Database Design and Query

Purwadhika
Startup and Coding School

Section 4 : Modifying Data

- Insert Data
- Update Data
- Delete Data

Insert Statement

- The MySQL **INSERT** statement allows you to insert one or more rows into a table. The following illustrates the syntax of the **INSERT** statement:

```
1 INSERT INTO table(column1,column2,...)
2 VALUES (value1,value2,...);
```

- First, you specify the table name and a list of comma-separated columns inside parentheses after the **INSERT INTO** clause.
- Then, you put a comma-separated values of the corresponding columns inside the parentheses followed the **VALUES** keyword.

Purwadhika
Startup and Coding School

Update Statement

- We use the **UPDATE** statement to update existing data in a table. We can use the **UPDATE** statement to change column values of a single row, a group of rows, or all rows in a table.

```
1 UPDATE [LOW_PRIORITY] [IGNORE] table_name
2 SET
3     column_name1 = expr1,
4     column_name2 = expr2,
5     ...
6 WHERE
7     condition;
```

Purwadhika
Startup and Coding School

- First, you specify the table name that you want to update data after the **UPDATE** keyword.
- Second, the **SET** clause specifies which column that you want to modify and the new values. To update multiple columns, you use a list comma-separated assignments. You supply the value in each column's assignment in the form of a literal value, an expression, or a subquery.
- Third, you specify which rows will be updated using a condition in the **WHERE** clause. The **WHERE** clause is optional. If you omit the **WHERE** clause, the **UPDATE** statement will update all rows in the table.
- Notice that the **WHERE** clause is so important that you should not forget. Sometimes, you may want to change just one row; However you forget the **WHERE** clause and accidentally updates all the rows in the table.
- MySQL supports two modifiers in the UPDATE statement.
  - The **LOW_PRIORITY** modifier instructs the **UPDATE** statement to delay the update until there is no connection reading data from the table. The **LOW_PRIORITY** takes effect for the storage engines that use table-level locking only, for example, MyISAM, MERGE, MEMORY.
  - The **IGNORE** modifier enables the **UPDATE** statement to continue updating rows even if errors occurs. The rows that cause errors such as duplicate-key conflicts are not updated.

Delete Statement

- To remove data from a table, you use the MySQL DELETE statement. The MySQL DELETE statement allows you to remove records from not only one table but also multiple tables using a single DELETE statement.

```
1 DELETE FROM table
2 [WHERE conditions] [ORDER BY ...] [LIMIT rows]
```

Purwadhika
Startup and Coding School

Section 5 : Querying Data

- Select Data
- Filtering Data
- Sorting Data
- Grouping Data
- Joining Tables
- Grouping
- Sub Query
- Union

Purwadhika
Startup and Coding School

Select Statement

- The **SELECT** statement allows you to get the data from tables. A table consists of rows and columns like a spreadsheet. Often, you want to see a subset rows, a subset of columns, or a combination of two. The result of the **SELECT** statement is called a result set that is a list of rows, each consisting of the same number of columns.

- Sample DB : http://bit.ly/fasttrack-db (resource/sampledatabase.sql)

Purwadhika
Startup and Coding School

Complete Select Statement

```
1  SELECT
2      column_1, column_2, ...
3  FROM
4      table_1
5  [INNER | LEFT |RIGHT] JOIN table_2 ON conditions
6  WHERE
7      conditions
8  GROUP BY column_1
9  HAVING group_conditions
10 ORDER BY column_1
11 LIMIT offset, length;
```

Purwadhika
Startup and Coding School

Complete Select Statement

- **SELECT** followed by a list of comma-separated columns or an asterisk (*) to indicate that you want to return all columns.
- **FROM** specifies the table or view where you want to query the data.
- **JOIN** gets data from other tables based on certain join conditions.
- **WHERE** filters rows in the result set.
- **GROUP BY** groups a set of rows into groups and applies aggregate functions on each group.
- **HAVING** filters group based on groups defined by GROUP BY clause.
- **ORDER BY** specifies a list of columns for sorting.
- **LIMIT** constrains the number of returned rows.

