Schema Design

Contents

[1. Schema Design 2](#_Toc185709693)

[Anomalies 2](#_Toc185709694)

[How to do Schema Design in Practice 5](#_Toc185709695)

[2. Schema Design Process (Recap) 9](#_Toc185709696)

[Sparse Relations in Database Design 10](#_Toc185709697)

[Relations and Their Attributes in Schema Design 13](#_Toc185709698)

[Mapping Tables and Keys 15](#_Toc185709699)

# Schema Design

* As a software developer, we do prepare design document.
  + Class Diagram - How will you implement application. (LLD)
  + Architectural Diagram - What infrastructure layers will be there. (HLD)
  + Schema Design - What tables will be there in our Database.
* Database Schema:
  + How will you store the data to be able to handle the given set of requirements.
  + Schema is blueprint of a real database. A pictorial representation of how database is going to be structured.
* Why do we need Schema design?
  + Should be able to handle all the requirements.
  + Handle requirements efficiently.
  + Avoid anomalies (issues such as redundancy which can cause inconsistency and more storage).

## Anomalies

* Issue due to redundancy.
* There are 3 types of anomalies...
  + Insertion anomaly
  + Deletion anomaly
  + Update anomaly
* When we have redundancy in database these anomalies happen.

### Insertion Anomaly

* Suppose there is a table Students with id, name, marks, batch\_id, batch\_name as columns as shown below...

A screenshot of a computer screen

Description automatically generated

|  |
| --- |
| CREATE TABLE IF NOT EXISTS students (      id INT PRIMARY KEY,      name VARCHAR(100) NOT NULL,      marks INT CHECK (marks >= 0 AND marks <= 100),      batch\_id INT NOT NULL,      batch\_name VARCHAR(50) NOT NULL  ); |

* New requirement has come: Create a new batch named December 2022. No students in that December 2022 batch yet.
* Since, no students are present in that batch, it is not possible to create a new row (id is primary key, name cannot be null.)
* Is this table Students is correctly created? Answer is no. There is lot of redundancy in the table.
* The batch\_name column has redundancy. Note that 'August 2022' is repeated many times. Because of this redundancy we have insertion anomaly.
* Insertion anomaly is *inability to store data about a particular entity till the time we have data about something else*.

### Deletion Anomaly

* New request has come. We need to delete the student with id 4 (Imagine that this student has registered for June 2022 batch first and is the only student in that batch)
* If we delete that student, the batch information is also deleted. Note that the student with id 4 is in June 2022 and there is only 1 student in that batch.

A screenshot of a computer

Description automatically generated

* Deletion anomaly is, at the time of deleting something, we might end up deleting something else.

### Update Anomaly

* A new requirement has come where we need to change the batch\_name 'August 2022' to 'Aug2022'.
* Imagine, we have written a query to update from 'August 2022' to 'Aug2022'. After updating couple of rows, machine has gone down… The state of table will look as show below…

A screenshot of a computer screen

Description automatically generated

* Update anomaly is, at the time of updating something, we might end up in inconsistencies.

### Database Normalization

* It is the techniques that we use to handle redundancy.
* There is something called normal forms. Normal forms are guidelines used in database design to reduce redundancy and improve data integrity by organizing data into tables.
  + 1NF
  + 2NF
  + 3NF
  + 4NF
  + 5NF
  + 6NF
  + BCNF (Boyce Codd Normal Form)
* We don’t use these NF in practice. We never use normal forms in reality. There is more practical way to approach database design.

## How to do Schema Design in Practice

* Schema design of Scaler.
* Let's list out the requirements.
  + Scaler has multiple students.
  + Scaler has multiple batches. Each student belong to exactly one batch at a time. one batch can have multiple students.
  + Every batch has a current instructor.
  + Every batch has multiple classes.
  + 1 class may involve students from multiple batchs
  + Every student has a student buddy.
  + Every student has a mentor.
  + For every mentor we store their company and number of session the mentor has taken.
  + We have to store for every batch a student belong to, date of joining that batch.
  + For every student for every class we have to store attendance.

