DBMS - Database Management System

- Is a program that store data, retrieve data & manipulate data.
- CRUD operations
  - Create -- Insert new record.
  - Retrieve -- Select existing records.
  - Update -- Modify existing record.
  - Delete -- Delete existing record.
- Generic program for any data e.g. students, financial, employees, exams, ...
- Example: Excel/Spreadsheets, Foxpro, Dbase, ...

RDBMS - Relational DBMS

- Is a generic program that store data, retrieve data & manipulate data i.e. perform CRUD operations efficiently.
- RDBMS organize data into **Tables**, **Rows** & **Columns**. These tables are **related** to each other -- Relational DBMS.

DBMS	RDBMS
Fields, Records, File	Columns, Rows, Tables
Relations in file are program	Built-in relational feature
Heavy programming	Built-in CRUD & other operations
Heavy network use (whole file)	Less network use (only record)
Client side processing	Server side processing
Slower	Faster
Huge data is not supported(MB)	Huge data support (100s of GB)
No networking support	Client-server arch
Single-user systems	Multi-user systems
File locking	Row level locking
No distribted db support	Built-in support for clustering
No security	Db auth, object level security

Different RDBMS

- MySQL: Michal - Sun - Oracle - Open Source (Community Ed).
  - Most used open source db.
  - MySQL 5.x -- MySQL 8.0.15

- MariaDb: Fully open source (clone of MySQL).
- Oracle: Enterprise database -- Express edition (educational).
- MSSQL server: Microsoft - Enterprise database -- Only windows.
- Informix: Fastest (Need assembly language knowledge).
- Sybase: Taken by SAP.
- PostreSQL: Open Source.
- DB2: IBM - Mainframe systems.
- MSAccess, Paradox, SQLite: Small RDBMS (not all features).

SQL - Structured Query Language

- RDBMS data processing is done using SQL queries.
- Client fire query on server, server process query, produce result and send result back to client.
- ANSI standardised -- 1987, 1989, ..., 2016.
- Case-insensitive language.
    - On Linux, table names & db names are case sensitive.
- Categories:
    - DDL: Data Definition Language
        - CREATE, DROP, RENAME, ALTER
    - DML: Data Manipulation Language*
        - INSERT, UPDATE, DELETE
    - DQL: Data Query Language*
        - SELECT
    - DCL: Data Control Language
        - CREATE USER, GRANT, REVOKE
    - TCL: Transaction Control Language
        - Transaction management
        - Transaction is "set of queries executed as single unit".
        - If any query fails, effect of remaining queries is discarded.
        - START TRANSACTION, SAVEPOINT, COMMIT, ROLLBACK

Naming conventions

- Table & column names can contains alphabets, digits and some special symbols e.g. $, #, _.
- Names must start with alphabets.
- If name contains special symbol then name must be enclosed in back-quotes e.g. `T1#`
- Names can be max 30 chars long.
- Names should be readable.

MySQL - RDBMS software

MySQL Installation - Ubuntu

```
cmd> sudo apt-get install mysql-community-server mysql-community-client
```

- This installs "MySQL server" i.e. **mysqld** and MySQL client i.e. **mysql**.
- mysqld:
  - Server. No UI.
  - Implemented in C/C++.
  - Installed on RDBMS server machine.
  - Process the data & generate result.
  - Can be accessed from any MySQL client program.
    - MySQL CLI (mysql)
    - MySQL Workbench (GUI)
    - phpmyadmin (Web based)
- mysql:
  - Command line client for MySQL

```
cmd> sudo systemctl status mysql

cmd> sudo systemctl stop mysql

cmd> sudo systemctl start mysql

cmd> sudo netstat -tlnp

cmd> mysql -V
```

- During installation of mysql server, its admin user is created with name **root**. User need to specify the password.

```
cmd> mysql -u root -p

cmd> mysql -h localhost -u root -p
    # client connect to mysql server of local machine.
    # localhost can be replaced by ip addr of different server machine.
```

