**Databases and DBMS**

**Database**: Container to store data

**DBMS**: Software that is used to manage a database.

\*Define \*Construct \*Manipulate \*Share

(Hierarchical Database and Network based Databases [1960s-1980s])

**Relational Database**: Where data is stored in a way that they are related to each other over a key

(E.F. Codd)

Multivalued and composite values are not supported (only atomic values)

\*Consistency \*Security \*Ease of Use

**Non-relational Database**: Each database store data differently and is used to manage specific kinds of data.

Ex: key-valued database (dictionary, Radis), document database (Json, MangoDB), search engine database, graph database (nodes and edges, Neo4j), wide-column database (Column Family- rows and columns, Cassandra) etc.

\*Flexibility \*Cost effective \*Suitable for small scale products

**Database Users**:

\*Database administrators \*Database designers \*End Users \*Data Analysts and SE

**Data Models**: define the structure of the database – to achieve data abstraction

\*High level or conceptual model - Entities, attributes, relationships

\*Low-level or physical model - how data is stored, defines access paths to search records efficiently

\*Representational or implementation model - Network based and hierarchical based (record structures)

*Data Schema*: subset of the meta data – blueprint of a database

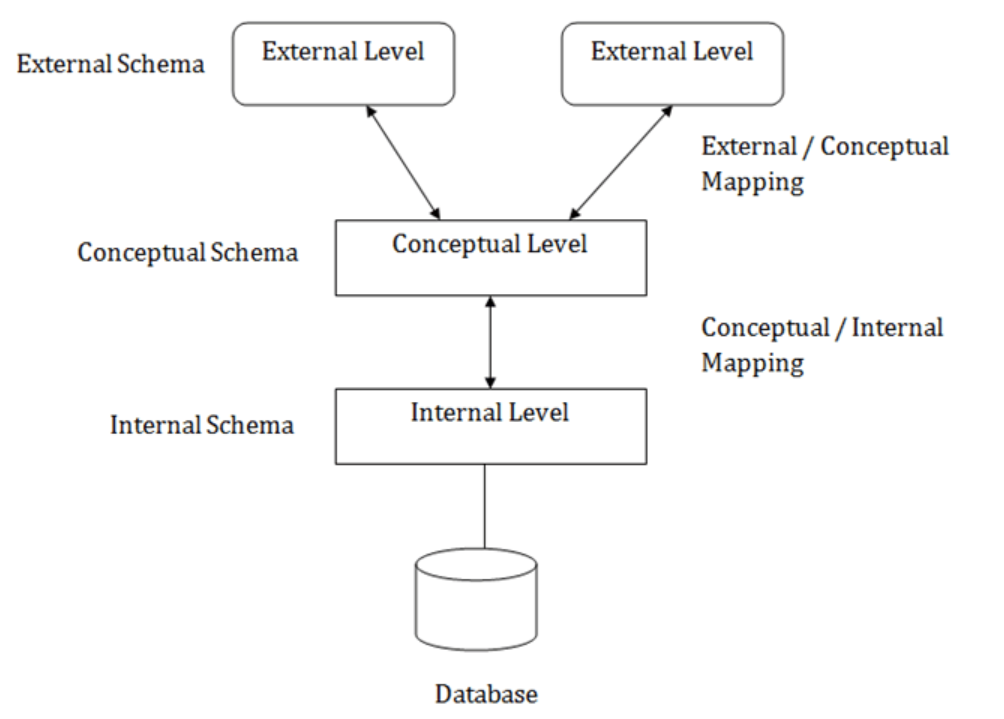
*Schema diagram*: diagrammatical representation of a schema

*Schema construct*: each object within a schema

*Schema Evolution*: when a schema of a database is changed

*Database State* (instance or snapshot): Data at a specific time

**Three-Schema Architecture**:



\*Way used to manage access to data

\*Separates the actual database and the end-users

\*Internal schema: Access paths, addresses, data types, memory

\*Conceptual schema: what DA and programmers see

\*External schema: what end –users see

**Data Independence**: changes don’t affect the higher levels

Logical DI: changes in conceptual level

Physical DI: changes in internal level

**Data Languages**:

*DDL (Data Definition Language)*: no strict separation of levels (also for Conceptual Schema) [Procedural (complex, set) and non-procedural (embedded, one record)]

*SDL (Stored Definition Language)*: internal schema

*VDL (View Data Language)*: user views

*DML (Data Manipulation Language)*: manipulate data stored in the database

*DCL (Data Control Language)*: control access to the data stored

**Database architecture**:

1.Centralized DB Architecture: Display Monitor <-------> Server

2.Two-tier Architecture: Client <---------> Server (because of PCs and workstations)