#### Steps

1. Find all the nouns that are there in the requirements.
   * Out of all the nouns find the nouns that we want to store information about.
   * Create 1 table for each such noun.
   * Good Practices:
     + Name of each table should ideally plural.
     + Represent table name in snake case. (Example: mentor\_sessions)
   * Nouns identified:
     + students
     + batches
     + instructors
     + classes
     + mentors
     + companies
2. For each of these nouns, find what all we need to store.
   * Create an id column
   * If no realation with another noun, create a column for that (primitive attributes). For example, Students will have name and name has no relation with other nouns. Hence it is a primitive attribute.

|  |  |
| --- | --- |
| CREATE TABLE students (      id INT PRIMARY KEY,      name VARCHAR(100) NOT NULL,      email VARCHAR(100) UNIQUE NOT NULL,      phone\_number VARCHAR(15),      graduation\_year INT  ); | CREATE TABLE batches (      id INT PRIMARY KEY,      name VARCHAR(100) NOT NULL,      start\_date DATE,      number\_of\_students INT  ); |
| CREATE TABLE instructors (      id INT PRIMARY KEY,      name VARCHAR(100) NOT NULL,      email VARCHAR(100) UNIQUE NOT NULL,      average\_rating DECIMAL(3, 2),      years\_of\_experience INT  ); | CREATE TABLE classes (      id INT PRIMARY KEY,      start\_time TIME,      title VARCHAR(100),      endtime TIME  ); |
| CREATE TABLE mentors (      id INT PRIMARY KEY,      name VARCHAR(100) NOT NULL,      email VARCHAR(100) UNIQUE NOT NULL,      number\_of\_mentees INT,      average\_rating DECIMAL(3, 2)  ); | CREATE TABLE companies (      id INT PRIMARY KEY,      name VARCHAR(100) NOT NULL  ); |

1. Now we have to represent relations.
   * But how to represent relation?
   * For relation,
     + Which two tables are related.
     + What relation between entities (Find cardinality).

* What is Cardinality?
  + *Cardinality* refers to the relationship between two entities. It defines how entities in one table relate to entities in another.
  + Let’s say we have Students and Batches
    - 1 Student is allotted to 1 Batches.
    - 1 Batches can have multiple Students.

A graph with a black line

Description automatically generated

* + Let’s say we have Husbands and Wifes table
    - 1 husband has 1 wife.
    - 1 wife has 1 husband.

A graph paper with a rectangle

Description automatically generated

* There are 4 types cardinality :
  1. 1:1
  2. 1:m
  3. m:1
  4. m:m
* Steps to find the cardinality…
  + Let’s say we have Movies and Shows table…
    - Go from Left to right and ask the question 1 movie how many shows?

A graph with a line

Description automatically generated

* + - Go from Right to left and ask the question 1 show can have how many movies?

A graph paper with a grid

Description automatically generated

* + - If there no m, put 1 else put M.

A graph with arrows pointing to the distance

Description automatically generated with medium confidence

* For relation in column, if cardinality is
  + 1:1, id of any 1 side on other side.
  + M:1, id of 1 side on m side. (Because we can have a list on 1 side and we cannot store list)
  + 1:M, id of 1 side on m side. (Because we can have a list on 1 side and we cannot store list)
  + M:M, Mapping table. (We have to create a new table, because there will be list on both sides)

A graph with a line and a line

Description automatically generated with medium confidence

* + - We create a new table, Student\_Course with student\_id, courese\_id as columns.

# Schema Design Process (Recap)

**Step 1: Identify Nouns (Entities)**

* Identify nouns for which information needs to be stored.
* Exclude nouns that are indirectly represented, e.g., "attendance" as it belongs to "student."

**Step 2: Create Tables for Identified Entities**

* Define tables for each identified noun.
* Ensure each table has a primary key (e.g., an ID column).
* Store primitive attributes (non-related data) in these tables.
  + Examples: numbers, dates, strings.

**Step 3: Determine Relationships Between Tables**

* Identify related tables.
* Specify how these tables are related (by which attribute).
  + Example: A "student" and a "batch" are related via "batch ID."
* Define the type of relationship (e.g., 1:1, 1:M, M:M).

**Step 4: Define Relationships and Cardinality**

* Analyse the type of relation (1:1, 1:M or M:1, M:M):
  + **1:1 Relationship**
    - The relationship can be stored in either table.
  + **1:M or M:1 Relationship**
    - Store the ID of the "1" side in the table on the "M" side to avoid lists.
    - Ensures compliance with 1NF (no multi-valued attributes).
  + **M:M Relationship**
    - Create a separate mapping table.
    - This table holds IDs from both related tables.
    - Avoids multi-valued attributes and maintains atomicity of values.

**Step 5: Use Cardinality to Finalize Design**

* Determine cardinality:
  + Left-to-right: Assume "1" on the left and find related entities on the right.
  + Right-to-left: Assume "1" on the right and find related entities on the left.
* Assign "1" or "M" based on identified relationships.

**Step 6: Create Mapping Tables for M:M Relationships**

* Mapping tables are necessary for M:M relationships.
* Store IDs from both related tables.
* Avoid violating normalization principles (e.g., 1NF).

