MySQL

- The most popular open source SQL database management system, is developed, distributed, and supported by Oracle Corporation.
- It is written in **C** and **C++**.
- It is named after co-founder Monty Widenius's daughter, **My**.
- The name of the MySQL Dolphin is **Sakila**.

Comments

- From a **#** character to the end of the line.
- From a **- - <SPACE>** sequence to the end of the line.
- From a **/* sequence to the following */** sequence that is multiline comments.

Data Types

[ ] code a likha lagbe na
[ ] er vitorer jinis lagbe / optional

Type	Details
BIT[(M)] - BIT, BIT(30)	▪ The default is 1 if M is omitted. ▪ M indicates the number of bits per value, from 1 to 64.
BOOL **BOOLEAN**	▪ Zero is considered **false** and nonzero values are considered **true**.
TINYINT[(M)] [UNSIGNED]	▪ 1 byte
SMALLINT [(M)] [UNSIGNED]	▪ 2 bytes
MEDIUMINT [(M)] [UNSIGNED]	▪ 3 bytes
INT[(M)] [UNSIGNED] - INT, INT(30), INT UNSIGNED	▪ **4 bytes**
BIGINT[(M)] [UNSIGNED]	▪ 8 bytes Here, M indicates the maximum display width (M <= 255)
FLOAT[(M,D)] [UNSIGNED]	▪ A small (single-precision) floating-point number. ▪ M is the total number of digits and D is the number of digits following the decimal point. If M and D are omitted, values are stored to the limits permitted by the hardware. ▪ The decimal point and the -ve sign are not counted in M. ▪ Permissible values are: • -3.402823466E+38 to -1.175494351E-38, 0, and • 1.175494351E-38 to 3.402823466E+38.
DOUBLE[(M,D)] [UNSIGNED] - DOUBLE, DOUBLE(10,3)	▪ A normal-size (double-precision) floating-point number. ▪ M is the total number of digits and D is the number of digits following the decimal point. If M and D are omitted, values are stored to the limits permitted by the hardware. ▪ The decimal point and the -ve sign are not counted in M. ▪ Permissible values are: • -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and • 2.2250738585072014E-308 to 1.7976931348623157E+308.

Decimal [(M, D)] [UNSIGNED]	▪ A packed “exact” fixed-point number. ▪ It is used when it is important to preserve exact precision, for example with monetary data. ▪ M (max 65) is the total number of digits (the precision) and D (max 30) is the number of digits after the decimal point (the scale). ▪ The decimal point and (for negative numbers) the - sign are not counted in M.
DATE	▪ ‘YYYY-MM-DD’
TIME	▪ ‘hh:mm:ss’
DATETIME	▪ ‘YYYY-MM-DD hh:mm:ss’
YEAR	▪ ‘YYYY’
CHAR[(M)] - CHAR, CHAR(10) **BINARY[(M)]**	▪ A fixed-length string that is always right-padded with spaces to the specified length when stored. M represents the column length in characters. The range of M is 0 to 255. If M is omitted, the length is 1 ▪ Binary byte string.
VARCHAR(M) - VARCHAR(20) **VARBINARY**	▪ A variable-length string. M represents the maximum column length in characters. The range of M is 0 to 65,535. ▪ Binary byte string (variable-length).
LONGTEXT **LONGBLOB**	▪ A TEXT column with a maximum length of 4,294,967,295 or 4GB (2^{32} – 1) characters. ▪ A BLOB column with a maximum length of 4,294,967,295 or 4GB (2^{32} – 1) bytes.
ENUM(‘val1’, ‘val2’, …)	▪ An enumeration. A string object that can have only one value, chosen from the list of values or NULL. ▪ It can have a maximum of 65535 distinct elements.

Constraints

Constraints	**Description**
NOT NULL	▪ In MySQL, NOT NULL constraint allows to specify that a column can not contain any NULL value.
DEFAULT def_value	▪ Sets a default value for a column when no value is specified. Ex: • DEFAULT 0 • DEFAULT (RAND() * RAND()) • DEFAULT (CURRENT_TIMESTAMP) • DEFAULT (CURRENT_TIMESTAMP) ON UPDATE CURRENT_TIMESTAMP
UNIQUE	▪ The UNIQUE index constraint in MySQL does not allow to insert a duplicate value in a column.
CHECK (expr)	▪ The CHECK clause enables the creation of constraints to be checked for data values in table rows. ▪ CHECK constraints are prohibited on columns used in foreign key referential actions. ▪ CHECK constraints are evaluated for INSERT, UPDATE, REPLACE, LOAD DATA statements and an error (warning) occurs if a constraint evaluates to FALSE.
PRIMARY KEY	▪ A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
FOREIGN KEY	▪ A FOREIGN KEY in MySQL creates a link between two tables by one(or more) specific column of both tables. ▪ The specified column in one table must be a PRIMARY KEY and referred by the column of another table known as FOREIGN KEY.