3.Three-tier Architecture: Client <---------> Web Server <-------> DBMS (because of WWW)

-----------------------------------------------------------------------------

**Structured Query Language (SQL)**

* Domain based – used only in DBs
* Declarative
* DDL: CREATE, ALTER, DROP, TRUNCATE, RENAME, COMMENT
* DCL: REVOKE, GRANT
* DML: SELECT, UPDATE, DELETE, INSERT
* PL-SQL: Procedural SQL (triggers, stored procedures)
* TCL (Transaction Control Language): COMMIT, ROLLBACK, SAVEPOINT
* Constraints: PRIMARY KEY, FORIEGN KEY, CHECK, UNIQUE, DEFAULT, NOT NULL, INDEX, ENUM

Creating a Database:

* Creates a database

*CREATE DATABASE* *database\_name;*

* Deletes a database if it already exists

*DROP DATABASE IF EXISTS database\_name;*

* Mentions which database is to be used to execute further queries

USE database\_name;

Creating a table:

* Creating a table named payment\_methods with columns. This is a comment in MySQL

*CREATE TABLE payment\_methods (*

*payment\_method\_id* *int(4) NOT NULL AUTO\_INCREMENT,*

*name* *varchar(50) NOT NULL,*

*PRIMARY KEY (payment\_method\_id)*

*) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8mb4*  *COLLATE=utf8mb4\_0900\_ai\_ci;*

Inserting data:

* Inserting values in the same order as the columns

*INSERT INTO table\_name VALUES (value1, value**2,... valueN);*

* Inserting specific column values in any order

*INSERT INTO table\_name (column1, column**2,...columnN) VALUES (value1, value**2,.... valueN);*

* Inserting multiple rows of data together

*INSERT INTO table\_name(column1, column**2,....columnN)*

*VALUES (value1, value**2,... valueN),*

*(value1, value**2,... valueN)…......*

*(value1, value**2,... valueN);*

Ex:

*INSERT INTO payment\_methods VALUES (1,'Credit Card');*

#or

*INSERT INTO payment\_methods(payment\_method\_id, name) VALUES (1,'Credit Card');*

Altering values:

* ALTER is used to add, delete or modify columns and also table constraints.
* Adding a column to an already existing table

Syntax:

*ALTER TABLE table\_name*  
*ADD column\_name datatype;*

Ex:

*ALTER TABLE Customers ADD email varchar (20);*

* Removing a column from a table

Syntax:

*ALTER TABLE table\_name*  
*DROP COLUMN column\_name;*

Ex:

ALTER TABLE Customers DROP COLUMN email;

* Changing the data type of a column name

Syntax:

ALTER TABLE *table\_name*  
*MODIFY* COLUMN *column\_name datatype*;

Ex:

ALTER TABLE Customers MODIFY COLUMN email varchar(255);

* Changing the name of a table

Syntax:

ALTER TABLE old\_*table\_name*  
*RENAME* new\_table\_name;

OR

RENAME TABLE old\_*table\_name TO new\_table\_name;*

*Ex:*

*RENAME TABLE customers TO new\_customers;*

Drop and Truncate:

* DROP is used to delete a table along with its data
* TRUNCATE is used to delete only data from the table

Syntax:

*DROP TABLE table\_name;*

*TRUNCATE TABLE table\_name;*

Ex:

*DROP TABLE customers;*

*TRUNCATE TABLE customers;*

Select Clause:

* returns data with all the columns in the mentioned table

*SELECT \* FROM table\_name;*

* returns data with the specified columns

*SELECT column1, column2, column3...columnN FROM table\_name;*

* returns data with the specified columns such that they have only distinct data, avoids duplicate entries

*SELECT DISTINCT column1, column2, column3,......columnN FROM table\_name;*

Ex:

*SELECT \*, (unit\_price \* quantity) AS total\_price*

*FROM order\_items*

*WHERE order\_id = 6 AND (unit\_price \* quantity) > 30*

Where Clause:

* returns all the data that fulfill the given condition (strings should be written within single or double quotes)

*SELECT \* FROM table\_name WHERE condition;*

* Arithmetic Operators: +, -, \*, /, % (Precedence: +, - < \*, /)
* Comparison Operators: =, >, <, >=, <=, != or <>, LIKE, BETWEEN, IN
* Logical Operators: AND, OR, NOT (Precedence: OR < AND)
* The precedence of operators can be altered using parenthesis.
* AND returns the records if both the statements evaluate to true

Syntax:

*SELECT column1, column2, ...columnN*  
*FROM table\_name*  
*WHERE condition1 AND condition2 AND condition3 ...;*

Ex:

*SELECT \* FROM Customers*  
*WHERE state = 'VA' AND points >= 1000;*

* OR returns the records if either of the statement evaluates to true

Syntax:

*SELECT column1, column2, ...columnN*  
*FROM table\_name*  
*WHERE condition1 OR condition2 OR condition3 ...;*

Ex:

*SELECT \* FROM Customers*  
*WHERE state = 'VA' OR state = 'MA';*

* NOT condition returns records opposite of the what the expression will be evaluated to.

