

Лабораторная работа №4

Часть 1. Реализация SQL-запросов на простую выборку данных

В лабораторной работе выполняется создание простых запросов на выборку данных на языке SQL с использованием предложений **SELECT**, **FROM** (**JOINS**), **WHERE** и **ORDER BY** оператора **SELECT**. В работе также требуется рассмотреть использование скалярных функций.

Всего: 15. Минимум два на каждый.

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 выборки, в котором НЕЛЬЗЯ использовать подзапросы и группировку данных (это еще будет в другой лабораторной работе);
- при использовании соединений нескольких таблиц обратить внимание на условие задания и сделать выбор между внутренним и внешним соединениями и их вариантами реализации;
- перед запуском запроса на выполнение, изучить данные в используемых запросом таблицах, и, если требуется, добавить в вашу схему необходимые новые данные, чтобы результат выборки был контролируемым и не пустым;
- выполнить запрос и проанализировать его результат – если есть расхождения между ожидаемыми данными и результатом запроса, то есть повод задуматься о проверке правильности выполнения этого задания.

3) Оформить *отчет*.

Оператор **SELECT**

Оператор **SELECT** выполняет выборку данных из таблиц базы данных и представляет результаты в виде одной финальной таблицы. Так как этот оператор относится к DML, то при его работе данные в таблицах не изменяются.

Чтобы использовать оператор **SELECT** для создания запросов необходимо:

- изучить структуру таблиц схемы данных (названия таблиц, названия столбцов и их типы данных, особенности формирования данных столбцов (ограничения));
- изучить связи между таблицами (ссылки и их смысл для соединения данных);
- учесть изменение данных таблиц во времени.

Для проверки работоспособности созданного запроса на выборку требуется:

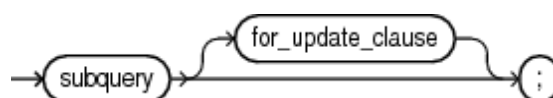
- чтобы этот запрос возвращал не пустые данные;
- проверить возвращаемые данные с помощью анализа данных таблиц.

Предложения оператора **SELECT**, используемые в этой лабораторной работе:

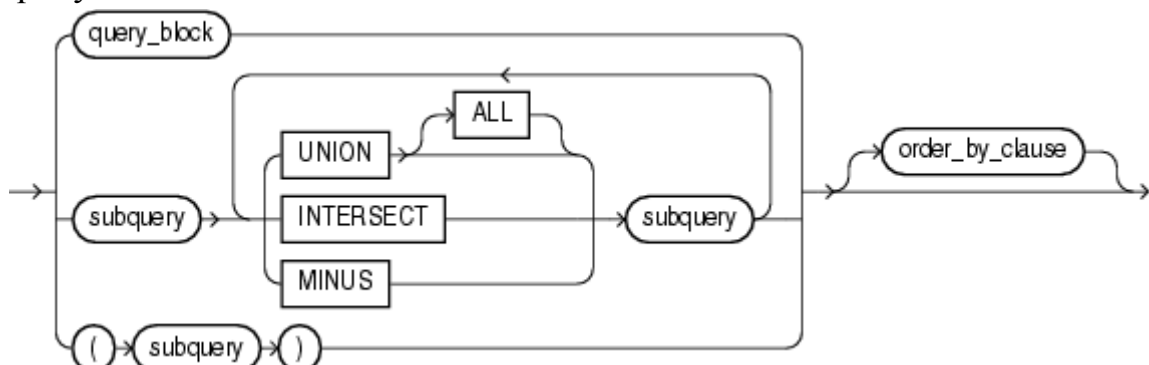
Предложение	Обязательное?	Назначение
SELECT	Да	Описание списка выборки – набора элементов разделенных запятыми: столбцов, констант, функций, выражений на их основе и т.п.
FROM	Да	Описание источника данных: набора таблиц, представлений, подзапросов и требуемых операций по их соединению.
WHERE	Нет	Описание фильтра строк источника данных по заданному логическому условию (предикату).
ORDER BY	Нет	Описание порядка сортировки строк результирующей таблицы.

Синтаксическая конструкция основных частей оператора SELECT приведена ниже.

