Jahong Liu

System Design Basics 14: SQL

SQL (Standard Query Language)

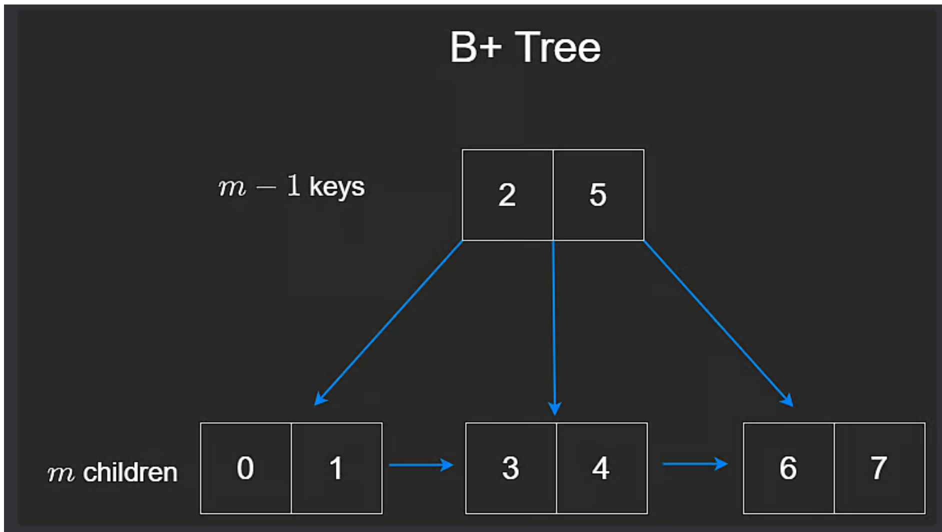
SQL Databases, also known as relational database management systems (RDBMS), are databases that store persistent data using tables. EDBMSs provide a way to structure data within disk so that reading from and writing to it is efficient. The data is formatted into precisely defined fields, so that it can be queried easily.

B+ Trees

What makes the queries efficient is the use of a data structure called B+ Tree. B+ Trees are multi-way trees that can have more than two children per node and store all their data in the leaf nodes which are linked in a sorted order. All the data is stored in the leaf nodes. This is different from a B-tree node, where the interior nodes contain only keys and pointers to other ndoes.

Furthermore, each node has at most M children. If a nodoe has M children, it must be the case that it has M-1 keys. For example, if the root node contains the values, [2, 5], it would have three children. The keys act as a separation value which divides its subtrees. The first child of the node containing [2,5] would be, [0,1], second being [3, 4] and the third being [6,7]. Notice that the first child contains all of the values that are smaller than 2. The second child contains values in between 2 and 5. The third child contains values greater than 5.

Another reason why B+ trees are used is because they provide indexing. Indexing is a way to improve the speed of data retrieval operations on a database table, coming at the cost of additional writes and storage space to maintain the index data structure. To give a quick example, if we had a bunch of names that mapped to their respective phone numbers, we could pick the name field as the index, allowing us to retrieve phone numbers faster in the future. However, there are multiple downsides to indexing, which are beyond scope of this discussion.

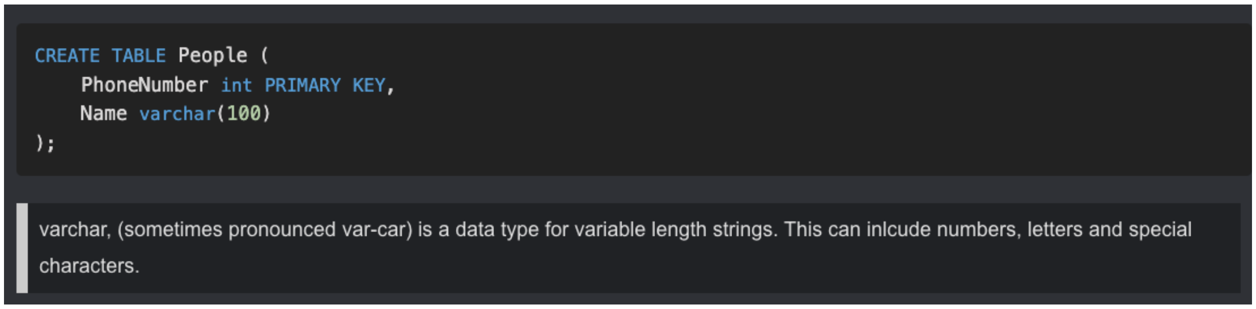


The visual above demonstrates a visualization of the B+ tree discussed above.

