***КОМАНДА SELECT***

Команда **SELECT** извлекает данные из столбцов одной или нескольких таблиц. Команда SELECT сам по себе является запросом. Если он используется как предложение внутри другого оператора, то он называется подзапросом. В операторе SELECT обязательно должно присутствовать предложение FROM. Остальные предложения не являются необходимыми.

Предложение SELECT может использоваться как:

* самостоятельная команда на получение и вывод строк таблицы, сформированной из столбцов и строк одной или нескольких таблиц (представлений);
* фраза выбора в командах CREAT VIEW, DECLARE CURSOR или INSERT;
* средство присвоения глобальным переменным значений из строк сформированной таблицы (INTO-фраза).

**Синтаксис команды:**

**SELECT** alias.столбец\_1, alias.столбец\_2 ….,

функция(столбец\_3), (столбец\_1+ столбец\_2) as выражение

**FROM** таблица\_1 alias, таблица\_2 alias, (запрос\_1) alias

**WHERE** условие\_1 [OR [NOT]| AND [NOT]] условие\_2

**GROUP BY** Групповой\_столбец\_1, групповой\_столбец2

**HAVING** условие\_по\_групповому\_столбцу

**ORDER BY** столбец\_1 **ASC**|**DESC**

После указания перечня выбираемых столбцов и операций над ними, указывается служебное слово FROM – оно определяет ресурс данным, т.е. откуда будут выбираться данные, в качестве ресурса данных могут выступать таблицы, представления, запросы.

Таблице или столбцу можно присвоить альтернативное имя (алиас - **alias**), действие которого будет действительно только в пределах оператора, в котором оно определено. Если альтернативное имя стоит после имени столбца в списке оператора SELECT, то оно будет использоваться вместо настоящего имени как заголовок для данного столбца. Использование имен столбцов и алиаса приведен в примере ниже.

|  |
| --- |
| select emp.first\_name "Имя", emp.last\_name "Фамилия"  from HR.employees emp |

Альтернативное имя таблицы можно использовать при соединении таблицы с самой собой в соотносящемся запросе. При использовании таблиц с одинаковыми именами полей требуется указание имени таблицы перед именем колонки таблицы.

Для выбора данных по условию применяют предложение **WHERE**. После его указания определяется перечень условий, разделяемые между собой логикой – элементы AND, NOT AND, OR, NOT OR. Условием в языке SQL называется сочетание одного или нескольких выражений и логических операторов, вырабатывающих значение TRUE (истина) или FALSE (ложь). Каждое условие можно представить в виде операторов сравнения (см. таблица).

|  |  |
| --- | --- |
| **Оператор** | **Значение/действие в SQL** |
| = | равно |
| != или <> | не равно |
| >= | больше или равно |
| <= | меньше или равно |
| IN | равен любому элементу в |
| NOT IN | не равен любому элементу в |
| ANY | Сравнивает с любым из значений в списке. Употребляется после =, !=, >, <, <=, >=. |
| ALL | Сравнивает с каждым значением в списке. Употребляется после =, !=, >, <, <=, >=. |
| BETWEEN | больше или равно значения\_1 и меньше или равно значение\_2 |
| NOT | не больше и не равно |
| EXISTS | Истина, если подзапрос извлекает хотя бы одну строку |
| IS NULL | Истина, значение есть NULL. Проверки типа x=NULL – являются неправильными. |
| IS NOT NULL | Истина, если значение не пустое |

**Виды соединений.**



**JOIN** — оператор языка SQL, который является реализацией операции соединения реляционной алгебры. Входит в предложение [FROM](https://ru.wikipedia.org/w/index.php?title=From_(SQL)&action=edit&redlink=1) операторов SELECT, UPDATE и DELETE.

