MySQL

Выборка из нескольких таблиц: соединения и подзапросы

Введение

В базе данных могут находиться десятки или даже сотни таблиц. Кроме того, правила нормализации говорят нам о том, что в разных таблицах могут храниться связанные между собой данные, которые было бы неплохо получить в рамках одного запроса. Конечно, при помощи языков программирования можно сначала сделать выборку из одной таблицы, потом сделать выборку значений из других таблиц и в конце концов соединить данные вместе, используя общие признаки. Однако делать много запросов для такий целей - крайне неэффективная стратегия. Поэтому в базах данных есть способы получить данные из нескольких таблиц при помощи единственного запроса.

Всего мы рассмотрим четыре варианта соединения таблиц - внутреннее соединение (**INNER JOIN**), левое соединение (**LEFT OUTER JOIN**), правое соединение (**RIGHT OUTER JOIN**), перекрестное соединение (**CROSS JOIN**), а также попробуем воспроизвести полное соединение (MySQL сама по себе не разрешает делать полное соединение таблиц, но этого все равно можно добиться с помощью дополнительных манипуляций, например при помощи оператора **UNION**).

Подготовка для выборки из нескольких таблиц (создание таблиц)

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

CREATE TABLE passports (
      employee_id INT UNSIGNED NULL UNIQUE,
      code VARCHAR(255) NOT NULL,
      nationality VARCHAR(255) NOT NULL,
      age INT UNSIGNED,
      sex VARCHAR(255) NOT NULL,
      CONSTRAINT fk_passport_employee_id FOREIGN KEY(employee_id) REFERENCES employees(id) ON DELETE CASCADE
);

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

CREATE TABLE projects (
      id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
      name VARCHAR(255) NOT NULL,
     employee_id INT UNSIGNED NOT NULL,
     client_id INT UNSIGNED NOT NULL,
     CONSTRAINT fk_project_employee_id FOREIGN KEY(employee_id) REFERENCES employees(id) ON DELETE CASCADE,
     CONSTRAINT fk_project_client_id FOREIGN KEY(client_id) REFERENCES clients(id) ON DELETE CASCADE
);
```

Подготовка для выборки из нескольких таблиц (заполнение таблиц)

```
INSERT INTO employees(id, name, position)
VALUES
(1, 'Barbara Schmidt', 'System Analytic'),
(2, 'Joe D'Amato', 'Android Programmer'),
(3, 'John Smith', 'Web Programmer');

INSERT INTO passports(employee_id, code, nationality, age, sex)
VALUES
(1, 's8f8798sdf', 'German', 35, 'woman'),
(2, '98dfg87dg8', 'Italian', 22, 'man'),
(NULL, '8s89sf889', 'Spanish', 18, 'man');

INSERT INTO clients(id, name)
VALUES
(1, 'Google'),
(2, 'Apple'),
(3, 'Microsoft');

INSERT INTO projects(name, employee_id, client_id)
VALUES
('CMS Development', 1, 2),
('Mobile App Development', 2, 1),
('Site Development', 1, 1);
```

Данные в таблицах

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

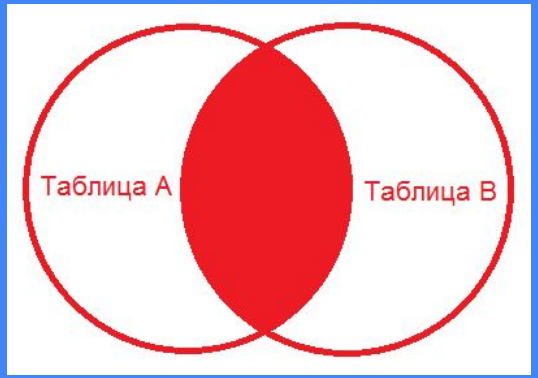
Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

Внутреннее соединение (INNER JOIN)



Введение

При использовании внутреннего соединения таблиц мы хотим получить данные из двух или более таблиц только для тех строк, которые имеют совпадающие данные как в первой, так и во второй таблице (третьей, четвертой и т.д.). Если у некой строки (не важно в какой из таблиц) такие совпадающие данные отсутствуют, то эта строка будет проигнорирована.

Для реализации внутреннего соединения в запросе после написания названия первой таблицы нам необходимо применить конструкцию **INNER JOIN** или просто **JOIN** (в MySQL **INNER** можно не писать), потом указать название следующей таблицы, а уже после названия поставить ключевое слово **ON** и вписать некие условия, которые должны выполняться для обеих соединяемых таблиц.

Простейший пример внутреннего соединения двух таблиц

Имена сотрудников хранятся в таблице **employees**, а национальности - в таблице **passports**. Общим условием будет равенство значения **id** в **employees** и **employee_id** в **passports**. Если эти значения равны, данные из обеих таблиц попадут в результирующую таблицу:

```
SELECT name, nationality FROM employees INNER JOIN passports ON id = employee_id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

РЕЗУЛЬТАТ

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian

Использование псевдонимов таблиц при соединении

```
-- 1 вариант (явное указание таблицы для столбца)
SELECT employees.name AS employee_name, projects.name AS project_name
FROM employees INNER JOIN projects ON employees.id = projects.employee_id;

-- 2 вариант (присвоение псевдонима для таблицы)
SELECT e.name AS employee_name, p.name AS project_name
FROM employees AS e INNER JOIN projects AS p ON e.id = p.employee_id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

РЕЗУЛЬТАТ

employee_name	project_name
Barbara Schmidt	CMS Development
Barbara Schmidt	Site Development
Joe D'Amato	Mobile App Development

Внутреннее соединение трех таблиц

```
SELECT e.name AS employee_name, p.name AS project_name, c.name AS client_name
FROM employees AS e
INNER JOIN projects AS p ON e.id = p.employee_id
INNER JOIN clients AS c ON p.client_id = c.id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

РЕЗУЛЬТАТ

employee_name	project_name	client_name
Joe D'Amato	Mobile App Development	Google
Barbara Schmidt	Site Development	Google
Barbara Schmidt	CMS Development	Apple

Несколько условий при соединении таблиц

При соединении таблиц важно понимать ещё один аспект - в качестве условия может выступать любое количество столбцов и их сравнений, которые мы можем перечислить с помощью ключевых слов **AND** или **OR**. Например, попробуем узнать названия тех проектов, которые заказал Google:

```
SELECT p.name FROM projects p
INNER JOIN clients c ON p.client_id = c.id AND c.name = 'Google';
```

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

РЕЗУЛЬТАТ

name
Mobile App Development
Site Development

Применение фильтрации и сортировки при соединении таблиц

При соединениях таблиц можно применять стандартные условия (**WHERE**) и сортировки (**ORDER BY**). В рамках этих конструкций можно использовать любые столбцы (как те, которые перечислены после **SELECT**, так и те, которые там не упомянуты) любых таблиц, но важно не забывать про псевдонимы этих таблиц. Например, получим работников и их возраст, но установим, что возраст должен быть меньше 30 лет. Кроме того, отсортируем результат по возрасту в возрастающем порядке:

```
SELECT e.name, p.age FROM passports p
INNER JOIN employees e ON p.employee_id = e.id
WHERE p.age < 30 ORDER BY p.age ASC;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

РЕЗУЛЬТАТ

name	age
Joe D'Amato	22

Применение звездочки при соединении таблиц

В числе прочего оператор звёздочка * может использоваться следующим образом - если мы используем псевдоним таблицы, то при при перечислении выбираемых столбцов можно указать звездочку после этого псевдонима и точки - тогда мы получим значения всех столбцов только той таблицы, чей псевдоним мы использовали:

```
SELECT e.* FROM passports p
INNER JOIN employees e ON p.employee_id = e.id
WHERE p.age < 30 ORDER BY p.age ASC;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

