RDBMS & Data-Warehouse Hand-Book 1.0

Author: ALGAE SERVICES
Mail: AlgaeServices@gmail.com
Online Notes: http://tutorials.algaeservice.com/

SYLLABUS

RDBMS & Data Warehouse

Unit 1: Introduction to DBMS and RDBMS

1. Database and Its Consumers
2. DBMS and its Models
3. RDBMS
4. Data Design and its modeling

Unit 2: Data Modeling and Normalization

1. E-R Diagram
2. Normalization
3. Design Databases

Unit 3: SQL Language

1. Basic SQL
2. SQL commands and its types
3. SQL constraints
4. SQL Joins
5. SQL Aggregate Functions
6. SQL functions

Unit 4: Transaction

1. Transaction Concept
2. ACID Properties
3. Transaction State
4. Database Recovery
5. Transaction Log
6. Deadlocks
7. Locks
8. Isolation Levels

Unit 5: Data warehouse

1. Understanding a Data Warehouse
2. Data Warehouse Features
3. Types of Data Warehouse
4. Difference between OLAP and OLTP
5. OLAP Unit

Chapter 1
Introduction to DBMS and RDBMS

A Database is a collection of related data organized in a way that data can be easily accessed, managed and updated. And management system which manages database is called as DBMS

A DBMS is a software that allows creation, definition and manipulation of database. It is a tool used to perform any kind of operation on data in database. It also provides protection and security to database. Once you add relationship among database entities it becomes RDBMS

DBMS: Database Management System
RDBMS: Relational Database Management system

1.1 Database and Its Consumers

A Database is a collection of related data organized in a way that data can be easily accessed, managed and updated. It is a logical container where related piece of information is stored and various operations can be performed on it.

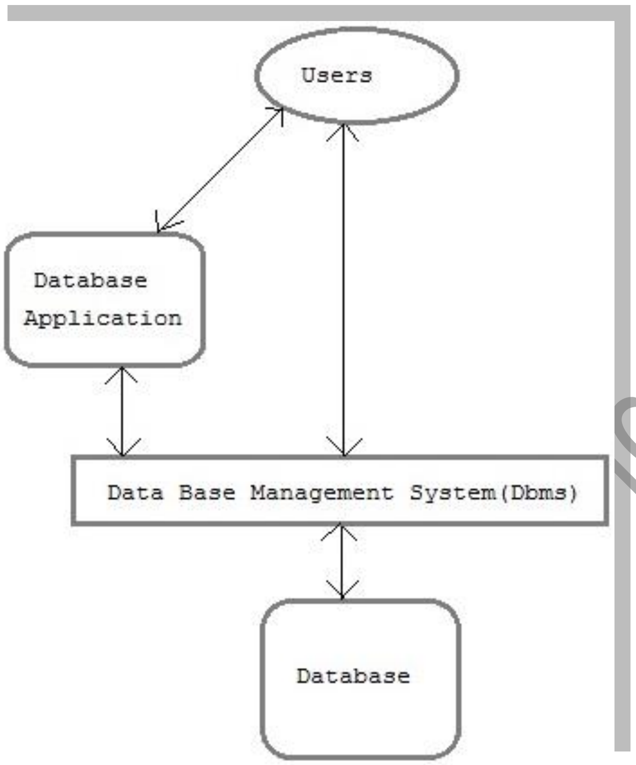
Example: name of your school.

Components of Database System:

There are 4 major components of database system:

1. Users: Users may be of various type such as DB administrator, System developer and End users.
2. Database application: Database application may be Personal, Departmental, Enterprise and Internal
3. DBMS: Software that allow users to define, create and manages database access, Ex: MySql, Oracle etc.

4. Database: Collection of logical data



1.2 DBMS and its Models

DBMS is defined as database management system. A DBMS is a software that allows creation, definition and manipulation of database. It is a tool used to perform any kind of operation on data in database. It provides protection and security to database and maintains data consistency in case of multiple users.

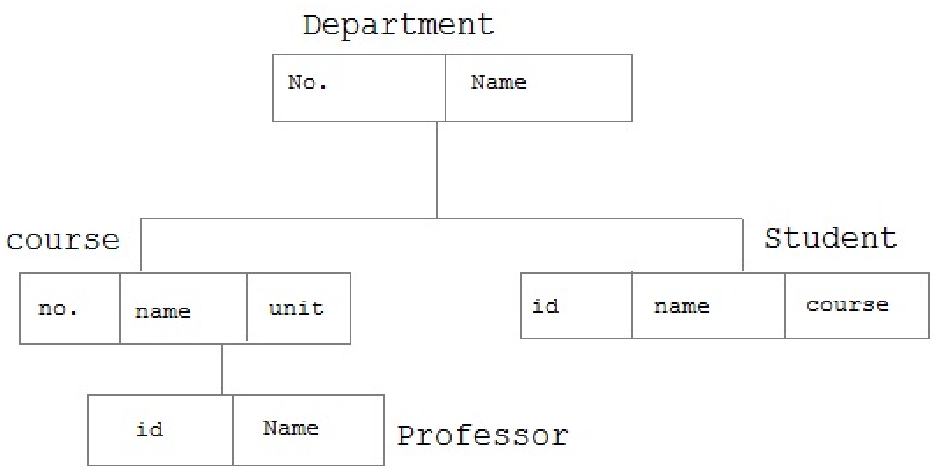
Advantages of DBMS:

1. Segregation of application program.
2. Minimal data duplicity.
3. Easy retrieval of data.
4. Reduced development time and maintenance need.

DBMS Data Model: A Database model defines the logical design of data. The model describes the relationships between different parts of the data. Historically, in database design, three models are commonly used. They are,

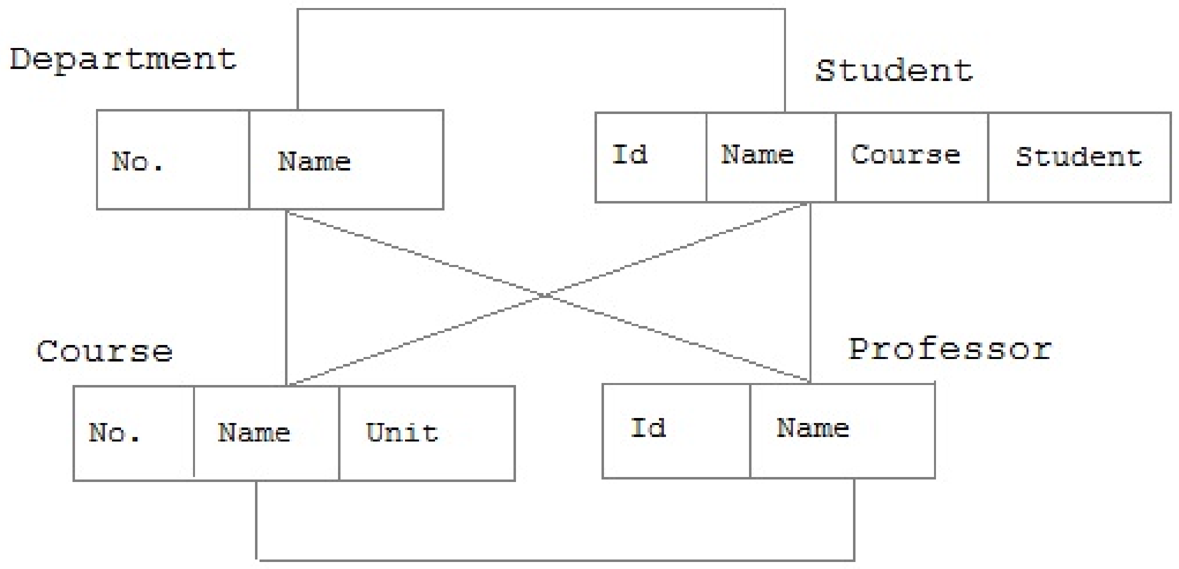
1. Hierarchical Model
2. Network Model
3. Relational Model

Hierarchical Model: In this model, each entity has only one parent but can have several children. At the top of hierarchy there is only one entity which is called Root.



Network Model:

- The network database model was invented by Charles Bachman in 1969 as an enhancement of the already existing database model, the hierarchical database model.
- Because the hierarchical database model was highly flaw, Bachman decided to create a database that is like the hierarchical database but with more flexibility and less defaults.
- The original and existing hierarchical database has one owner file linked strictly to one member file, creating a ladder affect that restricted the database to find relationships outside of its category.
- In the network model, entities are organized in a graph, in which some entities can be accessed through several paths



List of Software's using network model:

1. Integrated Data Store (IDS)
2. IDMS (Integrated Database Management System)
3. RDM Embedded
4. RDM Server
5. Turbo IMAGE

6. Univac DMS-1100

Advantages and Disadvantages of Network Model:

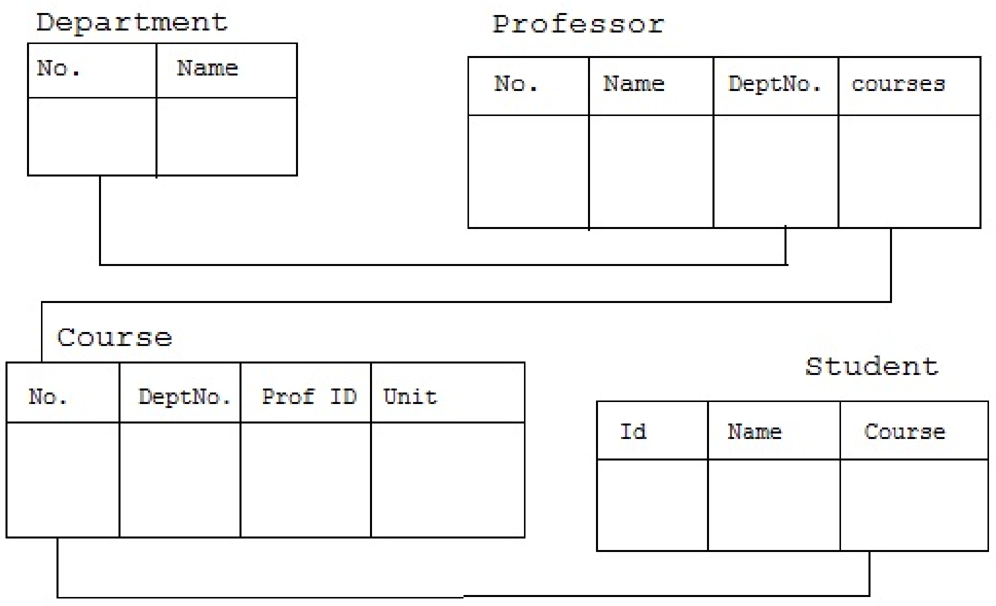
1. Because it has the many-many relationship, network database model can easily be accessed in any table record in the database
2. For more complex data, it is easier to use because of the multiple relationship founded among its data
3. Easier to navigate and search for information because of its flexibility

Disadvantage of a Network Database Model:

1. Difficult for first time users
2. Difficulties with alterations of the database because when information entered can alter the entire database

Relational Model:

In this model, data is organized in two-dimensional tables called relations. The tables or relation are related to each other.