AUTO_INCREMENT	▪ An integer or floating-point column can have the additional attribute AUTO_INCREMENT. When you insert a value of **NULL (recommended) or 0** into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically, this is *value+1*, where *value* is the largest value for the column currently in the table. AUTO_INCREMENT sequences begin with 1. ▪ There can be only one AUTO_INCREMENT column per table, it must be indexed, and it cannot have a DEFAULT value. An AUTO_INCREMENT column works properly only if it contains only positive values.

SQL Statements

- Data Definition Statements (DDL)

- CREATE, ALTER, DROP statements

To create and delete database schema

```
1. CREATE DATABASE [IF NOT EXISTS] database_name;
- An error occurs if the database exists and you didn’t specify IF NOT EXISTS.

2. DROP DATABASE [IF EXISTS] database_name;
- IF EXISTS is used to prevent an error from occurring if the database doesn’t exist.
```

To create and delete database table

```
3. CREATE TABLE [IF NOT EXISTS] table_name(

    col1 datatype [NOT NULL] [DEFAULT def_val] [~~UNIQUE~~] [AUTO_INCREMENT] [~~PRIMARY KEY~~] [~~CHECK(expr)~~],
    col2 datatype [NOT NULL] [DEFAULT def_val] [~~UNIQUE~~] [AUTO_INCREMENT] [~~PRIMARY KEY~~] [~~CHECK(expr)~~],
    .
    .
    .
    coln datatype [NOT NULL] [DEFAULT def_val] [~~UNIQUE~~] [AUTO_INCREMENT] [~~PRIMARY KEY~~] [~~CHECK(expr)~~],

    CONSTRAINT constraint_name PRIMARY KEY(col1, col2, ... ),

    CONSTRAINT constraint_name UNIQUE(col3, col4, ... ),

    CONSTRAINT constraint_name CHECK(expr),

    CONSTRAINT constraint_name FOREIGN KEY(col1, col2, ... )
                               REFERENCES ref_tablename(ref_col1, ref_col2, ... )
                               [ON DELETE CASCADE|SET NULL|RESTRICT]
                               [ON UPDATE CASCADE|SET NULL|RESTRICT]

);
```

- By default, tables are created in the default database, using the InnoDB storage engine.
- **IF NOT EXISTS** prevents an error from occurring if the table exists.
- If the constraint names are not defined, then MySQL automatically generates a constraint name.
- **CASCADE**: delete/update the child table matching rows when delete/update the parent table rows.
- **SET NULL**: sets the foreign key column to NULL when delete/update the parent table row.
- **RESTRICT**: rejects the delete/update operation for the parent table.

```
4. DROP TABLE IF EXISTS tablename1, tablename2, ...;
- With IF EXISTS, no error occurs for nonexisting tables.
```

```
To add new column and delete existing columns in database table

5. ALTER TABLE tablename
   ADD COLUMN colname datatype [NOT NULL] [DEFAULT def_val] [UNIQUE] [AUTO_INCREMENT] [PRIMARY KEY]
                                                                                   [CHECK(expr)];
6. ALTER TABLE tablename
   DROP COLUMN colname;


To add and delete primary key

7. ALTER TABLE tablename
   ADD CONSTRAINT constraint_name PRIMARY KEY(col1, col2, ... );

8. ALTER TABLE tablename
   DROP PRIMARY KEY;


To add/delete unique key

9. ALTER TABLE tablename
   ADD CONSTRAINT constraint_name UNIQUE(col3, col4, ... );

10. ALTER TABLE tablename
    DROP INDEX unique_constr_name;


To add/delete foreign key

11. ALTER TABLE tablename
    ADD CONSTRAINT constraint_name FOREIGN KEY(col1, col2, ... )
                                   REFERENCES ref_tablename(ref_col1, ref_col2, ... )
                                   [ON DELETE CASCADE|SET NULL|RESTRICT]
                                   [ON UPDATE CASCADE|SET NULL|RESTRICT];

12. ALTER TABLE tablename
    DROP FOREIGN KEY fk_constr_name;


To add/delete default constraint

13. ALTER TABLE tablename
    ALTER COLUMN colname SET DEFAULT def_value;

14. ALTER TABLE tablename
    ALTER COLUMN colname DROP DEFAULT;


To add/delete check constraint

15. ALTER TABLE tablename
    ADD CONSTRAINT constraint_name CHECK(expr);

16. ALTER TABLE tablename
    DROP CONSTRAINT check_constr_name;
```