Операция соединения, как и другие бинарные операции, предназначена для обеспечения выборки данных из двух таблиц и включения этих данных в один результирующий набор. Отличительными особенностями операции соединения являются следующие:

* в схему таблицы-результата входят столбцы обеих исходных таблиц (таблиц-операндов), то есть схема результата является «сцеплением» схем операндов;
* каждая строка таблицы-результата является «сцеплением» строки из одной таблицы-операнда со строкой второй таблицы-операнда.

Определение того, какие именно исходные строки войдут в результат и в каких сочетаниях, зависит от типа операции соединения и от явно заданного *условия соединения*. Условие соединения, то есть условие сопоставления строк исходных таблиц друг с другом, представляет собой логическое выражение (предикат).

При необходимости соединения не двух, а нескольких таблиц, операция соединения применяется несколько раз (последовательно).

SQL-операция JOIN является реализацией операции соединения реляционной алгебры только в некотором приближении, поскольку в реляционной модели данных соединение выполняется над отношениями, которые являются множествами, а в SQL — над таблицами, которые являются мультимножествами. Результаты операций тоже, в общем случае, различны: в реляционной алгебре результат соединения даёт отношение (множество), а в SQL — таблицу (мультимножество).

**Описание оператора**

**FROM**

Table1

{**INNER** | {**LEFT** | **RIGHT** | **FULL**} **OUTER** | **CROSS** } **JOIN**

Table2

{**ON** <condition> | **USING** (field\_name [,... n])}

В большинстве СУБД при указании слов LEFT, RIGHT, FULL слово OUTER можно опустить. Слово INNER также в большинстве СУБД можно опустить.

В общем случае СУБД при выполнении соединения проверяет условие (предикат) *condition*. Если названия столбцов, по которым происходит соединение таблиц, совпадают, то вместо ON можно использовать USING. Для CROSS JOIN условие не указывается.

Для перекрёстного соединения (декартова произведения) CROSS JOIN в некоторых реализациях SQL используется оператор «запятая» (**,**), что актуально для Oracle:

**FROM**

Table1,

Table2

**Виды оператора JOIN**

Для дальнейших пояснений будут использоваться следующие таблицы:

**City (Города)**

|  |  |
| --- | --- |
| Id | Name |
| 1 | Москва |
| 2 | Санкт-Петербург |
| 3 | Казань |

**Person (Люди)**

|  |  |
| --- | --- |
| Name | CityId |
| Андрей | 1 |
| Леонид | 2 |
| Сергей | 1 |
| Григорий | 4 |

**INNER JOIN**

Оператор *внутреннего соединения* INNER JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является коммутативным.

*Заголовок* таблицы-результата является объединением (конкатенацией) заголовков соединяемых таблиц.

*Тело* результата логически формируется следующим образом. Каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы, после чего для полученной «соединённой» строки проверяется условие соединения (вычисляется предикат соединения). Если условие истинно, в таблицу-результат добавляется соответствующая «соединённая» строка.

Описанный алгоритм действий является строго логическим, то есть он лишь объясняет результат, который должен получиться при выполнении операции, но не предписывает, чтобы конкретная СУБД выполняла соединение именно указанным образом. Существует несколько способов реализации операции соединения, например, соединение вложенными циклами ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *inner loops join*), соединение [хешированием](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *hash join*), соединение слиянием ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *merge join*). Единственное требование состоит в том, чтобы любая реализация логически давала такой же результат, как при применении описанного алгоритма.

В рамках работы в SQLDeveloper указывать слово INNER необязательно.

**SELECT** \*

**FROM** Person

**INNER JOIN** City **ON** Person.CityId = City.Id

ИЛИ

**SELECT** \*

**FROM** Person

**JOIN** City **ON** Person.CityId = City.Id

Результат:

|  |  |  |  |
| --- | --- | --- | --- |
| Person.Name | Person.CityId | City.Id | City.Name |
| Андрей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| Сергей | 1 | 1 | Москва |

**OUTER JOIN**

Соединение двух таблиц, в результат которого обязательно входят все строки либо одной, либо обеих таблиц.