Difference Between Network and Hierarchical models:

1	Many-to-many relationship	One-to-many relationship
2	Easily accessed because of the linkage between the information	Difficult to navigate because of its strict owner to member connection
3	Great flexibility among the information files because the multiple relationships among the files	Less flexibility with the collection of information because of the hierarchical position of the files

1.3 RDBMS

RDBMS stands for Relational Database Management Systems. It is called Relational Data Base Management System (RDBMS) because it is based on relational model introduced by E.F. Codd. In RDBMS data is represented in terms of tuples (rows).

Due to a collection of organized set of tables, data can be accessed easily in RDBMS. Relational database is most commonly used database. It contains number of tables and each table has its own primary key.

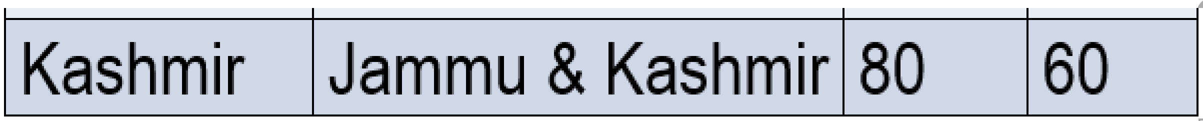
Examples of RDBMS:

1. MySql
2. Oracle
3. Sybase
4. Microsoft Access
5. IBM DB2

RDBMS Table: In Relational database, a table is a collection of data elements organized in terms of rows and columns. A table is also considered as convenient representation of relations. But a table can have duplicate tuples while a true relation cannot have duplicate tuples. Table is the simplest form of data storage. Below is an example of table.

City	State	High	Low
Bangalore	Karnataka	105	90
Mumbai	Maharashtra	101	92
Chennai	Tamilnadu	88	69
New Delhi	Delhi	77	60
Kashmir	Jammu & Kashmir	80	60

Record: A single entry in a table is called a Record or Row. A Record in a table represents set of related data. For example, the above table has 5 records. Following is an example of single record.



Kashmir	Jammu & Kashmir	80	60

Column: In Relational table, a column is a set of value of a particular type. The term Attribute is also used to represent a column.

Database Keys: Keys are very important part of Relational database. They are used to establish and identify relation between tables. They also ensure that each record within a table can be uniquely identified by combination of one or more fields within a table.

Types of Keys:

1. Super Key: Super Key is defined as a set of attributes within a table that uniquely identifies each record within a table. Super Key is a superset of Candidate key. A Super Key is a set of one or more attributes that are taken collectively and can identify all other attributes uniquely.
2. Candidate Key: Candidate keys are defined as the set of fields from which primary key can be selected. It is an attribute or set of attribute that can act as a primary key for a table to uniquely identify each record in that table. In other words candidate keys are minimal super keys.
3. Primary Key: Primary key is a candidate key that is most appropriate to become main key of the table. It is a key that uniquely identify each record in a table.
4. Composite Key: Key that consist of two or more attributes that uniquely identify an entity occurrence is called Composite key. But any attribute that makes up the Composite key is not a key in its own.
5. Secondary Alternate Key: The candidate key which are not selected for primary key are known as secondary keys or alternative keys
6. Non-Key: Non-key attributes are attributes other than candidate key attributes in a table.

Difference Between DBMS and RDBMS:

#	DBMS	RDBMS
1)	DBMS applications store data as file.	RDBMS applications store data in a tabular form.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	Normalization is not present in DBMS.	Normalization is present in RDBMS.
4)	DBMS does not apply any security with regards to data manipulation.	RDBMS defines the integrity constraint for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be no relation between the tables.	in RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS does not support distributed database.	RDBMS supports distributed database.
8)	DBMS is meant to be for small organization and deal with small data. it supports single user.	RDBMS is designed to handle large amount of data. it supports multiple users.

9)	Examples of DBMS are file systems, xml etc.	Example of RDBMS are mysql, postgre, sql server, oracle etc.

1.4 Data Design and its modeling

Data design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database where as a data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world.

Data Design is divided into below stages

1.Planning and Analysis
2.Conceptual Design
3.Logical Design
4.Physical Design
5.Implementation

- Planning and Analysis: Identify actual structure of the data and the sequence of data.

- Conceptual design: A technology-independent specification of the data to be held in the database. Usually represented as a straightforward diagram with supporting documentation. We can use E-R Approach

- Logical Data Model: Translation of the conceptual model into structures that can be implemented using a DBMS. This usually means the models specifies tables and columns.

- Physical Data Model: The logical model incorporating any changes necessary to achieve adequate performance and is also presented in terms of tables and columns, together with a specification of the physical storage, indexes, Keys. Physical data models are normalized.

- Implementation: Identify software, respective hardware, supporting database language.

Chapter 2
Data Modeling and Normalization

Once you get requirements for database design, there are 3 steps involve

1. Creating E-R Diagram
2. Normalize your E-R diagram
3. Design the databases

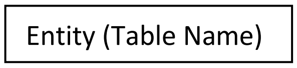
2.1 E-R Diagram

ER-Diagram is a visual representation of data that describes how data is related to each other. It was developed by Peter Chen in 1976. Since then Charles Bachman and James Martin have added some slight refinements to the basic ERD principles.

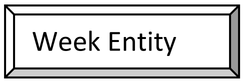
An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is a component of data. In other words, ER diagrams illustrate the logical structure of databases. At first glance an entity relationship diagram looks very much like a flowchart. It is the specialized symbols, and the meanings of those symbols, that make it unique.

E-R Diagram symbols: There are different components available in an ERD

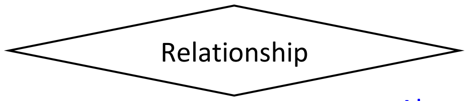
1. Entities: These are represented by rectangles. An entity is an object or concept about which you want to store information



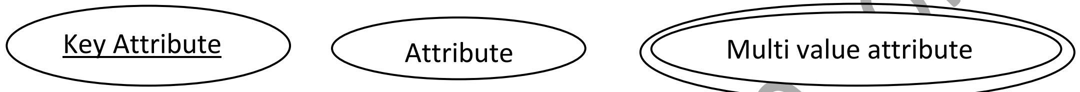
2. Weak entity: Entity that must defined by a F-k relationship with another entity as it cannot be uniquely identified by its own attributes alone.



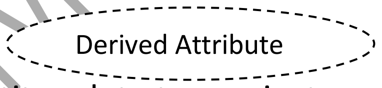
3. Relationship: This is represented by diamond shapes, show how two entities share information in the database.



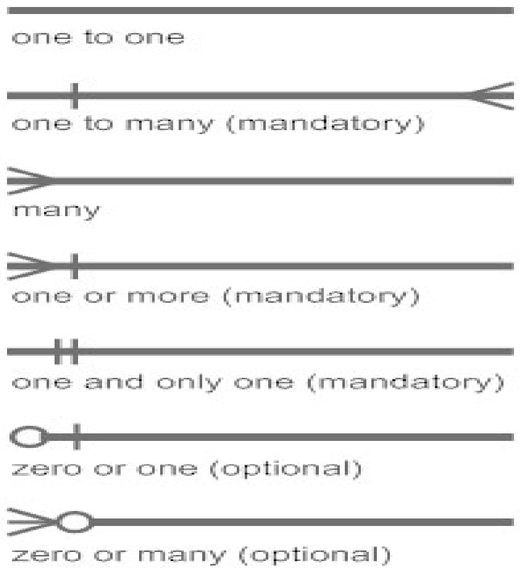
4. Attributes: These are nothing but table columns represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.
5. Multivalued attribute: Attribute can have more than one value. For example, an employee entity can have multiple skill values or employee name have first and last name.

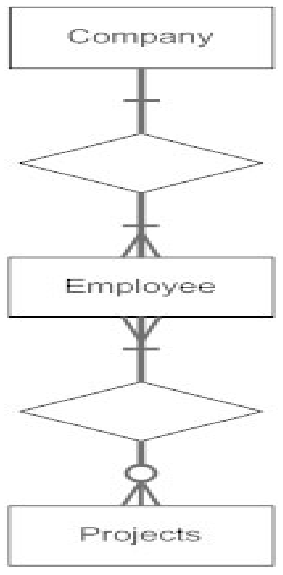


6. Derived attribute: Attribute based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary or age is based on DOB.



7. Cardinality: This specifies how many instances of an entity relate to one instance of another entity. Ordinality is also closely linked to cardinality. While cardinality specifies the occurrences of a relationship, ordinality describes the relationship as either mandatory or optional. In other words, cardinality specifies the maximum number of relationships and ordinality specifies the absolute minimum number of relationships.





