delivery)
FROM Order
GROUP BY fk_client;

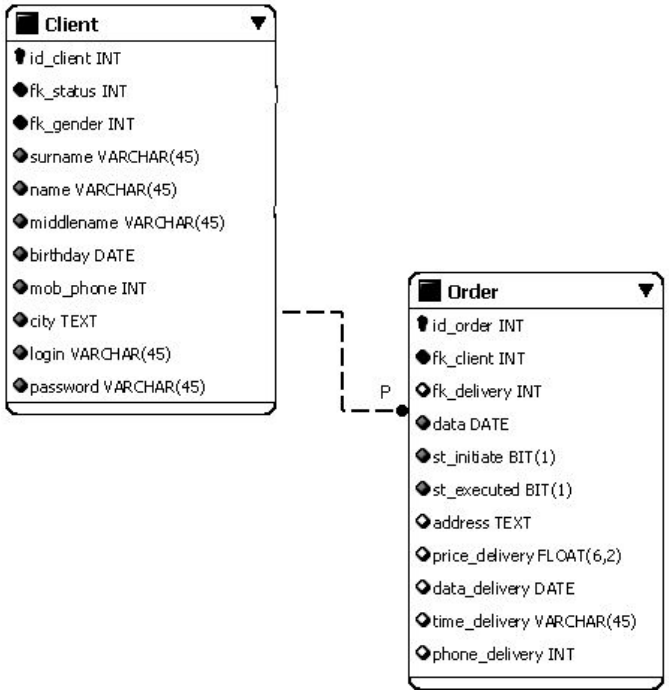
SELECT o.fk_client, c.surname, c.name, sum(price_delivery)
FROM Order o, Client c
WHERE o.fk_client=c.id_client
GROUP BY fk_client;

2. GROUP BY без агрегирующих функций работает аналогично DISTINCT

SELECT fk_client FROM Order
GROUP BY fk_client;

3. GROUP BY и NULL-значения. Все NULL-значения будут объединены в одну группу.

SELECT fk_client, sum(price_delivery)
FROM Order
GROUP BY fk_client;



←(все неизвестные клиенты (NULL) в заказе объединятся в одну группу и рассчитается суммарная стоимость доставки для всех NULL-клиентов).

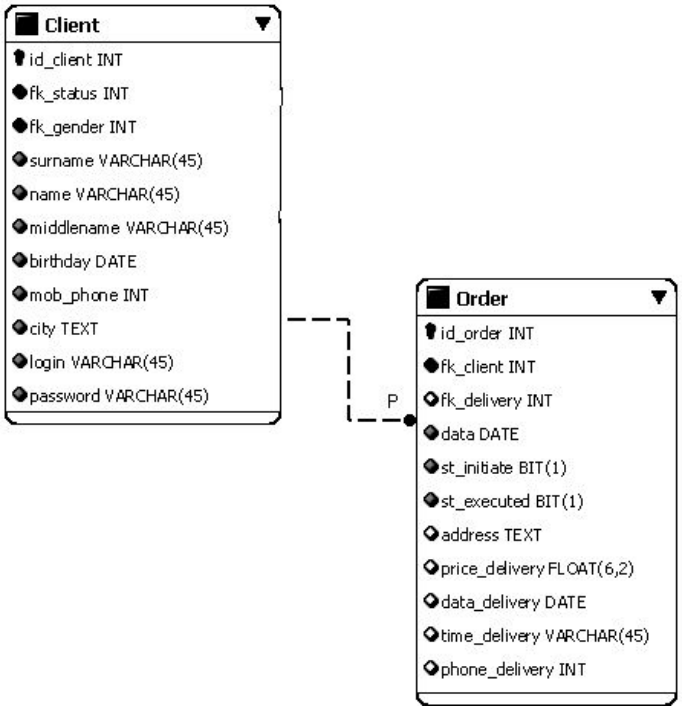
SELECT (продолжение) Группировка записей запросов GROUP BY (2)

4. GROUP BY и WHERE. При этом сначала производится выборка из таблицы с применением условия WHERE и лишь затем группировка результата GROUP BY. В предложении WHERE нельзя использовать агрегирующие функции.

5. GROUP BY и HAVING. При этом сначала происходит группировка таблицы и лишь затем выборка с применением условия HAVING. Допускается использование условия HAVING без предложения группировки GROUP BY.