How data is stored

In an SQL database, data is stored inside tables. Tables are a way to organiza data where each row contains information about a single primary key. A primary key uniquely identifies each record, where a record is a row.

If we go back to the phone numbers example, let's say we want to uniquely identify each person by their phone number and store their name in a table called people. We must declare the structure of the table first before inserting any records. In SQL, it can be defined as the following:

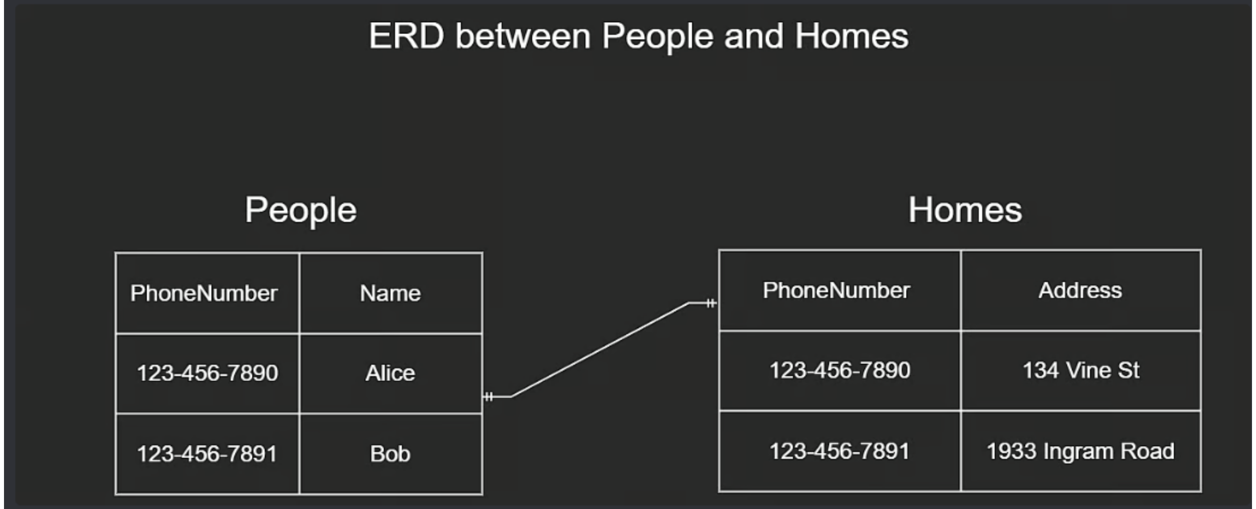


```sql
CREATE TABLE People (
    PhoneNumber int PRIMARY KEY,
    Name varchar(100)
);
```

varchar, (sometimes pronounced var-car) is a data type for variable length strings. This can inlcude numbers, letters and special characters.

If we wanted to have another table which would associate each PhoneNumber in our people table with an address, indicating that each person must have an address, we can do so in a table called HOMES. We want to ensure that no PhoneNumber that does not exist in People can be inserted into Homes. This is an example of a FOREIGN KEY constraint. This now means that each row in Homes table is associated with a row in the People table via the PhoneNumber.

```sql
CREATE TABLE Homes (
    PhoneNumber int,
    Address varchar(255),
    FOREIGN KEY (PhoneNumber) REFERENCES People(PhoneNumber)
);
```



People

PhoneNumber	Name
123-456-7890	Alice
123-456-7891	Bob

Homes

PhoneNumber	Address
123-456-7890	134 Vine St
123-456-7891	1933 Ingram Road

The above visual demonstrates the entity relationship diagram of the relation between `People` and `Homes`. Each person (as identified by a unique PhoneNumber) can have one home address, and each home (as identified by a unique PhoneNumber) can belong to one person.

