**XJTLU Entrepreneur College (Taicang)**

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

**Lab Manual (Lab 7)**

***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, subqueries, and set operations** | **Lab handouts** |
| 8 | SQL Joins | Lab handouts |
| 9 | SQL Single Row functions | Lab handouts |
| 10 | Application development | Lab handouts |

Table of Contents

[**GROUP Functions** 3](#_Toc95516415)

[SELECT Statement with Group Functions 3](#_Toc95516416)

[SELECT Statement with Group By Clause 5](#_Toc95516417)

[SELECT Statement with Group By and Having Clauses 8](#_Toc95516418)

[SELECT Statement with Set Operators 9](#_Toc95516419)

[Fundamentals of Subqueries 10](#_Toc95516420)

[Subquery Overview 11](#_Toc95516421)

[Single Row Subquery 12](#_Toc95516422)

[Multiple Row Subquery 13](#_Toc95516423)

[Lab Practice 17](#_Toc95516424)

[Subqueries 17](#_Toc95516425)

[Aggregate function 17](#_Toc95516426)

[Ordering of query results 17](#_Toc95516427)

[Grouping function 18](#_Toc95516428)

[Set operations 18](#_Toc95516429)

# **GROUP Functions**

In SQL, the following group functions can operate on a whole table or on a specific grouping of rows. Each function returns one result.

Group Functions include AVG, COUNT, MIN, MAX, SUM, VARIANCE, and STDDEV.

## SELECT Statement with Group Functions

| **Example** | **Output** |
| --- | --- |
| **Using MIN** | |
| SELECT MIN(life\_expect\_at\_birth)  AS "Lowest Life Exp"  FROM wf\_countries; | 32.62 |
| SELECT MIN(country\_name)  FROM wf\_countries; | Anguilla |
| SELECT MIN(hire\_date)  FROM employees; | 17/Jun/1987 |
| **Using MAX** | |
| SELECT MAX(life\_expect\_at\_birth)  AS "Highest Life Exp"  FROM wf\_countries; | 83.51 |
| SELECT MAX(country\_name)  FROM wf\_countries | Western Sahara |
| SELECT MAX(hire\_date)  FROM employees; | 29/Jan/2000 |
| **Using SUM** |  |
| SELECT SUM(area)  FROM wf\_countries  WHERE region\_id = 29; | 241424 |
| SELECT SUM(salary)  FROM employees  WHERE department\_id = 90; | 58000 |
| **Using AVG** | |
| SELECT AVG(area)  FROM wf\_countries  WHERE region\_id = 29; | 9656.96 |
| SELECT ROUND(AVG(salary), 2)  FROM employees  WHERE department\_id = 90; | 19333.33 |
| **Using VARIANCE** | |
| SELECT ROUND(VARIANCE(life\_expect\_at\_birth),4)  FROM wf\_countries; | 143.2394 |
| **Using STDDEV** | |
| SELECT ROUND(STDDEV(life\_expect\_at\_birth), 4)  FROM wf\_countries; | 11.9683 |
| **Using Several Group Functions** | |
| SELECT MAX(salary), MIN(salary), MIN(employee\_id) FROM employees WHERE department\_id = 60; | 9000 4200 103 |
| **Using COUNT** | |
| SELECT COUNT(job\_id) FROM employees; | 20 |
| SELECT COUNT(commission\_pct) FROM employees; | 4 |
| **Using COUNT All Rows** | |
| SELECT COUNT(\*)  FROM employees  WHERE hire\_date < '01/Jan/1996'; | 9 |
| **Using DISTINCT and COUNT** | |
| SELECT COUNT (DISTINCT job\_id)  FROM employees; | 12 |
| SELECT COUNT (DISTINCT salary)  FROM employees; | 18 |
| **Using NVL** | |
| SELECT AVG(commission\_pct) FROM employees; | .2125 |
| SELECT AVG(NVL(commission\_pct, 0))  FROM employees; | .0425  NVL(a, b)  b是在值为null的时候替换null都值 |

## SELECT Statement with Group By Clause

SELECT <*column1*>, … <*columnN*>

FROM <*table\_name*>

WHERE <*condition*>

GROUP BY <*column1*>;