Creating User & Database

- It is not recommeded to use root login for creating tables & doing CRUD operations.
- Better practice is to create a separate database/schema, db user and give permission to the user on that db.
- All db objects i.e. tables, constraints, relations, procedures, triggers, functions, etc should be created under some database/schema.
- Usually one project's all tables & other objects are kept in one database/schema.

```
cmd> mysql -u root -p
```

```
CREATE DATABASE dacdb;

SHOW DATABASES;

CREATE USER dac@localhost IDENTIFIED BY 'dac';

SELECT user, host FROM mysql.user;

GRANT ALL PRIVILEGES ON dacdb.* TO dac@localhost;

FLUSH PRIVILEGES;
-- activate new permissions

EXIT;
```

```
cmd> mysql -u dac -p
```

```
SHOW DATABASES;

SELECT USER(), DATABASE();

USE dacdb;

EXIT;
```

Using MySQL

```
USE dacdb;

CREATE TABLE contacts (id INT, name VARCHAR(40), mobile VARCHAR(12), email
VARCHAR(40), age INT);

SHOW TABLES;

DESCRIBE contacts;

INSERT INTO contacts VALUES (1, 'Nilesh Ghule', '9527331338',
'nilesh@sunbeaminfo.com', 35);

INSERT INTO contacts VALUES (2, 'Nitin Kudale', '9881208115',
'nitin@sunbeaminfo.com', 40), (3, 'Prashant Lad', '9881208114',
'prashant@sunbeaminfo.com', 40);

SELECT * FROM contacts;
```

```
INSERT INTO contacts (id, name) VALUES (5, 'Rahul Kale');

INSERT INTO contacts VALUES (5, 'Rahul Sansudi', NULL, NULL, NULL);

SELECT * FROM contacts;

SELECT id, name, mobile FROM contacts;
```

SELECT query (DQL)

```
SHOW TABLES;

SELECT * FROM BOOKS;

SELECT name, subject, price FROM BOOKS;

SELECT name, subject, price, price * 0.05 FROM BOOKS;

DESC BOOKS;
```

Computed Columns

```
SELECT name, subject, price, price * 0.05 AS gst FROM BOOKS;
-- AS keyword is optional

SELECT name, price, price * 0.05 AS gst, price + price * 0.05 total FROM BOOKS;
-- gst & total are computed columns.
```

LIMIT clause

```
SELECT * FROM BOOKS;

SELECT * FROM BOOKS LIMIT 5;
-- fetch first 5 rows

SELECT * FROM BOOKS LIMIT 4, 5;
-- skip first 4 rows and fetch next 5 rows
```

- In SQL query, strings & dates/times must in single quotes.

Database layout

Logical layout

- How data is represenated?
- "database" is like a namespace/container that stores all db objects related to a project.
- It contains tables, costraints, relations, stored procedures, functions, triggers, ...
- There are some system databases e.g. mysql, performance_schema, information_schema, sys, ... They contains db internal info.
    - SELECT user, host FROM mysql.user;
- The database contains tables.
- Tables have multiple columns. Each column is associated with a data-type & zero/more constaints.
- The data in table is in multiple rows. Each row have multiple values (as per columns).

Physical layout

- How data is stored on db server hard disk?
- In MySQL, the data is stored on disk in its **data directory** i.e. /var/lib/mysql
- Each database/schema is a separate sub-directory in data dir.
- Each table in the db, is a file on disk e.g. BOOKS table in classwork db is stored in file /var/lib/mysql/classwork/BOOKS.ibd.
- Note that data is stored in binary format.
- A file is not contiguously stored on hard disk & hence all data rows are not contiguous. They are scattered in the hard disk.
- In one row, all fields are consecutive.
- When records are selected, they are selected in any order. Hence we cannot fetch first n rows.

INSERT query - DML

```
USE classwork;

DESC BOOKS;

INSERT INTO BOOKS VALUES (9001, 'Atlas Shrugged', 'Ayn Rand', 'Novell', 456.562);

INSERT INTO BOOKS VALUES (9002, 'The Fountainhead', 'Ayn Rand', 'Novell',
423.123), (9003, 'The Alchemist', 'Paulo Coelo', 'Novell', 321.345);

CREATE TABLE BOOKS2 (id INT, name VARCHAR(50), author VARCHAR(50), subject
VARCHAR(50), price DOUBLE);

INSERT INTO BOOKS2 SELECT * FROM BOOKS;

SELECT * FROM BOOKS2;
```

