CSC316 DATABASE SYSTEMS II NOTES

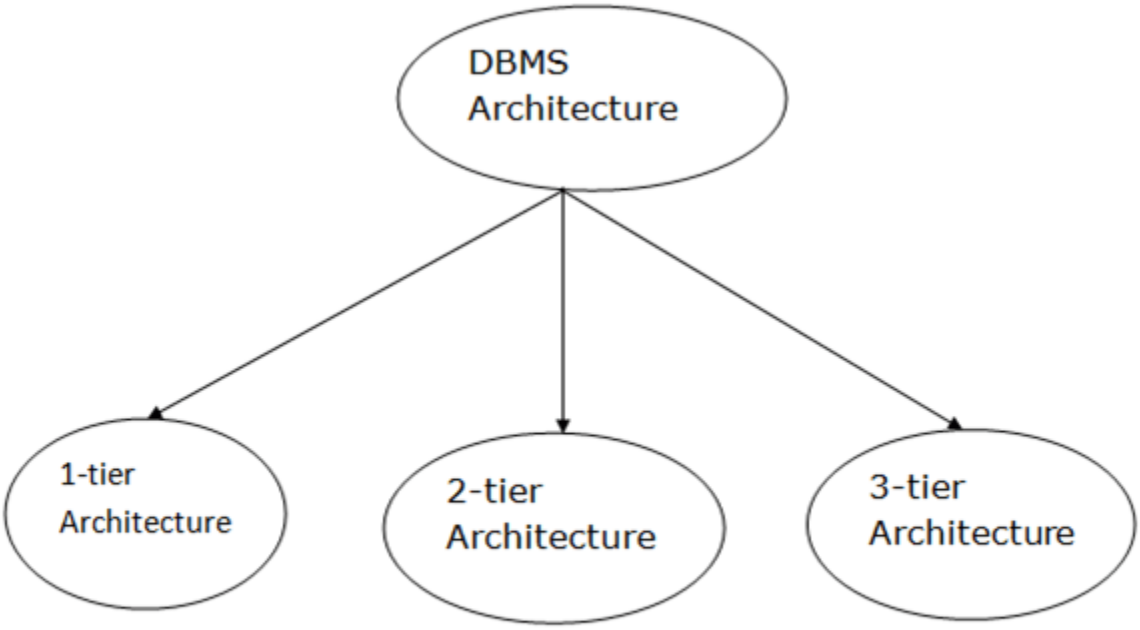
Course Name	**CSC 316: DATABASE SYSTEMS II**	
Credit Units	3	
Pre-requisite	CSC 221: Database Systems I	
Purpose	The purpose of this course is to introduce students to advanced topics in database systems	
Expected Learning Outcomes	At the end of the course, the students should be able to: 1. Design and implement databases using advanced techniques 2. Demonstrate an understanding of data warehouses 3. Demonstrate an understanding of the current and emerging trends in database systems	
Course Content	Overview of DBMS Architecture; Storage and indexing; Overview of relational databases: Entity-relationship model, Normalization; Relational operators (Relational Algebra, Relational Calculus); Query Processing and Query Optimization; Transaction Processing; Concurrency Control; Distributed Databases: Distributed database design, management of spatial data, and data from large scale distributed devices; Data Warehousing; Object-Oriented and Object-Relational Data; XML and Databases Emerging issues in Database systems: Database Integrity, Security, recovery and other emerging issues; Case study tools Oracle /SQL server/ My SQL.	
Mode of Delivery	Lectures, demonstrations, Group/class discussions and practical exercises	
Instructional Material and/or Equipment	Computers, Learning Management System, writing boards, writing materials, projectors etc.	
Course Assessment	Type	Weighting (%)
	Examination	70
	Continuous Assessment	30
	Total	100

Core Reading Materials	1. R. Ramakrishnan and J. Gehrke (2003), Database Management Systems, 3rd edition, McGraw-Hill. 2. Elmasri and Navathe (2004) Foundations of Database Systems, 4th, , Addison Wesley, 3. Silberschatz, Korth and Sudarshan, (2010), Database Systems Concepts, 6th edition, McGraw-Hill
Recommended Reading Materials	1. Garcia-Molina H., Ullman J. D. and Widom J. (2008) "Database Systems", 2nd edition, Prentice-Hall 2. Lewis P. M., Bernstein A., and Kifer M. (2002) "Database and Transaction Processing: An Application-Oriented Approach", Addison Wesley.

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



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

1-Tier Architecture

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

2-Tier Architecture

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

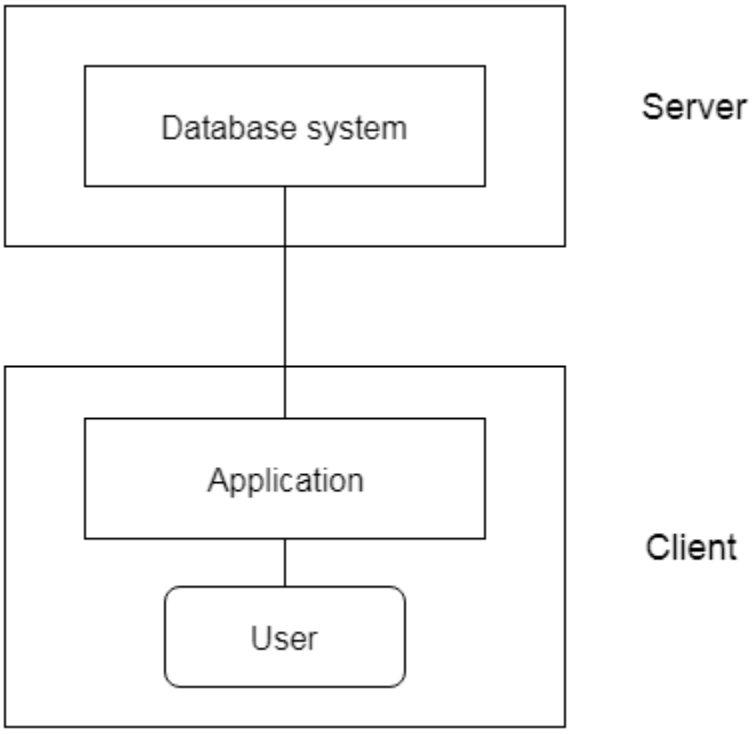


Fig: 2-tier Architecture

3-Tier Architecture

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

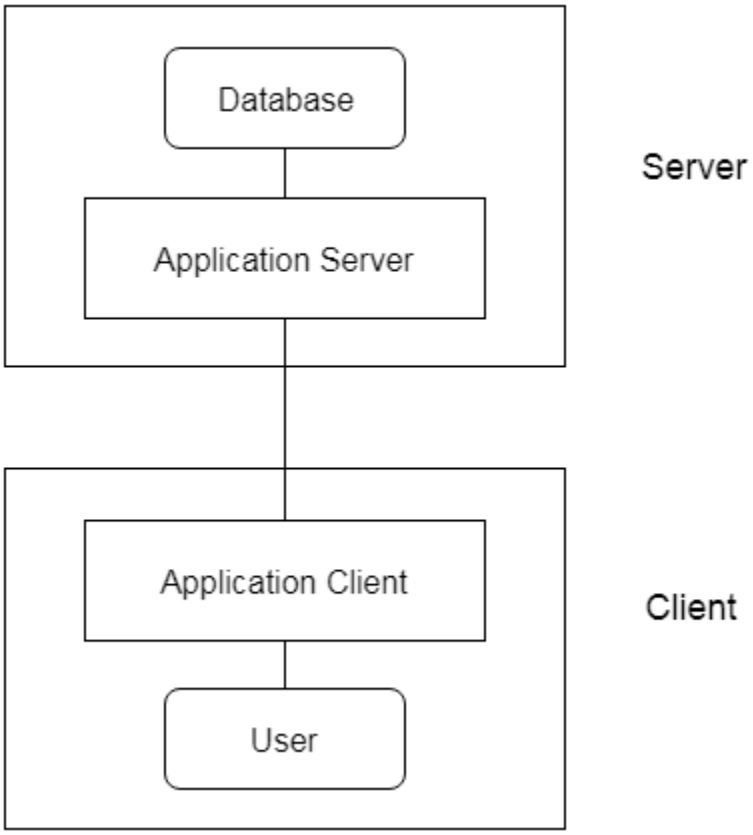


Fig: 3-tier Architecture

Indexing in DBMS

- Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Index structure:

Indexes can be created using some database columns.

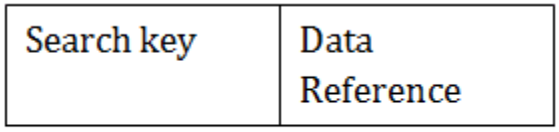
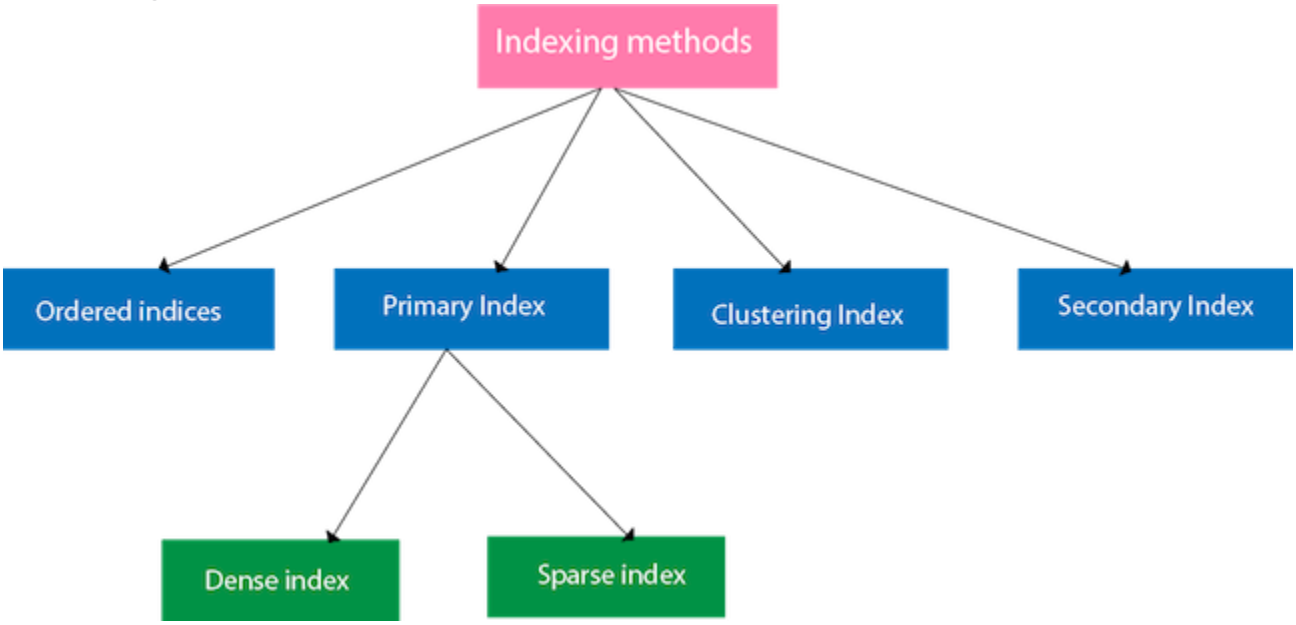


Fig: Structure of Index

- The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

Indexing Methods



Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

Example: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

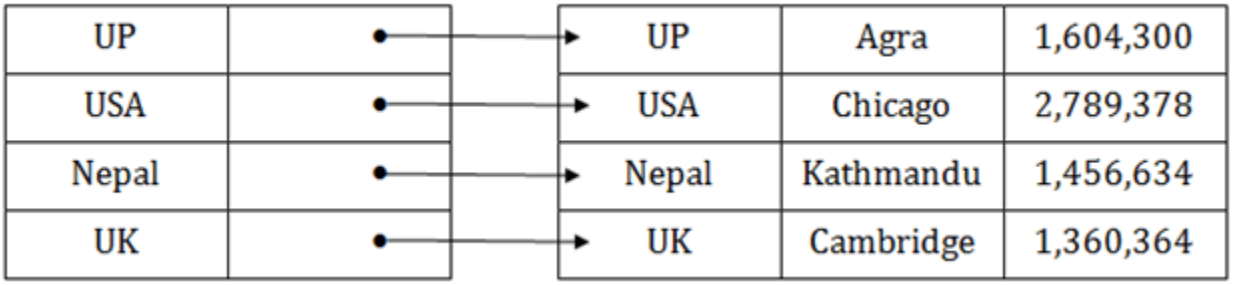
- In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543*10=5430 bytes.
- In the case of an index, we will search using indexes and the DBMS will read the record after reading 542*2= 1084 bytes which are very less compared to the previous case.

Primary Index

- If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
- As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- The primary index can be classified into two types: Dense index and Sparse index.

