# Лабораторная работа №5

## Часть 1. Реализация SQL-запросов на выборку данных с использованием подзапросов, агрегатных функций, группировки и операций над множествами

В лабораторной работе выполняется создание запросов на выборку данных на языке SQL с использованием подзапросов, агрегатных функций, а также группировки данных (предложение **`GROUP BY`** оператора **`SELECT`**) и операций над множествами (**`UNION`**, **`INTERSECT`**, **`MINUS`**).

### Порядок выполнения работы

1) Получить у преподавателя задания по вашей собственной схеме данных, созданной в лабораторной работе №2 и реализованной в виде таблиц в СУБД в лабораторной работе №3. Создать запросы по заданиям (по одному запросу на каждое задание).

2) Правила выполнения заданий:

- для каждого задания создать реализацию в виде одного оператора SQL **`SELECT`**, в котором можно использовать подзапросы и группировку данных;
- обратить внимание, что использование скалярных (особенно соотнесенных !) подзапросов в предложении **`SELECT`** следует ограничить, т.к. они ухудшают производительность и анализ запроса, поэтому, если запрос затрагивает несколько таблиц, то сначала надо собрать данные с помощью соединения данных таблиц, и только потом выполнять их обработку (например, группировать);
- перед запуском запроса на выполнение, изучить данные в используемых запросом таблицах, и если требуется добавить новые данные, чтобы результат выборки не был пустым;
- выполнить запрос и проанализировать его результат – если есть расхождения между изученными данными и результатом запроса, то есть повод задуматься о проверке правильности выполнения этого задания.

3) Оформить *отчет* (см. часть 2 данной лабораторной работы).

### Оператор `SELECT` и группировка данных

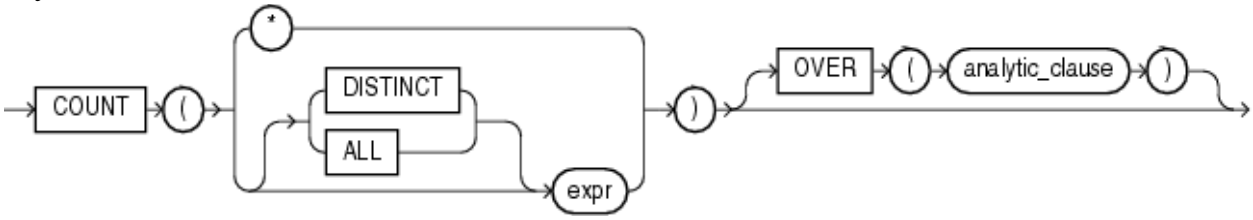
Агрегатные функции – это обобщающие функции, которые предназначены для выполнения вертикальных вычислений в таблицах – сжимают столбец значений до одного единственного значения.

Агрегатные функции общего назначения в СУБД Oracle: **AVG**, **COUNT**, **MAX**, **MEDIAN**, **MIN**, **SUM**.

Функция **AVG**:

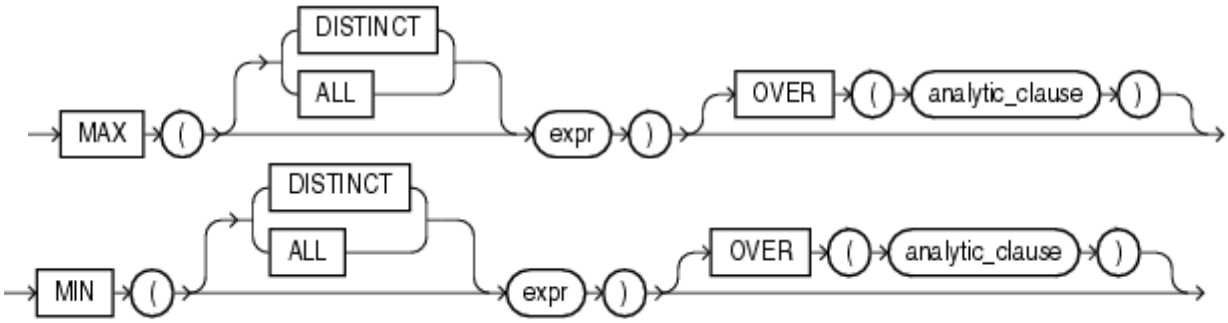
- возвращает среднее от всех значений в столбце;
- работает только с данными, которые можно представить в числовой форме;
- всегда подавляет **NULL**.

Функция **COUNT**:



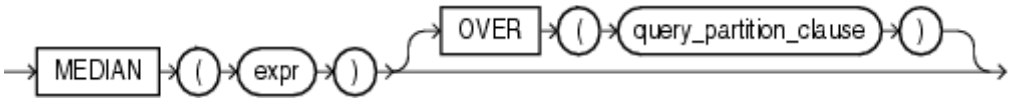

- возвращает число значений в столбце;
- при использовании в качестве аргумента звездочки (*) выполняет подсчет общего числа строк данных в таблице (и только в этом случае не подавляет **NULL**).