ORDER BY clause

- In db rows are scattered on disk. Hence not fetched in a fixed order.

```
SELECT * FROM EMP;

SELECT * FROM EMP ORDER BY sal;

SELECT * FROM EMP ORDER BY sal DESC;

SELECT * FROM EMP ORDER BY job ASC;

SELECT * FROM EMP ORDER BY deptno;

SELECT * FROM EMP ORDER BY deptno, sal DESC;

SELECT * FROM EMP ORDER BY deptno DESC, job, sal DESC;
```

- When query is fired from client:
    1. Db server will load the data from disk into its RAM.
    2. Using sorting algo data will be sorted there (on given col).
    3. Processed result is sent back to the client.
- More are columns to sort, more is the time taken for sorting.

```
SELECT * FROM EMP
ORDER BY sal DESC
LIMIT 1;
-- highest sal

SELECT * FROM EMP
ORDER BY sal ASC
LIMIT 2,1;
-- third lowest sal
```

Criteria - WHERE clause

Relational operators

- <, >, <=, >=, =, != or <>

```
-- SELECT cols FROM table WHERE condition;

SELECT * FROM EMP WHERE empno = 7900;

SELECT * FROM EMP WHERE sal > 2500;

SELECT ename, sal, comm FROM EMP WHERE comm > 0.0;
```

```
SELECT * FROM EMP WHERE comm = NULL; -- won't work
```

NULL related operators

- NULL values cannot be compared using above relational operators.
- There are special operators for comparting NULL values:
    - IS NULL or <=>
    - IS NOT NULL

```
SELECT * FROM EMP WHERE comm IS NULL;

SELECT * FROM EMP WHERE comm <=> NULL;

SELECT * FROM EMP WHERE comm IS NOT NULL;
```

Logical Operators

- AND, OR & NOT

```
SELECT * FROM EMP WHERE job = 'SALESMAN' AND sal <= 1500;
-- find all salesman whose sal is below 1500

SELECT * FROM EMP WHERE job = 'SALESMAN' AND comm IS NULL;
-- find all salesman who are not getting comm.
-- there are no such records

SELECT * FROM EMP WHERE sal >= 1000.00 AND sal <= 2000.00;

SELECT * FROM EMP WHERE job = 'SALESMAN' OR sal >= 2500;
-- find all emps who are either salesman or have sal >= 2500.

SELECT * FROM EMP WHERE empno = 7900 OR empno = 7788;

SELECT * FROM EMP WHERE job != 'SALESMAN' AND job != 'ANALYST';
-- find all emps who are neither salesman nor analyst.

SELECT * FROM EMP WHERE NOT (job = 'SALESMAN' OR job = 'ANALYST');
-- find all emps who are neither salesman nor analyst.
```

BETWEEN operator

```
SELECT * FROM EMP WHERE sal BETWEEN 1100 AND 1600;
-- find employee with sal greater than or equal 1100 & less than or equal 1600.
-- BETWEEN includes both ends of given range.
```

```
SELECT * FROM EMP WHERE hire BETWEEN '1982-01-01' AND '1982-12-31';
-- find employee joined in year 1982.
-- dates are in format 'yyyy-mm-dd'.

SELECT * FROM EMP ORDER BY ename;

SELECT * FROM EMP WHERE ename BETWEEN 'B' AND 'F';
-- find employees whose name start from B to F.
-- FORD is excluded (due to alphabetical order).

SELECT * FROM EMP WHERE ename BETWEEN 'B' AND 'G';
SELECT * FROM EMP WHERE ename BETWEEN 'B' AND 'G' AND ename != 'G';
```

IN operator

```
SELECT * FROM EMP WHERE empno = 7900 OR empno = 7782 OR empno = 7839;
-- find employees whose empno is 7900, 7782 or 7839.

SELECT * FROM EMP WHERE empno IN (7900, 7782, 7839);
-- find employees whose empno is 7900, 7782 or 7839.

SELECT * FROM EMP WHERE job IN ('SALESMAN', 'CLERK', 'PRESIDENT');
-- find all salesman, clerk & president.

SELECT * FROM EMP WHERE deptno NOT IN (10, 20);
-- find all emps not in dept 10 & 20.
```

