**XJTLU Entrepreneur College (Taicang)**

***School of* *AI and Advanced Computing***

**Lab Manual- Lab 6**

***CPT103TC:***

***Introduction to Database***

Shaheen Khatoon

Second Part of Semester 1

**Academic Year 2021-2022**

**Preface**

This laboratory manual serves as supplemental material for the laboratory class. Majority of its contents were taken from the materials of Oracle’s DB Design and Programming with SQL training.

The table below shows the delivery plan with reference material.

|  |  |  |
| --- | --- | --- |
| **Labs** | **Topics** | **Reference in DB Programming with SQL** |
| **1** | Introduction to Oracle Application Express  SQL Workshop for uploading and running scripts | Lab set-up  Oracle Application Development Foundation (Self-study)  SQL Scripts |
| **2** | Demonstrate and end to end application building process | Oracle Application Development Foundation (Self-study)  Project OracleFlix-demo |
| **3** | Data modeling using SQL Developer and ER Assistant | SQL Developer and ER Assistant Tutorial |
| **4** | SQL DDL command to create database objects and constraints | Demo CompanyScript.SQl; |
| **5** | Managing constraints and SQL DML | Lab handouts |
| **6** | Basic SQL | Lab handouts |
| **7** | SQL Group functions and subqueries | Lab handouts |
| **8** | SQL Joins | Lab handouts |
| **9** | SQL Single Row functions | Lab handouts |
| **10** | Application development | Lab handouts |

Table of Contents

[**Lab 6 –Introduction to Basic SQL** 3](#_Toc95516159)

[Anatomy of a SQL Statement 3](#_Toc95516160)

[SELECT Statement with a Condition 3](#_Toc95516161)

[Lab Tasks 5](#_Toc95516162)

[WHERE and ORDER BY 8](#_Toc95516163)

[Syntax with Examples 8](#_Toc95516164)

[Lab Exercises: 10](#_Toc95516165)

# **Lab 6 –Introduction to Basic SQL**

**Note:** For this lab, you must upload and run the **SQL script (SQL\_Schema.sql)** that came along with the lab. Follow data upload steps from lab 1 to upload required tables. A PDF document for the schema ERD and table design is provided. You are required to go through the pdf document to get yourself familiar with the underlying database.

SQL is a very high-level language. Say “what to do” rather than “how to do it.”

Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java.

Database management system figures out the “best” way to execute query called “query optimization.”

## Anatomy of a SQL Statement

*SELECT <attribute list>*

*FROM <table list>*

*WHERE <condition>;*

*where*

*■ <attribute list> is a list of attribute names whose values are to be retrieved by the query.*

*■ <table list> is a list of the relation names required to process the query.*

*■ <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.*

<>20 表示不等于 !=20

## SELECT Statement with a Condition

SELECT <*column1*>, … <*columnN*> FROM <*table\_name*> WHERE <*condition*> ;