Функции **MAX** и **MIN**:




- расчет максимального или минимального значения в столбце;
- всегда подавляют **NULL**.

Функция **MEDIAN**:



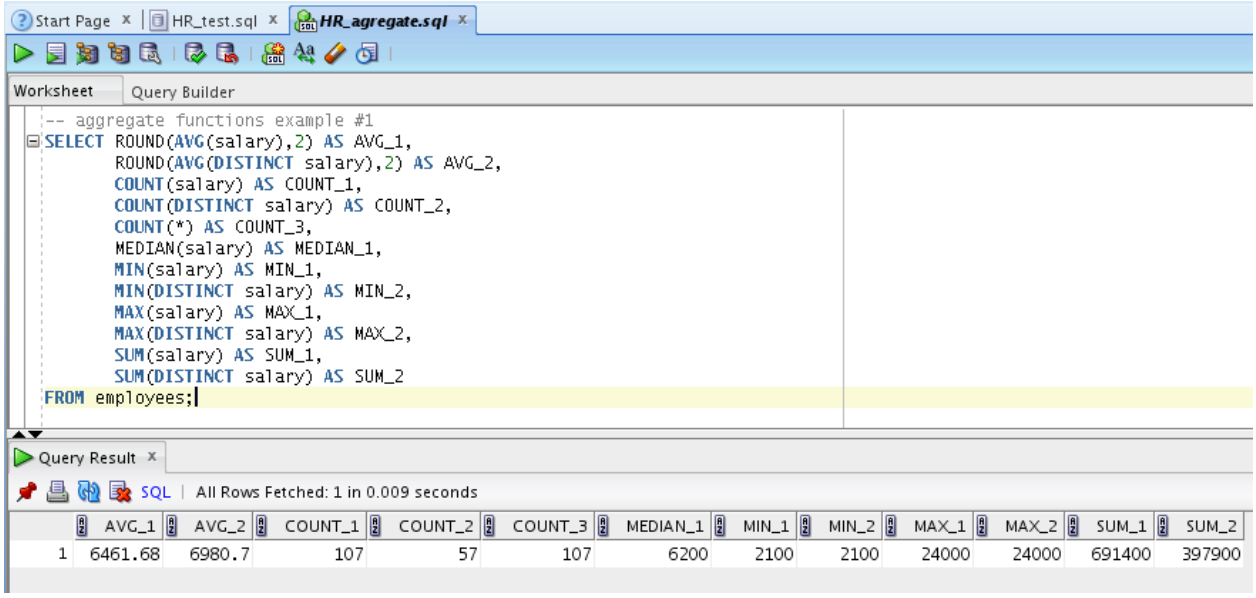

- возвращает медиану (значение *в средней позиции* в отсортированном списке значений) от всех значений в столбце;
- работает только с данными числовых (или конвертируемых в числовые) и временных типов;

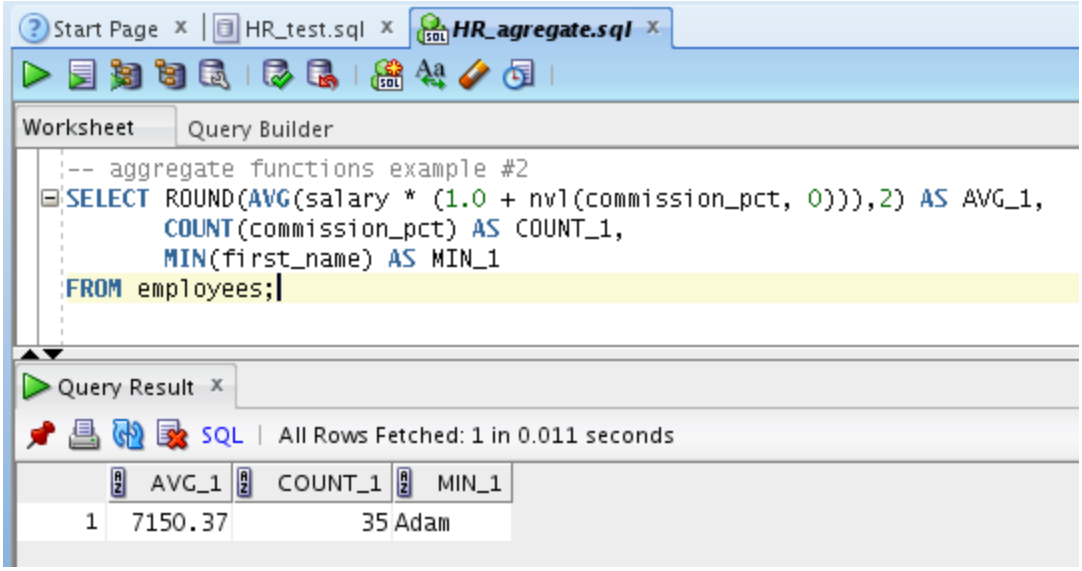
- всегда подавляет **NULL**.

