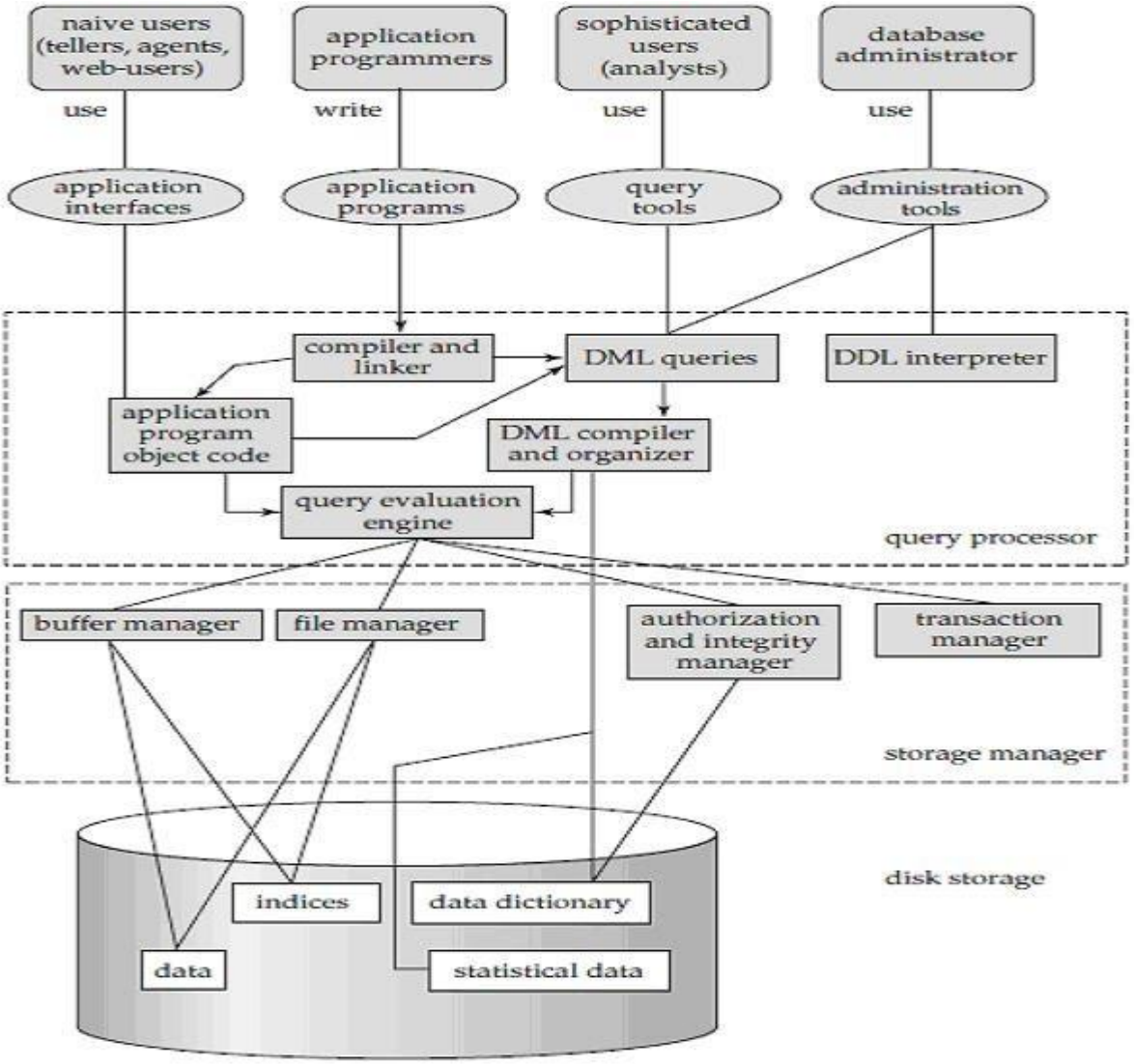
UNIT 3 …. Structure of Database Management System

Database Management System (DBMS) is software that allows access to data stored in a database and provides an easy and effective method of:-

- Defining the information.
- Storing the information.
- Manipulating the information.
- Protecting the information from system crashes or data theft.
- Differentiating access permissions for different users.

The Structure of Database Management System is also referred to as Overall **System Structure** or **Database Architecture** but it is different from the **tier architecture** of Database**.**
The database system is divided into three components: Query Processor, Storage Manager, and Disk Storage. These are explained as following below.



Architecture of DBMS

1. **Query Processor:**
   It interprets the requests (queries) received from end user via an application program into instructions. It also executes the user request which is received from the DML compiler.
   Query Processor contains the following components –

- **DML Compiler –**
  It processes the DML statements into low level instruction (machine language), so that they can be executed.

- **DDL Interpreter –**
  It processes the DDL statements into a set of table containing meta data (data about data).

- **Query evaluation Engine–**
  It executes the instruction generated by DML Compiler.

2. **Storage Manager:**
   Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executes the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database.

   It contains the following components –

- **Authorization Manager –**
  It ensures role-based access control, i.e., checks whether the particular person is privileged to perform the requested operation or not.

- **Integrity Manager –**
  It checks the integrity constraints when the database is modified.

- **Transaction Manager –**
  It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.

- **File Manager –**
  It manages the file space and the data structure used to represent information in the database.

- **Buffer Manager –**
  It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

3. **Disk Storage:** It contains the following components –

- **Data Files –**
  It stores the data.

- **Data Dictionary –**
  It contains the information about the structure of any database object. It is the repository of information that governs the metadata.

- **Indices** –

  The index is a type of data structure. It is used to locate and access the data in a database table quickly. **Index** records comprise search-key values and data pointers. Multilevel **index** is stored on the disk along with the actual database files.

What Does Database Manager (DB Manager) Mean?
Data managers or database managers are used to manage all access to the database, since their purpose is to serve as an interface between the database, the user and the applications.
Database managers have several capabilities including the ability to back up and restore, attach and detach, create, clone, delete and rename the databases.

Components of database manager

- **Authorization control:** This module checks in which the user has required authorization to carry out the essential function. (User want to access the database then db manager check that is authorize or not)
- **Command Processor**: Once the system has checked in which the user has influence to carry out the operation, control is passed to the command processor that changes commands to a logical series of steps.
- **Integrity checker:** For an operation in which vary the database, the integrity checker checks in which the requested operation satisfies all necessary integrity constraints like as key constraints.
- **Query Optimizer:** This module displays an optimal strategy for the query implementation.
- **Transaction Manager**: This module performs the essential processing of operations of various transactions. The transaction manager remains the tables of authorization Concurrency. The DBMS might use authorization tables to permit the transaction manager to make sure in which the user has permission to implement the desired operation on the database. The authorization tables could only be modified through properly authorized user commands that are themselves checked besides the authorization tables.
- **Scheduler:** This module is liable for ensuring in which transactions or concurrent operations on the database proceed without conflicting along with one another. It manages the associative order in that transaction operations are implemented. A database may also support concurrency control tables to prevent conflicts while concurrent, conflicting commands are implemented. The DBMS checks the concurrency control tables before implementing an operation to make sure that the data used through it is not locked by another statement.
- **Recovery Manager:** This module makes sure which the database remains in a constant state in the presence of failures. It is responsible for abort or transaction commit in which is failure or success of transaction.
- **Buffer Manager:** This module is responsible for the transmission of data between major memory and secondary storage, such as tape and disk. The buffer manager and the recovery manager are sometimes collectively referred to as data manager. The buffer manager is sometimes called as cache manager.

Responsibilities of Database manager

1. **Interaction with file manager: -** DB manager translate DML statement to low level (machine language) system command. Data storing, retrieving data and updating the database all are done by database manager.
2. **Integrity enforcement:-When** DBA creates any rules then those rules are followed or not these all are managed by database manager.
3. **Security Enforcement:-** Provide security
4. **Backup & Recovery:-**
5. **Concurrency Control: -** Concurrency control concept comes under the Transaction in database management system (DBMS). It is **a procedure in DBMS which helps us for the management of**

two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems.