* You use the GROUP BY clause to divide the rows in a table into smaller groups.
* You can then use the group functions to return summary information for each group.
* Group functions require that any column listed in the SELECT clause that is not part of a group function must be listed in a GROUP BY clause.
* If you include a group function (AVG, SUM, COUNT, MAX, MIN, STDDEV, VARIANCE) in a SELECT clause along with any other individual columns, each individual column must also appear in the GROUP BY clause.
* You cannot use a column alias in the GROUP BY clause.
* The WHERE clause excludes rows before they are divided into groups.

| **Example** | **Explanation** |
| --- | --- |
| SELECT department\_id, AVG(salary)  FROM employees  GROUP BY department\_id  ORDER BY department\_id; | The rows are being grouped by department\_id. The AVG function is then applied to each group. |
| SELECT MAX(salary) FROM employees GROUP BY department\_id; | To find the maximum salary of employees in each department  Column department\_id is added to tell which maximum salary belongs to which department  But how can we tell which maximum salary belongs to which department? |
| Usually we want to include the GROUP BY column in the SELECT list  SELECT department\_id, MAX(salary)  FROM employees  GROUP BY department\_id; |  |
| SELECT COUNT(country\_name), region\_id FROM wf\_countries GROUP BY region\_id ORDER BY region\_id; | This example shows how many countries are in each region.  Remember that group functions ignore null values, so if any country does not have a country name, it will not be included in the COUNT. |
| SELECT COUNT(\*), region\_id FROM wf\_countries GROUP BY region\_id ORDER BY region\_id; | Unlike the previous example, this one will count all of the rows in each region group, without the need to check which columns contained NULL values. |
| SELECT department\_id, MAX(salary)  FROM employees  WHERE last\_name != 'King'  GROUP BY department\_id; | WHERE Clause: We can also use a WHERE clause to exclude rows before the remaining rows are formed into groups  As employee King is excluded by the WHERE clause, the MAX(salary) for department 90 is returned as 17000. |
| SELECT region\_id, ROUND(AVG(population)) AS population  FROM wf\_countries  GROUP BY region\_id  ORDER BY region\_id; | Shows the average population of all countries in each region.  Rounds the average to a whole number |
| SELECT country\_id, COUNT(language\_id) AS "Number of languages"  FROM wf\_spoken\_languages  GROUP BY country\_id; | Counts the number of spoken languages for all countries. |
| SELECT department\_id, job\_id, count(\*)  FROM employees  WHERE department\_id > 40  GROUP BY department\_id, job\_id; | This example shows how many employees are doing each job within each department. |
| SELECT max(avg(salary))  FROM employees  GROUP by department\_id; | Group functions can be nested to a depth of two when GROUP BY is used. |

## SELECT Statement with Group By and Having Clauses

SELECT <*column1*>, … <*columnN*>

FROM <*table\_name*>

WHERE <*condition*>

GROUP BY <*column1*>

HAVING <*condition*>;

.

| **Example** | **Explanation** |
| --- | --- |
| Suppose we want to find the maximum salary in each department, but only for those departments which have more than one employee?  What is wrong with this example?  SELECT department\_id, MAX(salary)  FROM employees  WHERE COUNT(\*) > 1  GROUP BY department\_id; | A WHERE clause can be used only to include/exclude individual rows, not groups of rows. Therefore we cannot use group functions in a WHERE clause.  In the same way, you used the WHERE clause to restrict the rows that you selected, you can use the HAVING clause to restrict groups.  In a query using a GROUP BY and HAVING clause, the rows are first grouped, group functions are applied, and then only those groups matching the HAVING clause are displayed. |
| SELECT department\_id,MAX(salary)  FROM employees  GROUP BY department\_id  HAVING COUNT(\*)>1  ORDER BY department\_id; | The WHERE clause is used to restrict rows; the HAVING clause is used to restrict groups returned from a GROUP BY clause.  The query first finds the MAX salary for each department in the employees table. The HAVING clause then restricts the groups returned to those departments that have more than 1 employee. |
| The HAVING and GROUP BY clauses can use different columns. The example on GROUPs BY region\_id, but the HAVING clause restricts groups based on population.  SELECT region\_id,  ROUND(AVG(population))  FROM wf\_countries  GROUP BY region\_id  HAVING MIN(population)>300000  ORDER BY region\_id; | This query finds the average population of the countries in each region.  It then only returns the region groups with a lowest population greater than three hundred thousand.  You place each clause in the following order:  **SELECT column, group\_function**  **FROM table**  **WHERE**  **GROUP BY**  **HAVING**  **ORDER BY** |