Функция **SUM**:

- расчет суммы всех значений в столбце;
- работает только с данными, которые можно представить в числовой форме.

Ниже приведены примеры запросов на выборку с агрегатными функциями для схемы HR (см. лабораторную работу №4).



Группировка данных – обобщение данных в рамках строк с одинаковой комбинацией значений в группе (часть таблицы).

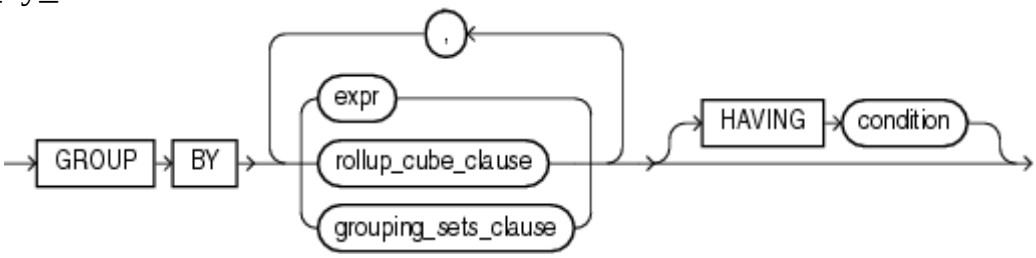
Группа – список столбцов таблицы, в рамках которых будут рассматриваться комбинации данных.

После выполнения группировки для каждой одинаковой комбинации в группе будет сформирована только одна результирующая строка (напоминает **DISTINCT** в списке выборки).

Особенности:

- на группировку поступают данные после фильтра **WHERE**;
- для получения статистики по группе нужно использовать агрегатные функции, зона действия которых будет зависеть от числа строк относящихся к конкретной группе;
- группы можно дополнительно фильтровать (прореживать).

Группировка выполняется в предложении **GROUP BY** оператора **SELECT**.
group_by_clause ::=




Пример группировки данных по одному столбцу:

```
-- grouping example #3
SELECT department_id,
       COUNT(*) AS employees_number
FROM employees
GROUP BY department_id
ORDER BY COUNT(*) DESC;
```

Query Result

SQL | All Rows Fetched: 12 in 0.017 se

| | DEPARTMENT_ID | EMPLOYEES_NUMBER |
|---|---|---|
| 1 | 50 | 45 |
| 2 | 80 | 34 |
| 3 | 100 | 6 |
| 4 | 30 | 6 |
| 5 | 60 | 5 |
| 6 | 90 | 3 |
| 7 | 20 | 2 |
| 8 | 110 | 2 |
| 9 | 40 | 1 |
| 10 | 10 | 1 |
| 11 | (null) | 1 |
| 12 | 70 | 1 |

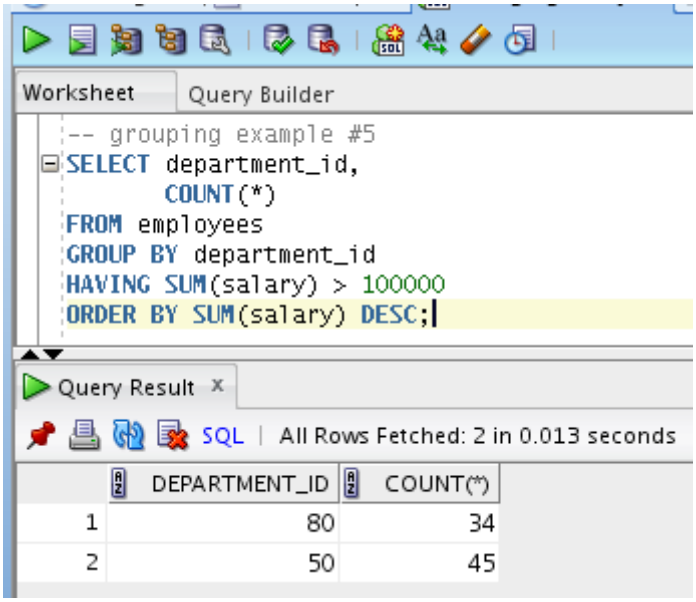
Фильтрация по группам выполняется в фильтре **HAVING** в предложении **GROUP BY**.

Условие (предикат) *condition* фильтра фактически аналогично по содержимому и обработке предикату в предложении **WHERE**.

Особенности:

- *expr* в предложении **HAVING** может строиться на основе использования агрегатных функций и фактически необходимо для удаления групп по статистическим данным;
- если в условии *condition* предложения **HAVING** отсутствует использование агрегатных функций, то, скорее всего, это условие нужно перенести в предложение **WHERE**, что повысит скорость обработки данных.

