Android Development with Kotlin

DBMS

Database Management Systems (DBMS)

Database Management System (DBMS) is a software for storing and retrieving users' data while considering appropriate security measures. It consists of a group of programs which manipulate the database. The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data. In large systems, a DBMS helps users and other third-party software to store and retrieve data.

DBMS allows users to create their own databases as per their requirement. The term "DBMS" includes the user of the database and other application programs. It provides an interface between the data and the software application.

DBMS Architecture

Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: 2-tier architecture and 3-tier architecture.

1-Tier Architecture

- In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.
- Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.
- The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: ODBC, JDBC are used.
- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

Relational Database

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

- **Tables** − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.
- **Tuple** − A single row of a table, which contains a single record for that relation is called a tuple.
- **Relation instance** − A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.
- **Relation schema** − A relation schema describes the relation name (table name), attributes, and their names.
- **Relation key** − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.
- **Attribute domain** − Every attribute has some pre-defined value scope, known as attribute domain.

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called key for that relation. If there are more than one such minimal subsets, these are called candidate keys.

Key constraints force that −

- *in a relation with a key attribute, no two tuples can have identical values for key attributes.*
- *a key attribute can* ***not have NULL values****.*

Key constraints are also referred to as ***Entity Constraints***.

Type of Keys

- **Primary Key** – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.
- **Super Key** – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table. instrumented tests are run on an emulator or an Android device using the Android framework.
- **Candidate Key** – A super key with no redundant attribute is known as candidate key
- **Alternate Key** – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
- **Composite Key** – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.
- **Foreign Key** – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Functional Dependency

A functional dependency (FD) is a relationship between two attributes, typically between the PK and other non-key attributes within a table. For any relation R, attribute Y is functionally dependent on attribute X (usually the PK), if for every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below

X ———> Y

The left side of the above FD diagram is called the determinant, and the right side is the dependent.

Redundancy

Redundancy means having multiple copies of same data in the database. This problem arises when a database is not normalized. Suppose a table of student details attributes are: student Id, student name, college name, college rank, course opted.

Student_ID	Name	Contact	College	Course	Rank
100	Himanshu	7300934851	GEU	Btech	1
101	Ankit	7900734858	GEU	Btech	1
102	Aysuh	7300936759	GEU	Btech	1
103	Ravi	7300901556	GEU	Btech	1

As it can be observed that values of attribute college name, college rank, course is being repeated which can lead to problems. Problems caused due to redundancy are: Insertion anomaly, Deletion anomaly, and Updation anomaly.

Normalization

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Types of Normal Forms have been discussed below:

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.

4. And the order in which data is stored, does not matter.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,

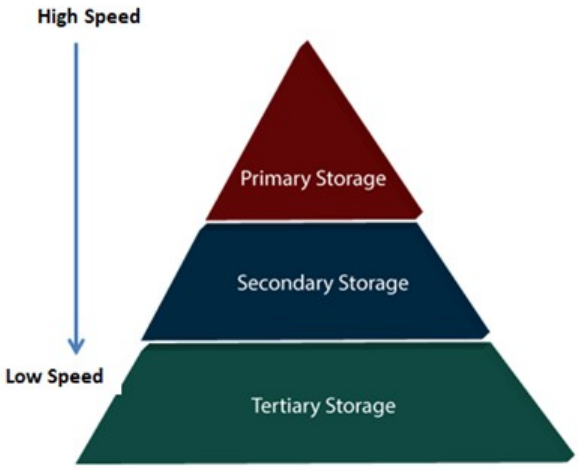
1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

Data Storage

A database system provides an ultimate view of the stored data. However, data in the form of bits, bytes get stored in different storage devices. In this section, we will take an overview of various types of storage devices that are used for accessing and storing data.

For storing the data, there are different types of storage options available. These storage types differ from one another as per the speed and accessibility. There are the following types of storage devices used for storing the data:

- Primary Storage
- Secondary Storage
- Tertiary Storage



Database File Organisation

File Organization defines how file records are mapped onto disk blocks. We have four types of File Organization to organize file records −

- **Heap File Organization**

  When a file is created using Heap File Organization, the Operating System allocates memory area to that file without any further accounting details. File records can be

placed anywhere in that memory area. It is the responsibility of the software to manage the records. Heap File does not support any ordering, sequencing, or indexing on its own.

- **Sequential File Organization**

  Every file record contains a data field (attribute) to uniquely identify that record. In sequential file organization, records are placed in the file in some sequential order based on the unique key field or search key. Practically, it is not possible to store all the records sequentially in physical form.

- **Hash File Organization**

  Hash File Organization uses Hash function computation on some fields of the records. The output of the hash function determines the location of disk block where the records are to be placed.

- **Clustered File Organization**

  Clustered file organization is not considered good for large databases. In this mechanism, related records from one or more relations are kept in the same disk block, that is, the ordering of records is not based on primary key or search key.

- **B+ Tree**

  The B+ tree is a balanced binary search tree. It follows a multi-level index format. In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height. In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.