DB Design Guidelines

- Learning Materials
- RDBMS Best Practices

Learning Materials

- Next Level Database Techniques for Developers

RDBMS Best Practices

There are 2 aspects that we need to be cognisant of while designing any relational database:

- Data storage
- Data retrieval

Both aspects have their own best practices to ensure the database is designed in a way that is most efficient to not only store but also retrieve data.

Data Storage:

- Data storage is the primary purpose of any database, however, **WHAT data we need to store** in it is the question that we need to answer every time while creating a table or column.
- **What to store:** As a general guideline, **only data that NEEDS to be persisted across user sessions and across user activities should be stored**. Any data which is relevant only for the user's current session or current activity need not be stored in the database. A good example for this is storing user preference of UI theme. In most cases, we don't want to store the user's preferred theme in the db but only keep it on the frontend for the user to experience either "Light Mode" or "Dark Mode". On the other hand, something like user's transactions or interactions with other users are a very important data point to be stored in the db.
- **File storage:** No files should ever be stored in the db. It's a very common malpractice to store user's profile picture/thumbnail in the db using a BLOB data type column. This is very vicious and can lead to database overloading very very fast. Hence **file storage should ALWAYS be kept separate from the database.**
- **Data types:** Deciding data types for columns is also a very important consideration that needs to be given due thought when designing a database. A very bad practice is to keep every column as a TEXT column or a VARCHAR regardless of what data is going to be stored in that column. If you know it's going to be an ID that will be saved in that column then why would you set it as a VARCHAR or TEXT? If you know its going to be a simple status flag which will only have values as 0 or 1, then why would you keep data type for that column as TEXT and not an ENUM or TINYINT?
- In depth knowledge of what the most common data types are, what each of them is meant to do is very important. Implication of assigning a data type to a column is also something that needs to be given due diligence, when the table grows and adds millions of records a wrong data type can create all sorts of chaos. You can learn more about it here.
- **Splitting relevant data:** Another important consideration while designing a database structure is which data point should we keep in just 1 table and reference using a JOIN or sub query as required. For example, if an application has a detailed

user profile feature then it might be a good idea to keep user profile data in a separate table from user authentication data. What this will do is, since user authentication table will be queried way more frequently that user profile, the user profile queries will not be blocked because the authentication table is locked. Similarly, a one-to-many relationship often requires an ID to stored as a foreign key in another table, in such cases ensure that you don't store any other data points from the first table in the second table.

- **Data redundancy:** Maintaining **data integrity** is also a very important aspect of storing data properly in a database. Whenever a record is getting deleted, due thought is to be given to identify if there are any child entities to that record. If there are, then a decision needs to be made as to whether we should delete all child entities or if we need any of those entities then should we do a soft delete of the parent entity. **There should never be orphan data in the database. If you are deleting an entity, make sure all related entries are deleted as well and if you are soft deleting an entity, make sure all other related entries are soft deleted as well.**

Data retrieval:

- **Indexing:** Indexes improve query performance by allowing the database to quickly locate rows based on the indexed columns. To decide which columns to index, check the WHERE and JOIN clauses in your queries. Indexes are used mainly for filtering, hence the columns which we are using in WHERE and JOIN clauses need to be indexed. Also, in the case of JOIN, ensure you keep data type and length same for both the columns that are being joined otherwise index will not work. Indexes also don't work on TEXT data type columns so avoid using TEXT as data type unless the column is going to hold a string that is user generated and we cannot be certain about its max length.
- **Write Efficient Queries:** Select only necessary columns in a query avoid using SELECT * as a default query practice. use filtering conditions to only retrieve data set that is required. Avoid unnecessary joins as they are very resource intensive and can cause multiple tables to lock until queries are resolved. As a rule of thumb, if you are going to join 2 tables, always join them on columns that are indexed and have the EXACT same data type, that way the performance will be much better.
- **Avoid SELECT DISTINCT**: Minimize resource usage by using filtering conditions instead of DISTINCT. SELECT DISTINCT is very resource intensive and its recommended to avoid using it.
- **Limit Result Set:** Never return huge chunks of data in any query, especially from large tables. Use pagination (LIMIT/OFFSET) for manageable data chunks to be retrieved and sent to the frontend.
- **Cleanup Indexes:** Regularly monitor and maintain index health. During the course of development, its possible that new columns get added to a table which replace a certain older column in the WHERE clause. In such cases, it's important to clean up indexes on such columns as unused indexes will hamper WRITE performance of the database.
- **Secure Queries:** Prevent SQL injection with parameterized queries or prepared statements. Always write queries using query builders so as to protect the

application against SQL injection attacks. For complex queries where query builders are not useful, ensure the input is sanitised and that the input does not contain any HTML tags or SQL syntax before being passed into the queries.

- **Optimize Network Usage:** Minimize data transferred between server and application. This best practice is particularly useful when there are bulk operations that need to be performed on the database. We should try to minimize the number of times our application connects to the database. For example, if you want to insert 100 records in the database instead of writing insert query in a loop and connecting to the database 100 times, you can bulk insert 100 records using the bulk insert syntax and connect to the database only once.