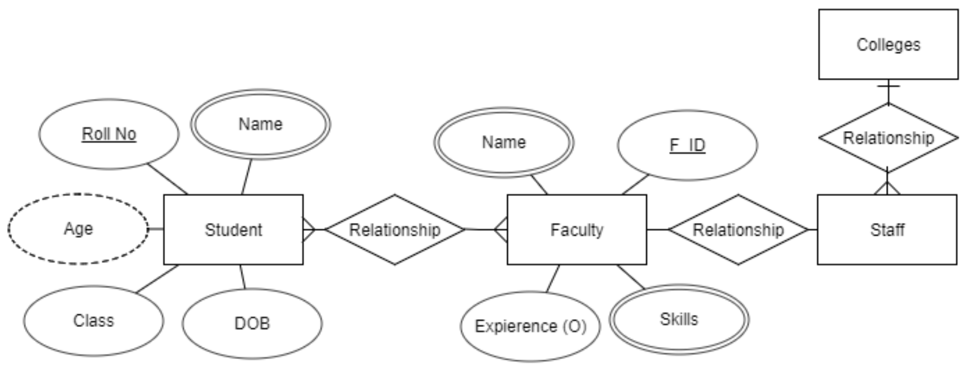
There are many notation styles that express cardinality:

1. Information Engineering Style (One you saw above)
2. Chen Style
3. Bachman Style
4. Martin Style

How to draw a basic ER diagram:

1. Purpose and scope: Define the purpose and scope of what you're analyzing or modeling.
2. Entities: Identify the entities that are involved. When you're ready, start drawing them in rectangles (or your system's choice of shape) and labeling them as nouns.
3. Relationships: Determine how the entities are all related. Draw lines between them to signify the relationships and label them. Some entities may not be related, and that's fine. In different notation systems, the relationship could be labeled in a diamond, another rectangle or directly on top of the connecting line.
4. Attributes: Layer in more detail by adding key attributes of entities. Attributes are often shown as ovals.
5. Cardinality: Show whether the relationship is 1-1, 1-many or many-to-many.

Example of ER-Diagram: You can try using https://erdplus.com/#/standalone



Uses of E-R Diagram

1. Database design: ER diagrams are used to model and design relational databases, in terms of logic and business rules.
2. Database troubleshooting: ER diagrams are used to analyze existing databases to find and resolve problems in logic or deployment.
3. Business information systems: The diagrams are used to design or analyze relational databases used in business processes.
4. Business process re-engineering (BPR): ER diagrams help in analyzing databases used in business process re-engineering and in modeling a new database setup.
5. Research: Since so much research focuses on structured data, ER diagrams can play a key role in setting up useful databases to analyze the data.

Limitations of ER diagrams and models:

1. Only for relational data: Understand that the purpose is to show relationships. ER diagrams show only that relational structure.

2. Not for unstructured data: Unless the data is cleanly delineated into different fields, rows or columns, ER diagrams are probably of limited use. The same is true of semi-structured data, because only some of the data will be useful.

3. Difficulty integrating with an existing database: Using ER Models to integrate with an existing database can be a challenge because of the different architectures.

Exercise:

7. Create ER Diagram for College having different departments (MBA, Eng., etc.), facilities (transport, library, etc.), Exam control (schedule, results, etc.)

8. Create ER diagram for online exam portal supports different type of exams, categories and level of questions,

2.2 Normalization

Normalization is a process that "improves" a database design by generating relations that are of higher normal forms. During this process, we increase the number of tables in the database, while decreasing the amount of data stored in each table.

In RDBMS, nothing considers as fully normalized. A design that has a lower normal form than another design has more redundancy. Uncontrolled redundancy can lead to data integrity problems. The benefit of higher normal forms is that update semantics for the affected data are simplified.

The *objective* of normalization: "*To create relations where every dependency is on the key, the whole key, and nothing but the key*".

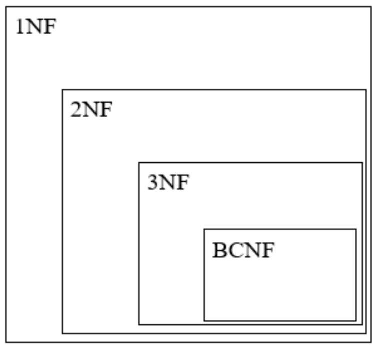
Normalization addresses 3 main problems:

1. INSERT ANOMALY: This refers to the situation when it is impossible to insert certain types of data into the database.
2. DELETE ANOMALY: The deletion of data leads to unintended loss of additional data, data that we had wished to preserve.
3. UPDATE ANOMALY: This refers to the situation where updating the value of a column leads to database inconsistencies (i.e., different rows on the table have different values).

There is a sequence to normal forms:

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyce-Codd Normal Form (BCNF)
5. Fourth Normal Form (4NF)
6. Fifth Normal Form (5NF)
7. Domain Key Normal Form (DKNF)

Most databases should be 3NF or BCNF to avoid the database anomalies.



a relation in BCNF, is also in 3NF

a relation in 3NF is also in 2NF

a relation in 2NF is also in 1NF

Normalization Concepts:

- *Functional dependency*
- Determinant
- Transitive dependency
- Partial dependency
- Candidate Key

Functional Dependencies: We say an attribute, B, has a functional dependency on another attribute, A, if for any two records, which have the same value for A, then the values for B in these two records must be the same.
We illustrate this as: A → B
Example: Suppose we keep track of employee email addresses, and we only track one email address for each employee. Suppose each employee is identified by their unique employee number. We say there is a functional dependency of email address on employee number:
employee number → email address

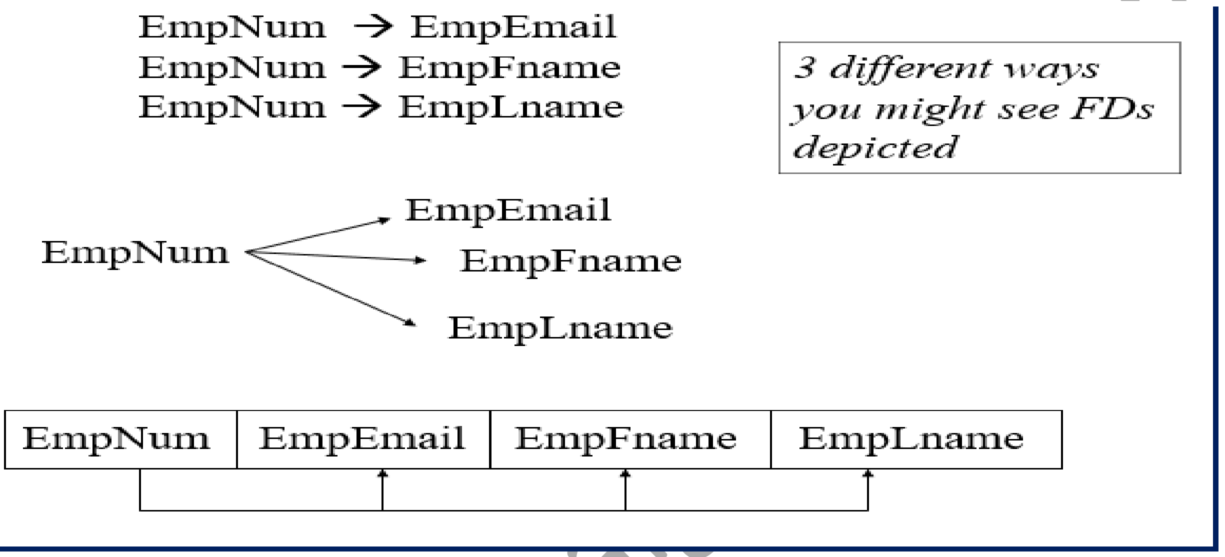
Employee table

EmpNum	Emp First Name	EMP Last Name	Emp email
1	Saurabh	Sinha	saurabh.x.sinha@gmail.com
2	Prashant	saxena	p.saxena@gmail.com
3	Aslam	Diwan	a.diwan@gmail.com
4	Amit	Kumar	Amit.kr@gmail.com
5	Rashi	Nigam	nigam.rashi@gmail.com

If EmpNum is the Primary Key, then the Functional Dependency must exist.

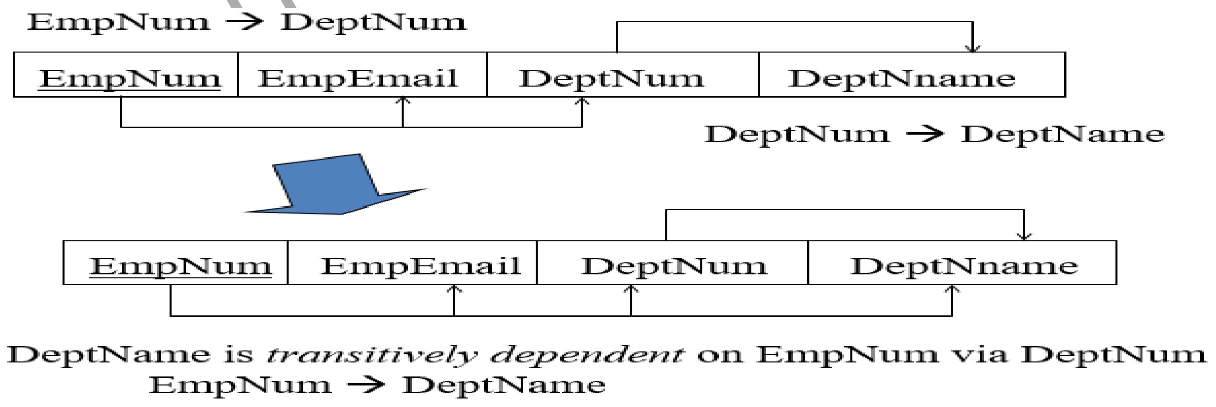
EmpNum → EmpEmail
EmpNum → EmpFirstname
EmpNum → EmpLastname

Different ways to visualise:

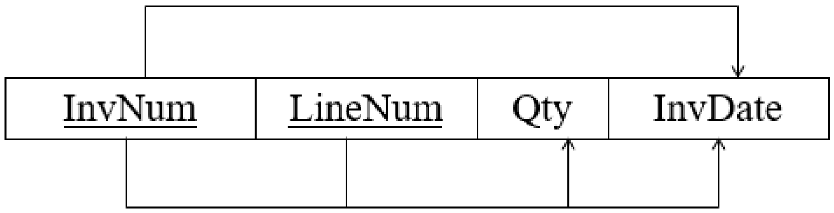


Determinant: In Functional Dependency EmpNum → EmpEmail
Attribute on the LHS is known as the **determinant i.e.** EmpNum is a determinant of EmpEmail

Transitive dependency: Consider attributes A, B, and C, and wheremA → B and B → C. Functional dependencies are transitive, which means that we also have the functional dependency A →C, we say that C is transitively dependent on A through B.



