

Unit 1 : Enhanced Entity Relationship Model and Relational Model

1.1 Entity relationship model Revised

1.1.1 Introduction to ER Model

ER Model is a high-level data model, developed by Chen in 1976. This model defines the data elements and relationships for a specified system. It is useful in developing a conceptual design for the database & is very simple and easy to design logical view of data.

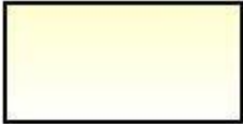
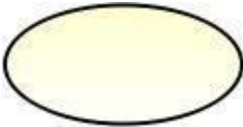
1.1.2 Importance of ER Model

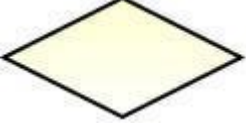

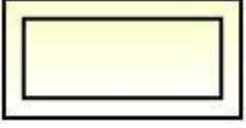
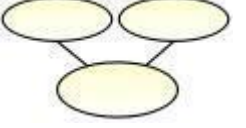

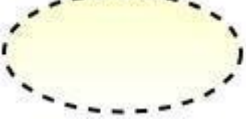

- ER Model is plain and simple for designing the structure.
- It saves time.
- Without ER diagrams you cannot make a database structure & write production code.
- It displays the clear picture of the database structure.

1.1.3 ER Diagrams

- ERD stands for Entity Relationship diagram.
- It is a graphical representation of an information system.
- ER diagram shows the relationship between objects, places, people, events etc. within that system.
- It is a data modeling technique which helps in defining the business process.
- It used for solving the design problems.

Following are the components of ER Diagram,

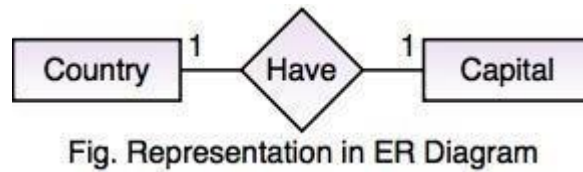
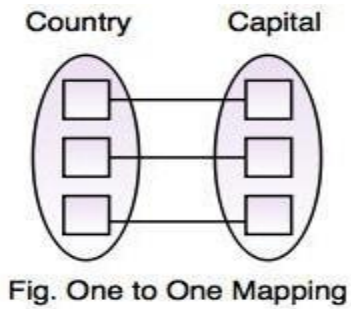
Notations	Representation	Description
	Rectangle	It represents the Entity.
	Ellipse	It represents the Attribute.

	Diamond	It represents the Relationship.
	Line	It represents the link between attribute and entity set to relationship set.
	Double Rectangle	It represents the weak entity.
	Composite Attribute	It represents composite attribute which can be divided into subparts For eg. Name can be divided into First Name and Last Name.
	Multi valued Attribute	It represents multi valued attribute which can have many values for a particular entity. For eg. Mobile Number.
	Derived Attribute	It represents the derived attribute which can be derived from the value of related attribute.
	Key Attribute	It represents key attribute of an entity which have a unique Value in a table. For eg. Employee → EmpId (Employee Id is Unique).

1.1.4 Types of Relationship Mapping

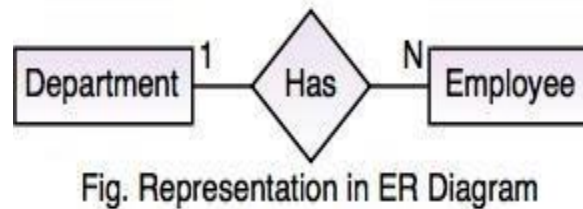
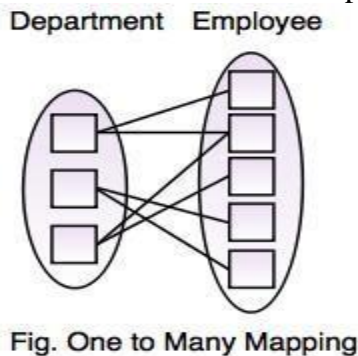
1. One - to - One Relationship

- In One - to - One Relationship, one entity is related with only one other entity.
- One row in a table is linked with only one row in another table and vice versa.
For example: A Country can have only one Capital City.



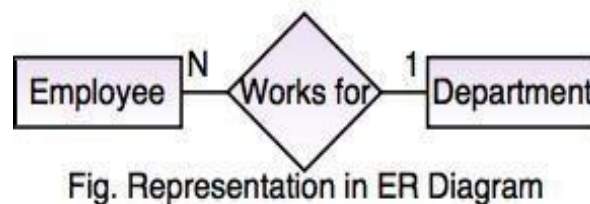
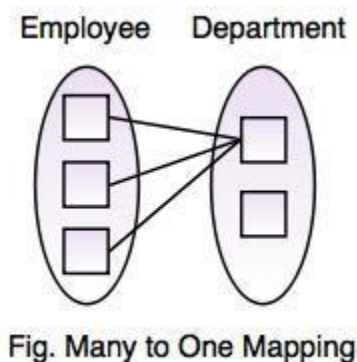
2. One - to - Many Relationship

- In One - to - Many Relationship, one entity is related to many other entities.
- One row in a table A is linked to many rows in a table B, but one row in a table B is linked to only one row in table A. For example: One Department has many Employees.



3. Many - to - One Relationship

- In Many - to - One Relationship, many entities can be related with only one other entity. For example: No. of Employee works for Department.
- Multiple rows in Employee table is related with only one row in Department table.



4. Many - to - Many Relationship

- In Many - to - Many Relationship, many entities are related with the multiple other entities.
- This relationship is a type of cardinality which refers the relation between two entities. For example: Various Books in a Library are issued by many Students.

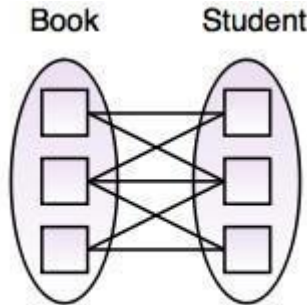


Fig. Many to Many Mapping

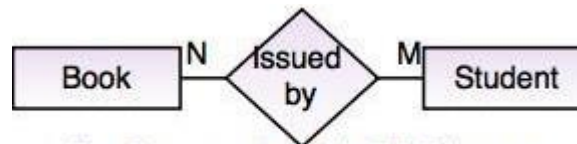


Fig. Representation in ER Diagram

1.2 Enhanced Entity Relationship Model (EER Model)

1.2.1 EER Model

EER is a high-level data model that incorporates the extensions to the original ER model. It is a diagrammatic technique for displaying the following concepts


- Sub Class and Super Class
- Specialization and Generalization
- Union or Category
- Aggregation

These concepts are used when they come in EER schema and the resulting schema diagrams are called as EER Diagrams.

1.2.2. Features of EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different entity types.

1.3 Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.
- The relationship between sub class and super class is denoted with  symbol.

1. Super Class

- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class.
For example: Shape super class is having sub groups as Square, Circle, Triangle.

2. Sub Class

- Sub class is a group of entities with unique attributes.
- Sub class inherits properties and attributes from its super class.
For example: Square, Circle, Triangle are the sub class of Shape super class.

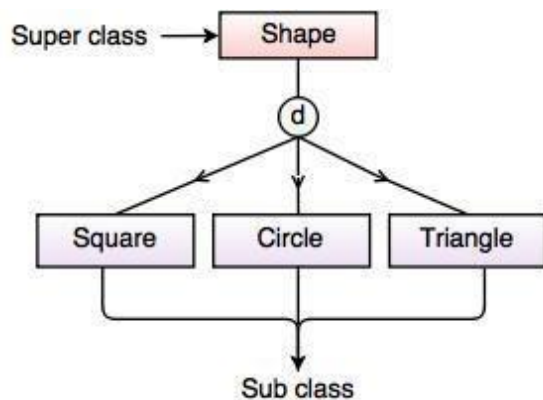


Fig. Super class/Sub class Relationship

1.4. Specialization and Generalization

1. Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom up approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features.
For example:

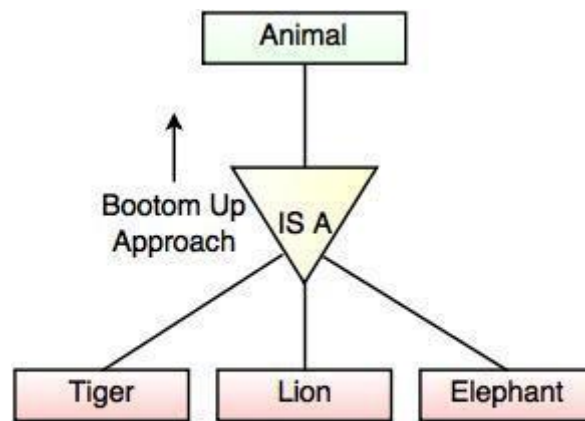


Fig. Generalization

In the above example, Tiger, Lion, Elephant can all be generalized as Animals.

2. Specialization

- Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.
- It is a top down approach, in which one higher entity can be broken down into two lower level entity.
- It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

For example

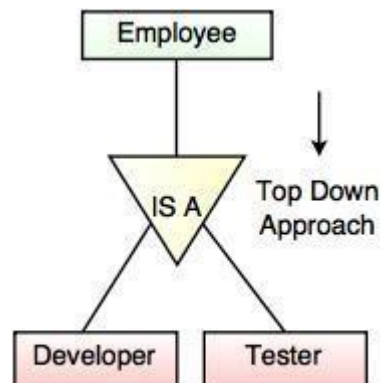


Fig. Specialization

In the above example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization

1.5 Constraints on specialization/generalization

There are three constraints that may apply to a specialization/generalization: membership constraints, disjoint constraints and completeness constraints.

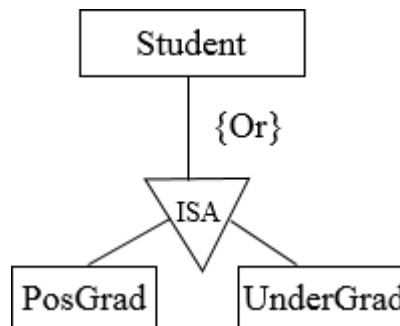
- **Membership constraints**

Condition defined: Membership of a specialization/generalization relationship can be defined as a condition in the requirements e.g. tanker is a ship where cargo = "oil".

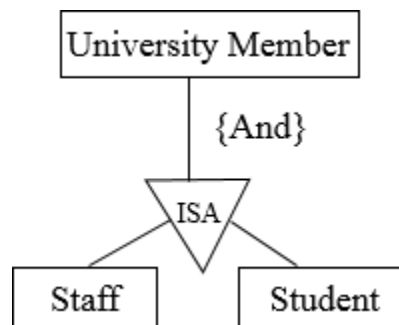
User defined: Sometimes the designer can define the superclass-subclass relationship. This can be done to simplify the design model or represent a complex relationship that exists between entities.

- **Disjoint constraints**

Disjoint: The disjoint constraint only applies when a superclass has more than one subclass. If the subclasses are disjoint, then an entity occurrence can be a member of only one of the subclasses, e.g. postgrads or undergrads — you cannot be both. To represent a disjoint superclass/subclass relationship, "Or" is used.

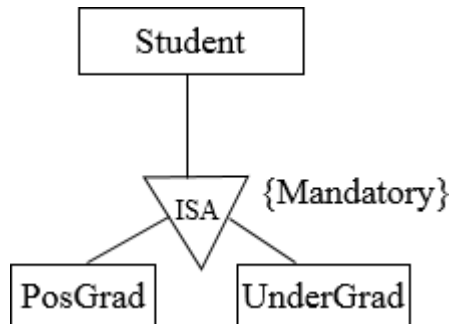


Overlapping: This applies when an entity occurrence may be a member of more than one subclass, e.g. student and staff — some people are both. "And" is used to represent the overlapping specialization/generalization relationship in the ER diagram.



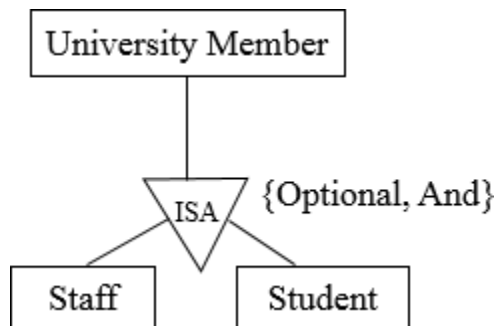
- **Completeness constraints**

Total: Each superclass (higher-level entity) must belong to subclasses (lower-level entity sets), e.g. a student must be postgrad or undergrad. To represent completeness in the specialization/generalization relationship, the keyword “Mandatory”™ is used.

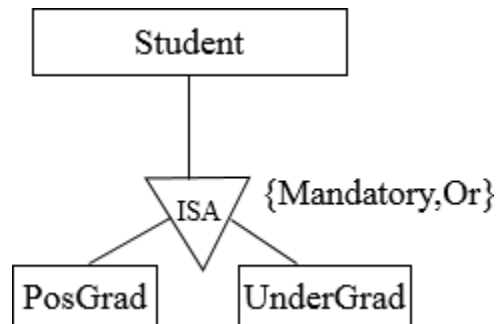


Partial: Some superclasses may not belong to subclasses (lower-level entity sets), e.g. some people at UCT are neither student nor staff. The keyword “Optional”™ is used to represent a partial specialization/generalization relationship.

We can show both disjoint and completeness constraints in the ER diagram. Following our examples, we can combine disjoint and completeness constraints.



Some members of a university are both students and staff. Not all members of the university are staff and students.



Difference between Generalization and Specialization :

GENERALIZATION

1. Generalization works in Bottom-Up approach.
2. In Generalization, size of schema gets reduced.
3. Generalization is normally applied to to group of entity
4. Generalization can be defined as a process of creating groupings from various entity sets
5. In Generalization, the difference and similarities between lower entities are ignored to form a higher entity
6. There is no inheritance in Generalization

SPECIALIZATION

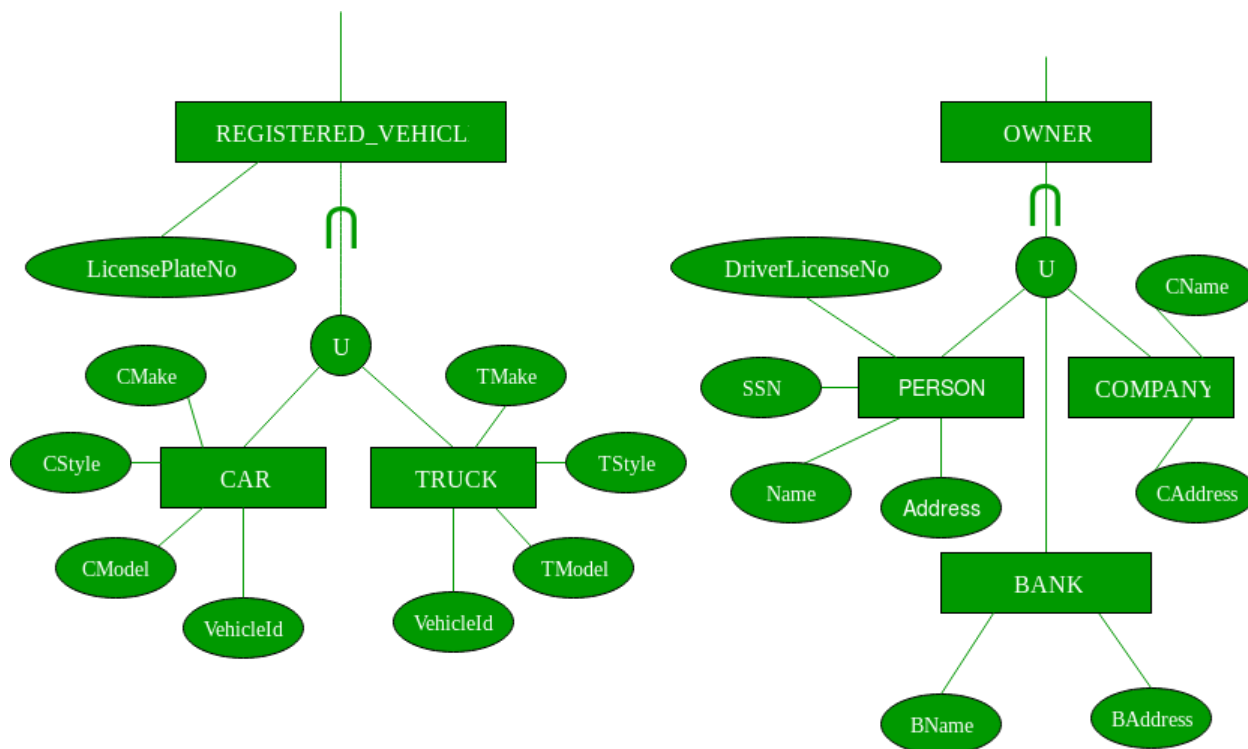
1. Specialization works in top-down approach.
2. In Specialization, size of schema gets increased.
3. We can apply Specialization to a single entity
4. Specialization can be defined as process of creating subgrouping within an entity set
- 5 In Specialization, a higher entity is split to form lower entities.
- 6There is inheritance in Specialization.

1.6 Union type or category

- Represents a single superclass/subclass relationship with more than one superclass
- Subclass represents a collection of objects that is a subset of the UNION of distinct entity types
- Attribute inheritance works more selectively
- Category can be total or partial

Set of Library Members is UNION of Faculty, Student, and Staff. A union relationship indicates either type; for example, a library member is either Faculty or Staff or Student.

- Below are two examples that show how UNION can be depicted in ERD – Vehicle Owner is UNION of PERSON and Company, and RTO Registered Vehicle is UNION of Car and Truck.



You might see some confusion in Sub-class and UNION; consider an example in above figure Vehicle is super-class of CAR and Truck; this is very much the correct example of the subclass as well but here use it differently we are saying RTO Registered vehicle is UNION of Car and Vehicle, they do not inherit any attribute of Vehicle, attributes of car and truck are altogether independent set, where is in sub-classing situation car and truck would be inheriting the attribute of vehicle class

1.7 . Aggregation

- Aggregation is a process that represent a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.
- It is a process when two entity is treated as a single entity.

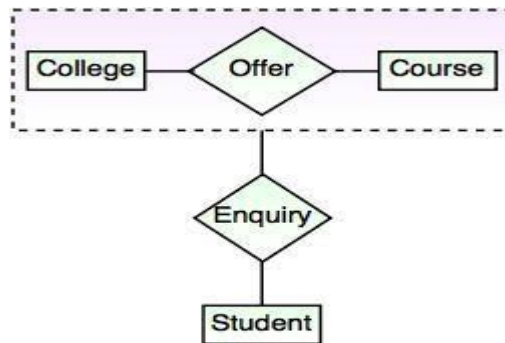


Fig. Aggregation

1.5 Relational Model Revised

Relational Model (RM) represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts in DBMS

1. Attribute: Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME, etc.
2. Tables – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3. Tuple – It is nothing but a single row of a table, which contains a single record.
4. Relation Schema: A relation schema represents the name of the relation with its attributes.
5. Degree: The total number of attributes which in the relation is called the degree of the relation.
6. Cardinality: Total number of rows present in the Table.
7. Column: The column represents the set of values for a specific attribute.
8. Relation instance – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. Relation key – Every row has one, two or multiple attributes, which is called relation key.
10. Attribute domain – Every attribute has some pre-defined value and scope which is known as attribute domain

Table also called Relation

Primary Key Domain
Ex: NOT NULL

@ guru99.com

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Tuple OR Row
Total # of rows is **Cardinality**

Column OR Attributes
Total # of column is **Degree**

1.9. Converting ER and EER model to Relational model

Converting ER model into Relational model

From the real world description of the organization, we were able to identify the following entities. These entities will become the basis for an entity-relationship diagram or model.

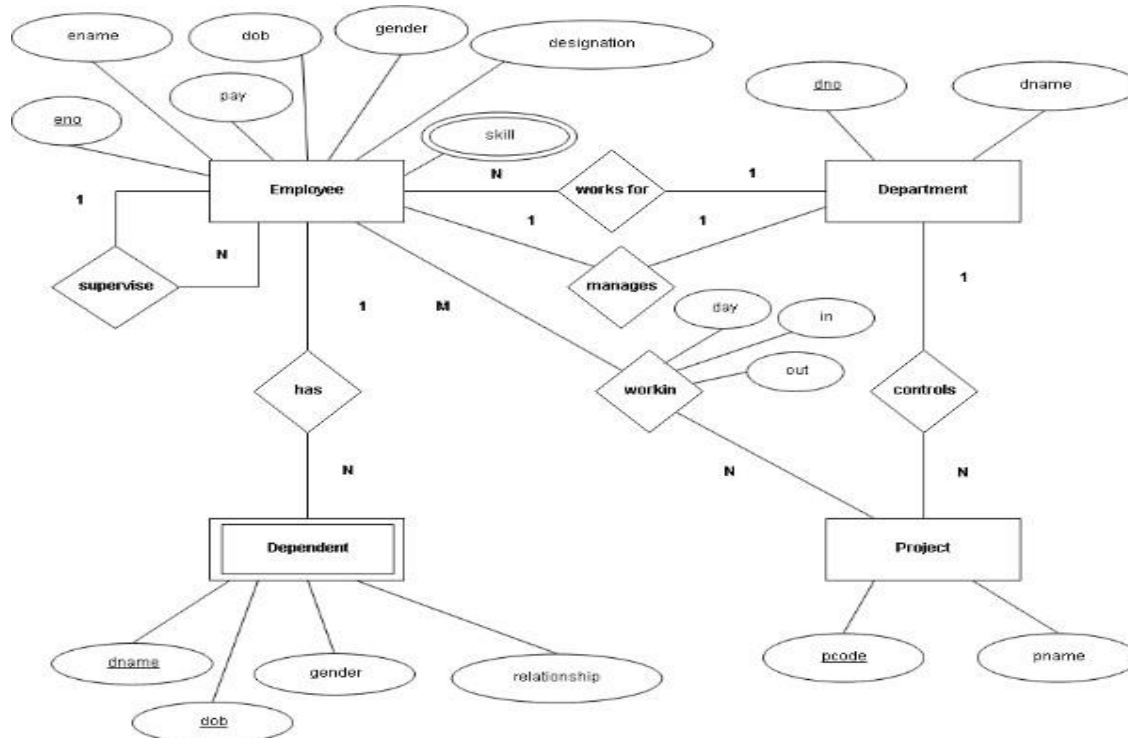
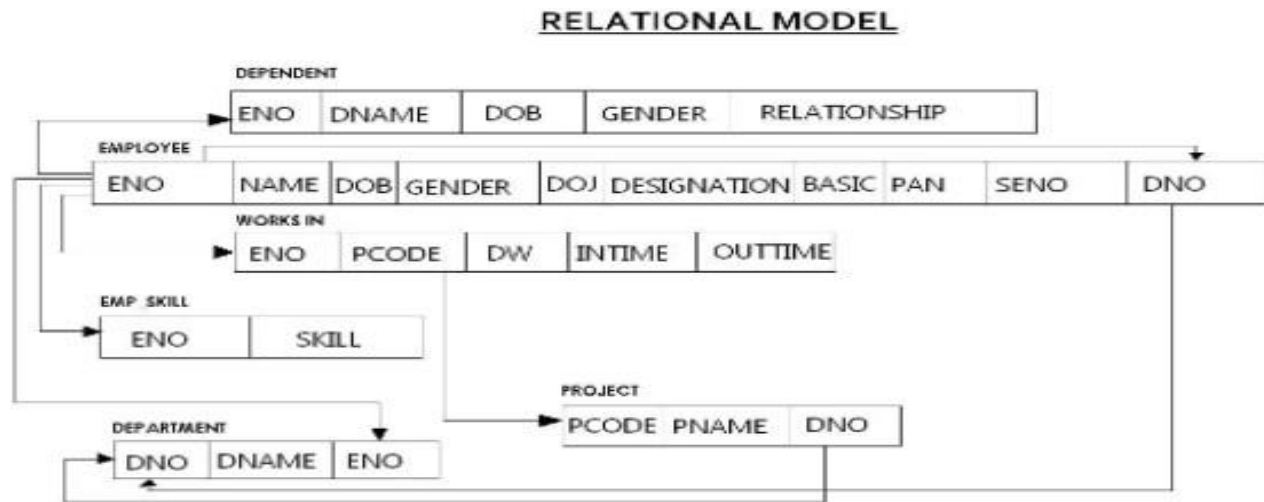


Figure : Converting ER model into Relational model

Convert to Relational Model

The next step in the database design is to convert the ER Model into the Relational Model. In the Relational Model, we will define the schema for relations and their relationships. The attributes from the entity-relationship diagram will become fields for a relationship and one of them is a primary field or primary key. It is usually underlined in the entity-relationship diagram. An entity in a relational model is a relation. For example, the entity Dependent is a relation in the relational model with all the attributes as fields – eno, dname, dob, gender, and relationship. Here is the Relational Model for above diagram of the company database. This is the result after converting ER model into relational model.



Relational Model of Database

1.9. SQL And Advance Features

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways,

Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

1.10.1 SQL contains of some important features and they are:

1. Data Definition language (DDL):

It contains of commands which defines the data. The commands are:

- create: It is used to create a table.

Syntax:

create table tablename(attribute1 datatype attributen datatype);

- drop: It is used to delete the table including all the attributes.

Syntax:

drop table tablename;

- alter: alter is a reserve word which modifies the structure of the table.

Syntax:

alter table

tablename add(new column1 datatype new columnx datatype);

- rename: A table name can be changed using the reserver `_rename`

Syntax:

rename old table name to new table name;

2. Data Manipulation Language (DML):

Data Manipulation Language contains commands used to manipulate the data.

The commands are:

- insert: This command is generally used after the create command to insert a set of values into the table. Syntax:

insert into tablename values(attribute1 datatype);

- delete: A command used to delete particular tuples or rows or cardinality from the table.

Syntax:

delete from tablename where condition;

- update: It updates the tuples in a table.

- Syntax:

update tablename set tuplename='attributename';

- **Triggers:**

Triggers are actions performed when certain conditions are met on the data.

A trigger contains of three parts.

- (i). event – The change in the database that activates the trigger is event.
- (ii). condition – A query or test that is run when the trigger is activated.
- (iii). action – A procedure that is executed when trigger is activated and the condition met is true.

3. Client server execution and remote database access:

Client server technology maintains a many to one relationship of clients (many) and server (one). We have commands in SQL that control how a client application can access the database over a network.

4 .Security and authentication:

SQL provides a mechanism to control the database meaning it makes sure that only the particular details of the database is to be shown the user and the original database is secured by DBMS.

5. Embedde SQL:

SQL provides the feature of embedding host languages such as C, COBOL, Java for query from their language at runtime.

6. Transaction Control Language:

Transactions are an important element of DBMS and to control the transactions, TCL is used which has commands like commit, rollback and savepoint.

- commit: It saves the database at any point whenever database is consistent.
Syntax: commit;
- rollback: It rollbacks/undo to the previous point of the transaction.

Syntax
rollback;

- savepoint: It goes back to the previous transaction without going back to the entire transaction.
Syntax:
savepoint;

SQL programming Language site lists commonly-used SQL statements, and is divided into the following sections:

- SQL Commands
- Advanced SQL

In this section, we discuss about the following Advance SQL keywords and concepts:

- SQL UNION
- SQL UNION ALL
- SQL Inline View
- SQL INTERSECT
- SQL MINUS
- SQL LIMIT
- SQL TOP
- SQL Subquery
- SQL EXISTS
- SQL CASE
- SQL Window Functions
- SQL DECODE
- SQL AUTO INCREMENT
- SQL IDENTITY
- SQL SEQUENCE and NEXTVAL

1 SQL Union

- The purpose of the SQL UNION query is to combine the results of two queries together while removing duplicates. In other words, when using UNION, only unique values are returned (similar to SELECT DISTINCT).
- Syntax
- The syntax of UNION in SQL is as follows:
- [SQL Statement1]
UNION
[SQL Statement 2];

Example

We use the following tables for our example.

Table *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	Jan-05-1999
San Diego	250	Jan-07-1999
Los Angeles	300	Jan-08-1999
Boston	700	Jan-08-1999

Txn_Date	Sales
Jan-07-1999	250
Jan-10-1999	535
Jan-11-1999	320
Jan-12-1999	750

Table *Internet_Sales*

To find all the dates where there is a sales transaction, we use the following SQL statement:

```
SELECT Txn_Date FROM Store_Information
```

```
UNION
```

```
SELECT Txn_Date FROM Internet_Sales;
```

Result:

Txn_Date

Jan-05-1999

Jan-07-1999

Jan-08-1999

Jan-10-1999

Jan-11-1999

Jan-12-1999

2 SQL Union All

The purpose of the SQL UNION ALL command is to combine the results of two queries together without removing any duplicates.

Syntax The syntax for UNION ALL is as follows:

[SQL Statement 1]

UNION ALL

[SQL Statement 2];

Example

We use the following tables for our example.

Table *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	Jan-05-1999
San Diego	250	Jan-07-1999
Los Angeles	300	Jan-08-1999
Boston	700	Jan-08-1999

Table *Internet_Sales*

Txn_Date	Sales
Jan-07-1999	250

Jan-10-1999	535
Jan-11-1999	320
Jan-12-1999	750

To find out all the dates where there is a sales transaction at a store as well as all the dates where there is a sale over the internet, we use the following SQL statement:

```
SELECT Txn_Date FROM Store_Information
UNION ALL
SELECT Txn_Date FROM Internet_Sales;
```

Result:

Txn_Date

Jan-05-1999

Jan-07-1999

Jan-08-1999

Jan-08-1999

Jan-07-1999

Jan-10-1999

Jan-11-1999

Jan-12-1999

2.1 UNION vs UNION ALL

UNION and UNION ALL both combine the results of two SQL queries. The difference is that, while UNION only returns distinct values, UNION ALL selects all values. If we use UNION in the above example,

```
SELECT Txn_Date FROM Store_Information
```

```
UNION
```

```
SELECT Txn_Date FROM Internet_Sales;
```

the result becomes,

Txn Date

Jan-05-1999

Jan-07-1999

Jan-08-1999

Jan-10-1999

Jan-11-1999

Jan-12-1999

3 SQL Inline View

An inline view is a SELECT statement in the FROM clause. As mentioned in the View section, a view is a virtual table that has the characteristics of a table yet does not hold any actual data. In an inline view construct, instead of specifying table name(s) after the FROM keyword, the source of the data actually comes from the inline view. Inline view is sometimes referred to as derived table. These two terms are used interchangeably.

Syntax

The syntax for an inline view is,

SELECT "column_name" FROM (Inline View);

Example

Without using an inline view, we can accomplish this in two steps:

Query 1

```
CREATE TABLE User_Higher_Than_200
SELECT User_ID, SUM(Score) FROM User_Score
GROUP BY User_ID
HAVING SUM(Score) > 200;
```

Query 2

```
SELECT a2.ZIP_CODE, COUNT(a1.User_ID)
FROM User_Higher_Than_200 a1, User_Address a2
WHERE a1.User_ID = a2.User_ID
GROUP BY a2.ZIP_CODE;
```

In the above code, we introduced a temporary table, *User_Higher_Than_200*, to store the list of users who scored higher than 200. *User_Higher_Than_200* is then used to join to the *User_Address* table to get the final result.

We can simplify the above SQL using the inline view construct as follows:

Query 3

```
SELECT a2.ZIP_CODE, COUNT(a1.User_ID)
```

```
FROM
```

```
(SELECT User_ID, SUM(Score) FROM User_Score GROUP BY User_ID HAVING  
SUM(Score) > 200) a1,
```

```
User_Address a2
```

```
WHERE a1.User_ID = a2.User_ID
```

```
GROUP BY a2.ZIP_CODE;
```

4 SQL Intersect

The INTERSECT command in SQL combines the results of two SQL statement and returns only data that are present in both SQL statements.

INTERSECT can be thought of as an AND operator (value is selected only if it appears in both statements), while UNION and UNION ALL can be thought of as an OR operator (value is selected if it appears in either the first or the second statement).

Syntax

The syntax for INTERSECT is as follows:

```
[SQL Statement 1]
```

```
INTERSECT
```

```
[SQL Statement 2];
```

Example Table *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	Jan-05-1999
San Diego	250	Jan-07-1999
Los Angeles	300	Jan-08-1999
Boston	700	Jan-08-1999

Table *Internet_Sales*

Txn_Date	Sales
Jan-07-1999	250
Jan-10-1999	535
Jan-11-1999	320
Jan-12-1999	750

To find out all the dates where there are both store sales and internet sales, we use the following SQL statement:

```
SELECT Txn_Date FROM Store_Information
```

```
INTERSECT
```

```
SELECT Txn_Date FROM Internet_Sales;
```

Result:

Txn_Date

Jan-07-1999

5 SQL Minus

The MINUS command operates on two SQL statements. It takes all the results from the first SQL statement, and then subtract out the ones that are present in the second SQL statement to get the final result set. If the second SQL statement includes results not present in the first SQL statement, such results are ignored.

Syntax

The syntax for MINUS is as follows:

[SQL Statement 1]

MINUS

[SQL Statement 2];

Example

We use the following tables for our example

Table *Store_Information*

Store_Name	Sales	Txn_Date
Los Angeles	1500	Jan-05-1999
San Diego	250	Jan-07-1999
Los Angeles	300	Jan-08-1999
Boston	700	Jan-08-1999

Table *Internet_Sales*

Txn_Date	Sales
Jan-07-1999	250
Jan-10-1999	535
Jan-11-1999	320
Jan-12-1999	750

To find all the dates where there are store sales but no internet sales, we use the following SQL statement:

```
SELECT Txn_Date FROM Store_Information
```

```
MINUS
```

```
SELECT Txn_Date FROM Internet_Sales;
```

Result:

Txn_Date

Jan-05-1999

Jan-08-1999

6 SQL Limit

The LIMIT clause restricts the number of results returned from a SQL statement. It is available in MySQL.

Syntax

The syntax for LIMIT is as follows:

[SQL Statement 1]

LIMIT [N];

where [N] is the number of records to be returned. Please note that the ORDER BY clause is usually included in the SQL statement. Without the ORDER BY clause, the results we get would be dependent on what the database default is.

Example

```
SELECT Store_Name, Sales, Txn_Date
FROM Store_Information
ORDER BY Sales DESC
LIMIT 2;
```

Result:

<u>Store_Name</u>	<u>Sales</u>	<u>Txn_Date</u>
Los Angeles	1500	Jan-05-1999
Boston	700	Jan-08-1999

7 SQL Top

The TOP keyword restricts the number of results returned from a SQL statement in Microsoft SQL Server. Syntax

The syntax for TOP is as follows:

SELECT TOP [TOP argument] "column_name" FROM "table_name";

where [TOP argument] can be one of two possible types:

1. [N]: The first N records are returned.
2. [M] PERCENT: The number of records corresponding to M% of all qualifying records are returned.

Examples

```
SELECT TOP 2 Store_Name, Sales, Txn_Date
FROM Store_Information
ORDER BY Sales DESC;
```

Result:

<u>Store_Name</u>	<u>Sales</u>	<u>Txn_Date</u>
Los Angeles	1500	Jan-05-1999
Boston	700	Jan-08-1999

Example 2: [TOP argument] is a percentage

To show the top 25% of sales amounts from Table *Store_Information*, we key in,
SELECT TOP 25 PERCENT Store_Name, Sales, Txn_Date

FROM Store_Information

ORDER BY Sales DESC;

Result:

<u>Store_Name</u>	<u>Sales</u>	<u>Txn_Date</u>
-------------------	--------------	-----------------

Los Angeles	1500	Jan-05-1999
-------------	------	-------------

8 SQL Subquery

A subquery is a SQL statement that has another SQL query embedded in the WHERE or the HAVING clause.

Syntax

The syntax for a subquery when the embedded SQL statement is part of the WHERE condition is as follows:

```
SELECT "column_name1"
FROM "table_name1"
WHERE "column_name2" [Comparison Operator]
(SELECT "column_name3"
FROM "table_name2"
WHERE "condition");
```

Example 1: Simple subquery

To use a subquery to find the sales of all stores in the West region, we use the following SQL statement:

```
SELECT SUM (Sales) FROM Store_Information
WHERE Store_Name IN
(SELECT Store_Name FROM Geography
```

```
WHERE Region_Name = 'West');
```

Result:

SUM (Sales)

2050

Example 2: Correlated subquery

If the inner query is dependent on the outer query, we will have a correlated subquery. An example of a correlated subquery is shown below:

```
SELECT SUM (a1.Sales) FROM Store_Information a1
WHERE a1.Store_Name IN
(SELECT Store_Name FROM Geography a2
```

```
WHERE a2.Store_Name = a1.Store_Name);
```

Result:

SUM (Sales)

2750

9 SQL Exists

EXISTS is a Boolean operator used in a sub query to test whether the inner query returns any row. If it does, then the outer query proceeds. If not, the outer query does not execute, and the entire SQL statement returns nothing.

Syntax

The syntax for EXISTS is:

```
SELECT "column_name1"
FROM "table_name1"
WHERE EXISTS
(SELECT *
FROM "table_name2"
```

```
WHERE "condition");
```

Example

```
SELECT SUM(Sales) FROM Store_Information
WHERE EXISTS
(SELECT * FROM Geography
WHERE Region_Name = 'West');
```

produces the result below:

SUM(Sales)

2750

10 SQL Case

CASE is used to provide if-then-else type of logic to SQL. There are two formats: The first is a Simple CASE expression, where we compare an expression to static values. The second is a Searched CASE expression, where we compare an expression to one or more logical conditions.

Simple CASE Expression Syntax

The syntax for a simple CASE expression is:

```
SELECT CASE ("column_name")
```

```
  WHEN "value1" THEN "result1"
```

```
  WHEN "value2" THEN "result2"
```

```
  [ELSE "resultN"]
```

```
END
```

```
FROM "table_name";
```

Example

```
SELECT Store_Name, CASE Store_Name
```

```
  WHEN 'Los Angeles' THEN Sales * 2
```

```
  WHEN 'San Diego' THEN Sales * 1.5
```

```
  ELSE Sales
```

```
END
```

```
"New Sales",
```

```
Txn_Date
```

```
FROM Store_Information;
```

Result:

<u>Store_Name</u>	<u>New Sales</u>	<u>Txn_Date</u>
Los Angeles	3000	Jan-05-1999
San Diego	375	Jan-07-1999
San Francisco	300	Jan-08-1999
Boston	700	Jan-08-1999

Searched CASE Expression Syntax

The syntax for a searched CASE expression is:

SELECT CASE

WHEN "condition1" THEN "result1"

WHEN "condition2" THEN "result2"

...

[ELSE "resultN"]

END

FROM "table_name";

SELECT Store_Name, Txn_Date, CASE

WHEN Sales >= 1000 THEN 'Good Day'

WHEN Sales >= 500 THEN 'OK Day'

ELSE 'Bad Day'

END

"Sales Status"

FROM Store_Information;

Result:

<u>Store_Name</u>	<u>Txn_Date</u>	<u>Sales Status</u>
Los Angeles	Jan-05-1999	Good Day
San Diego	Jan-07-1999	Bad Day
San Francisco	Jan-08-1999	Bad Day

Boston Jan-08-1999 OK Day

11 SQL Window Functions

With aggregate functions, you usually get a single result for each group that you look at (as specified by the columns in the GROUP BY clause). However, there are times when you want to apply the aggregate function to each row, and this is where window functions come in.

Window functions are called such because they operate on each row within a window. A window is defined by the OVER() clause. There are two major components in the OVER() construct:

- **PARTITION BY:** Defines the window partition into groups of rows. This is similar to the GROUP BY clause. Note that this is optional. You do not need to specify PARTITION BY if your window covers the entire data set.
- **ORDER BY:** Orders the value within each window partition. This is optional, although in most of the use cases, it makes sense to order your results in some way.

There are a number of aggregate functions that can act as a window function. The following are the common aggregate functions used as part of a window function:

- Common aggregate functions: SUM, AVG, COUNT
- Ranking functions: RANK, DENSE_RANK, ROW_NUMBER
- Grouping function: NTILE
- Before- and after-functions: LEAD, LAG

We go through each type of functions below:

Common Aggregate Functions: SUM, AVG, COUNT

We use the following table for our examples.

