SQL 13: SCHEMA DESIGN - 2

31/05/24

AGENDA

1. Scaler Schema Design - Complete Today
2. Represent Enums
3. Deciding PK of a mapping table
4. Represent FK
5. Represent Indexes
6. Case Study: Netflix Schema Design

Scaler Schema Design

1. batches
2. students
3. classes
4. instructors
5. mentors
6. mentor_sessions.

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.

① batches
- batch_id
- batch_name
- start_date
- inst_id
- bt_id

1 batches — m
1 instructors — 1
m : 1

② Students
- st_id
- st_name
- email
- phone_no
- univ_name
- grad_year
- buddy_id
- mentor_id

m Students — 1
1 batches — m
m : m

③ Student_batches
- st_id
- b_id
- date_of_start

④ Classes
- c_id
- c_name
- c_time
- inst_id

1 Class — m
1 instructor — 1
m : 1

⑤ instructors
- inst_id
- inst_name

⑥ mentors
- m_id
- m_name
- comp_name

1 1
Students mentors
m 1
m : 1

⑦ mentor_sessions
- ms_id
- ms_name
- duration
- st_rating
- mentor_rating
- start_time
- st_id
- mentor_id

Table: batches

①

b_id	b_name	b_type
1	A	Academy
2	B	DSML
3	C	DSML
4	D	Academy
5	E	Academy.
6	F	SST

Pros
① Readability
② Joins not required

Cons
① Error prone while inserting data. (WHILE)
② Memory - varchar(255) - Extra space.

Query

```
SELECT * FROM BATCHES
    WHERE B_TYPE = 'DSML';
```

③ String comparisons are slow.

②

Table: Batches

b_id	b_name	b_type
1	A	1
2	B	2
3	C	2
4	D	1
5	E	1
6	F	

ENUM

Code / Documentation

ACADEMY - 1

SST - 2

DSML - 3

Pros
① less space
② faster search / comparisons.

Cons
① No readability.
② Add/delete values in the middle
- can cause discrepancies
③ No info in DB about the mapping.

REPRENT ENUMS

③ Look up table.

table: batch_types

bt_id	bt_name
1	Academy
2	DSML
3	SST

Pros → All above problems are solved

Cons → Need to JOIN.

Mapping tables

① student_batches
- st_id
- b_id
- move_date

PK = (st_id, b_id)

Pros → ① no extra col (less space)
② → index (st_id, b_id)
→ joins on st_id faster

② student_batches
- sb_id
- st_id
- b_id
- move_date

Pros → ① size on index smaller. (1 col).

Requirement

UseCase(1): ① Scaler has exams

② for each a Student joins, he/she has to take take an exam (eg. contests, MBEs etc)

③ Each exam is associated with a batch.

Ques Find all the exams taken by a Student

Students
- st_id
- st_name

batches
- b_id
- b_name

join

Student_batches
- st_id
- b_id
- more_data

join

batch_exams
- be_id
- b_id
- e_id

join

exams
- e_id
- e_name

1 batches — m

m exams — 1

m : m

st_name	exam_name

Requirement:

UseCase ① Same exam will happen on different dates in different batches.

② If a student moves to another batch, he/she may have to give same exam again.

Ques How to store the marks obtained in each exam for every student?

→ marks
→ date-of-exam.

b-id

Student - exams

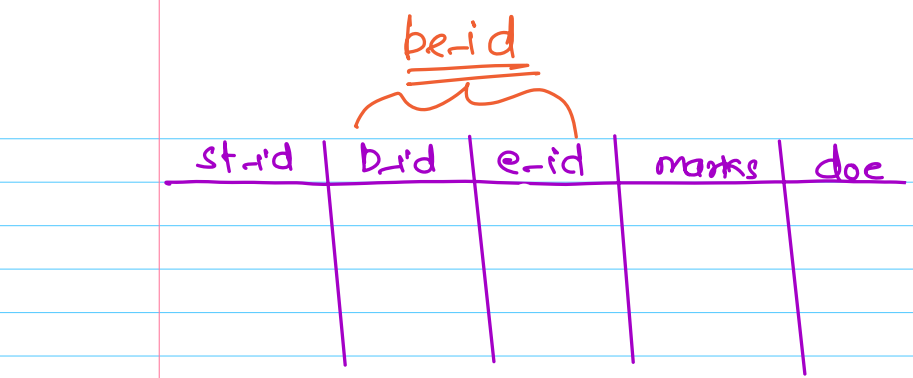
st-id	exam-id	marks	attempt_no	date_of_exam
1	101	50		
1	101	70		

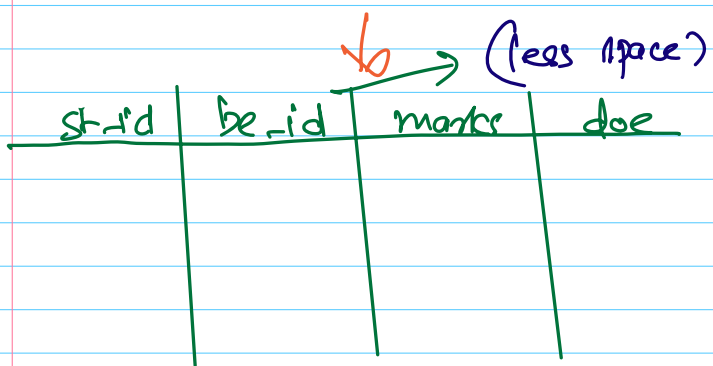
→ contest again in same batch.

Student-batch-exam

st-id	b-id	e-id	marks	doe	attempt
1	1	101	50		
1	1	101	70		

PK = ? (st-id, b-id, exam-id)





twitter Snowflake?

FK - All relations represented as FK

email → lot of queries

INDEXING

→ we will create an index on email. (A-Z).

Note: Indexes will be created based on the use-case.

NOTE: DO NOT CREATE INDEXES PREMATURELY.

BREAK TILL - 8:33 AM.

Netflix Schema Design

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.

Steps to do Schema Design

1. Find all nouns/entities to create tables.
2. Find all attributes (cols) that belong specifically to the tables. (PK)
3. Find the cardinalities of all relations

- add FK
- create mapping tables.

Tables.

1. users
2. videos
3. profiles
4. profile_types (ENUM)
5. actors
6. watch_status (ENUM)

table : batch_type

pt_id	bt_name

1. users
- u_id
- u_name
- email
- password.

2. profiles
- p_id
- p_name
- u_id
- pt_id

1 : m

users (1) — profiles (m)

3. profile_types
- pt_id
- pt_name

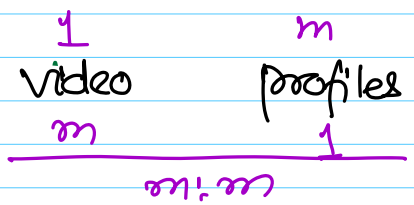
ADULT	1
KID	2

④ videos
- v_id
- v_name
- title
- desc
- ~~wst_id~~

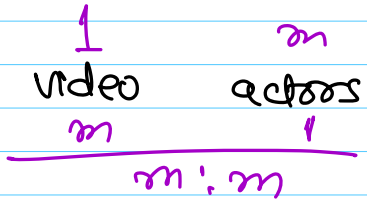
⑤ actors
- a_id
- a_name

⑥ watch_status_type
- wst_id
- wst_name

⑦ video_profiles
- vp_id
- video_id
- profile_id
- wst_id
- last_watch_timestamp.



⑧ video_actors
- video_id
- actor_id



PK of video_profiles. ?

PK = (profile_id, video_id)

Ques Why Range Queries are better in BBST than Hashmap.

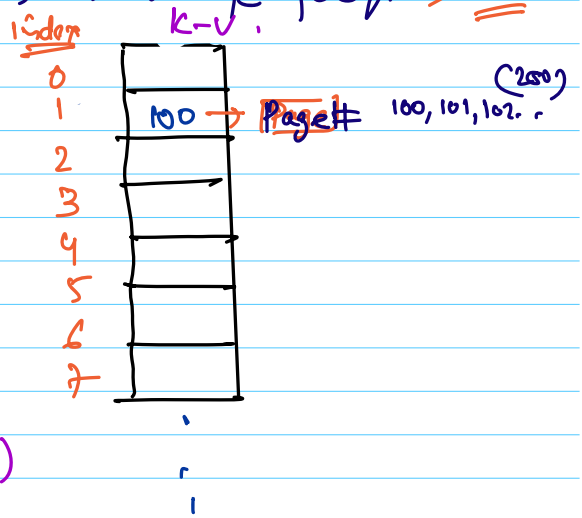
HashMap-

O(1) → Avg time.

Worst TC → O(N)

n = Total no. of elements → 10^{6} (100-200)

K = no. of elements in range query. → 100



hash(100) = 1

hash(200) = 1

O(1) → Avg

Worst O(N).

HM → K - O(n x K)

HM, TC = O(n*K)

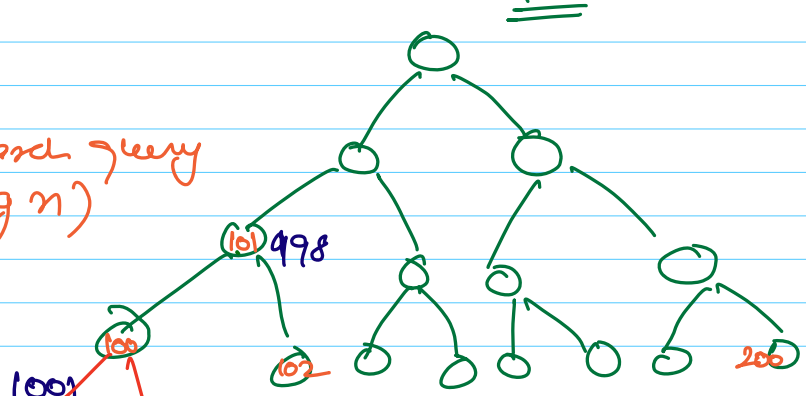
Balanced BST.

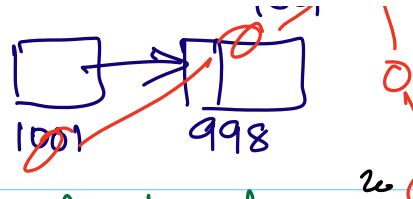
Search

Start of search query

TC = O(log n)

100 ∈ Start





10 → inorder traversal → (sorted) ascending order

TC (inorder traversal)

for (range - k) → O(k).

BBST = O(log n) + O(k)

n >> k.

HM = O(n * k)

(left, root, right)