CM1603 - Database Systems

Lecture 9| Retrieve Data in SQL

Dileeka Alwis – Senior Lecturer Grade II / Level Coordinator,

Department of Computing, IIT

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

Learning Outcomes

- Covers LO3 for Module - Use SQL as a data definition and data manipulation language, and to query a relational database.

- Partially covers LO4 for Module – Implement and test a relational database using a query language with a suitable interface.

- On completion of this lecture, students are expected to be able to:
  - Create DDL and DML statements
  - Use mySql to create databases and manipulate data

Lesson Outline

- Apply conditions with SQL WHERE Clause
- SQL Operators - Relational, Logical, Comparison operators
- Sort data using ORDER BY Clause
- Aggregate Functions in SQL
- Categorising data using GROUP BY Clause
- Apply conditions to the categories using HAVING Clause

SQL **WHERE** Clause

- Used to filter records that fulfill a specified condition.
- Mostly used in statements like:
  - SELECT
  - UPDATE
  - DELETE

```
SELECT <Field_Names>
FROM <Table_Name>
WHERE <Condition>
```

SQL Operators

- Relational Operators
  - Comparison operators used to test for equality, inequality, less than or grater than values
  - Mathematical operators like =, <, > , etc.

- Logical Operators
  - Evaluate whether the condition (logic) is TRUE or FALSE.
  - AND, OR, NOT

- Advanced Comparison Operators
  - IN, BETWEEN, LIKE, IS NULL etc.

Comparison Operators

Operator	Description
=	Equal to
>	Strictly greater than
>=	Greater than or equal to
<	Strictly less than
<=	Less than or equal to
<>	Not equal to
BETWEEN … AND …	Between 2 values (inclusive)
IN (SET)	Match any of the list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Logical Operators

Operator	Description
AND	Returns TRUE if **BOTH** conditions are true
OR	Returns TRUE if **EITHER** conditions are true
NOT	Returns TRUE if condition is **FALSE**

Examples

- Display ID and first name of all students whose surname is Smith.

```
SELECT StudentID, FirstName
FROM Student
WHERE Surname='Smith';
```

- Display all details of students whose first name is Tom or Mark.

```
SELECT *
FROM Student
WHERE FirstName ='Tom' OR FirstName='Mark';
```

BETWEEN Operator

- Used to selects values within a given range.
- The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN <Lowest_Value> **AND** <Highest_Value>

- Eg: Display details of students whose age is 10 – 20.

```
SELECT *
FROM Student
WHERE Age BETWEEN 10 AND 20;
```

Combining AND, OR and NOT

- Can combine AND, OR and NOT operators a single WHERE clause.
- Must use parenthesis "**( )**" to establish the order of precedence.
- If the parenthesis are not used, then the order of evaluation will be;
  1. **NOT**
  2. **AND**
  3. **OR**

- Eg: Display all details of students living in London or Manchester in England.

```
SELECT * FROM Student
WHERE Country='England' AND (City='London' OR City='Manchester');
```

IN Operator

- Allows to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

```
SELECT <Field names>
FROM <Table_Name>
WHERE <Condition>
IN (<Value1> , <Value2>, ...);
```

- Eg: Display all details of students that are from Germany, France, China and England.

```
SELECT * FROM Student
WHERE Country IN ('Germany', 'France', 'China', 'England');
```

IS NULL Operator

- Used to find whether a field does not contain a value (NULL value).
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- If expression is a NULL value, the condition evaluates to TRUE.
- If expression is not a NULL value, the condition evaluates to FALSE.

```
SELECT <Field_names>
FROM <Table_Name>
WHERE <Field_Name> IS NULL;
```

Examples

- Add a text as ‘Not given’ for the students whose address is not available in the system.

```
UPDATE Student
SET Address = ‘Not given’
WHERE Address IS NULL;
```

- Display all details of students whose DOB is available in the system.

```
SELECT * FROM Student
WHERE DOB IS NOT NULL;
```

LIKE Operator

- Used in a WHERE clause to search for a specified pattern in a field.
- There are two **wildcard characters** used in conjunction with the LIKE operator to substitute any other character(s) in a string.