Table *Store_Sales*

Store_ID	Salesperson	Sales
1	Aaron	374
1	Beatrice	492
1	Cathy	430

2	Dan	462
2	Elmo	747
2	Frank	1332
2	Gina	898
2	Harry	603
3	Isabel	247
3	Jimmy	1030
3	Kara	1030
3	Lamar	1314
3	Mary	462

To list the average sales amount of the store each salesperson belongs to, we use the following window function:

```
SELECT Store_ID, Salesperson, Sales, AVG(Sales) OVER (PARTITION BY Store_ID)
Avg_Store
FROM Store_Sales;
```

Result:

<u>Store_ID</u>	<u>Salesperson</u>	<u>Sales</u>	<u>Avg_Store</u>
1	Aaron	374	432
1	Beatrice	492	432
1	Cathy	430	432
2	Dan	462	808.4
2	Elmo	747	808.4
2	Frank	1332	808.4
2	Gina	898	808.4
2	Harry	603	808.4

3	Isabel	247	816.6
3	Jimmy	1030	816.6
3	Kara	1030	816.6
3	Lamar	1314	816.6
3	Mary	462	816.6

12 SQL DECODE Function

DECODE is a function in Oracle and is used to provide if-then-else type of logic to SQL. It is not available in MySQL or SQL Server.

Syntax

The syntax for DECODE is:

```
SELECT DECODE ( "column_name", "search_value_1", "result_1",  
["search_value_n", "result_n"],  
{ "default_result" } );
```

Example

```
SELECT DECODE (Store_Name,  
  
'Los Angeles', 'LA',  
  
'San Francisco', 'SF',  
  
'San Diego', 'SD',  
  
'Others') Area, Sales, Txn_Date  
  
FROM Store_Information;
```

Result:

<u>Area</u>	<u>Sales</u>	<u>Txn Date</u>
LA	1500	Jan-05-1999
SD	250	Jan-07-1999

SF 300 Jan-08-1999

Others 700 Jan-08-1999

13 SQL AUTO_INCREMENT

AUTO_INCREMENT is used in MySQL to create a numerical primary key value for each additional row of data.

Syntax

The syntax for AUTO_INCREMENT is as follows:

```
CREATE TABLE TABLE_NAME  
(PRIMARY_KEY_COLUMN INT NOT NULL AUTO_INCREMENT  
PRIMARY KEY (PRIMARY_KEY_COLUMN));  
CREATE TABLE USER_TABLE
```

```
(Userid int NOT NULL AUTO_INCREMENT,
```

```
Last_Name varchar(50),  
First_Name varchar(50),  
PRIMARY KEY (Userid));  
Upon creation, there is no data in this table.
```

We insert the first value:

```
INSERT INTO USER_TABLE VALUES ('Perry', 'Jonathan');
```

Now the table contains the following data:

Table *USER_TABLE*

Userid	Last_Name	First_Name
1	Perry	Jonathan

We then insert the second value:

```
INSERT INTO USER_TABLE VALUES ('Davis', 'Nancy');
```

Now the table has the following values:

Table *USER_TABLE*

Userid	Last_Name	First_Name
1	Perry	Jonathan
2	Davis	Nancy

Notice when we insert the first row, *Userid* is set to 1. When we insert the second row, *Userid* increases by 1 and becomes 2.

By default, `AUTO_INCREMENT` starts with 1 and increases by 1. To change the default starting value, we can use the `ALTER TABLE` command as follows:

```
ALTER TABLE TABLE_NAME AUTO_INCREMENT = [New Number];
```

where [New Number] is the starting value we want to use.

The `AUTO INCREMENT` interval value is controlled by the MySQL Server variable `auto_increment_increment` and applies globally. To change this to a number different from the default of 1, use the following command in MySQL:

```
mysql> SET @@auto_increment_increment = [interval number];
```

where [interval number] is the interval value we want to use. So, if we want to set the interval to be 5, we would issue the following command:

```
mysql> SET @@auto_increment_increment = 5;
```

14 SQL IDENTITY

`IDENTITY` is used in Microsoft SQL Server to automatically insert numerical primary key values to a table as new data is inserted. This is similar to the `AUTO INCREMENT` command in MySQL.

Syntax

The syntax for `IDENTITY` is as follows:

```
CREATE TABLE TABLE_NAME
```

```
(PRIMARY_KEY_COLUMN INT PRIMARY KEY IDENTITY ( [Initial_Value], [Interval] ),  
...);
```

Example

```
CREATE TABLE USER_TABLE
```

```
(Userid int PRIMARY KEY IDENTITY(2,1),
```

```
Last_Name nvarchar(50),
```

```
First_Name nvarchar(50));
```

Upon creation, the table is empty.

We insert the first value:

```
INSERT INTO USER_TABLE VALUES ('Washington', 'George');
```

Now the table has the following values:

Table *USER_TABLE*

Userid	Last_Name	First_Name
2	Washington	George

Userid is 2 because we had specified the initial value to be 2.

Next we insert the second value:

```
INSERT INTO USER_TABLE VALUES ('Jefferson', 'Thomas');
```

Now the table has the following values:

Table *USER_TABLE*

Userid	Last_Name	First_Name
2	Washington	George
3	Jefferson	Thomas

userid for the second row is 3 because it is 1 larger than the previous Userid, which is 2.

15 SQL SEQUENCE And NEXTVAL

Oracle uses the concept of SEQUENCE to create numerical primary key values as we add rows of data into a table. Whereas numerical primary key population for MySQL and SQL Server is tied to individual tables, in Oracle the SEQUENCE construct is created separately and is not tied to an individual table.

Syntax

The syntax for creating a sequence in Oracle is:

```
CREATE SEQUENCE SEQUENCE_NAME
```

```
[START WITH {Initial_Value}]
```

```
[INCREMENT BY {interval}];
```

Example

Assume we have a table with the following structure:

Table *USER_TABLE*

Userid	integer
Last_Name	varchar(50)
First_Name	varchar(50)

and we want to use the following sequence to generate the userid:

```
CREATE SEQUENCE SEQ_USER START WITH 5 INCREMENT BY 5;
```

We specify that we want to use the sequence and the NEXTVAL function in the INSERT INTO statements in the following order:

```
INSERT INTO USER_TABLE VALUES (SEQ_USER.NEXTVAL, 'Washington', 'George');
```

```
INSERT INTO USER_TABLE VALUES (SEQ_USER.NEXTVAL, 'Jefferson', 'Thomas');
```

Now the table has the following two rows:

Table *USER_TABLE*

Userid	Last_Name	First_Name
5	Washington	George
10	Jefferson	Thomas

It is worth noting that a sequence is independent of a table. In other words, a sequence can be used to generate primary key values for multiple tables, and the sequence continues even if it is being applied to a different table. So, let's say for example we have a second table, Table *NEW_USERS*, which has the same structure as table *USER_TABLE*, and we issue the following SQL command after executing the two SQL commands above:

```
INSERT INTO NEW_USER VALUES (SEQ_USER.NEXTVAL, 'Adams', 'John');
```

Table *NEW_USER* will have the following row:

Table *NEW_USER*

Userid	Last_Name	First_Name
15	Adams	John

Userid is 15 because that is the next value after 10.

1.10.File Structure

1 What are file structures?

A file structure is a combination of representations for data in files. It is also a collection of operations for accessing the data. It enables applications to read, write, and modify data. File structures may also help to find the data that matches certain criteria. An improvement in file structure has a great role in making applications hundreds of times faster.

The main goal of developing file structures is to minimize the number of trips to the disk in order to get desired information. It ideally corresponds to getting what we need in one disk access or getting it with as little disk access as possible.

1.11.1. File Organization

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

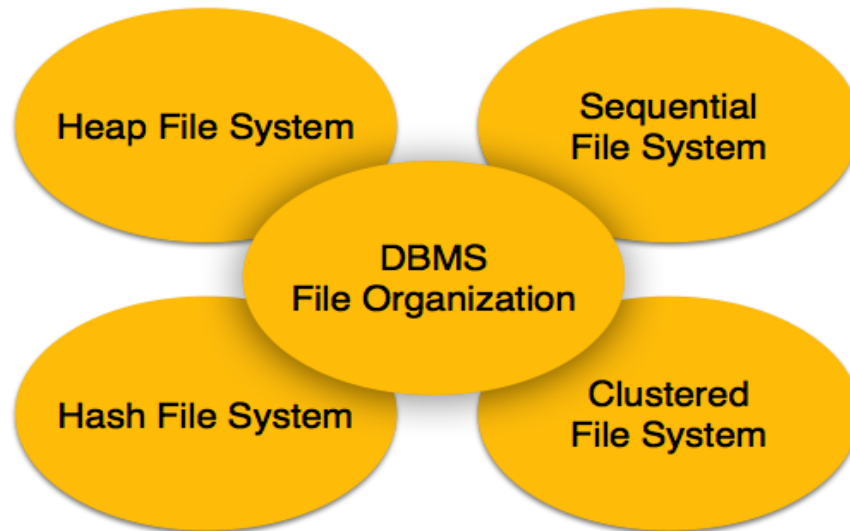


Figure : File Organization

1 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.

2 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.

3 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.

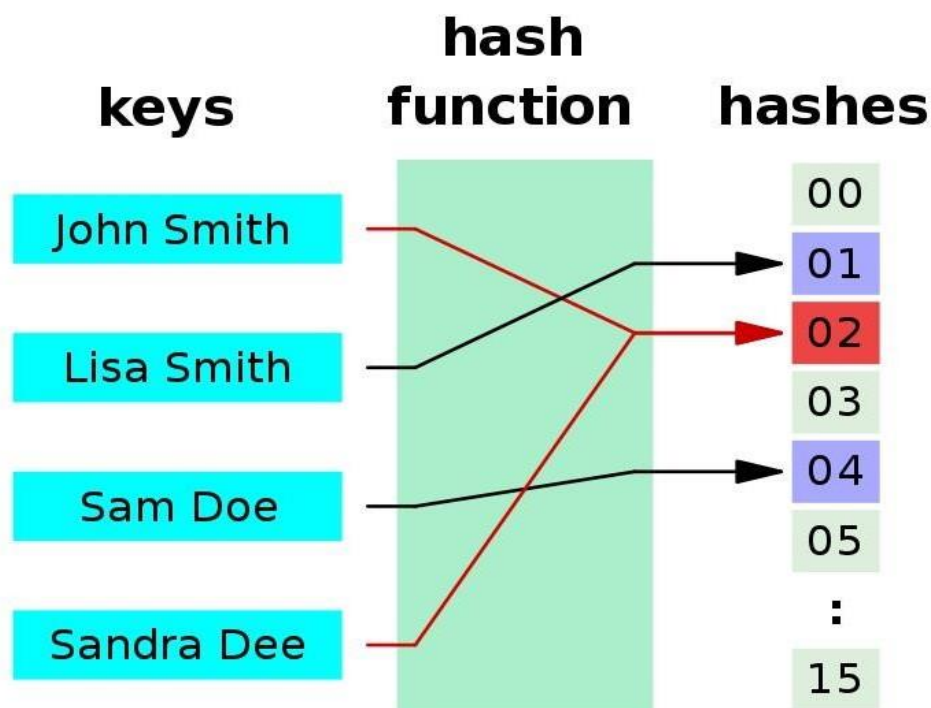
4 Clustered File Organization

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.

1.12 Hashing And Indexing

1.12.1 What is Hashing

In a large database, it is not possible to search all the indexes to obtain the required data. Hashing helps to find the direct location of a specific data record on the disk without using indexing. Here, data blocks, also called data buckets, store data. A hashing function is a mathematical function. It helps to generate the addresses of those data blocks. Furthermore, the hashing function can select any column value to generate the address, but it usually uses the primary key to generate the address of the data block.



There are two types of hashing as static and dynamic hashing. In static hashing, the resultant data bucket address is always the same. However, static hashing causes bucket overflowing. Dynamic hashing is a solution to this issue. In dynamic hashing, data bucket increases or decreases depending on the number of records

1.12.2 What is Indexing

When executing SQL queries, it takes some amount of time to access data from the disk. Herein, an index is a data structure that helps to find and access data in a table of a database quickly. Indexing technique reduces the number of disks accessed to process queries.

An index consists of two sections; a search key and a data reference. The search key contains the primary key or the candidate key of the table. Data reference holds the address of the disk block that has the value corresponding to that key.

Also, there are various types of indexes. Some of them are as follows.

Ordered Indexing – Indices are sorted, making data searching faster

Primary Indexing – When the index is based on the primary key of the table, it is called a primary index. There are two types of indexes in primary key called dense and spare index. The dense index contains an index record for every search key value in the data file. In the spare index, there are index records for some data items.

Clustered indexing – Uses a combination of two or more columns to create an index. A group of records consists of records with the same characteristics. And, these groups create the indexes.

Secondary indexing – Contains another level of indexing to minimize the size of mapping.

Difference between Indexing and Hashing in DBMS :

Indexing

It is a technique that allows to quickly retrieve records from database file.

It is generally used to optimize or increase performance of database simply by minimizing number of disk accesses that are required when a query is processed.

It offers faster search and retrieval of data to users, helps to reduce table space, makes it possible to quickly retrieve or fetch data, can be used for sorting, etc.

Its main purpose is to provide basis for both

Hashing

It is a technique that allows to search location of desired data on disk without using index structure.

It is generally used to index and retrieve items in database as it is faster to search that specific item using shorter hashed key rather than using its original value.

It is faster than searching arrays and lists, provides more flexible and reliable method of data retrieval rather than any other data structure, can be used for comparing two files for quality, etc.

Its main purpose is to use math problem to

Indexing

rapid random lookups and efficient access of ordered records.

It is not considered best for large databases and its good for small databases.

Types of indexing includes ordered indexing, primary indexing, secondary indexing, clustered indexing.

It uses data reference to hold address of disk block.

It is important because it protects file and documents of large size business organizations, and optimize performance of database.

Hashing

organize data into easily searchable buckets.

It is considered best for large databases.

Types of hashing includes static and dynamic hashing.

It uses mathematical functions known as hash function to calculate direct location of records on disk.

It is important because it ensures data integrity of files and messages, takes variable length string or messages and compresses and converts it into fixed length value.

Unit 2 : Object and object Relational Databases

2.1 Object database concept

An **object database** is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object databases are different from relational databases which are table-oriented. Object-oriented database management systems (OODBMSs) also called ODBMS (Object Database Management System) combine database capabilities with object-oriented programming language capabilities. OODBMSs allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS. Because the database is integrated with the programming language, the programmer can maintain consistency within one

environment, in that both the OODBMS and the programming language will use the same model of representation. Relational DBMS projects, by way of contrast, maintain a clearer division between the database model and the application.

Object database management systems added the concept of persistence to object programming languages. The early commercial products were integrated with various languages: GemStone (Smalltalk), Gbase (LISP), Vbase (COP) and VOSS (Virtual Object Storage System for Smalltalk). For much of the 1990s, C++ dominated the commercial object database management market. Vendors added Java in the late 1990s and more recently, C#.

Starting in 2004, object databases have seen a second growth period when open source object databases emerged that were widely affordable and easy to use, because they are entirely written in OOP languages like Smalltalk, Java, or C#, such as Versant's db4o (db4objects), DTS/S1 from Obsidian Dynamics and Perst (McObject), available under dual open source and commercial licensing.

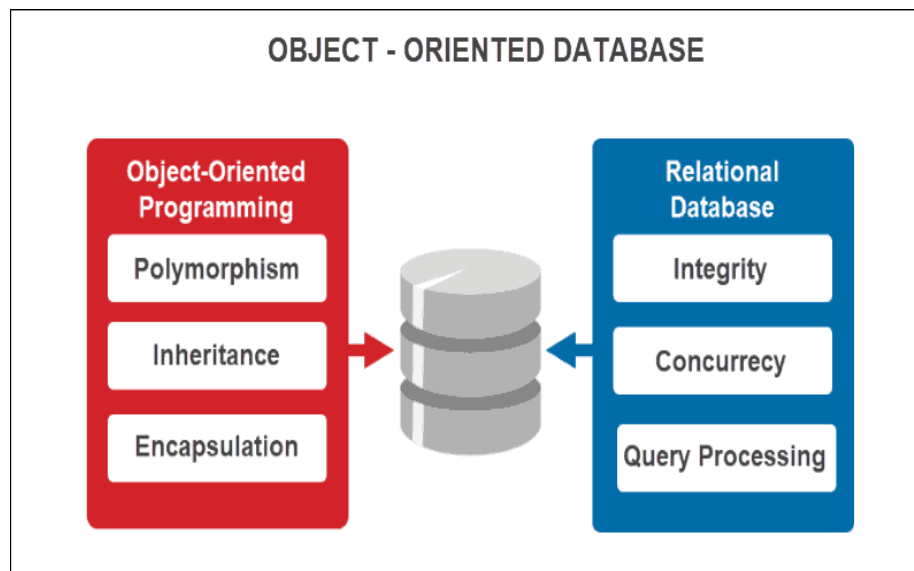


Figure: object oriented database concept

2.2 Object Database Extensions to SQL

The object-relational extensions to SQL include row types, collection types, user defined types (UDT), typed tables and reference types. The object-relational data model does not support all the features of the object-oriented data model.

Object-oriented analysis and design is a common methodology for analyzing and designing software systems. Frequently these systems include a database component. Standard rules exist to translate an object-oriented design of a database system into a relational database implementation but it would be better if the database software directly supported the object-oriented design. SQL is the standard relational database definition and manipulation language. Extensions to SQL in the 1999 and 2003 standards include object-oriented features [1][2]. These features support an extension of the relation data model called the object-relational data model. This model includes all the traditional ideas found in the relational data model and some features of the object-oriented data model. This paper presents an overview of the object-oriented data

model, the features of that model that are included in the object-relational extensions to SQL and a brief example illustrating the use of these features in the context of an Oracle 9i database server.

2.2.1. Object Oriented Data Model

The object-oriented data model is an extension of object-oriented ideas into the area of database systems. The fundamental concept is that a database consists of persistent objects that can be defined and manipulated by object-oriented languages such as C++ or Java. A persistent object is an object that continues to exist after the program that created the object is no longer running. The object can store any type of data. A database could include table objects but the database would not be restricted to only table objects. One advantage of the object-oriented data model is increased flexibility in the kinds of data that can be stored in a database. The relational data model is good at manipulating values like names and prices but is limited in its ability to store more complex objects such as images or movies. In an object-oriented database the contents can be queried through methods associated with objects in the database.

2.2.2. SQL Object-Relational Features

The object-relational extensions to SQL include row types, collection types, user defined types (UDT), typed tables and reference types. The object-relational data model does not support all the features of the object-oriented data model. Instead it represents an evolutionary step from the relational data model to the object-oriented data model

1. User-Defined Types and Complex Structures for Objects

To allow the creation of complex-structured objects, and to separate the declaration of a type from the creation of a table, SQL now provides **user-defined types (UDTs)**. In addition, four collection types have been included to allow for multivalued types and attributes in order to specify complex-structured objects rather than just simple (flat) records. The user will create the UDTs for a particular application as part of the database schema. A **UDT** may be specified in its simplest form using the following syntax:

```
CREATE TYPE TYPE_NAME AS (<component declarations>);
```

2. Object Identifiers Using Reference Types

Unique system-generated object identifiers can be created via the **reference type** in the latest version of SQL.

In general, the user can specify that system-generated object identifiers for the individual rows in a table should be created. By using the syntax:

```
REF IS <OID_ATTRIBUTE> <VALUE_GENERATION_METHOD> ;
```

3. Creating Tables Based on the UDTs

For each UDT that is specified to be instantiable via the phrase **INSTANTIABLE** (see Figure 11.4(b)), one or more tables may be created. This is illustrated in Figure 11.4(d), where we create a table **PERSON** based on the **PERSON_TYPE** UDT. Notice that the UDTs in Figure 11.4(a) are noninstantiable, and hence can only be used as types for attributes, but not as a basis for table creation. In Figure 11.4(b), the attribute **PERSON_ID** will hold the system-generated object identifier whenever a new **PERSON** record (object) is created and inserted in the table.

4. Encapsulation of Operations

In SQL, a **user-defined type** can have its own behavioral specification by specifying methods (or operations) in addition to the attributes. The general form of a UDT specification with methods is as follows:

```
CREATE TYPE <TYPE-NAME> (
  <LIST OF COMPONENT ATTRIBUTES AND THEIR TYPES>
  <DECLARATION OF FUNCTIONS (METHODS)>
);
```

5. Specifying Relationships via Reference

A component attribute of one tuple may be a **reference** (specified using the key-word **REF**) to a tuple of another (or possibly the same) table.

The keyword **SCOPE** specifies the name of the table whose tuples can be referenced by the reference attribute. Notice that this is similar to a foreign key, except that the system-generated value is used rather than the primary key value.

SQL uses a **dot notation** to build **path expressions** that refer to the component attributes of tuples and row types.

However, for an attribute whose type is **REF**, the dereferencing symbol **->** is used.

2.3. The ODMG Object Model and the Object Definition Language ODL

ODBMS stands for Object Database Management System. In ODBMS data is encapsulated and represented in the form of objects. It relates the concept of object-oriented programming with database systems. ODBMS grew out of research during the early 1970s as database support for graph-structured objects. In comparison with RDBMS, where data is stored in tables with rows and columns, ODBMS stores information as objects.

Characteristics

- **Easy to link with programming language:** The programming language and the database schema use the same type definitions, so developers may not need to learn a new database query language.
- **No need for user defined keys:** Object Database Management Systems have an automatically generated OID associated with each of the objects.
- **Easy modeling:** ODBMS can easily model real-world objects, hence, are suitable for applications with complex data.
- **Can store non-textual data** ODBMS can also store audio, video and image data.

Advantages

- **Speed:** Access to data can be faster because an object can be retrieved directly without a search, by following pointers.
- **Improved performance:** These systems are most suitable for applications that use object oriented programming.
- **Extensibility:** Unlike traditional RDBMS where the basic-datatypes are hardcoded, when using ODBMS the user can encode any kind of data-structures to hold the data.
- **Data consistency:** When ODBMS is integrated with an object-based application, there is much greater consistency between the database and the programming language since both use the same model of representation for the data. This helps avoid the impedance mismatch.
- **Capability of handling variety of data:** Unlike other database management systems, ODBMS can also store non textual data like-: images, videos and audios

Disadvantages

- **No universal standards:** There is no universally agreed standards of operating ODBMS. This is the most significant drawback as the user is free to manipulate data model as he wants which can be an issue when handling enormous amounts of data.
- **No security features:** Since use of ODBMS is very limited, there are not adequate security features to store production-grade data.
- **Exponential increase in complexity:** ODBMS become very complex very fast. When there is a lot of data and a lot of relations between data, managing and optimising ODBMS becomes difficult.
- **Scalability:** Unable to support large systems.
- **Query optimization is challenging:** Optimising ODBMS queries requires complete information about the data like-: type and size of data. This compromises the data-encapsulation feature that ODBMS had to offer.

2.4. Object Definition Language

Object Definition Language (ODL) is the specification language defining the interface to object types conforming to the ODMG Object Model. Often abbreviated by the acronym ODL. This language's purpose is to define the structure of an Entity-relationship diagram.

Class Declarations

Interface < name > { elements = attributes, relationships, methods }

Element Declarations

attributes (< type > : < name >);

relationships (< rangetype > : < name >);

Example

Type Data Tuple (Year, day, month)

Type Year , day, month integer

Interface Manager{

Keys id;

Attribute String name;

Attribute String phone;

relationship set<Employee> employees inverse Employee::manager}

interface Employee{

keys id;

attribute String name;

attribute Date Start Date;

relationship Manager inverse Manager::employee}

2.5. Object Database Conceptual Design

Conceptual design is the first stage in the database design process. The goal at this stage is to design a database that is independent of database software and physical details. The output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain. This design is descriptive and narrative in form.

The conceptual design has four steps, which are as follows.

1. Data analysis and requirements
2. Entity relationship modeling and normalization
3. Data model verification
4. Distributed database design

1. Data Analysis and Requirements:

The first step in conceptual design is to discover the characteristics of the data elements. Appropriate data element characteristics are those that can be transformed into appropriate information.

- Information needs. What kind of information is needed—that is, what output (reports and queries) must be generated by the system, what information does the current system generate, and to what extent is that information adequate?

- Information users. Who will use the information? How is the information to be used? What are the various end-user data views?
- Information sources. Where is the information to be found? How is the information to be extracted once it is found?
- Information constitution. What data elements are needed to produce the information? What are the data attributes? What relationships exist among the data? What is the data volume? How frequently are the data used? What data transformations are to be used to generate the required information? The designer obtains the answers to those questions from a variety of sources in order to compile the necessary information. Note these sources:
 - Developing and gathering end-user data views. The database designer and the end user(s) interact to jointly develop a precise description of end-user data views. In turn, the end-user data views will be used to help identify the database's main data elements.
 - Directly observing the current system: existing and desired output. The end user usually has an existing system in place, whether it's manual or computer-based. The designer reviews the existing system to identify the data and their characteristics.
 - Interfacing with the systems design group. The database design process is part of the Systems Development Life Cycle (SDLC). In some cases, the systems analyst in charge of designing the new system will also develop the conceptual database model.

2. Entity Relationship Modeling and Normalization:

The ER model, the designer must communicate and enforce appropriate standards to be used in the documentation of the design. The process of defining business rules and developing the conceptual model using ER diagrams can be described using the following steps.

1. Identify, analyze, and refine the business rules.
2. Identify the main entities, using the results of Step 1.
3. Define the relationships among the entities, using the results of Steps 1 and 2.
4. Define the attributes, primary keys, and foreign keys for each of the entities.
5. Normalize the entities. (Remember that entities are implemented as tables in an RDBMS.)
6. Complete the initial ER diagram.
7. Validate the ER model against the end users' information and processing requirements.
8. Modify the ER model, using the results of Step 7.

3. Data Model Verification:

The data model verification step is one of the last steps in the conceptual design stage, and it is also one of the most critical ones. In this step, the ER model must be verified against the proposed system processes in order to corroborate that the intended processes can be supported by the database model. Verification requires that the model be run through a series of tests against:

- End-user data views.
- All required transactions: SELECT, INSERT, UPDATE, and DELETE operations.
- Access rights and security.
- Business-imposed data requirements and constraints.

4. Distributed Database Design:

Although not a requirement for most databases, sometimes a database may need to be distributed among multiple geographically dispersed locations. Processes that access the database may also vary from one location to another. For example, a retail process and a warehouse storage process are likely to be found in different physical locations. If the database data and processes are to be distributed across the system, portions of a database, known as database fragments, may reside in several physical locations.

2.6 Object Query Language (OQL)

OQL is an SQL-like declarative language that provides a rich environment for efficient querying of database objects, including high-level primitives for object sets and structures. OQL provides a superset of the SQL-92 SELECT syntax. This means that most SQL SELECT statements which run on relational DBMSs tables work with the same syntax and semantics on ODMG collection classes.

OQL also includes object extensions to support object identity, complex objects, path expressions, operation invocation and inheritance. OQL's querying capabilities include the ability to invoke operations in ODMG language bindings, and OQL may be called from within an ODMG language binding for embedded operation. OQL maintains object integrity by invoking an object's defined operations, rather than using its own update operators.

In an effort to maintain compatibility with SQL SELECT in the future, the ODMG has created a strong working relationship with the ANSI X3H2 committee, which is defining the SQL3 standard. The objective is to ensure interoperability between OQL and the SQL-3 proposal.

2.7. Language Binding in the ODMG Standard

The ODMG standards' C++, Smalltalk and Java bindings define Object Manipulation Languages (OMLs) that extend the language standards to support manipulation of persistent objects. The bindings also include support for OQL, navigation and transactions. Because each language has its own OML, developers can work inside a single language environment without separate programming and database languages.

Java Language Binding: The ODMG language binding for Java is new in Release 2.0 of the ODMG standard. It adheres to the same principles as the Smalltalk and C++ bindings. The binding uses established Java language practice and style so as to be natural to the Java environment and programmers. Instances of existing classes can be made persistent without

changes to source code. As in the Smalltalk binding, persistence is by reachability: once a transaction is committed, any objects that can be reached from root objects in the database are automatically made persistent in the database. The ODMG binding to Java adds classes and other constructs to the Java environment to support the ODMG object model, including collections, transactions and databases, without altering the semantics of the language.

C++ Language Binding: The ODMG C++ binding provides language-transparent extensions that do not require a preprocessor to implement. These extensions provide facilities for object creation, naming, manipulation and deletion, and transactions and database operations. The C++ binding allows persistence-capable classes to be created by inheritance. OQL and OML can be interspersed, so that OQL queries may be built in-line to application code and conversely, OQL can call OML. The binding also includes recent ANSI C++ enhancements, like support for Standard Template Library (STL) algorithms (to provide sequential access to members of a collection) and exception handling.

Smalltalk Language Binding: The ODMG Smalltalk binding provides the ability to store, retrieve and modify persistent objects in Smalltalk. The binding includes a mechanism to invoke OQL and procedures for transactions and operations on databases. Smalltalk objects are made persistent by reachability. This means that an object becomes persistent when it is referenced by another persistent object in the database, and it is garbage-collected when it is no longer reachable.

The Smalltalk binding directly maps all of the classes defined in the ODMG object model to Smalltalk classes. Object Model collection classes and operations are mapped wherever possible to standard collection classes and methods. Relationships, transactions and database operations are also mapped to Smalltalk constructs.

Unit : 3 Query processing and optimization

3.1. Concept of Query processing

Query Processing is the activity performed in extracting data from the database. In query processing, it takes various steps for fetching the data from the database. The steps involved are:

1. Parsing and translation
2. Optimization
3. Evaluation

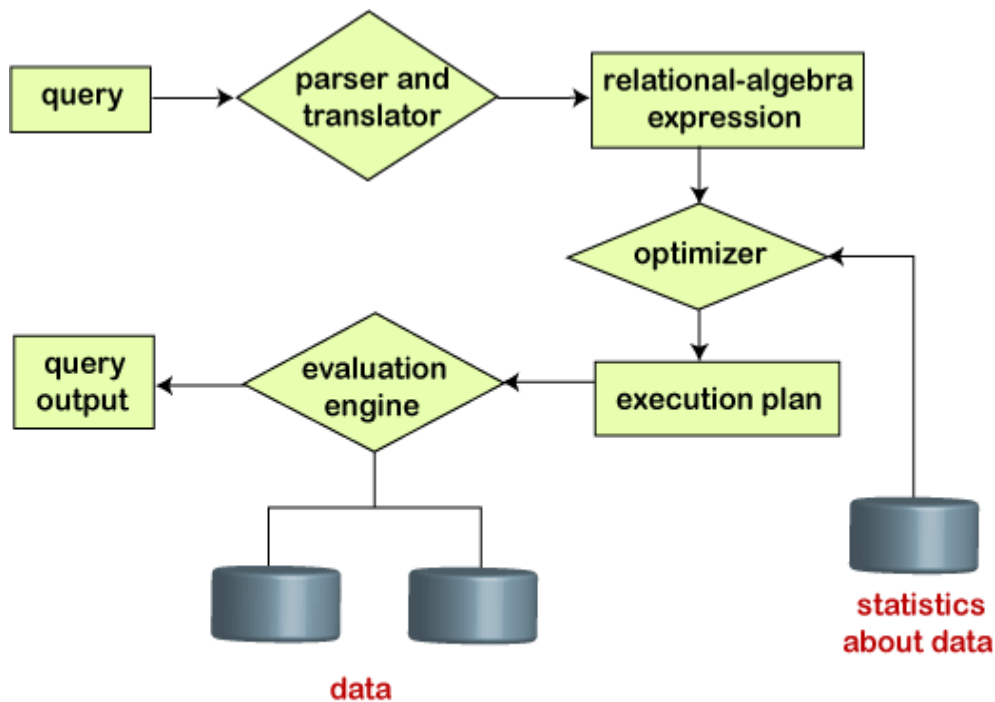
The query processing works in the following way:

1. Parsing and Translation

As query processing includes certain activities for data retrieval. Initially, the given user queries get translated in high-level database languages such as SQL. It gets translated into expressions

that can be further used at the physical level of the file system. After this, the actual evaluation of the queries and a variety of query -optimizing transformations and takes place. Thus before processing a query, a computer system needs to translate the query into a human-readable and understandable language. Consequently, SQL or Structured Query Language is the best suitable choice for humans. But, it is not perfectly suitable for the internal representation of the query to the system. Relational algebra is well suited for the internal representation of a query. The translation process in query processing is similar to the parser of a query. When a user executes any query, for generating the internal form of the query, the parser in the system checks the syntax of the query, verifies the name of the relation in the database, the tuple, and finally the required attribute value. The parser creates a tree of the query, known as 'parse-tree.' Further, translate it into the form of relational algebra. With this, it evenly replaces all the use of the views when used in the query.

Thus, we can understand the working of a query processing in the below-described diagram:



Steps in query processing

Suppose a user executes a query. As we have learned that there are various methods of extracting the data from the database. In SQL, a user wants to fetch the records of the employees whose salary is greater than or equal to 10000. For doing this, the following query is undertaken:

select emp_name from Employee where salary>10000;

Thus, to make the system understand the user query, it needs to be translated in the form of relational algebra. We can bring this query in the relational algebra form as:

- $\sigma_{\text{salary} > 10000} (\pi_{\text{salary}}(\text{Employee}))$
- $\pi_{\text{salary}} (\sigma_{\text{salary} > 10000}(\text{Employee}))$

After translating the given query, we can execute each relational algebra operation by using different algorithms. So, in this way, a query processing begins its working.

2. Evaluation

For this, with addition to the relational algebra translation, it is required to annotate the translated relational algebra expression with the instructions used for specifying and evaluating each operation. Thus, after translating the user query, the system executes a query evaluation plan.

Query Evaluation Plan

- In order to fully evaluate a query, the system needs to construct a query evaluation plan.
- The annotations in the evaluation plan may refer to the algorithms to be used for the particular index or the specific operations.
- Such relational algebra with annotations is referred to as **Evaluation Primitives**. The evaluation primitives carry the instructions needed for the evaluation of the operation.
- Thus, a query evaluation plan defines a sequence of primitive operations used for evaluating a query. The query evaluation plan is also referred to as **the query execution plan**.
- A **query execution engine** is responsible for generating the output of the given query. It takes the query execution plan, executes it, and finally makes the output for the user query.

3. Optimization

- The cost of the query evaluation can vary for different types of queries. Although the system is responsible for constructing the evaluation plan, the user does need not to write their query efficiently.
- Usually, a database system generates an efficient query evaluation plan, which minimizes its cost. This type of task performed by the database system and is known as Query Optimization.
- For optimizing a query, the query optimizer should have an estimated cost analysis of each operation. It is because the overall operation cost depends on the memory allocations to several operations, execution costs, and so on.

3.2 Query Parsing and Translation

Initially, the SQL query is scanned. Then it is parsed to look for syntactical errors and correctness of data types. If the query passes this step, the query is decomposed into smaller query blocks. Each block is then translated to equivalent relational algebra expression.

Steps for Query Optimization

Query optimization involves three steps, namely query tree generation, plan generation, and query plan code generation.

Step 1 – Query Tree Generation

A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.

During execution, an internal node is executed whenever its operand tables are available. The node is then replaced by the result table. This process continues for all internal nodes until the root node is executed and replaced by the result table.

For example, let us consider the following schemas –
EMPLOYEE

EmpID	EName	Salary	DeptNo	DateOfJoining
-------	-------	--------	--------	---------------

DEPARTMENT

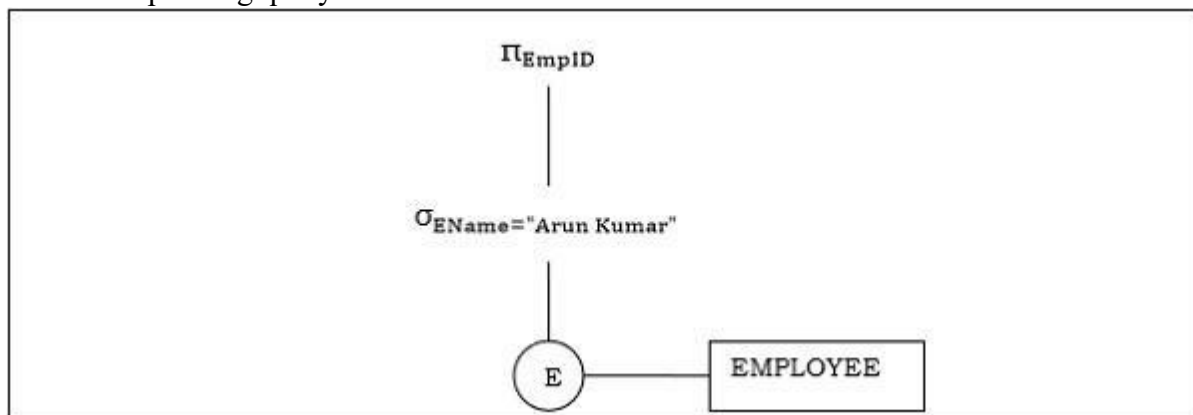
DNo	DName	Location
-----	-------	----------

Example 1

Let us consider the query as the following.

$\pi_{\text{EmpID}}(\sigma_{\text{EName} = \text{"ArunKumar"}}(\text{EMPLOYEE}))$

The corresponding query tree will be –

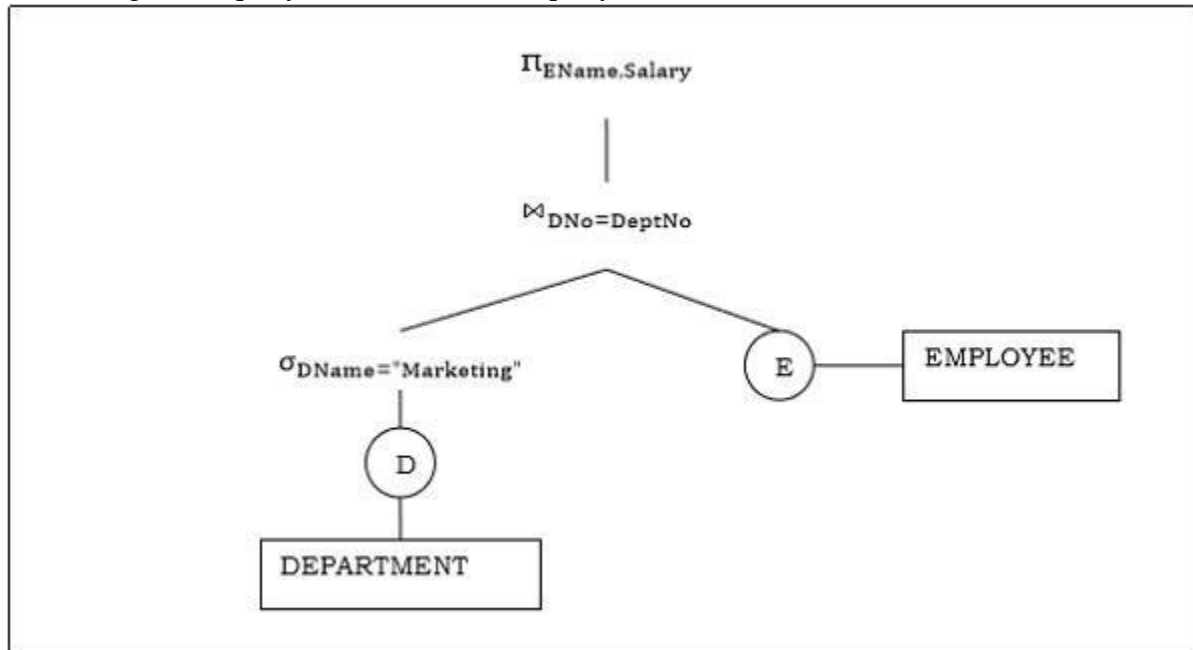


Example 2

Let us consider another query involving a join.

$\pi_{\{EName, Salary\}} (\sigma_{\{DName = \text{"Marketing"}\}} \{ (DEPARTMENT) \} \bowtie_{\{DNo=DeptNo\}} \{ (EMPLOYEE) \})$

Following is the query tree for the above query.

**Step 2 – Query Plan Generation**

After the query tree is generated, a query plan is made. A query plan is an extended query tree that includes access paths for all operations in the query tree. Access paths specify how the relational operations in the tree should be performed. For example, a selection operation can have an access path that gives details about the use of B+ tree index for selection.

Besides, a query plan also states how the intermediate tables should be passed from one operator to the next, how temporary tables should be used and how operations should be pipelined/combined.

Step 3– Code Generation

Code generation is the final step in query optimization. It is the executable form of the query, whose form depends upon the type of the underlying operating system. Once the query code is generated, the Execution Manager runs it and produces the results.

Approaches to Query Optimization

Among the approaches for query optimization, exhaustive search and heuristics-based algorithms are mostly used.

3.3. Exhaustive Search Optimization

In these techniques, for a query, all possible query plans are initially generated and then the best plan is selected. Though these techniques provide the best solution, it has an exponential time and space complexity owing to the large solution space. For example, dynamic programming technique.

3.4 Heuristic Based Optimization

Heuristic based optimization uses rule-based optimization approaches for query optimization. These algorithms have polynomial time and space complexity, which is lower than the exponential complexity of exhaustive search-based algorithms. However, these algorithms do not necessarily produce the best query plan.

Some of the common heuristic rules are –

- Perform select and project operations before join operations. This is done by moving the select and project operations down the query tree. This reduces the number of tuples available for join.
- Perform the most restrictive select/project operations at first before the other operations.
- Avoid cross-product operation since they result in very large-sized intermediate tables.

3.5 Choice of the Query Execution plans :

A **query plan** (or **query execution plan**) is a sequence of steps used to access data in a SQL relational database management system. This is a specific case of the relational model concept of access plans.

Since SQL is declarative, there are typically many alternative ways to execute a given query, with widely varying performance. When a query is submitted to the database, the query optimizer evaluates some of the different, correct possible plans for executing the query and returns what it considers the best option. Because query optimizers are imperfect, database users and administrators sometimes need to manually examine and tune the plans produced by the optimizer to get better performance.

Generating query plans:

A given database management system may offer one or more mechanisms for returning the plan for a given query. Some packages feature tools which will generate a graphical representation of a query plan. Other tools allow a special mode to be set on the connection to cause the DBMS to return a textual description of the query plan. Another mechanism for retrieving the query plan involves querying a virtual database table after executing the query to be examined. In Oracle, for instance, this can be achieved using the EXPLAIN PLAN statement.

Graphical plans:

The Microsoft SQL Server Management Studio tool, which ships with Microsoft SQL Server, for example, shows this graphical plan when executing this two-table join example against an included sample database:

```

SELECT *
FROM HumanResources.Employee AS e
  INNER JOIN Person.Contact AS c
    ON e.ContactID = c.ContactID
ORDER BY c.LastName

```

Textual plans:

It indicates that the query engine will do a scan over the primary key index on the Employee table and a matching seek through the primary key index (the ContactID column) on the Contact table to find matching rows.

StmtText

```

---
|--Sort(ORDER BY:([c].[LastName] ASC))
  |--Nested Loops(Inner Join, OUTER REFERENCES:([e].[ContactID], [Expr1004])
WITH UNORDERED PREFETCH)
    |--Clustered Index
Scan(OBJECT:([AdventureWorks].[HumanResources].[Employee].[PK_Employee_EmployeeI
D] AS [e]))
  |--Clustered Index
Seek(OBJECT:([AdventureWorks].[Person].[Contact].[PK_Contact_ContactID] AS [c]),

SEEK:([c].[ContactID]=[AdventureWorks].[HumanResources].[Employee].[ContactID] as
[e].[ContactID]) ORDERED FORWARD)

```

Cost Based optimization

Cost-Based Optimization (aka Cost-Based Query Optimization or CBO Optimizer) is an optimization technique in Spark SQL that uses table statistics to determine the most efficient query execution plan of a structured query (given the logical query plan). Cost-based optimization is disabled by default.

For a given query and environment, the Optimizer allocates a cost in numerical form which is related to each step of a possible plan and then finds these values together to get a cost estimate for the plan or for the possible strategy. After calculating the costs of all possible plans, the Optimizer tries to choose a plan which will have the possible lowest cost estimate. For that reason, the Optimizer may be sometimes referred to as the Cost-Based Optimizer. Below are some of the features of the cost-based optimization-

1. The cost-based optimization is based on the cost of the query that to be optimized.
2. The query can use a lot of paths based on the value of indexes, available sorting methods, constraints, etc.
3. The aim of query optimization is to choose the most efficient path of implementing the query at the possible lowest minimum cost in the form of an algorithm.
4. The cost of executing the algorithm needs to be provided by the query Optimizer so that the most suitable query can be selected for an operation.
5. The cost of an algorithm also depends upon the cardinality of the input.

Unit: 4 Distributed databases, NOSQL System, and Big data

4.1 Distributed Database Concept and advantages

A distributed database represents multiple interconnected databases spread out across several sites connected by a network. Since the databases are all connected, they appear as a single database to the users. Distributed databases utilize multiple nodes. They scale horizontally and develop a distributed system. More nodes in the system provide more computing power, offer greater availability, and resolve the single point of failure issue. Different parts of the distributed database are stored in **several physical locations**, and the processing requirements are distributed among processors on multiple database nodes.

A centralized distributed database management system (**DDBMS**) manages the distributed data as if it were stored in one physical location. DDBMS synchronizes all data operations among databases and ensures that the updates in one database automatically reflect on databases in other sites.

4.1.1. Distributed Database Features

Some general features of distributed databases are:

- **Location independency** - Data is physically stored at multiple sites and managed by an independent DDBMS.
- **Distributed query processing** - Distributed databases answer queries in a distributed environment that manages data at multiple sites. High-level queries are transformed into a query execution plan for simpler management.
- **Distributed transaction management** - Provides a consistent distributed database through commit protocols, distributed concurrency control techniques, and distributed recovery methods in case of many transactions and failures.
- **Seamless integration** - Databases in a collection usually represent a single logical database, and they are interconnected.
- **Network linking** - All databases in a collection are linked by a network and communicate with each other.
- **Transaction processing** - Distributed databases incorporate transaction processing, which is a program including a collection of one or more database operations. Transaction processing is an atomic process that is either entirely executed or not at all.

4.1.2. Types of Distributed database

1. Homogeneous Database:

In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structures used – all are the same at all sites. Hence, they're easy to manage.

2. Heterogeneous Database:

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

Advantages of Distributed database

Distributed database management basically proposed for the various reason from organizational decentralization and economical processing to greater autonomy. Some of these advantages are as follows:

- 1. Increased reliability and availability** – A distributed database system is robust to failure to some extent. Hence, it is reliable when compared to a Centralized database system.
- 2. Local control** – The data is distributed in such a way that every portion of it is local to some sites (servers). The site in which the portion of data is stored is the owner of the data.
- 3. Modular growth (resilient)** – Growth is easier. We do not need to interrupt any of the functioning sites to introduce (add) a new site. Hence, the expansion of the whole system is easier. Removal of site is also does not cause much problems.
- 4. Lower communication costs (More Economical)** – Data are distributed in such a way that they are available near to the location where they are needed more. This reduces the communication cost much more compared to a centralized system.
- 5. Faster response** – Most of the data are local and in close proximity to where they are needed. Hence, the requests can be answered quickly compared to a centralized system.
- 6. Reflects the organizational structure** – Normally, database is fragmented into various locations wherever we have controls.

7. Secured management of distributed data – Various transparencies like network transparency, fragmentation transparency, and replication transparency are implemented to hide the actual implementation details of the whole distributed system. In such way, Distributed database provides security for data.

8. Robust – The system is continued to work in case of failures. For example, replicated distributed database performs in spite of failure of other sites.

9. Complied with ACID properties – Distributed transactions demands Atomicity, Consistency, Isolation, and Reliability.

10. Improved performance and Parallelism in executing transactions can be achieved.

4.2 Data Fragmentation

Distributed Database systems provide distribution transparency of the data over the DBs. This is achieved by the concept called Data Fragmentation. That means, fragmenting the data over the network and over the DBs. Initially all the DBs and data are designed as per the standards of any database system – by applying normalization and de-normalization.

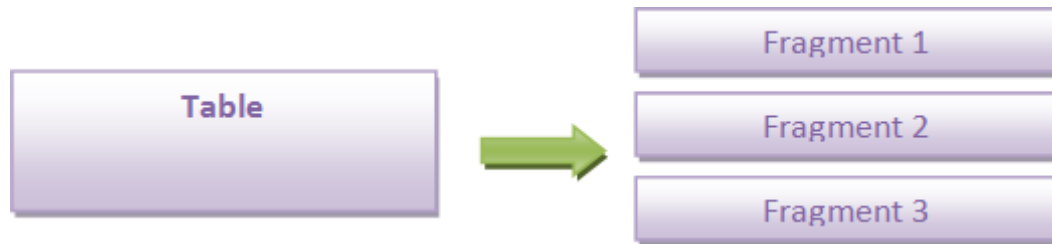
Fragmentation of data can be done according to the DBs and user requirement. But while fragmenting the data, below.

- **Completeness** : While creating the fragment, partial records in the table should not be considered. Fragmentation should be performed on whole table's data to get the correct result. For example, if we are creating fragment on EMPLOYEE table, then we need to consider whole EMPLOYEE table for constructing fragments. It should not be created on the subset of EMPLOYEE records.
- **Reconstructions** : When all the fragments are combined, it should give whole table's data. That means whole table should be able to reconstruct using all fragments. For example all fragments' of EMPLOYEE table in the DB, when combined should give complete EMPLOYEE table records.
- **Disjointedness** : There should not be any overlapping data in the fragments. If so, it will be difficult to maintain the consistency of the data. Effort needs to be put to create same replication in all the copies of data. Suppose we have fragments on EMPLOYEE table based on location then, there should not be any two fragments having the details of same employee.

4.2.1 Types of data fragmentations in DDBMS.

- **Horizontal Data Fragmentation** :

As the name suggests, here the data / records are fragmented horizontally. i.e.; horizontal subset of table data is created and are stored in different database in DDB.



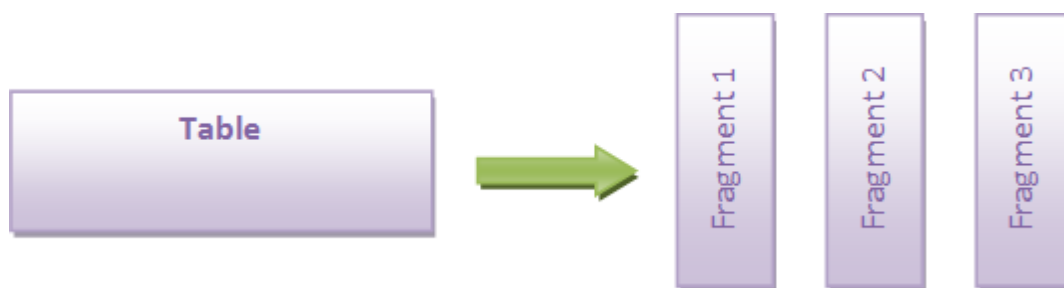
For example

Hence what we do is divide the entire employee table data horizontally based on the location. i.e.

```
SELECT * FROM EMPLOYEE WHERE EMP_LOCATION = _INDIA;  
SELECT * FROM EMPLOYEE WHERE EMP_LOCATION = _USA;  
SELECT * FROM EMPLOYEE WHERE EMP_LOCATION = _UK;
```

- **Vertical Data Fragmentation :**

This is the vertical subset of a relation. That means a relation / table is fragmented by considering the columns of it.



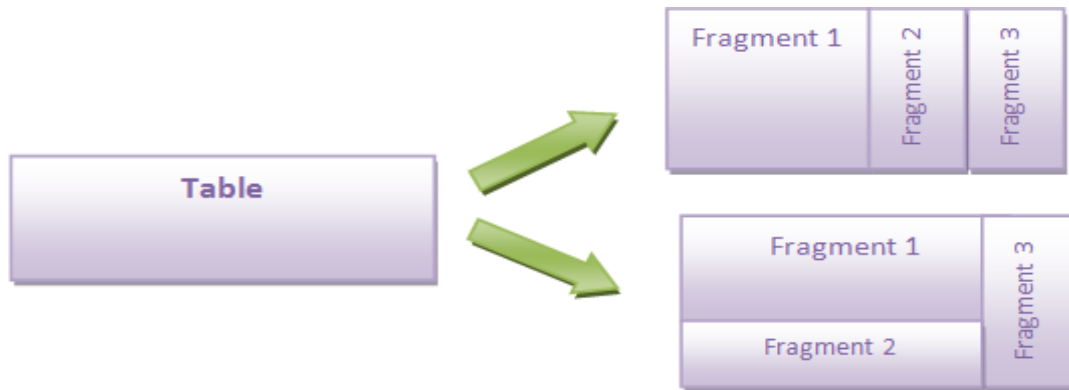
For example consider the EMPLOYEE table with ID, Name, Address, Age, location, DeptID, ProjID. The vertical fragmentation of this table may be dividing the table into different tables with one or more columns from EMPLOYEE.

i.e

```
SELECT EMP_ID, EMP_FIRST_NAME, EMP_LAST_NAME, AGE FROM EMPLOYEE;  
SELECT EMP_ID, STREETNUM, TOWN, STATE, COUNTRY, PIN FROM EMPLOYEE;  
SELECT EMP_ID, DEPTID FROM EMPLOYEE;  
SELECT EMP_ID, PROJID FROM EMPLOYEE;
```

- **Hybrid Data Fragmentation :**

This is the combination of horizontal as well as vertical fragmentation. This type of fragmentation will have horizontal fragmentation to have subset of data to be distributed over the DB, and vertical fragmentation to have subset of columns of the table.



As we observe in above diagram, this type of fragmentation can be done in any order. It does not have any particular order. It is solely based on the user requirement. But it should satisfy fragmentation conditions.

Consider the EMPLOYEE table with below fragmentations.

```
SELECT EMP_ID, EMP_FIRST_NAME, EMP_LAST_NAME, AGE
FROM EMPLOYEE WHERE EMP_LOCATION = _INDIA;
SELECT EMP_ID, DEPTID FROM EMPLOYEE WHERE EMP_LOCATION = _INDIA;
SELECT EMP_ID, EMP_FIRST_NAME, EMP_LAST_NAME, AGE
FROM EMPLOYEE WHERE EMP_LOCATION = _US;
SELECT EMP_ID, PROJID FROM EMPLOYEE WHERE EMP_LOCATION = _US;
```

4.3 Replication, and Allocation Techniques for Distributed Database Design

Replication is useful in improving the availability of data. The most extreme case is replication of the *whole database* at every site in the distributed system, thus creating a **fully replicated distributed database**. This can improve availability remarkably because the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries because the results of such queries can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module. The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent. This is especially true if many copies of the database exist. Full replication makes the concurrency control and recovery techniques more expensive than they would be if there was no replication, as we will see in Section 25.7.

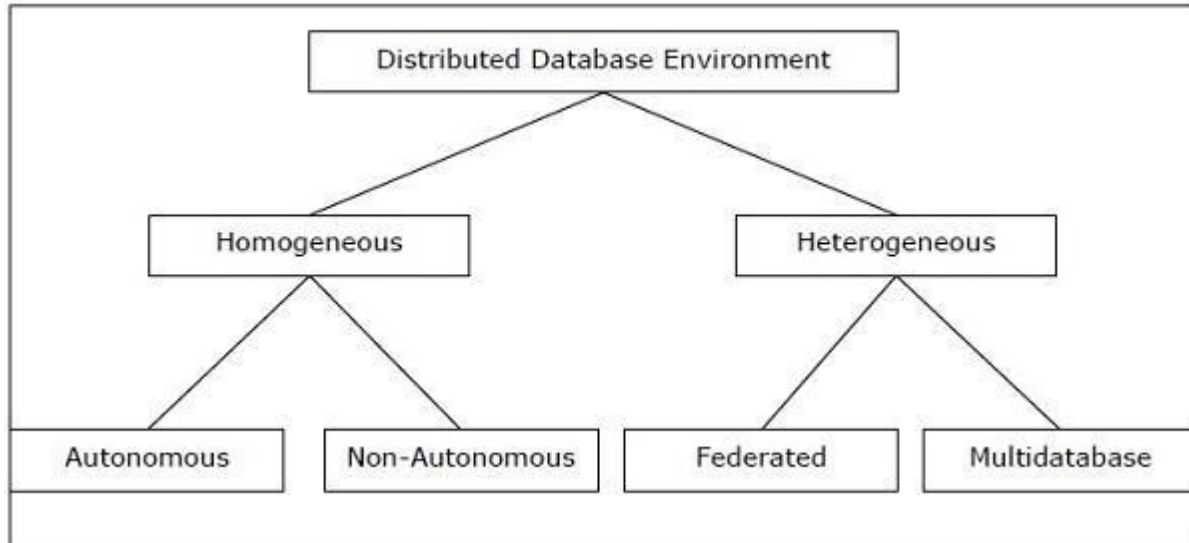
The other extreme from full replication involves having **no replication**—that is, each fragment is stored at exactly one site. In this case, all fragments *must be* dis-joint, except for the repetition of primary keys among vertical (or mixed) fragments. This is also called **nonredundant allocation**. Between these two extremes, we have a wide spectrum of **partial replication** of the data—that is, some fragments of the database may be replicated whereas others may not. The number of copies of each fragment can range from one up to the total number of sites in the distributed system. A special case of partial replication is occurring heavily in applications where mobile workers—such as sales forces, financial planners, and claims adjusters—carry partially replicated databases with them on laptops and PDAs and synchronize them periodically with the server database.⁷ A description of the replication of fragments is sometimes called a **replication schema**.

Each fragment—or each copy of a fragment—must be assigned to a particular site in the distributed system. This process is called **data distribution** (or **data allocation**). The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site. For example, if high availability is required, transactions can be submitted at any site, and most transactions are retrieval only, a fully replicated database is a good choice. However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication. Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.

4.4. Types of distributed database system:

A **distributed database system** allows applications to access data from local and remote databases. In a **homogenous distributed database system**, each database is an Oracle Database. In a **heterogeneous distributed database system**, at least one of the databases is not an Oracle Database.

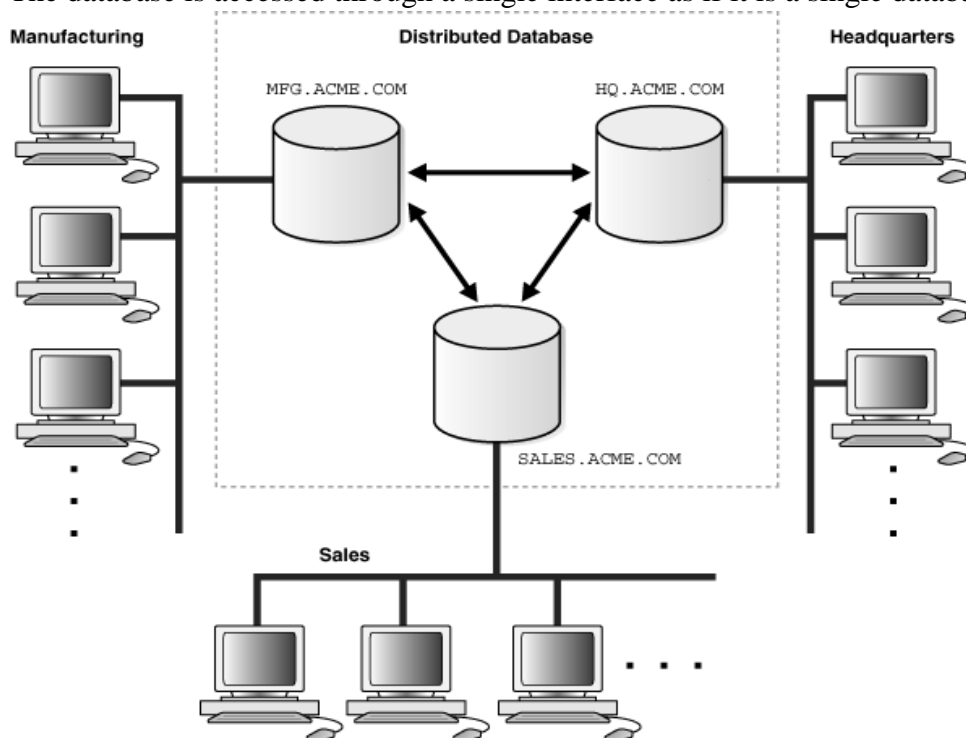
Distributed databases can be broadly classified into homogeneous and heterogeneous distributed database environments, each with further sub-divisions, as shown in the following illustration.



Homogeneous Distributed Databases

In a homogeneous distributed database, all the sites use identical DBMS and operating systems. Its properties are –

- The sites use very similar software.
- The sites use identical DBMS or DBMS from the same vendor.
- Each site is aware of all other sites and cooperates with other sites to process user requests.
- The database is accessed through a single interface as if it is a single database.



Types of Homogeneous Distributed Database

There are two types of homogeneous distributed database –

- **Autonomous** – Each database is independent that functions on its own. They are integrated by a controlling application and use message passing to share data updates.
- **Non-autonomous** – Data is distributed across the homogeneous nodes and a central or master DBMS co-ordinates data updates across the sites.

Heterogeneous Distributed Databases

In a heterogeneous distributed database, different sites have different operating systems, DBMS products and data models. Its properties are –

- Different sites use dissimilar schemas and software.
- The system may be composed of a variety of DBMSs like relational, network, hierarchical or object oriented.
- Query processing is complex due to dissimilar schemas.
- Transaction processing is complex due to dissimilar software.
- A site may not be aware of other sites and so there is limited co-operation in processing user requests.

Types of Heterogeneous Distributed Databases

- **Federated** – The heterogeneous database systems are independent in nature and integrated together so that they function as a single database system.
- **Un-federated** – The database systems employ a central coordinating module through which the databases are accessed.

Distributed DBMS Architectures

DDBMS architectures are generally developed depending on three parameters –

- **Distribution** – It states the physical distribution of data across the different sites.
- **Autonomy** – It indicates the distribution of control of the database system and the degree to which each constituent DBMS can operate independently.
- **Heterogeneity** – It refers to the uniformity or dissimilarity of the data models, system components and databases.

Architectural Models

Some of the common architectural models are –

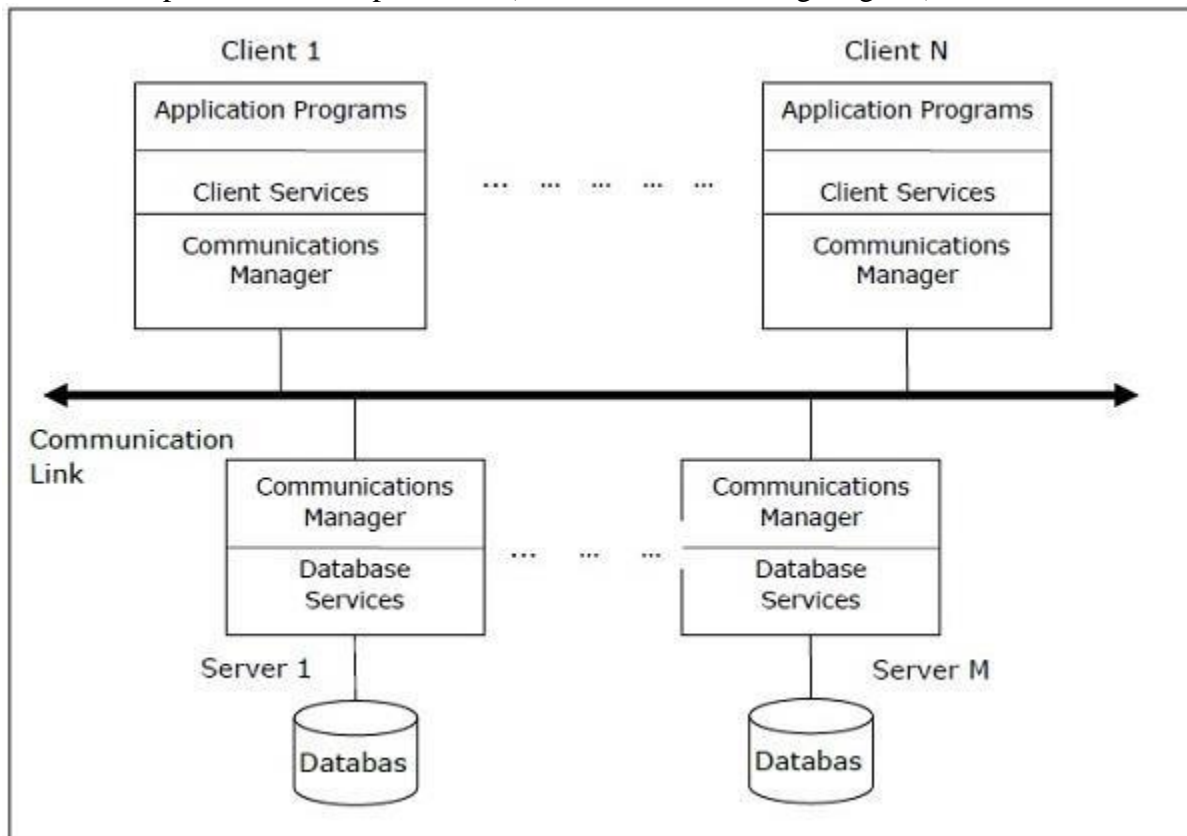
- Client - Server Architecture for DDBMS
- Peer - to - Peer Architecture for DDBMS
- Multi - DBMS Architecture

Client - Server Architecture for DDBMS

This is a two-level architecture where the functionality is divided into servers and clients. The server functions primarily encompass data management, query processing, optimization and transaction management. Client functions include mainly user interface. However, they have some functions like consistency checking and transaction management.

The two different client - server architecture are –

- Single Server Multiple Client
- Multiple Server Multiple Client (shown in the following diagram)

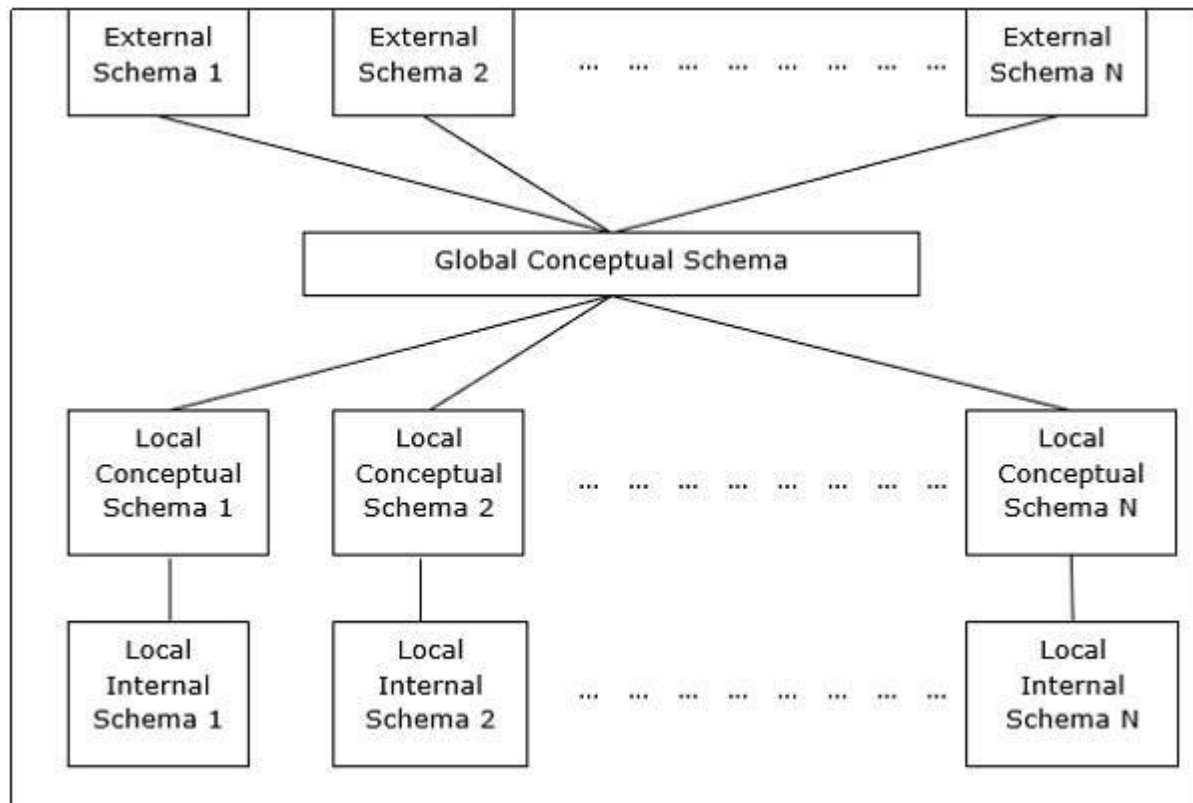


Peer- to-Peer Architecture for DDBMS

In these systems, each peer acts both as a client and a server for imparting database services. The peers share their resource with other peers and co-ordinate their activities.

This architecture generally has four levels of schemas –

- **Global Conceptual Schema** – Depicts the global logical view of data.
- **Local Conceptual Schema** – Depicts logical data organization at each site.
- **Local Internal Schema** – Depicts physical data organization at each site.
- **External Schema** – Depicts user view of data.



Multi - DBMS Architectures

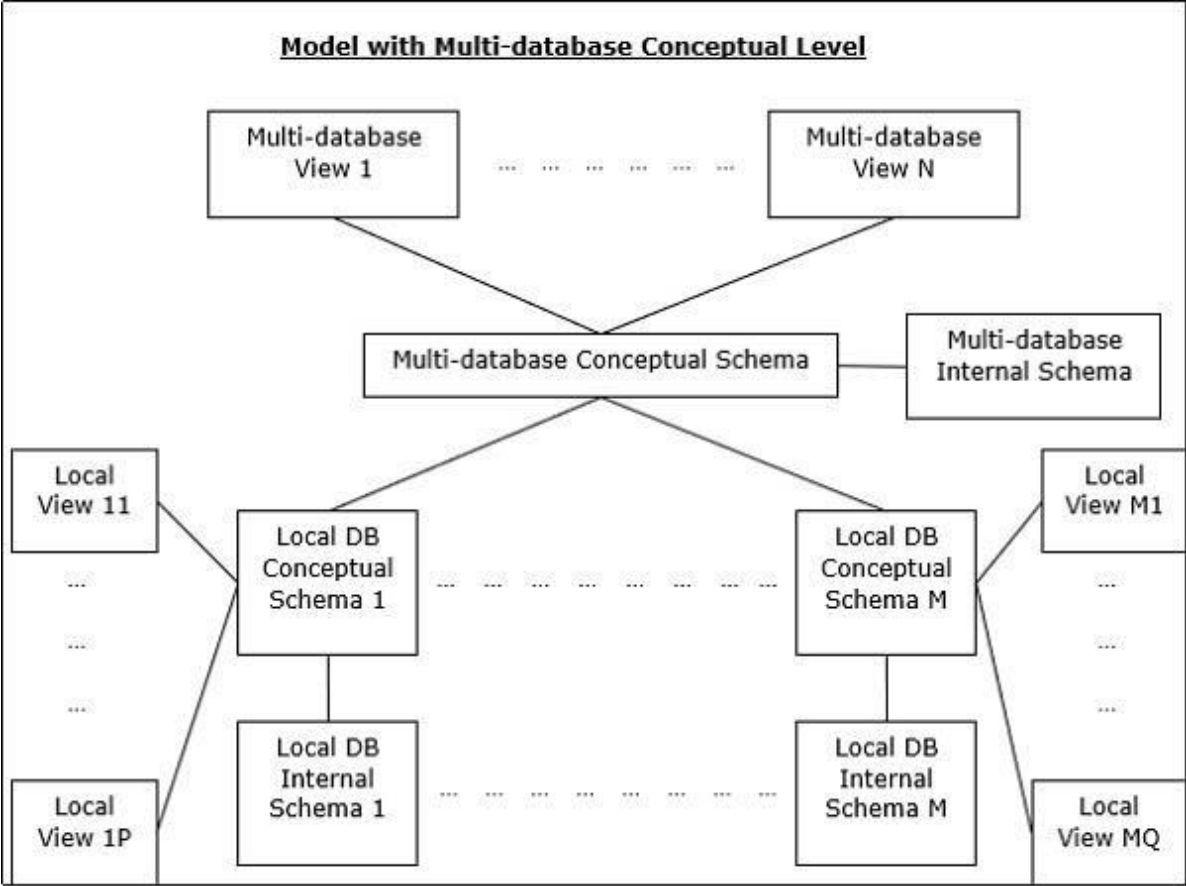
This is an integrated database system formed by a collection of two or more autonomous database systems.

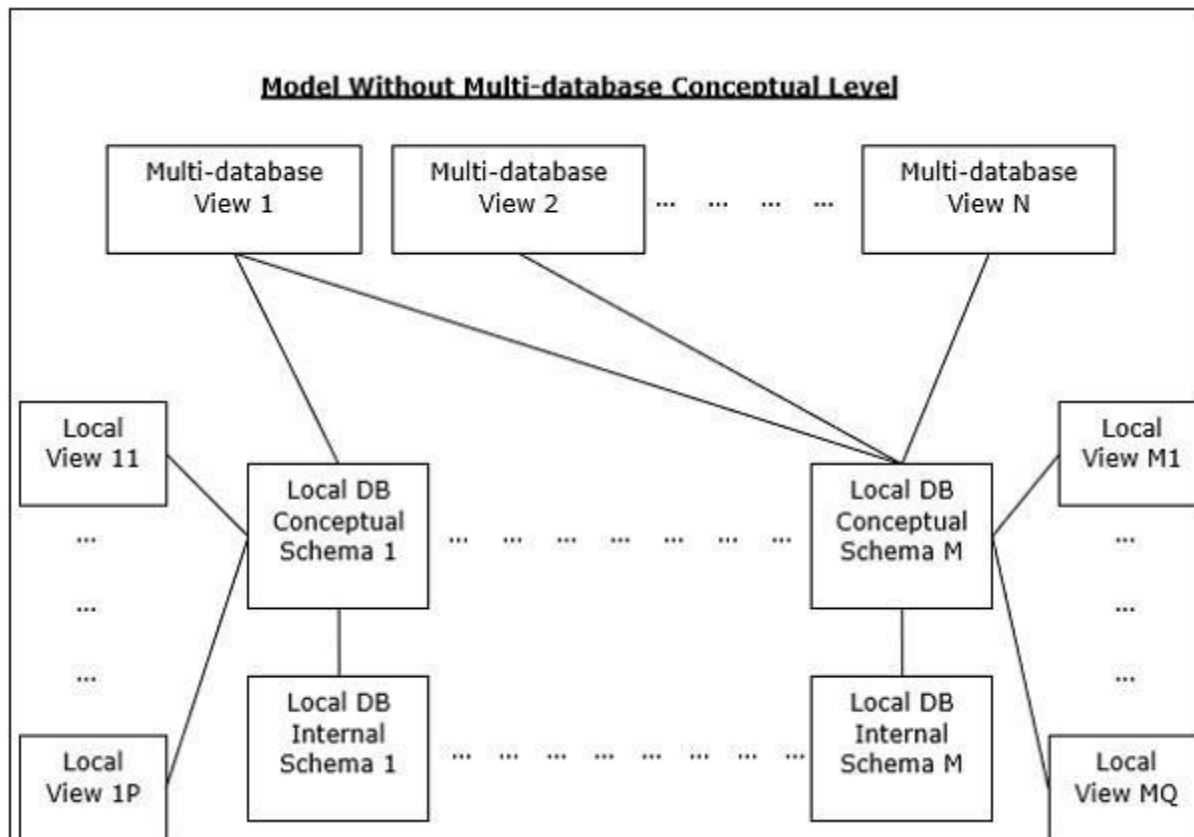
Multi-DBMS can be expressed through six levels of schemas –

- **Multi-database View Level** – Depicts multiple user views comprising of subsets of the integrated distributed database.
- **Multi-database Conceptual Level** – Depicts integrated multi-database that comprises of global logical multi-database structure definitions.
- **Multi-database Internal Level** – Depicts the data distribution across different sites and multi-database to local data mapping.
- **Local database View Level** – Depicts public view of local data.
- **Local database Conceptual Level** – Depicts local data organization at each site.
- **Local database Internal Level** – Depicts physical data organization at each site.

There are two design alternatives for multi-DBMS –

- Model with multi-database conceptual level.
- Model without multi-database conceptual level.





4.5 Introduction to NOSQL Systems

NoSQL **database** stands for –Not Only SQL or –Not SQL. Though a better term would be –NoREL, NoSQL caught on. Carl Stroz introduced the NoSQL concept in 1998. **NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

4.5.1 Types of NoSQL databases

There are four types of NoSQL databases:

- **Key value**

This is the most flexible type of NoSQL database because the application has complete control over what is stored in the value field without any restrictions.

- **Document**

Also referred to as document store or document-oriented databases, these databases are used for storing, retrieving, and managing semi-structured data. There is no need to specify which fields a document will contain.

- **Graph**

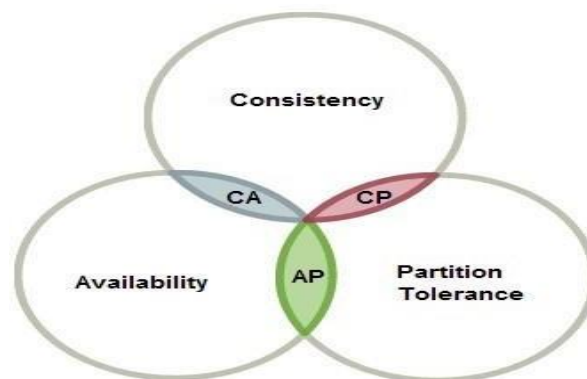
This database organizes data as nodes and relationships, which show the connections between nodes. This supports a richer and fuller representation of data. Graph databases are applied in social networks, reservation systems, and fraud detection.

- **Wide column**

These databases store and manage data in the form of tables, rows, and columns. They are broadly deployed in applications that require a column format to capture schema-free data.

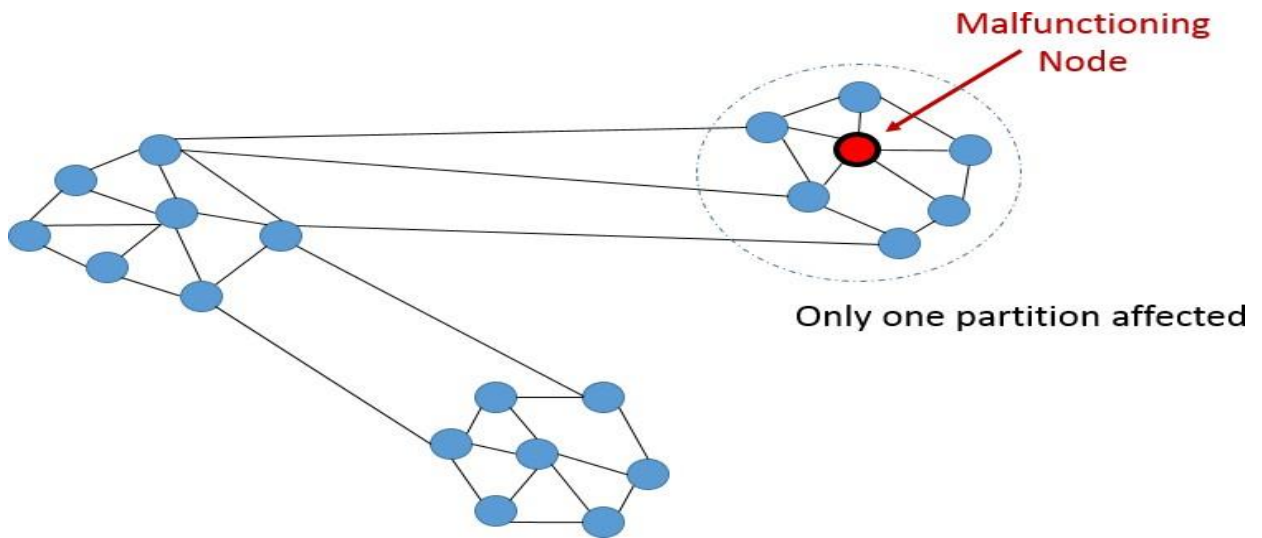
4.6 The CAP Theorem

CAP Theorem is a concept that a distributed database system can only have 2 of the 3: Consistency, Availability and Partition Tolerance.



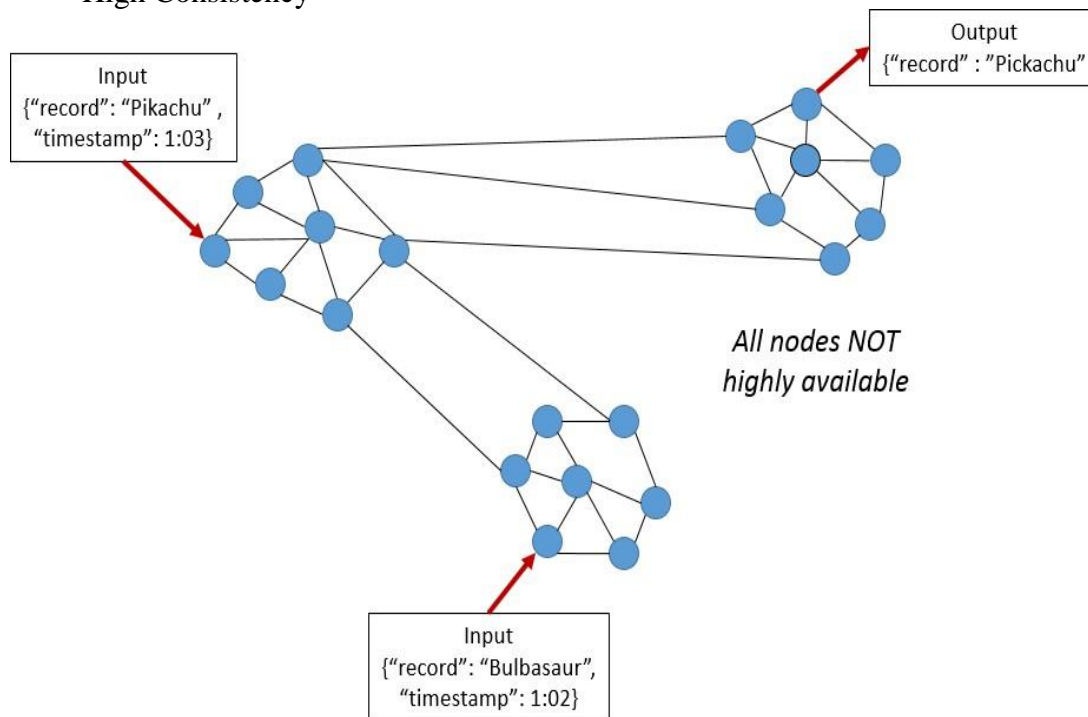
CAP Theorem is very important in the Big Data world, especially when we need to make trade off's between the three, based on our unique use case. On this blog, I will try to explain each of these concepts and the reasons for the trade off. I will avoid using specific examples as DBMS are rapidly evolving.

Partition Tolerance



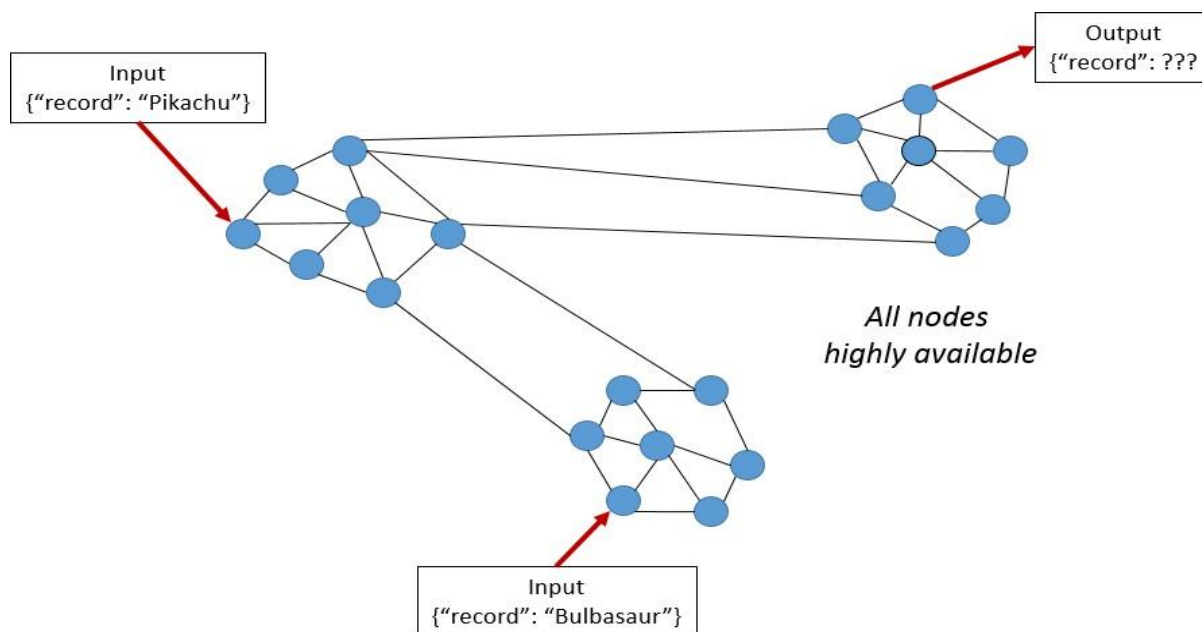
This condition states that the system continues to run, despite the number of messages being delayed by the network between nodes. A system that is partition-tolerant can sustain any amount of network failure that doesn't result in a failure of the entire network. Data records are sufficiently replicated across combinations of nodes and networks to keep the system up through intermittent outages. When dealing with modern distributed systems, Partition Tolerance is not an option. It's a necessity. Hence, we have to trade between Consistency and Availability.

High Consistency



This condition states that all nodes see the same data at the same time. Simply put, performing a *read* operation will return the value of the most recent *write* operation causing all nodes to return the same data. A system has consistency if a transaction starts with the system in a consistent state, and ends with the system in a consistent state. In this model, a system can (and does) shift into an inconsistent state during a transaction, but the entire transaction gets rolled back if there is an error during any stage in the process. In the image, we have 2 different records (–Bulbasaur and –Pikachu) at different timestamps. The output on the third partition is –Pikachu, the latest input. However, the nodes will need time to update and will not be Available on the network as often.

High Availability

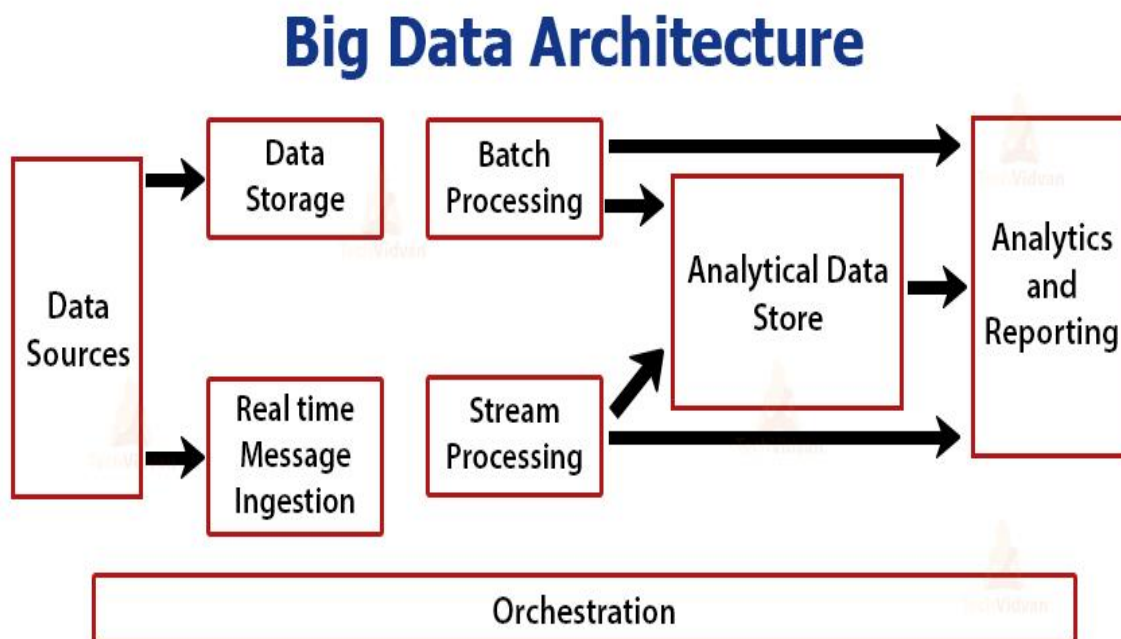


This condition states that every request gets a response on success/failure. Achieving availability in a distributed system requires that the system remains operational 100% of the time. Every client gets a response, regardless of the state of any individual node in the system. This metric is trivial to measure: either you can submit read/write commands, or you cannot. Hence, the databases are time independent as the nodes need to be available online at all times

4.7 Big Data:

It is changing our world and the way we live at an unprecedented rate. It is the new science of analyzing and predicting human and machine behaviour by processing a very huge amount of related data. It refers to speedy growth in the volume of structured, semi-structured and unstructured data. It is estimated to generate 50,000 Gb data per second in the year 2018. The speed at which data has generated a need to be stored and processed efficiently. Big Data engenders from multiple sources and arrives in multiple formats. Big Data in a way just means 'all data'. It can be described in terms of data management challenges that – due to increasing volume, velocity and variety of data – cannot be solved with traditional databases. It come from sensors, devices, video/audio, networks, log files, transactional applications, web, and social media – much of it generated in real-time and in a very large scale.

Architecture of Big data:



Data Sources:

All big data solutions start with one or more data sources. Examples includes: application data stores such as relational databases, Static files produced by applications such as web servers log files.

Data Storage:

Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats. This kind of store is often called a *data lake*. Options for implementing this storage include Azure Data Lake Store or blob containers in Azure Storage.

Batch processing:

Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files, processing them, and writing the output to new files. Options include running U-SQL jobs in Azure Data Lake Analytics, using Hive, Pig, or custom Map/Reduce jobs in an HDInsight Hadoop cluster, or using Java, Scala, or Python programs in an HDInsight Spark cluster.

Real-time message ingestion. If the solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing. This might be a simple data store, where incoming messages are dropped into a folder for processing. However, many solutions need a message ingestion store to act as a buffer for messages, and to support scale-out processing, reliable delivery, and other message queuing semantics. This portion of a streaming architecture is often referred to as stream buffering. Options include Azure Event Hubs, Azure IoT Hub, and Kafka.

Stream processing. After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis. The processed stream data is then written to an output sink. Azure Stream Analytics provides a managed stream processing service based on perpetually running SQL queries that operate on unbounded streams. You can also use open source Apache streaming technologies like Storm and Spark Streaming in an HDInsight cluster.

Analytical data store. Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical tools. The analytical data store used to serve these queries can be a Kimball-style relational data warehouse, as seen in most traditional business intelligence (BI) solutions. Alternatively, the data could be presented through a low-latency NoSQL technology such as HBase, or an interactive Hive database that provides a metadata abstraction over data files in the distributed data store. Azure Synapse Analytics provides a managed service for large-scale, cloud-based data warehousing. HDInsight supports Interactive Hive, HBase, and Spark SQL, which can also be used to serve data for analysis.

Analysis and reporting. The goal of most big data solutions is to provide insights into the data through analysis and reporting. To empower users to analyze the data, the architecture may include a data modeling layer, such as a multidimensional OLAP cube or tabular data model in Azure Analysis Services. It might also support self-service BI, using the modeling and visualization technologies in Microsoft Power BI or Microsoft Excel. Analysis and reporting can also take the form of interactive data exploration by data scientists or data analysts. For these scenarios, many Azure services support analytical notebooks, such as Jupyter, enabling these users to leverage their existing skills with Python or R. For large-scale data exploration, you can use Microsoft R Server, either standalone or with Spark.

Orchestration. Most big data solutions consist of repeated data processing operations, encapsulated in workflows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such Azure Data Factory or Apache Oozie and Sqoop.

The three V's of Big data are as follows.

- **Volume:** Large amounts of data from datasets with sizes of terabytes to zettabytes.
- **Velocity:** Large amounts of data from transactions with high refresh rate resulting in data streams coming at very high speed. As a result time required to act on these data streams will often be very short.
- **Variety:** Data is originated from different data sources. The data source can be either internal or external. Data can be in various formats due to various transactions and logs collected from different applications. It can be structured data (e.g. database table), semi-structured data (e.g. XML data), unstructured data (e.g. text, images, video streams, audio etc.).

Advantages:

- ➡ **Voluminous Collection:** A large amount of market data can be generated using big data analytics, and various graphical and mathematical representation can be made for easy analysis. This massive information is further helpful for deriving market based conclusion and predict consumer behavior.
- ➡ Big data analysis derives innovative solutions. Big data analysis helps in understanding and targeting customers. It helps in optimizing business processes.
- ➡ It helps in improving science and research.
- ➡ It improves healthcare and public health with availability of record of patients.
- ➡ It helps in financial tradings, sports, polling, security/law enforcement etc.
- ➡ Any one can access vast information via surveys and deliver an answer of any query.
- ➡ Every second additions are made.
- ➡ One platform carry unlimited information.

Disadvantages

- ➡ Traditional storage can cost lot of money to store big data.
- ➡ Lots of big data is unstructured.
- ➡ Big data analysis violates principles of privacy.
- ➡ It can be used for manipulation of customer records.
- ➡ It may increase social stratification.
- ➡ Big data analysis is not useful in short run. It needs to be analyzed for longer duration to leverage its benefits.

- ➔ Big data analysis results are misleading sometimes.
- ➔ Speedy updates in big data can mismatch real figures.

4.8 Introduction to MapReduce in Hadoop

MapReduce is a Hadoop framework used for writing applications that can process vast amounts of data on large clusters. It can also be called a programming model in which we can process

large datasets across computer clusters. This application allows data to be stored in a distributed form.

There are two primary tasks in MapReduce: map and reduce. We perform the former task before the latter. In the map job, we split the input dataset into *chunks*. Map task processes these chunks in parallel. The *map* we use outputs as inputs for the *reduce* tasks. Reducers process the intermediate data from the maps into smaller tuples, that reduces the tasks, leading to the final output of the framework.

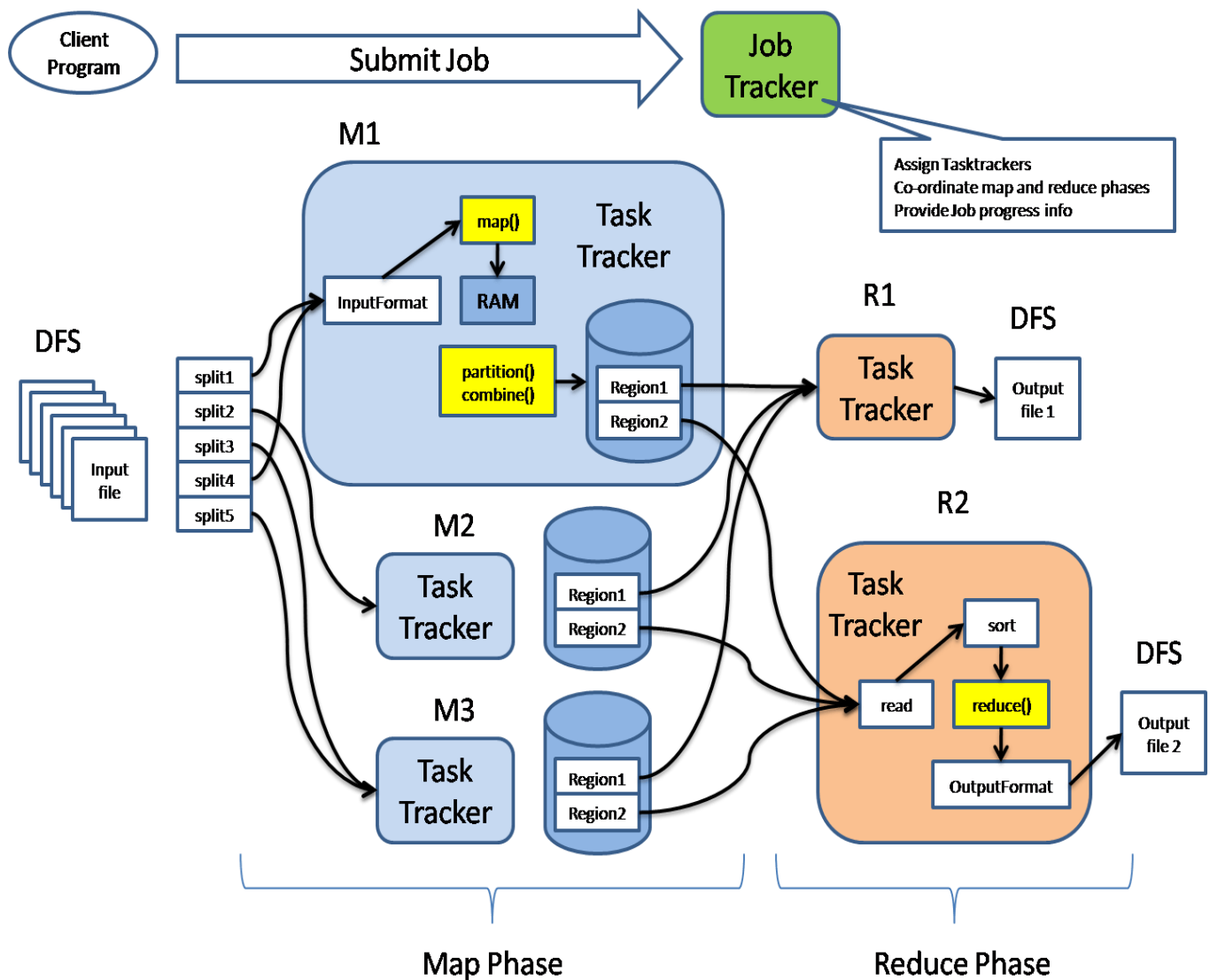
The MapReduce framework enhances the scheduling and monitoring of tasks. The failed tasks are re-executed by the framework. This framework can be used easily, even by programmers with little expertise in distributed processing. MapReduce can be implemented using various programming languages such as Java, Hive, Pig, Scala, and Python.

How MapReduce in Hadoop works

An overview of *MapReduce Architecture* and *MapReduce's phases* will help us understand how MapReduce in Hadoop works.

MapReduce architecture

The following diagram shows a MapReduce architecture.



MapReduce architecture consists of various components. A brief description of these components can improve our understanding on how MapReduce works.

- **Job:** This is the actual work that needs to be executed or processed
- **Task:** This is a piece of the actual work that needs to be executed or processed. A MapReduce job comprises many small tasks that need to be executed.
- **Job Tracker:** This tracker plays the role of scheduling jobs and tracking all jobs assigned to the task tracker.
- **Task Tracker:** This tracker plays the role of tracking tasks and reporting the status of tasks to the job tracker.
- **Input data:** This is the data used to process in the mapping phase.
- **Output data:** This is the result of mapping and reducing.

- **Client:** This is a program or Application Programming Interface (API) that submits jobs to the MapReduce. MapReduce can accept jobs from many clients.
- **Hadoop MapReduce Master:** This plays the role of dividing jobs into job-parts.
- **Job-parts:** These are sub-jobs that result from the division of the main job.

In the MapReduce architecture, clients submit jobs to the MapReduce Master. This master will then sub-divide the job into equal sub-parts. The job-parts will be used for the two main tasks in MapReduce: mapping and reducing.

The developer will write logic that satisfies the requirements of the organization or company. The input data will be split and mapped.

The intermediate data will then be sorted and merged. The reducer that will generate a final output stored in the HDFS will process the resulting output.

The following diagram shows a simplified flow diagram for the MapReduce program.



4.7.1 Phases of MapReduce

The MapReduce program is executed in three main phases: mapping, shuffling, and reducing. There is also an optional phase known as the combiner phase.

Mapping Phase

This is the first phase of the program. There are two steps in this phase: splitting and mapping. A dataset is split into equal units called *chunks (input splits)* in the splitting step. Hadoop consists of a RecordReader that uses `TextInputFormat` to transform input splits into key-value pairs.

The key-value pairs are then used as inputs in the mapping step. This is the only data format that a mapper can read or understand. The mapping step contains a coding logic that is applied to these data blocks. In this step, the mapper processes the key-value pairs and produces an output of the same form (key-value pairs).

