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";
```

Group By clause : Group rows that have the same values into summary rows.

→ It collects data from multiple records & groups the result by one or more columns.

e.g. 1) Pune
2) Delhi
3) Mumbai
} city

** Generally we use group by with some aggregation fn.

e.g. count no. of students in each city

Sol: SELECT city, COUNT(name) FROM student
GROUP BY city;

o/p →

city	count
Delhi	3
Pune	2

Practice Question :
Write the query to find avg marks in each city in asc. order.

Sol : SELECT city, AVG(marks) FROM student
GROUP BY city ORDER BY Avg(Marks) DESC;

** By default it is ascending for desc we add

101	Amit	78	C	Pune
102	Bhumi	93	A	Mumbai
103	chetan	85	B	Delhi
104	dhruv	96	A	Mumbai

Group by Grade

SELECT Grade, COUNT(Rno)
FROM student
GROUP BY Grade ;
ORDER BY Grade ;

HAVING clause : similar to where i.e. applies some condition on rows.

→ Used when we want to apply some any condition after grouping.

Having → Groups

e.g. count number of students in each city where max marks cross 90.

SELECT count(name), city FROM student
GROUP BY city HAVING Max(marks) >90;

GENERAL ORDER :

where → rows
Having → groups

SELECT column(s)
FROM tb_name
WHERE condition
GROUP BY column(s)
HAVING condition
ORDER BY column(s) ASC;

e.g. SELECT city
FROM student
WHERE grade="A"
GROUP BY city
HAVING Max(marks)>93;

o/p → Mumbai

Table related Queries

Update (to update existing rows)

UPDATE tb_name SET col1=val1, col2 = val2
WHERE condition ;

e.g. UPDATE student SET grade="O" WHERE grade="A";

safe mode is on which prevent us from making major changes. To off it :→ SET SQL_SAFE_UPDATES=0;
To on it :→ ∅ 1

SET marks = marks+1;
↓
update all marks by 1.

WHERE marks BETWEEN 80 & 90;
or
any condition

Delete (to delete existing rows)

DELETE FROM tb_name
WHERE condition.

e.g. DELETE FROM student WHERE marks <33;

** DELETE FROM student only this query can delete tb.

Foreign key :

Depat.

Id	Name
101	Science
102	English
103	Hindi

Parent table

FK

Teacher child table

Id	Name	Dept_Id
101	Adam	101
102	Bob	103
103	Casey	102
104	Donald	102

FOREIGN KEY(Dept_Id)
REFERENCE Depat

CREATE TABLE Depat(Id INT PRIMARY KEY, Name VARCHAR(50)
);

CREATE TABLE Teacher (Id INT PRIMARY KEY,
name VARCHAR(50), Dept_Id& INT,
FOREIGN KEY (dept_Id) RERERENCES Dept(Id));

cascading for FK :

On Delete Cascade :
when we create a FK using this option, it deletes the referencing rows in the child table when the referenced row is deleted in the parent table which has a PK.

On Update Cascade :
when we create a FK using UPDATE CASCADE, the referencing rows are updated in the child table when the referenced row is updated in the parent table which has a primary key (PK).

e.g. CREATE TABLE Student(
id INT PRIMARY KEY, courseID INT,
FOREIGN KEY(courseID) REFERENCES course(id)
ON DELETE CASCADE
ON UPDATE CASCADE

Alter (to change the schema)

→ design (cols, datatypes, constraint)

ADD column
ALTER TABLE tb_name
ADD column_name datatype constraint;

DROP column
ALTER TABLE tb_name
DROP COLUMN column_name;

RENAME Table
ALTER TABLE tb_name
RENAME TO new_tb_name;

CHANGE column (rename)
ALTER TABLE tb_name
CHANGE COLUMN old_name new_name new_datatyp
new constraint;

MODIFY column (modify datatype/constraint)
ALTER TABLE tb_name
MODIFY col_name new_datype new_constraint;

Truncate (to delete table's data).

```
TRUNCATE TABLE tb_name;
```

DROP → delete table

Truncate → del table's data (table still exists, we can add more data)

Practice Question: In a student table:
(a) change the name of column "name" to "full_name".
(b) Delete all the students who scored marks < 80.
(c) Delete the column for grades.

```
(a) ALTER TABLE student
    CHANGE name full_name VARCHAR (50);
(b) DELETE FROM student
    WHERE marks < 80;
(c) ALTER TABLE student
    DROP COLUMN grade;
```

JOINS IN SQL

used to combine rows from two or more tables, based on a related column between them.

Employee

id	name
101	
102	

Salary

id	salary
102	
103	

Types of Joins

inner join | Left Join | Right Join | Full Join

outer Join

Inner Join: Returns records that have matching values in both tables.