6. Группировка внутри группировки.

SELECT fk_client, st_executed, sum(price_delivery)
FROM Order
GROUP BY fk_client, st_executed;



SELECT (продолжение)
Сортировка результатов запроса
ORDER BY (1)

1. Сортировка по возрастанию ASC (по умолчанию) или по убыванию DESC.

```
SELECT o.fk_client, c.surname, c.name,
sum(price_delivery)
FROM Order o, Client c WHERE o.fk_client=c.id_client
GROUP BY fk_client
ORDER BY c.surname;
```

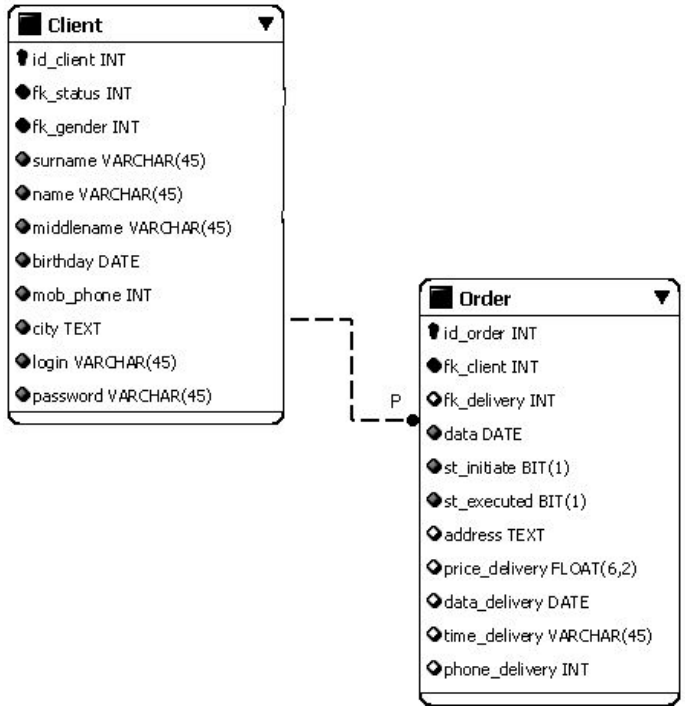
2. Сортировка по выражению (заголовку)

```
SELECT o.fk_client, c.surname AS `FAM`, c.name,
sum(price_delivery)
FROM Order o, Client c WHERE o.fk_client=c.id_client
GROUP BY fk_client
ORDER BY ` FAM` ASC;
```

3. Сортировка по позиции

```
SELECT surname, name
FROM Client
ORDER BY 1 DESC;

SELECT *
FROM Client
ORDER BY 1 DESC;
```



SELECT (продолжение)
Ограничение выборки LIMIT

Для управления количеством записей в результирующей таблице используется оператор LIMIT. Этот оператор записывается в самом конце запроса и имеет следующую конструкцию:

SELECT ... ... LIMIT [start,] amount

start - это номер строки в результирующей таблицы (от 0), от которой необходимо отсчитывать записи;

count - это число, которое означает то, сколько записей из результирующей таблицы необходимо отобрать, начиная от start.

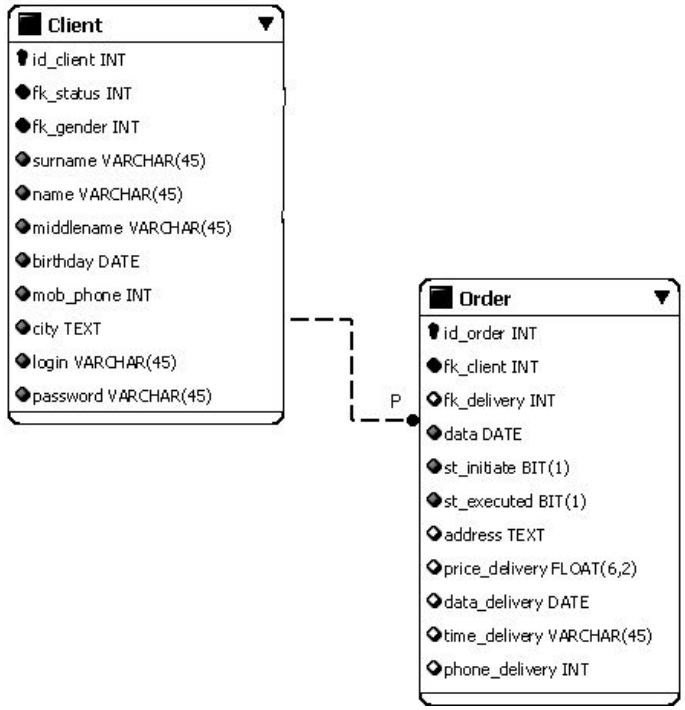
*SELECT * FROM Client LIMIT 1;*

*или SELECT * FROM Client LIMIT 0, 1;*

(получаем только 1-ю запись – соответствует 0-й в MySQL)