Partial dependency: A partial dependency exists when an attribute B is functionally dependent on an attribute A, and A is a component of a multipart candidate key.



Candidate key is {InvNum, LineNum}, InvDate is partially dependent on {InvNum, LineNum} as InvNum is a determinant of InvDate and InvNum is part of a candidate key

Normal Forms:

1. First Normal form: We say a relation is in 1NF if all values stored in the relation are single-valued and atomic. A database is in first normal form if it satisfies the following conditions:
    a. Contains only atomic values (An atomic value is a value that cannot be divided).
    b. There are no repeating groups (A repeating group means that a table contains two or more records in one columns that are closely related.)

1NF places restrictions on the structure of relations. Values must be simple.

Example: Below table in not in 1st normal form

Empid	Mobile	Degrees
1	123-456-789	B. E
2	123-789-456	B.E, M. E
3	123-456-789	B.E, MBA, LLB
4	123-789-456	MBA, LLB

Emp (Degrees) is a multi-valued field:
employee 2 has two degrees: B.E, M. E
employee 3 has three degrees: B.E, MBA, LLB
employee 4 has two degrees: MBA, LLB

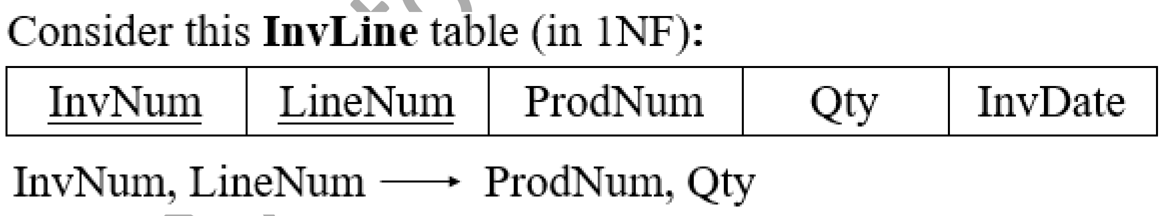
To obtain 1NF relations we must, without loss of information, replace the above with atomic values. Below itable is now normalized in 1st normal form

Empid	Mobile	Degrees
1	123-456-789	B. E
2	123-789-456	B. E
2	123-789-456	M.E
3	123-456-789	B.E
3	123-456-789	MBA
3	123-456-789	LLB
4	123-789-456	MBA
4	123-789-456	LLB

2. Second Normal Form: A table that is in 1st normal form and contains only a single key as the primary key is automatically in 2nd normal form. A relation in 2NF will not have any partial dependencies. A relation is in 2NF if

   a. Relation is in 1NF
   b. Every non-key attribute is fully dependent on Primary key (Composite key). (That is, we don't have any partial functional dependency.)

Example: Table InvLine have composite key (InvNum, LineNum)
Here InvNum is partial key which says InvNum Determinant INVdate

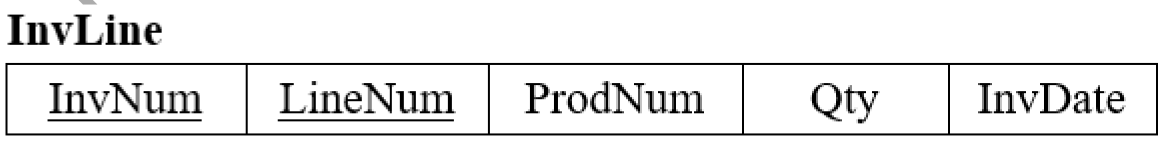


Consider this **InvLine** table (in 1NF):

InvNum	LineNum	ProdNum	Qty	InvDate

InvNum, LineNum $\longrightarrow$ ProdNum, Qty

Hence InvLine is in 1Nf only not 2NF since there is a partial dependency of InvDate on InvNum



InvLine

InvNum	LineNum	ProdNum	Qty	InvDate

The above relation has redundancies: the invoice date is repeated on each invoice line. We can improve the database by decomposing the relation into two relations:

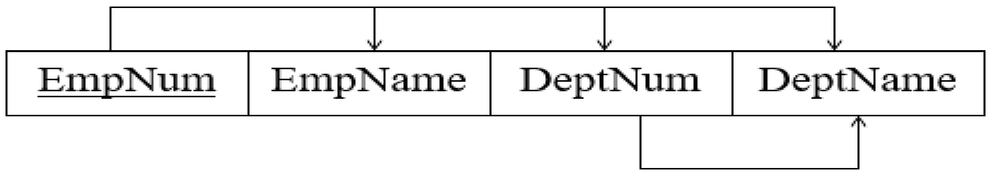
<u>InvNum</u>	<u>LineNum</u>	ProdNum	Qty

<u>InvNum</u>	InvDate

Now in relation 1st We have composite key (InvNum, LineNum) and in 2nd relation primary key is (InvNum)

3. <u>Third Normal Form</u>: A database is in third normal form if it satisfies the following conditions:
   a. It is in second normal form
   b. There is no transitive functional dependency (By transitive functional dependency, we mean A is functionally dependent on B, and B is functionally dependent on C. i.e. C is transitively dependent on A via B.)

Example: Consider this Employee relation



Is the relation in 3NF? … no
Is the relation in 2NF? … yes

Here

- EmpNum is Primary Key
- EmpName, DeptNum, and DeptName are non-key attributes.
- DeptNum determines DeptName, a non-key attribute, and DeptNum is not a primary key.

We correct the situation by decomposing the original relation into two 3NF relations.

EmpNum	EmpName	DeptNum

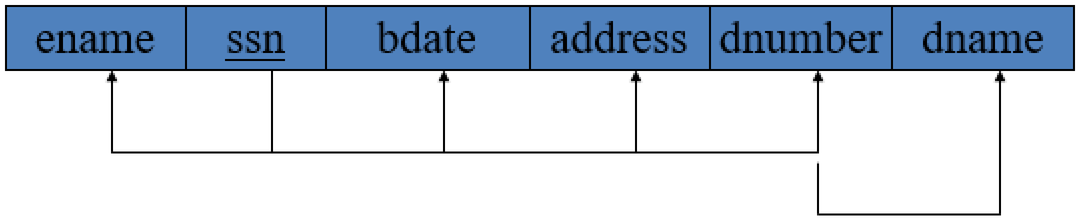
DeptNum	DeptName

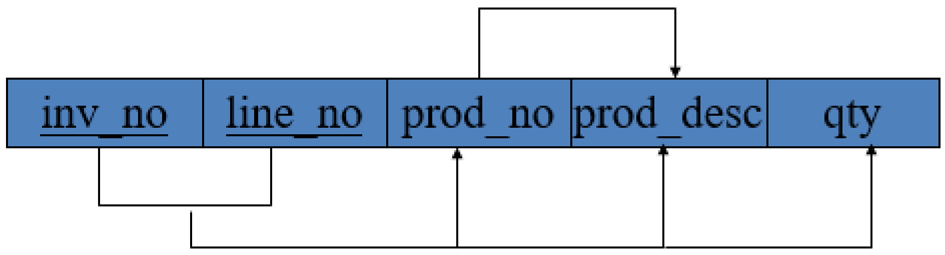
Now both tables are in 3rd Normal form because

- Both the tables have Primary key
- Both tables don't have Partial key
- Both table are not transitively dependent

Exercise: Find current normal form for both entities and convert into higher normal form:

EmployeeDept





Answers: 1.

- 1st NF: Yes
- 2nd NF: Yes, because no composite key
- 3rd NF: All non-key has determinant as PK
- BCNF: There is multiple overlapping we have:
  - dnumber → dname.
- Solution: Break into 2 tables
- ssn → ename, bdate, address

- Dnumber → dname

Answer 2:

- 1st NF: Yes
- 2nd NF: Yes, even we have composite key because none of non-key have determinant from composite key partially
- 3rd NF: All non-key has determinant as PK
- BCNF: There is multiple overlapping we have:
    - prod_no → prod_desc.
- Solution: Break into 2 tables
- (Inv_no, Line_no) → prodno, qty
- Prod_nor → prod_desc

2.3 Design Databases

Designing a database is in fact fairly easy, but there are a few rules to stick to. It is important to know what these rules are, but more importantly is to know why these rules exist, otherwise you will tend to make mistakes!