```
SELECT column(s) FROM tb_A
INNER JOIN tb_B
ON tb_A.col_name = tab_B.col_name;
```

* alias → alternate name
SELECT * FROM student as s;

e.g. student

st_id	name
101	adam
102	bob
103	casey

course

st_id	course
102	English
105	math
103	science
107	CS

```
SELECT * FROM student INNER JOIN course
ON student.st_id = course.st_id;
```

Result:

st_id	name	course
102	bob	english
103	casey	science

Left Join: Returns all records from the left table, & the matched records from right table.

```
SELECT * FROM student as s
LEFT JOIN course as c
ON s.st_id = c.st_id;
```

st_id	name	course
101	adam	null
102	bob	english
103	casey	science

Right Join: Returns all records from the right table & the matched records from left table.

```
SELECT * FROM student as s
RIGHT JOIN course as c
ON s.st_id = c.st_id;
```

st_id	course	name
102	English	bob
105	math	null
103	science	casey
107	CS	null

Full Join: Returns all records when there is a match in either left or right table.

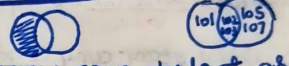
{ Left Join, Union, Right Join

```
SELECT * FROM student as a
LEFT JOIN course as b
ON a.id = b.id
UNION
SELECT * FROM student as a
RIGHT JOIN course as b
ON a.id = b.id
```

st_id	name	course
101	adam	null
102	bob	english
103	casey	science
105	null	math
107	null	CS

Some additional Joins

Left Exclusive Join



Right Exclusive Join

```
SELECT * FROR student as a
LEFT JOIN course as b ON a.id=b.id
WHERE b.id IS NULL;
```

SELF Join (regular Join but table is joined with itself).

```
SELECT col(s)
FROM table as a
JOIN table as b
ON a.col_name = b.col_name;
```

casey → adam → donald → bob

e.g. Employee

id	name	manager_id
101	adam	103 (casey)
102	bob	104 (donald)
103	casey	null
104	donald	103 (casey)

```
SELECT a.name as manager_name,
b.name
FROM employee as a
JOIN employee as b
ON a.id = b.manager_id;
```

UNION :- used to combine the result-set of two or more SELECT statements. → Gives UNIQUE records.
→ every SELECT should have same no. of cols.
→ cols must have similar datatypes.
→ cols in every SELECT should be in same order.

```
SELECT col(s) FROM tbA
UNION
SELECT col(s) FROM tb.B
```

UNION ALL → gives duplicate values also

SQL subqueries : A subquery or Inner query or a Nested query is a query within another SQL query.
→ It involves 2 select statements.

query
sub query

```
SELECT column(s)
FROM tb_name
WHERE col_name operator
(subquery);
```

1) select
2) FROM
3) where

Roll no.	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

Example :
Get name of all students who scored more than class average.
step1 : Find avg of class
step2 : Find the names of students with marks > avg.

```
SELECT name, marks
FROM student
WHERE marks > (SELECT AVG(marks) FROM student);
```

dynamic

Ques. Find the names of all students with even roll numbers.

```
SELECT roll_no FROM student ;- step-1
WHERE roll no. % 2 = 0;
SELECT name FROM student WHERE roll no IN
(102, 104, 106); → step-2
```

OR using subquery

```
SELECT name, roll no FROM student
WHERE roll no IN (SELECT rollno FROM student
                  WHERE roll no. % 2=0) ;
```

Ques. Example with FROM
Find the max marks from the students of Delhi.
step1 : Find students from Delhi
step2 : Find their max marks using sublist in step1

+ city
Pune
Mum
Mum
Delhi
Delhi
Delhi

```
SELECT MAX (marks)
FROM (SELECT * FROM student WHERE
      city = "Delhi") AS temp;
```

OR

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

---

more e.g.

```
SELECT (SELECT Max(marks) FROM student), name
FROM student;
```

MYSQL views : It is a virtual table based on the result-set of an SQL statement.

table → real
views → virtual

```
CREATE VIEW view1 AS
SELECT roll no, name FROM student;
SELECT * FROM view1;
```

real → real data → operations

** A view always shows up-to-date data. The db engine recreates the view, every time a user queries it.

```
DROP VIEW view1; (doesn't affect actual data).
```