DBMS Lecture 2

DB Schema:

The very first step is to design the database.

In SQL database, we normally mention that we are having these tables, these are the attributes and the cardinality of the relation.

==Schema is like the Pictorial representation of how the database is going to be structured.== - Design Document - Schema Design is also a part of Design Doc.

Example: If someone starts with a particular schema and later understand that it has a problem, then will have to run a **DB migration.** DB Migration are often not easy to run, they might require a downtime, some corner cases to handle, so that's why a Schema should be designed properly, as early as possible.

How to approach Schema Design:

Case Study: Model Scaler

Requirements: 1. There are several batches at scaler. Each batch has name, start month, instructor(current), students. 2. Each batch has multiple classes. Each class has a name, instructor. 3. Every student has name, grad year, university name, email id, phone no(only 1 number). 4. Each student has a student buddy. 5. A student may move from one batch to other-> we want to know the entry date of a student on a batch So, for every student and for every batch they were a part of, we want to store when they join as well as when did they leave. 6. Every Student has a Mentor. Every mentor has a name, DOB.

Design part:

1. Find all ==entities== in the requirements->Entities are anything for which we are storing details, causing any behaviour in the system, might be a real object(student,person), concept in the system(batch,class).

- How to find entities: - Find all Nouns in the requirements. For every noun if we have to store any information about that Create a table about this. - Visualization(See in LLD classes).

Entities: Batches, instructor,students,classes,mentor. Create tables for each entities.

Nomenclatures for tables: Some Conventions: 1. Prefer having plural names for tables. 2. Normally go with the company conventions.

Then write the attributes of each table. again, **Conventions for attributes:** 1. currentInstructor(Camelcase) 2. current_instructor(More common convention in SQL). 3. Go with the convention that your comapany uses. 4. current instructor(we can have this, but don't do this highly wrong)

Now, In batches table, create id for primary key. **Convention of id:** 1. id (more popular). 2. {table_name}_id-> (batch_id): This convention helps in natural joins.

On a first go, We had just put requirements into tables. Not the perfect design.

The datatype of id is INT The datatype of current_instructor is instructor(but again this is not making any sense).

Cardinalitites of relationship between Entities. Two Entities can be related to each other. *Example:* In above requirement, every batch has an instructor. so, batch is related to instructor.

==Now,If there is a relation between A and B, Cardinality says, How many of A are related to How many of B.== - 1:1 - 1:M - M:1 - M:M

Cardinality of batch and instructor.

So, **How to find Cardinality of a relation b/w A and B** 1. Find which relation between A and B are we talking about.

For Example: students have 2 types of relation with batches. - current_batches - previous_batches Here, Let's assume A: Student B: Batch Relation: Current_batch

2. Two step approach.
    - Put 1 on the left side
    - 1 student can have how many current_batches? 1

- Now, In right side, 1 batch can be a current_batch of how many students? M
- If we have M on any side, put M there, otherwise 1

3. If M on any side, Put M on that side, else 1 on that side.

Now, A: Student B: Batch Relation: previous_batch

Now, A: Class B: Instructor

Now, A: Batch B:current_instructor

Now, A:Batch B:Class We can merge the batches, then 1 class can be for many batches.

Schema Design Now:

How to represent different cardinalities in database.

1. 1:1 We don't have any 1:1 relationship in above use cases, so let's take an example of: A:Husband B:Wife

Now, How can we store the information, that husband has that wife in a table: Use Foreign key. Put foreign key in any table.

So, In 1:1 relation, put the id of other(any) side on the different side. - either husband_id oin wives table OR - wives_id in husband table.

We can do both, but leads to redundant data, not necessary.

2. 1:M AND M:1

1 class has 1 instructor 1 instructor can teach mutiple classes.

2 Options: - The table which is related to many, can have a list. - The table that is related to 1 can have a single id. But the second option is not good. ==Multivalued attributes are not recommended.==

So, If cardinality of relation b/w A:B is M:1 or 1:M,put the id of **One side on M side.**

3. M:M

- Create a new table just to store this relation-> Mapping table/Lookup table

The Primary key of this mapping table: - Combination of both foreign keys-> Composite key(batch_id,class_id).(Always go with this approach unless a reason not to go with this) - A new Column.

Reasons for not going with 1^st^ approach: 1. If the mapping table can also have relations with other table. 2. There are attributes of the relation and the relation kind of acts like an entity. For Example: Let's say for every batch, every class we had an assignment as well.

For every batch, every class there can be different assignment. There can be M:M relation, so create a seperate table.