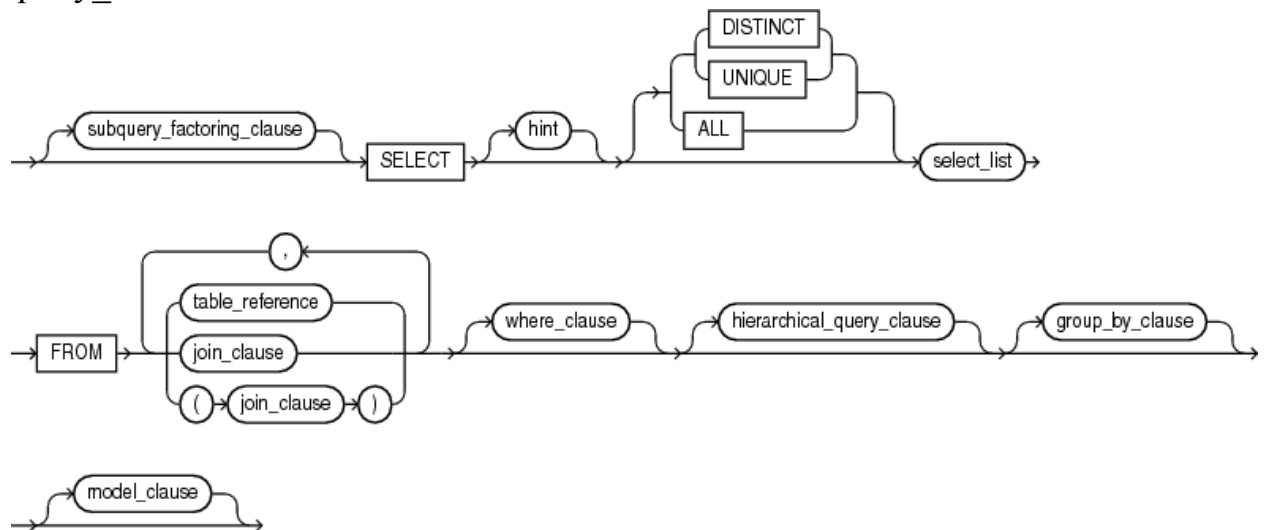
select::=



subquery::=

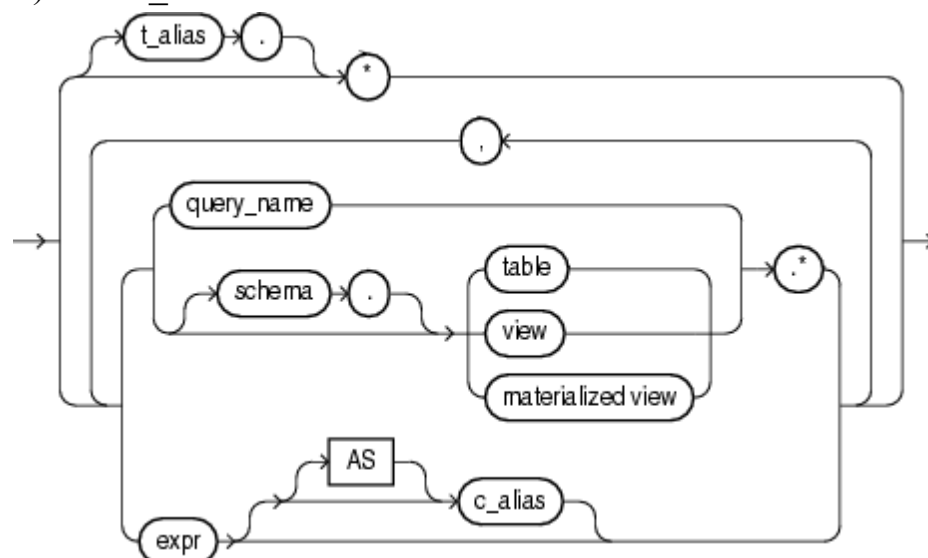


query_block::=



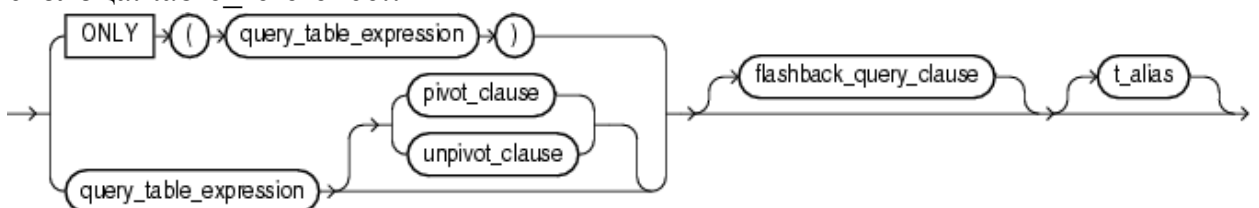
- *hint* – комментарий с инструкциями для оптимизатора запросов;
- **DISTINCT** или **UNIQUE** – запрет дублирования результирующих строк;
- **ALL** – разрешение вывода дубликатов строк (по

умолчанию). select_list::=

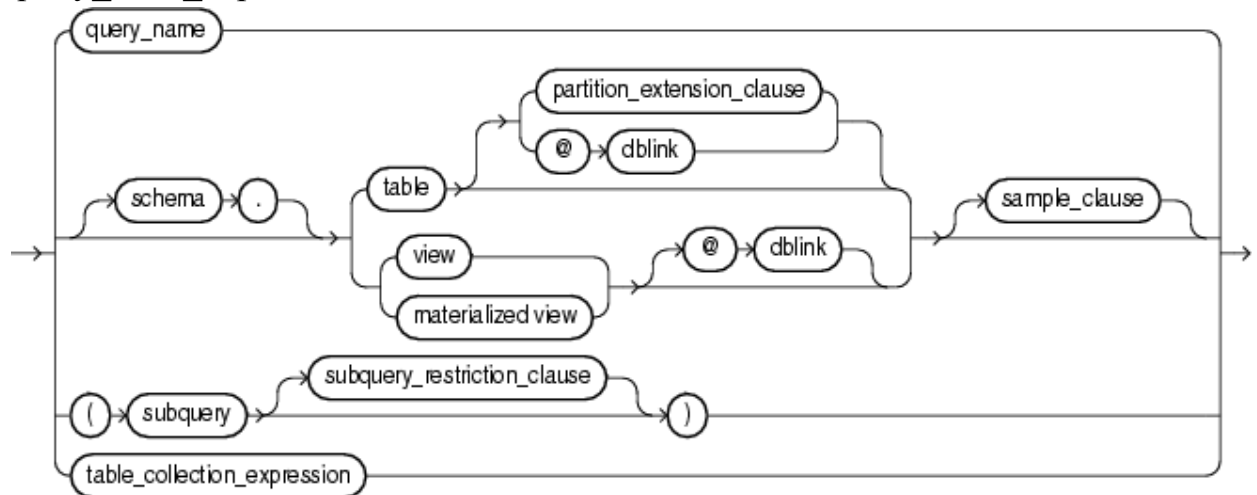


- * – выбор всех столбцов;
- *expr* – имя столбца из источника данных, константа, выражения с ними и т.п.;
- *c_alias* – новое название

