Date: 26/12/2017.
Tuesday.

Introduction

Data Base Management System (DBMS)

Data Base

* A data base is collection of interrelated data

DBMS

* DBMS is collection of interrelated data and set of application programs which are used to perform some operations on the data

eg: oracle, DB2, mysql etc

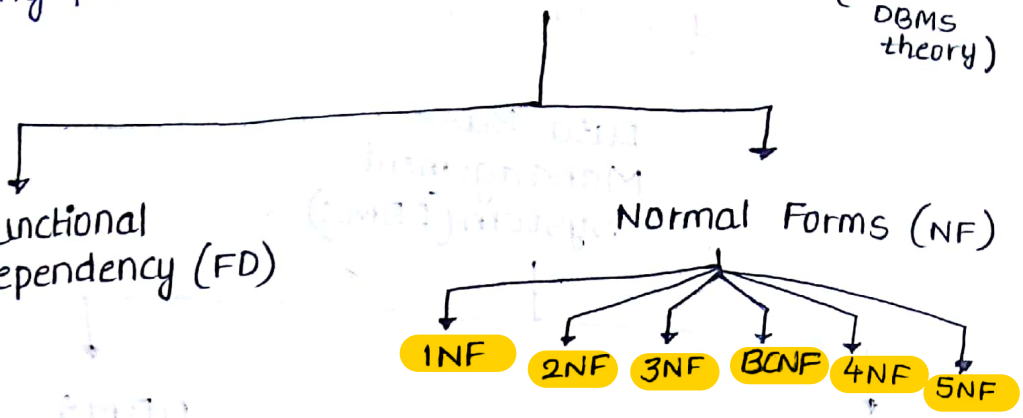
Data Modelling

* → Entity Relationship Model (pictorial representation)
** → Relational Model (All DBMS are RDBMS. commercial.)
→ Network Model
→ Hierarcheal Model
→ Object oriented Model.

Query Languages

- Mathematical Query Language
  - Relational Algebra
  - Relational calculus
    - Tupple (TRC)
    - Domain (DRC)
- Practical Query Language
  - Structured Query Language (SQL)

⇒ After designing DBMS, we conduct a testing process called "Normalization" (Its a DBMS theory)



⇒ Normalization is used to design efficient Data Base.

Transaction Management (TM)

- Transaction processing (TP)
- Transaction Management (TM)
- Transaction Failure (TF)
- Concurrent transactions (CCT)

Query Processor

- Query processing (QP)
- Query Evaluating (QE)
- Query Optimisation (QO)

Indexing

- Tree Based
  - ISAM
  - B Tree
  - B+ Tree
- Hash Based
  - → Extendible hashing

* Projects

1. Creat Data Base (other than student DB & ATM DB)
2. Download open source code (Identify & modify using efficient Algorithms)

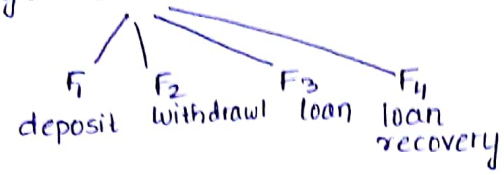
28/12

File systems Vs DBMS

⇒ Different files are meant for different purposes

⇒ No Data Redundacy*

Eg:- Bank



F_i → data + Applications

Drawback: customer information is repeating.

There is "Data Redundancy"

⇒ No Data Inconsistency

→ space complexity

→ Data inconsistency

Eg:- data must be updated in all file systems.

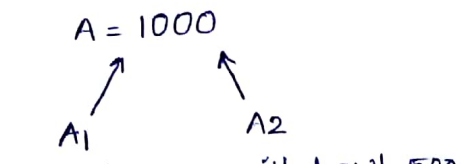
⇒ Ex:- if we want, the customers info whose balance is above 1,00,000 This is a new application.

It is difficult to develop a new application with the existing applications and integrating them.

⇒ No problem of Integrating new application with existing application.

⇒ Easy Data Access.

⇒



But none is true.

⇒ No concurrent execution problem.

∴ Concurrent execution problems.

⇒ A —500→ B
1000 2000

F_1 {
1) Read A
2) A.bal = A.bal - 500;
3) write A

F_2 {
4) Read B
5) B.bal = B.bal + 500
6) write B

Due to some bugs, OS problems if the transaction is failed after step 3, there is no recovery mechanism in file systems.

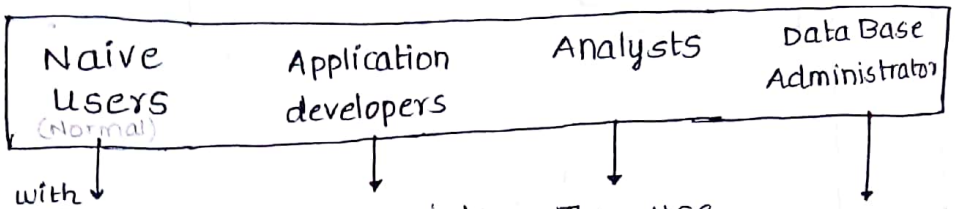
⇒ There is a data recovery mechanism in DBMS, during failures.

2/11/18

Architecture of DBMS

Components of DBMS

Data Base Management system users

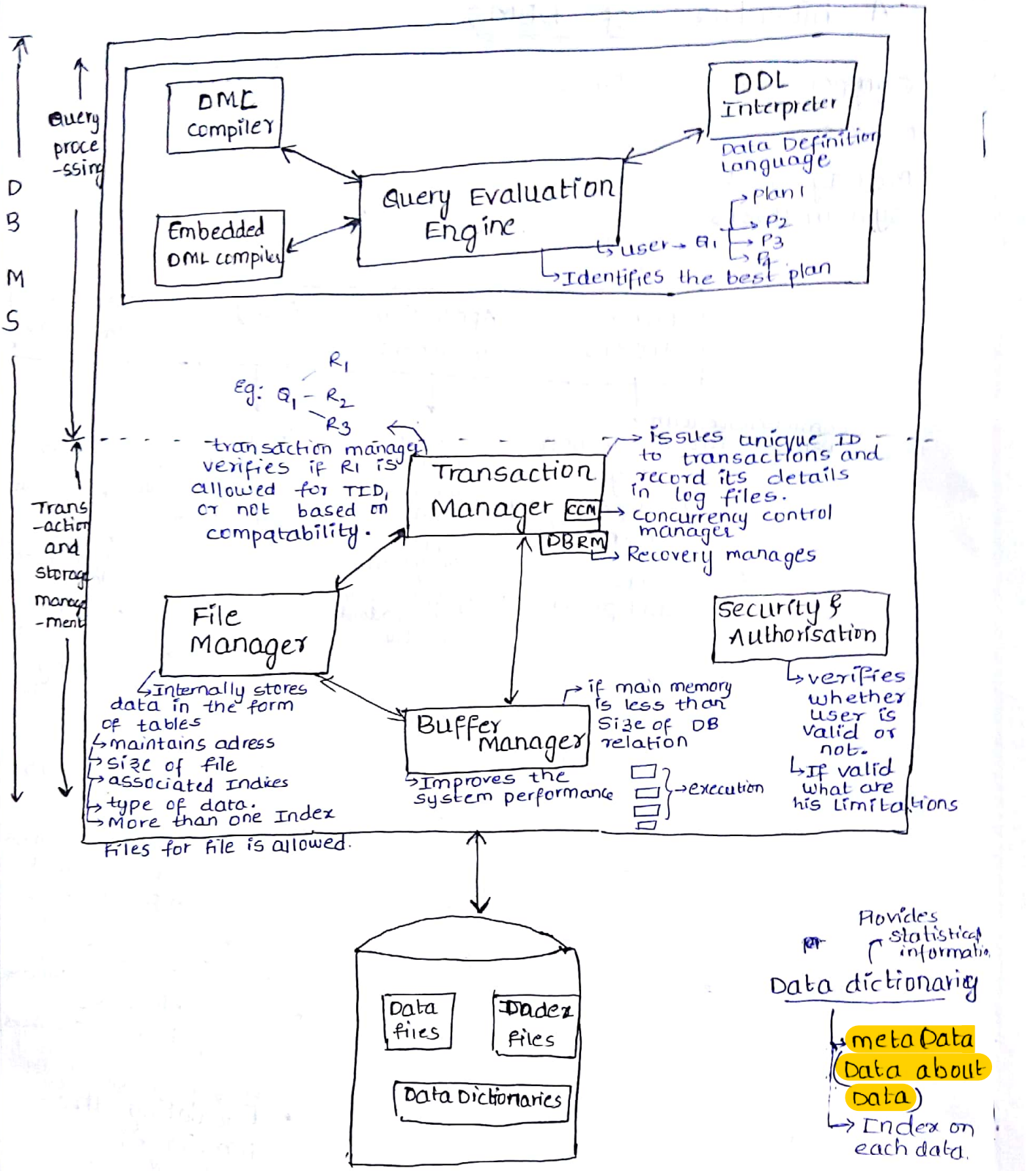


Naive users (Normal)
- communicate with DBMS with Application program developed by other users.
- Eg:- Students of Student portal

Application developers
- communicate with DBMS with their own application program.
- Eg: Coreteam of student portal

Analysts
- They use some 'tools' to communicate with DBMS

Data Base Administrator
- TOOLS.
- can creat DB
- creation of relations in database
- creation of Indices
- Adding users to DB.
- Delete user from DB.
- Eg:- MySQL
  - student ← DB1
  - Faculty ← DB2
  - admini ← DB3
- Grant previlages to different types of users
  - Eg:- U1, U2, U3, U4
  - ↓ Student, ↓ Faculty, ↓ HOD, ↓ Dean academic
- Revoke previous previlages.
  - Eg:- if HOD is changed, authentication should be given to new HOD.
- Esculating the previlages
  - Incrementing the previlages
  - Eg:- A admin making other user as admin.



Provides statistical information

Data dictionaries

- meta Data
- Data about Data
- Index on each data.

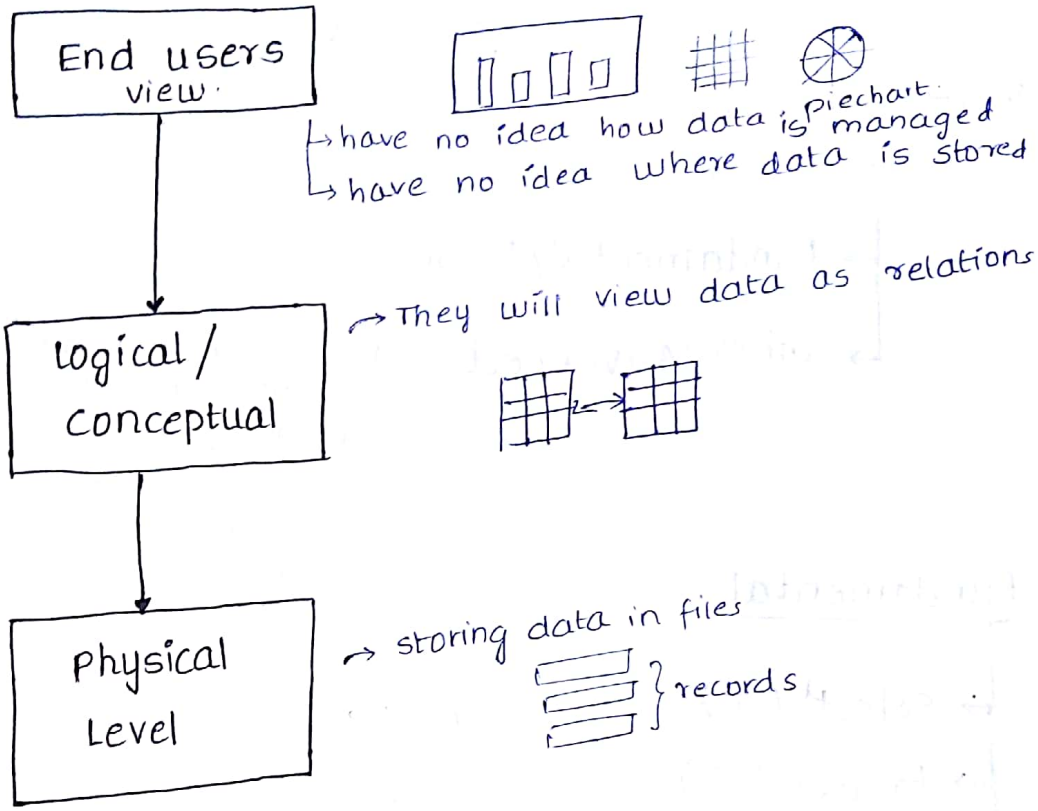
Eg: of statistical information

attribute, domain, size etc.

Statistical data present in data dictionaries → which helps the Query engine to prepare plans

Data Abstraction

3 - level schema :



Advantage of 3 level schema

⇒ Change of data representation at one level does not effect other levels. This is called 'data Independency'.

⇒ Hiding the complexities at different Level.

Query language
- Mathematical
  - Relational Algebra
  - Relational Calculus
    - Tuple
    - Domain
- structured Query language

Relational Algebra

- Fundamental/Basic
- other/Advanced. (we can use fundamental operations to develop new operations)

Fundamental

- Select (σ)
- Project (π)
- cartesian product ($\times$)
- Set union ($\cup$)
- set Difference ($-$)
- Rename (ρ)

Advanced

- Join
  - NJ (Natural)
  - TJ (Theta)
  - outer join
    - Left
    - Right
    - full
- set instruction.
- Database Modification
  - Insert
  - delete
  - update
- Aggregate operations

Tuple:- A horizontal row in a relation.

Select (σ) :-

It divides horizontally, the given relation.

→ only from main memory not from DB

it deletes those tuple which doesn't satisfy given condition and outputs the remaining

$\sigma_{\theta}(Rel)$

Rel → original relation in D/B (or) Result of a R.A expression.

θ → Simple / Compound.

(condition) $\theta = \theta_1 \wedge \theta_2 \wedge \theta_3$

$\theta = \theta_1 \vee \theta_2$

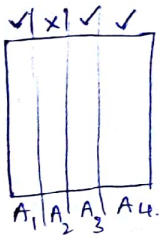
Eg:- $\sigma_{m>50 \wedge d='CSE'}(Student)$

its equivalent structured query

```
SELECT *
FROM student
WHERE m>50
AND d='CSE';
```

Project (Π) :-

Attribute: A column in a relation.



Π_{A_1, A_3, A_4} → It projects only those attributes mentioned.

$\Pi_{SNO}(Student)$

Selection followed by Projection :-

$$\Pi_{sno}\left(\sigma_{m>50}(student)\right)$$

⇒ outputs only student roll no whose marks > 50 from 'student' relation

SELECT sno FROM student WHERE m > 50;

$t_1 \in r_1$

$r_1 = \sigma_\theta(R)$

$r_2 = \Pi_{A_1 A_2 A_3}(R)$

no. of tuples in r_1 → min = 0, max } depends on number of condition satisfied tuples

no. of tuples in r_2 = max

no. of Attributes in r_1 = max

no. of attributes in r_2 = 3

Projection followed by selection.

$$\Pi_{SNO,SName}\left(\sigma_{m>50}(student)\right) \equiv \Pi_{SNO,SName}\left(\sigma_{m>50}\left(\Pi_{SNO,SName\ U\ m}(student)\right)\right)$$

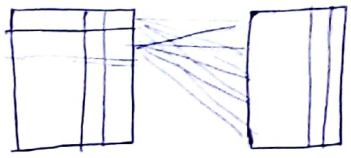
* Note :-

$$\Pi_{A_1,A_2}\left(\sigma_\theta(R)\right) \equiv \Pi_{A_1,A_2}\left(\sigma_\theta\left(\Pi_{A_1,A_2 U Attr(\theta)}(R)\right)\right)$$

Cartesian product :-

$DB = \{R_1, R_2, \ldots, R_n\}$

When we require some portion from R_1 & some from R_2 first combine both R_1, R_2 using cartesian product



$\gamma = R_1 \times R_2$
$(m) \quad (n)$

- no. of tuples in $\gamma = m * n$

$t_1 \in \gamma$

- Size of t_1 is = size $T_1 \in R_1$ + size of $T_2 \in R_2$ (Attributes)

Eg:1 $R_1(ABC)$
$R_2(DE)$

$R_1 \times R_2 =$

A	B	C	D	E

Eg:2 $R_1(ABC)$
$R_2(CDE)$

$R_1 \times R_2 =$

A	B	C	C	D	E

Set operations :-

Union :

$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{1, 3, 5\}$

$S_1 \cup S_2 = \{1, 2, 3, 4, 5\}$ → eliminates duplicates (Internal Algorithm)

→ if we need duplicates perform union all.

Pre-conditions to perform set operations

1) Arity (no. of attributes) should match.

2) Domain value of i^{th} attribute in R_1 should match with Domain value of i^{th} attribute in R_2.

$R_1 = \{A_1, A_2, A_3\}$

$R_2 = \{X_1, X_2, X_3\}$

s.no s.no ✓

s.no Id ✗

s.name s.name ✓

Set difference (−) :-

$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{1, 3, 5\}$

$S_1 - S_2 = \{2, 4\}$ → elements in S_1 but not in S_2

Rename (ρ) :

$\rho_{Newname}(oldname)$

$\rho_{X(A_1, A_2, A_3)}(R)$

R → original relation in D.B
→ Result of relational algebra expression.

Eg :- $\rho_{Result}(\Pi_{sno, grad}(Student))$

Intersection (∩) :

$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{1, 3, 5\}$

$S_1 \cap S_2 = \{1, 3\}$

$S_1 \cap S_2 = S_1 - (S_1 - S_2)$

join ($\bowtie$) :-

$$R_1 \bowtie_{JC} R_2 = \sigma_{\theta}(R_1 \times R_2)$$

Eg :- $R_1 = (A, B, C)$

$R_2 = (C, D, E)$

$$R_1 \bowtie_{R_1.C = R_2.C} R_2 = \sigma_{R_1.C = R_2.C}(R_1 \times R_2)$$

R_1 = (SNO, SName, Add)

R_2 = (SNO, marks, grade)

$$\Pi_{SName, marks}\left(R_1 \bowtie_{R_1.SNO = R_2.SNO} R_2\right) = \Pi_{SName, marks}\left(\sigma_{R_1.SNO = R_2.SNO}(R_1 \times R_2)\right)$$

$r = R_1 \bowtie_{JC} R_2$

no. of attributes in r = no. of attributes in R_1 + no. of attributes in R_2.

no. of tuples in r → min = 0.

→ max = min(R_1, R_2)

$\bowtie$ → Natural join (includes only one attribute of duplicate attributes)

→ Theta join

Equivalent Query in SQL

```
SELECT Snam, marks
FROM   R1, R2
WHERE  R1.SNO = R2.SNO;
```

(or)

```
R1 Natural join R2
```

Aggregate Operations

- Min
- Max
- Avg
- Sum
- Count

* group by Attribute $\mathcal{G}_{\text{Aggregate operation}}$ (Rel)

$$_{\text{dept}}\mathcal{G}_{\max(\text{sal})}(\text{emp})$$

* Generalised Projection.

$$\Pi_{A_1 F_1 A_2 F_2}(R)$$

Eg: $\Pi_{SNO, m+5}(\sigma_{m<30}(\text{student}))$

Adding 5 marks to students having marks less than 30.

10/1

Database Modification

1. Insert
2. Delete
3. Update.

} advanced → using basic operations.

Aggregate Operations

- Min
- Max
- Avg
- Sum
- Count

* group by Attribute $\mathcal{G}_{\text{Aggregate operation}}$ (Rel)

$$_{\text{dept}}\mathcal{G}_{\max(\text{sal})}\ (\text{emp})$$

* Generalised Projection.

$$\Pi_{A_1 F_1 A_2 F_2}\ (R)$$

Eg:- $\Pi_{\text{SNO}, m+5}\left(\sigma_{m<30}(\text{student})\right)$

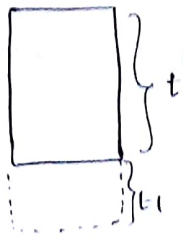
Adding 5 marks to students having marks less than 30.

Database Modification

1. Insert
2. Delete
3. Update.

} advanced ↓ using basic operations.

Insert :- R_1 (union)



$= t + t_1$

$R_1 \leftarrow R_1 \cup t_1$

Delete :- (1) search (set difference)
(2) Delete

$R = t$

after search t_2
after delete $t - t_2$

$t_1 \leftarrow \sigma_\theta (R_1)$
$R_1 \leftarrow R_1 - t_1$
} deletion.

Update : delete & insert

$t_1 \leftarrow \sigma_\theta (R_1)$
$R_1 \leftarrow R_1 - t_1$
} delete
$t_2 \leftarrow t_1'$
} insert
$R_1 \leftarrow R_1 \cup t_2$

— x —

Eg:- Bank : DataBase

customer (cid, cname, Addrs)
Account (ACNO, bal, bid)
Loan (lno, amt, bid)
Depositor (Cid, Acc NO)
Borrower (cid, lNO)
Branch (Bid, Bname, loc)

1. Find all customer id's those who are having an account at 'ABC' branch.
2. Find all customer Names those who are having an accout at 'ABC' branch.
3. Find all customer Name those who are having an account and a loan in the bank.

1. Select cid

From

$$\Pi_{cid}\left(\left(\sigma_{S.bform='ABC'}\left(\rho_S\left(\rho_P\left(\rho_\gamma\left(loan \bowtie_{loan.bid=Bran.bid} Branch\right) \bowtie_{\gamma.lno=Borrower.lNO} Borrower\right)\right)\right)\right) \bowtie_{P.cid=cust.cid} Customer\right)$$

4. Find all customers names those who are having an account in the bank.
5. similarly a loan in the bank.

4) Select distinct cname
From customer natural join Depositor.

$$\Pi_{cname}\left(Customer \bowtie_{Customer.cid=Depositor.cid} Depositor\right)$$

5) $\Pi_{cname}\left(\left(Customer \bowtie_{cust.cid=Bor.cid} Borrower\right)\right)$

```
select distinct cname
from customer natural join
Borrower;
```

③ $\left(\Pi_{cname}(customer \bowtie_{cust.cid=Depos.cid} Depositor) \cup \Pi_{cname}(customer \bowtie_{co.cid=B.cid} Borrower)\right)$

•• Natural join (or) Theta join retrieves the data that is common in both the relations and

∴ loss of the data. so, to overcome this we use outer join.

Eg:-

R_1

A	B	C
1	b_1	c_1
2	b_2	c_2
3	b_3	c_3
4	b_4	c_4
5	b_5	c_5

R_2

D	E
1	e_1
3	e_3
5	e_5

R_1 join R_2 (Theta)

A	B	C	D	E
1	b_1	c_1	1	e_1
3	b_3	c_3	3	e_3
5	b_5	c_5	5	e_5

Outer join :- It is an extension of Natural join.

It join all tuples from R_1 OR R_2 OR BOTH without missing any tuples.

Outerjoin

↳ left outer join : ⟕ $R_1 ⟕ R_2$

all tuples from R_1

Eg:

R_1

A	B	C
1	b_1	c_1
2	b_2	c_2
3	b_3	c_3
4	b_4	c_4

R_2

D	E
1	e_1
3	e_3
5	e_5

$l =$ $R_1 ⟕ R_2$

A	B	C	D	E
1	b_1	c_1	1	e_1
3	b_3	c_3	3	e_3
2	b_2	c_2	null	null
4	b_4	c_4	null	null

Right outer join : ⟖

$r = R_1 ⟖ R_2$

A	B	C	D	E
1	b_1	c_1	1	e_1
3	b_3	c_3	3	e_3
null	null	null	5	e_5

Full outer join : ⟗

$f = R_1 ⟗ R_2$

A	B	C	D	E	F
1	b_1	c_1	1	e_1	
null	null	null	5	e_5	
3	b_3	c_3	3	e_3	
2	b_2	c_2	null	null	
4	b_4	c_4	null	null	

$t_1 \in l$ → no. of tuples = no. of tuples in left side relation

$t_2 \in r$ → no. of tuples = no. of tuples in right side relation

$t_3 \in f$ → no. of tuples = no. of tuples in left side + no. of tuples in right side − no. of tuples ($R_1 ⋈ R_2$)

natural join

VIEWS

↓ SQL

create view view-name as <R.A expression>

Eg: create view V_1 as $\Pi_{A_1,A_2,A_3}(\sigma_D(R_1 \bowtie R_2))$

Database Design life cycle

SDLC
(subway design life cycle)

- Problem definition
- Analysis of Problem
  - type
  - complexities
  - possible solutions
- Design
  - Algorithm
  - Flow Diagram

DBDLC
(Database Development for life cycle)

- Define the problems for DB
- Problem Analysis
  - Identifying the entities (obj) → (relation)
  - Identifying the properties of an entities → (attributes)
  - Identify the relationship exists b/w entities
  - type of data, user, facilities, system (app)
- Design → pictorical representation of DB
  - Modelling
    - ER model
      - NO DBMS to implement it. So convert → Relational model (Tabular)
      - All, current DBMS's are RDBMS
        - $(R_1, R_2, \ldots R_n)$ in acceptible forms
          - Size of relations (no data redundancy)
          - Insert, Delete, Update } problems
          - Testing these anomalies { - 1NF, - 2NF, - 3NF, - BCNF } Normalisation

- Implementation

- Testing

- Implementation
  (i.e., creation of tables using any DBMS)
- Testing
  (insertion, deletion, retrieval).

22/1/18

RELATIONAL CALCULUS

⇒ Non procedural query Language where as Relational algebra and SQL are procedural query languages

Procedural QL
↳ To perform operation on Data.
we write procedure.

Relational calculus
- non procedural QL
- TRC → based on tuples.
- DRC → based on domain value.

TRC :-

$\{t / P(t)\}$
↓ set of output tuples
↳ Predicate of relation.

eg:- 1. $\{t / t \in \text{student}\}$ ≡ select * from student
$\equiv \Pi_{\text{All attributes}}(\text{student})$

2. $\{t / t \in \text{student}(t[m] > 70)\}$ ≡ select * from student where marks > 70 ;

$\equiv \Pi_{\text{All-attributes}}(\sigma_{m>70}\ \text{student})$

** no. of tuples → 0, m (size of relation)

To output only specific attributes.

3. $\{t / \exists t,s \in \text{student}(s[\text{SNO}] = t[\text{SNO}] \wedge s[m] > 70)\}$

O/P:- S.NO.
=
=
=

4) $\{ t \mid \exists s, t \in \text{student} (s[SNO] = t[SNO] \wedge s[Name] = t[Name] \wedge s[m] > 70) \}$

o/p

SNO	Name

5) combining two relations

student (SNO, Name, m)

Reg (SNO, dept).

$\{ t \mid \exists s \in \text{student} (s[SNO] = t[SNO] \wedge s[m] > 70 \wedge s[Name] = t[Name])$

$\wedge \exists r \in Reg (r[SNO] = t[SNO] \wedge r[dept] = 'CSE' \wedge r[dept] = t[dept]$

$\wedge \quad s[SNO] = r[SNO] \}$

o/p

SNO	Name	dept

DRC :-

$\{ \langle A_1, A_2, \ldots A_n \rangle \mid Q(\langle A_1, A_2, \ldots, A_n \rangle) \}$

↱ output Attributes

↱ Attributes pasticipating in query condition ∪ output attributes.

eg 1.

Student (SNO, Name, Marks)

$\{ \langle SNO, Name, Marks \rangle \mid \langle SNO, Name, Marks \rangle \in \text{student} \wedge marks > 70 \}$

≡ select * from student where marks > 70 ;

2. $\{ \langle SNO \rangle \mid \exists \langle SNO, marks \rangle \in \text{student} \wedge marks > 70 \}$.

≡ select SNO from student where marks > 70

3. combining two relations.

Reg(S, d)

Student(S, N, M)

$$\{ \langle S, N, d \rangle \mid \exists t_1 \langle S, N, M \rangle \in \text{student} \wedge M > 70 \wedge \exists t_2 \langle S, d \rangle \in \text{Reg} \wedge d = \text{'CSE'} \wedge t_1[S] = t_2[S] \}$$

— x — x —

In both TRC & DRC

Select customers ids who have an account and loan in the bank.

Given

dep(ACNO, cid)

Bor(LNo, cid).

1) TRC

$$\{ t \mid \exists r \in \text{dep}(t[cid] = r[cid]) \wedge \exists s \in \text{Bor}(t[cid] = s[cid]) \wedge r[cid] = s[cid] \}$$

2) DRC

$$\{ \langle cid \rangle \mid \exists r \langle cid \rangle \in \text{dep} \wedge \exists s \langle cid \rangle \in \text{Bor} \wedge r[cid] = s[cid] \}$$

DATABASE DESIGN

* ER model is a pictorial representation of database.

Database Design
- Database Modelling
  - Data Models
    - object based
      - object-oriented (class & objects)
      - * Entity-Relationship Model
    - Record-based (works on set of records)
      - * Relational
      - Network
      - Hierarchial

⇒ object is a real life entity which is distinguishable from other with a set of properties.

⇒ ER Model.
- Works on entities (objects)
- Attributes (properties)

ER Model
- Entity → eg:- S1 101 xyz.
  - an object
  - Attributes (properties) eg:

Entity set : set of similar entities. → eg:- S1, S2, S3

Types of Attributes

1. Simple Attribute :- Value of an attribute is constructed from a single component. No subparts in it.

   eg:- Gender → male, Female

2. Composite Attribute :- value of an attribute is constructed from more than one Components. subparts will be there.

Eg:- Name → First name + Middle name + Last name

Address → Door no + street name + country

3. Single value Attribute :-
only one value for each entity.

Eg:- Rollno of student.

4. Multivalue Attribute :-
More than one value for the same entity.

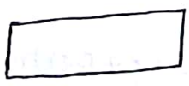
Eg:- Subjects registered by a student.

phonenumbers

5. Derived Attribute :-
If the value of an attribute is derived from other attribute.

Eg:- Age derived from D.O. Birth
Experience " " D.O. join.
Total marks of a student in S1 + S2 + S3

Symbols for an ER model

1) Entity set  (rectangle)

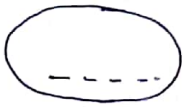
2) Attribute -  (ellipse)
(simple, composite, Single value)

3) Multivalue Attribute -  (double ellipse)

4) Derived Attribute -  (dashed ellipse)

5) primary key Attribute -

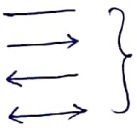
6) Descriminator -



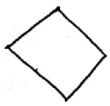
(an attribute part of weak entity set)

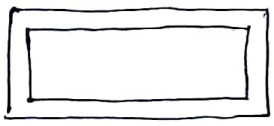
Lines:

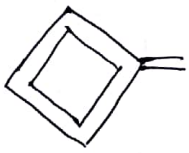
—— between Entity and its attributes.



between two entities with a Relationship.

Relationship:  (diamond)

Weak-Entity set:  (double rectangle)

Total participation. 

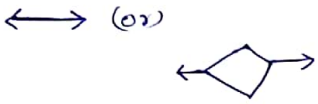
Relationship:

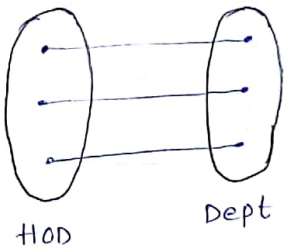
It is an association among entities.

Relationship set: set of similar relations

Types of Relationset.

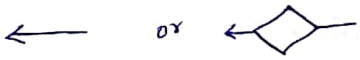
1) one - to - one

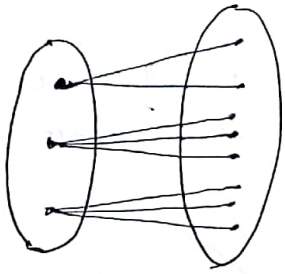




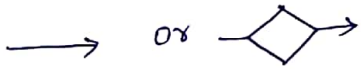
only one faculty is HOD for one Dept, and only one HOD for one dept.

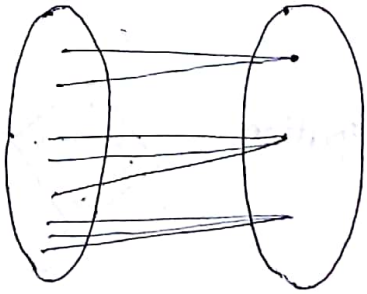
2) one - to - many



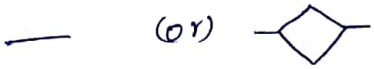


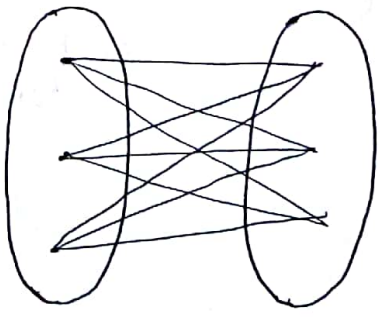
3) Many to one





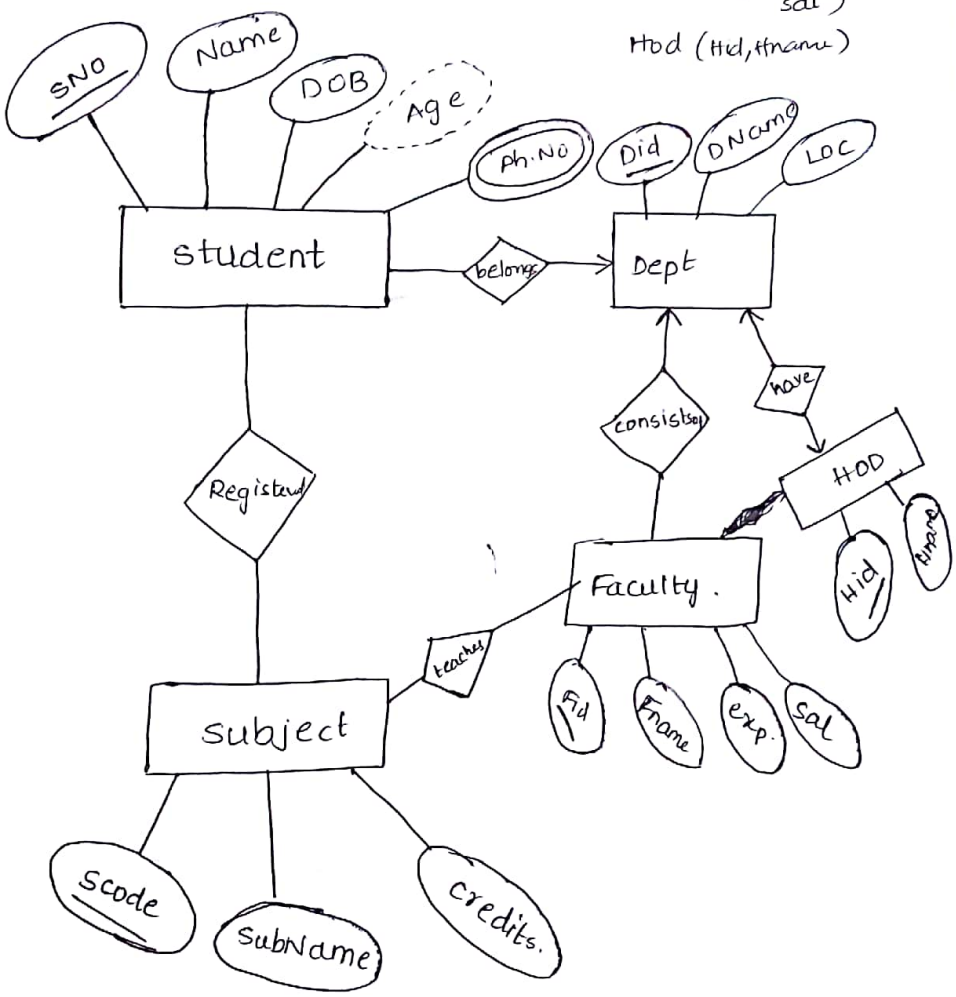
4) Many to many





Example ER model for student database.

Dept (Did, DName, loc)
Lab (Lid, Lname, LL)
Faculty (Fid, Fname, exp, sal)
Hod (Hid, Hname)

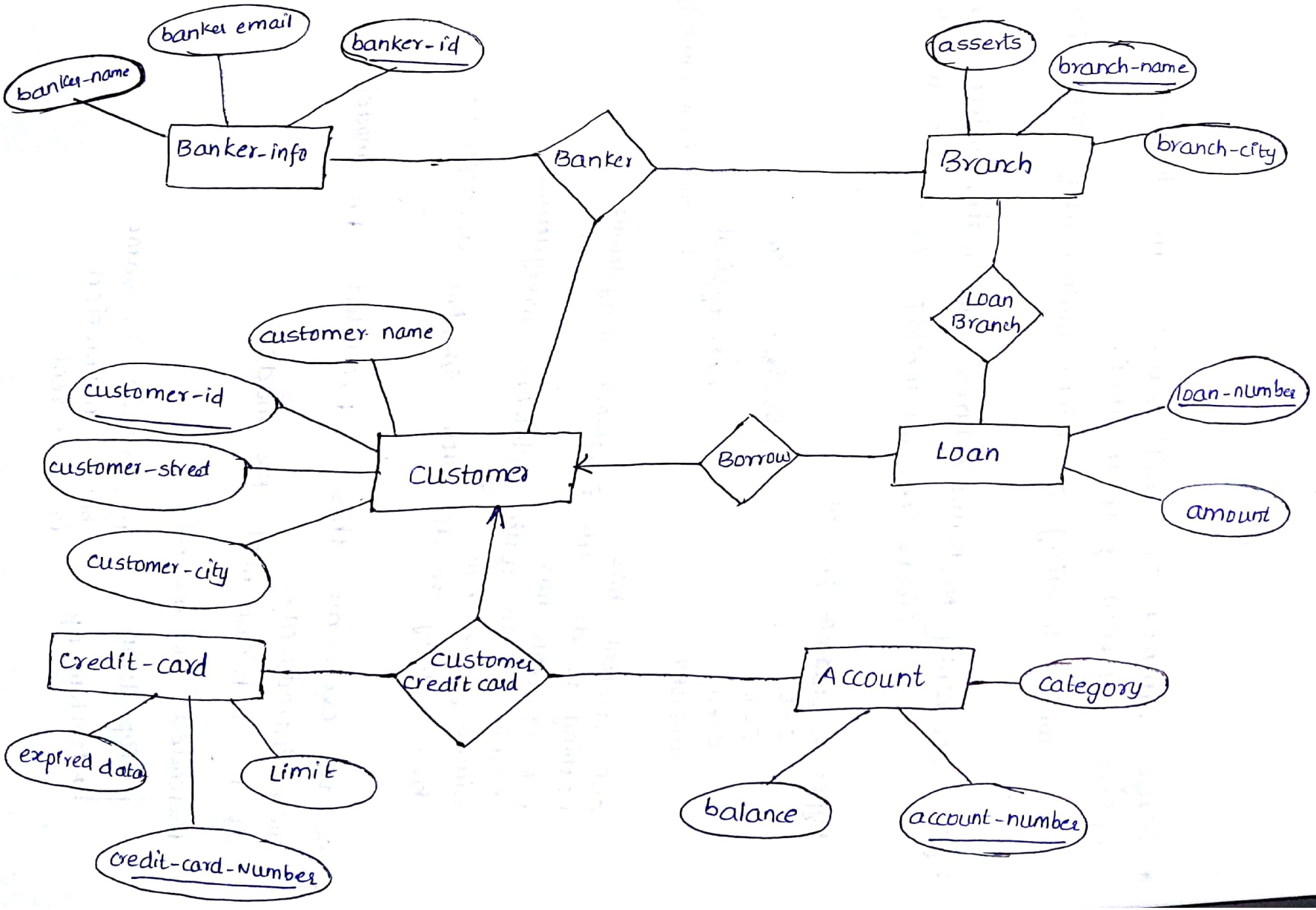


Assumptions :-

1. many students may belong to each department, where as each student cannot belong to more than one department
2. Each subject can be registered by many students, each student can register to more than one subject.
3. Each subject can be taught by more than one faculty and each faculty can teach more than one subject.
4. Each faculty belong to only one department, one department can have more than one faculty.

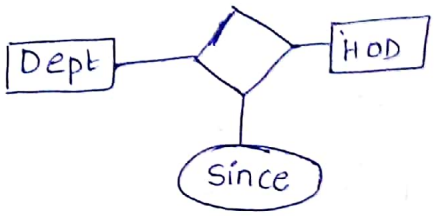
5) Each department will have only one HOD, and each HOD will be there for each department

ER Model. Bank database



Note:- joining attribute to a related entity.

* This attribute, ~~indi~~ separetly with each relation doesn't ~~make~~ seem meaningul.



* Large number of 'null' values in database effect system performance.

like while loading data into main memory lack of space etc.

Eg:- Student (SNO, Name, Address, practicals, projectwork, fieldwork, s/w develop, h/w develop, surveydetails)

types of students present

CSE
ECE
CIVIL
HISTORY

$S_1, S_2, S_3, ----S_n$

non typical

↳ so first classify

CSE students has nothing to do with fieldwork, survey details

ECE students has nothing to do with fieldwork, survey details

similary CIVIL & HISTORY has nothing to do with s/w, h/w development.

* So, to overcome this Extended ER model was introduced.

It is a hierarchial ER model.

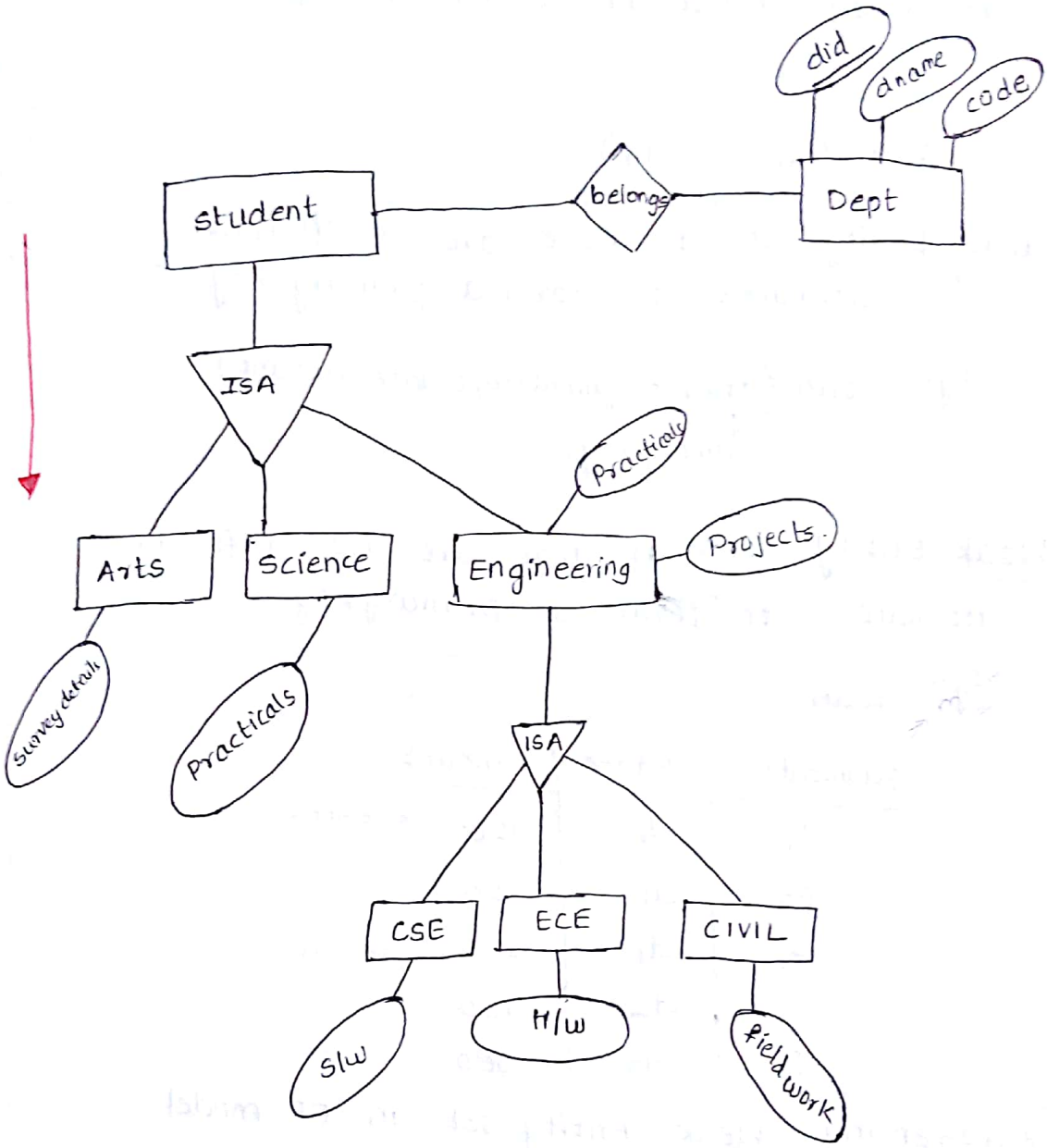
Extended ER model

- Top-down
- Bottom up

} appearance is same but construction is different

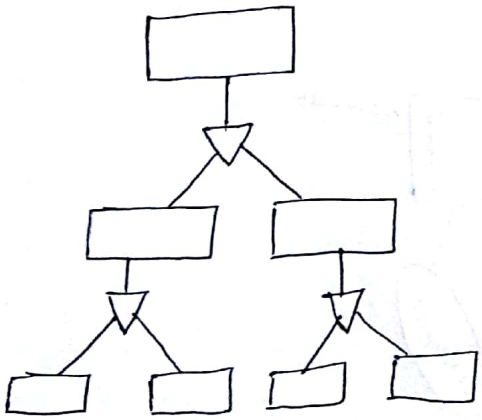
Top-down method

⇒ "Specialization"



Bottom-up method

⇒ "Generalization"



Key attribute: one or more attributes which (Primary key) uniquely identifies a tuple.

* Primary key should not be Null & should be unique.

$E_1 (a_1, a_2, a_3, ---- a_n)$

* Strong Entity set: If there are sufficient attributes to form a primary key.

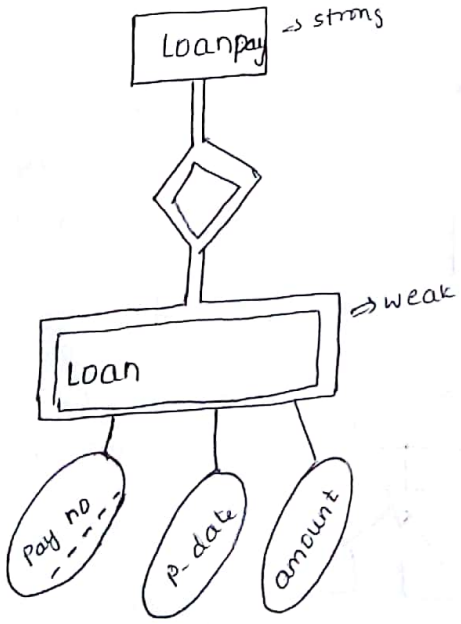
Eg:- Loan (Lno, paymentno, p-date, amount)
↓
primary key.

* Weak Entity set :- If there are no sufficient attributes to form a primary key.

Eg:- Loan

paymentno	Pdate	amount
P_1	d_1	100 → x person's
P_2	d_1	200
P_1	d_1	300 → y person
P_1	d_2	400
P_2	d_2	500

Representing Weak Entity set in ER model



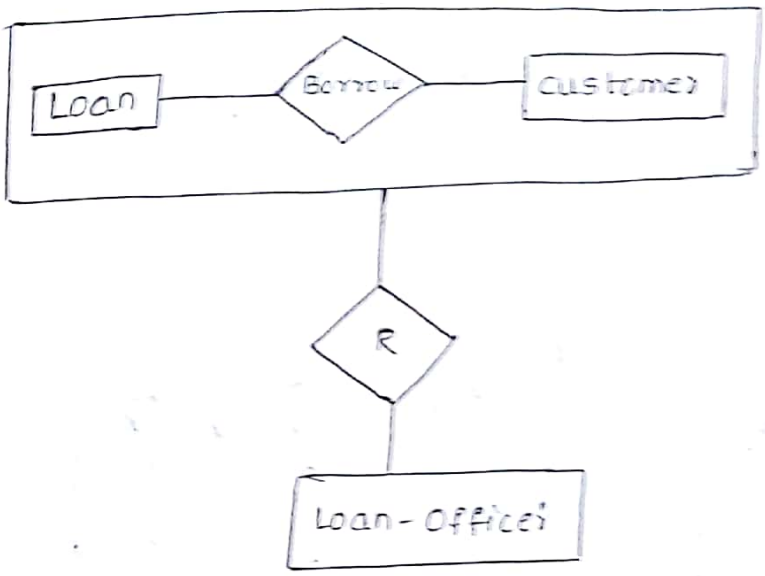
Descriminator:

It is an attribute, addition of it to a weak Entity set changes it to a strong Entity set.

eg:-

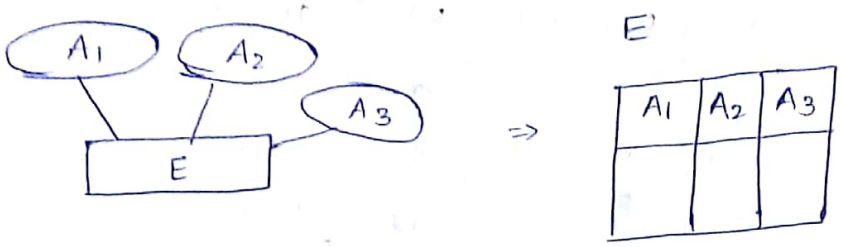
LNO	Payment no	pdate	amount
L_1	P_1	d_1	100
L_3	P_2	d_1	200
L_2	P_1	d_1	300
L_4	P_1	d_2	400
L_1	P_2	d_2	500
L_2	P_2	d_2	600

Aggregation



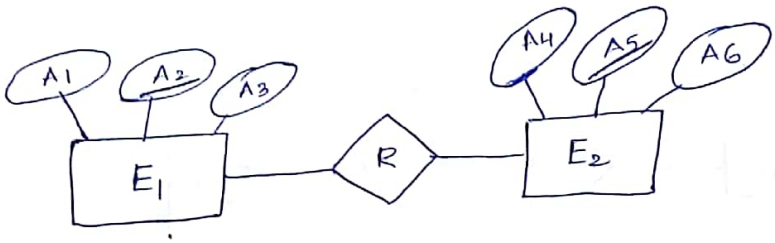
Converting an ER model to Relational Model

① Converting an Entity into relation (Table 1)



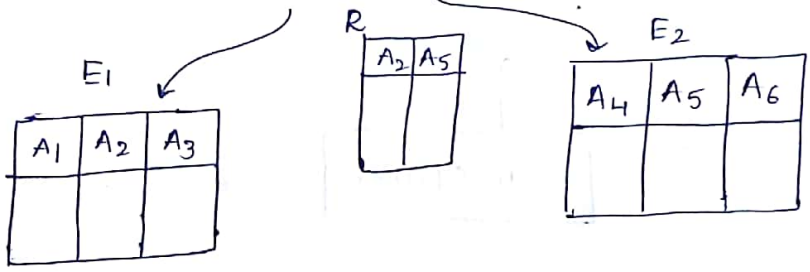
* Relation name is same as Entity name

② converting a Relationship into relation (Table 2)



Att (R) = PK(E1) U PK(E2') U Attr(R)

= A2 U A5 U null = A2 U A5



③ Converting a multivalued attribute to Relation (Table 3)

student

	S NO	SName	phno
data redundancy	101	x	P_1
	101	x	P_2
	101	x	P_3

$Attr(T_3) = mv\,Attr \cup PK(E)$

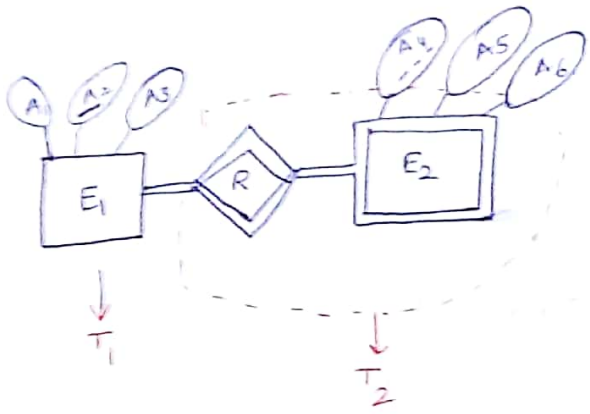
Eg:- $E(A_1, A_2, \underline{A_3}, A_4, \underline{A_5})$

multi valued

$Attr(T_3) = A_2 \cup A_3, A_5$

$= A_2, A_3, A_5$

4) Weak Entityset in to Table

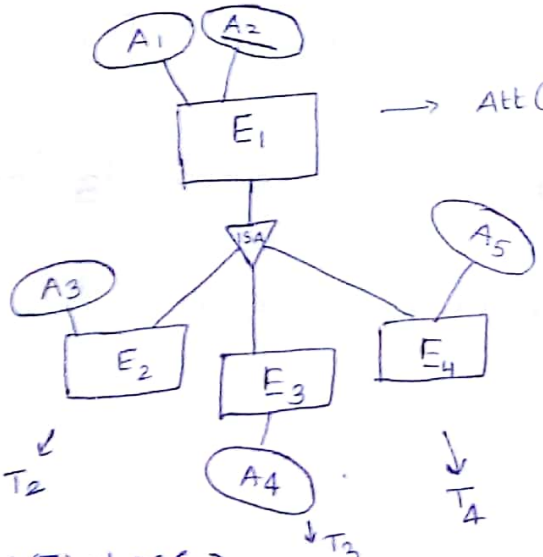


$Attr(T_2) = Att(\text{weak Entityset}) \cup Pk(\text{strong Entity})$

$= (A_4, A_5, A_6) \cup A_2$

$= A_2 A_4 A_5 A_6$

5) Conversion of specialization / Generalization into Relation (Table)



$Att(T_1) = Att(E_1)$

$= A_1, A_2$

$Att(T_2) = Att(E_2) \cup PK(E_1)$

$= A_3, A_2$

Relational Model for student database,

ER model is discussed before (10 pages)

Attr(student) = SNO, Name, DoB, Age, phno

Attr (phno) = phNO U (SNO)

= phNO, SNO.

Attr(Dept) = Did, Dname, loc

Attr(HOD) = Hid, Hname

Attr(faculty) = fid, fname, exp, sal

Attr (subject) = scode, subName, credit

Attr(belongs) = SNO, did.

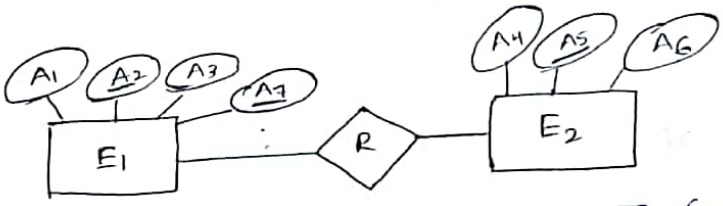
Attr (have) = Did, Hid

Attr (consists of) = Fid, Did

Attr (Registered) = SNO, scode

Attr (teaches) = scode, Fid.

step: 2



$T_1(A_1, A_2, A_3, A_7)$ $T_2(A_2, A_7, A_5)$ $T_3(A_4, A_5, A_6)$

$T_1(A_1, A_2, A_3, A_5, A_7)$, $T_3(A_4, A_5, A_6)$

minimum number of tables = 2

Need For Database design :

Student (SNO, SName, Address, DNO, DName, loc)

1	A	A_1	d_1	x	L_1
2	B	A_2	d_1	x	L_1
3	C	A_1	d_1	y	L_1
4	D	A_2	d_2	y	L_2
5	E	A_3	d_2	y	L_2
6	F	A_4	d_3	z	L_3
7	G_1	A_5	d_3	z	L_3
8	H	A_5	d_4	w	L_4
9	I	A_6	d_1	x	L_4
10	J	A_7	d_5	p	L_5

Problems with above relation

1) Data Redundancy.

2) Insertion anamoly.

- Eg:- if we insert a dept d_6 in which no student registered then studentcount gives 11 but there are only 10 students

3) Deletion problem

Eg:- if last tuple is deleted
then dept count = 4
but physically there are 5 departments

4) Updation anamoly

Eg:- if loc, DName of tuples 1,2 are done
but tuple 3 is not done ⇒ Inconsistent

Database Design

↳ Normalization.

↳ It is a process or theory of Decomposing the larger Relations with anamolies into smaller relations without anamolies based on Functional Dependancy theory.

↳ It is a process to elminate data redundancy

Functional Dependancy :- Dependancies exists b/w the Attribute of the relation

$DB = (R_1, R_2 \ldots R_n)$

$R_1(A_1, A_2 \ldots A_n)$

Let 'R' be a given relation

$X, Y \in R$.. X, Y are the set of Attributes

$X \longrightarrow Y$

X → Determinent, Y → Dependent.

X is functionally determines Y

Y is functionally depending on X

Let $r_1(R)$ is an instance of R (1000 tuples; 100 attributes)

$t_1, t_2 \in r_1$

if $t_1[X] = t_2[X]$ then

$t_1[Y] = t_2[Y]$

Eg:

	S.No	Name
t_1	1	A
	2	B
	3	C
	4	D
t_2	1	A
	2	B
	3	C
	4	D
	5	All
	6	B,

⇒ $F = \{ f_1, f_2, f_3 \}$ → Let R be a given set $\alpha, \beta, \gamma, \delta \in R$

F: set of FD's. f_1, f_2, f_3: FD's. $\alpha, \beta, \gamma, \delta$: set of Attributes

If one FD is known then to know hidden FD's.

Closure properties of FD's

1) Reflexivity: if $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$

2) Transitivity: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

3) Augementation: if $\alpha \rightarrow \beta$ and γ then $\gamma\alpha \rightarrow \gamma\beta$

4) Union: if $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ exists then $\alpha \rightarrow \beta\gamma$

5) Decomposition: if $\alpha \rightarrow \beta\gamma$ exists then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ exists

6) pseudo transitivity: if $\alpha \rightarrow \beta$ and $\gamma\beta \rightarrow \delta$ then $\gamma\alpha \rightarrow \delta$

R (A, B, C, D)

RNo → Name
A → B
RNo → Marks
A → C

} A → BC
RNo → {Name, Marks}

Consider Bank Database

Customer (cid, cname, Address, Age)
Depositor (AccNo, bal, bid, bname, loc)
Borrower (LNo, amt, bid, bname, loc)

AccNO → bal
AccNO → bid
AccNO → bname
AccNO → loc

} AccNO → {bal, bid, bname, loc}

bid → bname
bid → loc

} bid → {bname, loc}

Finding ~~candidate~~ a key

input: R is a given relation, set of F
output: set of candidate key's

Step1: α is a set of attributes $\alpha \in R$

Step 2: if $\alpha^+ \rightarrow R$ and posses the following two properties.

Step 3: Uniquness: Uniquely identifies each tuple in R ($\alpha^+ \rightarrow R$)

Irredusability: β is any subset of α if β ~~→R~~ ~~then~~ is uniquely identifies each tuple in R then α is a super key and β is candidate key.

closer of attributes

i/p : R, F, α

o/p : α^+

step 1 : result = α

step 2 : For each FD $\beta \rightarrow \gamma$ in F

if $\beta \subseteq$ result then

result = result $\cup \gamma$

step 3 : if any changes in result then goto step 2

step 4 : α^+ = result

Eg:① R (A, B, C, D)

F : $A \rightarrow B$
$A \rightarrow C$
$B \rightarrow D$

B^+ : $\alpha = B$ result = B.

$\beta \rightarrow \gamma$		
$A \rightarrow B$	$A \subseteq B$	NO
$A \rightarrow C$	$A \subseteq B$	No
$B \rightarrow D$	$B \subseteq B$	Yes.

result = BUD = BD

change in result, so

$A \rightarrow B$,	$A \subseteq BD$	NO
$A \rightarrow C$,	$A \subseteq BD$	No.
$B \rightarrow D$,	$B \subseteq BD$	Yes.

result = BD(U)B = BD.

no change in result

so, $\alpha^+ = BD$

$B^+ = BD$

Eg ②: R (A B C G H I)

$F = \{ A \to B,\ A \to C,\ CG \to H,\ CG \to I,\ B \to H \}$

find AB^+, AC^+, AG^+, AH^+, AI^+

AB^+ :-

$\alpha = AB$, result = AB

$A \to B$, $A \subseteq AB$ Yes

result = B U AB = AB.

$A \to C$, $A \subseteq AB$, Yes

result = C U AB = ABC

$CG \to H$, $CG \subseteq ABC$, NO

$CG \to I$, $CG \subseteq ABC$, NO.

$B \to H$, $B \subseteq ABC$, Yes.

result = H U ABC = ABCH

NO, change in result, so loop once again

so, $\alpha^+ = ABCH$

$\boxed{AB^+ = ABCH}$

AC^+ :

$\alpha = AC$, result = AC

$A \to B$, $A \subseteq AC$, Yes

result = B U AC = ABC

$A \to C$, $A \subseteq ABC$, Yes

result = C U ABC = ABC

$CG \to H$, $CG \subseteq ABC$, No.

$CG \to I$, $CG \subseteq ABC$, NO.

$B \to H$, $B \subseteq ABC$, yes

result = H U ABC = ABCH.

change in result, so

$A \to B$, $A \subseteq ABCH$, Yes

result = B U ABCH = ABCH

$A \to C$, $A \subseteq ABCH$, yes

result = $C \cup ABCH = ABCH$

$CG \to H$, $CG \subseteq ABCH$, No

$CG \to I$, $CG \subseteq ABCH$, No.

$B \to H$, $B \subseteq ABCH$, Yes

result = $H \cup ABCH = ABCH$.

No change in result

$\therefore \alpha^+ = ABCH$

$\boxed{AC^+ = ABCH}$

— x —

Eg: for finding superkey and candidate key

$R(ABCD)$

$X^+ \to R$

Key, $A \in X$

super key

$(X-A)^+ \to R$

$A_i \in X-A \to$ superkey

$((X-A)-A_i)^+ \to R$.

F: $A \to B$
$A \to C$
$B \to D$
$D \to A$.

{A, B, AB, BC, AC, AD, BD, CD
C, D, ~~ABC~~, ABD, ACD, BCD, ABCD}

~~$\alpha = AB$~~

$(A)^+$: $\alpha = A$ result = A.

$A \to B$

$A \subseteq A$, yes result = AUB = AB

$A \to C$, $A \subseteq AB$, result = ABC

$B \to D$, $B \subseteq ABC$, result = ABCD

$D \to A$, $D \subseteq ABCD$, result = ABCD.

change in result, so repeat

we get $A^+ = ABCD$

A is candidate key,

Similarly check all subsets of R.

$(B)^+$: $\alpha = B$, result = B

$A \to B$

$A \not\subseteq B$, No.

$A \to C$

$A \not\subseteq B$ No

$B \to D$

$B \subseteq B$, Yes, result = BD

$D \to A$,

$D \subseteq BD$, Yes result = ABD

change in result, so repeat

$B^+ = ABCD$

$A \to B$

$A \subseteq ABD$, Yes, result = $B \cup ABD = ABD$

$A \to C$

$A \subseteq ABD$, Yes, result = $C \cup ABD = ABCD$

$B \to D$. " " = ABCD

$D \to A$ " " = ABCD

change in result, repeat

$B^+ \to R$

$(B)^+ = ABCD$ $\therefore$ B is candidate key.

2. R(A B C D E F G H) → 9 ndependent

$F = \{ CH \rightarrow G,\ A \rightarrow BC,$

$B \rightarrow CFH,\ E \rightarrow A,$

$F \rightarrow EG \}$

List all candidate keys

{ A, B, C, E, F, G, H, AB, AC, AE, AF, AG, AH, BC, BE, BF, BG, BH,
CE, CF, CG, CH, EF, EG, EH, FG, FH, GH, ABC, ABE, ABF, ABG,
ABH . . . . . }

Closure of FD's (F^+)

F is a given FD's on R.

F^+ is logically derived from F using closure properties of FD's

Eg 1. R(A B C D)

$A \to B$

$A \to C$

$B \to D$

i/p: R, F

o/p: F^+ $A \to B, A \to C \Rightarrow A \to BC$

$A \to B, B \to D \Rightarrow A \to D$

Step ① for all possible Attribute set X, calculate X^+

Step ②: For each attribute $a \in X^+$ Add $X \to a$ to F^+

Step 3: return F^+

end

Note: $A \to A$

$Y \subseteq X$ then

$X \to Y$ is trivial

For Eg:- 1

$A^+ = ABCD$

$A \in A^+$ $B \in A^+$ $C \in A^+$ $D \in A^+$

$A \to A$, $A \to B$ $A \to C$ $A \to D$

$F_1^+ = \{ A \to A, A \to B, A \to C, A \to D \}$

$B^+ = BD$

$B \in B^+$ $D \in B^+$

$B \to B$, $B \to D$

$F_2^+ = \{ B \to B, B \to D \}$

C^+ is $\alpha = C$, result = C

$A \to B$, $A \notin C$

$A \to C$, $A \notin C$

$B \to D$, $B \notin C$

no. change in result

$C^+ = C$

$F_3^+ = \{C \rightarrow C\}$

$(D)^+$: $\alpha = D$, result = D.

$A \rightarrow B$, $A \subseteq D$, No

$A \rightarrow C$, $A \subseteq D$, No

$B \rightarrow D$, $B \subseteq D$, No

No change in result

$D^+ = D$

$F_4^+ = \{D \rightarrow D\}$

similarly calculate for, $AB^+, AC^+, AD^+, BC^+, BD^+, CD^+, ABC^+$ - - - -

$F^+ = F_1^+ \cup F_2^+ \cup F_3^+ \cup F_4^+ \cup$. - - - - - F_n^+

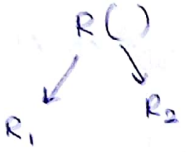
$= \{ A \rightarrow A,\ A \rightarrow B,\ A \rightarrow C,\ A \rightarrow D,$
$B \rightarrow B,\ B \rightarrow D,$
$C \rightarrow C,$
$D \rightarrow D$ - - - - - - $\}$

Decomposition of Relation :-

To overcome, drawbacks of large relation like redundancy, insertion, updation & deletion anamolies we need to Decompose.

$$R(A_1, A_2, \ldots, A_n)$$

Valid decomposition:- if it is possible to retrive data from the decomposed relations.



$R_1 \cup R_2 = R$

$Att(R_1) \cup Att(R_2) = Att(R)$ & $R_1 \cap R_2 \rightarrow R_1 / R_2$

$R_1 \cap R_2 \neq \phi$

lossless join:- if data is not lost after joining the decomposed relations.

Lossy decomposition :- if information is lost during decomposition.

Eg:-

$R(A\ B\ C\ D)$

$A \rightarrow B$
$A \rightarrow C$
$B \rightarrow D$

$A \rightarrow BC$ $R_1(A\ B\ C)$

$R_2(B, D)$

$R_1 \cap R_2 = B$

$R_1 \cap R_2 \rightarrow R_2$

$B \rightarrow R_2$ ✓

∴ It is a lossless decomposition or valid decomposition.

Normalization

It is process of successive decomposition of relation in to number of relations with any anamolies

1 NF
2 NF
3 NF
BC NF
4 NF
5 NF.

1NF :- A relation R is in 1NF, if all values are atomic.

- NO repeated columns & NO set of values

Not in 1NF → convert into 1NF

Eg:

SNO	Name	phno
101	X	{111, 222}
102	Y	{333, 444, 555}

SNO	Name	phno
101	X	111
101	X	222
102	Y	333
102	Y	444
102	Y	555

* But it creates data redundancy.

2NF :- A relation R is in 2NF, iff it is in 1NF and all non key attributes are fully dependent on key

key → candidate key, not superkey

Eg:- Company (CNO, pno, qty, city, status)

101	P1	100	C1	A
101	P2	200	C1	A
102	P1	200	C1	B
102	P2	300	C1	A
103	P1	400	C2	B
103	P2	500	C2	B

FD = { CNO → city
CNO → status
city → status.
{CNO, pno} → qty
}

city is partially dependant on CNO so not

$(CNO)^+$ → city, status.
it is violating the property of 2NF

So, decompose it

Company (cno, pno, qty, city, status)

- $cno \rightarrow city, status$: R_1 (cno, city, status) $= \Pi_{cno, city, status}$
- $\{cno, pno\} \rightarrow qty$: R_2 (cno, pno, qty) $= \Pi_{cno, pno, qty}$

Now, check if the decomposition is valid or not

$R_1 \cap R_2 = cno$

$R_1 \cap R_2 \rightarrow R_1$ [$\because cno \rightarrow \{city, status\}$]

$\therefore$ It is a valid lossless join decomposition

Dependancy Preservation :

Let R be a given relation

F is a set of F.D's

$R \begin{cases} R_1 \\ R_2 \end{cases}$

$(F_{R_1} \cup F_{R_2})^+ = F^+$

$cno \rightarrow city, status = \{ cno \rightarrow city, state$

$\cup$

$\{cno, pno\} \rightarrow qty \quad \{cno, pno\} \rightarrow qty$

$\{ cno, pno \} \rightarrow qty$

$cno \rightarrow city, pno \}$

— × — × —

R_2 → is in 2NF ($\because$ all non key attributes are fully depending on key attribute)

R_1 → is in 2NF

cno	city	status
101	C_1	A
102	C_1	A
103	C_1	A
104	C_2	B
105	C_2	B
106	C_2	B
107	C_3	C
108	C_3	C
-	C_4	D

Insertion anomoly

3NF :- If a ~~relation~~ R is in 3NF iff it is in 2NF and non-key attributes are non-transitively dependent on PK attribute.

Eg :- From the above table.

$cno \rightarrow city$
$city \rightarrow status$

PK = cno.

$(cno)^+ \rightarrow R_1$

non-key = city, status

$cno \rightarrow status$

is depending transitively on

∴ city → status is violating the property of 3NF

So, decompose it.

R_1 —($cno \rightarrow city$)→ R_3 (cno, city)

R_1 —($city \rightarrow status$)→ R_4 (city, status)

$R_3 \cap R_4 \rightarrow R_4$

∴ Its a valid decomposition.