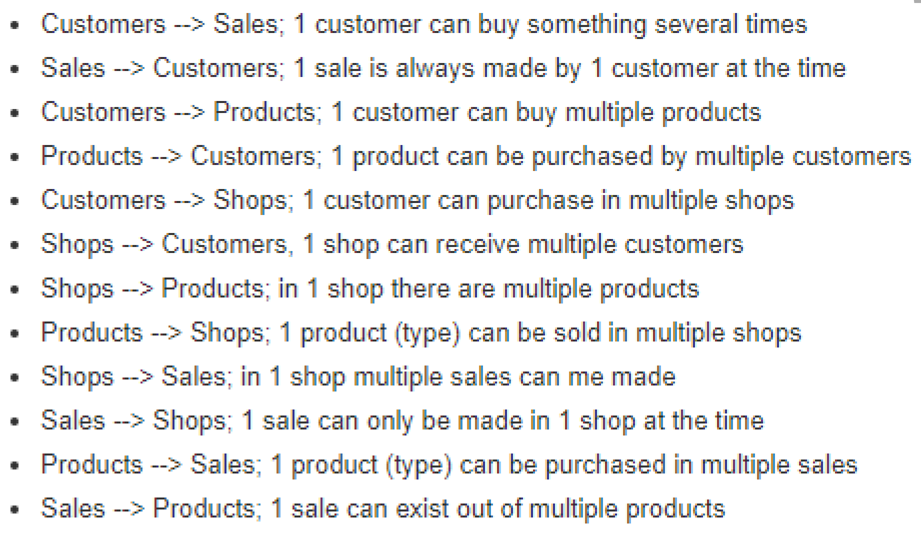
Standardization makes your data model flexible and that makes working with your data much easier. Please, take the time to learn these rules and apply them! The database used in this article is designed with our database design and modeling tool for Databases.

A good database design starts with a list of the data that you want to include in your database and what you want to be able to do with the database later. This can all be written in your own language, without any SQL. In this stage, you must try not to think in tables or columns, but just think: "What do I need to know?" Don't take this too lightly, because if you find out later that you forgot something, usually you need to start all over. Adding things to your database is mostly a lot of work.

1. Identifying Entities: The types of information that are saved in the database are called 'entities'. These entities exist in four kinds: people, things, events, and locations. Everything you could want to put in a database fits into one of these categories. If the information you want to include doesn't fit into these categories, then it is probably not an entity but a property of an entity, an attribute.
   For example, imagine that you are creating a website for a shop, what kind of information do you have to deal with? In a shop, you sell your products to customers. The "Shop" is a location; "Sale" is an event; "Products" are things; and "Customers" are people. These are all entities that need to be included in your database. But what other things are happening when selling a product? A customer comes into the shop, approaches the vendor, asks a question and gets an answer. "Vendors" also participate, and because vendors are people, we need a vendor's entity.

2. Identifying Relationships: The next step is to determine the relationships between the entities and to determine the cardinality of each relationship. The relationship is the connection between the entities, just like in the real world: what does one entity do with the other, how do they relate to each other? For

example, customers buy products, products are sold to customers, a sale comprises products, a sale happens in a shop. The cardinality shows how much of one side of the relationship belongs to how much of the other side of the relationship. First, you need to state for each relationship, how much of one side belongs to exactly 1 of the other side. For example: How many customers belong to 1 sale? How many sales belong to 1 customer? How many sales take place in 1 shop?



- Customers --> Sales; 1 customer can buy something several times
- Sales --> Customers; 1 sale is always made by 1 customer at the time
- Customers --> Products; 1 customer can buy multiple products
- Products --> Customers; 1 product can be purchased by multiple customers
- Customers --> Shops; 1 customer can purchase in multiple shops
- Shops --> Customers, 1 shop can receive multiple customers
- Shops --> Products; in 1 shop there are multiple products
- Products --> Shops; 1 product (type) can be sold in multiple shops
- Shops --> Sales; in 1 shop multiple sales can me made
- Sales --> Shops; 1 sale can only be made in 1 shop at the time
- Products --> Sales; 1 product (type) can be purchased in multiple sales
- Sales --> Products; 1 sale can exist out of multiple products

Types of relationship:

a. One to One: For one row in table X there is a only one row in table Y
b. One to Many: One customer can by multiple products
c. Many to many: One student has many degrees and one degree can be obtain by many students.
d. Recursive Relationships: Sometimes an entity refers to itself. For example, think of a work hierarchy: an employee has a boss; and the boss is an employee too
e. Redundant Relationships: Sometimes in your model you will get a relationship that are already indicated by other relationships, although not directly. Example there is a direct relationship between customers and products. But there are also relationships from customers to sales and from sales to products, so indirectly there already is a relationship between customers and products through sales.

3. Identifying Attributes: The data elements that you want to save for each entity are called 'attributes'.

ALGAE

   a. About the products that you sell, you want to know, for example, what the price is, what the name of the manufacturer is, and what the type number is.
   b. About the customers, you know their customer number, their name, and address.
   c. About the shops, you know the location code, the name, the address. Of the sales, you know when they happened, in which shop, what products were sold, and the sum of the sale.
4. Categorize items like
   a. Entity as weak or strong entity
   b. Attributes as Key, Non-Key, Partial
   c. Relationship as 1:1, 1:M, Redundant, self-relationship.
5. Create ER Diagram
6. Choose Technology to Build your database based on different characteristics like price, hardware, brand, availability.
7. Defining the Attribute's Data Type: One should define the datatype for attributes like integer, character, float, date time, money.
8. Normalize: Normalize your database.

Chapter 3
SQL Language

SQL is defined as Structured Query Language. It is a computer language for storing, manipulating and retrieving data stored in relational database. SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language. SQL is an ANSI (American National Standards Institute) standard

Although SQL is an ANSI (American National Standards Institute) standard, there are different versions of the SQL language. However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

3.1 Basic SQL

SQL can do below activities:

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.

- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views

History of SQL:

- 1970 -- Dr. E. F. "Ted" of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974 -- Structured Query Language appeared.
- 1978 -- IBM worked to develop Codd's ideas and released a product named System/R.
- 1986 -- IBM developed the first prototype of relational database and standardized

3.2 SQL commands and its types

The standard SQL commands to interact with relational databases are divided into 4 parts

1. DDL (Data Definition language)
2. DML (Data Manipulation language)
3. DCL (Data Control language)
4. TCL (Transaction Control Languages)

Data Definition Language:

Command	Description
Create	Creates a new table, a view of a table, or other object in database
Alter	Modifies an existing database object, such as a table.
Drop	Deletes an entire table, a view of a table or other object in the database.
Truncate	Deletes data for entire table

Syntax and Examples:

/************Create Table******************/

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255)

);
/************Alter Table*****************/
ALTER TABLE table_name
ADD column_name datatype;

ALTER TABLE Persons
ADD DateOfBirth date;
/************Truncate Table*****************/
TRUNCATE TABLE table_name;

TRUNCATE TABLE Shippers;
/************Drop Table*****************/

DROP TABLE table_name;

DROP TABLE Shippers;
/**********************************/
```

Data Manipulation language:

Command	Description
Insert	Create new record
Update	Modify existing record
Delete	Delete existing record
Select	Retrieve existing record

Syntax and Examples:

```
/************Insert Table*****************/
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode,
Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
/************Update Table*****************/
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;

UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
/************Delete Table*****************/
DELETE FROM table_name
WHERE condition;

DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste';
/************Select Table*****************/
SELECT column1, column2, ...
FROM table_name
WHERE condition;
SELECT *
FROM table_name
WHERE condition;


SELECT * FROM Customers
WHERE Country='Mexico';
/*************************************/
```

Data Control Language:

Command	Description
Grant	Gives a privilege to user
Revoke	Takes back privileges granted from user

Transaction Control language:

Command	Description
Commit	Apply changes permanently
Rollback	Revert changes

Common Data Types:

`char(size)`	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
`varchar(size)`	Variable-length character string. Max size is specified in parenthesis.
`Int`	For any number or integer
`date`	Date value
`Decimal(size,d)`	Decimal value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

3.3 SQL constraints

Constraints are the rules enforced on data columns on table.
These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be column level or table level. Column level constraints are applied only to one column, whereas table level constraints are applied to the whole table.
Below constraints are supported by SQL

1. NOT NULL Constraint: Ensures that a column cannot have NULL value.
2. DEFAULT Constraint: Provides a default value for a column when none is specified.
3. UNIQUE Constraint: Ensures that all values in a column are different.
4. PRIMARY Key: Uniquely identified each rows/record in a database table.
5. FOREIGN Key: Uniquely identified a rows/records in any another database table.
6. CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.

3.4 SQL JOINs

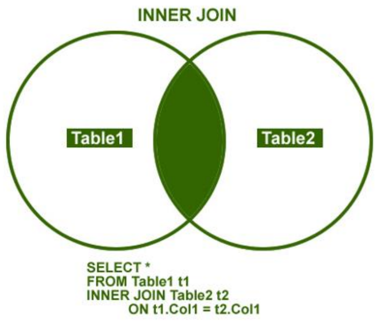
An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them. The most common type of join is: SQL INNER JOIN (simple join). An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Types:

1. INNER JOIN: Returns all rows when there is at least one match in BOTH tables
2. LEFT JOIN: Return all rows from the left table, and the matched rows from the right table
3. RIGHT JOIN: Return all rows from the right table, and the matched rows from the left table
4. FULL JOIN: Return all rows when there is a match in ONE of the tables
5. SELF JOIN: Similar like inner join but on same table

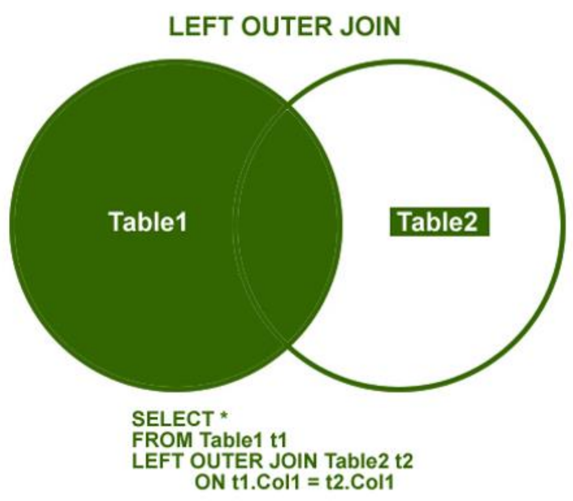
INNER JOIN:
The INNER JOIN keyword selects records that have matching values in both tables.