Shuffling phase

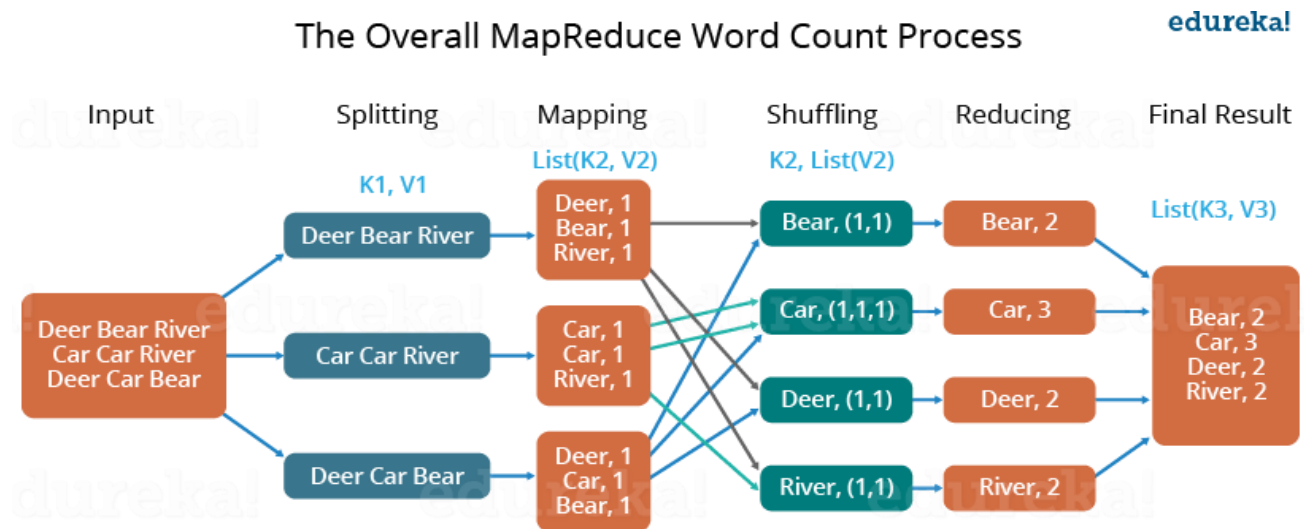
This is the second phase that takes place after the completion of the Mapping phase. It consists of two main steps: sorting and merging. In the sorting step, the key-value pairs are sorted using the keys. Merging ensures that key-value pairs are combined.

The shuffling phase facilitates the removal of duplicate values and the grouping of values. Different values with similar keys are grouped. The output of this phase will be keys and values, just like in the Mapping phase.

Reducer phase

In the reducer phase, the output of the shuffling phase is used as the input. The reducer processes this input further to reduce the intermediate values into smaller values. It provides a summary of the entire dataset. The output from this phase is stored in the HDFS.

The following diagram shows an example of a MapReduce with the three main phases. Splitting is often included in the mapping stage.



Benefits of Hadoop MapReduce

- **Speed:** MapReduce can process huge unstructured data in a short time.
- **Fault-tolerance:** The MapReduce framework can handle failures.
- **Cost-effective:** Hadoop has a scale-out feature that enables users to process or store data in a cost-effective manner.

- **Scalability:** Hadoop provides a highly scalable framework. MapReduce allows users to run applications from many nodes.
- **Data availability:** Replicas of data are sent to various nodes within the network. This ensures copies of the data are available in the event of failure.
- **Parallel Processing:** In MapReduce, multiple job-parts of the same dataset can be processed in a parallel manner. This reduces the time taken to complete a task.

4.7.2 Applications of Hadoop MapReduce

The following are some of the practical applications of the MapReduce program.

E-commerce

E-commerce companies such as Walmart, E-Bay, and Amazon use MapReduce to analyze buying behavior. MapReduce provides meaningful information that is used as the basis for developing product recommendations. Some of the information used include site records, e-commerce catalogs, purchase history, and interaction logs.

Social networks

The MapReduce programming tool can evaluate certain information on social media platforms such as Facebook, Twitter, and LinkedIn. It can evaluate important information such as who liked your status and who viewed your profile.

Entertainment

Netflix uses MapReduce to analyze the clicks and logs of online customers. This information helps the company suggest movies based on customers' interests and behavior.

Unit 5: Advanced database models, System and Applications

5.1 Active Database Concept and Triggers

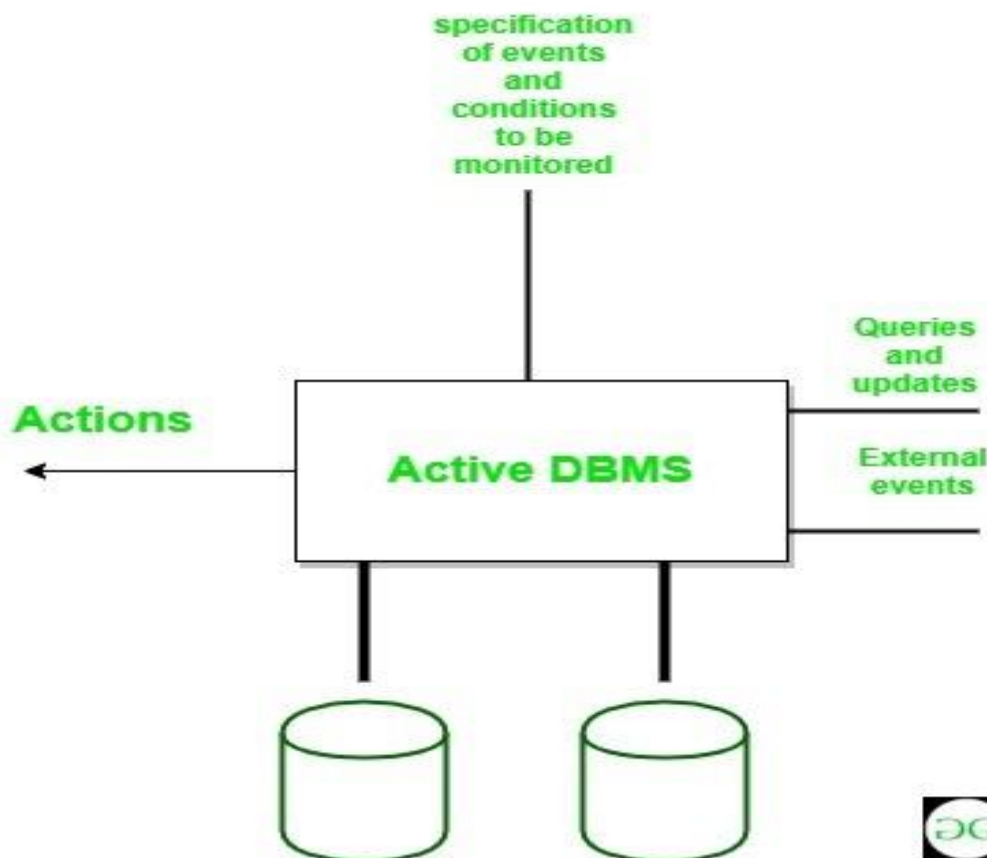
A trigger is a procedure which is automatically invoked by the DBMS in response to changes to the database, and is specified by the database administrator (DBA). A database with the set of associated triggers is generally called an active database. These database are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database is activated or not, prior to executing the statement. If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement.

Active Database is a database consisting of set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers.

In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database) is activated or not, prior to executing the statement.

If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement.

In such situation, DBMS processes each of the trigger randomly. The execution of an action part of a trigger may either activate other triggers or the same trigger that Initialized this action. Such types of trigger that activates itself is called as 'recursive trigger'. The DBMS executes such chains of trigger in some pre-defined manner but it effects the concept of understanding.



Features of Active Database:

1. It possess all the concepts of a conventional database i.e. data modelling facilities, query language etc.
2. It supports all the functions of a traditional database like data definition, data manipulation, storage management etc.
3. It supports definition and management of ECA rules.
4. It detects event occurrence.
5. It must be able to evaluate conditions and to execute actions.
6. It means that it has to implement rule execution.

Advantages :

1. Enhances traditional database functionalities with powerful rule processing capabilities.

2. Enable a uniform and centralized description of the business rules relevant to the information system.
3. Avoids redundancy of checking and repair operations.
4. Suitable platform for building large and efficient knowledge base and expert systems.

A trigger is a procedure which is automatically invoked by the DBMS in response to changes to the database, and is specified by the database administrator (DBA). A database with a set of associated triggers is generally called an active database.

Parts of trigger

A triggers description contains three parts, which are as follows –

- **Event** – An event is a change to the database which activates the trigger.
- **Condition** – A query that is run when the trigger is activated is called as a condition.
- **Action** – A procedure which is executed when the trigger is activated and its condition is true.

Create database trigger

To create a database trigger, we use the CREATE TRIGGER command. The details to be given at the time of creating a trigger are as follows –

- Name of the trigger.
- Table to be associated with.
- When trigger is to be fired: before or after.
- Command that invokes the trigger- UPDATE, DELETE, or INSERT.
- Whether row-level triggers or not.
- Condition to filter rows.
- PL/SQL block is to be executed when trigger is fired.

The syntax to create database trigger is as follows –

```
CREATE [OR REPLACE] TRIGGER triggername
{BEFORE|AFTER}
{DELETE|INSERT|UPDATE[OF COLUMNS]} ON table
[FOR EACH ROW { WHEN condition}]
[REFERENCE [OLD AS old] [NEW AS new]]
BEGIN
PL/SQL BLOCK
END.
```

5.2 Temporal database concepts:

A temporal database is a database that has certain features that support time-sensitive status for entries. Where some databases are considered current databases and only support factual data considered valid at the time of use, a temporal database can establish at what times certain entries are accurate.

Temporal databases support managing and accessing temporal data by providing one or more of the following features:

- A time period datatype, including the ability to represent time periods with no end (infinity or forever)
- The ability to define valid and transaction time period attributes and bitemporal relations
- System-maintained transaction time
- Temporal [primary keys](#), including non-overlapping period constraints
- Temporal constraints, including non-overlapping uniqueness and [referential integrity](#)
- Update and deletion of temporal records with automatic splitting and coalescing of time periods
- Temporal queries at current time, time points in the past or future, or over durations
- Predicates for querying time periods, often based on [Allen's interval relations](#)

5.3 Spatial database concepts

A spatial database is a database that is enhanced to store and access spatial data or data that defines a geometric space. These data are often associated with geographic locations and features, or constructed features like cities. Data on spatial databases are stored as coordinates, points, lines, polygons and topology. Some spatial databases handle more complex data like three-dimensional objects, topological coverage and linear networks.

The main goal of a spatial database system is the effective and efficient handling of spatial data types in two, three or higher dimensional spaces, and the ability to answer queries taking into consideration the spatial data properties. Examples of spatial data types are:

- point: characterized by a pair of (x,y) values,
- line segment: characterized by a pair of points.
- rectangle: characterized by its lower-left and upper-right corners,
- polygon: comprised by a set of points, defining its corners.

1. Spatial Query Processing In traditional database systems user queries are usually expressed by SQL statements containing conditions among the attributes of the relations (database tables). A spatial database system must be equipped with additional functionality to answer queries containing conditions among the spatial attributes of the database objects, such as location, extend and geometry. The most common spatial query types are:

- topological queries (e.g., find all objects that overlap or cover a given object),
- directional queries (e.g., find all objects that lie north of a given object),
- distance queries (e.g., find all objects that lie in less than a given distance from a given object).

5.4 Multimedia database concept

Multimedia database is a collection of multimedia data which includes text, images, graphics (drawings, sketches), animations, audio, video, among others. These databases have extensive amounts of data which can be multimedia

and multisource. The framework which manages these multimedia databases and their different types so that the data can be stored, utilized, and delivered in more than one way is known as a multimedia database management system.

The multimedia database can be classified into three types. These types are:

1. Static media
2. Dynamic media
3. Dimensional media

Multimedia Database Applications:

1. **Documents and record management:** Industries which keep a lot of documentation and records. Ex: Insurance claim industry.
2. **Knowledge dissemination:** Multimedia database is an extremely efficient tool for knowledge dissemination and providing several resources. Ex: electronic books
3. **Education and training:** Multimedia sources can be used to create resources useful in education and training. These are popular sources of learning in recent days. Ex: Digital libraries.
4. **Real-time monitoring and control:** Multimedia presentation when coupled with active database technology can be an effective means for controlling and monitoring complex tasks. Ex: Manufacture control
5. Marketing
6. Advertisement
7. Retailing
8. Entertainment
9. Travel

5.5 Deductive database concepts

A **deductive database** is a [database system](#) that can make [deductions](#) (i.e. conclude additional facts) based on [rules](#) and [facts](#) stored in the (deductive) database. [Data log](#) is the language typically used to specify facts, rules and queries in deductive databases. Deductive databases have grown out of the desire to combine [logic programming](#) with [relational databases](#) to construct systems that support a powerful formalism and are still fast and able to deal with very large datasets. Deductive databases are more expressive than relational databases but less [expressive](#) than logic programming systems. In recent years, deductive databases such as data log have found new application in [data integration](#), [information extraction](#), networking, [program analysis](#), security, and cloud computing.

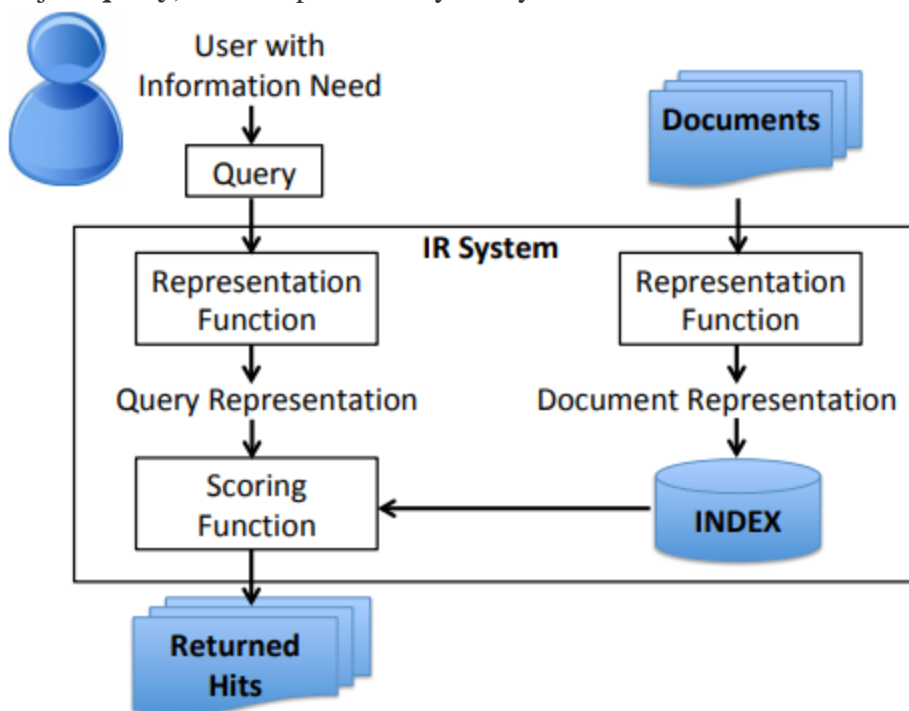
Deductive databases reuse many concepts from logic programming; rules and facts specified in the deductive database language Data log look very similar to those in [Prolog](#). However important differences between deductive databases and logic programming:

- Order sensitivity and procedurally: In Prolog, program execution depends on the order of rules in the program and on the order of parts of rules; these properties are used by programmers to build efficient programs. In database languages (like SQL or Data log), however, program execution is independent of the order of rules and facts.
- Special predicates: In Prolog, programmers can directly influence the procedural evaluation of the program with special predicates such as the [cut](#), this has no correspondence in deductive databases.

- Function symbols: Logic Programming languages allow [function symbols](#) to build up complex symbols. This is not allowed in deductive databases.
- [Tuple](#)-oriented processing: Deductive databases use set-oriented processing while logic programming languages concentrate on one tuple at a time.

5.6 Introduction to Information Retrieval and Web Search

Information retrieval deals mainly with *unstructured data*, and the techniques for indexing, searching, and retrieving information from large collections of unstructured documents. In this chapter we will provide an introduction to information retrieval. This is a very broad topic, so we will focus on the similarities and differences between information retrieval and database technologies, and on the indexing techniques that form the basis of many information retrieval systems. **information retrieval** is “the discipline that deals with the structure, analysis, organization, storage, searching, and retrieval of information” as defined by Gerald Salton, an IR pioneer. We can enhance the definition slightly to say that it applies in the context of unstructured documents to satisfy a user’s information needs. This field has existed even longer than the database field, and was originally concerned with retrieval of cataloged information in libraries based on titles, authors, topics, and keywords. In academic programs, the field of IR has long been a part of Library and Information Science programs. Information in the context of IR does not require machine-understandable structures, such as in relational database systems. Examples of such information include written texts, abstracts, documents, books, Web pages, e-mails, instant messages, and collections from digital libraries. Therefore, all loosely represented (unstructured) or semi structured information is also part of the IR discipline. IR systems go beyond database systems in that they do not limit the user to a specific query language, nor do they expect the user to know the structure (schema) or content of a particular database. IR systems use a user’s information need expressed as a **free-form search request** (sometimes called a **keyword search query**, or just **query**) for interpretation by the system.



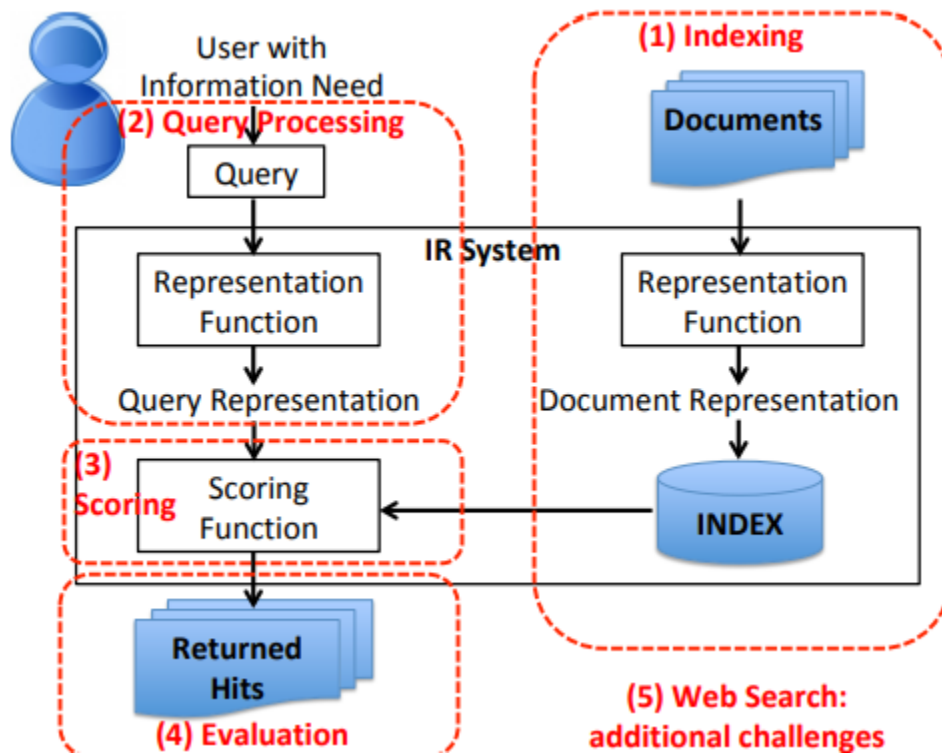


Figure: Structure of IR System

An IR system can be characterized at different levels

- 1.Types of users
- 2.Types of Data
3. Types of Information Need

Types of Users. The user may be an *expert user* (for example, a curator or a librarian), who is searching for specific information that is clear in his/her mind and forms relevant queries for the task, or a *layperson user* with a generic information need. The latter cannot create highly relevant queries for search (for example, students trying to find information about a new topic, researchers trying to assimilate different points of view about a historical issue, a scientist verifying a claim by another scientist, or a person trying to shop for clothing).

Types of Data. Search systems can be tailored to specific types of data. For example, the problem of retrieving information about a specific topic may be handled more efficiently by customized search systems that are built to collect and retrieve only information related to that specific topic. The information repository could be hierarchically organized based on a concept or topic hierarchy. These topical domain-specific or vertical IR systems are not as large as or as diverse as the generic World Wide Web, which contains information on all kinds of topics. Given that these domain-specific collections exist and may have been acquired through a specific process, they can be exploited much more efficiently by a specialized system.

Types of Information Need. In the context of Web search, users' information needs may be defined as navigational, informational, or transactional. **Navigational search** refers to finding a particular piece of information (such as the Georgia Tech University Website) that a user needs quickly. The purpose of **informational search** is to find current information about a topic (such as research activities in the college of computing at Georgia Tech—this is the classic IR system task). The goal of **transactional search** is to reach a site where further interaction happens (such as joining a social network, product shopping, online reservations, accessing databases, and so on).