| **Example** | **Explanation** |
| --- | --- |
| **Using Comparison Operators** | |
| SELECT DISTINCT country\_id  FROM locations  WHERE location\_id<2000 | It displays unique country\_id from table locations for all rows that have location\_id less than 2000.  **Note: To see the effect, repeat the statement without the word DISTINCT.** |
| SELECT employee\_id, first\_name, email, hire\_date FROM employees  WHERE employee\_id<200; | It displays 4 of 12 columns and all rows of table **employees** whose employee\_id is less than 200.  **Reminder: Use DESC to know the structure of a table.** |
| SELECT employee\_id, first\_name, hire\_date FROM employees  WHERE hire\_date<TO\_DATE('1-1-2000','dd-mm-yyyy'); | It displays 3 of 12 columns and all rows of table **employees** who were hired before 1-Jan-2000. |
| **Using Concatenation Operator** | |
| SELECT 'A city from the US is ' || city  FROM locations  WHERE country\_id='US'; | It displays a concatenation of literal value and column city for all rows in table **locations** whose country\_id is ‘US’.  结果是“”里的句子加上符合条件的city |
| SELECT street\_address||' ' ||city as full\_address FROM locations  WHERE country\_id='US'; | It displays a concatenation of columns street\_address and city with alias full\_address for all rows in table **locations** with country\_id of ‘US’.  street\_address||' ' ||city合并成一个full\_address |
| SELECT department\_id || ' '||  department\_name  FROM departments; | ||' '|| is used to make a space between the department\_id and department\_name  The 'space' character in single quotation marks creates a space between the column values |
| SELECT department\_id ||' '||  department\_name AS " Department Info "  FROM departments;  " Department Info "这个需要引号因为不是一个单词  SELECT first\_name ||' '||  last\_name AS "Employee Name"  FROM employees; | Column aliases (e.g. Department Info) are useful when using the concatenation operator so that the default SELECT line does not appear as the column heading. |
| **Concatenation and Literal Values** | You can also include numbers as literal values. In the following example, the number 1 is concatenated to the strings, 'has a 'and 'year salary of ' |
| SELECT last\_name ||' has a '|| 1 ||' year salary of '||  salary\*12 || ' dollars.' AS Pay  FROM employees; |  |
| **Using Between, In, and Like** | |
| SELECT employee\_id, salary, salary\*0.1 as rate rate是salary\*0.1之后的名字  FROM employees  WHERE salary BETWEEN 10000 AND 12000; | It displays 2 of 12 existing columns plus 1 derived column called “rate” (alias) and all rows of table **employees** with salaries between 10000 and 12000. |
| SELECT \*  FROM employees  WHERE job\_id IN ('SA\_MAN', 'SA\_REP'); | It displays all columns and all rows from table **employees** with job\_id either ‘SA\_MAN’ or ‘SA\_REP’ (Sales Manager or Sales Representative) |
| SELECT \*  FROM employees  WHERE first\_name LIKE 'J%'; | It displays all columns and all rows from table **employees** whose first\_name starts with ‘J’.  The character ‘%’ represents none, one or more characters. |
| SELECT \*  FROM employees  WHERE first\_name LIKE '\_e\_'; | It displays all columns and all rows from table **employees** whose first\_name starts with any character, 2nd character ‘e’, and any 3rd character. The name should consist of only 3 letters (e.g Lex).  The character ‘\_’ represents one and only one character. |
| SELECT \*  FROM employees  WHERE first\_name LIKE '\_e%';  [charlist]里面都是不同的要求 | As the above example, the 1st character can be anything, the 2nd must be ‘e’ but the next characters can be none, one or more. |
| **Using IS NULL** | |
| SELECT \*  FROM departments  WHERE manager\_id is null; | It displays all columns and all rows of **departments** table whose manager\_id is null (empty). |

## Lab Tasks

1. Create a SQL statement that displays only the first\_name and salary of an employee whose salary is greater than 10,000.

|  |
| --- |
|  |

1. Create a SQL statement that displays only the first\_name and salary of an employee whose salary is between 10000 and 15000.

|  |
| --- |
|  |

1. Create a SQL statement that displays the first\_name, the word ‘earns’ and salary of an employee.

e.g. ‘Smith earns 10000’

|  |
| --- |
|  |

1. Create a SQL statement that displays the first\_name and last\_name of employees whose last\_name is ‘Smith’.

|  |
| --- |
|  |

1. Create a SQL statement that displays the first\_name and last\_name of employees whose last\_name is either ‘Smith’, ‘King’, or ‘Rogers’.

|  |
| --- |
|  |

1. Create a SQL statement that displays the first\_name and last\_name of employees whose last\_name starts with ’S’.

|  |
| --- |
|  |

# WHERE and ORDER BY

* create SELECT Statements using logical operators
* create SELECT Statements using ORDER BY
* create SELECT Statements using column aliases
* create SELECT Statements using sorting with other columns
* create SELECT Statements using sorting with multiple columns

Logical conditions combine the result of two or more conditions to produce a single result. In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <, <=, >, >=, and, or, <>.