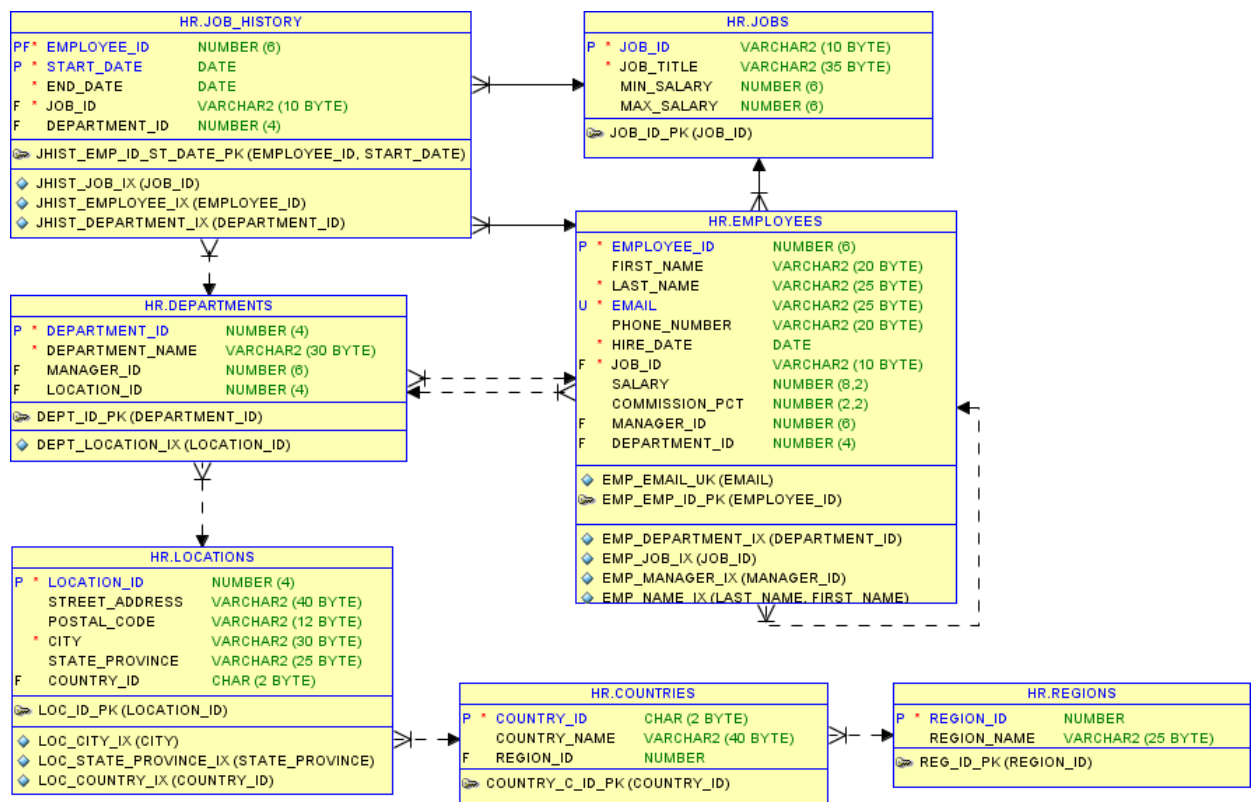
столбца. table_reference::=



query_table_expression::=



Примеры, приводимые ниже, используют готовую учебную схему данных HR (СУБД ORACLE 11g):



Ниже приведен пример простого однотабличного SQL-запроса на выборку данных для схемы данных HR и результат его выполнения в Oracle SQL Developer:

The screenshot shows an SQL Worksheet with a query and its results. The query is as follows:

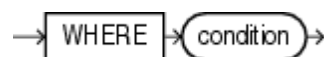
```
-- simple query example
/*
  other comment for query
*/
SELECT emps.first_name,
       emps.last_name,
       emps.salary,
       '$ <==> BY', -- const column
       9730,         -- const column
       emps.salary * 9730 AS "Employee Salary BY" -- column alias
FROM HR.employees emps; -- table alias
```

The results are displayed in a table with 7 rows and 6 columns:

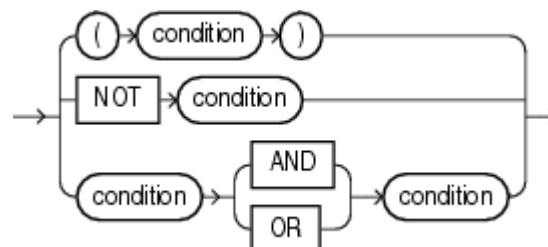
	FIRST_NAME	LAST_NAME	SALARY	'\$ <==> BY'	9730	Employee Salary BY
1	Steven	King	24000	\$ <==> BY	9730	233520000
2	Neena	Kochhar	17000	\$ <==> BY	9730	165410000
3	Lex	De Haan	17000	\$ <==> BY	9730	165410000
4	Alexander	Hunold	9000	\$ <==> BY	9730	87570000
5	Bruce	Ernst	6000	\$ <==> BY	9730	58380000
6	David	Austin	4800	\$ <==> BY	9730	46704000
7	Valli	Pataballa	4800	\$ <==> BY	9730	46704000

Предложение **WHERE** позволяет выполнять операции выборки строк данных, удовлетворяющих условию (предикату) *condition*.

where_clause ::=



compound_condition ::=



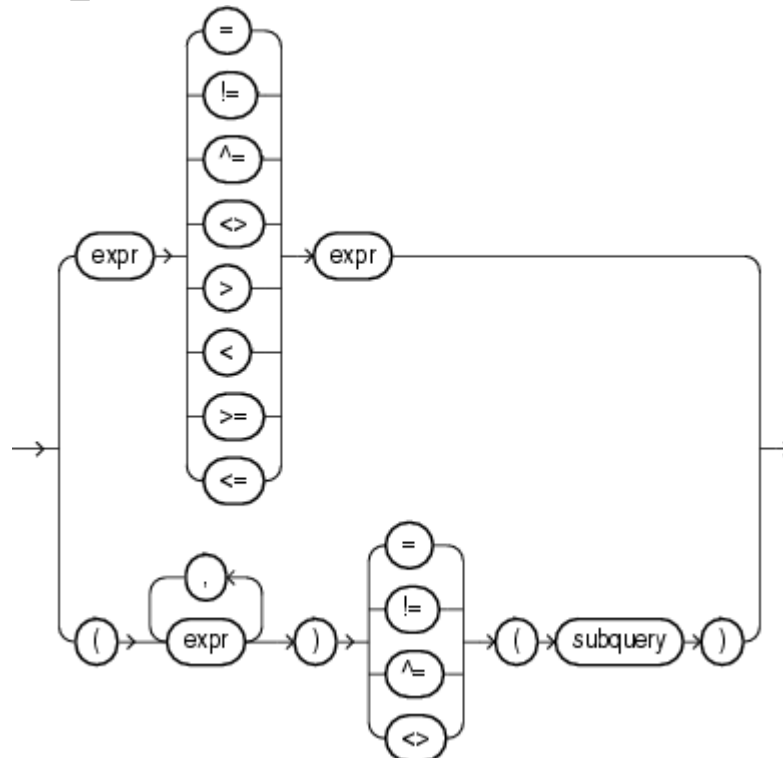
Для составных условий следует учитывать трехзначную логику сравнения:

--	TRUE	FALSE	UNKNOWN
NOT	FALSE	TRUE	UNKNOWN

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

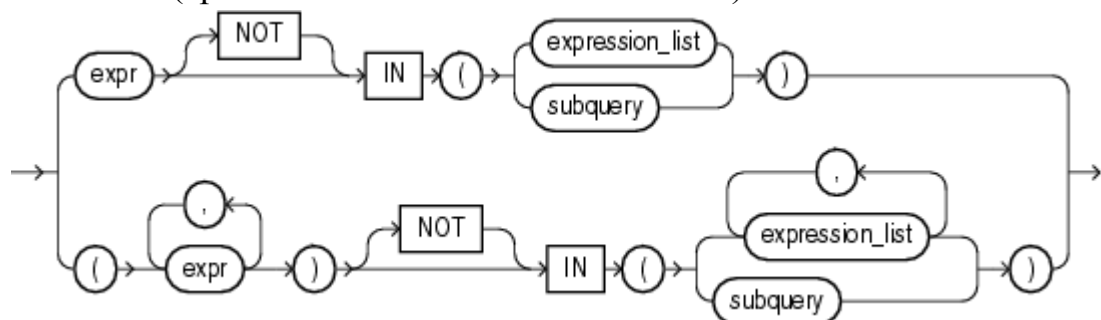
simple_comparison_condition::=



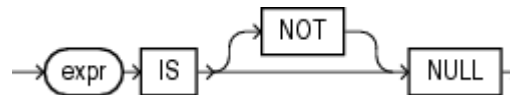
between_condition::= (сравнение с диапазоном значений)



in_condition::= (сравнение с множеством значений)