**LEFT OUTER JOIN**

Оператор *левого внешнего соединения* LEFT OUTER JOIN соединяет две таблицы. Порядок таблиц для оператора важен, поскольку оператор не является коммутативным.

*Заголовок* таблицы-результата является объединением (конкатенацией) заголовков соединяемых таблиц.

*Тело* результата логически формируется следующим образом. Пусть выполняется соединение левой и правой таблиц по предикату (условию) *p*.

1. В результат включается внутреннее соединение (INNER JOIN) левой и правой таблиц по предикату *p*.
2. Затем в результат добавляются те строки левой таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких строк столбцы, соответствующие правой таблице, заполняются значениями NULL.

В рамках работы в SQLDeveloper указывать слово OUTER необязательно.

**SELECT** \*

**FROM**

Person *-- Левая таблица*

**LEFT OUTER** **JOIN**

City *-- Правая таблица*

**ON** Person.CityId = City.Id

ИЛИ

**SELECT** \*

**FROM** Person *-- Левая таблица*

**LEFT JOIN** City **ON** Person.CityId = City.Id *-- Правая таблица*

Результат:

|  |  |  |  |
| --- | --- | --- | --- |
| Person.Name | Person.CityId | City.Id | City.Name |
| Андрей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| Сергей | 1 | 1 | Москва |
| Григорий | 4 | NULL | NULL |

**RIGHT OUTER JOIN**

Оператор *правого внешнего соединения* RIGHT OUTER JOIN соединяет две таблицы. Порядок таблиц для оператора важен, поскольку оператор не является коммутативным.

*Заголовок* таблицы-результата является объединением (конкатенацией) заголовков соединяемых таблиц.

*Тело* результата логически формируется следующим образом. Пусть выполняется соединение левой и правой таблиц по предикату (условию) *p*.

1. В результат включается внутреннее соединение (INNER JOIN) левой и правой таблиц по предикату *p*.
2. Затем в результат добавляются те строки правой таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких строк столбцы, соответствующие левой таблице, заполняются значениями NULL.

В рамках работы в SQLDeveloper указывать слово OUTER необязательно.

**SELECT** \*

**FROM**

Person *-- Левая таблица*

**RIGHT OUTER** **JOIN**

City *-- Правая таблица*

**ON** Person.CityId = City.Id

ИЛИ

**SELECT** \*

**FROM** Person *-- Левая таблица*

**RIGHT** **JOIN** City **ON** Person.CityId = City.Id *-- Правая таблица*

Результат:

|  |  |  |  |
| --- | --- | --- | --- |
| Person.Name | Person.CityId | City.Id | City.Name |
| Андрей | 1 | 1 | Москва |
| Сергей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| NULL | NULL | 3 | Казань |

**FULL OUTER JOIN**

Оператор *полного внешнего соединения* FULL OUTER JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является коммутативным.

*Заголовок* таблицы-результата является объединением (конкатенацией) заголовков соединяемых таблиц.

*Тело* результата логически формируется следующим образом. Пусть выполняется соединение первой и второй таблиц по предикату (условию) *p*. Слова «первой» и «второй» здесь не обозначают порядок в записи выражения (который неважен), а используются лишь для различения таблиц.

1. В результат включается внутреннее соединение (INNER JOIN) первой и второй таблиц по предикату *p*.
2. В результат добавляются те строки первой таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких строк столбцы, соответствующие второй таблице, заполняются значениями NULL.
3. В результат добавляются те строки второй таблицы, которые не вошли во внутреннее соединение на шаге 1. Для таких строк столбцы, соответствующие первой таблице, заполняются значениями NULL.

В рамках работы в SQLDeveloper указывать слово OUTER необязательно.