Introduction to Database Administrator

A database administrator is defined as, the database is used to store and run a huge amount of data and the database administration is the set of activities performed by the database administrator to make sure the database is always available when needed, the main task of a database administrator is to maintain the complexity of the database, fundamentally a database administrator has to take care of a database, a database administrator is a person who can perform the roles like taking care of the database, managing it, maintaining the successful environment by performing all related pursuit using generalized software to keep the data secure.

Database administrator overview

The database administrator is also known as a DBAs so that in an organization the role of DB can be performed by the person, the responsibility of the database administrator is to create and design the database, the requirements are also analyzed and monitored by them, the troubleshooting skills are required, and also to make sure the data should be secure, the issues of the database are also handled by DBAs.

Different Types of Database Users

1. Database Administrator
3. Naive Users
4. Application Programmers
5. Sophisticated Users
6. Specialized Users

1. Database Administrator (Super Users)
A single person or a team of members can be a database administrator.

An administrator has full control of the database. Their account is called a superuser account.

An administrator defines the logical and physical schemas and manages all three levels of the database.

Even they can control view level schemas as well.

They grant/revoke authorization permission to all other users.

They design the overall structure of the database including, layouts, functioning, procedures, and motives.

They are responsible for routine maintenance, backup, and recovery of the database.

They perform all admin related activities like time to time updating, insert new required values /functions,

modify the existing, etc.

They provide technical support or arrange the same.

They control these all operations as well:

Security, integrity, redundancy, concurrency, hardware and software management.

2. Naive Users (End Users)

These are the actual users who use the database to fill the information, but most of the naive users have no knowledge of a database and how to operate it. They just use the view level with the help of interface methods provided.

These are again dived into two categories:

Parametric Users

These types of users just use predefined programs like booking online tickets, filling online forms, applying for online facilities, etc.

Casual Users

All parametric users are also casual users, but these types of users can use basic programming to fill the data in the database. They are provided with little training or user guide to have a little knowledge about the database technology.

3. Application Programmer (Backend Users)

These are the people who do all programming to make the database a real-world entity.
We can classify these in below three categories:

Designers

These are the first contact person if you want a database to be created. They try to understand your requirements related to a database like layout, looks, functioning, programming, costing, technologies or techniques, etc. After getting all the information they design the final layout for programmers to code it. Also, they create the structure of a database like tables, relationships, procedures, constraints, views, communication, etc.

System Analysts

These are the persons who check current similar options and do the needful to change or update the final layout to make it unique. Also, they check and gather all the information related to resources. They make sure that the buyer should be satisfied with the final product.

Programmers

These are the professional persons who finally do the coding to make the final layout a real-world entity. The programming is subdivided into 2 categories:

- **With tools programming:** Use available tools to make development faster.
- **Without tools programming:** Do all the programming required themselves.

4. Sophisticated Users

These users have knowledge of DDL, DML and use these both to write down queries to make their own databases or access the current database. These users are like a little techno savvy person or are provided with the complete training to do the needful. They can be engineers, analysts, scientists of the same organization, or others.

Roles and Responsibilities of DBA (Database Administrator)

1. Software installation and Maintenance

A DBA is frequently involved in the initial installation and configuration of a new Oracle, SQL Server, or other databases. The system administrator configures the database server's hardware and implements the operating system, after which the DBA installs and configures the database software. The DBA is in charge of ongoing maintenance, such as updates and patches.

In addition, if a new server is implemented, the DBA is in charge of transferring data from the existing system to the new platform.

2. Managing Data Integrity

DBAs primarily handle the overall integrity of database. They make sure that the Data integrity is carefully managed because it protects data from unauthorized use.

3. Takes Care of Data Extraction, Transformation, and Loading

DBAs are responsible for Data extraction, transformation, and loading, also known as (ETL), which refers to the efficient import of large amount of data extracted from multiple systems into a data warehouse environment. The external data is cleaned and transformed to fit the required format before being imported into a central repository.

4. Monitoring Performance

Only implementing a database is not the task of the database administrator. Once the database is implemented, they are required to monitor databases for performance issues. If a system component slows down processing, the DBA may need to change the software configuration or add more hardware capacity. There are numerous monitoring tools available, and understanding what they need to track to improve the system is part of the DBA's job.

5. Data Handling

Each company's success today revolves around massive databases. Companies nowadays maintain massive databases containing unstructured data types such as images, documents, or sound and video files. Managing an extensive database (VLDB) may necessitate higher-level skills, as well as additional monitoring and tuning, which a DBA possesses.

6. Decides Data Recovery and Back up method

If any company is having a big database, then it is likely to happen that database may fail at any instance. It is require that a DBA takes backup of entire database in regular time span. DBA has to decide that how much data should be backed up and how frequently the back should be taken. Also the recovery of data base is done by DBA if they have lost the database.

7. Database Security

One of the most critical responsibilities of a DBA is identifying and correcting any flaws in the database software. No system is entirely secure; however, DBAs mitigate risks by implementing best practices. A DBA must be able to identify potential flaws in the database software and the overall system of the company and take appropriate steps to mitigate risks.

8. Database Integrity

DBAs are primarily responsible for the overall integrity of a company's database. This includes putting the database in place, keeping it safe from loss and corruption, making it easily accessible, ensuring it works properly, and constantly tweaking it for ease of use and maximum productivity. In addition, the database administrator is also in charge of training eligible employees on how to access and use the database so that they can perform their duties.

9. Database Accessibility

Setting up employee access is a critical component of database security. DBAs decide who has access and what kind of access they have. They create a subschema to control database accessibility.

They also determine which users will have access to the database and which users will use data. Without the permission of the DBA, no user has the authority to access the database.

10. Provides Support to Users

If a user requires assistance at any time, it is the DBA's responsibility to assist him. The DBA provides complete support to users who are new to the database.

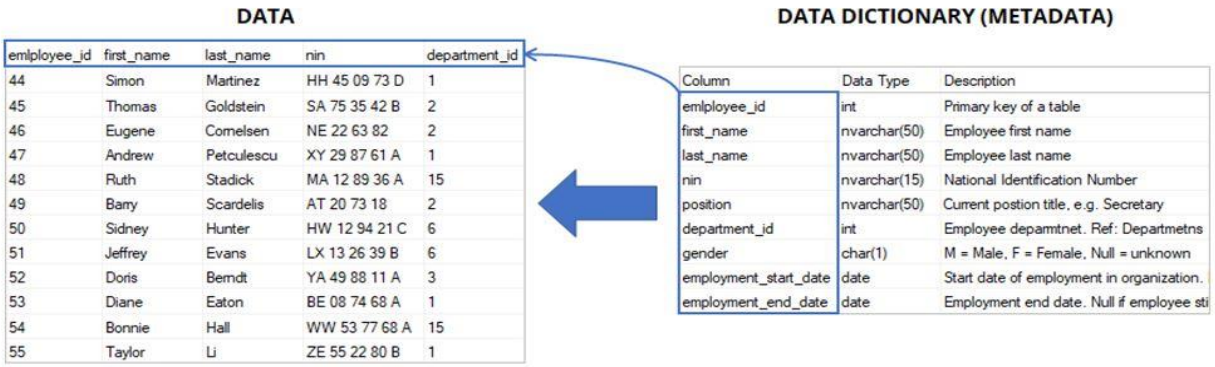
What is Data Dictionary

Data dictionary contains metadata i.e. data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary; it is only handled by the database administrators.

The data dictionary in general contains information about the following −

- Names of all the database tables and their schemas.
- Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
- Physical information about the tables such as where they are stored and how.
- Table constraints such as primary key attributes, foreign key information etc.
- Information about the database views those are visible.

This is a data dictionary describing a table that contains employee details.



emlployee_id	first_name	last_name	nin	department_id
44	Simon	Martinez	HH 45 09 73 D	1
45	Thomas	Goldstein	SA 75 35 42 B	2
46	Eugene	Cornelsen	NE 22 63 82	2
47	Andrew	Petculescu	XY 29 87 61 A	1
48	Ruth	Stadick	MA 12 89 36 A	15
49	Barry	Scardelis	AT 20 73 18	2
50	Sidney	Hunter	HW 12 94 21 C	6
51	Jeffrey	Evans	LX 13 26 39 B	6
52	Doris	Berndt	YA 49 88 11 A	3
53	Diane	Eaton	BE 08 74 68 A	1
54	Bonnie	Hall	WW 53 77 68 A	15
55	Taylor	Li	ZE 55 22 80 B	1

Column	Data Type	Description
emlployee_id	int	Primary key of a table
first_name	nvarchar(50)	Employee first name
last_name	nvarchar(50)	Employee last name
nin	nvarchar(15)	National Identification Number
position	nvarchar(50)	Current postion title, e.g. Secretary
department_id	int	Employee deparmtnet. Ref: Departmetns
gender	char(1)	M = Male, F = Female, Null = unknown
employment_start_date	date	Start date of employment in organization.
employment_end_date	date	Employment end date. Null if employee sti

Types of Data Dictionary

Here are the two types of data dictionary −

Active Data Dictionary

The DBMS software manages the active data dictionary automatically. The modification is an automatic task and most RDBMS has active data dictionary. It is also known as integrated data dictionary.

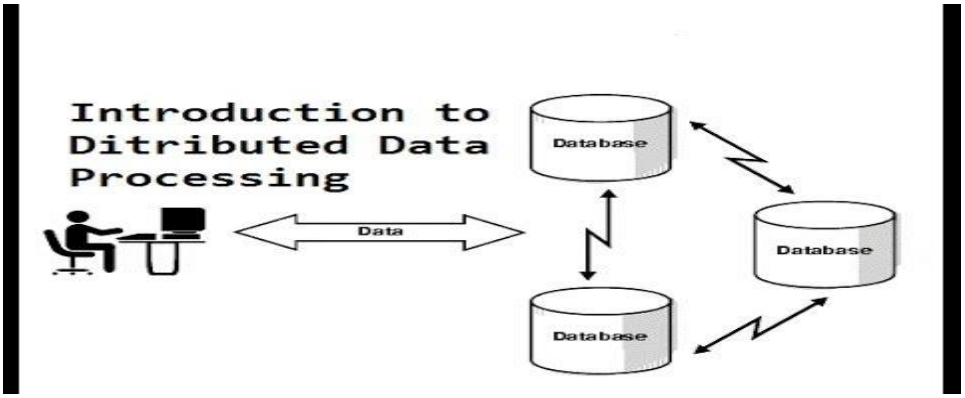
Passive Data Dictionary

Managed by the users and is modified manually when the database structure change. Also known as non-integrated data dictionary.

Distributed Processing

In distributed processing, a database's logical processing is shared among two or more physically independent sites that are connected through a network. For example, the data input/output (I/O), data selection, and data validation might be performed on one computer, and a report based on that data might be created on another computer. The distributed processing system uses only a single-site database.

Having different database file located at different sites in a network is known as distributed data processing.



Advantages

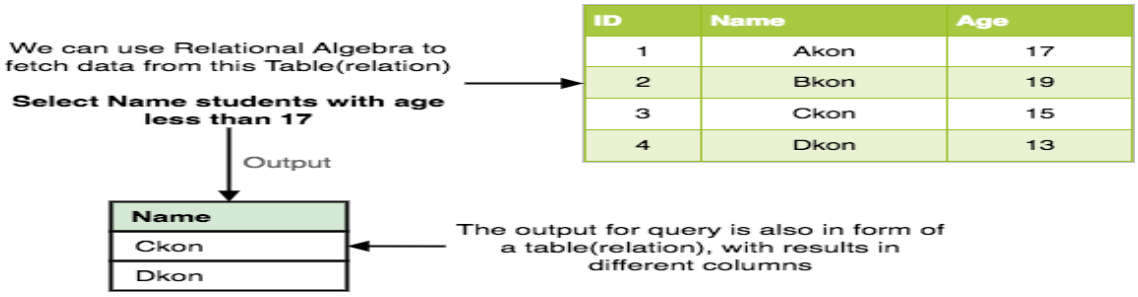
1. User want interface with each other while accessing the data. (When user is going to visit access or manipulate any data then he will get the data from its nearest location. In such way that multiple user not interferes with each other.)
2. Speed process (File will be accessed from nearest location then speed will be enhanced)
3. If one site fails, the system still can run.
4. Reduce costs (shared desktop performance instead of mainframe)
5. Ability to share data across multiple servers.

Disadvantages

1. Synchronize multiple database file takes more time (result is slow processing)
2. Data replication (Same data will be place in different database file)

Relational algebra
Relational algebra is a **procedural** query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, and delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.
On the other hand **relational calculus** is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it. We will discuss relational calculus in a separate tutorial.

Relational Algebra is a procedural query language used to query the database tables to access data in different ways.
In relational algebra, input is a relation (table from which data has to be accessed) and output is also a relation (a temporary table holding the data asked for by the user).



ID	Name	Age
1	Akon	17
2	Bkon	19
3	Ckon	15
4	Dkon	13

Relational Algebra works on the whole table at once, so we do not have to use loops etc to iterate over all the rows (tuples) of data one by one. All we have to do is specify the table name from which we need the data, and in a single line of command, relational algebra will traverse the entire given table to fetch data for you.

Basic SQL Relational Algebra Operations

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations from Set Theory

- UNION (υ)
- INTERSECTION (∩),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

Unary Relational Operations

PROJECT (symbol: π)

Projection is used to project required column data from a relation. It is **used to select desired columns (or attributes) from a table (or relation)**. Project operator in relational algebra is similar to the Select statement in SQL.

Example:

R

(A B C)

---

1 2 4
2 2 3
3 2 3
4 3 4

π (BC)

B C

---

2 4

2 3

3 4

Note: By Default projection removes duplicate data.

Selection (σ)

Selection is used to select required tuples of the relations.

For the above relation

σ (c>3)R will select the tuples which have c more than 3.

Note: selection operator only selects the required tuples but does not display them. For displaying, data projection operator is used.

For the above selected tuples, to display we need to use projection also.

π (σ (c>3)R) will show following tuples.

A B C

---

1 2 4
4 3 4

Rename (ρ)

Rename is a unary operation used for renaming attributes of a relation. ρ (a/b)R will rename the attribute 'b' of relation by 'a'.

Relational Algebra Operations from Set Theory

Union (U)
Union operation in relational algebra is same as union operation in set theory, only constraint is for union of two relation both relation must have same set of Attributes.

Intersection
An intersection is defined by the symbol ∩
A ∩ B
Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.

Visual Definition of Intersection

Example:
A ∩ B

Table A ∩ B	
column 1	**column 2**
1	1

Set Difference (-)
Set Difference in relational algebra is same set difference operation as in set theory with the constraint that both relation should have same set of attributes.

Cross Product (X)
Cross product between two relations let say A and B, so cross product between A X B will results all the attributes of A followed by each attribute of B. Each record of A will pairs with every record of B. Below is the example

```
      A                        B
  (Name   Age Sex)        (Id   Course)
  ------------------      -------------
  Ram    14   M            1    DS
  Sona   15   F            2    DBMS
  kim    20   M
```

A X B

Name	Age	Sex	Id	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Note: if A has ‘n’ tuples and B has ‘m’ tuples then A X B will have ‘n*m’ tuples.

Binary Relational Operations

Join

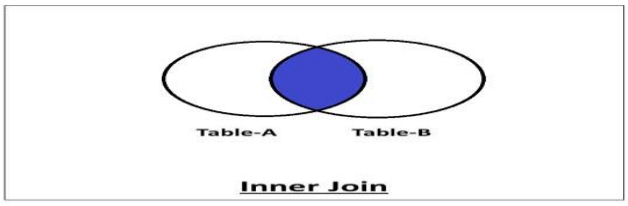
In DBMS, a join statement is mainly used to combine two tables based on a specified common field between them. If we talk in terms of Relational algebra, it is the cartesian product of two tables followed by the selection operation. Thus, we can execute the product and selection process on two tables using a single join statement.

A Join can be broadly divided into two types:

1. **Inner Join**
2. **Outer Join**

Inner Join

Inner Join is a join that can be used to return all the values that have matching values in both the tables. Inner Join can be depicted using the below diagram.



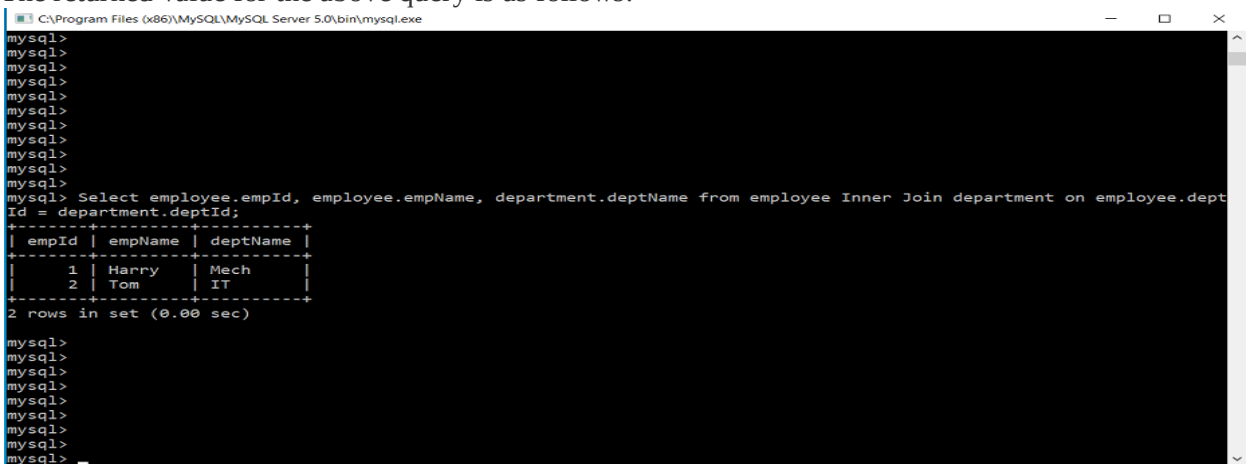
The inner join can be further divided into the following types:

1. **Equi Join**
2. **Natural Join**

Now let us learn about these inner joins one-by-one.

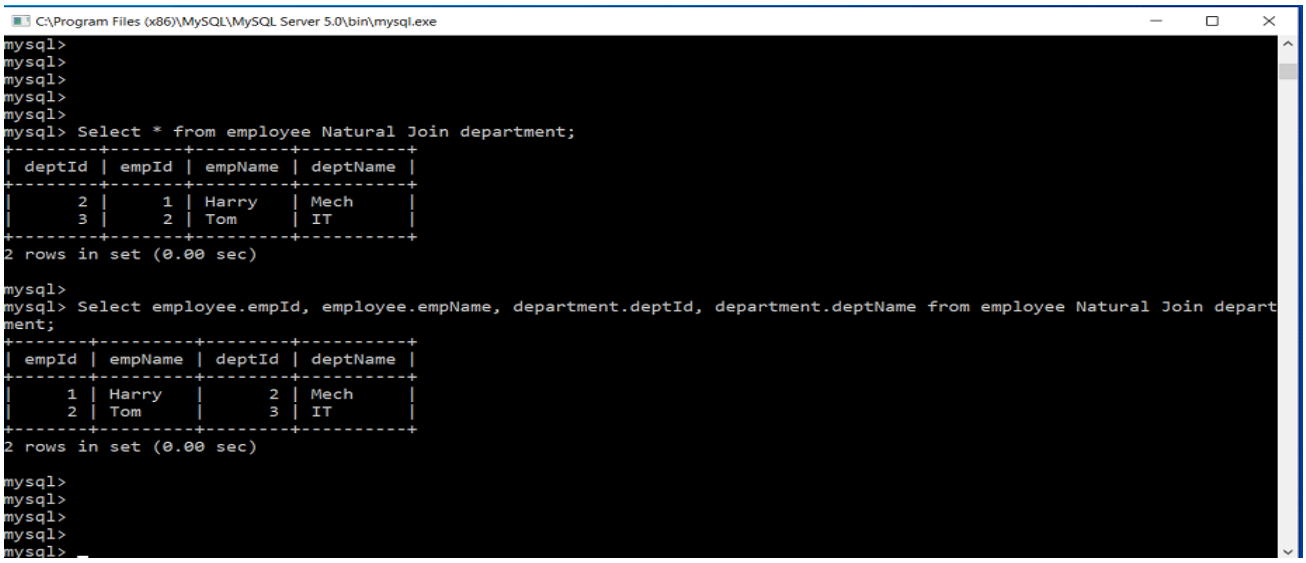
1. **Equi Join**
2.

Equi Join is an inner join that uses the equivalence condition for fetching the values of two tables.
The MySQL query for equi join can be as follows:
Select employee.empId, employee.empName, department.deptName from employee Inner Join department on employee.deptId = department.deptId;
The returned value for the above query is as follows:



2. Natural Join

Natural Join is an inner join that returns the values of the two tables on the basis of a common attribute that has the same name and domain. It does not use any comparison operator. It also removes the duplicate attribute from the results.
The MySQL query for natural join can be as follows:
Select * from employee Natural Join department;
The above query will return the values of tables removing the duplicates. If we want to specify the attribute names, the query will be as follows:
Select employee.empId, employee.empName, department.deptId, department.deptName from employee Natural Join department;
The returned values for the above two queries are as follows:



Outer Join

Outer Join is a join that can be used to return the records in both the tables whether it has matching records in both the tables or not.

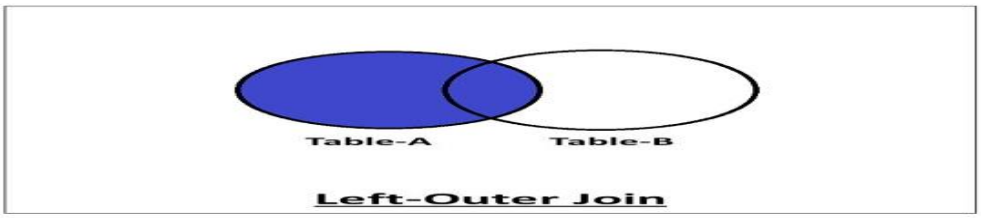
The outer join can be further divided into three types:

1. **Left-Outer Join**
2. **Right-Outer Join**
3. **Full-Outer Join**

We'll learn about these outer joins one-by-one.

1. Left-Outer Join:

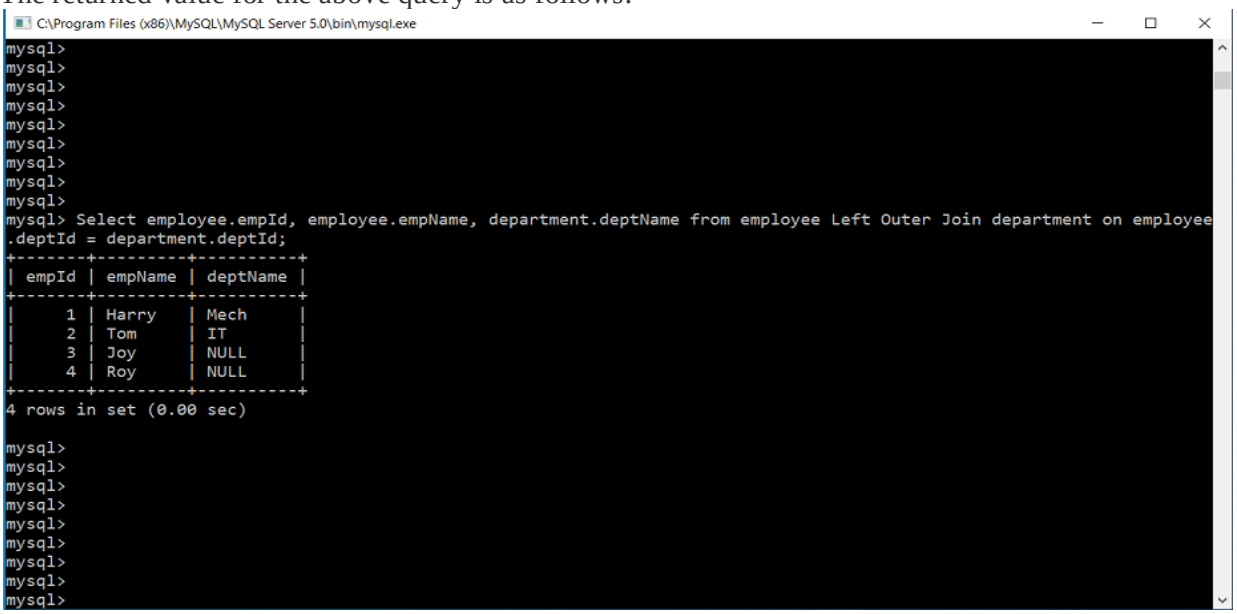
The Left-Outer Join is an outer join that returns all the values of the left table, and the values of the right table that has matching values in the left table. If there is no matching result in the right table, it will return null values in that field. The Left-Outer Join can be depicted using the below diagram.



The MySQL query for left-outer join can be as follows:

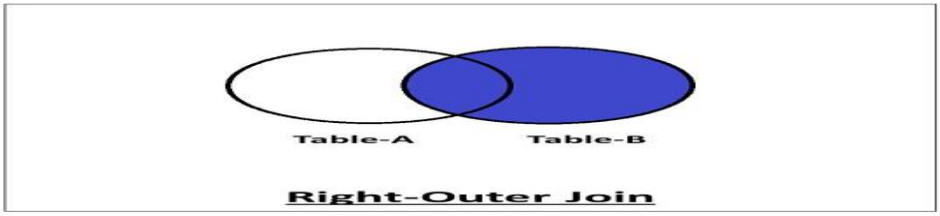
- Select employee.empId, employee.empName, department.deptName from employee Left Outer Join department on employee.deptId = department.deptId;

The returned value for the above query is as follows:



2. Right-Outer Join:

The Right-Outer Join is an outer join that returns all the values of the right table, and the values of the left table that has matching values in the right table. The Right-Outer Join can be depicted using the below diagram.



The MySQL query for right-outer join can be as follows:

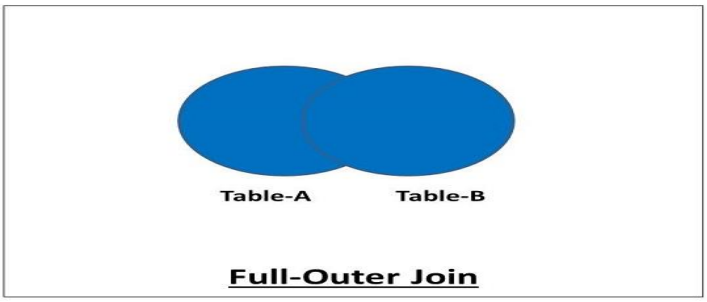
```
Select employee.empId, employee.empName, department.deptName from employee Right Outer Join department on employee.deptId = department.deptId;
```

The returned values for the above query is as follows:

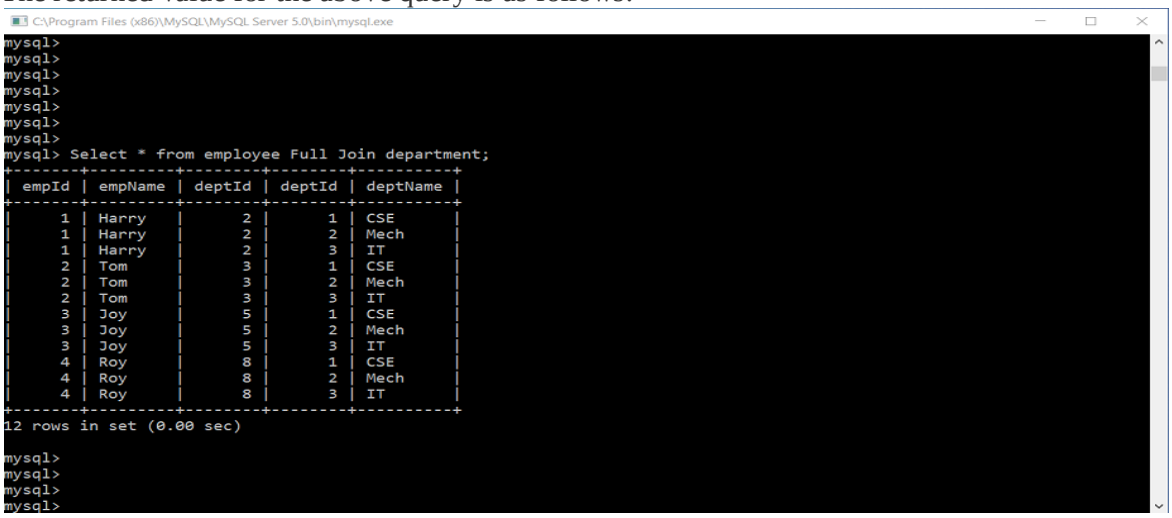
```
C:\Program Files (x86)\MySQL\MySQL Server 5.0\bin\mysql.exe
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> Select employee.empId, employee.empName, department.deptName from employee Right Outer Join department on employe
e.deptId = department.deptId;
+-------+---------+----------+
| empId | empName | deptName |
+-------+---------+----------+
|  NULL | NULL    | CSE      |
|     1 | Harry   | Mech     |
|     2 | Tom     | IT       |
+-------+---------+----------+
3 rows in set (0.00 sec)

mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
```

3. Full-Outer Join:

The Full-Outer join contains all the values of both the tables whether they have matching values in them or not. The Full-Outer Join can be depicted using the below diagram.



The MySQL query for full-outer join can be as follows:
Select * from employee Full Join department;
The returned value for the above query is as follows:



This is all about join in DBMS. Hope you learned something new today. That's it for this blog.

Division operation

The division operator is used for queries which involve the 'all'.
R1 ÷ R2 = tuples of R1 associated with all tuples of R2.

Example

Retrieve the name of the subject that is taught in all courses.

Name	Course
System	Btech
Database	Mtech
Database	Btech
Algebra	Btech

÷

Course
Btech
Mtech

=

Name
database

The resulting operation must have all combinations of tuples of relation S that are present in the first relation or R.

Example

Retrieve names of employees who work on all the projects that John Smith works on.

Consider the Employee table given below −

Name	Eno	Pno
John	123	P1
Smith	123	P2
A	121	P3

÷

Works on the following −

Eno	Pno	Pname
123	P1	Market
123	P2	Sales

=

The result is as follows

Eno
123

The expression is as follows

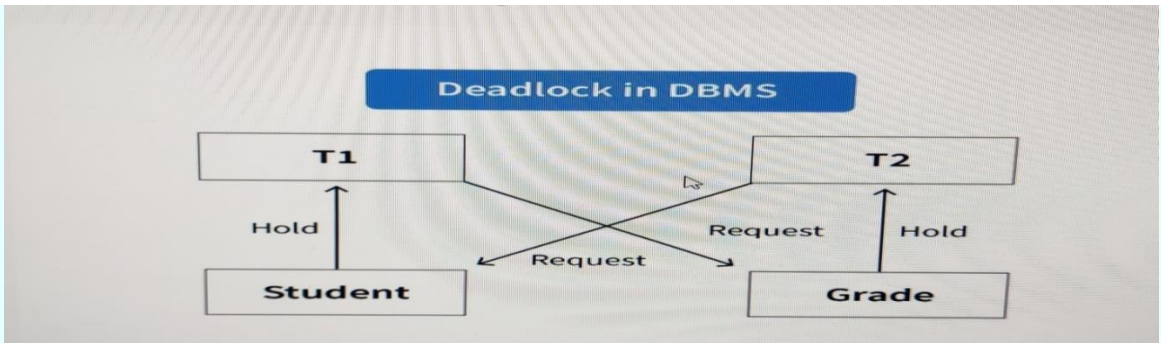
Smith <- ΠPno(σEname = ‘john smith’ (employee * works on Pno=Eno))

Deadlock

Deadlock in a database management system (DBMS) is an undesired **situation in which two or more transactions have to wait indefinitely for each other in order to get terminated, but none of the transactions is willing to give up the allocated CPU and memory resources that the other one needs**. Deadlock brings the whole system to a halt as no task ever gets finished and is in waiting state forever.