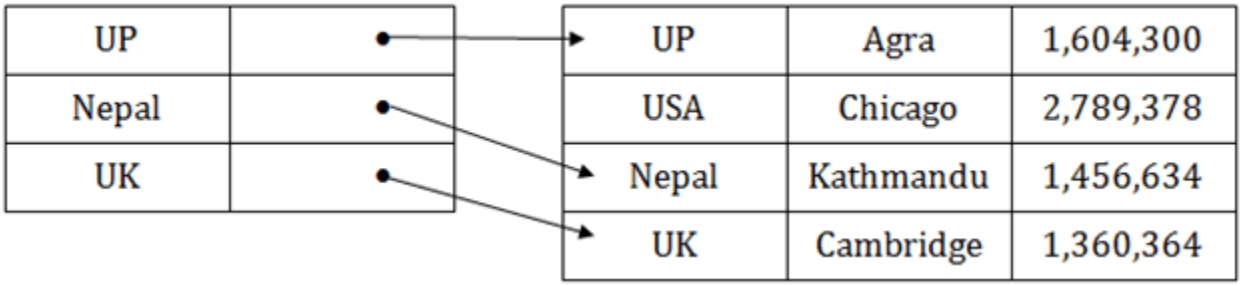
Dense index

- The dense index contains an index record for every search key value in the data file. It makes searching faster.
- In this, the number of records in the index table is same as the number of records in the main table.
- It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



Sparse index

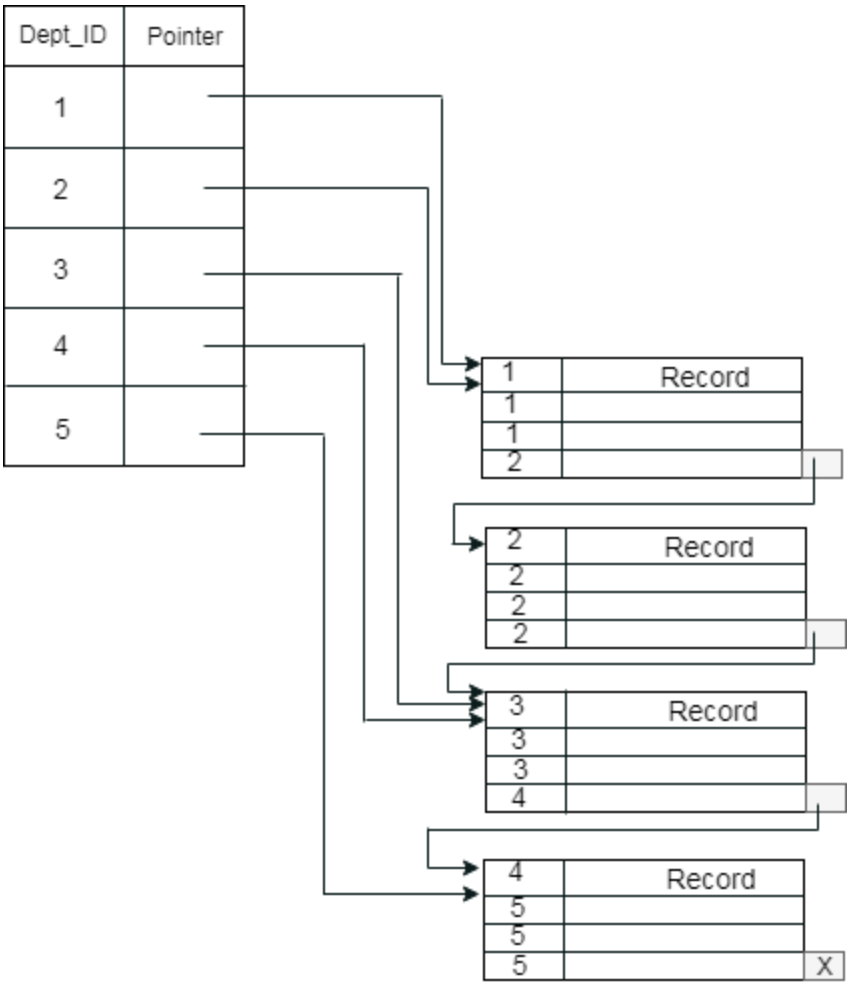
- In the data file, index record appears only for a few items. Each item points to a block.
- In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



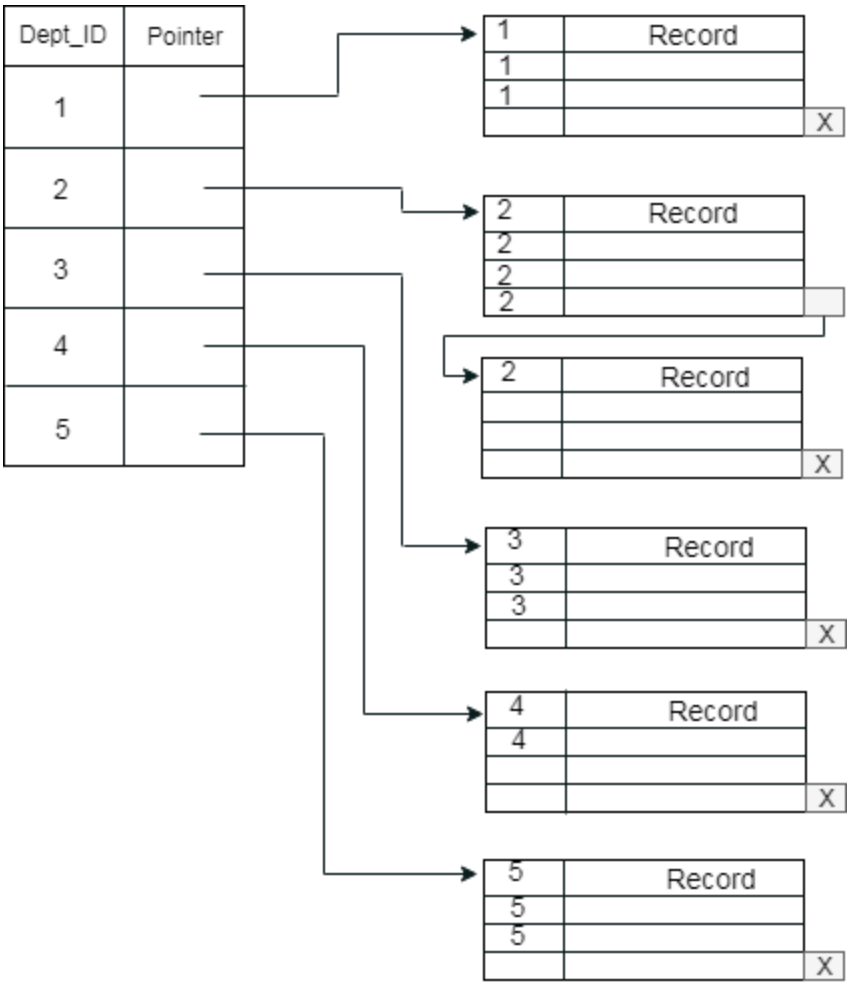
Clustering Index

- A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
- In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
- The records which have similar characteristics are grouped, and indexes are created for these group.

Example: suppose a company contains several employees in each department. Suppose we use a clustering index, where all employees which belong to the same Dept_ID are considered within a single cluster, and index pointers point to the cluster as a whole. Here Dept_Id is a non-unique key.



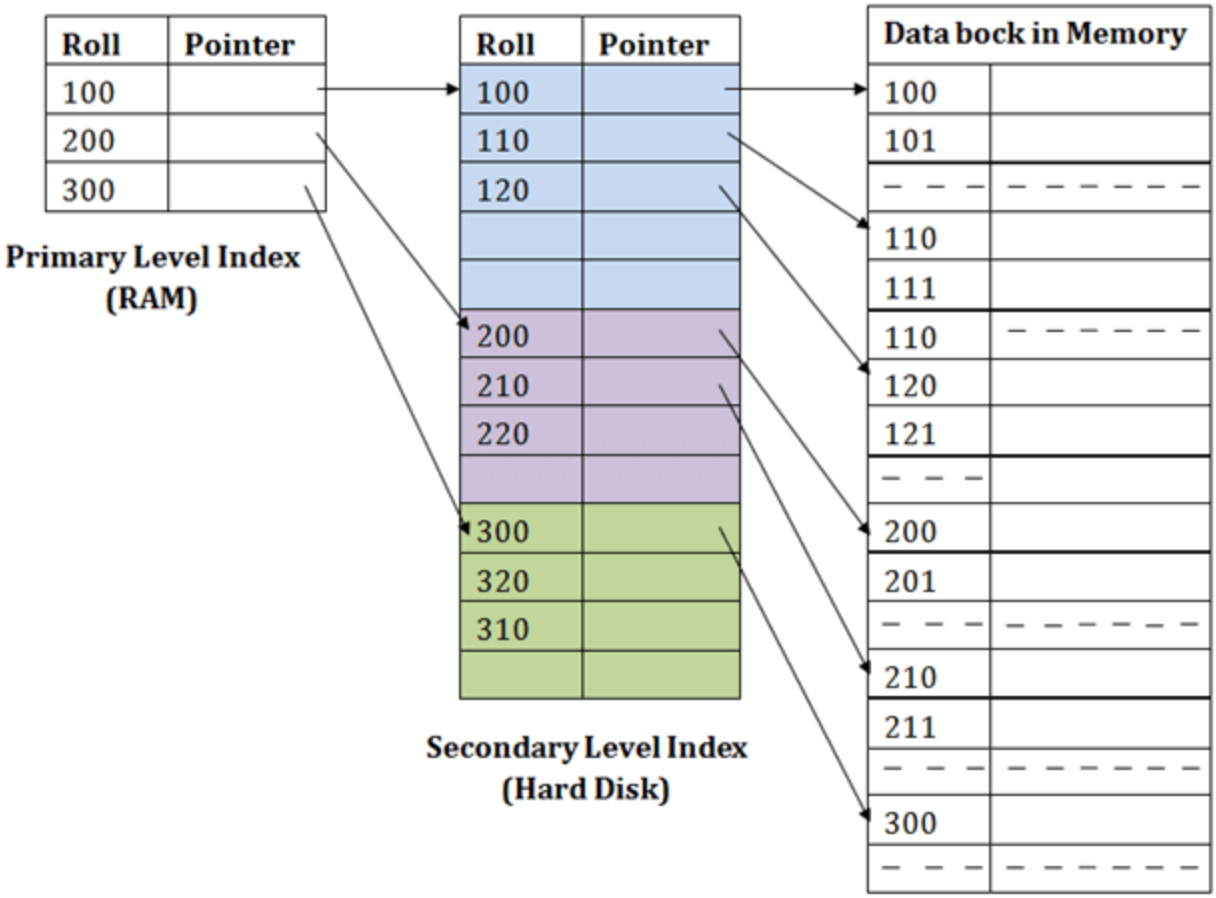
The previous schema is little confusing because one disk block is shared by records which belong to the different cluster. If we use separate disk block for separate clusters, then it is called better technique.



Secondary Index

In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).

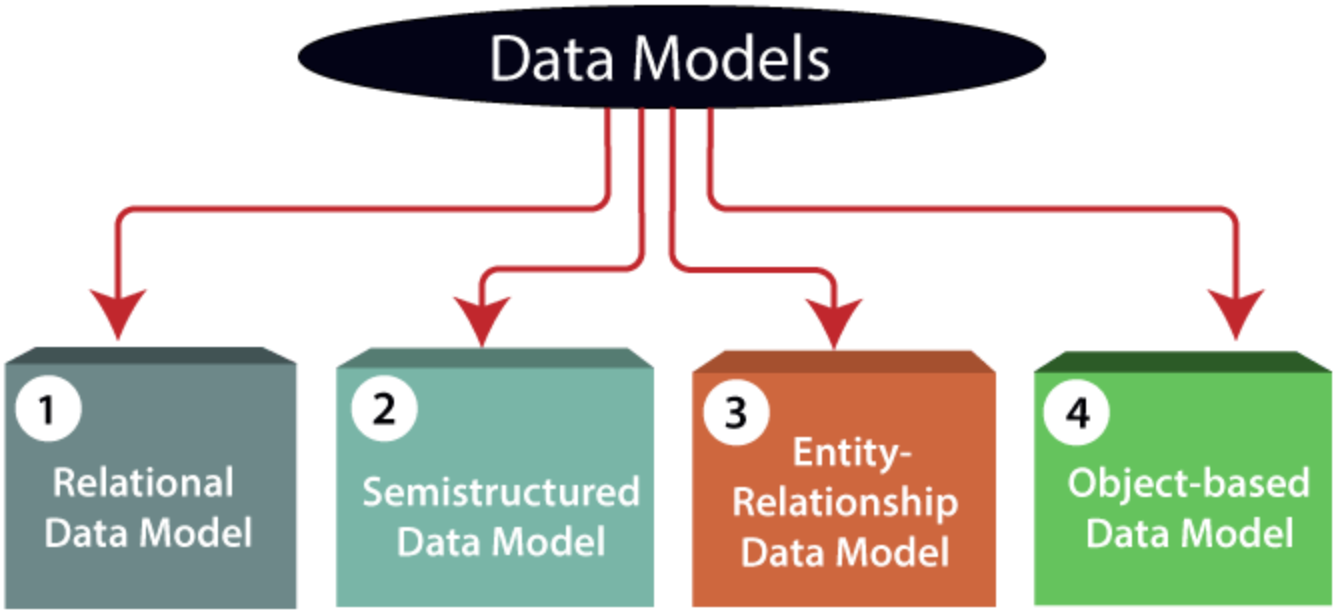


For example:

- If you want to find the record of roll 111 in the diagram, then it will search the highest entry which is smaller than or equal to 111 in the first level index. It will get 100 at this level.
- Then in the second index level, again it does max (111) <= 111 and gets 110. Now using the address 110, it goes to the data block and starts searching each record till it gets 111.
- This is how a search is performed in this method. Inserting, updating or deleting is also done in the same manner.