**SELECT** \*

**FROM**

Person

**FULL** **OUTER** **JOIN**

City

**ON** Person.CityId = City.Id

ИЛИ

**SELECT** \*

**FROM** Person

**FULL** **JOIN** City **ON** Person.CityId = City.Id

Результат:

|  |  |  |  |
| --- | --- | --- | --- |
| Person.Name | Person.CityId | City.Id | City.Name |
| Андрей | 1 | 1 | Москва |
| Сергей | 1 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| NULL | NULL | 3 | Казань |
| Григорий | 4 | NULL | NULL |

**CROSS JOIN**

Оператор *перекрёстного соединения*, или *декартова произведения* CROSS JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является коммутативным.

*Заголовок* таблицы-результата является объединением (конкатенацией) заголовков соединяемых таблиц.

*Тело* результата логически формируется следующим образом. Каждая строка одной таблицы соединяется с каждой строкой второй таблицы, давая тем самым в результате все возможные сочетания строк двух таблиц.

**SELECT** \*

**FROM**

Person

**CROSS** **JOIN**

City

или

**SELECT** \*

**FROM** Person, City

Результат:

|  |  |  |  |
| --- | --- | --- | --- |
| Person.Name | Person.CityId | City.Id | City.Name |
| Андрей | 1 | 1 | Москва |
| Андрей | 1 | 2 | Санкт-Петербург |
| Андрей | 1 | 3 | Казань |
| Леонид | 2 | 1 | Москва |
| Леонид | 2 | 2 | Санкт-Петербург |
| Леонид | 2 | 3 | Казань |
| Сергей | 1 | 1 | Москва |
| Сергей | 1 | 2 | Санкт-Петербург |
| Сергей | 1 | 3 | Казань |
| Григорий | 4 | 1 | Москва |
| Григорий | 4 | 2 | Санкт-Петербург |
| Григорий | 4 | 3 | Казань |

Если в предложении *WHERE* добавить условие соединения (предикат *p*), то есть ограничения на сочетания кортежей, то результат эквивалентен операции INNER JOIN с таким же условием:

**SELECT** \*

**FROM**

Person,

City

**WHERE**

Person.CityId = City.Id

Таким образом, выражения t1, t2 WHERE p и t1 INNER JOIN t2 ON p синтаксически являются альтернативными формами записи одной и той же логической операции внутреннего соединения по предикату *p*. Синтаксис CROSS JOIN + WHERE для операции соединения называют устаревшим, его не рекомендует стандарт SQL ANSI.

**Запросы с группированием (фразы Group BY и HAVING).**

## Описание

SQL оператор **GROUP BY** можно использовать в SELECT для сбора данных по нескольким записям и группировки результатов одного или нескольких столбцов.

**Оператор HAVING** используется в сочетании с оператором GROUP BY, чтобы ограничить группы возвращаемых строк только теми, для которых выполняется некоторое условие.

## Синтаксис

SELECT expression1, expression2, … expression\_n,  
aggregate\_function (aggregate\_expression)  
FROM tables  
[WHERE conditions]  
GROUP BY expression1, expression2, … expression\_n

[HAVING conditions]

## Параметры или аргументы

expression1, expression2, … expression\_n — выражения, которые не входят в aggregate\_function и должны быть включены в предложение GROUP BY в конце оператора SQL.

aggregate\_function — это функция, такая как SUM, COUNT, функции MIN, MAX или AVG.

aggregate\_expression – это столбец или выражение, которое используется в aggregate\_function.

Tables – таблицы из которых вы хотите выгрузить данные. После оператора FROM должна быть указана хотя бы одна таблица.

WHERE conditions. Необязательный. Условия, которые должны быть выполнены для записей, которые будут выбраны.

HAVING conditions — это дополнительные условия применяемые только к агрегированным результатам для ограничения групп возвращаемых строк. В результирующий набор будут включены только те группы, состояние которых соответствует TRUE.

## Пример использования функции MIN и MAX

Определить номера департаментов с минимальной зарплатой больше или равной 2500 и максимальной зарплатой меньшей 10000.