**Key Concepts to Remember**

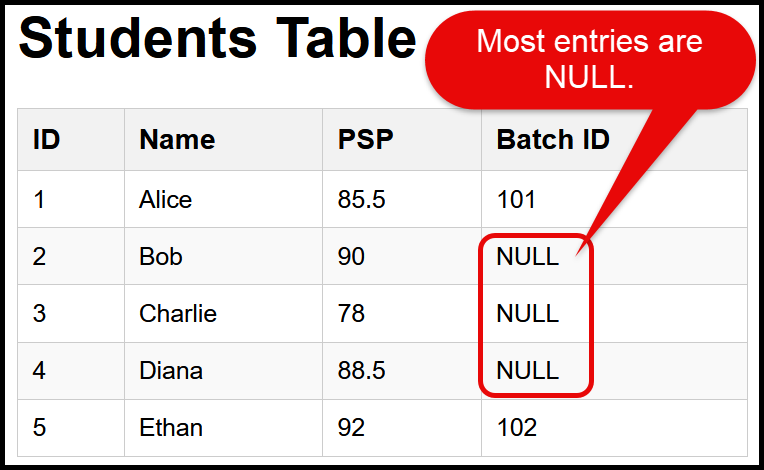
1. **Normalization**
   * Ensures no multi-valued attributes.
   * Maintains atomic values in relational models.
2. **Cardinality**
   * Describes the number of relationships between entities.
   * Helps decide how to store relationships in the schema.
3. **Mapping Tables**
   * Essential for M:M relationships.
   * Prevents anomalies and ensures adherence to the relational model.

## Sparse Relations in Database Design

**Key Concepts:**

1. **Cardinality of Relations**
   * **1:1 (One-to-One):** Each entity in A is related to exactly one entity in B and vice versa.
   * **1:M (One-to-Many):** An entity in A can relate to multiple entities in B, but each entity in B relates to only one entity in A.
   * **M:M (Many-to-Many):** Entities in A can relate to multiple entities in B, and vice versa.
2. **Normal Representation of Relations**
   * For **1:1** or **M:1**, place the foreign key of A in B (or vice versa, depending on the relation).
   * For **M:M**, use a mapping table with foreign keys from both entities.

**Sparse Relations**

* A **sparse relation** occurs when **most entities in one table do not have a corresponding entity in the related table**.
* Example:
  + **Student-Batch Relation:** Not all students are associated with a batch (e.g., free students).

A table with numbers and letters

Description automatically generated

* + Out of 10,000 students, only 1,000 may belong to a batch, leading to 9,000 null values in the batch\_id column.

**Problems with Sparse Relations**

* **Storage Waste:**
  + Columns with many nulls occupy unnecessary storage space (e.g., batch\_id column with 9,000 nulls wastes 36,000 bytes if each null requires 4 bytes).
* **Practicality:**
  + Nulls may not provide meaningful information and complicate data processing.

**Solution for Sparse Relations**

* **Use a Mapping Table:**
  + A table with numbers and letters

    Description automatically generatedFor any sparse relation, regardless of cardinality, create a separate table to represent the relationship.
  + Example for Student-Batch Relation:
    - students table:
      * Columns: Id, Name, PSP

A white box with black text

Description automatically generated

* Batches table:
  + Columns: Id, Name
* A white rectangular box with black text

  Description automatically generatedMapping table student\_batch:
  + Columns: student\_id, batch\_id
  + Only include rows in the mapping table for students who are related to a batch, eliminating nulls.

**Generalized Steps for Sparse Relations**

1. **Identify Sparsity:**
   * Determine if the relation is sparse by assessing the percentage of entities with no corresponding relation (typically >50%, but case-specific).
2. **Replace Foreign Key Columns with Mapping Table:**
   * Remove the direct foreign key column from the primary table.
   * Introduce a mapping table with foreign keys from both entities.
3. **Advantages of Mapping Table:**
   * Saves storage space by avoiding null values.
   * Improves clarity and efficiency in database design.

**Examples**

**Student-Batch Relation (1:M)**

* **Sparse Relation:** Not all students belong to a batch.
* **Solution:**
  + Mapping table student\_batch with only relevant rows.

**Husband-Wife Relation (1:1)**

* **Sparse Relation:** Not all men are married.
* **Solution:**
  + Mapping table husband\_wife with husband\_id and wife\_id columns.

**Key Takeaways**

1. Sparse relations are specific to scenarios where **many entities in one table are not related to any entity in another table**.
2. For sparse relations:
   * **Replace foreign key columns with mapping tables** to save storage.
   * The problem is not nulls but wasted storage.
3. This design principle applies regardless of the cardinality of the relation.

## Relations and Their Attributes in Schema Design

**1. Introduction to Relations with Attributes:**

* Relations between two entities can have associated information (attributes).
* Example:
  + Facebook friendship relation:
    - Date of friendship (e.g., "friends since date").
  + Husband-wife relation:
    - Date of marriage, date of engagement, venue, etc.
* These attributes provide additional context or metadata about the relation.

