RELATIONAL DATABASES

SELECT STATEMENT

OBJECTIVES

- Apply SQL syntax to restrict the rows returned from a query
- Demonstrate application of the WHERE clause syntax
- Explain why it is important, from a business perspective, to be able to easily limit data retrieved from a table
- Construct and produce output using a SQL query containing character strings and date values
- Apply comparison operators
- BETWEEN, IN, LIKE
- IS NULL, IS NOT NULL

SELECT STATEMENT

- The SELECT statement allows you to get data from tables in your database.

```
SELECT
    column_1, column2, …
FROM
    table_1
[INNER|LEFT|right] JOIN table_2 ON conditions
WHERE
    conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1;
```

SELECT STATEMENT

```
SELECT
    column_1, column2, …
FROM
    table_1
[INNER|LEFT|right] JOIN table_2 ON conditions
WHERE
    conditions
GROUP BY column_1
HAVING group_conditions
ORDER BY column_1;
```

Each select statement must have a select clause, and a from clause.

All other clauses are optional.

Order by clause is always last

The table names must match tables in the databases.

The column names must match columns on those tables.

SELECT STATEMENT

```
SELECT student_id, last_name, first_name
FROM graduates;
```

This query would return all of the students you inserted into your graduates table created in lab8.

What if we only wanted to see the students that have more than 60 credits?

WHERE CLAUSE

- When retrieving data from the database, you may need to limit the rows of data that are displayed.
- You can accomplish this using the WHERE clause.
- A WHERE clause contains a condition that must be met, and directly follows the FROM clause in a SQL statement.

WHERE CLAUSE – COMPARISON OPERATORS

- The following operators can be used to compare one expression to another:

= equal to

> greater than

>= greater than or equal to

< less than

<= less than or equal to

<> not equal to

!= not equal to

WHERE CLAUSE

```
SELECT student_id, last_name, first_name
FROM graduates
WHERE credits > 60;
```

This now returns only those students whose credits are more than 60.

WHERE CLAUSE

- In the example below, the department_id column is used in the WHERE clause, with the comparison operator =
- All employees with a department_id of 90 are returned.

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

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90

WHERE CLAUSE

- Character strings and dates in the WHERE clause must be enclosed in single quotation marks ‘’
- Number however, should not be enclosed in single quotation marks.

```
SELECT first_name, last_name
FROM employees
WHERE last_name = ‘Taylor’;
```

WHERE CLAUSE

- What do you think will happen if the WHERE clause is written as:

```
SELECT first_name, last_name
FROM employees
WHERE last_name = ‘jones’;
```

- All character searches are case sensitive.
- No rows are returned by this statement as all last name are stored in proper case.
- There are functions which help to avoid errors due to case, UPPER, LOWER, INITCAP

WHERE CLAUSE

- Comparison operators can be used in the following ways:

```
WHERE hire_date < ‘2000-01-01’

WHERE salary >= 6000

WHERE job_id = ‘IT_PROG’
```

WHERE CLAUSE

- What do you think will happen with the following where clause?

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

Will employees with salary of 3000 be included in the results?

BETWEEN…AND

- The BETWEEN…AND operator is used to select and display rows based on a range of values.
- When used with the WHERE clause, the BETWEEN…AND condition will return a range of values between and inclusive of the specified lower and upper limits.
- Values specified with the BETWEEN condition are said to be inclusive.
- Note also that the lower limit value must be listed first.

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 9000 AND 11000;
```

BETWEEN…AND

- Using the BETWEEN…AND is the same as using the following expression:

```
WHERE salary >= 9000 AND salary <=11000;
```

- There is no performance benefit in using one method over the other.

IN

- The IN condition is also known as the “membership condition”
- It is used to test whether a value is IN a specified set of values
- For example, IN could be used to identify students whose identification numbers are 2349, 7354, or 4333 or people whose international phone calling code is 1735, 82, or 10.
- This example shows those locations whose country id is UK or CA

```
SELECT city, state_province, country_id
FROM locations
WHERE country_id IN('UK', 'CA');
```

IN

- In this example, the WHERE clause could also be written as a set of OR conditions:

```
SELECT city, state_province, country_id
FROM locations
WHERE country_id IN('UK', 'CA');
…
WHERE country_id = 'UK' OR country_id = 'CA';
```

- Either method works efficiently

LIKE

- When you are not entirely sure what you are looking for in the database the LIKE operator allows you to select rows that match either characters, dates, or number patterns.
- Two symbols: the % and the _ (underscore) are wildcard characters, and can be used to construct a search string.
- The % is used to represent any sequence of zero or more characters.
- The _ underscore symbol is used to represent a single character.

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

LAST_NAME
Kochhar
Lorentz
Mourgos

IS NULL, IS NOT NULL

- NULL is unavailable, unassigned, unknown
- Being able to test for NULL is desirable.
- You may want to know all the dates in June that right now do not have a concert scheduled.
- You may want to know all the clients who do not have phone numbers
- IS NULL tests for unavailable, unassigned, unknown data
- IS NOT NULL tests for data

IS NULL, IS NOT NULL

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

LAST_NAME
King

- Employee King is the President of the company, so has no manager.