Data Models

Data Model is the modeling of the data description, data semantics, and consistency constraints of the data. It provides the conceptual tools for describing the design of a database at each level of data abstraction. Therefore, there are following four data models used for understanding the structure of the database:



1) Relational Data Model: This type of model designs the data in the form of rows and columns within a table. Thus, a relational model uses tables for representing data and in-between relationships. Tables are also called relations. This model was initially described by Edgar F. Codd, in 1969. The relational data model is the widely used model which is primarily used by commercial data processing applications.

2) Entity-Relationship Data Model: An ER model is the logical representation of data as objects and relationships among them. These objects are known as entities, and relationship is an association among these entities. This model was designed by Peter Chen and published in 1976 papers. It was widely used in database designing. A set of attributes describe the entities. For example, student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

3) Object-based Data Model: An extension of the ER model with notions of functions, encapsulation, and object identity, as well. This model supports a rich type system that includes structured and collection types. Thus, in 1980s, various database systems following the object-oriented approach were developed. Here, the objects are nothing but the data carrying its properties.

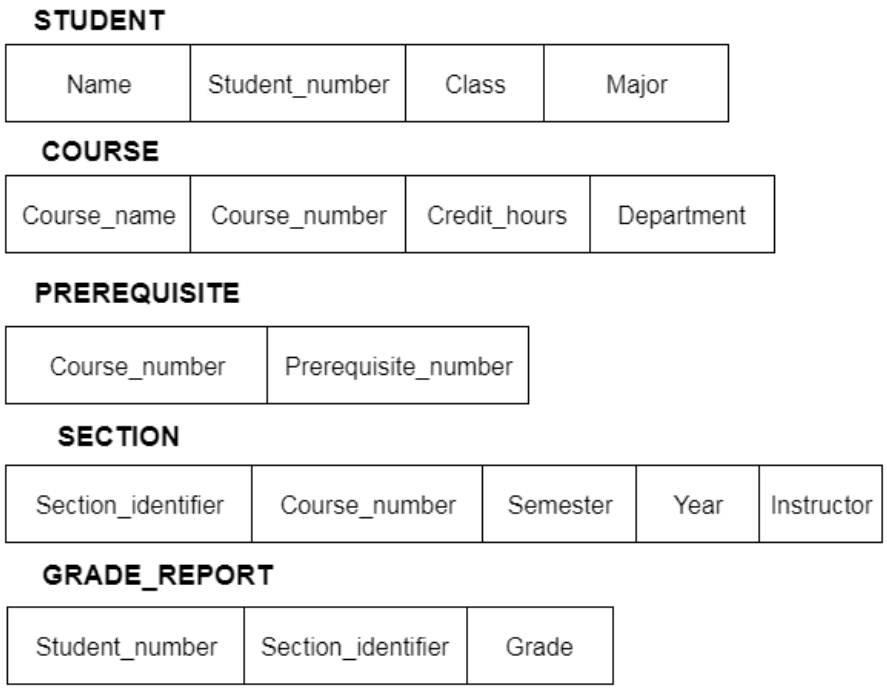
4) Semistructured Data Model: This type of data model is different from the other three data models (explained above). The semistructured data model allows the data specifications at places where the individual data items of the same type may have different attributes sets. The Extensible Markup Language, also known as XML, is widely used for representing the semistructured data. Although XML was initially designed for including the markup information to the text document, it gains importance because of its application in the exchange of data.

Data model Schema and Instance

- The data which is stored in the database at a particular moment of time is called an instance of the database.
- The overall design of a database is called schema.
- A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
- A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
- A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.
- A database schema is designed by the database designers to help programmers whose software will interact with the database. The process of database creation is called data modeling.

A schema diagram can display only some aspects of a schema like the name of record type, data type, and constraints. Other aspects can't be specified through the schema diagram. For example, the given figure neither show the data type of each data item nor the relationship among various files.

In the database, actual data changes quite frequently. For example, in the given figure, the database changes whenever we add a new grade or add a student. The data at a particular moment of time is called the instance of the database.



Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

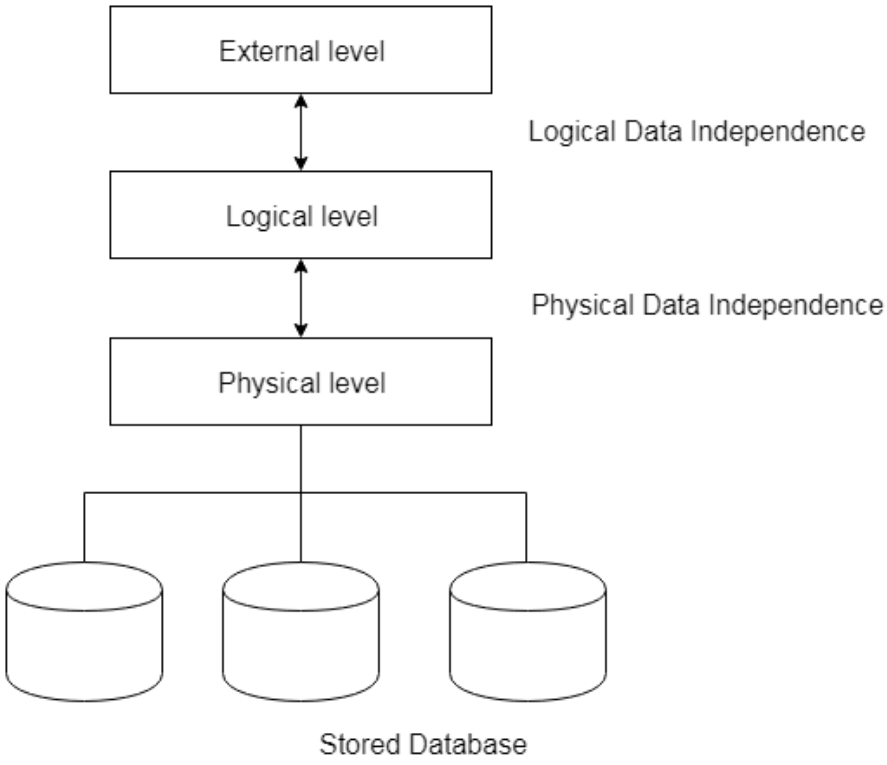
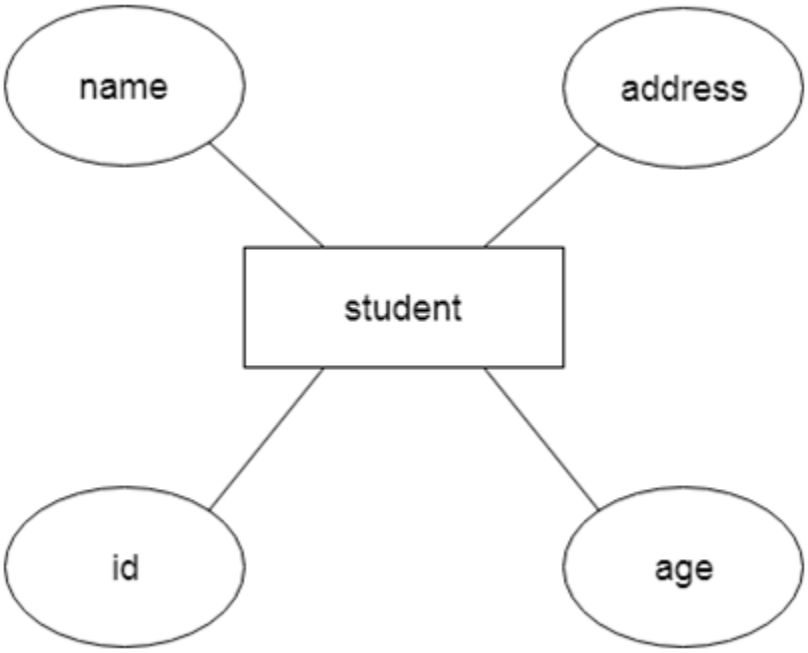


Fig: Data Independence

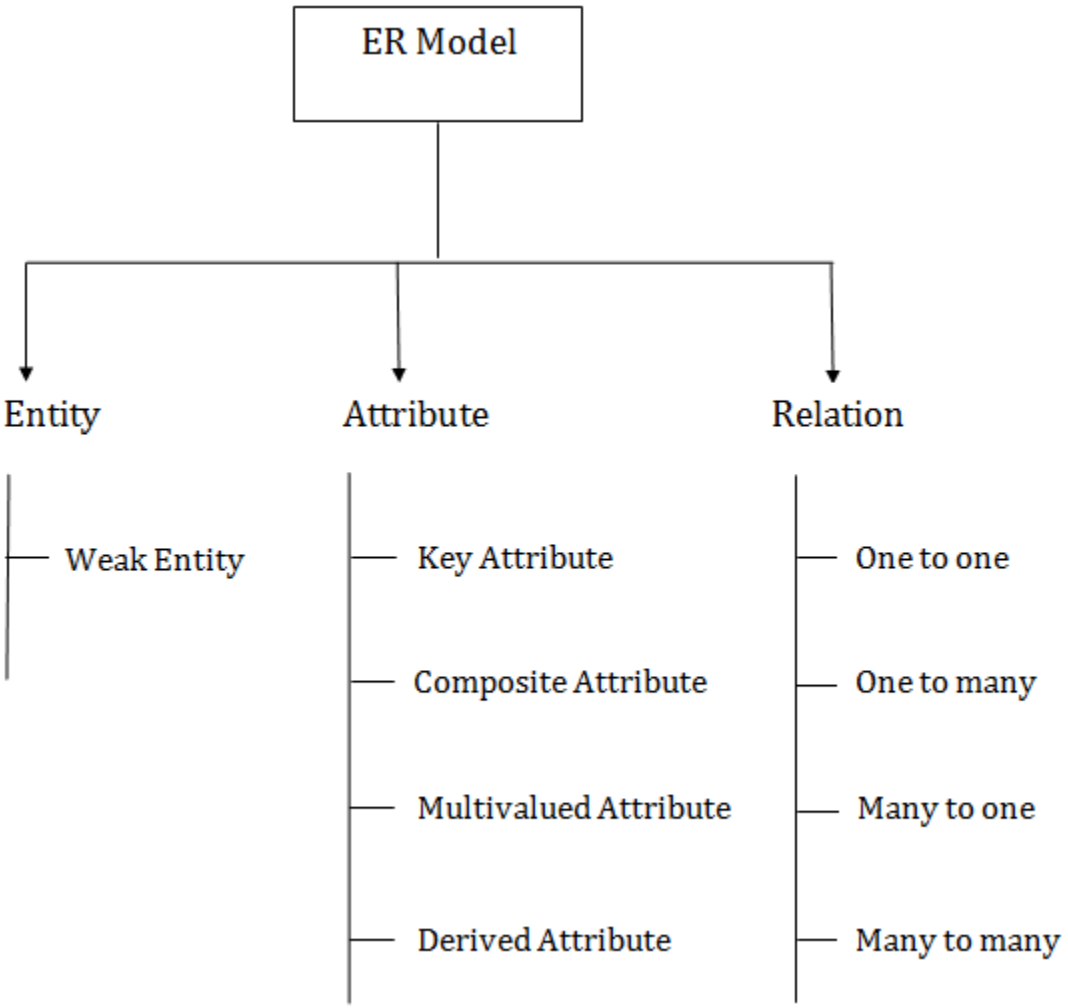
ER (Entity Relationship) Diagram in DBMS

- ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system.
- It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an entity-relationship diagram.

For example, Suppose we design a school database. In this database, the student will be an entity with attributes like address, name, id, age, etc. The address can be another entity with attributes like city, street name, pin code, etc and there will be a relationship between them.



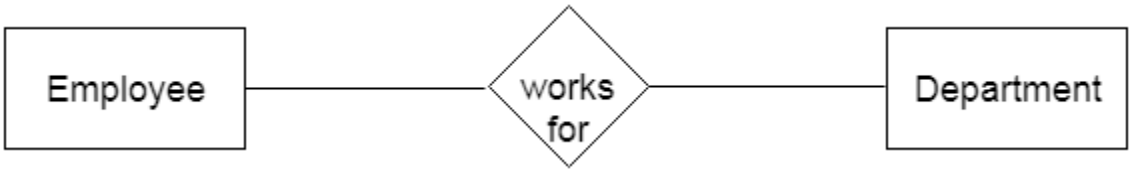
Component of ER Diagram



1. Entity:

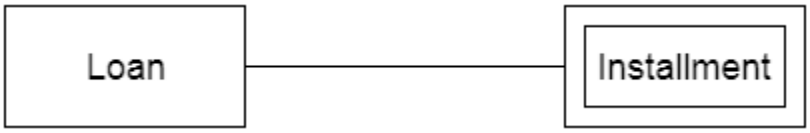
An entity may be any object, class, person or place. In the ER diagram, an entity can be represented as rectangles.

Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.



a. Weak Entity

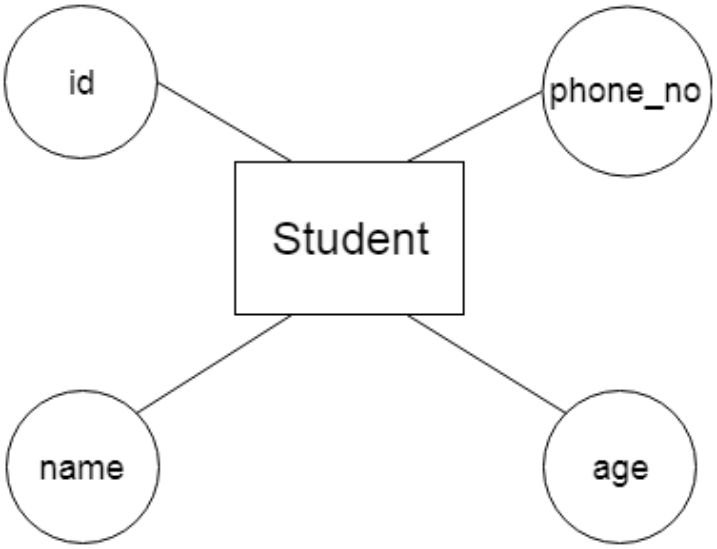
An entity that depends on another entity called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a double rectangle.



2. Attribute

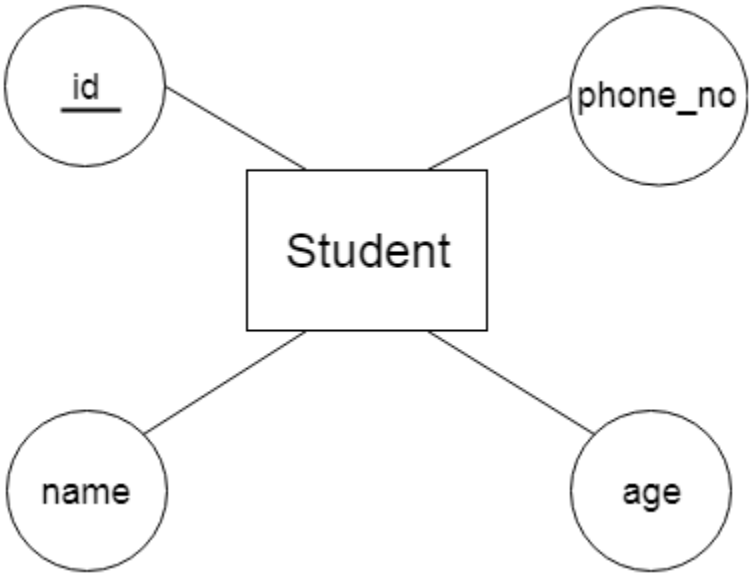
The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

For example, id, age, contact number, name, etc. can be attributes of a student.



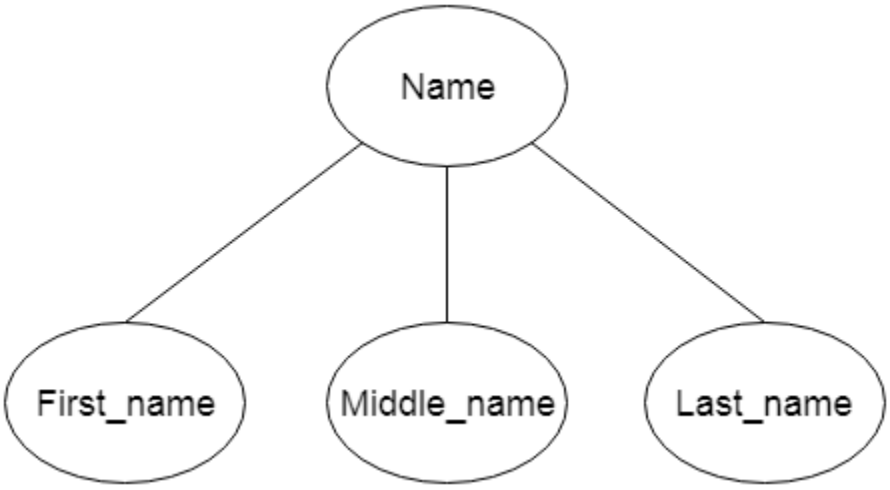
a. Key Attribute

The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.



b. Composite Attribute

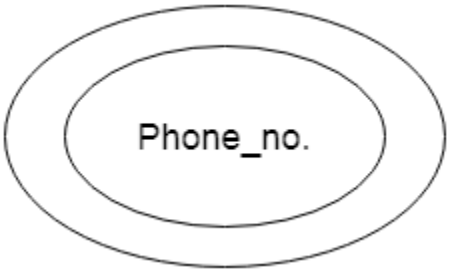
An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.



c. Multivalued Attribute

An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.

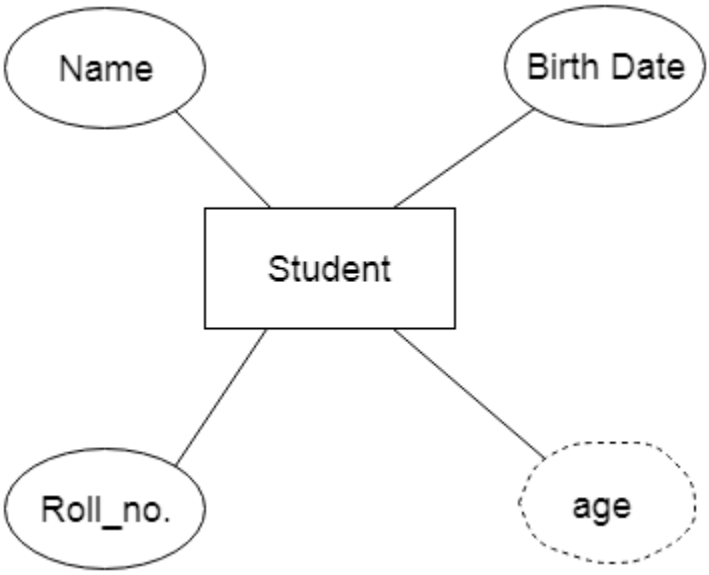
For example, a student can have more than one phone number.



d. Derived Attribute

An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

For example, A person's age changes over time and can be derived from another attribute like Date of birth.



3. Relationship

A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

Types of relationship are as follows:

a. One-to-One Relationship

When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

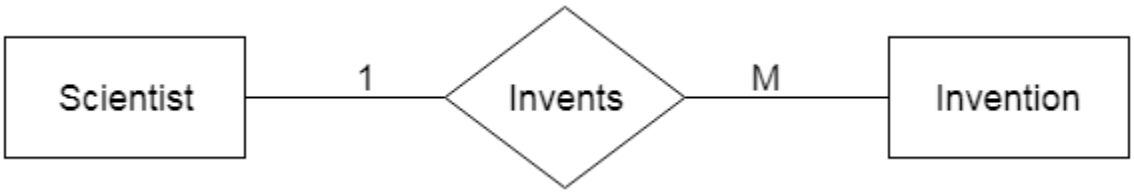
For example, A female can marry to one male, and a male can marry to one female.

Advertisement

b. One-to-many relationship

When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

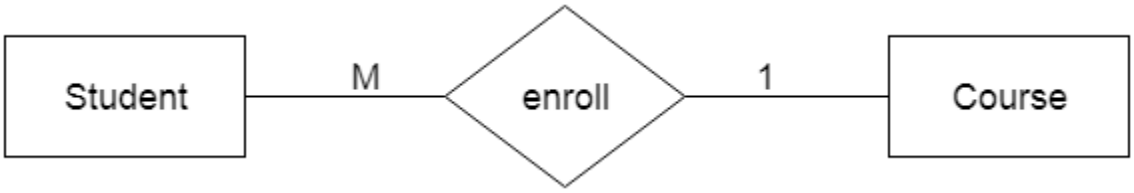
For example, Scientist can invent many inventions, but the invention is done by the only specific scientist.



c. Many-to-one relationship

When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.

For example, Student enrolls for only one course, but a course can have many students.



d. Many-to-many relationship

When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

For example, Employee can assign by many projects and project can have many employees.

Relational Model in DBMS

Relational model makes the query much easier than in hierarchical or network database systems. In 1970, E.F Codd has been developed it. A relational database is defined as a group of independent tables which are linked to each other using some common fields of each related table. This model can be represented as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute. It is well knows in database technology because it is usually used to represent real-world objects and the relationships between them. Some popular relational databases are used nowadays like Oracle, Sybase, DB2, MySQL Server etc.

Relational Model Terminologies:

Following are the terminologies of Relational Model:

Relation	Table

Tuple	Row, Record
Attribute	Column, Field
Domain	It consists of set of legal values
Cardinality	It consists of number of rows
Degree	It contains number of columns

Let's explain each term one by one in detail with the help of example:

Example: STUDENT Relation

Stu_No	S_Name	PHONE_NO	ADDRESS	Gender
10112	Rama	9874567891	Islam ganj	F
12839	Shyam	9026288936	Delhi	M
33289	Laxman	8583287182	Gurugram	M
27857	Mahesh	7086819134	Ghaziabad	M
17282	Ganesh	9028939884	Delhi	M

Relation: A relation is usually represented as a table, organized into rows and columns. A relationship consists of multiple records. **For example:** student relation which contains tuples and attributes.

Tuple: The rows of a relation that contain the values corresponding to the attributes are called tuples. **For example:** in the Student relation there are 5 tuples.

The value of tuples contains (10112, Rama, 9874567891,islam ganj, F) etc.

Data Item: The smallest unit of data in the relation is the individual data item. It is stored at the intersection of rows and columns are also known as cells. **For Example:** 10112, "Rama" etc are data items in Student relation.

Domain: It contains a set of atomic values that an attribute can take. It could be accomplish explicitly by listing all possible values or specifying conditions that all values in that domain must be confirmed. **For example:** the domain of gender attributes is a set of data values "M" for male and "F" for female. No database software fully supports domains typically allowing the users to define very simple data types such as numbers, dates, characters etc.

Attribute: The smallest unit of data in relational model is an attribute. It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai). **For example:** Stu_No, S_Name, PHONE_NO, ADDRESS, Gender are the attributes of a student relation. In relational databases a column entry in any row is a single value that contains exactly one item only.

Cardinality: The total number of rows at a time in a relation is called the cardinality of that relation. For example: In a student relation, the total number of tuples in this relation is3 so the cardinality of a relation is 3. The cardinality of a relation changes with time as more and more tuples get added or deleted.

Degree: The degree of association is called the total number of attributes in a relationship. The relation with one attribute is called unary relation, with two attributes is known a binary relation and with three attributes is known as ternary relation. **For example:** in the Student relation, the total number of attributes is 5, so the degree of the relations is 5. The degree of a relation does not change with time as tuples get added or deleted.

Relational instance: In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

Relational schema: A relational schema contains the name of the relation and name of all columns or attributes.

Relational key: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Properties of Relations

- Each attribute in a relation has only one data value corresponding to it i.e. they do not contain two or more values.

E_Id	Ename	Deptt.	Salary	Address
102	Anil	Prog.	20000	38-B, Lwr Rd 90-A, Basant Av.

javaTpoint

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- tuple has no duplicate value
- Order of tuple can have a different sequence
- It also provides information about metadata.

Merits of Relational Model:

Following are the various merits of relational model:

- This provides an abstract view of the data. It abstracts the physical structure from the logical structure of data.
- This model is very easy to design. Tables can use different attributes as per requirements.
- The relational model supports data independence. In a relational database the data is stored in tables so that we can modify the data without changing the physical structure.
- Relational database helps the user to use a query language to query the database.
- It offers more flexibility than other models.
- By moving sensitive attributes, we can also implement database security control and authorization in a particular table into a separate relation with its authorization controls.
- Relational database helps the user to use a query language to query the database.
- A relational model consists of simple relationships. The characteristics of a database that make it immune to certain maintenance problems have been developed in the context of relational models.
- It is useful for representing most real world objects and the relationships between them. It is very easy to implement a relationship through the use of a composite key, so this model persistence method dominates the market.

Demerits of Relational Model:

Most of the drawbacks of the relational database is not because of the shortcoming but because of the way it is being implemented, we can avoid the drawbacks of the relational model by using proper designing techniques and proper database standards are enforced. **Following are the various demerits of relational model:**

- The main disadvantage of relational models is that they do not support binary data **for example:** images, documents, spreadsheets etc.
- The relational model can easily adapt to new hardware so incurs large hardware overhead.
- Relational databases use a simple mapping of logical tables to physical structures.

- This mostly limits performance and allows non-relational systems such as object oriented management systems to perform better on specialised applications such as CAD, CAM etc.
- Enforcing data integrity in relational models is difficult because no single piece of hardware has control over the data.
- The relational model is suitable for small databases but not suitable for complex databases because the user needs to know the complex physical data storage details. So, while designing the databases they don't come to light when they may cause problems. When a database grows it will slow down the system and will result in performance degradation and data corruption.