These correspond to the relational algebra operators =, <, ≤, >, ≥, and ≠, respectively, and to the C/C++ programming language operators =, <, <=, >, >=, and !=. The main syntactic

difference is the not equal operator. SQL has additional comparison operators that we will present gradually.

### 

## Syntax with Examples

SELECT <*column1*>, … <*columnN*> FROM <*table\_name*> WHERE <*condition*>

**ORDER BY** <*column*> [DESC]

**The order of execution is as follows:**

* FROM clause: locate the table that contains data
* WHERE clause: restrict the rows to be returned
* SELECT clause: selects from the reduced data set columns requested
* ORDER BY clause: orders the result set

| **Example** | **Explanation** |
| --- | --- |
| **Using Logical Operators** | |
| SELECT last\_name, department\_id, salary  FROM employees  WHERE department\_id > 50 AND salary > 12000; | In the query shown, the results returned will be rows that satisfy BOTH conditions specified in the WHERE clause. |
| SELECT last\_name, hire\_date, job\_id  FROM employees  WHERE hire\_date > '01/jan/1998' AND job\_id LIKE 'SA%'; | Another example of using AND in the where clause. |
| SELECT department\_name, manager\_id, location\_id  FROM departments  WHERE location\_id = 2500 OR manager\_id=124; | all rows returned have a location\_id of 2500 OR they have a manager\_id equal to 124 |
| SELECT department\_name, location\_id  FROM departments  WHERE location\_id NOT IN (1700,1800); | The NOT operator will return rows that do NOT satisfy the condition in the WHERE clause. |
| SELECT last\_name||' '||salary\*1.05  As "Employee Raise"  FROM employees  WHERE department\_id IN(50,80)  AND first\_name LIKE 'C%'  OR last\_name LIKE '%s%'; | The precedence order of operators is Arithmetic, Concatenation, Comparison, IS NULL, LIKE, IN, BETWEEN, NOT, AND, and the last is OR. |
| **Using Order By** | |
| SELECT last\_name, hire\_date  FROM employees  ORDER BY hire\_date; | uses the ORDER BY clause to order hire\_date in ascending (default) order |
| SELECT last\_name, hire\_date  FROM employees  ORDER BY hire\_date DESC; | uses the ORDER BY clause to order hire\_date in descending (DESC) order |
| **Using Column Aliases** | |
| SELECT last\_name, hire\_date as "Date Started"  FROM employees  ORDER BY "Date Started"; | The alias used in the SELECT statement is referenced in the ORDER BY clause. |
| **Sorting with Other Columns** | |
| SELECT employee\_id, first\_name  FROM employees  WHERE employee\_id < 105  ORDER BY last\_name; | the data is sorted by the last\_name column even though this column is not listed in the SELECT statement  It is difficult to verify your results when you sort by a column that you're not SELECTing. In the real world, you would run your query selecting the last\_name column until you were sure you were getting the right data. Then you could remove that column from your SELECT statement. |
| **Sorting with Multiple Columns** | |
| SELECT department\_id, last\_name  FROM employees  WHERE department\_id <= 50  ORDER BY department\_id, last\_name; | Employees are first ordered by department number (from lowest to highest), then for each department, the last names are displayed in alphabetical order (A to Z). |
| SELECT department\_id, last\_name  FROM employees  WHERE department\_id <= 50  ORDER BY department\_id DESC, last\_name; | If you want to reverse the sort order of a column, add DESC after its name. |

## Lab Exercises:

Use the company database to perform the following operations. Use order by and column aliases to make the results meaningful.

To perform this lab activity, you first need to upload companySchema.SQl by following the data upload steps, you already learned.

**Example:**

Retrieve the birth date and address of the employee(s) whose name is ‘John B. Smith’.

SELECT Bdate, Address

FROM EMPLOYEE

WHERE Fname = 'John' AND Minit = 'B' AND Lname = 'Smith';

**Practice:**

1. Retrieve the name and address of all employees who work for the ‘Research’ department. Sort results by the first name.
2. For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.
3. For each employee, retrieve the employee’s first and last name and the first and last name of his or her immediate supervisor.