BIC 21404 DATABASE SYSTEM

LAB SHEET 4

Name: _____ Matric No.: _____

Title : Creating and managing tables
Objectives : At the end of the session, students are able to:
i. Limit the rows that are retrieved by a query
ii. Sort the rows that are retrieved by a query
iii. Use ampersand substitution to restrict and sort output at run time
Duration : 2 Hours

Limiting Rows Using a Selection

Assume that you want to display all the employees in department 90. The rows with a value of 90 in the `DEPARTMENT_ID` column are the only ones that are returned. This method of restriction is the basis of the `WHERE` clause in SQL.

The syntax:

```
SELECT * | { [DISTINCT] column | expression [alias], …}
FROM table
[WHERE condition(s)];
```

The SQL statement:

```
SELECT employee_id, last_name, job_id, department_id
FROM employees
WHERE department_id = 90;
```

The `SELECT` statement retrieves the employee ID, last name, job ID, and department number of all employees who are in department 90.

Character Strings and Dates

- Character strings and date values are enclosed with single quotation marks.
- Character values are case-sensitive and date values are format-sensitive
- The default date display format is DD-MON-RR

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'Whalen';

SELECT last_name
FROM employees
WHERE hire_date = '17-FEB-96';
```

Comparison Operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN … AND …	Between two values (inclusive)
IN (set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Comparison operators are used in conditions that compare one expression with another value or expression. They are used in the `WHERE` clause in the following format:

Syntax

```
… WHERE expr operator value
```

Example

```
… WHERE hire_date = '01-JAN-95'
… WHERE salary >= 6000
… WHERE last_name = 'Smith'
```

Note:

- An alias cannot be used in the `WHERE` clause
- The symbols `!=` and `^=` can also represent the *not equal to* condition

Using Comparison Operators

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

In the example, the `SELECT` statement retrieves the last name and salary from the `EMPLOYEES` table for any employee whose salary is less than or equal to RM3000. Note that there is an explicit value supplied to the `WHERE` clause. The explicit value of `3000` is compared to the salary value in the `SALARY` column of the `EMPLOYEES` table.

Range Conditions Using the `BETWEEN` Operator

Use the BETWEEN operator to display rows based on a range of values:

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

You can display rows based on a range of values using the BETWEEN operator. The range that you specify contains a lower limit and an upper limit. The SELECT statements above returns rows from the EMPLOYEES table for any employee whose salary is between RM2,500 and RM3,500. Values that are specified with the BETWEEN operator are inclusive. However you must specify the lower limit first.

Membership Condition Using the `IN` Operator

Use the `IN` operator to test for values in a list:

```
SELECT employees_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201);
```

The condition defined using the `IN` operator is also known as the membership condition. The example above displays employees number, last names, salaries, and managers' employee numbers for all the employees whose manager's employee number is 100, 101, or 201.

Note: The set of values can be specified in any random order, for example, (201,100,101).

The `IN` operator can be used with any data type. The following example returns a row from the `EMPLOYEES` tabls, for any employee whose last name is included in the list of names in the `WHERE` clause:

```
SELECT employee_id, manager_id, department_id
FROM employees
WHERE last_name IN ('Harstein', 'Vargas');
```

Pattern Matching Using the `LIKE` Operator

- Use the `LIKE` operator to perform wildcard searches of valid search string values
- Search conditions can contain either literal characters or numbers:
  - `%` denotes zero or many characters
  - `_` denotes one character

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%';
```

The `SELECT` statement will returns the first name from the `EMPLOYEES` table for any employee whose first name begins with the letter "S". Note the uppercase "S". Consequently, names beginning with a lowercase "s" are not returned.

The `LIKE` operator can be used as a shortcut for some `BETWEEN` comparisons. The following example displays the last names and hire dates of all employees who joined between January 1995 and December 1995:

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date LIKE '%95'
```

Combining Wildcard Characters

- You can combine the two wildcard characters (%,_) with literal characters for pattern matching:

```
SELECT last_name
FROM employees
WHERE last_name LIKE '_o%';
```

This statement will displays the names of all employees whose last names have the letter "o" as the second character.

- You can use the ESCAPE identifier to search for the actual % and _ symbols:

```
SELECT last_name, job_id
FROM employees
WHERE job_id LIKE 'SA\_%' ESCAPE '\';
```

This option specifies what the escape character is. (e.g. If you want to search for strings that contain `SA_`).

Using the NULL Conditions

```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL;
```

- The `NULL` conditions include the `IS NULL` and the `IS NOT NULL` condition
- The `IS NULL` condition tests for nulls. A null value means that the value is unavailable, unassigned, unknown or inapplicable. Therefore, you cannot test with `=`, because a null cannot be equal or unequal to any value. The example above retrieves the last names and managers of all employees who do not have a manager.
- Here is another example: To display the last name, job ID, and commission for all employees who are not entitled to receive a commission:

```
SELECT last_name, job_id, commission_pct
FROM employees
WHERE commission_PCT IS NULL;
```

Using the `AND` Operator

`AND` requires both the component conditions to be true:

```
SELECT employee_id, last_name, salary
FROM employees
WHERE salary >= 10000
AND job_id LIKE '%MAN%';
```

Both the component conditions must be true for any record to be selected. Therefore only those employees who have a job title that contains the string 'MAN' *and* earn $10,000 or more are selected.

Using the OR Operator

OR requires either component condition to be true

```
SELECT employee_id, last_name, salary
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%';
```

Either component condition can be true for any record to be selected. Therefore, any employees who have a job ID that contains the string 'MAN' or earns $10,000 *or* more are selected.

Using the NOT Operator

```
SELECT last_name, job_id
FROM employees
WHERE job_id NOT IN ('IT_PROG','ST_CLERK', 'SA_REP')
```

The statements will displays the last name and job ID of all employees whose job ID *is not* IT_PROG, ST_CLERK or SA_REP.

Rules of Precedence

The rules of precedence determine the order in which expression are evaluated and calculated. The table below lists the default order of precedence. However, the default order can be overridden by using parenthesis around the expressions that you want to calculate first.

Order	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

Example:

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR    job_id = 'AD_PRES'
AND   salary > 15000;
```

There are two conditions in this example:

- The first condition is that the job ID is AD_PRESS and the salary is greater than $15,000
- The second condition is that the job ID is SA_REP

Therefore the SELECT statement reads as follows:

"Select the row if an employee is a president *and* earns more than $15,000, *or* if the employee is a sales representative"

Using parenthesis example:

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR    job_id = 'AD_PRES')
AND   salary > 15000;
```

There are two conditions in this example:

- The first condition is that the job ID is AD_PRESS or SA_REP
- The second condition is that the salary is greater than $15,000

Therefore the SELECT statement reads as follows:

"Select the row if an employee is a president *or* a sales representative, *and* if the employee earns more than $15,000"

Using the `ORDER BY` Clause

- Sort the retrieved rows with the ORDER BY clause:
  - ASC: Ascending order, default
  - DESC: Descending order
- The ORDER BY clause comes last in the SELECT statement.

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date;
```

Note: Use the keywords NULLS FIRST or NULLS LAST to specify whether returned rows containing null values should appear first or last in the ordering sequence.

- Sorting in descending order

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date DESC;
```

- Sorting by column alias

```
SELECT last_name, job_id, department_id, salary*12 annsal
FROM employees
ORDER BY annsal;
```

- Sorting by using column's numeric position

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY 3;
```

- Sorting by multiple column

```
SELECT last_name, job_id, department_id, salary
```

```
FROM employees
ORDER BY department_id, salary DESC;
```

Substitution Variables

- Use substitution variables to:
  - Temporarily store values with single-ampersand (&) and double-ampersand(&&) substitution
- Use substitution variables to supplement the following:
  - WHERE conditions
  - ORDER BY clause
  - Column expressions
  - Table names
  - Entire SELECT statements

Using the Single-Ampersand Substitution Variable

Use a variable prefixed with an ampersand (&) to prompt the user for value:

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num;
```

Character and Date Values with Substitution Variable

Use single quotation marks for date and character values:

```
SELECT last_name, department_id, salary*12
FROM employees
WHERE job_id = '&job_title';
```

Specifying Column Names, Expressions, and Text

```
SELECT employee_id, last_name, job_id, &column_name
FROM employees
WHERE &condition
ORDER BY &order_column;
```

Using the Double-Ampersand Substitution Variable

Use double ampersand (&&) if you want to reuse the variable value without prompting the user each time:

```
SELECT employee_id, last_name, job_id, &&column_name
FROM employees
ORDER BY &column_name;
```

Using the `DEFINE` Command

- DEFINE – create and assign a value to a variable
- UNDEFINE – remove a variable

```
DEFINE employee_num = 200
```

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE employee_id = &employee_num;

UNDEFINE employee_num
```

Exercise : Complete the following exercises:

Instructions: Save all your statement as a <matric_no>_lab4_<exercise_no>.sql script.

1. The HR department needs a report that displays the last name and salary of employees who earn more than $12,000. Save your SQL statement as a file named matricno_lab_04_01.sql.
2. Create a report that displays the last name and department number for employee number 176.
3. The HR department needs to find high-salary and low-salary employees. Modify lab_04_01.sql to display the last name and salary for any employee whose salary is not in the range of $5,000 to $12,000.
4. Create a report to display the last name, job ID, and hire date for employees with the last names of Matos and Taylor. Order the query in ascending order by the hire date.
5. Display the last name and department ID of all employees in departments 20 or 50 in ascending alphabetical order by name.
6. Modify lab_04_03.sql to display the last name and salary on employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary, respectively. Save as lab_04_06.sql
7. The HR department needs a report that displays the last name and hire date for all employees who were hired in 1994.
8. Create a report to display the last name and job title of all employees who do not have a manager.
9. Create a report to display the last name, salary, and commission of all employees who earn commission. Sort data in descending order of salary and commissions. Use the column's numeric position in the ORDER BY clause.
10. Members of the HR department want to have more flexibility with the queries that you are writing. They would like a report that displays the last name and salary or employees who earn more than an amount that the user specifies after a prompt. Save this query to a file named lab_04_10.sql. If you enter 12000 when prompted, the report displays the following results:

	LAST_NAME	SALARY
1	Hartstein	13000
2	King	24000
3	Kochar	17000
4	De Haan	17000

11. The HR department wants to run reports based on a manager. Create a query that prompts the user for a manager ID and generates the employee ID, last name, salary and department for that manager's employees. The HR department wants the ability to sort the report on a selected column.
12. Display all employee last names in which the third letter of the names is "a"

13. Display the last names of all employees who have both an “a” and an “e” in their last name.
14. Display the last name, job, and salary for all employees whose jobs are either those of a sales representative or of a stock clerk, and whose salaries are not equal to $2,500, $3,500, or $7,000.
15. Modify lab_04_06.sql to display last name, salary, and commission for all employees whose commission is 20%.