Operations on Relational Model:

List of the following basic operations that can be performed on a relational model:

- Insertion Operation
- Deletion Operation
- Update Operation
- Retrieval Operation

Let's explain each operation one by one.

Insert operation: It is used to insert a new record in the table. Adding new records to the table is much easier than other models. Data values will not be found in a relation when the following condition occurs:

- If we try to insert a duplicate value for the field that is selected as a primary key.
- If we insert a NULL value in the attribute that contains primary key.
- If we try to enter a data value in the foreign key attribute that does not exist in corresponding primary key attribute.
- If an attribute is assigned a value that does not exist in the corresponding domain.

DELETE operation: This operation is used to delete records from the table but problems arise when the rows to be deleted have some attributes which are foreign key attributes.

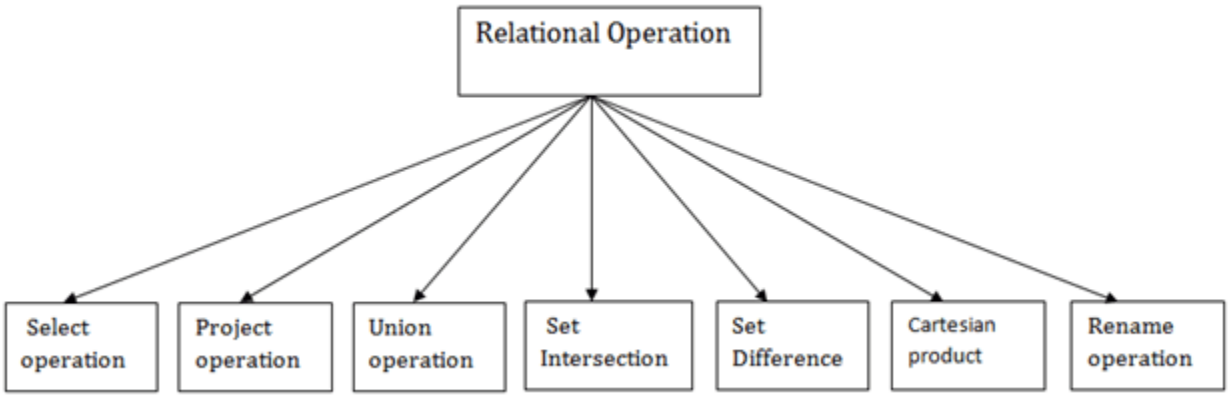
Update operation: It is used to modify or change the data value of a record in a table. Updating an attribute that is neither a primary key nor a foreign key requires only checking that the new value is of the correct data type and domain. If we modify a data value of a primary key and foreign key attribute then need to check:

- The modified value does not contain the value of the corresponding foreign key value.
- The new values must not alrea**Retrieval operation:** It is used to save a record from a relation. This operation is very simple and homogeneous.

Relational Algebra

Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

1. Notation: σ p(r)

Where:

σ is used for selection prediction
r is used for relation
p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500

Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

1. σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by ∏.

1. Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye

Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

1. ∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

1. Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15

Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry

Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

1. Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

1. Notation: R - S

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME

Jackson
Hayes
Willians
Curry

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

1. Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales

C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. ρ(STUDENT1, STUDENT)

Relational Calculus

There is an alternate way of formulating queries known as Relational Calculus. Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results. The relational calculus tells what to do but never explains how to do. Most commercial relational languages are based on aspects of relational calculus including SQL-QBE and QUEL.

Why it is called Relational Calculus?

It is based on Predicate calculus, a name derived from branch of symbolic language. A predicate is a truth-valued function with arguments. On substituting values for the arguments, the function result in an expression called a proposition. It can be either true or false. It is a tailored version of a subset of the Predicate Calculus to communicate with the relational database.

Many of the calculus expressions involves the use of Quantifiers. There are two types of quantifiers:

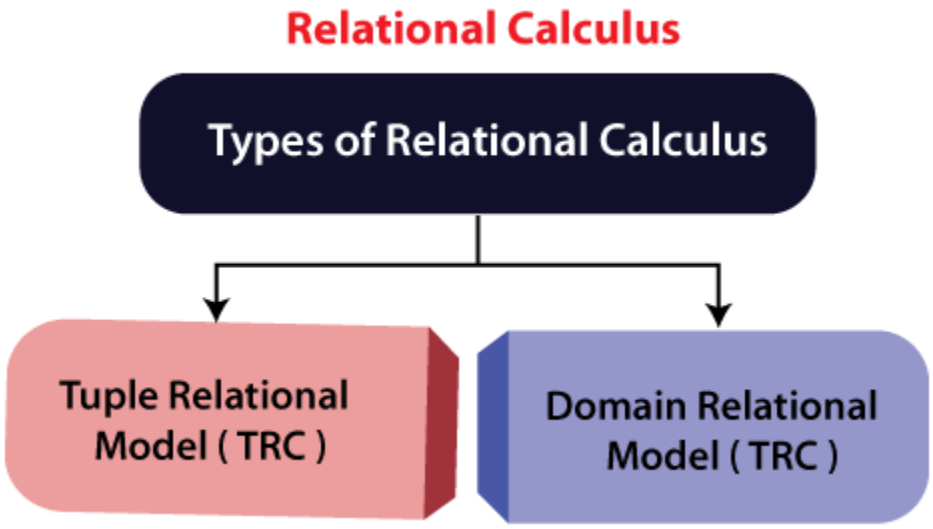
- **Universal Quantifiers:** The universal quantifier denoted by ∀ is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.
- **Existential Quantifiers:** The existential quantifier denoted by ∃ is read as for all which means that in a given set of tuples there is at least one occurrences whose value satisfy a given condition.

Before using the concept of quantifiers in formulas, we need to know the concept of Free and Bound Variables.

A tuple variable t is bound if it is quantified which means that if it appears in any occurrences a variable that is not bound is said to be free.

Free and bound variables may be compared with global and local variable of programming languages.

Types of Relational calculus:



1. Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples.

Notation:

A Query in the tuple relational calculus is expressed as following notation

1. {T | P (T)} or {T | Condition (T)}
   Where

   T is the resulting tuples

   P(T) is the condition used to fetch T.

   For example:

1. { T.name | Author(T) AND T.article = 'database' }
   Output: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

   TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

   For example:

1. { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}
   Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ¬ (not). It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

Notation:

1. { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}
   Where

a1, **a2** are attributes
P stands for formula built by inner attributes

For example:

1. {< article, page, subject > | ∈ javatpoint ∧ subject = 'database'}
   Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

Normalization

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

- Making relations very large.
- It isn't easy to maintain and update data as it would involve searching many records in relation.
- Wastage and poor utilization of disk space and resources.
- The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

- Normalization is the process of organizing the data in the database.

- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

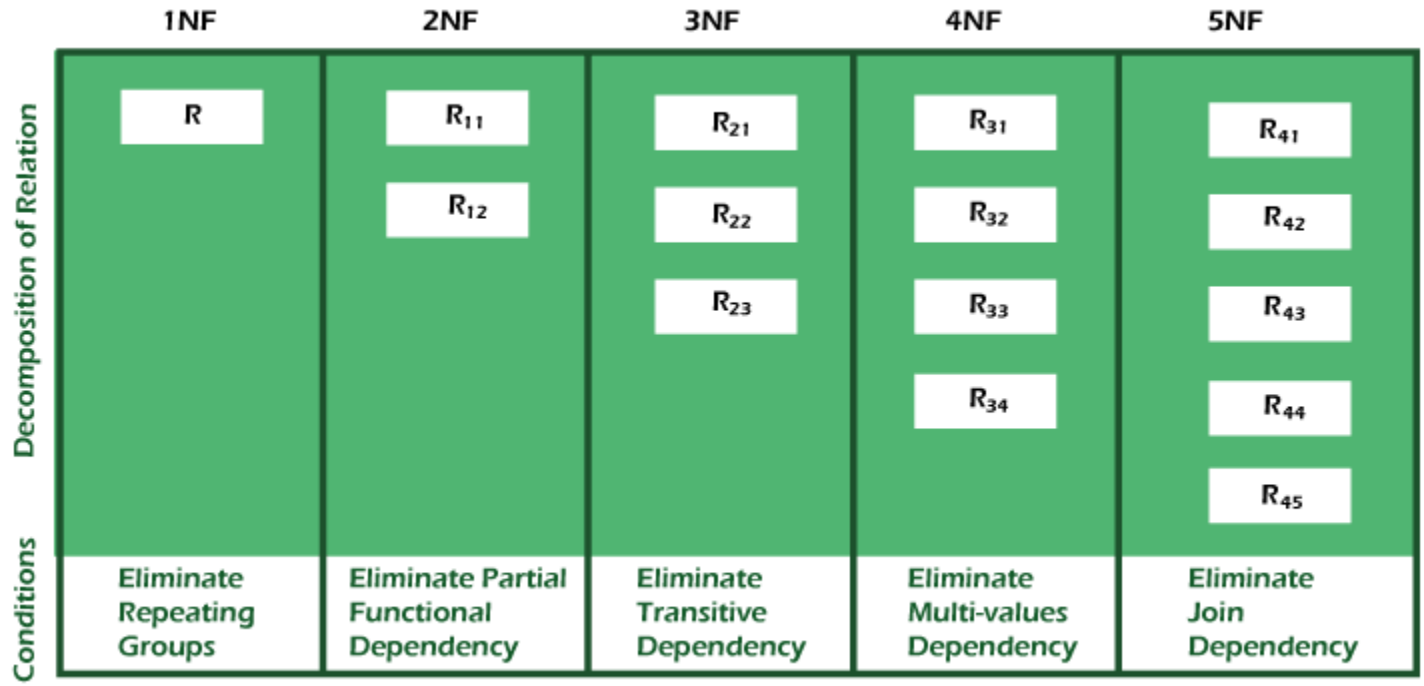
Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms:



Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Advantages of Normalization

- Normalization helps to minimize data redundancy.
- Greater overall database organization.
- Data consistency within the database.
- Much more flexible database design.
- Enforces the concept of relational integrity.

Disadvantages of Normalization

- You cannot start building the database before knowing what the user needs.
- The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
- It is very time-consuming and difficult to normalize relations of a higher degree.
- Careless decomposition may lead to a bad database design, leading to serious problems.

Query Processing in DBMS

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:

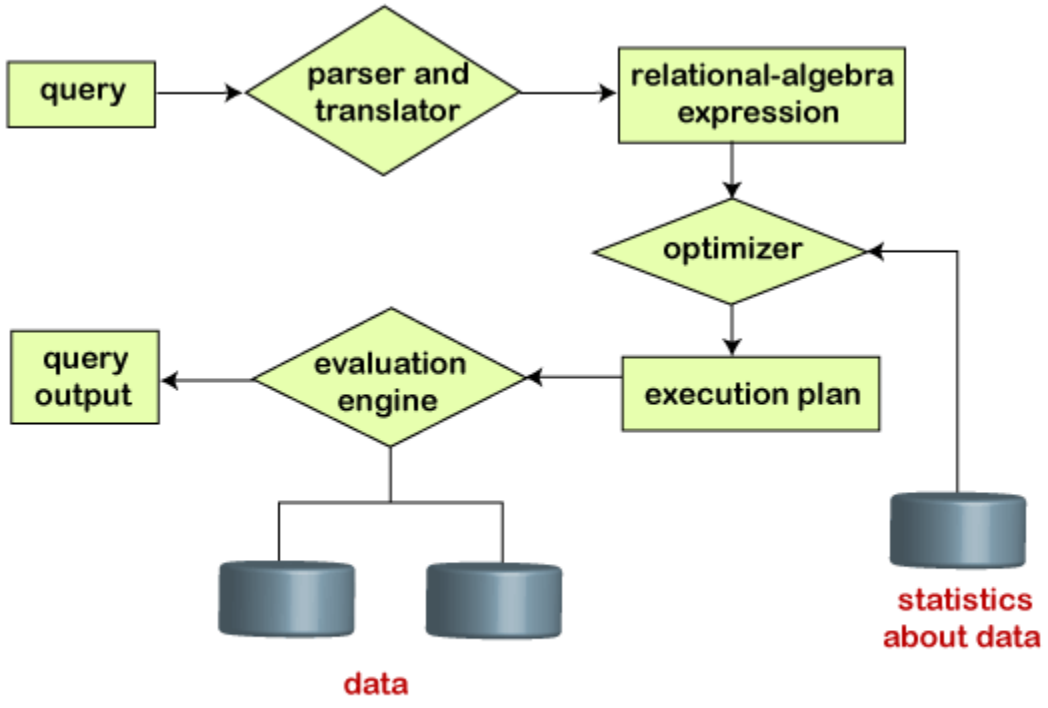
1. Parsing and translation
2. Optimization
3. Evaluation

The query processing works in the following way:

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:

- $\sigma_{salary>10000}$ **(**π_{salary} **(Employee))**
- π_{salary} **(**$\sigma_{salary>10000}$ **(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.

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.

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.

Finally, after selecting an evaluation plan, the system evaluates the query and produces the output of the query.