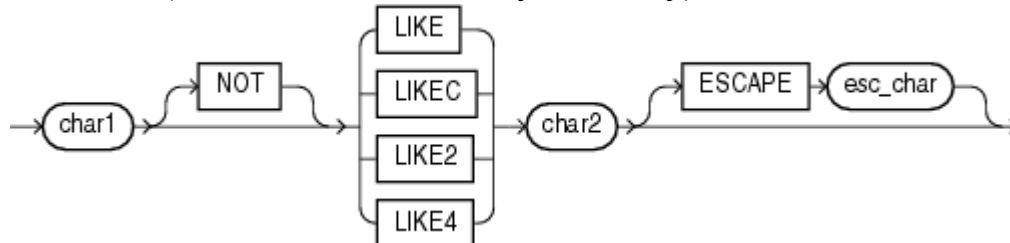


null_condition ::= (сравнение с **NULL**)



- обычные условия сравнения с **NULL** всегда дают неопределенный логический результат (UNKNOWN)

like_condition ::= (поиск по символьному шаблону)



- *char1* – столбец с символьными данными для поиска;
- *char2* – строка-шаблон для поиска;
- **LIKE** – поиск строки по заданному паттерну с использованием спецсимволов:
 - символ процента (%) – любая последовательность символов,
 - символ подчеркивания () – любой одиночный символ;
- *esc_char* – escape символ – для возможности поиска в строке спецсимволов;
- **LIKEC**, **LIKE2**, **LIKE4** – разные варианты сравнения для Unicode символов: Unicode, UCS-2 и UCS-4.

Ниже приведены примеры оператора **SELECT** с предложением **WHERE**:

SQL Worksheet: HR_test.sql x EMPLOYEES x

Worksheet: Query Builder

```
-- WHERE example #1
SELECT emps.employee_id,
       emps.last_name AS "LAST",
       emps.first_name AS "FIRST",
       emps.salary
FROM HR.employees emps
WHERE (emps.salary BETWEEN 12000 AND 17000) AND
      emps.last_name LIKE '%rt%' AND
      emps.commission_pct IS NOT NULL;
```

Query Result x

SQL | All Rows Fetched: 1 in 0.008 seconds

EMPLOYEE_ID	LAST	FIRST	SALARY
1	146 Partners	Karen	13500

SQL Worksheet: HR_test.sql x EMPLOYEES x

Worksheet: Query Builder

```
-- WHERE example #2
SELECT emps.employee_id,
       emps.last_name AS "LAST",
       emps.first_name AS "FIRST",
       emps.salary
FROM HR.employees emps
WHERE emps.employee_id IN (100, 120, 200);
```

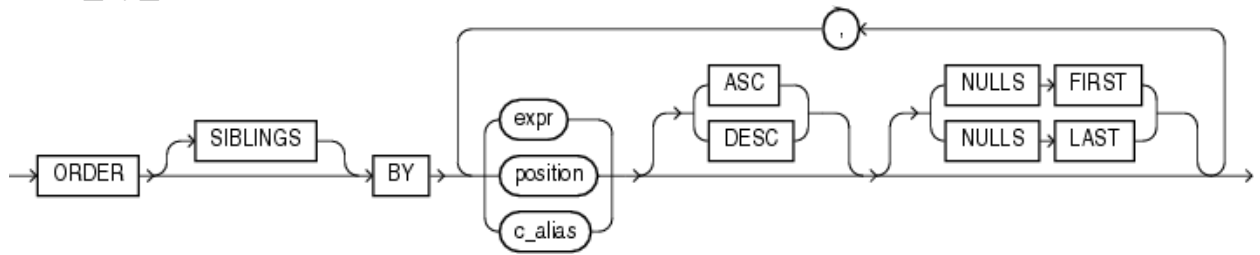
Query Result x

SQL | All Rows Fetched: 3 in 0.008 seconds

EMPLOYEE_ID	LAST	FIRST	SALARY
1	100 King	Steven	24000
2	120 Weiss	Matthew	8000
3	200 Whalen	Jennifer	4400

Предложение **ORDER BY** позволяет выполнять сортировку строк результирующей таблицы.

`order_by_clause ::=`



- *expr* – имена столбцов результирующей таблицы и выражения на их основе;
- *position* – целочисленный индекс положения столбца в результирующей таблице;
- *c_alias* – ранее заданное новое имя столбца;
- **ASC** – сортировка данных столбца по возрастанию (по умолчанию);
- **DESC** – сортировка данных столбца по убыванию;
- **NULLS FIRST** | **NULLS LAST** – порядок расположения значений **NULL** при сортировке данных столбца.

Ниже приведены примеры оператора **SELECT** с предложением **ORDER BY**:

SQL Worksheet: HR_test.sql

```
-- ORDER BY example #1
SELECT emps.employee_id,
       emps.last_name AS "LAST",
       emps.first_name AS "FIRST"
FROM HR.employees emps
ORDER BY 2 ASC, "FIRST" DESC;
```

Query Result: Fetched 50 rows in 0.053 seconds

EMPLOYEE_ID	LAST	FIRST
10	Bernstein	David
11	Bissot	Laura
12	Bloom	Harrison
13	Bull	Alexis
14	Cabrio	Anthony
15	Cambrault	Nanette
16	Cambrault	Gerald
17	Chen	John
18	Chung	Kelly
19	Colmenares	Karen
20	Coolidge	Curtis

SQL Worksheet: HR_test.sql

```
-- ORDER BY example #2
SELECT emps.employee_id,
       emps.last_name AS "LAST",
       emps.first_name AS "FIRST",
       emps.salary
FROM HR.employees emps
ORDER BY emps.salary DESC, 1;
```