% (Percentage sign):

Represents zero, one, or multiple characters

_ (Underscore):

Represents a single character

Examples

- Display all the details of students whose first name starts with ‘J’.

```
SELECT * FROM Students
WHERE FirstName LIKE ‘J%’;
```

- Display all the details of students whose surname contains ‘i’ as the third letter.

```
SELECT * FROM Students
WHERE Surname LIKE ‘_ _i%’;
```

Character List Wildcard

- Used to define sets and ranges of characters (list of characters) to match or not match.
- **[charlist]** - Defines sets and ranges of characters to match
- **[^charlist]** or **[!charlist]** - Defines sets and ranges of characters **NOT** to match

Examples

- Display all the details of students whose first name starts with ‘a’ or ‘b’ or ‘c’.

```
SELECT * FROM Students
WHERE FirstName LIKE ‘[a-c]%’;
```

- Display all the details of students whose surname ends with a vowel.

```
SELECT * FROM Students
WHERE Surname LIKE ‘%[aeiou]’;
```

ORDER BY Clause

- Used to arrange (sort) the rows in the result set according to specific criteria.
- **ASC -** Order rows in ascending order (Default)
- **DESC -** Order rows in descending order

SELECT <Field_Names>
FROM <Table_Name>
WHERE <Condition>
ORDER BY <Attribute>

Example

- Display all details of male students in ascending order of first name.

```
SELECT *
FROM Student
WHERE Gender = 'M'
ORDER BY FirstName;
```

- Display name and gender of students in ascending order of gender field and descending order of first name.

```
SELECT FirstName, Surname, Gender
FROM Student
ORDER BY Gender, FirstName DESC;
```

Aggregate Functions

- Used to provide summarization information for SQL statements, which return a single value.
  - MIN(<Attribute>)
  - MAX(<Attribute>)
  - AVG(<Attribute>)
  - SUM(<Attribute>)
  - COUNT(<Attribute>)

Function	Description
MIN	returns the smallest value in the selected column
MAX	returns the largest value in the selected column
AVG	returns the average of values in the selected column
SUM	returns the sum of values in the selected column
COUNT	returns the number of values in the selected column

- Note: when using aggregate functions, NULL values are not considered, except in COUNT(*) .

GROUP BY Clause

- This clause is used to group the result-set by one or more columns.
- Often used with aggregate functions.

```
SELECT <Field_Names>
FROM <Table_Name>
WHERE <Condition>
GROUP BY <Field_name>
```

Note: All the columns in the SELECT list that are not in group function MUST BE in the GROUP BY clause!

Examples

- Display the total marks of each student.

```
SELECT StudentID, SUM(Marks) AS Total
FROM Marks
GROUP BY StudentID;
```

- Count the number of records for each student.

```
SELECT StudentID, Count(Marks)
FROM Marks
GROUP BY StudentID;
```

StudentID	ModuleID	Marks
1001	M2	54
1002	M3	67
1003	M1	84
1001	M1	94
1001	M3	38
1002	M1	54
1003	M3	67
1001	M7	82
1002	M4	55
1003	M2	25

HAVING Clause

- Used with the GROUP BY clause to apply a filter condition to the columns that appear in the GROUP BY clause.
- If the GROUP BY clause is omitted, the HAVING clause behaves like the WHERE clause.
- HAVING clause applies the condition to each group of rows. WHERE clause applies the condition to each individual row.

- Find the students whose average is above 75.

```
SELECT StudentID, AVG(Marks) AS Average
FROM Marks
GROUP BY StudentID
HAVING AVG(Marks)>75;
```

Order of Execution

SELECT <Attribute and Function list>

FROM <Table list>

WHERE <Condition>

GROUP BY <Grouping Attribute(s)>

HAVING <Group Condition>

ORDER BY <Attribute list>

Thank you

Contact: dileeka.a@iit.ac.lk

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY