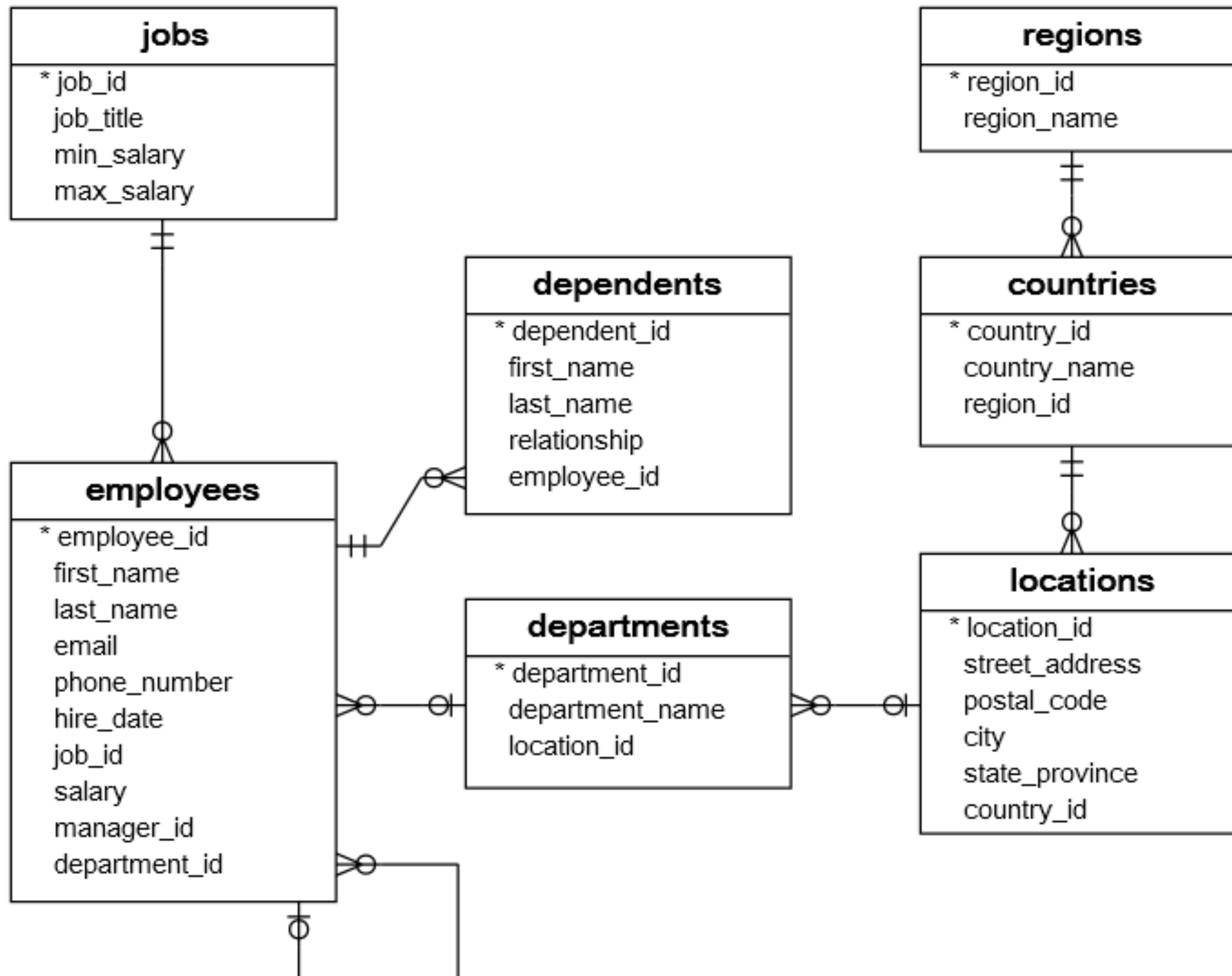
MySQL DML. SELECT Constructions 2.