Operators

Bitwise

&, ~, |, ^, <<, >>

Arithmetic

DIV (integer div), / (floating point div)
- (minus), - (negative sign)
%, MOD (modulus)
+ (plus)
* (multiplication)

Logical

AND, &&
OR, ||
NOT, !
XOR

Assignment

= (to assign value)

Comparison

>, >=, <, <=, !=, <> (not equal), = (equality check), <=>

BETWEEN … AND …
NOT BETWEEN … AND …

IN(val1, val2, …)
NOT IN(val1, val2, …)

LIKE pattern
NOT LIKE pattern
here, % = 0 to many chars and _ = exactly 1 char

IS boolean
IS NOT boolean

IS NULL
IS NOT NULL

COALESCE(val1, val2, … …)

```
SELECT *
FROM employees
WHERE   DEPARTMENT_ID IN(10, 50, 100)
        AND FIRST_NAME LIKE "1%"
        AND SALARY BETWEEN 2000 AND 15000
        AND COMMISSION_PCT IS NOT NULL
        AND MANAGER_ID>0 IS TRUE
        AND LAST_NAME LIKE "___%"
```

SQL Statements

- Data Manipulation Statements (DML)
    - INSERT, UPDATE, DELETE statements

```
To insert data records into database table

1. INSERT INTO tablename[(col1, col2, col3, ... ...)] VALUES(val1, val2, val3, ... ...);
```

```
To delete data records from database table

2. DELETE FROM tablename
   WHERE condition;

To update data records in database table

3. UPDATE tablename
   SET col1=val1, col2=val2, ... ...
   WHERE condition;
```

Flow Control Operators and Functions

CASE WHEN … WHEN … ELSE … END Statements
 	
CASE
WHEN [condition] THEN result
WHEN [condition] THEN result
… …
ELSE result
END
	`SELECT  EMPLOYEE_ID,`
	`        CASE`
	`            WHEN SALARY>20000 THEN 'A'`
	`            WHEN SALARY BETWEEN 15001 AND 20000 THEN 'B'`
	`            WHEN SALARY BETWEEN 10001 AND 15000 THEN 'C'`
	`            ELSE 'D'`
	`        END AS "Salary Grade"`
	`FROM employees;`

IF(expr1, expr2, expr3)
 	
here,
If expr1 is TRUE (expr1 <> 0 and expr1 <> NULL), then IF() returns expr2.
 	
Otherwise, it returns expr3.
	`SELECT  EMPLOYEE_ID,`
	`        IF(SALARY>20000,`
	`           'A',`
	`           IF(SALARY>10000, 'B', 'C')`
	`        ) AS 'SALARY GRADE'`
	`FROM employees;`

IFNULL(expr1, expr2)
 	
here,
If expr1 is not NULL,
IFNULL() returns expr1;
otherwise it returns expr2.
	`SELECT IFNULL(NULL,10);`
	`-- Output: 10`
	
	`SELECT IFNULL(1,0);`
	`-- Output: 1`

Numeric Functions (1, .2, 3.4, -5, -6.78, +9.10, 1.2E3)

ABS(x)
- returns the absolute value of x
	`SELECT ABS(-1), ABS(10)`
	`-- Output: 1     10`

FLOOR(x)
- returns the largest integer value not greater than x
 	
CEIL(x)
- returns the smallest integer value not less than x
	`SELECT FLOOR(1.2), FLOOR(-1.2),`
	`       CEIL(1.2) , CEIL(-1.2)`
	
	`-- Output: 1     -2          2         -1`

ROUND(x) / ROUND(x,D)
- returns the argument x rounded to D(default 0) decimal places
 	
TRUNCATE(x,D)
- returns the number x, truncated to D decimal places
	`SELECT ROUND(1.34,1), ROUND(1.35,1),`
	`       TRUNCATE(1.34,1), TRUNCATE(1.35,1)`
	`-- Output: 1.3     1.4    1.3      1.3`

Other functions POW(x,y), EXP(x), LOG(B,x), SQRT(x), RAND(), CONV(x, from_base, to_base)	
Other functions PI(), DEGREES(x), RADIANS(x), SIN(x), COS(x), TAN(x), COT(x), ASIN(x), ACOS(x), ATAN(x)	

String Functions ('a string', "another string")

LENGTH(str)
- returns the length of the string str.
	`SELECT LENGTH('abcd'), LENGTH(''), LENGTH(NULL);`
	`-- Output: 4               0           NULL`

LOWER(str)
- returns the string str with all characters changed to lowercase.
 	