```
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
     ON t1.Col1 = t2.Col1
```

- SELECT *column_name(s)*
  FROM *table1*
  INNER JOIN *table2*
  ON *table1.column_name=table2.column_name*;
- SELECT *column_name(s)*
  FROM *table1*
  JOIN *table2*
  ON *table1.column_name=table2.column_name*;

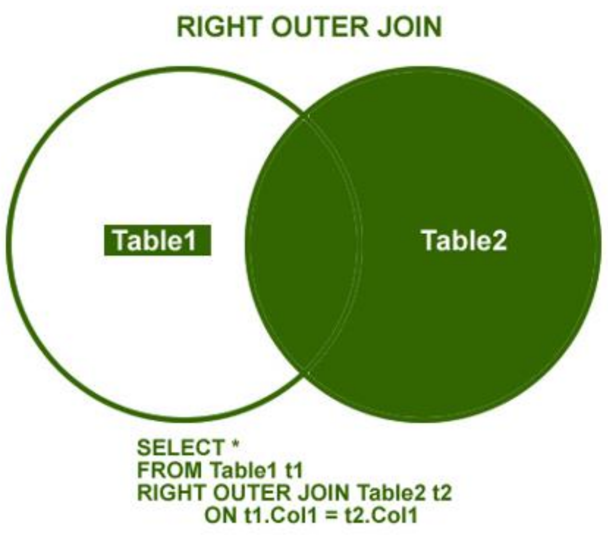
SQL LEFT JOIN:
The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.



- SELECT *column_name(s)*
  FROM *table1*
  LEFT JOIN *table2*
  ON *table1.column_name=table2.column_name*;
- SELECT *column_name(s)*
  FROM *table1*

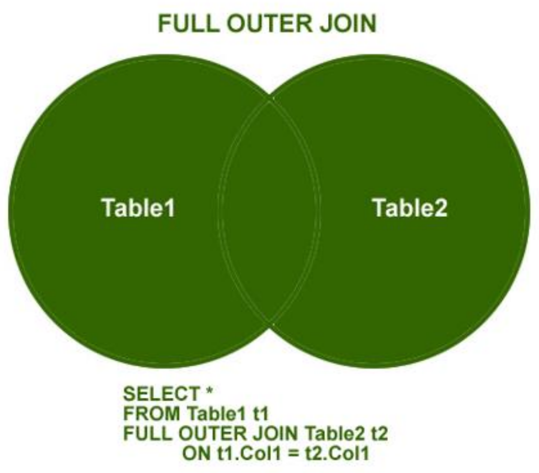
LEFT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;

SQL RIGHT JOIN:
The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.



- SELECT *column_name(s)*
  FROM *table1*
  RIGHT JOIN *table2*
  ON *table1.column_name=table2.column_name*;
- SELECT *column_name(s)*
  FROM *table1*
  RIGHT OUTER JOIN *table2*
  ON *table1.column_name=table2.column_name*;

FULL OUTER JOIN:
The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).



```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

SQL Self JOIN:

A self-JOIN is a regular join, but the table is joined with itself.

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

3.5 SQL Aggregate Functions

SQL has many built-in functions for performing calculations on data. SQL aggregate functions return a single value, calculated from values in a column.

Useful aggregate functions:

1. AVG () - Returns the average value
2. COUNT () - Returns the number of rows
3. MAX () - Returns the largest value
4. MIN () - Returns the smallest value
5. SUM () - Returns the sum

Syntax:

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;

SELECT MAX(column_name)
FROM table_name
WHERE condition;

SELECT SUM(column_name)
FROM table_name
WHERE condition;

SELECT AVG(column_name)
FROM table_name
WHERE condition;

SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

SQL Group BY:
The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY clause is used in a SELECT statement to collect data across multiple records and group the results by one or more columns

Syntax:

```
SELECT expression1, expression2, ... expression_n,
       aggregate_function (expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n;
```

3.6 SQL OPERATORS

SQL operators are used with where clause in SQL queries.

For example:

- SELECT *column_name,column_name*
  FROM *table_name*
  WHERE *column_name operator value*;

- SELECT *id*, *name*
  FROM *emp*
  WHERE *salary = 100*;

Below is most commonly used operators in SQL Server

=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal to
!=	In some versions of SQL this operator is used instead on <>
LIKE	Explained later
In	To specify multiple possible values for a column
Between	Between an inclusive range

Chapter 4
Transaction

A transaction is a unit of work that you want to treat as "a whole". It must either happen in full, or not at all. In computer programming, a transaction usually means a sequence of information exchange and related work (such as database updating) that is treated as a unit for the purposes of satisfying a request and for ensuring database integrity. For a transaction to be completed and database changes to made permanent, a transaction must be completed in its entirety.

Example: Transferring money from one bank account to another. To do that you must first withdraw the amount from the source account, and then deposit it to the destination account. The operation must succeed in full. If you stop halfway, the money will be lost.

4.1 Transaction Concept

A transaction is a unit of program execution that accesses and possibly updates various data items.

Example: Below is transaction to transfer $50 from account A to account B:

1. read(A)
2. A = A – 50
3. write(A)
4. read(B)
5. B = B + 50
6. write(B)

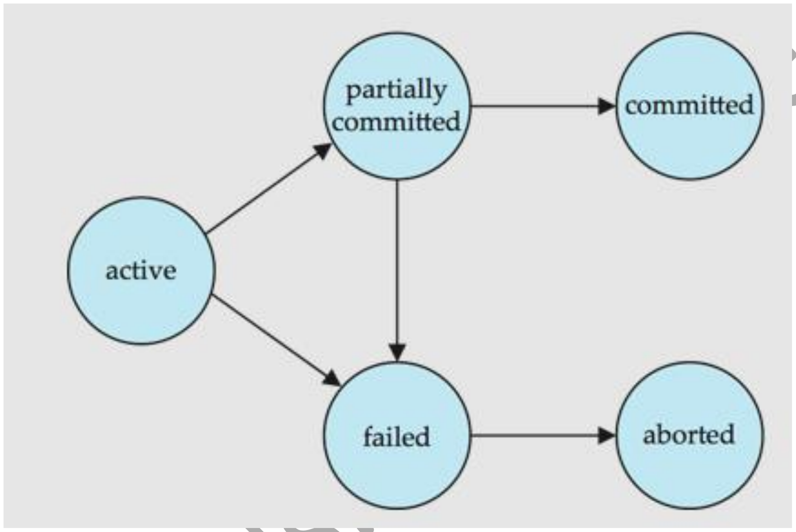
In any transaction two main issues to deal with are:

a. Failures of various kinds, such as hardware failures and system crashes
b. Concurrent execution of multiple transactions

Transaction State:

1. **Active** – The initial state; the transaction stays in this state while it is executing
2. **Partially committed** – after the final statement has been executed.

3. **Failed --** after the discovery that normal execution can no longer proceed.
4. **Aborted** – after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
   – Restart the transaction
      • can be done only if no internal logical error
   – Kill the transaction
5. **Committed** – after successful completion.



Transaction Lifecycle: Transaction must be tracked for recovery once transaction starts till it terminates or commits.

1. Begin_Transaction: Marks the beginning of a transaction execution;
2. End_Transaction: Specifies that the read and write operations have ended and marks the end limit of transaction execution (but may be aborted because of concurrency control);
3. Commit_Transaction: signals a successful end of the transaction. Any updates executed by the transaction can be safely committed to the database and will not be undone;
4. Rollback (or Abort): Signals that the transaction has ended unsuccessfully. Any changes that the transaction may have applied to the database must be undone;

5. Undo: Like ROLLBACK but it applies to a single operation rather than to a whole transaction;
6. Redo: specifies that certain transaction operations must be redone to ensure that all the operations of a committed transaction have been applied successfully to the database;

4.2 Acid Properties

ACID properties are defined as

a. Atomicity
b. Consistency
c. Isolation
d. Durability

All RDBMS systems follow these properties religiously. If these properties were not followed can cause database crash.

1. Atomicity: It is an all-or-none proposition. It states that database modifications must follow an "all or nothing" rule. Each transaction is said to be "atomic." If one part of the transaction fails, the entire transaction fails.
   A transaction can
    1. Commit after completing its actions
    2. Abort because of
        a. Internal DBMS decision: restart
        b. System crash: power, disk failure, …
        c. Unexpected situation: unable to access disk, data value, …
    3. A transaction interrupted in the middle could leave the database inconsistent

DBMS needs to remove the effects of partial transactions to ensure atomicity: either all a transaction's actions are performed or none

- DBMS ensures atomicity by undoing the actions of partial transactions
- To enable atomiciy, the DBMS maintains a record, called a log, of all writes to the database
- The component of a DBMS responsible for this is called the recovery manager

2. Consistency: This states that only valid data will be written to the database. If, for some reason, a transaction is executed that violates the database's consistency rules, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules.
   The database must be in a consistent state both before the transaction begins and after the transaction has completed, irrespective of whether the transaction was a success (and therefore committed) or a failure

Example: If an insert violates a check constraint or the referential integrity of the table in question, then the transaction must be rolled back or a compensating action

3. Isolation: This keeps transactions separated from each other until they're finished. It requires that multiple transactions occurring at the same time not impact each other's execution. The isolation property does not ensure which transaction will execute first, merely that they will not interfere with each other.
    Isolation guarantees that even though transactions may be interleaved, the net effect is identical to executing the transactions serially
    For example, if transactions T1 and T2 are executed concurrently, the net effect is equivalent to executing
    - T1 followed by T2, or
    - T2 followed by T1