WHERE clause execution

- The condition in WHERE clause is checked for each row in table.
- Example:

```
SELECT * FROM EMP WHERE deptno = 30 AND sal > 1500;
```

LIKE operator

- % is any number of any characters.
- _ is any one character.

```
SELECT * FROM EMP WHERE ename LIKE 'S%';
-- find emps whose name start with "S"

SELECT * FROM EMP WHERE ename LIKE '%ER';
-- find emps whose name end with "ER"
```

```
SELECT * FROM EMP WHERE ename LIKE '%U%';
-- find emps whose name contains "U".

SELECT * FROM EMP WHERE ename LIKE '____';
-- find emps whose names are of 4 chars.

SELECT * FROM EMP WHERE ename LIKE '%O__';
-- find emps whose name contains "O" & then two chars.
```

CASE-WHEN statement

- Used to generate a computed column in SELECT.

```
SELECT ename, deptno FROM EMP;

SELECT deptno, dname FROM DEPT;

-- 10 - ACCOUNTING, 20 - RESEARCH, 30 - SALES, 40 - OPERATIONS
SELECT ename, deptno,
CASE
WHEN deptno = 10 THEN 'ACCOUNTING'
WHEN deptno = 20 THEN 'RESEARCH'
WHEN deptno = 30 THEN 'SALES'
ELSE 'OPERATIONS'
END
FROM EMP;

SELECT ename, deptno,
CASE
WHEN deptno = 10 THEN 'ACCOUNTING'
WHEN deptno = 20 THEN 'RESEARCH'
WHEN deptno = 30 THEN 'SALES'
ELSE 'OPERATIONS'
END AS dname
FROM EMP;
```

UPDATE query - DML

- To change one or more rows in a table.

```
SELECT * FROM BOOKS;

-- syntax:
-- UPDATE table SET col1=value1, ... WHERE condition;

UPDATE BOOKS SET price=150.123 WHERE id = 1004;
```

```
-- change price of book 1004 to 150.123.

UPDATE BOOKS SET price=160.123, author='K & R', name='C Programming Language'
WHERE id = 1004;
-- change price of book 1004 to 160.123, author to "K & R" and name to "C
Programming Language".

UPDATE BOOKS SET price=price+price*0.10 WHERE subject='Java Programming';
-- increase price of all java books by 10%.

UPDATE BOOKS SET price=150.123;
-- change all books price to 150.123;

UPDATE BOOKS SET id=1005, name='C Lang', author='Ritchi', subject='C', price=123
WHERE id = 1004;
-- change all columns -- not used commonly.
```

DELETE query - DML

- One or more rows of a table are deleted.

```
DELETE FROM BOOKS WHERE id=1005;
-- delete one row

DELETE FROM BOOKS WHERE subject='Java Programming';
-- delete multiple rows

DELETE FROM BOOKS;
-- delete all rows
```

TRUNCATE query -- DDL

```
TRUNCATE TABLE BOOKS2;
```

DELETE	TRUNCATE
Table structure is not changed	Table structure is not changed
Delete all rows if no condn	Delete all rows
Can have WHERE clause	No WHERE clause
DML query	DDL query
Can be rollbacked (using tx)	Cannot be rollbacked

DELETE	TRUNCATE
File size is not change	File size is shrinked
Slower	Faster

DROP query -- DDL

```
DROP TABLE BOOKS3;

DROP TABLE BOOKS7; -- error if table not present

DROP TABLE IF EXISTS BOOKS7;
```

DROP	TRUNCATE
Table struct + data is deleted	Only data is deleted
DDL Query	DDL Query
Table file is deleted.	Table file size is shrinked

Data Types

- Various data types are supported in MySQL.
- Different db support different data types (they may be similar, but not same).
- Categories:
    - Numeric types
        - Integer types
            - TINYINT (1 byte)
            - SMALLINT (2 bytes)
            - MEDIUMINT (3 bytes)
            - INT (4 bytes)
            - BIGINT (8 bytes)
            - integer types can signed (default) or unsigned.
            - BIT -- number of bits can be given.
        - Floating point types
            - Approx precision
                - FLOAT (4 bytes) - single precision
                - DOUBLE (8 bytes) - double precision
            - Exact precision
                - DECIMAL - size depend on given precision
                    - (m,n): m digits & n digits after decimal pt
    - Date/Time types