UPPER(str)
- returns the string str with all characters changed to uppercase.
 	
REVERSE(str)
- returns the string str with the order of the characters reversed.
	`SELECT LOWER('AbCd'), UPPER('AbCd'), REVERSE('AbCd')`
	`-- Output: abcd           ABCD           dCdA`

CONCAT(str1, str2, str3, … …)
- returns the string that results from concatenating the arguments.
	`SELECT CONCAT('MySQL',' ','is',' ','fun')`
	`-- Output: MySQL is fun`

SUBSTR(str, pos)
- returns a substring from string str starting at position pos
 	
SUBSTR(str, pos, len)
- returns a substring that is len characters long from str, starting at position pos.
 	
LEFT(str, len)
- returns the leftmost len characters from the string str.
 	
RIGHT(str, len)
- returns the rightmost len characters from the string str
	`SELECT SUBSTR('abcdef',3), SUBSTR('abcdef',-3)`
	`-- Output: cdef                def`
	`-- string indexing starts with 1`
	
	`SELECT SUBSTR('abcdef',3,2), SUBSTR('abcdef',-3,2)`
	`-- Output: cd                  de`
	
	`SELECT LEFT('abcd', 3), RIGHT('abcd',3)`
	`-- Output:    abc             bcd`

LPAD(str, len, padstr)
- returns the string str, left-padded with the string padstr to a length of len characters.
 	
RPAD(str, len, padstr)
- returns the string str, right-padded with the string padstr to a length of len characters.
	`SELECT LPAD('abcd', 8, 'xyz'), RPAD('abcd',6,'x')`
	`-- Output:   xyzxabcd                abcdxx`

TRIM(str)
TRIM(remstr FROM str)
TRIM(LEADING remstr FROM str)
	`SELECT TRIM('   abc   '),`
	`       TRIM('x' FROM 'xxxabcxxx'),`
	`       TRIM(LEADING 'x' FROM 'xxxabcxxx'),`
	`       TRIM(TRAILING 'x' FROM 'xxxabcxxx')`

TRIM(TRAILING remstr FROM str) - returns the string str with all remstr(default space) prefixes or suffixes or both(default) removed.	`-- Output: abc        abc         abcxxx      xxxabc`
INSERT(str, pos, len, newstr)	- replaces the substring(pos to pos+len-1) with newstr
LOCATE(substr, str [, pos])	- returns the position of the first occurrence of substring substr within str
REPLACE(str, from_str, to_str)	- replaces all occurrences of from_str with to_str

Date and Time Functions ('YYYY-MM-DD hh:mm:ss', 'YYYY-MM-DD', 'hh:mm:ss')

NOW()
- returns the current datetime
CURDATE()
- returns the current date
CURTIME()
- returns the current time
	`SELECT NOW(), CURDATE(), CURTIME()`
	`-- Output: 2019-10-18 12:29:34  2019-10-18  12:29:34`

DATE(datetime)
- only date part
TIME(datetime)
- only time part
	`SELECT DATE('2019-10-18 12:29:34'),`
	`       TIME('2019-10-18 12:29:34')`
	`-- Output: 2019-10-18      12:29:34`

HOUR(datetime)
- only hour part
MINUTE(datetime)
- only minute part
SECOND(datetime)
- only second part
	`SELECT HOUR('2019-10-18 12:29:34'),`
	`       MINUTE('2019-10-18 12:29:34'),`
	`       SECOND('2019-10-18 12:29:34')`
	`-- Output: 12          29            34`

DAY(datetime)
- only day part
MONTH(datetime)
- only month part
YEAR(datetime)
- only year part
	`SELECT DAY('2019-10-18 12:29:34'),`
	`       MONTH('2019-10-18 12:29:34'),`
	`       YEAR('2019-10-18 12:29:34')`
	`-- Output: 18          10          2019`

DATEDIFF(datetime1, datetime2)
 	
