DBMS LECTURE 1

Intro to DBMS and SQL

DBMS: Database Management System.

Let's understand the word DBMS, Database Management System Management and System are already defined words.

A management system is going to help us manage something.

Database is composed of a word Data.

Q- What is Data?? Ans- Data is any information about something.

Example: A user clicks on a particular button in a website.The information about a user clicking on a button is Data.

So, Data is any information that is important for someone to do their work..

Example: For Scaler, how many students attended the class is a data.

Every infomation that is around the world howsoever small it may be is a data.

Example: Facebook: facebook also tracks a lot of data for us. As soon as a user scroll, likes a picture, upload something even that is the data for facebook.

Now, Let's imagine the Scaler software system, □ So any information that is helpful for Scaler to make it's business decisions or to execute it's business decisions is a data for scaler.

Airbase: Place to store air craft. Similarly, Database: Place to store Data.

A database is a collection of related data. Example: In Scaler codebase, the different related data items might be, Students, mentor, assignment, question. Every data that is related to scaler is one particular database.

Scaler will not be storing the analytics related data, like how many people clicked on a particular button, scroll of a user etc in the same database because it's not the related data.

Now, How to decide that this data should be in same database often depends upon technical architecture.

Example: People who build a microservice, for every microservice they create their own database or the type of use cases.

Example: Scaler has one type of use case: it needs to serve users whereas their might be some analytics related information. So there needs to be seperate database to serve both the purpose. Often what happens is: The database that are actually used to serve users like MySQL, mongoDB are optimized for different set of queries, whereas the databases like Redshift/ Bigtable which are **not** optimized for user's query but are optimized for analytical query.

==So, Databases are seperated by purpose they serve.==

DBMS: The way/The software system used to manage database systems. Hierarchy of the infomation:

□ DBMS like MYSQL, PostgreSQL,MongoDb all of this allow to have multiple databases inside them. In every database one can have multiple data that are related to each other.

Types of DBMS:

The purpose of a DBMS is to store databases and provide ways to interact with data.

Assume, The OS is like DBMS. In one dbms there can be different databases, likewise every database is a different folder. Now In folder1, User create one file called students.txt, then there is Instructors.txt file, then assignments.txt. So each of these is a text file in which like, every row of students.txt have information about 1 student similarly instructors and assignment.txt also. □

The real databases or the very first database that came into existence is this: file based DBMS.

1. File Based DBMS: * Stores data as files on disk. Example: It's very similar to previous example, the information about students might be maintained in 1 file, similarly information about instructors might be maintained in one file.

Cons of file based system: * **Retrieval is not Easy:** In file traversal happens one by one. file is nothing but a series of bytes so one start reading from the very first byte till the place where the result is. So Because the traversal is sequential hence retreival is slow and inefficent.

- **Possiblity of error:** *For Example:* There is a student and a assignement file. We need to find a relation that this which student have this assignment, That means one have to store a relation between students and assignments. Here Let's say this is the students file. In this file, there is a student with id 1, Name 'Naman', email 'xyz' then assignments are there.

Now, Let's say there is no assignment with no. 241 Is the file system going to through some error? No, File system is just storing bytes that's it. So there is a possibility of errors specially errors while storing relations. Those realtions might not have the data on the other side. This is actually **Inconsistency.**

For Example: Let's say student data is needed by finance team. So finance team also has students data. Now the students data is also needed by instructors team, so instructor team will have their own folder and they will also have students data. This is also needed by business team. Similarly business team also has students data in their own folder.

Now, The problem is **Duplication.** And because of Duplication, ==Maintaining consistency becomes a problem.==

- **Inconsistency and Duplication:** Let's say the student Naman decided to change his email. Scaler made an update on the students table as well as on the Finance table but was not aware of the instructor's table. So now at two places Naman has different email id's. So there is no way to know which one is correct or the latest one. This can lead to inconsistent issues. ==The same information has two different values at two different places and it's difficult to examine which one is actually correct.==

- **Security:** Anyone opens the file and can have the whole database. So no security is offered by file databases.

- **No support for Concurrency:** In a file based DBMS, if there are 2 people who are trying to access the data at the same time. They cannot able to do that. Since the data eventually is DISC.

Now, **How a Disc works?** In disc there is a pointer and this pointer is basically moving, it traverses the disc goes to the place where the user wants to find the data, it gives the data and then moves ahead. And there is only 1 pointer per disc.

Disc can access 1 information at 1 time, because pointer will only go to one particular place at one time. So, there is no possibility of concurrent access. The pointer cannot go to more than one place at the same time, it will first serve 1 request then another request. That means in real world systems, the large scaled systems this DBMS is not going to be very efficient.

So that is how came the use case of **Dedicated DBMS**, rather than relying on file systems to store our data these days, more sophisticated, complicated and feature full DBMS came into existence.

2. Dedicated DBMS: *Features:*

1. Backups: Data Backups are very important thing because In reality Data loss is very common, the hatdware in which the data is being storted can get messsed up, the disc on which data is stored might get corrupted. So backups are really imporatant.

2. Concurrency: These days concurrency is very important because the scale of applications has become huge.

3. Security: Support for Encryption, even file level encryption etc are properties that are provided by DBMS these days.

4. Efficient Storage and Retrieval: Different databases like MongoDb, MYSQL, PostgreSQL have different efficency. For Example: MongoDB's are most efficient for these types of operations, similarly MYSQL in another set of operations. So Software implementations of DBMS offer very high storage and efficient retrieval.

Q- How does the database store data? Does data never go to Disks in that case? Now If the machine shuts down, then also data should remain persistent. But what happens is, DBMS is a software system, it's an application, so it has a copy or it has some part of the data in memory. Every operation doesn't involve directly going into the disc, it has memory in front of it to speed up the operation. So that's how real DBMS work these days.

For Example: DBMS have indexes and indexes are stored in memory, so whenever a user try to access something, first it will be searched in memory where is this particular thing stored, then we go to that location in disc.

Non-Relational DBMS(NO SQL DBMS): * Don't follow the relational model: where data is not stored as rows in the table.

In relational database we have a table, if we want to find something in table, then searching starts in linear fashion, row by row and then get the data. That's not efficient in all the cases.

For Example: Use case of facebook friends, Let's say facebook has a user's table like this, User 1 has following friends.

Let's say for each user we want to find their friends.

Finding friends stored in this particular fashion is not efficient.

How could the data have been stored so that it is the best scenerio? In the same record. Like all the friends are stored immediately after this.

That's going to be more efiicient, like just directly get that memory block. So there are many DBMS that exist these days that don't actually store data in rows, store data in a different way, which is suitable for different type of operations. Known as NO SQL DBMS.

Based on how they stored data, further characterized further. Types of NOSQL. 1. Graph Based: Neo4J is the example. 2. Colummar: Where the data is not actually stored in rows, but in columns. Example: Cassandra

---

Relational Databases:

- This is the most widely used model.

```
 Start with SQL Database and then contradict why SQL will not work.
If there is an application where there are many writes, then in thast case SQL will not work.
```

- The model behind SQL.
- SQL is the language for Relational databases.
- Represent data as collection of relations.

Firsty database is the collection of data whereas relational databases are collection of relations. Both lines are true so relations is nothing but data.

According to relational database, a **Relation** is nothing but table of values.

So every table we see in MYSQL database, that is a relation. For Example: In scaler's database, There is relations called student's relation because it's a students table, or batches relation, or courses relation.

`It's not relationship, it's relation..`

Now inside the relations there are going to be many rows, this is the data, whatever the user put in these rows that is the data. ==Data is put inside different relations and different relations are put inside MYSQL database.==

So again, Relational database is going to store data in diferent **relations** and each relations is nothing but **tables**.

Now, each of these rows is actually an information of an individual student or an individual batch.
so each of these is known as an ==Entity.== **Each row is an information about a real Entity.**

==So Relational databases say's that, collect data into relations. A relation in logical form is nothing but a table, and inside the table there are multiple rows.==

Terms: 1. Relation: Table

2. Attributes: Properties of relation: Columns

3. Tuple: Row of a table.

4. Degree: Number of attributes of a relation: Number of Columns of a table. For Example: What is the degree of student's relation, How to find that? Just count the number of columns.

5. Cardinality: Number of tuples in the relation: Number of rows in a table.

The cardinality of a table change more often than degree. Because whenever we insert a new value, the cardinality is going to change.

Properties of a relation/table:

1. Each row is unique: Means atleast one value should be different. We cannot have a relation called student's where every information is same.

There are 2 students whose psp is 90, now there are duplicates in relation.

This is resolved by having **id column.**

2. The values in cells should be atomic: No list/JSON/collection MYSQL 5 or before 8, doesn't actually allow to store list/JSON/collection because it actually confirms to the real relational model.

Phone numbers are not single value, they are list of values. Traditional SQL databases, not allowed to have this information.

```
Now, MYSQL 8 still don't have array, but allow JSON.
```

Prefered way:

3. Order of columns shouldn't matter: Order of the column should never be assumed or should never be relied upon. One should name the particular column which is to be accesses since no one should expect a relational database the same order of columns. this order might differ.

So rather than saying select * from students and first column is f_name, be precise, ==select fname,l_name from students.==

Theoretically, Should never hardcore the order of columns, practically they do.

4. Order of rows should not matter: We should never expect that if we make a query, like ==Select * from students where name=naman==, never expect that this query gives 3 rows, never expect that these 3 rows will come in the exact same order the next time we run the query.

These contraints exist in relational model because relational mode of DBMS is based on a mathematical concept: Set Theory.

Relation=Set Every table/relation is a set. In Set order of values doesn't matter ==Set, {A,B}={B,A} List, [A,B]!=[B,A]==

So if every set is a relation that means every tuple is a value inside set, so every table is nothing but a set of tuples. ==Set,{(T1),(T2),(T3)}== So order of these tuples will not matter. That's why order of rows should not matter since every row is nothing but a tuple.

Cardinality of Set: Number of values in Set, Cardinality of relation: Number of rows in relation.

The set{A,A,B,C}={C,B,A}

In HashSet we cannot put multiple keys with same values because duplicate don't matter.. ==Same in relation, each row is 100% unique.==

In Students table, multiple students are there.

What are the values in this table that can guarantee that only 1 person will come up. Email,Phone_number -> Key

Key:

Attribute/set of attributes that can uniquely identify a row in a relation.

In this table,

Types of Keys: 1. Super Key: A super key most minimal constraints. It is a set of attributes that can uniquely identify a relation. So whatever is defined about key, that is Super key.

2. Candidate key: A key of the minimum size that can uniquely identify a relation. Minimum size: If we remove any of the attributes, the resulting attribute will not be a key.

Since after removing 1 from (email,phone_no) still we are getting minimal set. Super Key of minimum size.

3. Primary Key: One of the candidate keys chosen by the database architect/creator as the key of the table. ==Only/Exactly 1 per table.== We have 2 candidate keys, the database administrator will choose 1 from them,

But email is also not a good option, since email is a string, so string is actually repeated multiple times in the mapping tables. A string repeated multiple time is wastage of storage. Now, also a user can change email, therefore email is not a good choice from design perspective.

So, 1. It can change. 2. It will be duplicated in the mapping/referring tables. ==Ideally key should never change.==

In this case, No row is there which will not change. So, In practicality, we create another column called student_id..

Now, even if the email of a person change student_id, remain same..

Student_id is super key, candidate key and choosen to be Primary key Now, if we take student_id as primary key we are just repeating integer not string.

==Primary keys are actually automatically generated integers or some value which is not the user's information.==

4. Composite Key: A key with greater than or equal to 2 values. In above example, primary key was not a composite key.

5. Foreign key: Now, Let's imagine students table, and every student has a batch in which they are part of, every student can be part of 1 batch only.

Primary key of students table: **id**

Now, the responsibility of batch_id is it's going to uniquely identify **batches**. But batches are not stored in students table instead in some other table. ==So, this batch_id which is uniquely identifying batches(stored in different table) is known as foreign key.== ==Foreign key is not associated to one's table, it is associated to other table.==

Primary key of Batches table: id

Basic SQL:

Links to download for using MYSQL: https://dev.mysql.com/downloads/mysql/ MYSQL workbench https://dev.mysql.com/downloads/workbench/

Open MYSQL workbench, then click on :heavy_plus_sign: icon near MYSQL connection. Then write connection name(any name) and if password required put that, then click Ok.

Now after that we'll see a Panel where we can edit, write some queries there.

SQL:

Structured Query Language.

Query Language means to query over something. It's a language that is used to query over relational databases since relational databases are in structured format as in rows and columns therefore It's called Structured Query Language.

==Language used to manipulate data(insert,update,read,delete).==

Create Database:

`Create database scaler_class;` Every SQL statement will end with semicolon. Execute any SQL statement using command+enter.

Just refresh, and can see a scaler_class.

Now, the problem is if someone shares this particular file with anyone and the person runs it, They will see an error. Because it's already created.

So don't just write create database but instead, `create database if not exist scaler_class;` Now it has not shown error, just warning.

Delete Database: `Drop database scaler_class;`

Same problem again therefore, `Drop database if exists scaler_class.` Now, just showing warning, not error.

Now, for all the queries, we will be using scaler_class database.

`Use scaler_class;` Now, Whatever queries we are going to write they are going to automatically assume that use scaler_class database only.

For Example: Create a table.

```
 create table students(
student_id INT(data type),
first_name VARCHAR(10),
last_name VARCHAR(10)
);
```

==Ideally, whenever we create a table, create it with Primary Key only.==

```
 create table students(
student_id INT(data type) PRIMARY KEY,
first_name VARCHAR(10),
last_name VARCHAR(10)
);
```

Now, it created this table in scaler_class database because we have used scaler_class.

Now, Let's say we want to create this table in bakery database.

```
 create table bakery.students(
student_id INT(data type) PRIMARY KEY,
first_name VARCHAR(10),
last_name VARCHAR(10)
);
```

```
 create table scaler_class.students(
student_id INT(data type) PRIMARY KEY,
first_name VARCHAR(10),
last_name VARCHAR(10)
);
```

==So whenever we use the table, prefix it with name of database or use USE keyword.==

`DROP table scaler_class.students;`

Also, write if not exists and if exists:

```
 create table if not exists scaler_class.students(
student_id INT(data type) PRIMARY KEY,
first_name VARCHAR(10),
last_name VARCHAR(10)
);
```

`DROP table if exists scaler_class.students;`

Now, create one more table batches,

```
 CREATE table if not exists batches(
batch_id INT PRIMARY KEY,
batch_name VARCHAR(20),
instructor_name VARCHAR(20)
);
```

We have 2 tables in scaler_class database.

Now, we want that every student should have batch_id. Delete the table and again create the table but this time add batch_id column as well.

```
 create table if not exists scaler_class.students(
student_id INT(data type) PRIMARY KEY,
first_name VARCHAR(10),
last_name VARCHAR(10),
batch_id INT
);
```

Problem: Existing data will be gone, We can't always do that.

So There are two ways:

Create the table, after that modify them. ==Modify table: ALTER keyword is used.==

```
 alter table students
add batch_id INT;
```

Run this and it's updated.

batch_id is foreign key.

```
 Alter table students
add foreign key fk_students_batches (batch_id)
references batches (batch_id);
```

- Foreign key convention: fk_tables name to refer(column name on which the relation is going to made) column name what does it refers to.