## SELECT Statement with Set Operators

* Set operators are used to combine the results from different SELECT statements into one single result output
* Sometimes you want a single output from more than one table
* If you join the tables, the rows that meet the join criteria are returned, but what if a join will return a result set that doesn't meet your needs?
* This is where SET operators come in
* They can return the rows found in multiple SELECT statements, the rows that are in one table and not the other, or the rows common to both statements
* There are a few rules to remember when using SET operators:
* The number of columns and the data types of the columns must be identical in all of the SELECT statements used in the query.
* The names of the columns need not be identical.
* Column names in the output are taken from the column names in the first SELECT statement.
* So any column aliases should be entered in the first statement as you would want to see them in the finished report.
* The clauses we will use are UNION, UNION ALL, INTERSECT, and MINUS.

| **Example** | **Explanation** |
| --- | --- |
| SELECT hire\_date, employee\_id, job\_id  FROM employees  UNION  SELECT TO\_DATE(NULL),employee\_id,  job\_id  FROM job\_history;  The employees table contains a hire date, employee id and a job id.  The job history table contains employee id and job id, but does not have a hire date column.  The two tables have the employee id and job id in common, but job history does not have a hire\_date. You can use the TO\_CHAR(NULL) function match columns in a SELECT list.  One NULL is included for each missing column. Furthermore, NULL is formatted to match the data type of the column it is standing in for, so TO\_CHAR, TO\_DATE, or TO\_NUMBER functions are used to achieve identical SELECT lists. | The rows with NULL values for hire\_date are from the job\_history table |
| SELECT hire\_date, employee\_id, job\_id  FROM employees  UNION  SELECT TO\_DATE(NULL),employee\_id, job\_id  FROM job\_history  ORDER BY employee\_id; | Using the previous query example, we could ORDER BY employee\_id to see the jobs each employee has held |
| **Exercise:** Modify the above queries by replacing union with other set operators such as MINUS, INTERSECT, and UNION ALL and analyse the result set to observe the difference | Union all 保留两个table中完全相同的row |

# Fundamentals of Subqueries

* Throughout this course, you have written queries to extract data from a database
* What if you wanted to write a query, only to find out you didn't have all the information you needed to construct it?
* You can solve this problem by nesting queries—placing one query inside the other query
* The inner query is called a "subquery."

## Subquery Overview

* A subquery is a SELECT statement that is embedded in a clause of another SELECT statement
* A subquery executes once before the main query
* The result of the subquery is used by the main or outer query
* Subqueries can be placed in a number of SQL clauses, including the WHERE clause, the HAVING clause, and the FROM clause
* The subquery syntax is:

SELECT select\_list

FROM table

WHERE expression operator

(SELECT select\_list

The SELECT statement in parentheses is the inner query or 'subquery'. It executes first, before the outer query

FROM table);

* The two types of subqueries are:
  + **Single-row subqueries** that use single-row operators (>, =, >=, <>, <=) and return only one row from the inner query
  + **Multiple-row subqueries** that use multiple-row operators (IN, ANY, ALL) and return more than one row from the inner query

| **Example** | **Explanation** |
| --- | --- |
| SELECT first\_name, last\_name, hire\_date  FROM employees  WHERE hire\_date >  (SELECT hire\_date  FROM employees  WHERE last\_name = 'Vargas'); | What if you wanted to find out the names of the employees that were hired after Peter Vargas?  The first thing you need to know is the answer to the question, "When was Peter Vargas hired?"  Once you know his hire date, then you can select those employees whose hire dates are after his. |
| SELECT last\_name FROM employees WHERE department\_id = (SELECT department\_id FROM employees WHERE last\_name = 'Grant'); | If a subquery returns a null value or no rows, the outer query takes the results of the subquery (null) and uses this result in its WHERE clause.  The outer query will then return no rows, because comparing any value with a null always yields a null. |

### Single Row Subquery

* They return only one row
* They use single-row comparison operators (=, >,>=, <, <=, <>)
* Always enclose the subquery in parentheses.
* Always place the subquery on the right hand side of the comparison condition.
* The outer and inner queries can get data from different tables.
* Only one ORDER BY clause can be used for a SELECT statement, and if specified, it must be the last clause in the main SELECT statement.
* The only limit on the number of subqueries is the buffer size that the query uses.

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name, job\_id, department\_id  FROM employees  WHERE department\_id =  (SELECT department\_id  FROM departments  WHERE department\_name = 'Marketing')  ORDER BY job\_id; | The outer and inner queries can get data from different tables.  The subquery finds the department\_id for 'Marketing', the outer query uses the returned department\_id to display rows from the employees table. |
| SELECT last\_name, job\_id, salary, department\_id  FROM employees  WHERE job\_id =  (SELECT job\_id  FROM employees  WHERE employee\_id = 141)  AND department\_id =  (SELECT department\_id  FROM departments  WHERE location\_id = 1500); | More than one subquery can return information to the outer query.  The first subquery returns the job\_id of employee 141 (ST\_CLERK). The second subquery uses the departments table to find the department\_id at location\_id 1500 (50). The outer query then returns rows from the employees table that match both these values. |
| **"Which employees earn less than the average salary?"**  SELECT last\_name, salary  FROM employees  WHERE salary <  (SELECT AVG(salary)  FROM employees); | Group function (AVG) in subquery. The subquery first finds the average salary for all employees, the outer query then returns employees with a salary of less than the average. |
| **Which departments have a lowest salary that is greater than the lowest salary in department 50?**  SELECT department\_id, MIN(salary)  FROM employees  GROUP BY department\_id  HAVING MIN(salary) >  (SELECT MIN(salary)  FROM employees  WHERE department\_id = 50); | In this example, the subquery selects and returns the lowest salary in department 50. The outer query uses this value to select the department ID and lowest salaries of all the departments whose lowest salary is greater than that number. The HAVING clause eliminated those departments whose MIN salary was less than department 50's MIN salary. |

### Multiple Row Subquery

* Subqueries that return more than one value are called multiple-row subqueries.
* Because we cannot use the single-row comparison operators (=, <, and so on), we need different comparison operators for multiple-row subqueries.
* The multiple-row operators are: IN, ANY, and ALL
* The NOT operator can be used with any of these three operators.