**SELECT** emp.department\_id, min(emp.salary), max(emp.salary)

**FROM** employees emp

**GROUP BY** emp.department\_id

**HAVING** min(emp.salary)>=2500 AND max(emp.salary)<10000

**ORDER BY** min(emp.salary)

**Вложенные запросы (подзапросы) в SQL Oracle**

**Цели лабораторной работы**

Изучить возможности SQL Oracle по формулировке и обработке подзапросов.

Приобрести практический опыт по формулировке и обработке подзапросов с использованием SQL\*Plus.

**Теоретические основы**

***Запрос*** – это операция, которая позволяет отыскивать данные из одной или несколько таблиц. При наличии вложенных запросов запрос верхнего уровня называется предложением SELECT, а запрос, вложенный в предложение SELECT называется подзапросом. Таким образом, ***подзапрос*** (вложенный запрос) – это запрос, результат которого передается в качестве аргумента в другой запрос. Подзапросы позволяют связывать в единое целое несколько запросов.

Подзапросы используются для:

определения множества строк, который должны быть вставлены в целевую таблицу в предложениях INSERT или CREATE TABLE;

определения одного или более значений, присваиваемых существующим строка в предложении UPDATE;

получения значений для фраз WHERE, HAVING или START WITH в предложениях SELECT, UPDATE, и DELETE;

определения значений указанного столбца в списке INSERT ... VALUES;

определения таблицы, которая используются соответствующим запросом.

Это производится путем размещения подзапроса во фразе FROM соответствующего запроса как если бы это было именем таблицы. Вы можете также использовать таким образом подзапросы вместо таблиц в предложениях INSERT, UDPATE и DELETE.

Используемые таким образом подзапросы могут использовать переменные связывания (correlation variables), однако только такие, которые определены только в самом подзапросе, ссылки на внешние переменные не допустимы. Внешние ссылки (подзапросы с левой корреляцией - left-correlated subqueries) допустимы только во фразе FROM предложения SELECT .

Подзапрос дает ответ на содержательные запросы, имеющие сложную структуру. Например, для определения, кто работает на кафедре Иванова, вы сначала используете подзапрос для определения кафедры, на которой работает Иванов, а затем отвечаете на основной запрос путем формулировки предложения SELECT.

Подзапрос может содержать другие подзапросы. Oracle не ограничивает глубину вложенности подзапросов.

Если таблица в подзапросе имеет такое же имя, что и таблица внешнего запроса, то для ссылки на столбцы внешнего запроса их необходимо уточнять именем таблицы или алиасом таблицы. Чтобы ваши запросы было легче воспринимать, всегда квалифицируйте столбцы в подзапросе именем или алиасом таблицы.

Oracle выполняет ***корреляционные (связанный) подзапрос***, когда подзапрос ссылается на столбец таблицы внешнего запроса. Связанный подзапрос вычисляется для каждой строки, обрабатываемой внешним предложением. Внешним предложением может быть SELECT, UPDATE или DELETE.

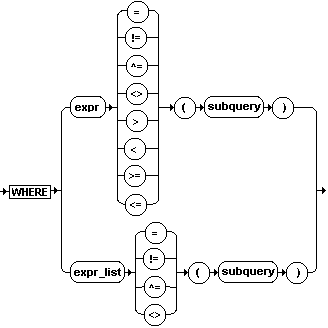
Связанный подзапрос дает ответы на такие содержательные запросы, ответы которых требуют вычисления подзапросов для каждой строки внешнего запроса. Например, связанный подзапрос используется для определения преподавателей, которые зарабатывают больше, чем средняя зарплата по кафедре. В этом случае связанный подзапрос для каждого преподавателя вычисляет среднюю зарплату на его кафедре.

Далее мы обсудим использование подзапросов в предложении SELECT.

