03. Accessing The DBMS

Accessing the DBMS: Options

OPTION 1: Command Line Tools

OPTION 2: Graphical User Interfaces

MySQL References

- Help Facility: **`Help`**, **`Help Contents`**
- Online Reference Manual
- MySQL Workbench Documentation
- Internet Forums, Websites, Youtube
- MySQL Textbooks
- Database Systems Textbooks

- …
- …

Database Languages *(SQL Sublanguages)*

- **Data Definition Language (DDL)**
  - DDL is used to define the structure of the database schema.
  - It includes commands such as CREATE, ALTER, DROP, TRUNCATE, and COMMENT.
  - **Examples:** Creating, modifying structure, dropping database objects.
- **Data Manipulation Language (DML)**
  - DML is used to manipulate data stored in the database.
  - It includes commands such as SELECT, INSERT, UPDATE, DELETE, and MERGE.
  - **Examples:** Retrieving data from tables, adding new records, updating existing records, deleting records, etc.
- **Data Query Language (DQL):**
  - Primarily concerned with querying and fetching data
  - Allows users to specify the criteria for selecting and filtering data
- **Transaction Control Language (TCL)**
  - TCL is used to manage transactions within the database.
  - It includes commands such as COMMIT, ROLLBACK, and SAVEPOINT.
  - **Examples:** Committing changes made by transactions, rolling back changes, setting savepoints within transactions, etc.
- **Data Control Language (DCL)**
  - DCL is used to control access to data within the database.
  - It includes commands such as GRANT and REVOKE.
  - **Examples:** Granting specific privileges to users or roles, revoking privileges, etc.

MySQL Datatypes

Numeric data types	Integer Types: INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT
	Fixed-Point Types: DECIMAL, NUMERIC
	Floating-Point Types: FLOAT, DOUBLE
	Bit-Value Type: BIT
Date and time data types	DATE, DATETIME, TIMESTAMP, TIME, YEAR
String data types	CHAR, VARCHAR
	BINARY, VARBINARY
	BLOB, TEXT
	ENUM
	SET
Spatial data types	GEOMETRY, POINT, LINESTRING, POLYGON
	MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION
JSON data types	Enables efficient access to data in JavaScript Object Notation(JSON) documents

DDL

- Examples:

```
CREATE Database KULibrary;

CREATE TABLE Author (
    AuthorID INT PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255)
);

ALTER TABLE Author Add dob DATE;

Drop Database KULibrary;
```

```
CREATE TABLE city_copy
AS
SELECT * FROM world.city;

CREATE VIEW ke_cities AS
SELECT name, countrycode FROM city_cpy
WHERE countrycode LIKE '%ke%';

CREATE PROCEDURE sp_List_Authors(IN Lim Int)
BEGIN
     SELECT name, email FROM Author LIMIT Lim;
END
```

Database Objects

- Database: A container that holds tables, views, stored procedures, functions, and other objects. It's used to organize and manage data into logical groups.
- Table: The fundamental building blocks of a relational database system. A structured collection of data organized into rows and columns.
- View: A virtual table derived from one or more tables or other views. It presents data from the underlying tables in a predefined format, without storing the data itself.
- Procedure: A set of SQL statements stored in the database and executed as a single unit. It can accept input parameters, perform operations, and return results to the caller. Procedures are used to encapsulate frequently executed tasks, enhance code reusability, and improve performance by reducing network traffic.

Database Objects

- Function: A reusable block of code that accepts input parameters, performs computations, and returns a single value. Functions enhance the flexibility and modularity of SQL code by encapsulating common logic and calculations.
- Event: An event is a scheduled task that performs a specific action at a predefined time or interval.It can be used to automate database maintenance tasks, generate reports, or execute other SQL statements.
- Trigger: A database object that automatically executes in response to certain database events, such as INSERT, UPDATE, or DELETE operations on a table. Triggers are defined to enforce business rules, maintain data integrity, or audit changes to the database.

They contain SQL code that executes when the specified event occurs, allowing for complex database logic and actions to be implemented.

Using scripted Instructions

Why?

