Week 2 - MYSQL NOTES

Created by	Dakshil Gorasiya

Sub Query

Sub query is a SELECT statement inside SQL statement

It is useful to solve complex query in steps

Correlated subquery can be slower use it carefully

Subquery is always enclosed in parentheses ()

It is used within WHERE, FROM, SELECT, HAVING, INSERT, UPDATE, DELETE

There is two type of subquery

1. Scaler subquery (Return a single value)

   Ex:

   ```
   -- To find list of student having marks more than average
   SELECT
       T01F01,
       T01F02,
       T03F04
   FROM
       T01
       INNER JOIN T03 ON T01F01 = T03F02
   WHERE
       T03F04 > (SELECT AVG(T03F04) FROM T03)
   ```

If subquery return more than one rows it will generate error

If it find (col1, col2) > (val1, val2) it first check col1 > val1 it true it is true and col1 = val1 then compare col2 > val2 (so basically if first comparison it true or false it will not check condition and if equal than only go to next condition)

2. Multi-row subquery (Used with IN, NOT IN, ANY, ALL)

IN - It will try to find current value in result given by subquery

```
-- To find employee working in sales and marketing
SELECT
    T02F01,
    T02F02,
    T02F03
FROM
    T02
WHERE
    T02F08 IN (SELECT T01F01 FROM T01 WHERE T01F02 IN ('SALES', 'MARKETING'));
```

ANY - It compares all values given by sub query and if any one is true it will give true

= ANY is same as IN

> ANY means greater than minimum value

< ANY means less than maximum value

```
-- To find employee whose salary is greater than any of sales employee
SELECT
    T02F01,
    T02F02,
    T02F03
FROM
    T02
WHERE
    T02F07 > ANY(SELECT T02F07 FROM T02 WHERE T02F08 = 2);
```

ALL - column value will be compared with each row of sub query and all should result as true

> ALL means greater than maximum value

< ALL means less than minimum value

```
SELECT
    T02F01,
    T02F02,
    T02F03
FROM
    T02
WHERE
    T02F07 > ALL(SELECT T02F07 FROM T02 WHERE T02F08 = 2);
```

Correlated sub query

EXISTS - It will return true if subquery result have 1 or more rows and false otherwise

Often used in correlated sub query

```
-- To find department having atleas one employee
SELECT
    T01F02
FROM
    T01
WHERE
    EXISTS (SELECT 1 FROM T02 WHERE T02F08 = T01F01);
```

For each row of outer query it run inner query and check EXISTS condition

For other subquery like IN, ANY, ALL inner query runs only once

Correlated sub query has performance overhead as it have to to run inner query multiple time

It is not mandatory that all correlated subquery have EXISTS the inner query which use column of outer table is known as correlated sub query

Ex:

```
-- To list employee whose salary more than department average
SELECT
    T02F01,
    T02F02,
    T02F03,
    T02F07
FROM
    T02 AS D1
WHERE
    T02F07 > (SELECT AVG(T02F07) FROM T02 AS D2 WHERE D1.T02F08 = D2.T02F08);
```

Subquery in from

Alias name is mandatory for dept_avg

```
-- To find department with maximum average salary
SELECT
    MAX(avg_dept_salary)
FROM
    (SELECT
        T02F08,
        AVG(T02F07) AS avg_dept_salary
    FROM
        T02
    GROUP BY
        T02F08)
    AS dept_avg;
```

Subquery in select

```
-- To list with their department
SELECT
    T02F01,
    T02F02,
    T02F03,
    (SELECT T01F02 FROM T01 WHERE T01F01 = T02F08)
FROM
    T02;
```

UNION VS UNION ALL

Union combines result of two or more SELECT statements and removes duplicate rows

Because it has to check for and eliminate duplicates, it performs an implicit DISTINCT operation on the combined result set. This process involves sorting the data and comparing rows, which can add performance overhead, especially on large datasets.

Syntax:

```
SELECT column1, column2, ... FROM table1
UNION
SELECT column1, column2, ... FROM table2;
```

Union all also combines results of two SELECT statements but it does not remove duplicate rows

Rules

1. Both select must have same number of columns
2. Both select should have compatible data types otherwise it try to convert internally
3. Column name of first select is taken
4. Use ORDER BY and LIMIT at last after last select statement mind that use name from first select statement (If need order by or limit inside select use it in parenthesis)

Ex:

```
-- UNION
SELECT * FROM T02 WHERE T02F08 = 1
UNION
SELECT * FROM T02 WHERE T02F08 = 1;

-- UNION ALL
SELECT * FROM T02 WHERE T02F08 = 1
UNION ALL
SELECT * FROM T02 WHERE T02F08 = 1;
```

error : as have different number of columns

```
SELECT * FROM T01
UNION
SELECT * FROM T02;
```

Other operator are INTERSECT and EXCEPT(or MINUS)

Views

View is virtual table which is a stored query with no real data

A view contains rows and columns, just like a regular table. The key difference is that a view does not store any data itself. The data is physically stored in the underlying base tables from which the view is created.

syntax:

```
CREATE VIEW view_name AS select_query [WITH CHECK OPTION];
```

Updatable vs Non-updatable view
A view is generally **updatable** if MySQL can trace the rows in the view directly back to the rows in a single base table. The rules are:

- The SELECT statement must not contain DISTINCT.
- It must not contain aggregate functions (SUM(), COUNT(), MIN(), MAX(), etc.).
- It must not contain GROUP BY or HAVING clauses.

- It must not contain UNION or UNION ALL.
- It must not have subqueries in the SELECT list.
- The FROM clause must reference a base table (not another non-updatable view).

Since views are stored queries, they run every time you query them. If a view is based on a very complex and slow query, accessing the view will also be slow. Views don't store data, so they don't improve the performance of the underlying query.

Ex:

```
CREATE OR REPLACE VIEW EMP_DEPT AS
(SELECT
   T02F01,
   T02F02,
   T02F03,
   T01F02
FROM
   T01
   INNER JOIN T02 ON T01F01 = T02F08);

SELECT * FROM EMP_DEPT;
```

DELETE FROM EMP_DEPT; (ERROR: cannot delete from join view)

It will delete all data from original table

```
CREATE OR REPLACE VIEW EMP_NAME AS
(SELECT
   T02F01,
   T02F02,
   T02F03
FROM
   T02);

   DELETE FROM T02;
```

ALTER VIEW view_name AS query will modify existing view without modifying permissions

The `WITH CHECK OPTION` is a clause used when creating a view that acts as a data integrity rule. It ensures that any data inserted or updated through the view must conform to the conditions in the view's `WHERE` clause.

In simple terms, it prevents us from using a view to make changes to data that the view itself wouldn't be able to see.

Without this option, we can create a logical loophole. we could update a row through a view in such a way that the row "disappears" from the view immediately after the update. This can be confusing and lead to inconsistent data.

