BASIC TERMINOLOGIES

Database: Database is a collection of **data which are o**rganized in particular way to enable its users to easily manage and update the data.

DBMS: DBMS is a system used to Define, Create, and Update and Control the data in database. Example: XML files, MS ACCESS.

RDBMS: Relational Database Management System (RDBMS) is an advanced version of a DBMS. RDBMS is a system software which is used to store the data which are in the form of tables. Here, data is stored in rows and columns which is known as tuples and attribute. Example of RDBMS is MySQL, Oracle, SQL Server, etc.

DBMS	RDBMS
• DBMS stores data as a file	• RDBMS, data is stored in the form of tables.
• DBMS supports single users	• RDBMS supports multiple users.
• DBMS does not support client-server architecture	• RDBMS supports client-server architecture.
• DBMS has low software and hardware requirements	• RDBMS has higher hardware and software requirements
• In DBMS, data redundancy is Occur	• In RDBMS, keys and indexes do not allow data redundancy
• Ex: XML Files, MICROSOFT ACCESS	• MySQL, SQL SERVER

DATABASE MODEL:

A Database model defines how data will be stored, accessed and updated in a database management system.

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

Hierarchical model:

In hierarchical model, data is organised into tree-like structure. one-to-many relationship between data. For example, one department can have many courses. Child node has only one parent

Network Model:

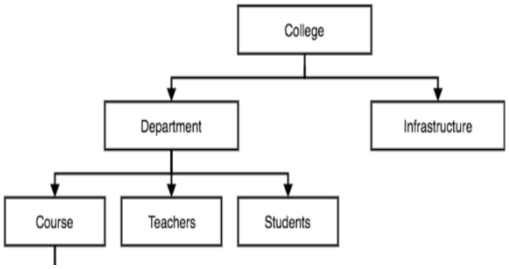
This is an extension of the Hierarchical model. In this model data is organised more like a graph. Child node may has more than one parent node. Many-to-many data relationships between data.

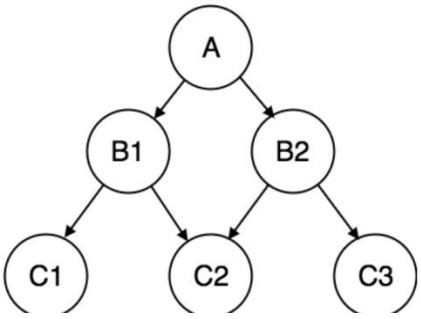
ER model: Entity-relationship Model used to represent the relationships into pictorial form .This model is good to design a database, which can converted into tables in relational model

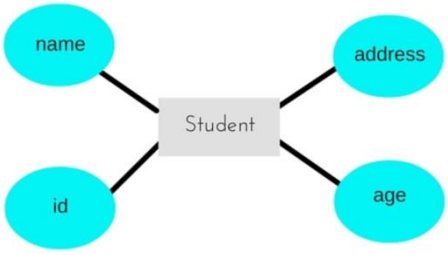
Let's take an example, If we have to design a School Database, then **Student** will be an **entity** with **attributes** name, age, address etc. As **Address** is generally complex, it can be another **entity** with **attributes** street name, pin code, city etc., and there will be a relationship between them.

RELATIONAL MODEL:

In this model, data is organised in two-dimensional table. This model used widely.







student_Id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

File vs Database Management approach

File management system

- A file management system is a collection of programs that manage and store data in files .
- It is the easiest way to store the data like text, videos, images, audios, etc. in general files.
- File systems contain information about size of the file, access permission, location and date of created in the form of metadata. File systems use metadata to store and retrieve files from Hard disk.
- Data redundancy is high in file management system. So, Data consistency is not met.

- For example, NTFS (New Technology file system) is the Windows file system, and EXT (Extensible file system) is the Linux file system.

Limitations of File Management System:

1. Data Redundancy:

It is possible that the same information may be duplicated in different files. This leads to data redundancy and cause for memory wastage.

2. Data Inconsistency:

Because of data redundancy, it is possible that data may not be in consistent state.

Note: Data consistency means any changes made to the one database are immediately reflected to all other database.it is possible only when data has no redundancy.

3. Integrity Problems:

Data integrity means that the data contained in the database is correct and consistent. But consistent is not possible in File based system.so Integrity problem occur.

4. Limited Data Sharing:

Data are found in various files. Also different files may have different formats and these files may be stored in different folders. So, due to this data isolation, it is difficult to share data among different applications.

5. Fixed Queries

Query or reports needed by the organization has to be developed by the application programmer.

Advantages of Database approach:

1. Reducing Data Redundancy

In a database as there is a single database and any change in it is reflected immediately. Because of this, there is no chance of encountering duplicate data.

2. Data Consistency

Data consistency is ensured in a database because there is no data redundancy. Any changes made to the database are immediately reflected to all the users and there is no data inconsistency.

3. Data Integrity

Data integrity means that the data is correct and consistent in the database. There are multiple databases in a DBMS. Data integrity can be ensure in DBMS.

4. Sharing of Data

In DBMS, the users of the database can share the data among themselves. There are various levels of authorisation to access the data. Many remote users can also access the database simultaneously and share the data between themselves.

5. Data Security

Data Security is vital concept in a database. Only authorised users should be allowed to access the Database.

6. Backup and Recovery

Database Management System automatically takes care of backup and recovery. The users don't need to backup data periodically because this is taken care of by the DBMS. It also restores the database after a crash or system failure.

Disadvantage of DBMS:

1. Increased Cost
2. Maintenance Cost
3. Frequent Upgrade needed

Three schema Architecture

The three schema architecture is used to separate the user applications and physical database. The three schema architecture contains three-levels.

1. **Internal Level**

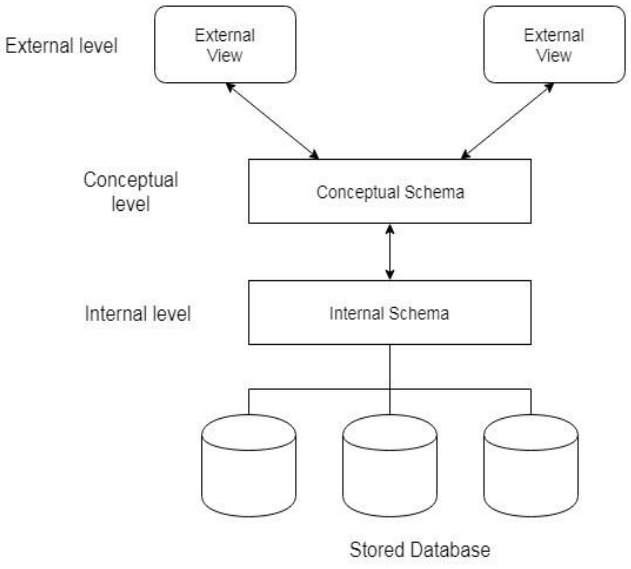
-describes the database structure.
- define how data will be stored in a block.
-Internal level is also known as physical level.

2. **Conceptual Level**

-describes what data are to be stored in the database and also describes what relationship exists among those data.
-Conceptual level is also known as logical level.
-Programmers and database administrators work at this level.

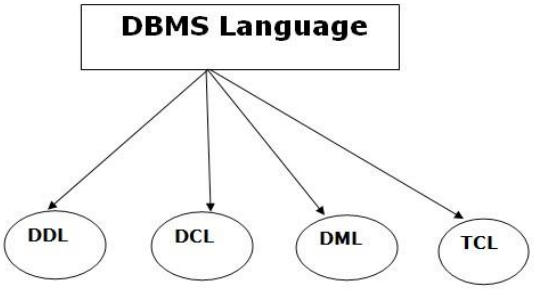
3. **External Level**

-At the external level, a database contains several schemas called as subschema.



-It describe the database which is interested by user group and hides the remaining database from that user group.
-External schema is also known as view schema.

Types of Database Language



1. Data Definition Language

-DDL stands for Data Definition Language.
-used to define database structure.
-Task come under the DDL: - **Create, Alter, Drop.**

2. Data Manipulation Language

-DML stands for Data Manipulation Language.
-Used for accessing and manipulating data in a database & handles user requests.
-Task come under the DDL: - **select ,insert, update**

3. Data Control Language

-DCL stands for Data Control Language.
- Used to retrieve the saved data.
-Task come under the DCL:-

- **Grant**: Used to give permission for user to access the database.
- **Revoke**: Used to take back permission from the user.

4. Transaction Control Language

- used to save the changes made by the DML statement.
-tasks that come under TCL:

- **Commit:** Used to save the transaction on the database.
- **Rollback:** Used to restore the database to its original state

ER MODEL

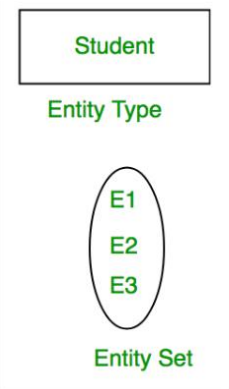
-used to represent the relationships into pictorial form .So, Different clients can easily understand. This model is good to design a database. It can converted into tables in relational model.

Entity, Entity Set –

An Entity is a table which used to represent the object with a physical existence – a particular person, car, house, or employee – or the object with a conceptual existence – a company, a job, or a university course.

For example, in a school database, students, teachers and courses offered can be considered as different entities. All these entities have some attributes.