**Подзапрос во фразе WHERE**

**Подзапрос в простом условии сравнения**

**Синтаксис:**



**Описание:**

При использовании простых условий сравнения с подзапросом во фразе WHERE применяются следующие правила:

Подзапрос должен возвращать единственную строку.

Если левая часть равна *expr*, то подзапрос должен возвращать единственную строку с единственным значением с типом, совместимым с типом *expr*.

Если левая часть является списком выражений (*expr\_list*), то подзапрос должен возвращать единственную строку со списком значений, который соответствует по количеству и типу значениям из *expr\_list*. В этом случае оператор сравнения дает TRUE, если каждое значение в *expr\_list* равно (в случае =) или не равно (в случае !=, ^=, <>) каждому значение, возвращаемому подзапросом.

**Примеры**:

1. Выбрать кафедры, которые располагаются в том же корпусе, что факультет информатики:

SELECT Name

FROM DEPARTMENT

WHERE Building = (SELECT Building

FROM FACULTY

WHERE UPPER(Name) = 'INFORMATICS');

2. Выбрать факультеты, чьи фонды меньше фонда кафедры CAD:

SELECT Name

FROM FACULTY

WHERE Fund < (SELECT Fund

FROM DEPARTMENT

WHERE UPPER(Name) = 'CAD');

3. Выбрать преподавателей, у которых salary + commission превышает более чем на 100 половину salary + commission преподавателя Bill:

SELECT Name

FROM TEACHER

WHERE Salary + Commission + 100 > (SELECT (Salary + Commission) / 2

FROM TEACHER

WHERE UPPER(Name) = 'BILL');

4. Выбрать преподавателей, которые работают на той же кафедре, что и Bill и занимают ту же должность, что и Bill:

SELECT Name

FROM TEACHER

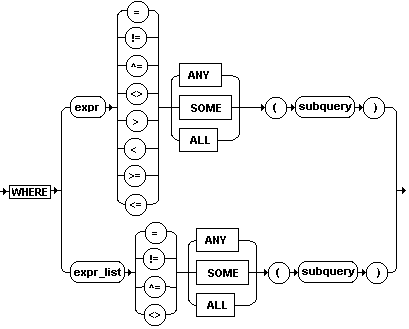
WHERE (DepNo, Post) = (SELECT DepNo, Post

FROM TEACHER

WHERE UPPER(Name) = 'BILL');

**Подзапрос в условии сравнения групп**

**Синтаксис:**



**Описание:**

При использовании условий сравнения групп с подзапросом во фразе WHERE применяются следующие правила:

Подзапрос может возвращать ноль или более строк.

Если левая часть равна *expr*, то подзапрос должен возвращать строки с единственным значением, которые совместимы по типу с *expr*.

Если левая часть равна *expr\_list*, то подзапрос должен возвращать строки со списком значений, который соответствует по количеству и типу с *expr\_list*.

ANY и SOME эквивалентны и сравнивают значение слева с каждым значением списка справа, возвращаемого подзапросом. Подзапрос может вернуть ноль или более строк. Условие равно TRUE, если по крайней мере одна строка подзапроса удовлетворяет условию (соответствует оператору сравнения) по отношению к значению (списку значений) определенному левым операндом, в противном получаем FALSE. Если подзапрос не возвращает строк, то получаем FALSE.

ALL сравнивают значение слева с каждым значением списка справа, возвращаемого подзапросом. Дает TRUE, если ВСЕ строки, возвращаемые подзапросом, удовлетворяют условию (соответствуют оператору сравнения) по отношению к значению (списку значений) определенному левым операндом, в противном получаем FALSE. Если подзапрос не возвращает строк, то получаем TRUE

**Примеры:**

1. Выдать кафедры, фонд которых больше фонда по крайней мере одного из факультетов:

SELECT Name

FROM DEPARTMENT

WHERE Fund > ANY (SELECT Fund FROM FACULTY);

**ANY,** ALL **и агрегатные функции**. Обратите внимание, что оператор “< ANY” эквивалентно утверждению: “*левое значение меньше, чем максимальное значение из множества, задаваемого правым операндом*”, а оператор “> ANY” эквивалентен следующему утверждению “*левое значение больше, чем минимальное значение из множества, задаваемого правым операндом*”. Поэтому операторы ANY могут быть выражены через функции MAX и MIN в подзапросе. В свою очередь, “< ALL” (“> ALL”) эквивалентно утверждению: «*левое значение меньше (больше), чем минимальное (максимальное) значение из множества, задаваемого правым операндом* ».

