SQL

Database : Collection of data in a format that can be easily accessed (Digital).

DBMS : Software application used to manage DB.

user → DBMS → DB

Relational (RDB)	Non-Relational
• data stored in table	• data not stored in tables.
e.g. MySQL, oracle	e.g. mongoDB

** we use SQL to work with RDBMS

SQL : (Structured Query Language)

→ Programming Language used to interact with RDBMS.

→ Used to perform CRUD operations.

Create Read update Delete

Creating our first Database

```
CREATE DATABASE db_name;
DROP DATABASE db_name;      (Deletion)
USE db_name;
```

Creating our First Table

```
USE db_name;
CREATE TABLE table_name (
  column_name1 datatype constraint,
  column_name2 datatype constraint
  );
```

e.g.
```
CREATE DATABASE college;
USE college;
CREATE TABLE student (        → table schema (design)
  id INT PRIMARY KEY,         → not null, unique
  name VARCHAR(50),           → columns
  age INT NOT NULL
  );
```

DATATYPE	Description	Usage
CHAR	string (0-255), can store char. of fixed length	CHAR(50)
VARCHAR	string (0-255), can store char. of upto given length.	VARCHAR(50)
BLOB	str (0-65535), store binary large object	BLOB(1000)
INT	integer (-2,147,483,648 to +ve)	INT
TINYINT	integer (-128 to 127)	TINYINT
BIGINT	integer (-9,223,372,036... to +ve)	BIGINT
BIT	x-bit values (1-64)	BIT(2)
FLOAT	Decimal no. with 23 digits precision	FLOAT
DOUBLE	Decimal no. with 24-53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN

Datatypes

DATE	YYYY-MM-DD format (1000-01-01 to 9999-12-31)	DATE
YEAR	years in 4 digit (1901 - 2155)	YEAR

CHAR(10) → P U N E

VARCHAR(10) → P U N E

Signed & Unsigned Datatypes

(+ve & -ve) (+ve)

e.g. TINYINT UNSIGNED (0 to 255)

TINYINT (-128 to 127) → Range increases

To add column

```
INSERT INTO Tb_name VALUES (1, "abc", 26);
```

Types of SQL Commands

DDL (Data Definition Lang)	Create, alter, rename, truncate & drop
DQL (Data Query Lang)	select
DML (Data Manipulation Lang)	insert, update & delete
DCL (Data Control Lang)	grant & revoke permission to users
TCL (Transaction Control Lang)	start transaction, commit, rollback

```
CREATE DATABASE IF NOT EXISTS db_name;
DROP DATABASE IF NOT EXISTS db_name;
SHOW DATABASES;
SHOW TABLES;
SELECT * FROM table_name;      → select & view all columns
```

Practice Question :

Qu. Create a database for your company named XYZ.
step1: Create a table inside this DB to store employee info (id, name & salary). step2: Add following info in the DB:
1, "adam", 25000
2, "bob", 30000
3, "casey", 40000
step3: select & view all your table data.

Soln.
```
CREATE DATABASE xyz_company;
USE xyz_company;
CREATE TABLE Employee (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  salary INT );
INSERT INTO Employee (id, name, salary)
  VALUES (1, "Adam", 25000),
         (2, "Bob", 30000),
         (3, "Casey", 40000);
SELECT * FROM Employee;
```

Keys :

Primary key : A column (or set of columns) in a table that uniquely identifies each row.
→ A unique ID
→ only 1 PK & cannot be null.

Foreign key : A column (or set of columns) in a table that refers to the PK in another table.
→ FKs can have duplicate & null values.
→ multiple FKs.

Constraints : specific rules for data in a table.

NOT NULL	columns cannot have a null value
UNIQUE	all values in column are different
PRIMARY KEY	makes a column unique & not null but used only for one.
FOREIGN KEY	Prevent actions that would destroy links b/w tables.
DEFAULT	sets the default value of a column
CHECK	it can limit the values allowed in a col.

Example :

```
CREATE TABLE Departments(
 dept_id INT PRIMARY KEY,
 dept_name VARCHAR(50)
 );
CREATE TABLE Employees(
  emp_id INT PRIMARY KEY,
  name VARCHAR(50) NOT NULL,
  age INT CHECK (age>=18),
  dept_id INT,
  status VARCHAR(20) DEFAULT 'Active',
  FOREIGN KEY(dept_id) REFERENCES Departments
                              (dept_id)
  );
```

SELECT : used to select any data from the DB.

Basic Syntax - SELECT col1,col2 FROM tb_name;

To select ALL - SELECT * FROM tb_name;

To show Distinct values - SELECT DISTINCT col_name FROM tb_name;

** SELECT is used to show the table data.

WHERE clause : To define some conditions.

SELECT col1, col2 FROM tb_name WHERE conditions;

e.g.
SELECT * FROM student WHERE marks >80;
SELECT * FROM student WHERE marks>80 AND city="Mumbai";

Operators in WHERE

Arithmetic operators	+, -, *, /, %
Comparison operators	=, !=, >, >=, <, <=
Logical operators	AND, OR, NOT, IN, BETWEEN, ALL, LIKE, ANY
Bitwise operators	&, | (Bitwise AND, OR)

AND (to check for both conditions to be true).
SELECT * FROM student WHERE marks >80 AND city = "Mumbai";

OR (to check for one of the conditions to be true).
SELECT * FROM student WHERE marks >80 OR city="Pune";

BETWEEN (selects from a given range)
SELECT * FROM student WHERE marks BETWEEN 80 AND 90;

In (matches any value in the list)
SELECT * FROM student WHERE city IN ("Delhi", "Mumbai");

NOT (to negate the given condition)
SELECT * FROM student WHERE city NOT IN ("Delhi", Mumbai");

Limit clause :
sets an upper limit on number of(tuples) rows to be returned.

SELECT * FROM student LIMIT 3;
(we want only ~~and~~ 3 students' data)

Order By clause
To sort in ascending (ASC) or descending (DESC) order.

SELECT col1, col2 FROM tb_name ORDER BY col_name(s) ASC;

e.g. SELECT * FROM student ORDER BY city ASC;
o/p - Delhi → mumbai - Pune.

Aggregate Functions : perform a calculation on a set of values, & return a single value.

COUNT() → To count values
MAX() → To return max value
MIN() → To return min value
SUM() → returns the sum
AVG() → returns avg

Syntax :

```
SELECT COUNT(marks) FROM student;
SELECT MAX(marks) FROM student WHERE
                        city="Pune";
```