A. Database System

a. Data models

Data models are essential ideas in database administration and design, particularly the Entity-Relationship (ER) and relational models.

1. Entity-Relationship (ER) Model:

   Real-world entities, their characteristics, and the connections between them are represented by the ER model, a high-level, conceptual data model. The ER model acts as a guide for database design, assisting in the comprehension of information structure and data requirements prior to implementation. Entity-Relationship Diagrams (ERDs) are frequently used to illustrate it visually.

2. Relational Model:

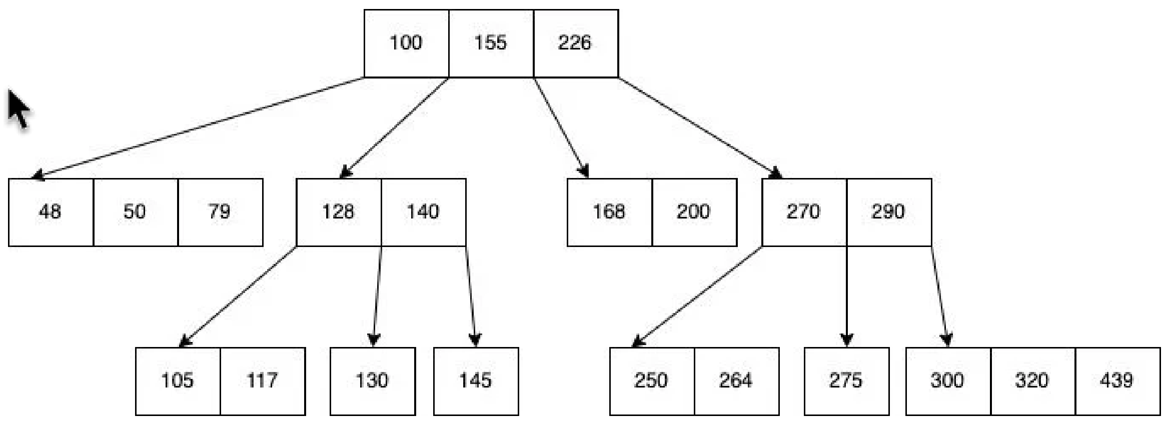
   Data is logically arranged into tables (relations) using the relational paradigm. The majority of contemporary database management systems (DBMS) are built on it. The useful framework for managing, organizing, and querying data in relational databases is provided by the relational model. It uses structured query language (SQL) to provide flexible data retrieval and stresses data integrity through the use of keys.

b. Indexing

For every database system to operate as efficiently as possible, database indexing is essential. Your database queries may become sluggish and ineffective without proper indexing, which could result in a bad user experience and lower productivity. We'll look at some best practices for building and utilizing database indexes in this post.

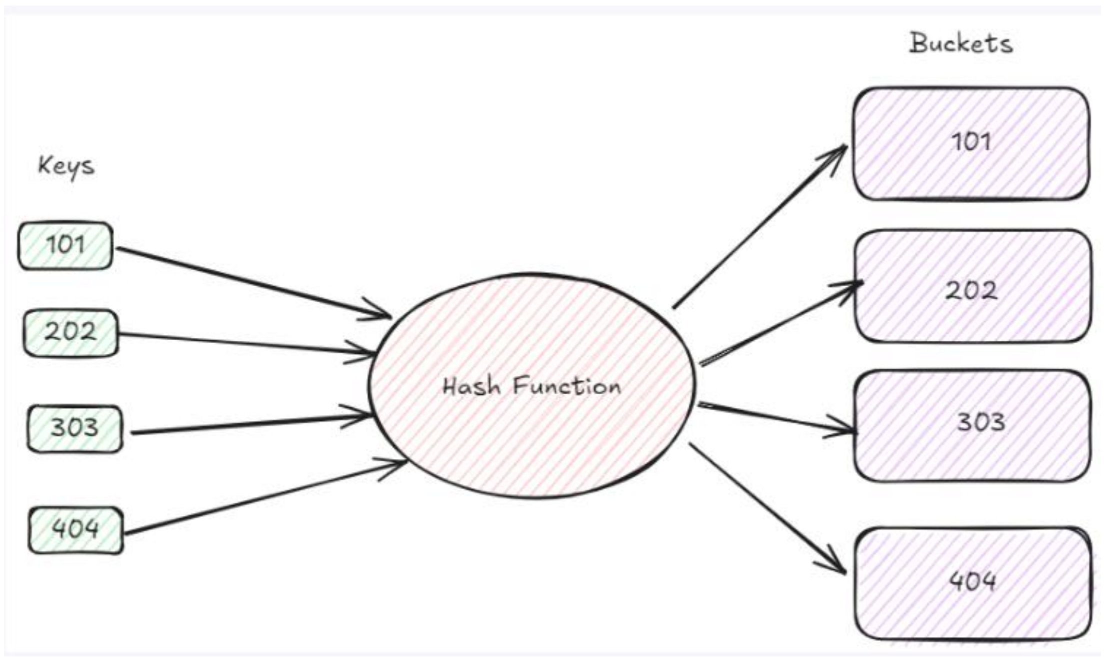
B-Tree Indexing:

Data structures that offer sorted data and enable sequential access, attachments, deletions, and searches in a sorted manner are known as B-trees. When it comes to systems that write big data blocks, the B-tree is incredibly powerful. Nodes with more than two children are permitted in the B-tree, which streamlines the binary search tree. A sample B-tree is shown below.



Hash Indexing:

A method called hash indexing is intended to improve search effectiveness, particularly in high-dimensional settings such as vector databases. It functions differently from B-Trees and is especially helpful for handling complicated and big datasets.



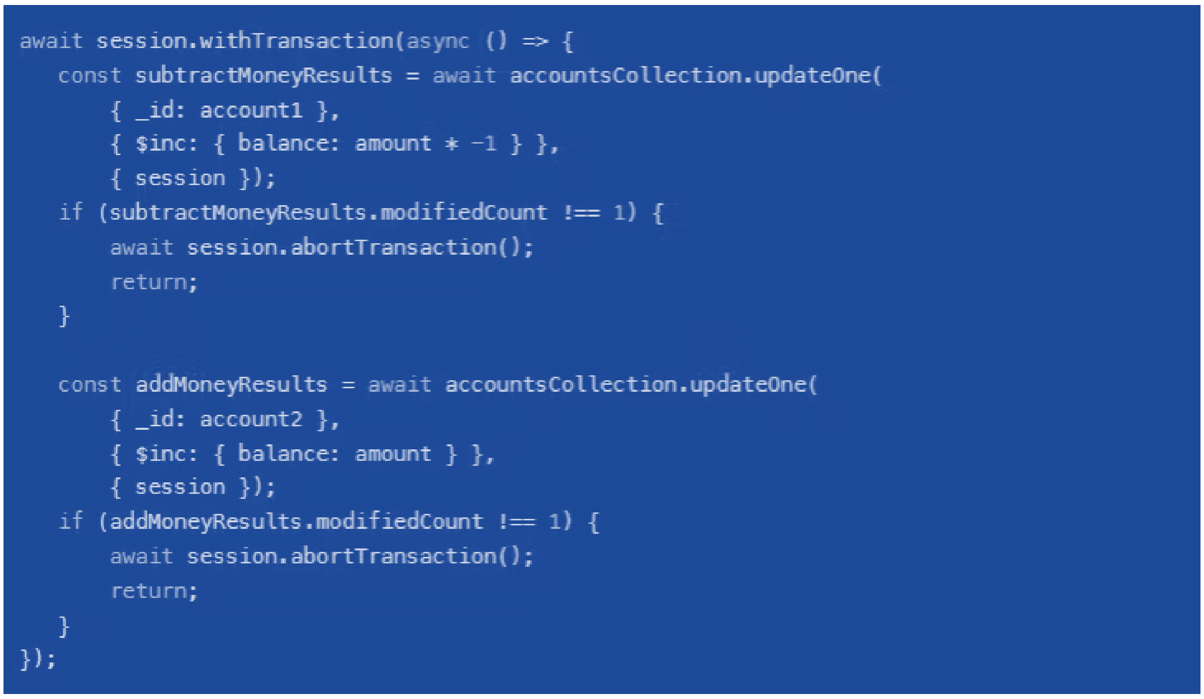
c. **Transaction ACID**

Atomicity: A transaction's result can be entirely successful or unsuccessful; if one component fails, the entire transaction must be rolled back;

Consistency: Transactions preserve integrity constraints by transferring the database between valid states;

Isolation: Concurrent transactions are isolated from one another, guaranteeing data accuracy; and

Durability: Once a transaction is committed, its changes are still in effect even in the case of a system failure.



```
await session.withTransaction(async () => {
   const subtractMoneyResults = await accountsCollection.updateOne(
       { _id: account1 },
       { $inc: { balance: amount * -1 } },
       { session });
   if (subtractMoneyResults.modifiedCount !== 1) {
       await session.abortTransaction();
       return;
   }

   const addMoneyResults = await accountsCollection.updateOne(
       { _id: account2 },
       { $inc: { balance: amount } },
       { session });
   if (addMoneyResults.modifiedCount !== 1) {
       await session.abortTransaction();
       return;
   }
});
```

B. Software Engineering