Syntax:

*SELECT column1, column2, ...columnN*  
*FROM table\_name*  
*WHERE NOT condition;*

Ex:

*SELECT \* FROM Products*  
*WHERE NOT quantity\_in\_stock = '56';*

* BETWEEN is used to get records between a range of values
* Includes value1 and value2

Syntax:

*SELECT column1, column2, ...columnN*  
*FROM table\_name*  
*WHERE column\_name BETWEEN value1 AND value2;*

*Ex:*

*SELECT \* FROM Customers*  
*WHERE birth\_date BETWEEN '1990-01-01' AND '2000-01-01';*

* IN keyword is used to get the records that match with anyone of the values specified within the parenthesis

Syntax:

*SELECT column1, column2, ...columnN*  
*FROM table\_name*  
*WHERE column\_name IN (value1, value2, ...);*

Ex:

*SELECT \* FROM Customers*  
*WHERE state IN ('VA', 'MA', 'IL');*

* LIKE keyword is used to get the records that match a specified pattern
* % - matches any number of characters (%field% - returns records which have ‘field’ in them)
* \_ – matches only one character (\_\_a – returns records that have ‘a’ exactly after two other characters)
* \_, % - also called as wildcard characters

Syntax:

*SELECT column1, column2, ...columnN*  
*FROM table\_name*  
*WHERE column\_name LIKE pattern;*

Ex:

*SELECT \* FROM Customers*  
*WHERE first\_name LIKE 't%';*

Regular Expressions:

* REGEXP is used to match a pattern
* ^field – data starts with field
* field$ - data ends with field
* ^field1|feild2|feild3$ - logical OR
* [abcd] or [a-d]
* [a-f0-9] - range of values
* [dfg]e – de, fe, ge

Ex:

*SELECT \**

*FROM customers*

*WHERE last\_name REGEXP 'ey$|on$';*

*SELECT \**

*FROM customers*

*WHERE last\_name REGEXP '^my|[se]';*

*SELECT \**

*FROM customers*

*WHERE last\_name REGEXP 'b[ru]';*

IS NULL operator:

* IS NULL operator is used to get the records that have the specified column value as NULL
* Can be used along with NOT operator to get the non-null records

Syntax:

*SELECT column\_names*  
*FROM table\_name*  
*WHERE column\_name IS NULL;*

Ex:

*SELECT \**  
*FROM orders*  
*WHERE shipper\_id IS NULL;*

*SELECT \**  
*FROM orders*  
*WHERE shipper\_id IS NOT NULL;*

Order by clause:

* ORDER BY clause is used to order the result in ascending or descending order
* By default, the records will be ordered in the ascending order of the primary key
* In MySQL, column names that are not there in the select clause can also be used in the order by clause

Syntax:

*SELECT column1, column2, ...columnN*  
*FROM table\_name*  
*ORDER BY column1, column2, ... ASC|DESC;*

Ex:

*SELECT \**

*FROM Order\_items*

*WHERE order\_id = 2*

*ORDER BY quantity \* unit\_price DESC;*

LIMIT Clause:

* LIMIT is used to limit the number of records returned by the query

Syntax:

*SELECT column1, column2, ...columnN*

*FROM table\_name*  
*WHERE condition*  
*LIMIT number;*

Ex:

*SELECT \**

*FROM customers*

*ORDER BY points DESC*

*LIMIT 3;*

JOINS: (Used to combine columns)

Inner Joins:

* Joins are used to get records from multiple tables based on a common column name and if they have same values
* In MySQL JOIN keyword performs inner join be default
* Tables across the databases can also be joined together

Syntax:

*SELECT column1, column2, ...columnN*

*FROM table1*  
*INNER JOIN table2*  
*ON table1.column\_name = table2.column\_name;*

Ex:

--Alias name for the table

--column name is prefixed with table alias name to avoid ambiguity as to which column name is to be selected

--This is an explicit inner join

*SELECT order\_id, oi.product\_id, quantity, name*

*FROM order\_items oi*

*JOIN products p ON oi.product\_id = p.product\_id;*

--Implicit Inner Join. Same as the above one but is not recommended to use.