**2. Dilution of Table Purpose:**

* Storing relation attributes directly in one entity's table (e.g., husband's table) dilutes the table's purpose.
  + Example:
    - Adding date\_of\_marriage and venue to the husband's table mixes personal data with relation-specific data.
  + The table's design becomes inconsistent and harder to maintain.

**3. Solution: Mapping Table for Relations with Attributes:**

* A separate mapping table should be created to store the relationship and its attributes.
* Benefits:
  + Avoids diluting the purpose of individual entity tables.
  + Centralizes relation-specific information, making it easier to query and manage.

A screenshot of a table

Description automatically generated**4. Examples:**

**a) Husband-Wife Relation:**

* **Husband Table**:
  + husband\_id, name

A table with names and a couple of names

Description automatically generated with medium confidence

* **Wife Table**:
  + wife\_id, name
* **Husband-Wife Mapping Table**:
  + husband\_id, wife\_id, date\_of\_marriage, date\_of\_engagement, venue

A white rectangular table with black text and numbers

Description automatically generated

**b) Student-Batch Relation:**A screenshot of a computer

Description automatically generated

* **Student Table**:
  + student\_id, name, psp
* **Batch Table**:
  + batch\_id, name
* **Student-Batch Mapping Table**:
  + student\_id, batch\_id, date\_of\_joining

**5. Guidelines for Designing Relations with Attributes:**

* **Always use a mapping table for relations with attributes.**
  + Regardless of the cardinality (1:1, 1:M, M:M).
* **Attributes are stored in the mapping table.**
  + Ensures clear separation of entity data and relationship data.

**Summary**

A white and blue box with black text

Description automatically generated

## Mapping Tables and Keys

**Primary Key in Mapping Tables**

* A **mapping table** typically has a composite key.
  + Example: For a Student-Batch Mapping Table, the primary key would be both student\_id and batch\_id.
  + Example: For a Husband-Wife Mapping Table, the primary key would be husband\_id and wife\_id.
* **Composite Key**:
  + A primary key composed of more than one column is called a **composite key**.
  + Example: (student\_id, batch\_id) or (husband\_id, wife\_id).

**When to Use Composite Key vs. Separate ID**

1. **Composite Key**:
   * **Advantages**:
     + Data is **sorted by the first column** of the composite key (e.g., sorted by husband\_id in (husband\_id, wife\_id)).
     + Queries like SELECT \* WHERE husband\_id = X are faster.
   * **Disadvantages**:
     + Cannot represent relations between mapping tables effectively.
     + Limited flexibility for scenarios where relations need unique identifiers.
2. **Separate ID**:
   * **Advantages**:
     + Can represent relations between mapping tables or entities related to the mapping (e.g., a Child table linked to a specific Husband-Wife relation).
   * **Disadvantages**:
     + Queries may be slower since the table is not sorted by the first column of the composite key by default.
     + Requires additional storage for the unique ID column.

**Sorting in Tables**

* **By Composite Key**:
  + Tables are sorted by the **first column** of the composite key.
  + If there are ties, the second column in the key is used for sorting.
  + Example: In (husband\_id, wife\_id), rows are sorted by husband\_id, and if two rows have the same husband\_id, they are further sorted by wife\_id.
* **By Separate ID**:
  + The table is sorted by the unique ID, not by husband\_id or wife\_id.
  + This can lead to less intuitive ordering but supports greater flexibility for linking relations.

**Why Choose a Separate ID?**

* **Scenarios with Relations Between Mapping Tables**:
  + Example: A Child table linked to specific marriages.
    - Child table would have a foreign key pointing to the unique ID in the Husband-Wife Mapping Table.

**Key Design Insights**

1. Use **composite keys** when:
   * Sorting and query speed for specific columns is a priority.
   * There are no additional relations dependent on the mapping table.
2. Use **separate ID** when:
   * Relations between mapping tables or other entities exist.
   * A unique identifier is needed for flexibility.

**Example: Sorting Behavior**

* **With Composite Key (husband\_id, wife\_id)**:
  + Example rows…

A table with numbers and text

Description automatically generated

* + Sorting:
* Rows are sorted by husband\_id first, then wife\_id for tie-breaking.
* **With Separate ID**:
  + Example rows…

A table with numbers and text

Description automatically generated

* + Sorting:
* Rows are sorted by the unique id column.

**Summary**

* Mapping tables can use either a **composite key** or a **separate ID** based on use case requirements.
* Composite keys prioritize query speed and sorting by specific columns.
* Separate IDs prioritize flexibility, particularly for relationships involving mapping tables.

Internally, databases sort tables by the primary key (either composite or separate).