Functional Dependencies

What is a Functional Dependency?

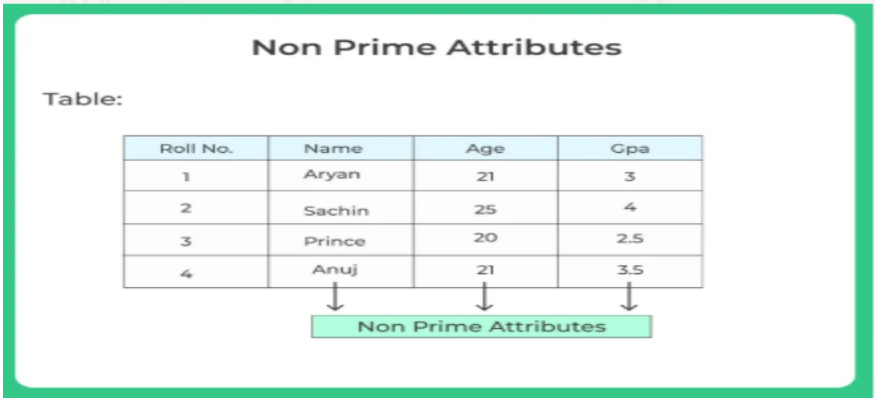
If one attribute is determined by another attribute in a DBMS system then it is a functional dependency.

Functional Dependencies

Functional Dependency plays an important role to find the difference between good and bad database design.

A functional dependency is denoted by an arrow →

The functional dependency of A on B is represented by A →B



Examples :

Consider the following table:

ID	NAME	AGE	CITY	PHONE_NO
67	Luke	22	Delhi	8353355366
68	Haley	19	Noida	7323598665
69	Luke	21	Mumbai	8656864365

- [ID] : Every individual will have a ***unique ID*** so it is a prime attribute.
- [NAME] : Two different persons ***might have same name***, so it is non prime attribute
- [AGE] : Two different persons ***might have same age***, so it is non prime attribute
- [CITY] : Two different persons ***might be living in the same city***, so it is non prime attribute

- [PHONE_NO] : ***No two persons will have same phone number***, so it is not a non prime attribute

Rules of Functional Dependencies

- Reflexive rule :. If A is a set of attributes and B is subset of A, then A holds a value of B.
- Augmentation rule : When A -> B holds, and C is attribute set, then AC->BC also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule : This rule is very much similar to the transitive rule in algebra if A->B holds and B->C holds, then A->C also holds. A->B is called as functionally that determines B.

Multivalued Dependency

- Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table.
- A multivalued dependency is a complete constraint between two sets of attributes in a relation.

Example:

Car_Model	Manufactre_Year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

- In this example, Manufacture_year and colour are independent of each other but dependent on Car_model. In this example, these two columns are said to be multivalue dependent on Car_model.
- This dependence can be represented like this:
    - Car_model -> Manufacture_year

- Car_model-> colour

Trivial Functional dependency

- The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.
- A->B is a trivial functional dependency if B is a subset of A.

Example:

Stu_ID	Stu_Name
3015	Priya
1402	Rajesh
2109	Ganga

In the above table,{Stu_ID, Stu_Name} -> Stu_ID is a trivial functional dependency as Stu_ID is a subset of {Stu_ID,Stu_Name}.

Non trivial functional dependency

- In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.
- A nontrivial dependency occurs when A->B holds true where B is not a subset of A.

Example:

Stu_ID	Stu_Name	Marks
3015	Priya	97
1402	Rajesh	83
2109	Ganga	89

- {Stu_ID} -> {Stu_Name} (if we know the Stu_ID, we knows the Stu_Name)
- But Stu_Name is not a subset of Stu_ID, and hence it’s non-trivial functional dependency.

Transitive dependency

A transitive is a type of functional dependency which happens when it is indirectly formed by two functional dependencies.

Example:

Stu_ID	Stu_Name	Marks
3015	Priya	97
1402	Rajesh	83
2109	Ganga	89

- {Stu_ID} -> {Stu_Name} (if we know the Stu_ID, we know its Stu_Name)
- {Stu_Name} -> {Marks} If we know the Stu_Name, we know the Marks
- Therefore according to the rule transitive dependency: {Stu_ID} -> {Marks} should hold, that makes sense because if we know the Stu_ID, we can know his/her marks.

Advantages of Functional Dependency

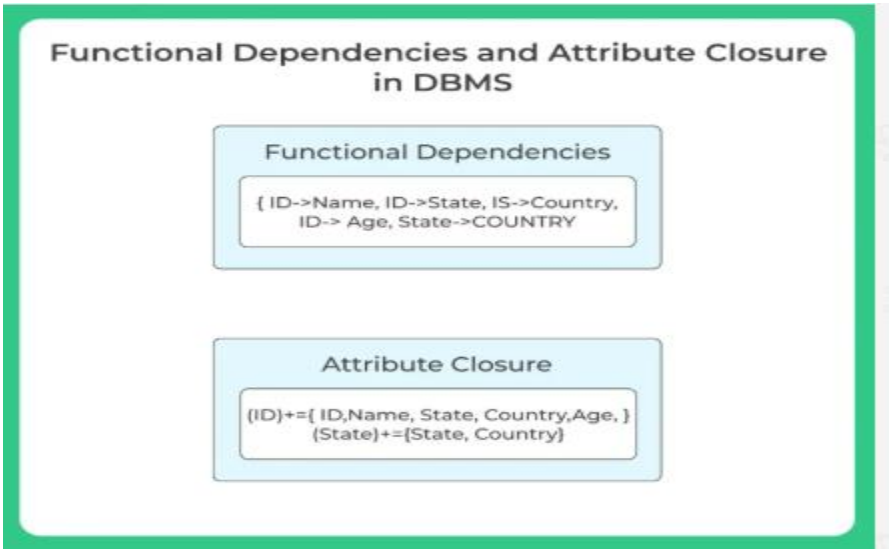
- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to define meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

Functional dependency and Attribute Closure in DBMS

Functional dependency and Attribute Closure in DBMS

- A Relation A->B is said to be a functional dependency whenever two tuples are having the same value for both attributes A and attribute B

- Simply Functional dependency is a relationship that exists when one attribute uniquely determines another attribute
- Consider an example of table student
    - Id->name, id->addr are functional dependencies, where as name->addr is not a functional dependency.



Finding functional dependencies in a relation (table)

Functional dependencies in a relation are dependent on the domain (set of possible values that a column can accept) in that table.

Consider the table student

ID	Name	Age	Address	Country	State
65	Priya	20	Chennai	India	Tamil Nadu
66	Rishi	21	Mumbai	India	Maharashtra
67	Keerthi	19	Delhi	India	Delhi
68	Amitha	20	Chennai	India	Tamilnadu

- As we know, ID is capable of defining uniquely each record in the table hence ID-> Name, ID-> Address, ID->Age hold true

- In the same way State ->Country will also be true because there are chances for two rows to have the same state then they will have the same country as well.

Functional dependency set

- Functional dependency set is nothing but the set of all functional dependencies that are existed in a table or relation
- For example, in the table student, we can observe the functional dependencies as
    - {ID ->Name, ID ->State, ID-> Country, ID-> age, State-> Country}

Attribute closure

The set "A*" is said to be the closure set of "A" if the set of attributes are functionally dependent on the attributes of "A"

Finding attribute closure of an attribute set

- First, add elements of the attribute set to the result set
- Add elements to the result set which are functionally determined from the result set recursively

In the relation student an attribute closure is determined as

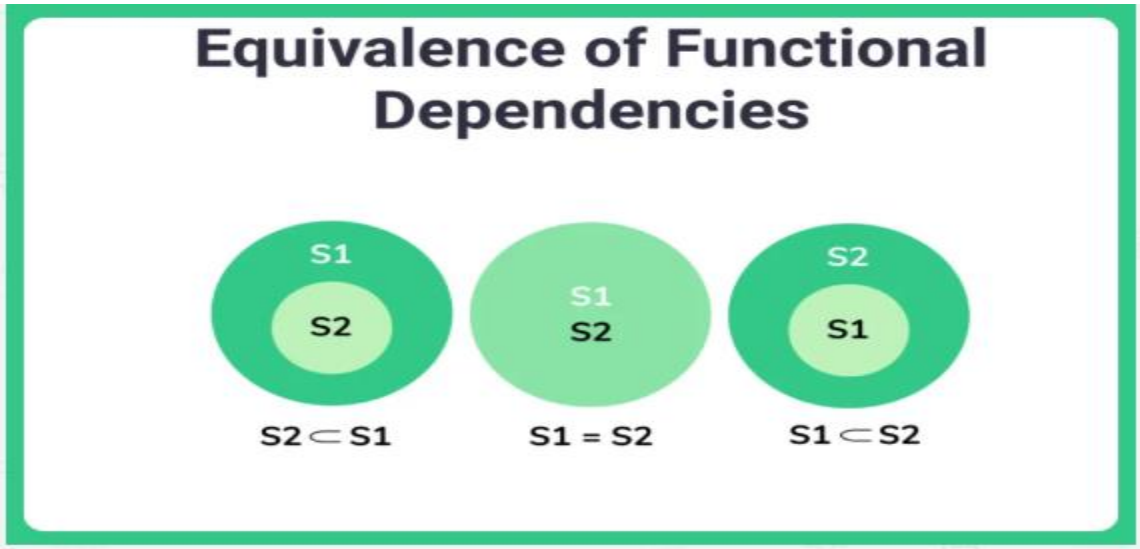
- (ID)+ = {ID, Name, State,Country,Age}
- (State)+ = {State, Country}

Finding candidate keys and super keys through attribute closure

- If all attributes of the relation are present in attribute closure of an attribute set then such an attribute set will have a super key for that relation
- If all attributes of the relation or determined by none of the attribute set then such a set will be a candidate key as well
    - (ID, Name)+ = {ID, Name, State, Country, Age}
    - (ID)+ = {ID, Name,State,Country, Age}
- (ID, Name) will be super key but not as a candidate key because its subset (ID) + is equal to all attributes present in the relation. Hence, ID will be a candidate key.

Equivalence of functional dependencies in DBMS

- Equivalence of functional dependencies is nothing but we need to determine whether all functional dependencies existed for the relation are equal or not
- You will be given a relation with different functional dependency sets of that relationship you need to check whether one functional dependency is a subset of other or both are equal.



Finding a relationship between two to FD (functional dependency) sets

Consider S1 and S2 are two FD sets for a relation R.

1.
    1. ***If all FDs of S1 can be determined from FDs that are present in S2***, we can conclude that ***S2 ⊃ S1***.
    2. ***If all FDs of S2 can be determined from FDs that are present in S1***, we can conclude ***FD1 ⊃ FD2***.
    3. If ***1 and 2 are satisfied*** then, ***S1=S2.***

Check for the equivalence of functional dependencies for A relation R (A, B, C, D) having two FD sets S1 = {A->B, B->C, AB->D} and S2= {A->B, B->C, A->C, A->D}

Solution:

- Step 1. Checking whether all FDs of set S1 is present in set S2
    - FD A->B in set S1 is present in set S2.
    - FD B->C in set S1 is also present in set S2.
    - FD AB->D in present in set S1 but not in S2 but we have to check whether we can derive it or not. In set S2, (AB) + = {A, B, C, D}. It means that A,B,C,D can be determined functionally from AB which implies AB->D holds true for set S2
    - As all FDs in set S1 also hold in set S2, S2 ⊃ S1 is true.

- Step 2. Checking whether all FDs of S2 are present in S1
    - FD A->B in set S2 is present in set S1.
    - FD B->C in set S2 is also present in set S1.
    - FD A->C is present in S2 but not in S1 but we have to check whether we can derive it or not. In set S1, (A) + = {A, B, C, D}. It means that A,B,C,D can be determined functionally from A which implies A->C holds true for set S1.
    - FD A->D is present in S2 but not in S1 but we have to check whether we can derive it or not. In set S1, (A) + = {A, B, C, D}. It means that A,B,C,D can be determined functionally from A which implies A->D holds true for set S1.
    - As all FDs in set S2 also hold in set S1, S1 ⊃ S2 is true.

- Step 3. We can conclude that S1=S2 as both S2⊃ S1 and S1⊃ S2 are true. Hence these two FD sets are semantically equivalent.

Canonical Cover in DBMS

Features of Canonical cover

- The canonical cover ***is free from all irrelevant functional dependencies***
- The ***closure of canonical cover is the same as that of a given set of functional dependency***

- The ***canonical cover is not unique*** because there may be ***more than one chemical cover for a given set of functional dependency.***

What's the need for this Canonical cover

Reduces computational time: When a functional dependency is free from useless dependency e then computational time and working with that set becomes very easy.

PRACTICE PROBLEM BASED ON FINDING CANONICAL COVER-

You will be given some 3 to 5 functional dependencies and asked to find the simplified version that is Canonical cover

Change WXYZ to ABCD

Problem :

The following functional dependencies hold true for the relational scheme R (W, X, Y, Z) –

$X \rightarrow W$

$WZ \rightarrow XY$

$Y \rightarrow WXZ$

Write the irreducible equivalent for the given set of functional dependencies.

Solution-

STEP 1: Write all the functional dependencies such that each dependency contains only one attribute on its right side i.e-

$X \rightarrow W$

$WZ \rightarrow X$

$WZ \rightarrow Y$

$Y \rightarrow W$

$Y \rightarrow X$

$Y \rightarrow Z$

STEP 2: Check each functional dependency one by one whether it is essential or not

- For X → W:

Including X → W, (X) + = {X, W}

Ignoring X → W, (X) + = {X}

Now, as the the two results are different, it is concluded that X→W is essential and cannot be eliminated.

- For WZ → X:

Including WZ → X, (WZ) + = {W, X, Y, Z}

Ignoring WZ → X, (WZ) + = {W, X, Y, Z}

Now, as the the two results are same, it is concluded that WZ→X is not essential and can be eliminated.

After eliminating WZ→X ,the resultant set of functional dependencies is :

X → W

WZ → Y

Y → W

Y → X

Y → Z

Now, we will consider this reduced set in further checks.

- For WZ → Y:

Including WZ → Y, (WZ) + = {W, X, Y, Z}

Ignoring WZ → Y, (WZ) + = {W, Z}

Now, as the the two results are different, it is concluded that WZ→Y is essential and cannot be eliminated.

- For Y → W:

Including Y → W, (Y) + = {W, X, Y, Z}

Ignoring Y → W, (Y) + = {W, X, Y, Z}

Now, as the the two results are same, it is concluded that Y→W is not essential and can be eliminated.

After eliminating Y→W,the resultant set of functional dependencies is :

X → W

WZ → Y

Y → X

Y → Z

- For Y → X:

Including Y → X, (Y) + = {W, X, Y, Z}

Ignoring Y → X, (Y) + = {Y, Z}

Now, as the the two results are different, it is concluded that Y→X is essential and cannot be eliminated.

- For Y → Z:

Including Y → Z, (Y) + = {W, X, Y, Z}

Ignoring Y → Z, (Y) + = {W, X, Y}

Now, as the the two results are different, it is concluded that Y→Z is essential and cannot be eliminated.

- From here, our essential functional dependencies are-

X → W

WZ → Y

Y → X

Y → Z

Normalization in DBMS

What is Normalization ?

We define Normalization as a step-by-step process via which we reduce the complexity of a database. The database may have a lot redundancies (unnecessary information that has become outdated or no longer useful) like attendance record for former students in a university.

Normalization in DBMS

A lot of anomalies may be caused in a database because of various cause **insertion, deletion and updation operation or concurrency errors**.

There are 6 different normalisation benchmarks used which are –

- 1 NF (normal form)
- 2 NF
- 3 NF
- BCNF (Boyce-Codd Normal Form
- 4NF
- 5NF (not used anymore)
- 6NF (not used anymore)

1NF

Name – 1 Normal Form

Prerequisite –

The relation **should not have any multivalued attribute** at all in it. For example, for Student table in the database (relation). In the phone number column there must only be one phone number (value).

How to solve –

Decompose the table into single values attributes. Either the second phone number must be deleted or it can be decomposed into primary and secondary number.

Not 1 NF

ID	FName	Phone_Number
1	Bran	915988589, 989458602
2	Sansa	790614709
3	Jon	700293958

Is 1 NF

ID	FName	Primary_No	Secondary_No
1	Bran	915988589	989458602
2	Sansa	790614709	–
3	Jon	700293958	–

Is 1 NF

ID	FName	Phone_Number
1	Bran	915988589
2	Sansa	790614709
3	Jon	700293958

2NF

To understand the conditions for 2NF, one must know the following –

- What is non prime attribute
- What is an candidate Key

Candidate Key – A candidate key is a column, or set of columns, in a table that can uniquely identify any database record without referring to any other data. The best set of columns which identity it uniquely is chosen as candidate key.

Non Prime Attribute – Is the row that doesn't belong to the candidate key for the table.

Example –

In the table below, if you want to identify any row of the table uniquely, we need both {StudentID, CourseID} thus, it becomes candidate key.

And since CourseName is not part of candidate key. It becomes non-prime attribute.

Enrolled Courses Table

Student_ID	Course_ID	Course_Name
1	CSE101	C
1	CSE102	C++
2	CSE101	C
2	CSE102	C++
3	CSE103	DBMS

What is 2NF Finally?

To be 2NF in DBMS. The relation should be –

- Already 1NF
- No Non Prime attribute should be dependent on any candidate key element, which is called not having partial dependency. This maybe is difficult to understand from definitions but easier from example.

In table, the candidate key is {StudentID, CourseID}, the name of the course is obviously dependent on the CourseID right? CourseID, is non prime also. Thus there is partial dependency in it. Thus, the relation is not 2NF.

To make it 2NF we can break the table as follows –

Enrolled Courses Table

Student_ID	Course_ID
1	CSE101
1	CSE102

2	CSE101
2	CSE102
3	CSE103

Courses Table

Course_ID	Name
CSE101	C
CSE102	C++
CSE103	DBMS

3NF

For a relation to be in 3NF form the following must hold true –

- Should be 2NF already
- There should not be any transitive dependency for non prime attributes

Don’t worry if you don’t know what transitive dependency is, we will understand this with an example.

Student Table

Student_ID	Name	Age	State	Country
1	Bran	16	UP	India
2	Jon	21	UP	India
3	Arya	17	UP	India
4	Jamie	41	Gujarat	India
5	Tyrion	41	Gujarat	India

Now, for the table above. The primary key obviously is StudentID. However, the following dependencies are there –

1. StudentID -> Name

2. StudentID -> Age
3. **StudentID -> State**
4. **State -> Country**

Now, when we look at point 3 and 4 they form transitive dependency as they are of format A -> B and B -> C. Thus, this forms transitive dependency is not in 3NF.

BCNF

The BCNF is the short for Boyce-Codd Normal Form the following are the conditions for a relation to be in BCNF form –

- Should be 3NF
- If every non-trivial functional dependency A –> B, A is a super key, i.e. LHS should always be a super key.

Let's have a look onto an example to understand this better. Let's say a student enrolling a course can also register for courses from other branches and the course registration table for student is stored in the following way –

Course Registered Table

Student_ID	Name	CourseName	Branch_ID	Course_ID
1	Bran	C	CSE	101
1	Bran	Fluid Mechanics	MECH	101
1	Bran	C++	CSE	102
2	Jamie	C	CSE	101
2	Jamie	Basic Electronics	EEE	101

Candidate Key – {StudentID, CourseName}

- To identify each row uniquely we need {StudentID, CourseName}
- Thus, its the candidate key

Functional dependencies are –

1. StudentID -> Name
2. CourseName -> {BranchID, CourseID}

Now, neither the 2nd is a dependency of type X -> Y, nor CourseName is a super key. **Note –** Set of attributes that can help us uniquely identify a record or tuple in the database is called super key.

Student_ID	Name
1	Bran
1	Bran
1	Bran
2	Jamie
2	Jamie

Course Table

Course_Name	Branch_ID	Course_ID
C	CSE	101
Fluid Mechanics	MECH	101
C++	CSE	102
C	CSE	101
Basic Electronics	EEE	101

4NF

To be in 4NF form the follwing must be true

1. Should be in BCNF form
2. Should not have any Multi-Valued Dependency.

Example –

A -> BC

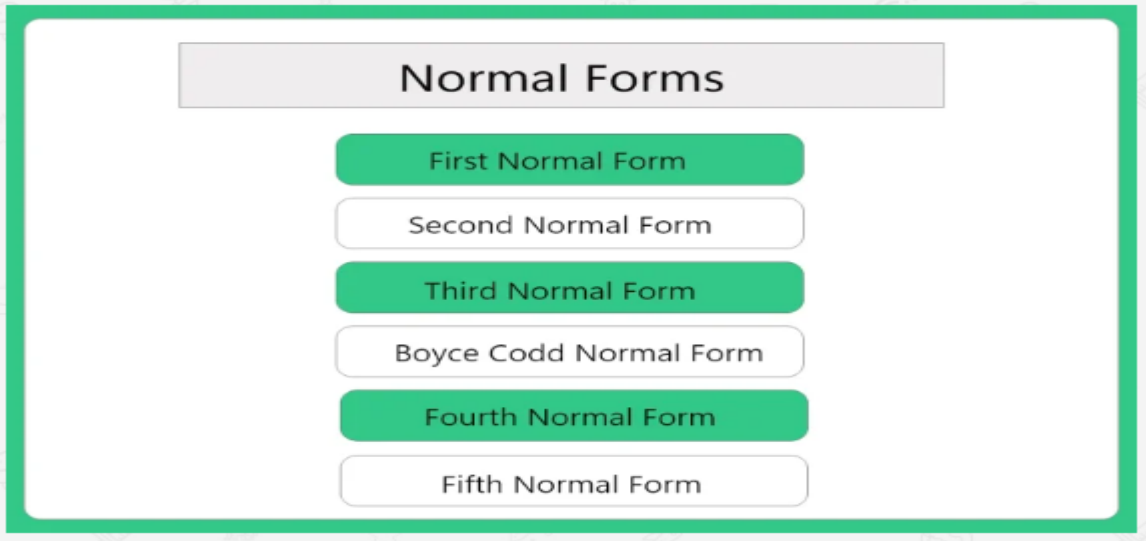
Normal Forms in DBMS

Normal Forms in DBMS

Will discuss how a table suffers from problems due to infection and updation and deletion in upcoming tutorials.

This normal form helps in

- Data integrity
- Eliminates Data redundancy
- As a result of data retrieval and data insertion, updation becomes smooth, faster, optimized



There are 5 of there are five different levels of Normal forms let's see a brief introduction about each normal form

First Normal Form(1NF)

-
  - A table is said to be in First Normal Form if it is free from multivalued or composite attributes i.e each attribute should be atomic
  - A multivalued attribute is nothing but it contains more than one value in a single cell.

Second Normal Form(2NF)

A table is said to be in Second Normal Form if it satisfies the following two properties

1. It must be in First Normal Form
2. It should be free from all kinds of partial dependencies.

Third Normal Form(3NF)

A table is said to be in Third Normal Form if it satisfies the following properties

1.
    1. If it is in second normal form and
    2. There is no transitive dependency.

BCNF(Boyce Codd Normal Form)

The table is said to be in BCNF if it satisfies the following properties

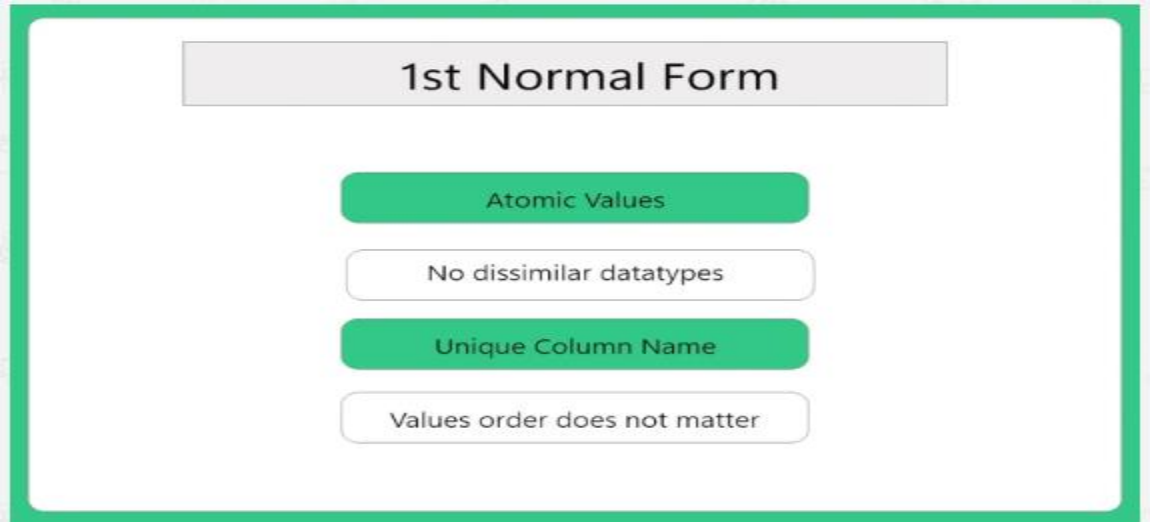
-
    - It should be in 3nf
    - For every functional dependency, A-> B, A must be a superkey i.e in any functional dependency LHS attribute must be a Superkey.

4NF(Fourth Normal Form)

The table is said to be in the 4th Normal Form it is it satisfies the following properties

-
    - It should be in Boyce Codd normal form(BCNF)
    - It is free from multivalued dependency

1-NF form in DBMS



First Normal Form(1NF)

- **Definition**: A table is said to be in First Normal Form if it is free from multivalued or composite attributes i.e each attribute should be atomic
- A multivalued attribute is nothing but it contains more than one value in a single cell

Problem Table:

Consider a table where a student can have multiple phone numbers

STUDENT

Roll no	Name	Phone
66	Trishaank	P1
73	Prashant	P2,P3
79	Sanjay	P4
82	Srinivas	P5

From the above figure, we can see that a ***cell is containing two values(two phone numbers)*** which breaks our rule This problem can be solved in three approaches

Approach 1:

Eliminate the ***multivalued attribute by*** **introducing** ***a composite key***i.e roll number, phone number combination would act as a primary key

Problem Table

Roll no	Name	Phone
66	Trishaank	P1
73	Prashant	P2,P3
79	Sanjay	P4
82	Srinivas	P5

Normalized to 1NF

Roll no	Name	Phone

66	Trishaank	P1
73	Prashant	P2
73	Prashant	P3
79	Sanjay	P4
82	Srinivas	P5

Drawback:

This approach causes serious drawback of update anomaly due to redundancy in the table (Prashant data stored redundantly)

Approach 2:

Divide the table into two parts such that all the ***multivalued attributes in one table at single-valued attributes in another table*** and add primary key attribute of the original table to each newly formed tables

Student Phone

Roll no	Phone
66	P1
73	P2
73	P3
79	P4
82	P5

Student

Roll no	Name
66	Trishaank
73	Prashant
79	Sanjay

82	Srinivas

In this example, multivalued attributes phone numbers and primary key attribute roll number will from one table and all the remaining single-valued attributes name and primary key attribute roll number will from another table

Approach 3:

Make each and every attribute as atomic i.e ***introduce n separate columns for n multivalues***

Problem Table

Roll no	Name	Phone
66	Trishaank	P1
73	Prashant	P2,P3
79	Sanjay	P4
82	Srinivas	P5

Normalized to 1NF

Roll no	Name	Phone1	Phone2	Phone3
66	Trishaank	P1	———	———
73	Prashant	P2	P3	———
79	Sanjay	P4	———	———
82	Srinivas	P5	———	———

Drawbacks:

It's it not necessary that each and every student would have multiple phone numbers, as a result, empty cells have to be maintained which results in memory wastage.

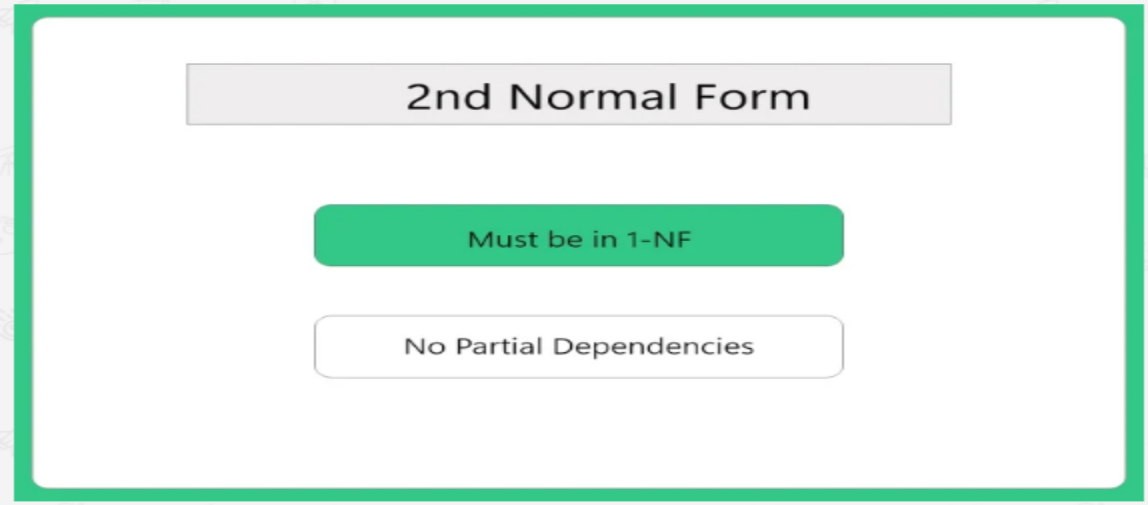
2-NF form in DBMS

2-NF form in DBMS

In this article, we will learn about the 2-NF form in DBMS.

Normalization is a process where a table is broken into further tables to ensure the best that they are free from problems due to insertion updation and deletion

Even if the table is in First Normal Form there are still the anomalies of insertion updation and deletion hence we go for second normal form.



Definition

A table is said to be in Second Normal Form if it satisfies the following two properties

1. It must be in First Normal Form 2. It should be free from all kinds of partial dependencies

Problem Table: Score table

student_id	subject_id	marks	teacher
191	13A	70	Java Teacher
191	15D	75	C++ Teacher
191	13A	80	Java Teacher

- The above table is in First Normal Form as it is free from multivalued attributes
- The combination of student_id and subject_id columns will become the primary key for the table.
- But the column teacher entirely depends on subject_id column and independent of student_id.
- So even if the table is in First Normal Form it suffers from problems due to insertion updation and deletion hence we need to go for second normal form.

Procedure to convert a table into 2-NF

- Identify the partial and full dependencies and apply decomposition rule
- In the above table column teacher is dependent on subject_id
- The simple solution is to create a separate table for subject_id and teacher and removing it from Score table
- Resulting tables which are in 2 normal form are

Score Table

student_id	subject_id	marks
191	13A	70
191	15D	75
191	13A	80

Subject Table

subject_id	teacher
13A	Java Teacher
15D	C++ Teacher

3-NF form in DBMS

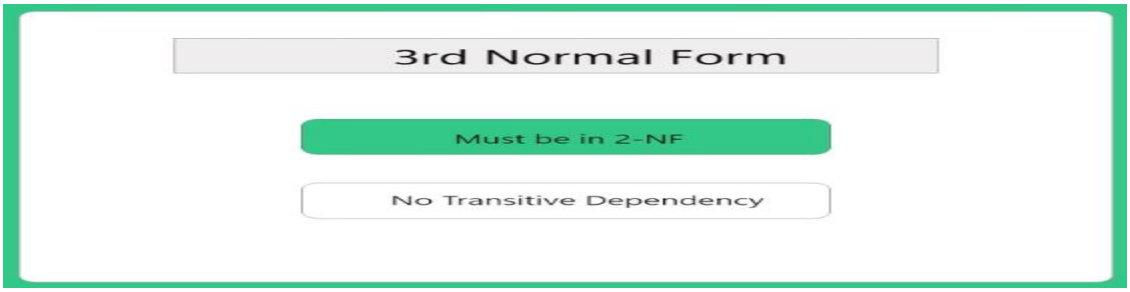
3-NF form in DBMS

Normalization is a process of making a table optimized by eliminating problems caused due to insertion updates and deletion.

Definition:

A table is said to be in Third Normal Form if it satisfies the following properties

1. If it is in second normal form and
2. There is no transitive dependency.



What is a transitive dependency?

When non-prime attribute depends on another non-prime attribute rather than depending upon the prime attributes or primary key

Employee table

Emp_id	Name	Project-id	Date-completion
64	Arun	101	20-4-21
65	Sindhu	102	30-4-21
66	Trishaank	101	20-4-21
67	Vamshi	103	10-2-21

- In the employee table, we have dependency as shown
  Project id -> date completion
- Project id rather than depending on the primary key of the table *employee ID* it is depending on a non-prime attribute *date completion*

The table is in First Normal Form because it does not have any multivalued attributes

The table is in the second form because there is no composite and no chances of any partial dependency.

But still, the stables office from the insert, update and delete anomalies

- Insert anomaly: Project ID cannot be assigned until an employee is assigned to it
- Update anomaly: Updating the values of date completion requires multiple updates because it is stored redundantly table
- Delete anomaly: The deletion of employee results in the deletion of entire project data even if there is even a single employee in that project the tables suffers from
- Even if the table is in second Normal Form it suffers from all the anomalies because of the presence of a transitive dependency as shown

Note: Table with any kind anomaly will have a transitive dependency.

How to eliminate transitive dependency?

Solution steps

- Transitive dependency is converted into a separate table
- The determinant of each transitive dependency becomes a primary key attribute of the corresponding table
- Add primary key of each table resulted from each transitive dependency as the foreign key in an original table along with all remaining attributes

we have a transitive dependency *[project id -> date completion]*

- Break the original employee table in two tables where one table project consists of transitive dependency second table employees consistently meaning attributes
- In a newly formed projects table would contain project ID, date completion and determinant project ID would serve as a primary key attribute of this newly formed table

Project table

Project_id	Date-Completion
101	20-4-21
102	30-4-21
103	10-2-21

Employee table

Emp_id	Name	Project-id
64	Arun	101
65	Sindhu	102
66	Trishaank	101
67	Vamshi	103

- In the newly formed employees, table project id is derived as a foreign key attribute from the original employee table along with the remaining attributes.

Now compare this newly formed tables the previous problem table

- Project ID could be added without the employee information

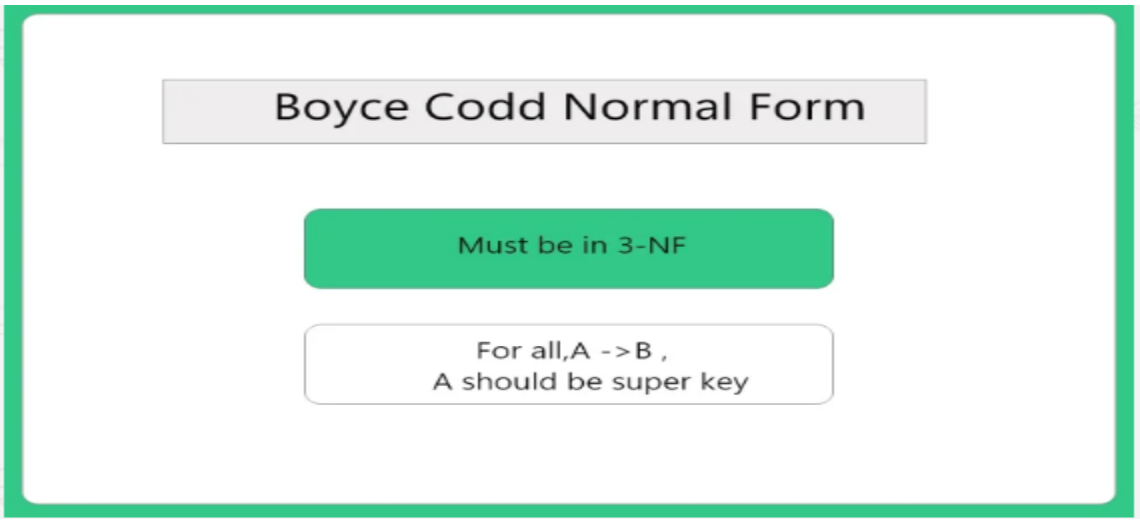
- Deletion of employee data would not result in the deletion of project data in the project table, updating the values of project-completion would not result in multiple updates because it is free from redundancy.

BCNF form in DBMS

BCNF form in DBMS

In this article, we will learn about the Boyce Codd Normal Form i.e. BCNF form in DBMS.

- Normalization is a process where the table is decomposed into further tables so that newly formed tables are free from all problems of insertion, deletion, and updation of data
- **BCNF**: Advanced version of 3nf and stricter then 3nf



Definition of Boyce Codd Normal Form:

The table is said to be in BCNF if it satisfies the following properties

-
  - It should be in 3nf
  - For every functional dependency, A-> B, A must be a superkey i.e in any functional dependency LHS attribute must be a Superkey.

Problem Table

EMPLOYEE table:

EID	EMP_LOCATION	EMP_DEPT	DEPT_NO	EMP_DEPT_NO
66	India	HR	D1	101
66	India	Marketing	D1	102
73	CANADA	Exports	D2	103
73	CANADA	Finance	D2	104

- Consider a table employee when an employee can work in more than one department, these dependencies cause insertion, updation and deletion problems
- It is in First Normal Form because of no multiple attributes and It is in second normal form and third Normal Form also because it is free from all kinds of partial and transitive dependencies.

Procedure to convert a table into BCNF

In the above table Functional dependencies are as follows:

- **EMP → EMP_COUNTRY**
  EMP_DEPT → {DEPT_NO, EMP_DEPT_NO}
- Candidate key: {EID, EMP_DEPT}
- From the table, we can see that we must need a candidate key combination {EID, EMP-DEPT} because single employee working in more than two departments hence his ID is duplicated so that only employee ID cannot be served as a primary key
- Create three tables from the original employee table
- Create a table called EMP_COUNTRYwhich contains employees country and employee ID and here each record can be uniquely identified from this table.

EMP_COUNTRY table:

EID	EMP_LOCATION
66	India
73	UK

- Create another table which will consist of employee department type number and the employee Department number.

EMP_DEPT

EMP_DEPT	DEPT_NO	EMP_DEPT_NO
HR	D1	101

Marketing	D1	102
Exports	D2	103
Finance	D2	104

- Create one more table EMP_DEPT_MAPPING which consists of employee ID and departmentEMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT_NO
D1	101
D2	102
D3	103
D4	104

Observed functional dependencies

-
  - **EID → EMP_COUNTRY**
  - **EMP_DEPT → {DEPT_NO, EMP_DEPT_NO}**

Conclusion: The table is in BCNF because the left side of each dependency is a superkey in all the three tables.

4-NF form in DBMS

4-NF form in DBMS

In this article, we will learn about the 4-NF form in DBMS.

Normalization is a process of breaking a table into two smaller ones that are free from insertion and deletion and update problems

4-NF is the highest level of normalization.

Fourth Normal Form

Definition of 4th Normal Form

The table is said to be in 4th Normal Form it is it satisfies the following properties

-
  - It should be in Boyce Codd normal form(BCNF)
  - It is free from multivalued dependency.

What is a multivalued dependency

A Dependency A->B for a single value of A ***there exist multiple values of B,*** then search dependencies are called as a multivalued dependency

Problem table: STUDENT

STU_ID	Course	Interest
66_3sk	Programming	Drums
66_3sk	Math	Ayurveda
73_prash	Data Mining	Drums
79_sanju	WPS	Sports
82_srinu	Math	Aerobics

- Given table is in First Normal Form because there is no multivalued attribute, it is in second normal form and third Normal Form because it is free from all kinds of partial dependency, full dependency and transitive dependency and it is in BCNF because in all the dependencies left side attribute is a super key
- Here student id 66 appears two times in courses and interests column and it leads to unnecessary repetition of data.

Procedure to Convert a table to 4-NF

Break the given tables into two different tables ensure that on multiple values are not mapped for a single attribute **STUDENT_COURSE**

STU_ID	COURSE
66_3sk	Programming
66_3sk	Math
73_prash	Data Mining
79_sanju	WPS
82_srinu	Math

STUDENT_INTEREST

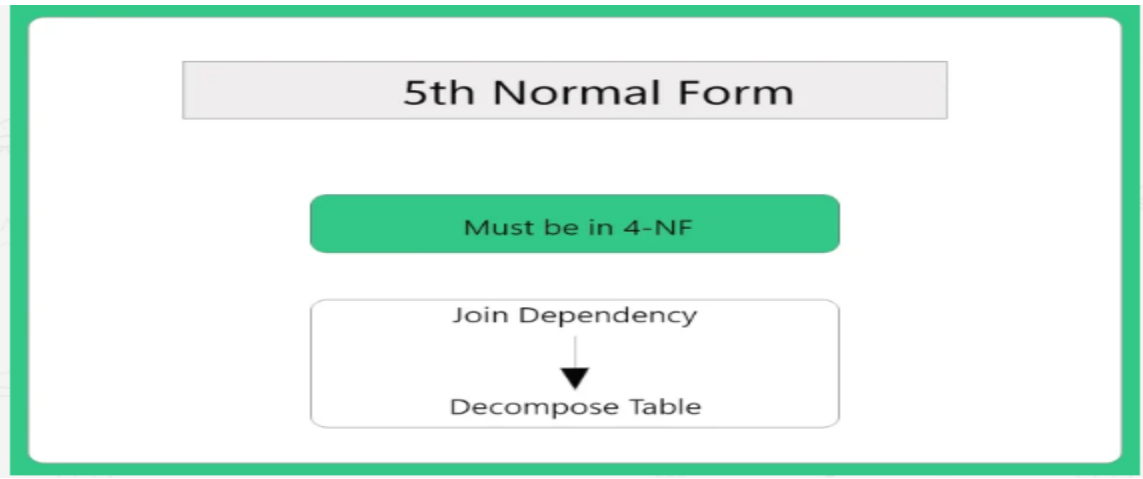
STU_ID	Interest
66_3sk	Drums
66_3sk	Ayurveda
73_prash	Drums
79_sanju	Sports
82_srinu	Aerobics

5-NF

5-NF

In this article, we will learn about 5-NF in DBMS.

Normalization is a process of making a table optimized by eliminating problems caused due to insertion updation and deletion.



Definition of 5-NF:

A table is said to be in Fifth Normal Form if it satisfies the following properties

1.
    1. If it is in fourth normal form and
    2. It should not have join dependency

It is also called as ***Project Join Normal Form(PJNF)***

If the table is having join dependency then it should be broken into smaller tables.

Problem table : SPC table

Supplier	Product	Customer
S1	P1	C1
S1	P2	C2
S1	P1	C2
S2	P2	C1
S1	P2	C1

- The above ***Supplier-Product-Customer table is in 4th normal form.***
- But still there exists anomalies
- The above table is defined by ternary relationship among supplier,product and customers
- A single supplier can supply a product which will be ***used by more than one customers***

- A customer ***can buy two or more products*** from a single supplier
- A product used by a customer ***can be supplied by more then one supplier.***

How to eliminate join dependency ?

- Join dependency is ***converted to separate tables***
- The simplest solution to eliminate join dependency from the above table is to ***break the table into 3 binary one-one relationships***
- The resultant tables which are in 5th normal form are

Supp_Pro table

Supplier	Product
S1	P1
S1	P2
S2	P2

Supp_Cust table

Supplier	Customer
S1	C1
S1	C2
S2	C1

Cust_Pro table

Product	Customer
P1	C1
P2	C2
P1	C2

P2	C1

- When we combine the above three tables, the resultant table will be the original SPC table
- But this ***might not be true in all cases*** i.e., when we combine the separated tables, the resultant will not be equivalent to the original table, in that case ***we should not break down the original table.***

Joins in DBMS

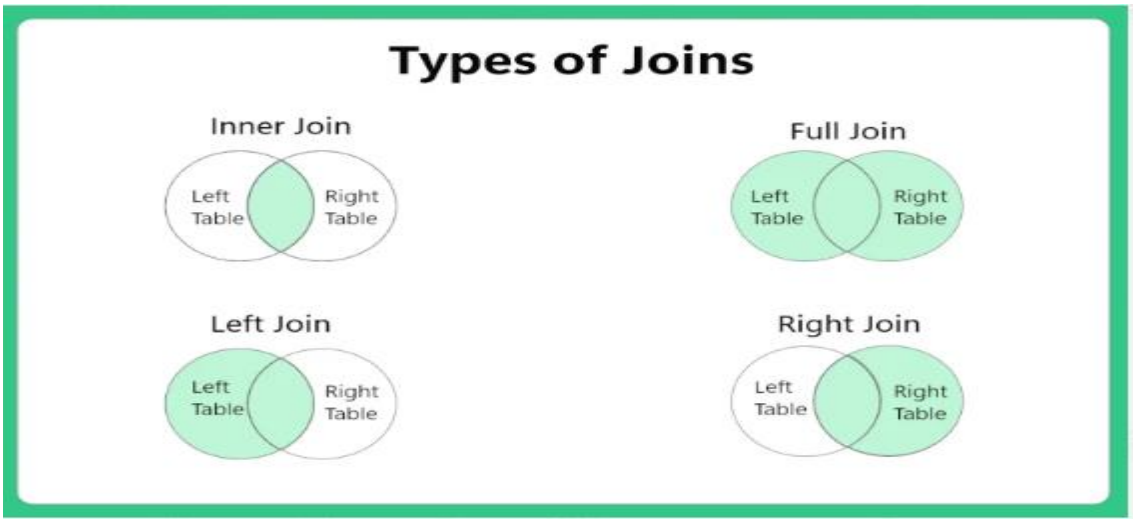
Joins in DBMS

Joints are used to retrieve data from multiple tables i.e it is used for data merging

there are five different types of joints provided by ANSI, let's discuss each of them in brief, Also called as 9I joins.

There are multiple types of joins in DBMS :

1. Inner Join
2. Right Join
3. Left Join
4. Full Join
5. Cross Join
6. Natural Join



Inner Join

- **Definition**: The inner join returns only matching rows, here columns that are used in join condition must belong to the same type, when tables having common column then only we are allowed how to use inner join
- Inner join performance is very high compared to Oracle 8i equal join

Inner Join Example

```
select ename,loc from
emp
join
dept
on emp.deptno=dept.deptno
where loc='CHICAGO';
```

- consider two sample tables emp and dept

Emp table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTN
7839	KING	PRESIDENT	–	17-NOV-81	5000	–	10
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	–	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	–	10
7566	JONES	MANAGER	7839	02-APR-81	2975	–	20
7788	SCOTT	ANALYST	7566	19-APR-87	3000	–	20
7902	FORD	ANALYST	7566	03-DEC-81	3000	–	20
7369	SMITH	CLERK	7902	17-DEC-80	800	–	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100	–	20

7900	JAMES	CLERK	7698	03-DEC-81	950	–	30
7934	MILLER	CLERK	7782	23-JAN-82	1300	–	10

Dept table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

O/P(inner join)

ROWNUM	ENAME	LOC
1	BLAKE	CHICAGO
2	ALLEN	CHICAGO
3	WARD	CHICAGO
4	MARTIN	CHICAGO
5	TURNER	CHICAGO
6	JAMES	CHICAGO

Left outer join

Definition: Left join always returns all rows from the left table and matching rows from right side table and also returns null values in the place of nonmatching rows in another table

Left join Example

```
select rownum ,ename,loc from
emp
LEFT join
dept
on emp.deptno=dept.deptno
where loc='DALLAS';
```

O/P

ROWNUM	ENAME	LOC
1	JONES	DALLAS
2	SCOTT	DALLAS
3	FORD	DALLAS
4	SMITH	DALLAS
5	ADAMS	DALLAS

Right outer join

Definition: Right join always return all rows from right side table and matching rows from the left table and also returns null values in the place of nonmatching rows in another table

Example for Right Outer join

```
select rownum,ename,loc from
emp
RIGHT join
dept
on emp.deptno=dept.deptno where loc!='CHICAGO' ;
```

O/P

ROWNUM	ENAME	LOC
1	KING	NEW YORK
2	CLARK	NEW YORK
3	JONES	DALLAS
4	SCOTT	DALLAS
5	FORD	DALLAS
6	SMITH	DALLAS
7	ADAMS	DALLAS
8	MILLER	NEW YORK

9	–	BOSTON

Full Outer join

Definition: Full join returns all rows from both tables because it is viewed as a union of a left, right and inner join, It also returns null values in the place of nonmatching rows tuples

Full Outer join example

```
select rownum,ename,loc,job,mgr from
emp
FULL OUTER JOIN
dept
on emp.deptno=dept.deptno ;
```

O/P

ROWNUM	ENAME	LOC	JOB	MGR
1	KING	NEW YORK	PRESIDENT	–
2	BLAKE	CHICAGO	MANAGER	7839
3	CLARK	NEW YORK	MANAGER	7839
4	JONES	DALLAS	MANAGER	7839
5	SCOTT	DALLAS	ANALYST	7566
6	FORD	DALLAS	ANALYST	7566
7	SMITH	DALLAS	CLERK	7902
8	ALLEN	CHICAGO	SALESMAN	7698
9	WARD	CHICAGO	SALESMAN	7698
10	MARTIN	CHICAGO	SALESMAN	7698
11	TURNER	CHICAGO	SALESMAN	7698
12	ADAMS	DALLAS	CLERK	7788
13	JAMES	CHICAGO	CLERK	7698

14	MILLER	NEW YORK	CLERK	7782
15	–	BOSTON	–	–

Natural join

Natural Join also returns matching rows only, this join performance is very high compared to inner join **Definition:** In this join, you are not required to use join condition explicitly, in this case, resource tables must have a common column, based on this oracle server itself internal establishes join condition

Natural Join Example

```
select rownum,ename,loc from
emp
NATURAL join
dept
where loc='CHICAGO';
```

O/P

ROWNUM	ENAME	LOC
1	BLAKE	CHICAGO
2	ALLEN	CHICAGO
3	WARD	CHICAGO
4	MARTIN	CHICAGO
5	TURNER	CHICAGO
6	JAMES	CHICAGO

Cross join

Definition: In this type of join data in each and, every row in one table is added to all the rows in another table Here ***cross product*** operation performed

Here we have 14 rows in emp table and 4 rows in dept table hence 14*4=56 records are displayed if the cross join is performed on these tables without condition specified

Cross Join example

```
select rownum,ename,sal,dname,loc from
emp
cross join
dept
where loc='CHICAGO';
```

O/P

ROWNUM	ENAME	SAL	DNAME	LOC
1	KING	5000	SALES	CHICAGO
2	BLAKE	2850	SALES	CHICAGO
3	CLARK	2450	SALES	CHICAGO
4	JONES	2975	SALES	CHICAGO
5	SCOTT	3000	SALES	CHICAGO
6	FORD	3000	SALES	CHICAGO
7	SMITH	800	SALES	CHICAGO
8	ALLEN	1600	SALES	CHICAGO
9	WARD	1250	SALES	CHICAGO
10	MARTIN	1250	SALES	CHICAGO
11	TURNER	1500	SALES	CHICAGO
12	ADAMS	1100	SALES	CHICAGO
13	JAMES	950	SALES	CHICAGO
14	MILLER	1300	SALES	CHICAGO

Heera 14 records are displayed because there is one row in dept table with location Chicago and that row is appended to all the rows in emp table

Dependency Preservation in DBMS

Dependency Preserving Decomposition in DBMS

- Decomposition of a relation in relational model is done to convert it into appropriate normal form
- A relation R is decomposed into two or more only if the decomposition is both lossless join and dependency preserving.

Dependency Preserving Decomposition

- If we decompose a relation R into relations R1 and R2, all dependencies of R must be part of ***either R1 or R2 or must be derivable from combination of functional dependencies(FD) of R1 and R2***
- Suppose a relation R(A,B,C,D) with FD set ***{A->BC}*** is decomposed into R1(ABC) and R2(AD) which is dependency preserving because ***FD A->BC is a part of R1(ABC).***

Theory

Consider a schema ***R(A,B,C,D)*** and functional dependencies ***A->B and C->D*** which is decomposed into ***R1(AB) and R2(CD)***

This decomposition is ***dependency preserving decompostion*** because

- ***A->B*** can be ensured in ***R1(AB)***
- ***C->D*** can be ensured in ***R2(CD)***

Example

Let a relation R(A,B,C,D) and set a FDs F = { A -> B , A -> C , C -> D} are given.
A relation R is decomposed into –

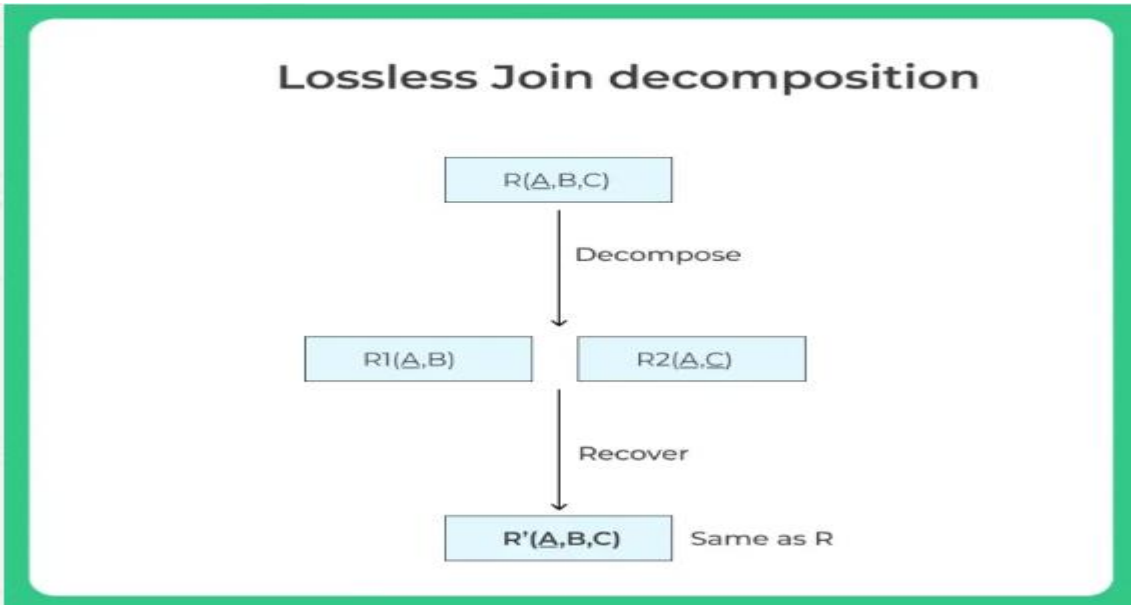
R_1 = (A, B, C) with FDs F_1 = {A -> B, A -> C}, and
R_2 = (C, D) with FDs F_2 = {C -> D}.
$F' = F_1 \cup F_2$ = {A -> B, A -> C, C -> D}
so, F' = F.
And so, $F'^{+} = F^{+}$.

Thus, the decomposition is dependency preserving decomposition.

Lossless Join Decomposition in DBMS

Lossless Join Decomposition in DBMS.

- Decomposition of a relation in relational model is done to convert it into appropriate normal form
- A relation R is decomposed into two or more only if the decomposition is both lossless join and dependency preserving.



Lossless join decomposition

There are two possibilities when a relation R is decomposed into R1 and R2.They are

-
  - Lossy decomposition i.e., ***R1⋈R2⊃R***
  - Lossless decomposition i.e., ***R1⋈R2=R***

<u>For a decomposition to be lossless, it should hold the following conditions :</u>

- Union of attributes of R1 and R2 must be equal to attribute R. each attribute of R must be either in R1 or in R2 i.e., ***Att(R1) ∪ Att(R2) = Att(R)***
- Intersection of attributes of R1 and R2 must not be null i.e., ***Att(R1) ∩ Att(R2) ≠ Ø***
- Common attribute must be a key for atleast one relation(R1 or R2) i.e., ***Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2)->Att(R2)***

Example

A relation R(A,B,C,D) with FD set {A->BC} is decomposed into R1(ABC) and R2(AD). This is lossless join decomposition because

-
  - First rule holds true as Att(R1) ⋃ Att(R2)=***(ABC) ⋃ (AD)= (ABCD)*** = Att(R)
  - Second rule holds true as Att(R1) ⋂ Att(R2) = ***(ABC) ⋂ (AD) ≠ Ø***
  - Third rule holds true as ***Att(R1) ⋂ Att(R2) = A is a key of R1(ABC) because A->BC*** is given

Lossless join and Dependency preserving Decomposition in DBMS

- Decomposition of a relation in relational model is done to convert it into appropriate normal form
- A relation R is decomposed into two or more only if the decomposition is both lossless join and dependency preserving.

Learn more about Other Decompositions here on this page.

Lossless join decomposition

There are two possibilities when a relation R is decomposed into R1 and R2.They are

- Lossy decomposition i.e., ***R1⋈R2⊃R***
- Lossless decomposition i.e., ***R1⋈R2=R***

For a decomposition to be lossless, it should hold the following conditions

- Union of attributes of R1 and R2 must be equal to attribute R. each attribute of R must be either in R1 or in R2 i.e., ***Att(R1) ⋃ Att(R2) = Att(R)***
- Intersection of attributes of R1 and R2 must not be null i.e., ***Att(R1) ⋂ Att(R2) ≠ Ø***
- Common attribute must be a key for atleast one relation(R1 or R2) i.e., ***Att(R1) ⋂ Att(R2) -> Att(R1) or Att(R1) ⋂ Att(R2)->Att(R2)***

Example
A relation R(A,B,C,D) with FD set {A->BC} is decomposed into R1(ABC) and R2(AD). This is lossless join decomposition because

- First rule holds true as Att(R1) ⋃ Att(R2)=***(ABC) ⋃ (AD)= (ABCD)*** = Att(R)
- Second rule holds true as Att(R1) ⋂ Att(R2) = ***(ABC) ⋂ (AD) ≠ Ø***
- Third rule holds true as ***Att(R1) ⋂ Att(R2) = A is a key of R1(ABC) because A->BC*** is given

Dependency Preserving Decomposition

- If we decompose a relation R into relations R1 and R2, all dependencies of R must be part of ***either R1 or R2 or must be derivable from combination of functional dependencies(FD) of R1 and R2***
- Suppose a relation R(A,B,C,D) with FD set ***{A->BC}*** is decomposed into R1(ABC) and R2(AD) which is dependency preserving because ***FD A->BC is a part of R1(ABC)***

Example
Consider a schema ***R(A,B,C,D)*** and functional dependencies ***A->B and C->D*** which is decomposed into ***R1(AB) and R2(CD)***

This decomposition is ***dependency preserving decompostion*** because

- ***A->B*** can be ensured in ***R1(AB)***
- ***C->D*** can be ensured in ***R2(CD)***

Transactions and Concurrency Control

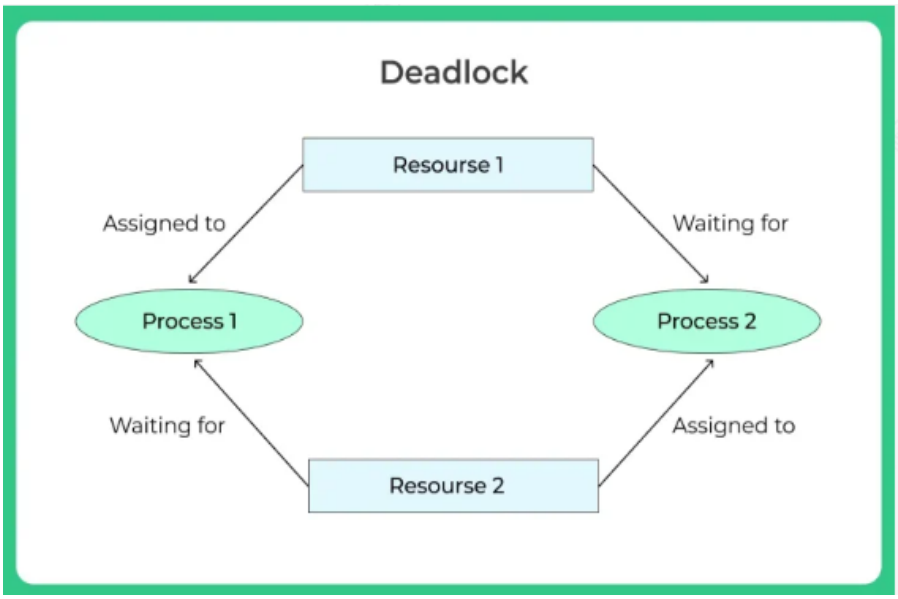
Deadlock in DBMS

Deadlock in DBMS

In this article, we will learn about Deadlock in DBMS.

- Every process need some resource for its execution and these resources are granted in sequential order
    1. First the process *request some resource*.

2. OS grants the resource to the process if it is available or else it places the request in the wait queue.
3. The process uses it and releases on the completion so that it can be granted to another process
4. Deadlock is a situation where two or more transactions are waiting indefinitely for each other to give up their locks.



Example:

Transaction ***T1 is waiting for transaction T2*** to release the lock and similarly transaction ***T2 was waiting for transaction T1*** to release Its lock, as a result, all activities come to halt state and remain at standstill .This continues until the DBMS detects that ***deadlock has occurred and abort some of the transactions.***

Necessary conditions for Deadlocks

Mutual Exclusion

It implies if **two processes cannot use the same resource at the same time.**

Hold and Wait

A process ***waits for some resources while holding another resource*** at the same time.

No pre-emption

The process which ***once scheduled will be executed till the completion***. No other process can be scheduled by the scheduler meanwhile.

Circular Wait

All the ***processes must be waiting for the resources in a cyclic manner*** so that the last process is waiting for the resource which is being held by the first process.

Deadlock avoidance

- Whenever a database is ***stuck in a deadlock*** it's ***better to abort the transactions*** that are ***resulting deadlock*** but this is a waste of resources and time
- Deadlock avoidance mechanism used ***to detect any deadlock situation in advance*** by using some techniques like WEIGHTS FOR GRAPH but this wait for graph method is suitable only for smaller databases, for large databases other prevention techniques are used

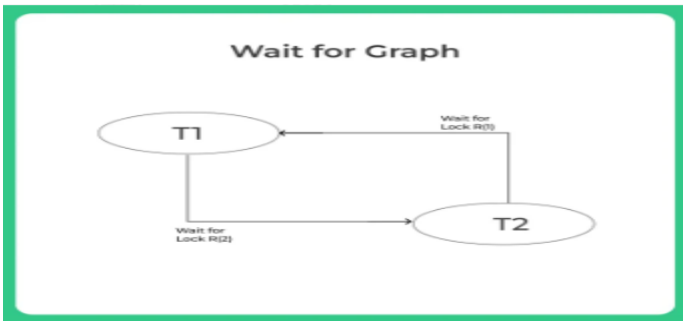
Deadlock detection (Wait for Graph)

- Whenever a transaction waits indefinitely to obtain a lock DBMS should check whether the transaction is involved in a deadlock or not .For this purpose a ***lock manager maintains wait-for graph*** for the detection of deadlock cycles
- Wait for graph method is ***suitable only for smaller databases***

Steps

1. In this method, ***a graph is created based on transactions and their locks***
2. While Creating the graph ***if a cycle or closed loop is encountered then there is a deadlock***

This wait for graph is maintained by the system for every transaction that is waiting for some resource held by others



Deadlock prevention

- Prevention techniques are ***used for larger databases .***

- Resource ***allocation is done in such a way that deadlock never occurs***
- DBMS ***analyse the transactions to determine whether any deadlock situation can arise or not***, if there is any possibility then DBMS never allows such transaction for execution

Wait-Die scheme

- Whenever there is a conflict of resource between transactions DBMS simply ***check the timestamp of both the transactions***
- There are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T
- If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS

Check if TS(Ti) < TS(Tj) :

- If Ti is older transaction and Tj has some resource held then ***Ti is allowed to wait until data item is available for execution***
- Simply if ***older transaction is waiting for a resource*** which is ***locked by younger transaction*** then ***older transaction is allowed to wait*** for resource until it is available

Check if TS(Ti) < TS(Tj)

- If Ti is older transaction and has held some resource and if Tj is waiting for it, then ***Tj is killed and restarted later with the random delay but with the same timestamp.***

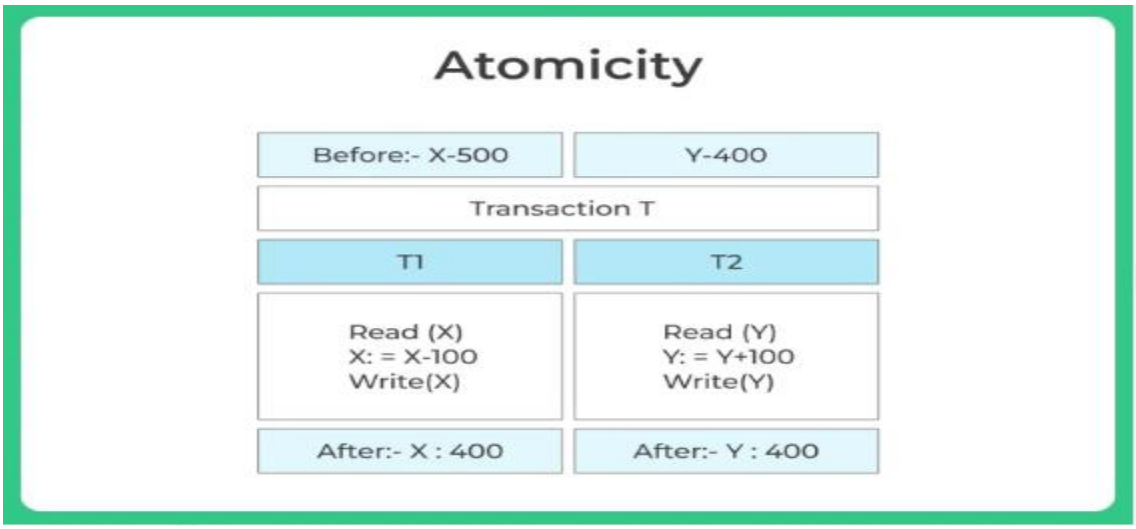
Wound wait scheme

- Whenever an older transaction request for a resource which is held by a younger transaction, ***older transaction forces the younger transaction to kill its transaction and release the resource to the elder one***
- After minute delay ***younger transaction is restarted within the same timestamp***
- Whenever older transaction has held a resource which is requested by the Younger transaction, then the ***younger transaction is asked to wait until older releases it.***

ACID Properties in DBMS

Atomicity (A)

- An atomic transaction simply means that the transaction happens only if it can be completed and achieve its purpose or if not it doesn't happen at all.
- Atomicity defines that there are no transactions that occur partially hence the atomicity is also known as the **"all or nothing rule"**.



It is associated with two operations

- **Abort**: If a transaction is aborted i.e. it is incomplete, changes made to the database are not visible.
- **Commit**: If a transaction is committed i.e. it is complete changes made to the database or visible.

Example

Consider a transaction T which consists of T1 and T2: ***Task is to transfer 100 from account X to account Y***

- **T1:** Deduct the amount from account X
- **T2:** Credit the amount to account Y

If the transaction fails after the completion of T1 and before completion of T2 then the amount will be deducted from account X but it will not be added to the account Y which ultimately results in an inconsistent database state.

Consistency(C)

- Integrity constraints (maintain certain rules to authenticate database) must be maintained *to* ensure your database is consistent before and after the transaction.
- Consistency refers to the correctness of the database.

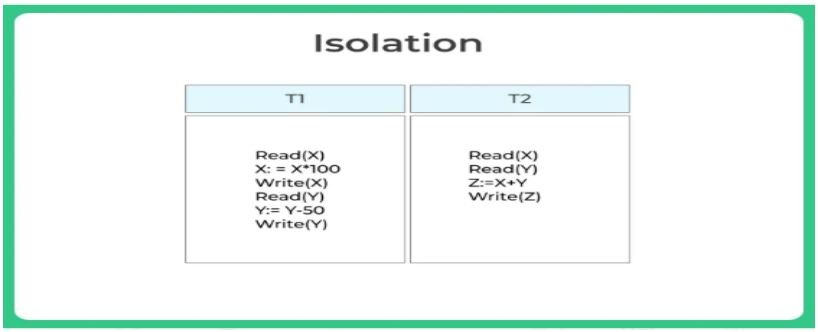
Example

The total amount before and after the transaction must be maintained.

- Total amount ***before T occurs = 500 + 200 = 700.***
- Total amount ***after T occurs = 400 + 300 = 700.***
- The database is said to be ***inconsistent if T1 is completed but fails as a result total.***
- Transaction ***P is incomplete.***

Isolation (I)

- Isolation ensures that multiple transactions can occur at the same time provided each transaction is independent and shall not interfere in another transaction.
- Changes in one particular transaction are visible to any other transaction unless a particular change in the transaction is written to the memory.



Durability (D)

- Durability ensures that once after the completion of the transaction execution the updates and modifications to the database are stored and returned to a disc so that they can be used whenever a system failure occurs.

- So that all the changes become permanent and stored in non-volatile memory so that any action can be referred to and never lost.

What is the purpose of these ACID properties?

- Provides a mechanism for the correctness and consistency of a database system.
- As a result, each transaction is independent, consistent with each other and all actions are stored properly and permanently and support failure recovery.

Concurrency Control in DBMS

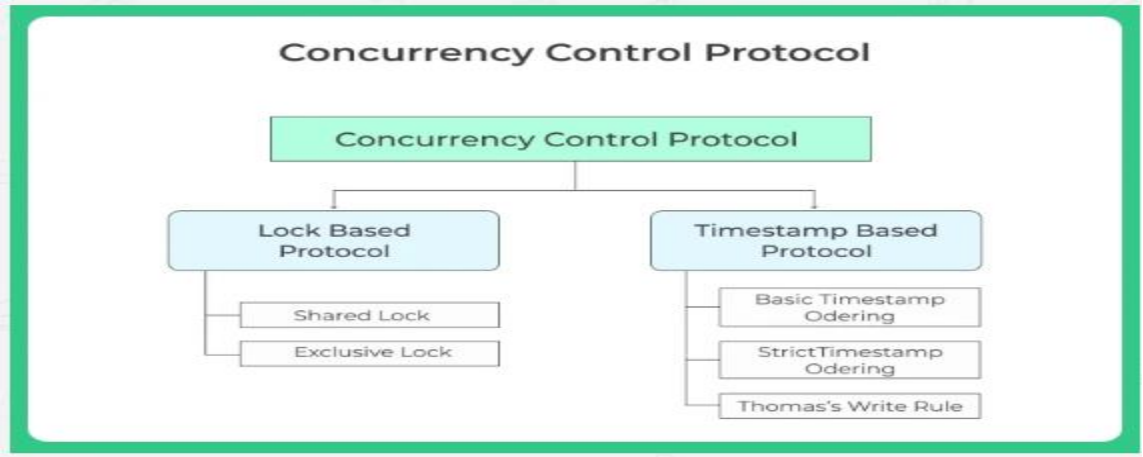
Concurrency Control Protocols in DBMS

In this article, we will learn about Concurrency Control Protocols in DBMS.

- Concurrency Control Protocols in DBMS are procedures that are used for managing multiple simultaneous operations without conflict with each other
- Concurrency control ensures the speed of the transactions but at the same time we should address conflicts occurring in a multi-user system and make sure the database transactions are performed concurrently without violating the Data integrity of the respective databases

Concurrency control can be broadly divided into two protocols

- Lock Based Protocol
- Timestamp Based Protocol



Lock-Based Protocol

- Lock based protocol mechanism is very crucial in concurrency control which controls concurrent access to a data item
- It ***e***nsures that one transaction should not retrieve and update record while another transaction is performing a write operation on it

Example

In traffic light signal that indicates stop and go, when one signal is allowed to pass at a time and other signals are locked, in the same way in a database transaction, only one transaction is performed at a time meanwhile other transactions are locked

- If this locking is not done correctly then it will display inconsistent and incorrect data
- It maintains the order between the conflicting pairs among transactions during execution
- There are two lock modes,
    1. Shared Lock(S)
    2. Exclusive Lock(X)

Shared lock(S)

- Shared locks can only read without performing any changes to it from the database
- Shared Locks are represented by S.
- S – lock is requested using lock – s instruction.

Exclusive Lock(X)

- The data items accessed using this instruction can perform both read and write operations
- Exclusive Locks are represented by X.
- X – lock is requested using lock – X instruction.

Lock Compatibility Matrix

	Shared	**Exclusive**
Shared	True	False

Exclusive	False	False

- Lock compatibility Matrix controls whether this multiple transactions can acquire locks on the same resource at a time or not
- If a resource is already locked by another transaction, then a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock.
- There can be any number of transactions for holding shared locks on an item but if any transaction holds exclusive lock then item no other transaction may hold any Lock on the item.

Timestamp Based Protocol

- It is the most commonly used concurrency protocol
- Timestamp based protocol helps DBMS to identify transactions and determines the serializability order
- It has a unique identifier where each transaction is issued with a timestamp when it is entered into the system
- This protocol uses the system time or a logical counter as a timestamp which starts working as soon as the transaction is created

Timestamp Ordering Protocol

This protocol ensure serializability among transactions in their conflicting read/write operations

- ***TS(T)***: transaction of timestamp (T)
- ***R–timestamp(X)***: Data item (X) of read timestamp
- ***W–timestamp(X):*** Data item (X) of write timestamp

Timestamp Ordering Algorithms

1. Basic Timestamp ordering

-
  - It ensures transaction execution is not violated by comparing the timestamp of T with Read_TS(X) and Write_TS(X)

- When it finds that transaction execution sequence is violated transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp

2. Strict Timestamp ordering

It makes sure that the schedules are both strict for easy recoverability and conflict serializability

3.Thomas's Write Rule

-
- It does not enforce conflict serializability
- It rejects some write operations, by modifying the checks for the write_item(X) operation as follows.

Read_TS(X) > TS (T)

- Abort and rollback transaction T and reject the operation when read timestamp is greater than timestamp transaction

Write_TS(X) > TS(T)

- Whenever write timestamp is greater than the timestamp of the entire transaction then do not execute the write operation but continue processing
- Because sometimes transaction with set and timestamp may be greater than TS(T) and after T in the timestamp has already written the value of X.

If neither 1 nor 2 occurs As mentioned above

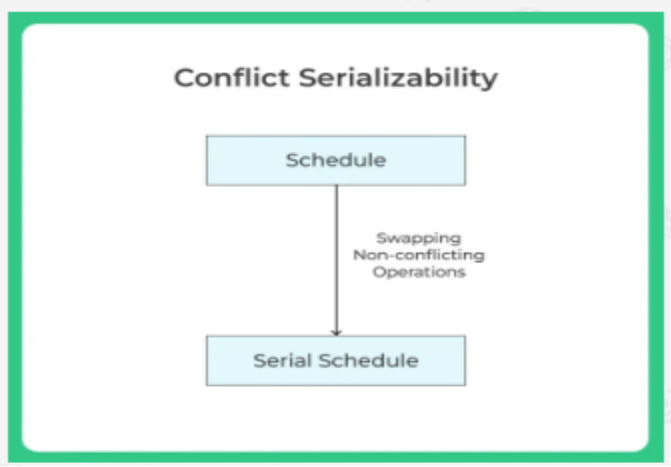
- Execute the Write item(X) operation of transaction T and set Write_TS(X) to TS(T).

Conflict Serializability in DBMS

Conflict Serializability in DBMS

- Serial schedules will have less performance because it cannot allow multiple transactions run concurrently, hence to improve the performance we need to execute multiple transactions at the same time.

- But sometimes because of the concurrency of transactions database may become inconsistent like when two or more transactions try to access the same data item.
- Hence to avoid this we need to verify whether these concurrent schedules are serializable or not.



What are the conflicting operations?

Two operations are said to be conflicting if both are different operations fighting for the same data item in which at least one is write operation

Examples for conflicting and no conflicting operations

- (R1 (A), W2 (A)): Conflicting because R1 and W2 are different operations fighting for the same data item A and one is a write operation
- (W1 (A), W2 (A)) and (W1 (A), R2 (A)): Conflicting because both are different operations and fighting for the same data item A involving at least one write operation
- (R1 (A), W2 (B)): Non conflicting because these R1 and W2 are different operations and are accessing different data items A and B respectively
- (W1 (A), W2 (B)): Non-conflicting because W1 and W2 are different operations accessing different data items A and B.

Checking whether a given schedule is conflict serializable or not

Consider the following schedule:

S1: R1 (A), W1 (A), R2 (A), W2 (A), R1 (B), W1 (B), R2 (B), W2 (B)

If i and j are two operations in a transaction and i< j (i.e. i is executed before j), the same order will follow in the schedule as well. Using this property, we can get two transactions of schedule S1 as:

T1: R1 (A), W1 (A), R1 (B), W1 (B)

T2: R2 (A), W2 (A), R2 (B), W2 (B)

T1->T2 or T2->T1 are possible serial schedules

- Interchanging R2 (A) and R1 (B) non-conflicting operations in S1, the schedule becomes,

S11: R1 (A), W1 (A), R1 (B), W2 (A), R2 (A), W1 (B), R2 (B), W2 (B)

- In the same way, interchanging W2 (A) and W1 (B) non-conflicting operations in S11, the schedule becomes,

S12: R1 (A), W1 (A), R1 (B), W1 (B), R2 (A), W2 (A), R2 (B), W2 (B)

- S12 is a serial schedule in which all operations of T1 are performed before starting any operation of T2. Since S1 has been transformed into a serial schedule S12 by interchanging non-conflicting operations of its or , S1 is conflict serializable.

Let us take another Schedule:

S2: R2 (A), W2 (A), R1 (A), W1 (A), R1 (B), W1 (B), R2 (B), W2 (B)

The two transactions are:

T1: R1 (A), W1 (A), R1 (B), W1 (B)

T2: R2 (A), W2 (A), R2 (B), W2 (B)

T1->T2 or T2->T1 are possible serial schedules.

- Original Schedule is:

S2: R2 (A), W2 (A), R1 (A), W1 (A), R1 (B), W1 (B), R2 (B), W2 (B)

- Interchanging R1 (A) and R2 (B) non-conflicting operations in S2, the schedule becomes,

S21: R2 (A), W2 (A), R2 (B), W1 (A), R1 (B), W1 (B), R1 (A), W2 (B)

- In the same way, interchanging W1 (A) and W2 (B) non-conflicting operations in S21, the schedule becomes,

S22: R2 (A), W2 (A), R2 (B), W2 (B), R1 (B), W1 (B), R1 (A), W1 (A)

- In schedule S22, even though all operations of T2 are performed first, but the operations of T1 are not in order (the order should be R1 (A), W1 (A), R1 (B), W1 (B)). So S2 is not conflict serializable.

Conflict Equivalent: Two schedules are said to be conflict equivalent when one can be transformed into another by swapping non-conflicting operations. In the example discussed above, S11 is a conflict equivalent to S1 (S1 can be converted to S11 by swapping non-conflicting operations). Similarly, S11 is conflict equivalent to S12 and so on.

NoteAlthough S2 is not conflict serializable, still it is conflict equivalent to S21 and S21 because S2 can be converted to S21 and S22 by swapping non-conflicting operations.

NoteThe schedule which is conflict serializable is always conflict equivalent to one of the serial schedule. S1 schedule discussed above (which is conflict serializable) is equivalent to serial schedule (T1->T2).

View Serializability in DBMS

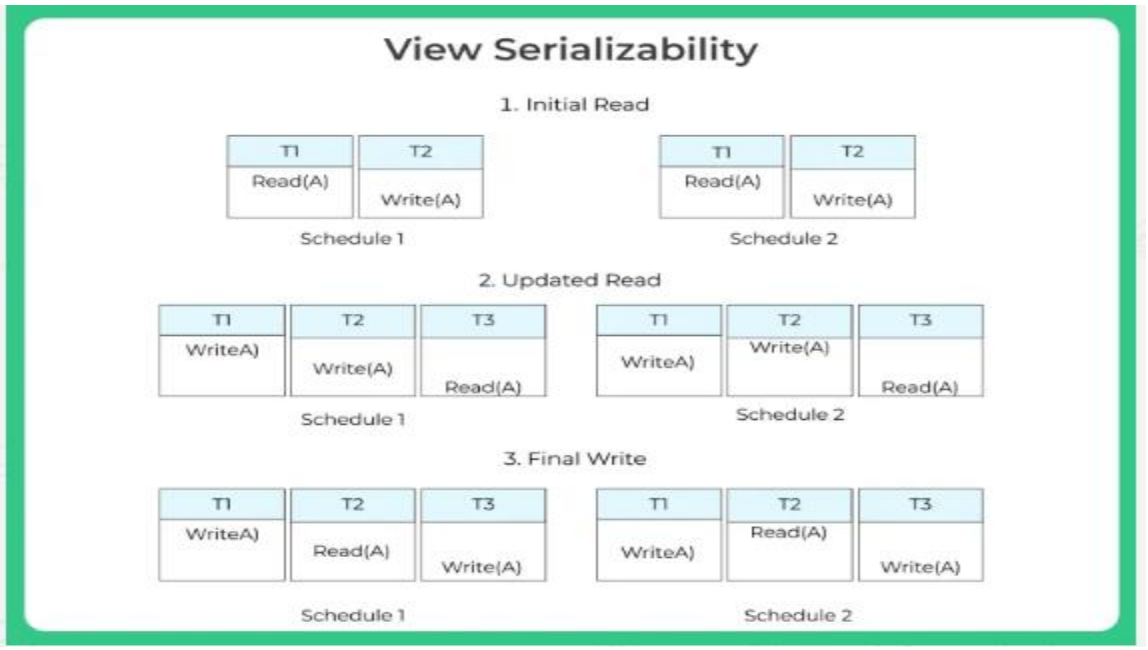
View Serializability

DBMS View Serializability in DBMS is a technique to find out that a detailed schedule is either view serializable or now not. To prove whether a special schedule is view serializable, the consumer includes trying out whether or not the agreed time table is View equivalent to its serial schedule. considering there is no synchronized transactions execution, we are able to affirm that a serial schedule will simply now not go away the database unpredictable.

View Serializability in DBMS

- A schedule is view serializable when it is view equivalent to a serial schedule.
- All conflict serializable schedules are view serializable.

- The view serializable which is not a conflict serializable contains blind writes.



View Equivalent

Two view equivalent schedules S1 and S2 should satisfy the following conditions:

1.Initial Read

- The initial read of both the schedules ***must be in the same transaction.***
- Suppose two schedule S1 and S2. In schedule S1, if a transaction ***T1 is reading the data item A, then in S2, transaction T1 should also read A.***
- The two schedules S1 and S2 are view equivalent because Initial read operation in S1 is done by T1 and in S2 also it is done by T1.

2.Updated Read

- Suppose in schedule S1, if transaction ***Tm is reading A which is updated by transaction Tn then in S2 also, Tm should read A which is updated by Tn.***
- The two schedules are ***not view equal*** because, ***in S1,transaction T3 is reading A updated by transaction T2 and in S2, transaction T3 is reading A which is updated by transaction T1.***

3.Final Write

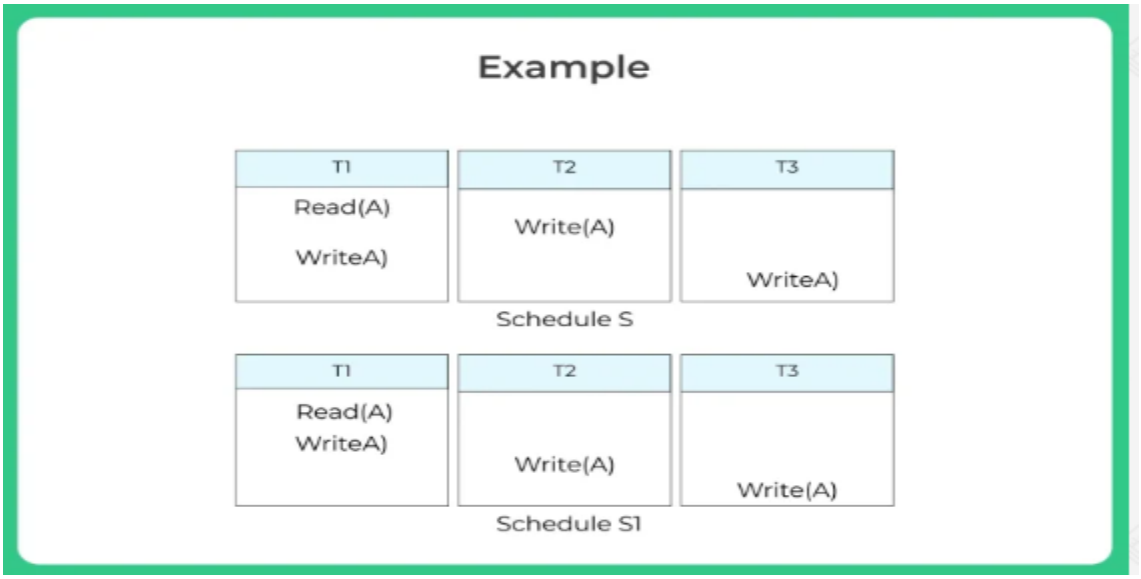
- A final write must be the ***same in both the schedules.***
- Suppose in schedule ***S1, if a transaction T1 updates A in the last, then in S2 final write operation should also be done by transaction T1.***
- The two schedules is view equal because Final write operation in S1 is done by T3 and in S2 also the final write operation is done by T3.

Example:

Consider a schedule S with 3 transactions.

The total number of possible schedules is 3!=6.They are

S1, S2 , S3 , S4, S5, S6



Considering the first schedule

Schedule S1

- **Step 1:** Final updation on data items

In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

- **Step 2:** Initial Read

The initial read operation ***in S is done by T1 and in S1, it is also done by T1.***

- **Step 3:** Final Write

The final write operation ***in S is done by T3 and in S1, it is also done by T3***. So, S and S1 are view Equivalent.

-
  - The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.**Hence, view equivalent serial schedule is**

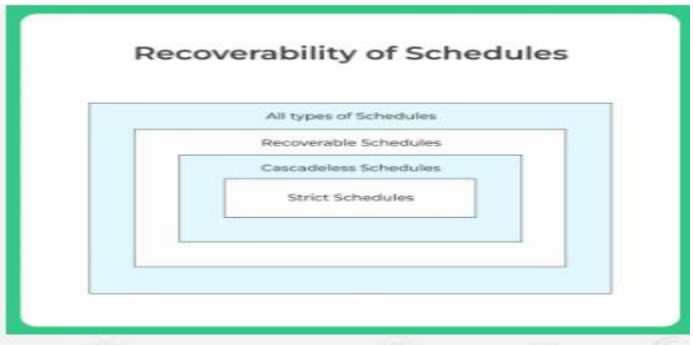
T1 → T2 → T3

Recoverability of Schedules in DBMS

Recoverability of Schedules in DBMS

Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure. In that case, the failed transaction has to be rollback. But some other transactions may also have used value produced by the failed transaction. So we also have to rollback those transactions. This process is called recoverability of schedules.

- Different types of recoverability of schedules are
- Recoverable Schedule
- Cascadeless Schedule
- Strict Schedule
- Cascading Abort



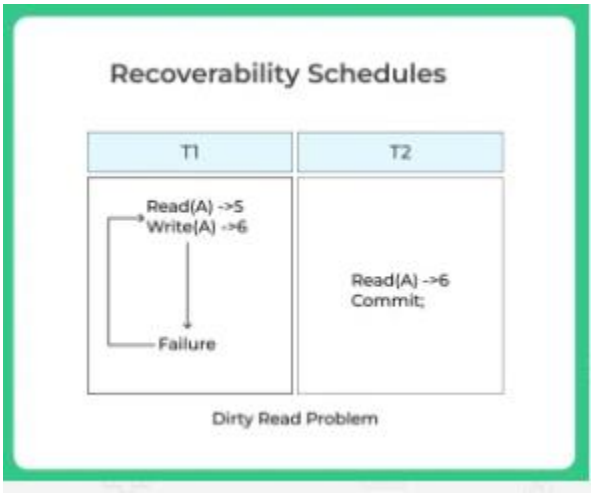
.Recoverable Schedule:

- A schedule is said to be recoverable if it is recoverable as name suggest i.e only reads are allowed before write operation on same data.
- Only reads (Tm->***Tn)*** is permissible.

Example:

S1: R1(x), **W1(x)**, R2(x), R1(y), R2(y), **W2(x)**, W1(y), **C1**, **C2**;

- Given schedule follows order of ***Tm->Tn => C1->C2***.
- Transaction T1 is executed before T2 hence there is no chance of conflict occur.
- R1(x) appears before W1(x) and transaction T1 is committed before T2 i.e. completion of first transaction performed first update on data item x, hence given schedule is recoverable.



Example of irrecoverable schedule:

S2: R1(x), R2(x), R1(z), R3(x), R3(y), W1(x), **W3(y)**, R2(y), W2(z), **W2(y)**, C1, **C2**, **C3**;

- ***Tm->Tn => C2->C3*** but W3(y) executed before W2(y) which leads to conflicts thus it must be committed before T2 transaction. So given schedule is unrecoverable.
- If ***Tm->Tn => C3->C2*** is given in schedule then it will become recoverable schedule.

Note:

A committed transaction should never be rollback. It means that reading value from uncommitted transaction and commit it will enter the current transaction into inconsistent or unrecoverable state this is called ***Dirty Read problem.***

2. Cascadeless Schedule:

When no read or write-write occurs before execution of transaction then corresponding schedule is called cascadeless schedule.

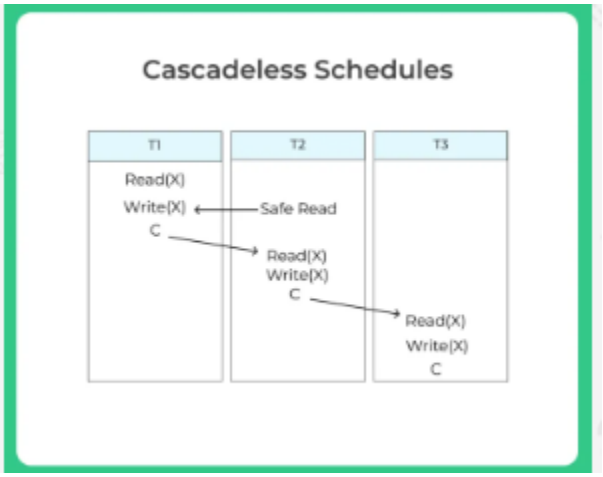
Example

S3: R1(x), R2(z), R3(x), R1(z), R2(y), R3(y), W1(x), C1, W2(z), **W3(y)**, **W2(y)**, **C3**, **C2**;

In this schedule ***W3(y)*** *and* ***W2(y)*** *overwrite conflicts* and there is no read, therefore given schedule is cascadeless schedule

Special Case

A committed transaction desired to abort. As given in image all the transactions are reading committed data, hence it is cascadeless schedule.



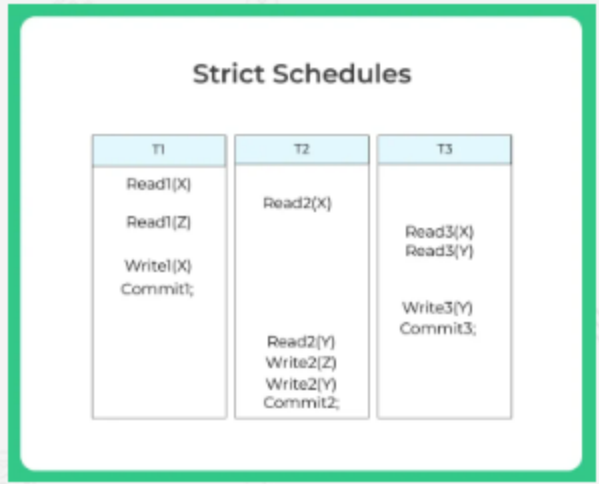
3. Strict Schedule

- If the schedule contains no read or writes before commit then it is known as a strict schedule.
- A strict schedule is strict in nature.

Example

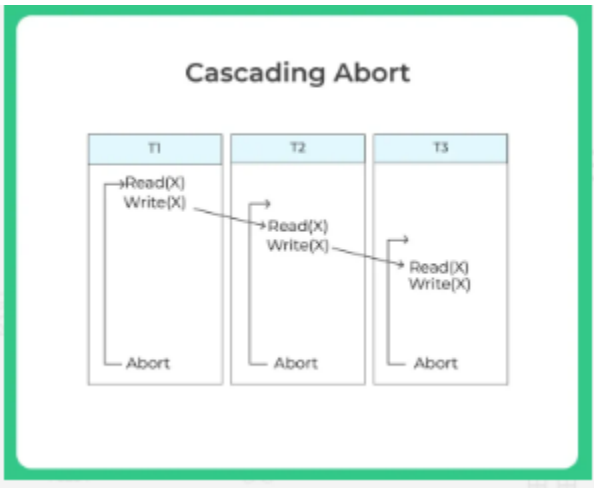
S4: R1(x), R2(x), R1(z), R3(x), R3(y), **W1(x)**, **C1**, **W3(y)**, **C3**, R2(y), **W2(z)**, **W2(y)**, **C2**;

In this schedule, no read-write or write-write conflict arises before commit hence it is a strict schedule



Cascading Abort

- Cascading Abort can also be rollback.
- If transaction T1 abort as T2 read data that written by T1 which is not committed,therefore, it is cascading rollback.



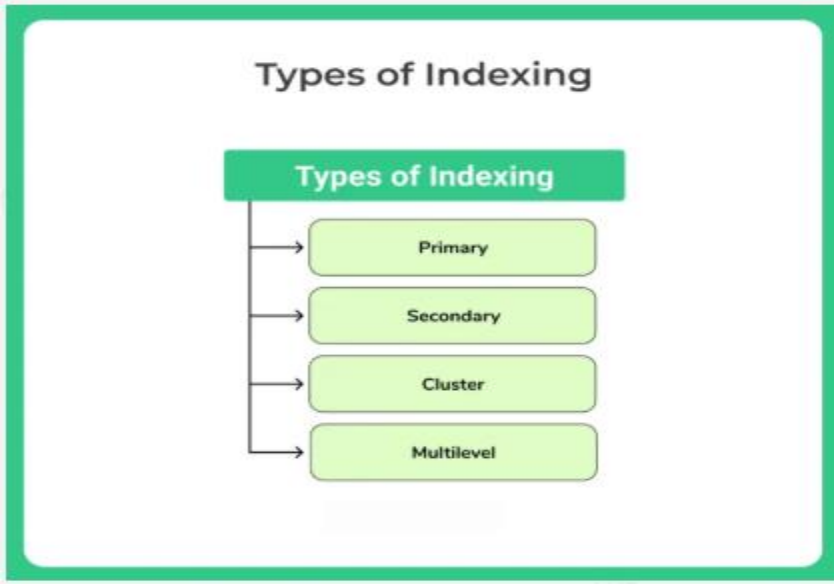
Indexing, B and B+ trees

Indexing and it’s Types

Indexing and it’s Types

- An index
    - Takes a search key as input.
    - Efficiently returns a collection of matching records.

- An Index is a compact table with two columns.
    - The first column stores a duplicate of the primary or candidate key from table.
    - The second column contains pointer that hold the disk block addresses where the corresponding key-value is stored.
- There exist four primary types of indexing method.
    - Primary Indexing
    - Secondary Indexing
    - Cluster Indexing
    - Multilevel Indexing

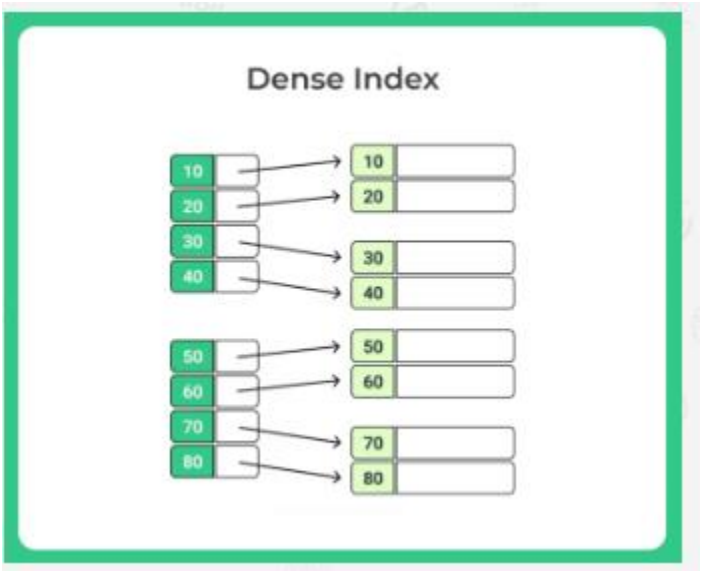


Primary Indexing

- Primary indexing helps in efficiently retrieving records based on their primary key values.
- It creates an index structure that maps primary key values to disk block addresses.
- The primary index consists of a sorted list of primary key values and their corresponding disk block pointers.
- It speeds up data retrieval by minimizing search efforts and is particularly useful for primary key-based queries.
- Primary Indexing is further divided into two types.
    - Dense Index
    - Sparse Index

Dense Index

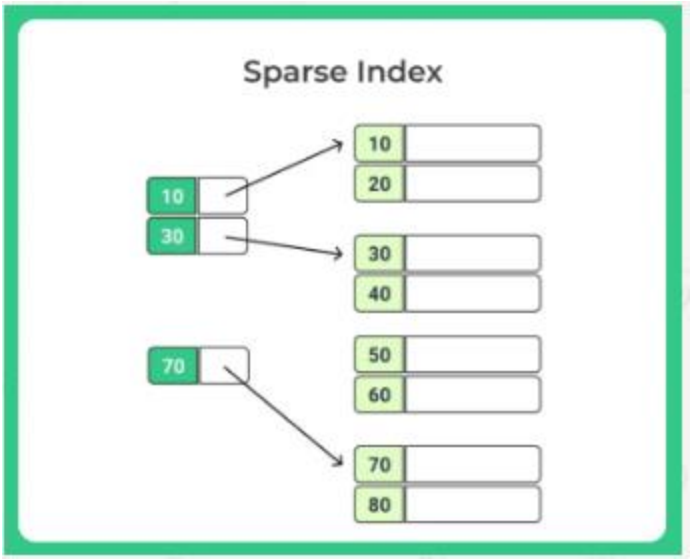
- A Dense Primary Index is a type of primary index in a database management system (DBMS).
- It contains an index entry for every record in the table, resulting in a one-to-one mapping between the index and the table.
- Dense Primary Indexes are particularly useful in scenarios where random access to individual records is frequent or essential.
- Enables direct access to any record by using the primary key.
- As every record requires an index entry, the Dense Primary Index can consume significant storage resources, especially for large tables.



Sparse Index

- A Sparse Primary Index is a type of primary index in a database management system (DBMS).
- Unlike a Dense Primary Index, it contains index entries for only a subset of records in the table, rather than every record.
- It also improves the efficiency of index maintenance operations, such as inserts, updates, and deletes, as fewer index entries need to be modified.

- This allows for efficient access to specific records based on their primary key values, as the index provides a direct path to the relevant data.
- However, it's important to note that the efficiency of using a Sparse Primary Index depends on the selectivity of the index, meaning the ratio of the number of indexed records to the total number of records in the table.

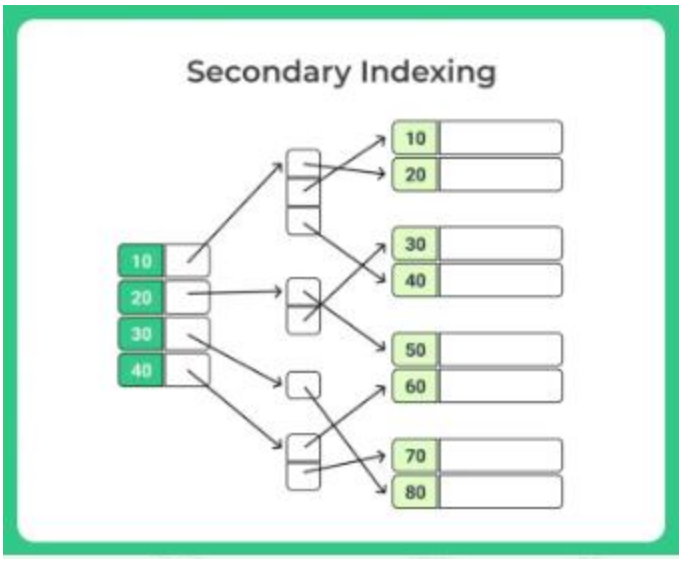


Secondary Indexing

- Secondary indexing is used to improve the search performance of frequently queried non-key attributes/columns in a database table.
- Unlike primary indexing, which is based on the primary key, secondary indexing focuses on non-key columns.
- It creates an index structure that maps the values of the indexed column to the corresponding disk block addresses or pointers where the records are stored.
- Secondary indexes enable faster access to records based on the values in the indexed column, reducing the need for full table scans.

Example

- **Creation of Secondary Index:** A secondary index is created on the "Location" column of the customer database table.
- **Index Structure:** The secondary index structure maps distinct "Location" values to their corresponding disk block addresses or pointers.
- **Efficient Location-based Queries:** With the secondary index, queries that involve searching for customers based on a specific location can be executed more efficiently.



Cluster Indexing

- Cluster indexing is a method of indexing used in database management systems (DBMS) to physically order records in a table based on the indexed columns.
- The indexed column(s) in a cluster index determine the physical order of the records in the table.
- Cluster indexing is particularly useful for range queries and sequential access patterns.
- Cluster indexing can be applied to both primary and secondary indexes, depending on the DBMS capabilities and requirements.

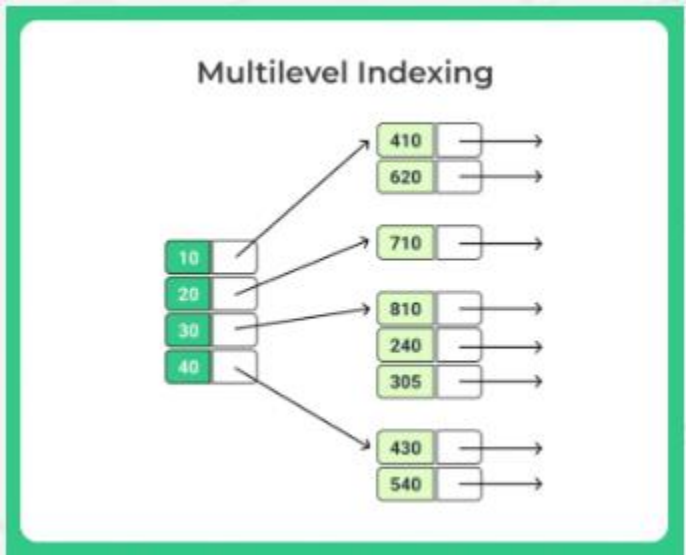
Example

- Let's consider a database table called "Employees" with columns like "EmployeeID," "Name," "Department," and "Salary."
- We decide to create a cluster index on the "Department" column.
- The cluster index organizes the data physically on disk based on the "Department" values.
- Rows with the same "Department" value are stored together in the same or nearby data pages.
- For example, all employees from the "Sales" department will be stored together, followed by the "Marketing" department, and so on.

- When a query is executed that involves filtering or sorting based on the “Department” column, the database can quickly locate the relevant data pages.
- This improves query performance, as the database doesn’t need to scan the entire table but can directly access the clustered data pages.

Multilevel Indexing

- Multilevel indexing is a technique used in database systems to efficiently index large amounts of data.
- It involves creating multiple levels of indexes to navigate and access the data.
- The first level index, also known as the primary index, provides an entry for each block or page of data.
- The primary index is usually based on the primary key of the table.
- Each entry in the primary index points to a block or page containing a secondary index.
- The secondary index is created on a secondary key or non-key attribute of the table.
- The secondary index provides a way to locate specific records within a block or page.

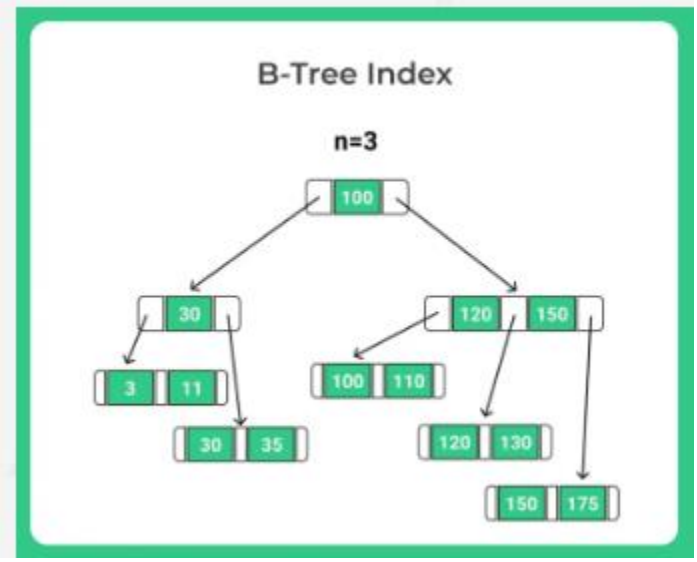


B-Tree Indexing

- B-tree indexing is a widely used indexing technique in database systems for efficient data storage and retrieval.
- It is specifically designed for disk-based storage systems, where data is stored on hard drives or solid-state drives (SSDs).
- A B-tree is a balanced tree data structure that allows for efficient search, insertion, and deletion operations.
- It is called a “B-tree” because it is a balanced version of a binary tree.
- The B-tree consists of nodes, where each node contains multiple keys and pointers to child nodes.

Properties of B-Tree

- In a B-tree, the paths from the root to the leaves tend to have approximately equal lengths.
- For any node in a B-tree that is neither the root nor a leaf, the number of its children falls within the range of n/2 to n, where n represents the degree of the B-tree.



Advantages of Indexing

- **Improved Query Performance:** Indexing allows for faster data retrieval by providing a quick and efficient way to locate specific data records. Queries that involve indexed columns can benefit from reduced search time and improved query performance.
- **Efficient Data Filtering:** Indexes enable efficient data filtering by narrowing down the search space. By using index-based conditions in queries, the database engine can quickly identify the relevant data, resulting in faster response times.
- **Accelerated Sorting and Ordering:** Indexes facilitate speedy sorting and ordering of query results. When a query requires sorting based on an indexed column, the database can leverage the index structure to retrieve the data in the desired order more rapidly.
- **Enhanced Concurrency:** Indexing can improve concurrency in database systems. With indexes, multiple users can simultaneously access and modify the data without causing significant contention, as the indexed data is organized and accessed efficiently.
- **Increased Data Integrity:** Indexes can enforce data integrity constraints, such as primary key or unique key constraints. They help ensure that duplicate or inconsistent data entries are prevented or quickly identified.
- **Optimal Disk I/O Utilization:** By reducing the number of disk reads required to retrieve data, indexes optimize disk I/O utilization. This leads to more efficient usage of storage resources and improved overall system performance.

Disadvantages of Indexing

- **Increased Storage Space:** Indexes require additional storage space to store the index structure. This can result in increased disk space usage, especially for large databases with multiple indexes. The storage overhead of indexes should be considered when designing a database.
- **Overhead on Data Modification:** Indexes introduce additional overhead when modifying data. Inserts, updates, and deletes not only affect the data itself but also require updating the

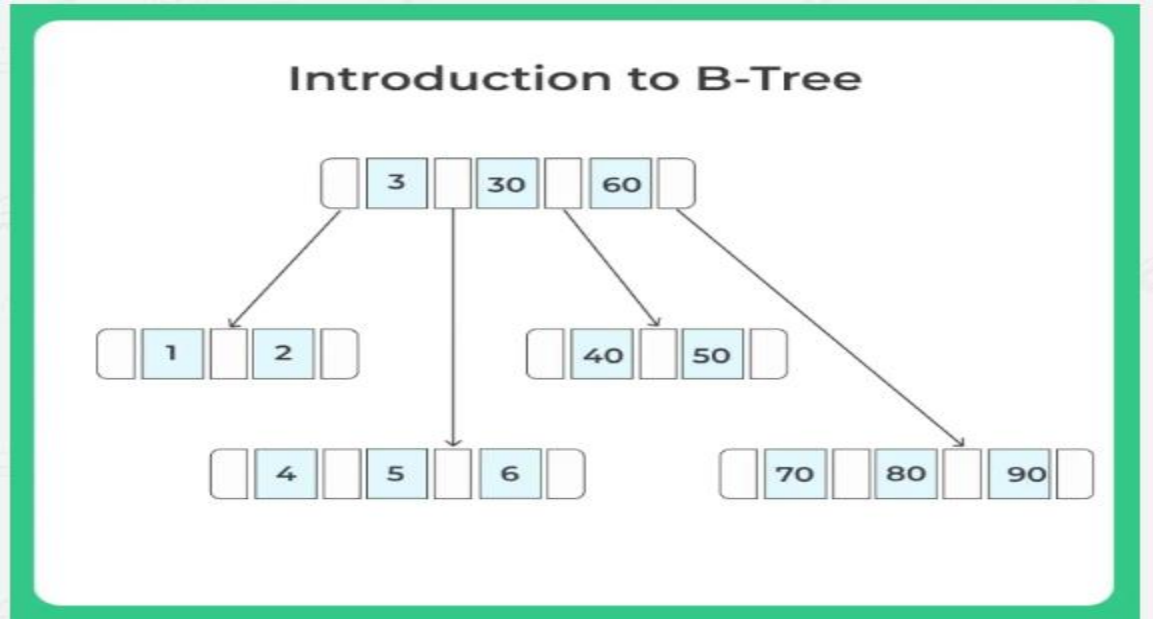
corresponding indexes. This can lead to slower data modification operations, especially on heavily indexed tables.

- **Index Maintenance Overhead:** Indexes require ongoing maintenance to remain accurate and effective. As data is modified, indexes need to be updated accordingly. This maintenance overhead can impact system performance, especially in scenarios with frequent data modifications.
- **Increased Complexity and Overhead on Data Loading:** When bulk loading or importing large amounts of data into a database, indexes can slow down the process. Each data insertion requires updating the associated indexes, resulting in increased complexity and longer data loading times.
- **Index Selection Overhead:** The database engine needs to evaluate and choose the appropriate indexes for query execution. In complex database systems with numerous indexes, the overhead of index selection and optimization can impact query performance and system response times.

Introduction to B-Trees

Introduction to B-Trees

B-Tree is a self-balancing search tree. Generally, a B-Tree node size is kept equal to the disk block size.As height is low for B-Tree, total disk accesses for most of the operations are reduced significantly compared to other search trees.



Properties of B-Tree

- All leaves are at same level.
- A B-Tree is defined by the term minimum degree ‘n’. The value of n depends upon disk block size.
- Every node except root must contain at least n-1 keys. Root may contain minimum 1 key.
- All nodes (including root) may contain at most 2n – 1 keys.
- Number of children of a node is equal to the number of keys in it plus 1.
- All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in the range from k1 and k2.
- B-Tree grows and shrinks from the root.
- Time complexity to search, insert and delete is O(logn).

Search

- Let the key to be searched be k.
- We start from the root and recursively traverse down.
- For every visited non-leaf node, if the node has the key, we simply return the node.
- Otherwise, we recur down to the appropriate child (the child which is just before the first greater key) of the node.
- If we reach a leaf node and don’t find k in the leaf node, we return NULL.

Traverse

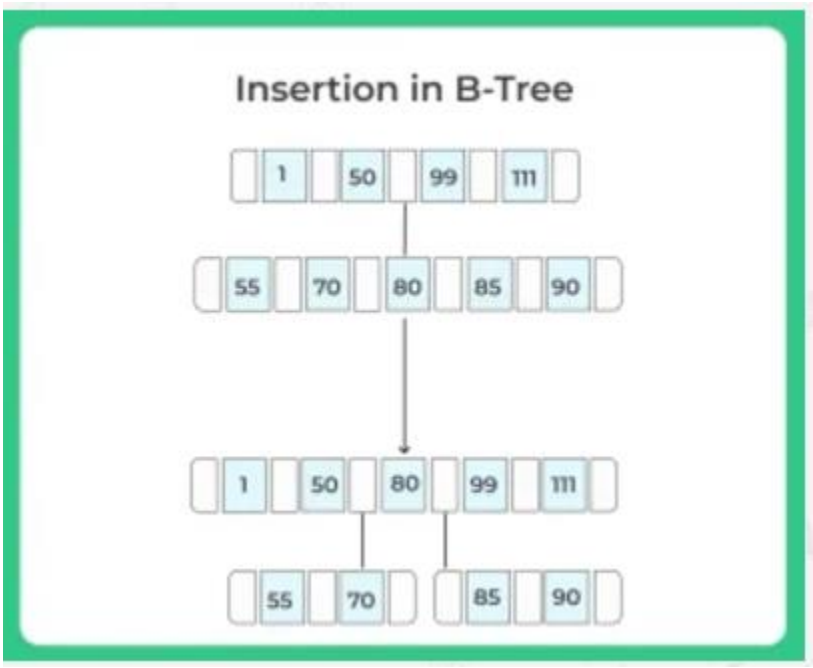
- Traversal is also similar to in-order traversal of Binary Tree.
- We start from the leftmost child, recursively print the leftmost child, then repeat the same process for remaining children and keys.
- In the end, recursively print the rightmost child.

Insertion in B-Tree

Insertion in B-Tree

In this article, we will learn about Insertion in B-Tree.

- In a B-Tree new key is always inserted at the leaf node.
- In B-tree, we start from the root and traverse down till we reach a leaf node.
- After we reach a leaf node, we insert the key in that leaf node.
- In this technique, we have a predefined range on the number of keys that a node can contain.
- So before inserting a key to the node, we make sure that the node has extra space.
- For this,we use an operation called splitChild() that is used to split a child of a node.



Algorithm for insertion in B-Tree

Step 1: Initialize x as root.

Step 2: While x is not leaf, do following

-
  - Find the child of x that is going to be traversed next. Let the child be y.
  - If y is not full, change x to point to y.

- If y is full, split it and change x to point to one of the two parts of y. If k is smaller than mid key in y, then set x as the first part of y else second part of y. When we split y, we move a key from y to its parent x.

Step 3: The loop in step 2 stops when x is leaf. x must have space for 1 extra key as we have been splitting all nodes in advance. So simply insert key to x.

Example

Consider a sequence of integers 10, 20, 30, 40, 50, 60, 70, 80 and 90 to be inserted in an initially empty B-Tree with minimum degree n=3

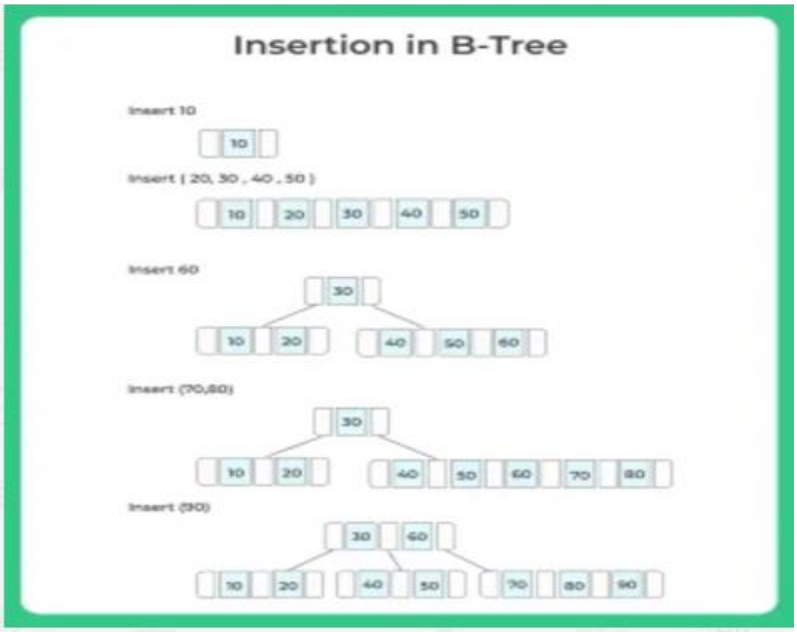
Step1: Initially root is NULL. Let us first insert 10.

Step2: Now insert 20, 30, 40 and 50. They all will be inserted in root because the maximum number of keys a node can accommodate is 2n–1 which is 5.

Step3: Now insert 60. As root node is full, it will first split into two, then 60 will be inserted into the appropriate child.

Step4: Let us now insert 70 and 80. These new keys will be inserted into the appropriate leaf without any split.

Step5: Let us now insert 90. This insertion will cause a split. The middle key will go up to the parent.

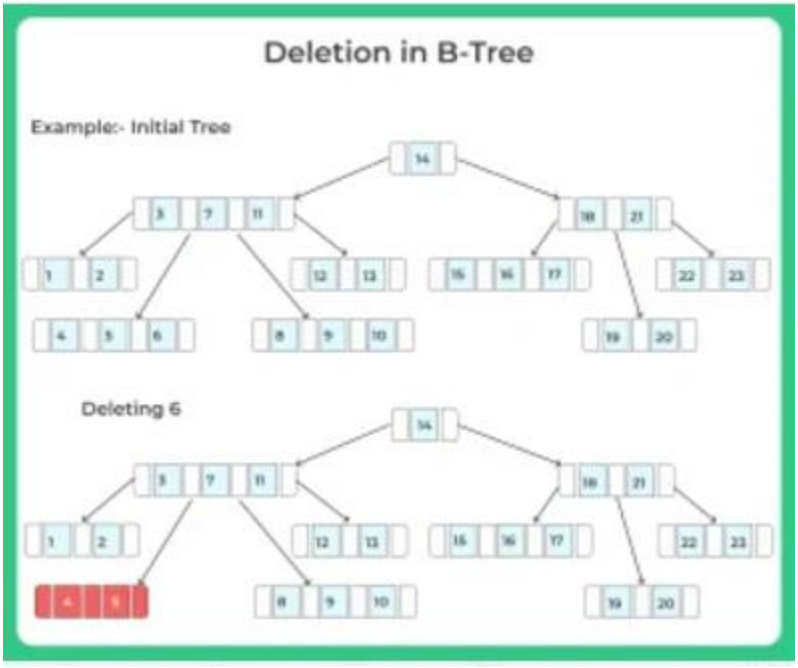


Deletion in B-Tree

Deletion in B-Tree

In this article, we will learn about Deletion in B-Tree.

- Deletion from a B-tree is more complicated than insertion
- We can delete a key from any node and when we delete a key from an internal node, we will have to rearrange the node's children.
- We must make sure that deletion process doesn't violate the B-tree properties.
- We must ensure that a node doesn't get too small during deletion (except that the root is allowed to have fewer than the minimum number n-1 of keys, where n is the degree).
- For deletion, we follow such an approach which might have to back up if a node along the path to where the key is to be deleted has the minimum number of keys.



Various Cases of Deletion

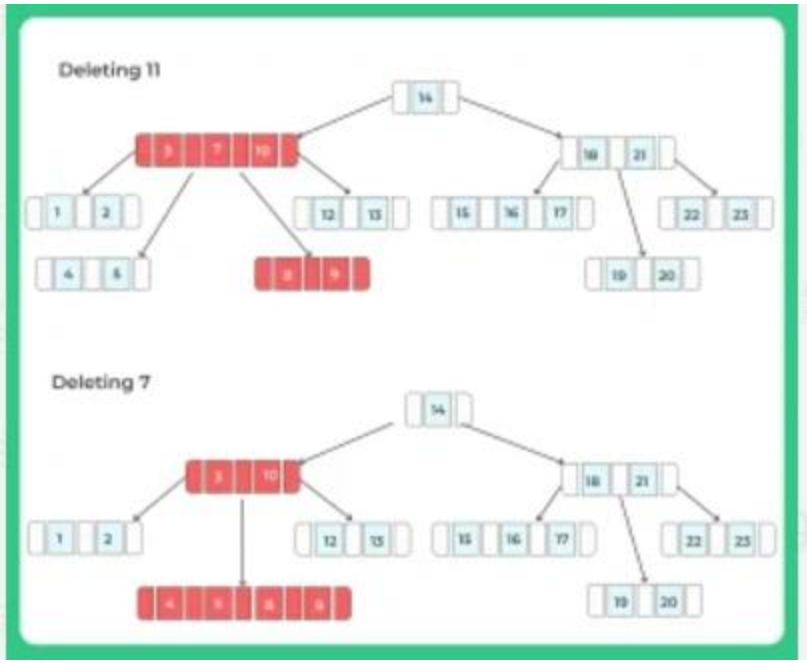
Case 1.

If the key k is in node x and x is a leaf, delete the key k from x.

Case 2

If the key k is in node x and x is an internal node, do the following.

- If the child y that precedes k in node x has at least n keys, then find the predecessor k0 of k in the sub-tree rooted at y. Recursively delete k0, and replace k with k0 in x. (We can find k0 and delete it in a single downward pass.)
- If y has fewer than n keys, then, symmetrically, examine the child z that follows k in node x. If z has at least n keys, then find the successor k0 of k in the subtree rooted at z. Recursively delete k0, and replace k with k0 in x. (We can find k0 and delete it in a single downward pass.)



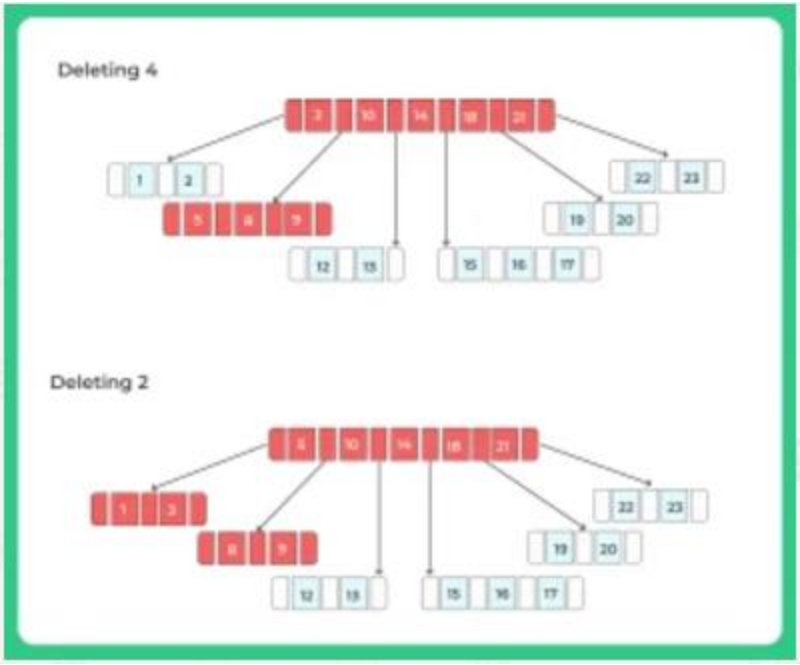
- Otherwise, if both y and z have only n-1 keys, merge k and all of z into y, so that x loses both k and the pointer to z, and y now contains 2n-1 keys. Then free z and recursively delete k from y.

Case 3

If the key k is not present in internal node x, determine the root x.c(i) of the appropriate subtree that must contain k, if k is in the tree at all. If x.c(i) has only n-1 keys, execute step 1 or 2 in this case as necessary to guarantee that we descend to a node containing at least n keys. Then finish by recursing on the appropriate child of x.

- If x.c(i) has only n-1 keys but has an immediate sibling with at least n keys, g*ive x.c(i) an extra key by moving a key from x down into x.c(i), moving a key from x.c(i) 's immediate left or right sibling up into x,*and moving the appropriate child pointer from the sibling into x.c(i).

- If x.c(i) and both of x.c(i)'s immediate siblings have n-1 keys, merge x.c(i) with one sibling, which involves moving a key from x down into the new merged node to become the median key for that node.



Hashing in DBMS

What is Hashing?

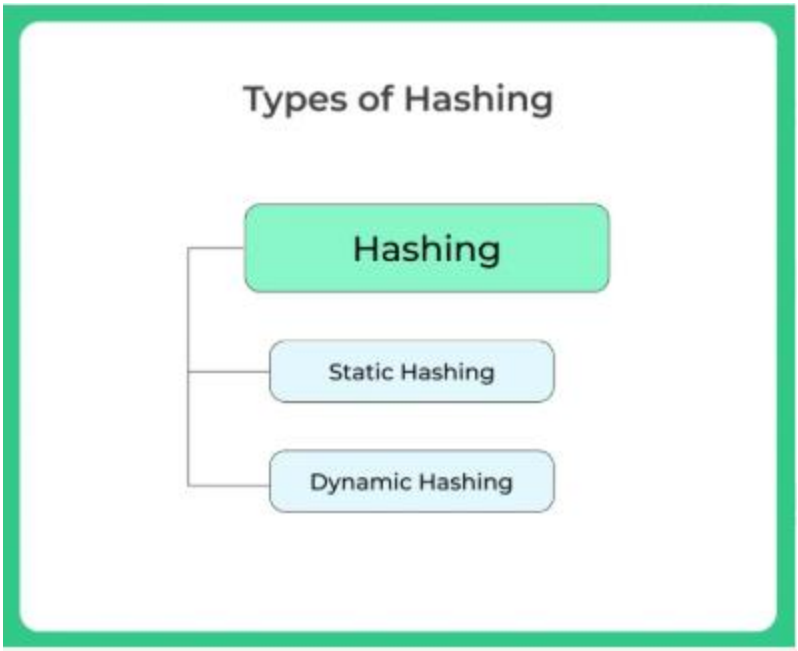
Hashing is an effective technique to calculate the direct location of the data record on the disk using a function key without using a sequential index structure as a result data retrieval time decreases.

Hashing in DBMS

In this article, we will learn about Hashing in DBMS.

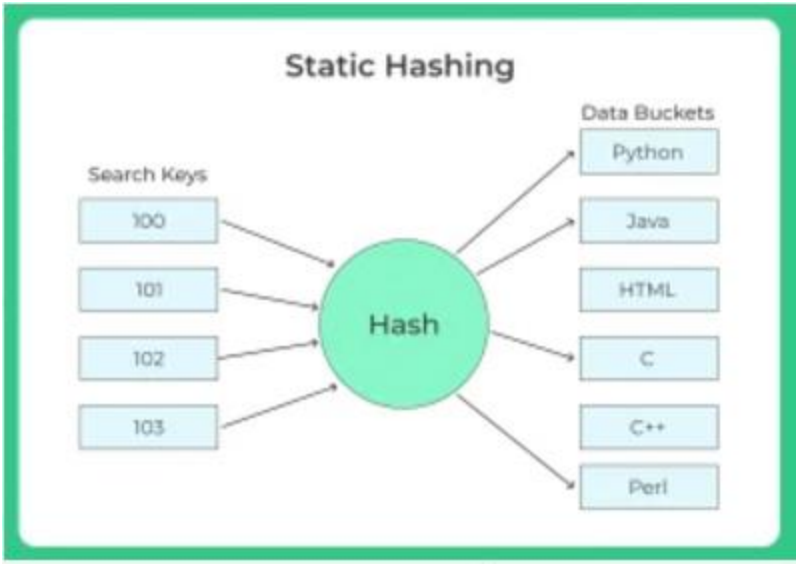
- In a huge database structure, it is difficult to search all index values sequentially and then reach the destination data block to get the desired data
- Hashing is an effective technique to calculate the direct location of the data record on the disk using a function key without using a sequential index structure as a result data retrieval time decreases.

- ***Bucket***: Hash file stores data in bucket format, a bucket is nothing but a unit of storage. Typically 1 complete disc block which in turn can store one or more records
- ***Hash Function***: a Hash Function returns a value which is nothing but the address of the desired data block



Static hashing

- In static hashing, ***a search key value is provided by the designed Hash Function always computes the same address***
- For example, if ***mod (4)*** hash function is used, ***then it shall generate only 5 values.***
- The ***number of buckets*** provided ***remains unchanged*** at all times
- ***Bucket address = h (K)***: the address of the desired data item which is ***used for insertion updating and deletion operations***

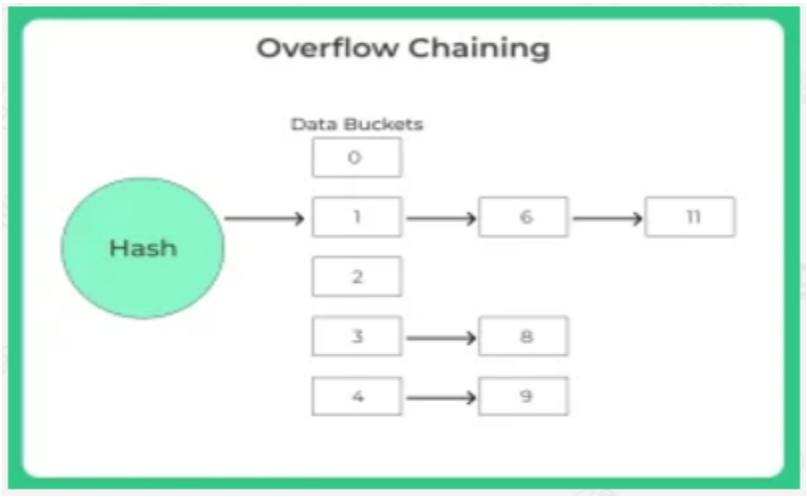


Collision and its resolution

- Collision is a condition when there is a ***fight between multiple data items for the same address***
- Simply collision ***arises when a hash function generates an address at which data is already stored***

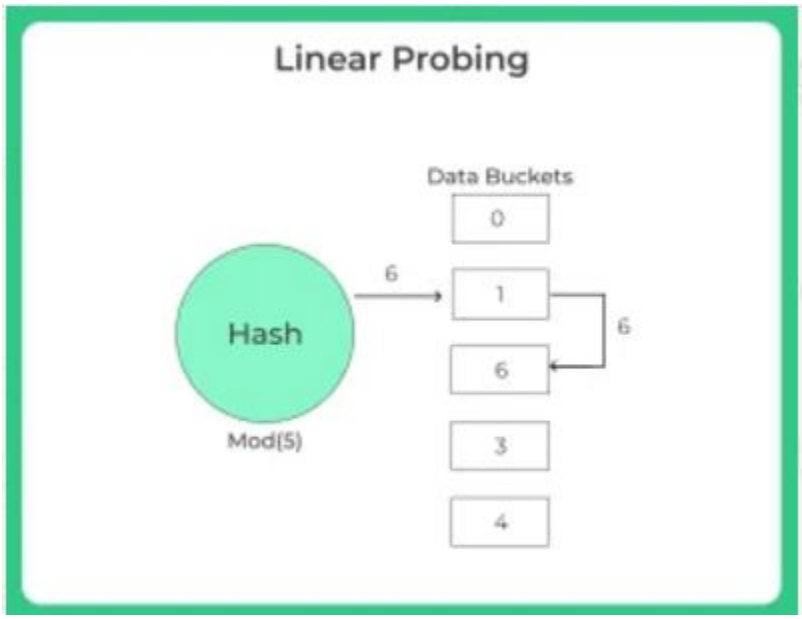
<u>Overflow chaining (closed hashing)</u>

- When buckets are full ***a new bucket is allocated*** for the same hash result which is then ***linked after the previous one***
- For example, if ***h (k) =3 and if some data is already present*** in this location, now ***a new bucket is linked to the existing location 3***



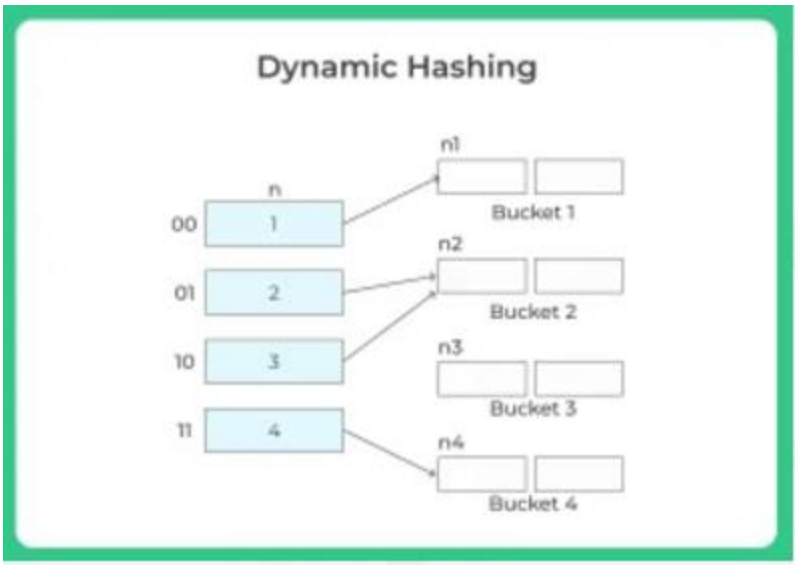
Linear probing (open hashing)

- Hash Function is generating a key at which ***data is already stored then the next free available bucket is allocated***
- For example, if ***h (k) =6 and is already occupied now check next possible available bucket that is free***



Dynamic hashing (extensible hashing)

- The ***drawback of static hashing*** is that it ***does not expand or shrink its size based on the requirement*** of the database
- Dynamic hashing ***provides a mechanism in which data buckets are added and removed*** dynamically and on-demand
- Generally, Hash Function in dynamic hashing is designed to ***produce a large number of values such that only a few of them are used at initial stages***



Organization

- The ***prefix of an entire hash value*** is taken as a ***hash index***.
- Only ***a portion*** of the hash value is ***used for computing bucket addresses***.
- ***Every hash index has a depth value*** to represent how many bits are used for computing a hash function.
- These bits can address ***2n buckets.***
- When all these bits are consumed i.e, ***when all the buckets are full then the depth value is increased*** linearly and twice the buckets are allocated.

Conclusion

- Hashing is ***not advantages when data is organized in some ordering*** and queries require a range of data.
- Hashing does ***best when the data is discrete and random***
- Hashing algorithms are having ***high complexity than indexing***
- All the ***hash operations are done at a constant time***