Simple Example

- Even though there are many columns in products table, the SELECT statement just returns data of selected columns.

```
1 SELECT productCode, productName FROM products
```

- If you want to get data for all columns in the employees table, you can list all column names in the SELECT clause. Or you just simply use the asterisk (*) to indicate that you want to get data from all columns of the table

```
1 SELECT * FROM products
```

Purwadhika
Startup and Coding School

Distinct Operator

- When querying data from a table, you may get duplicate rows. In order to remove these duplicate rows, you use the **DISTINCT** clause in the SELECT statement.

```
SELECT DISTINCT
    columns
FROM
    table_name
WHERE
    where_conditions;
```

Purwadhika
Startup and Coding School

Filtering Data Using “Where” Clause

- If you use the **SELECT** statement to query the data from tables without the **WHERE** clause, you will get all rows in the tables that may be not necessary. The tables accumulate data from business transactions all times. It does not make sense to get all rows from a table especially for big tables like employees, sales orders, purchase orders, production orders, etc., because we often want to analyze a set of data at a time e.g, sales of this quarter, sales of this year compared to last year, etc.

- The **WHERE** clause allows you to specify exact rows to select based on a particular filtering expression or condition.

Filtering Data Using “Where” Clause

```sql
SELECT
    lastname, firstname, jobtitle
FROM
    employees
WHERE
    jobtitle = 'Sales Rep' AND officeCode = 1;
```

Purwadhika
Startup and Coding School

- The following table lists the comparison operators that you can use to form filtering expressions in the WHERE clause.

Operator	Description
=	Equal to. You can use it with almost any data types.
<> or !=	Not equal to.
<	Less than. You typically use it with numeric and date/time data types.
>	Greater than.
<=	Less than or equal to
>=	Greater than or equal to

“And” Operator

- The AND operator is a logical operator that combines two or more Boolean expressions and returns true only if both expressions evaluate to true. The AND operator returns false if one of the two expressions evaluate to false

```
WHERE boolean_expression_1 AND boolean_expression_2
```

- The following illustrates the results of the AND operator when combining true, false, and null.

	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Purwadhika
Startup and Coding School

“Or” Operator

- The OR operator combines two Boolean expressions. It returns true when either condition is true.

```
WHERE boolean_expression_1 OR boolean_expression_2
```

- The following illustrates the results of the OR operator when combining true, false, and null.

	TRUE	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

“In” Operator

- The IN operator allows you to determine if a specified value matches any one of a list or a subquery. The following illustrates the syntax of the IN operator.

```
SELECT
    column1,column2
FROM
    table_name
WHERE
 (expr|column_1) IN ('value1','value2',...);
```

Purwadhika
Startup and Coding School

- You can use a column or an expression (expr) with the **IN** operator in the **WHERE** clause.
- The values in the list must be separated by a comma (,).
- The IN operator can also be used in the **WHERE** clause of other statements such as **INSERT**, **UPDATE**, **DELETE**, etc.

```
1 SELECT
2     officeCode, city, phone, country
3 FROM
4     offices
5 WHERE
6     country IN ('USA' , 'France');
```

Purwadhika
Startup and Coding School

“Like” Operator

- The **LIKE** operator is commonly used to select data based on patterns. Using the **LIKE** operator in the right way is essential to increase the query performance.
- The **LIKE** operator allows you to select data from a table based on a specified pattern. Therefore, the **LIKE** operator is often used in the **WHERE** clause of the **SELECT** statement.
- MySQL provides two wildcard characters for using with the **LIKE** operator, the **percentage (%)** and **underscore (_)** .

Percentage (%) wildcard

- The **percentage (%)** wildcard allows you to match any string of zero or more characters.

```
WHERE firstName LIKE '%a';
```

```
WHERE firstName LIKE 'a%';
```

```
WHERE firstName LIKE '%a%';
```

Purwadhika
Startup and Coding School

Underscore (_) wildcard

- The **underscore (_)** wildcard allows you to match any single character.

```
WHERE firstName LIKE 'T_m';
```

Purwadhika
Startup and Coding School

“IS NULL” Operator

- To test whether a value is NULL or not, you use the **IS NULL** operator. The following show syntax of the IS NULL operator:

```
WHERE value IS NULL
```

Purwadhika
Startup and Coding School

Sorting Data Using “Order By” Clause

- When you use the SELECT statement to query data from a table, the result set is not sorted in any orders. To sort the result set, you use the ORDER BY clause. The ORDER BY clause allows you to:
  - Sort a result set by a single column or multiple columns.
  - Sort a result set by different columns in ascending or descending order.

```
1 SELECT column1, column2
2 FROM tbl
3 ORDER BY column1 [ASC|DESC], column2 [ASC|DESC]
```