*SELECT * FROM Client LIMIT 1, 4;*

(получаем 4 записи, начиная со 2-й)



Вставка записей INSERT

Синтаксис оператора INSERT	Описание
INSERT [LOW_PRIORITY | DELAYED | [IGNORE]] [INTO] tbl_name [(col_name [,col_name] ...)] {VALUES | VALUE} [(value [,value] ...)] ... [ON DUPLICATE KEY UPDATE col_name=value [,col_name=value]] *#или* INSERT [LOW_PRIORITY | DELAYED | [IGNORE]] [INTO] tbl_name SET col_name = value [,col_name = value] ... [ON DUPLICATE KEY UPDATE ...] *#или* INSERT [LOW_PRIORITY | DELAYED | [IGNORE]] [INTO] tbl_name [(col_name [, col_name] ...)] SELECT select_expr ... WHERE where_condition [ON DUPLICATE KEY UPDATE ...]	1. LOW_PRIORITY – выполнение INSERT будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. Не рекомендуется для таблиц MyISAM, поскольку отключает параллельные вставки. 2. DELAYED – игнорируется сервером MySQL. 3. IGNORE – выполнение INSERT не будет прервано, даже если возникнет ошибка дублирования ключей. Записи, для которых возникают конфликтные ситуации, обновлены не будут. 4. Порядок и количество полей col_name и присваиваемых значений value из списка VALUE должны совпадать. 5. Имена и количество полей col_name и полей из списка select_expr оператора SELECT должны совпадать. 6. Поля имеющие значения по умолчанию (DEFAULT) можно не указывать – значения установятся автоматически. Можно указать значение DEFAULT в явном виде или использовать функцию DEFAULT(col_name). 7. Для первичных ключей с атрибутом AUTO_INCREMENT в списке VALUE нужно указать значение NULL или использовать функцию LAST_INSERT_ID(). 8. Для вставки нескольких записей необходимо в VALUE указать несколько списков (), (), ... через запятую.

Примеры использования INSERT

1. Конструкция INSERT...VALUES

INSERT INTO client VALUES (NULL, 1, 1, 'Petrov', 'Ivan', 'Ivanovich', '2018-10-01', 11111111, 'Kiev', 'abcd', 'ivan123456');

2. Конструкция INSERT...SET используются для вставки записи на основе явно заданных значений.

INSERT INTO client SET

id_client =NULL, fk_status=1, fk_gender=1,

surname ='Petrov', name='Ivan', middlename='Ivanovich', birthday='2018-10-01',

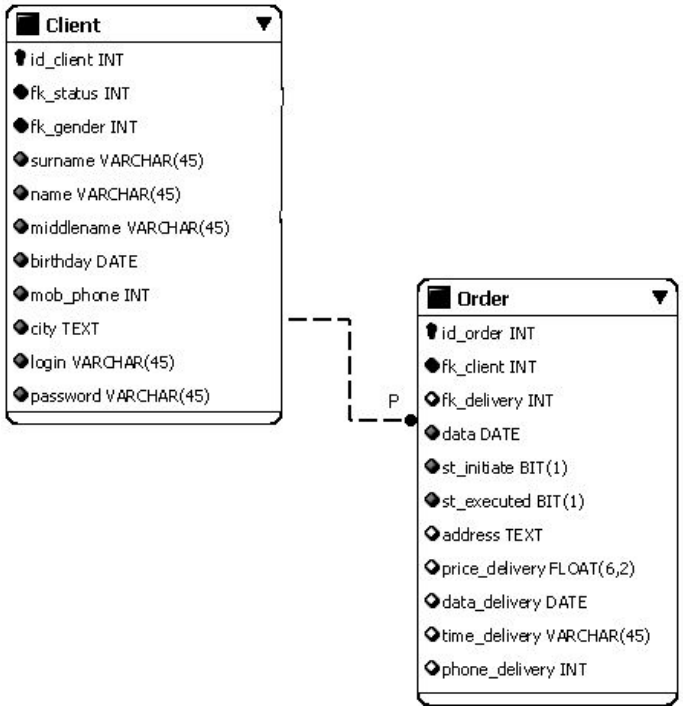
mob_phone=11111111, city='Kiev',

login='abcd', password='ivan123456');

3. Конструкция INSERT...SELECT используется для вставки записей, выбранных из другой таблицы или таблиц.