Summary

- 1. HR sample database
- 2. SELECT Statement Full Syntax
- 3. SELECT - Filtering Data
    - 3.1. DISTINCT - Remove Duplicates from the result set.
    - 3.2. LIMIT / OFFSET of returned rows
    - 3.3. WHERE Clause Constructions
        - 3.3.1. Comparison operators (=, !=, <>, <, >, <=, >=)
        - 3.3.2. SQL Triple Logic Overview
            - 3.3.2.1. IS NULL operator and the NULL concepts
            - 3.3.2.2. AND operator
            - 3.3.2.3. OR operator
            - 3.3.2.4. NOT operator
        - 3.3.3. BETWEEN operator
        - 3.3.4. IN operator
        - 3.3.5. LIKE operator
- 4. SELECT - Aliases
- 5. SELECT - Sorting Data
    - ORDER BY Clause
- 6. SELECT - Grouping Data
    - GROUP BY Clause
- 7. SELECT - Filtering Group
    - HAVING Clause
- 8. SELECT - Joining Multiple Tables
    - 8.1. INNER JOIN
    - 8.2. LEFT [OUTER] JOIN
    - 8.3. RIGHT [OUTER] JOIN
    - 8.4. FULL [OUTER] JOIN
    - 8.5. [CROSS] JOIN
    - 8.6. Self Joins
- 9. SELECT UNION and UNION ALL
- 10. SELECT Subqueries

1. HR sample database.

You can use SQL Tutorial site https://www.sqltutorial.org/seeit/ for online testing examples and exercises on real DB.



2. SELECT Statement Full Syntax

To query data from a table, you use the SQL SELECT statement, where contains the syntax for selecting columns, selecting rows, grouping data, joining tables, and performing simple calculations.

```
-- Complete SELECT query
SELECT DISTINCT column, AGG_FUNC(column_or_expression), …
FROM mytable
    JOIN another_table
      ON mytable.column = another_table.column
    WHERE constraint_expression
    GROUP BY column
    HAVING constraint_expression
    ORDER BY column ASC/DESC
    LIMIT count OFFSET COUNT;
```

Each query begins with finding the data that we need in a database, and then filtering that data down into something that can be processed and understood as quickly as possible. Because each part of the query is executed sequentially, it's important to understand the order of execution so that you know what results are accessible where.

The SELECT statement is one of the most complex commands in SQL include many clauses:

- **SELECT** − This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.
- **FROM** − This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.
- **JOIN** – for querying data from one, two or multiple related tables
- **WHERE** – This clause defines predicate or conditions for filtering data based on a specified condition.
- **GROUP BY** – for grouping data based on one or more columns
- **HAVING** – for filtering groups
- **ORDER BY** – for sorting the result set
- **LIMIT** – for limiting rows returned

You will learn about these clauses in the subsequent tutorials on Practice Works PW-01, PW-02, PW-03 and PW-04.

4. SELECT Aliases

- Aliases (different name) – make your query shorter and more understandable.
- Alias use in a SELECT, DELETE, UPDATE statements.
- Alias allows you to assign a table or a column a temporary name during the execution of a query.
- There are two types of aliases:
  - **table alias**,
  - **column alias**.
- An alias only exists for the duration of the query.
- Almost all relational database management system supports both column and table aliases.

```
SELECT
    employee_id,
    concat(first_name, ' ', last_name) fullname
                                       Column alias
FROM                 Table alias
    employees e

INNER JOIN departments d ON d.department_id = e.department_id
                         Table alias
```

Aliases can be useful when:

- There are more than one table involved in a query
- Functions are used in the query
- Column names are big or not very readable
- Two or more columns are combined together

© Yuriy Shamshin

4.1. Column Aliases

When we design the tables, we often keep the column names short e.g., **so_no** for **sales order number** and **inv_no** for the **invoice number**. Then we use the SELECT statement to query the data from the table, the **output is not descriptive**.

- To assign a column a new name in a query, you use the column alias.
- The column alias is just a temporary name of the column during the execution of a query.
- If the **alias name contains space**, we have to use either sing (‘) or double (“) quotes or square brackets ([]) to surround the alias.

Syntax of the Column Alias.