    NOTE: The DBMS provides no guarantee of effective order of execution

4. Durability: It guarantees that the database will keep track of pending changes in such a way that the server can recover from an abnormal termination. It ensures that any transaction committed to the database will not be lost. Durability is ensured using database backups and transaction logs that facilitate the restoration of committed transactions despite any subsequent software or hardware failures

Reference: http://saurabhsinhainblogs.blogspot.in/2014/02/sql-server-acid-properties.html

4.3 Database Recovery & Transaction Log

Database recovery defines as committing or rollback transactions in case of disaster. Disasters like

- System crashes
- Power failures
- Disk crashes
- User mistakes
- Sabotage
- Natural disasters

Post disaster RDBMS suggest 2 type of operations

1. Undo: Roll back all transactions since last checkpoint
2. Redo: Re-apply all transactions till last checkpoint

To recover transactions (committing or rollback) once database is available again post disaster, RDBMS uses logfiles So question comes up

- What is a log file?
- What is the significance of log file for SQL engine
- Why database need log file
- Architecture of log file
- Pros and cons of log file

What is a log file?
Log file is a physical file requires satisfying one of the properties of Relational database systems. This property is called as Durability (ACID Properties). As per durability any transaction occurring over RDBMS system should be durable. Either complete or should be able to rollback and to rollback any transaction DBMS should know what is done till now.
So, Log files contain Sql at-least to rollback any transaction in case of system failure.
Every RDBMS database has a transaction log that records all transactions and the database modifications made by each transaction. The transaction log must be truncated on a regular basis to keep it from filling up. However, some factors can delay log truncation

What is the significance of log file for SQL engine?

All database software's are bound with rules of RDBMS and for durability server engine have implemented protocol "Write Ahead Log" also called as WAL. And this protocol makes sure that at-least undo part of every transaction should first go to log file and then to data cache. So, to implement WAL server uses log buffer, physical log file and data cache.

The transaction log is a critical component of the database and, if there is a system failure, the transaction log might be required to bring your database back to a consistent state. The transaction log should never be deleted or moved unless you fully understand the behavior

Why database need log file?
As discussed Log file write every transaction detail in log file before moving to data pages but still why separate file, why can't it adjust all logs in same data file?

So here we have 2 basic reasons
1. If I keep both log and data in same file and there is corruption occur then even though I have full back up I can't do point in time recovery
2. The basic idea of data file is to hold data and keep reusing whereas log file to keep data only till transaction is running (durability) and once transaction is completed or (kept in log backup) it's no longer required.

Architecture of log file:

1. Log file is made of VLF's also called as virtual log file, each "chunk" that is added, is divided into VLFs at the time the log growth.
2. Log file is a circular linked list of VLF's
3. Sql server writes transaction in log file serially.
4. If any database is having more than 1 log file then also, second can't be used, until first file is full
5. No. of Vlfs in a log file can be identified by using command "DBCC LOGINFO"
6. VLF allows log file to get reused once all transactions in VLF file is committed or rolled back.

No. of VLF added on every growth in log file depends on below formula for SQL Server only:

Chunks less than 64MB = 4 VLFs
Chunks of 64MB and less than 1GB = 8 VLFs
Chunks of 1GB and larger = 16 VLFs

Cons of log file:

1. Transactions written in log file are serial in nature, adding one more file will not help to increase I/O performance.
2. While the log is 'growing' then it is essentially locked, any processes trying to do insert/update/delete activity will block until the growth has completed.
3. Too many VLF can cause fragmentation (slow down database startup and also log backup and restore operations)
4. Few VLF with large size of VLF can cause VLF will remain active and can take much time to clear VLF

4.5 Locks

In database, a lock is used to access a database concurrently for multiple users. This prevents data from being corrupted or invalidated when multiple users try to write to the database. Any single user can only modify those database records (that is, items in the database) to which they have applied a lock that gives them exclusive access to the record until the lock is released.

From the programmer's point of view, there are two mechanisms for locking data in a database:

- Pessimistic locking is where a record or page is locked immediately when the lock is requested. With pessimistic locking, it should be guaranteed that the record will be updated.
- Optimistic locking is where a record or page is only locked when the changes made to that record are updated. The Optimistic locking situation is only appropriate when there is less chance of someone needing to access the record while it is locked; otherwise it cannot be certain that the update will succeed because the attempt to update the record will fail if another user updates the record first.

In the sight of database, there are three mechanisms for locking data in a database:

1. Exclusive Locking: The resources have been locked only allow accessing the locking operation, other operations will not be accepted. Implementing the command of update data, that is, INSERT, UPDATE or DELETE command, database will use an exclusive lock automatically. However, when the object has other locks, we cannot use exclusive lock. Exclusive lock can be released until the end of transaction.
2. Shared locking: The resources which have been locked only allow other users reading, but not modifying. In SELECT command, database usually locked the object with shared lock. Usually when the data with the shared lock has been read, the shared lock would be released immediately.
3. Update locking: Update lock is created to avoid deadlock. When SQL Server updates the data, it will lock the data with update lock and the data can be read, but cannot be modified. When SQL Server sure to update operation of data, it will change to exclusive lock automatically. However, there are other locks in object, it cannot be locked with updated lock.

4.6 Deadlock

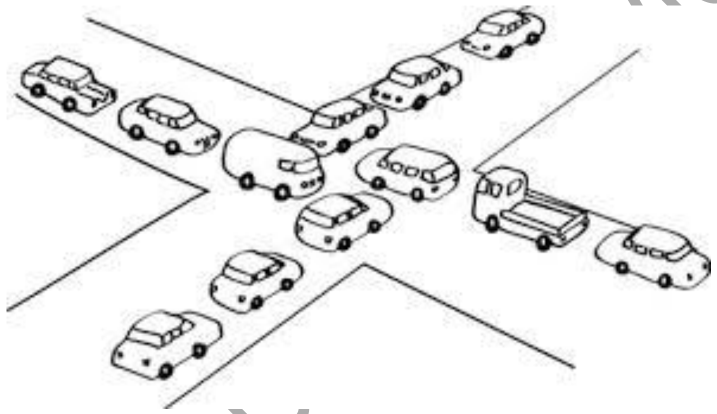
A situation, typically one involving opposing parties, in which no progress can be made.

OR

A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function.

OR

In concurrent computing, a deadlock is a state in which each member of a group of actions is waiting for some other member to release a lock.



Deadlock between Program1 and Program 2:

- Program 1 requests resource A and receives it.
- Program 2 requests resource B and receives it.
- Program 1 requests resource B and is queued up, pending the release of B.
- Program 2 requests resource A and is queued up, pending the release of A.

A deadlock situation can arise if and only if all the following conditions hold simultaneously in a system

1. Mutual exclusion: The resources involved must be unshareable otherwise, the processes would not be prevented from using the resource when necessary. Only one process can use the resource at any given instant of time.
2. Hold and wait or resource holding: A process is currently holding at least one resource and requesting additional resources which are being held by other processes.

3. No preemption: A resource can be released only voluntarily by the process holding it.
4. Circular wait: each process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, P = {P1, P2, …, PN}, such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3 and so on until PN is waiting for a resource held by P1.

Deadlock Detection: Deadlock detection is performed by a lock monitor thread that periodically initiates a search through all the tasks in an instance of the Database Engine.

The following points describe the search process:

- The default interval is 5 seconds.
- If the lock monitor thread finds deadlocks, the deadlock detection interval will drop from 5 seconds to as low as 100 milliseconds depending on the frequency of deadlocks.
- If the lock monitor thread stops finding deadlocks, the Database Engine increases the intervals between searches to 5 seconds.
- If a deadlock has just been detected, it is assumed that the next threads that must wait for a lock are entering the deadlock cycle. The first couple of lock waits after a deadlock has been detected will immediately trigger a deadlock search rather than wait for the next deadlock detection interval. For example, if the current interval is 5 seconds, and a deadlock was just detected, the next lock wait will kick off the deadlock detector immediately. If this lock wait is part of a deadlock, it will be detected right away rather than during next deadlock search.

4.7 Isolation Levels

Isolation levels in RDBMS control the way locking works between transactions. This keeps transactions separated from each other until they're finished. It requires that multiple transactions occurring at the same time not impact each other's execution. The isolation property does not ensure which transaction will execute first, merely that they will not interfere with each other.

Isolation guarantees that even though transactions may be interleaved, the net effect is identical to executing the transactions serially.

Before we go any further it is important to understand these two terms….

- Dirty Reads – This is when you read uncommitted data, when doing this there is no guarantee that data read will ever be committed meaning the data could well be bad.
- Phantom Reads – This is when data that you are working with has been changed by another transaction since you first read it in. This means subsequent reads of this data in the same transaction could well be different.

There are 4 types of Isolation level:

1. Read Uncommitted
2. Read Committed (The default)
3. Repeatable Read
4. Serializable

Read Uncommitted: This is the lowest isolation level. Read uncommitted causes no shared locks to be requested which allows you to read data that is currently being modified in other transactions. It also allows other transactions to modify data that you are reading. As you can probably imagine this can cause some unexpected results in a variety of different ways.

Example: Data returned by the select could be in a half way state if an update was running in another transaction causing some of your rows to come back with the updated values and some not to.

Read Committed: This is the default isolation level and means selects will only return committed data. Select statements will issue shared lock requests against data you're querying this causes you to wait if another transaction already has an exclusive lock on that data. Once you have your shared lock any other transactions trying to modify that data will request an exclusive lock and be made to wait until your Read Committed transaction finishes.

Repeatable Read: This is like Read Committed but with the additional guarantee that if you issue the same select twice in a transaction you will get the same results both times. It does this by holding on to the shared locks it obtains on the records it reads until the end of the transaction, this means any transactions that try to modify these records are forced to wait for the read transaction to complete.