Why prefer muti-valued or composite Primary key: By default, the table is sorted by pk and index on pk. If the pk is (b_id,c_id) table will be sorted by b_id(first column) implies that queries like finding all classes of batch will be faster. This is the reason we should prefer this aproach.

Caveats: 1:1 / 1:M/ M:1

Let's say we prefer going with having wife_id in husbands table. Now, There are suppose, 5 Million husbands out of which 3 million don't have wives. That means there will be a lot of NULL in the table containing foreign key.

Same thing can happen in 1:M and M:1 as well.

Let's say a scenerio can be:

Students and batches tables. Every student has a batch_id. But not every student belong to a batch. For those students, batch_id will be NULL.

Now, Let's say 25k students join Scaler Masterclass. Now 500 will enroll for the course that means 24.5k NULL. It will waste storage.A lot of Null's.

1. If a relation representation can have a lot of NULL(sparse table), Considering representing the relation via mapping table.
2. A relation can also have attributes.

For example: A: Student B: Batch Relation: Past batches So, we represent M:M relation using Mapping table. This relation has some attributes in requirements. Start_date and end_date. Now, This is actually kind of acting like an entity.

Can 1:1 or 1:M or M:1 have attributes? Yes.

So In husbands and wives table, there can be marriage_date as attribute. It should be stored in either of the table. Now, there can be other dates as well. So we put all those in husbands table.

2. Whenever any relation(1:1/1:M/M:1) starts acting like an entity(a lot of attributes) considering it like an entity(a mapping table with an id).

SumUp:

Now, Let's model schema:

Since, Batches and current_instructor have M:1 relation. So, current_instructor_id in batches table.

Now, batches and class: M:M Therefore in mapping table called batch_classes will have b_id, c_id.

In students table, cardinality of relation b/w student and current_batch: M:1, so put curr_batch_id in students table. There is an attribute b/w them, start date: put curr_batch_st_date in students. In future if we get multiple attributes then make a mapping table.

Similarly, cardinality b/w students and prev_batches, M:M Create a mapping table b/w student_batch_prev_batches where we have id, batch_id, st_date, end_date.

In classes table, cardinality of relation b/w class and instructor: M:1 Just put instructor_id in classes table.

Cardinality of relation b/w students and mentor. So again in students table, put mentor_id.

Complete schema:

SQL Data Types:

Whenever we do programming in any language, whenever we create variable, we do specify the data type of the variable. The Data Type is needed because it helps the computer to decide how much memory to allocate to the variable, or how much memory to take from heap,etc.

Data Types of columns in the tables: - When creating a table, do specify data type of every column.

Types of Data Types:(MYSQL 8)

1. Strings
2. Integers
3. Floating Points
4. Booleans
5. Enum
6. Date Time
7. JSON

String Data Type: There are 3 types of strings data types:

1. Char(X):
    - Strings of fixed maximum length.

        - X can be 0 to 255
        - CHAR(4) Can we be able to put 'abc' here? YES. Internally it will be stored as "abc "
    - Now, If we put 'abcd' here, It will be stored like, "abcd"
        - Now, If we put 'a' here, It will be stored as "a "
    - If we put 'abcde' here,
    - Some version might throw error or it will store 1^st^ 4 characters "abcd".
    - Example: Phone no, Pin no.
2. Varchar(X):
    - String of variable length
    - X can be between 0 to 65535
    - X mean indication that this can be an average sixe.
    - Initial 1 to 2 bytes of varchar are used to store length of the string.
    - Example: If we want to store 'ab' then it will stored like, 2ab
    - Byte1: 2, Byte2: a, Byte3: b
    - Now, Let's say we want to store 300 chars,
    - Byte1: 30, Byte2: 0, then 1 byte for every character.
    - Now, Let's say we are storing an empty byte "": Size is 0 therefore 1 byte just to store size.
    - 'a': Takes 2 bytes, Byte1 1, Byte2: a.
    - 'ab': Takes 3 bytes, Byte1: 2, Byte2: a, Byte3: b
    - 'abc': Takes 4 bytes, 3abc.
    - 'abcd': Takes 5 bytes.
    - Varchar should be used to store name, email, etc.
3. Text:
    - means there are different types of text.
    - TINYTEXT: 255Bytes
    - TEXT: 64KB
    - MEDIUMTEXT: 16MB
    - LONGTEXT: 4GB
    - Only difference is the size of string stored in them.
    - Medium and long text are very big so consider storing in ==files== rather than in data types.

Difference between Varchar and Text is: TEXT columns can not be indexed(discuss later). *For Example:* `Select * where name='naman'` so name should be a varchar because you might want to index on names.