РЕЗУЛЬТАТ

id	name	position
2	Joe D'Amato	Android Programmer

Соединение нескольких таблиц без оператора JOIN

```
SELECT e.name AS employee_name, p.name AS project_name, c.name AS client_name
FROM employees AS e, projects AS p, clients AS c
WHERE e.id = p.employee_id AND p.client_id = c.id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

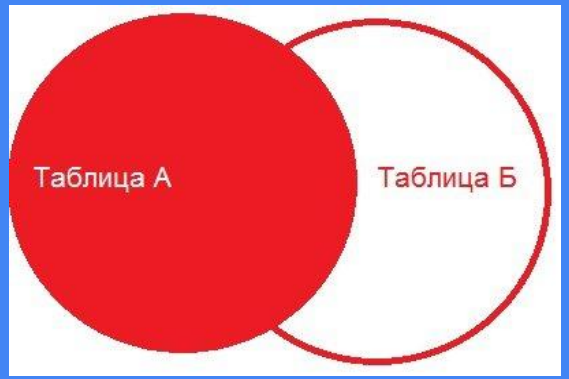
Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

РЕЗУЛЬТАТ

employee_name	project_name	client_name
Joe D'Amato	Mobile App Development	Google
Barbara Schmidt	Site Development	Google
Barbara Schmidt	CMS Development	Apple

Левое внешнее соединение (LEFT OUTER JOIN)



Введение

Левое внешнее соединение синтаксически мало чем отличается от внутреннего соединения (само собой, вместо **INNER JOIN** следует писать **LEFT OUTER JOIN**, кроме того, **OUTER** можно игнорировать - достаточно только **LEFT JOIN**), но практически различие очевидно. Данное соединение гарантирует, что строки таблицы (или таблиц), которая была запрошена вначале (условно говоря - слева), будут гарантированно включены в результирующую выборку (хотя бы один раз) - даже в том случае, если значения совпадающих столбцов отсутствуют. Вместо отсутствующих значений столбцов будут возвращены значения **NULL**.

Пример левого внешнего соединения двух таблиц (employees - passports)

```
SELECT e.name, p.nationality FROM employees e
LEFT OUTER JOIN passports p ON e.id = p.employee_id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

РЕЗУЛЬТАТ

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian
John Smith	NULL

Пример левого внешнего соединения двух таблиц (passports - employees)

```
SELECT name, nationality FROM passports p
LEFT OUTER JOIN employees e ON p.employee_id = e.id;
```

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

РЕЗУЛЬТАТ

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian
NULL	Spanish

Левое внешнее соединение трех таблиц

```
SELECT e.name AS employee_name, p.name AS project_name, c.name AS client_name
FROM employees AS e
LEFT JOIN projects AS p ON e.id = p.employee_id
LEFT JOIN clients AS c ON p.client_id = c.id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

РЕЗУЛЬТАТ

employee_name	project_name	client_name
Barbara Schmidt	CMS Development	Apple
Barbara Schmidt	Site Development	Google
Joe D'Amato	Mobile App Development	Google
John Smith	NULL	NULL

Использование левого внешнего и внутреннего соединения в одном запросе

```
SELECT e.name AS employee_name, p.name AS project_name, c.name AS client_name
FROM employees AS e
LEFT JOIN projects AS p ON e.id = p.employee_id
INNER JOIN clients AS c ON p.client_id = c.id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

РЕЗУЛЬТАТ

employee_name	project_name	client_name
Joe D'Amato	Mobile App Development	Google
Barbara Schmidt	Site Development	Google
Barbara Schmidt	CMS Development	Apple

Правое внешнее соединение (RIGHT OUTER JOIN)

Введение

Как можно понять из названия, правое внешнее соединение является абсолютным антиподом левого внешнего соединения. При соединении таблиц нам гарантируется, что все строки таблицы, которая была запрошена позже (условно “справа”), будут нам возвращены в любом случае (хотя бы один раз) - даже если для них не выполняется условие соединения. Синтаксис очень похож на **INNER JOIN** и **LEFT OUTER JOIN** - сохраняются все предыдущие особенности, но главная конструкция называется **RIGHT OUTER JOIN** (как и ранее, слово **OUTER** можно не писать - в MySQL предоставляется возможность писать только **RIGHT JOIN**).

Пример правого внешнего соединения двух таблиц (employees - passports)

```
SELECT e.name, p.nationality FROM employees e
RIGHT OUTER JOIN passports p ON e.id = p.employee_id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

РЕЗУЛЬТАТ

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian
NULL	Spanish

Пример правого внешнего соединения двух таблиц (passports - employees)

```
SELECT e.name, p.nationality
FROM passports p
RIGHT OUTER JOIN employees e ON p.employee_id = e.id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

РЕЗУЛЬТАТ

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian
John Smith	NULL

Использование правого внешнего и внутреннего соединения в одном запросе

```
SELECT e.name AS employee_name, p.name AS project_name, c.name AS client_name
FROM employees AS e
INNER JOIN projects AS p ON e.id = p.employee_id
RIGHT JOIN clients AS c ON p.client_id = c.id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

РЕЗУЛЬТАТ

employee_name	project_name	client_name
Barbara Schmidt	CMS Development	Apple
Barbara Schmidt	Site Development	Google
Joe D'Amato	Mobile App Development	Google
NULL	NULL	Microsoft

Использование правого и левого внешнего соединений в одном запросе