*SELECT order\_id, oi.product\_id, quantity, name*

*FROM order\_items oi, products p*

*WHERE oi.product\_id = p.product\_id;*

--Joining multiple tables

*SELECT o.order\_id, o.product\_id, quantity, name*

*FROM ((order\_items o*

*JOIN products p ON o.product\_id = p.product\_id)*

*JOIN order\_item\_notes oin ON o.order\_id = oin.order\_id);*

*SELECT p.payment\_id, c.client\_id, c.name, invoice\_id, date, amount, pm.name, c.city, c.state, c.phone*

*FROM payments p*

*JOIN payment\_methods pm ON p.payment\_method = pm.payment\_method\_id*

*JOIN clients c ON c.client\_id = p.client\_id;*

--Joining tables from multiple databases

*SELECT \**

*FROM order\_items oi*

*JOIN sql\_inventry.products p ON oi.product\_id = p.product\_id;*

Outer Join:

* Inner join returns records that only match the condition, whereas OUTER JOIN returns the records whether the condition matches or not.
* LEFT OUTER JOIN or LEFT JOIN returns all the records from the left table along with the matched record.
* RIGHT OUTER JOIN or RIGHT JOIN returns all the records from the right table along with the matched record.

Syntax:

*SELECT column1, column2, ...columnN*

*FROM table1*  
*RIGHT JOIN table2*  
*ON table1.column\_name = table2.column\_name;*

Ex:

*SELECT order\_id, oi.product\_id, quantity, name*

*FROM order\_items oi*

*RIGHT JOIN products p ON oi.product\_id = p.product\_id;*

Self Joins:

* Joining a table with itself
* Can be done using inner joins and outer joins

Syntax:

*SELECT column1, column2, ...columnN*

*FROM table1 T1*   
*JOIN table1 T2 ON condition;*

Ex:

*SELECT e.employee\_id, e.first\_name, e.job\_title, e.reports\_to, m.first\_name AS manager*

*FROM employees e*

*JOIN employees m ON e.reports\_to = m.employee\_id;*

*SELECT e.employee\_id, e.first\_name, e.job\_title, e.reports\_to, m.first\_name AS manager*

*FROM employees e*

*LEFT JOIN employees m ON e.reports\_to = m.employee\_id;*

Using clause:

* Matches if the two columns with the same name exist in two different tables.
* Applicable only if the names match otherwise cannot be used.

*SELECT o.order\_id, o.product\_id, quantity, name*

*FROM order\_items o*

*LEFT JOIN products p*

*USING (product\_id)*

*JOIN order\_item\_notes oin*

*USING (order\_id);*

Natural Joins:

* Joins two tables based on a common value. No need to mention separately the columns that need to be matched

Ex:

*SELECT oi.order\_id, oi.product\_id, quantity*

*FROM order\_items oi*

*NATURAL LEFT JOIN orders o;*

Cross Joins:

* Returns the records from both the tables.
* WHERE clause is not specified
* If there is a relationship between the two tables and a WHERE clause is specified, the result of the CROSS JOIN will be same as the INNER JOIN

Syntax:

*SELECT column1, column2, ...columnN*

*FROM table1*  
*CROSS JOIN table2;*

Ex:

--Explicit Cross join

*SELECT \**

*FROM products p*

*CROSS JOIN shippers sh;*

--Implicit Cross Join

*SELECT \**

*FROM products p, shippers sh;*

Union clause:

* UNION is used to combine rows
* It outputs only distinct rows
* To output all the rows, we should use UNION ALL
* Should have same number of columns selected to combine
* Column names selected in the first query will be used to display column names in the result.

Syntax:

*SELECT column\_name(s) FROM table1*  
*UNION*  
*SELECT column\_name(s) FROM table2;*

*SELECT column\_name(s) FROM table1*  
*UNION ALL*  
*SELECT column\_name(s) FROM table2;*

Ex:

*SELECT customer\_id, first\_name, points, 'Bronze' AS type*

*FROM customers*

*WHERE points < 2000*

*UNION*

*SELECT customer\_id, first\_name, points, 'Silver' AS type*

*FROM customers*

*WHERE points BETWEEN 2000 AND 3000*

*UNION*

*SELECT customer\_id, first\_name, points, 'Gold' AS type*

*FROM customers*

*WHERE points > 3000*

*ORDER BY first\_name;*

Group By Clause:

* GROUP BY is used to summarize rows with same values
* It is used along with aggregate functions (min(), max(), avg(), count(), sum())

Syntax:

*SELECT column\_name(s)*  
*FROM table\_name*

*WHERE condition*  
*GROUP BY column\_name(s)*  
*ORDER BY column\_name(s);*

*Ex:*

*SELECT count(state) AS state\_count, state*

*FROM customers*

*GROUP BY state*

*ORDER BY state\_count;*

Having Clause*:*

* *HAVING clause was added because WHERE cannot be used with aggregate functions.*