TIMEDIFF(datetime1, datetime2)
	`SELECT DATEDIFF('2019-10-19 00:00:00',`
	`                '2019-10-18 23:59:59'),`
	`       TIMEDIFF('2019-10-21 00:00:00',`
	`                '2019-10-18 23:59:59')`
	`-- Output: 1           48:00:01`

DATE_ADD(datetime, INTERVAL n unit)
 	
DATE_SUB(datetime, INTERVAL n unit)
 	
unit = SECOND /MINUTE /HOUR /
DAY /MONTH /YEAR
	`SELECT DATE_ADD('2008-12-31 23:59:59',INTERVAL 1 SECOND)`
	`-- Output: 2009-01-01 00:00:00`

DATE_FORMAT(date, format)
- date to string
 	
STR_TO_DATE(string, format)
- string to date
 	
format =
%Y – YYYY, %y – yy
%M – January, %b – Jan, %m – 01..12,   %c – 1..12
%D – 0th, 1st ;   %d – 00,   %e – 0
	`SELECT DATE_FORMAT('1900-10-04 22:23:00', '%D %M, %Y %l:%i %p')`
	`-- Output:   4th October, 1900 10:23 PM`
	
	`SELECT STR_TO_DATE('May 01, 2013','%M %d,%Y')`
	`-- Output: 2013-05-01`

%H – 00..23, %k – 0..23, %h – 01 .. 12, %ℓ – 1..12 %i – 00..59, %s – 00..59 %p – 'AM', 'PM', %a – 'Sun' , %W – 'Sunday'	

LAST_DAY(date) – returns the last date of that month	`SELECT LAST_DAY('2019-12-01')`
	`-- output: 2019-12-31`

SQL Statements

- Data Manipulation Statements (DML)

- Basic Search Operations (SELECT, WHERE, ORDER BY, LIMIT clauses)

```
1. To show the whole database table data (all columns, all rows)

        SELECT *
        FROM tablename;

2. Row filter (showing specific rows)

        SELECT *
        FROM tablename
        WHERE condition;

3. Column filter (showing specific columns)

        SELECT col1, col2*5, col3+col4, function(col5), ... ... ...
        FROM tablename
        [WHERE condition];

4. Sorting table rows/data (ordering data records)

        SELECT *|col1, col2*5, col3+col4, function(col5), ... ... ...
        FROM tablename
        [WHERE condition]
        ORDER BY col1 [ASC|DESC], col2 [ASC|DESC], ... ...;

5. Showing distinct data/removing duplicate data

        SELECT [DISTINCT] col1, col2*5, col3+col4, function(col5), ... ... ...
        FROM tablename
        [WHERE condition]
        [ORDER BY col1 [ASC|DESC], col2 [ASC|DESC], ... ...];

6. Column aliasing (can be used in GROUP BY, ORDER BY, HAVING clauses)

        SELECT [DISTINCT] col1, col2*5 AS 'newcol2', col3+col4 AS 'newcol3',
                                             function(col5) AS 'newcol4', ... ... ...
        FROM tablename
        [WHERE condition]
        [ORDER BY col1 [ASC|DESC], col2 [ASC|DESC], ... ... ];

7. Limiting no. of rows

         SELECT [DISTINCT] col1, col2*5 [AS 'newcol2'], col3+col4 [AS 'newcol3'],
                                              function(col5) [AS 'newcol4'], ... ...
         FROM tablename
         [WHERE condition]
         [ORDER BY col1 [ASC|DESC], col2 [ASC|DESC], ... ... ]
         LIMIT [offset,] rowcount;

  - Default LIMIT 0, total_row_count
```

SQL Statements

- Data Manipulation Statements (DML)

- Aggregate Operations (GROUP BY, HAVING clauses)

Aggregate/Group Functions

1. **AVG([DISTINCT] expr)**
   - Returns the average value of expr for each group.
   - The DISTINCT option can be used to return the average of the distinct values of expr.
   - If there are no matching rows, AVG() returns NULL.

2. **SUM([DISTINCT] expr)**
   - Returns the sum of expr for each group.
   - If the return set has no rows, SUM() returns NULL.
   - The DISTINCT keyword can be used to sum only the distinct values of expr.

3. **COUNT(expr)**
   - Returns a count of the number of non-NULL values of expr within each group.
   - The result is a BIGINT value.
   - If there are no matching rows, COUNT() returns 0.

4. **COUNT(*)**
   - It is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain NULL values.

5. **COUNT(DISTINCT expr)**
   - Returns a count of the number of rows with different non-NULL expr values.