Joins

If we wanted to retrieve the names and addresses of the people based on their phone numbers, we would need to perform a join of the two tables. Joins are a way to combine records from two or more tables, based on a related column between them. In this case, that column is PhoneNumber.

```sql
SELECT People.name, Homes.address
FROM People
JOIN Homes ON People.phone = Homes.phone;
```

Trade-offs of Relational Database Management Systems

Relational Databases follow what's called **ACID**. ACID is a property that stands for **Atomicity**, **Consistency**, **Isolation**, and **Durability** respectively. This acronym refers to four key properties of a transaction. A transaction, in the context of a database, refers to a sequence of one or more SQL operations that are treated as a unit.

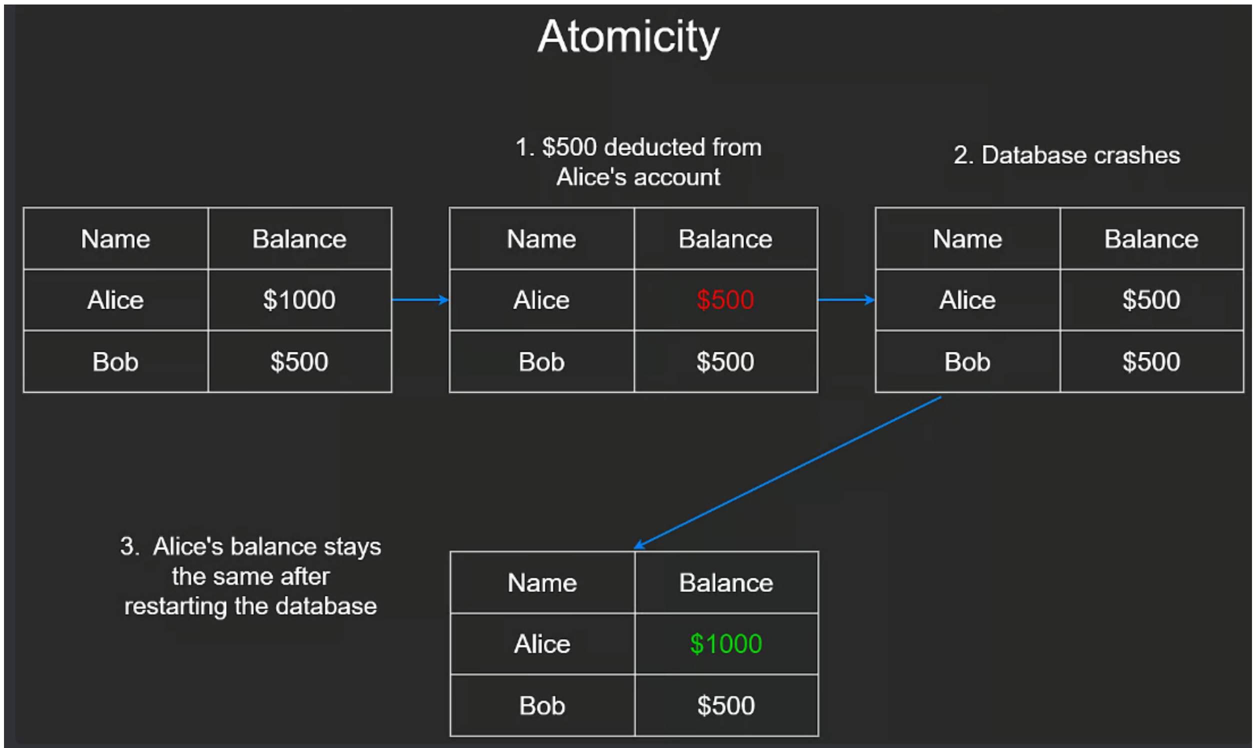
Durability

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, all transactions will be remembered. So, if the power goes out after a transaction occurs, the transaction will still be recorded in the database.

Atomicity

All changes to data are performed as if they are a single operation.

That is, all the changes are performed, or none of them are. When we have the begin and commit. If one of the steps fail in the transaction, it means that the entire transaction is nor committed. The transaction will only commit if all the operations were successful. We can explain this using an example. Given Alice and Bob who have 1000and500 in their bank accounts respectively, Alice chooses to send Bob $500. If the database crashes while the transfer is taking place, atomicity will ensure that money is not taken out of Alice's account. Again, either all the steps get done, or none of the steps get done. Otherwise, the money would be destroyed and disappear.