We can give access to view using GRANT to other user without giving access of entire table and if given update permission user can only modify columns which are present in view

FUNCTION

Function is a named block of SQL code that performs a specific operation and return a single value

There is two type of functions - build-in and user defined

Build-in function are CONCAT(), UPPER(), LOWER(), LENGTH(), ROUND(), CEIL(), FLOOR(), ABS(), NOW(), CURRDATE(), COUNT(), SUM(), MIN(), MAX(), AVG()

User defined function

It must return exactly one value

It cannot use OUT or INOUT parameter

Syntax:

```
DELIMITER $$

CREATE FUNCTION function_name(p_parameter1 DATATYPE, p_parameter2 DATATYPE, ...)
RETURNS return_datatype
[CHARACTERISTICS]
BEGIN
    -- Declare variables if needed
    DECLARE v_variable_name DATATYPE;

    -- function logic (calculations, etc.)
    -- ...

    -- Return the final value
    RETURN value_to_return;
END$$

DELIMITER ;
```

Characteristics : DETERMINISTIC, NOT DETERMINISTIC, NO SQL, READS SQL DATA, WRITES SQL DATA

Characteristics is used for optimization by MYSQL

Ex:

```
DROP FUNCTION IF EXISTS CALCULATE_AGE;
DELIMITER $$
CREATE FUNCTION CALCULATE_AGE(p_birth_date DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE v_age INT;
    SET v_age = TIMESTAMPDIFF(YEAR, p_birth_date, CURDATE());
    RETURN v_age;
END$$
DELIMITER ;

SELECT CALCULATE_AGE("2010-10-10");
```

To get all functions of DB

```
SHOW FUNCTION STATUS WHERE Db = 'employee';
```

To get function defination

```
SHOW CREATE FUNCTION CALCULATE_AGE;
```

PROCEDURE

A procedure is similar to FUNCTION but it cannot return values

It can use OUT and INOUT parameter to return multiple values

It cannot be called by SELECT it is call by CALL procedure_name

It can reduce network traffic as we are not sending all SQL statements

It can optimize performance as it is precompiled means parsed and execution plan is stored

It can use transaction unlike functions

Syntax:

```
DELIMITER $$

CREATE PROCEDURE procedure_name(
    [parameter_mode] parameter1_name DATATYPE,
    [parameter_mode] parameter2_name DATATYPE,
    ...
)
BEGIN
    -- Declare variables if needed
    -- DECLARE variable_name DATATYPE;

    -- SQL statements and logic go here
    -- ...
END$$

DELIMITER ;
```

To call it

```
CALL procedure_name
```

Ex:

```
DROP PROCEDURE IF EXISTS GET_EMPLOYEE_BY_DEPARTMENT;
DELIMITER $$
CREATE PROCEDURE GET_EMPLOYEE_BY_DEPARTMENT(IN p_dept_name VARCHAR(50))
BEGIN
    SELECT
        T02F01,
        T02F02,
        T02F03
    FROM
        T02
        INNER JOIN T01 ON T02F08 = T01F01
    WHERE
        T01F02 = p_dept_name;
END$$
```

```
DELIMITER ;

CALL GET_EMPLOYEE_BY_DEPARTMENT("Sales");
```

With out parameter

```
DROP PROCEDURE IF EXISTS COUNT_EMPLOYEE;
DELIMITER $$
CREATE PROCEDURE COUNT_EMPLOYEE(IN p_dept_name VARCHAR(50), OUT p_count INT)
BEGIN
    SELECT
        COUNT(T02F08)
    INTO
        p_count
    FROM
        T02
        INNER JOIN T01 ON T02F08 = T01F01
    WHERE
        T01F02 = p_dept_name;
END$$
DELIMITER ;

CALL COUNT_EMPLOYEE("Sales", @total);
SELECT @total;
```

To use INOUT first

SET @variable = value

and then pass

Ex:

```
DELIMITER $$
CREATE PROCEDURE double_number(INOUT p_num INT)
BEGIN
    SET p_num = p_num * 2;
END$$
DELIMITER ;

SET @num = 5;
CALL DOUBLE_NUMBER(@num);
SELECT @num;
```

To use transaction control

```
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
    END;

    START TRANSACTION;

        -- sql statements
    COMMIT;
END$$

DELIMITER ;
```

If else

```
IF condition1 THEN
    -- statements
ELSEIF condition2 THEN
    -- statements
ELSE
    -- statements
END IF;
```

Loops

```
WHILE condition DO
    -- statements
END WHILE;
```

```
REPEAT
    -- statements
UNTIL condition
END REPEAT;
```

```
[label:] LOOP
    -- some statements
    IF condition THEN
        LEAVE label;  -- break
        ITERATE label; -- continue
    END IF;
END LOOP label;
```

To show all procedure

```
SHOW PROCEDURE STATUS WHERE Db = 'database_name';
```

To show specific procedure

```
SHOW CREATE PROCEDURE procedure_name;
```

Syntax to declare a variable

```
DECLARE variable_name DATATYPE(size) DEFAULT default_value;
```

Assign value

```
SET variable_name = value
```

Security Definer, Invoker

Definer: The procedure runs with the permission of the user who defined it.

Invoker: The procedure runs with the permission of the user who invoked it.

Ex:

```
CREATE PROCEDURE procedure_name()
SQL SECURITY INVOKER
BEGIN
    -- CODE
END;
```

SQL SECURITY DEFINER is default setting

It is useful when instead of giving access on specific table we can give access of a script

Exception Handling

```
DECLARE handler_action HANDLER FOR condition_value [, condition_value] ...
    statement;
```

handler_action: CONTINUE(Go on next line), EXIT(Exit current BEGIN...END block)

condition_value: MySQL error code, SQL state, Named condition like SQLEXCEPTION(all sql exception), SQLWARNING(all sql warning), NOT FOUND(for cursor)

Ex:

```
 DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        -- Code to run when any SQL error occurs
        SELECT 'An error occurred!';
        ROLLBACK;
    END;
```

Cursor:

When we open cursor it run select query and store result in temporary table

When we FETCH it move cursor to next row

Cursor is slower compared to SELECT use it only when row by row processing is required

```
DROP PROCEDURE IF EXISTS FETCH_EMPLOYEE;
DELIMITER $$
CREATE PROCEDURE FETCH_EMPLOYEE(OUT p_employee TEXT)
BEGIN
    DECLARE v_done INT DEFAULT FALSE;
    DECLARE v_name VARCHAR(100);

    DECLARE cur_employee CURSOR FOR SELECT CONCAT(T02F02, T02F03) FROM T02;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_done = TRUE;
    SET p_employee = '';

    OPEN cur_employee;

    READ_LOOP: LOOP
        FETCH cur_employee INTO v_name;
        IF v_done THEN
            LEAVE READ_LOOP;
        END IF;

        IF p_employee = '' THEN
            SET p_employee = v_name;
        ELSE
            SET p_employee = CONCAT(p_employee, ", ", v_name);
        END IF;

    END LOOP;

    CLOSE cur_employee;
END$$
DELIMITER ;

CALL FETCH_EMPLOYEE(@list);
SELECT @list;
```

PREPARE EXECUTE

It is used to write dynamic SQL query

When a session ends, whether normally or abnormally, its prepared statements no longer exist.