INSERT INTO client_man SELEST * FROM client WHERE fk_gender=1;

(при этом таблица Client_man должна быть создана заранее с помощью оператора CREATE TABLE...) *(в частном случае она должна полностью совпадать с таблицей Client)*



Альтернатива INSERT

OAD DATA [LOW_PRIORITY | CONCURRENT]
[LOCAL] INFILE 'file_name.txt'
[REPLACE | IGNORE]
INTO TABLE tbl_name [FIELDS [TERMINATED BY '\t']
[[OPTIONALLY] ENCLOSED BY ''] [ESCAPED BY '\\']]
[LINES TERMINATED BY '\n']
[IGNORE number LINES] [(col_name,...)]

Команда LOAD DATA INFILE читает строки из текстового файла и вставляет их в таблицу с очень высокой скоростью.

LOAD DATA INFILE "data.txt" INTO TABLE db2.my_table;

LOAD DATA INFILE "/tmp/file_name" INTO TABLE test IGNORE 1 LINES;

LOAD DATA INFILE 'data.txt' INTO TABLE table2 FIELDS TERMINATED BY ',';

LOW_PRIORITY - выполнение команды LOAD DATA будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы.

CONCURRENT - при работе с таблицами MyISAM, то другие потоки могут извлекать данные из таблицы во время выполнения команды LOAD DATA.

При применении опции *LOCAL* выполнение может происходить несколько медленнее в сравнении с предоставлением серверу доступа к файлам напрямую, поскольку содержимое файла должно переместиться с клиентского хоста на сервер. Ключевые слова *REPLACE* и *IGNORE* управляют обработкой входных записей, которые дублируют существующие записи с теми же величинами уникальных ключей. Если указать *REPLACE*, то новые строки заменят существующие с таким же уникальным ключом. Если указать *IGNORE*, то входные строки, имеющие тот же уникальный ключ, что и существующие, будут пропускаться. Если не указан ни один из параметров, то при обнаружении дублирующегося значения ключа возникает ошибка и оставшаяся часть текстового файла игнорируется.

Обновление записей UPDATE

Синтаксис оператора UPDATE	Описание
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name SET col_name=value [,col_name2=value ...] [WHERE where_condition] [ORDER BY ...] [LIMIT row_count] *#Синтаксис обновления записей в* *#нескольких таблицах* UPDATE [LOW_PRIORITY] [IGNORE] tbl_name [, tbl_name ...] SET col_name1=expr1 [, col_name2=expr2 ...] [WHERE where_condition]	1. LOW_PRIORITY – выполнение UPDATE будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. Не рекомендуется для таблиц MyISAM, поскольку отключает параллельные замены. 2. IGNORE – выполнение UPDATE не будет прервано, даже если возникнет ошибка дублирования ключей. Записи, для которых возникают конфликтные ситуации, обновлены не будут. 3. Поля, имеющие значения по умолчанию (DEFAULT) можно не указывать – значения установятся автоматически. 4. При обновлении записей в нескольких таблицах запрещено использовать ORDER BY и LIMIT.