6. **MAX(expr)**
   - Returns the maximum value of expr.
   - If there are no matching rows, MAX() returns NULL.

7. **MIN(expr)**
   - Returns the minimum value of expr.
   - If there are no matching rows, MIN() returns NULL.

```
1. To group the whole table as 1 group

        SELECT groupfn(col1) [AS 'newcolname'], groupfn1(col2) [AS 'newcolname1'], ... ...
        FROM tablename
        [WHERE condition]
        ...
        ...

2. To group the whole table into several groups

        SELECT col1, col2, groupfn(col3), groupfn1(col4), ... ...
        FROM tablename
        [WHERE condition]
        GROUP BY col1, col2;

        Note: You can only show the columns stated in group by statement i.e. col1 and col2
        directly. All the other columns must be within group functions.
```

```
          or,

          SELECT expression, groupfn(col2), groupfn1(col3), ... ...
          FROM tablename
          [WHERE condition]
          GROUP BY expression;

          Note: You can also use expression as group by criteria.
```

3. Group filtering (to show specific groups)

```
          SELECT col1, col2, groupfn(col3), groupfn1(col4), ... ...
          FROM tablename
          [WHERE condition]
          GROUP BY col1, col2
          HAVING condition
          [ORDER BY col1 [ASC|DESC], col2 [ASC|DESC], ... ...]
          [LIMIT [offset,] rowcount];

          Note: Having condition may involve only col1 or, col2 and any other conditions that
          use aggregate functions.
```

SQL Statements

- Data Manipulation Statements (DML)

- Table Join Operations (JOIN, LEFT JOIN clauses)

Types of Join

1. **JOIN/ INNER JOIN / CROSS JOIN**
2. **LEFT JOIN / LEFT OUTER JOIN**
3. **RIGHT JOIN / RIGHT OUTER JOIN**
4. **NATURAL JOIN/NATURAL INNER JOIN/NATURAL LEFT JOIN/NATURAL RIGHT JOIN**

Notes:

a) For code portability across databases, it is recommended that you use LEFT JOIN instead of RIGHT JOIN.

b) Natural JOIN/ Natural LEFT JOIN is semantically equivalent to an INNER JOIN or a LEFT JOIN with a USING clause that names all columns that exist in both tables.

c) The search_condition used with ON is any conditional expression of the form that can be used in a WHERE clause.

d) In MySQL, JOIN, CROSS JOIN, and INNER JOIN are syntactic equivalents (they can replace each other).

e) INNER JOIN and COMMA(,) are semantically equivalent in the absence of a join condition.

f) STRAIGHT_JOIN is similar to JOIN, except that the left table is always read before the right table.

1. Table aliasing/renaming

```
       SELECT *|col1, col2*5, col3+col4, function(col5), ... ...
       FROM tablename [AS 'new table name']
       ...
       ...
       ...
```

2. INNER JOIN Operation **(max^{m} 61 tables)**

```
        - joining two tables
        SELECT t1.col1, t2.col2, ... ...
        FROM tablename1 AS t1

             JOIN
             tablename2 AS t2
             ON join_condition


        [WHERE condition]
        ...
        ...

        - joining three tables
        SELECT t1.*, t2.*, t3.col1, t3.col2, ... ...
        FROM tablename1 AS t1

             JOIN
             tablename2 AS t2
             ON join_condition

             JOIN
             tablename3 AS t3
             ON join_condition

        [WHERE condition]
        ...
        ...
```

3. LEFT OUTER JOIN Operation

```
        - joining two tables
        SELECT t1.col1, t2.col2, ... ...
        FROM tablename1 AS t1

             LEFT JOIN
             tablename2 AS t2
             ON join_condition


        [WHERE condition]
        ...
        ...
```

SQL Statements

- Data Manipulation Statements (DML)

- Subquery Operations (Scalar Subquery, Column Subquery, Row Subquery, Correlated Subquery, Derived table)

Subquery: A subquery is a SELECT statement within another statement.

Example:

```
DELETE FROM t1
WHERE s11 > ANY
               (SELECT COUNT(*)
                FROM t2
                WHERE NOT EXISTS
                       (SELECT *
                        FROM t3
                        WHERE ROW(5*t2.s1,77)=
                                                   (SELECT 50,11*s1
                                             FROM t4
                                             UNION
                                             SELECT 50,77
                                             FROM (
                                                           SELECT *
                                                    FROM t5
                                                          ) AS t5
                                                   )
                      )
               );
```

a) A subquery can return a scalar (a single value), a single row (multi-column), a single column (multi-row), or multi-row multi-column (derived table).

b) A subquery can contain many of the keywords that an ordinary SELECT can contain: DISTINCT, GROUP BY, ORDER BY, LIMIT, joins, UNION constructs, comments, functions, and so on.

c) A subquery's outer statement can be any one of: SELECT, INSERT, UPDATE, DELETE, SET, or DO.

d) A subquery must always appear within parentheses.