Atomicity
1. $500 deducted from Alice's account
2. Database crashes
3. Alice's balance stays the same after restarting the database

Name	Balance
Alice	$1000
Bob	$500

Name	Balance
Alice	$500
Bob	$500

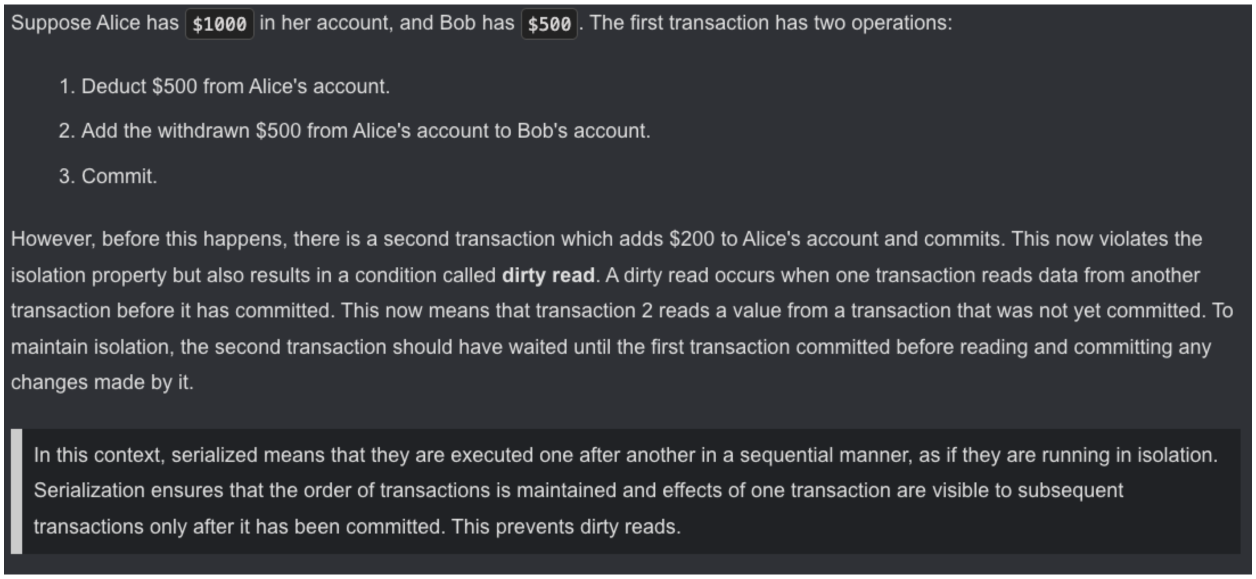
Name	Balance
Alice	$500
Bob	$500

Name	Balance
Alice	$1000
Bob	$500

The visual above demonstrates atomicity from the example discussed.

Isolation

Isolation refers to an intermediate state of a transaction, such that a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized. The isolation property ensures that the concurrent transactions do not interfere with each other and the effects of one transaction are isolated from other transactions until it is committed.



Suppose Alice has `$1000` in her account, and Bob has `$500`. The first transaction has two operations:

1. Deduct $500 from Alice's account.
2. Add the withdrawn $500 from Alice's account to Bob's account.
3. Commit.

However, before this happens, there is a second transaction which adds $200 to Alice's account and commits. This now violates the isolation property but also results in a condition called **dirty read**. A dirty read occurs when one transaction reads data from another transaction before it has committed. This now means that transaction 2 reads a value from a transaction that was not yet committed. To maintain isolation, the second transaction should have waited until the first transaction committed before reading and committing any changes made by it.

> In this context, serialized means that they are executed one after another in a sequential manner, as if they are running in isolation. Serialization ensures that the order of transactions is maintained and effects of one transaction are visible to subsequent transactions only after it has been committed. This prevents dirty reads.

Consistency

Consistency in the ACID model ensures data integrity. Simply put, consistency in databases refers to the adherence to predefined rules and constrains that maintain the validity of the data throughout the execution of multiple transactions. Therefore, it ensures that any transaction the database performs brings it from one valid state to another.

In databases, we have the ability to define what consistency constrains should be followed. In our example, the rule that an account balance can’t be negative is a consistency constraint defined by the database.