```
SELECT e.name AS employee_name, p.name AS project_name, c.name AS client_name
FROM employees AS e
LEFT JOIN projects AS p ON e.id = p.employee_id
RIGHT JOIN clients AS c ON p.client_id = c.id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **projects**:

id	name	employee_id	client_id
1	CMS Development	1	2
2	Mobile App development	2	1
3	Site Development	1	1

Таблица **clients**:

id	name
1	Google
2	Apple
3	Microsoft

РЕЗУЛЬТАТ

employee_name	project_name	client_name
Barbara Schmidt	CMS Development	Apple
Barbara Schmidt	Site Development	Google
Joe D'Amato	Mobile App Development	Google
NULL	NULL	Microsoft

Перекрестное соединение (CROSS JOIN)

Введение

Далее рассмотрим крайне любопытный вид соединения - перекрестное соединение, которое также известно как “декартово произведение”. Суть его состоит в том, чтобы каждой строке из первой соединяемой таблицы была соотнесена каждая строка из второй соединяемой таблицы - таким образом мы можем получить все комбинации этих строк. Данное соединение крайне редко используется в настоящих проектах - чаще всего просто для генерации большого количества тестовых данных.

Синтаксис перекрестного соединения представляет собой конструкцию **CROSS JOIN**, которой можно не прописывать какие либо условия.

Пример перекрестного соединения двух таблиц (employees - passports)

```
SELECT e.name, p.nationality FROM employees e CROSS JOIN passports p;
```

name	nationality
John Smith	German
Joe D'Amato	German
Barbara Schmidt	German
John Smith	Italian
Joe D'Amato	Italian
Barbara Schmidt	Italian
John Smith	Spanish
Joe D'Amato	Spanish
Barbara Schmidt	Spanish

Пример перекрестного соединения двух таблиц с условием

CROSS JOIN не запрещает использование условий соединения, просто после внедрения таких условий **CROSS JOIN** ведет себя как **INNER JOIN**:

```
SELECT e.name, p.nationality FROM employees e
CROSS JOIN passports p ON e.id = p.employee_id;
```

Таблица **employees**:

id	name	position
1	Barbara Schmidt	System Analytic
2	Joe D'Amato	Android Programmer
3	John Smith	Web Programmer

Таблица **passports**:

employee_id	code	nationality	age	sex
1	s8f8798sdf	German	35	woman
2	98dfg87dg8	Italian	22	man
NULL	8s89sf889	Spanish	18	man

РЕЗУЛЬТАТ

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian

Пример перекрестного соединения без оператора JOIN

Декартово произведение можно осуществить и без конструкции **CROSS JOIN** - всего навсего можно перечислить таблицы через запятую без всяких условий:

```
SELECT e.name, p.nationality
FROM employees e, passports p;
```

name	nationality
John Smith	German
Joe D'Amato	German
Barbara Schmidt	German
John Smith	Italian
Joe D'Amato	Italian
Barbara Schmidt	Italian
John Smith	Spanish
Joe D'Amato	Spanish
Barbara Schmidt	Spanish

Полное внешнее соединение

Введение

Иногда нам может потребоваться запрос, в рамках которого мы должны гарантированно получить все строки как из “левой” таблицы, так и из “правой” таблицы (хотя бы один раз). MySQL не имеет конструкций конкретно для этой задачи, но предоставляет оператор **UNION**, который позволяет объединить результаты двух подзапросов. Таким образом, мы можем сначала сделать подзапрос с левым соединением, потом подзапрос с правым соединением, а уже после этого объединить результаты в одну конечную таблицу.

Применение оператора **UNION** накладываем на нас два ограничения. Во-первых, результаты подзапросов должны иметь одинаковое количество и одинаковый порядок выбранных столбцов. Во-вторых, типы выбираемых столбцов должны быть совместимыми (если в первом подзапросе первый столбец имеет тип **INT**, то и во втором подзапросе первый столбец должен иметь тип **INT**).

Кроме ограничений, оператор **UNION** имеет одну интересную особенность. Если при объединении таблиц в результирующей выборке будут полностью идентичные строки, то **UNION** не будет их показывать все, а покажет только уникальные варианты. Если нам по какой-то причине нужны дублирующиеся строки, то вместо **UNION** следует использовать **UNION ALL**.

Пример полного внешнего соединения (UNION) двух таблиц (employees - passports)

```
(
    SELECT e.name, p.nationality FROM employees e
    LEFT JOIN passports p ON e.id = p.employee_id
)
UNION
(
    SELECT e.name, p.nationality FROM employees e
    RIGHT JOIN passports p ON e.id = p.employee_id
);
```

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian
John Smith	NULL
NULL	Spanish

Пример полного внешнего соединения (UNION ALL) двух таблиц (employees - passports)

```
(
    SELECT e.name, p.nationality FROM employees e
    LEFT JOIN passports p ON e.id = p.employee_id
)
UNION ALL
(
    SELECT e.name, p.nationality FROM employees e
    RIGHT JOIN passports p ON e.id = p.employee_id
);
```

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian
John Smith	NULL
Barbara Schmidt	German
Joe D'Amato	Italian
NULL	Spanish

Применение INTERSECT при полном внешнем соединении

INTERSECT также объединяет таблицы, но оставляет только те строки, одинаковые комбинации которых присутствуют как в результате первого подзапроса, так и в результате всех последующих:

```
(
    SELECT e.name, p.nationality
    FROM employees e
    LEFT JOIN passports p ON e.id = p.employee_id
)
INTERSECT
(
    SELECT e.name, p.nationality
    FROM employees e
    RIGHT JOIN passports p ON e.id = p.employee_id
);
```

name	nationality
Barbara Schmidt	German
Joe D'Amato	Italian

Применение EXCERPT при полном внешнем соединении

Оператор **EXCEPT** делает следующее - при объединении таблиц он оставляет в конечной выборке только те комбинации строк, которое есть в результате первого подзапроса - если такая комбинация присутствует и в результате следующих подзапросов, то эта строка будет удалена из конечной таблицы:

```
(
   SELECT e.name, p.nationality
   FROM employees e
   LEFT JOIN passports p ON e.id = p.employee_id
)
EXCEPT
(
   SELECT e.name, p.nationality
   FROM employees e
   RIGHT JOIN passports p ON e.id = p.employee_id
);
```

name	nationality
John Smith	NULL

Подзапросы

Введение

Очень часто складывается ситуация, когда нам необходимо получить данные из какой-то одной таблицы, но условие для выборки строк должно полагаться на данные из другой таблицы. То есть получается, что нам не надо объединять таблицы - нам необходимо получить некий набор промежуточных значений из другой таблицы, чтобы опереться на них при выборке из первой таблицы. Такие запросы во вторую таблицу называются подзапросами.

Подготовка для выполнения подзапросов (создание таблиц)

```
CREATE TABLE customers (
    id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    debt DECIMAL(10,2) NOT NULL,
    city VARCHAR(255) NOT NULL
);

CREATE TABLE shops (
    id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    brand VARCHAR(255) NOT NULL,
    city VARCHAR(255) NOT NULL
);

CREATE TABLE orders (
    id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    customer_id INT UNSIGNED NOT NULL,
    shop_id INT UNSIGNED NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    CONSTRAINT fk_order_customer_id FOREIGN KEY(customer_id) REFERENCES customers(id) ON DELETE CASCADE,
    CONSTRAINT fk_order_shop_id FOREIGN KEY(shop_id) REFERENCES shops(id) ON DELETE CASCADE
);
```

Подготовка для выполнения подзапросов (заполнение таблиц)

```
INSERT INTO customers(id, name, debt, city)
VALUES
(1, 'John Smith', 120, 'Rīga'),
(2, 'Brandon Adams', 0, 'Liepaja'),
(3, 'Mary Sue', 0, 'Rēzekne'),
(4, 'Paul George', 300, 'Rīga'),
(5, 'Connor MacLeod', 0, 'Daugavpils');


INSERT INTO shops(id, brand, city)
VALUES
(1, 'Maxima', 'Rīga'),
(2, 'Lidl', 'Rīga'),
(3, 'Rimi', 'Daugavpils'),
(4, 'Top', 'Liepaja');

INSERT INTO orders(id, customer_id, shop_id, price)
VALUES
(1,  1, 1, 999.99),
(2,  1, 2, 505.69),
(3,  1, 2, 656.17),
(4,  3, 4, 15.22),
(5,  3, 2, 199.76),
(6,  3, 2, 1059.13),
(7,  4, 1, 236.87),
(8,  4, 1, 517.89),
(9,  5, 3, 1000.0),
(10, 5, 1, 700.11);
```