2. Выдать кафедры, фонд которых больше фонда по крайней мере одного из факультетов:

SELECT Name

FROM DEPARTMENT

WHERE Fund > ANY (SELECT Fund FROM FACULTY);

SELECT Name

FROM DEPARTMENT

WHERE Fund > (SELECT MIN(Fund) FROM FACULTY);

3. Выдать группы, которые имеют рейтинг больше, чем рейтинг всех групп пятого курса кафедры “DBMS”:

SELECT Num

FROM SGROUP

WHERE Rating >ALL (SELECT Rating

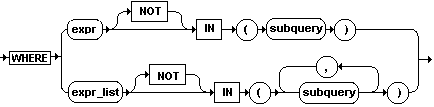
FROM SGROUP, DEPARTMENT

WHERE SGROUP.DepNo = DEPARTMENT.DepNo AND

UPPER(DEPARTMENT.Name) = 'DBMS' AND SGROUP.Course = 5);

**Подзапрос в условии проверки вхождения элемента во множество**

**Синтаксис:**



**Описание:**

Это условие в таком синтаксисе проверят вхождение элемента (списка элементов) во множество (множество списков), создаваемое подзапросом.

**Пример:**

1. Выбрать преподавателей, которые имеют лекции по крайней мере одному такому предмету, по которым читает лекции преподаватель Bill:

SELECT Name

FROM TEACHER T, LECTURE L

WHERE T.TchNo = L.TchNo AND

SbjNo IN (SELECT SbjNo

FROM TEACHER TCH, LECTURE LEC

WHERE TCH.TchNo = LEC.TchNo AND UPPER(TCH.Name) = 'BILL');

**Подзапрос в условии EXISTS**

**Синтаксис:**



**Описание:**

Дает TRUE, если подзапрос возвращает по крайней мере одну строку.

Так как EXISTS обычно используется в связанных подзапросах, мы его обсудим подробнее позже.

**Связанные подзапросы**

Для того, чтобы связать подзапрос с внешним запросом (предложением), необходимо в подзапросе была ссылка на столбец внешнего запроса. Подзапрос вычисляется для каждой строки, обрабатываемой внешним запросом (предложением). В качестве внешнего предложения могут выступать SELECT, UPDATE или DELETE.

Следующие примеры дают общий синтаксис использования связанных подзапросов:

SELECT select\_list

FROM table1 t\_alias1

WHERE expr operator

(SELECT column\_list

FROM table2 t\_alias2

WHERE t\_alias1.column operator t\_alias2.column);

UPDATE table1 t\_alias1

SET column =

(SELECT expr

FROM table2 t\_alias2

WHERE t\_alias1.column = t\_alias2.column);

DELETE FROM table1 t\_alias1

WHERE column operator

(SELECT expr

FROM table2 t\_alias2

WHERE t\_alias1.column = t\_alias2.column);

Далее мы обсудим использование связанных подзапросов во фразе WHERE предложения SELECT.

**Связанные подзапросы во фразе WHERE**

**Примеры:**

1. Выдать преподавателей, которые имеют по крайней мере одну лекцию:

SELECT Name

FROM TEACHER

WHERE EXISTS (SELECT \*

FROM LECTURE

WHERE LECTURE.TchNo = TEACHER.TchNo);

Здесь в условии LECTURE.TchNo = TEACHER.TchNo подзапроса мы ссылаемся на внешний запрос. Поэтому подзапрос является связанным.