*Syntax:*

*SELECT column\_name(s)*  
*FROM table\_name*  
*GROUP BY column\_name(s)*

*HAVING condition*  
*ORDER BY column\_name(s);*

*Ex:*

*SELECT count(state) AS state\_count, state*

*FROM customers*

*GROUP BY state*

*HAVING count(state) <> 1*

*ORDER BY state\_count;*

Existsclause*:*

* *EXISTS clause is used to check whether a record exists or not in the table.*
* *It returns TRUE, if subqueries return 1 or more rows.*

*Syntax:*

*SELECT column\_name(s)*  
*FROM table\_name*  
*WHERE EXISTS*  
*(SELECT column\_name FROM table\_name WHERE condition);*

Ex:

*SELECT \* FROM customers c*

*WHERE EXISTS (SELECT \* FROM orders o WHERE c.customer\_id = o.customer\_id)*

Case clause:

* Like if...else statement.
* If no conditions match returns NULL.

Syntax:

*CASE*  
 *WHEN condition1 THEN result1*  
 *WHEN condition2 THEN result2*  
 *WHEN conditionN THEN resultN*  
 *ELSE result*  
*END;*

Ex:

*SELECT \*,*

*CASE*

*WHEN points > 3000 THEN "Gold Customer"*

*WHEN points > 1500 THEN "Silver Customer"*

*ELSE "Bronze Customer"*

*END AS Customer\_status*

*FROM customers;*

MySQL Constraints:

* Constraints are defined to make sure that the data that is to be inserted follows a certain set of rules.
* If the data does not verify the constraint set, then the insertion will be aborted.
* Primary Key, Foreign Key, Unique, NOT NULL, AUTO\_INCREMENT, CHECK, DEFAULT, Create Index
* *NOT NULL*: Ensures that the data is not inserted into the table without the value being specified to that column.
* *UNIQUE*: Ensures that the column with this constraint does not have duplicate values.
* *PRIMARY KEY*: This column uniquely identifies the table and should be NOT NULL.
* *FOREIGN KEY*: Foreign key is the Primary key of another table. Adding this to the child table ensures that there is a proper relationship between the parent and the child table, and no invalid data is inserted into the child table.
* *CHECK*: It is a condition that is to be checked on column before adding a value to it. If the condition is fulfilled the record will be inserted otherwise not.
* *DEFAULT*: It is used to set default values to column if the value is not specified while inserting the data.
* *AUTO INCREMENT*: AUTO\_INCREMENT keyword in MySQL is used to auto create values to certain columns. This is often used to define the Primary Key in a table.
* *CREATE INDEX*: Index is created to a table so that it is easier to access the table.

Ex:

*CREATE TABLE employee(*

*id int PRIMARY KEY AUTO\_INCREMENT,*

*first\_name varchar(50) NOT NULL UNIQUE,*

*last\_name varchar(50),*

*age int CHECK(age>=18),*

*city varchar(50) DEFAULT 'Bengaluru',*

*department ENUM('Management','HR','Development','Testing'),*

*customer\_id int NOT NULL,*

*FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)*

*);*

*CREATE INDEX emp\_index ON employee(id);*

*SELECT \* FROM employee USE INDEX(emp\_index);*

*SHOW INDEX IN employee;*

*ALTER TABLE employee CHANGE last\_name last\_name varchar(50) NOT NULL;*

Creating a copy of a table with data:

Syntax:

*CREATE TABLE new\_table\_name*

*SELECT \* FROM old\_table\_name;*

Ex:

*CREATE TABLE orders\_archived*

*SELECT \* FROM orders;*

Copying data from one table to an already created empty table:

Syntax:

*INSERT INTO empty\_table\_name*

*SELECT \* FROM table\_name;*

Ex:

*INSERT INTO orders\_archived*

*SELECT \* FROM orders;*

Update:

* UPDATE keyword is used to update any records in the table.
* If the WHERE clause in the UPDATE query is missed, all the records will be updated with the specified column values.
* While updating multiple rows of a table from MySQL Workbench, safe mode needs to be unchecked. This can be done as follows: Edit --> Preferences --> SQL Editor --> Uncheck “Safe Updates” and restart workbench.

Syntax:

*UPDATE table\_name*  
*SET column1 = value1, column2 = value2, ...columnN = valueN*  
*WHERE condition;*

Ex:

UPDATE customers

SET points = points + 50

WHERE birth\_date <='1990-01-01';

Delete:

* This keyword is used to delete a record or multiple records from the table.