- Scripted instructions are convenient for query sets that are run very frequently or repeatedly
- To help generate queries similar to those used earlier
- Especially useful for multiline/complex queries so that in case of errors, then only the erroneous part ought to be fixed
- To be able to distribute the query/query set to other people
- To catch the output in a file for further processing

```
CREATE DATABASE sit301;

USE sit301;

CREATE TABLE employee(
EmpNo    INTEGER PRIMARY KEY,
EmpName  VARCHAR(50)     NOT NULL,
Dob      DATE,
Address  VARCHAR(40),
Town     VARCHAR(20),
Salary   DECIMAL,
DeptNo INTEGER
);

Create Table Department(
DeptNo  Integer  Primary Key,
DeptName  Varchar(20),
DeptBase  Varchar(20),
MgrEmpNo  Integer,

FOREIGN KEY (MgrEmpNo) REFERENCES Employee(empno)
);


ALTER TABLE Employee
ADD CONSTRAINT FOREIGN KEY(DeptNo) REFERENCES Department(DeptNo);

INSERT INTO Department
VALUES
('17', 'Finance', 'Town Center', NULL),
('18', 'Marketing', 'Westlands', NULL),
('19', 'Production', 'Industrial Area', NULL),
('20', 'Admin', 'Town Center', NULL),
('21', 'Human Resources', 'Town Center', NULL);


INSERT INTO Employee
(EmpNo, EmpName, Dob, Address, Town, Salary, DeptNo)
VALUES
('1001', 'Eugene Wambua', '1970-11-02', 'Box 12345', 'Mombasa', '25000', '21'),
('1002', 'Joyce Wanjiku', '2001-03-30', 'Box 11111', 'Nakuru', '35000', '19'),
```

Examples

1. Batch Command File	**At the shell prompt:**
	
	`C:\Program Files\MySQL\MySQL Server 8.0\bin>Mysql -uSampleUser -p <"C:\myfolder\mysqlscript.sql"`
	
	`C:\Program Files\MySQL\MySQL Server 8.0\bin>Mysql -uSampleUser -p <\\10.2.17.120\scripts$\mysqlscript.sql`
	
	At the MySql Prompt (open a file and executes commands contained therein):
	
	`mysql> \. C:\myfolder\mysqlscript.sql`

2. Loading Data from a text file	**At the MySql Prompt**
	
	`mysql> LOAD DATA INFILE 'C:\myfolder\CustomerData.txt' INTO TABLE Customer`
	
	`mysql> LOAD DATA INFILE 'CustomerData.txt' INTO TABLE Customer`
	
	*** instructions are subject to security controls by the server*

Data Manipulation Language (DML)

- Key Terms:
  - INSERT INTO
  - UPDATE
  - DELETE

DML COMMAND	General Syntax	Example
INSERT INTO	INSERT INTO *table_name* (*column1*, *column2*, *column3*, ...) VALUES (*value1*, *value2*, *value3*, ...); INSERT INTO *table_name* VALUES (*value1*, *value2*, *value3*, ...);	INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) VALUES ('Paul', 'Michael Onyango', PO Box 123456', 'Nairobi', '00100', 'Kenya'); INSERT INTO Customers VALUES ('Paul', 'Michael Onyango', PO Box 123456', 'Nairobi', '00100', 'Kenya');

Data Manipulation Language

DML COMMAND	General Syntax	Example
UPDATE Modifies existing records in a table.	UPDATE *table_name* SET *column1* = *value1*, *column2* = *value2*, ... WHERE *condition*;	/* Updates single record, multiple columns */ UPDATE Customers SET ContactName = 'John Owino', City= 'Nakuru' WHERE CustomerID = 135; /*(Potentially) updates multiple records, single column*/ UPDATE Customers SET Title='Herr' WHERE Country='Germany'

Data Manipulation Language

DML COMMAND	General Syntax	Example
DELETE Erases existing records in a table.	DELETE FROM *table_name* WHERE *condition*; DELETE FROM *table_name*;	DELETE FROM Customers WHERE CustomerName='John Owino' DELETE FROM Customers;
Care should be taken when using the DELETE statement, as deleted data cannot be easily recovered.		