Purwadhika
Startup and Coding School

Grouping Data

- The **GROUP BY** clause, which is an optional part of the **SELECT** statement, groups a set of rows into a set of summary rows by values of columns or expressions. The **GROUP BY** clause returns one row for each group. In other words, it reduces the number of rows in the result set.
- We often use the **GROUP BY** clause with aggregate functions such as **SUM**, **AVG**, **MAX**, **MIN**, and **COUNT**. The aggregate function that appears in the **SELECT** clause provides the information about each group.

```
1 SELECT
2     c1, c2,..., cn, aggregate_function(ci)
3 FROM
4     table
5 WHERE
6     where_conditions
7 GROUP BY c1 , c2,...,cn;
```

Purwadhika
Startup and Coding School

Grouping Data Example

```sql
SELECT
    status, COUNT(*)
FROM
    orders
GROUP BY status;
```

Purwadhika
Startup and Coding School

Having Clause

- The MySQL **HAVING** clause is used in the **SELECT** statement to specify filter conditions for a group of rows or aggregates.
- The MySQL **HAVING** clause is often used with the **GROUP BY** clause. When using with the **GROUP BY** clause, we can 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.
- Notice that the **HAVING** clause applies the filter condition to each group of rows, while the **WHERE** clause applies the filter condition to each individual row.

Having Clause Example

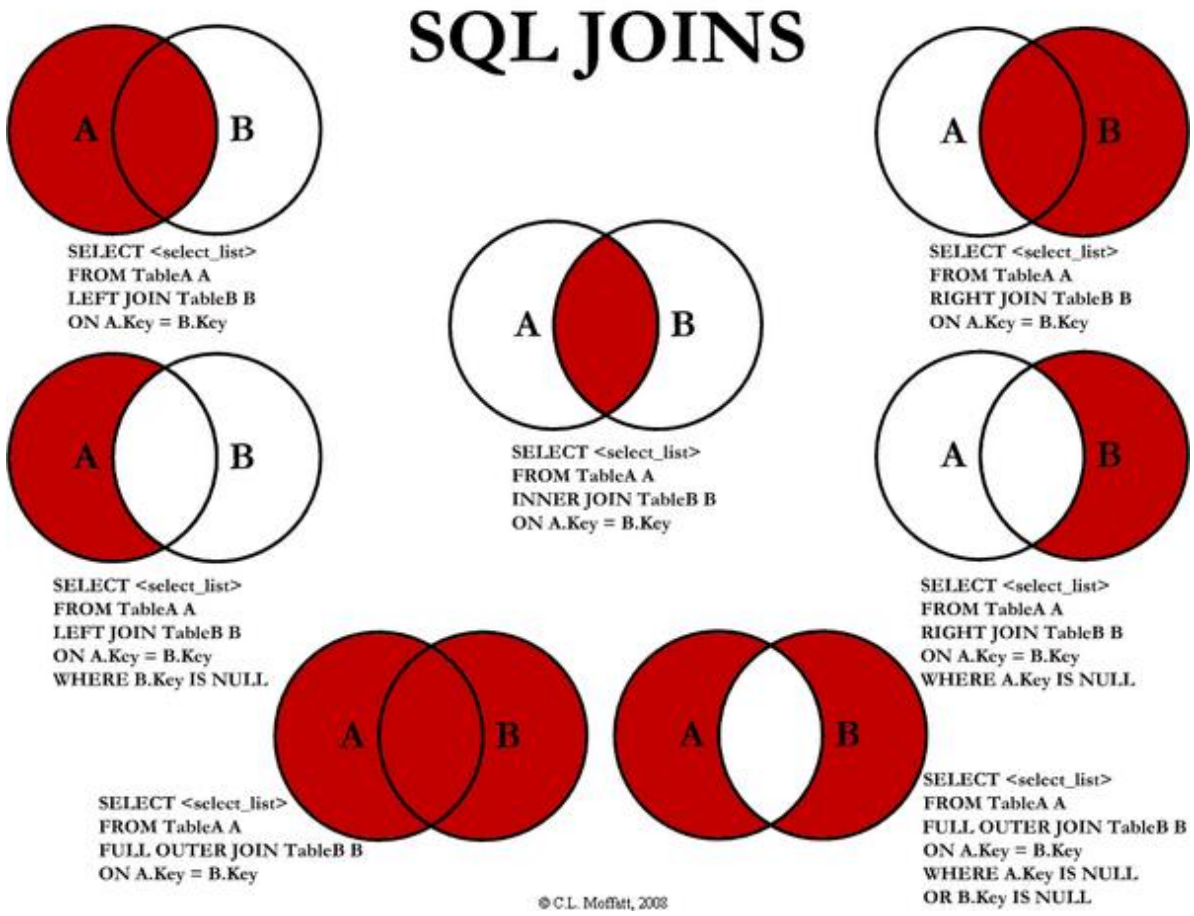
```sql
SELECT
    ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach) AS total
FROM
    orderdetails
GROUP BY ordernumber
HAVING total > 1000;
```