Syntax:

*DELETE FROM table\_name WHERE condition;*

Ex:

*DELETE FROM Customers WHERE customer\_id = 3;*

Aggregate Functions:

* Min(): returns minimum value of the column name specified
* Max(): maximum value of the specified column name
* Count(): number of records with the specified column name
* Sum(): sum of all the values of the specified column name
* Avg(): average of all the values of the specified column name
* Max(), Min(), Count(): column can be numeric or string
* Sum(), Avg(): column should be numeric otherwise returns default value 0

Syntax:

*SELECT function\_name(column\_name)*

*FROM table\_name*

*WHERE condition;*

Ex:

*SELECT max(first\_name)*

*FROM customers;*

Temporary Tables:

* Temporary tables can be created to hold the result of complex queries.
* This helps reduce computing time.
* They are available only during the session and will be destroyed once the connection is closed or the session is over.
* Insertion, deletion or any other operations can be done the same as the actual table.

Syntax:

*CREATE TEMPORARY TABLE temp\_table\_name;*

Ex:

*CREATE TEMPORARY TABLE temp\_customers*

*SELECT DISTINCT c.customer\_id, c.first\_name, c.last\_name, c.birth\_date, c.state, c.points*

*FROM customers c*

*JOIN orders o ON c.customer\_id = o.customer\_id;*

*SELECT \* FROM temp\_customers;*

Creating user accounts in MySQL and GRANT/REVOKE Privileges:

* Creating a new user to access the MySQL database

*CREATE USER user\_name IDENTIFIED BY password;*

*CREATE USER 'test'@'localhost' IDENTIFIED BY 'abcd';*

* Creating multiple user accounts

*CREATE USER 'test1'@'localhost' IDENTIFIED BY 'abcde',*

*'test2'@'localhost' IDENTIFIED BY 'abcdef';*

* Creating a new user to access the MySQL database with any hostname

*CREATE USER 'test3'@'%' IDENTIFIED BY 'abcd3';*

* Creating a new user to access the MySQL database with specific hostname but any subdomain

*CREATE USER 'test4'@'%.subdomain' IDENTIFIED BY 'abcd4';*

* Shows the grants given to the user. When a new user is created, no grant is provided to the user

*SHOW GRANTS FOR user\_name;*

*SHOW GRANTS FOR test@localhost;*

* Get the users and hosts

*SELECT User, Host FROM mysql.user;*

* Deleting a user account

*DROP USER user\_name;*

*DROP USER test1@localhost;*

* Deleting multiple user accounts

*DROP USER test2@localhost, test4@.subdomain, test3@localhost;*

* Show current users

*SHOW PROCESSLIST;*

* Kill the running user process

*KILL Id\_number;*

* After the process is killed drop the user account as before

*DROP USER root@localhost;*

* Granting privileges to a user. Only SELECT privilege is granted. Cannot do any other operations

*GRANT [SELECT, UPDATE, INSERT, DELETE, INDEX, CREATE, ALTER, DROP, GRANT, ALL] ON [database\_name|table\_name] TO user\_name;*

*GRANT EXECUTE ON [FUNCTION|PROCEDURE] function\_or\_procedure\_name TO user\_name;*

*GRANT SELECT ON customers TO test@localhost;*

* Revoking privileges to a user

*REVOKE [SELECT, UPDATE, INSERT, DELETE, INDEX, CREATE, ALTER, DROP, GRANT, ALL] ON [database\_name|table\_name] FROM user\_name;*

*REVOKE EXECUTE ON [FUNCTION|PROCEDURE] function\_or\_procedure\_name FROM user\_name;*

*REVOKE SELECT ON customers FROM test@localhost;*

* Locking user accounts: locking a user account denies access to the user to make a connection to MySQL.

*CREATE USER 'test5@localhost' IDENTIFIED BY '12345' ACCOUNT LOCK;*

* Unlocking user accounts: To allow access again the account needs to be unlocked.

*ALTER USER 'test5@localhost' IDENTIFIED BY '12345' ACCOUNT UNLOCK;*

Stored Procedures:

* Stored Procedures are pre-compiled SQL statements that are called frequently.
* They can call functions, other stored procedures as well as themselves recursively.
* A stored procedure can have parameters such us IN, OUT, INOUT
* IN: used when an argument needs to be passed to the procedure
* OUT: used when a value is returned by the procedure
* INOUT: used when an argument is passed, and a value is returned by the procedure
* A stored procedure can have any number of parameters

Syntax:

*DELIMITER &&*

***CREATE******PROCEDURE*** *procedure\_name(parameters)*

***BEGIN***

*Declaration\_section*

*Executable\_section*

***END*** *&&*

*DELIMITER ;*

Ex:

*1. DELIMITER &&*

*CREATE PROCEDURE sp\_getCustomers()*

*BEGIN*

*SELECT \* FROM customers;*

*END &&*

*DELIMITER ;*

*CALL sp\_getCustomers();*

*2. DELIMITER &&*

*CREATE PROCEDURE sp\_getCustomersCount(OUT var int)*

*BEGIN*

*SELECT Count(\*) INTO var FROM customers;*

*END &&*

*DELIMITER ;*

*CALL sp\_getCustomersCount(@CusCount);*

*SELECT @CusCount;*

*3. DELIMITER &&*

*CREATE PROCEDURE sp\_getHighestCustomers(IN var int)*

*BEGIN*

*SELECT \* FROM customers ORDER BY points DESC LIMIT var;*

*END &&*

*DELIMITER ;*

*CALL sp\_getHighestCustomers(8);*

*4. DELIMITER &&*

*CREATE PROCEDURE sp\_getCustomersName(INOUT var varchar(50))*

*BEGIN*

*SELECT Count(first\_name) INTO var FROM customers WHERE birth\_date > var;*

*END &&*

*DELIMITER ;*

*SET @var1 = '1986-03-28';*

*CALL sp\_getCustomersName(@var1);*

*SELECT @var1;*

*Dropping a stored procedure:*

*DROP PROCEDURE IF EXISTS stored\_procedure\_name;*

*DROP PROCEDURE IF EXISTS sp\_getCustomersName;*

Triggers:

* Triggers are stored procedures that are called automatically when an event occurs.
* Data validation
* Chceks integrity of data
* Triggers are called automatically only to the rows that are affected by INSERT, UPDATE and DELETE (DML).
* Trigger time has two variants AFTER and BEFORE.
* AFTER: Trigger is executed after the operation on the table is performed (after data is successfully added if something needs to be done)
* BEFORE: Trigger is executed before the operation on the table us performed (data needs to be checked before adding it to the database)
* BEFORE INSERT --> Can access and change NEW values
* AFTER INSERT --> Can access NEW values but cannot change
* BEFORE UPDATE --> OLD values are accessible but cannot be changed. NEW values are accessible and can be changed.
* AFTER UPDATE --> OLD and NEW values are accessible but cannot be changed.
* BEFORE DELETE --> OLD rows are accessible but cannot be changed
* AFTER DELETE --> OLD rows are accessible but cannot be changed

Syntax:

*CREATE TRIGGER trigger\_name [AFTER|BEFORE] [INSERT|UPDATE|DELETE]*

*ON table\_name FOR EACH ROW*

*BEGIN*

*--variable declarations*

*--trigger code*

*END;*

Ex:

*DELIMITER //*

*CREATE TRIGGER trigger\_customers BEFORE INSERT*

*ON customers FOR EACH ROW*

*BEGIN*

*IF NEW.points < 0 THEN SET NEW.points = 0;*

*END IF;*

*END //*

*INSERT INTO customers (first\_name, last\_name, birth\_date, phone, address, city, state, points) VALUES("Dwayne","Johnson","1980-01-01","804-427-9456","14187 Commercial Trail",'Hampton','VA', 1765),*

*("Richard","Thompson","1985-01-01","804-427-9556","14687 Commercial Trail",'Hampton','MA', -0908);*

* Enabling and disabling triggers: MySQL does not support disabling and enabling triggers temporarily. We have to drop the triggers if we no longer want them.

Syntax:

*DROP TRIGGER IF EXISTS trigger\_name;*

Ex:

*DROP TRIGGER IF EXISTS trigger\_customers;*

Views:

* Views are virtual tables created using queries.
* They are created every time the query is executed.
* Any changes made to the underlying tables or to the view itself will be reflected in the views and vice versa.
* Any DML operations can be performed on the view.

Syntax:

*CREATE OR REPLACE VIEW view\_name AS*  
*SELECT column1, column2, ...*  
*FROM table\_name*  
*WHERE condition;*

Ex:

*CREATE VIEW customers\_view AS*

*SELECT c.customer\_id, first\_name, birth\_date, points*

*FROM customers c*

*JOIN orders o*

*USING (customer\_id);*

* Deleting a view

Ex:

*DROP VIEW IF EXISTS customers\_view;*

Functions:

* User Defined Functions in MySQL
* Can have 0 or more parameters but must have a return statement
* Can be called from a SELECT statement and from other functions or stored procedures
* Any one of the below options should be mentioned
* DETERMINISTIC/NOT DETERMINISTIC - Whether the function returns the same output for the same set of inputs or not
* READS SQL DATA/MODIFIES SQL DATA - Whether the function reads or modifies data
* NO SQL/CONTAINS SQL - Whether the function contains SQL statements or not