Пример фильтра по группам:




## Подзапросы в операторе SELECT

Подзапрос (*subquery*) – оператор **SELECT**, помещенный в круглых скобках в некоторое предложение основного оператора на языке SQL (т.е. некоторое вложение оператора выборки в другой оператор выборки).

Подзапрос в свою очередь может быть основным оператором для другого подзапроса и т.д. (максимальный уровень вложенности обычно равен *255*).

Назначение подзапроса – выборка данных в виде таблицы для использования в основном операторе **SELECT**. Ограничения - подзапрос не может содержать **ORDER BY**.

Подзапросы классифицируются по структуре возвращаемой таблицы и связи между данными:

- скалярный подзапрос (*scalar subquery*) – возвращает всегда только одно единственное значение (одну строку и один столбец в результирующей таблице);
- однострочный подзапрос (*single-row subquery*) –возвращает всегда только одну строку;
- многострочный подзапрос (*multiple-row subquery*) –возвращает несколько строк данных (от нуля до много);
- многостолбцовый подзапрос (*multiple-column subquery*) – возвращает всегда более чем один столбец данных;
- коррелированный (соотнесенный) подзапрос (*correlated subquery*) – выборка данных в подзапросе зависит от текущей строки данных в основном запросе.

В данной лабораторной работе подзапрос в операторе **SELECT** можно использовать в следующих предложениях:

- **SELECT** – только скалярные подзапросы для построения выражений (не использовать для замены соединения данных нескольких таблиц !);
- **FROM** – все виды подзапросов (подзапрос служит источником данных (*inline view*));
- **WHERE**, **HAVING** – все виды подзапросов (подзапрос служит источником данных для предиката).

Пример использования скалярных подзапросов с агрегатными функциями (всегда возвращают только одно единственное значение) в предложении **SELECT**:

Worksheet | Query Builder

```
SELECT department_id, SUM(salary) AS dep_sum,
       (SELECT SUM(salary) FROM employees) AS total_sum,
       round((SUM(salary) / (SELECT SUM(salary) FROM employees) * 100), 2) AS dep_prc
FROM employees
GROUP BY department_id
ORDER BY department_id;
```

Query Result

SQL | All Rows Fetched: 12 in 0.01 seconds

| | DEPARTMENT_ID | DEP_SUM | TOTAL_SUM | DEP_PRC |
|---|---|---|---|---|
| 1 | 10 | 4400 | 691400 | 0.64 |
| 2 | 20 | 19000 | 691400 | 2.75 |
| 3 | 30 | 24900 | 691400 | 3.6 |
| 4 | 40 | 6500 | 691400 | 0.94 |
| 5 | 50 | 156400 | 691400 | 22.62 |
| 6 | 60 | 28800 | 691400 | 4.17 |
| 7 | 70 | 10000 | 691400 | 1.45 |
| 8 | 80 | 304500 | 691400 | 44.04 |
| 9 | 90 | 58000 | 691400 | 8.39 |
| 10 | 100 | 51600 | 691400 | 7.46 |
| 11 | 110 | 20300 | 691400 | 2.94 |

Примеры использования подзапросов в предложении **WHERE**:

a) использование скалярного подзапроса с агрегатной функцией для формирования простого сравнения:

Worksheet | Query Builder

```
SELECT employee_id, email
FROM employees
WHERE salary > (SELECT MAX(salary) FROM employees WHERE department_id = 30)
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 10 in 0.013 seconds

| | EMPLOYEE_ID | EMAIL |
|---|---|---|
| 1 | 100 | SKING |
| 2 | 101 | NKOCHHAR |
| 3 | 102 | LDEHAAN |
| 4 | 108 | NGREENBE |
| 5 | 145 | JRUSSEL |
| 6 | 146 | KPARTNER |
| 7 | 147 | AERRAZUR |
| 8 | 168 | LOZER |
| 9 | 201 | MHARTSTE |
| 10 | 205 | SHIGGINS |

b) использование многострочного (*multiple-row*) подзапроса с одним столбцом и оператора **IN** (проверка на вхождение элемента в множество):

Worksheet | Query Builder

```
SELECT employee_id, last_name, first_name
FROM employees
WHERE salary IN (SELECT salary FROM employees WHERE first_name = 'John')
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 5 in 0.024 seconds

| | EMPLOYEE_ID | LAST_NAME | FIRST_NAME |
|---|---|---|---|
| 1 | 110 | Chen | John |
| 2 | 121 | Fripp | Adam |
| 3 | 126 | Mikkilineni | Irene |
| 4 | 139 | Seo | John |
| 5 | 145 | Russell | John |

c) использование многострочного (*multiple-row*) подзапроса с одним столбцом и предикатом **ANY** (при использовании связки с оператором равенства (**= ANY**) результат аналогичен работе оператора **IN**):