- basic syntax

```
SELECT column1, column_name2 AS alias_name, column3
FROM table_name;
```

- without AS

```
SELECT column1, column_name alias_name, column3
FROM table_name;
```

- use expression

```
SELECT expression AS alias_name
FROM table_name;
```

Example 1.

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Result

ID	Customer
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería
4	Around the Horn

Example 2.

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

Result

Customer	Contact Person
Alfreds Futterkiste	Maria Anders
Ana Trujillo Emparedados y helados	Ana Trujillo
Antonio Moreno Taquería	Antonio Moreno
Around the Horn	Thomas Hardy

Example 3.

Without aliases

```
SELECT CustomerName, Address, PostalCode, City, Country
FROM Customers;
```

Result

CustomerName	Address	PostalCode	City	Country
Alfreds Futterkiste	Obere Str. 57	12209	Berlin	Germany
Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	05021	México D.F.	Mexico
Antonio Moreno Taquería	Mataderos 2312	05023	México D.F.	Mexico
Around the Horn	120 Hanover Sq.	WA1 1DP	London	UK

Use aliases. The following SQL statement creates two aliases named "Customer Name" and "Customer Address", one for the CustomerName column and one for the combine four columns (Address, PostalCode, City and Country).

```
SELECT CustomerName AS [Customer Name], Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS [Customer Address]
FROM Customers;
-- or equivalents aliases with CONCAT function
SELECT CustomerName [Customer Name], CONCAT(Address,', ',PostalCode,' ',City,', ',Country) [Customer Address]
FROM Customers;
```

Result

Customer Name	Customer Address
Alfreds Futterkiste	Obere Str. 57, 12209 Berlin, Germany
Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222, 05021 México D.F., Mexico
Antonio Moreno Taquería	Mataderos 2312, 05023 México D.F., Mexico
Around the Horn	120 Hanover Sq., WA1 1DP London, UK

4.2. Table Aliases

- We often assign a table a different name temporarily in a SELECT statement. We call the new name of the table is the table alias.
- Notice that assigning an alias does not actually rename the table. It just gives the table another name during the execution of a query.
- In practice, we keep the table alias short and easy-to-understand. For example, e for employees, d - departments, j - jobs, l - locations.

Syntax of the Table Alias.

```
SELECT column_name(s)
FROM table_name AS alias_name;
-- or without AS
SELECT column_name(s)
FROM table_name alias_name;
```

So why do we have to use the table alias?

Example 1. The first reason to use the table alias is to save time typing a lengthy name and make your query more understandable.

```
SELECT d.department_name
FROM departments AS d
```

Example 2. The second reason to use the table alias is when you want to refer to the same table multiple times in a single query. You often find this kind of query in the INNER JOIN, LEFT JOIN, and SELF-JOIN.

The following query selects data from employees and departments tables using the INNER JOIN clause. To make the query shorter, you use the table aliases, e for employees table and d for departments table as the following query:

```
SELECT employee_id, first_name, last_name, department_name
FROM employees e
INNER JOIN departments d ON d.department_id = e.department_id
ORDER BY first_name;
```

Example 3. Both column and table aliases

```
SELECT o.OrderID [Order ID], o.OrderDate [Order Date], c.CustomerName [Customer Name]
FROM Customers AS c, Orders AS o
WHERE c.CustomerName Like 'A%' AND c.CustomerID=o.CustomerID;
```

5. SELECT - Sorting Data.

5.1. ORDER BY Clause – sort the data by one or more columns in the ascending and/or descending order.

When you use the SELECT statement to query data from a table, the order of rows in the result set is undetermined or unpredictable.

- In some cases, the rows that appear in the result set are in the order that they are stored in the table physically.
- In case the ***query optimizer*** uses an index to process the query, the rows will appear as they are stored in the index key order.

To specify exactly the order of rows in the result set, you add use an ORDER BY clause in the SELECT statement after FROM clause or after other clauses (WHERE, GROUP BY, .. if exist) as follows:

```
SELECT
  column2, column1, column3
FROM
  table_name
ORDER BY column1 ASC,
         column2,
         column3 DESC;
```

To sort the result set, you specify the column in which you want to sort and the kind of the sort order: ascending (ASC) or descending (DESC).