2. Выдать преподавателей, которые не имеют ни одной лекции:

SELECT Name

FROM TEACHER

WHERE NOT EXISTS (SELECT \*

FROM LECTURE

WHERE LECTURE.TchNo = TEACHER.TchNo);

**Простые и связанные подзапросы во фразе HAVING**

Вы можете использовать простые и связанные подзапросы во фразе HAVING.

Если вы используете связанный подзапрос в фразе HAVING, то в подзапросе можно ссылаться на те столбцы внешнего запроса, которые могут использоваться в фразе HAVING (обычно это столбцы, по которым производится группирование).

**Примеры:**

1. Перечислить факультеты, у которых сумма фондов финансирования всех их кафедр превышает более чем на 20000 фонд финансирования той кафедры факультета, которая имеет максимальный фонд.

SELECT F1.Name

FROM FACULTY F1, DEPARTMENT D1

WHERE F1.FacNo = D1.FacNo

GROUP BY F1.Name

HAVING SUM(D1.Fund) > (SELECT 200000 + MAX(D2.Fund)

FROM FACULTY F2, DEPARTMENT D2

WHERE F2.FacNo = D2.FacNo AND F1.Name = F2.Name);

**Простые подзапросы во фразе FROM**

Фраза FROM может содержать не только список имен таблиц, но и подзапросы. Для ссылки на такие таблицы-подзапросы следует приписать подзапросу алиас.

Имеется класс запросов, которые не могут быть выражены без подзапроса во фразе FROM. К ним, в частности, относятся такие запросы, которые требуют независимого вычисления двух или более запросов, и затем совместного использования результатов такого запроса.

**Пример:**

Выдать средний фонд финансирования факультетов и среднюю зарплату преподавателей:

SELECT Fac.AvgFund, Tch.AvgSalary

FROM (SELECT AVG(Fund) AS AvgFund FROM FACULTY) Fac,

(SELECT AVG(Salary) AS AvgSalary FROM TEACHER) Tch;

**Подзапросы во фразе SELECT**

Во фразе SELECT можно использовать простые (независимые, несвязанные) и связанные (коррелированные) запросы. **В обоих случаях подзапрос должен возвращать одно значение**.

Подзапрос является простым, если в нем не используются атрибуты таблиц, определенных в основном (внешнем запросе). При использовании простого подзапроса он вычисляется однократно и возвращенное им значение вставляется в соответствующее место во все строки, формируемые внешним запросом.

Пример. Для каждого факультета вывести его название, фонд финансирование, а также максимальный и минимальный фонды финансирования среди всех кафедр ВУЗа

SELECT Name AS "Факультет",

Fund AS "Фонд факультета",

(SELECT MAX(Fund) FROM DEPARTMENT) AS "МАКС фонд кафедр",

(SELECT MIN(Fund) FROM DEPARTMENT) AS "МИН фонд кафедр"

FROM FACULTY

Подзапрос является связанным (коррелированным), если в нем используются атрибуты таблиц, определенных во внешней запросе. В этом случае подзапрос вычисляется для каждой строки, формируемой для фразы SELECT.

Пример. По каждому факультету, расположенному в корпусе 6, вывести:

- название факультета

- количество групп этого факультета с рейтингом, более 20

- количество преподавателей-профессоров

SELECT Name AS "Факультет",

(SELECT COUNT (DISTINCT GrpPK)

FROM DEPARTMENT d, SGROUP g

WHERE f.FacPK=d.FacFK AND d.DepPK=g.DepFK AND g.Rating > 20) AS "К-во групп",

(SELECT COUNT (DISTINCT TchPK)

FROM DEPARTMENT d, TEACHER t

WHERE f.FacPK=d.FacFK AND d.DepPK=t.DepFK AND

UPPER(t.Post)= 'профессор') AS "К-во профессоров"

FROM FACULTY f

WHERE Building= '6';