Lab 2: Review of SQL Queries

(Source: sqltutorial.org)

HOFSTRA UNIVERSITY®
FRANK G. ZARB SCHOOL OF BUSINESS

Submission instructions:

- This lab assignment must be electronically submitted through Blackboard (you can find the link under Assignments). Please DO NOT send them via email.
- Submit ONLY ONE Word file named according to the following format:

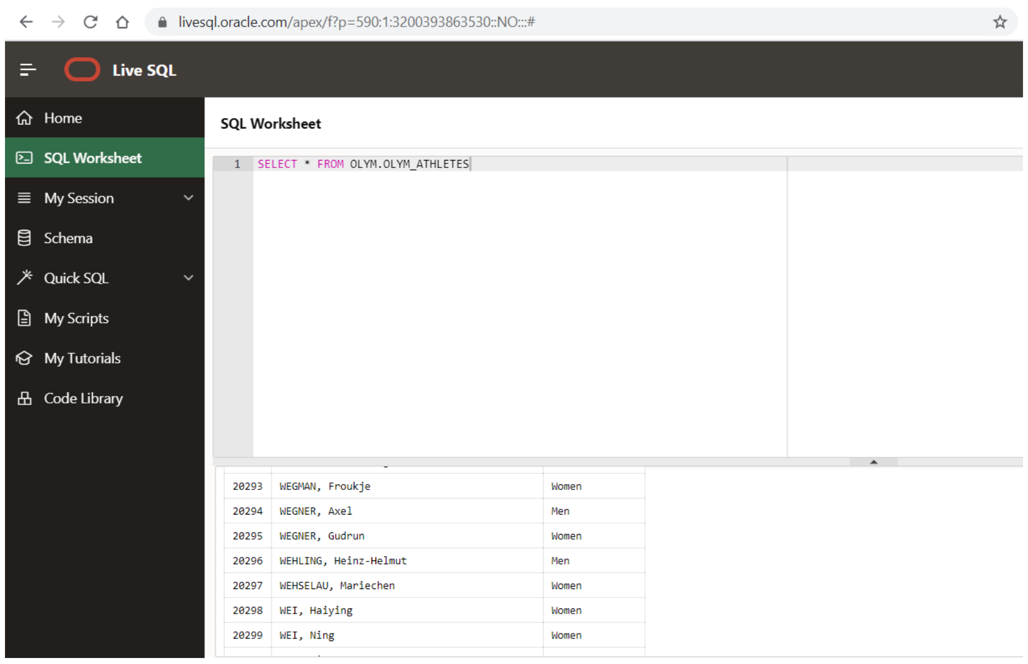
  FirstName_LastName_Lab2.doc (e.g. amir_gandomi_Lab2.doc)

- Lab assignments are to be completed individually. Plagiarism in any form will be treated as an academic misconduct. For more information about academic integrity, please refer to "Policy on Academic Honesty" described in the course outline.

Lab instructions:

The purpose of this assignment is to review the process of developing simple SQL queries to extract data from a relational database. To this end, you will use Oracle's *Live SQL* application, which is a free online fully-functioning database tool. Live SQL is pre-populated with schema that you can use to practice SQL queries.