When you include more than one column in the ORDER BY clause, the database system first sorts the result set based on the first column and then sort the sorted result set based on the second column, and so on.

If you don’t specify the sort order, the database system typically sorts the result set in ascending order (ASC) by default.

Example for DB: dependent(depId, depFname, depLname, relationship, empId) >0---have---|- employees(empId, empFname, empLname, salary)

```
SELECT *
FROM
  employees
ORDER BY
  empFname, empLname;
```

© Yuriy Shamshin

6. SELECT - Grouping Data

6.1. GROUP BY Clause

- Grouping is one of the most important tasks that you have to deal with while working with the databases.
- GROUP BY combine rows into groups and apply an aggregate function to each group.
- The GROUP BY clause is an optional clause of the SELECT statement.
- You often use the GROUP BY in conjunction with an aggregate function such as MIN, MAX, AVG, SUM, or COUNT to calculate a measure that provides the information for each group.

Syntax of the GROUP BY clause.

```
SELECT
	column1,
	column2,
	AGGREGATE_FUNCTION (column3)
FROM
	table1
GROUP BY
	column1,
	column2;
```

- It is **not mandatory** to include an aggregate function in the SELECT clause.
- However, if you use an aggregate function (COUNT, MIN, MAX, AVG, SUM), it will calculate the summary value for each group.
- If you want to filter the **rows** before grouping, you add a WHERE clause.
- However, to filter **groups**, you use the HAVING clause.
- It is important to emphasize that the WHERE clause is applied before rows are grouped whereas the HAVING clause is applied after rows are grouped. In other words, the WHERE clause is applied to rows whereas the HAVING clause is applied to groups.
- To sort the groups, you add the ORDER BY clause after the GROUP BY clause.
- The columns that appear in the GROUP BY clause are called *grouping columns*. If a grouping column contains NULL values, all NULL values are summarized into a single group because the GROUP BY clause considers NULL values are equal.

© Yuriy Shamshin

Example 1. Find the headcount of each department (SQL GROUP BY with COUNT() function):

```
SELECT department_id,
  COUNT(employee_id) as EmployeesCount
FROM  employees;
```

department_id	EmployeesCount
11	40

```
SELECT department_id,
  COUNT(employee_id) as EmployeesCount
FROM  employees
GROUP BY department_id;
```

department_id	EmployeesCount
1	1
2	2
3	6
...	...

Example 2. Find the min salary of each department (SQL GROUP BY with MIN() function):

```
SELECT department_id, MIN(salary)
FROM  employees;
```

department_id	MIN(salary)
3	2500

```
SELECT department_id, MIN(salary)
FROM  employees
GROUP BY department_id;
```

department_id	MIN(salary)
1	4400
2	6000
3	2500
...	...

Example 3. Find the max salary of each department (SQL GROUP BY with MAX() function):

```
SELECT department_id, MAX(salary)
FROM  employees;
```

department_id	SUM(salary)
9	24000

```
SELECT department_id, MAX(salary)
FROM  employees
GROUP BY department_id;
```

department_id	SUM(salary)
1	4400
2	13000
3	11000
...	...

Example 4. Find the average salary of each department (SQL GROUP BY with AVG() function):

```
SELECT department_id, AVG(salary)
FROM  employees;
```

department_id	AVG(salary)
11	8600

```
SELECT department_id, AVG(salary)
FROM  employees
GROUP BY department_id;
```

department_id	AVG(salary)
1	4400
2	9500
3	4150
…	…

Example 5. Find the sum salary of each department (SQL GROUP BY with SUM() function):

```
SELECT department_id, SUM(salary)
FROM  employees;
```

department_id	SUM(salary)
11	322400

```
SELECT department_id, SUM(salary)
FROM  employees
GROUP BY department_id;
```