Purwadhika
Startup and Coding School

Joining Tables

- **JOIN** clause is used to combine rows from two or more tables, based on a common field between them.
- The most common type of join is: **INNER JOIN (simple join)**. An **INNER JOIN** returns all rows from multiple tables where the join condition is met.

Joining Tables



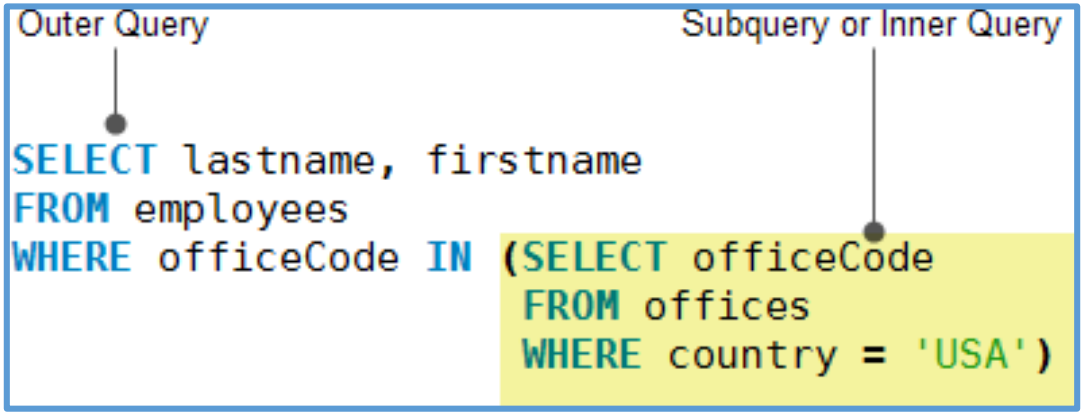
Purwadhika
Startup and Coding School

Subquery

- A MySQL **Subquery** is a query that is nested inside another query such as **SELECT**, **INSERT**, **UPDATE** or **DELETE**. In addition, a MySQL **Subquery** can be nested inside another **Subquery**.
- A MySQL **Subquery** is also called an inner query while the query that contains the **Subquery** is called an outer query.

Subquery

- Let's take a look at the following subquery that returns employees who locate in the offices in the USA.
  - The subquery returns all offices codes of the offices that locate in the USA.
  - The outer query selects the last name and first name of employees whose office code is in the result set returned by the subquery.



Purwadhika
Startup and Coding School

Union Operator

- MySQL **UNION** operator allows you to combine two or more result sets from multiple tables into a single result set. The syntax of the MySQL **UNION** is as follows:

```
1 SELECT column1, column2
2 UNION [DISTINCT | ALL]
3 SELECT column1, column2
4 UNION [DISTINCT | ALL]
```

Union Operator

- There are some rules that you need to follow in order to use the **UNION** operator:
  - The number of columns appears in the corresponding **SELECT** statements must be equal.
  - The columns appear in the corresponding positions of each **SELECT** statement must have the same data type or, at least, convertible data type.
- By default, the **UNION** operator eliminates duplicate rows from the result even if you don't use **DISTINCT** operator explicitly. Therefore, it is said that **UNION** clause is the shortcut of the **UNION DISTINCT**.
- If you use the UNION ALL explicitly, the duplicate rows, if available, remain in the result. The **UNION ALL** performs faster than the **UNION DISTINCT**.