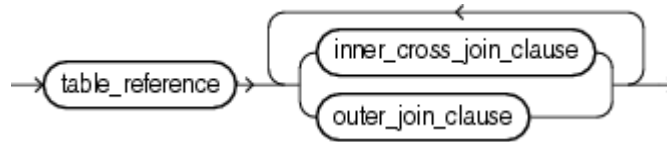
Query Result: Fetched 50 rows in 0.012 seconds

EMPLOYEE_ID	LAST	FIRST	SALARY
1	King	Steven	24000
2	Kochhar	Neena	17000
3	De Haan	Lex	17000
4	Russell	John	14000
5	Partners	Karen	13500
6	Hartstein	Michael	13000
7	Greenberg	Nancy	12000
8	Errazuriz	Alberto	12000
9	Higgins	Shelley	12000
10	Ozer	Lisa	11500

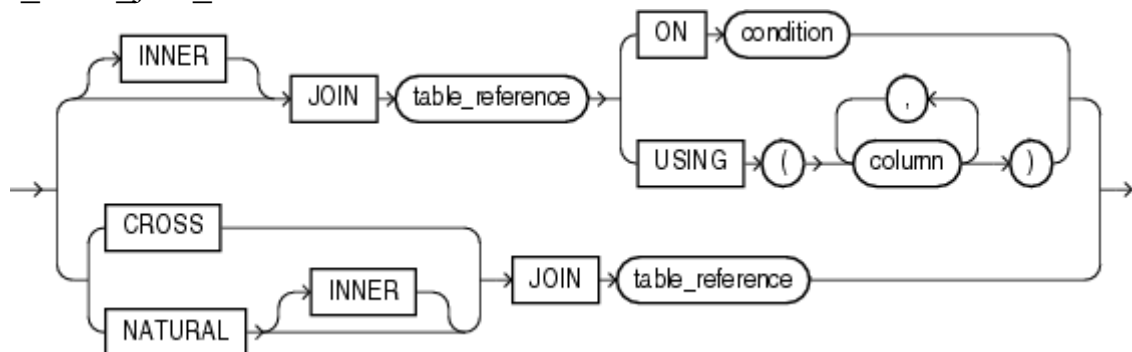
Соединение (*join*) – операция позволяющая формировать одну общую таблицу данных из нескольких таблиц.

В основе соединения лежит операция *декартова произведения* отношений, которая позволяет получить все возможные сочетания кортежей двух реляционных отношений.

join_clause ::=



inner_cross_join_clause ::=



Декартово произведение (**CROSS JOIN**) дает все возможные сочетания строк данных двух таблиц:

SQL Worksheet: HR_test.sql

```
-- CARTESIAN PRODUCT example #1
SELECT R.last_name as RNAME,
       S.last_name as SNAME
FROM HR.employees R, HR.employees S;
```

Script Output: All Rows Fetched: 11449 in 2.069 seconds

RNAME	SNAME
11438 Zlotkey	Taylor
11439 Zlotkey	Tobias
11440 Zlotkey	Tucker
11441 Zlotkey	Tuvault
11442 Zlotkey	Urman
11443 Zlotkey	Vargas
11444 Zlotkey	Vishney
11445 Zlotkey	Vollman
11446 Zlotkey	Walsh
11447 Zlotkey	Weiss
11448 Zlotkey	Whalen
11449 Zlotkey	Zlotkey

SQL Worksheet: HR_test.sql

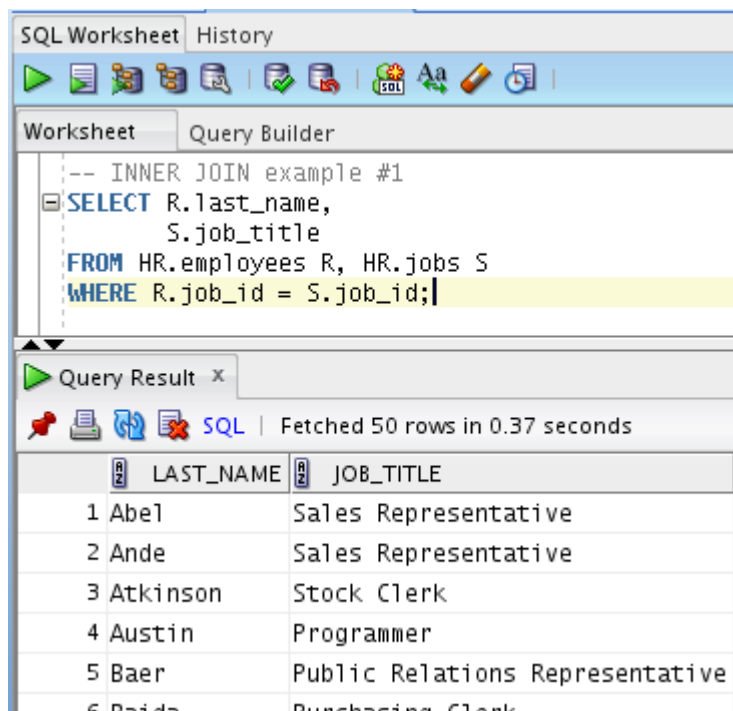
```
-- CARTESIAN PRODUCT example #2
SELECT R.last_name as RNAME,
       S.last_name as SNAME
FROM HR.employees R CROSS JOIN HR.employees S;
```

Script Output: Query Result

RNAME	SNAME
11438 Zlotkey	Taylor
11439 Zlotkey	Tobias
11440 Zlotkey	Tucker
11441 Zlotkey	Tuvault
11442 Zlotkey	Urman
11443 Zlotkey	Vargas
11444 Zlotkey	Vishney
11445 Zlotkey	Vollman
11446 Zlotkey	Walsh
11447 Zlotkey	Weiss
11448 Zlotkey	Whalen
11449 Zlotkey	Zlotkey