Удаление записей DELETE

Синтаксис оператора DELETE	Описание

`DELETE [LOW_PRIORITY] [QUICK]`
`[IGNORE] FROM tbl_name`
`[WHERE where_condition]`
`[ORDER BY ...]`
`[LIMIT row_count]`
#Синтаксис удаления записей в
#нескольких таблицах
`DELETE [LOW_PRIORITY] [QUICK]`
`[IGNORE] tbl_name[.*]`
`[, tbl_name[.*]] ...`
`FROM tbl_name`
`[WHERE where_condition]`
#или
`DELETE [LOW_PRIORITY] [QUICK]`
`[IGNORE]`
`FROM tbl_name[.*]`
`[, tbl_name[.*]] ...`
`USING table_references`
	1. `LOW_PRIORITY` – выполнение `DELETE` будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. Не рекомендуется использовать с таблицами `MyISAM`.
	2. `IGNORE` – выполнение `UPDATE` не будет прервано, даже если возникнет ошибка. Записи, для которых возникают конфликтные ситуации, удалены не будут.
	3. Если задан `QUICK`, то обработчик таблицы при выполнении удаления не будет объединять индексы. Не рекомендуется для InnoDB. Для MyISAM рекомендуется после удаления выполнить команду `OPTIMIZE TABLE`.
	3.Выражение `where_condition=TRUE` для каждой удаляемой строки.
	4. Для таблиц MyISAM и InnoDB поля с атрибутом AUTO_INCREMENT, после удаления записи с его максимальным значением, не восстанавливаются.
	5. При удалении записей в нескольких таблицах

Объединение выборок UNION

Синтаксис оператора UNION	Описание
SELECT ... select_expr UNION [ALL | DISTINCT] SELECT ... select_expr [UNION [ALL | DISTINCT] SELECT ...] SELECT ... [LIMIT] [ORDER BY] Если для всего результата UNION необходимо применить оператор ORDER BY, следует заключить предложения SELECT в круглые скобки: (SELECT ... select_expr [ORDER BY] [LIMIT]) UNION [ALL | DISTINCT] (SELECT ... select_expr... [ORDER BY] [LIMIT]]) [ORDER BY] [LIMIT]	1. Поля, перечисленные в выражениях запроса select_expr должны быть одинакового типа. 2. Имена полей, указанные в select_expr первого SELECT будут использованы как имена полей для всего результата UNION. 3. Для всех SELECT-запросов UNION: DISTINCT – объединяет только уникальные записи; ALL – объединяет все записи. 4. Если ALL и DISTINCT не используются, по умолчанию подразумевается DISTINCT для всего результата UNION. 5. Во всех SELECT-запросах UNION могут использоваться операторы LIMIT и ORDER BY. 6. Использование ORDER BY во втором и последующих SELECT-запросах с применение имен полей в предложениях select_expr приводит к ошибке. Это обусловлено тем, что результат UNION использует имена полей из select_expr первого SELECT. В этом случае рекомендуется использовать псевдонимы для таблиц и полей, совпадающих с именами первого SELECT.

Объединение таблиц WHERE, JOIN

Источник запроса – перечень таблиц, разделенных запятой, в выражении FROM, представляет собой декартово произведение (полное или перекрестное объединение), которое возвращает полный набор комбинаций записей (кортежей).

Условия WHERE, принимающие значения TRUE или FALSE, в теории множеств называют предикатами. Использование предикатов превращает источник запроса (FROM) в объединение по эквивалентности, ограничивающее число возвращаемых записей. Инструкция JOIN является альтернативой WHERE и также позволяет задать условия объединения.

Для связывания нескольких таблиц используется объединение по равенству (equi-join), а имена таблиц указываются в предложении FROM, где запятая (,) выступает операндом объединения. Для объединения таблиц с помощью инструкции WHERE требуется задать условия объединения. Для этого обычно используют условие на равенство уникальных ключей связанных таблиц (чаще всего используются первичные и внешние ключи):

WHERE pk_tbl_A = fk_tbl_B.

При связывании нескольких таблиц, используются однотипные равенства в предложении WHERE, объединенные логическим оператором AND.

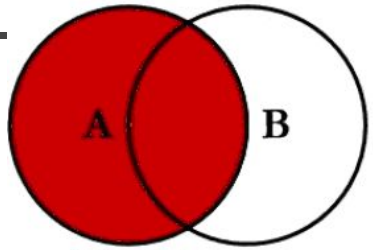
Для объединения таблиц также используется инструкция JOIN. Она эквивалентна оператору объединения «запятая» (,) в инструкции FROM. Условие соединения (join_condition) задается так же, как и с использованием WHERE.

Объединение таблиц WHERE, JOIN (продолжение)

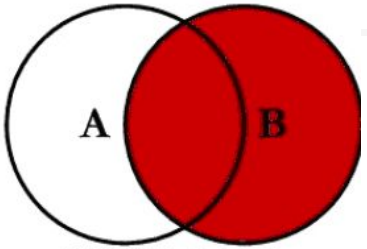
Синтаксис инструкции JOIN	Описание

FROM tbl_name
[STRAIGHT_JOIN]
  | [INNER | CROSS]
  | [NATURAL [{LEFT|RIGHT}]]
JOIN
tbl_name [[AS] alias]] ON
join_condition
...
 	