- String types
  - Char-wise storage.
    - Each char can be 1 bytes (ASCII), 2 bytes (UNICODE) & so on ... depending on char encoding.
  - CHAR(n) - n can be max 255.
    - Fixed length.
    - If given small string, rest of space is not used.
    - If given bigger string, gives error.
    - Very fast access.
  - VARCHAR(n) - n can be max 65535 (64K).
    - Variable length.
    - Internally stores chars + length.
    - Depending on entered string, space is allocated.
    - If given small string, will take less space.
    - If given bigger string, gives error.
  - TEXT
    - CHAR & VARCHAR are stored inline to the row.
    - TEXT are stored outside the record (its addr is in record).
    - Used for huge text data.
    - Very slow.
    - Different sizes:
      - TINYTEXT (max 255 chars)
      - TEXT (max 64K chars)
      - MEDIUMTEXT (max 16M chars)
      - LONGTEXT (max 4G chars)
- Binary types
  - CHAR -- BINARY
  - VARCHAR -- VARBINARY
  - BLOB -- Similar to TEXT types
    - Stored external to record.
    - Byte-wise storage.
  - Used for binary files e.g. images, audio, video, pdf, ...
  - Using binary types increase db size quickly & slow down processing. Hence usually not recommended.
    - Different sizes:
      - TINYBLOB (max 255 bytes)
      - BLOB (max 64K bytes)
      - MEDIUMBLOB (max 16M bytes)
      - LONGBLOB (max 4G bytes)
- Misc types
  - BOOL - Like TINYINT i.e. 1 byte.
    - TRUE - 1
    - FALSE - 0
  - ENUM
  - SET

```
CREATE TABLE t4 (c1 CHAR(10), c2 VARCHAR(10), c3 TEXT(10));

INSERT INTO t4 VALUES ('abc', 'abc', 'abc');

INSERT INTO t4 VALUES ('abcdefghij', 'abcdefghij', 'abcdefghij');

INSERT INTO t4 VALUES ('abcdefghijk', 'abcdefghijk', 'abcdefghijk'); -- error

INSERT INTO t4 VALUES ('abcdefghij', 'abcdefghij', 'abcdefghijk');
```

```
- Date/Time types
    - DATE - 3 bytes - num of days from 1-1-1000.
    - TIME - 3 bytes - timespan.
    - DATETIME - 8 bytes - store both date & time
    - TIMESTAMP - 4 bytes - num of secs from 1-1-1970
        - auto updated on DML operation.
    - YEAR - 1 byte - year 1901 to 2155
```

```
CREATE TABLE t5 (name VARCHAR(20), birth DATETIME, modified TIMESTAMP);

DESCRIBE t5;

INSERT INTO t5 (name, birth) VALUES ('Nilesh', '1983-09-28');
INSERT INTO t5 (name, birth) VALUES ('Vishal', '1980-01-01');

SELECT * FROM t5;

INSERT INTO t5 (name, birth, modified) VALUES ('Rahul', '1980-12-31', NOW());
```

DUAL table

- A dummy/in-memory a table having single row & single column.
- It is used for arbitary calculations, testing functions, ...

```
SELECT NOW() FROM DUAL;

SELECT 2 + 3 * 4 FROM DUAL;

SELECT USER(), DATABASE() FROM DUAL;
```

- Historically this table is first time is used by Oracle. At that time they develop table with two rows & hence name given was DUAL. Later on table changed to single row, but name kept same.
- This keyword is accepted by ANSI. But lot of db (including MySQL) keep it optional.

```
SELECT NOW();

SELECT 2 + 3 * 4;

SELECT USER(), DATABASE();
```

Help

- MySQL client provide help on commands & functions.

```
-- syntax: HELP topic;

HELP INSERT;

HELP SELECT;

HELP Functions;

HELP String Functions;
```

Assignments

1. Install mysql server in a Ubuntu VM. Create a user and some database. Access that server from host machine's MySQL client.
2. Execute the queries in the above notes.