A statement prepared in stored program context cannot refer to stored procedure or function parameters or local variables because they go out of scope when the program ends and would be unavailable were the statement to be executed later outside the program

It can refer to user defined variables

Use ? only for data not for table name, column name or sql statements

Ex:

```
SET @salary = "100000";
PREPARE GET_EMPLOYEE FROM 'SELECT T02F02, T02F03, T02F07 FROM T02 WHERE T02F07 > ?';
EXECUTE GET_EMPLOYEE USING @salary;
DEALLOCATE PREPARE GET_EMPLOYEE;
```

Advantages:

Improve performance as parsed and compiled only once.

Protect against SQL injection as sql treat variable as data not as sql statement.

TRIGGER

Trigger is a special type of stored program that automatically execute in response to a specific event on a table

Cascade foreign key actions do not activate trigger

Syntax:

```
DELIMITER $$

CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
BEGIN
    -- Trigger body
END$$

DELIMITER ;
```

NEW : Available in INSERT and UPDATE point to data after operation

OLD : Available in DELETE and UPDATE point to data before operation

Ex:

```
DROP TRIGGER IF EXISTS LOG_DELETE_EMPLOYEE;
DELIMITER $$
CREATE TRIGGER LOG_DELETE_EMPLOYEE
BEFORE DELETE ON T02
FOR EACH ROW
BEGIN
    INSERT INTO T03 (T03F01, T03F02, T03F03, T03F04) VALUES
    (OLD.T02F01, OLD.T02F02, OLD.T02F03, CURDATE());
END$$
DELIMITER ;
```

To get all triggers of database

```
SHOW TRIGGERS;
```

BACKUP RESTORE

Physical backup : Feature of MySQL enterprise it copy files so fast and smaller in size perfect for large database (Use mysqlbackup), not portable must be restored to similar version

Logical backup : Store backup in SQL format store create and insert entry (Use mysqldump)

Hot backup : No database downtime all operation are allowed (MySQL enterprise backup for INNODB)

Warm backup : Database is online but only read operation is allowed (mysqldump)

Cold backup : Stop the database and then backup usually used with replica

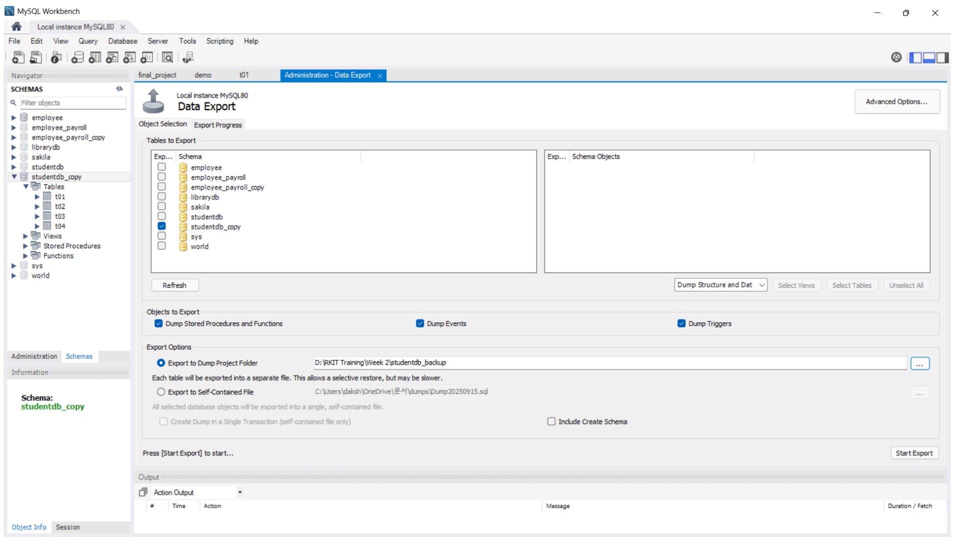
There is two options to export using MySQL workbench which use mysqldump

If Include Create Schema is ticked it will include statement to create database and use it.

1. Export to dump project folder

Allows to restore selectively

Can be slower


MySQL Workbench
Local instance MySQL80
File Edit View Query Database Server Tools Scripting Help
final_project demo t01 Administration - Data Export
Data Export
Object Selection Export Progress
Advanced Options...
Refresh
Select Views Select Tables Unselect All
Start Export

Tables to Export

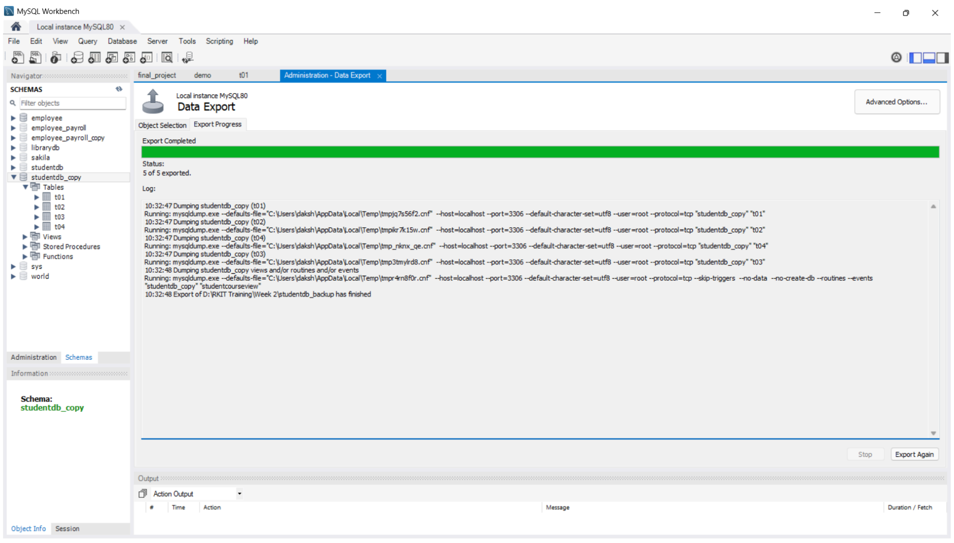
Exp...	Schema
☐	employee
☐	employee_payroll
☐	employee_payroll_copy
☐	librarydb
☐	sakila
☐	studentdb
☑	studentdb_copy
☐	sys
☐	world

Dump Structure and Dat

Objects to Export
☑ Dump Stored Procedures and Functions ☑ Dump Events ☑ Dump Triggers