Let's understand deadlock in DBMS with an example of students database, where transaction 1 holds a lock on certain records of the Student table and needs to update those records in the Grade table. Simultaneously, there is transaction 2, which holds locks on some other records in the Grade table and needs to update those records in the Student table, which are already held by transaction 1.

Now, the main problem of deadlock arises when transaction 1 is waiting for Transaction 2 to release its lock and similarly, transaction Transaction 2 is waiting for Transaction 1 to release its lock. All activities come to a halt. *Both the transaction involved in such a situation will remain at a standstill until the DBMS detects the deadlock and aborts one of the two transactions that are causing deadlock.*



The three basic techniques to control deadlocks are:

• Deadlock prevention

A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. If the transaction is aborted, all changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.

• Deadlock detection

The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the "victim") is aborted (rolled back and restarted) and the other transaction continues.

• Deadlock avoidance

The transaction must obtain all of the locks it needs before it can be executed. This technique avoids the rolling back conflicting transactions by requiring that locks be obtained in succession. However, the serial lock assignment required in deadlock avoidance increases action response times.

Concurrency Control in DBMS

Concurrency control concept comes under the Transaction in database management system (DBMS). It is a procedure in DBMS which helps us for the management of two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems. Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data. For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

Advantages of Concurrency:

In general, concurrency means, that more than one transaction can work on a system.

The advantages of a concurrent system are:

- **Waiting Time:** It means if a process is in a ready state but still the process does not get the system to get execute is called waiting time. So, concurrency leads to less waiting time.
- **Response Time:** The time wasted in getting the response from the cpu for the first time, is called response time. So, concurrency leads to less Response Time.
- **Resource Utilization:** The amount of Resource utilization in a particular system is called Resource Utilization. Multiple transactions can run parallel in a system. So, concurrency leads to more Resource Utilization.
- **Efficiency:** The amount of output produced in comparison to given input is called efficiency. So, Concurrency leads to more Efficiency.

Major goals of concurrency control mechanisms

- **Serializability: -** Serializability is the classical concurrency scheme. It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. It assumes that all accesses to the database are done using read and write operations.
- Recoverability.
- Distributed serializability and commitment ordering.
- Distributed recoverability.
- Recovery.
- Replication.

Concurrency control techniques
The concurrency control techniques are as follows –

Locking
Lock guaranties exclusive use of data items to a current transaction. It first accesses the data items by acquiring a lock, after completion of the transaction it releases the lock. Types of Locks

Introduction to Locking

Locking is one of the most commonly used concurrency control schemes in DBMS. This works by associating a variable *lock* on the data items. This variable describes the status of the data item with respect to the possible operations that can be applied on it. The value of this variable is used to control the concurrent access and the manipulation of the associated data item.

The concurrency control technique in which the value of the lock variable is manipulated is called **locking**. The technique of locking is one way to ensure Serializability in DBMS.

In DBMS, locking is the responsibility of a subsystem called **lock manager**.

The types of locks are as follows –

Binary Locks

A binary lock has two states or values associated with each data item. These values are:

1. Locked – 1
2. Unlocked – 0

If a data item is **locked**, then it cannot be accessed by other transactions i.e., other transactions are forced to wait until the lock is released by the previous transaction.

But, if a data item is in the **unlocked** state, then, it can be accessed by any transaction and on access the lock value is set to locked state.

These locks are applied and removed using **Lock** () and **Unlock** () operation respectively.

In binary locks, at a particular point in time, only one transaction can hold a lock on the data item. No other transaction will be able to access the same data concurrently. Hence, Binary locks are very **simple** to apply but are **not used practically**.

1. **Shared Lock:-**Transaction can read only the data item values] 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: -** Used for both read and write data item values] 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.

Time Stamping

Time stamp is a unique identifier created by DBMS that indicates relative starting time of a transaction. Whatever transaction we are doing it stores the starting time of the transaction and denotes a specific time.

This can be generated using a system clock or logical counter. This can be started whenever a transaction is started. Here, the logical counter is incremented after a new timestamp has been assigned.

Optimistic

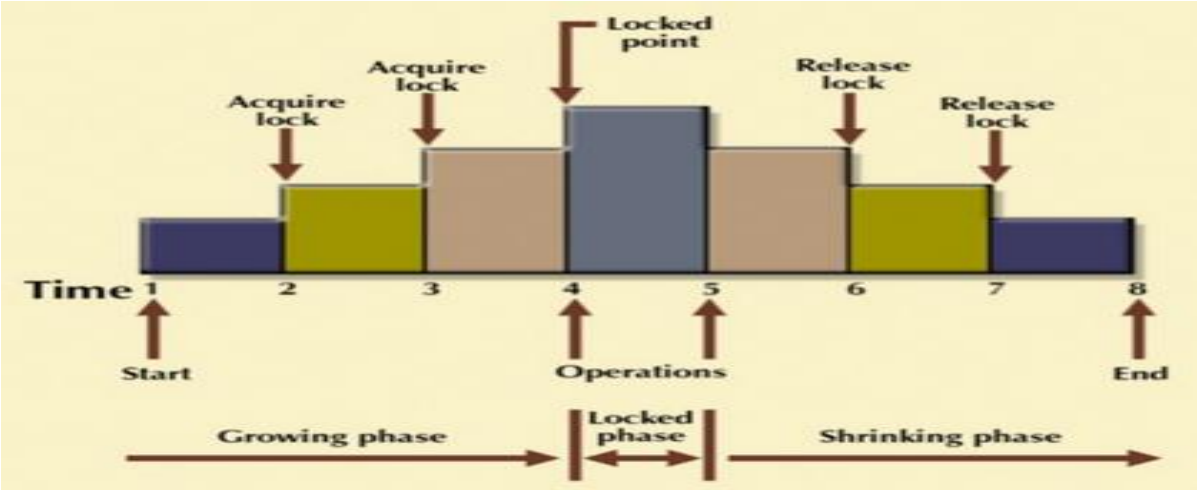
It is based on the assumption that conflict is rare and it is more efficient to allow transactions to proceed without imposing delays to ensure serializability.

Concurrency Control Protocols

Two Phase Locking Protocol

Two Phase Locking Protocol also known as 2PL protocol is a method of concurrency control in DBMS that ensures serializability by applying a lock to the transaction data which blocks other transactions to access the same data simultaneously. Two Phase Locking protocol helps to eliminate the concurrency problem in DBMS.
This locking protocol divides the execution phase of a transaction into three different parts.

- In the first phase, when the transaction begins to execute, it requires permission for the locks it needs.
- The second part is where the transaction obtains all the locks. When a transaction releases its first lock, the third phase starts.
- In this third phase, the transaction cannot demand any new locks. Instead, it only releases the acquired locks.



The Two-Phase Locking protocol allows each transaction to make a lock or unlock request in two steps:

- **Growing Phase**: In this phase transaction may obtain locks but may not release any locks.
- **Shrinking Phase**: In this phase, a transaction may release locks but not obtain any new lock

It is true that the 2PL protocol offers serializability. However, it does not ensure that deadlocks do not happen.
In the above-given diagram, you can see that local and global deadlock detectors are searching for deadlocks and solve them with resuming transactions to their initial states.

Timestamp-based Protocols

Timestamp based Protocol in DBMS is an algorithm which uses the System Time or Logical Counter as a timestamp to serialize the execution of concurrent transactions. The Timestamp-based protocol ensures that every conflicting read and write operations are executed in a timestamp order.