Reference:

- https://dev.mysql.com/doc/refman/8.0/en/subqueries.html

1. Scalar Subquery:

- A subquery is a scalar subquery that returns a single value.
- A scalar subquery is a simple operand, and you can use it almost anywhere **a single column value or literal** is legal.
- If the subquery result is empty, the result is **NULL.**

Example 1: Show the employee id, salary for only those employees having greater salary than the employee id 150. Also show the salary of employee id 150 with each employee record.

```
SELECT  EMPLOYEE_ID,
        SALARY,
        ( SELECT SALARY
          FROM employees
          WHERE EMPLOYEE_ID=150
        ) AS "150 id's salary"
FROM employees
WHERE SALARY > ( SELECT SALARY
                 FROM employees
                 WHERE EMPLOYEE_ID=150
               )
```

Example 2: Show all the employee details for only those employees receiving salary higher than the average salary of all the employees. Also show the average salary of all employees with each employee record.

```
SELECT  e1.*,
        ( SELECT AVG(e2.SALARY)
          FROM employees AS e2
        ) AS whole_avg
FROM employees AS e1
WHERE e1.SALARY > ( SELECT AVG(e3.SALARY)
                    FROM employees AS e3
                  )
```

Example 3: Show those employee details, average salary of his own department who receives higher salary then the average salary of his department.

```
SELECT  e1.*,
        ( SELECT AVG(e2.SALARY)
          FROM employees as e2
          WHERE e2.DEPARTMENT_ID = e1.DEPARTMENT_ID
        ) AS Dept_Avg
FROM employees as e1
WHERE e1.SALARY > ( SELECT AVG(e3.SALARY)
                    FROM employees as e3
                    WHERE e3.DEPARTMENT_ID = e1.DEPARTMENT_ID
                  )
ORDER BY Dept_Avg ASC
```

References:

- https://dev.mysql.com/doc/refman/8.0/en/scalar-subqueries.html
- https://dev.mysql.com/doc/refman/8.0/en/comparisons-using-subqueries.html

Column Subquery: When the subquery returns a Single Column but multiple rows.

Use operators:

- **ANY** – return TRUE if the comparison is TRUE for ANY of the values in the column that the subquery returns.
- **ALL** – return TRUE if the comparison is TRUE for ALL of the values in the column that the subquery returns.
- IN – it is equivalent to (**= ANY**) operator.
- SOME – it is equivalent to **ANY** operator.

Example 4(ALL operator): Show those employee details for only those employees receiving salary higher than all other employee salaries of department number 50. Also show the highest salary of department number 50 with each employee record.

```
SELECT  e1.*,
        ( SELECT MAX(e3.SALARY)
          FROM employees AS e3
          WHERE e3.DEPARTMENT_ID=50
        ) AS 'max sal of dept no 50'
FROM employees AS e1
WHERE e1.SALARY > ALL( SELECT e2.SALARY
                       FROM employees AS e2
                       WHERE e2.DEPARTMENT_ID=50
                     )
```

Example 5(ANY operator): Show those employee details for only those employees receiving salary higher than any of the employee salaries of department no 50. Also show the lowest salary of department no 50 with each employee record.

```
SELECT  e1.*,
        ( SELECT MIN(e3.SALARY)
          FROM employees AS e3
          WHERE e3.DEPARTMENT_ID=50
        ) AS 'min sal of dept no 50'
FROM employees AS e1
WHERE e1.SALARY > ANY( SELECT e2.SALARY
                       FROM employees AS e2
                       WHERE e2.DEPARTMENT_ID=50
                     )
```

References:

- https://dev.mysql.com/doc/refman/8.0/en/any-in-some-subqueries.html
- https://dev.mysql.com/doc/refman/8.0/en/all-subqueries.html

Practices:

1. Find out those employee's last name and salary who is assigned to the same job type as employee id 141.
2. Find out the employee (last name and salary) who receives the highest salary.
3. Find out the manager details who handles minimum no of employees.
4. Find out those employees who don't work in 'IT_PROG' job type and also receive lower salary than any other employees in 'IT_PROG' job type.