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

1. Open_Account(X)
2. Old_Balance = X.balance
3. New_Balance = Old_Balance - 800
4. X.balance = New_Balance
5. Close_Account(X)

Y's Account

1. Open_Account(Y)
2. Old_Balance = Y.balance
3. New_Balance = Old_Balance + 800
4. Y.balance = New_Balance
5. Close_Account(Y)

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

1. 1. R(X);
2. 2. X = X - 500;
3. 3. W(X);

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A) A:= A-100 Write(A)	Read(B) Y:= Y+100 Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600+300=900
2. Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.

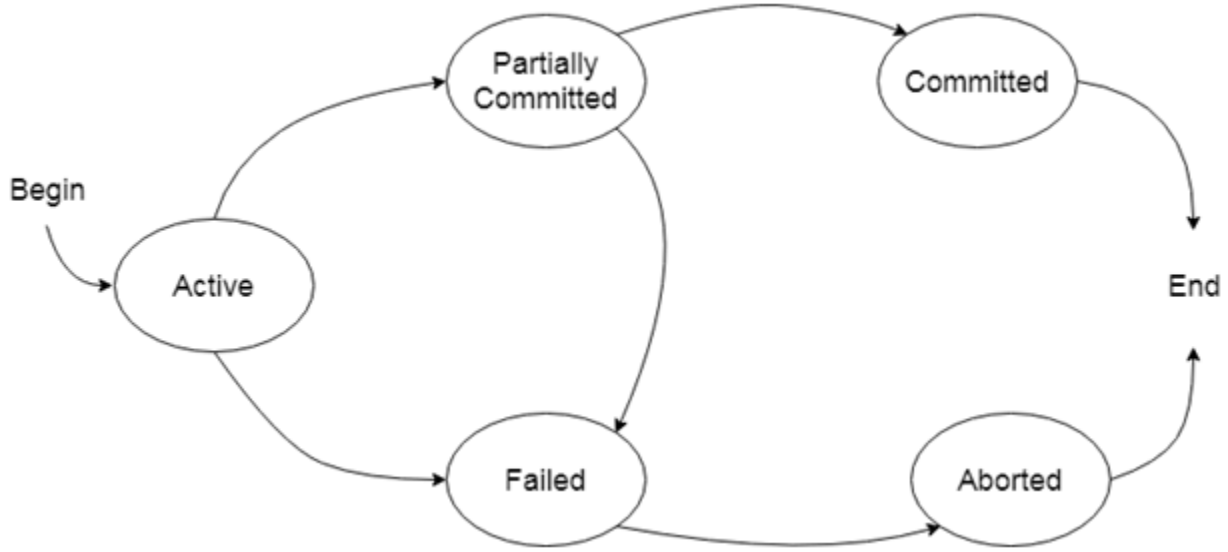
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states -



Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

A transaction is said to be in a committed state if it executes all its operations successfully. In this state, all the effects are now permanently saved on the database system.

Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    - Re-start the transaction
    - Kill the transaction

DBMS Concurrency Control

Concurrency Control is the management procedure that is required for controlling concurrent execution of the operations that take place on a database.

But before knowing about concurrency control, we should know about concurrent execution.

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.

- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Problems with Concurrent Execution

In a database transaction, the two main operations are **READ** and **WRITE** operations. So, there is a need to manage these two operations in the concurrent execution of the transactions as if these operations are not performed in an interleaved manner, and the data may become inconsistent. So, the following problems occur with the Concurrent Execution of the operations:

Problem 1: Lost Update Problems (W - W Conflict)

The problem occurs *when two different database transactions perform the read/write operations on the same database items in an interleaved manner (i.e., concurrent execution) that makes the values of the items incorrect hence making the database inconsistent*.

For example:

Consider the below diagram where two transactions T_X and T_Y, are performed on the same account A where the balance of account A is $300.

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A - 50	
t_3	—	READ (A)
t_4	—	A = A + 100
t_5	—	—
t_6	WRITE (A)	—
t_7		WRITE (A)

LOST UPDATE PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300 (only read).

- At time t2, transaction T_X deducts $50 from account A that becomes $250 (only deducted and not updated/write).
- Alternately, at time t3, transaction T_Y reads the value of account A that will be $300 only because T_X didn't update the value yet.
- At time t4, transaction T_Y adds $100 to account A that becomes $400 (only added but not updated/write).
- At time t6, transaction T_X writes the value of account A that will be updated as $250 only, as T_Y didn't update the value yet.
- Similarly, at time t7, transaction T_Y writes the values of account A, so it will write as done at time t4 that will be $400. It means the value written by T_X is lost, i.e., $250 is lost.

Hence data becomes incorrect, and database sets to inconsistent.

Dirty Read Problems (W-R Conflict)

The dirty read problem occurs *when one transaction updates an item of the database, and somehow the transaction fails, and before the data gets rollback, the updated database item is accessed by another transaction. There comes the Read-Write Conflict between both transactions.*

For example:

Consider two transactions T_x and T_y in the below diagram performing read/write operations on account A where the available balance in account A is $300:

Time	T_x	T_y
t_1	READ (A)	—
t_2	A = A + 50	—
t_3	WRITE (A)	—
t_4	—	READ (A)
t_5	SERVER DOWN ROLLBACK	—

DIRTY READ PROBLEM

- At time t1, transaction T_X reads the value of account A, i.e., $300.
- At time t2, transaction T_X adds $50 to account A that becomes $350.
- At time t3, transaction T_X writes the updated value in account A, i.e., $350.
- Then at time t4, transaction T_Y reads account A that will be read as $350.

- Then at time t5, transaction T_X rollbacks due to server problem, and the value changes back to $300 (as initially).
- But the value for account A remains $350 for transaction T_Y as committed, which is the dirty read and therefore known as the Dirty Read Problem.

Unrepeatable Read Problem (W-R Conflict)

Also known as Inconsistent Retrievals Problem that occurs when in a transaction, two different values are read for the same database item.

For example:

Consider two transactions, T_x and T_Y, performing the read/write operations on account A, having an available balance = $300. The diagram is shown below:

Time	T_x	T_y
t_1	READ (A)	—
t_2	—	READ (A)
t_3	—	A = A + 100
t_4	—	WRITE (A)
t_5	READ (A)	—

UNREPEATABLE READ PROBLEM

- At time t1, transaction T_X reads the value from account A, i.e., $300.
- At time t2, transaction T_Y reads the value from account A, i.e., $300.
- At time t3, transaction T_Y updates the value of account A by adding $100 to the available balance, and then it becomes $400.
- At time t4, transaction T_Y writes the updated value, i.e., $400.
- After that, at time t5, transaction T_X reads the available value of account A, and that will be read as $400.
- It means that within the same transaction T_X, it reads two different values of account A, i.e., $ 300 initially, and after updation made by transaction T_Y, it reads $400. It is an unrepeatable read and is therefore known as the Unrepeatable read problem.

Thus, in order to maintain consistency in the database and avoid such problems that take place in concurrent execution, management is needed, and that is where the concept of Concurrency Control comes into role.

Concurrency Control

Concurrency Control is the working concept that is required for controlling and managing the concurrent execution of database operations and thus avoiding the inconsistencies in the database. Thus, for maintaining the concurrency of the database, we have the concurrency control protocols.

Concurrency Control Protocols

The concurrency control protocols ensure the *atomicity, consistency, isolation, durability* and *serializability* of the concurrent execution of the database transactions. Therefore, these protocols are categorized as:

- Lock Based Concurrency Control Protocol
- Time Stamp Concurrency Control Protocol
- Validation Based Concurrency Control Protocol

We will understand and discuss each protocol one by one in our next sections.

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

There are four types of lock protocols available:

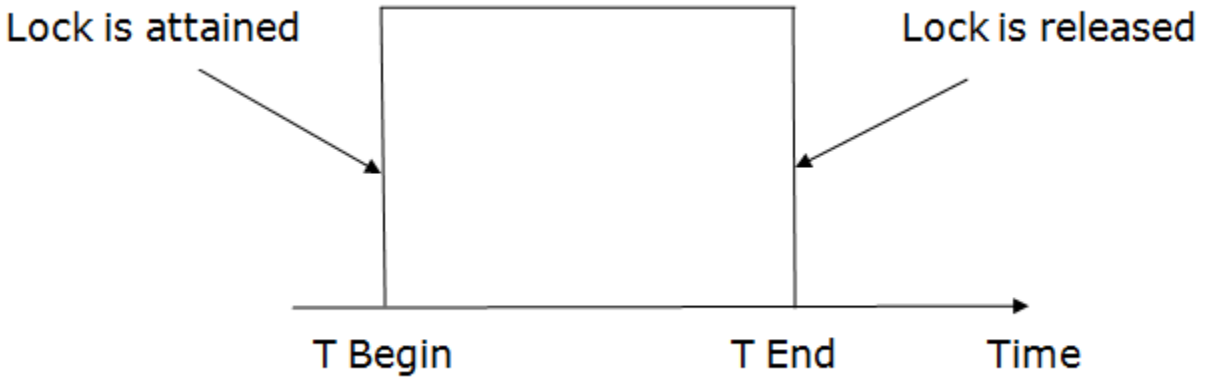
1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

2. Pre-claiming Lock Protocol

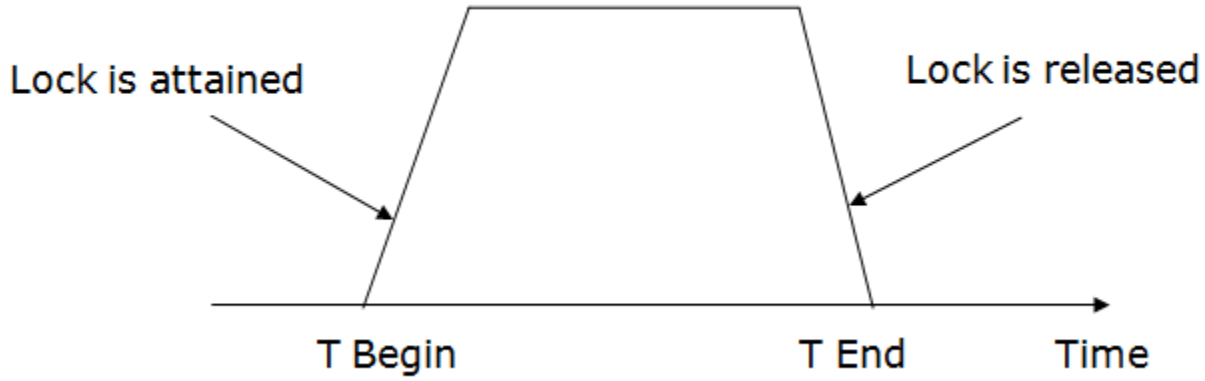
- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.

- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Example:

	T1	**T2**
0	LOCK-S(A)	
1		LOCK-S(A)
2	LOCK-X(B)	
3	——	——
4	UNLOCK(A)	
5		LOCK-X(C)
6	UNLOCK(B)	
7		UNLOCK(A)
8		UNLOCK(C)
9	——	——

The following way shows how unlocking and locking work with 2-PL.

Transaction T1:

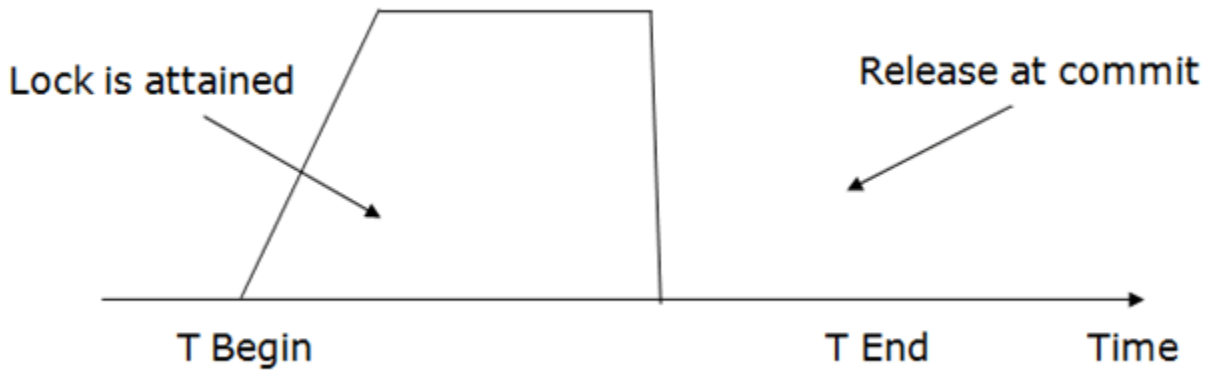
- **Growing phase:** from step 1-3
- **Shrinking phase:** from step 5-7
- **Lock point:** at 3

Transaction T2:

- **Growing phase:** from step 2-6
- **Shrinking phase:** from step 8-9
- **Lock point:** at 6

4. Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.



It does not have cascading abort as 2PL does.

Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.

- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

- If TS(Ti) < R_TS(X) then the operation is rejected.
- If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:

Image: Precedence Graph for TS ordering

- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade- free.

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

Start(Ti): It contains the time when Ti started its execution.

Validation (T_i): It contains the time when Ti finishes its read phase and starts its validation phase.

Finish(Ti): It contains the time when Ti finishes its write phase.

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.
- Thus it contains transactions which have less number of rollbacks.

Thomas write Rule

Thomas Write Rule provides the guarantee of serializability order for the protocol. It improves the Basic Timestamp Ordering Algorithm.