department_id	SUM(salary)
1	4400
2	19000
3	24900
…	…

Example 6. Find the salary sum of each department and order by salary sum (SQL GROUP BY with SUM() function and ORDER BY):

```
SELECT department_id, SUM(salary)
FROM employees
GROUP BY  department_id
ORDER BY  sum(salary);
```

department_id	SUM(salary)
1	4400
4	6500
7	10000
…	…

Example 7. Find the salary sum of each department and order by salary sum (SQL GROUP BY with SUM() function and ORDER BY DESC):

```
SELECT department_id, SUM(salary)
FROM employees
GROUP BY  department_id
ORDER BY  sum(salary) DESC;
```

department_id	SUM(salary)
9	58000
8	57700
10	51600
…	…

© Yuriy Shamshin

Difference between ORDER BY and GROUP BY clause:

NR	GROUP BY	ORDER BY
1.	Group by statement is used to group the rows that have the same value.	Whereas Order by statement sort the result-set either in ascending or in descending order.
2.	It may be allowed in CREATE VIEW statement.	While it does not use in CREATE VIEW statement.
3.	In select statement, it is always used before the order by keyword.	While in select statement, it is always used after the order by keyword.
4.	Attribute cannot be in the group by statement under aggregate function.	Whereas in order by statement, attribute can be under aggregate function.
5.	In group by clause, the tuples are grouped based on the similarity between the attribute values of tuples.	Whereas in order by clause, the result-set is sorted based on ascending or descending order.
6.	Group by controls the presentation of tuples(rows).	While order by clause controls the presentation of columns.

7. SELECT - Filtering Group

7.1. HAVING Clause

- The HAVING clause is often used with the GROUP BY clause in the SELECT statement to specify a condition for filtering groups.
- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
- The WHERE clause is applied to rows whereas the HAVING clause is applied to groups.
- If you use a HAVING clause without a GROUP BY clause, the HAVING clause behaves like the WHERE clause.

Syntax of the HAVING clause:

```
SELECT
      column1,
      column2,
      AGGREGATE_FUNCTION (column3)
FROM
      table1
WHERE
      column_condition
GROUP BY
      column1,
      column2
HAVING
      group_condition;
```

- The WHERE clause applies the condition to individual rows before the rows are summarized into groups by the GROUP BY clause.
- However, the HAVING clause applies the condition to the groups after the rows are grouped into groups.
- Therefore, it is important to note that the HAVING clause is applied after whereas the WHERE clause is applied before the GROUP BY clause.

© Yuriy Shamshin

Example 1. HAVING with COUNT function

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

COUNT(CustomerID)	Country
9	Brazil
11	France
11	Germany
7	UK
13	USA

Example 2. HAVING with SUM function

The following statement calculates the sum of salary that the company pays for each department and selects only the departments with the sum of salary between 20000 and 30000.

```
SELECT department_id, SUM(salary)
FROM employees
GROUP BY department_id
HAVING SUM(salary) BETWEEN 20000 AND 30000
ORDER BY SUM(salary);
```

department_id	SUM(salary)
11	20300.00
3	24900.00
6	28800.00

Example 3. HAVING with MIN function

To find the department that has employees with the lowest salary greater than 10000, you use the following query:

```
SELECT
     e.department_id,
     department_name,
     MIN(salary)
FROM employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY e.department_id
HAVING MIN(salary) >= 10000
ORDER BY MIN(salary);
```

department_id	department_name	MIN(salary)
7	Public Relations	10000.00
9	Executive	17000.00

Difference between Where and Having Clause in SQL:

Nr.	WHERE Clause	HAVING Clause
1.	is used to filter the records from the table based on the specified condition.	is used to filter record from the groups based on the specified condition.
2.	can be used without GROUP BY Clause	cannot be used without GROUP BY Clause
3.	implements in row operations	implements in column operation
4.	cannot contain aggregate function	can contain aggregate function
5.	can be used with SELECT, UPDATE, DELETE statement.	can only be used with SELECT statement.
6.	is used before GROUP BY Clause	is used after GROUP BY Clause
7.	is used with single row function like UPPER, LOWER etc.	is used with multiple row function like SUM, COUNT etc.

© Yuriy Shamshin