Database

17 December 2021 09:29

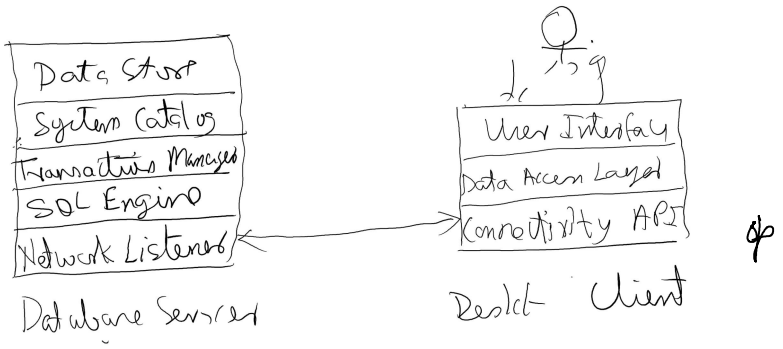
Database System: It is a software which manages persistence of structured data and provides an API for consuming this data. A database system generally handles storage and sharing of

1. **Relational Data** - Each part of data is logically represented by a *table* consisting of *rows* and *columns* with following characteristics:
   Schema - the structure and type of data stored in the rows of a table is specified by the definition of the columns of that table.
   Constraint - the column definition can indicate the restrictions on the data that can be stored in the rows of the table to maintain the integrity of that data.
   Relationship - one or more columns of one (child) table can reference columns of another (parent) table so that the data stored in those columns of first table match with the data stored in the referenced columns of second table

2. **Transactional Data** - Different parts of data can be updated within a single unit of work known as *transaction* which supports *commit* operation for persisting the new state of data and *rollback* operation to restore data to its old state with following (ACID) characteristics:
   Atomic - a transaction does not perform any update whose effect on data cannot be cancelled by the rollback operation.
   Consistent - a transaction executes rollback operation as soon as any update of data violates the integrity of data.
   Isolated - a transaction does not allow any operation executing outside of its scope to access the data it has updated until it ends.
   Durable - a transaction always ends with either commit operation or rollback operation.

Business Application: It is a software that automates a certain business task by processing transactional data using a set of pre-defined rules known as business logic. A business application is characterized by its *maintainability* which indicates how quickly it can be adapted to any change in its business logic and by its *scalability* which indicates how quickly it can be adapted to a significant increase int the number of its users.
A business application is generally designed using *client-server architecture* in which its implementation is divided to execute as two separate but interacting processes with following responsibilities:
Backend - It manages persistence of transactional data for the application and may optionally implement its business logic for increasing its maintainability. Backend is commonly supported using database system installed on a central machine.
Frontend - It provides the user-interface for the application and may optionally implement its business logic for increasing its scalability.

Frontend is separately installed on each machine where it is required.

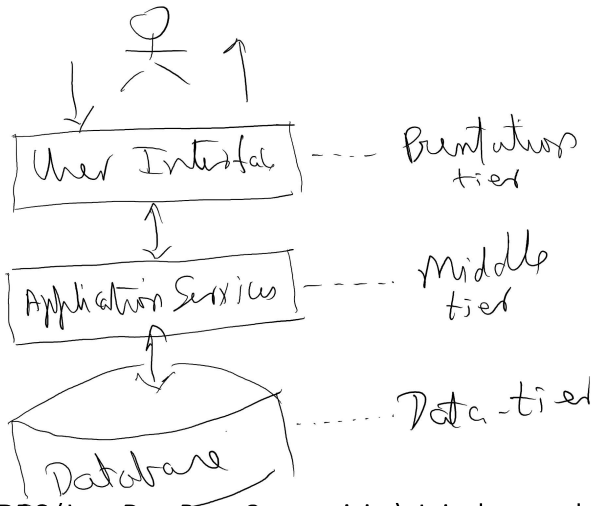


Enterprise Application: It is a business application which pro cesses large volumes of transactional data using highly complex business logic and is concurrently accessed by large number of users with a very low tolerance for downtimes. Such an application requires high maintainability as well as high scalability and as such it is designed using *N-tier architecture* in which its implementation is divided to execute as multiple separate but interaction processes with following responsibilities:

Data Tier - It manages persistence of transactional data for the application. Data-tier is generally supported using one or more database systems but can also include flat-file based legacy programs and ERPs.

Middle Tier - It implements business logic for the application on top of the data-tier. Middle tier is generally designed using *service oriented architecture* (SOA) in which its implementation is divided into autonomous sets of operations called services each of which only shares the description of the operations it implements and these operations can only be consumed by sending messages to the process hosting that service which can execute on its own machine.

Presentation Tier - It provides the user-interface for the application on top of the middle-tier. Presentation-tier is commonly supported as separately installable *rich client* or a web-browser based *thin-client*.



JDBC (~~Java Data~~Base Connectivity)**:** It is the standard Java API (available

in java.sql package) for consuming relational data managed by a database system using SQL. The implementations of interfaces specified by JDBC for a particular database system is called its JDBC driver and it includes support for following objects:

1. **Connection** - it opens a communication session with the database system using the information specified in its URL.
2. **Statement** - it dispatches SQL commands to the database system using its Connection object.
3. **ResultSet** - it fetches the rows resulting from execution of a query (SELECT) command dispatched by the Statement object.

JPA (Java Persistence API)**:** It specifies standard support (through javax.persistence package) for accessing relational data using instances of Java classes which are unaware of their persistence context (source of data). The implementation of JPA (such as *EclipseLink* and *Hibernate* which are built on top of JDBC) is called the JPA Provider and it includes support for the following

1. **Entity** - It is a serializable *plain old Java object* (POJO) containing at least one *identity* field (for uniquely identifying such objects) and whose class and properties can be respectively mapped (through annotations of META-INF/orm.xml) to a relational database table and its columns.
2. **Entity Manager** - It loads entities from rows of their mapped database tables (using *Java persistence query language* - JPQL) and handles persistence of such entities in those tables in a transactional manner.
3. **Entity Manager Factory** - It creates the entity manager from its *persistence unit* (configured in META-INF/persistence.xml) which contains information required for identifying and initializing the JPA provider.

RMI (Remote Method Invocation)**:** It is programming support offered by the Java runtime for invoking methods exposed by a Java object activated within a JVM executing on a remote machine across the network. It provides following mechanisms for building pure Java distributed systems:

1. **Object Export** - It binds a Java object to a network socket so that it can handle method invocation requests from its remote client. Under RMI only a remote object i.e an object whose class implements at least one remote interface (extending java.rmi.Remote) can be exported over a TCP/IP endpoint (using java.rmi.server.UnicastRemoteObject).
2. **Location Transparency** - It allows a client to invoke a method of a remote object in a manner which is identical to invoking a method of a local object (using dot operator). The client actually calls a remote method on a stub (proxy) object whose class implements all the remote interfaces supported by the target exported remote object to transport the invocation request to that object.
3. **Initial Bootstrapping** - It allows a client to obtain the stub of a remote object it needs to consume. The server process binds the endpoint information of its exported remote object to a well-known name within local RMI registry (a TCP/IP based name service - default port 1099) so that the client can lookup for such name in

this registry from a remote location and initialize the stub from the information bound to that name.