Data Query Language

- Basic statement : The SELECT statement

```
SELECT
      [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
      select_expr [, select_expr] ...
      [into_option]
      [FROM table_references
        [PARTITION partition_list]]
      [WHERE where_condition]
      [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
      [HAVING where_condition]
      [WINDOW window_name AS (window_spec)
          [, window_name AS (window_spec)] ...]
      [ORDER BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
      [LIMIT {[offset,] row_count | row_count OFFSET offset}]
      [into_option]
      [FOR {UPDATE | SHARE}
          [OF tbl_name [, tbl_name] ...]
          [NOWAIT | SKIP LOCKED]
        | LOCK IN SHARE MODE]
      [into_option]

into_option: {
      INTO OUTFILE 'file_name'
          [CHARACTER SET charset_name]
          export_options
    | INTO DUMPFILE 'file_name'
    | INTO var_name [, var_name] ...
}
```


Data Query Language: Select

- Used to Retrieve Data and make other general queries from the system

SELECT column_name FROM table_name [WHERE conditions];	**SELECT** - Starts the query and instructs MySQL on what task it must perform. **FROM** (mandatory) - Specifies the table or tables involved in the query. If multiple tables, there's need to separate their names and specify how they are to be related to each other. **WHERE** (optional) - specifies the criteria to define the required data
Other optional clauses	**GROUP BY** organizes the retrieved data by grouping them by columns. **HAVING** relates to GROUP BY. When this clause is applied, it selects and returns only those rows having the specified conditions TRUE. **ORDER BY** sorts the records in the returned result-set. **DISTINCT** removes duplicates from the result-set. **LIMIT** controls the number of rows retrieved by the query.

Conditions in SQL

- Used for filtering and matching records
- Conditions must be evaluable into a True or False value when tested against records
- Simple Conditions e.g.

```
WHERE Town='Naivasha'
```

- Complex Conditions e.g.

```
WHERE Town='Naivasha' AND Balance>10000;
```

- Comparison Operators: `= ; < ; <= ; > ; >=; != or <> ;`
- Logical Operators: **AND ; OR; NOT**
- Comparison Operator for special value NULL: **IS**

=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Sorting Records

- Use the ORDER BY clause

Primary, Secondary ... Sort Order

```
SELECT ContactName, email, Dob,
Town, Balance
FROM Contacts
ORDER BY Town, Balance;
```

Sorting by Single Column

```
SELECT ContactName, email, Dob,
Town, Balance
FROM Contacts
ORDER BY Balance;
```

Ascending vs Descending Sort Order

```
SELECT ContactName, email, Dob,
Town, Balance
FROM Contacts
ORDER BY Town, Balance Desc;
```

Popular MySQL Functions

Date & Time Functions	Current_Date, CurDate Current_Time, Current_TimeStamp, CurTime AddDate, AddTime Date_Add, DateDiff Date Day, DayName, DayofMonth, DayofWeek, DayofYear Week, Weekday, WeekofYear, Year, Yearweak
String Functions	Mid ASCII BIN, HEX, OCT CHAR_LENGHT, LENGTH FORMAT INSTR LCASE, UCASE, LOWER, UPPER

Popular MySQL Functions

Numeric Functions	ABS, SQRT COS, COT, ACOS, SIN, TAN LN, LOG, LOG10, LOG2 PI, POWER, RADIANS RAND ROUND TRUNCATE
Statistical Functions	COUNT, MAX, MIN, SUM, AVG STDDEV, VARIANCE

Data Control Language (DCL):

- Statements that are used to control user access to database objects

User Management	`CREATE USER 'usr1'@'localhost' IDENTIFIED BY 'usr1pwd';`
	`ALTER USER 'usr1'@'localhost' IDENTIFIED BY 'newpwd';` `FLUSH PRIVILEGES /*to reload the grant tables */`
	`DROP USER 'usr1'@'localhost';`

Allocating and Deallocating Privileges	**Key Statements:** GRANT, REVOKE
	Privileges:
	All; CREATE; DROP; DELETE; INSERT; SELECT; UPDATE; EXECUTE;
	Objects: TABLE, VIEW, PROCEDURE, FUNCTION, PACKAGE or SEQUENCE
	Example:
	`GRANT SELECT, DELETE, UPDATE ON Customers TO 'usr1'@'localhost'`
	`REVOKE DELETE, UPDATE ON Customers FROM 'usr1'@'localhost'`

Transaction Control Language (TCL)

