
Agenda

- Nuances when representing relations
- Scaler Schema Design - continued
- Deciding Primary Keys of a mapping table
- Representing Foreign keys and indexes
- Case Study - Schema design of Netflix

Scaler Schema Design - continued

For reference from previous class, Scaler Schema Design:

The requirements are as follows:

1. Scaler will have multiple batches.
2. For each batch, we need to store the name, start month and current instructor.
3. Each batch of Scaler will have multiple students.
4. Each batch has multiple classes.
5. For each class, store the name, date and time, instructor of the class.
6. For every student, we store their name, graduation year, University name, email, phone number.
7. Every student has a buddy, who is also a student.
8. A student may move from one batch to another.
9. For each batch a student moves to, the date of starting is stored.
10. Every student has a mentor.
11. For every mentor, we store their name and current company name.
12. Store information about all mentor sessions (time, duration, student, mentor, student rating, mentor rating).
13. For every batch, store if it is an Academy-batch or a DSML-batch.

Tables

`batches`

batch_id	name	start_month	curr_inst_id

`students`

student_id	name	email	phone_number	grad_year	univ_name	batch_id

`batch_classes`

batch_id	class_id

Now, let's continue from here. What is the cardinality between `class` and `instructor`. As this is m:1 cardinality, `instructor_id` will be included in `classes`.

`classes`

class_id	name	schedule_time	instructor_id

Every student has a buddy. Here, the cardinality of the buddy relation between a student and another student is m:1.

student	--- buddy ---	student
1	-->	1
m	<--	1

So, the `students` table will have one more column called `buddy_id`.

`students`

student_id	name	email	phone_number	grad_year	univ_name	batch_id	bu

When a student is moved from one batch to another, this date is an attribute of the relation between `students` and `batches`. So, we will create a new table like this:

`student_batches`

student_id	batch_id	move_date

As we have included `batch_id` here, we can remove it from `students` table but that will decrease the performance because everytime we will have to query on this new table also. So, for ease, we will keep the `batch_id` in `students` also.

Every student has a mentor, the cardinality between student and mentor is m:1. So, the `students` table will have `mentor_id`.

`students`

student_id	name	email	phone_number	grad_year	univ_name	batch_id	bu

Now, for mentor sessions we will add the `student_id` and `mentor_id` in the `mentor_sessions` table.

`mentor_sessions`

mentor_session_id	time	duration	student_rating	mentor_rating	student_id

Now, for the batch type, it can be DSML or Academy. Here, the batch type is enum (enum represents one of the given fixed set of values).

Eg:

```
enum Gender{
    male,
    female
};
```

So, we will have a `batch_types` table.

`batch_types`

id	value

Cardinality between `batches` and `batch_types` will be m:1. In `batches` table we will have `batch_type_id`.

`batches`

batch_id	name	start_month	curr_inst_id	batch_type_id

This was a brilliant example of how to create a Schema Design.

How to represent enum

1. **Using strings**

   `batches`

batch_id	name	type
1	b1	DSML
2	b2	Academy
3	b3	Academy
4	b4	DSML

```
**Pros:**
- Readability.
- No joins are required.

**Cons:**
- The problem in storing enums this way is that it will take a lot of space.
- It will have slow string comparison.
```

2. **Using integers** Here, 0 means DSML type batch and 1 means Academy type batch.

```
`batches`

| batch_id | name | type_id |
| -------- | ---- | ------- |
| 1        | b1   | 0       |
| 2        | b2   | 1       |
| 3        | b3   | 1       |
| 4        | b4   | 0       |


**Pros:**
- Less space
- Faster to search

**Cons:**
- No readability.
- We can not add or delete values (enums) in between as it will cause discrepencies.
- Also, what a particular value represents is not in the database.
```

3. **Lookup table** It will have id and value columns where each type is stored as separate. The `type_id` of `batches` will refer to the `id` column of

`batch_types`. All the above cons are solved with this method.

`batch_types`

id	value
1	Academy
2	DSML
3	Neovarsity
4	SST

So, the best way to represent enums is to use lookup table.

Deciding Primary Keys of a mapping table

Example from previous discussion:

For `student_batches` the primary key will be (student_id, batch_id).

`student_batches`

student_id	batch_id	move_date

OR

`student_batches`

id	student_id	batch_id	move_date

If in case we have our table like this, the primary key will be `id`. **Size of index will be lesser here.**

Now, can there be a possibility that we might want to join a mapping table with another mapping table? **Answer:** Yes!

Example 2

1. Scaler has exams.
2. For each batch a student joins, they will have to take exams of that batch.
3. Each exam is associated to a batch.