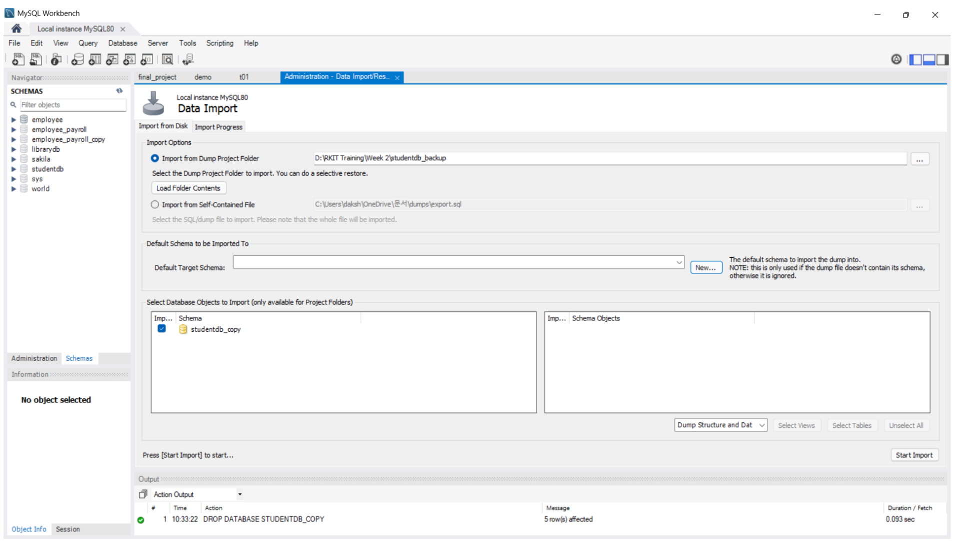
Export Options
[✓ Export to Dump Project Folder] D:\RKIT Training\Week 2\studentdb_backup
Each table will be exported into a separate file. This allows a selective restore, but may be slower.
Export to Self-Contained File C:\Users\daksh\OneDrive\문서\dumps\Dump20250915.sql
All selected database objects will be exported into a single, self-contained file.
☐ Create Dump in a Single Transaction (self-contained file only) ☐ Include Create Schema

Press [Start Export] to start...



Export Completed
Status:
5 of 5 exported.
Log:

```
10:32:47 Dumping studentdb_copy (t01)
Running: mysqldump.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmpjq7s56f2.cnf"  --host=localhost --port=3306 --default-character-set=utf8 --user=root --protocol=tcp "studentdb_copy" "t01"
10:32:47 Dumping studentdb_copy (t02)
Running: mysqldump.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmpikr7k15w.cnf"  --host=localhost --port=3306 --default-character-set=utf8 --user=root --protocol=tcp "studentdb_copy" "t02"
10:32:47 Dumping studentdb_copy (t04)
Running: mysqldump.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmp_nknx_qe.cnf"  --host=localhost --port=3306 --default-character-set=utf8 --user=root --protocol=tcp "studentdb_copy" "t04"
10:32:47 Dumping studentdb_copy (t03)
Running: mysqldump.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmp3tmylrd8.cnf"  --host=localhost --port=3306 --default-character-set=utf8 --user=root --protocol=tcp "studentdb_copy" "t03"
10:32:48 Dumping studentdb_copy views and/or routines and/or events
Running: mysqldump.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmpr4rn8f0r.cnf"  --host=localhost --port=3306 --default-character-set=utf8 --user=root --protocol=tcp --skip-triggers  --no-data  --no-create-db --routines --events
"studentdb_copy" "studentcourseview"
10:32:48 Export of D:\RKIT Training\Week 2\studentdb_backup has finished
```


MySQL Workbench
Local instance MySQL80
final_project demo t01 Administration - Data Import/Res...
Data Import
Import from Disk Import Progress
Load Folder Contents
New...
Select Views Select Tables Unselect All
Start Import

Import Options
[✓ Import from Dump Project Folder] D:\RKIT Training\Week 2\studentdb_backup
Select the Dump Project Folder to import. You can do a selective restore.
Import from Self-Contained File C:\Users\daksh\OneDrive\문서\dumps\export.sql
Select the SQL/dump file to import. Please note that the whole file will be imported.

Default Schema to be Imported To
Default Target Schema:
The default schema to import the dump into.
NOTE: this is only used if the dump file doesn't contain its schema, otherwise it is ignored.

Select Database Objects to Import (only available for Project Folders)

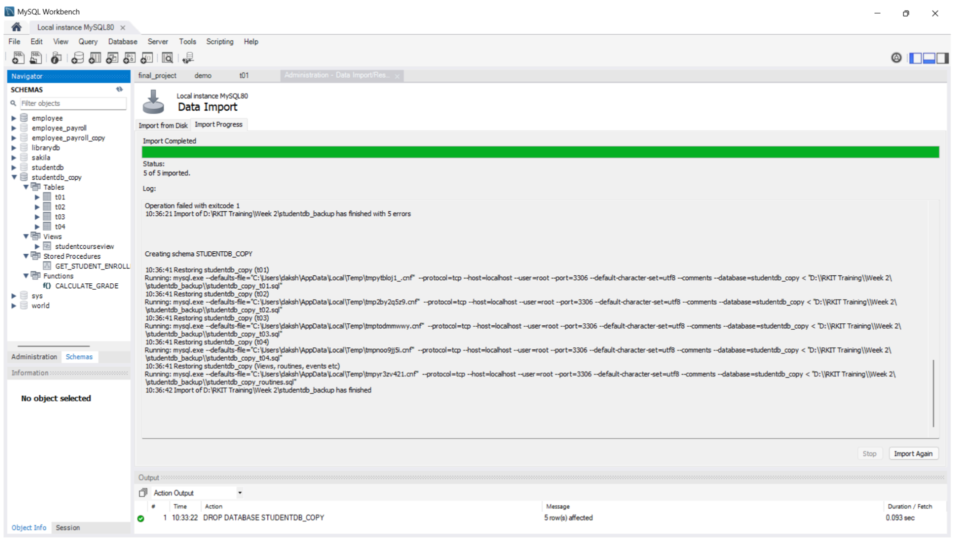
Imp...	Schema
☑	studentdb_copy

Dump Structure and Dat

Press [Start Import] to start...

Output

#	Time	Action	Message	Duration / Fetch
1	10:33:22	DROP DATABASE STUDENTDB_COPY	5 row(s) affected	0.093 sec


Local instance MySQL80
Data Import

Import Completed

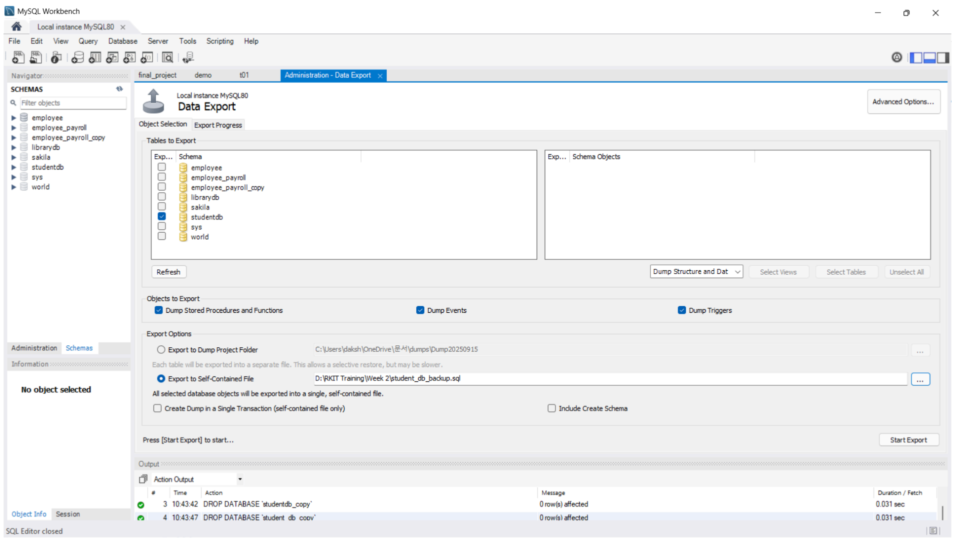
Status:
5 of 5 imported.

Log:

```
Operation failed with exitcode 1
10:36:21 Import of D:\RKIT Training\Week 2\studentdb_backup has finished with 5 errors



Creating schema STUDENTDB_COPY

10:36:41 Restoring studentdb_copy (t01)
Running: mysql.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmpytbloj1_.cnf"  --protocol=tcp --host=localhost --user=root --port=3306 --default-character-set=utf8 --comments --database=studentdb_copy < "D:\\RKIT Training\\Week 2\
\studentdb_backup\\studentdb_copy_t01.sql"
10:36:41 Restoring studentdb_copy (t02)
Running: mysql.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmp2by2q5z9.cnf"  --protocol=tcp --host=localhost --user=root --port=3306 --default-character-set=utf8 --comments --database=studentdb_copy < "D:\\RKIT Training\\Week 2\
\studentdb_backup\\studentdb_copy_t02.sql"
10:36:41 Restoring studentdb_copy (t03)
Running: mysql.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmptodmmwwy.cnf"  --protocol=tcp --host=localhost --user=root --port=3306 --default-character-set=utf8 --comments --database=studentdb_copy < "D:\\RKIT Training\\Week 2\
\studentdb_backup\\studentdb_copy_t03.sql"
10:36:41 Restoring studentdb_copy (t04)
Running: mysql.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmpnoo9jj5i.cnf"  --protocol=tcp --host=localhost --user=root --port=3306 --default-character-set=utf8 --comments --database=studentdb_copy < "D:\\RKIT Training\\Week 2\
\studentdb_backup\\studentdb_copy_t04.sql"
10:36:41 Restoring studentdb_copy (Views, routines, events etc)
Running: mysql.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmpyr3zv421.cnf"  --protocol=tcp --host=localhost --user=root --port=3306 --default-character-set=utf8 --comments --database=studentdb_copy < "D:\\RKIT Training\\Week 2\
\studentdb_backup\\studentdb_copy_routines.sql"
10:36:42 Import of D:\RKIT Training\Week 2\studentdb_backup has finished
```

```
#  Time      Action                          Message             Duration / Fetch
1  10:33:22  DROP DATABASE STUDENTDB_COPY    5 row(s) affected   0.093 sec
```

2. Export to self contained file


Local instance MySQL80
Data Export

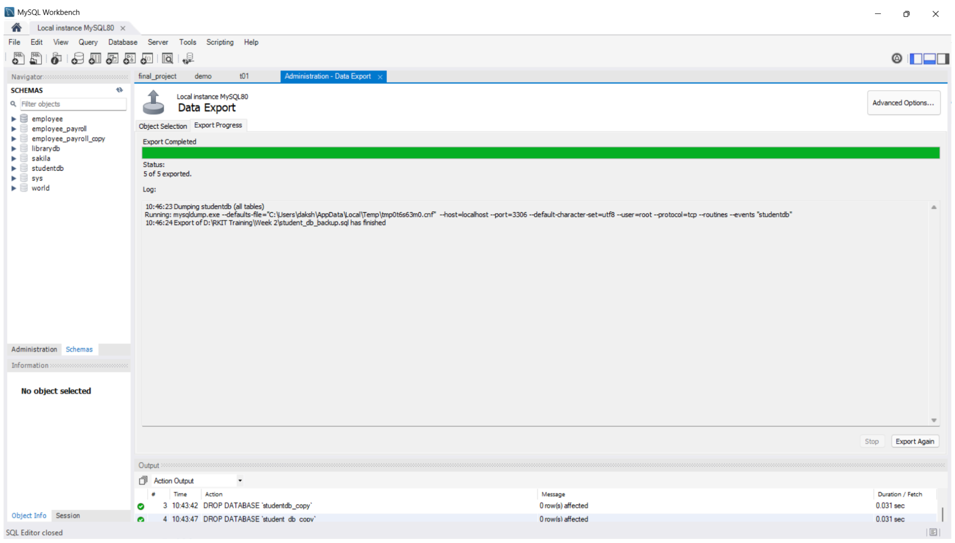
Tables to Export: employee, employee_payroll, employee_payroll_copy, librarydb, sakila, [✓ studentdb], sys, world

Objects to Export
- [✓] Dump Stored Procedures and Functions
- [✓] Dump Events
- [✓] Dump Triggers

Export Options
- ( ) Export to Dump Project Folder: C:\Users\daksh\OneDrive\문서\dumps\Dump20250915
  Each table will be exported into a separate file. This allows a selective restore, but may be slower.
- (●) Export to Self-Contained File: D:\RKIT Training\Week 2\student_db_backup.sql
  All selected database objects will be exported into a single, self-contained file.
- [ ] Create Dump in a Single Transaction (self-contained file only)
- [ ] Include Create Schema

Press [Start Export] to start...

```
#  Time      Action                             Message             Duration / Fetch
3  10:43:42  DROP DATABASE `studentdb_copy`     0 row(s) affected   0.031 sec
4  10:43:47  DROP DATABASE `student_db_copy`    0 row(s) affected   0.031 sec
```


Local instance MySQL80
Data Export

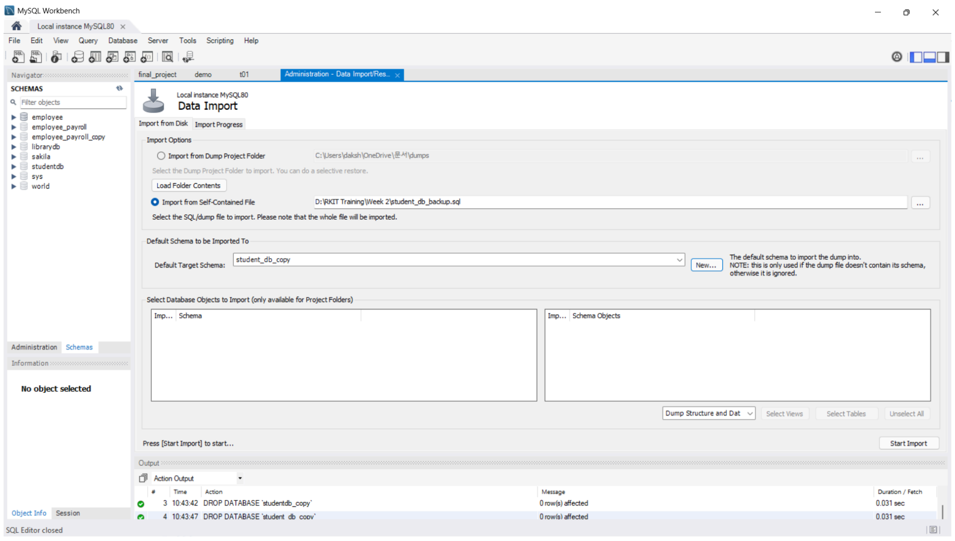
Export Completed

Status:
5 of 5 exported.

Log:

```
10:46:23 Dumping studentdb (all tables)
Running: mysqldump.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmp0t6s63m0.cnf"  --host=localhost --port=3306 --default-character-set=utf8 --user=root --protocol=tcp --routines --events "studentdb"
10:46:24 Export of D:\RKIT Training\Week 2\student_db_backup.sql has finished
```

```
#  Time      Action                             Message             Duration / Fetch
3  10:43:42  DROP DATABASE `studentdb_copy`     0 row(s) affected   0.031 sec
4  10:43:47  DROP DATABASE `student_db_copy`    0 row(s) affected   0.031 sec
```



Local instance MySQL80
Data Import

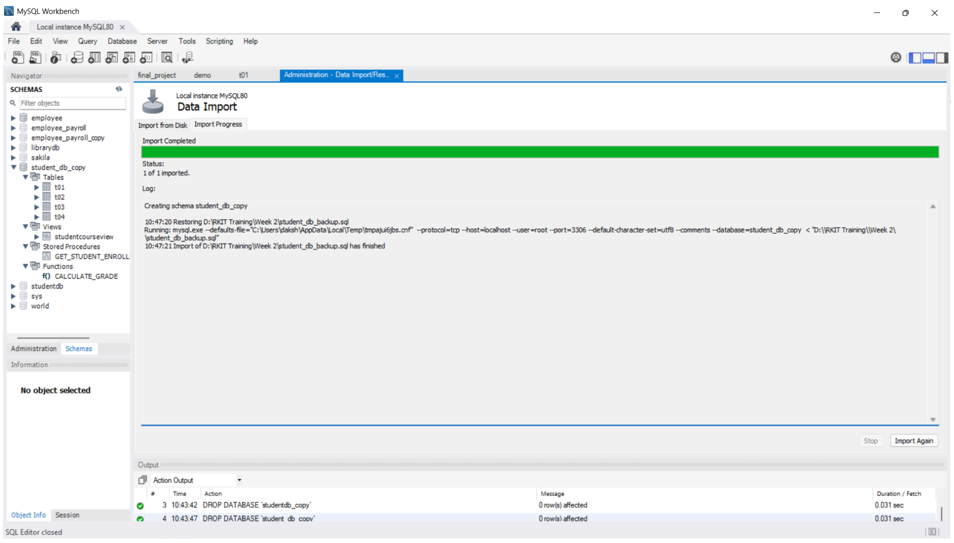
Import Options
( ) Import from Dump Project Folder: C:\Users\daksh\OneDrive\문서\dumps
Select the Dump Project Folder to import. You can do a selective restore.
(●) Import from Self-Contained File: D:\RKIT Training\Week 2\student_db_backup.sql
Select the SQL/dump file to import. Please note that the whole file will be imported.

Default Schema to be Imported To
Default Target Schema: student_db_copy
The default schema to import the dump into.
NOTE: this is only used if the dump file doesn't contain its schema, otherwise it is ignored.

Select Database Objects to Import (only available for Project Folders)
Dump Structure and Dat

Press [Start Import] to start...

```
#  Time      Action                                Message             Duration / Fetch
3  10:43:42  DROP DATABASE `studentdb_copy`        0 row(s) affected   0.031 sec
4  10:43:47  DROP DATABASE `student_db_copy`       0 row(s) affected   0.031 sec
```



Import Completed
Status:
1 of 1 imported.

Log:

```
Creating schema student_db_copy

10:47:20 Restoring D:\RKIT Training\Week 2\student_db_backup.sql
Running: mysql.exe --defaults-file="C:\Users\daksh\AppData\Local\Temp\tmpajui6jbs.cnf"  --protocol=tcp --host=localhost --user=root --port=3306 --default-character-set=utf8 --comments --database=student_db_copy  < "D:\\RKIT Training\\Week 2\\student_db_backup.sql"
10:47:21 Import of D:\RKIT Training\Week 2\student_db_backup.sql has finished
```

EXPLAIN

Explain is a diagnostic tool that shows execution plan for given SQL statement

Syntax:

```
EXPLAIN sql_query
```

Ex:

```
EXPLAIN SELECT
    T02F01,
    T02F02,
    T02F03,
    T01F02
FROM
    T01
    INNER JOIN T02 ON T01F01 = T02F08
```

```
WHERE
    T02F04 = "PRIYA.PATEL@EXAMPLE.COM";
```

id	select_type	table	partitions	type	possible_keys	key
1	SIMPLE	T02		const	T02F04_UNIQUE,FK_T02F08	T02F04_UNIQ
1	SIMPLE	T01		const	PRIMARY	PRIMARY

As it is using index it is optimized and need 1 row to lookup

Output of EXPLAIN

`id` : A sequential identifier for each SELECT within the query

`select_type` : The type of SELECT query

SIMPLE : no subquery, no union

PRIMARY : outermost select

SUBQUERY

DERIVED : Subquery in FROM clause

UNION

`table` : the table to which row of output refers

`type` : Most crucial for query optimization

system > const > eq_ref > ref > range > index > all

system/const : The table has at most one matching row, which is read at the start of the query.

eq_ref : Used in joins when all parts of a primary key or unique index are used.

ref : All matching rows are read from an index for each combination of rows from the previous tables. Used for non-unique indexes

range : Only rows that are in given range is retrieved using an index. Common for BETWEEN, IN, <, >

index : Reading entire index

all : Full table scan.

`possible_keys` : Shows which index MYSQL can use to find rows in table

`key` : The index which is used (If NULL no index is used)

`key_len` : The length of key that was used. For a multi-column index, this tells how many parts of the index MySQL is using.

`ref` : shows which column is compared with index

`rows` : Number of row MYSQL have to scan to get output (Lower is better)

`filtered` : An estimated percentage of table rows will be filtered by the table condition

`Extra` : Contains addition information

Using index : Good. All information needed for query was retrieved from the index itself, without reading actual table rows

Using where : Normal. Where clause is being used to filter records after retrieved from storage.

Using index condition : Good. Filtering done at the storage engine level

Using temporary : Bad. MySQL need to create an internal temporary table to process the query.

Using filesort : Bad. . This means MySQL must do an extra pass to sort the rows in order to satisfy an ORDER BY clause.

INDEX

An index is a special lookup table that the database search engine can use to speed up data retrieval.

Mind that INDEX also slows down INSERT, UPDATE, DELETE query as it also require to update index.

Index classification based on data structure

1. B-Tree index
    a. It store data in a sorted, balanced tree structure, which allows efficient lookups, insertion, deletion
    b. Best for operator like =, >, <, BETWEEN, LIKE
2. Hash index
    a. Use hashing for search
    b. Best for comparison operation like finding user by email
    c. Cannot be used for ORDER BY or range query
3. Full-text index
    a. Designed specifically for searching text within a column in a more natural, word-based way. Instead of matching the entire string, it breaks the text into individual words and indexes them.
    b. Best for searching through long text fields like a blog post's body
4. Spatial Index (R-Tree)
    a. Used for indexing geographical data, such as coordinated
    b. Best for location based query like find nearest petrol pump

Index classification based on column composition

1. Single-column index : Created on single column
2. Multi-column (Composite) index : Created on multiple columns, order of column is very important
    a. The index is sorted by the first column, then by the second column within each value of the first, and so on.
3. Unique index : This index ensures that all values in the indexed column are unique
    a. A primary key is special case of unique index
4. Clustered vs. Non-clustered index :
    a. clustered index : This type determines the physical order of data in a table, there can be only one clustered index per table, MySQL INNDB, the primary key is the clustered index. If there is no primary key, InnoDB uses the first UNIQUE index with no NULL values.
    b. non-clustered index : The data is stored in one location, and the index is in another. The index contains pointers back to the location of the data rows. A single table can have multiple non-clustered indexes. All indexes that are not the clustered index are non-clustered.

Syntax:

```
CREATE INDEX index_name ON table_name (column1, column2, ...);
```

To create unique index

```
CREATE UNIQUE INDEX index_name ON table_name (column1, column2, ...);
```

To create a full-text index

```
CREATE FULLTEXT INDEX index_name ON table_name (column1, column2, ...);
```

To create spatial index(geometric)

```
CREATE SPATIAL INDEX index_name ON table_name(column_name)
```

To show existing index

```
SHOW INDEX FROM table_name;
```

To drop index

```
DROP INDEX index_name ON table_name;
```

CASE in SELECT

syntax:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE else_result
END
```

Ex:

```
SELECT
    T02F01,
    T02F02,
    T02F03,
    CASE
                WHEN T02F07 > 100000 THEN 'HIGH'
        ELSE 'AVERAGE'
    END
FROM T02;
```

Syntax:

```
CASE expression
    WHEN value1 THEN result1
    WHEN value2 THEN result2
    ...
    ELSE else_result
END
```

Ex:

```
SELECT
    product_name,
    CASE product_name
        WHEN 'Laptop' THEN 'High-value item'
        WHEN 'Monitor' THEN 'Fragile item'
        ELSE 'Standard item'
    END AS shipping_note
FROM
    products;
```

IF ELSE

```
IF(condition, value_if_true, value_if_false)
```

```
SELECT
        T02F01,
    T02F02,
    T02F03,
    IF (T02F07 > 100000, "HIGH", "AVERAGE")
FROM
    T02;
```

UPSERT

Update + insert

If row exists → update it

If row doesn't exists → insert it

INSERT ... ON DUPLICATE KEY UPDATE

```
INSERT INTO T01(T01F01, T01F02, T01F03) VALUES
(1, "Engineering", "Delhi")
ON DUPLICATE KEY UPDATE
    T01F02 = VALUES(T01F02),
    T01F03 = "Anand";
```

Updates existing row, inserts if not exists

REPLACE INTO

```
REPLACE INTO T01(T01F01, T01F02, T01F03) VALUES
(6, "HR", "Anand");
```

Delete row and then inserts new one

GROUP_CONCAT

Ex:

```
SELECT
    T01F02,
    GROUP_CONCAT(T02F02)
FROM
    T02
    INNER JOIN T01 ON T01F01 = T02F08
GROUP BY
    T02F08;
```

T01F02	GROUP_CONCAT(T02F02)
Engineering	Priya,Amit,Rajesh
Sales	Anjali,Vikram,Pooja
Human Resources	Sunita,Deepak
Finance	Neha

Loopup table as alternative to ENUM

Create a table with id and name and description if needed

Insert enum types into it Ex: For order status - Pending, Shipped, Delivered

Use forign key to use ENUM in main table

PARTITION

Partition is a way to divide a very large table into smaller, more manageable pieces call partition, while still having it behave as a single table for most operation.

When query include partition key it only scan related partition which improves performance known as partition pruning

Type of partition

1. RANGE partition

Ex:

```
ALTER TABLE T03
    PARTITION BY RANGE (YEAR(T03F04)) (
        PARTITION P2020 VALUES LESS THAN (2020),
        PARTITION P2021 VALUES LESS THAN (2021),
        PARTITION P2022 VALUES LESS THAN (2022),
        PARTITION P2023 VALUES LESS THAN (2023),
        PARTITION P2024 VALUES LESS THAN (2024),
        PARTITION P_FUTURE VALUES LESS THAN MAXVALUE
    );
```

It can also be done when creating table

```
CREATE TABLE table_name (defination)
PARTITION BY RANGE ...
```

2. List partitioning

partition based on specific list of discrete values

```
CREATE TABLE customers (
    customer_id INT,
    name VARCHAR(100),
    region_code INT,
    PRIMARY KEY (customer_id, region_code)
)
```

```
PARTITION BY LIST (region_code) (
    PARTITION p_north VALUES IN (1, 5, 9),
    PARTITION p_south VALUES IN (2, 6, 10),
    PARTITION p_east VALUES IN (3, 7, 11),
    PARTITION p_west VALUES IN (4, 8, 12)
);
```

3. Hash partitioning

Distributes data evenly across a specified number of partitions. You provide a column and the number of partitions, and MySQL uses a hash function on the column's value to decide where to place the row. This is useful for ensuring an even data distribution when there's no obvious range or list.

```


ALTER TABLE T03
PARTITION BY HASH(T03F01)   -- It excepts a expression
PARTITIONS 8;
```

4. KEY partitioning

Similar to HASH but MySQL generates the hash internally using one or more columns, typically primary keys.

```
ALTER TABLE T03
PARTITION BY KEY(T03F01)
PARTITIONS 8;
```

5. Subpartitioning

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );
```

> Every unique including primary key must include partition key

To get all partition details

```
SELECT
    PARTITION_NAME,
    TABLE_ROWS,
    PARTITION_DESCRIPTION
FROM
    INFORMATION_SCHEMA.PARTITIONS
WHERE
    TABLE_SCHEMA = 'employee'
    AND TABLE_NAME = 't03';
```

To remove partitioning

```
ALTER TABLE table_name REMOVE PARTITIONING;
```

To remove a partition

It will remove data without invoking trigger

```
ALTER TABLE table_name DROP PARTITION partition_name_1, partition_name_2;-
```

Window function

window function is a special type of SQL function that performs a calculation across a set of table rows that are somehow related to the current row.

The key difference between a window function and a standard aggregate function is that window function do not collapse the rows.

An aggregate function with GROUP BY returns a single summary row for each group, whereas a window function returns a value for *every single row*.
It is used to find running total, ranking, moving averages.

Ranking function

1. ROW_NUMBER() : Assigns a unique, sequential number to each row (e.g. 1,2,3,4)
2. RANK() : Assigns a rank based on the ORDER BY clause, (1,2,2,4)
3. DENSE_RANK() : Assigns a rank but does not leave gaps (1,2,2,3)

Value window function

LAG(col, offset, default) : Accesses data from previous row in the partition

LEAD(col, offset, default) : Access data from a subsequent row in the partition)

FIRST_VALUE(col) : Gets the value of col from first row

LAST_VALUE(col) : Gets the value of col from last row

Ex:

```
SELECT
    T02F01,
    T02F02,
    T02F03,
    T02F07,
    T01F02,
    ROW_NUMBER() OVER W  AS SALARY_RANK,
    SUM(T02F07) OVER W AS RUNNING_TOTAL,
    LAG(T02F07, 1, 0) OVER W AS PREVIOUS_SCANNED_SALARY
FROM
    T02
    INNER JOIN T01 ON T01F01 = T02F08
WINDOW W AS (PARTITION BY T02F08 ORDER BY T02F07 DESC);
```

Keyset pagination alternative to LIMIT OFFSET

If offset become larger for big tables it is not best to way to use it to provide pagination support

So we can use Keyset pagination in which we directly go to desired row using where and then use LIMIT

For which we can use sequential Id or indexed date or any other related field

Ex:

```
SELECT * FROM products
ORDER BY id
LIMIT 10 OFFSET 1000;

SELECT * FROM products
WHERE id > 1000
ORDER BY id
LIMIT 10;
```

Both query is give same result but 2nd will execute faster

INFORMATION AND PERFORMANCE SCHEMA

both information_schema and performance_schema are special, read-only databases that provide insights into the server itself, but they serve very different purposes.

- **information_schema** is about **metadata** (data about the data). It tells you *what* exists in your databases.
- **performance_schema** is about **performance metrics**. It tells you *what's happening* inside the server and how efficiently it's running.

Official documentation

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

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

```
-- Give information about database
SELECT
   SCHEMA_NAME,
   DEFAULT_CHARACTER_SET_NAME
FROM
   information_schema.SCHEMATA;

-- Give information about tables
SELECT
   TABLE_NAME,
   ENGINE,
   TABLE_ROWS,
   (DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024 AS total_size_mb
FROM
   information_schema.TABLES
WHERE
   TABLE_SCHEMA = 'employee';

-- Give information about columns
SELECT
   COLUMN_NAME,
   DATA_TYPE,
   IS_NULLABLE, COLUMN_DEFAULT
FROM
   information_schema.COLUMNS
WHERE
   TABLE_SCHEMA = 'employee' AND TABLE_NAME = 'T02'
ORDER BY
```

```
    ORDINAL_POSITION;

-- Information about index
SELECT
    INDEX_NAME,
    COLUMN_NAME,
    NON_UNIQUE
FROM
    information_schema.STATISTICS
WHERE
    TABLE_SCHEMA = 'employee' AND TABLE_NAME = 'T02';

-- Constraints
SELECT
    CONSTRAINT_NAME,
    TABLE_NAME,
    COLUMN_NAME,
    REFERENCED_TABLE_NAME,
    REFERENCED_COLUMN_NAME
FROM
    information_schema.KEY_COLUMN_USAGE
WHERE
    TABLE_SCHEMA = 'employee' AND TABLE_NAME = 'T02';
```

```
-- avg_timer_wait is measured in picosecond (10^-12)
SELECT
    digest_text,
    avg_timer_wait
FROM
    performance_schema.events_statements_summary_by_digest
ORDER BY
    avg_timer_wait DESC
LIMIT 5;
```

Recursive CTE

```
WITH RECURSIVE numbers AS (
    SELECT 1 AS n      -- Base case: Start with 1
    UNION ALL
    SELECT n + 1       -- Recursive case: Add 1 each time
    FROM numbers
    WHERE n < 10       -- Stopping condition
)
SELECT * FROM numbers;
```

Ex:

```
WITH RECURSIVE emp_hierarchy AS (
    SELECT id, name, manager_id, 0 AS level
    FROM employees
    WHERE manager_id IS NULL      -- Base case: CEO
    UNION ALL

    SELECT e.id, e.name, e.manager_id, eh.level + 1
    FROM employees e
    INNER JOIN emp_hierarchy eh ON e.manager_id = eh.id
)
SELECT * FROM emp_hierarchy;
```

JSON Functions

```
-- creating json
SELECT JSON_OBJECT('ID', 1, 'NAME', 'ABC');  -- create json object
SELECT JSON_ARRAY(1, NULL, 'ABC', 3.14); -- create json array

-- search json
SELECT JSON_EXTRACT('{"user": {"name": "Bob"}}', '$.user.name');  -- find at given path  $ is document
SELECT JSON_UNQUOTE(JSON_EXTRACT('{"user": {"name": "Bob"}}', '$.user.name'));  -- to unquote

SELECT JSON_CONTAINS('[1, 2, 3]', '2');  -- check if array containt
SELECT JSON_CONTAINS_PATH('{"a": 1, "b": {"c": 2}}', 'one', '$.a', '$.d');  -- check if json contain given one or all paths

SELECT JSON_KEYS('{"id": 1, "name": "Alice"}');   -- give array of keys
SELECT JSON_SEARCH('{"a": "hello", "b": ["world"]}', 'one', 'world'); -- give path given string

-- modify json
SELECT JSON_SET('{"a": 1, "b": 2}', '$.a', 10, '$.c', 3);  -- replace old one add if not exists
SELECT JSON_INSERT('{"a": 1}', '$.a', 10, '$.b', 2);  -- don't replace add only if not exists
SELECT JSON_REPLACE('{"a": 1}', '$.a', 10, '$.b', 2);   -- only replace don't add new
SELECT JSON_REMOVE('{"a": 1, "b": [2, 3]}', '$.b');  -- remove
SELECT JSON_ARRAY_APPEND('[1, 2]', '$', 3);   -- apped into json array
SELECT JSON_MERGE_PRESERVE('{"a": 1, "b": 2}', '{"a": 3, "c": 4}');  -- merge two json preserving both's data


-- other
SELECT JSON_TYPE('{"a": 1}');  -- check typr OBJECT, ARRAY, STRING, INTERGER, NULL
SELECT JSON_VALID('{"a": 1}'); -- check if json is valid or not
SELECT JSON_LENGTH('{"a": 1, "b": [10, 20]}', '$.b');  -- find number of keys for object and number of element for array
SELECT JSON_DEPTH('{"a": {"b": {"c": 1}}}'); -- find depth of json
```