Database

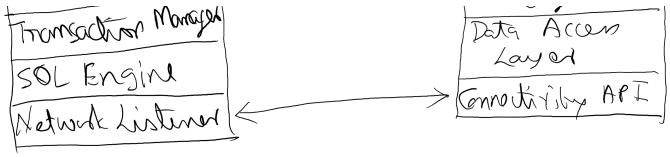
28 April 2022 09:37

Database System: It is a software that manages persistence of structured data and provides an API for consuming this data. A database system commonly provides support for storing and sharing data with following features

1. **Relational** - Each part of data is stored in a logical table with following characteristics
   (a) **Schema** - The definitions of columns of a table specify the type of data stored in the rows of that table.
   (b) **Constraints** - The definitions of columns of table indicate the restrictions on the data that can be stored in the rows of that table.
   (c) **Relationship** - Columns of one *child* table can reference columns of another *parent* table so that these columns in child table can only contain data which matches with the data in the referenced columns of parent table.
2. **Transactional** - Updates to different parts of data can be grouped together in a unit of work called a *transaction* which supports *commit* operation to persist the new state of data and *rollback* operation to restore data to its original state. A transaction has following (ACID) properties
   (a) **Atomic** - A transaction only performs an update whose effect can be cancelled by the rollback operation.
   (b) **Consistent** - A transaction performs a rollback as soon as any of its updates violates the integrity of the target data.
   (c) **Isolated** - A transaction does not allow a concurrent operation running outside of its scope to change its updated data until it ends.
   (d) **Durable** - A transaction always ends with a commit or a rollback operation.

Business Application: It is a software that automates a particular business task by processing some transactional data using a predefined set of rules known as the business logic. It is generally designed using *client-server architecture* in which its implementation is divided to execute as two separate but communicating processes with following responsibilities:

1. **Backend** - It manages persistence of transactional data for the application and can optionally implement business logic for increasing the maintainability of that application. It supported using a database server which is installed on a central machine on the network.
2. **Frontend** - It provides a user interface for the application and can optionally implement business logic for increasing the scalability of that application. It is supported using a client program which can be installed on multiple machines on the network.



Enterprise Application: It is a business application which processes large volumes of transactional data using highly complex business logic and is concurrently accessed by large number of users with very low tolerance for down times. Since such an application requires high maintainability as well as high scalability it is designed using *N-tier* architecture in which its implementation is divided to execute as three or more separate but communicating processes with following responsibilities:
Data Tier - It manages persistence of transactional data for the application. It is commonly supported using one or more database servers but may also include ERPs and legacy applications which persist data in flat files.
Middle Tier - It implements business logic for the application on top of the data-tier. It is commonly designed using *service oriented architecture* (SOA) in which its implementation is divided into (small) autonomous sets of operations called (*micro*)*services* each of which only shares the description of its operations and such operations can only be consumed by sending messages to the (isolated) host responsible for publishing it on a well-known network endpoint.
Presentation Tier - It provides a user-interface for the application on top of the middle-tier. It is commonly supported using separately installable *rich clients* or web-browser based *thin clients*.

Java Database Connectivity (JDBC)**:** It is a standard Java API (available through java.sql package) for consuming data provided by a relational database system using SQL. The implementation of this API 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 from the information provided in a URL.
2. **Statement** - It executes SQL command on the database system using the Connection object.
3. **ResultSet** - It fetches rows resulting from an execution of query (SELECT) command using the statement object.

Java Persistence API (JPA)**:** It specifies a standard support (through javax.sql package) for accessing relational data using instances of Java classes. An implementation of this API (like EclipseLink or Hibernate) is called JPA provider and it includes support for

1. **Entity** - It is a *serializable plain old Java object* (POJO) with one or more *identity* fields and whose class is mapped to a database table and whose fields are mapped to columns of that table (through META-INF/orm.xml or annotations).
2. **EntityManager** - It loads entities (specified using Java persistence query language -JPQL) from the rows of their mapped tables and

handles persistence of these entities into those tables in a transactional manner.

3. **EntityManagerFactory** - It creates an entity manager from its *persistence unit* (configured in META-INF/persistence.xml) which contains the information required by the JPA provider for connecting to database system and mapping its tables to entities.

Remote Method Invocation (RMI)**:** It is the programming support offered by Java runtime which enables a Java program running within one JVM to invoke methods exposed by an object activated in a remote JVM across the network. It includes support for:

1. **Remote Object** - It is a Java object whose class implements at least one remote interface (which extends java.rmi.Remote). When a remote object is exported (through java.rmi.server.UnicastRemoteObject) it is bound to a TCP/IP listener socket so that it can receive method invocation requests from remote endpoints.
2. **Remote Stub** - It is a proxy of an exported remote object whose (runtime generated) class implements all the remote interfaces implemented by the class of that remote object. When a method is called on an exported remote object's stub the corresponding invocation request is transported over a TCP/IP connection to the same method implemented by that remote object.
3. **RMI Registry** - It is a TCP/IP based name service which supports discovery of remote objects exported on its host machine. The server program binds the information required by the stub of its exported remote object to a well-known name within the local RMI registry so that a client program can look-up for this name from a remote location and acquire the stub of the remote object bound to that name.