The basic Thomas write rules are as follows:

- If TS(T) < R_TS(X) then transaction T is aborted and rolled back, and operation is rejected.
- If TS(T) < W_TS(X) then don't execute the W_item(X) operation of the transaction and continue processing.
- If neither condition 1 nor condition 2 occurs, then allowed to execute the WRITE operation by transaction Ti and set W_TS(X) to TS(T).

If we use the Thomas write rule then some serializable schedule can be permitted that does not conflict serializable as illustrate by the schedule in a given figure:

T1	T2
R(A) W(A) Commit	 W(A) Commit

Figure: A Serializable Schedule that is not Conflict Serializable

In the above figure, T1's read and precedes T1's write of the same data item. This schedule does not conflict serializable.

Thomas write rule checks that T2's write is never seen by any transaction. If we delete the write operation in transaction T2, then conflict serializable schedule can be obtained which is shown in below figure.

T1	T2
R(A) W(A) Commit	Commit

Figure: A Conflict Serializable Schedule

A Distributed Database System is a kind of database that is present or divided in more than one location, which means it is not limited to any single computer system. It is divided over the network of various systems. The Distributed Database System is physically present on the different systems in different locations. This can be necessary when different users from all over the world need to access a specific database. For a user, it should be handled in such a way that it seems like a single database.

Parameters of Distributed Database Systems:

- **Distribution:**

It describes how data is physically distributed among the several sites.

- **Autonomy:**

It reveals the division of power inside the Database System and the degree of autonomy enjoyed by each individual DBMS.

- **Heterogeneity:**

It speaks of the similarity or differences between the databases, system parts, and data models.

Common Architecture Models of Distributed Database Systems:

- **Client-Server Architecture of DDBMS:**

This architecture is two level architecture where clients and servers are the points or levels where the main functionality is divided. There is various functionality provided by the server, like managing the transaction, managing the data, processing the queries, and optimization.

- **Peer-to-peer Architecture of DDBMS:**

In this architecture, each node or peer is considered as a server as well as a client, and it performs its database services as both (server and client). The peers coordinate their efforts and share their resources with one another.

- **Multi DBMS Architecture of DDBMS:**

This is an amalgam of two or more independent Database Systems that functions as a single integrated Database System.

Types of Distributed Database Systems:

- **Homogeneous Database System:**

Each site stores the same database in a Homogenous Database. Since each site has the same database stored, so all the data management schemes, operating system, and data structures will be the same across all sites. They are, therefore, simple to handle.

- **Heterogeneous Database System:**

In this type of Database System, different sites are used to store the data and relational tables, which makes it difficult for database administrators to do the transactions and run the queries into the database. Additionally, one site might not even be aware of the existence of the other sites. Different operating systems and database applications may be used by various computers. Since each system has its own database model to store the data, therefore it is

required there should be translation schemes to establish the connections between different sites to transfer the data.

Distributed Data Storage:

There are two methods by which we can store the data on different sites:

- **Replication:**

This method involves redundantly storing the full relationship at two or more locations. Since a complete database can be accessed from each site, it becomes a redundant database. Systems preserve copies of the data as a result of replication.

This has advantages because it makes more data accessible at many locations. Additionally, query requests can now be handled in parallel.

However, there are some drawbacks as well. Data must be updated frequently. Any changes performed at one site must be documented at every site where that relation is stored in order to avoid inconsistent results. There is a tonne of overhead here. Additionally, since concurrent access must now be monitored across several sites, concurrency control becomes far more complicated.

- **Fragmentation:**

According to this method, the relationships are divided (i.e., broken up into smaller pieces), and each fragment is stored at the many locations where it is needed. To ensure there is no data loss, the pieces must be created in a way that allows for the reconstruction of the original relation.

Since Fragmentation doesn't result in duplicate data, consistency is not a concern.

Ways of fragmentation:

- **Horizontal Fragmentation:**

In Horizontal Fragmentation, the relational table or schema is broken down into a group of one and more rows, and each row gets one fragment of the schema. It is also called **splitting by rows**.

- **Vertical Fragmentation:**

In this fragmentation, a relational table or schema is divided into some more schemas of smaller sizes. A common candidate key must be present in each fragment in order to guarantee a lossless join. This is also called **splitting by columns**.

Note: Most of the time, a hybrid approach of replication and fragmentation is used.

Application of Distributed Database Systems:

- Multimedia apps use it.
- The manufacturing control system also makes use of it.
- Another application is by corporate management for the information system.
- It is used in hotel chains, military command systems, etc.

Distributed databases' benefits

Using distributed databases has a lot of benefits.

- As distributed databases provide modular development, systems may be enlarged by putting new computers and local data in a new location and seamlessly connecting them to the distributed system.
- With centralized databases, failures result in a total shutdown of the system. Distributed database systems, however, continue to operate with lower performance when a component fails until the issue is resolved.
- If the data is near to where it is most often utilized, administrators can reduce transmission costs for distributed database systems. Centralized systems are unable to accommodate this<

Data Warehouse Tutorial

Data Warehouse is a relational database management system (RDBMS) construct to meet the requirement of transaction processing systems. It can be loosely described as any centralized data repository which can be queried for business benefits. It is a database that stores information oriented to satisfy decision-making requests. It is a group of decision support technologies, targets to enabling the knowledge worker (executive, manager, and analyst) to make superior and higher decisions. So, Data Warehousing support architectures and tool for business executives to systematically organize, understand and use their information to make strategic decisions.

Data Warehouse environment contains an extraction, transportation, and loading (ETL) solution, an online analytical processing (OLAP) engine, customer analysis tools, and other applications that handle the process of gathering information and delivering it to business users.

What is a Data Warehouse?

A Data Warehouse (DW) is a relational database that is designed for query and analysis rather than transaction processing. It includes historical data derived from transaction data from single and multiple sources.

A Data Warehouse provides integrated, enterprise-wide, historical data and focuses on providing support for decision-makers for data modeling and analysis.

A Data Warehouse is a group of data specific to the entire organization, not only to a particular group of users.

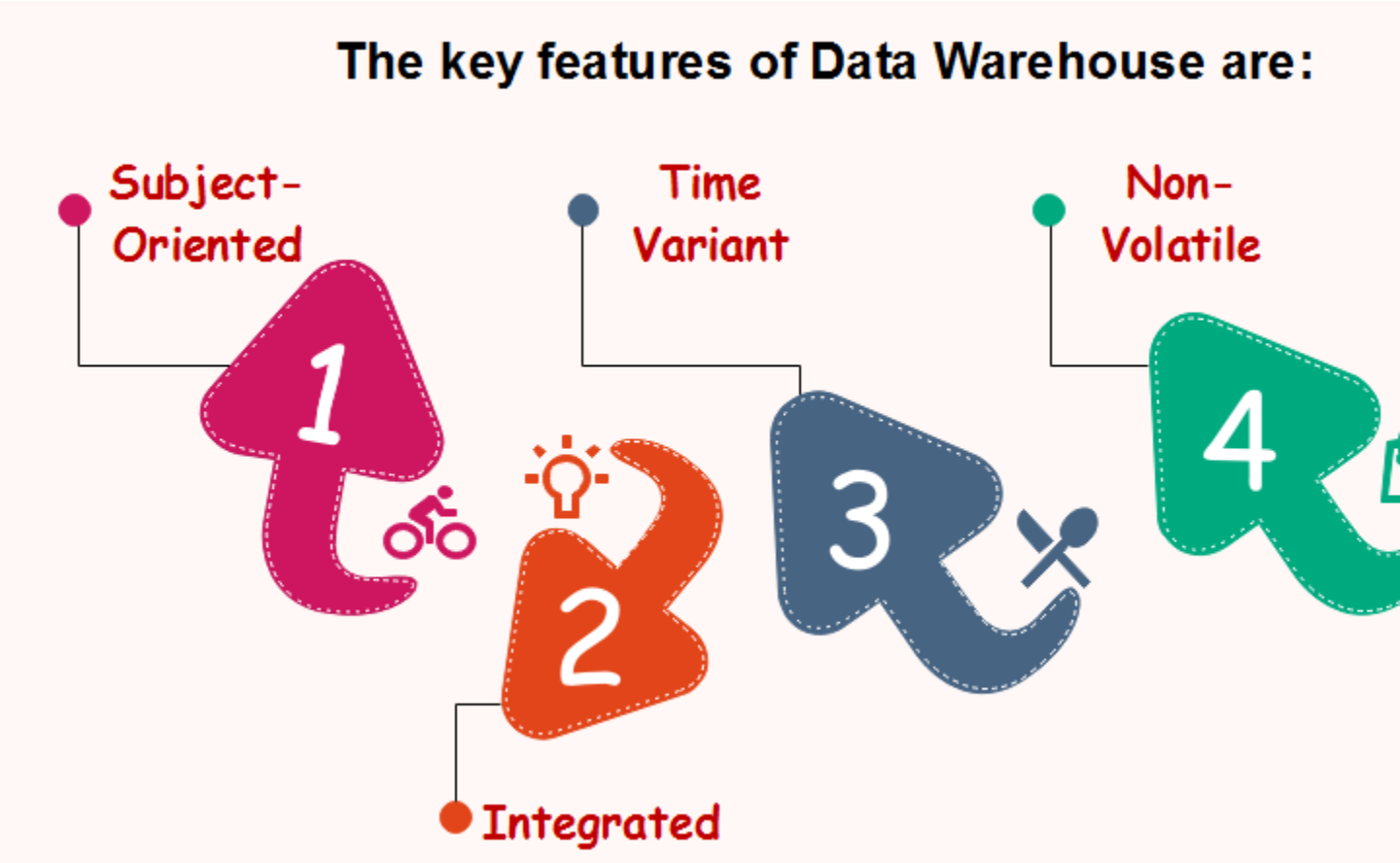
It is not used for daily operations and transaction processing but used for making decisions.

A Data Warehouse can be viewed as a data system with the following attributes:

- It is a database designed for investigative tasks, using data from various applications.
- It supports a relatively small number of clients with relatively long interactions.
- It includes current and historical data to provide a historical perspective of information.
- Its usage is read-intensive.
- It contains a few large tables.

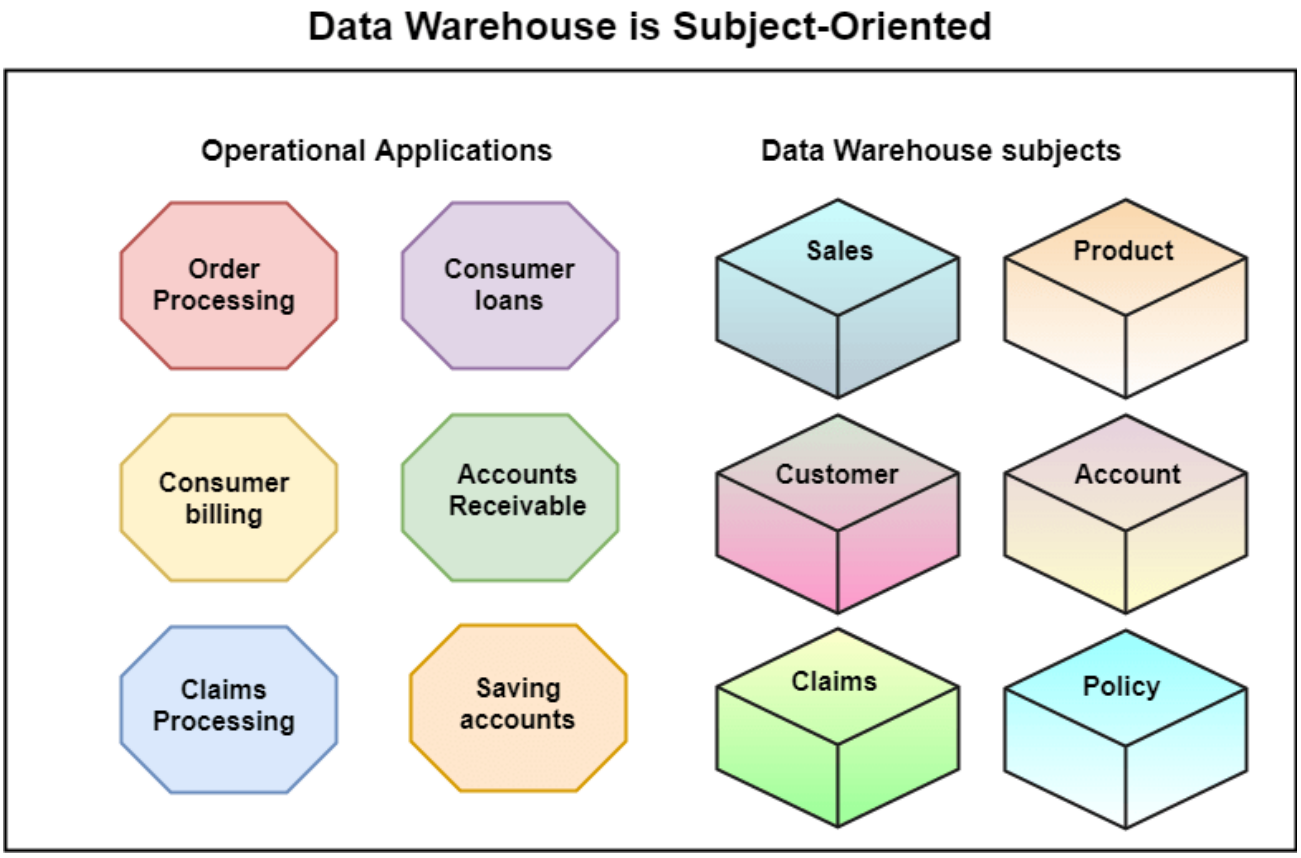
"Data Warehouse is a subject-oriented, integrated, and time-variant store of information in support of management's decisions."

Characteristics of Data Warehouse



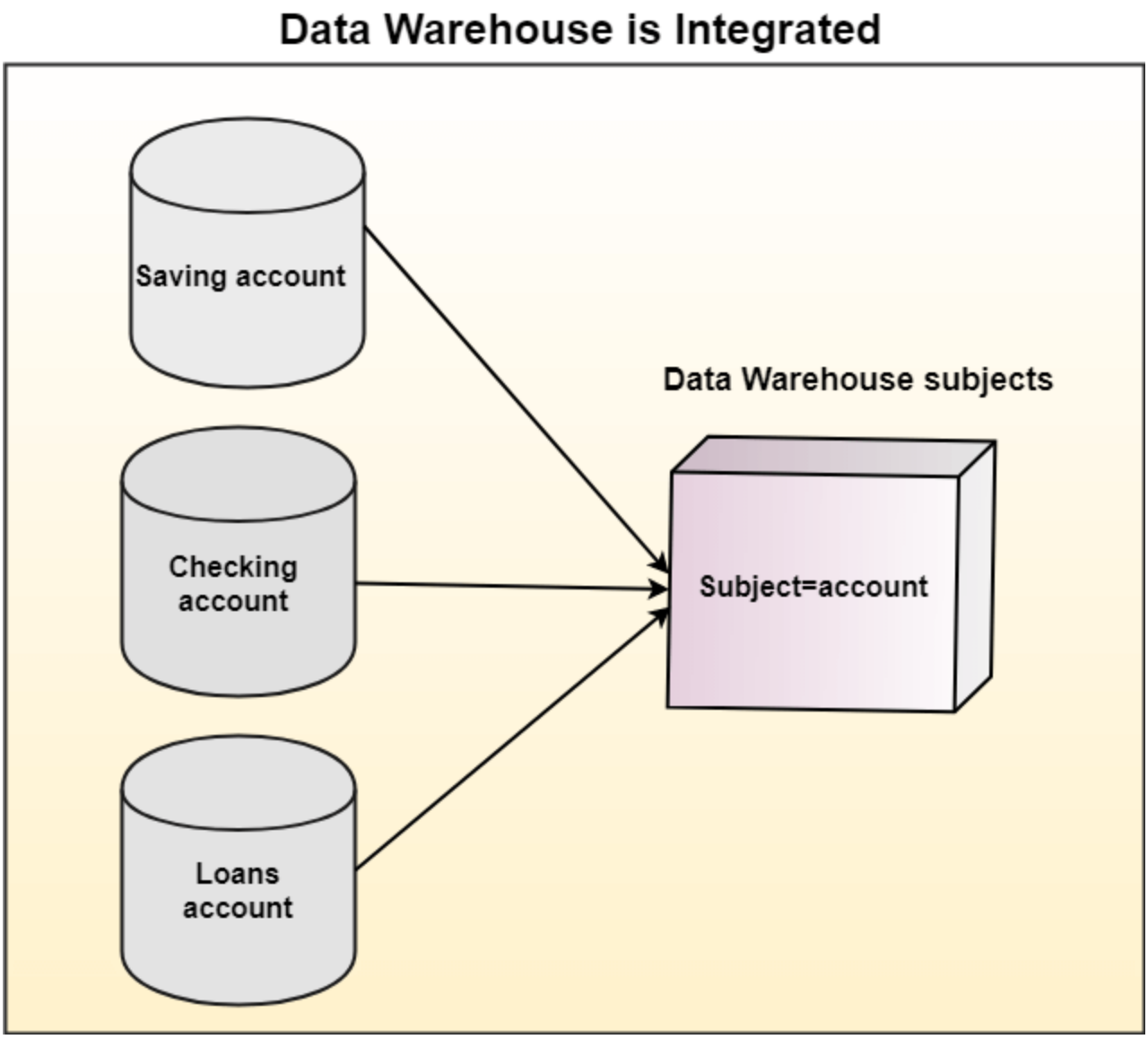
Subject-Oriented

A data warehouse target on the modeling and analysis of data for decision-makers. Therefore, data warehouses typically provide a concise and straightforward view around a particular subject, such as customer, product, or sales, instead of the global organization's ongoing operations. This is done by excluding data that are not useful concerning the subject and including all data needed by the users to understand the subject.



Integrated

A data warehouse integrates various heterogeneous data sources like RDBMS, flat files, and online transaction records. It requires performing data cleaning and integration during data warehousing to ensure consistency in naming conventions, attributes types, etc., among different data sources.

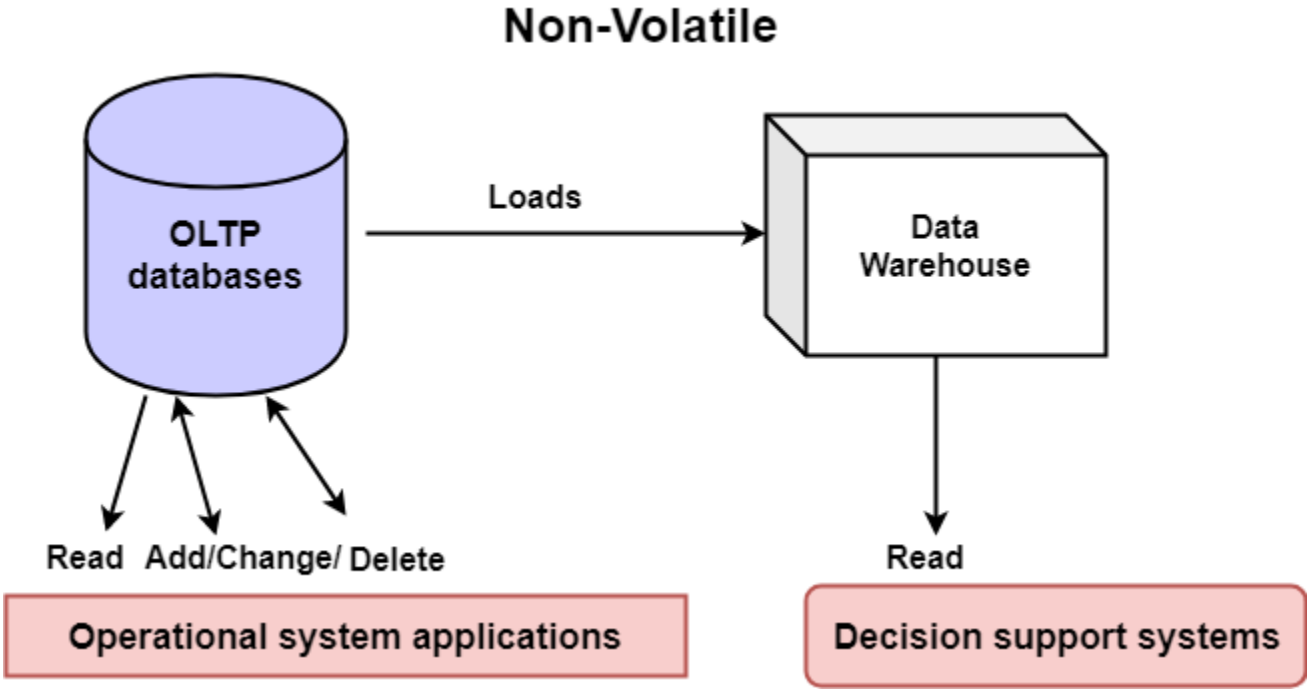


Time-Variant

Historical information is kept in a data warehouse. For example, one can retrieve files from 3 months, 6 months, 12 months, or even previous data from a data warehouse. These variations with a transactions system, where often only the most current file is kept.

Non-Volatile

The data warehouse is a physically separate data storage, which is transformed from the source operational RDBMS. The operational updates of data do not occur in the data warehouse, i.e., update, insert, and delete operations are not performed. It usually requires only two procedures in data accessing: Initial loading of data and access to data. Therefore, the DW does not require transaction processing, recovery, and concurrency capabilities, which allows for substantial speedup of data retrieval. Non-Volatile defines that once entered into the warehouse, and data should not change.



History of Data Warehouse

The idea of data warehousing came to the late 1980's when IBM researchers Barry Devlin and Paul Murphy established the "Business Data Warehouse."

In essence, the data warehousing idea was planned to support an architectural model for the flow of information from the operational system to decisional support environments. The concept attempt to address the various problems associated with the flow, mainly the high costs associated with it.

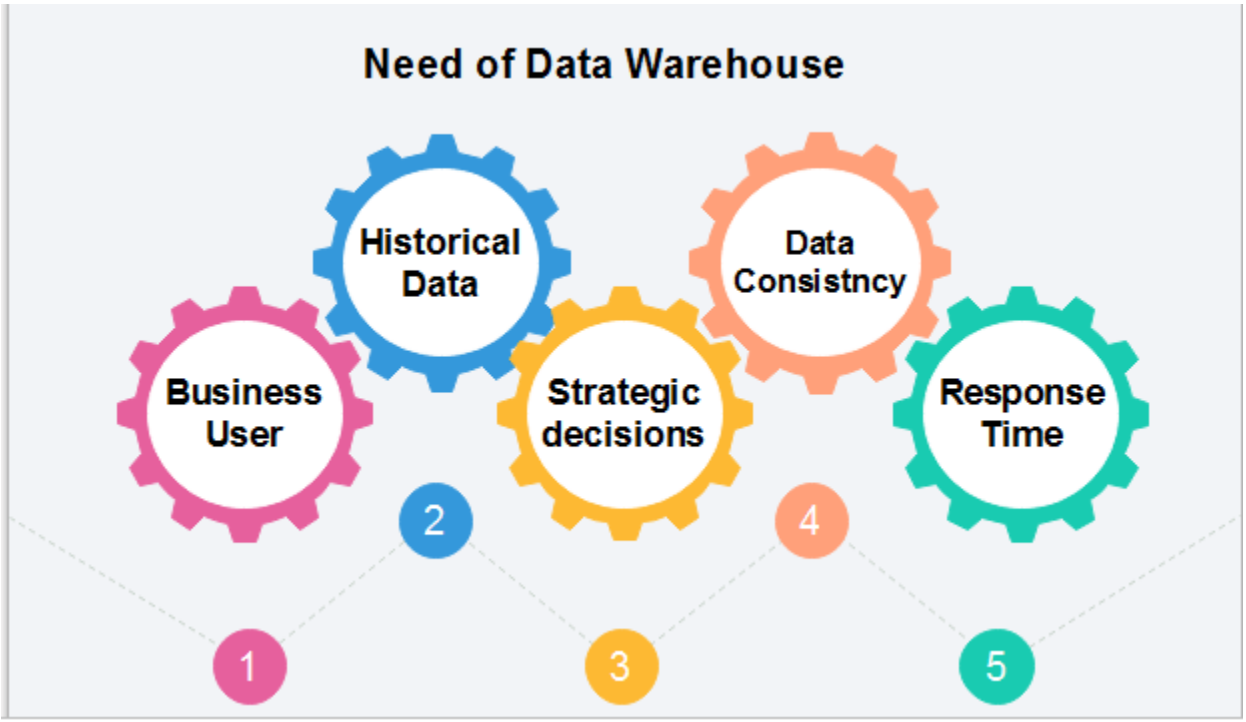
In the absence of data warehousing architecture, a vast amount of space was required to support multiple decision support environments. In large corporations, it was ordinary for various decision support environments to operate independently.

Goals of Data Warehousing

- To help reporting as well as analysis
- Maintain the organization's historical information
- Be the foundation for decision making.

Need for Data Warehouse

Data Warehouse is needed for the following reasons:



1. 1) **Business User:** Business users require a data warehouse to view summarized data from the past. Since these people are non-technical, the data may be presented to them in an elementary form.
2. 2) **Store historical data:** Data Warehouse is required to store the time variable data from the past. This input is made to be used for various purposes.
3. 3) **Make strategic decisions:** Some strategies may be depending upon the data in the data warehouse. So, data warehouse contributes to making strategic decisions.
4. 4) **For data consistency and quality:** Bringing the data from different sources at a commonplace, the user can effectively undertake to bring the uniformity and consistency in data.
5. 5) **High response time:** Data warehouse has to be ready for somewhat unexpected loads and types of queries, which demands a significant degree of flexibility and quick response time.

Benefits of Data Warehouse

1. Understand business trends and make better forecasting decisions.
2. Data Warehouses are designed to perform well enormous amounts of data.
3. The structure of data warehouses is more accessible for end-users to navigate, understand, and query.
4. Queries that would be complex in many normalized databases could be easier to build and maintain in data warehouses.
5. Data warehousing is an efficient method to manage demand for lots of information from lots of users.
6. Data warehousing provide the capabilities to analyze a large amount of historical data.