Worksheet Query Builder

```
SELECT employee_id, last_name, first_name
FROM employees
WHERE salary = ANY (SELECT salary FROM employees WHERE first_name = 'John')
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 5 in 0.01 seconds

| | EMPLOYEE_ID | LAST_NAME | FIRST_NAME |
|---|---|---|---|
| 1 | 110 | Chen | John |
| 2 | 121 | Fripp | Adam |
| 3 | 126 | Mikkilineni | Irene |
| 4 | 139 | Seo | John |
| 5 | 145 | Russell | John |

d) использование многострочного (*multiple-row*) подзапроса с одним столбцом и предикатом **ALL** (при использовании связки с оператором неравенства (**<> ALL**) результат аналогичен работе оператора **IN** с отрицанием **NOT**):

Worksheet Query Builder

```
SELECT employee_id, last_name, first_name
FROM employees
WHERE salary <> ALL (SELECT salary FROM employees WHERE first_name = 'John')
ORDER BY employee_id;
```

Query Result

SQL | Fetched 50 rows in 0.022 seconds

| | EMPLOYEE_ID | LAST_NAME | FIRST_NAME |
|---|---|---|---|
| 1 | 100 | King | Steven |
| 2 | 101 | Kochhar | Neena |
| 3 | 102 | De Haan | Lex |
| 4 | 103 | Hunold | Alexander |
| 5 | 104 | Ernst | Bruce |
| 6 | 105 | Austin | David |
| 7 | 106 | Pataballa | Valli |
| 8 | 107 | Lorentz | Diana |
| 9 | 108 | Greenberg | Nancy |
| 10 | 109 | Faviet | Daniel |

e) использование многостолбцового (*multiple-column*) подзапроса и оператора **EXISTS** (если есть хоть одна строка данных в результате подзапроса, то результат предиката будет TRUE):

Worksheet Query Builder

```
SELECT employee_id, last_name, first_name
FROM employees
WHERE department_id = 60 AND
      EXISTS (SELECT * FROM job_history WHERE department_id = 60)
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 5 in 0.006 seconds

| | EMPLOYEE_ID | LAST_NAME | FIRST_NAME |
|---|---|---|---|
| 1 | 103 | Hunold | Alexander |
| 2 | 104 | Ernst | Bruce |
| 3 | 105 | Austin | David |
| 4 | 106 | Pataballa | Valli |
| 5 | 107 | Lorentz | Diana |

Соотнесенные или коррелированные подзапросы – требуют учета внешних данных и должны заново выполняться для каждой строки-кандидата из внешнего оператора **SELECT**. Пример соотнесенного подзапроса в предложении **WHERE**:

Worksheet Query Builder

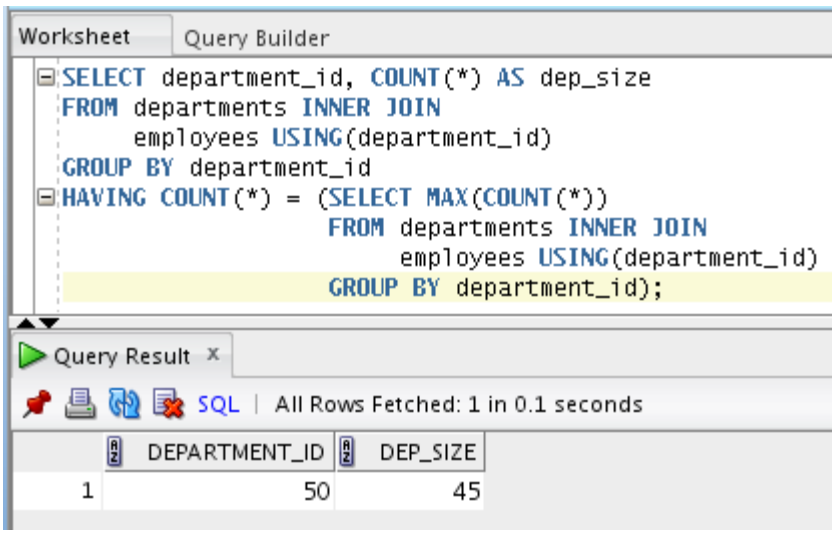
```
SELECT Q.employee_id, Q.last_name, Q.first_name, Q.job_id
FROM employees Q
WHERE salary = (SELECT MAX(SQ.salary) FROM employees SQ
                WHERE SQ.job_id = Q.job_id)
ORDER BY Q.salary DESC;
```