Данные в таблицах

Таблица **shops**:

id	brand	city
1	Maxima	Rīga
2	Lidl	Rīga
3	Rimi	Daugavpils
4	Top	Liepaja

Таблица **customers**:

id	name	debt	city
1	John Smith	120.00	Rīga
2	Brandon Adams	0.00	Liepaja
3	Mary Sue	0.00	Rēzekne
4	Paul George	300.00	Rīga
5	Connor MacLeod	0.00	Daugavpils

Таблица **orders**:

id	customer_id	shop_id	price
1	1	1	999.99
2	1	2	505.69
3	1	2	656.17
4	3	4	15.22
5	3	2	199.76
6	3	2	1059.13
7	4	1	236.87
8	4	1	517.89
9	5	3	1000.00
10	5	1	700.11

Подзапрос для получения одного значения для сравнения (=)

Представим, что нам надо разослать сообщения для тех людей, которые живут в том городе, в котором расположен некий магазин. Однако мы не знаем, в каком конкретно городе расположен магазин, но знаем его id (например, id = 2). Для этого мы должны получить город магазина (это и будет подзапрос), а потом использовать этот город для сравнения с городами посетителей.

```
SELECT id, name FROM customers WHERE city = (SELECT city FROM shops WHERE id = 2);
```

id	brand
1	John Smith
4	Paul George

Подзапрос для получения набора значений для сравнения (IN)

С помощью оператора **IN** можно проверить, находится ли запрашиваемое условием значение (в данном случае - идентификатор пользователя) в том наборе данных, которые вернул подзапрос (в данном случае - идентификаторы пользователей, которые совершали заказы).

```
SELECT id, name FROM customers WHERE id IN (SELECT customer_id FROM orders);
```

id	name
1	John Smith
3	Mary Sue
4	Paul George
5	Connor MacLeod

Использование оператора IN без подзапроса

Внутри оператора **IN** вовсе необязательно выполнять подзапрос - можно проставить необходимые нам значения вручную:

```
SELECT id, name FROM customers WHERE id IN (1, 2, 3);
```

id	name
1	John Smith
2	Brandon Adams
3	Mary Sue

Подзапрос для получения набора значений для сравнения (ANY)

Оператор **ANY** (имеет псевдоним **SOME**) схож с оператором **IN**, его также можно использовать для определения того, что некое значение находится в значениях, присланных подзапросом. Однако **ANY** также поддерживает не только прямое равенство (=), но и другие операции сравнения (>, >=, <, <=, <>, !=) - условие будет выполняться, если хотя бы какое-то значение будет подходить под выбранный подвид сравнения. Это позволяет, например, получить всех клиентов, которые имеют долг больше, чем цена какого-либо из их предыдущих заказов (клиентов, которые много себе позволяют):

```
SELECT c.id, c.name, c.debt FROM customers c
WHERE c.debt > ANY(SELECT price FROM orders o WHERE o.customer_id = c.id);
```

id	name	debt
4	Paul George	300.00

Подзапрос для получения набора значений, которым нельзя равняться (NOT IN)

С помощью оператора **NOT IN** можно проверить, не входит ли запрашиваемое условием значение в тот наборе данных, которые вернул подзапрос - если входит, то условие считается невыполненным. Таким образом, **NOT IN** является антиподом оператора **IN**. Получим при помощи него всех клиентов, которые не совершали заказов.

```
SELECT id, name FROM customers WHERE id NOT IN (SELECT customer_id FROM orders);
```

id	name
2	Brandon Adams

Коррелирующие подзапросы

Введение

Когда мы получали список злостных должников, то мы должны были обратить внимание на получение данных внешнего основного запроса внутри подзапроса. Такой вид подзапроса называется коррелирующим, и он очень часто может привести к нехорошим последствиям. Дело в том, что при коррелирующем подзапросе этот подзапрос происходит не один раз, а столько раз, сколько у нас строк в основном запросе, т.к. проверка значения осуществляется для каждой строки. Например, если у нас 1000 клиентов, то общее количество запросов будет равно 1000.