Testing dependancy preservation.

$$(F_{R3} \cup F_{R4})^+ \quad F^+$$

$$\begin{pmatrix} cno \to city \cup \\ city \to status \end{pmatrix} = \begin{pmatrix} cno \to city \\ city \to status \end{pmatrix}^+$$

Check if individual relations are in 3NF or not

$R_3 \to$ in 3NF

$R_4 \to$ is also in 3NF.

city	status
C_1	A
C_2	B
C_3	C

cno	city
101	C1
102	C1
103	C1
104	C2
105	C2
106	C2
107	C3
108	C3

} this much percentage of redundacy is acceptible.

Question ① : $R(A, B, C, D)$ Test if it is in 3NF

$A \to B$
$A \to C$
$B \to D$

$R(ABCDE)$ ⇒ keep the relation in it highest NF
$A \to B$
$B \to C$
$D \to E$

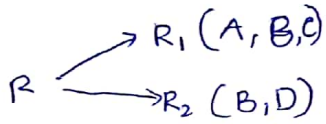
$(A)^+ =$ $A \to B$ res = A
$A \subseteq A$ res = AB
$A \to C$
$A \subseteq AB$ res = ABC
$B \to D$
$B \subseteq ABC$ res = ABCD

$(B)^+ =$ $A \subseteq B$ no
$A \subseteq B$ no
$B \subseteq B$ yes, res = BD

$(B)^+ = BD$

$(C)^+ = C$

$(D^+) = D$



$R_1 \cap R_2 = B$

$B \rightarrow R_2$ ✓

∴ It is a valid decomposition

$(F_{R1} \cup F_{R2})^+ = (F)^+$ dependancy preservation

R_1 is in 2NF

R_2 is in 2NF

R_1 (A, B, C)

$A \rightarrow B$

$A \rightarrow C$

non key attributes B, C are depending non transitively on 'A'

∴ R_1 is in 3NF

R_2 (B, D)

$B \rightarrow D$

R_2 is also in 3NF.

Question 2: R(A, B, C, D, E)

$A \rightarrow B$

$B \rightarrow C$

$D \rightarrow E$

keep the relation in its highest normal form.

7/9/18

1) R = (A B C D E F G H) keep the relation in highest Normal form.

$$F = \{ CH \to G,\ A \to BC,\ B \to CFH,\ E \to A,\ F \to EG \}$$

Sol :- Assume that R is 1 NF

'D' is an idependent key

candidate keys:- AD, ED, FD, BD [∵ A → R, E → A, F → E, B → F]

Non key Attributes:- G, C, H, A, B, D, E, F

∵ non key attributes are not fully depending on key attributes

∴ R is not in 2NF then decompose it

R(ABCDEFGH)

- $A \to BC$ → $R_1(ABC)$
- $B \to CFH$ → $R_2(BCFH)$
- $CH \to G$ → $R_3(CHG)$
- R_4 (~~BCADH~~)
- $E \to A$ → $R_5(EA)$
- $F \to EG$ → $R_6(EGF)$
- $R_7(AD)$

BCNF :-

⇒ If a relation is in BCNF.

⇒ All the determinants are candidate key's

determinant
X → Y
↓ ↳ dependant
C.K.

⇒ Eg :-

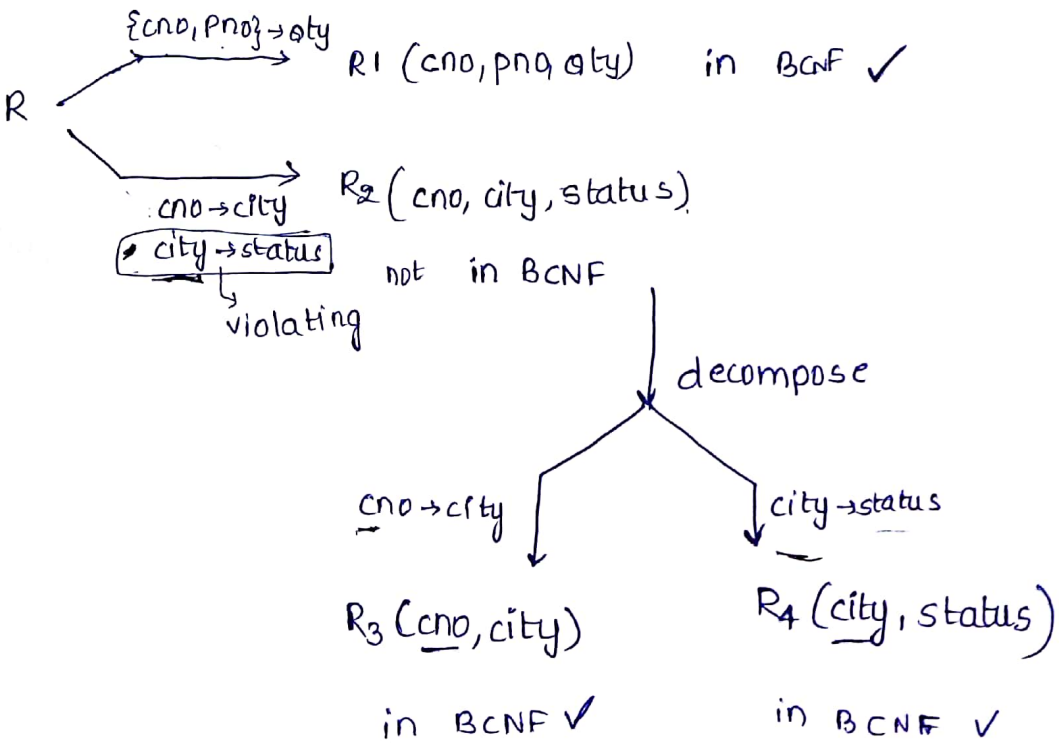
Company (cno, pno, qty, city, status)

FD's :- → cno → city
city → status
{cno, pno} → qty.

CK {CNO, PNO}

∵ cno → city, city → status } is violating BCNF rule.

so, decompose



Find R(A, B C D) in which NF ?

FDs :- AB → D
BC → D
A → B
B → A

Sol :-

8/2/18

In Which NF ?

S.NO	subject	Language
1	S1	L1
1	S1	L2
1	S2	L1
1	S2	L2
2	S1	L1
2	S1	L2
2	S2	L1
2	S2	L2
3	S1	L3
3	S1	L2
3	S1	L1

→ data redundancy

Here, there is no Functional dependency.

It is 2NF ∵ no partial dependancies

It is 3NF ∵ no transitive dependancies

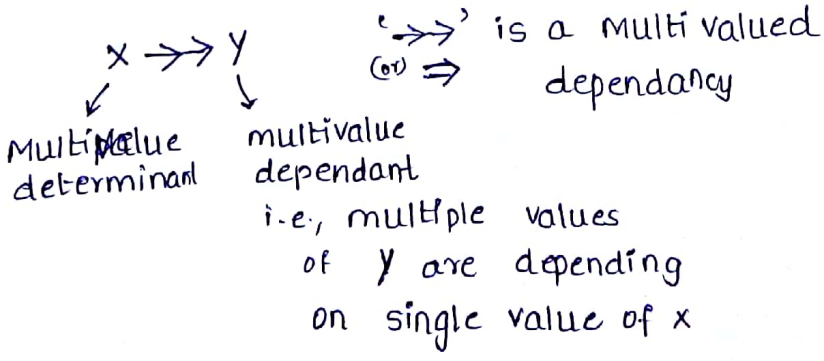
It is in BCNF also.

But still there is Data Redundancy.

4NF:-

multi valued Dependancy (MVD):-

Let R be a given Relation x, Y ∈ R are set of Attributes



$t_1, t_2, t_3, t_4 \in R$

$t_1[X] = t_2[X] = t_3[X] = t_4[X]$

$t_1[Y] = t_3[Y]$

$t_2[Y] = t_4[Y]$

$t_1[R-Y] = t_4[R-Y]$

$t_2[R-Y] = t_3[R-Y]$

so, for the previous example

mvd's are SNO $\twoheadrightarrow$ Language.

4NF

⇒A Relation R is in 4NF, if it is in BCNF and there are no multivalued independent parts in the primary key

The previous example is not in 4NF, so, decompose it

mvd SNO $\twoheadrightarrow$ subject R_1 (SNO, subject)

SNO $\twoheadrightarrow$ Language R_2 (SNO, Language)

R_1 (SNO | subject)

SNO	subject
1	S1
1	S2
2	S1
2	S2
3	S1

R_2 (SNO | Language)

SNO	Language
1	L1
1	L2
2	L1
2	L2
3	L1
3	L2
3	L3

DBMS

- Transaction Management
- Query processing.

$\sigma_\theta(R)$
- random order $O(N)$
- sorted order (i.e., sorting of attribute participating attribute) $O(\log N)$

Indexing :- Indexing is used for improving system performance i.e., speed up access to desired data.

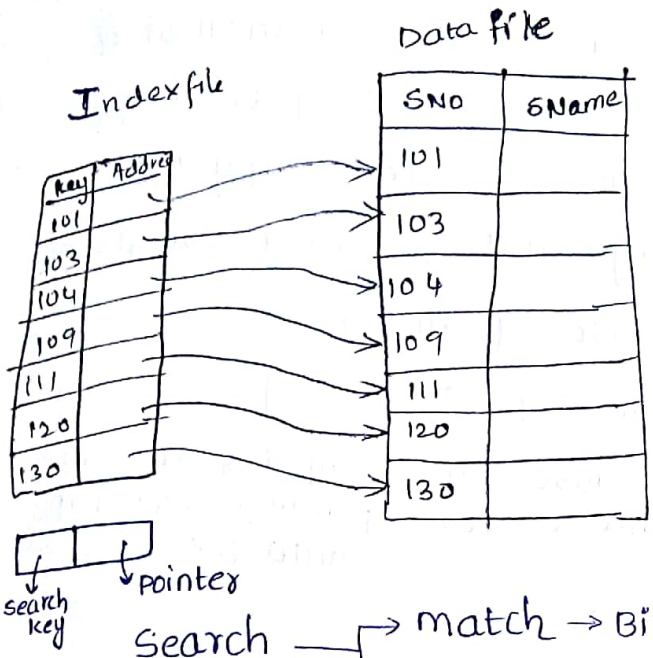
Indexing

1. Ordered Indexing
   - ① Primary Indexing
     - dense
     - sparse
   - ② Secondary Indexing
   - ③ Multilevel Indexing
2. Tree-based Indexing
   - ISAM
   - B-Tree
   - B^+-Tree
3. Hash-based Indexing
   - chaining
   - Probing
   - Re hashing
   - extendable hashing

Ordered Indexing

1) Primary Indexing :- Assumes that data file is in sorted order.

Dense :- For each search key value we find an entry in Index File. corresponding to data file.



Search → match → Binary search
→ Range

Eg:- search for search key value = 115
No data found.

LB → UB
Scan records one by one from LB to UB

if a record is found it will point to that and does sequential scan upto upper boundary.

Insert → Eg: insert a new record with search key value = 105.

First it searches for the record with search key value just less than 105 i.e., 104

After inserting, update the Index file, with new record 105

Deletion → search the data record with the given search key value.

Update Index file because [no. of records in Index file = Data file] in "dense" Indexing

Advantages of Primary dense Indexing

1) When size of D.B is greater than that of main memory, we need not partition the data file, just we can copy index file to main memory and perform operations on it. [Because size of index file is much less than size of data file]

2) Searching time is less, because for every key value in index file we will find a corresponding record in data file

Disadvantage:-

1) search → match → $t_1 + C$
search → range → $t_1 + t_2$
↓
i.e., we may find only two matching records in 100 records.

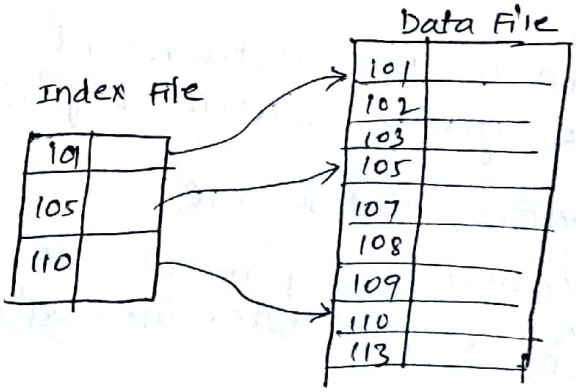
Insert → $T = t_1 + t_2 + C$ (updating, inserting)

Delete → $T = t_1 + t_2 + C$

2) When no. of data file records are increasing, no. of index file records increases and it may lead to partitioning of Index File.

Sparse primary Indexing:-

Here only Few records of data file are present in Index File



search → match → search for the search key value less than the required search key value.

If found, from there do sequential search till we find the required record.

$T = t_1 + c$

→ Range :- similar to match query but start searching from search key value less than Lower boundry

$T = t_1 + t_2$

Insert → $T = t_1 + t_2 + c$ → inserting

- t_1: search for search key value less than or equal to inserting record key value
- t_2: linear search to find exact position

** no need to update.

Delete :- $T = t_1 + t_2 + c$

- t_1: search for search key value nearest to the required key value
- t_2: linear search till we ~~get~~ find exact record
- c: delete.

** no need to update.

Note: Don't decide by scanning Index file that a record is found.

Advantages of sparse Indexing

⇒ It reduces the size of the index file saving space

⇒ We need not update the index file all the time

Disadvantages of sparse Indexing :

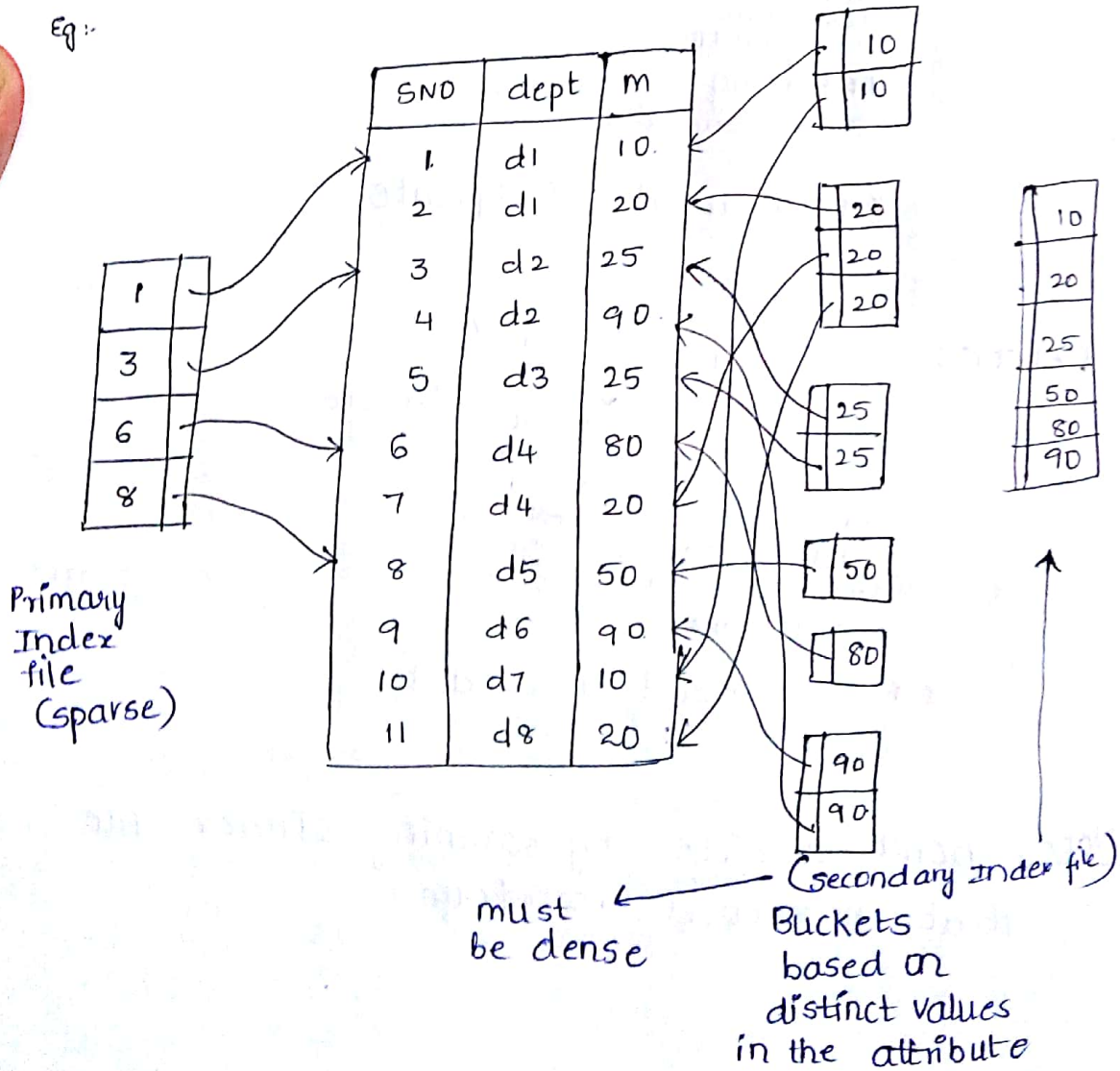
⇒ Generally slower than dense indexing

Secondary Indexing :-

⇒ Index File is in sorted order.

⇒ If queries are based on other attributes other than primary key, create secondary index based on other attributes.