#если поля в связываемых таблицах
#совпадают, вместо:
tbl_name [[AS] alias]] ON
join_condition
#используется:
tbl_name [[AS] alias]]
USING (col_name, col_name,
...)
	1. Предложения JOIN, INNER JOIN и CROSS JOIN являются синонимами.
	2. STRAIGHT_JOIN – задает объединение таблиц в том порядке, в каком они перечислены в предложении FROM.
	3. NATURAL [LEFT] JOIN определяется как семантически эквивалентное INNER JOIN или LEFT JOIN со списком USING, который содержит все одинаковые поля в таблицах.
	4. JOIN – при объединении двух таблиц будут выбраны поля, значения которых совпадают в обеих таблицах.
	5. LEFT JOIN – будут выбраны поля, значения которых совпадают в обеих таблицах, а также поля левой таблицы, значения в которых не совпали со значениями в правой таблице.
	6. RIGHT JOIN – аналогично п.5 с противоположной связью таблиц.

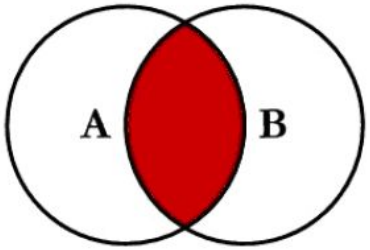
Визуальное представление JOIN (общее)



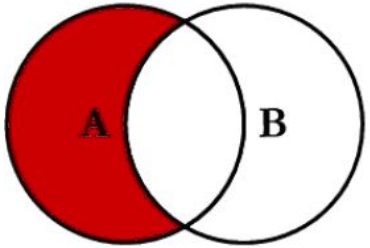

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



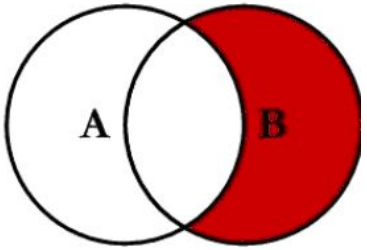

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



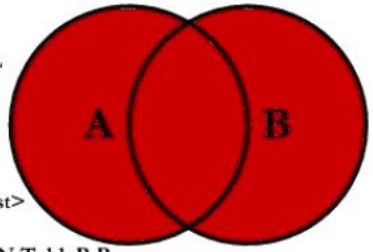

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



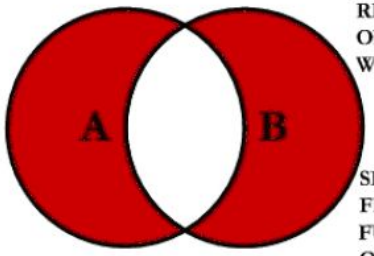

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

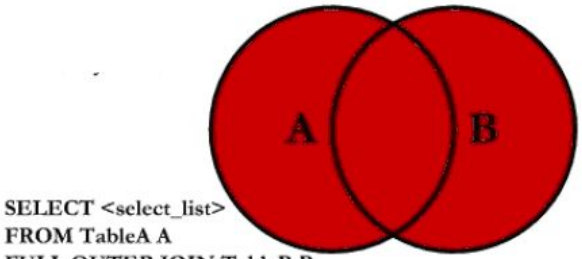



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

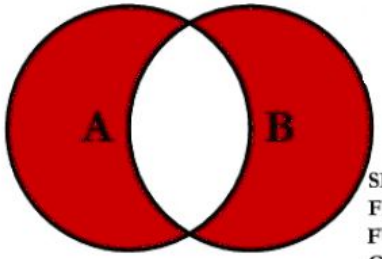



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Альтернатива FULL JOIN




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

Полное внешнее соединение (FULL JOIN) не поддерживается в MySQL. Это «избыточная» операция, т.к. она представляется через объединение левого и правого внешних соединений.

```
SELECT <select_list> FROM TableA A
LEFT JOIN TableB B
    ON A.Key = B.Key
UNION
SELECT <select_list> FROM TableA A
RIGHT  JOIN TableB B
    ON A.Key = B.Key;
```