В контексте коррелирующих запросов следует понимать, что корреляция не связана с использованием какой-либо конкретной конструкции (**IN**, **ANY** и т.д.). Именно обращение к данным внешнего запроса делает любой подзапрос коррелирующим. К сожалению, мы не может исправить наш оригинальный запрос, т.к. это единственный вариант получить всех злостных должников:

```
SELECT c.id, c.name, c.debt FROM customers c
WHERE c.debt > ANY(SELECT price FROM orders o WHERE o.customer_id = c.id);
```

Применение коррелирующих запросов в контексте оператора ALL

Чтобы проиллюстрировать работу коррелирующих запросов, обратимся к нового для нас оператора **ALL** - он используется схожим с **ANY** образом, поддерживает операторы сравнения , >=, <, <=, =, <>, требует возвращение только одного столбца, но одновременно считает условие истинным, если значения всех (а не хотя бы какого либо одного, как у **ANY**) строк будут соответствовать проверяемому значению. Например, попробуем выбрать всех посетителей, которые делали заказы в магазинах только тех городов, где они и живут:

```
SELECT c.id, c.name, c.city FROM customers c WHERE c.city = ALL(
   SELECT city FROM shops s WHERE s.id IN (
      SELECT o.shop_id FROM orders o WHERE o.customer_id = c.id
   )
);
```

id	name	city
1	John Smith	Rīga
2	Brandon Adams	Liepaja
4	Paul George	Rīga

Применение коррелирующих запросов в контексте оператора SELECT

Еще один яркий пример коррелирующего подзапроса можно проиллюстрировать на примере того, как использовать подзапрос не в качестве участника фильтрации выборки, а в качестве присоединенного динамического столбца, т.е. выполним подзапрос сразу после **SELECT**. Для этого попробуем получить в контексте запроса посетителей тот магазин (любой один), который находится в том городе, где посетитель живет.

```
SELECT c.name, (SELECT s.brand FROM shops s WHERE s.city = c.city LIMIT 1) AS shop_brand
FROM customers c;
```

name	city
John Smith	Maxima
Brandon Adams	Top
Mary Sue	NULL
Paul George	Rīga
Connor MacLeod	Rimi

Применение оператора EXISTS

Рассмотрим еще один оператор, который позволяет осуществлять подзапрос - **EXISTS**. Ему вообще не нужно значение для сравнения, не нужно следить за количеством столбцов или строк - условие будет считаться выполненным, если **EXISTS** найдет хотя бы одну любую строку. Но **EXISTS**, как и все остальные операторы, рассмотренные в нашем уроке, имеет доступ к значениям верхних запросов, поэтому он позволяет осуществлять коррелирующие запросы. Добавим, что у **EXISTS** есть конструкция-антипод **NOT EXISTS**, которая считает условие выполненным, если подзапрос не вернул ни одной строки. Попробуем использовать оператор **EXISTS**, чтобы получить тех посетителей, у которых есть заказы на сумму 1000 евро или более:

```
SELECT c.id, c.name FROM customers c
WHERE EXISTS(SELECT * FROM orders o WHERE o.customer_id = c.id AND o.price >= 1000);
```

id	name
3	Mary Sue
5	Connor MacLeod

Применение оператора NOT EXISTS

Теперь сделаем наоборот - при помощи **NOT EXISTS** получим тех посетителей без заказов на сумму 1000 или более евро:

```
SELECT c.id, c.name FROM customers c
WHERE NOT EXISTS(SELECT * FROM orders o WHERE o.customer_id = c.id AND o.price >= 1000);
```

id	name
1	John Smith
2	Brandon Adams
4	Paul George

Решение проблемы корреляции EXISTS/NOT EXISTS

Оба предыдущих запроса сделаны коррелирующими только в целях знакомства с конструкциями **EXISTS / NOT EXISTS** - их можно переписать и без корреляции:

```
-- есть заказы на 1000 или более 1000 евро
SELECT id, name FROM customers
WHERE id IN(SELECT customer_id FROM orders WHERE price >= 1000);

-- нет заказов на 1000 или более 1000 евро
SELECT id, name FROM customers
WHERE id NOT IN(SELECT customer_id FROM orders WHERE price >= 1000);
```