Eg :-



size of bucket = no. of records with that value.

search
- M → search in secondary index file using Binary search.
- R → start from L-B

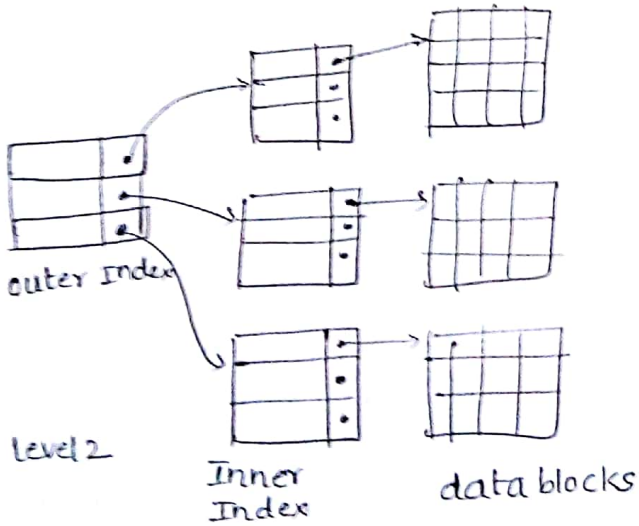
Insert :- start From primary index file

delete :- start from secondary index file.
if after deletion if bucket becomes empty delete record from secondary index file.

MultiLevel Indexing :-

⇒ As the size of the database grows, size of index file also grows.

⇒ Multi-level indexing helps in breaking down the index into several smaller index files in order to make the outermost level so small that it can be saved on prim main memory



search
- M
- R

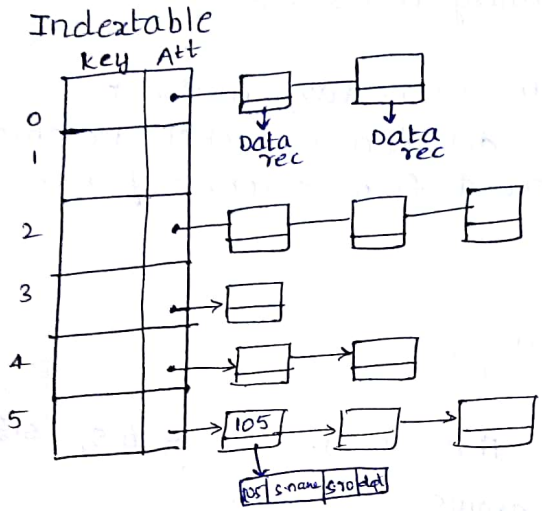
$T = t_1 + t_2 + t_3$
- t_1 → in level 1
- t_2 → in level 2
- t_3 → in data file

start from LB

Hash based Indexing

- Separate chaining.
- Extendable Hashing

Separate chaining



hash Function = $h(x)$

hash key $h(x) = x \bmod \text{Tablesize}$

Insertion → Exact (Match query)

Eg:- $\sigma_{RNO=105}$

$hk(105) = 5$

Find the hash key and insert a node in the corresponding key in the Index table.

Delete → search using hash key.

If the node is not found in the corresponding list then decide it is not present.

Else delete the record.

Update:- Go and search in the hash key value list of Index table.

If found update it
else decide not exist.

Insert = T = C + T_1 + C (C → hashkey generation, T_1 → linear search)

Deletion = T = C + T_1 + C

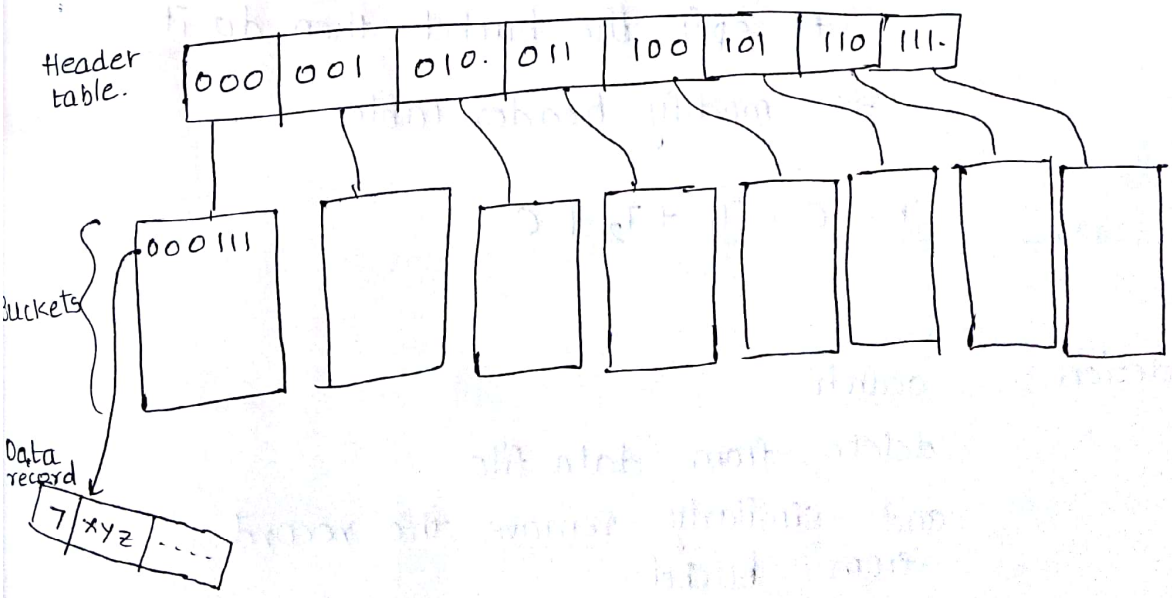
Update = T = C + T_1 + C

Range Query

Eg:- R.NO 110 - 120.

⇒ It has to scan Index table almost totally so, hash based indices are not good for range queries.

Extendable Hashing



Search

Exact

Eg: $\sigma_{RNO=7}(S)$ → 0000 0111

↓

Search in header file

Then search in corresponding bucket

$T = C + T_1 + C$

- C → Conversion to binary
- T_1 → depends on bucket size & search key values in bucket

Insertion :-

Eg: 

insert this record in datafile.

and based on first three bits search in header file.

if bucket is full and is possible to split the bucket then do it.

else modify header table.

$T = C + T_1 + T_2 + C$

Deletion :- search

delete from data file

and similarly remove the record from bucket.

If, after removal if bucket is empty then merge the adjacent buckets

↳ merging time

updation: similar to search.

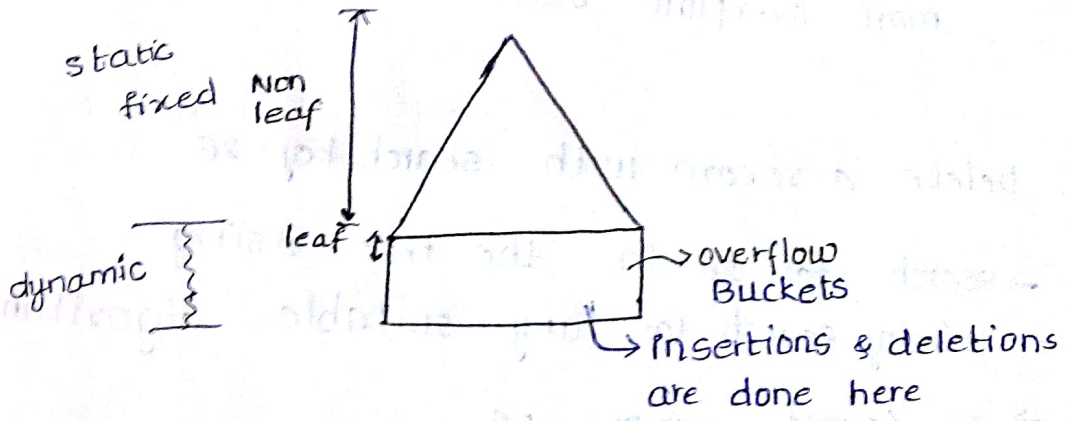
range search:-

It is not necessary that all the search key values are found in the same bucket. we need to scan all buckets almost. ∴ It is not a good practice.

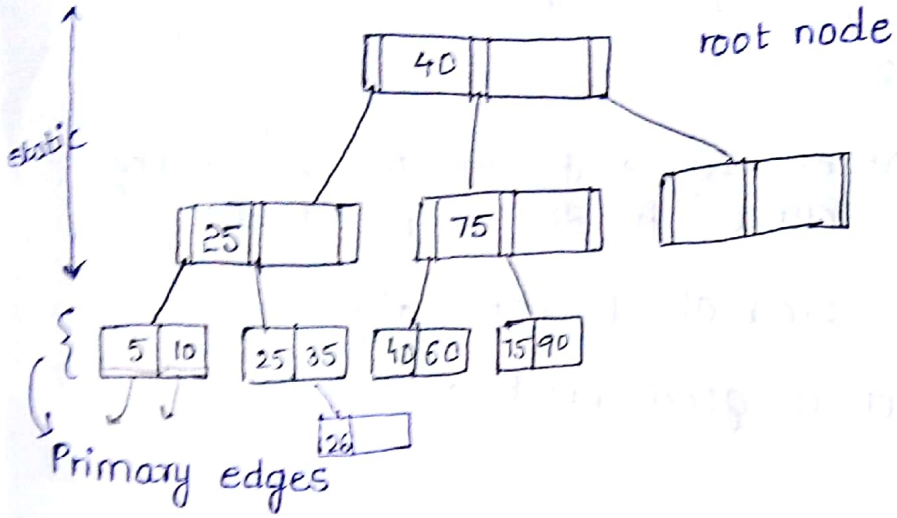
TREE BASED INDEXING TECHNIQUES

- ISAM
- B-Tree
- B^+-Tree

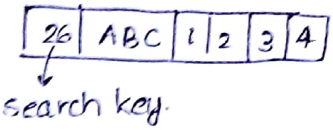
ISAM: node structure is same as B^+ Tree



Indexed Sequential Access Method (ISAM)



Eg:-1 Insert



→ perform Binary search on root node.

→ If space is not found at leaf node.

so, creat overflow bucket.

→ Even, if space is not found in overflow bucket, then create one more overflow bucket

2. Delete a record with search key = 26.

→ search for 26 in the tree using binary search (or) any suitable algorithm

→ If found remove 26

and don't update (or) merge nodes

→ Nodes are merged only when node is empty.

Errors

if it doesn't exist in leaf node then decide data rec doesn't exist

TRANSACTION MANAGEMENT

Transaction

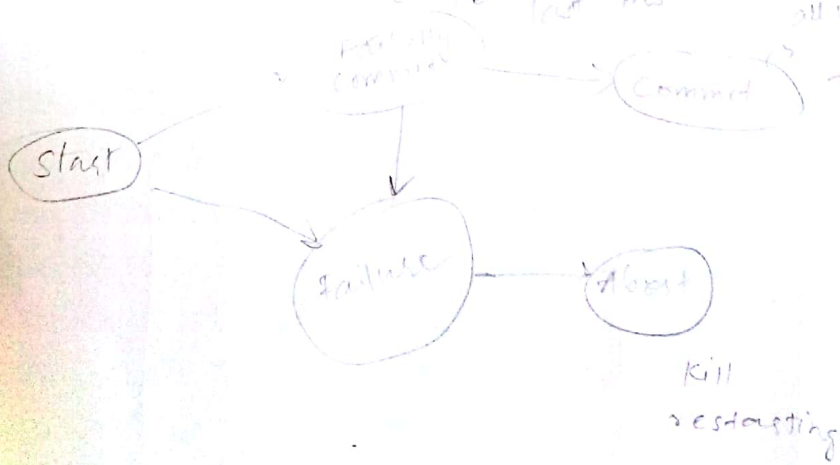
→ set of operations

Eg: Bank database

T_1 : A → 500 → B

T_1 :
Read A → selection search
A.bal = A.bal - 500 → update
write A
Read B → search
B.bal = B.bal + 500 → Deposit update
write B.
end;

Each transaction is uniquely identified by unique TID issued by transaction manager



Transaction Properties (ACID)

➤ **Atomicity :** All the updations are done

or

None of the updations are done

It avoids inconsistency of Database.

If transaction is

Successful then discards old one and points to new copy

else points to old copy only.

➤ **Consistency :** Before and after transaction DB must be consistent.

Eg:- A (1000) $\xrightarrow{100}$ B (500) A+B = 1500.

A (900) → B (510) A+B = 1500.

➤ **Isolation :** Let t_1, t_2 be two concurrent executing transaction

Isolation property states that:

at a time only one transaction takes place in serial order

$t_1 \rightarrow t_2$

(or) $t_2 \rightarrow t_1$

➤ Durability: Once the transaction executes 'COMMIT' all the updations are permanent.

Serial scheduling:-

S_1

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
commit	
	R(A)
	W(A)
	R(B)
	W(B)
	commit

↓

waiting time(T_2) = Execution time(T_1)

S_2

T_1	T_2
	R(A)
	W(A)
	R(B)
	W(B)
	commit
R(A)	
W(A)	
R(B)	
W(B)	
commit	

↓

$WT(T_1) = ET(T_2)$

S_3

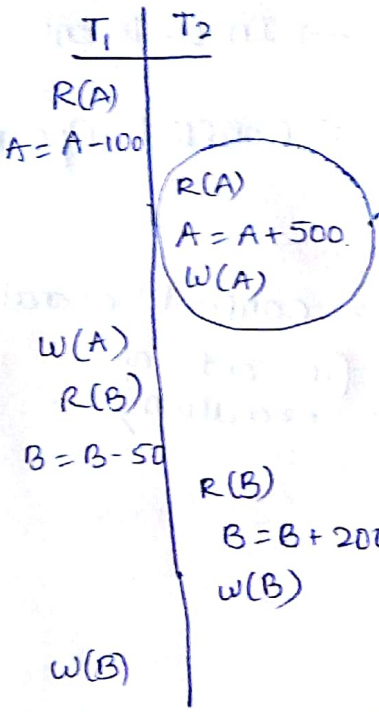
T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
commit	
	R(B)
	W(B)
	commit

waiting time decreased.

S_4

T_1	T_2
	R(A)
	W(A)
R(A)	
W(A)	
	R(B)
	W(B)
	commit
R(B)	
W(B)	
commit	

Example for S_3



→ not effecting the value of A Leads to Inconsistency.

∴ We need a control at Database level for concurrent executions.

S → is a serializable.

↳ Allowed to execute.

↳ guarantees for Consistency.

Eg:- S1, S2

But waiting time is more.

* S is called a serializable schedule if it is equivalent to serial schedule.

→ Conflict Serializable schedule

→ View Serializable schedule.

Conflict Serializable schedule

↳ conflict operation.

Eg:- T_1 : Read(D) T_2 : Read(D)

(oR)

→ consistent

∴ Non conflict

T_1 followed by T_2

T_2 " " T_1

② T_1 : Read (A) T_2 : Read (B) → Non conflict

③ T_1 : Read (D) T_2 : write (D) → Inconsistent

if it happens before T_2 ↓ if it ↓ ∴ Conflict Operations

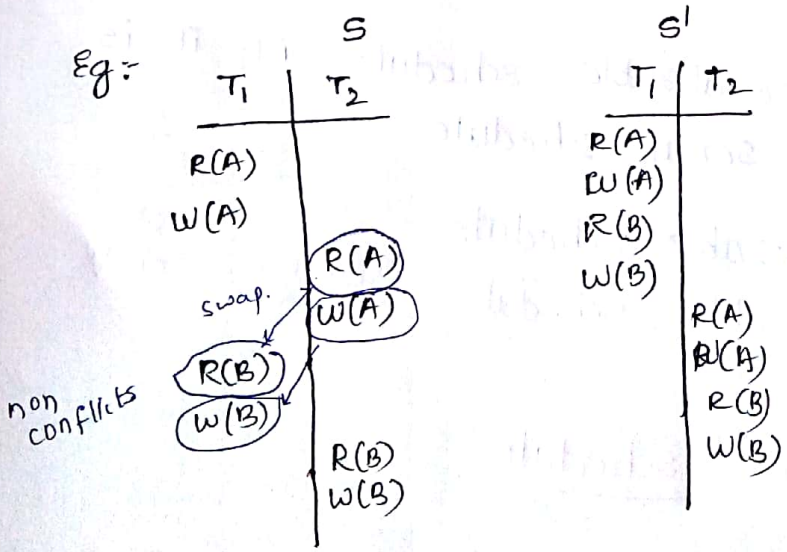
④ T_1 : write (D) T_2 : Read (D) → conflict operations (in order of execution)

⑤ T_1 : write (D) T_2 : write (D)

conflict in order of executions

* S, S' are conflict equivalent schedules if one schedule is obtained by swapping some non conflict operations of other schedule

Eg:-



* if S' is serial schedule then S is conflict serializable schedule

Let T_i and T_j be two transactions

$G(V, E)$ — G is directed graph
V is vertex set
E is Edge set

$E = \{ e_1, e_2, \ldots e_n \}$

$V \rightarrow$ Transactions in S

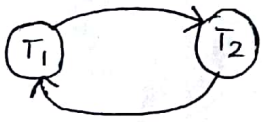
$e = T_i \rightarrow T_j$, Data 'D'

1. If T_i executes read(D) before the T_j executes write(D)
2. If T_i executes write(D) before the T_j executes Read(D)
3. If T_i executes write(D) before the T_j executes write(D)

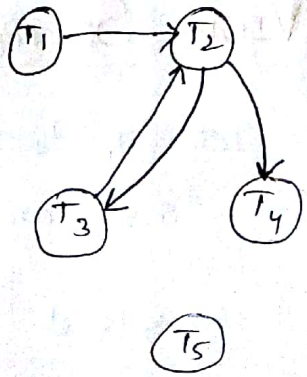
Eg :

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
R(B)	
W(B)	

$V = \{ T_1, T_2 \}$



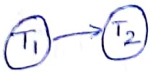
T_1	T_2	T_3	T_4	T_5
R(A)				
W(A)				
	R(A)			
	W(A)			
		R(B)		
		W(B)		
	R(B)			
	W(B)			
	R(C)			
	W(C)			
		R(C)		
		W(C)		
	R(D)			
	W(D)			
R(E)			R(D)	
W(E)			W(D)	



* A schedule is said to be conflict serializable if their graphs are equivalent.

Eg:

T_1	T_2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)



T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)

*

VIEW SERIALIZATION

* Let T_i, T_j, D dataitem and S, S' are two schedules.

S & S' are view equivalent if it satisfies the following conditions

1) If T_i reading the initial value of D in S, the T_i also read the initial value of D in S'

2) If T_i re... other T_j T_i read(Q)

3) If T_i per... in S' also

15/3/18

S →

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	comm
R(B)	
W(B)	
R(C)	
W(C)	
Commit	

Rollback

Fail,

* All non-re... allowed t... only recove...

* In a shed... roll-backs ... are called cascadless

* ~~Non-cascadi...~~ cascading

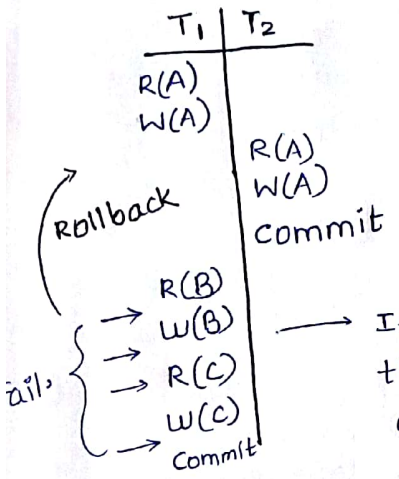
T_1
R(A)
W(A)

2) If T_i reads (D) when written by the other T_j in S, then there exists T_i read(D) where T_j in S'

3) If T_i performs final write on (D) in S, in S' also T_i only performs the final write (D)

13/8

S ⟶ Non recoverable schedule.



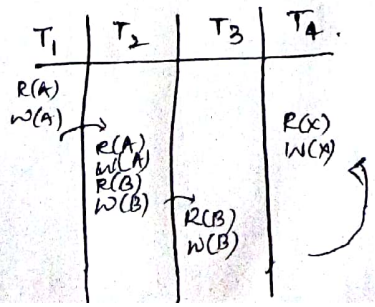
⟶ If a failure occurs here all the updations done previously cannot be removed because of 'commit' after W(A).

* All non-recoverable schedules are not allowed to execute.

only recoverable schedules are allowed.

* In a shedule when there are series of roll-backs are there, then those schedules are called cascading schedules.

* ~~Non-cascading~~ cascadless schedules are allowed but not cascading schedules.



Concurrency Control Management

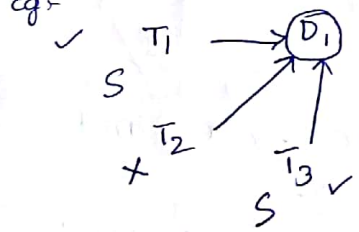
Lock :

- Shared lock — S-lock(D) → Read(D)
- Exclusive lock — X-lock(D) → Read and Write(D)

Lock Compatable Table

	S	X
S	T	F
X	F	F

Eg :-



T_2 waits in waiting queue
∵ it is not compatible

T_1	Concurrency Control Manager (CCM)
Begin	
lock_X(A)	T_1 requesting lock_x(A) grant lock_x(A)
	T_1 permit read on A
R(A)	
W(A)	
unlock(A)	
lock_X(B)	
R(B)	
W(B)	
unlock(B)	
commit	

T_1	T_2
lock_X(A)	
R(A)	
W(A)	
unlock(A)	
lock_X(B)	lock_X(A)
	R(A)
	W(A)
R(B)	
W(B)	
unlock(B)	
	lock_X(B)
	R(B)
	W(B)
	unlock(B)

Eg:- Bank

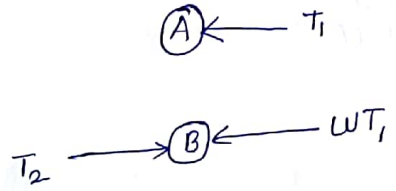
waiting queue (w.q): T_2 withdraw } → This has to wait till the completion of all other transaction.

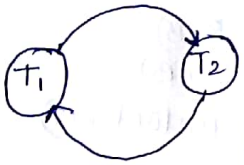
This is known as Starvation.

* But our schedule must be of starvation free.

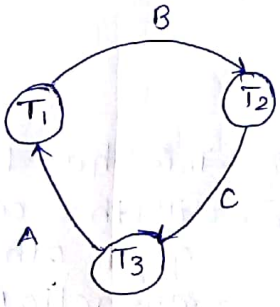
T_1	T_2
lock_X(A)	
R(A)	lock_X(B)
W(A)	R(B)
	W(B)
lock_X(B)	lock-X(A)
R(B)	
W(B)	
unlock(A)	R(A)
unlock(B)	W(A)
	unlock(A)
	unlock(B)

granted x_lock on A to T_1
granted x_lock on B to T_2





Dead lock

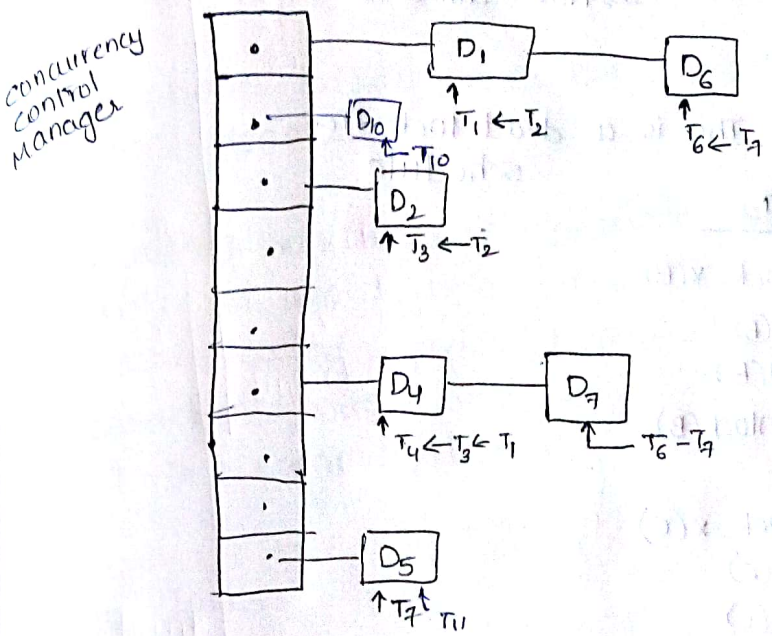


T_1	T_2	T_3
lock_X(A)		
R(A)	lock_X(B)	
W(A)	R(B)	lock_X(C)
	W(B)	R(C)
		W(C)
lock(X(B)		
R(B)	lock_X(C)	Lock_X(A)
W(B)	R(C)	R(A)
	W(C)	W(A)
unlock(A)	~~unlock~~	
unlock(B)	unlock(B)	
	unlock(C)	unlock(C)
		unlock(A)

→ our schedule must be dead lock free

Lock Table

Data structure which maintains waiting queue



Eg :- T_{10} (D_1, D_4, D_{10})

if D_{10} is not there in the Lock Table then D_{10} will be placed according to hashing.

⇒ If T_2 is rolled back then T_2 will be removed from lock Table.

⇒ Implementation :

using hash table + linked list.

How to identify starvation situation occured or not ?

→ count

How to identify DeadLock situation ?

→ Graph, check cycles.

2-PHASE Locking Mechanism

↳ To avoid non-recoverability

① Growing phase → allowed to lock
→ not allowed to unlock

② Shrinking phase → allowed to unlock
→ NOT allowed to lock the data item.

Ex:- This is a dead lock free schedule.

T_1	T_2
Lock_X(A)	Lock_X(B)
R(A)	R(B)
W(A)	W(B)
unlock(A)	unlock(B)
Lock_X(B)	lock_X(C)
R(B)	R(C)
W(B)	W(C)
unlock(B)	unlock(C)
	lock_X(A)
	R(A)
	W(A)
	unlock(A)

⇓

2 phase locking schedule

T_1	T_2
lock_X(A)	lock_X(B)
lock_X(B)	lock_X(C)
R(A)	lock_X(A)
W(A)	R(B)
R(B)	W(B)
W(B)	R(C)
unlock(A)	W(C)
unlock(B)	R(A)
→ commit	W(A)
	unlock(A)
	unlock(C)
	unlock(B)
	commit

There is possibility for cascading.

∴ It is helpful for recovery.

* This mechanism/protocol assures serializability. It can be proved that the transactions can be serialized in the order of their lock points (i.e, the point where a transaction acquired its final lock)

* Two-phase locking does not ensure freedom from deadlocks

* Cascading roll_back is possible under two-phase locking. To avoid this, follow a modified protocol called strict two-phase locking. Here a transaction must hold all its exclusive locks till it commits/aborts.

* Rigorous two phase locking is even stricter: here all locks

Lock_conversions

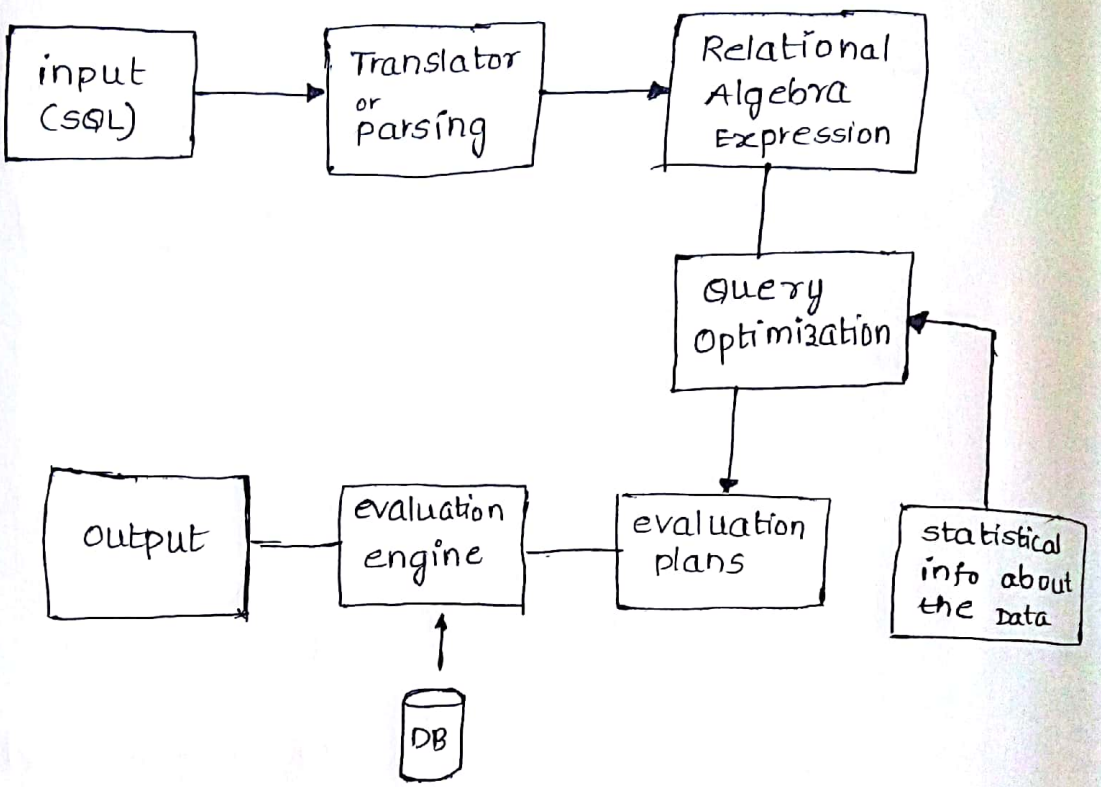
* Two phase locking with lock conversions.

- First phase:
  - can acquire a lock_s on item.
  - can acquire a lock_x on item.
  - can convert a lock_s to a lock_x (upgrade)

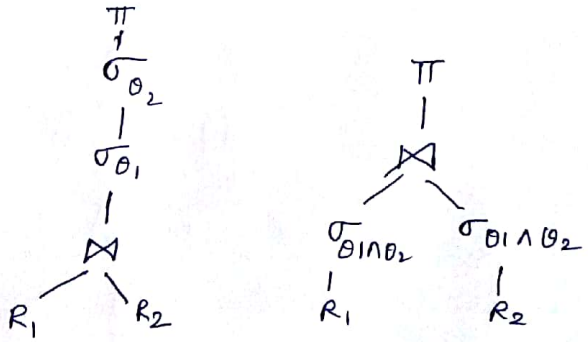
- second phase
  - can release a lock-S
  - can release a lock_X
  - can convert a lock_x to a lock_s (downgrade).

* This protocol assures serializability. But still relies on the programmer to

QUERY PROCESSING

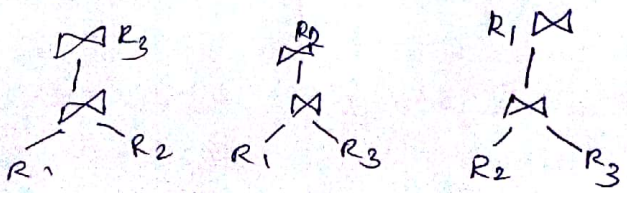


evaluation plans for $\Pi(\sigma_{\theta 1 \wedge \theta 2}(R_1 \bowtie R_2))$



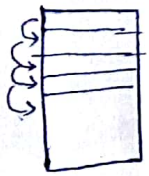
chose a cheaper plan.

$R_1 \bowtie R_2 \bowtie R_3$



Algorithms for selection

Simple selection $\sigma_{\theta_1}(R_1)$



① Linear scan (Linear search)
'n' number of comparisons

② θ_1 based on A_1 attribute

Binary search (if input data is sorted based on A_1)

③ Index based selection. (if index is available use hash function)

④ compound selection.

$\sigma_{\theta_1 \wedge \theta_2}(R)$

If there is no index or it is unsorted then use Linear scan

If there is index (or) if it is sorted then use $\sigma_{\theta_2}(\sigma_{\theta_1}(R))$

For compound selection on $\theta_1, \theta_2, \ldots \theta_n$

$\theta_1 \wedge \theta_2 \wedge \ldots \wedge \theta_n$

$$\sigma_{\theta_n}(\ldots(\sigma_{\theta_2}(\sigma_{\theta_1}(R)))\ldots)$$

↓

here use
Linear scan
(or) Binary search
(or) Index based search

For $\theta_1 \vee \theta_2 \vee \theta_3$ → t_3 tuples

t_1 tuples → t_2 tuples

$t_1 \cup t_2 \cup t_3$

For Range Queries

$x_1 - x_2$

$\theta = A.val \geq x_1 \wedge A.val \leq x_2$

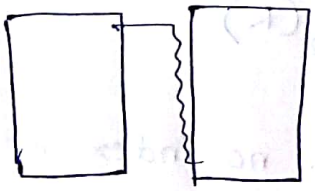
Algorithms for join

① Nested loop join

$\sigma_\theta (R_1 \times R_2)$

$\bowtie$

$2(m * n)$



Algorithm

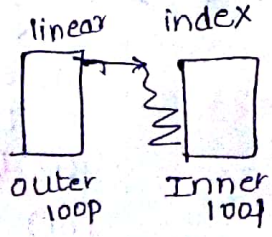
1) For each tuples t_i in R_1

For each tuples t_j in R_2

if (t_i, t_j) satisfies the condition

then result = combine(t_i, t_j)

2) Index based loop join



For each tuple t_i in R_1

$x = \text{Index}(t_i, A)$

For each tuple t_j in R_2

if $(\theta(t_j, x))$

then combine (t_i, t_j)

③ Block Nested loop join.

If the size of Relation is greater than size of memory, then we go for Block nested loop join.

For each block b_L of R_1 do begin
For each block b_K of R_2 do begin
For each tuple t_i in B_L of R_1 do begin

For each tuple t_j in B_K of R_2 do begin

if (t_i, t_j) satisfies the condition

then combine (t_i, t_j)

end

end

end

end.

④ Merge - Join.

1. Sort both the relations on their join attribute.
2. Merge the sorted relations to join them.

→ join step is similar to the merge stage of sort-merge algorithm

→ Main difference is handling of duplicate values in join attribute - every pair with same value on join attribute must be matched

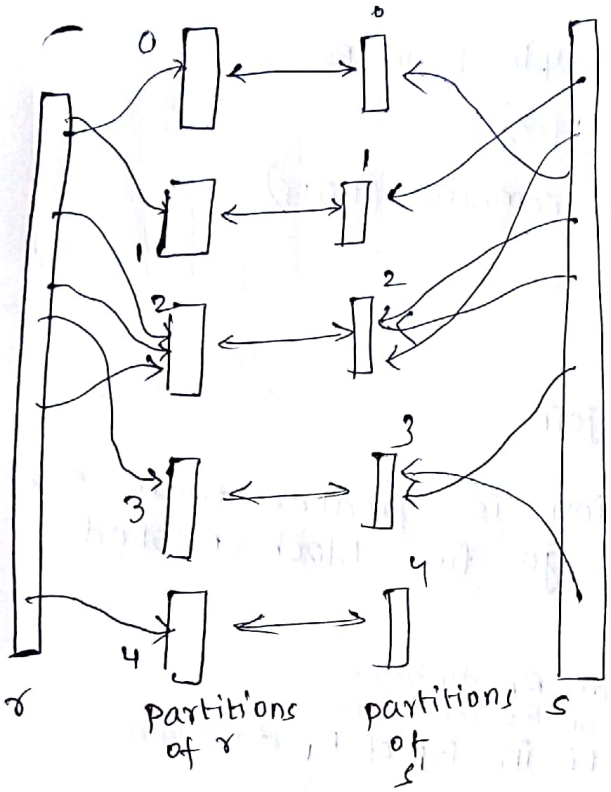
pr →

a_1	a_2
a	3
b	1
c	8
d	13
f	2
m	5

ps →

a_1	a_3
a	A
b	G
c	L
d	N

5) Hash join.



Complex joins

$r \bowtie_{\theta_1 \wedge \theta_2 \cdots \wedge \theta_n} s$

⇒ either use nested loops / block nested loops

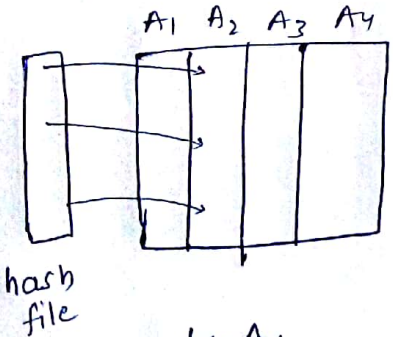
$r \bowtie s$
$\theta_1 \vee \theta_2 \vee \cdots \vee \theta_n$

⇒)

* Duplicate elimination:
  can be implemented via hashing or sorting.
  - on sorting duplicates will come adjacent to each other, and all but one set of duplicates can be deleted.
  - Optimization :- duplicates can be deleted during run generation as well as at intermediate merge steps in external sort merge.
  - Hashing is similar - duplicates will come into the same bucket.

* Projection.
  - perform projection on each tuple
  - followed by duplicate elimination.

Projection



$t_i \cdot A_4$.

$T = t_1 + t_2$

t_1 → linear scanning of all records

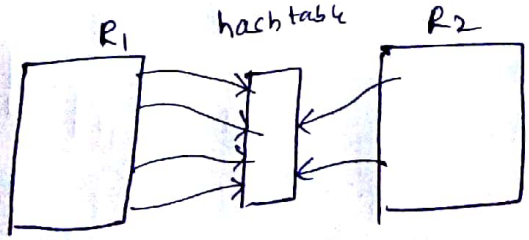
t_2 → to verify if it is a duplicate value or not by hashing.

Set Operation

$\cup$, $\cap$, $-$

Union ($\cup$):

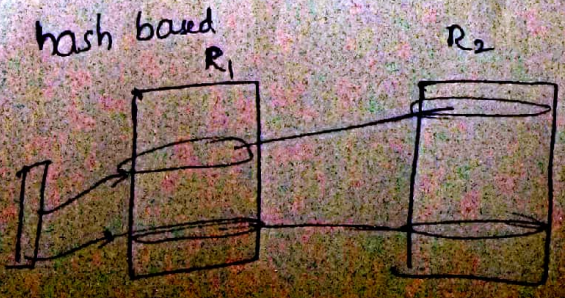
hash based



sort based

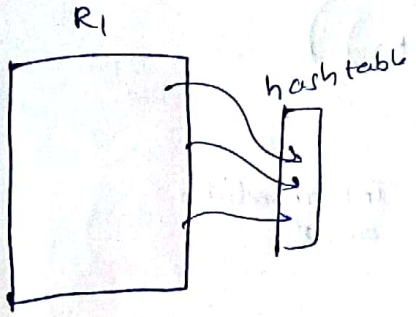
intersection ($\cap$):

hash based



set difference (−):

$R_1 - R_2$



Now scan R_2, for each tuple of R_1 ; if its corresponding hash value is already, delete its hash value.

Result = hash table.

Aggregate Operations

- SUM
- COUNT
- MIN
- MAX

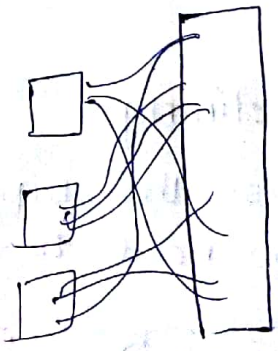
= (group by)

Sort based

→ calculations like min, max, sum are done on the resultant tuples present consecutively

hash based.

→ buckets relation.



perform, calculation like min, max, count on each bucket

Evaluation of Expressions.

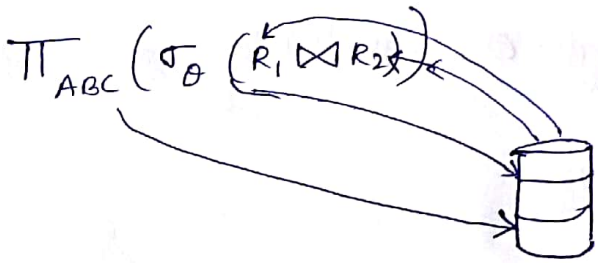
$\Pi_{ABC}(\sigma_\theta(R_1 \bowtie R_2))$

↓

output = intermediate output;

Materialization

Generates results of an expression whose inputs are relations or are already computed, materialize (store) it on disk. Repeat.

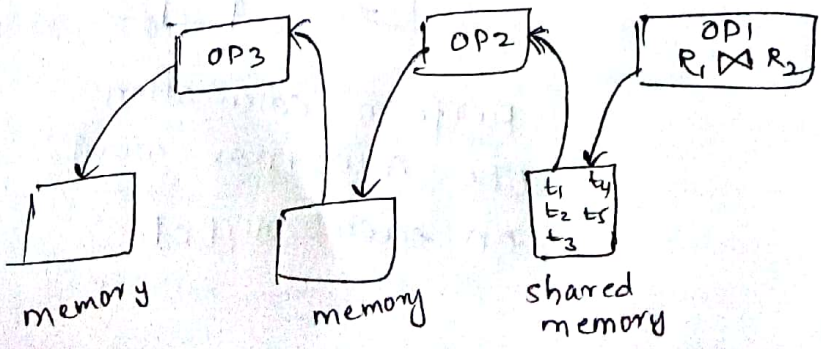


Drawbacks

More number of I/o operations are present which are costlier.

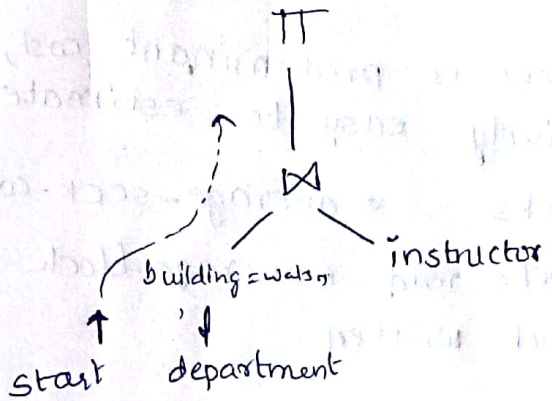
Pipelining:

Pass the tuples to next parent operation. No need to store just pass on.



Eg:- for Materialized

$$\Pi_{name}\left(\sigma_{building="watson"}(department) \bowtie Instructor\right)$$



⇒ pipeline may not always be possible
eg.., sort, hash-join.

⇒ For pipeline to be effective, use evaluation algorithms that generate output tuples even as tuples are received for inputs to the operation.

⇒ pipeline can be executed in two ways:

demand driven

⇒ Producer starts producing only on the demand of the consumer.

Producer driver

⇒ if there is no overflow, just keep on producing (or) writing.

Query cost :- Total elapsed time for answering query.

Factors contributing to time cost.

→ disk accesses, CPU or network communication

Typically disk access is predominant cost, and is also relatively easy to estimate.

- Number of seeks * average-seek-cost
- Number of blocks read (average-block-read-cost)
- Number of blocks written

⇒ cost of write is greater than cost of read

(Data is read back after being written to ensure that the write was successful

t_T → time to transfer one block.

t_S → time for one seek

∴ cost of b block transfers + S seeks

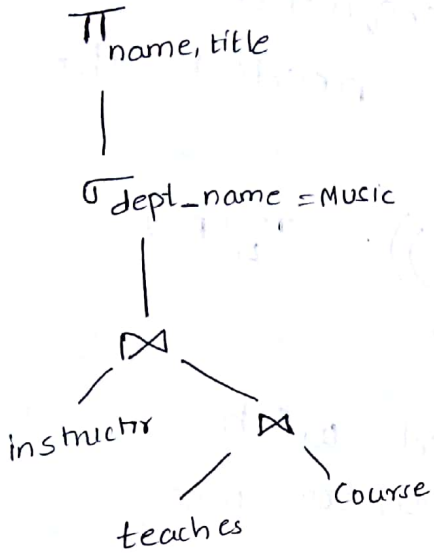
$$\boxed{\text{cost} = b * t_T + S * t_S}$$

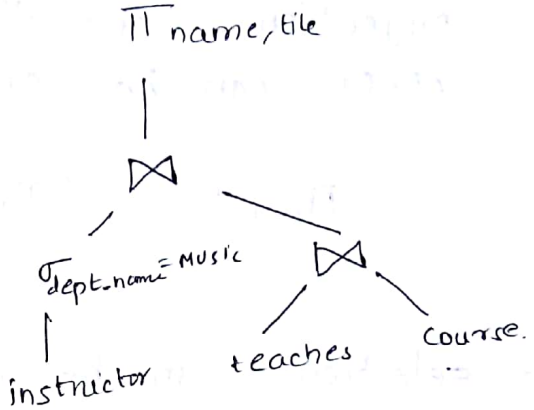
Note:- we ignore CPU costs for simplicity.

(Real systems do take CPU cost into account).

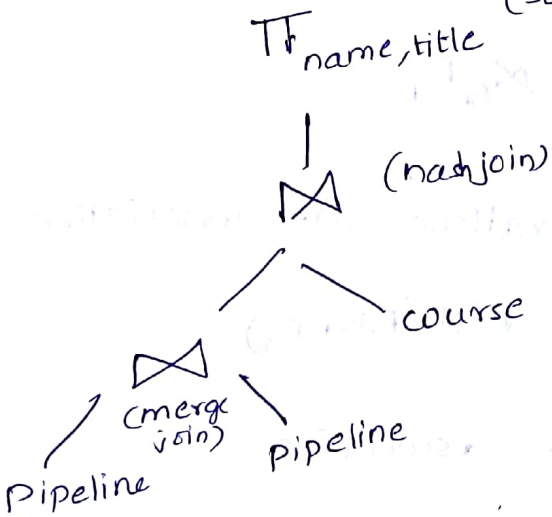
Cost of operation = Access time (cost) + Execution cost + Communication cost.

QUERY OPTIMIZATION





* An evaluation plan defines exactly what algorithm is used for each operation and how the execution of the operation is co-ordinate



If two queries are accepting same input and giving same output then they are Equivalent

* Equivalente Rules

1) conjunctive selection Operations can be deconstructed.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. selection operations are commutative

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Only the last in a sequence of projection operations is needed, the others can be omitted.

$$\Pi_{L_1}(\Pi_{L_2}\cdots(\Pi_{L_n}(E))\cdots) = \Pi_{L_1}(E)$$

4. selections can be combined with cartesian products and theta joins.

a) $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$

** b) $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5. Theta-join operations (and natural joins) are commutative

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

6. (a) Natural join operations are associative.

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(b) Theta joins are associative

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

7. Selection operation distributes over the theta join only when

a) when all the attributes in θ_0 involve only the attributes of one of the expression (E_1) being joined

$$\sigma_{\theta_0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_{\theta} E_2$$

b) when θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_{\theta} (\sigma_{\theta_2}(E_2))$$

8) Projection Operation distributes over the theta join only when

a) If θ involves only attributes from $L_1 \cup L_2$

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = (\Pi_{L_1}(E_1)) \bowtie_{\theta} (\Pi_{L_2}(E_2))$$

b) Consider a join $E_1 \bowtie_{\theta} E_2$

→ Let L_1 and L_2 be sets of attributes from E_1 and E_2 respectively.

→ Let L_3 be attributes of E_1 that are involved in join condition θ, but are not in $L_1 \cup L_2$ and

→ Let L_4 be attributes of E_2 that are involved in join condition θ, but are not in $L_1 \cup L_2$

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_{\theta} E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_4}(E_2)))$$

9) set operations union and intersection are commutative

$$E_1 \cup E_2 = E_2 \cup E_1$$

$$E_1 \cap E_2 = E_2 \cap E_1$$

(set difference is not comutative

10) Associative

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$

$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11) selection Operation distributes over $\cup, \cap$ & -

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta (E_2)$$

and similarly for $\cup$ and $\cap$ in place of -

Also $\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - E_2$

and similarly for $\cap$, in place of '-'

but not for $\cup$

12) Projection Operation distributes over union

$$\Pi_L (E_1 \cup E_2) = \Pi_L (E_1) \cup \Pi_L (E_2)$$

** Practical Query optimizers incorporate the following approaches.

→ search all the plans and choose the best plan in a cost-based fashion.

→ uses heuristics to choose a plan.

Cost-Based Optimization with Equivalence Rules

1) Physical Equivalence rules allow logical query plan to be converted to physical query plan specifying what algorithms are used for each step.

2) Efficient optimizer based on equivalent rules depends on

→ A space efficient representation of expressions which avoids making multiple copies of subexpression.

⇒ Techniques for detecting duplicate derivation of expressions.

→ A form of dynamic programming based on memoization, which stores the best plan for a subexpression the first time it is optimized, and reuses it on repeated optimization calls on same sub-expression.

Heuristic Optimization

→ Cost-based optimization is expensive, even with dynamic programming.

→ Systems may use heuristics to reduce the number of choices that must be made in a cost-based fashion.

** Heuristic Optimization transforms the query-tree by using a set of rules that typically (but not in all cases) improves execution performance.

(i) Perform selection early (reduces the number of tuples)

(ii) Perform Projection early (reduces the number of attributes)

(iii) Perform most restrictive selection and join operations (i.e., with smallest result size) before other similar operations.

(iv) Some systems use only heuristics, others combine heuristics with partial cost-based Optimization

Recovery Systems

- Failure classification
- storage structure
- Log-Based Recovery
- ARIES

Failure classification

1) Transaction Failure.
   → logical errors:

   → system errors:

2) System crash:

3) Disk Failure:

storage structure:

1) Volatile storage:
   does not survive system crashes.
   Ex: main memory, cache memory

2) Non volatile storage:
   survives system crashes.
   Ex: disk, tape, flash memory,
   non-volatile (battery backed up) RAM
   but may still fail losing data.

3) Stable storage

- Maintain multiple copies of each block on separate disks.

- Failure during data transfer can still result in inconsistent copies.

Log-Based Recovery :-

Recovery algorithms.

- During normal transaction processing to ensure enough information exists to recover from failure
- Actions taken after a failure to recover the database contents to a state that ensures atomicity, consistency and durability

Log :

- History of actions executed by the DBMS
- Sequence of log records, and maintains a record of update activities on the database.
- Stored in stable storage.

LSN

- Every log record given a unique ID called the log sequence number.
- LSN are given in increasing order.

PageLSN :-

Every page in DB contains the LSN of the most recent log record that describes a change to this page

① when transaction T_i starts, it registers itself by writing a $\langle T_i \text{ start} \rangle$ log record.

② Before T_i executes write(x), a log record $\langle T_i, X, V_1, V_2 \rangle$ is written,
where V_1 is the value of x before the write (the old values) and V_2 is the value to be written to x (the new value)

③ When transaction T_i finishes it last statement the log record
$\langle T_i \text{ commit} \rangle$ is written

④ When transaction T_i has aborted $\langle T_i \text{ abort} \rangle$

undo & Redo

undo of a log record $\langle T_i, X, V_1, V_2 \rangle$ writes the old value V_1 to X.

Redo of a log record $\langle T_i, X, V_1, V_2 \rangle$ writes the new value V_2 to X.

checkpoint

streamline recovery procedure by periodically performing checkpointing.

→ Output all log records currently residing in main memory onto stable storage.
→ Output all modified buffer blocks to the disk.
→ write a log record $\langle \text{checkpoint } L \rangle$ on to stable storage where L is list of all transactions active at the time of checkpoint.
→ All updates are stopped while doing checkpointing.

SECURITY AND AUTHORIZATION

* Secrecy :- Users should not be able to see things they are not supposed to be

  Eg:- A student can't see other student's grade.

* Integrity :- Users should not be able to modify things they are not supposed to.

  Eg:- only instructors can assign grades.

* Availability :- Users should be able to view and modify data.

** A Security policy specifies who is authorized to do what.

** A security mechanism allows us to enforce a chosen security policy.

* Two main mechanisms at the DBMS level.
  i) Discretionary access control.
  ii) Mandatory access control.

Discretionary access control.

* Based on the concept of access rights or privileges for objects (tables and views) and mechanisms for giving users privileges (and revoking privileges).

* Creator of a table or a view automatically gets all privileges on it.
  - DBMS keeps track of who subsequently gains and looses privileges, and ensures

requests from users who have the necessary privileges (at the time the request is issued) are allowed.

GRANT command

GRANT privileges ON object TO users [WITH GRANT OPTION]

Privileges

1. select : can read all columns
2. INSERT : can insert tuples
3. DELETE : can delete tuples.
4. REFERENCES : can define foreign keys that refer to this column

* If a user has a privilege with the GRANT OPTION can pass privilege on to other users.

* only owner can execute CREATE, ALTER, and DROP

Eg:- 1. GRANT INSERT, SELECT ON Sailors TO Horatio.

2. GRANT DELETE ON SAILORS TO Yuppy WITH GRANT OPTION.

3. GRANT UPDATE(rating) ON sailors TO Dustin,

4. GRANT SELECT ON Activesailors TO Guppy, Yuppy.

5. REVOKE : when a privileges is revoked from x

"THERE IS NO SUBSTITUTE
FOR HARD WORK. NEVER GIVE UP.
NEVER STOP BELIEVING."

- Manam Ramu