Syntax:

*DELIMITER $$*

*CREATE FUNCTION function\_name(parameters)*

*RETURNS return\_type*

*[DETERMINISTIC|NOT DETERMINISTIC]*

*[READS SQL DATA|MODIFIES SQL DATA]*

*[NO SQL|CONTAINS SQL]*

*BEGIN*

*DECLARE var data\_type;*

*Statements;*

*RETURN variable;*

*END $$*

*DELIMITER ;*

Ex:

*DELIMITER $$*

*CREATE FUNCTION showMessage(firstName varchar(50), lastName varchar(50))*

*RETURNS varchar(50)*

*DETERMINISTIC*

*BEGIN*

*RETURN concat(firstName, lastName);*

*END $$*

*DELIMITER ;*

* Calling a function

*SELECT showMessage("Jane", "Doe") AS Name;*

* Dropping a function

*DROP FUNCTION showMessage;*

* Show all the functions defined in the database

*SHOW FUNCTION STATUS WHERE db = 'sql\_store'*

Transactions:

* Transactions - Set of statements that will be executed and committed to database only if all the statements are successfully executed
* ACID Properties - Any transcation defined should have these properties
* Atomicity - ALL or None
* Consistancy - DB should be consisitent before and after transaction
* Isolation - Each transaction should be independent of each other
* Durability - Changes of a successful transaction should be committed even if there is a system crash

*START TRANSACTION;*

*INSERT INTO customers (customer\_id, first\_name, last\_name, birth\_date, phone, address, city, state, points)*

*VALUES(19,"Jane","Doe","1989-01-01","904-427-9456","14187 Avenue Trail",'Hampton','MA', 1895);*

*INSERT INTO orders (order\_id, customer\_id, order\_date, status, comments, shipped\_date, shipper\_id)*

*VALUES(11,19,"2023-09-29",1,DEFAULT,DEFAULT,DEFAULT);*

*SELECT \* FROM customers;*

*COMMIT;*

* ROLLBACK - If any of the statement inside the transaction ROLLBACK can be called to restore the database to the previous position
* Cannot be used on CREATE, UPDATE, ALTER, DROP database/tables
* After execution of transaction statements either COMMIT or ROLLBACK can be called

*START TRANSACTION;*

*SELECT \* FROM order\_item\_notes;*

*DROP TABLE order\_item\_notes;*

*SELECT \* FROM order\_item\_notes;*

*ROLLBACK;*

* SAVEPOINT saves the instance of the database at specified point
* ROLLBACK TO SAVEPOINT rolls back to this save point where the statements executed after the last save point are dropped
* RELEASE SAVEPOINT deletes the savepoint specified

*START TRANSACTION;*

*INSERT INTO orders (order\_id, customer\_id, order\_date, status, comments, shipped\_date, shipper\_id)*

*VALUES(12,19,"2023-09-29",1,DEFAULT,DEFAULT,DEFAULT);*

*SAVEPOINT save\_point;*

*INSERT INTO orders (order\_id, customer\_id, order\_date, status, comments, shipped\_date, shipper\_id)*

*VALUES(13,19,"2023-09-29",1,DEFAULT,DEFAULT,DEFAULT);*

*ROLLBACK TO SAVEPOINT save\_point;*

*INSERT INTO orders (order\_id, customer\_id, order\_date, status, comments, shipped\_date, shipper\_id)*

*VALUES(14,19,"2023-09-29",1,DEFAULT,DEFAULT,DEFAULT);*

*RELEASE SAVEPOINT save\_point;*

*COMMIT;*

Table locking:

* Locking and Unlocking tables
* Provides read access to the table - write operation cannot be performed

*LOCK TABLES customers READ;*

* Throws error : Error Code: 1099. Table 'customers' was locked with a READ lock and can't be updated

*INSERT INTO customers (first\_name, last\_name, birth\_date, phone, address, city, state, points) VALUES("Jane1","Doe1","1989-01-01","904-427-9456","14187 Avenue Trail",'Hampton','MA', 1895);*

* Provides write access to the table and the data to the table can be inserted only in the current session
* Table needs to be UNLOCKED for it to be accessible in other sessions

*LOCK TABLES customers WRITE;*

* Will be executed in this session. Does not execute throws table locked error in other sessions

*INSERT INTO customers (first\_name, last\_name, birth\_date, phone, address, city, state, points) VALUES("Jane1","Doe1","1989-01-01","904-427-9456","14187 Avenue Trail",'Hampton','MA', 1895);*

* removes lock from all the tables

*UNLOCK TABLES;*