```
SELECT last_name, commission_pct
FROM employees
WHERE commission_pct IS NOT NULL;
```

LAST_NAME	COMMISSION_PCT
Zlotkey	.2
Abel	.3
Taylor	.2
Grant	.15

- IS NOT NULL returns the rows that have a value in the commission_pct column.

LOGICAL OPERATORS

- In SQL it is often desirable to be able to restrict the rows returned by a query based on two or more conditions.
- You may want to know the names of your staff who are either cooks or order takers.
- Conditional operators such as AND, NOT, and OR make these requests easy to do
- Logical conditions combine the result of two component conditions to produce a single result.
- For example to attend a concert you need to buy a tick and have transportation to get there.
- If both conditions are met you can go to the concert.

LOGICAL OPERATORS

- A logical operator combines the results of two or more conditions to produce a single result.
- A result is returned only if the overall result of the condition is true.
- AND – returns TRUE if both conditions are true.
- OR – returns TRUE if either condition is true.
- NOT – returns TRUE if the condition is false.

LOGICAL OPERATORS

```
SELECT last_name, department_id, salary

FROM employees

WHERE department_id > 50 AND salary > 12000;
```

LAST_NAME	DEPARTMENT_ID	SALARY
King	90	24000
Kochhar	90	17000
De Haan	90	17000

LOGICAL OPERATORS

```
SELECT last_name, hire_date, job_id

FROM employees

WHERE hire_date > ‘1998-01-01’ AND job_id LIKE ‘SA%’;
```

LAST_NAME	HIRE_DATE	JOB_ID
Zlotkey	29/Jan/2000	SA_MAN
Taylor	24/Mar/1998	SA_REP
Grant	24/May/1999	SA_REP

LOGICAL OPERATORS

- If the WHERE clause uses the OR condition, the results returned from a query will be rows that satisfy either one of the OR conditions.
- In other words, all rows returned have a location_id of 2500 OR they have a manager_id equal to 124.

```
SELECT department_name, manager_id, location_id
FROM departments
WHERE location_id = 2500 OR manager_id = 124;
```

DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
Shipping	124	1500
Sales	149	2500

LOGICAL OPERATORS

- The NOT operator will return rows that do NOT satisfy the condition in the WHERE clause.

```
SELECT department_name, location_id
FROM departments
WHERE location_id NOT IN (1700, 1800);
```

DEPARTMENT_NAME	LOCATION_ID
Shipping	1500
IT	1400
Sales	2500

PRECEDENCE

- In what order are expressions evaluated and calculated?

```
SELECT last_name || ‘ ‘ || salary * 1.05 AS “Employee Raise”,
       department_id, first_name
FROM employees
WHERE department_id IN (50, *))
AND first_name LIKE ‘C%’
OR last_name LIKE ‘%s%’;
```

PRECEDENCE

- The AND operator is evaluated before the OR operator.
- In the previous example if either of the conditions in the AND statement are not met then the OR operator is used to select the rows.

ORDER	OPERATORS
1	Arithmetic + - * /
2	Concatenation ||
3	Comparison <, <=, >, >=, <>
4	IS (NOT) NULL, LIKE, (NOT) IN
5	(NOT) BETWEEN
6	NOT
7	AND
8	OR

PRECEDENCE

- In this example, the order of the OR and AND have been reversed from the previous example.

```
SELECT last_name || ‘ ‘ || salary * 1.05 AS “Employee Raise”,
      department_id, first_name
FROM employees
WHERE department_id IN (50, *))
OR first_name LIKE ‘C%’
AND last_name LIKE ‘%s%’;
```

What is the outcome?

PRECEDENCE

- Adding parenthesis changes the way the WHERE clause is evaluated, and the rows that are returned.

```
SELECT last_name || ‘ ‘ || salary * 1.05 AS “Employee Raise”,
      department_id, first_name
FROM employees
WHERE (department_id IN (50, *))
OR first_name LIKE ‘C%’)
AND last_name LIKE ‘%s%’;
```