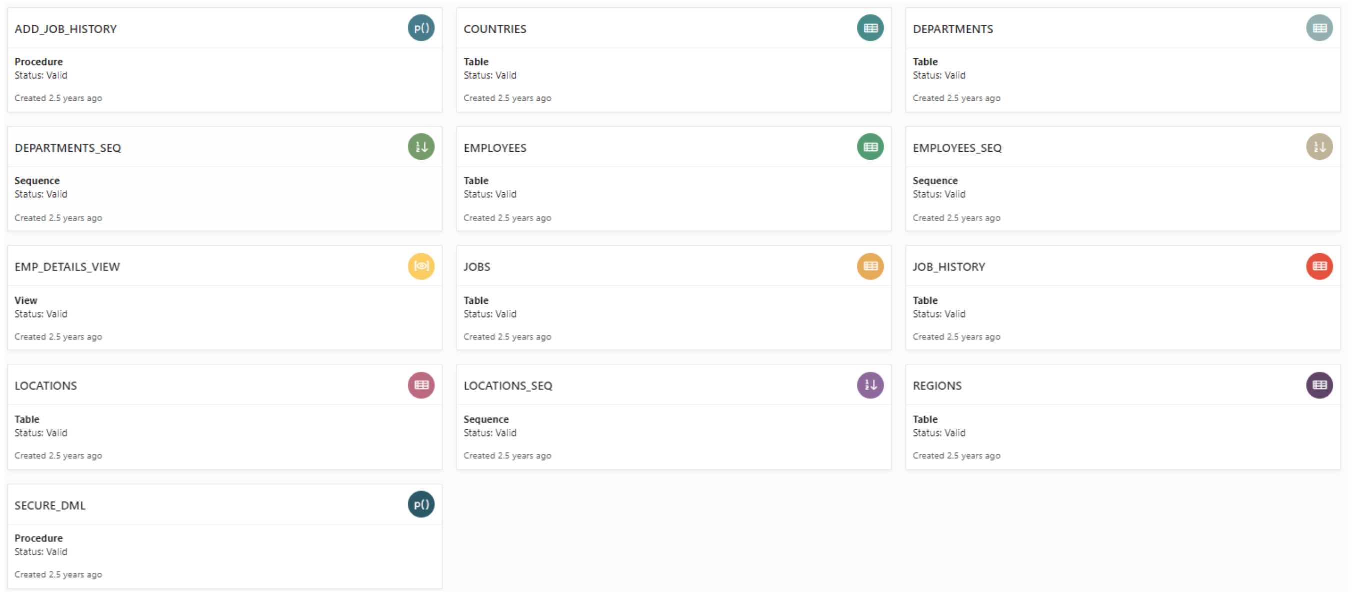
First, click here to access Live SQL and then click Start Coding Now. You will be promoted to create a free Oracle account. Follow the steps to create and log in to your account. You will use SQL Worksheet to create and execute queries as shown below.



```
SELECT * FROM OLYM.OLYM_ATHLETES
```

```
20293   WEGMAN, Froukje          Women
20294   WEGNER, Axel             Men
20295   WEGNER, Gudrun           Women
20296   WEHLING, Heinz-Helmut    Men
20297   WEHSELAU, Mariechen      Women
20298   WEI, Haiying             Women
20299   WEI, Ning                Women
```

Pre-populated databases are available through Schema link in the left panel. We will use the Human Resources (HR), Oracle's HR sample schema with the following tables:


ADD_JOB_HISTORY
Procedure
Status: Valid
Created 2.5 years ago

COUNTRIES
Table
Status: Valid
Created 2.5 years ago

DEPARTMENTS
Table
Status: Valid
Created 2.5 years ago

DEPARTMENTS_SEQ
Sequence
Status: Valid
Created 2.5 years ago

EMPLOYEES
Table
Status: Valid
Created 2.5 years ago

EMPLOYEES_SEQ
Sequence
Status: Valid
Created 2.5 years ago

EMP_DETAILS_VIEW
View
Status: Valid
Created 2.5 years ago

JOBS
Table
Status: Valid
Created 2.5 years ago

JOB_HISTORY
Table
Status: Valid
Created 2.5 years ago

LOCATIONS
Table
Status: Valid
Created 2.5 years ago

LOCATIONS_SEQ
Sequence
Status: Valid
Created 2.5 years ago

REGIONS
Table
Status: Valid
Created 2.5 years ago

SECURE_DML
Procedure
Status: Valid
Created 2.5 years ago

To call a table in any query, use the syntax `DatabaseName.TableName`. (e.g., HR.EMPLOYEES).

In some exercises, we will use Olympic database (OLYM), which contains about 260,000 separate results for competitors at all events from 1896 to 2004, including both summer and winter Olympics.

1. Introduction to SQL SELECT statement

To query data from a table, you use the SQL `SELECT` statement. The `SELECT` statement contains the syntax for selecting columns, selecting rows, grouping data, joining tables, and performing simple calculations.

The `SELECT` statement is one of the most complex commands in SQL. In this lab, we'll focus on the basics only.

The following illustrates the basic syntax of the `SELECT` statement that retrieves data from a single table.

```
SELECT
    select_list
FROM
    table_name;
```

In this syntax:

First, specify a list of comma-separated columns from which you want to query the data in the `SELECT` clause. Then, specify the table name in the `FROM` clause. When evaluating the `SELECT` statement, the database system evaluates the `FROM` clause first and then the `SELECT` clause.

In case you want to query data from all columns of a table, you can use the asterisk (*) operator, like this:

```
SELECT * FROM table_name;
```

Notice that SQL is case-insensitive. It means that the `SELECT` and `select` keywords are the same.

To make the SQL statements more readable, we will use the uppercase letters for the SQL keywords such as `SELECT` and `FROM` and the lowercase letters for the identifiers such as table and column names.

Besides the `SELECT` and `FROM` clauses, the `SELECT` statement can contain many other clauses such as

WHERE – for filtering data based on a specified condition.

ORDER BY – for sorting the result set.

LIMIT – for limiting rows returned.

JOIN – for querying data from multiple related tables.

GROUP BY – for grouping data based on one or more columns.

HAVING – for filtering groups.

You will learn about some of these clauses in this lab. We use the HR.EMPOYEES table in the sample database for the demonstration purposes.

Example. The following example retrieves data from all the columns of the `employees` table:

```
SELECT * FROM HR.EMPLOYEES;
```

Example. If you use want to view the employee id, first name, last name, and hire date of all employees, you use the following query:

```
SELECT
    employee_id,
    first_name,
    last_name,
    hire_date
FROM
    HR.EMPLOYEES;
```

The `SELECT` statement allows you to perform simple calculations.

Example. The following query calculates the year of services of employees on January 1st, 2020 using the FLOOR() , DATEDIFF() and CURRENT_DATE functions:

```
SELECT
    employee_id,
    first_name,
    last_name,
    hire_date,
    FLOOR((TO_DATE('2020-09-01','YYYY-MM-DD') - hire_date)/365) AS YoS
FROM
    HR.EMPLOYEES;
```

In this example, `TO_DATE('2020-09-01','YYYY-MM-DD')` returns the current date and time. To calculate the year of service, we divide the result of the substraction by 365. The `FLOOR()` function returns the largest integer less than or equal the result of a numeric expression. The `YoS` is the *column alias* for the expression below to display a user-friendly heading in the returned result set.

2. ORDER BY clause

To specify exactly the order of rows in the result set, you add use an `ORDER BY` clause in the `SELECT` statement as follows:

```
SELECT
    column1, column2
FROM
    table_name
ORDER BY column1 ASC ,
         column2 DESC;
```

In this syntax, the `ORDER BY` clause appears after the `FROM` clause. In case the `SELECT` statement contains a `WHERE` clause, the `ORDER BY` clause must appear after the `WHERE` clause. To sort the result set, you specify the column in which you want to sort and the kind of the sort order:

- Ascending (`ASC`)
- Descending (`DESC`)

If you don't specify the sort order, the database system typically sorts the result set in ascending order (`ASC`) by default.

Example. The following statement selects employee data and sorts the result set by salary in the descending order:

```
SELECT
      employee_id,
      first_name,
      last_name,
      hire_date,
      salary
FROM
      HR.EMPLOYEES
ORDER BY
      salary DESC;
```

Exercise 1. Develop a query to retrieve employee id, first name, last name, hire date, and salary and sort the results by the first name in ascending order and the last name in descending order.

Note: For each exercise, once you develop the proper Select statements and achieved the desired result, copy the query and paste it in a Word document. Write the exercise number before the answer. Do this for each exercise and at the end submit the file through Blackboard.

To remove duplicates from a result set, you use the `DISTINCT` operator in the `SELECT` clause as follows:

```
SELECT DISTINCT
    column1, column2, ...
FROM
    table1;
```

Example. The following statement retrieves the job id and salary from the `employees` table. With the `DISTINCT` operator, the database system uses values in both `job_id` and `salary` columns to evaluate and remove duplicates. In to better understand, run the query without the `DISTINCT` operator and compare the results.

```
SELECT DISTINCT
      job_id,
      salary
FROM
      HR.EMPLOYEES
ORDER BY
      job_id,
      salary DESC;
```

3. WHERE clause

To select certain rows from a table, you use a `WHERE` clause in the `SELECT` statement. The following illustrates the syntax of the `WHERE` clause in the `SELECT` statement:

```
SELECT
    column1, column2, ...
FROM
    table
WHERE
    condition;
```

The `WHERE` clause appears immediately after the `FROM` clause. The `WHERE` clause contains one or more *logical expressions* that evaluate each row in the table. If a row that causes the condition evaluates to true, it will be included in the result set; otherwise, it will be excluded.

Note that the logical expression that follows the `WHERE` clause is also known as a predicate. You can use various operators to form the row selection criteria used in the `WHERE` clause.