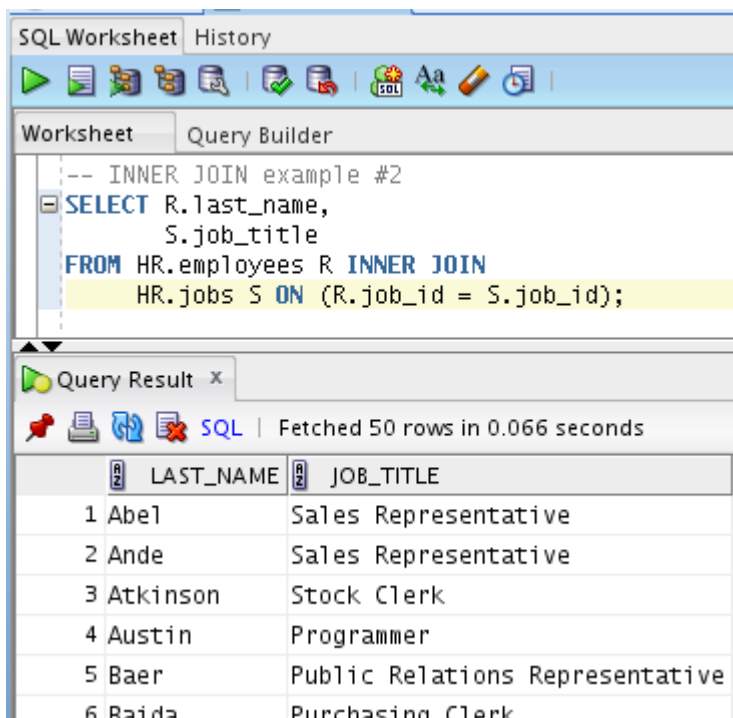
Наиболее часто используется внутреннее соединение (**INNER JOIN**), с использованием оператора равенства (=), или соединение по эквивалентности, которое позволяет «собирать» воедино данные связанные ссылками (FK = PK).

Ниже приведены примеры такого вида соединения, позволяющие получить в результирующей таблице данные сотрудника и его должности, которые в базе данных расположены в разных таблицах:



```
-- INNER JOIN example #1
SELECT R.last_name,
       S.job_title
FROM HR.employees R, HR.jobs S
WHERE R.job_id = S.job_id;
```

	LAST_NAME	JOB_TITLE
1	Abel	Sales Representative
2	Ande	Sales Representative
3	Atkinson	Stock Clerk
4	Austin	Programmer
5	Baer	Public Relations Representative
6	Raida	Purchasing Clerk

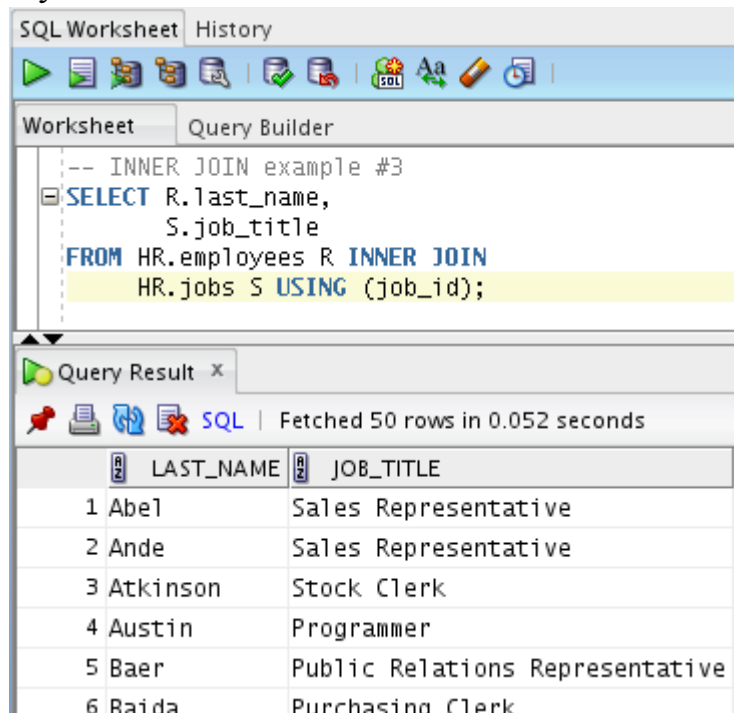


```
-- INNER JOIN example #2
SELECT R.last_name,
       S.job_title
FROM HR.employees R INNER JOIN
     HR.jobs S ON (R.job_id = S.job_id);
```

	LAST_NAME	JOB_TITLE
1	Abel	Sales Representative
2	Ande	Sales Representative
3	Atkinson	Stock Clerk
4	Austin	Programmer
5	Baer	Public Relations Representative
6	Raida	Purchasing Clerk

Если поля FK, и связанного с ним, PK названы одинаково, то можно

использовать следующий синтаксис:

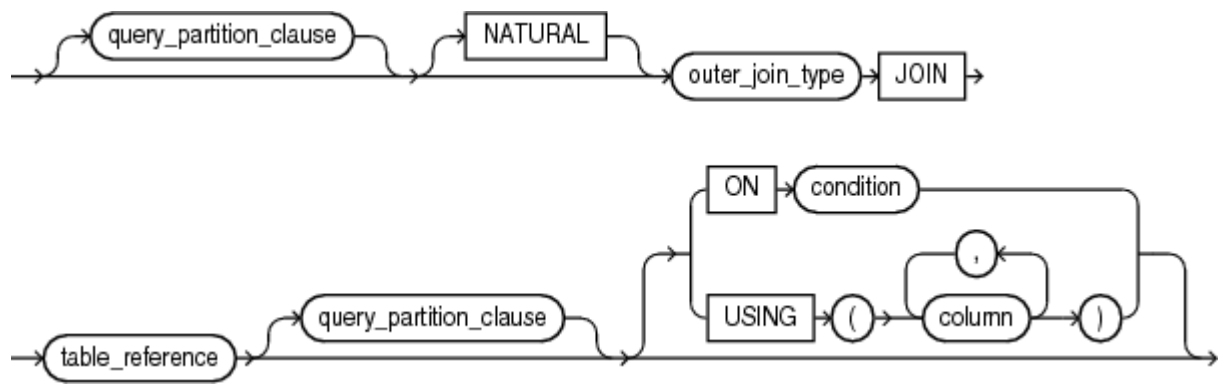


Внешние соединения (**OUTER JOIN**) сохраняют кортежи, которые были бы утрачены при использовании внутреннего соединения.