Query Result

SQL | All Rows Fetched: 20 in 0.117 seconds

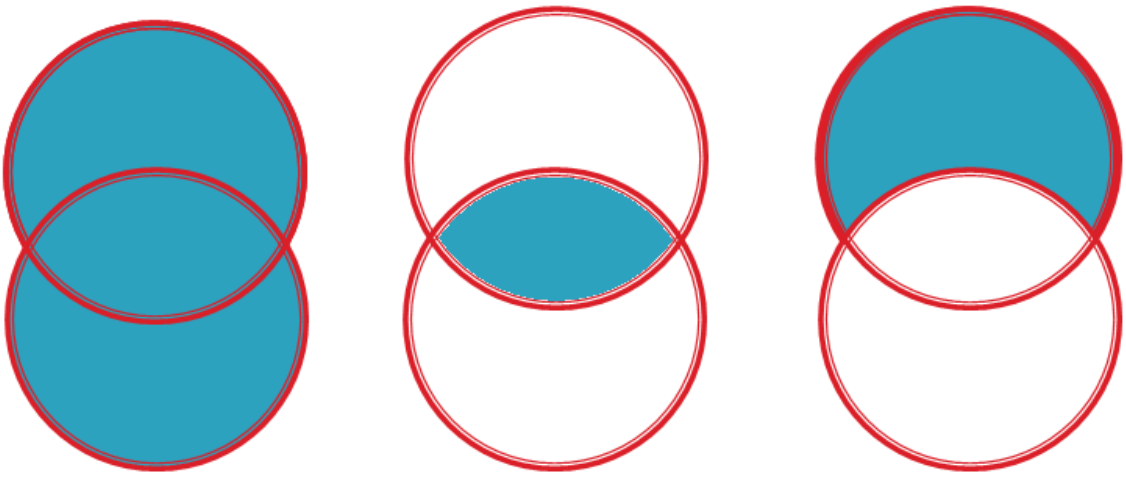
| | EMPLOYEE_ID | LAST_NAME | FIRST_NAME | JOB_ID |
|---|---|---|---|---|
| 1 | 100 | King | Steven | AD_PRES |
| 2 | 101 | Kochhar | Neena | AD_VP |
| 3 | 102 | De Haan | Lex | AD_VP |
| 4 | 145 | Russell | John | SA_MAN |
| 5 | 201 | Hartstein | Michael | MK_MAN |
| 6 | 108 | Greenberg | Nancy | FI_MGR |
| 7 | 205 | Higgins | Shelley | AC_MGR |
| 8 | 168 | Ozer | Lisa | SA_REP |
| 9 | 114 | Raphaely | Den | PU_MAN |
| 10 | 204 | Baer | Hermann | PR_REP |
| 11 | 103 | Hunold | Alexander | IT_PROG |

Использование подзапросов в предложении **HAVING** практически аналогично по вариантам и способам обработки подзапросам в предложении **WHERE**. Пример подзапроса в предложении **HAVING**:




## Операции над множествами

Операции над множествами предназначены для *объединения* результатов одинаковых по своей структуре (заголовкам) в одну общую таблицу (множество) с таким же заголовком:



***R* UNION *S*** ***R* INTERSECT *S*** ***R* MINUS *S***

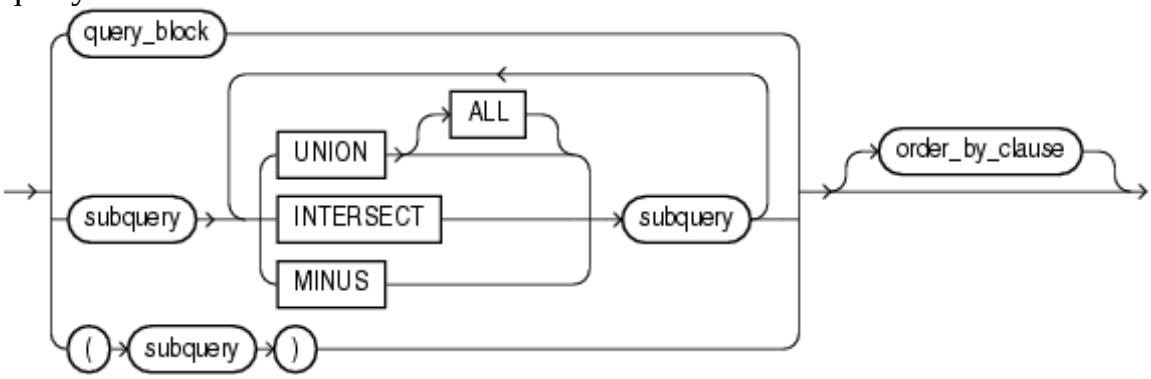
Особенности операций над множествами:

- данные для операций формируются с помощью подзапросов, но при этом подзапросы не обязательно заключать в круглые скобки;
- исходные множества (таблицы) должны иметь одинаковые заголовки, т.е. число столбцов в заголовках должно совпадать и парные столбцы должны иметь совмещенные типы данных, однако парные столбцы НЕ обязательно должны иметь одинаковые имена.
- операции формируют строки результирующей таблицы, число столбцов при этом не изменяется.
- заголовок результирующей таблицы будет иметь имена столбцов по названиям в первом (левом) подзапросе.