The following table shows the SQL comparison operators:

Operator	**Meaning**
=	Equal to
<> (!=)	Not equal to
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

To form a simple expression, you use one of the operators above with two operands that can be either column name on one side and a literal value on the other, for example: `salary > 1000` asks a question: “Is salary greater than 1000?”

Example. The following query finds employees who have the salaries greater than 14,000 and sorts the result set based on the salary in descending order.

```
SELECT
      employee_id,
      first_name,
      last_name,
      salary
FROM
      HR.EMPLOYEES
WHERE
      salary > 14000
ORDER BY
      salary DESC;
```

Example. The following query finds all employees who work in the department id 50.

```
SELECT
      employee_id,
      first_name,
      last_name,
      department_id
FROM
      HR.EMPLOYEES
WHERE
      department_id = 50
ORDER BY
      first_name;
```

Exercise 2. Develop a query to retrieve employee id, first name, last name, hire date, and salary of all the employee whose last name is Gates.

Exercise 3. Develop a query to retrieve employee id, first name, last name, hire date, and salary of all the employees who were hired in 2007. You will need to use the EXTRACT function. Look it up!

Exercise 4. Develop a query to retrieve employee id, first name, last name, hire date, and salary finds employees whose salaries are greater than or equal 9,000. Sort the results by salary.

Exercise 5. Use the AND operator to find all employees whose salaries are greater than 5,000 and less than 7,000.

Exercise 6. The `IS NULL` operator compares a value with a null value and returns true if the compared value is null; otherwise, it returns false. Use `IS NULL` to find the first name, last name, and phone number of of all employees who do not have a phone number.

Exercise 7. The `IN` operator compares a value to a list of specified values. The `IN` operator returns true if the compared value matches at least one value in the list; otherwise, it returns false. Use the `IN` operator to find all employees who work in the department id 10, 20, or 30.

The `LIKE` operator compares a value to similar values using a wildcard operator. SQL provides two wildcards used in conjunction with the `LIKE` operator:

- The percent sign (`%`) represents zero, one, or multiple characters.
- The underscore sign (`_`) represents a single character.

Example. The following statement finds all employees whose first name starts with the letter J:

```
SELECT
    employee_id, first_name, last_name
FROM
    HR.EMPLOYEES
WHERE
    first_name LIKE 'J%'
ORDER BY first_name;
```

Note: Unlike majority of database management systems, Oracle's `LIKE` operator is case-sensitive. You can use the UPPER function to perform a case-insensitive match, as in this condition:

```
UPPER(last_name) LIKE 'SM%'
```

Exercise 8. Develop a query to find all employees with the first names whose the second character is h.

4. SQL GROUP BY clause

Grouping is one of the most important tasks that you have to deal with while working with the databases. To group rows into groups, you use the GROUP BY clause.

The GROUP BY clause is an optional clause of the SELECT statement that combines rows into groups based on matching values in specified columns. One row is returned for each group.

You often use the GROUP BY in conjunction with an aggregate function such as MIN, MAX, AVG, SUM, or COUNT to calculate a measure that provides the information for each group.

The following illustrates the syntax of the GROUP BY clause.

```
SELECT
      column1,
      column2,
      AGGREGATE_FUNCTION (column3)
FROM
      table1
GROUP BY
      column1,
      column2;
```

The columns that appear in the GROUP BY clause are called *grouping columns*.

It is not mandatory to include an aggregate function in the SELECT clause. However, if you use an aggregate function, it will calculate the summary value for each group.

If you want to filter the rows before grouping, you add a WHERE clause. However, to filter groups, you use the HAVING clause.

To sort the groups, you add the ORDER BY clause after the GROUP BY clause.

Example. To find the headcount of each department, you group the employees by the department_id column, and apply the COUNT function to each group as the following query:

```
SELECT
      department_id,
      COUNT(employee_id) headcount
FROM
      HR.EMPLOYEES
GROUP BY
      department_id;
```

Exercise 9. Develop a query to retrieve the total number of medals won by each country (NOC column) in 2008 Olympics (EDITION column) from OLYM_MEDALS_VIEW table in OLYM database. Use ORDER BY to sort the results in descending order. This is the top three rows of the results you should get.

NOC	MEDAL_COUNT
USA	315
CHN	184
AUS	149

Exercise 10. The OLYM database includes medal counts for all Olympics from 1896 to 2008. Use GROUP BY to find the athlete with most medals during theses years from the OLYM_MEDALS_VIEW table.