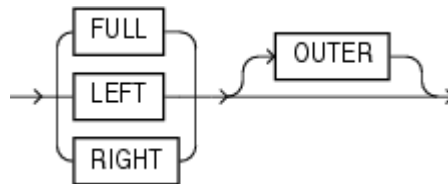
Если есть соединение двух таблиц вида $R \text{ join } S$, то:

- *левое внешнее соединение* (**LEFT OUTER JOIN**) – соединение по эквивалентности, в результат которого дополнительно включены кортежи отношения R не имеющие совпадающих значений в столбцах, по которым выполняется соединение, в отношении S (отсутствующие значения из отношения S заменяются на NULL);
- *правое внешнее соединение* (**RIGHT OUTER JOIN**) – соединение по эквивалентности, в результат которого дополнительно включены кортежи отношения S не имеющие совпадающих значений в столбцах, по которым выполняется соединение, в отношении R (отсутствующие значения из отношения R заменяются на NULL);
- *полное внешнее соединение* (**FULL OUTER JOIN**) – объединяет результаты, как левого, так и правого внешних соединений.

outer_join_clause::=



outer_join_type::=



Пример левого и правого внешнего соединения приведен ниже:

SQL Worksheet History

Worksheet Query Builder

```
-- LEFT OUTER JOIN example #1
SELECT R.department_name,
       S.last_name
FROM HR.departments R LEFT OUTER JOIN
     HR.employees S ON (R.department_id = S.department_id);
```

Query Result x

SQL | All Rows Fetched: 122 in 0.113 seconds

	DEPARTMENT_NAME	LAST_NAME
103	Finance	Sciarra
104	Finance	Greenberg
105	Accounting	Gietz
106	Accounting	Higgins
107	Treasury	(null)
108	Corporate Tax	(null)
109	Control And Credit	(null)
110	Shareholder Serv...	(null)

SQL Worksheet History

Worksheet Query Builder

```
-- RIGHT OUTER JOIN example #1
SELECT R.department_name,
       S.last_name
FROM HR.departments R RIGHT OUTER JOIN
     HR.employees S ON (R.department_id = S.department_id);
```

Query Result x

SQL | Fetching next 50 rows in 0.084 seconds

	DEPARTMENT_NAME	LAST_NAME
76	Sales	Hutton
77	Sales	Taylor
78	Sales	Livingston
79	(null)	Grant
80	Sales	Johnson
81	Shipping	Taylor
82	Shipping	Fleaur

Пример полного внешнего соединения:

The screenshot shows an SQL Worksheet interface. The top toolbar includes icons for running queries, saving, and editing. Below the toolbar, the 'Worksheet' tab is active, displaying the following SQL query:

```
-- FULL OUTER JOIN example
SELECT R.department_name,
       S.last_name
FROM   HR.departments R FULL OUTER JOIN
       HR.employees S ON (R.department_id = S.department_id)
WHERE  R.department_name IS NULL OR
       S.last_name IS NULL;
```

The 'Query Result' tab is also visible, showing the results of the query. The status bar indicates 'All Rows Fetched: 17 in 0.027 seconds'. The results are displayed in a table with two columns: DEPARTMENT_NAME and LAST_NAME.

	DEPARTMENT_NAME	LAST_NAME
1	(null)	Grant
2	NOC	(null)
3	Manufacturing	(null)
4	Government Sales	(null)
5	IT Support	(null)
6	Benefits	(null)
7	Shareholder Services	(null)
8	Retail Sales	(null)
9	Control And Credit	(null)
10	Recruiting	(null)

Скалярные функции

Скалярные (*single row*) функции - функции, которые выполняются для каждой строки данных таблицы и формируют одно значение.

Скалярные функции могут применяться в списке выборки и предикатах фильтров оператора SELECT.

Ниже приведены (справочно) группы функций, которые желательно самостоятельно изучить и использовать в данной работе:

- функции для работы с числами: **ROUND (number)** , **TRUNC (number)** ;
- функции для работы со строками (возвращают символьный результат): **INITCAP**, **TRANSLATE**, **TRIM**, **CONCAT**, **LOWER**, **SUBSTR**, **UPPER**;
- функции для работы со строками (возвращают числовой результат): **INSTR**, **LENGTH**;
- функции для работы с временными типами данных: **ADD_MONTHS**, **LAST_DAY**, **MONTHS_BETWEEN**, **SYSDATE**, **TO_CHAR (datetime)** ;

- функции для преобразования данных: **TO_CHAR(datetime)**, **TO_CHAR(number)**, **TO_CHAR(number)**;
- функции кодирования и декодирования: **DECODE**;
- функции для работы с *NULL*: **NVL**, **NVL2**.

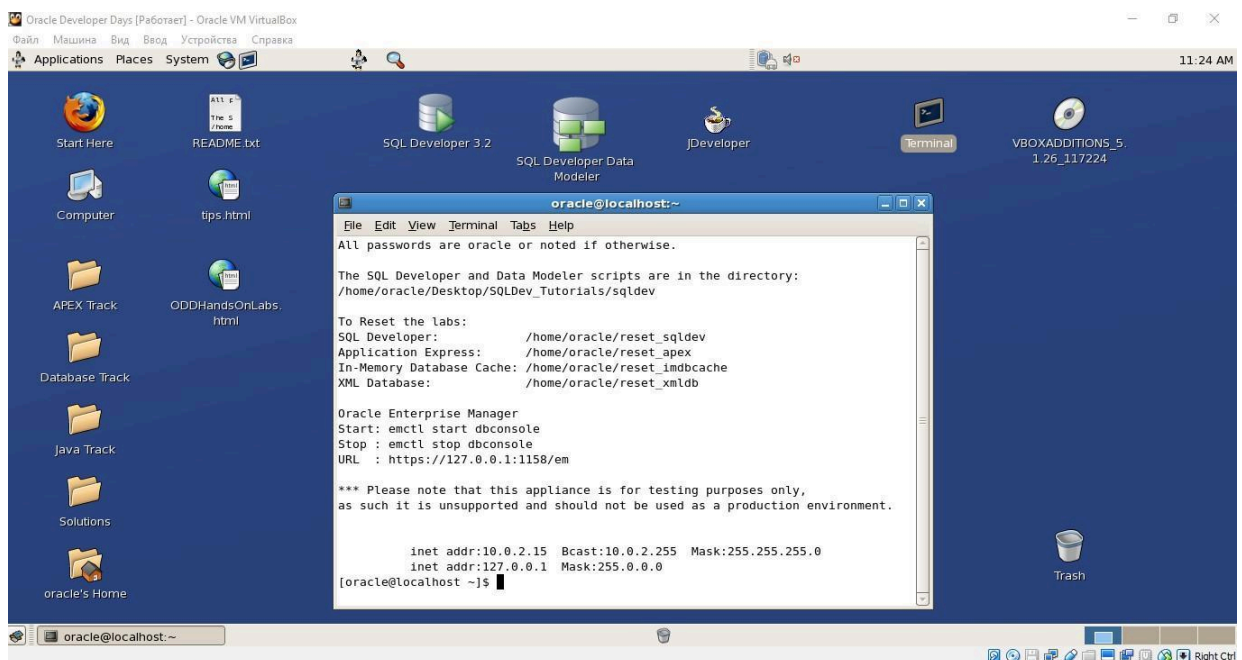
Часть 2. Использование Oracle SQL Developer

В этой части приводятся рекомендации по использованию Oracle SQL Developer для решения ряда задач данной лабораторной работы.

