7086CEM

Data Management Systems

Objectives

- SQL SELECT statement
- Aggregate functions

SQL – SELECT

- Use a SELECT statement or subquery to retrieve data from one or more tables, views or materialized views.

Syntax:

SELECT <*column_list*>
FROM <*table_list*>
[WHERE <*condition_expression*>]
GROUP BY <*group_columns*>
HAVING <*group_conditional_expression*>
ORDER BY <col_name>

SQL – SELECT

Consider tables

Student (s_id, last_name, first_name, date_of_birth, c_id, grade, loan);

Course(c_id, c_name).

Example: find the name of student with id, 000001.

```
SELECT s_id, last_name, c_id
FROM Student
WHERE s_id = '000001';
```

List all the details of the students:

```
SELECT  *  FROM Student.
```

SQL - SELECT

- Comparison Operators

=	Equal to
<> or !=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

SQL - SELECT

- Comparison Operators

BETWEEN…AND…	Compares a range of values
IN	Tests against values in a list
LIKE	Compares a character pattern

Note:

1. The NOT Boolean operator can be used with the following operators to negate results:

NOT BETWEEN…AND…

NOT IN

NOT LIKE

SQL - SELECT

- List the names of all students with a loan greater than 5000:

```
SELECT s_id, last_name
FROM Student
WHERE loan > 5000;
```

- List the names of all the students who do not take ‘M26CDE’. List the course id as well:

```
SELECT s_id, last_name, c_id
FROM Student
WHERE c_id <> ‘M26CDE’;
```

SQL - SELECT

- Comparison Operators

2. The LIKE operator recognises two following character symbols:

%	represents any sequence of zero or more characters
_	represents any single character

Example 1: assume that two students' names, 'J Smith' and 'P Smith', are stored in the database. To select them:

```
SELECT last_name
FROM Student
WHERE last_name LIKE '%Smith';
```

Example 2: Assume that two students' ids, '8801' and '8802', are stored in the database.

```
SELECT s_id
FROM Student
WHERE s_id LIKE '88__';
```

SQL - SELECT

Student (s_id, last_name, first_name, date_of_birth, c_id, grade, loan);

- To select the persons with a last name alphabetically between ‘Akinwale’ and ‘Smith’ from the table above.

```
SELECT  *  FROM Student
WHERE last_name
BETWEEN  ‘Akinwale ‘ AND  ‘Smith’;
```

To select students with a grade less than 40 or greater than 50:

```
SELECT  last_name FROM Student
WHERE salary
NOT BETWEEN  40 and 50;
```

SQL - SELECT

- Boolean Operators – AND , OR

  Syntax:

  SELECT <*column_list*>
  FROM <*table_list*>
  WHERE <*expression*> <*Boolean_operator*> <*expression*>…

Boolean Operators – AND

Example: List the students who have the loan between 1000 and 5000 pounds.

```
SELECT last_name, loan
FROM Student
WHERE loan >= 1000 AND loan <= 5000;
```

SQL - SELECT

- Boolean Operators – OR

  Example: List the students whose ids are ‘8801’, ‘8802’ and ‘8803’.

```
SELECT s_id, last_name
FROM Student
WHERE s_id = ‘8801’ OR s_id = ‘8802’ OR s_id = ‘8803’;
```

This is equivalent to:

```
SELECT s_id, last_name
FROM Student
WHERE s_id  IN (‘8801’, ‘8802’, ‘8803’);
```

SQL - SELECT

Ordering the Rows of a Result

Syntax: SELECT <*column_list*>
FROM <*table_list*>
[WHERE <*condition_expression*>]
ORDER BY <*column_1*> <*specified_order*>, ...,
<*column_n*> < *specified_ order*>

Examples:

```
SELECT last_name
FROM Student
ORDER BY last_name ASC;
```

Or the equivalent default:

```
SELECT last_name
FROM Student
ORDER BY last_name;
```

SQL - SELECT

- Order Rows on Multiple Columns
  - To order the results on a list of columns, the columns are separated by a comma (,) .