Exercise 11. Develop a query to find the name of countries that have won at least 10 gold medals in 2004 Olympics. Include the gold medal count and order the results based on the count in descending order (HINT: you need to use both WHERE and HAVING clause in your query).

This is the top three rows of the results you should get.

NOC	GOLD_COUNT
USA	116
CHN	52
AUS	49

5. SQL INNER JOIN clause

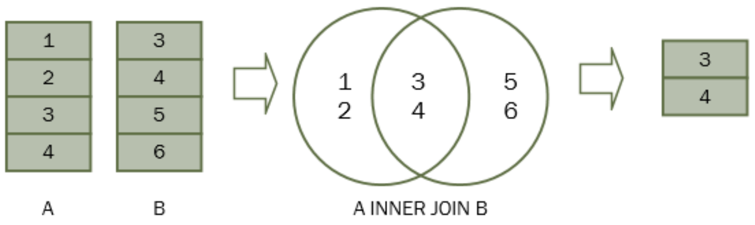
So far, you have learned how to use the SELECT statement to query data from a single table. However, the SELECT statement is not limited to query data from a single table. The SELECT statement can link multiple tables together.

The process of linking tables is called joining. SQL provides many kinds of joins such as inner join, left join, right join, full outer join, etc. This lab focuses on the inner join.

The inner join clause links two (or more) tables by a relationship between two columns. Whenever you use the inner join clause, you normally think about the *intersection*.

It is much easier to understand the inner join concept through a simple example.

Suppose, we have two tables: A & B. Table A has four rows: (1,2,3,4) and table B has four rows: (3,4,5,6). When table A joins with the table B using the inner join, we have the result set (3,4) that is the intersection of the table A and table B.



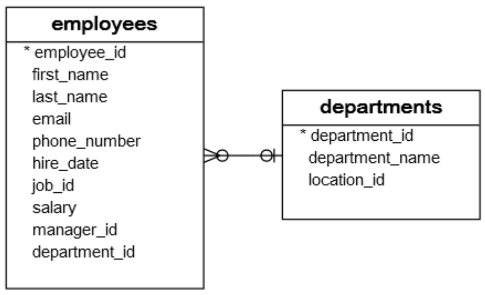
For each row in table A, the inner join clause finds the matching rows in the table B. If a row is matched, it is included in the final result set.

Suppose the column name of the A & B tables is n, the following statement illustrates the inner join clause:

```
SELECT
  A.n
FROM A
INNER JOIN B ON B.n = A.n;
```

The INNER JOIN clause appears after the FROM clause. The condition to match between table A and table B is specified after the ON keyword. This condition is called *join condition* i.e., *B.n = A.n*

Example. Each employee belongs to one and only one department while each department can have more than one employee. The relationship between the employees and departments table is one-to-many:



The department_id column in the employees table is the foreign key column that links the employees to the departments table. To get the information of the department id 10, 20, and 30, you use the following statement.

```
SELECT
      department_id,
      department_name
FROM
      HR.departments
WHERE
      department_id IN (10, 20, 30);
```

Now, to get the information of employees who work in the department id 10, 20 and 30, you use the following query:

```
SELECT
      first_name,
      last_name,
      department_id
FROM
      HR.employees
WHERE
      department_id IN (10, 20, 30)
ORDER BY
      department_id;
```

To combine data from these two tables, you use an inner join clause as the following query:

```
SELECT
    first_name,
    last_name,
    HR.employees.department_id,
    HR.departments.department_id,
    department_name
FROM
    HR.employees
        INNER JOIN
    HR.departments ON HR.departments.department_id = HR.employees.department_id
WHERE
    HR.employees.department_id IN (10 , 20, 30);
```

For each row in the employees table, the statement checks if the value of the department_id column equals the value of the department_id column in the departments table.

If the condition HR.employees.department_id = HR.departments.department_id is satisfied, the combined row that includes data from rows in both employees and departments tables are included in the result set.

Notice that both employees and departments tables have the same column name department_id, therefore we had to qualify the department_id column using the syntax table_name.column_name.

Exercise 12. Develop a query to list the name of each employee along with their job title by joining EMPLOYEES and JOBS tables. Sort the results by employees' last name in ascending order.

Exercise 13. Develop a query to list the name of each employee with a salary of over $10,000 along with their job title by joining EMPLOYEES and JOBS tables. Sort the results by employees' last name in ascending order.