Transaction = a logical unit of work that contains one or more SQL statements

Transaction Example: Transfer KSh 10,000 from **Savings Account no 123456** to **Current Account no 456789**

Stages	
1. Subtract amount from source account	`UPDATE account SET balance=balance-10000 WHERE AccountNo=123456;`
2. Increase balance of destination account	`UPDATE account SET balance=balance-10000 WHERE AccountNo=456789 ;`
3. Record transaction in transactions journal	`INSERT INTO journal (TransactionDate, Description, SourceAccount, DestinationAccount, Amount) VALUES (Current_Date(),'Bank Transfer', '123456','456789','10000');`

Transaction Control Language (TCL): Key Commands

- **COMMIT:** The command saves changes invoked by a transaction to the database.
- **ROLLBACK:** Undoes the suggested effects of the current transaction, canceling its changes.
- **SAVEPOINT:** Command used for dividing (or) breaking a transaction into sub-units so that the user has a chance of roll backing the transaction up to a specified point instead of the entire transaction.
- **START TRANSACTION:** Marks the beginning of the transaction

```
START TRANSACTION;
     UPDATE account SET balance=balance-10000 WHERE AccountNo=123456;
     UPDATE account SET balance=balance-10000 WHERE AccountNo=456789 ;
     INSERT INTO journal VALUES (Current_Date(), '123456','456789','10000');
COMMIT;
```

- By default, MYSQL runs with autocommit enabled
  - As soon as you execute a statement that update modifies the table and the change is permanent. The change cannot be rolled back
- To disable autocommit explicitly:

```
SET autocommit =0;
```


Working with Views

1. Definition

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;
```

2. Data Retrieval:

```
SELECT * FROM view_name;
```

3. **Data modification**: Depending on the complexity of the view definition, you can often perform inserts, updates, and deletes on views. However, certain conditions must be met, e.g. the view must not be based on constructs like aggregate functions or joins.

4. **Security**: Views can be used to restrict access to specific columns or rows of a table, providing a layer of security. Users can be granted permissions to access views without having direct access to the underlying tables.

Working With Views

- **Performance:** Views **can** improve performance by precomputing complex queries and storing their results, reducing the need to repeatedly execute the same complex logic.
- **Updatability:** Not all views are updatable. MySQL imposes certain restrictions on updatable views. Generally, views are updatable if they meet certain criteria like containing only one table in the FROM clause, not using aggregate functions, not using GROUP BY or HAVING clauses, and so on.
- **Maintenance:** Views can simplify maintenance by encapsulating complex logic in a single object. If the underlying tables change, you only need to update the view's definition rather than modifying multiple queries throughout your application.
- **Nested Views:** MySQL allows you to create views based on other views, which can be useful for breaking down complex queries into manageable parts

Working with Stored Procedures

- MySQL procedures are stored routines that allow you to group one or more SQL statements into a reusable unit and execute them when needed.
- Advantages of MySQL Procedures:
  - Modularity and Reusability:
    - Procedures allow one to encapsulate SQL code into reusable units, making it easier to manage and maintain. They reducing code duplication and promote modularity.
  - Improved Performance:
    - MySQL procedures are precompiled and stored in the database, reducing parsing and optimization overhead during execution. Procedures therefore can minimize network traffic and round-trips between the application and the database server, leading to improved performance
  - Enhanced Security:
    - Procedures can help enforce security policies by encapsulating sensitive SQL logic within the database. Users only need appropriate permissions to execute procedures, reducing the risk of unauthorized access to underlying data or SQL statements.
  - Encapsulation of Business Logic:
    - Procedures enable one to implement complex business logic directly within the database, closer to the data it operates on.
    - This helps maintain consistency and integrity of data, as well as enforce business rules consistently across applications.
  - Version Control and Maintenance:
    - Procedures can be version-controlled along with other database objects, allowing you to track changes and revert to previous versions if necessary.
    - Centralizing logic in procedures simplifies maintenance tasks, as updates or fixes can be applied in one place without affecting multiple parts of the application.
  - Improved Code Readability:
    - By abstracting complex SQL logic into named procedures with meaningful names, you can improve the readability and maintainability of your codebase. This makes it easier for developers to understand and collaborate on database-related tasks.