→ Database is a collection of data in a format that can be easily accessed. allows user to efficiently store, retrieve and manage data

There are different types of databases

Relational - mysql, posgreSQL

NOSQL - mongoDB

→ A software application used to manage our DB is called DBMS

→ DBMS is a software that interacts with users, applications and the database itself to capture and analyze data. Allows user to define, create, maintain and control access to database

user → DBMS → Database
(SQL to interact with DBMS)

→ types of databases

→ RDMS

→ mysql, posgresql, oracle, SQLserver

→ data stored in tables (rows and columns)

with relationships defined between tables

→ use case: complex queries, enterprise system

financial applications

→ NOSQL databases

→ MongoDB (JSON)

→ Redis (key-value)

→ Cassandra (columns)

→ Neo4j (nodes and relationships)

→ hierarchal databases (IMS)

→ Network databases (IDS)

→ object oriented databases (objectDB)

→ time-series databases (InfluxDB)

→ columnar databases (Apache HBase)

→ graph database (Neo4j)

→ SQL, Structed Query language (IBM created)

→ interact with relational databases

→ perfoms CRUD operation

create, read, update, delete

→ Before it was named SEQUEL

→ used for managing and manipulating relational databases

→ DQL (query / retrieve, select)

→ DDLC define / modify, create alter, drop, truncate) rename

→ DML (manipulate, insert, update, delete)

→ DCLC control access, Grant, Revoke) & TCL (commit, roll back)

→ Interrelated tables together are stored in a database

→ colums (structure / schema)

→ rows (Individual data)

creating database

```
CREATE DATABASE db_name;
```

drop database (delete, remove)

```
DROP DATABASE db_name;
```

creating first table

```
CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
```

age INT NOT NULL

);

→ varchar is the data type in general, varchar2 is specific to oracle and ignores the trailing spaces

Add data to table

INSERT INTO (1, "AMNN", 26);

SQL Datatypes

They define the type of values that can be stored in a column

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

→ char and varchar, char- fixed length varchar- variable length

left (char)

(varchar)

→ for more string lengths, we use blob(0-65535)

→ signed, unsigned

→ if an column is sure that we don't use negatives we give unsigned int, to increase it's range

CREATE DATABASE IFNOTEXISTS (throws warning)

DROP DATABASE IFNOTEXISTS

SHOW DATABASES (inside server)

SHOW TABLES (only of db being used)

USE db_name

→ create table helps u define the table schema (design)

Select and view all columns

SELECT * FROM table_name;

Insert data into tables

INSERT INTO table_name

(colname 1, colname 2)

VALUES

(dA1, dA2), (dB1, dB2);

keep in mind the order of insertion

→ float datatype can introduce rounding errors, not suitable where exact precision is required, so use DECIMAL(10, 2) → total digits 10, (including decimal) right to the decimal 2

Keys

Primary key

A column(s) that uniquely determines each row, uniquely (a unique id)

→ only 1 PK and should not be null

→ A collection of colums are also primary key for ex:- student id & course id should be unique for course enrollments

foreign key

A column (set of column) in a table that refers to the PK of another table

ex:- student Id, cityId(city)

city id (FK) in student & cityid(PK) in city

→ there can be duplicates and multiple FK and

can be null

constraints

→ UNIQUE, col2 INT UNIQUE

→ col1 INT NOTNULL

→ primary key (id, roll no)

→ cust_id INT,

FOREIGN KEY (cust_id) references customer(id)

→ default (sets default value for column)

Salary INT DEFAULT 25000;

→ check (can limit values added in column)

CONSTRAINT age_check CHECK (age >=18 AND

city = "Delhi")

while defining age INT CHECK (age >=18)

select

SELECT col1, col2 FROM table_name;

→ reserved keyword (cannot be used for table

or column name)

→ * is the wild card character

DISTINCT

→ This keyword is used in a select statement to eliminate duplicates rows from the result

→ not efficient for large datasets, can use filtering instead

→ Treats null values as equal and returns one null value

where

```
SELECT col, col2 FROM table_name
WHERE marks >80 AND CITY = "mumbai".
```

Arithmetic operators: +, −, *, /, %

comparison operators: =, !=, >, >=, <, <=

logical operators: AND, OR, NOT, IN, BETWEEN, ALL, LIKE, ANY

Bitwise operators: &, |

AND — Both should be true

OR - Any of the two should be true

BETWEEN - want between any range (INCLUSIVE)

use >,< for exclusive or NOT BETWEEN

IN - matches any value in the list

SELECT * FROM student WHERE city IN ("delhi", "mumbai");

→ considered IN over OR for better performance

NOT - Negate the condition

LIMIT

Sets an upper limit on number of (tuples) rows to be returned

SELECT * FROM student LIMIT 3;

SELECT col1, col2 FROM STUDENT LIMIT 3;

SELECT * FROM student WHERE marks > 75 LIMIT 3;

OFFSET

SELECT * FROM student LIMIT 5, 10;

→ This query skips the first 5 rows and then returns the next 10 rows (row 6 to 15)

→ limit is effective in performance but when combined with a large offset can impact

performace because MySQL still has to scan and skip preceeding rows

→ can use index pagination for better performance in the above case

ORDER BY CLAUSE

TO sort in ascending (asc) or descending order (DESC)

```
SELECT * FROM student ORDER by city ASC;
```

To sort the results of a query based on one or more columns, either in ascending or descending order

→ No order is specified, deafuat will be ASC

```
SELECT * FROM STUDENTS ORDER by class, score DESC
```

→ first sorts the rows by class column in ascending order and then within each class, sorts the score

→ Sql considers NULL as the smallest value, it returns null along with asc/desc order

→ can use index for better performance on larger data sets

Aggregate functions

they perform calculation on a set of values, and return a single value

- → COUNT()
- → MAX()
- → MIN()
- → SUM()
- → AVG()

```
SELECT MAX(marks) FROM student;

SELECT COUNT(*) FROM student;
```

→ count, counts duplicates as well, if you want to avoid this

```
SELECT count (DISTINCT name) FROM student
```

→ GROUP_CONCAT()

concatnates values from multiple rows into a single string

```
SELECT GROUP_CONCAT(name) FROM students;
```

Adam, Bob, casey

SELECT department, GROUP_CONCAT(name) FROM Students GROUP by department;

CSE Aman, bob

ECE casey

GROUP by clause

groups rows that have the same values into summary rows

it collects data from multiple records and groups the result by one or more column

SELECT city, COUNT(name) from students GROUP by city;

SELECT city, name, COUNT(name) from students GROUP by city, name;

mumbai chetan 1

mumbai swarna 1

→ if there were two chetan's in mumbai we would get

mumbai chetan 2

HAVING CLAUSE

Similar to where i.e, applies some conditions on rows

used when we want to apply any condition after grouping

```
SELECT count(name), city
FROM student GROUP BY city
HAVING max(marks) >90
```

→ it is used to filter records after a group clause

General order

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

table Related queries

update(update existing Rows)

```
UPDATE table_name
SET col1=val1, col2=val2
WHERE condition;
```

```
UPDATE student
  SET grade = "O"
  WHERE grade = "A"
```

→ There is a safe mode in MySQL while updating table, to turn it off write

```
SET SQL_SAFE_UPDATES = 0;
```

Delete

To delete existing rows

```
DELETE FROM table_name
  WHERE marks < 30;
```

FOREIGN KEY

```
FOREIGN KEY (dept_id) REFERENCES dept(id);
```

cascading for fk

→ on delete cascade

when we create a foreign key 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 primary key

→ on update cascade

When we create a foreign key 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

→ Add the following to make reflections in the child table

ON UPDATE CASCADE

ON DELETE CASCADE

→ it refers to a foreign key constraint that specifies what should happen in the child table when changes occur in the parent table

Types of foreign key cascading actions

→ cascade

→ set null

→ set default

→ Restrict

→ NO action

Benefits of cascading

→ Referal integrity

→ simplifie managment

ALTER (To change the schema)

→ Add column

ALTER TABLE table_name

ADD COLUMN column_name datatype constraint;

ALTER TABLE student

ADD COLUMN age INT;

→ drop column

ALTER TABLE table_name

DROP COLUMN column_name;

ALTE TABLE Student

DROP COLUMN age;

→ Rename table

ALTER TABLE table_name

```
RENAME TO new_table_name;
```

→ change column (rename)

```
ALTER TABLE table_name
CHANGE column old_name new_name new_datatype
new_constraint;
```

→ modify column (modify datatype (constraint)

```
ALTER TABLE table_name
MODIFY col_name new_datatype new_constraint;
```

→ truncate (to delete table's data)

```
TRUNCATE TABLE table_name
```

JOINTS

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

→ foreign key is not compulsory for doing joins

Types of JOINS

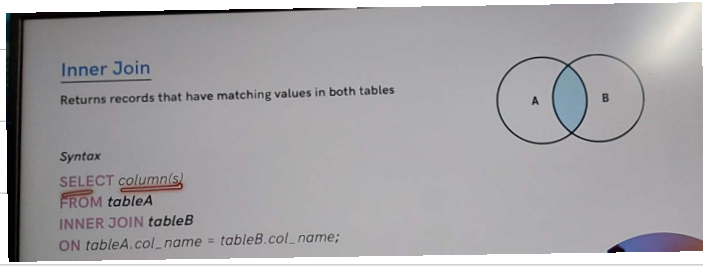
→ Inner join

→ outer join

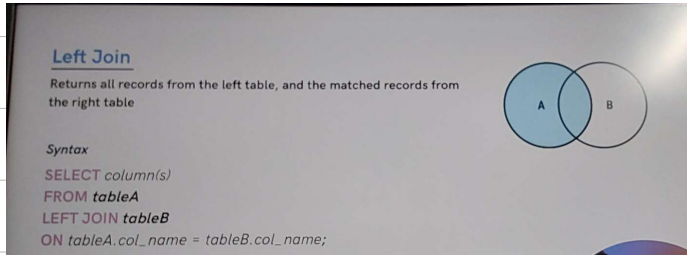
→ left join

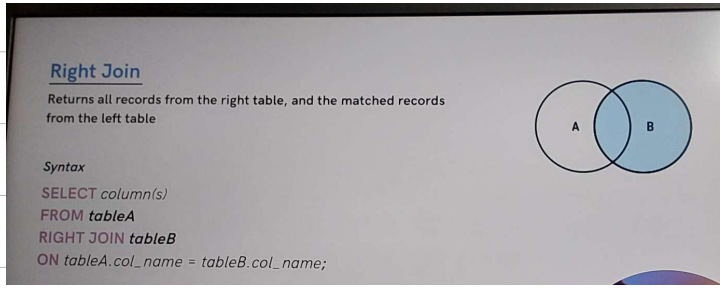
→ right join

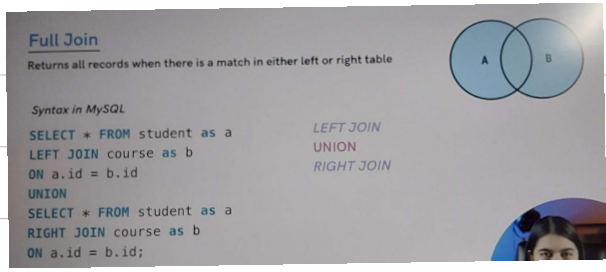
→ Full join



→ Inner join of A with B and vice versa is same



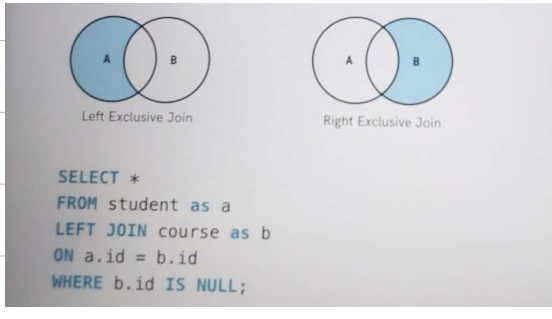




To perform Full join in MSQL we do union of left join and Right join

→ left exclusive join

→ Right exclusive join



→ Full exclusive join

→ self join

→ A regular join but the table is joined with itself

SELECT (COLUMN(S)

```
FROM table_name as a
JOIN table_name as b
ON a.col_name = b.col_name;
```

UNION

→ It is used to combine the result-set of two or more select statements (gives unique records)

→ TO use it;

→ every select should have same no. of columns

→ columns must have similar data types

→ colums in every select should be in same order

```
SELECT column(s) FROM tableA
UNION
SELECT column(s) FROM tableB
```

CROSS JOIN

→ Returns the cartesian product of two tables

of two tables

→ Results in large dataset, combining every row from one table with every row from the other

```
SELECT a.table_name, b.table_name
from employees
CROSS JOIN departments;
```

→ union returns uniqe values and union ALL returns duplicates as well

SQL Sub queries

A subquery, inner query or nested query is a query within another SQL query

```
SELECT column(s) FROM table_name
WHERE colname operator
(subquery);
```

MySQL Views

A view is a virtual table based on the result-set of an SQL statement

```
CREATE VIEW view1 AS

SELECT rollno, name FROM student;

SELECT * FROM view1
```