2. **Row Subquery:** A row subquery is a subquery variant that returns a single row and can thus return more than one column value.

Example 1: Show those employees employee id, first name, job id, department id, job id of employee number 150, department id of employee number 150 who works in the same department and same job type as employee having employee number 150.

```
SELECT  e1.EMPLOYEE_ID, e1.FIRST_NAME, e1.JOB_ID, e1.DEPARTMENT_ID,
        ( SELECT JOB_ID
          FROM employees
          WHERE EMPLOYEE_ID=150
        ) AS "150 id's job_id",
        ( SELECT DEPARTMENT_ID
          FROM employees
          WHERE EMPLOYEE_ID=150
        ) AS "150 id's department_id"
FROM employees AS e1
WHERE (JOB_ID, DEPARTMENT_ID) = ( SELECT JOB_ID, DEPARTMENT_ID
                                  FROM employees AS e2
                                  WHERE EMPLOYEE_ID=150
                                )
```

Practice:

1. Find out those employees whose is assigned to the same job type as employee id 144 and receives the same salary as employee id 144.

Example 2(multiple row, multiple column): Find out those employees employee id, department id, his salary and the maximum salary of his department who receives the highest salary of his own department.

```
SELECT  e1.EMPLOYEE_ID,
        e1.DEPARTMENT_ID,
        e1.SALARY,
        (SELECT MAX(e2.SALARY)
         FROM employees AS e2
         WHERE e2.DEPARTMENT_ID=e1.DEPARTMENT_ID
        ) AS 'max sal of his dept'
FROM employees AS e1
WHERE (DEPARTMENT_ID, SALARY) IN ( SELECT DEPARTMENT_ID, MAX(SALARY)
                                   FROM employees
                                   GROUP BY DEPARTMENT_ID
                                 )
```

Reference:

- https://dev.mysql.com/doc/refman/8.0/en/row-subqueries.html

3. **Derived Table:** A derived table is an expression that generates a table within the scope of a query FROM clause. A subquery in a SELECT statement FROM clause is a derived table.

   - **[AS] tablename** clause is mandatory because every table in a FROM clause must have a name.
   - A derived table cannot normally refer to (depend on) columns of preceding tables in the same FROM clause.
   - A derived table may be defined as a lateral derived table to specify that such references are permitted.

Example 1: Show the maximum value of the department wise total salary.

```
SELECT MAX(dt1.c1) AS 'max total salary of a dept'
FROM (SELECT SUM(e2.SALARY) AS c1
      FROM employees AS e2
      GROUP BY e2.DEPARTMENT_ID
     ) AS dt1
```

Example 2: For each employee show his employee id, total no of employees hired after him and total no of employees hired before him.

```
SELECT h_after.af_id, h_after.AFTER, h_before.BEFORE
FROM (  SELECT  e1.employee_id AS af_id,
                count(*) AS "AFTER"
        FROM employees e1
             JOIN employees e2
             ON e1.HIRE_DATE < e2.HIRE_DATE
        GROUP BY e1.EMPLOYEE_ID
     ) AS h_after

    JOIN

    (   SELECT  e3.employee_id AS bf_id,
                count(*) AS "BEFORE"
        FROM employees e3
             JOIN employees e4
             ON e3.HIRE_DATE > e4.HIRE_DATE
        GROUP BY e3.EMPLOYEE_ID
     ) AS h_before

    ON  h_after.af_id = h_before.bf_id
```

Reference:

- https://dev.mysql.com/doc/refman/8.0/en/derived-tables.html

Practices:

1. Show the maximum no of employees handled by a manager.
2. Show the minimum amount of total salary provided by a department.

4. **Correlated Subquery:** A correlated subquery is a subquery that contains a reference to a table that also appears in the outer query.

Example 1:

```
SELECT *
FROM t1
WHERE column1 = ANY (SELECT column1
                     FROM t2
                     WHERE t2.column2 = t1.column2
                    );
```

Example 2: Show those employee details receiving highest salary in his job type.

```
SELECT e1.EMPLOYEE_ID, e1.JOB_ID, e1.SALARY
FROM employees AS e1
WHERE SALARY=( SELECT MAX(SALARY)
               FROM employees AS e2
               WHERE e2.JOB_ID=e1.JOB_ID
              )
```

Reference:

- https://dev.mysql.com/doc/refman/8.0/en/correlated-subqueries.html

Reference: https://dev.mysql.com/doc/refman/8.0/en/