Example: Consider Employee(e_id, dept_no, salary) and

List the employees' salaries in descending order within the departments where they work in ascending order.

```
SELECT e_id, dept_no, salary
FROM Employee
ORDER BY dept_no, salary DESC;
```

Find the employees who work in department D003 and list their ids in ascending order.

```
SELECT dept_no, e_id
FROM Employee
WHERE dept_no = 'D003'
ORDER BY e_id;
```

SQL – Rename Result Columns

- **Syntax:**

  SELECT <*column_name*> AS <*new_name*>
  FROM <*table_list*>

  Example: Product (p_id, name, price)

  SELECT p_id, price * 1.175 AS Price_Inc_VAT
  FROM Product;

  Results can be displayed with a more meaningful title such as:

  SELECT pi_id, price * 1.175 “Price Including VAT”
  FROM product;

SQL – Aggregate Functions

- Aggregate (Group) Functions perform a variety of actions such as counting the rows in a table, averaging a column's data and summing numeric data. They can also be used to search a table to find the highest or lowest values in a column.

- Aggregate functions return a single valued result, i.e. a calculated column with only one row.

  COUNT(column-name) returns the number of non-null values
  COUNT(*) returns the number of rows
  MAX(column-name) returns the highest value
  MIN(column-name) returns the lowest value
  SUM(column-name) calculates the total of values
  AVG(column-name) returns the average value

SQL – Aggregate Functions

Student (s_id, last_name, first_name, date_of_birth, c_id, grade, loan)

Examples of the use of aggregate functions:

1. Find the total number of students:

   SELECT COUNT(s_id) FROM Student;

2. Find the average grade of student:

   SELECT AVG(grade) "Average grade" FROM Student.

3. Find the youngest student.

   SELECT MAX(date_of_birth) FROM Student;

4. Find the total loans of 'Adam' and 'Smith'

   SELECT SUM(loan) FROM Student
   WHERE last_name IN ('Adam', 'Smith');

5. Find the number of students with a grade higher than the average:

   SELECT COUNT(s_id) FROM Student
   WHERE grade > (select AVG(grade) FROM Student);

SQL – GROUP BY

- A GROUP BY clause groups a result into subsets that have matching values for one or more columns.
- It can be used to answer questions such as: what is the highest salary in each department?. What is the average grade in each course?

Syntax:

SELECT <*column_list*>
FROM <*table_list*>
[WHERE <*condition_expression*>]
GROUP BY <*group_columns*>

SQL – GROUP BY

Consider the table:

Student (s_id, last_name, first_name, date_of_birth, c_id, grade, loan);

1. List the number of students enrolled for each course.

```
SELECT c_id, COUNT(s_id)  FROM Student
GROUP BY c_id;
```

2. List the lowest grade for each course:

```
SELECT c_id, MIN(grade)  FROM Student
GROUP BY c_id;
```

3. List the number of students for each grade:

```
SELECT grade, COUNT(s_id)  FROM Student
GROUP BY grade;
```

4. List the youngest student for each course:

```
SELECT c_id, MAX(date_of_birth)  FROM Student
GROUP BY c_id;
```

SQL - HAVING

- A HAVING clause restricts the results of a GROUP BY expression. The HAVING clause is applied to each group of the grouped table, much as a WHERE clause is applied to a select list.

  Syntax: SELECT <*column_list*>
  FROM <*table_list*>
  [WHERE <*condition_expression*>]
  GROUP BY <*group_columns*>
  HAVING <*group_conditional_expression*>

Example: Student (s_id, last_name, first_name, date_of_birth, c_id, grade, loan)

List the courses on which more than 50 students have enrolled, and whose s_id starts with '88'.

```
SELECT c_id, COUNT(s_id)
FROM Student
WHERE s_id LIKE '88__'
GROUP BY c_id
HAVING COUNT(s_id) > 50;
```

SQL - HAVING

Student (s_id, last_name, first_name, date_of_birth, c_id, grade, loan)

List the grades greater than 50 which were obtained by more than 10 students:

```
SELECT grade, COUNT(s_id)
FROM Student
WHERE grade > 50
GROUP BY grade
HAVING COUNT(s_id) > 10;
```

For each course list the number of students who obtained a grade higher than the average.

```
SELECT  c_id, COUNT(s_id)  FROM Student
WHERE grade >  (SELECT AVG(grade) FROM student)
GROUP BY  c_id;
```