Serializable: This isolation level takes Repeatable Read and adds the guarantee that no new data will be added eradicating the chance of getting Phantom Reads. It does this by placing range locks on the queried data. This causes any other transactions trying to modify or insert data touched on by this transaction to wait until it has finished.

Chapter 5
Data warehouse

The term "Data Warehouse" was first coined by Bill Inmon in 1990. Per Inmon, a data warehouse is a subject oriented, integrated, time-variant, and non-volatile collection of data. This data helps analysts to take informed decisions in an organization.

A data warehouse is constructed by integrating data from multiple heterogeneous sources. It supports analytical reporting, structured and/or adhoc queries and decision making.

5.1 Understanding a Data Warehouse

A data warehouse is a database, which is kept separate from the organization's operational database. There is no frequent updating done in a data warehouse.
It possesses consolidated historical data, which helps the organization to analyze its business. A data warehouse helps executives to organize, understand, and use their data to take strategic decisions.

Data warehouse systems help in the integration of diversity of application systems. A data warehouse system helps in consolidated historical data analysis.

A data warehouses is kept separate from operational databases due to the following reasons:

1. An operational database is constructed for well-known tasks and workloads such as searching records, indexing, etc. In contract, data warehouse queries are often complex and they present a general form of data.
2. Operational databases support concurrent processing of multiple transactions. Concurrency control and recovery mechanisms are required for operational databases to ensure robustness and consistency of the database.
3. An operational database query allows to read and modify operations, while an OLAP query needs only read only access of stored data.
4. An operational database maintains current data. On the other hand, a data warehouse maintains historical data.

Data Warehouse Architecture: Different data warehousing systems have different structures. Some may have an ODS (operational data store), while some may have multiple data marts. Some may have a small number of data sources, while some may have dozens of data sources. In view of this, it is far more reasonable to present the different layers of a data warehouse architecture rather than discussing the specifics of any one system.

In general, all data warehouse systems have the following layers:

1. Data Source Layer
2. Data Extraction Layer
3. Staging Area
4. ETL Layer
5. Data Storage Layer
6. Data Logic Layer
7. Data Presentation Layer
8. Metadata Layer
9. System Operations Layer

1.2 Data Warehouse Features

1. Subject Oriented - A data warehouse is subject oriented because it provides information around a subject rather than the organization's ongoing operations. These subjects can be product, customers, suppliers, sales, revenue, etc. A data warehouse does not focus on the ongoing operations, rather it focuses on modelling and analysis of data for decision making.
2. Integrated - A data warehouse is constructed by integrating data from heterogeneous sources such as relational databases, flat files, etc. This integration enhances the effective analysis of data.
3. Time Variant - The data collected in a data warehouse is identified with a time. The data in a data warehouse provides information from the historical point of view.
4. Non-volatile - Non-volatile means the previous data is not erased when new data is added to it. A data warehouse is kept separate from the operational database and therefore frequent changes in operational database is not reflected in the data warehouse.

5.3 Types of Data Warehouse

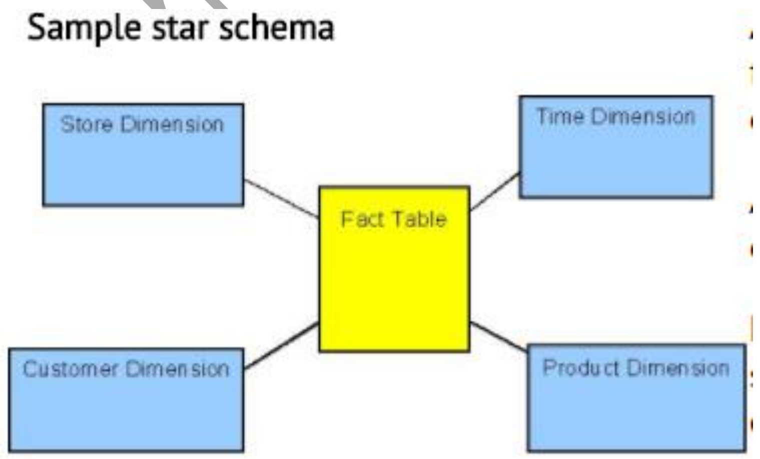
1. Information Processing - A data warehouse allows to process the data stored in it. The data can be processed by means of querying, basic statistical analysis, reporting using crosstabs, tables, charts, or graphs.

2. Analytical Processing - A data warehouse supports analytical processing of the information stored in it. The data can be analyzed by means of basic OLAP operations, including slice-and-dice, drill down, drill up, and pivoting.

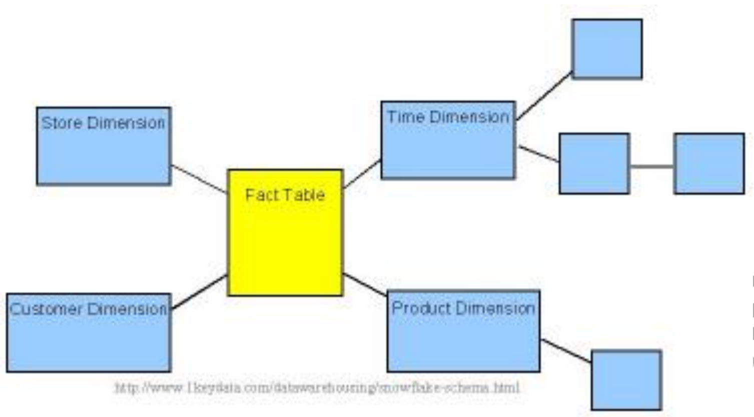
3. Data Mining - Data mining supports knowledge discovery by finding hidden patterns and associations, constructing analytical models, performing classification and prediction. These mining results can be presented using the visualization tools.

Data warehouse supports different data models:

Star schema: A single object (the fact table) sits in the middle and is radically connected to other surrounding objects (dimension lookup tables) like a star. Each dimension is represented as a single table. The primary key in each dimension table is related to a foreign key in the fact table.



Snowflake schema: The snowflake schema is an extension of the star schema, where each point of the star explodes into more points. In a star schema, each dimension is represented by a single dimensional table, whereas in a snowflake schema, that dimensional table is normalized into multiple lookup tables, each representing a level in the dimensional hierarchy.



5.4 OLAP

1. OLAP is an acronym for Online Analytical Processing.
2. OLAP performs multidimensional analysis of business data and provides the capability for complex calculations, trend analysis, and sophisticated data modeling.
3. It is the foundation for many kinds of business applications for Business Performance Management, Planning, Budgeting, Forecasting, Financial Reporting, Analysis, Simulation Models, Knowledge Discovery, and Data Warehouse Reporting.
4. OLAP enables end-users to perform ad hoc analysis of data in multiple dimensions, thereby providing the insight and understanding they need for better decision making.

Types of OLAP:

1. Relational OLAP (ROLAP)
2. Multidimensional OLAP (MOLAP)
3. Hybrid OLAP (HOLAP)

ROLAP: This methodology relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. Each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement.

ROLAP Advantages:

1. Can handle large amounts of data: The data size limitation of ROLAP technology is the limitation on data size of the underlying relational database. In other words, ROLAP itself places no limitation on data amount.
2. Can leverage functionalities inherent in the relational database: Often, relational database already comes with a host of functionalities. ROLAP technologies, since they sit on top of the relational database, can therefore leverage these functionalities.

ROLAP Disadvantages:

1. Performance can be slow: Because each ROLAP report is essentially a SQL query (or multiple SQL queries) in the relational database, the query time can be long if the underlying data size is large.
2. Limited by SQL functionalities: Because ROLAP technology mainly relies on generating SQL statements to query the relational database, and SQL statements do not fit all needs (for example, it is difficult to perform complex calculations using SQL), ROLAP technologies are therefore traditionally limited by what SQL can do.

MOLAP: This is the more traditional way of OLAP analysis. In MOLAP, data is stored in a multidimensional cube. The storage is not in the relational database, but in proprietary formats.

MOLAP Advantages:

1. Excellent performance: MOLAP cubes are built for fast data retrieval, and are optimal for slicing and dicing operations.
2. Can perform complex calculations: All calculations have been pre-generated when the cube is created. Hence, complex calculations are not only doable, but they return quickly.

MOLAP Disadvantages:

1. Limited in the amount of data it can handle: Because all calculations are performed when the cube is built, it is not possible to include a large amount of data in the cube itself. This is not to say that the data in the cube cannot be derived from a large amount of data. Indeed, this is possible. But in this case, only summary-level information will be included in the cube itself.
2. Requires additional investment: Cube technology are often proprietary and do not already exist in the organization. Therefore, to adopt MOLAP technology, chances are additional investments in human and capital resources are needed.

HOLAP: HOLAP technologies attempt to combine the advantages of MOLAP and ROLAP. For summary-type information, HOLAP leverages cube technology for faster performance. When detail information is needed, HOLAP can "drill through" from the cube into the underlying relational data

Difference Between OLAP and OLTP

#	Data Warehouse (OLAP)	Operational Database(OLTP)
1	It involves historical processing of information.	It involves day-to-day processing.
2	OLAP systems are used by knowledge workers such as executives, managers, and analysts.	OLTP systems are used by clerks, DBAs, or database professionals.
3	It is used to analyze the business.	It is used to run the business.
4	It focuses on Information out.	It focuses on Data in.
5	It is based on Star Schema, Snowflake Schema, and Fact Constellation Schema.	It is based on Entity Relationship Model.
6	It focuses on Information out.	It is application oriented.
7	It contains historical data.	It contains current data.
8	It provides summarized and consolidated data.	It provides primitive and highly detailed data.
9	It provides summarized and multidimensional view of data.	It provides detailed and flat relational view of data.
10	The number of users is in hundreds.	The number of users is in thousands.
11	The number of records accessed is in millions.	The number of records accessed is in tens.
12	The database size is from 100GB to 100 TB.	The database size is from 100 MB to 100 GB.
13	These are highly flexible.	It provides high performance.