- **UNION** – объединение исходных множеств, в результате операции будет получена таблица включающая в себя только различные строки данных из обоих источников;
- **UNION ALL** – объединение исходных множеств, в результате операции будет получена таблица включающая в себя все строки данных из обоих источников, включая и дубликаты;
- **INTERSECT** – пересечение исходных множеств, в результате операции будет получена таблица включающая в себя только те строки, которые присутствуют в обоих исходных множествах;
- **MINUS** – операция вычитания множеств, в результате операции будет получена таблица включающая в себя только те строки, которые присутствуют в первом множестве и отсутствуют во втором, т.е. результат зависит от расположения данных.

Синтаксис операций над множествами описан в диаграмме оператора **SELECT**:

subquery::=




Предложение **ORDER BY** нельзя использовать в подзапросах (*subquery*).

Все примеры работы с операциями над множествами будут приведены для одинаковых исходных множеств (на основе подзапросов для схемы HR):

```
SELECT employee_id AS emp_id_R, job_id FROM employees WHERE department_id = 20;
```

Query Result

SQL | All Rows Fetched: 2 in 0.005 seconds

| | EMP_ID_R | JOB_ID |
|---|---|---|
| 1 | 201 | MK_MAN |
| 2 | 202 | MK_REP |

```
SELECT employee_id AS emp_id_S, job_id FROM job_history WHERE department_id = 20;
```

Query Result

SQL | All Rows Fetched: 1 in 0.007 seconds

| | EMP_ID_S | JOB_ID |
|---|---|---|
| 1 | 201 | MK_REP |

Пример работы операции **UNION**:

Worksheet | Query Builder

```
SELECT employee_id FROM employees WHERE department_id = 20
UNION
SELECT employee_id FROM job_history WHERE department_id = 20
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 2 in 0.031 seconds

| | EMPLOYEE_ID |
|---|---|
| 1 | 201 |
| 2 | 202 |

Пример работы операции **UNION ALL**:

Worksheet | Query Builder

```
SELECT employee_id FROM employees WHERE department_id = 20
UNION ALL
SELECT employee_id FROM job_history WHERE department_id = 20
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 3 in 0.013 seconds

| | EMPLOYEE_ID |
|---|---|
| 1 | 201 |
| 2 | 201 |
| 3 | 202 |

Пример работы операции **INTERSECT**:

Worksheet | Query Builder

```
SELECT employee_id FROM employees WHERE department_id = 20
INTERSECT
SELECT employee_id FROM job_history WHERE department_id = 20
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 1 in 0.017 seconds

| | EMPLOYEE_ID |
|---|---|
| 1 | 201 |

Пример работы операции **MINUS**:

Worksheet | Query Builder

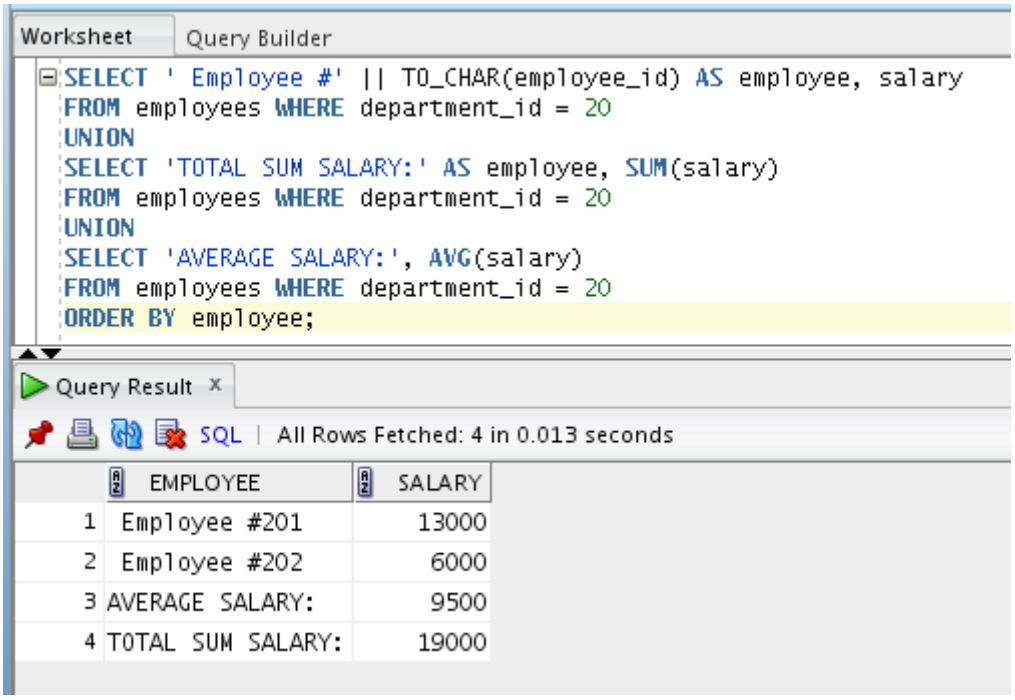
```
SELECT employee_id FROM employees WHERE department_id = 20
MINUS
SELECT employee_id FROM job_history WHERE department_id = 20
ORDER BY employee_id;
```

Query Result

SQL | All Rows Fetched: 1 in 0.015 seconds

| | EMPLOYEE_ID |
|---|---|
| 1 | 202 |

Использование операции **UNION** позволяет получать в результирующей таблице сочетание разных данных (например, реальных данных и статистики по этим данным), но подзапросы должны быть построены таким образом, чтобы быть совместимыми для соединения:

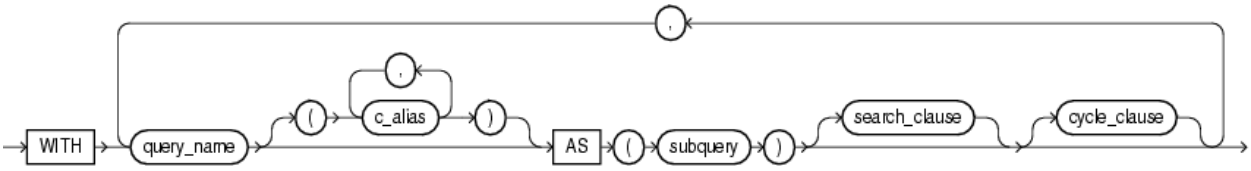



Для упрощения выше приведенного примера, а также повторного использования подзапросов без их копирования, применимо предложение **WITH**:

- предложение **WITH** предназначено для задания имен подзапросам (факторизация запроса); при этом именованные подзапросы фактически выходят на уровень встроенных представлений (*inline view* или временных таблиц);
- подзапросы в предложении **WITH** записываются в круглых скобках, через запятую;
- эти подзапросы можно теперь будет использовать в основном запросе **SELECT** фактически как имя таблицы, причем можно указывать это имя несколько раз;
- остальные части предложения **WITH** – необязательные и используются для построения рекурсивных запросов (*recursive subquery factoring*).

Общий синтаксис предложения **WITH**:

subquery_factoring_clause::=



Пример реализации выше приведенного запроса с использованием предложения **WITH**:

Worksheet | Query Builder

```
WITH
sq_saldep20 AS (SELECT ' Employee #' || TO_CHAR(employee_id) AS employee, salary
                FROM employees WHERE department_id = 20),
sq_sumdep20 AS (SELECT 'TOTAL SUM SALARY:' AS employee, SUM(salary) AS sum_salary
                FROM employees WHERE department_id = 20),
sq_avgdep20 AS (SELECT 'AVERAGE SALARY:' AS employee, AVG(salary) AS avg_salary
                FROM employees WHERE department_id = 20)
SELECT employee, salary FROM sq_saldep20
UNION
SELECT employee, sum_salary FROM sq_sumdep20
UNION
SELECT employee, avg_salary FROM sq_avgdep20
ORDER BY employee;
```

Query Result

SQL | All Rows Fetched: 4 in 0.055 seconds

| | EMPLOYEE | SALARY |
|---|---|---|
| 1 | Employee #201 | 13000 |
| 2 | Employee #202 | 6000 |
| 3 | AVERAGE SALARY: | 9500 |
| 4 | TOTAL SUM SALARY: | 19000 |

## Часть 2. Оформление отчета по лабораторной работе

Для защиты лабораторной работы требуется составить электронный отчет – файл в формате MS Office Word (*.doc(x)*) или Adobe Acrobat Reader (*.pdf*), в который должны войти:

- данные о номере лабораторной работы, ФИО студента, номер группы, название темы (модели), при этом можно не делать отдельный титульный лист;
- решение заданий по вашей схеме для каждого задания в виде:
  - текста условия задания;
  - текста SQL-запроса;
  - скриншота, показывающего результат работы запроса, на котором видно число выбранных строк.
- если какое либо задание не выполнено, то требуется указать его номер и краткое описание проблем.

Отчет сохранить в файле с названием «*ФИО* БД ЛР5 *ГРУППА* 2019», где:

- *ФИО* – ваша фамилия, имя и отчество (можно «*Фамилия Имя*» или «Фамилия», но так чтобы не попасть на однофамильцев в группе);
- *ГРУППА* – номер группы;
- пробелы допустимо заменить символом подчеркивания.

После защиты отчета по лабораторной работе у преподавателя, файл отчета переслать на почту kalabukhov@bsuir.by, тема письма «БД ЛР5», тело письма может быть пустым, но не забыть прикрепить к письму файл отчета.