`exams`

id	name	start_date	end_date

Between batch and exam, each exam is associated to a batch, we will have to create a mapping table. One batch can have multiple exams, One exam can be present fo multiple batches.

`exam_batches`

exam_id	batch_id

Similarly we also have a table called `student_batches`.

`student_batches`

student_id	batch_id	date

To figure out which student went through which exams, we will need to join `student_batches` with `exam_batches`. Basically, we are forming a relation between two mapping tables.

Example 3

1. One student can belong to multiple batches.
2. Every batch has exams.
3. Same exam may happen on different batches on different dates.
4. If a students moves the batch, they may have to give some exams again.

`student_batches`

student_id	batch_id	date

Cardinality between batches ad exams is m:m. So, we will have a `batch_exams` table. Date is also an attribute of this relation.

`batch_exams`

batch_id	exam_id	date

Between students and exams also the cardinality is m:m. But if we have (student_id, exam_id) as primary key of the new `student_exams` table, it will not allow one student to take a particular exam twice. So, we will have to add `batch_id` also in PK. The below `student_batch_exams` will be our new table.

`student_batch_exams`

student_id	batch_id	exam_id	marks

Hence, we can see that sometimes a mapping may also have a relation with another entity. In these cases, not having a primary key can cause problems.

Advantages of a separate key: If a relation is being mapped to another entity or relation, it saves space.

Advantages of NO separate key: Queries on first column will become faster because the table will be sorted by that column. A mapping table is often used for relationships and thus will require joins. Having no separate key makes things faster.

Representing Foreign keys and indexes

Along with Schema Design questions, use cases are also mentioned. These use cases govern what indexes will be there. For example, we need a function to find all the classes of a batch. For this, we will simply have an index on `batch_id`.

`batch_classes`

batch_id	class_id

Let's say that the learners often search mentor by a name. This is a use case. On which column of which table will you create an index for this? You have to create an index on `name` column of `mentors` table.

`mentors` | mentor_id | name | company_name | |-----------|------|--------------|

Now, foreign key is mentioned alongside creating Schema during the third step (representing relationships). You will mention after creating the attributes that this `column_A` of `table_A` will have a foreign key referring to the `column_B` of `table_B`.

After drawing the complete Schema, mention the indexes.

This was all about Schema Design!

Case Study - Schema design of netflix

Following is tge link to Netflix's requirements: [Netflix Schema Design]

Problem Statement Design Database Schema for a system like Netflix with following Use Cases. **Use Cases**

1. Netflix has users.
2. Every user has an email and a password.
3. Users can create profiles to have separate independent environments.
4. Each profile has a name and a type. Type can be KID or ADULT.
5. There are multiple videos on netflix.
6. For each video, there will be a title, description and a cast.
7. A cast is a list of actors who were a part of the video. For each actor we need to know their name and list of videos they were a part of.
8. For every video, for any profile who watched that video, we need to know the status (COMPLETED/ IN PROGRESS).
9. For every profile for whom a video is in progress, we want to know their last watch timestamp.

Let's approach this problem as one should in an interview.

1. Finding all the nouns to create tables.

- `users`
- `profiles`
- `profile_type` (lookup table)
- `videos`
- `actors` (cast is nothing but a mapping between videos and actors)
- `watch_status_type` (enum, it is an attribute of relation between profile and videos)

2. Finding attributes of particular entites.

   `users`

id	email	password

   `profiles`

id	name

`profile_type`

id	value

`videos`

id	name	description

`actors`

id	name

`watch_status_type`

id	value

3. Representing relationships.

   Now, there are no relationships in the first and second use cases. Moving forward, what is the cardinality between `users` and `profiles`? One user can have multiple profiles but one profile is associated with one user. Therefore, it is 1:m, id of user will be in `profiles` table.

   `profiles`

id	name	user_id

   What is the cardinality between `profiles` and `profile_type`? It is m:1, `profiles` will have another column `profile_type_id`.

   `profiles`

id	name	user_id	profile_type_id

   What is the cardinality between `videos` and `actors`? One video can have multiple actors and one actor could be in multiple videos. So, it is m:m.

   `video_actors`

video_id	actor_id

   Status is an information about relation between `videos` and `profiles`. Hence, a new table is created. Last watch timestamp is also an attribute on these two.

   `video_profiles`

video_id	profile_id	watch_status_type_id	watched_till

Time for a some follow up questions.

Quiz 1

What should be the primary key of `video_profiles`?

Choices

- ☐ (video_id, profile_id)
- ☐ (profile_id, video_id)
- ☐ (id) a new column

Quiz explanation

In this question, (profile_id, video_id) is the best option because as soon as we open netflix and a particular profile, it shows us the videos we are currently watching.
So, to make this query faster, primary key should have `profile_id` first.

This is all about the Schema Design!