The older transaction is always given priority in this method. It uses system time to determine the time stamp of the transaction. This is the most commonly used concurrency protocol.
Lock-based protocols help you to manage the order between the conflicting transactions when they will execute. Timestamp-based protocols manage conflicts as soon as an operation is created.
Example:
Suppose there are there transactions T1, T2, and T3.
T1 has entered the system at time 0010
T2 has entered the system at 0020
T3 has entered the system at 0030
Priority will be given to transaction T1, then transaction T2 and lastly Transaction T3.
Advantages:

- Schedules are serializable just like 2PL protocols
- No waiting for the transaction, which eliminates the possibility of deadlocks!

Disadvantages:
Starvation is possible if the same transaction is restarted and continually aborted
(Starvation is the situation when a transaction needs to wait for an indefinite period to acquire a lock.)

Method used for DBMS Deadlock Recovery

The most common method to recover from a deadlock is to rollback one or more transactions until a no deadlock situation is reached. The **actions** that would be taken for deadlock recovery will be as follows:

- **Choice of the Victim Transaction**
- **Roll back**
- **Starvation**

Choice of the Victim Transaction

The transaction which is chosen for a rollback when a deadlock situation is reached is known as a **victim transaction** and the method is known as **victim selection**.

The victim may be selected based on any of the following **criterions**.

- Transaction may be selected randomly.
- Transaction will be selected depending on when the transaction started to run. That is the youngest transaction may be selected as the victim because it would have done the least amount of work.
- The transaction which needs a greater number of data items to be locked.
- The transaction which has done least number of data updates to the database because all the data written or changed in the database must be undone in the case of all a rollback.

- The transaction which would affect the roll back of a least number of other transactions. This means the victim transaction must lead to **least** amount of **cascade rollback**.

  So, we can see that the victim transaction must be such a transaction which would incur the **minimum cost**.

Roll back

Once the victim transaction is selected the next important point to decide is how far this transaction should be rolled back. There are two ways in which the transaction can be rolled back. These are as follows:

- One of the simplest solutions is to **roll back** the transaction **entirely** and restart it later. This is referred to as a **full rollback**.
- On the other hand, some transactions can be rolled back to a **particular point** which is enough to break the deadlock. This type of rollback is known as a **partial rollback**. In such transactions additional information regarding the state of the current transaction execution, bookmark information, locks required etc. must be maintained by the system.

Starvation

Starvation is a condition which arises when one same transaction is always picked as the victim transaction. When the same transaction is rolled back again and again it will never complete its execution.

In other words, we can say that when a transaction cannot proceed for an indefinite period of time and other transactions in the system continue to execute normally, such a situation is called **starvation** or **Live Lock**.

For example, let us assume there are two transactions T1 and T2 which are trying to acquire a lock on a data item X. The scheduler grants the lock to T1. As a result, T2 has to wait for the lock. Now beforeT1 unlocks the data item X, a new transaction T3 also wants to acquire the lock on the data item X. Now, if the scheduler grants the lock to T3, T2 again has to wait for the lock. As a result, T2 has to wait indefinitely even if there is no deadlock. Hence, this situation is called starvation.

Solution to Starvation

In order to avoid the situation of starvation to implement DBMS Deadlock Recovery we can choose one of the following ways:

- A transaction must be picked as the victim transaction only a finite number of times. This can be done by maintaining an index to how many times a transaction has aborted and rolled back.
- The scheduler must follow the policy of first come first serve basis.

Transaction processing

Transaction processing means dividing information processing up into individual, indivisible operations, called transactions, that complete or fail as a whole; a transaction can't remain in an intermediate, incomplete, state (so other processes can't access the transaction's data until either the transaction has completed or it has been "rolled back" after failure). Transaction processing is designed to maintain database integrity (the consistency of related data items) in a known, consistent state.

- 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. R(X);
2. X = X - 500;
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.

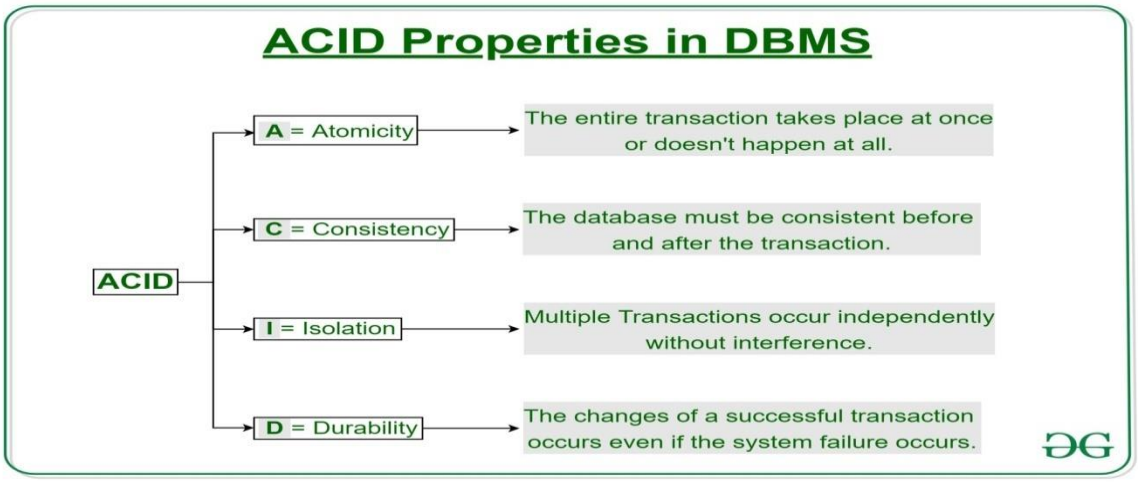
To solve this problem, we have two important operations:

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

Rollback: It is used to undo the work done.

ACID Properties in DBMS

A **transaction** is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations. In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.



Atomicity:
By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.
—**Abort**: If a transaction aborts, changes made to the database are not visible.
—**Commit**: If a transaction commits, changes made are visible.
Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

Consistency:

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,
The total amount before and after the transaction must be maintained.
Total **before T** occurs = **500 + 200 = 700**.
Total **after T occurs** = **400 + 300 = 700**.
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

Isolation:

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Let **X**= 500, **Y** = 500.
Consider two transactions **T** and **T".**

T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write (Y)	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result, interleaving of operations takes place due to which **T''** reads the correct value of **X** but the incorrect value of **Y** and sum computed by
T'': (X+Y = 50, 000+500=50, 500)
is thus not consistent with the sum at end of the transaction:
T: (X+Y = 50, 000 + 450 = 50, 450).
This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system

failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Some important points:

Property	Responsibility for maintaining properties
Atomicity	Transaction Manager
Consistency	Application programmer
Isolation	Concurrency Control Manager
Durability	Recovery Manager

The **ACID** properties, in totality, provide a mechanism to ensure the correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

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.

Anomalies in DBMS

Anomaly:-Problems that can occur in poorly planned, un-normalized databases where all the data is stored in one table

There are three types of **anomalies that occur when the database is not normalized**. These are: Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: A manufacturing company stores the employee details in a table Employee that has four attributes: Emp_Id for storing employee's id, Emp_Name for storing employee's name, Emp_Address for

storing employee's address and Emp_Dept for storing the department details in which the employee works. At some point of time the table looks like this:

Emp Id	Emp Name	Emp Address	Emp Dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

This table is not normalized.

We will see the problems that we face when a table in database is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if Emp_Dept field doesn't allow null.