| **Example** | **Explanation** |
| --- | --- |
| **Query Comparison** |  |
| Whose salary is equal to the salary of an employee in department 20 ?  SELECT first\_name, last\_name  FROM employees  WHERE salary =  (SELECT salary  FROM employees  WHERE department\_id = 20); | This example returns an error because more than one employee exists in department 20, the subquery returns multiple rows    We call this a multiple-row subquery  The problem is the equal sign (=) in the WHERE clause of the outer query  How can one value be equal to (or not equal to) more than one value at a time? |
| The **IN** operator is used within the outer query WHERE clause to select only those rows which are IN the list of values returned from the inner query. For example, we are interested in all the employees that were hired the same year as an employee in department 90  SELECT last\_name, hire\_date  FROM employees  WHERE EXTRACT(YEAR FROM hire\_date) IN  (SELECT EXTRACT(YEAR FROM hire\_date)  FROM employees  WHERE department\_id=90);  EXTRACT function can be used to extract YEAR, MONTH or DAY fields from a DATE data type | The inner query will return a list of the years that employees in department 90 were hired.  The outer query will then return any employee that was hired the same year as any year in the inner query list. |
| The ANY operator is used when we want the outer-query WHERE clause to select the rows which match the criteria (<, >, =, etc.) of at least one value in the subquery result set.  SELECT last\_name, hire\_date  FROM employees  WHERE EXTRACT(YEAR FROM hire\_date) < ANY  (SELECT EXTRACT(YEAR FROM hire\_date)  FROM employees  WHERE department\_id=90); | The example shown will return any employee whose year hired is less than at least one year hired of employees in department 90 |
| The ALL operator is used when we want the outer-query WHERE clause to select the rows which match the criteria ( <, >, =, etc.) of all of the values in the subquery result set.  SELECT last\_name, hire\_date FROM employees  WHERE EXTRACT(YEAR FROM hire\_date) < ALL  (SELECT EXTRACT(YEAR FROM hire\_date)  FROM employees  WHERE department\_id=90); | The ALL operator compares a value to every value returned by the inner query.  As no employee was hired before 1987, no rows are returned.  =ALL: How can one value equal every one of a set of values? For this reason, =ALL is rarely used. |
| **Dealing with Null Values** |  |
| SELECT last\_name,  employee\_id  FROM employees  WHERE employee\_id IN  (SELECT manager\_id  FROM employees);  SELECT last\_name, employee\_id  FROM employees  WHERE employee\_id <= ALL  (SELECT manager\_id  FROM employees); | Suppose that one of the values returned by a multiple-row subquery is null, but other values are not.  If IN or ANY are used, the outer query will return rows which match the non-null values.  If ALL is used, the outer query returns no rows because ALL compares the outer query row with every value returned by the subquery, including the null.  And comparing anything with null results in null. |
| **GROUP BY and HAVING**  GROUP BY clause and the HAVING clause can also be used with multiple-row subqueries. | |
| What if you wanted to find the departments whose minimum salary is less than the salary of any employee who works in department 10 or 20? We need a multiple-row subquery which returns the salaries of employees in departments 10 and 20. The outer query will use a group function (MIN) so we need to GROUP the outer query BY department\_id.  SELECT department\_id, MIN(salary)  FROM employees  GROUP BY department\_id  HAVING MIN(salary) < ANY  (SELECT salary  FROM employees  WHERE department\_id IN (10,20))  ORDER BY department\_id; |  |
| **Multiple-Column Subqueries**  Subqueries can use one or more columns.  If they use more than one column, they are called multiple-column subqueries.  A multiple-column subquery can be either **pair-wise comparisons or non-pair-wise** comparisons. | |
| The example below shows a multiple-column **pair-wise subquery**  SELECT employee\_id, manager\_id, department\_id  FROM employees  WHERE(manager\_id,department\_id) IN  (SELECT manager\_id,department\_id  FROM employees  WHERE employee\_id IN (149,174))  AND employee\_id NOT IN (149,174); | The query lists the employees whose manager and departments are the same as the manager and department of employees 149 or 174.  First, the subquery to retrieve the MANAGER\_ID and DEPARTMENT\_ID values for the employees with EMPLOYEE\_ID 149 or 174 is executed. These values are compared with the MANAGER\_ID column and the DEPARTMENT\_ID column of each row in the EMPLOYEES table. If the values match, the row is displayed. In the output, the records of the employees with the EMPLOYEE\_ID 149 or 174 will not be displayed. |
| SELECT employee\_id,  manager\_id,  department\_id  FROM employees  WHERE manager\_id IN  (SELECT manager\_id  FROM employees  WHERE employee\_id IN (149,174))  AND department\_id IN  (SELECT department\_id  FROM employees  WHERE employee\_id IN (149,174))  AND employee\_id NOT IN(149,174); | A **non-pair-wise multiple-column subquery** also uses more than one column in the subquery, but it compares them one at a time, so the comparisons take place in different subqueries:  First, the subquery to retrieve the MANAGER\_ID values for the employees with the EMPLOYEE\_ID 149 or 174 is executed. Similarly, the second subquery to retrieve the DEPARTMENT\_ID values for the employees with the EMPLOYEE\_ID 149 or 174 is executed. The retrieved values are compared with the MANAGER\_ID and DEPARTMENT\_ID column for each row in the EMPLOYEES table. |
| SELECT last\_name AS "Not a Manager"  FROM employees emp  WHERE NOT EXISTS  (SELECT \*  FROM employees mgr  WHERE mgr.manager\_id = emp.employee\_id); | In this example, the subquery is selecting the employees that are managers.  The outer query then returns the rows from the employee table that do NOT EXIST in the subquery. |

# Lab Practice

Apply the SQL concepts learn in this lab on the company database for performing the following retrieval operations. *Please refer to the textbook Ch. 5 and 6 examples queries for the correct answers.*

## Subqueries

1. Returns the names of employees whose salary is greater than the salary of all the employees in department 5
2. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

## Aggregate function

1. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.
2. Find the sum of the salaries of all employees of the ‘Research’ department, as well as the maximum salary, the minimum salary, and the average salary in this department.
3. Retrieve the total number of employees in the company (Q1) and the number of employees in the ‘Research’ department (Q2)
4. Count the number of distinct salary values in the database.

## Ordering of query results

1. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.
2. For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

## Grouping function

1. For each department, retrieve the department number, the number of employees in the department, and their average salary.
2. For each project, retrieve the project number, the project name, and the number of employees who work on that project.
3. For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.
4. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

## Set operations

1. Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project.
2. Use all set operators in the above query to observe the results sets