An entity set contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school.



Relationship

--participation (all the rows(tuples) participating or not)

Type of relationship

- one to one
- 

- one to many or many to one

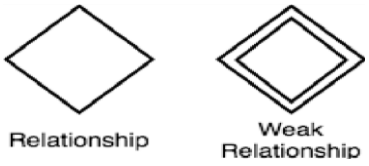
One to many: one customer has many account

Many to one: many student have one teacher

- many to many

ENTITY

Relationships between Entities - Weak and Strong



Weak Entity

- is an entity that depends on another entity.

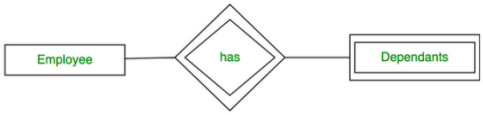
-Like strong entity, weak entity does not have any primary key

-In ER MODEL, Weak entities represented with **double rectangular** box.

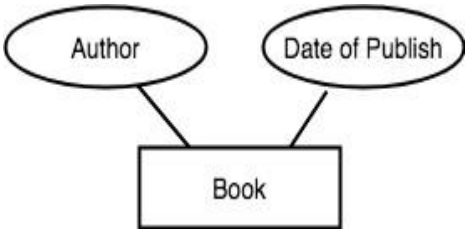
-Weak entity always has total participation but Strong entity may not have total participation.

Example:

A company may store the information of dependants (Parents, Children, Spouse) of an Employee. But the dependents don't have existence without the employee. So Dependent will be weak entity type and Employee will be strong entity type for Dependant.

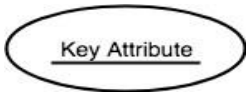


Attributes for any Entity



Key Attribute for any Entity

The attribute which **uniquely identifies each entity** in the entity set is called key attribute. For example, Roll_No



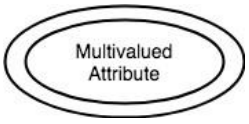
Types of Attribute:

1. composite(fast name, last name) & simple attribute(name)
2. single value(pan no) & multivalued attribute(mobile no)
3. derived attribute (age)
4. complex attribute(address)

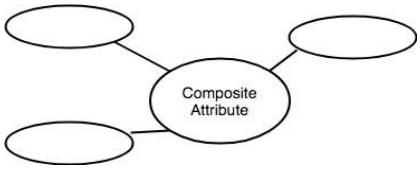
Derived Attribute for any Entity

Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth

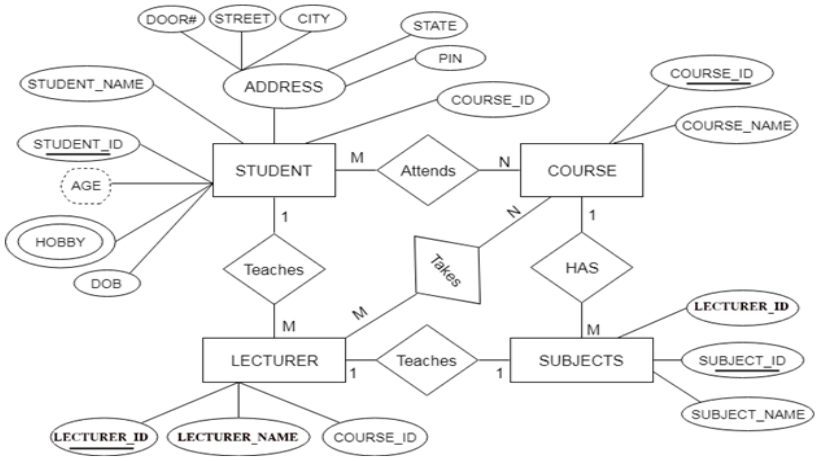
Multivalued Attribute for any Entity: ex. phone number



Composite Attribute for any Entity: Ex. address



The ER diagram is given below:



The improvement in DBMS:

1. Generalization
2. Specialization
3. Aggregation

These are 3 approaches added to Traditional ER Model for Database Modelling.

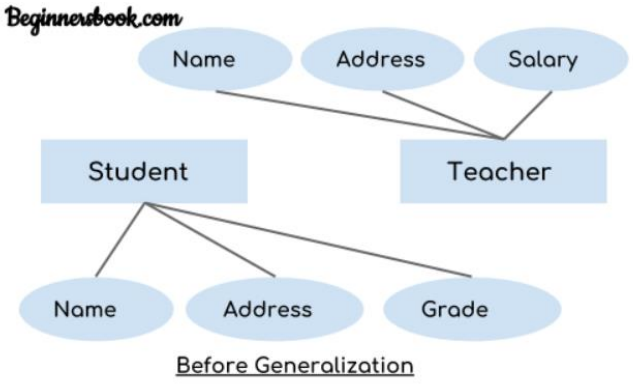
Generalization

-Generalization uses bottom-up approach

- Two or more than two lower level entities combine together to form a higher level new entity.

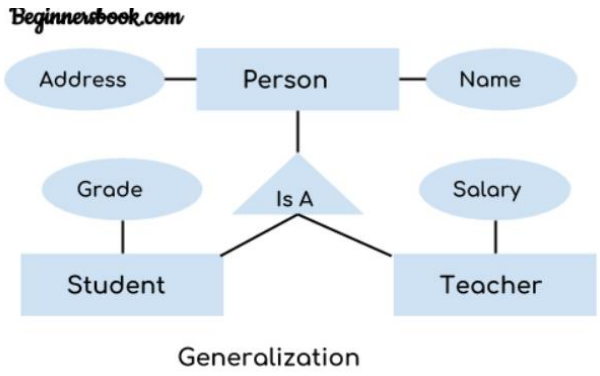
Example:

Let's say we have two entities Student and Teacher.
Attributes of Entity Student are: Name, Address & Grade
Attributes of Entity Teacher are: Name, Address & Salary



These two entities have two common attributes: Name and Address, we can make a generalized entity with these common attributes

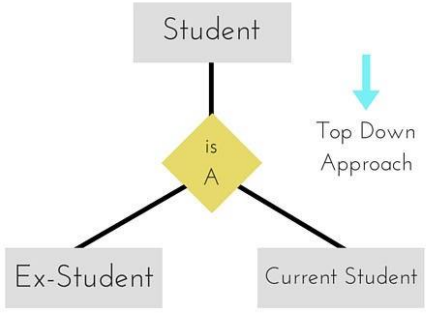
After generalization:



Specialization

-Specialization is a top-down approach, and it is opposite to Generalization.

- In specialization, one higher level entity can be broken down into two lower level entities.



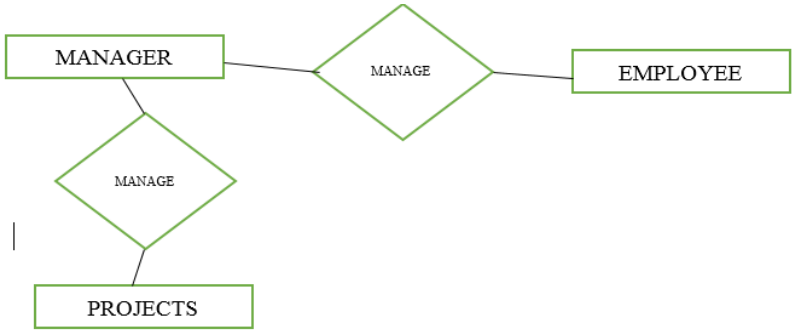
Aggregation

In Aggregation relation between two entities is treated as a **single entity**.

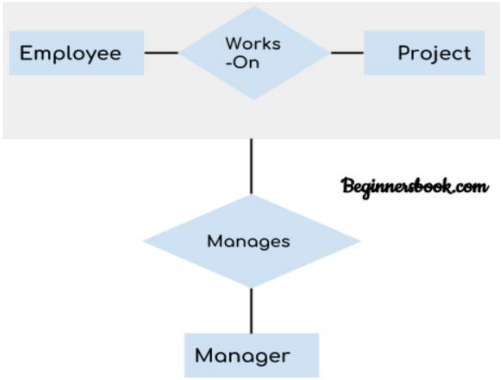
Ex:

Let's consider the situation. There is a manager who manages Employee as well as projects.

Before aggregation:



After Aggregation:



A manager not only manages the employee working under them but he has to manage the project as well. In such scenario if entity "manager" makes a "manages" relationship with either "employee" or "project" entity alone then it will not make any sense because he has to manage both. In our example, the relationship "works-on" between "employee" & "project" acts as one entity that has a relationship "manages" with the entity "manager".

RELATIONAL MODEL:

In this model, data is organised in two-dimensional table which contain row and coloumn . This model used widely.

Ex: Oracle Database, MySQL, Microsoft SQL Server

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20

IMPORTANT TERMINOLOGIES in Relational model

1. **Relation Schema:** A relation schema represents the structure of the relation(table), name of the relation(name of table), with its attributes.
   e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT.
2. **Tuple:** Each row in the relation is known as tuple.
3. **Degree:** The number of attributes in the relation is known as degree of the relation.
4. **Cardinality:** The number of tuples in a relation is known as cardinality.
5. **Attribute:** Each column in a Table known as attribute. e.g., Student_Rollno, NAME,etc.
6. **NULL Values:** The value which is not known or unavailable is called NULL value. It is represented by blank space

Introduction to Database Keys

1. **Super Key**
2. **Candidate Key**
3. **Primary Key**
4. **Foreign Key**
5. **Unique Key**
6. **Composite Key**

Keys used to identify relationships between tables and also to uniquely identify any row of data inside a table. A Key can be a single attribute or a group of attributes.

1. Super Key

Super Keys are set of column which can uniquely identify each row within a table. Super Key is a superset of Candidate key.

2. Candidate Key (primary attribute)

Candidate keys are the minimal set of column which can uniquely identify each row in a table.

3. Primary Key

Primary key is a candidate key which is used to uniquely identify each row(tuple) in a table. Primary key has no null value. One table may contain only one primary key.

4. Foreign key

Foreign key is the column of table which is to point to the primary key of another table.Its used to describe relationship b/w two tables.

For example, In a company, every employee works in a specific department .Here employee and department are two different entities. So we can't store the information of the department in the employee table.

We add primary key of DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table. Now in the EMPLOYEE table, Department_Id is the foreign key and both the tables are related.

5. Composite Key

Composite key is a set of two or more columns that uniquely identify each row in a table.

6. Unique Key

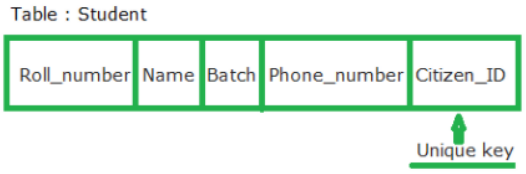
A unique key is a set of one or more than one columns of a table that uniquely identify each row in a table. Unique key can have NULL value. Multiple unique key can be found in a given table.

7. Secondary or Alternative key

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

Note:

- Composite means more than one field makes the row unique. So a composite key has more than one field. A Unique key can have 1 or more fields. So, all composite keys are unique keys but all unique keys are not composite keys.
- Primary key will not accept NULL values whereas unique key can accept one NULL value. A table can have only primary key whereas there can be multiple unique key on a table.



Here, Roll number is primary key and Citizen_ID is unique key. Citizen_ID column should be unique because each citizen of a country must have his or her Unique identification number like Aadhaar Number. But if student is migrated to another country in that case, he or she would not have any Citizen_ID and the entry could have a NULL value.

Prime attribute: The prober subset of candidate keys.
Non-prime Attributes: Attributes other than Prime attributes.

Join Operations:

A JOIN is used to combine rows from two or more tables, based on a related column between them.

Inner Join (Natural Join): An inner join combines two different tables and returns rows where the key exists in both tables.

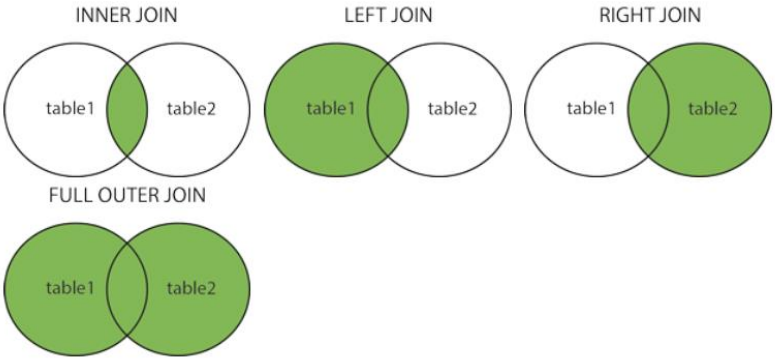
Outer Join:

- **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

CROSS JOIN (Cartesian Join): The CARTESIAN JOIN is also known as CROSS JOIN. In a CARTESIAN JOIN there is a join for each row of one table to every row of another table. This usually **happens when the matching column or WHERE condition is not specified.** i.e., the number of rows in the result-set is the product of the number of rows of the two tables.

SELF JOIN: As the name signifies, in SELF JOIN a table is joined to itself. In other words we can say that it is a join on certain condition between two copies of the same table. Here, the table has a FOREIGN KEY which references its own PRIMARY KEY.Ex: The employee table might be joined to itself in order to show the manager name and the employee name in the same row.

Equi Join: If you have written INNER join using where clause than using an equality operator as = will be known as an equijoin. The main difference between Self Join and Equi Join is that In Self Join we join one table to itself rather than joining two tables. Both Self Join and Equi Join are types of INNER Join in SQL.



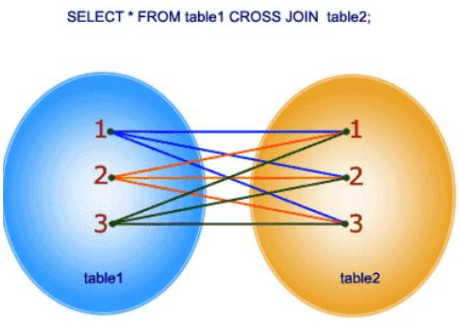


Table A:

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

Table B:

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Natural Join (Inner Join): Table contain Only Common row (foreign key of A which are found in primary key of B) from A AND B
Natural join of above table:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

Left Outer Join: Table contain all row from table and their respective row from right table using foreign key .if any foreign key of left table which is primary key of right table is missing in right table, that respective column of row filled as NULL value.

Left join of above table:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

Right Outer Join:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

Full Outer Join:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

Integrity Constraints

Data Integrity:

Data integrity means that the data is correct and consistent in the database.
Note: Data consistency means any changes made to the one database are immediately reflected to all other database.it is possible only when data has no redundancy.

Example: Let us imagine we have a customer database where we have two tables i.e. '*customer_table*'(customer_id, customer_name, purchase_id) and '*purchase_table*'(purchase_id, purchhased_item). These two tables are related by purchase

id.Therefore any purchase is made by the customer then that data of the purchased item will be stored in the *purchase_table.*

Integrity Constraints (Rules) in Relational Model

- Integrity Constraints are set of rules which are need to follow while entering data into the database table

Types of Integrity Constraint

- **Domain Integrity constraint**
- **Entity integrity constraint**
- **Referential Integrity constraint**
- **Key integrity constraints**

Domain integrity Constraints:

These are attribute level constraints. An attribute can only take values which lie inside the domain range. Example: If a constraint AGE>0 is applied, inserting negative value of AGE will result in failure.

Entity integrity constraints

The entity integrity constraint states that primary key value can't be null. This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows. A table can contain a null value other than the primary key field.

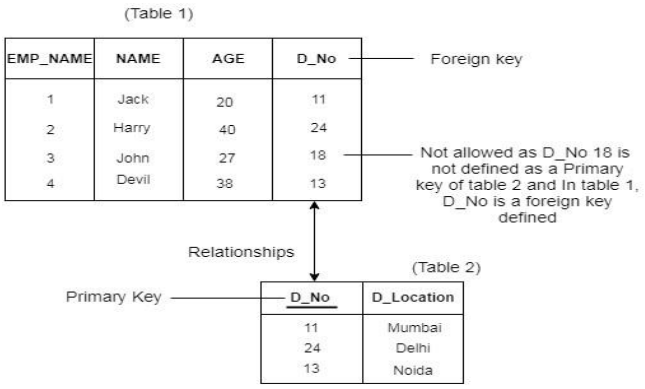
Referential Integrity Constraints:

Referential Integrity: it is specified between two tables.

In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2 the every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Let us suppose we have 2 relations

BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH table .



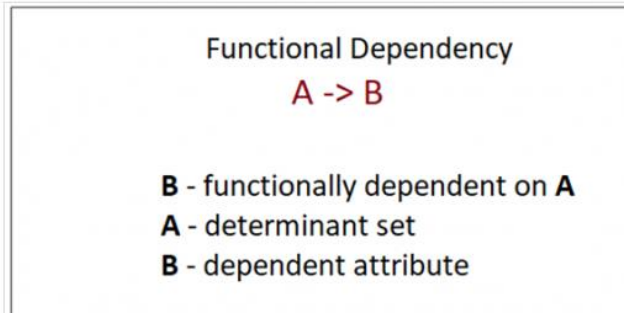
Key Integrity Constraints: It is used to maintain integrity of different keys which are available in table.

The types of key constraints-

- Primary key constraints- It uniquely identifies a row in a table. Not contain Null value
- Unique key constraints- provides a unique/distinct values to specified columns.
- Foreign Key constraints- ensures referential integrity of the relationship
- NOT NULL constraints- ensures that the specified column doesn't contain a NULL value.
- Check constraints- checks for the predefined conditions before inserting the data inside the table.
- DEFAULT constraints: provides a default value to a column if none is specified.

Functional Dependency

- The functional dependency is a relationship that exists between two attributes. 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).
- Functional dependencies used in Normalization to reduce the Data redundancy.



Types of Functional Dependencies

- **Trivial functional dependency**:
- **Non-trivial functional dependency**
- **Multivalued dependency**
- **Transitive dependency**

Trivial Functional Dependency

It occurs when B is a subset of A in:

```
A ->B
```

Example

We are considering the same **<Department>** table with two attributes to understand the concept of trivial dependency.

The following is a trivial functional dependency since **DeptId** is a subset of **DeptId** and **DeptName**

```
{ DeptId,  DeptName } -> Dept Id
```

Non –Trivial Functional Dependency

It occurs when B is not a subset of A in:

```
A ->B
```

Example

```
DeptId ->  DeptName
```

The above is a non-trivial functional dependency since DeptName is a not a subset of DeptId.

Multivalued dependency in DBMS: if one attribute determinant more than one attributes which have no dependency between them.

Transitive dependency:

A transitive functional dependency happens when a functional dependency is indirectly formed by two functional dependencies.

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{Company} -> {CEO} (if we know the compay, we know its CEO's name)
{CEO } -> {Age} If we know the CEO, we know the Age
Therefore according to the rule of rule of transitive dependency:
{ Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.

Partial Dependency

If the proper subset of candidate key determines non-prime attribute, is called partial dependency

Normalization

Keyword:

Data redundancy-same data occur different place

Data consistent- Data consistency means any changes made to the one database are immediately reflected to all the users or all other database.it is possible only when data has no redundancy.

Not consistent-the age of person x is different in different table

Anomaly-something which different from what is expected/normal

Non-prime attribute-attribute which is not part of any candidate key

Proper subset of candidate key-if AB is candidate key. The proper subset of candidate key is {A,B}
Partial Dependency– If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Normalization of Database

Normalization is a method of organizing the data in the database which helps you to avoid data redundancy, insertion, update & deletion anomaly in data base.

Problems without Normalization

-If a table is not properly normalized and have data redundancy then it will take extra memory space.
-Data redundancy leads to Data inconsistent. Insertion, Updation and Deletion Anamolies are very frequent if database is not normalized.
Example:

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

Above table is not normalized. There will be chance for data inconsistency if hod of branch CSE will update. I think above table is Not in 3NF.Because,non key is determinant non key value.
branch->hod
branch->office_tel
The normalized version of above table:

Rollno	Name	Branch id
401	Akon	101
402	Bkon	101
403	Ckon	101
404	Dkon	101

Branch id	Branch	Hod	Office_tel
101	CSE	Mr.X	53337

Type of Anomalies:

There are three types of anomalies: insertion, update and deletion anomalies.

Insertion Anomaly

Unable to add new data to the database due to absence of other data .

-Let's say we have a table that has 4 columns. Student ID, Student Name, Student Address and Student Grades. Now when a new student enrol in school, even though first three attributes can be filled but 4th attribute will have NULL value because he doesn't have any marks yet. If student grade not accept null values, this student data could not entered into the database. This results in database inconsistencies.

Updation Anomaly

-occur when forgetting to update the value of data in multiple places.

Let say we have 10 columns in a table out of which 2 are called employee Name and employee address. Now if one employee changes its location then we would have to update the table. if the table is not normalized one employee can have multiple entries(data redundancy) and while updating one of them might get missed. It will lead to data inconsistency.

Deletion Anomaly

-Deletion Anomalies happen when the deletion of unwanted information causes important information to be deleted as well.

For example, if a single database record contains information about a particular product along with information about a salesperson for the company. If the salesperson quits, then information about the product is deleted along with salesperson information.

Type of Normalization

First Normal Form (1NF)

As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic value(single value).

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

1 normalized form:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

1. Second normal form (2NF)

A table is said to be in 2NF if both the following conditions hold:

- Table is in 1NF (First normal form)
- relation must not contain any partial dependency.

Note – If the proper subset of candidate key determines non-prime attribute(attribute that is not part of any candidate key) , it is called partial dependency.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Functional dependency: teacher_id ->teacher_age
Candidate Keys: {teacher_id and subject}
Primary key: {teachere_id and subject}
Non prime attribute: teacher_age
Proper Subset of Candidate Key: teacher_id,subject
Prime attribute:teacher_id,subject
The table is in 1 NF because each attribute has atomic values. non prime attribute teacher_age is dependent on teacher_id which is a proper subset of candidate key. This is a partial dependency and so this relation is not in 2NF.
2 normalized form:

teacher_details table:

teacher_id	teacher_age
111	38
222	38
333	40

teacher_subject table:

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

Third Normal Form (3NF)

What are transitive functional dependencies?

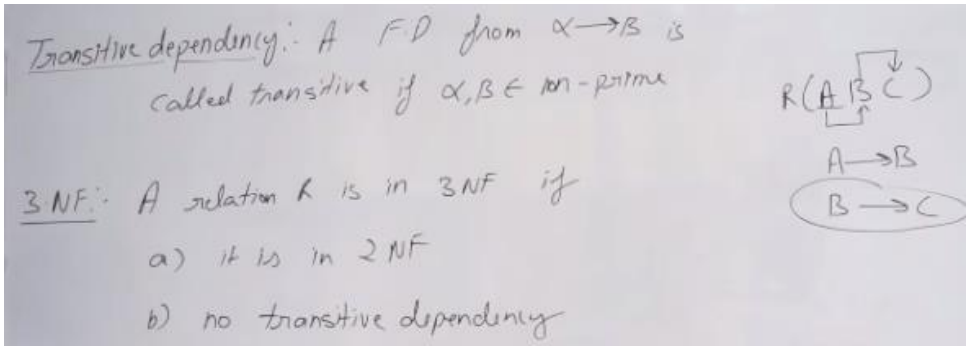
A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change

Consider the table 1. Changing the non-key column Full Name may change Salutation.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

Change in Name

May Change Salutation



Transitive dependency:

Non key->non key

Boyce and Codd Normal Form (BCNF)

This normal form is also referred as 3.5 normal forms

1. R must be in 3rd Normal Form
2. for each functional dependency (X → Y), X should be a super Key.

Example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}
Candidate key: {emp_id, emp_dept}
The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:
emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}
Candidate keys:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}
This is now in BCNF as in both the functional dependencies left side part is a key.

Denormalization in Databases

- Denormalization is the process of increasing the redundancy in the database.
- It is the opposite process of normalization.
- It is mostly done **for improving the performance**.
- Denormalization adds redundant data into normalized database for reducing the problems during database queries which combine data from the various tables into a single table.

Denormalization method:

- **Adding Redundant columns:**

In this method, only the redundant column which is frequently used is added to the main table.

```
SELECT e.EMP_ID, e.EMP_NAME, e.ADDRESS, d.DEPT_NAME
FROM EMPLOYEE e, DEPT d
WHERE e.DEPT_ID = d.DEPT_ID;
```

Example:
We have to generate a report where we have to show employee details and his department name. Here we need to have join EMPLOYEE with DEPT to get department name.

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEPT_ID	PROJ_ID
100	Joseph	Clinton Town	10	206
101	Rose	Fraser Town	20	205
102	Mathew	Lakeside Village	10	206
103	Stewart	Troy	30	204
104	William	Holland	30	202

DEPARTMENT

DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

But joining the huge EMPLOYEE and DEPT table will affect the performance of the query. But we cannot merge DEPT with EMPLOYEE. In this case, what we can add the redundant column DEPT_NAME to EMPLOYEE, so that it avoids EMPLOYEE TABLE join with DEPT TABLE and increasing the performance.

redundancy of data on DEPT_NAME.

EMPLOYEE

EMP_ID	EMP_NAME	ADDRESS	DEPT_ID	PROJ_ID	DEPT_NAME
100	Joseph	Clinton Town	10	206	Accounting
101	Rose	Fraser Town	20	205	Quality
102	Mathew	Lakeside Village	10	206	Accounting
103	Stewart	Troy	30	204	Design
104	William	Holland	30	202	Design

DEPARTMENT

DEPT_ID	DEPT_NAME
10	Accounting
20	Quality
30	Design

Pros of Denormalization:-

1. Retrieving data is faster.
2. Queries to retrieve can be simpler(and therefore less likely to have bugs), since we need to look at fewer tables.

Cons of Denormalization:-

1. Updates and inserts are more expensive.
2. Data may be inconsistent.

TRANSACTION

- Transaction is a unit consist of logically related operation like(read, write)

For example, you are transferring money from your bank account to your friend's account, the set of operations would be like this:

1. Read your account balance
2. Deduct the amount from your balance
3. Write the remaining balance to your account
4. Read your friend's account balance
5. Add the amount to his account balance
6. Write the new updated balance to his account

This whole set of operations can be called a transaction.

In DBMS, we write the above 6 steps transaction like this:

Lets say your account is A and your friend's account is B, you are transferring 10000 from A to B, the steps of the transaction are:

```
1. R(A);
2. A = A - 10000;
3. W(A);
4. R(B);
5. B = B + 10000;
6. W(B);
```

In the above transaction **R** refers to the **Read operation** and **W** refers to the **write operation**.

Transaction failure in between the operations

The transaction may fail before finishing the all the operations in the set. This can happen due to power failure, system crash etc.

Assume that transaction fail after third operation (see the example above) then the amount would be deducted from your account but your friend will not receive it.

To solve this problem, we have the following two operations.

- **Commit:** If all the operations in a transaction are completed successfully then commit operation help to save those changes to the database permanently.
- **Rollback:** If any one of the operation fails then roll back operation help to restore the database into its original state (before applying transaction).

Even though these operations can help us avoiding several issues that may arise during transaction but they are not sufficient when two transactions are running concurrently. To handle those problems we need to understand database ACID properties.

Transaction property (ACID)

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

Property of Transaction

Atomicity

- The transaction cannot occur partially. Each transaction is treated as one unit. Atomicity ensure that the transaction either completed successfully or not executed at all.
- Transaction control manager component responsible for atomicity.

Atomicity involves the following two operations:

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

Roll back: If a transaction fails then all the changes are not 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)	Read(B)
A:= A-100	Y:= Y+100
Write(A)	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

- It ensure that if the system is consistent before and after transaction
- User /application programmer responsible for 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

Isolation

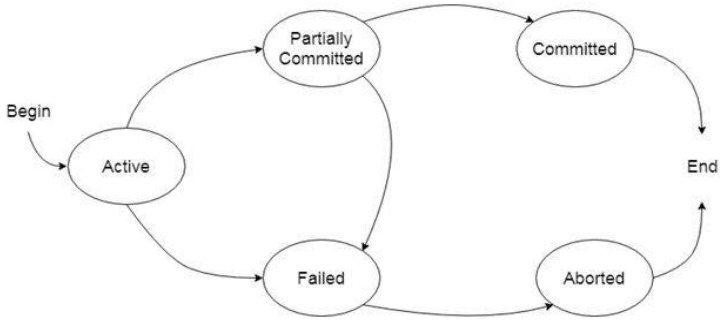
- It ensure that no transaction should be affected by other parallelly executed transaction.
- In isolation, if the transaction T1 is being executed and is using the data item X then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control manager component responsible for isolation property.

Durability

- It ensure that all the updates made by transaction must become permanent in DB irrespective of hardware and software failure.
- The recovery manager component is responsibility for durability.

States of Transaction

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



Active state

- Active state is the first state of the every transaction. In this state, 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.

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.

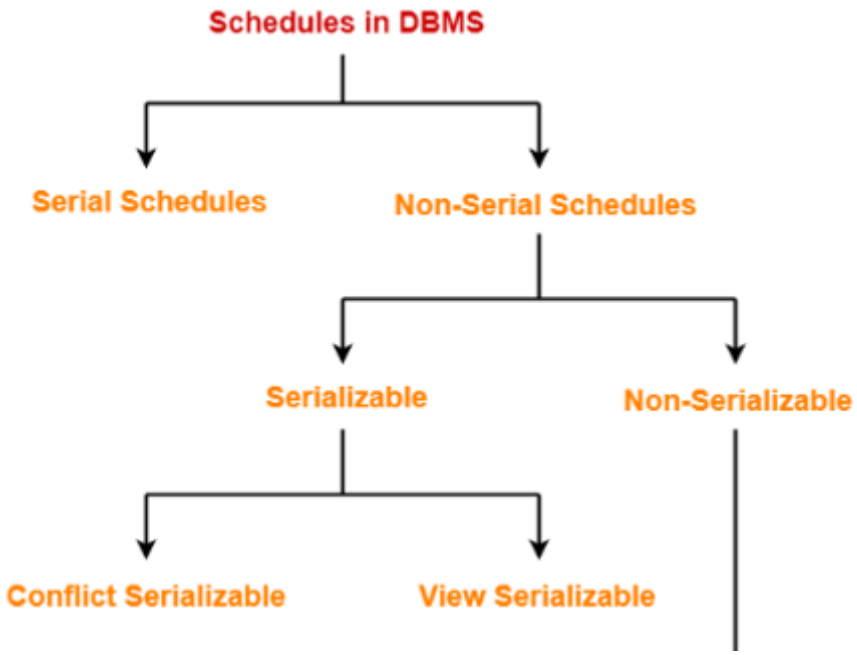
Failed state

- If any of the query made from the database system fails then the transaction is said to be in 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 transaction reached failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state .

Schedule

Transactions are set of operations on database. If multiple transactions are running concurrently, we need to perform the only one operation on the database. This sequence of operations is known as Schedule.



1. Serial Schedule

In **Serial schedule**, one transaction is executed completely before execution of another transaction. This type of execution also known as **non-interleaved** execution

```
T1      T2
----    ----
R(A)
R(B)
W(A)
commit
        R(B)
        R(A)
        W(B)
        commit
```

2. Non-serial Schedule

In non-serial schedules,

- Multiple transactions execute concurrently.
- Operations of all the transactions are inter leaved or mixed with each other.

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

a. Serializable schedule

The non-serial schedule is said to be in a serializable schedule only when it is equivalent to the serial schedules, for an n number of transactions. Here, multiple transactions can execute concurrently

Serial Schedules	Serializable Schedules
No concurrency is allowed. Thus, all the transactions necessarily execute serially one after the other.	Concurrency is allowed. Thus, multiple transactions can execute concurrently.

Conflict Serializable: A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

View Serializable: To check whether a given schedule is view serializable, we need to check whether the given schedule is **View Equivalent** to its serial schedule.

Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions:

- Initial Read: Initial read of each data item in transactions must match in both schedules.
- Final Write: Final write operations on each data item must match in both the schedules.
- Update Read For example, In schedule S1, T1 performs a read operation on X after the write operation on X by T2 then in S2, T1 should read the X after T2 performs write on X.

4. Non-Serializable Schedules

A non-serial schedule which is not serializable is called as a non-serializable schedule.

A non-serializable schedule may or may not consistent and may or may not recoverable.

Concurrency Control:

- Concurrency control enables the multiple users to access shared database at the same time.
- When multiple users(transaction) are accessing the database at the same time, and at least one is updating data, there may be the chance of conflicts, which can result in data inconsistencies.

Ex: Assume that two people who go to two different electronic machine at the same time to buy a movie ticket for the same movie and the same show time.
However, there is only one seat left in for the movie show in that particular theatre. Without concurrency control, it is possible that both person will end up purchasing a ticket. However, concurrency control method does not allow this to happen. Concurrency control only provides a ticket to the buyer who has completed the transaction process first.

We have concurrency control protocols to ensure atomicity and isolation of concurrent transactions.

Problems of concurrency control

Following are the three problems in concurrency control.

1. Lost updates
2. Dirty read
3. Unrepeatable read(phantom read)

1. Lost update problem

- If two transactions T1 and T2 read a data and then update the data one after another , the data updation of T1 will be overwritten by the T2.It will leads to Lost update problem.(i.e.update made by one transaction is lost here)

Transaction-X	Time	Transaction-Y
——	t1	——
Read A	t2	——
——	t3	Read A
Update A	t4	——
——	t5	Update A
——	t6	——

At time t2, transaction-X reads A's value.

- At time t3, Transaction-Y reads A's value.
- At time t4, Transactions-X writes A's value on the basis of the value seen at time t2.
- At time t5, Transactions-Y writes A's value on the basis of the value seen at time t3.
- So at time t5, the update of Transaction-X is lost because Transaction Y overwrites it without looking at its current value.
- Such type of problem is known as Lost Update Problem as update made by one transaction is lost here.

2. Dirty Read(unpredictable read)

- Let consider this scenario. Transaction Y update A data.and transaction X read the data A from database. But transaction Y is not complete due to system failure so the previous value of data A rollback into the database. Now the transaction hold the value of data A which is incorrect. This is called dirty read.

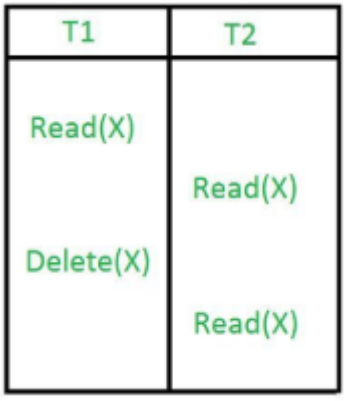
Transaction-X	Time	Transaction-Y
——	t1	——
——	t2	Update A
Read A	t3	——
——	t4	Rollback
——	t5	——

At time t2, transaction-Y writes A's value.

- At time t3, Transaction-X reads A's value.
- At time t4, Transactions-Y rollbacks. So, it changes A's value back to that of prior to t1.
- So, Transaction-X now contains a value which has never become part of the stable database.

3.Phantom Read Problem:

The phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.



In the above example, once transaction 2 reads the variable X, transaction 1 deletes the variable X without transaction 2's knowledge. Thus, when transaction 2 tries to read X, again it is not able to it.

Concurrency Control Protocol

Concurrency control protocols ensure atomicity and isolation of concurrent transactions. The concurrency control protocol can be divided into 3 categories:

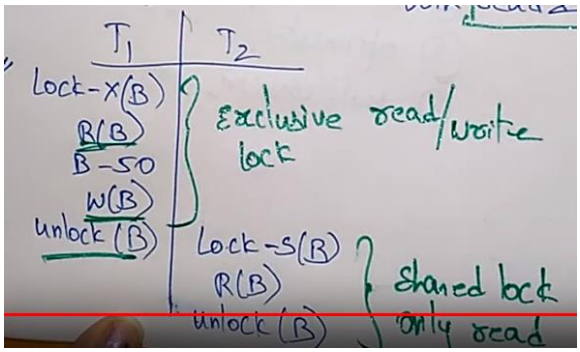
1. Lock based protocol

A) Shared lock(lock-s)

Transaction can only perform read operation on data items

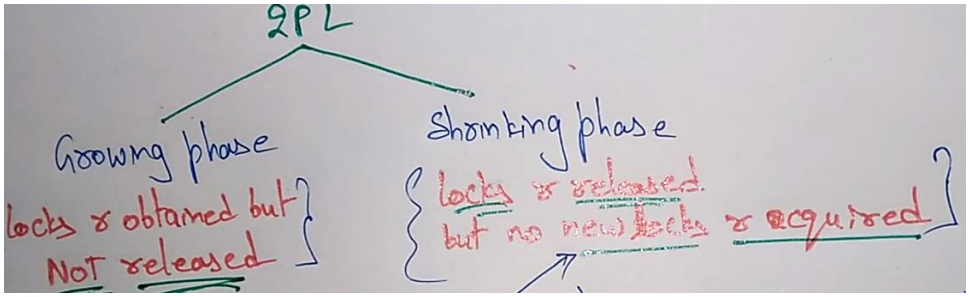
B) Exclusive lock(lock-x)

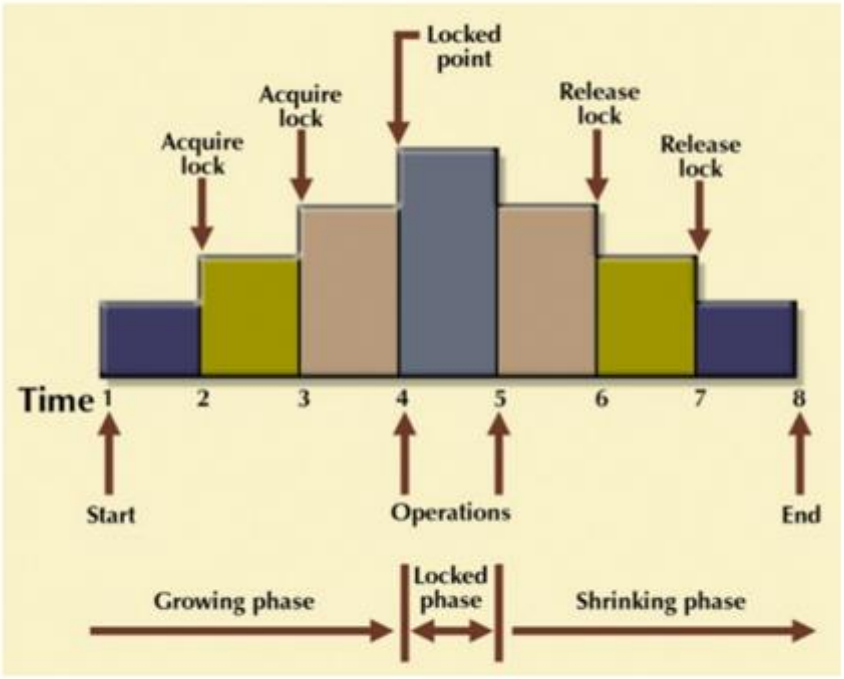
Transaction can perform read and write operation on data items.



2. Two phase locking protocol

- **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

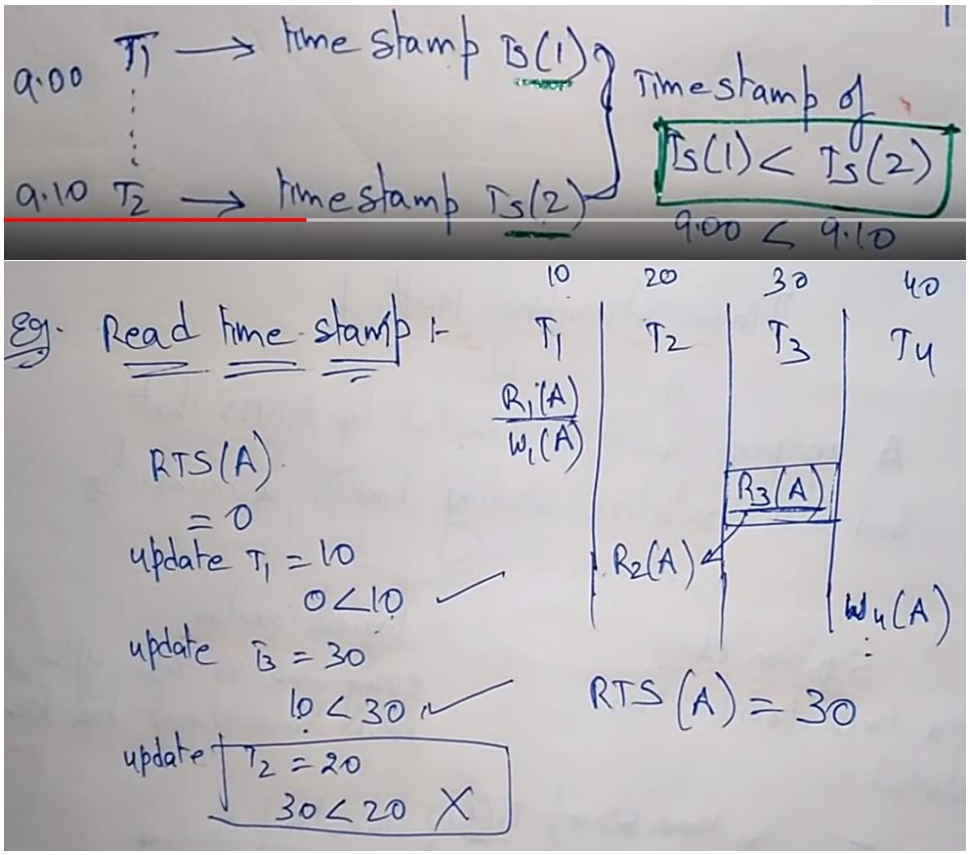




3. Time-stamp protocol

A unique identifier created by DBMS which indicate the relative starting time of each transaction.

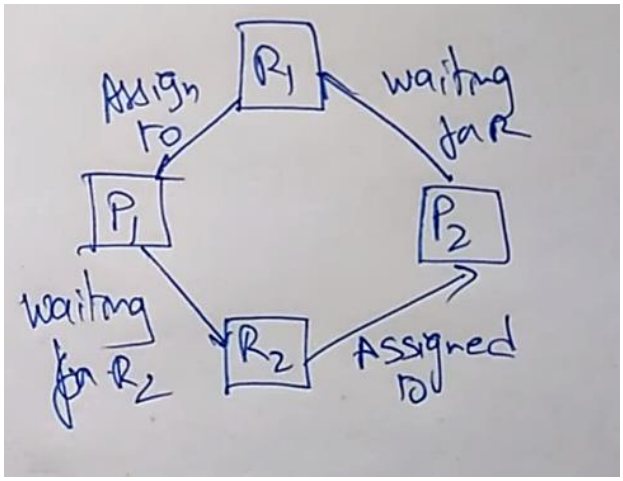
Condition: **Ts(1)<Ts(2)**

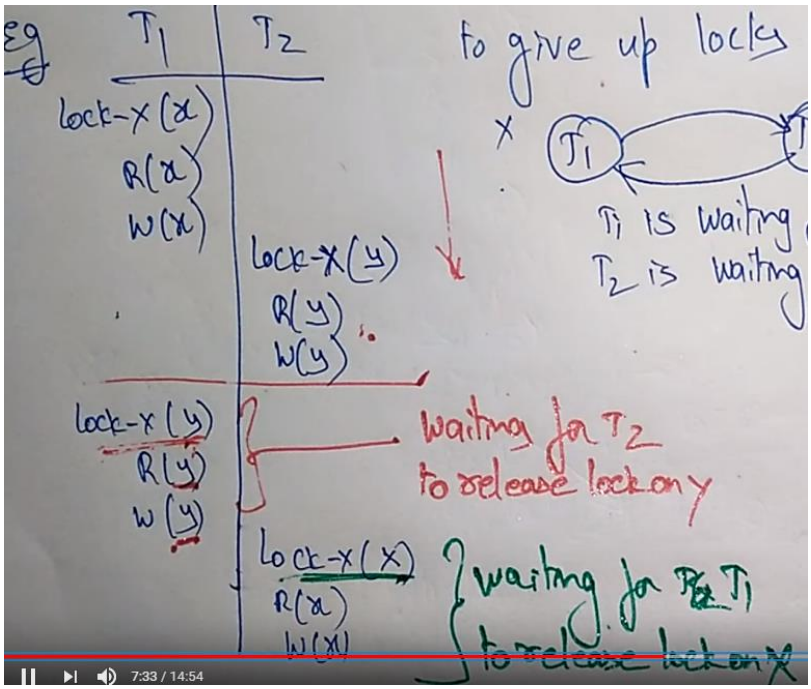


Dead lock in DBMS:

It is a situation in which Two or more transaction hold lock and waiting for one another to release locks. Real life example:

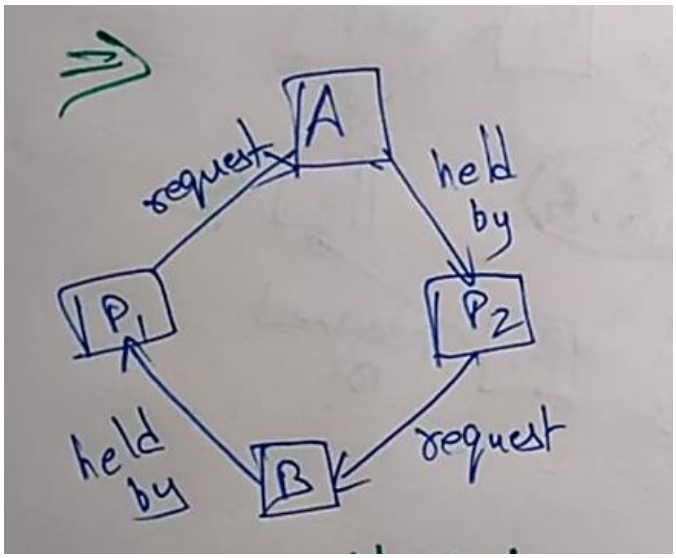
A) 2 person call each other at a same time. result: both are getting busy state.
B) Exam paper and pen: person A have pen and person B have paper. To complete the exam successfully Person A need paper and Person B need pen.so, this is not possible .so dead lock occur.



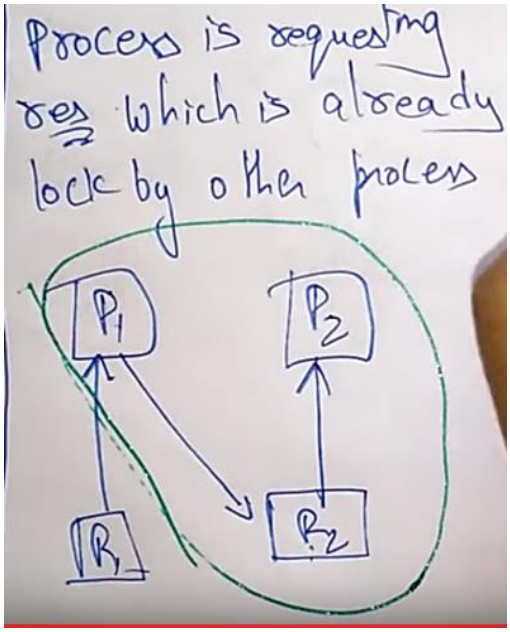


Necessary condition for Dead lock:

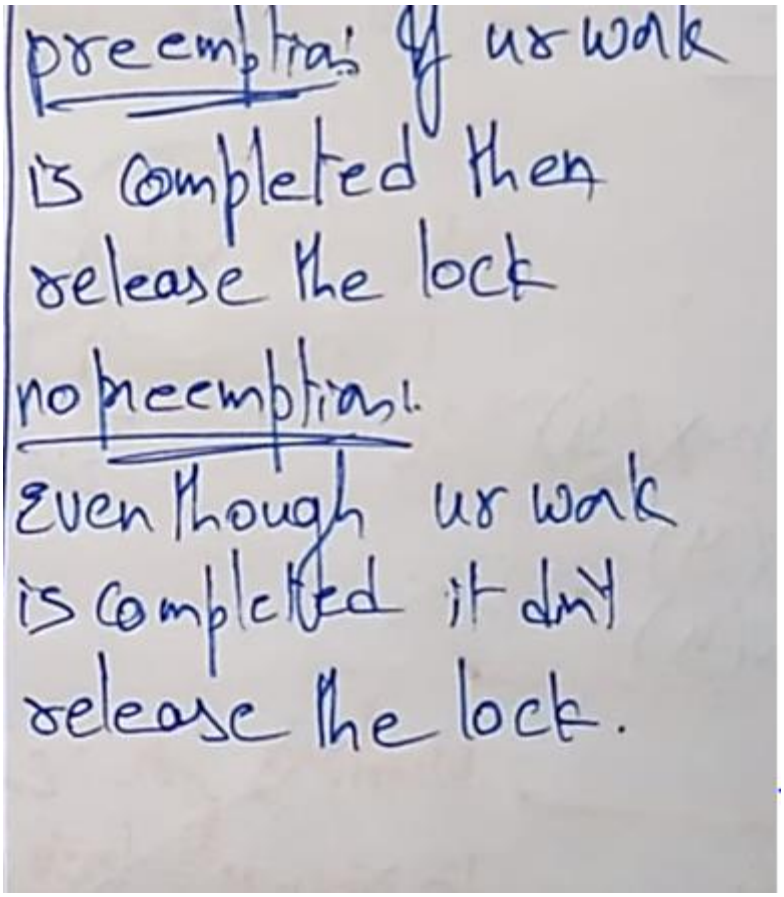
1. Hold and wait



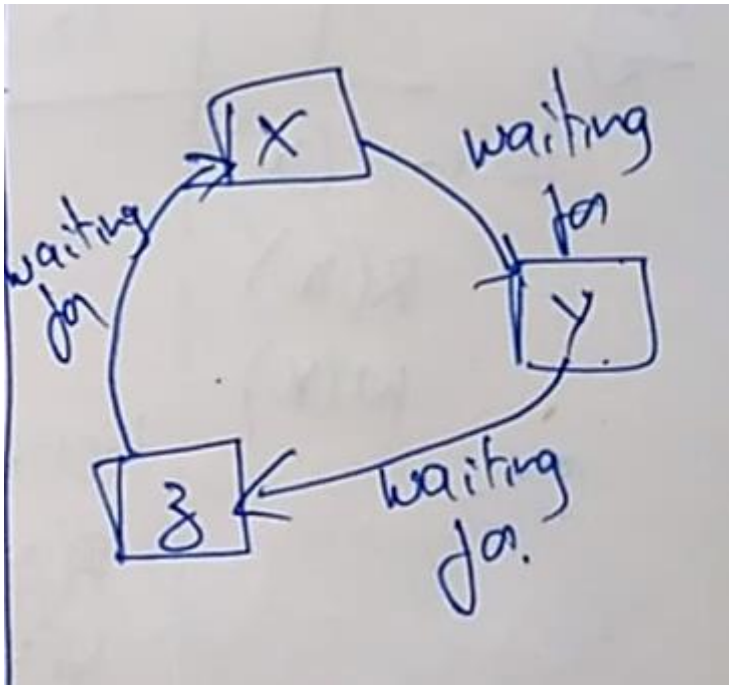
2.Mutual exclusion:



3.No Pre-emption:



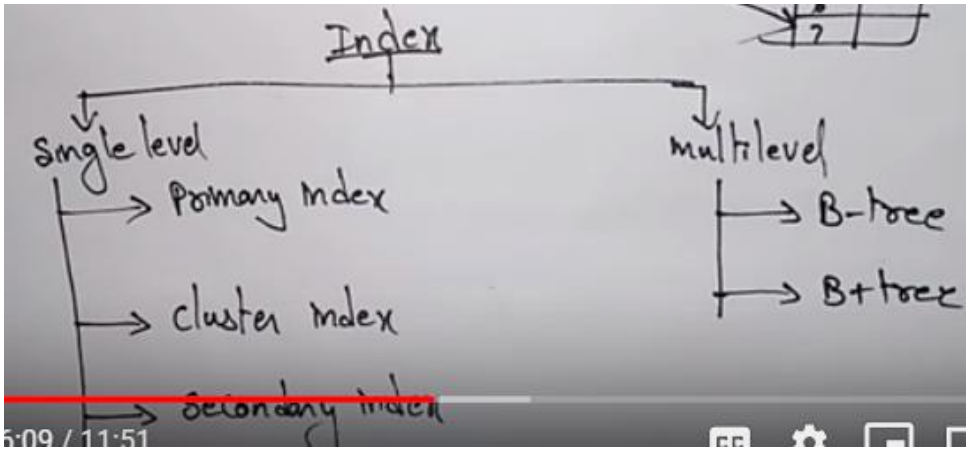
4.circular wait:



Starvation:

Starvation can be best explained with the help of an example – Suppose there are 3 transactions namely T1, T2, and T3 in a database that are trying to acquire a lock on data item ' I '. Now, suppose the scheduler grants the lock to T1(maybe due to some priority), and the other two transactions are waiting for the lock. As soon as the execution of T1 is over, another transaction T4 also come over and request unlock on data item 'I'. Now, this time the scheduler grants lock to T4, and T2, T3 has to wait again. In this way if new transactions keep on requesting the lock, T2 and T3 may have to wait for an indefinite period of time, which leads to **Starvation**.

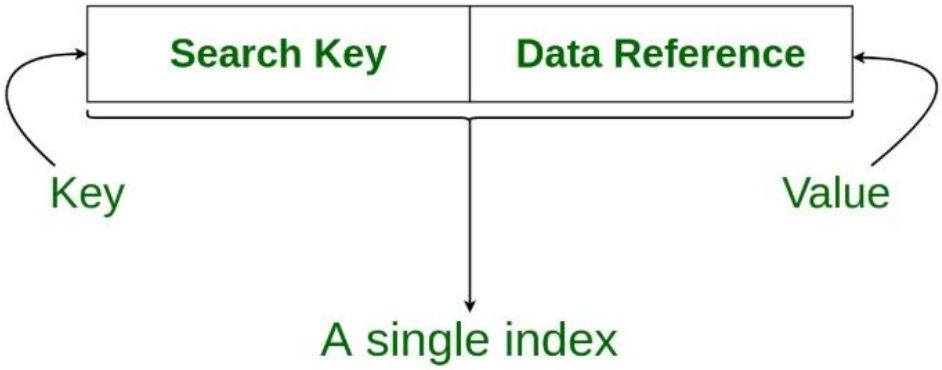
INDEXING



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

Ex. Indexing in database systems is similar to what we see in books.

Structure of a index:



Type of index:

- Single level index
- Multi-level index

Single level index:

Here index table not divided into further small index.

Primary (ordered) Index –

– Primary index is defined on an ordered data file. The index is created on primary key column.

Types of Primary index:

- Dense Index
- Sparse Index

Dense Index

In dense index, there is an index row for every primary key value in the database. This makes searching faster but requires more space to store index records itself. Each Index row contain primary key value and a pointer to the actual record on the disk.

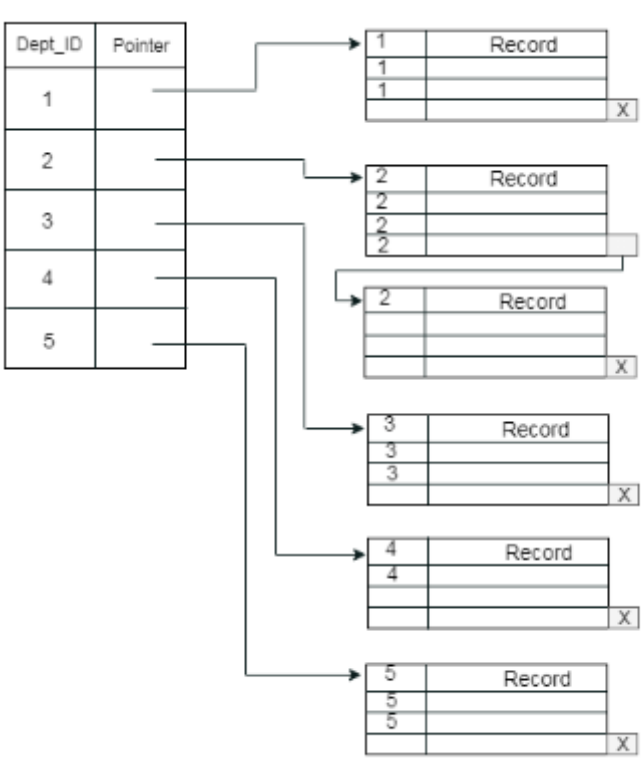
Sparse Index

In sparse index, index records are not created for every primary key. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not found in index record, we directly reach by following the index, then the system starts sequential search until the desired data is found

Clustering Index

- A clustered index can be defined as an ordered data file. The index record is created on non-primary key column.
- The rows 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.

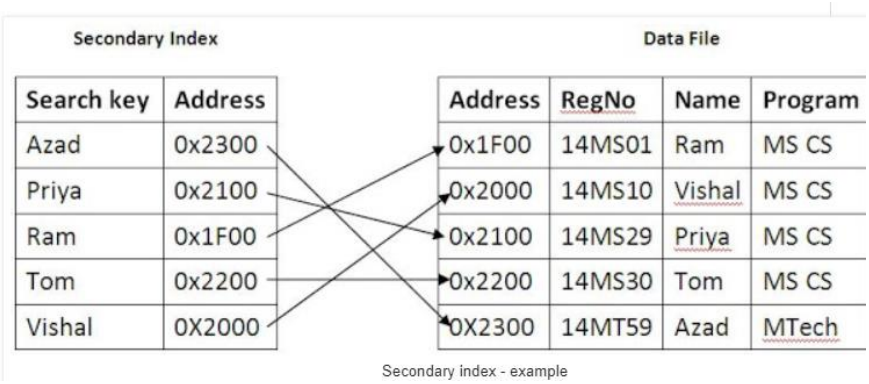


Secondary:

– Secondary index generated by using candidate key.

Secondary index enables accessing data file without using primary key. Secondary index must be a dense index(index defined on each row). Order of the search key values in the index file and the actual records are in different order.

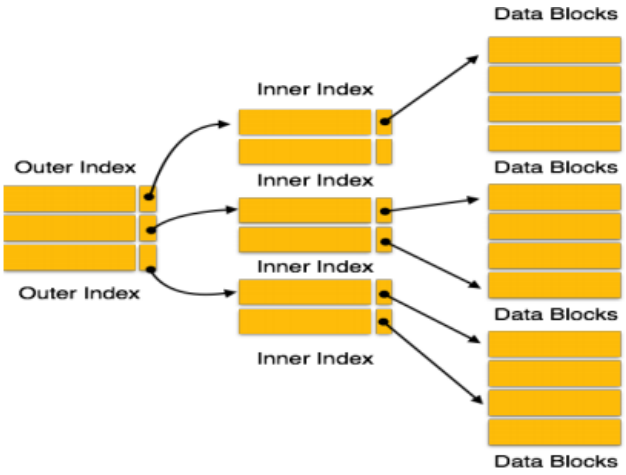
- A file can have several secondary indexes.



Secondary index - example

Multilevel Index

When stored large number of data it may result to large number of index. With help of multi index we can overcome this problem. In Multi-level Index ,the actual index table divided into several smaller index table in order to make the outermost level as small which can be saved in a single disk block.so that index table can easily be accommodated anywhere in the main memory.



B tree

A B-Tree of order m can have at most m-1 keys and m children.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

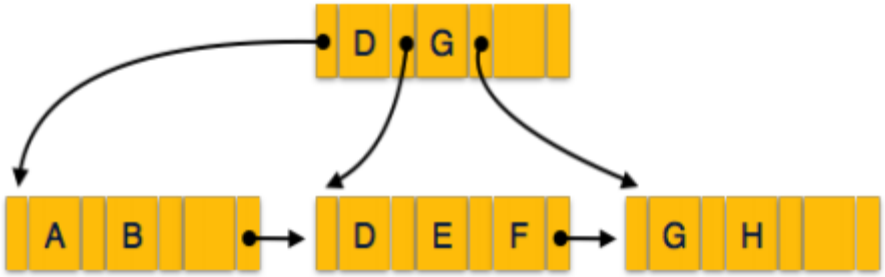
1. Every node in a B-Tree contains at most m children.
2. Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
3. The root nodes must have at least 2 children nodes.
4. All leaf nodes must be at the same level.

B+ Tree

A B+ tree is a balanced binary search tree that follows a multi-level index format. The leaf nodes of a B+ tree denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height, thus balanced.

Additionally, the leaf nodes are linked using a link list; therefore, a B+ tree can support random access as well as sequential access.

Structure of B+ Tree



Every leaf node is at equal distance from the root node. A B+ tree is of the order n where n is fixed for every B+ tree.

Internal nodes –

- Internal (non-leaf) nodes contain at least ⌈n/2⌉pointers, except the root node.
- At most, an internal node can contain n pointers.

Leaf nodes –

- Leaf nodes contain at least ⌈n/2⌉record pointers and ⌈n/2⌉key values.
- At most, a leaf node can contain n record pointers and n key values.
- Every leaf node contains one block pointer P to point to next leaf node and forms a linked list.

B Tree VS B+ Tree

SN	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Deletion of internal nodes are so complicated and time consuming.	Deletion will never be a complexed process since element will always be deleted from the leaf nodes.
5	Leaf nodes can not be linked together.	Leaf nodes are linked together to make the search operations more efficient.

Imp points from above table: 1,2,5

Note:

B+ tree widely used in data base system.

Avl tree, red black tree not used in data base system. Because they have growth in depth wise not in breath wise.so, they take more accessing time. B , B+ tree grow breadth wise and they take less access time compare to other tree data structure.

Recovery management system

Atomicity is property of Transaction .It describes transaction as a unit.so, every transaction has only two state either done or not done. For maintaining this property DBMS used two types of recovery system.

1. Log based recovery
2. Using checkpoint (used for concurrent transactions)

1. Log-based Recovery

The log is a sequence of records (row). Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.

2. Checkpoint

Recovery with Concurrent Transactions

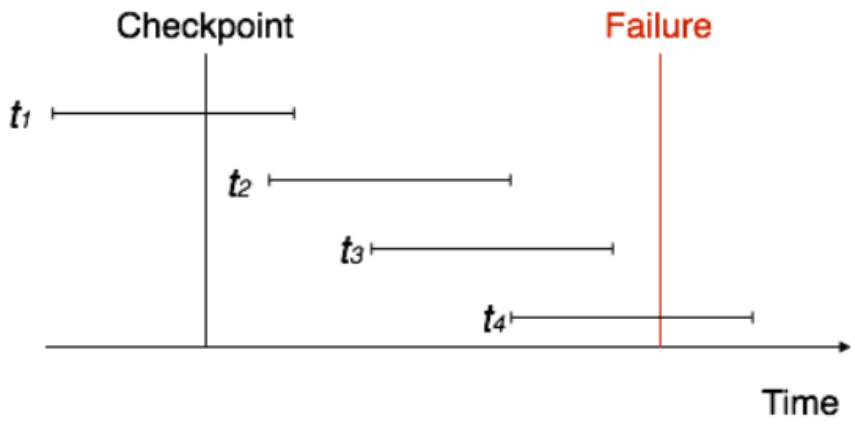
When more than one transaction are being executed in parallel, the logs are interleaved (so, logs not completed for particular transaction). At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering.

And also whenever transaction happened it stored in the log file. Because, in the long run the log file may grow too big. Keeping and maintaining logs in real time may fill out all the memory space available in the system.

To overcome this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint points to a point where all the previous transactions were completed successfully.

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



- The recovery system reads the logs backwards.
- It maintains two lists, an undo-list and a redo-list.
- If the recovery system sees a log with <T_n, Start> and <T_n, Commit> or just <T_n, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <T_n, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are undone and their logs are removed. All the transactions in the redo-list redone and their logs are saved.

All the best...

With