Порядок выполнения работы

1) Запустить виртуальную машину для практической работы:

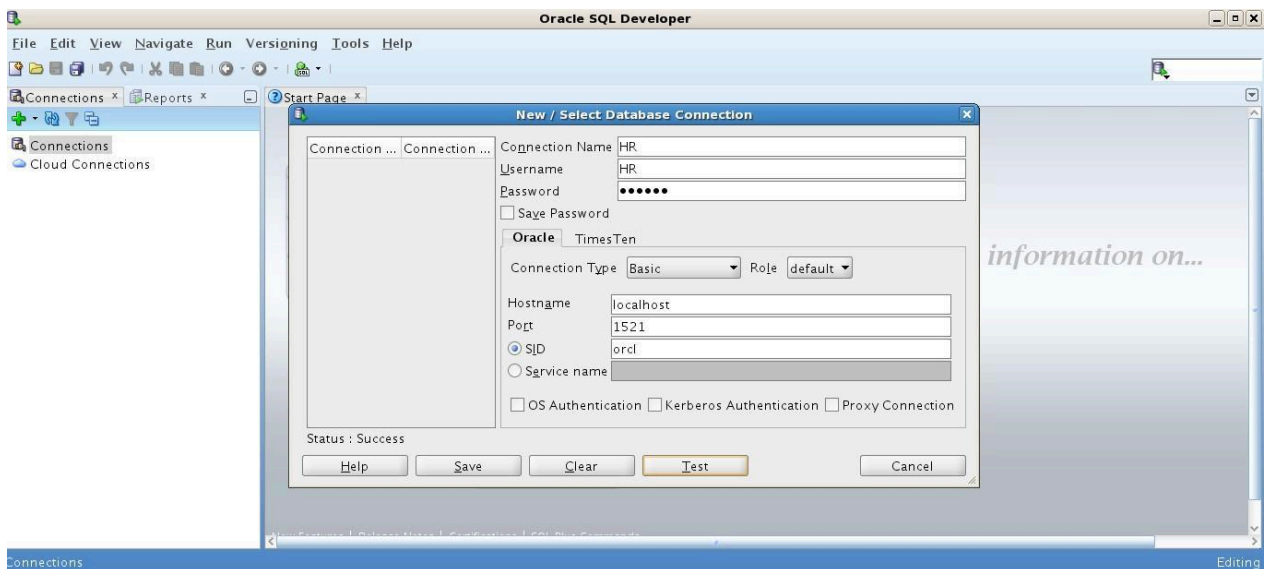
- Войти под рабочим пользователем:
 - o username: oracle
 - o password: oracle



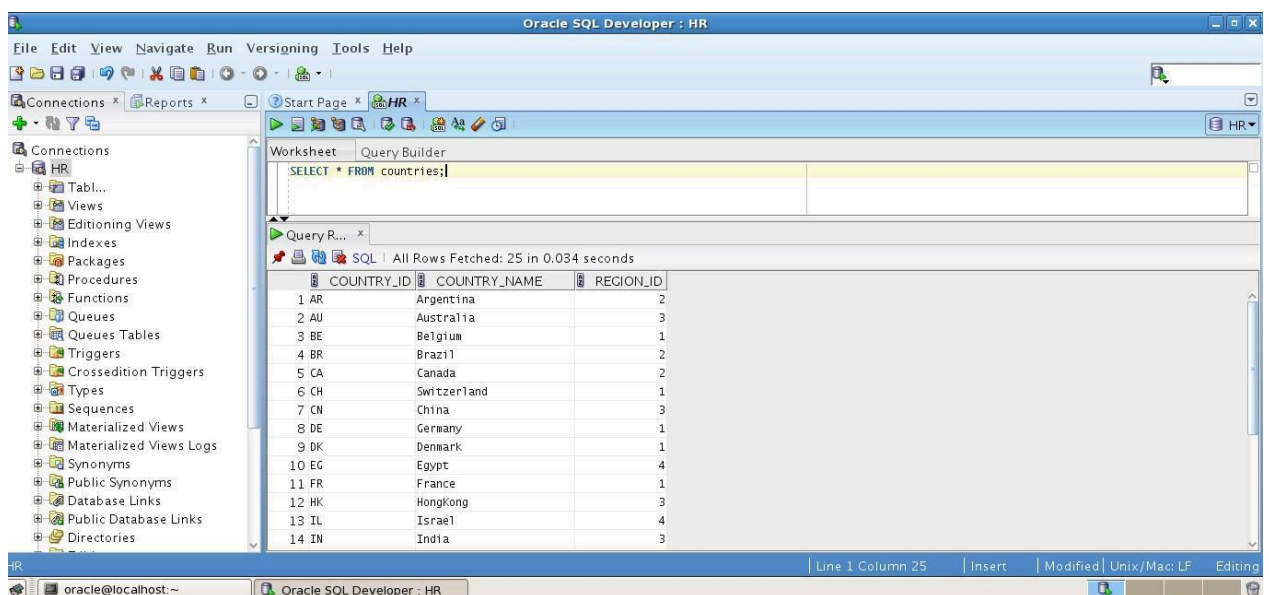
2) Пример работы с SQL Developer по созданию запросов на выборку данных в схеме HR:

- Запустить SQL Developer;
- Создать соединение со схемой HR (правой кнопкой мыши на Connections → New Connection или на кнопку «+»):
 - o username: HR
 - o password: oracle
 - o SID: orcl

(желательно выполнить сначала Test, чтобы проверить правильность заданных параметров, и только потом Save).



- Подключиться к схеме HR (правая кнопка мыши на HR → Connect) и протестировать доступ к данным простым запросом в закладке Worksheet (например, **SELECT * FROM countries;**).



- Теперь можно создавать запросы к схеме HR по полученному заданию:
 - все запросы можно создать в закладке Worksheet (текст запросов можно сохранить в файл с помощью File → Save (Save As...) или скопировать через буфер обмена);
 - для запуска запросов поодиночке можно использовать пиктограммы (признак окончания оператора – знак ';' '):
 - Run Statement – выполнение выделенного (позиция курсора в операторе или выделенный фрагмент текста) оператора – результат в виде таблицы;
 - Run Script – выполнение выделенного (только выделенный фрагмент текста) оператора – результат в виде консоли;
 - использование закладки Query Builder – работа с конструктором

запросов – в лабораторной работе не предусмотрена, но ее можно использовать, чтобы получить предварительный оператор на языке SQL.

3) Для создания запросов к собственной схеме необходимо сначала подключиться к ней (аналогично выше приведенному пункту 2) и только потом создавать запросы.