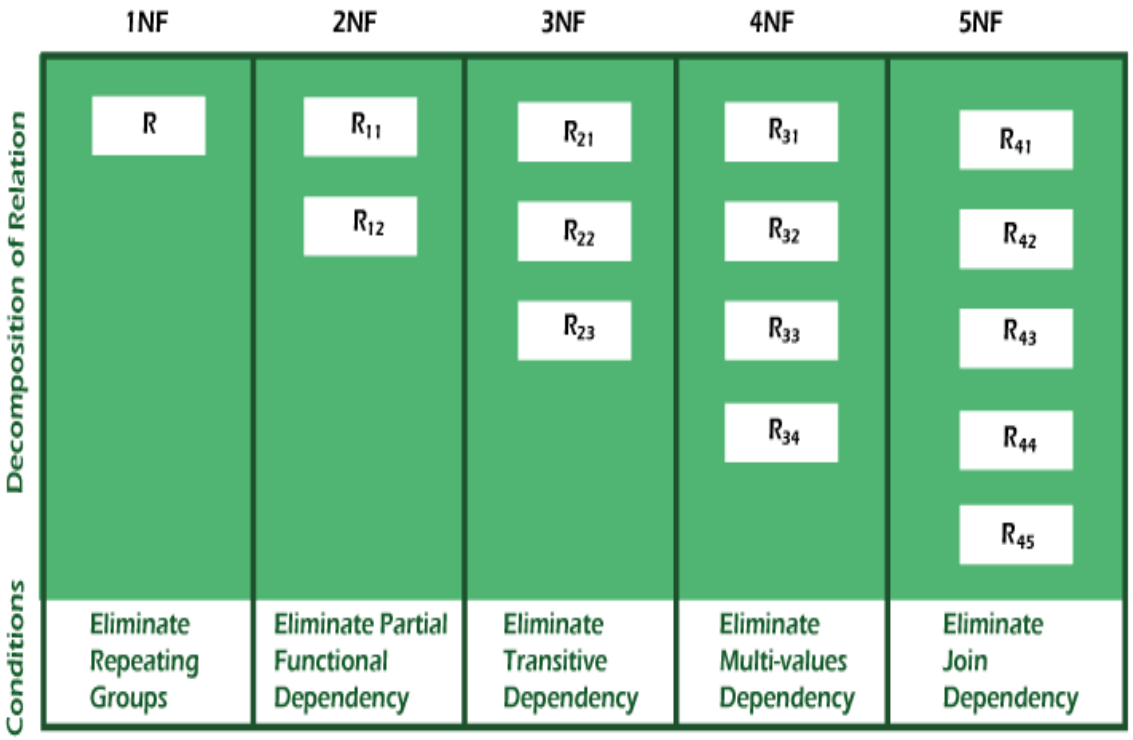
Delete anomaly: Let's say in future, company closes the department D890 then deleting the rows that are having Emp_Dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

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.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key
- **Example**: Let's say a school wants to store the data of teachers and the subjects they teach. They create a table Teacher that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

Teacher Id	Subject	Teacher Age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

- **Candidate Keys**: {Teacher_Id, Subject}

  Non prime attribute: Teacher_Age

- This table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute Teacher_Age is dependent on Teacher_Id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says **"no non-prime attribute is dependent on the proper subset of any candidate key of the table"**.

- To make the table complies with 2NF we can disintegrate it in two tables like this: **Teacher Details table:**

Teacher_Subject table:

Teacher Id	Teacher Age
111	**38**
222	**38**
333	**40**

Teacher Id	Subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Now the tables are in Second normal form (2NF).

Third Normal form (3NF)

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- No **Transitive dependency** .

An attribute that is not part of any **candidate key** is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Let's say a company wants to store the complete address of each employee, they create a table named Employee_Details that looks like this:

Emp_Id	Emp_Name	Emp_Zip	Emp_State	Emp_City	Emp_District
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {Emp_Id}, {Emp_Id, Emp_Name}, {Emp_Id, Emp_Name, Emp_Zip}…so on
Candidate Keys: {Emp_Id}
Non-prime attributes: all attributes except Emp_Id are non-prime as they are not part of any candidate keys.

Here, Emp_State, Emp_City & Emp_District dependent on Emp_Zip. Further Emp_zip is dependent on Emp_Id that makes non-prime attributes (Emp_State, Emp_City & Emp_District) transitively dependent on super key (Emp_Id). This violates the rule of 3NF.

To make this table complies with 3NF we have to disintegrate the table into two tables to remove the transitive dependency:

Employee Table:

Emp_Id	Emp_Name	Emp_Zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

Employee_Zip table:

Emp_Zip	Emp_State	Emp_City	Emp_District
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.
To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:
For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}
Now, this is in BCNF because left side part of both the functional dependencies is a key.

Functional dependency in DBMS

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally:
If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example
STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency **(Join dependency is a constraint which is similar to functional dependency or multivalued dependency.)** and joining should be lossless **(The common attribute of the sub relations is a super key of any one of the relation).**
-
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
-
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

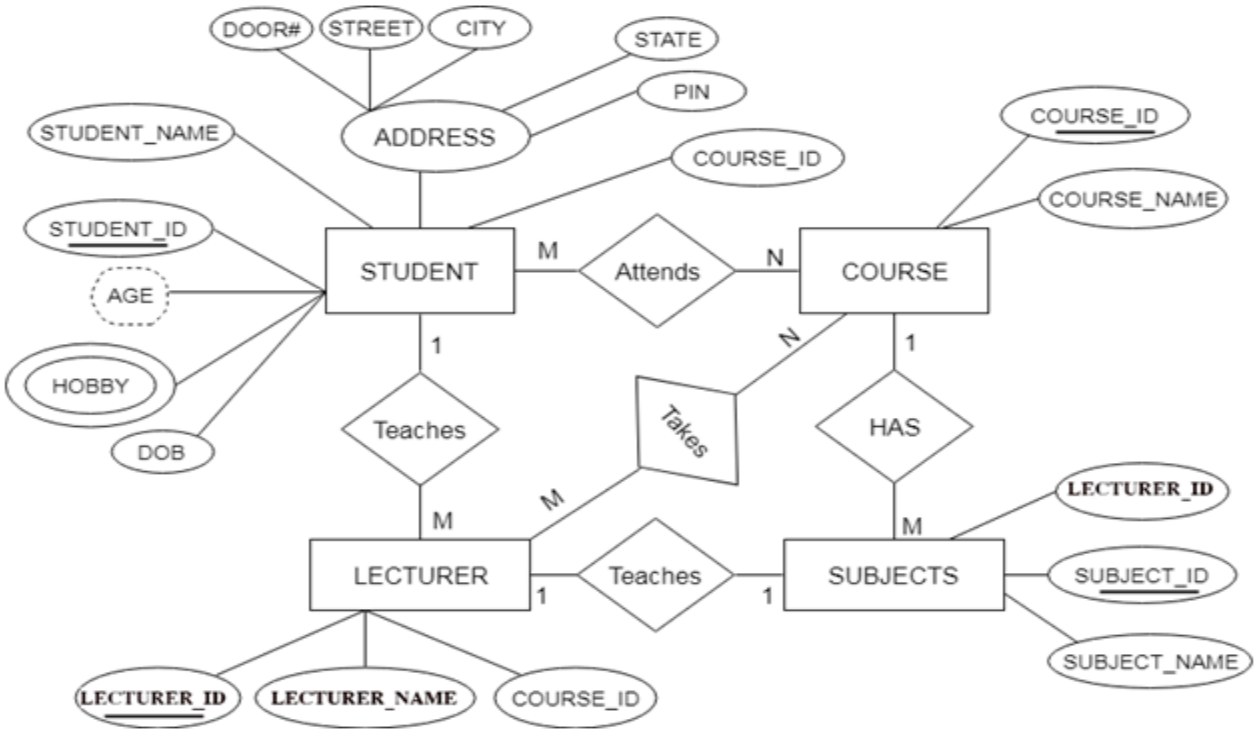
SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

ER diagram to Relational database

The database can be represented using the notations, and these notations can be reduced to a collection of tables.

In the database, every entity set or relationship set can be represented in tabular form.

The ER diagram is given below:



There are some points for converting the ER diagram to the table:

- **Entity type becomes a table.**

In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

- **All single-valued attribute becomes a column for the table.**

In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.

- **A key attribute of the entity type represented by the primary key.**

In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.

- **The multivalued attribute is represented by a separate table.**

In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.

- **Composite attribute represented by components.**

In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.

- **Derived attributes are not considered in the table.**

In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

Using these rules, you can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below: