Relational Databases

Types of databases based on structure

1. Relational Databases
2. Non relational databases

Non Relational Database

Non relational database dont impose a tabular structure to the data stored in them Non Relational databases do have a structure but it is far more flexible. Eg in the payments table shown below adding an extra attribute (say Acc. No) would not be possible once the schema has been defined. So a new table has to be created but in non relational database keys (new columns) can be added with much ease.

Relational Databases

Data is stored in the form of tables. These tables represents an entity. The below table represents payments done by a customer.

Payments Table

customer-name	processed-at	amount
praveen	12/04/1999	20
naveen	13/03/1995	30
panda	14/12/1996	15

each row in a table is known as an record

each column in the table is known as an attribute

All the table in a relational database has a defined schema (specific set of rules / plans on how the data should be stored)

SQL Database is often synonymously used with relational database

Most relational databases support SQL (Structured Query language). SQL is a programming language used to perform complex queries on the database. Eg: PostgreSQL (popular relational database)

Why Python / JavaScript cannot be used to query the database?

When dealing with large scale distributed system

(there will be tera-bytes of data) and if we had written a python / JS script to query the data all the data has to be loaded in the memory for the script to process that data (this is borderline impossible). SQL allows to perform the queries directly in the database without having to load the data into the memory.

ACID Transactions ⇒ SQL Databases must use ACID Transaction

An ACID transaction is an operation in the database that has the properties mentioned below

A → Atomicity

The operation that constitutes the transaction will either all succeed or all fail. There is no in-between state

Eg: Database transaction to transfer funds from one bank account to another bank account This database transaction consists of two operations

→ Detecting funds from one bank account

→ Increasing funds in another bank account.

The atomicity of a transaction dictates that if a database transaction consists of more than one operation if any one of the operations fail then the entire database transaction will fail.

C ⇒ Consistency

Consistency ensures that the database is never in a half completed state. whatever happens in the middle of a transaction

If there is an error in the transaction or there is a system failure then all the changes that have been made will be rolled back automatically ie. the database will be restored to the state it had before the transaction Eg In the Database transaction to transfer funds example if money has been debited from one bank account but not has been credited to another bank account due to network error in this case money will be

credited back to the account from which it has been debited.

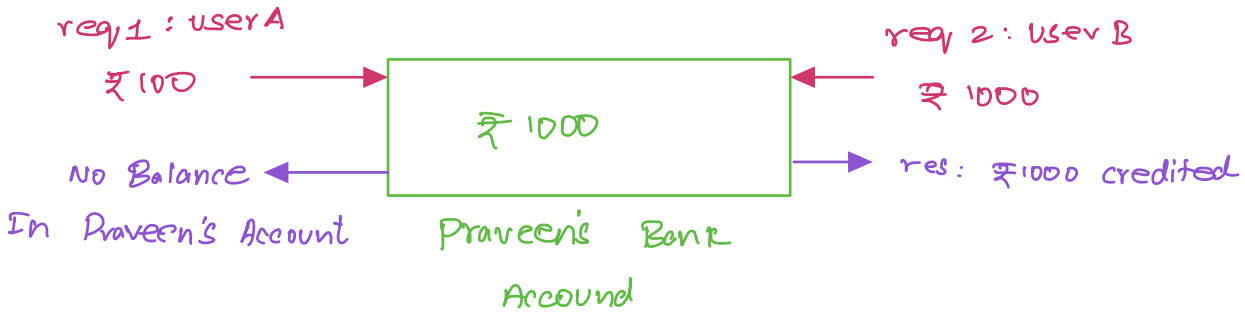
I ⇒ Isolation

The execution of multiple database transactions Concurrently / Parallely will have the same effect as if they had been executed sequentially

Eg: Users A and B are trying to debit money from a bank account (say Praveen's Bank acc)

If user-A sends request 1ST his req. will be processed while the user-B's req will be put on hold

*Both the users are sending the request at the time.



The database randomly picks the request (in this case let's assume the database picks req 2) it will perform the operation (for req 2) then it will proceed with req 1

D ⇒ Durability

Any committed transaction is written to the disk → non volatile storage and not to the memory.

Database Index → mostly an index is created on the columns which are specified in the "WHERE" clause of a query.

An index is a data structure that organizes data records on the disk in a way that makes retrieval of data efficient.

In practice an index is created on one or multiple columns in the database to greatly speed up the read queries that run very often

Eg: This is a table which holds the amount paid by customers Assume that this table is queried a lot of times based on the amount. Eg: Return customer

Index for the amount column

amount	pointer
200	
300	
950	
9000	

This index can

customer_name	processed_at	amount
praveen	01/12/2021	200
Rekha	01/01/2022	9000
Santy	06/04/2022	300
Sai	11/05/2022	950

be made up of a Binary Tree

names who have paid above 800. This operation takes O(N) but with an index the operation can be brought down to O(log N) (since index in this case is made up of Binary Tree)

Having an index for multiple columns comes with the downside of

① Slightly longer writes to the database, since the writes should also take place in the relevant index / indexes

② Each extra index requires additional storage space

③ Indexes on non-primary keys might have to be changed on updates whereas an index on the primary key might not (as updates don't modify the primary key)

④ Creation of indexes takes some time.

SQL Database :

Any database that supports SQL. This is often synonymously used with Relational Database though not every relational database supports SQL.