[8-10 marks] DBMS

Topics:

1. Integrity Constraints & E.R. Model
2. Normalisation
3. Queries → Relational Algebra
   → SQL
   → Relational Calculus
4. File Organisation and indexing
5. Transactions and Concurrency control.

Textbooks:

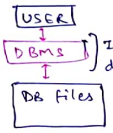
→ DBMS - Raghuramakrishna (Solve exercise problems)
→ DBMS - Navathe

Faculty:

Ravi Kumar Pedappu

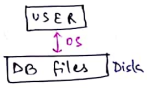
Database:
→ Collection of related data.
Eg. set of students info

DBMS: S/w used to manage and access DB files in an efficient way.



Interface b/w user and database files.

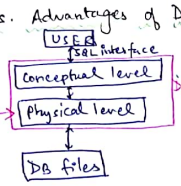
Flat File System [os File System]:
→ DB files stored in disk and managed without DBMS (using O.S)



→ Small DB: if DB size KBs/MBs, can manage using flat files.
→ Huge DB: if DB size GBs/TBs, then, flat filesystem failed to mange DB.

Disadvantages of flat filesystem vs. Advantages of DBMS filesystem:

① Too complex to manage storage details of DB files by user &
Too difficult to develop application programs & manage application programs.



• Data Independency: Users can access DB files w/o Storage details.
• easy to access data
~~easy to manage~~.

② Flat Filesystem: More access cost to access data from DB file.
DBMS filesystem: DBMS maintains indexing to reduce access cost.

③ Degree of concurrency very less. → More degree of concurrency.

④ Too complex to implement different levels of access control. → Easy to implement different levels of access control because of virtual tables [VIEWS].

Integrity Constraints of RDBMS:
→ Relational DBMS: Widely used data models.
• R.J. Codd
* also called Codd data model.
• Codd proposed 12 rules to design RDMS software.

Rule 1: • Data in DB file must be in tabular format [set of rows and column].
• No two records of DB file is same.

attribute/field

DBMS file

Student

SId	Sname	Age
S1	A	20
S2	B	20
S3	B	21
S4	A	22

record/tuple

→ Arity = 3
Cardinality = 4

Flat file

S1, A, 20#, S2, A, 20#,
S3, B, 21#, S4, A, 22#,
- - - - - - - - - -

[OS file]

* Arity: No. of fields of the DB file (table)
* Cardinality: No. of records of DB file.
* Relational Schema: Definition or structure of DB table.
Student (SId, Sname, Age)
↑ Table name ↑ Field names
* Relational Instance: record set of the DB file.
• also called snapshot.
* Candidate Key: Minimal set of attributes used to differentiate records of the Table uniquely. (Denoted by underlining → SId)

Eg: ① Student (Sid, Sname, age) {Sid ⇒ candidate key}

{Sid Sname ⇒ NOT candidate key}

② Enroll (Sid Cid fee) "Student can enroll in many courses"

S1	C1	5000
S1	C2	6000
S2	C2	6000
S3	C2	6000

a course can be enrolled by many students.

{Sid Cid : Candidate Key}

③ R

A	B	C
4	6	8
4	5	8
4	6	4
4	5	4
5	6	8

A B C : Candidate Key.

* X is some set of attribute of R.
X is candidate key of Rel R.

Iff: ① No two records with same field values over over "X" attribute. //Unique

② No proper subset of "X" can differentiate records uniquely. //Minimal

R (X (cand.key) — —)

④ Emp

Eid	Ename	DOB	panid	ppno	Bank	Acc.No.
E1	A		P3	X5	SBI	101
E2	NULL		P5	NULL	SBI	102
E3	B		P4	X2	ICICI	102
E4	C		P1	NULL	ICICI	101

Assume No two emp's with same bank & acc.no.

Candidate key {(Eid), (PanId, Ppno, BankAccNo)}

PRIMARY KEY — ALTERNATIVE KEYS

NULL ⇒ Unknown value / unexisting value.

⇒ Primary Key: Any one candidate key of relation whose field values must be NOT NULL. [UNIQUE & NOT NULL]

* If 2 attributes are combined, then both should be NOT NULL.

→ Every field of primary key must be NOT NULL. [NULL values not allowed for any primary key field].

→ for any DB relation, at most one primary key is allowed.

⇒ Alternative Keys: All candidate keys of relation except primary key.

* Alternative key fields allowed NULL values.
* More than one alternative key allowed to define for relation.

* Create Table Emp
(Eid varchar(10) Primary Key,
Ename varchar(30),
DOB date,
PanID varchar(10) UNIQUE NOT NULL,
Ppno varchar(15) UNIQUE,
bank varchar(20),
AccNo integer(15),
UNIQUE (bank, AccNo)
);

Integrity Constraints: Conditions to maintain correctness.

Prime Attribute: Attribute which belongs to any candidate key of the relation.

Prime Attribute Set: Set of all candidate key attributes of relation (DB Table).

E.g: ⇒ Prime Attr set {Eid, panID, ppno, Bank, AccNo} of Emp relation.

Non prime / Nonkey attribute: attribute which doesnot belong to any candidate key of the relation.

Non prime attribute set: set of all non-prime attributes of relation "R".

E.g: {Ename, DOB}

⇒ For any relational database table [RDBMS Table], there must be atleast one candidate key and atleast one candidate key whose field values must be NOTNULL.

E.g:
```
Create Table R
( A int UNIQUE,
  B int UNIQUE,
  C int
);
```

A	B	C
2	12	-
4	NULL	-
6	15	-
N	16	-

Rel R not allowed in RDBMS

```
Create Table R
( A int UNIQUE NOTNULL,
  B int UNIQUE,
  C
);
```

Allowed in RDBMS.

(or)
```
Create Table R
( A int Primary key,
  B int UNIQUE,
  C int
);
```

Allowed for RDBMS (Preferred)

UNIQUE + NOT NULL ≠ Primary Key

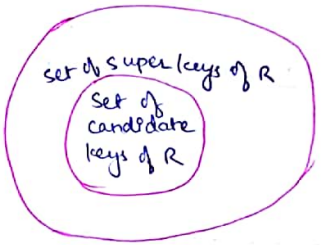
⇒ Superkey: Set of attributes of relation (R) which can differentiate records uniquely but may not be minimal attribute set.

∴ Candidate key = Minimal superkey.

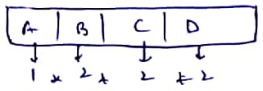
Student (SId , Sname Age)

Cand. Key = Sid

Super key = { Sid ,
Sid Sname ,
Sid Age ,
Sid Sname Age }



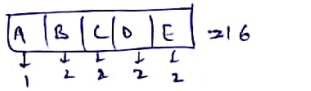
Q. R(A,B,C,D)
How many superkeys in Rel(R) if A: candidate key?



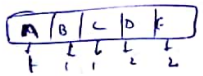
= 8

A. [any subset of B,C,D]
$2^3 = 8$

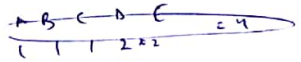
Q. R(A,B,C,D,E)
How many superkeys in R if {A, BC} is candidate key.



= 16



= 4



= 4

Total = 20

For A $= 2^4$ {B,C,D,E}
A, AB, AC, AD, AE, ACD, ACE, ABD, ADE, ABDE, ACDE

For B $= 2^3$ {A,D,E}
B, BC, BCD, BCE, BCDE, ABC, ABCD, ABCE, ABCDE

ABC{D,E}
$= 2^2 = 4$

$n(A \cup B) = n(A) + n(B) - n(A \cap B)$
$= 16 + 8 - 4$
$= 20$

Q. R(A,B,C,D,E,F) ; {AB, BCD}

A	B	C	D	E	F
1	1	2	2	2	2

= 16

A B C D E F : 4 ⇒ 20
1 1 1 1 2 2

(or) $2^{AB}_4 + 2^{BCD}_3 - 2^{ABCD}_2 = 20$

Q. R(A,B,C,D,E), How many sks, if A, BC, CD?

A|B|C|D|E = 16 (1 2 2 2 2) A B C D F = 4 (1 1 1 2 2) A B C D E = 2 (1 1 1 1 2)

22

A BC CD
$= 2^4 + 2^3 + 2^3 - 2^2 - 2^2 - 2^2 + 2 =$
$= 16 + 16 - 12 + 2$
$= 22$

Q R(A1 A2 A3 A4 .. An)
How many sk's if:

(a) {A1} cand. key 2^{n-1}

(b) {A1 A2, A3 A4} cand. keys

(c) {A1, A2, A3} cand. keys

~~(d) {A1A2, A3A4, A5A6}~~

(d) {A1A2, A2A3, A4A5} cand. keys

(e) if each attribute of R is candidate key, then, sk's of R: ____

(f) if all attributes of R form one cand. key ≠ 1 ____

(b) [A1|A2|A3|A4| .. |An] — n

$2^{n-2} + 2^{n-2} - 2^{n-4}$

[A1 A2 A3 A4 — —] — n−2

$= 2^{n-2} + 2^{n-4}$

$= \cancel{2^{n-4}}$

(c) A1 A2 A3 — — (1 1 1) 2^{n-3}
A1 A2 A4 — — (1 1 2) 2^{n-2}
A1 A2 An — — (1 1 1) 2^{n-3}

$\Rightarrow 2^{n-1} + 2^{n-2} + 2^{n-3}$ $7 \times 2^{n-3}$

(d) A1 A2 (1 1) A3 A4 (2 2) A5 A6 .. n

$2^{n-2} + 2^{n-2} + 2^{n-2}$
$- 2^{n-3} - 2^{n-4} - 2^{n-4}$
$+ 2^{n-5}$
$= 17 \cdot 2^{n-5}$

(e) $2^n - 1$ ⇒ Max no. of superkeys.

(f) 1

# of possible sk's in R with n attributes:

(a) 2^n (b) $2^n - 1$ (c) n! (d) (n−1)!

FOREIGN KEY: [Referential Key]

→ Used to relate data between tables.

Example:

Foreign key — Referencing relation

student(sid, sname, age) — (Referenced / Parent relation)

sid	sname	age
S1	A	20
S2	B	25
S3	B	22
S4	C	21
S5	B	22

All students

Enroll((sid), cid, fee)

sid	cid	fee
S1	C1	5000
S1	C2	5000
S2	C3	4000
S2	C3	5000
S5	C4	5000
~~S8~~	C1	5000

sid: Primary key

```
Create Table student
( sid Varchar(10) Primary key,
  sname Varchar(30),
  age integer,
  login varchar UNIQUE),
```

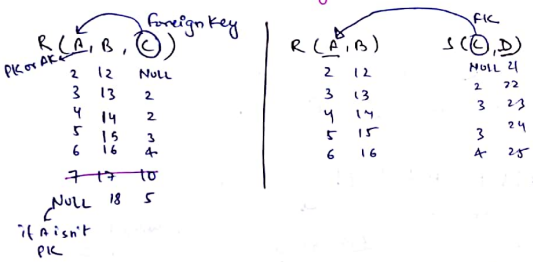
sid,cid: Primary key

```
Create Table Enroll
( sid varchar(10),
  cid varchar(30);
  fee integer,
  primary key(sid, cid),
  FOREIGN KEY(sid) REFERENCES Student
  ON DELETE CASCADE
  ON UPDATE CASCADE
);
```

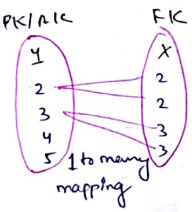
specify in case of alt. key

by default references primary key

Foreign Key: Set of attributes references to primary key / alternative key of same relation or some other relation.



* Record with Foreign Key value NULL is not in relationship.



* Each record of referencing relation can relate to at most one records of referenced relation.
* Each referenced record can be related by 0 or more (many) records of referencing relation.

04/11/2017

Referential Integrity Constraints: [FK constraints]

Referenced relation	Referencing relation
Insertion: No violation	Insertion: May cause violation
Deletion: May cause violation	Deletion: No violation
Updation: May cause violation	Updation: May cause violation

① Referenced Relation [student]:

(i) Insertion: No violation

(ii) Deletion: May cause violation.

(a) ON DELETE NO ACTION (default): Deletion of referenced record restricted if FK violation occurs.

(b) ON DELETE CASCADE: Because of deletion of referenced records DBMS deletes related referencing records.

(c) ON DELETE SET NULL: Allowed to delete referenced record & related FK field values set to NULL.

(iii) Updation: May cause violation.

(a) ON UPDATE NO ACTION (default)

(b) ON UPDATE CASCADE

(c) ON UPDATE SET NULL

② Referencing Relation [Enroll]:

(i) Insertion: Not allowed

(ii) Deletion: Allowed

(iii) Updation: Not allowed.

* There are no solutions as for referenced relation.

Check Constraint : [Constraint to Attribute values]

→ Used to specify set of accepted values of attribute (or) range of values to given attribute.

```
Create Table stud
( sid  varchar(10) Primary key,
  sname  varchar(30),
  age  Integer check age >= 20 and age <= 30, (NULL Not allowed by default)
  games varchar(10) check ("cricket", "chess", NULL)
);
```

Q (i) R (A B C)
AB: Primary key
S(D E F) D: Primary key
Attribute E of relation "S" foreign key ref. to rel. R allowed or not?
NOT allowed

R (A	B	C)
4	5	
4	8	
5	4	

NOT Allowed S(D E F)

(ii) check constraints can replace by foreign key. (FALSE)

R(A) S(D)
20
21
:
30
31

20-30

#NORMALISATION:*

→ Used to eliminate or reduce redundancy in DB tables.

→ Redundancy in DB table : If two or more independent relations stored in single RDBMS table.

Sid → Sname age [Stuols Info] CPd → Cname Instructor [Course Info] Sid cid → fee [Mapping data]

R

Sid	Sname	age	CPd	Cname	Instructor	fee
S1	B	20	C1	DB	Korth	5000
S2	B	20	C1	DB	Korth	4000
S3	A	22	C1	DB	Korth	4000
S3	A	22	C2	Algo	Cormen	6000
S3	A	22	C3	Algo	Allen weiss	5000

* Problems because of redundancy [Because of redundancy may cause inconsistency]

DB Anomalies:

(i) Updation anomaly: If one redundant copy is updated and some other redundant copy is not updated, which causes inconsistency.

(ii) Deletion anomaly: Because of deletion of some data, we may lose other useful data.

(iii) Insertion anomaly: May not be possible to insert one independent data w/o other independent information.

*→ If redundancy reduced/eliminated, then DB anomalies also reduced/eliminated.

→ Decompose the DB relation into two or more subrelation to reduce/eliminate redundancy.

R_1(Sid Sname age) R_2(Sid Cid fee) R_3(Cid Cname Instructor)

R_1	R_2	R_3
S1	S1 C1	C1
S2	S2 C1	C2
S3	S3 C1	C3
	S3 C2	
	S3 C3	

Normalised DB design:
- No redundancy
- No anomalies

Functional Dependency [FD]:

→ also called single valued dependency.

→ X → Y [FD representation]

R	X	Y
t_1	x_1	y_5
t_2	x_1	y_5

X → Y implied in R only if $t_1.x = t_2.x$ then $t_1.y = t_2.y$

[for each 'X' value of R, there must be only one 'Y' value]

R	A	B	C
	4	8	2
	6	5	3
	4	8	4
	6	5	6
	4	8	5
	7	5	7
	9	6	8

C.K.

A → B → implied

B → A is not implied

C → A implied

C → B implied

↓

superkey.

* Every superkey property is applicable to candidate key also.

X ⟶ Y
(Determinant) (Determiner)

Trivial FD: [self dependency] attribute determine themselves.

→ X → Y is trivial FD only if X ≥ Y

- A → A
- B → B
- C → C
- AB → A
- AB → AB

} Trivial FD

{* every possible trivial FD exists in R}

Non Trivial FD:

→ No common attribute over determinant and determiner.

$X \cap Y = \phi$

X → Y is non trivial FD.

E.g.
- A → B
- C → A
- C → B
- A → B

} Non trivial FD

Semi- non trivial FD:

→ Combination of trivial and non-trivial.

E.g. ① AC → BC
- AC → B
- AC → C

② A → AB
- A → A
- A → B

③ B → AC
- ~~B → A~~
- BC → A
- BC → C

Q. R

A	B	C
2	4	6
3	4	7
2	4	5
5	4	7
6	5	7

Find set of non-trivial FDs for the given instance of data records?

A → B

~~AB → C~~

AC → B

Q. R (A B C) Find set of non trivial FDs for given instance:

A	B	C
4	4	7
4	7	7
4	4	5
4	7	5
6	4	7

~~A → B~~

None

{ } // empty set of N.T. FDs

Eg ~~*~~ Stud (Sid Sname age) {Sid → Sname age}

Sid	Sname
S_1	A
S_2	B
S_3	C
S_4	C

defined from constraints of DB design.

R(MEid did) ; MEid : emp who manages
- each dept. ~~has~~ must have one manager
- each emp can manage many dept.

MEid	did
e1	d1
e1	d2
e2	d3
e3	d4

[did → MEid]

E.g. R (A B C)

A	B	C
4	5	1
4	5	2
6	7	3
7	7	4
4	8	4

new entry

~~B → A~~ is not implied TRUE

A → B implied in R, FALSE
may/may not

∴ FDs of R are not defined over given data, they are defined over the constraints of relation.

Armstrong Rules over FDs:

→ X Y Z are some set of attributes over relation R.

① Reflexivity: X → X always true.

② Transitivity: if X → Y & Y → Z implied in relation R.
then X → Z also implied in relation R.

R (A B C D)

A	B	C	D
4	5	8	
4	5	8	
5	7	9	
5	7	9	
6	7	9	
6	7	9	

if {B → C, C → D} exists in R
then B → D
also in R.

~~B → C ⇒ AB → AC~~

③ Augmentation: if X → Y then XZ → YZ

E.g ~~A~~ B → C ⇒ AB → AC

But AB → AC ⇏ B → C [may or may not be TRUE]

④ Split and Merge Rule: if X → YZ, then X → Y
split → X → Z

similarly X → Y, X → Z, then X → YZ
merge

But ⇒ XY → Z ⇏ X → Z
Y → Z [may or may not be TRUE]

Attribute Closure : (X^+)

X^+ = { set of all possible attributes which are determined by X recursively }

Given FD set
{A → B, BC → D, CD → E, B → C, EF → G}

$(A)^+ = \{A, B, C, D, E\}$ A → ABCDE

$(B)^+ = \{B, C, D, E\}$ B → BCDE

$(AF)^+ = \{A, B, C, D, E, F, G\}$ AF → ABCDEFG

Superkey:

→ Relation R with FD set (F)

R(X Y Z) X: superkey of R

2
3
4
5
6

{X → YZ}

X is superkey of relation R iff X^+ determines all attributes of R.

Eg R(A B C D) {A→B, B→CD}

2 4
3 5
7 6
8 6 7 8
9 6 7 8

super key.

$B^+ = \{B\ C\ D\}$ → not SK

$(A)^+ = \{A\ B\ C\ D\}$ → SK

Candidate Key: [Minimal superkey]

X is candidate key of relation iff:

① X must be superkey of rel R. [X^+ should determine all attr.]

② No proper subset of X is superkey.

$Y \subset X$ s.t. Y^+ should not determine all attributes.

R(A B C D E) {AB → C, B → D, A → E}

~~SK~~ $(AB)^+ = \{A, B, C, D, E\}$

$(A)^+ = \{A, E\}$

$(B)^+ = \{B, D\}$

Not SK.

AB is cand. key of R.

Q. Find the no. of candidate keys of given rel R? ***

① R(A B C D) {A → B, B → C}

cand key $(AD)^+ = \{A, B, C, D\}$

$A^+ = \{A\ B\ C\}$

$D^+ = \{D\}$

Determinants	Determines
(A)	B
B	C

* X must be a part of every candidate key of R only if:

- X doesn't exists in (or)
- X belongs to some determinant but not any determiner of non trivial FDs of in R.

* if X → Y, non trivial FD with "Y" is prime attribute of rel R. then R has atleast two candidate keys.

X → Y

Prime attribute

cand. key $(WY)^+ = \{$All attributes of R$\}$

X → Y

$(WX)^+ = \{W X Y, \_\ \_\}$

$(WY)^+ = \{W X Y, \_\ \_\}$

Q.2. R(A B C D E)

{AB → C, C → D, D → EA}

Soln:

$(CB)^+ = \{A, B, C, D, E\}$

$(AB)^+ = \{A, B, C, D, E\}$

$(BD)^+ = \{A, B, C, D, E\}$

{AB, BD, BC}

To test minimal sk:

[1] [proper subset of SK]$^+$ = should not determine all attributes.

(or)

[2] (some attr. of SK) should not determine (some other attr. of SK)

Q3 Find no. of candidate keys:

R(ABCD) { AB → CD, C → A, D → B }

✓ (AB)⁺ = {A, B, C, D} ✓ (AD)⁺ = {A, B, C, D}

CK ✓ (CB)⁺ = {A, B, C, D}

~~(BD)⁺ = {A, B, C, D}~~

✓ (CD)⁺ = {A, B, C, D}

∴

✗ (ABC)⁺ = {A, B, C, D}

Q4. R(ABCDE) { A → BC, CD → E, E → A, B → D }

(A)⁺ = {A, B, C, D, E}

(E)⁺ = {A, B, C, D, E}

~~(ACD)⁺~~ (CD)⁺ = {C, D, E, A, B}

(BC)⁺ = {A, B, C, D, E}

Q5. R(ABCDEF) { AB → C, C → DE, E → F, F → B }

(AB)⁺ = {A, B, C, D, E, F}

(AF)⁺ = {A, B, C, D, E, F}

(AE)⁺ = {A, B, C, D, E, F}

(AC)⁺ = {A, B, C, D, E, F}

Q6. R(ABCDEF) { AB → C, C → D, CD → E, DE → F, EF → B }

(AB)⁺ = {A, B, C, D, E, F}

(AEF)⁺ = {A, B, C, D, E, F}

(AED)⁺ = {A, B, C, D, E, F}

✗ (ADC)⁺ = {A, C, D, E, F, B}

(AC)⁺ = {A, C, D, E, F, B}

Q7. R(ABC). No non trivial FD in R.

ABC : Candidate key.

Membership Test :

→ Given FD set (F);

X → Y FD is logically implied (member of) in FD set iff X^+ determines Y in FD set (F).

{ . . . . . } ⇐ X → Y

if X^+ : consist Y, then X → Y is member of FD set

otherwise X → Y is not member of FD set.

E.g { AB → C, BC → D, D → E, DE → F }

(a) AB → F member of FD set?

(AB)⁺ = {A, B, C, D, E, F} ∴ AB → F is member of FD set.

(b) BC → A member of FD set?

(BC)⁺ = {B, C, D, E, F} BC → A is not member of FD set.

Closure of FD set:

→ Given FD set (F)

F^+ : set of all possible FD's which can ~~don't~~ be derived by using given FDs.

F = { A → B, B → C }

F^+ { (A)⁺ = ABC : A → A, A → B, A → C, A → AB, A → BC, A → AC, A → ABC

(B)⁺ = BC : B → BC, B → B, B → C

(C)⁺ = C : C → C

(AB)⁺ = ABC : AB → A, AB → B, AB → C, AB → AB, AB → BC, AB → AC, AB → ABC

(BC)⁺ = BC : BC → B, BC → C, BC → BC

(AC)⁺ = ABC : AC → A, AC → B, AC → C, AC → AB, AC → BC, AC → AC, AC → ABC

(ABC)⁺ = ABC : ABC → A, ABC → B, ABC → C, ABC → AB, ABC → BC, ABC → AC, ABC → ABC }

* FD set closure identification is NP complete problem. [exponential TC]

~~F = F⁺~~] ~~Expressive power term.~~

[F ≡ F⁺] expressive power equal expressive term.

Equality of FD sets:

→ Given FD sets F & G

⇒ F & G equal expressive iff $F^+ = G^+$

⇒ F & G equal expressive iff

ⓐ F covers G: Every FD of G set must be ~~not~~ member of F set.

$F \supseteq G$

&& ⓑ G covers F: Every FD of F set must be member of G set.

$F \subseteq G$.

*

F covers G	G covers F	Result
True	True	F = G
True	False	F ⊃ G
False	True	F ⊂ G
False	False	F & G are not comparable

Q. $F = \{A \to B, B \to C, C \to A\}$

$G = \{A \to BC, B \to AC, BC \to A, AB \to C\}$

Which is true?

ⓐ F ⊂ G ⓑ F ⊃ G ⓒ F = G ⓓ None

F covers G ⇒ F: $A \to B$, $B \to C$, $C \to A$ [True]

⇐ G: $A \to BC$ ✓, $B \to AC$ ✓, $BC \to A$ ✓, $AB \to C$ ✓

$(A)^+ = \{A, B, C\}$

$(B)^+ = \{C, A, B\}$

$(BC)^+ = \{B, C, A\}$

$(AB)^+ = \{A, B, C\}$

G covers F ⇒ G: $A \to BC$, $B \to AC$, $BC \to A$, $AB \to C$ [False]

⇐ F: $A \to B$ ✓, $B \to C$ ✓, $C \to A$ ✗

$(A)^+ =$ ·

$(C)^+ = \{C\}$

Q. $F = \{A \to BCDEF, BC \to ADEF, B \to F, D \to E\}$

$G = \{A \to BC, BC \to AD, B \to F, D \to E\}$

ⓐ F ⊂ G ⓑ F ⊃ G ✓ ⓒ F = G ⓓ None

F covers G ⇒ True: $A \to BCDEF$, $BC \to ADEF$, $B \to F$

G covers F ⇒ True: $D \to E$

$A \to BC$, $BC \to AD$, $B \to F$, $D \to E$

Minimal Cover (or) Canonical Cover:

→ Given FD set F:

• Minimal cover of FD set(F): Minimal possible FDs which are logically equal to F.

* Procedure to find minimal cover:

Given FD set(F)

① Remove extraneous attributes from every determinant of FD set(F).

if $WX4 \to Z$, $W^+ = X$,

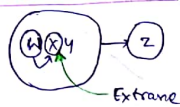
then attr. "X" is extraneous in ~~wxy~~ $WX4 \to Z$

② Remove all redundant FDs from step ① result:

$X \to Y$ redundant in FD set F_1, iff. $X \to Y$ must be member of $\{F_1 - (X \to Y)\}$ FD set.

* Extraneous Attribute over determinant :

{ WXY → Z, W → X }
≡ { WY → Z, W → X }



Eg: ① { ABCD → E , AB → F , F → D }

ABC → E , AB → F , F → D

② { AB → E , AC → F , F → B , C → D }

AC → EF , F → B , C → D (or) AC → F , AC → F , F → B , C → D

③ { A → B , C , A → B , B → A } (or) { AB → C , A → B , B → A }

{ B → C , A → B , B → A } (or) { A → C , A → B , B → A }

* Redundant FD in FD set :

→ X → Y in FD set (F) is redundant iff X → Y must implied in FD set { F − (X → Y) }

{ F − (X → Y) } ⇐ X → Y must be member.

∴ X → Y is redundant in FD set (F).

Eg: ① F = { A → B, B → C, A → C } : A → C is member

② F = { AB → C, BC → D, AB → D, CD → E, BC → E }

* When checking for redundancy, assume that the FD is not present in FD set, and then try to produce the respective FD.

Q. F = { A → BCDEF, BC → ADEF, ACF → D, D → E, B → F } , find minimal covers.

① A → BCDEFF, BC → ADEF, ~~AE~~ A → D, D → E, B → F

② if A → D is redundant

~~A → BCDEF, BC → ADEF, D → E, B → F~~ A → B, A → C, A → D, A → E, A → F ; BC → A, BC → D, BC → E, BC → F ; D → E ; B → F

{ A → BC , BC → AD , D → E , B → F }

---

if BC → D is redundant

{ A → BCD, BC → A, D → E, B → F }

* Minimal covers of FD set (F) may not be unique but all minimal covers are logically unique [equally expressive].

$F_{m_1} \equiv F_{m_2} \equiv F$

Properties of Decomposition : ***

→ Decomposition of Rel (R) into sub relation not causes any integrity violation if :

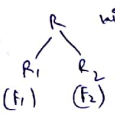
* lossless join decomposition &&
* dependency preserving decomposition

① Lossless join ~~decomposition~~ decomposition :

→ JOIN result of subrelations must be equal to original relation.

$R_1 \bowtie R_2 \bowtie R_3 = R$

② Dependency preserving decomposition :



with FD set (F) . Because of decomposition should not lose any FD of FD set (F).

Lossless Join Decomposition :

→ Rel R decomposed into $R_1\ R_2\ R_3 \dots R_n$ sub relations.

In general,

$[R_1 \bowtie R_2 \bowtie \dots \bowtie R_n] \supseteq R$

* result of SQL queries is inconsistent.

* If $(R_1 \bowtie R_2 \bowtie \dots \bowtie R_n) \equiv R$, then Lossless join decomposition.

* If $(R_1 \bowtie R_2 \bowtie \dots \bowtie R_n) \supset R$, then Lossy join decomposition.

Eg.① R

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S3	B	C3

{Sid → Sname}

Candidate key : Sid Cid

① decomposed into R_1 (Sid Sname) R_2 (Sid Cid) {Sid → Sname}

R_1 Sid	Sname	R_2 Sid	Cid
S1	A	S1	C1
S2	B	S1	C2
S3	B	S2	C2
		S3	C3

(natural join) → $R_1 \bowtie R_2$

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S3	B	C3

$R_1 \bowtie R_2 \equiv R$

Lossless Join Decomposition

② decomposed into R_1 (Sid Sname) R_2 (Sname Cid)

R_1 Sid	Sname	R_2 Sname	Cid
S1	A	A	C1
S2	B	A	C2
S3	B	B	C2
		B	C3

$R_1 \bowtie R_2$

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S2	B	C2
S2	B	C3
S3	B	C2
S3	B	C3

Lossy Join Decomposition

spurious tuples

$R_1 \bowtie R_2 \supset R$

Eg.② R (A B C) {A → B, B → C}

4 5 4

5 5 4

6 7 5

7 7 5

8 6 5

{A} → candidate key.

① decomposed R into

R_1 (A B) R_2 (B C) B → C

A	B
4	5
5	5
6	7
7	7
8	6

B	C
5	4
7	5
6	5

$R_1 \bowtie R_2 \equiv R$

Lossless Join decomposition

② decomposed R into

R_1 (A C) R_2 (B C)

A	C
4	4
5	4
6	5
7	5
8	5

B	C
5	4
7	5
6	5

$R_1 \bowtie R_2 \supset R$

Lossy Join Decomposition

***

Relation R with FD set (F) decomposed into R_1, R_2 subrelations

Given decomposition is lossless join decomposition iff :

① $(R_1 \cup R_2) \equiv R$

R(ABCDE) → R(ABC) R(CD) ×

R(ABCDE) → R(ABC) R(CDE) ✓ //every attribute of R in some subrelation

* No attribute should be lost because of decomposition.

&& ② $(R_1 \cap R_2) \xrightarrow{\text{determines}} R_1$ //$R_1 \cap R_2$ is sk for R_1

(or)

$(R_1 \cap R_2) \xrightarrow{\text{determines}} R_2$ //$R_1 \cap R_2$ is sk for R_2

* No spurious tuples/records.

$\{(R_1 \bowtie R_2) \equiv R\} \Leftrightarrow \{(R_1 \cup R_2 \equiv R)$ and $R_1 \cap R_2 \to R_1$ or $R_1 \cap R_2 \to R_2\}$

Q. R(A B C D E)

{AB → C, C → D, B → E} ; Test which given decompositions are lossless join or lossy join.

(i)(ii) & (iii) are 3 possibilities of lossy decomposition.

(i) ✗ R_1 (ABC) R_2 (CD) ×

$R_1 \cup R_2 \neq R$ lossy because of attribute "E" lost because of decomposition

(ii) ✗ R_1 (ABC) R_2 (DE) ×

$R_1 \cup R_2 = R$ ✓ and $R_1 \cap R_2 = \phi$ lossy join decomposition.

(iii) ✗ R_1 (ABC) R_2 (CDE) ×

$R_1 \cup R_2 = R$ ✓ and $R_1 \cap R_2 \to R_1$ or R_2 ×

C is not superkey for any relation.

(iv) R_1 (ABCD) R_2 (BE) ✓

$R_1 \cup R_2 = R$ ✓ and $R_1 \cap R_2 = R_2$ ✓

Lossless join decomposition

$(B)^+ = BE$

(v) R_1 (ABCD) R_2 (ABE) ✓

$R_1 \cup R_2 = R$ and $R_1 \cap R_2 \to R_1/R_2$

$(AB)^+ = ABCDE$

Lossless join decomposition.

05/11/2017

Q. R(ABCDEFG)
{AB→C , BC→A, AC→B, B→D, DE→F, F→G}

① D_1 {ABC, DEF, FG}
R_1 R_2 R_3

- $R_1 \cup R_2 \cup R_3 \equiv R$ ✓
- Join relations only if common attribute is key for one of sub relation. (LLJ).

$R_1(ABC)$ $R_2(DEF)$ $R_3(FG)$

$R_1(ABC)$ $R_{23}(DEFG)$] ⇒ Lossy join

② D_2 {ABC, BD, DEF, FG, ABDE}

$R_1(ABC)$ $R_2(BD)$ $R_3(DEF)$, $R_4(FG)$ $R_5(ABDE)$

$R_{12}(ABCD)$ $R_{34}(DEFG)$ $R_5(ABDE)$ or $R_{12}(ABCD)$ R_{345}

$R_{125}(ABCDE)$ $R_{34}(DEFG)$ $R_{345}(ABDEFG)$ $R_{12}(ABCD)$

~~R_{1} R_{2} R_{3}~~ $R_{12534}(ABCDEFG)$ Lossless join

Dependency Preserving Decomposition:

→ Relational schema R with FD set (F) decomposed into $R_1 R_2 R_3 \ldots R_n$ subrelations with F_1 F_2 F_3 ..... F_n FDs.

In general $\{F_1 \cup F_2 \cup F_3 \cup \ldots \cup F_n\} \subseteq F$

→ if $[F_1 \cup F_2 \cup \ldots \cup F_n] \equiv F$, then dependency preserving decomposition.
→ if $[F_1 \cup F_2 \cup \ldots \cup F_n] \subset F$, then NOT dependency preserving decomposition
↳ // Because of decomposition some FD of F is lost.
↳ // every FD of F preserved in decomposition.

Q. R(ABCDEFG)
{AB→C, BC→A, AC→B, B→D, DE→F, F→G}
D_2 {ABC, BD, DEF, FG, ABDE}, Dependency preserving or NOT?

$R_1(ABC)$	$R_2(BD)$	$R_3(DEF)$	$R_4(FG)$	$R_5(ABDE)$
AB→C BC→A AC→B	B→D	DE→F	F→G	B→D

} Dependency preserving decomposition.

$F_1 \cup F_2 \cup F_3 \cup F_4 \cup F_5 \cup F_6 \equiv F$

* Q. R(ABCDE) {A→B, B→C, C→D, D→BE} → (A)+ = A,B,C,D,E; (B)+ = B,C,D,E; (C)+ = C,D,B,E; (D)+ = D,B,E,C; (E)+ = E

D_1 {~~AB~~ AB, BC, CD, DE}

$R_1(AB)$	$R_2(BC)$	$R_3(CD)$	$R_4(DE)$
A→B	B→C	C→D	D→E
	C→B	D→C	
F_1	F_2	F_3	F_4

A→B in R_1
B→C in R_2
C→D in R_3
D→BE in D→E in R_4
D→B in R_3 and R_2

$F_1 \cup F_2 \cup F_3 \cup F_4 \equiv F$ ∴ dependency preserving decomposition.

Q2 R(ABCD) {AB→CD, D→B}

D{ACD, BD, ABC}

$R_1(ACD)$	$R_2(BD)$	$R_3(ABC)$
~~A→C~~	~~D→D~~	AB→C
AD→C	D→B	
F_1	F_2	F_3

$(AB)^+ = AB,C,D$
$(D)^+ = D,B$
$(A)^+ = A$
$(B)^+ = B$
$(AD)^+ = (A,D,B,C)$

AB→C in R_3

AB→C ⇒ ~~AB→D~~, ~~D→B~~ ✗

$F_1 \cup F_2 \cup F_3 \subset F$

Not dependency preserving decomposition.
AB→D is lost.

Q4 R(ABCDEF)

{A→BCDEF, BC→ADEF, B→F, D→E}

D{ABC, BCD, BF, DE}

$R_1(ABC)$	$R_2(BCD)$	$R_3(BF)$	$R_4(DE)$
A→BC	BC→D	B→F	D→E
B→A			
~~BC~~→F	f_2	f_3	f_4

$(A)^+ = ABCDEF$
$(B)^+ = BCDEFA$

A→BCDEF in R_1, R_2, R_3, R_4
BC→ADEF in R_1, R_2, R_3, R_4
B→F in R_3
D→E in R_4

$F_1 \cup F_2 \cup F_3 \cup F_4 \equiv F$; Dependency preserving decomposition.

Normal Forms :***

→ Used to identify degree of redundancy [also used to eliminate/reduce redundancy].

→ Redundancy in relation (R) can be because of

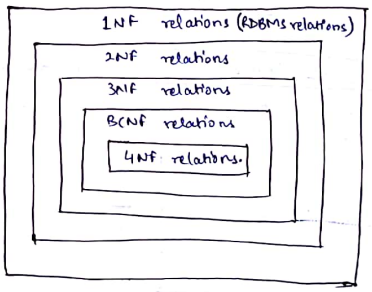
Non trivial FD(X→Y) [Single valued dependency]	Non Trivial MVD(X→→Y) [Multi valued dependency]
→ To eliminate redundancy over FDs, relation should decompose into BCNF.	→ To eliminate redundancy over MVDs relation should decompose into 4NF.

* Normal Forms:

Defined over FDs: 1NF, 2NF, 3NF, BCNF ⇒ 0% redundancy over FD set of relation if relation in BCNF. But may contain redundancy over MVDs.

Defined over MVDs: 4NF ⇒ 0% redundancy over FDs & MVDs if Rel in 4NF.

X→→Y



First Normal form (1NF):

→ default NF of RDBMS relations.

→ definition: Relation R in 1NF iff no Multivalued attribute in R.

(or)

Every attribute of R must be atomic [single valued].

⇒ R (Sid | Sname | (Cid)) ← multi valued attribute

Sid	Sname	Cid
S1	A	c1/c2
S2	B	c2/c3
S3	B	c3

[Set of Multiple values over attribute for any record.]

{Sid → Sname} NOT in 1NF

↓ 1NF design

1NF relation ✓ allowed in RDBMS

R (Sid | Sname | Cid) {Sid → Sname}

Sid	Sname	Cid
S1	A	c1
S1	A	c2
S2	B	c2
S2	B	c3
S3	B	c3

{Sid Cid : cand. key}

1NF design ⟺ design cand. keys based on RDBMS constraint.

Eg: ① R (Sid | Sname | age | Cid | phno. | email) {Sid → Sname. age} {Sid : cand. key}

Sid	Sname	age	Cid	phno.	email
S1	A	20	c1/c2 (2)	P1/P2/P3 (3)	e1/e2/e3/e4 (4)
S2	B	22	c2/c3 (2)	P4 (1)	e5/e6 (2)

↓ 1NF design

R (Sid | Cid | phno. | email | Sname | age) {Sid → Sname. age} {Sid Cid phno email : CK}

Sid	Cid	phno.	email	Sname	age	
S1	C1	P1	e1	A	20	24 tuples
S1	C1	P1	e2			
:	:	:	:	:	:	
S1	C2	P3	e4	A	20	
S2	C2	P4	e5	B	22	4 tuples
:	:	:	:	:	:	
S2	C3	P4	e6	B	22	

*Drawback: Degree of redundancy is high because 1NF design goal is to convert data into RDBMS but not decrease redundancy.

→ X → Y, non-trivial FD forms redundancy in Rel R iff Non trivial FD with X is not superkey.

(X) → Y : Non Trivial FD
not SK

R (Sid | Sname | Cid) Sid → Sname Not SK

redundancy

Sid	Sname	Cid
S1	A	C1
S1	A	C2
S1	A	C3
S2	B	C3

→ X → Y, FD in R not forms any redundancy iff:

① Trivial FD: X ≥ Y
(or)

② X must be superkey.

① R (Sid | Sname | age) {(Sid) → Sname. age} (Sid: SK)

Sid	Sname	age
S1	A	20
S2	A	20
S3	B	22
S4	B	22

② R (A | B | C) {A → BC} (A: SK)

A	B	C
1	2	5
2	2	5
3	3	7
4	3	7
5	3	7

③ R (A | B | C) {A → B, B → C} (A: SK; B: not SK)

A	B	C	
1	2	5	
2	2	5	redundancy
3	3	7	
4	3	7	
5	3	7	

④ R (A | B | C | D | E | F) Prime | Non Prime

{AB → C, B → D, D → E, AE → F, C → A}

{AB, BC} (AB: SK; B: not SK; D: not SK; AE: not SK; C: not SK)

(AB)+ = {A, B, C, D, E, F}

(B)+ = {B, D, E}

(C)+ = {A, C}

(D)+ = {D, E}

(AE)+ = {A, E}

Prime → Prime → ~~SK~~ (No redundancy)

Prime → Non-prime → Redundancy

Non-prime → non-prime → Redundancy.

⇒ X : Not superkey of Rel R :

① X proper subset of cand. key
(or)

② X non-prime attributes
(or)

③ X proper subset of CK combines with non-prime attributes.

⇒ Non Trivial FD X→Y with "X" not superkey	1NF	2NF	3 NF	BCNF [Not allowed any non trivial FDs which form redundancy]
① [Proper subset of CK] → [Non Prime attri.] Not SK B → D	Allowed	Not Allowed	Not allowed	Not allowed
② [Non Prime attr.] → [Non Prime attr.] Not SK D → E	Allowed	Allowed	Not allowed	Not allowed
③ [Proper subset of CK & non prime attr.] → [Non Prime attr.] Not SK AE → F	Allowed	Allowed	Not allowed	Not allowed
④ [Proper subset of one CK] → [Proper subset of other cand. key] Not SK C → A	Allowed	Allowed	Allowed	Not allowed

Q. If rel R is in 3NF but not in BCNF, then, which FD must be ~~pres~~ present in R?

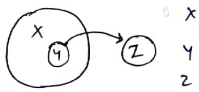
⇒ [Proper subset of Cand. key] → [Proper subset of other CK] must exist in R.

Q. If relation is in 3NF and no non trivial FD such that [Proper subset of one cand. key] → [Proper subset of other CK], then rel. R ~~atts~~ also must be in BCNF.

Second Normal Form (2NF):

→ Relational Schema (R) is in 2NF iff:
- No partial dependency in relation R.

* Partial Dependency:



X : any cand. key
Y ⊂ X
Z : Non Prime attribute

Y ⊂ X
Y → Z : Partial dependency.

Third Normal Form (3NF):

→ Relational Schema (R) in 3NF iff: (or)
Every non trivial FD X→Y in R with
① X must be super key
(or)
② Y must be prime attribute
⇒ X → Y : X super key (or) Y Prime Key

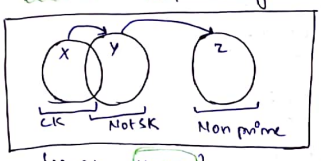
{ X → Y, Y → Z }
X : super key, Z : Prime

(Cand. key) → Non Prime

(Cand. key (or) Not superkey) → Prime

Definition:
Rel R in 3NF iff
No Transitive Dependency

* Transitive Dependency:



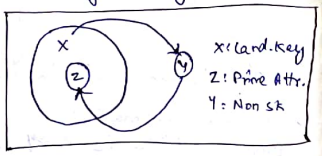
$\{X \to Y, (Y \to Z)\}$ → Transitive Dependency

* Non-prime attribute (Z) transitively determined by superkey or candidate key.
* Transitive Dependency [Not allowed in 3NF]

$\{\underbrace{X}_{SK} \to Y, \underbrace{Y}_{Not\ SK} \to \underbrace{Z}_{Non\ prime}\}$

⇒ Prime attribute transitively determined by superkey allowed in 3NF.

$\{\underbrace{X}_{SK} \to Y, \underbrace{Y}_{Not\ SK} \to \underbrace{Z}_{Prime}\}$



e.g. R(ABC) $\{AB \to C, C \to A\}$

CK: $\{AB, BC\}$ — SK: AB

⇒ Allowed in 3NF

⇒ C → A forms redundancy in relation.

BCNF [Boyce Codd Normal Form]:

→ Relational Schema R in BCNF iff every non trivial FD $X \to Y$ in R, with X must be superkey.

* All non trivial FDs such that determinant must be superkey.

Superkey → Prime/Non Prime

$X \to Y$, $X \geq Y$ → No redundancy.

→ {No redundancy in R over FDs} ⇒ Rel. R is in BCNF.

→ {Redundancy exists in R over FDs} ⇒ Rel. R not in BCNF

Highest NF of given relation: * * *

[FD ⇒ Partial dependency]

Q1. R(ABCDE)

$\{ABD \to C, BC \to D, CD \to E\}$

(a) 1NF (b) 2NF (c) 3NF (d) BCNF

Sol^n: Candidate Keys: $(ABD)^+ = \{A, B, C, D, E\}$, $(ABC)^+ = \{A, B, C, D, E\}$ → {ABD, ABC}

Prime ⇒ ABCD

Non Prime ⇒ E

* Remove trivial FD if given.

(ABD) → C, (BC) → D, (CD) → E

ABD: SK; C: prime; BC: not SK but prime; D: prime; CD: not SK nor prime; E: non prime

	ABD → C	BC → D	CD → E
⇒ BCNF ⇒	✗ ✓	X	X
⇒ 3NF ⇒	✗ ✓	✓	X
⇒ 2NF ⇒	Not partial dependency	Not PD	Not PD

BC → D, CD → E ⇒ (BC → E) → PD

* 2NF Test : $\forall$ (Proper subset of cand. key)$^+$ = {always prime}

↳ even if there is anyone non prime attribute, rel. not in 2NF (PD exists)

* ABD ⇒ $A^+ = A$, $B^+ = B$, $D^+ = D$, $AB^+ = AB$, $BD^+ = BD$, $AD^+ = AD$

ABC ⇒ $C^+ = C$, $AC^+ = AC$, $BC^+ = B, C, D, E$ → Non prime

rel. R not in 2NF.

BC → E PnR is PD.

Q② R(ABCDE) {AB→C, C→D, B→E} AB : cand. key.

Partial dependency

Highest NF = 1NF.

Q③ R(ABCD) {AB→C, C→A, AC→D}

Cand. key: {AB, BC} Prime ⇒ A, B, C Non prime ⇒ D

~~AB~~	AB→C	C→A	AC→D
BCNF	✓	x	x
3NF	✓	✓	X
2NF	~~NOT~~ PD	NOT PD	NOT PD
~~1NF~~			

1NF

AB ⇒ $A^+ = A$, $B^+ = B$ ~~$AB^+ = ABCD$~~

BC ⇒ $B^+ \cap C = B$, $C^+ = A, D$

C → D → PD

Q④ R(ABCD) {AB→C, BC→D}

CK = {AB} P ⇒ A, B NP ⇒ C, D

	AB→C	BC→D
(2NF)		
BCNF ⇒	~~✓~~	x
3NF ⇒	✓	x

AB ⇒ $A^+ = A$, $B^+ = B$

2NF

Q⑤ R(ABCDEF) {AB→C, C→D, D→AE, DE→F, EF→B}

CK = {AB, AEF, ~~DEF~~, C} P ⇒ ABCDEF NP ⇒ ~~{}~~

No BCNF

3NF

Q⑥ R(A,B,C,D,E,F) {AB→C, C→DE, E→F, F→B}

CK = {AB, AF, AE, AC} P ⇒ A B C EF NP ⇒ D

No BCNF ~~AB ⇒ A^+~~ 1NF

No 3NF

No 2NF ⇒ C→D is PD

Q⑦ R(ABCDE) {A→BC, CD→E, E→A, B→D}

CK = {A, E, BC, CD} P ⇒ ABCDE NP ⇒ {}

No BCNF 3NF

Q⑧ R(ABCD) {AB→C, BC→A, AC→B}

CK = ~~{ABD}~~ {ABD, ACD, BCD} P ⇒ ABCD NP ⇒

No BCNF

3NF

Q9. R(ABCD) {AB→CE, BD→F, E→F}

CK = AB → C CK = ABD PA = ABD, NPA = C

AB → C is PD Not 2 NF ∴ 1NF

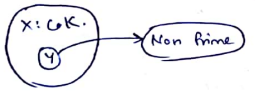
Q10. R(ABC) {AB→E, DE→C}

∴ R(ABC) with no non trivial FDs.

∴ R is in BCNF.

Q11. If rel R with only simple cand. keys (only 1 attribute) which is true about R?

(a) R in BCNF
(b) R in 3NF but may not BCNF
(c) R in 2NF but may not 3NF
(d) R in 1NF but may not 2NF



Not possible [only simple CKs in R]

Eg R(ABCD) {A→B, B→AC, C→D} C→D: Not 3NF

CK = {A, B}

Eg R(ABC) {A→B, B→AC} 2NF ✓ 3NF ✓ BCNF ✓

CK = {A, B}

Q12. If every attr. of R is Prime attribute, then, R

(a) (b) 3NF but may not BCNF. (c) (d)

Q13. If R is in 3NF (and) at most one compound candidate key in Rel R. (remaining candidate keys are simple), then

(a) R also in BCNF
(b) R may not BCNF
(c) R not BCNF
(d) can not decide

Q14. R(ABC) with no non trivial FDs, highest NF of R.

BCNF

Non non trivial FDs ⇒ No redundancy over FDs.

Q15. Relation R with only two attributes Rel R always in BCNF [also in other higher NFs]

R(AB) CK = AB (or) B→A

① R(AB) {A→B} BCNF
② R(AA) {B→A} BCNF
③ R(AB) {A→B, B→A} BCNF
④ R(AB) {Non non-trivial FDs} BCNF

Q16 Eg. R(ABCDEFGHI) » CK = {AB, C} PA = ABC NP = DE

CK = {A, BC} R(ABCDEFGHI) CK = (AB, C, D, G, H)

AD: BCE DE→AE A→D

A→D BC→E DE→A B→C C→A

Decomposition into higher NF :-

→ Rel R decompose into 2NF, 3NF, BCNF with
- Lossless join decomposition
- Dependency preservation

① R(ABCDE) {AB → C, C → D, B → E}

AB: CK — R in 1NF but not in 2NF — B → E: partial dependency

2NF decomposition:

Min 2 rel. tables, 1 FK

R1(ABCD): AB → C, C → D; AB: CK
R2(BE): B → E; B: CK

- Lossless join
- Dependency preserving
- No PD

3NF decomposition:

~~2M~~ Min 3 rel. tables, 2 FK

R1(ABC): AB → C; CK
R2(CD): C → D; CK
R3(BE): B → E; CK

- Lossless join
- Dependency preserving
- AT 3NF
- BCNF

② R(ABCDEF) {AB → C, B → D, A → E, E → F}

CK: AB

Partial dependency

2NF

(ABC): AB → C
(BD): B → D
(AEF): A → E, E → F

LLJ ✓
DP ✓
No PD ✓
2NF

3NF

(ABC): AB → C
(BD)
(AE)
(EF)

LLJ ✓
DP ✓
3NF ✓
BCNF ✓

③ R(ABCD) find no. of relational tables required for 2 NF.

{A → C, B → D}

CK = AB

(ABC): A → C
(BD): B → D
(AB)
(AC): A → C
(BD): B → D

Min 3

④ R(ABCDEF) {A → D, B → E, C → F}

CK = ABC

(ABC)
(AD): A → D
(BE): B → E
(CF): C → F

4 relation tables

⑤ R(ABCDE) {AB → C, BC → A, AC → B}

{ABDF, ACDE, BCDF} C.K.

R in 3NF but not in BCNF

(ABC): AB → C, BC → A, AC → B; {AB, BC, AC}
(DE)
(ABDE): {AB}; No non trivial FDs.

LLJ ✓
DP ✓
2 relation tables

⑥ R(ABC) {AB → C, C → A}

{AB, BC} 3NF

(ABC): AB → C
(CA): C → A

Non trivial FDs
SK

LLJ ✓
BCNF ✓
DP ✗
2 relation tables

Rel R not possible to decompose BCNF with dependency preserving decomposition.

DB design Goals	1NF	2NF	3NF (Most accurate Normal form)	BCNF
① 0% redundancy	NO	NO	NO	YES over FDs NO over MVDs
② Lossless join * decomposition	YES	YES	YES	YES Lossless join decomposition of 1NF, 2NF, 3NF, BCNF possible for every relation.
③ Dependency * Preservation Decomposition	YES	YES	YES Every relation can decompose into ~~1NF~~, ~~2NF~~, 3NF with dependency preserving decomposition.	May not Not every relation can decompose into BCNF with ~~fate~~ dependency preserving decomposition

* 3NF is considered as most accurate normal form.

* Lossless join decomposition and dependency preservation are of highest priority.

If you are achieving BCNF ~~but~~ at the cost of Dependency preservation, there is no need to ~~conv~~ decompose into BCNF, as it may cause inconsistency in the database.

QUERIES

⇒ Relational Algebra (Mathematical formula) — Procedural Query language
Formulation of What data retrieved (and) How data retrieved

⇒ SQL → used non procedural
⇒ Relational Calculus (Mathematical formula)
Non procedural Query language
Formulation of What data retrieved

#Relational Algebra:

* Basic Operators:

- π : Projection (Unary)
- σ : selection (Unary)
- $\times$: Cross product
- $\cup$: Union
- $-$: Minus (set difference)
- ρ : Rename (Unary)

* Derived Operators:

- $\cap$: Intersection [using $-$]
- $\bowtie$: Join [using $\pi, \sigma, \times$]
- $/$: Division [using $\pi, \times, -$]

* π : Projection:

→ used to project required list of attributes from Rel(R)

$\pi_{Attr\_list}(R)$

* σ : selection:

→ used to retrieve records which satisfy the predicate condition (P).

$\sigma_p(R)$

E.g. R

A	B	C
4	6	8
7	5	8
5	6	8
8	3	5

$\pi_{BC}(R)$: distinct tuples only in result

B	C
6	8
5	8
3	5

$\sigma_{A \le 6}(R)$

A	B	C
4	6	8
5	6	8

$\pi_{BC}(\sigma_{A \le 6}(R))$

B	C
6	8

* X : Cross Product :

→ R X S results all attributes of R followed by all attributes of S and each record of R pairs with every record of S.

X no. of attr.

R

A	B	C
2	4	5
3	4	6
7	5	5

n distinct tuples

Y no. of attr.

S

C	D
7	9
5	8

m distinct tuples

⇒ R X S : n x m distinct tuples ← X+Y no. of attr. →

A	B	C	C	D
2	4	5	7	9
2	4	5	5	8
3	4	6	7	9
3	4	6	5	8
7	5	5	7	9
7	5	5	5	8

NOTE

Rel R with n tuples
Rel S with 0 tuples // empty
# of tuples in R X S = 0 tuples. // empty

$A \times \underline{B} = \{\}$

$\{\}$

JOINS :

1. Natural Join ($\bowtie$)
2. Conditional Join ($\bowtie_c$)
3. Outer Joins
   (i) Left outer join ($\overset{\frown}{\bowtie}$ or $\overset{\frown}{\bowtie}_c$)
   (ii) Right outer join ($\overset{\smile}{\bowtie}$ or $\overset{\smile}{\bowtie}_c$)
   (iii) Full outer join ($\overset{\frown\smile}{\bowtie}$ or $\overset{\frown\smile}{\bowtie}_c$)

④ Equi Join ($\bowtie_=$)

Conditional join with equal conditions

Natural Join :

$$R \bowtie S \equiv \pi_{\text{distinct attributes}} \left(\sigma_{p} (R \times S) \right)$$

p : equality of common attr. of R & S

E.g. R X S

R ⋈ S

A	B	C	C	D
2	4	5	7	9
2	4	5	5	8
3	4	6	7	9
3	4	6	5	8
7	5	5	7	9
7	5	5	5	8

E.g. $R \bowtie S \equiv \pi_{ABCD} \left(\sigma_{R.C=S.C} (R \times S) \right)$

R ⋈ S

A	B	C	D
2	4	5	8
7	5	5	8

E.g. $T_1(A\,B\,C) \quad T_2(B\,C\,D)$

$$T_1 \bowtie T_2 = \pi_{ABCD} \left(\sigma_{\substack{T_1.B=T_2.B \\ T_1.C=T_2.C}} (T_1 \times T_2) \right)$$

E.g. $T_1(A\,B) \quad T_2(C\,D)$

T_1: (4,5), (6,7) $\quad T_2$: (3,5), (4,5)

$T_1 \bowtie T_2 = T_1 \times T_2 = \pi_{ABCD} \left(\sigma_{\phi} (T_1 \times T_2) \right)$

Natural join equal to cross product, if join condition empty.

$T_1 \bowtie T_2$

A	B	C	D
4	5	3	5
4	5	4	5
6	7	3	5
6	7	4	5

R(A B C D)

$R_1(A\,B) \quad R_2(C\,D)$

$R_1 \bowtie R_2 = R_1 \times R_2 \supset R$ ∴ Lossy decompaition.

E.g. R

A	B	C
4	6	8
7	5	8
5	6	8
8	3	5

$\pi_{BC}(R)$: distinct tuples only in result

B	C
6	8
5	8
3	5

$\sigma_{A \le 6}(R)$

A	B	C
4	6	8
5	6	8

$\pi_{BC}(\sigma_{A \le 6}(R))$

B	C
6	8

* X : Cross Product :

→ R X S results all attributes of R followed by all attributes of S and each record of R pairs with every record of S.

X no. of attr. | Y no. of attr. | ←X+Y no. of attr.→

R

A	B	C
2	4	5
3	4	6
7	5	5

n distinct tuples

S

C	D
7	9
5	8

m distinct tuples

⇒ R X S : n x m distinct tuples

A	B	C	C	D
2	4	5	7	9
2	4	5	5	8
3	4	6	7	9
3	4	6	5	8
7	5	5	7	9
7	5	5	5	8

NOTE

Rel R with n tuples

Rel S with 0 tuples // empty

# of tuples in R X S = 0 tuples. // empty

A X B = { } = ϕ

JOINS :

① Natural Join ($\bowtie$)

② Conditional Join ($\bowtie_c$)

③ Outer Joins

(i) Left outer join ($\overset{L}{\bowtie}$ or $\overset{L}{\bowtie}_c$)

(ii) Right outer join ($\overset{R}{\bowtie}$ or $\overset{R}{\bowtie}_c$)

(iii) Full outer join ($\overset{F}{\bowtie}$ or $\overset{F}{\bowtie}_c$)

④ Equi Join ($\bowtie_=$)

Conditional join with equal condition

Natural Join :

$$R \bowtie S \equiv \pi_{\text{distinct attributes}} \left(\sigma_{p} (R \times S) \right)$$

p : equality of common attr. of R & S

E.g. R ⋈ S

R X S

A	B	C	C	D
2	4	5	7	9
2	4	5	5	8
3	4	6	7	9
3	4	6	5	8
7	5	5	7	9
7	5	5	5	8

E.g. $R \bowtie S \equiv \pi_{ABCD} \left(\sigma_{R.C=S.C} (R \times S) \right)$

R ⋈ S

A	B	C	D
2	4	5	8
7	5	5	8

E.g. $T_1(A\,B\,C)$ $T_2(B\,C\,D)$

$T_1 \bowtie T_2 = \pi_{ABCD} \left(\sigma_{T_1.B=T_2.B \,\wedge\, T_1.C=T_2.C} (T_1 \times T_2) \right)$

E.g. $T_1(A\,B)$ $T_2(C\,D)$

$T_1 \bowtie T_2 = T_1 \times T_2 = \pi_{ABCD} (\sigma_{\phi} (T_1 \times T_2))$

Natural join equal to cross product, if join condition empty.

$T_1 \bowtie T_2$

A	B	C	D
4	5	3	5
4	5	4	5
6	7	3	5
6	7	4	5

R(A B C D)

R1(A B) R2(C D)

$R_1 \bowtie R_2 = R_1 \times R_2 \supset R$ ∴ Lossy decomposition.

Conditional Join : ($\bowtie_c$)

$R \bowtie_c S \equiv \sigma_c (R \times S)$

e.g. $R \bowtie_{R.C<S.C} S \equiv \sigma_{R.C<S.C}(R \times S)$

$\equiv$

A	B	C	~~K~~	~~D~~
2	4	5	7	9
3	4	6	7	9
7	5	5	7	9

A	B	C	D	E
2	4	5	7	9
2	4	5	5	8
3	4	6	7	9
3	4	6	5	8
7	5	5	7	9
7	5	5	5	8

e.g. $R \bowtie_{R.A<5} S \equiv \sigma_{R.A<5}(R \times S)$

$\equiv$

A	B	C	C	D
2	4	5	7	9
2	4	5	5	8
3	4	6	7	9
3	4	6	5	8

$\sigma_{R.A<5}(R \bowtie S) \equiv \sigma_{R.C=S.C \wedge R.A<5}(R \times S)$

A	B	C	D
2	4	5	8

Outer Joins :

(i) Left outer join ($\bowtie$)

$R \bowtie S \equiv R \bowtie S$ tuples && tuples of R that failed join condition

R

A	B	C
4	5	7
4	6	3
4	7	5

S

C	D	E	F
7	4	3	5
7	7	5	6
6	9	8	9

$R \bowtie S \equiv$

A	B	C	D	E	F
4	5	7	4	3	5
4	5	7	7	5	6
4	6	3	NULL	N	N
4	7	5	N	N	N

(ii) Rightouter join ($\bowtie$)

$R \bowtie S \equiv$

A	B	C	D	E	F
4	5	7	4	3	5
4	5	7	7	5	6
N	N	6	9	8	9

R

A	B	C
4	5	7
4	6	3
4	7	5

S

C	D	E	F
4	4	3	5
7	7	5	6
6	9	8	9

(iii) Full Outer Join ($\bowtie$) :

$R \bowtie S \equiv (R \bowtie S) \cup (R \bowtie S)$

A	B	C	D	E	F
4	5	7	4	3	5
4	5	7	7	5	6
4	6	3	N	N	N
4	7	5	N	N	N
N	N	6	9	8	9

06/11/2017

$\Rightarrow R(A\ldots)\ S(B\ldots)$

Retrieve "A" values of Rel R those are more than ~~some~~ "B" values of S. (or any/atleast one)

Soln: R(A..) S(B..)

2, 4, 6 | 3, 5

$\Rightarrow R \times S$

A...	B...
2	3
2	5
4	3
4	5
6	3
6	5

$= \pi_{R.A}(\sigma_{R.A>S.B}(R \times S))$ — $R \bowtie_{R.A>S.B} S$

A
4
6

$\equiv \pi_{R.A}(R \bowtie_{R.A>S.B} S)$

$R \bowtie_c S$, $\sigma_c(R \times S)$, $R \bowtie S$ } used to retrieve records of relation "R" which are satisfied given condition with atleast one/any/some record of S.

$\Rightarrow R(A\ldots)\ S(B\ldots)$

Retrieve "A" values of Rel R those are more than every "B" of Rel S.

$\neg(> All) \equiv \leq Som$

Soln: R(B..) S(B..)

2, 4, 6 | 3, 5

$\equiv \pi_A(R) - \pi_A(R \bowtie_{R.A \leq S.B} S)$

* A values of R those are more than every B of S

$\equiv$

{All A values of R} − {A values of R those are less than equal to some B of S}

* ρ : Rename :

→ used to rename table name or attributes.

Stud (Sid Sname age)

① ρ (Temp, stud) : Temp(Sid Sname age)

② $\rho_{I,N,A}$ (Stud) : stud (I, N, A)

③ $\rho_{I\to I,\ 3\to A}$ (Stud : stud (I, Cname, A)

Eg: Stud (Sid Sname age)

Retrieve sids whose age more than age of some stud. (M_c)

Stud

Sid	Sname	age
S_1		10
S_2		20
S_3		30
S_4		40

×

Stud

Sid	Sname	age
S_1		10
S_2		20
S_3		30
S_4		40

$$\pi_{Stud.Sid}\left(Stud \underset{Stud.age > Temp.age}{\bowtie} \rho(Temp, Stud)\right)$$

(or) $\pi_{age}\left(Stud \underset{age > A}{\bowtie} \rho_{I,N,A}(Stud)\right)$

② Retrieve Sid's whose age (≥ every age) maximum.

[Sid's of all students] − [Sids of students whose age less than some student's age]

$$\pi_{sid}(Stud) - \pi_{sid}\left(Stud \underset{age < A}{\bowtie} \rho_{I,N,A}(Stud)\right)$$

Sid
S_1
S_2
S_3
S_4

Stud

Sid	age
S_1	10
S_2	20
S_3	30
S_4	40

<

Stud

Sid	age
S_1	10
S_2	20
S_3	30
S_4	40

Retrieve Sids whose age minimum.

$$\pi_{Sid}(Stud) - \pi_{Sid}\left(Stud \underset{age > A}{\bowtie} \rho_{I,N,A}(Stud)\right)$$

(or)

$$\pi_{Sid}(Stud - \pi_{A}(Stud \underset{age < A}{\bowtie} \rho_{I,N,A}(Stud)))$$

⇒ Retrieve Sids whose age maximum for each name.

Stud

Sid	Sname	age
S_1	A	10
S_2	A	20
S_3	B	30
S_4	B	40

Stud

Sid	Sname	age
S_1	A	10
S_2	A	20
S_3	B	30
S_4	B	40

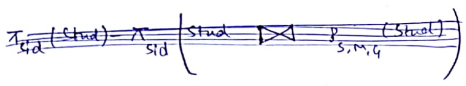
{Sids whose age max for each name} ≡ {Sids of all Stud} − {Sids whose age < some Stud age of same name} (M_c)

Sid
S_2
S_4

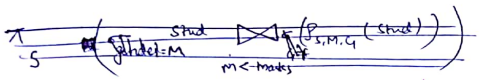
$$= \pi_{Sid}(Stud) - \pi_{Sid}\left(Stud \underset{SName = N \wedge age < N}{\bowtie} \rho_{I,N,A}(Stud)\right)$$

⇒ Stud (Sid, marks, gender)

① Retrieve sid's of female students who scored less marks than some male students.



Sid Marks gender
S1
S2
S3
S4



π_I (Stud $\bowtie_{\text{gender}=M,\ G_1=F \wedge M<\text{marks}}$ $\rho_{I,M,G}$ (Stud)) | π_{Sid} (Stud $\bowtie_{M>\text{marks}_1 \wedge \text{Gender}_1=\text{"female"} \wedge G=\text{"male"}}$ $\rho_{I,M,G}$ (Stud))

② Retrieve Sid's of female students who scored less marks than every male student.

$\pi_{Sid}(\sigma_{\text{gender}=\text{female}}(\text{Stud})) - \pi_{Sid}(\text{Stud} \bowtie_{\text{gender}=\text{"female"} \wedge G=\text{male} \wedge \text{marks} \geq M} \rho_{I,M,G}(\text{Stud}))$

SET OPERATORS:

∪ : Union
− : Minus (Set difference) } Basic operators
∩ : Intersection } Derived operator

⇒ To apply set operations, relations must be union compatible.

⇒ R & S union compatible iff:

① # of attr. of R ≡ # of attr. of S
and ② domain of each attr. of R must be same as that of S.

E.g. R(A1, A2) S(B1, B2)

E.g. X $\pi_{Sid}(\ldots) \cup \pi_{Sid\ Sname}(\ldots)$ — $\{S_2, S_3\}$ — $\{(S_1, A), (S_2, B)\}$

X $\pi_{Sid\ Sname}(\ldots) \cap \pi_{Sid\ age}(\ldots)$

✓ $\pi_{Sid\ Sname}(\ldots) \cap \pi_{I,N}(\ldots)$
(I: set of stud. Ids; N: set of stud. names)

⇒ R∪S, R∩S, R−S result always treated as distinct tuples.

R

A
2
2
2
3
3
4

S

B
2
2
3
3
5
5

⇒ R∩S

A
2
3

distinct tuples.

⇒ R∪S, R∩S, R−S, expressions resulted in schema same as schema of left side relation, i.e., "R" here.

E.g. R

A	B	C
2	4	6
3	5	7
4	6	5

S

D	E	F
3	5	7
4	6	5
2	4	8

* U : Union:

$R \cup S \equiv \{X / X \in R \text{ (V) of } X \in S\}$

R ∪ S =

A	B	C
2	4	6
3	5	7
4	6	5
2	4	8

* "—" MINUS :

$R - S \equiv \{x \mid x \in R \wedge x \notin S\}$ (But not)

(Records of R but not of S) ≡ Records of only R

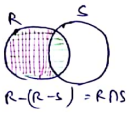
$R - S \equiv$

A	B	C
2	4	6

* "∩" Intersection :

$R \cap S \equiv R - (R - S)$

$R \cap S \equiv \{x \mid x \in R \wedge x \in S\}$ (∧ AND)



R − (R − S) = R ∩ S

$R \cap S \equiv$

A	B	C
2	5	7
4	6	5

If T_1 & T_2 same attributes

T_1 (A B)

A	B
2	4
6	8

T_2 (A B)

A	B
2	4
6	5

$T_1 \cap T_2 \equiv T_1 \bowtie T_2$

A	B
2	4

$T_1 \cup T_2 \equiv T_1 \bowtie T_2$

$T_1 \cap T_2 = T_1 - (T_1 - T_2)$

A	B
2	4
6	8
6	5

DIVISION (/ or ÷) : * *

Enroll (Sid Cid)

Sid	Cid
S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S3	C1

course (Cid - -)

Cid
C1
C2
C3

All courses.

Retrieve Sid's enrolled in all courses.

$\pi_{Sid,Cid}(Enroll) / \pi_{Cid}(course) \equiv$

Sid
S1

Sid	Cid
S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S3	C1

Cid
C1
C2
C3

* Expansion of "/" :

→ Sid's enrolled in every course

$\pi_{Sid,Cid}(Enroll) / \pi_{Cid}(course)$

Step 1 : Sid's not enrolled every course [Sid's enrolled proper subset of all courses]

$\pi_{sid}\left(\pi_{sid}(Enroll) \times \pi_{Cid}(course) - \pi_{Sid,Cid}(Enroll) \right)$

$\pi_{sid}(\pi_{sid}(Enroll) \times \pi_{Cid}(course))$: Every student enrolled in every course [All possible pairs]

$\pi_{Sid,Cid}(Enroll)$: Actual pairs

( ) : disqualified records of division

Sid	Cid
S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S2	C3
S3	C1
S3	C2
S3	C3

−

Sid	Cid
S1	C1
S1	C2
S1	C3
S2	C1
S2	C2
S3	C1

⇒

Sid	Cid
S2	C3
S3	C2
S3	C3

π_{sid} ⇒

Sid
S2
S3

S2, S3 : sid's not enrolled in every course

Step 2 : [Sid enrolled in every course] ≡ [Sid's enrolled some course] − [Sid's not enrolled every course]

$\pi_{Sid,Cid}(Enroll) / \pi_{Cid}(course) \equiv \pi_{Sid}(Enroll) - \pi_{Sid}\left(\pi_{Sid}(Enroll) \times \pi_{Cid}(course) - \pi_{Sid,Cid}(Enroll) \right)$

$$\pi_{AB}(R) \,/\, \pi_{B}(S) \equiv \pi_{A}(R) - \pi_{A}\left(\pi_{A}(R) \times \pi_{B}(S) - \pi_{AB}(R)\right)$$

"A" value of relation R which are paired by every "B" value of S.

Basic operators used to express division $\pi, \times, -$

⇒ Stud (Sid Sname age) [All students]
Course (Cid Cname Instructor) [All course info]
Enroll (Sid Cid fee) [Sid's enrolled course] Mapping of student & course

① Retrieve Sid's enrolled some course taught by Korth.

$$\pi_{Sid}\left(\text{Enroll} \bowtie_{\substack{\text{Instructor}\\=\text{Korth}\\ \wedge \text{Enroll.Cid}=\text{Course.Cid}}} \text{Course}\right)$$

Sid	Cid	fee
s_1	c_1	
s_2	c_2	
s_2	c_1	
s_3	c_2	
s_3	c_2	

Cid	Cname	Ins.
c_1		Korth
c_2		Korth
c_1		...

$\equiv \pi_{Sid}\left(\sigma_{\substack{\text{Enroll.Cid}=\text{Course.Cid}\\ \wedge \text{Inst}=\text{Korth}}} (\text{Enroll} \times \text{Course})\right)$ 2×(7×4) = 56 comp.

$\equiv \pi_{Sid}\left(\sigma_{\text{Enroll.Cid}=\text{Course.Cid}} (\text{Enroll}) \times \sigma_{\text{Instructor}=\text{KORTH}} (\text{Course})\right)$ 4+(7×2) = 18 comp.

=

Enroll

Sid	Cid
s_1	c_1
s_1	c_2
s_2	c_2
s_3	c_3
s_3	c_4
s_1	c_3
s_1	c_3

Course

Cid	Inst
c_1	Korth
c_2	Korth
c_3	Navathe
c_4	Navathe

$\equiv \pi_{Sid}\left(\sigma_{\text{Inst}=\text{KORTH}} (\text{Enroll} \bowtie \text{Course})\right)$

$\equiv \pi_{Sid}\left(\text{Enroll} \bowtie \left(\sigma_{\text{Inst}=\text{KORTH}} (\text{Course})\right)\right)$

② Retrieve Sid's enrolled every course taught by Korth.

$\equiv \pi_{Sid\,Cid}(\text{Enroll}) \,/\, \pi_{Cid}\left(\sigma_{\text{Inst.}=\text{KORTH}} (\text{Course})\right)$

$\equiv \pi_{Sid}(\text{Enroll}) - \pi_{Sid}\left(\pi_{Sid}(\text{Enroll}) \times \pi_{Cid}\left(\sigma_{\text{Inst}=\text{KORTH}} (\text{course})\right) - \pi_{Sid\,Cid}(\text{Enroll})\right)$

- $\pi_{Sid}(\text{Enroll})$: Sids not enrolled every KORTH course
- $\pi_{Sid}(\text{Enroll}) \times \pi_{Cid}(\sigma_{\text{Inst}=\text{KORTH}}(\text{course}))$: every sid enrolled every KORTH course
- $\pi_{Sid\,Cid}(\text{Enroll})$

③ Retrieve Cid's enrolled by some stud whose age is more than 20.

Enroll Sid Cid Stud Sid age

$\pi_{Cid}\left(\sigma_{age>20} (\text{Enroll} \bowtie \text{Stud})\right)$

$\equiv \pi_{Cid}\left(\text{Enroll} \bowtie \sigma_{age>20} (\text{Stud})\right)$

④ Retrieve Cid's enrolled by every stud whose age more than 20.

$\pi_{Sid\,Cid}(\text{Enroll}) \,/\, \pi_{Sid}\left(\sigma_{age>20} (\text{Stud})\right)$

or

Cid Sid

Cid

***

① Pair with every ... ⇒ "/"

② > every - - - ⇒ All - {≤ some} value

⇒ Retrieve Sid's enrolled some course taught by Korth or Navathe.

π_{Sid} (Enroll ⋈ $\sigma_{Instructor = "KORTH" \vee Instructor = "Navathe"}$ (Course))

≡ π_{Sid} (Enroll ⋈ $\sigma_{Instr = "Korth"}$ (Course)) ∪ π_{Sid} (Enroll ⋈ $\sigma_{Inst = "Navathe"}$ (Course))

⇒ Retrieve Sid's enrolled some course taught by Korth and Navathe.

π_{Sid} (Enroll ⋈ $\sigma_{Inst = "Korth" \wedge Instr = "Navathe"}$) ∩ π_{Sid} (Enroll ⋈ $\sigma_{Inst = "Korth" \wedge Inst = "Navathe"}$ (Course)) ✗

≡ π_{Sid} (Enroll ⋈ $\sigma_{Inst = "Korth"}$ (Course)) ∩ π_{Sid} (Enroll ⋈ $\sigma_{Inst = "Navathe"}$ (Course))

⇒ Retrieve Sid's enrolled every course taught by Korth & Navathe.

Course

Cid	Inst
C1	Korth
C2	Korth
C3	Navathe
C4	Navathe

Enroll

Sid	Cid
S1	C1
S1	C2
S2	C3
S2	C4
S3	C1
S3	C2
S3	C3
S3	C4
S4	C1
S4	C3

Q1. $\pi_{Sid,Cid}$ (Enroll) / π_{Cid} ($\sigma_{Inst = Korth \vee Inst = Navathe}$ (Course)) ⇒ {S1, S3}

Q2. { $\pi_{Sid,Cid}$ (Enroll) / π_{Cid} ($\sigma_{Inst = Korth}$ (Course)) } ∩ { $\pi_{Sid,Cid}$ (Enroll) / π_{Cid} ($\sigma_{Inst = Navathe}$ (Course)) }

Which is true?

(a) Only Q1 correct
(b) only Q2 correct
(c) both Q1, Q2 correct
(d) both Q1, Q2 incorrect

⇒ Retrieve Sid's enrolled only Korth course.

~~$\pi_{Sid,Cid}$ (Enroll) / π_{Cid} ($\sigma_{Inst = Korth}$)~~

≡ { Sid's enrolled some course } − { Sid's enrolled some non-Korth course }

≡ π_{Sid} (Enroll) $\overset{but\ not}{-}$ π_{Sid} (Enroll ⋈ $\sigma_{Inst \neq Korth}$ (Course))

S1 S2 S3 S4 − S1 S2 S3 = S4

Courses

Enroll

S1	C1
S1	C2
S2	C2
S3	C1
S3	C3
S4	C1

Course

C1	Korth
C2	Navathe
C3	Ullman

"−" : only but not

#Self Cross Product:

σ_C (R×R) : To compare every possible two records of R.

σ_C (R×R×R) : To compare every possible three records of R.

⇒ Retrieve Sids of enrolled at least two courses.

Enroll

Sid	Cid	fee
S1	C1	
S1	C2	
S1	C3	
S3	C1	
S3	C2	
S2	C1	

×

Sid	Cid	fee
S1	C1	
S1	C2	
S1	C3	
S3	C1	
S3	C2	
S2	C1	

$t_1.sid = t_2.sid \wedge t_1.cid \neq t_2.cid$

$\pi_{Enroll.Sid}$ ($\sigma_{Enroll.sid = T.sid \wedge Enroll.cid \neq T.cid}$ (Enroll × ρ(T, Enroll))) ≡ π_{Sid} (Enroll ⋈ ρ(Enroll) Sid=S, C,F ∧ Cid≠C)

⇒ Retrieve Sid's enrolled atleast 3 courses.

$$\pi_{Sid}\left(\sigma_{\substack{Sid=S1=S \\ \wedge S1=S2 \\ \wedge Cid \neq C1 \\ \wedge C1 \neq C2 \\ \wedge Cid \neq C2}}\left(Enroll \times \rho_{S1,C1,F1}^{(Enroll)} \times \rho_{S2,C2,F2}^{(Enroll)}\right)\right)$$

$$\equiv \pi_{T1.sid}\left(\sigma_{\substack{T1.Sid=T2.Sid \\ \wedge T2.Sid=T3.Sid \\ \wedge T1.Cid \neq T2.Cid \wedge T2.Cid \neq T3.Cid \\ \wedge T1.Cid \neq T3.Cid}}\left(\begin{matrix}\rho(T1, Enroll) \\ \rho(T2, Enroll) \\ \rho(T3, Enroll) \\ (T1 \times T2 \times T3)\end{matrix}\right)\right)$$

⇒ Retrieve Sid's enrolled only one course.
[Sid's enrolled atleast one course but not enrolled atleast two courses]

$$\pi_{Sid}(Enroll) - \pi_{Sid}\left(Enroll \underset{\substack{Sid=S \\ \wedge Cid \neq C}}{\bowtie} \rho_{S,C,F}(Enroll)\right)$$

⇒ Retrieve Sid's enrolled ~~only~~ exactly two courses.

[Sid's enrolled atleast two courses] − [Sid's enrolled at least three courses]

$$\equiv \pi_{Sid}\left(Enroll \underset{\substack{Sid=S \\ \wedge Cid \neq C}}{\bowtie} \rho_{S,C,F}(Enroll)\right) - \pi_{Sid}\left(\sigma_{\substack{Sid=S1 \\ \wedge S1=S2 \\ \wedge Cid \neq C1 \\ \wedge C1 \neq C2 \\ \wedge Cid \neq C2}}\left(Enroll \times \rho_{S1,C1,F1}^{(Enroll)} \times \rho_{S2,C2,F2}^{(Enroll)}\right)\right)$$

⇒ Retrieve Sid's enrolled atmost one course.

$$\pi_{Sid}(Stud) - \pi_{Sid}\left(Enroll \underset{\substack{Sid=S \\ \wedge Cid \neq C}}{\bowtie} \rho_{S,C,F}(Enroll)\right)$$

⇒ Sid's enrolled atmost two courses.

[All students $\pi_{Sid}(Stud)$] − [Sid's enrolled atleast three courses]

⇒ Retrieve Sid's who paid maximum fee.

$$\pi_{Sid}\left(\pi_{Sid,Cid}^{(Enroll)} - \pi_{Sid,Cid}\left(Enroll \underset{Fee \leq F}{\bowtie} \rho_{S,C,F}^{(Enroll)}\right)\right)$$

Sid	Cid	fee
S1	C1	80
S1	C2	70
S1	C3	50
S3	C1	50
S3	C2	60
S2	C1	80

Sid	Cid	fee
S1	C1	80
S1	C2	70
S1	C3	50
S3	C1	50
S3	C2	60
S2	C1	80

⇒ Retrieve Sid's who paid max fee for each course.

$$\pi_{Sid}\left(\pi_{Sid,Cid}^{(Enroll)} - \pi_{Sid,Cid}\left(Enroll \underset{\substack{Fee \leq F \\ \wedge Cid=C}}{\bowtie} \rho_{S,C,F}(Enroll)\right)\right)$$

{All stud} − Sid's who paid < some stud for same course

Sid	Cid	fee
S1	C1	50
S2	C1	80
S3	C1	70
S4	C2	70
S5	C2	60

Sid	Cid	fee
S1	C1	50
S2	C1	80
S3	C1	70
S4	C2	70
S5	C2	60

⇒

⇒ Stud (Sid Sname) with 200 tuples.
Enroll (Sid Cid) with 100 tuples.
How many maximum & minimum records in result of :
Stud ⋈ Enroll?

(a) (200,100)
(b) (200,0)
(c) (100,100)
(d) (20000,0)

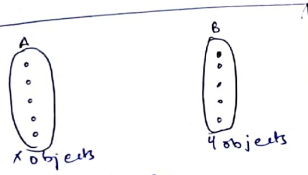
Sid	Sname
S1	A
S2	B
S3	C
S4	
S5	
200	

Sid	Cid
S2	C1
S3	C2
S3	C2
100	

① Max tuples Stud ⋈ Enroll = 100

② Min tuples Stud ⋈ Enroll = 100 FK exists
= 0 No FK exists.

S1 S2
S1 S3
~~S2 S2~~
S2 S3
~~S3 S3~~
S5 S3



① 1 : M ⇒ Y pairs
② M : 1 ⇒ X pairs
③ 1 : 1 ⇒ min(X,Y) pairs
④ M : N ⇒ X.Y pairs.

Q. Rel R(A B C) with "n" tuples
S(B D E) with "m" tuples. [Assume no NULLS]
Max tuples in R ⋈ S = ~~m*n~~ n
Min tuples in R ⋈ S = n with FK
0 without FK

③ R(A B C) with n tuples
S(A D E) with m tuples
Max tuples in R ⋈ S = min(m,n) tuples
Min tuples in R ⋈ S = min(m,n) with FK
0 w/o FK

Q④ R(A B C) with n tuples
S(D B E) with m tuples
Max tuples in R ⋈ S = m*n
Min tuples in R ⋈ S = 0

Q. R(A B C) S(B D F) with n & m tuples &&
{A → B, B → C, C → D, D → F} are FDs.
Max tuples in the result of R ⋈ S? n tuples.

#SQL: Structured Query Language

Sub languages of SQL:

Data Definition Lang. (DDL)	Data Manipulation Lang. (DML)	Data Control Lang. (DCL)
→ Used to define or modification of structure of DB table & integrity constraints. [PK, AK, FK, check, etc.]	→ Used to modify data records but not allowed to modify structure of Tables. (OR) used to access data from DB Tables.	→ data control for consistency: • lock() • shared() • exclusivelock() • commit() • Rollback etc. (Used for transaction & Concurrency Control)
→ DDL Commands: • CREATE TABLE <tname> • DROP TABLE <tname> • ALTER TABLE <tname> • Add/remove attributes • Modify integrity constraints. • CREATE INDEX ..... Etc.	→ DML Commands: • INSERT INTO <tname>... • DELETE FROM <tname>.. • UPDATE <tname> SET... • SELECT * FROM tables where condition;	→ data control for security: • grant • revoke, etc.
→ Not modifies frequently & control of DDL under DBA.		

SQL Query vs. RA Expression:

SQL: SELECT DISTINCT $A_1, A_2 \ldots A_n$ → Projection Operation (π)
FROM $R_1, R_2 \ldots R_m$ → Cross Product ($\times$)
WHERE P; → Selection operation (σ_P)

RA: $\pi_{A_1 A_2 \ldots A_n}\left(\sigma_P (R_1 \times R_2 \times \ldots \times R_m)\right)$

E.g: Sid's enrolled some course taught by Korth.

$\pi_{sid}\left(\sigma_{\substack{Enroll.cid = Course.Cid \\ \wedge Instr = Korth}} (Enroll \times Course)\right)$

```
SELECT DISTINCT sid
FROM Enroll AS T1, Course AS T2      → Rename operator(?)
WHERE T1.cid = T2.cid and
      T2.Inst = "Korth";
```

Sid	Cid
S1	C1
S1	C2
S2	C2

=

Cid	Instructor
C1	Korth
C2	Korth

⇒

Sid	Cid	Cid	Instructor
S1	C1	C1	Korth
S1	C2	C2	Korth
S2	C2	C2	Korth

Basic SQL clauses:

⑤ SELECT [DISTINCT] ⑥ $A_1, A_2 \ldots A_n$
① FROM $R_1, R_2, R_3 \ldots R_n$
② [WHERE P] → condition for each record.
③ [GROUP BY (Attributes)
④ [HAVING condition]]
⑦ [ORDER BY (Attributes) [DESC]] ;

Execution Flow:

FROM : Cross Product (×)
↓
WHERE : selection operator (σ)
↓
GROUP BY
↓
HAVING
↓
SELECT
↓
DISTINCT → Projection operator (π)
↓
ORDER BY

Aggregate functions of SQL:

1. COUNT( ) → "NULL" values not considered for aggregation.
2. SUM( ) → Aggregate function computes aggregation of NON NULL values.
3. AVG( )
4. MIN( )
5. MAX( )

Create table R
(A int,
B int
);

R

Row-ID	A	B
1	6	8
2	8	7
3	NULL	10
4	8	NULL
5	5	5
6	NULL	NULL
7	5	6

*** Write SQL Query to delete duplicate records. ****

* COUNT( ):

- count(*) : # of records of table.
- count(A) : # of NON NULL values of attr. A.
- count(DISTINCT A) : # of distinct NON NULLS of attr "A"

Eg: SELECT COUNT(*) AS A1, COUNT(A) AS A2,
COUNT(DISTINCT A) AS A3,
FROM R;

o/p

A1	A2	A3
7	5	3

~~DISTINCT~~

~~SUM~~ E.g.

1. SELECT SUM(A) AS A1, SUM(DISTINCT A) AS A2,
AVG(A) AS A3, AVG(DISTINCT A) AS A4,
MIN(A) AS A5, MAX(A) AS A6,
FROM R;

o/p

A1	A2	A3	A4	A5	A6
32	19	6.4	6.33	5	8
		SUM(A)/COUNT(A) = 32/5	SUM(DISTINCT A)/COUNT(DISTINCT A) = 19/3		

2. SELECT AVG(A), B
FROM R;

Error: Incorrect usage of aggregation.

AVG(A)	B

* If aggregate function used in SELECT clause, then, not allowed to select any attribute if GROUP BY clause doesn't exists.

3. SELECT A
FROM R
WHERE A = max(A);

(for each record computes aggregation)

6 max(6) ✓
8 max(8) ✓

o/p

A
6
8
8
5
5

Group By Clause:

→ Used to group data records based on specified attribute values.

R

A	B	C
a1	b1	80
NULL	b2	70
a3	b3	50
NULL	b2	90
a1	b1	60
a3	b3	60
a1	b5	NULL

Group by (A) →

R

A	B	C
a1	b1	80
a1	b1	60
a1	b5	NULL
NULL	b2	70
NULL	b2	90
a3	b3	50
a3	b3	60

Group by (AB) →
OLAP application (Data Mining Queries)

R

A	B	C
a1	b1	80
a1	b1	60
a1	b5	NULL
NULL	b2	70
NULL	b2	90
a3	b3	50
a3	b3	60

* SQL Standard:

→ Every attribute of group by must be in select clause and allowed use aggregate fun in SELECT ⇔ clause [Aggregation of SELECT clause computes aggregation of each group]
But not allowed to select attribute not in group by

① 3. SELECT A, AVG(C)
1. FROM R
2. GROUP BY A;

A	avg(c)
a1	70
NULL	80
a3	55

② SELECT A
FROM R
GROUP BY (A);

A
a1
NULL
a3

③ SELECT B
FROM R
GROUP BY (A);

ERROR (Not Allowed)

④ SELECT A, B
FROM R
GROUP BY (A);

ERROR (Not Allowed)

⑤ SELECT AVG(C)
FROM R
GROUP BY (A);

SQL Standard Not Allowed

Oracle SQL

Avg(c)
70
80
55

⇓

SELECT C1
FROM (SELECT A, Avg(C) C1
FROM R
GROUP BY A);

⑥ SELECT A
FROM R
GROUP BY (A,B)

SQL Standard Not Allowed

Oracle SQL

A
a1
a1
NULL
a3

Q Which is not allowed:

(a) Select A, Avg(B)
FROM R; most wrong

(b) Select Avg(c)
FROM R
Group by A;

(c) Select A
from R
Group by (A,B);

(d) Select A, count(*)
FROM R
Group by A;

Q. Which is not allowed?

(a) Select A
from R
Group by (A, B);

(b) Select A
from R
Group by A;

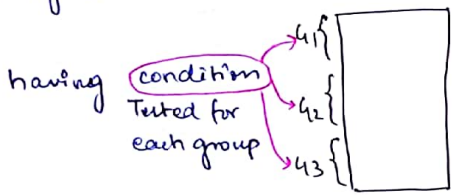
(c) Select A, count(*)
from R
Group by (A);

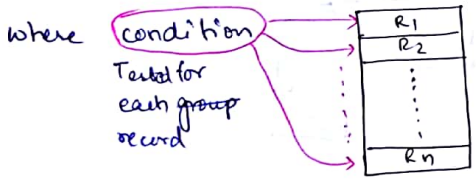
(d) Select A, B
From R
Group by (A, B);

Having Clause:

→ Used to select group based on specified condition.

→ Having clause condition tested for each group.





SQL Standard:

→ Having clause allowed to use only if group by clause exists.

→ Condition of having clause must be over aggregation or over function some(), every(), but not allowed direct attribute.

E.g. SELECT A
FROM R
Group by A ERROR
having C > 70;

SELECT A
FROM R
Group by A
having AVG(C) > 60;

A
a1
NULL

E.g. 1. SELECT A
1. FROM R
2. Group by A
3. having some(C) > 70;

A
a1
NULL

Select A
from R
Group By A
having ~~so~~ every(C) ≥ 70;

A
NULL

* If Group By based on oneInt type attribute, then, having clause condition can use group by attribute.

E.g. R (A B)

× { 2 2
✓ { 4 4 4
✓ { 8 8

Select A
FROM R
Group by A
having A > 3;
allowed

Order By Clause:

→ Used to sort data records of Query result based on specified attribute [Ascending / Descending order].

E.g.
5. Select X, Y, Z
1. FROM ...
2. where ...
3. Group by ...
4. having ...
6. Order By (X, Y);

o/p

X	Y	Z
2	6	10
2	8	8
3	4	1
3	4	0
NULL	5	3
NULL	6	4

order By (X, Y)

Nested Query: ***

① Nested Query (without corelation)
② Co-related nested query.

Nested Query:

Select Attr
From Tables
WHERE condition
GROUP by Attr.
having condition
order by attr.

(Inner Query)

Nested Query without co-relation:

Inner Query independent
&
Outer Query uses result of inner query.

E.g SELECT T1.sid
FROM stud T1
WHERE MARKS = (Select max(marks) FROM stud T2);

80 ← scalar value

① Bottom Query
② Top Query.

* Execution Flow: Bottom Query to Top Query.

⇒ Inner Query computes only once.

Co-related Nested Query: ** [For most complex applications/Queries]

→ Inner Query uses attributes defined in outer Query & outer query uses result of inner query.

→ Used in WHERE Clause & HAVING Clause of outer query, used to ~~co~~ co-relate with inner query.

→ SELECT A
① FROM R
③ WHERE (Select count(*)
② FROM S
③ WHERE R.A < S.B) ≤ 2

R	A
✗	10
✗	20
✓	30
✓	40
✓	50

S	B
	5
	15
	25
	35
	45

⇒

A
30
40
50

~~→ If correlation where clause for each record of outer query from clause from clause~~

→ If co-relation in WHERE clause for each record of outer query FROM clause computes inner query.

* Execution Flow: (Top-Bottom-Top) in loop.

* Corelation in HAVING clause:

SELECT A
① FROM R AS T1
② GROUP BY A
③④ HAVING AVG(C) > (SELECT AVG(C)
⑤ FROM S AS T2
⑥ WHERE T2.A = T1.A);

R	A	C
✗	a1	40
✗	a1	70
✓	a3	80
✓	a3	90

S	A	C
	a1	80
	a1	70
	a3	60
	a3	50

O/P:

A
a1

→ If co-relation in HAVING clause, for each group of outer query, re-computes inner query.

Stud

Sid	Branch	Marks
S1	CS	70
S2	IT	80
S3	CS	90
S4	IT	95
S5	EC	85

Eg: SELECT Sid
FROM Stud
WHERE Marks = (select Marks from Stud where branch = 'CS');

ERROR: S1 CS 70 = ?

marks
70
90
85

Direct comparison not allowed

set of values

SQL functions used for nested queries:

⇒ IN / NOT IN
⇒ ANY
⇒ ALL
⇒ EXISTS / NOT EXISTS

Best suitable for Nested queries with no co-relation

- Used to compare set of values of inner query result with outer query record value in where / having clause.

→ Best suitable for co-related nested query.

① IN : Membership Test

x IN [set of ~~values~~ Y values of Inner Query result]

True if X value, member of set of Y values.

X IN {2, 3, 5, 6}
True if X ∈ set

Eg: R(A B) S(C D)

$\pi_{A,B}(R \bowtie_{R.B=S.C} S)$ ≡ Select * FROM R where B IN (Select C FROM S)

IN = =ANY = equal to atleast one = some = any

Same as equijoin

⇒ T_1(A B C) T_2 (D E F)

$\pi_{ABC}(T_1 \bowtie_{T_1.B=T_2.D \wedge T_1.C=T_2.E} T_2)$ ≡ Select * FROM T_1 where (B, C) IN (Select D, E FROM T_2)

=ANY

⇒ $\pi_{A,B}(R \bowtie_{R.B>S.C} S)$ • Not possible to implement using IN function.

ANY & ALL Functions:

→ X op ANY [set of Y values]

op: any comparison operation (<, <=, >, >=, =, <>, ≠)

TRUE if X satisfies given comparison with atleast one or some values of set Y.

→ X op ALL [set of Y values]

TRUE if X satisfies given comparison (op) with every value of set Y.

Eg: Select *
FROM R
where A =ANY IN (Select B from S)

[=ANY] ⇒ IN

=ANY function can be replaced by IN function.

<>ALL fn

<>ALL function can be replaced by NOT IN function

R

A
10
30
40

S

B
20
30
40

Q

R

A	B
4	2
6	4
7	5

S

C	D
4	3
7	6
9	8

(a) 0 (b) 1 (c) 2 (d) 3

# of tuples in result of SQL Query:

① SELECT *
FROM R
where B > ANY (Select C FROM S where D > 10);

C
Empty

Scanned by CamScanner

① Select *
FROM R
WHERE B > ALL(Select C
FROM S
where D>10);

(a) 1 (b) 2 (c) ✓ 3 (d) 0

X = { } //empty set
∃x (X>10) : False
∀x (X>10) : True

∃x P(x): True for (some) x (at least one) P(x) is true
False for every x P(x) is false
∀x P(x): True for every x P(x) is true
False for some x P(x) is false

* X IN {Empty Set} : FALSE
X NOT IN {Empty Set} : TRUE

EXISTS | NOT EXISTS :

EXISTS: function return TRUE if inner query result Not empty otherwise false.

R(A...) S(B...)
✗ 20 25
✓ 30 35
✓ 40 45

Select A
FROM R
where EXISTS (Select *
FROM S
where R.A > S.B);

F T T
{}, {25}, {25,35}

TRUE: Some record or atleast one record of S should satisfy R.A > S.B

~~20~~ 30

o/p

A
30
40

Q. R

A	B
4	2
6	4
7	5

S

C	D
4	3
7	6
9	8

# of tuples in result of SQL query ?

① Select *
FROM R
where EXISTS (Select C
FROM S
where D>10 and R.B < S.C);

{ } empty

(a) ✓ 0 (b) 1 (c) 2 (d) 3

② Select *
FROM R
where EXISTS (Select count(*)
FROM S
where D>10 and R.B<S.C);

Count
0

} Not empty

(a) 1 (b) 2 (c) ✓ 3 (d) 0

Q. R(A...) S(B....)
Retrieve "A" value of R which are more than some "B" of S.

RA: $\pi_A \left(R \underset{R.A > S.B}{\bowtie} S \right)$

SQL: ① select distinct A
FROM R, S
where R.A > S.B;

② select A
FROM R

Emp (Eid , dno , sal , gender)

① ⇒ Retrieve Eid's who gets highest salary.

$\pi_{Eid} - \pi_{Eid}\left(Emp \underset{sal<s}{\bowtie} \rho_{I,D,S,G}(Emp)\right)$

Eid dno sal gender.

— : EXCEPT

SQL: ① SELECT Eid → All Eids
FROM Emp
EXCEPT
SELECT T1.Eid
FROM Emp T1, Emp T2
WHERE T1.sal < T2.sal; → Emp. ids who don't get highest salaries

② SELECT Eid
FROM Emp
Where Sal = (Select MAX(sal)
FROM Emp);

② ⇒ Retrieve Empid's who gets second highest salary.

$\rho\left(Temp, \pi_{Eidsid}\left(Emp \underset{sal<s}{\bowtie} \rho_{I,D,S,G}(Emp)\right)\right)$

Eidsal of all emp except highest sal.

$\pi_{Eid}(Temp) - \pi_{Eid}\left(Temp \underset{Sal<s}{\bowtie} \rho_{I,D,S,G}(Temp)\right)$

~~SQL : ① WITH~~

SQL : ① WITH Temp (Eid, Sal) as
(SELECT DISTINCT T1.eid, T1.sal
FROM Emp T1, Emp T2
where T1.sal < T2.sal)
SELECT Eid
FROM Temp
EXCEPT
Select T1.eid
FROM Temp T1, Temp T2
where T1.sal < T2.sal;

Join Query

② Select ~~distinct~~ eid
FROM Emp
where sal = (select max(sal) → 2nd max
FROM Emp
where sal < (select max(sal) → 1st max
FROM Emp))

Nested Query

③ Emp (Eid sal)

E1	80
E2	80
E3	70
E4	70
E5	60
E6	50

Emp (Eid sal)

E1	80
E2	80
E3	70
E4	70
E5	60
E6	50

Correlated nested query

80	80
~~70~~	~~70~~
60	60
50	50

Select T1.eid
FROM Emp T1
WHERE (SELECT COUNT (DISTINCT T2.Sal)
FROM Emp T2
where T1.Sal < T2.Sal) = 1

By using ② :
kth highest
- k-1 levels of nested query
- k-1 instances of Emp.

By using ③ :
kth highest :
- where (SELECT COUNT (DISTINCT T2.sal)
FROM Emp T2
where T1.sal < T2.sal) = k-1
- 2 levels of emptable.

② ⇒ ~~Retrieve department dno which consist atleast three employees.~~

write queries

- Eid's who get top 3 highest salaries < 3
- Eid's who get top 3 least salaries ~~$>$~~ $T_1.sal > T_2.sal$
- Eid's who get 3rd highest salary $= 2$
- Eid's who get 3rd least salary $(T_1.sal > T_2.sal) = 2$

③ ⇒ Retrieve ~~department~~ dno which consists atleast three employees

Relational algebra solution

① SELECT DISTINCT $T_1.dno$
FROM Emp T_1, Emp T_2, Emp T_3
where $T_1.dno = T_2.dno$ and $T_2.dno = T_3.dno$ and
$T_1.eid <> T_2.eid$ and $T_2.eid <> T_3.eid$ and
$T_1.eid <> T_3.eid$;

② SELECT dno
FROM Emp
Group by dno
HAVING count(*) ≥ 3 ;

⇔

Temp

dno	cant(*)
d_1	3
d_2	2
d_3	4

eid	dno	
e_1	d_1	
e_2	d_1	
e_3	d_1	≥ 3 ✓
e_4	d_2	
e_5	d_2	≥ 3 ✗
e_6	d_3	
e_7	d_3	
e_8	d_3	≥ 3 ✓
e_9	d_3	

(or)

SELECT dno
FROM (Select dno, count(*) c
From Emp
Group By dno) temp
where $c \geq 3$;

* HAVING clause query can be rewritten by using where clause.

④ ⇒ Retrieve ~~dept~~ dno such that average salary of female employees of department more than average salary of all male employees of company.

① SELECT dno
FROM emp
where gender = female
group by dno
having AVG(sal) > (Select AVG(sal)
From Emp
where gender = male) ;

AVG(sal): female emp avg. sal of each dept.

avg sal of all males of all departments

fmp.

eid	dno	sal	gender	
e_1	d_1		F	✓
e_3	d_1		F	
e_5	d_2		F	
e_7	d_2		F	

② SELECT $T_1.dno$
FROM (Select dno, Avg(sal) sal
From Emp
where gender = female
Group by dno) T_1,
(Select Avg(sal) sal
FROM Emp
where gender = male) T_2
where $T_1.sal > T_2.sal$;

dno	Avg of fem

> ... Avg of M

⑤ ⇒ Retrieve dno such that avg. sal of female emp's of each department more than avg. sal of male emp. of same department.

① Select dno
FROM Emp T_1
where $T_1.gender$ = female
Top ① Group by $T_1.dno$
Top ③ having AVG($T_1.sal$) > (Select AVG(sal)
FROM Emp T_2
where $T_2.gender$ = male and
$T_2.dno = T_1.dno$)
Bottom ② ;

② Select T_1.dno
FROM (Select dno, AVG(sal) sal
FROM Emp
where gender = female
Group by dno) T_1,
(Select dno, AVG(sal)sal
FROM Emp
Group by dno
where gender = male) T_2
where T_1.sal > T_2.sal and T_1.dno = T_2.dno;

---

⑥ ⇒ Retrieve Eid's whose salary more than some emp salary of dept 5. (compare dept 5 with dept 5 also)

RA: $\pi_{eid}\left(\text{Emp} \underset{\substack{sal>s \\ \wedge d=5}}{\bowtie} \rho_{T,D,S,G}(\text{Emp})\right)$

SQL: ① SELECT DISTINCT T_1.eid — Join Query
FROM Emp T_1, Emp T_2
WHERE T_1.sal > T_2.sal and
T_2.dno = 5;

Nested ② Select eid
FROM emp
where sal > ANY (Select sal FROM Emp where dno = 5); — sal > (min sal of dept 5)

O(n^2) ③ Select T_1.eid
FROM Emp T_1
where EXISTS (Select *
FROM Emp T_2
where T_2.dno = 5
and T_1.sal > T_2.sal); — Corelated Nested Query

T_1	Eid	dno	sal
✗	e1	4	10
✗	e2	5	20
✓	e3	4	30
✓	e4	5	40
✓	e5	3	50

T_2	Eid	dno	sal	
	e1	4	10	✗ ✗ ✗ ✗ ✗
	e2	5	20	✗ ✗ ✓ ✓ ✓
	e3	4	30	✗ ✗ ✗ ✗ ✗
	e4	5	40	✗ ✗ ✗ ✗ ✓
	e5	3	50	✗ ✗ ✗ ✗ ✗

⑦ ⇒ Retrieve eid's whose sal more than every emp of dept. 5.

RA ⇒ $\pi_{eid}(emp) - \pi_{eid}\left(\text{Emp} \underset{\substack{sal<s \\ \wedge d=5}}{\bowtie} \rho_{T,D,S,G}(emp)\right)$

SQL ⇒ ① Select eid from emp
Join Query EXCEPT
Select T_1.eid
FROM Emp T_1, Emp T_2
where T_1.sal <= T_2.sal and
T_2.dno = 5;

② SELECT Eid
Nested Query FROM Emp T_1
Where sal > ALL (SELECT sal FROM Emp where dno = 5);
sal > ALL: sal = 20, 40
where sal > max(Select sal From Emp where dno = 5)

③ Select T_1.eid
FROM Emp T_1,
where NOT EXISTS (Select *
FROM Emp T_2
where T_2. dno = 5 and
T_2, sal ≥ T_1.sal);
T_1.sal ≥ every T_2.sal

$\neg(T_1.sal \le$ some sal of $d=5) \equiv T_1.sal$ more than every sal of dept 5.
$T_1.sal \le$ some sal of $d=5$

eid	dno	sal
e1	4	10
e2	5	20
e3	4	30
e4	5	40
e5	3	50

exists ($T_1.sal \le T_2.sal$)
atleast one $T_1.sal \le T_2.sal$.

Q. R(A...) S(B....) ***

① "A" value of R more than some B value of rel S.

ⓐ RA $\pi_A(R \underset{R.A > S.B}{\bowtie} S)$

ⓑ SQL

① SELECT DISTINCT R.A
FROM R, S
where R.A > S.B

② SELECT R.A
FROM R
where R.A > ANY (Select B ~~from S~~
from S);

③ SELECT R.A
FROM R
where EXISTS (SELECT *
FROM S
where R.A > S.B);
or
S.B < R.A

Q. R (A...) S(B...) ***

. Retrieve 'A' value which are more than every "B" of S.

ⓐ RA $\pi_A(R) - \pi_A(R \underset{R.A \leq S.B}{\bowtie} S)$

ⓑ SQL ① SELECT A
FROM R
EXCEPT
Select A
FROM R, S
WHERE R.A ≤ S.B;

② Select A
From R
where A > ALL (Select B
FROM S);

③ Select A
FROM R
WHERE NOT EXISTS (Select *
FROM S
where R.A ≤ S.B
(or)
S.B ≥ R.A);

SQL Queries equal to "/" of RA :

Stud (Sid Sname age) Course (Cid Cname Instr) Enroll (Sid Cid fee)

⇒ Retrieve Sid's enrolled every course taught by Korth.

ⓐ RA : $\pi_{Sid,Cid}(Enroll) / \pi_{Cid}(\sigma_{Inst = Korth}(Course))$

ⓑ SQL Co-related Query:

Enroll T1

Sid	Cid
S1	C1
S1	C2
S1	C3
S2	C1

Course

Cid	Instr
C1	Korth
C2	Korth
C3	Korth
C4	Navathe

− Enroll T2

Sid	Cid
S1	C1
S1	C2
S1	C3
S2	C1

= ϕ

Retrieve T_1.Sid from Enroll(T_1) only if [All Cid's of Korth] − {Cid's of Enroll(T_2) which are enrolled by T_2.Sid}. = ϕ empty

✓ S_1 ⇒ $\begin{bmatrix} C1 \\ C2 \\ C3 \end{bmatrix} - \begin{bmatrix} C1 \\ C2 \\ C3 \end{bmatrix} = \phi$

✗ S_2 ⇒ $\begin{bmatrix} C1 \\ C2 \\ C3 \end{bmatrix} - [C1] = \begin{bmatrix} C2 \\ C3 \end{bmatrix}$

① Select T_1.sid
① FROM Enroll T_1
③ Where NOT EXISTS (
② Select Cid
FROM Course
where Instr = Korth
EXCEPT
Select T_2.Cid
From Enroll T_2
where T_2.sid = T_1.sid) ;

② $\pi_{Sid\,Cid}(Enroll) / \pi_{Cid}(\sigma_{Inst=Korth}(Course))$

$\equiv \pi_{sid}(Enroll) - \pi_{sid}\Big(\underbrace{\pi_{sid}(Enroll) \times \pi_{Cid}(\sigma_{Instr=Korth}(Course)) - \pi_{Sid\,Cid}(Enroll)}_{R}\Big)$

```
SELECT Sid
FROM Enroll
EXCEPT
 Select Sid
 FROM ( Select Enroll.sid, Course.Cid
        FROM Enroll, Course
        where Instructor = Korth
        EXCEPT
        SELECT Sid, Cid
        FROM Enroll
      ) R ;
```

⇒ Retrieve Cid's enrolled by every student whose age more than 20.

$\pi_{Cid\,Sid}(Enroll) / \pi_{Sid}(Enroll \bowtie Stud)_{age>20}$

Retrieve T_1.cid from Enroll only if

[all Sid's whose age more than 20] − [Sid of enroll(T_2) who are enroll by T_1.Cid] = ϕ

Select T_1.Cid
① FROM Enroll T_1
③ where NOT EXISTS (
② Select Sid
FROM Stud
where age > 20
EXCEPT
Select T_2.sid
FROM Enroll T_2
where T_2.Cid = T_1.Cid);

* Catalog (Supplierid, partId)

Sid	Pid
S1	P1
S1	P2
S1	P3
S2	P2

Parts (Pid, Col)

Pid	Col
P1	Red
P2	Red
P3	Red
P4	Green

⇒ Retrieve Sid supplied every red pot.

~~Retrieve Sid only if~~

$\pi_{Sid\,Pid}(Catalog) / \pi_{Pid}(\sigma_{Col=red}(Parts))$

Retrieve T_1.Sid from Catalog only if

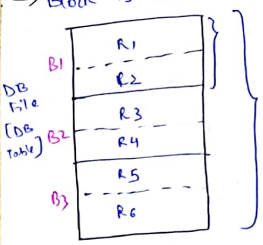
[all red part Pid's] − [Pids of Catalog(T_2) which are supplied by T_1.Sid] = ϕ

```
Select T1.sid
(1) From catalog T1
(3) Where NOT_EXISTS (
        Select pid
        From Parts
        Where color = red
    (2) EXCEPT
        SELECT T2.pid
        FROM CATALOG T2
        Where T2.sid = T1.sid) ;
```

11/11/2017

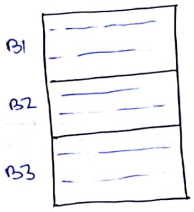
File Organization & Indexing: ***

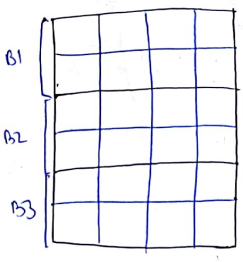
→ DB is collection of files [tables].
→ File is collection of pages [Blocks]
→ Block is collection of records.



→ Data access from disk block by block.

OS Flat file



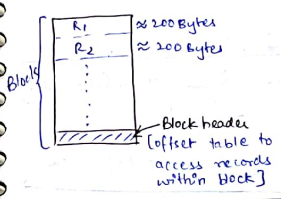


Records of DB file:

① Fixed Length:

```
Create table R
( A char (100),
  B char (50),
  C char (50),
);
```

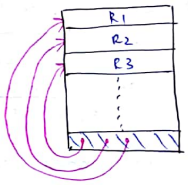
- Size of record ≈ 200
- All records are of same size.



② Variable Length:

```
Create tables
( D char (100)
  E char (50)
  F text
);
```

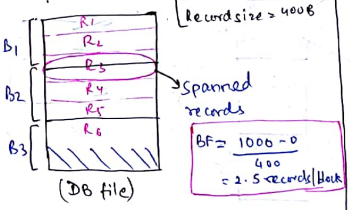
- Records of file(s) may not be of fixed length.



⇒ Records of the file can be organized in two ways:

① Spanned Organization:

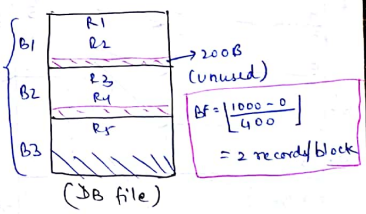
- Record allowed to span/store in more than one block.



- Possible to allocate file without Internal fragmentation.
- More access cost.
- Too complex to organize.

② Unspanned organization:

- complete record must be stored in one block.



- May not possible to allocate file without Internal fragmentation.
- Less access cost.
- Easy to organize.

* → To allocate DB file with fixed length records, unspanned organisation preferred.

* → To allocate DB file with variable length records, spanned organization preferred.

Block Factor:

→ Maximum possible records per block..

Block size : B Bytes

Block header size : H Bytes

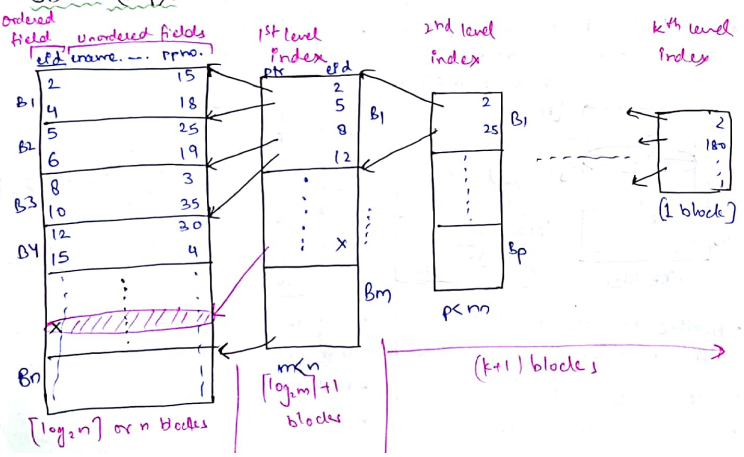
Record size : R Bytes

* → Block factor for unspanned organisation (by default) = $\left\lfloor \frac{B-H}{R} \right\rfloor$ records/block.

→ Block factor for spanned organisation = $\frac{B-H}{R}$ records/block

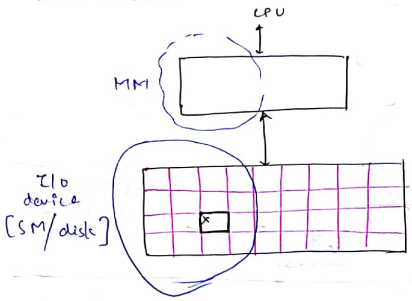
Indexing:

DB File (Emp):



I/O Cost [Access Cost]:

→ # of secondary memory blocks (disk blocks) required to transfer from disk to main memory in order to access some record.



→ I/O cost to access record from DB file of n blocks without index:

(a) based on ordered field access:

```
SELECT *
from emp
WHERE eid = x;
```

$\lceil \log_2 n \rceil$ blocks ~~access~~ = Access cost

•under assumption, that middle block can be accessed directly.

(b) base on ~~ord~~ unordered field access:

```
SELECT *
from emp
where pno = x;
```

n blocks = access cost

* Level of indexing

(0) w/o indexing

(1) 1st level indexing

(2) Multilevel indexing

Access cost

$\lceil \log_2 n \rceil$ or n blocks

$\lceil \log_2 m \rceil + 1$ blocks

$(k+1)$ blocks

Index File :

→ Each entry of index file consists of two fields:

< search key, pointer >

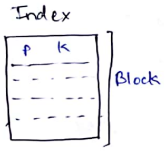
[Field used for indexing]

- Key / Non key
- ordered field / Unordered field

→ **Index file block factor** = Max possible index entries {[key, ptr] pairs} in index block

(key → search key)

Index



Block factor of index file = $\left\lfloor \frac{B-H}{K+P} \right\rfloor$ entries / block

B : size of block
H : size of block header
K : size of search key
P : size of ptr.

* Multilevel Index :

→ Index to index until last level one block.

→ ~~Ideal~~ Ideal I/O cost to access record using multilevel index is k+1 blocks.

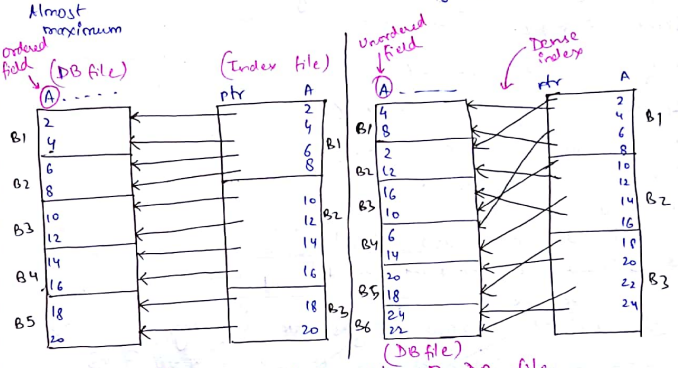
Categories of Index :

1. Dense Index
2. Sparse Index

* Dense Index : [only at 1st level]

→ More/Max entries used for indexing of DB file.

Almost maximum



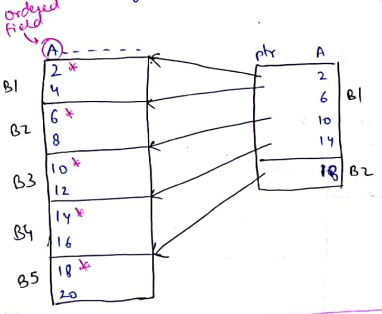
No. of index file entries = No. of entries of DB file.

* Sparse Index :

→ Less entries in index file.

→ for set of records, there exists entry in index file.

→ Sparse index possible only if search key is ordered field of the DB file.



✓ No. of index file entries < # of entries of DB file.

for * problems No. of index file entries = # of Blocks of DB file

Q. File consists 50,000 records with record size: 100 Bytes.
Block size = 1024 Bytes. Search key = 10 Bytes
Pointer = 5 Bytes.

① # of dense index blocks?
② I/O cost using dense index?
③ No. of sparse index blocks?
④ I/O cost using sparse index?
⑤ I/O cost w/o index:
   ⓐ Based on ordered field
   ⓑ Based on unordered field.

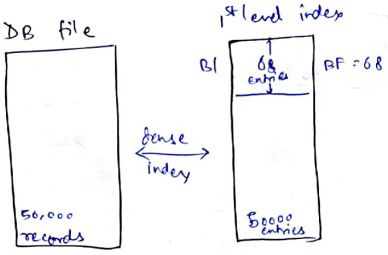
① ~~entries 50,000 * entry size 15 = 7,50,000 Bytes for indexing.~~ ✗
~~# of dense index blocks = 750000 / 1024 = 750 733~~

② ① Block factor ⇒ Entry size = 15 bytes = $\frac{1024-0}{10+5}$

$= \frac{1024}{15} \approx 68$ entries in one block

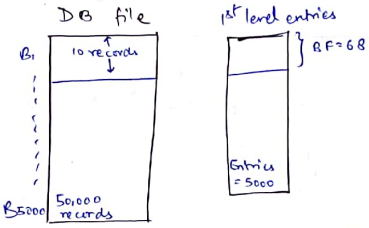
$\frac{50000}{68} = 735.29 \approx 736$

③ ⇒ Dense index



③

⇒ Sparse index:



No. of entries in sparse index file = No. of blocks in DB.

BF of DB $= \left\lfloor \frac{B-H}{R} \right\rfloor$

$= \left\lfloor \frac{1024-0}{100} \right\rfloor = 10$

# of sparse index blocks $= \left\lceil \frac{5000}{68} \right\rceil = 74$

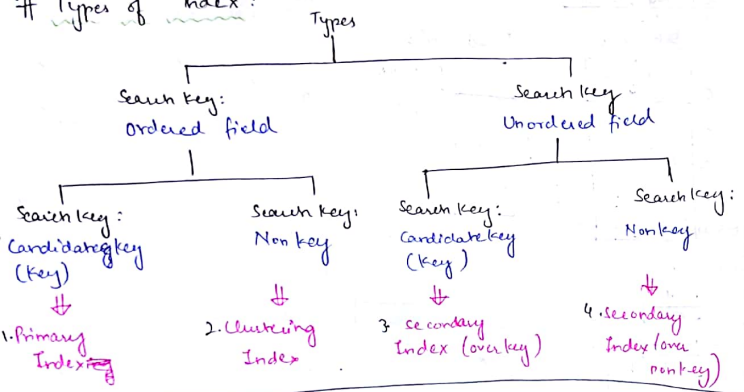
② I/O cost using dense index $= \lceil \log_2 m \rceil + 1 = \lceil \log_2 736 \rceil + 1$
$= 11$ blocks

④ $\lceil \log_2 74 \rceil + 1 \Rightarrow 7 + 1 = 8$ blocks

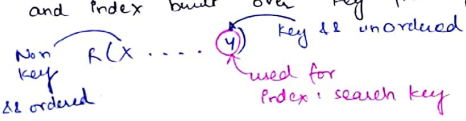
⑤ ⓐ $\lceil \log_2 n \rceil = \lceil \log_2 5000 \rceil = 13$ blocks

ⓑ Unordered ⇒ $n = 5000$ blocks.

Types of index:



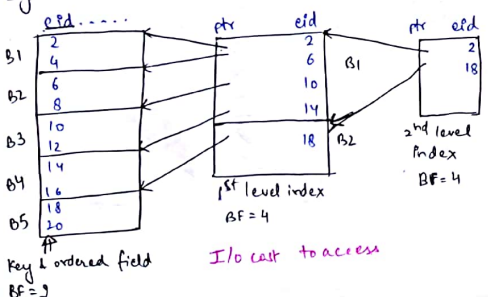
Q. Records of relation R physically ordered over nonkey field and index built over key field of rel R....



* Primary Index:

→ Search key: Key and ordered field of DB file.

→ E.g.



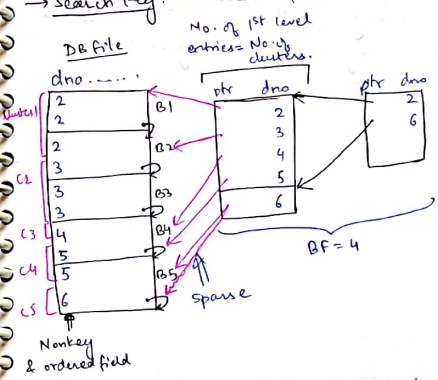
* I/o cost to access record using primary index with multilevel indexing = (k+1) blocks

→ For any database relation, atmost one primary index is possible [because search key is ordered field and rest all fields are unordered].

→ Primary index can be dense or sparse but sparse primary index is preferred.

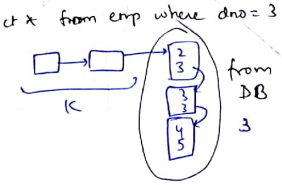
* Clustering Index:

→ Search key: Ordered field and non key.



* I/o cost to access cluster of records using clustering index with multilevel index = [k + one or more DB blocks access until next cluster begins] blocks

Select * from emp where dno = 3; = 5



→ Clustering index mostly sparse index [Clustering index can be dense, if each cluster with only one record.].

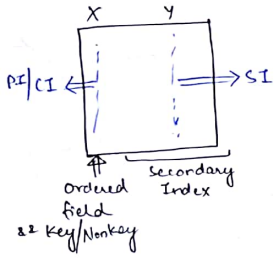
→ For any database relation at most one clustering index is possible.

→ For any database relation, either primary index or clustering index is possible, but not both.

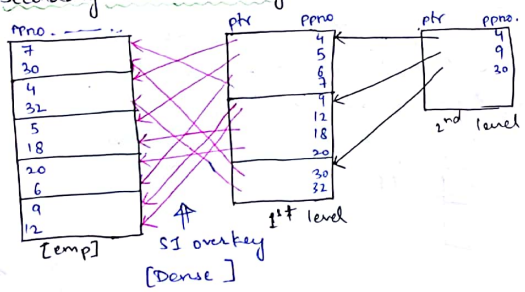
* Secondary Index:

→ Search Key: Unordered fields and
- key / Nonkey.

→ Secondary index is secondary possible way to access data even if Primary Index/Clustering Index already exist.

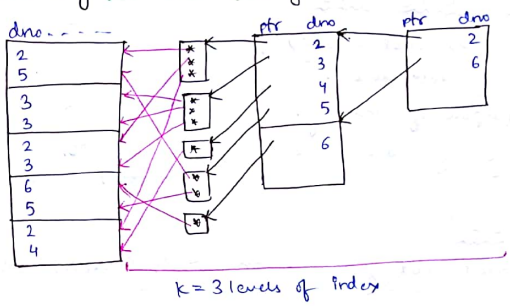


Secondary Index over key:



I/o cost to access records using secondary indexing with MLI
= (K+1) blocks

Secondary index over non key:



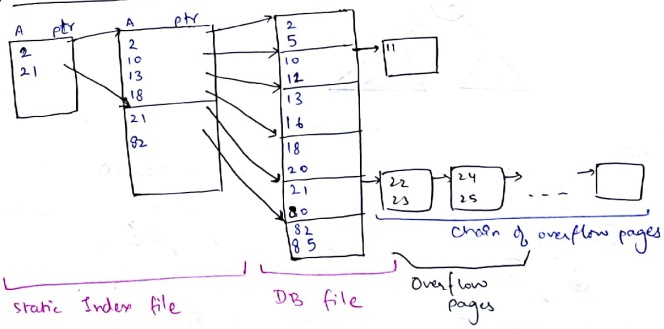
I/o cost to access all records of same non-key value
= K + # of DB blocks, i.e., no. of pointers in given redirect page.

Eg select * from emp where dno = 3 → 6 blocks

Q.U pgs

Multilevel Index (MLI):

① Static MLI:



→ Index file is constant and newly inserted records are stored in overflow pages.

Disadvantage of static MLI:

1. → Worst case access cost = O(n)
   where, n ⇒ no. of data blocks.
   - defeats the purpose of indexing.
2. Minimum usage of index block can become 0%.

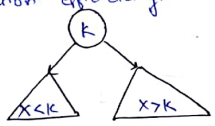
2. Dynamic MLI:

→ Index file must modify based on records insertion/deletion.

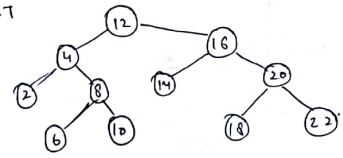
→ *** Standard data structure used for dynamic MLI:
- B Tree
- B+ Tree ***
  - Balanced

⇒ Search Tree:

→ DS used to store keys/elements in tree format and every parent key is more than keys of left subtree and less than keys of right subtree to perform search operation efficiently.



E.g.: BST



X = 10 //search
12, 4, 8, 10 → successful search, probe sequence

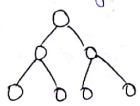
X = 17 //search
12, 16, 20, 18 →

→ Probe sequence to search key in search tree is exactly one node access from each level.

→ Search tree of n keys:

(a) Minimum height = O(log n)

Advantage

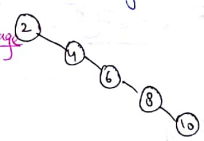


θ(log₂n)

Search cost = O(log n)

(b) Maximum height = O(n)

Disadvantage



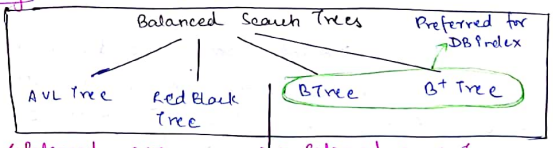
Height = O(n)

Search cost = O(n)

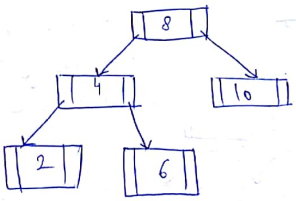
Balanced Search Tree: [Height restricted Search Tree]

→ Max height of search tree for "n" distinct keys should not exceed O(log n).

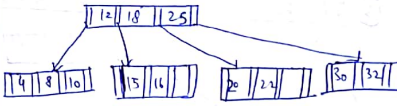
Advantage: Worst case search cost to search element = O(log n).



← Balanced BST →



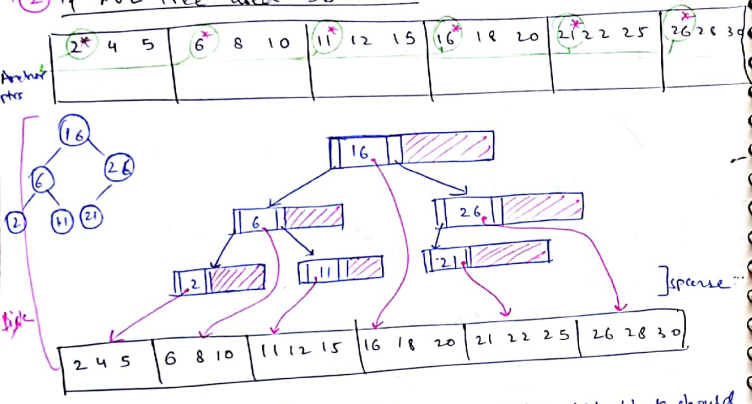
← Balanced search trees with degree of node "p" →



⇒ Why B/B+ Tree used for DB file index rather than Balanced BST?

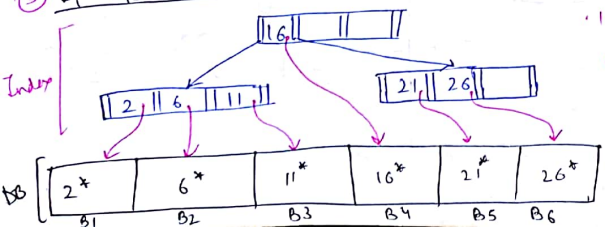
① Because to reduce access cost.
[DB file stored in disk, index to DB file must be also in Disk &
"Data Access from Disk in terms of blocks."]

② If AVL Tree used DB index:



→ If AVL Tree used for DB index, one entire disk block should be allocated for index node and only one key stored in one node, which leads to wastage of disk space allocated for index and more access cost.

③ If B/B+ Tree used for indexing:



- less I/O cost
- less wastage of disk space

Conclusion

MM Application
- Search over MM data. [keys stored in Main m/m]
  - less # of keys, n = 50
    - Preferred DS: AVL Trees
  - Huge # of keys, n = 1,00,000
    - ✗ If AVL tree used, then: ⇒ 1,00,000 nodes, 2,00,000 pointers
    - Preferred DS = BTree

SM Application
- Search over secondary m/m block (data)
  - less # of keys
  - Huge # of keys
    - → Preferred DS = BTree / B+Tree
    - → I/O cost reduction more important

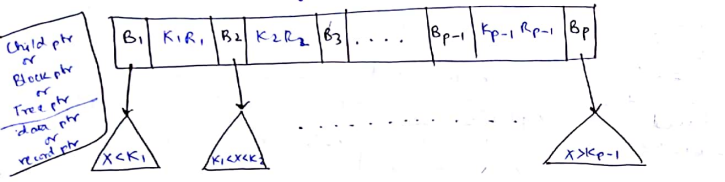
12/11/2017

B Tree definition:

→ Order (p): Max possible child pointers in BTree node. (degree)

① Structure of node:

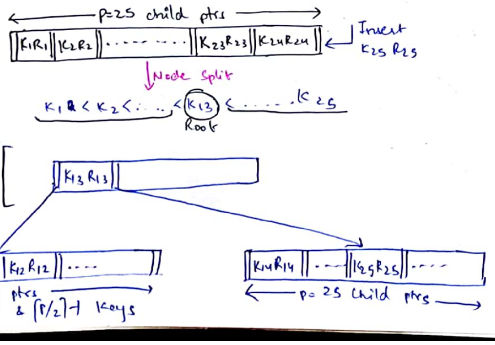
p child ptrs, (p-1) [key, record ptr] pairs.



② Every internal node except root with atleast $\lceil p/2 \rceil$ child ptrs & $(\lceil p/2 \rceil - 1)$ keys atmost p child ptrs & (p-1) keys must be stored

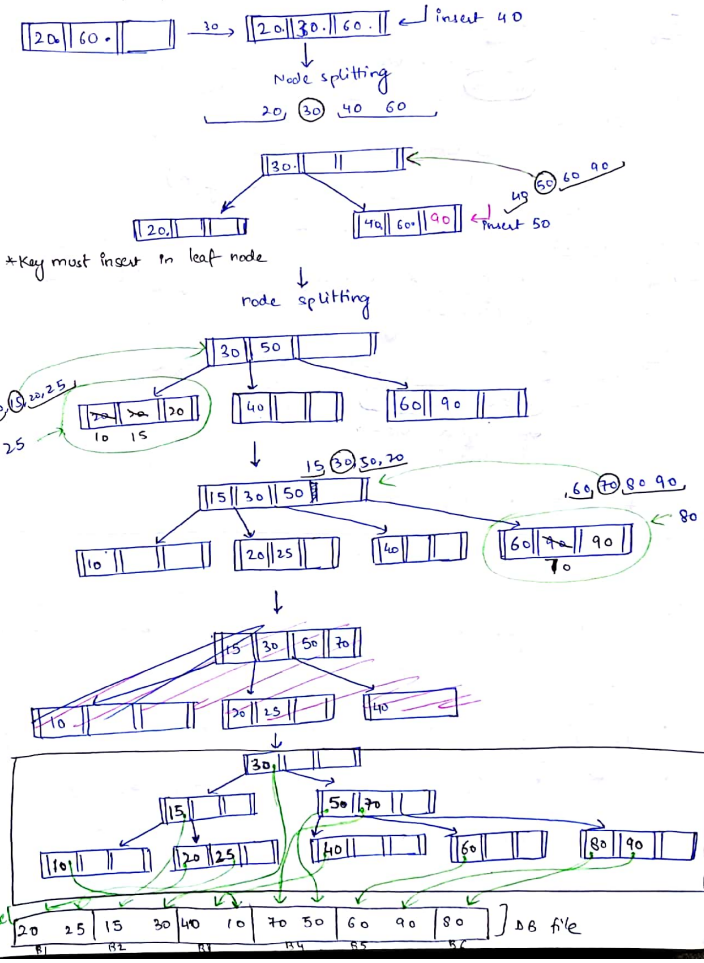
③ Root can be atleast 2 children with 1 key and atmost p child pointers with (p-1) keys should be stored.

④ Every leaf node must be at same level and keys in node should be in sequence order.



Eg Construct B Tree with order p=4 [Max CP per node] and following sequence of keys:

20, 60, 30, 40, 90, 50, 10, 15, 25, 70, 80



*Advantages:

→ BTree index best suitable for random access of any one record.

E.g. SELECT *
FROM R
WHERE (A = 25); ← any one record access of file.

I/O cost = (k+1) blocks access
k ≈ $\theta(\log_p n)$
I/O cost of random access query = $\theta(\log_p n)$

*Disadvantage:

→ BTree index not best suitable for sequential access of range of records.

E.g. Select *
FROM R
WHERE (A ≥ 25 && A ≤ 40); ← sequential access of range of records
16 keys possible →
[Successful Unsuccessful]

I/O Cost = $\theta(X \cdot \log_p n)$

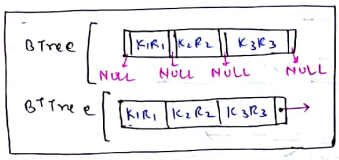
B^+ Tree Definition:

→ order (p) : Max possible pointers in B^+ Tree node.
[degree]

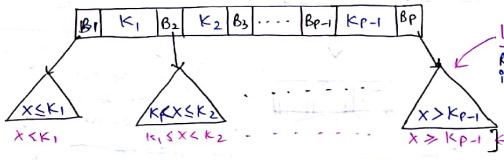
① Structure of node:

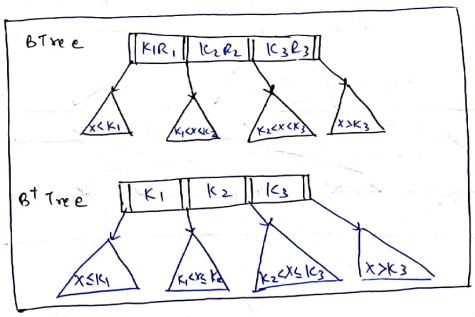
ⓐ Leaf node: consist only (Key, Rp) pairs and one block pointer pointed next leaf node.

P: # of ptrs. | K_1R_1 | K_2R_2 | K_3R_3 | ... | K_{p-1} R_{p-1} | → Block pointer to next leaf node

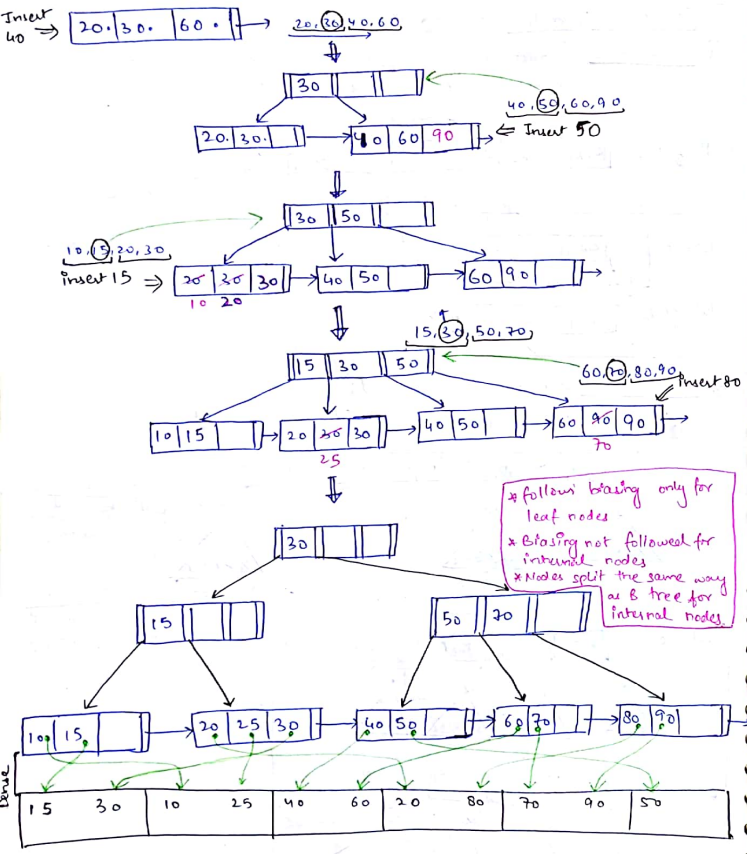


ⓑ Internal node: - consists only keys and child pointers.
- No record pointer in the internal node.





Q. Construct B+ Tree with order p=4 [max possible pointers per node]
for the sequence:
20, 60, 30, 40, 90, 50, 10, 15, 25, 70, 80



* follow biasing only for leaf nodes
* Biasing not followed for internal nodes
* Nodes split the same way as B tree for internal nodes

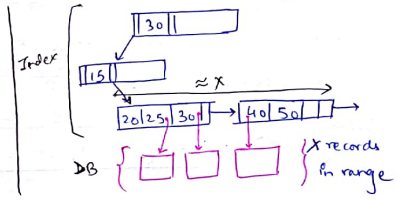
* Advantage:

① → B+ Tree index best suitable for random access of any one record.
(Same as B Tree).

I/O cost of random access query = $\Theta(\log_p n)$

② → B+ Tree index best suitable for sequential access of range of records.

E.g. SELECT *
FROM R
Where A >= 25 && A <= 40;



I/o cost = $\log_p n + X + X$
$\approx \Theta(X + \log_p n)$

→ In B+ Tree:

① All keys that are required for indexing must be stored at leaf level.
② Every leaf node pointed to next leaf node.
③ Leaf node consists of key-data pointer pairs and one block pointer to point next leaf node.
④ Internal node consists only keys and child pointers.
⑤ Entire tree fully left biased or right biased.

• → In left biased B+ Tree, keys stored in internal nodes are maximum keys of each leaf node except the last leaf node.
[No. of internal node keys = No. of leaf nodes − 1]

Problems:

① Find best possible order of

B Tree node / B+ Tree node:

Q. Block size = 1024 Bytes Block ptr = 6 Bytes
Search Key = 9 Bytes Record ptr = 8 Bytes

What is best possible order of BTree node, if order "p".
Max possible block ptrs in BTree node?

Order p:
Max Bp per node

Block (BTree node)

$p \times Bp + (p-1)[Key + R_p]$

$p \times Bp + (p-1)[Key + R_p] \leq$ Block size

$6p + (p-1)[9 + 8] \leq 1024$

$23p \leq 1024 + 9$

$p = \left\lfloor \frac{1033}{23} \right\rfloor = 45 = p$

Q2. Block size = 512 Bytes
Bp = Rp = 10 Bytes
Search key = 15 Bytes
What is order of B Tree node of order P is max possible keys per node?

Block [BTree]

$(p+1)Bp + p[Keys + Rp]$

$(p+1)10 + p[10+15] \leq 512$

$35p \leq 502$

$p = \left\lfloor \frac{502}{35} \right\rfloor = 14$ [max possible keys per node]

Q3. Block size = 2048 Search key = 40 B.
Bp = Rp = 20 Bytes
What is best possible order of B Tree node if order P: b/w 1 to 2P keys in root
b/w P to 2P keys in other nodes?

BTree node

$(2p+1)Bp + 2p[Key + Rp]$

$(2p+1)20 + 2p(40+20) \leq 2048$

$p = 12$
[order p = 12]
Capacity of node = 24

Q4 Block size = 1024 B Block ptr = 6 B
Search key = 9 B Record ptr = 8 B
What is best possible order of B+ Tree ~~node~~ leaf node and internal node if order p = max possible pointers in B+ Tree node?

* If Rp is

Soln: Internal node:
P: # of Block ptrs

$p \times Bp + (p-1)Keys$

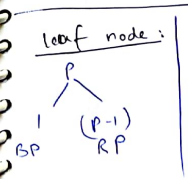
$p \times Bp + (p-1)Key \leq$ Block size

$6p + (p-1)9 \leq 1024$

$15p \leq 1033$

p (order) $= \left\lfloor \frac{1033}{15} \right\rfloor = 68$

Leaf node:



$(p-1)[Keys, R_p] + 1 \times Bp$

$(p-1)(Key + R_p) + Bp \leq$ Block size

$p-1(9+8) + 6 \leq 1024$

$17p \leq 1035$

$p = \left\lfloor \frac{1035}{17} \right\rfloor = 60$

* If the size of record pointer is same as the size of block pointers, then, the order of B^+ Tree leaf node is equal to B^+ Tree internal node.
if $R_p = B_p$, then $P_{internal} = P_{leaf}$

Q.5 Block size = 512 Bytes Search key = 15 Bytes
$B_p = R_p = 10$ B

Best possible order for B^+ tree?
Order is max possible keys per node.

P[keys] + (P+1) Bp | P[keys, Rp] + 1 Bp

Internal:
$P * 15 + (P+1)10 \leq 512$
$15P + 10P + 10 \leq 512$
$25P \leq 502$
$P = \lfloor \frac{502}{25} \rfloor = 20$

Leaf: same for both internal as well as leaf nodes as $R_p = B_p$.

Q.6 Block size = 2048 What is best possible order for B^+ tree?
$B_p = R_p = 20$ B
Search key = 40 By.
Order p: b/w 1 to 2p keys for root node
b/w p to 2p keys for other nodes.

(2p+1)Bp + 2p[keys] | (2p) [keys, Rp] + 1Bp

$(2p+1) * 20 + 2p * 40 \leq 2048$
$40p + 20 + 80p \leq 2048$
$120p \leq 2028$
$p = \lfloor \frac{2028}{120} \rfloor = 16$

Capacity of node = 32

NOTE (sm application)

① → If same size blocks allocated for "B Tree nodes" and "B^+ Tree nodes", then, "order of B^+ Tree node" > "order of B Tree node"

[# of keys can store in B^+ Tree node] > [# of keys can store in B Tree node]

⇒ If same size blocks allocated for B Tree index and B^+ Tree index, then

* [# of B Tree index nodes for n distinct keys] ≥ [# of B^+ Tree index nodes for n distinct keys]
* [# of levels of B Tree index for n-distinct keys] ≥ [# of levels of B^+ tree index for n-distinct keys]
* [I/O cost of B Tree index] ≥ [I/O cost of B^+ Tree index]

B^+ Tree index preferred over B Tree index for database Table because I/O cost of range query as well as I/O cost of random access query, both are less in B^+ Tree index.

② * If order of BTree node = order of B^+ Tree node. [M.M. Application possibly]
Then,
[# of B Tree index nodes for n distinct keys] ≤ [# of B^+ Tree index nodes for n distinct keys]
[# of B Tree index levels for n distinct keys] ≤ [# of B^+ Tree index levels for n distinct keys]

Eg: B Tree order p=14 keys

$15 B_p + 14(\text{keys} + B_p)$

B^+ Tree order p=14 keys.

$14[B_p + \text{keys}] + B_p$

$14\,\text{keys} + 15 B_p$

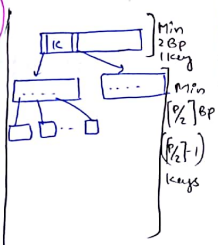
② Compute Min | Max keys in BTree | B^+ Tree index for given height (h):

* Minimum possible keys in B|B^+ Tree of order p [Max child ptrs per node] and height (h | [0, 1, ... h]

Assume: Root → Min 2 Bp & 1 key

Other nodes → Min $\lceil p/2 \rceil$ Bp & $\left(\lceil p/2 \rceil - 1\right)$ keys.

Height	Min Nodes $\approx \Theta\left(\lceil \frac{p}{2} \rceil^h\right)$	Min BP	Min keys $\approx \Theta\left(\lceil \frac{p}{2} \rceil^h\right)$
Root 0	1	2	1
1	2	$2\lceil p/2 \rceil$	$\left(2\lceil p/2 \rceil - 1\right)$
2	$2\lceil p/2 \rceil$	$2\lceil p/2 \rceil^2$	$2\lceil p/2 \rceil\left(\lceil p/2 \rceil - 1\right)$
3	$2\lceil p/2 \rceil^2$	$2\lceil p/2 \rceil^3$	$2\lceil p/2 \rceil^2\left(\lceil p/2 \rceil - 1\right)$
⋮			
h	$2\lceil p/2 \rceil^{h-1}$	$2\lceil p/2 \rceil^h$ (Point to Node at height "h")	$2\lceil p/2 \rceil^{h-1}\left(\lceil p/2 \rceil - 1\right)$



* Min possible keys in BTree with order p and height 'h':

$$= 1 + 2\left(\lceil p/2 \rceil - 1\right) + 2\lceil p/2 \rceil\left(\lceil p/2 \rceil - 1\right) + 2\lceil p/2 \rceil^2\left(\lceil p/2 \rceil - 1\right) + \ldots + 2\lceil p/2 \rceil^{h-1}\left(\lceil \frac{p}{2} \rceil - 1\right)$$

$$= 1 + 2\left(\lceil \frac{p}{2} \rceil - 1\right)\left[1 + \lceil \frac{p}{2} \rceil + \lceil \frac{p}{2} \rceil^2 + \ldots + \lceil \frac{p}{2} \rceil^{h-1}\right]$$

$$\Rightarrow 1 + 2\left(\lceil \frac{p}{2} \rceil - 1\right)\frac{\left(\lceil \frac{p}{2} \rceil^h - 1\right)}{\left(\lceil \frac{p}{2} \rceil - 1\right)} = 1 + 2\left[\lceil \frac{p}{2} \rceil^h - 1\right]$$

$$= 2\lceil \frac{p}{2} \rceil^h - 1$$

Minimum possible keys in B^+ Tree with order p and height (h):

$$= 2\lceil \frac{p}{2} \rceil^{h-1}\left(\lceil \frac{p}{2} \rceil - 1\right) \approx \Theta\left(\lceil \frac{p}{2} \rceil^h\right)$$

Maximum possible keys B|B^+ Tree with order p and height (h):

⇒ Assume max p, Bp and (p-1) keys.

Height	Max node $\approx \Theta(p^h)$	Max BP	Max keys $\approx \Theta(p^h)$
0	1	p	p-1
1	p	p×p	p×(p-1)
2	p^2	$p^2 \times p$	$p^2 \times (p-1)$
3	p^3	$p^3 \times p$	$p^3 \times (p-1)$
⋮	⋮		
h	p^h	—	$p^h(p-1)$

Max nodes $\left\{\frac{p^{h+1} - 1}{p-1}\right\} \approx \Theta(p^h)$

Max keys in BTree with order p and height (h):

$$\left(p^{h+1} - 1\right) \approx \Theta(p^h)$$

Height of B Tree with order p and n - distinct keys

(a) **Minimum height:** [Max possible keys per node]

$n = p^{h+1} - 1$

$\log(n+1) = (h+1)\log_p p \Rightarrow h+1 = \log_p(n+1)$

$\Rightarrow \boxed{h = \log_p(n+1) - 1} \approx \theta(\log_p n)$

(b) **Maximum height:** [Min possible keys per node]

$n = 2\left\lceil \frac{p}{2} \right\rceil^h - 1$

$\log_{\lceil p/2 \rceil}\left(\frac{n+1}{2}\right) = h \Rightarrow \boxed{h = \log_{\lceil p/2 \rceil}\left(\frac{n+1}{2}\right)} \approx \theta(\log_p n) \approx \boxed{\log_{\frac{p}{2}} \frac{n}{2}}$

Keys of B|B+	$\Rightarrow \theta\left(\lceil p/2 \rceil^h\right)$ ... to ... $\theta(p^h)$
Height	$\Rightarrow \log_p n$ to $\log_{p/2} n/2$

Q① Order p: blw 2 to p block ptrs in root node.
blw $\lceil \frac{p}{2} \rceil$ to p block ptrs in other internal nodes.
blw $\lceil \frac{p}{2} \rceil$ to $(p-1)$ keys in leaf node

of B|B+ Tree.

(i) What is min keys and nodes in B|B+ Tree of order = 5 & level = 4?
(ii) What is max keys and nodes in B|B+ Tree of order = 5 & level = 4?

Min keys = $2 \times \lceil \frac{p}{2} \rceil^h - 1$ (B tree) — Root to leaf

Soln. Min keys/nodes of level 4, order p = 5

$\lceil \frac{p}{2} \rceil = 3 \rightarrow$ BP/node except root

(i)

level	Min nodes	Min BP	Min keys
(Root) 1	1	2	1
2	2	2×3	2×2
3	6	6×3	6×2
4	18	~~18×3~~ =	18×2
			53

Min keys of in B Tree of 4 levels = 53

Min keys in B+ Tree of 4 levels = 18×2 = 36

Min nodes in B|B+ Tree = 27 = 1+2+6+18

(ii)

level	Max nodes	Max BP	Max keys
1	1	5	4
2	5	5×5	5×4
3	25	25×5	25×4
4	125	〃	125×4
~~Max~~	156		

Max keys in B Tree of 4 levels = 624

Max keys in B+ Tree of 4 levels = 500

Max nodes in B|B+ Tree = 156

Bulk loading B^+ Tree :

→ Construction of B^+ Tree in bottom up approach.

→ Design of B^+ Tree from leaf nodes to root.

Procedure:

1. Sort the keys in ascending order which are used for indexing.
2. Design leaf nodes:
   distribute keys into leaf nodes.
   [# of leaf nodes].
3. Design internal nodes:
   If "X" nodes at level (l), then, we need to maintain "X" child ptr at level (l-1).
   Distribute "X" child ptrs into nodes
   [# of nodes at (l-1)]

Repeat 3 until root.

Q. # of keys for index = 2500

Order P=7 [max ptrs per node of B^+ Tree]

[Assume 2 to p ptrs root
$\lceil p/2 \rceil$ to p ptrs other nodes]

(i) Min levels of B^+ Tree index [Min nodes] ?

(ii) Max levels of B^+ Tree index [Max nodes]

// Solve using Bulk loading B^+ Tree.

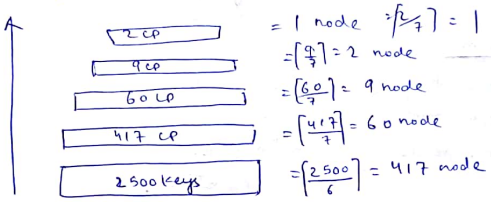
Leaf to root

(i) Min levels of B^+ tree index:

Take max node fill factor.

Max: 7 cptrs = p

6 keys per node



= 1 node $= \lceil 2/7 \rceil = 1$

$= \lceil 9/7 \rceil = 2$ node

$= \lceil 60/7 \rceil = 9$ node

$= \lceil 417/7 \rceil = 60$ node

$= \lceil 2500/6 \rceil = 417$ node

Min levels = 5

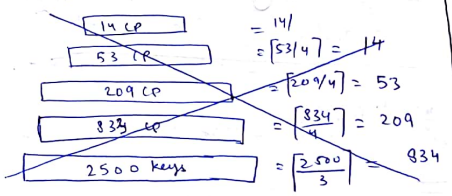
Min nodes = 417+72 = 489 nodes.

(ii) Max levels of B^+ Tree index.

Take min node fill factor

Min: Root → 2 cp & 1 key

Other nodes $\Rightarrow \lceil p/2 \rceil = 4$ cp & 3 keys



$= 14/$

$= \lceil 53/4 \rceil = 14$

$= \lceil 209/4 \rceil = 53$

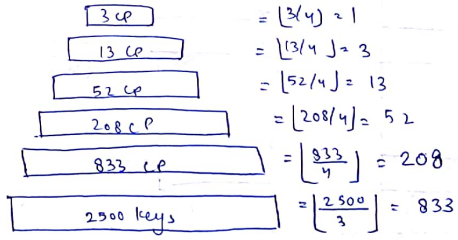
$= \lceil 834/4 \rceil = 209$

$= \lceil 2500/3 \rceil = 834$

Max levels = 6

Max index blocks = 833 + 277

= 1110 nodes.



$= \lfloor 3/4 \rfloor = 1$

$= \lfloor 13/4 \rfloor = 3$

$= \lfloor 52/4 \rfloor = 13$

$= \lfloor 208/4 \rfloor = 52$

$= \lfloor 833/4 \rfloor = 208$

$= \lfloor 2500/3 \rfloor = 833$

BTree Problem:

(i) Find min/max keys of given "l" levels.

(ii) Find min/max levels of given "n" keys.
Design B Tree from root to leaf.

B+ Tree Problem:

(i) Find min/max keys of given "l" levels.
Design B+ Tree from root to leaf.

(ii) Find min/max levels of given "n" keys.
Design leaf to root [Bulk loading B+ Tree]

JOIN Algo: → Nested loop join algo
→ Block nested loop join algo.

* Nested Loop Join Algo:

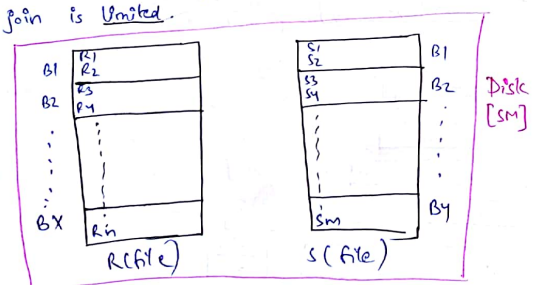
→ Relation "R" with "n" tuples occupies in "X" blocks.

→ Relation "S" with "m" tuples occupies in "Y" blocks.

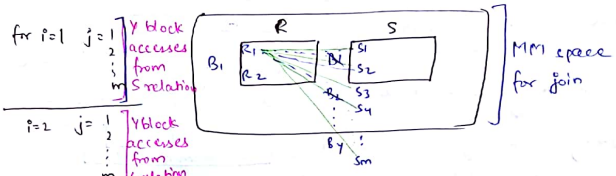
R ⋈ S ⇒ for (i=1 ; i≤n ; i++) → Record no. of outer rel (R)
for (j=1; j≤m; j++) → Record no. of inner rel. (S)
comp (Ri, Sj Join condition)

* Disadvantage of Nested loop join:

- More access cost required to join if MM space allocated to join is limited.



Case 1: Assume only two blocks of MM allocated for join.
[At any time one block of R data and one block of S data can store in MM]

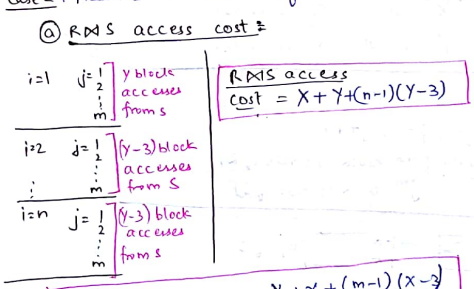


(a) R ⋈ S access cost = {X + n*Y} blocks

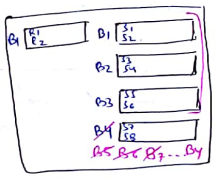
(b) S ⋈ R access cost = {Y + m*X} blocks

Case 2: Assume 5 blocks of MM allocated for join.

(a) R ⋈ S access cost =



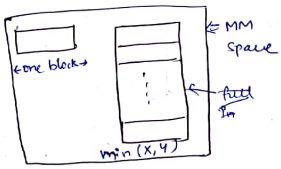
R ⋈ S access cost = X + Y + (n-1)(Y-3)



(b) S ⋈ R access cost = Y + X + (m-1)(X-3)

Case 3: How many minimum MM blocks required to join R, S without any repeated data access from Disk to MM using nested loop join algo?

* full inner relation should be in MM.
* one block for outer relation.



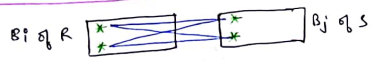
{min(X,Y)+1} blocks of MM required to avoid repeated data access from Disk to MM to perform Join of R & S using Nested Join Algo.

I/O Cost = X+Y

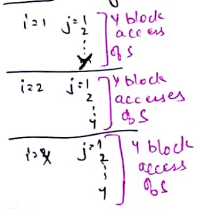
Block Nested loop Join Algo:

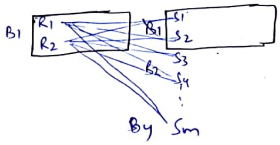
→ Relation "R" with "X" blocks of records.
→ Relation "S" with "Y" blocks of records.

```
R⋈S ⇒ for (i=1 ; i≤X ; i++)
        for ( j=1; j≤Y; j++)
          Join (ith block records of R,
                jth block records of S)
```



Case 1: Only 2 blocks of main memory allocated for Join.



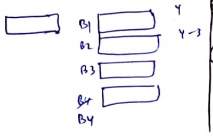


ⓐ R⋈S access cost : (X+XY) blocks

ⓑ S⋈R access cost : (Y+Y.X) blocks

Case 2: ~~5~~ Assume 5 blocks of MM allocated for join.

ⓐ R⋈S access cost ⓑ S⋈R access cost.



ⓐ R⋈S = X+Y+(X-1)(Y-3)

ⓑ S⋈R = Y+X+(Y-1)(X-3)

ER MODEL :

→ ER Diagrams used to represent high level DB design (Diagrammatic representation of DB design)

DB development steps:

Requirement of DB design
⇩
ER Diagrams
⇩
RDBMS Tables design

(High level database design)

↓
Normalization
↓
Design Index
↓
Design user access Rules.

(Low level database design)

#Main Components:

① Attribute
② Entity Sets
③ Relationship sets.

* Attribute :

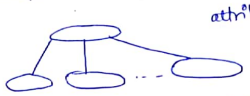
- Key attribute
- Multivalued attribute
- Derived attribute : Value of attribute derived from values of stored attribute.

E.g.: DoB (Stored attribute), age (Derived attribute)

age = system date − DoB

E.g. Select Sid, (Sysdate − DoB) as age
FROM Student.

- Composite attribute: Attribute can represent by two or more attributes.

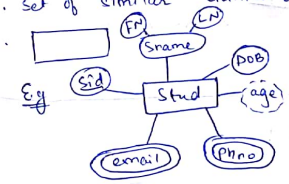


E.g. name →

address →

Entity Set :

- Set of similar entities [objects / tuples].



E.g.

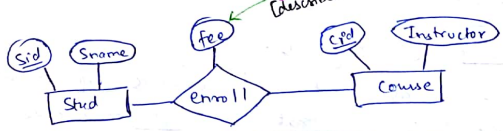
* Integrity constraints :

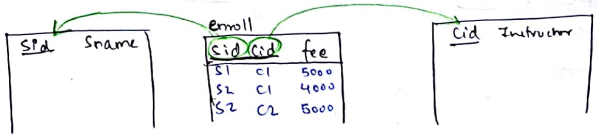
- Primary key / Alternative keys

* Relationship Sets :

→ used to relate two or more entity sets.

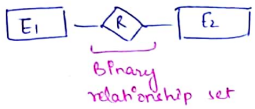
E.g.:



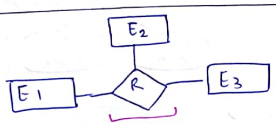


* Relationship Set Integrity constraints:

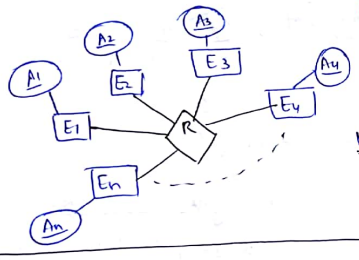
* Primary key / Alternate keys
* Foreign keys.



Binary relationship set



Ternary Relationship set



R(A1, A2, ... An ....)

Min: n attributes & n foreign keys

Constraints of Relationship set : * * *

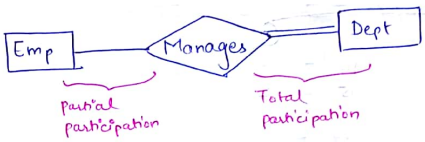
① Mapping (Cardinality)

② Participation

* Participation:

→ If every entity of entity set must related to relationship set, then, Total participation [100% participation]
otherwise, partial participation. [≤ 100% participation]

E.g: Emp & Dept are entity sets.
Manages is relationship set used to relate emp who are manager of dept such that every dept. must have some manager.



* Mapping Constraint:

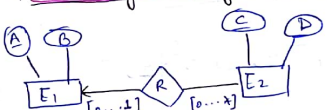
- one (→) : Entity allowed to relate atmost one entity.
- many (—) : Entity allowed to relate 0 or more entities.

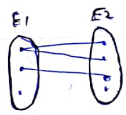
Binary relationship set possible mapping:

① One : Many mapping
② Many : one mapping
③ Many : Many mapping
④ One : One mapping

Candidate keys of relationship set depends on mapping constraint.

⇒ One : Many Mapping relationship set (R) b/w E1 E2 entities:





E1

A	B
2	-
3	-
4	-
5	-

R

A	C
2	14
2	15
3	16

E2

C	D
14	-
15	-
16	-
17	-

(1:M) key

Cand. key of relationship set R(A C) is {C}. for 1:M mapping.

* RDBMS tables design for ERD:

E_1 (A B)

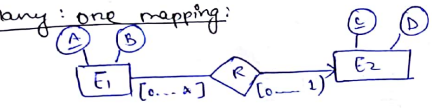
A	B
2	-
3	-
4	-
5	-

$R E_2$ (C D A)

C	D	A
14	-	2
15	-	2
16	-	3
17	-	NULL

→ Minimum 2 relational tables with one FK required for above ERD.

=) Many : one mapping:



E_1 (A B)

A	B
2	-
3	-
4	-
5	-

R (A C)

A	C
2	14
3	14
4	15

key (M:1)

E_2 (C D)

C	D
14	-
15	-
16	-
17	-

Candidate key of relationship set R(A C) is {A} for M:1 mapping.

* RDBMS Tables design for ERD:

R E_1 (A B C)

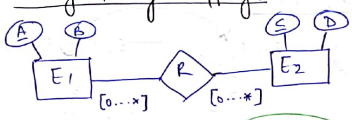
A	B	C
2	-	14
3	-	14
4	-	15
5	NULL	NULL

E_2 (C D)

C	D
14	-
15	-
16	-
17	-

→ Minimum : 2 relational tables with 1 Foreign key

=) Many : Many Mapping :



E_1 (A B)

A	B
2	-
3	-
4	-
5	-

R (A C)

A	C
2	14
2	15
3	15
4	15

E_2 (C D)

C	D
14	-
15	-
16	-
17	-

Many : Many mapping

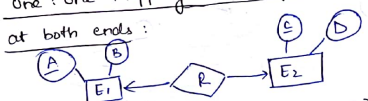
Candidate Key of relationship set R(A,C) is {A C} for M:N Mapping.

* RDBMS design for above ERD:

E_1 (A B) R (A C) E_2 (C D)

Minimum : 3 relational tables && 2 foreign keys

=) One : One Mapping relationship b/w E_1 & E_2 with partial participation at both ends :



E_1 (A B)

A	B
2	-
3	-
4	-
5	-

R (A C)

A	C
2	15
3	17
5	14

key key

E_2 (C D)

C	D
14	-
15	-
16	-
17	-

⇒

X Not allowed in RDBMS

A	B	C	D
2	-	15	-
3	-	17	-
4	-	NULL	NULL
5	-	14	-
NULL	NULL	16	-

Candidate Keys of relationship set R(A,C) is {A, C} for 1:1 mapping.

* RDBMS design for 1:1 Mapping ERD:

E_1R (A B C) — E_2 (C D)

{A, C}

A	B	C
2	-	15
3	-	17
4	-	NULL
5	-	14

(A: Primary key, C: Alt Key)

C	D
14	-
15	-
16	-
17	-

E_1 (A B)

A	B
2	-
3	-
4	-
5	-

E_2R (A C D)

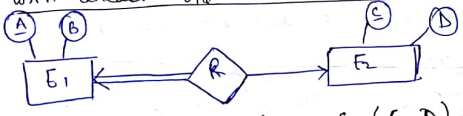
A	C	D
5	14	-
3	15	-
NULL	16	-
2	17	-

(A: Alk, C: PK)

→ Minimum 2 Relational tables and 1 foreign key required.

⇒ One: One Mapping Relationship set b/w E_1 E_2 entity sets with atleast one end is total participation:

(a)



E_1 (A B)

A	B
~~2~~	~~-~~
3	-
~~4~~	~~-~~
5	-

R (A C)

A	C
2	15
3	17
5	14

E_2 (C D)

C	D
14	-
15	-
16	-
17	-

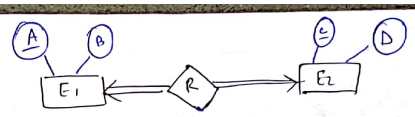
E_1RE_2 (A B C D) — {A, C}

A (AK)	B	C (PK)	D
2	-	15	-
3	-	17	-
5	-	14	-
NULL	-	16	-

Eg: E_1, E_2 with n & m entities respectively. 'R' relation with 1:1 & total participation at E_1.

[n ≤ m] guaranteed.

(b)



E_1RE_2 (A B C D) — {A, C}

m = n

E_1 with n attributes
E_2 with m attributes, with total participation at both ends.

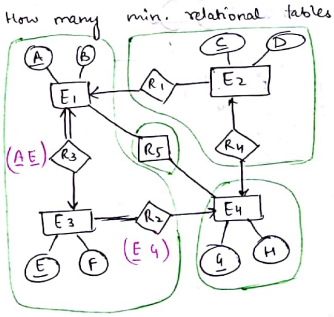
13/11/2017

Q1. E_1 E_2 entity sets, R_1 R_2 relationship sets b/w E_1 & E_2 with 1:M & M:N Mapping. How many minimum relational tables required?

(a) 1 (b) 2 (c) 3 (d) 4

Q2. E_1 E_2 E_3 entity sets
R_1 rel b/w E_1 E_2 with 1:M mapping
R_2 rel b/w E_2 E_3 with M:1 mapping
R_3 rel b/w E_1 E_3 with 1:1 mapping.
How many min. relational tables required?

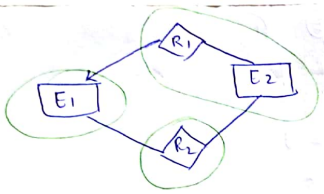
Q3.



4 Tables.

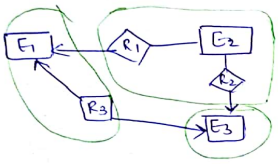
(i) Min relational tables? 4
(ii) # of foreign keys in RDBMS design? 5
(iii) Total no. of attributes in RDBMS design? 5+4+2+2 = 13

(1)



3 Tables.

(2)



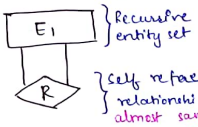
3 Tables.

(3)

- $E_1 R_3 E_2 R_2$ [A B EF G]
  Primary key = E
  Alternative key = A
  Foreign key = G
- $R_1 E_2 R_4$ [C D A G]
  Primary key = C
  Alternative key = G
  Foreign key = A, G
- E_4 [G H]
  Primary key = G
- R_5 [A G]
  Primary key = AG
  Foreign key = A, G

Self Referential relationship set:

→ Entities of entity set (E_1) related to some other entity of same entity set (E).



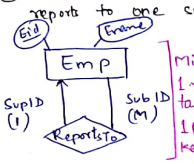
Recursive entity set

Self referential relationship almost same as binary relationship

1:M, M:1, M:N, 1:1 mappings possible

Eg: Emp is entity set and Reports_to relationship set related b/w supervisor and subordinate.

(a) Each supervisor can supervise many subordinates and each subordinate reports to one supervisor.



Minimum 1 relational tables & 1 foreign key

Emp

Eid	Ename
e1	A
e2	B
e3	A
e4	C

Reports-To (1:M)

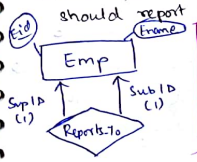
Sup ID	Sub ID
E1	E2
E1	E3
E2	E4

*RDBMS design:

Emp-Reports-To

eid	ename	supID → fk
e1	A	NULL
e2	B	e1
e3	A	e1
e4	C	e2

(b) Each supervisor should supervise one subordinate and each sub-ordinate should report to one supervisor only.



Min^m relational tables = 1

Foreign key = 1

Emp

Eid	Ename
e1	A
e2	B
e3	A
e4	C

R-To (1:1)

Sup ID	Sub ID
e1	e2
e2	e3
e3	e4

* RDBMS design:

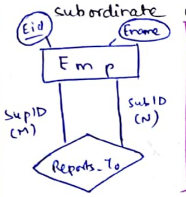
Emp - R-To

Combining Eid & sub ID

Eid	Ename	Sub-ID (FK)
e1	A	NULL
e2	B	e1
e3	A	e2
e4	C	e3

Sub ID: e2, e3, e4, NULL

ⓒ Each supervisor can supervise many sub-ordinates and each subordinate can report to many supervisors.



M:M
2 tables

Emp: Eid Ename

R-To: SupID SubID (M:M)

* RDBMS design:

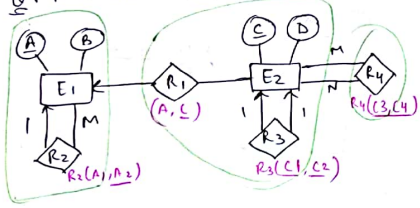
emp

eid	ename
e1	
e2	
e3	
e4	

Reports-to

Sup ID	Sub ID
e1	e2
e1	e3
e2	e3
e3	e4

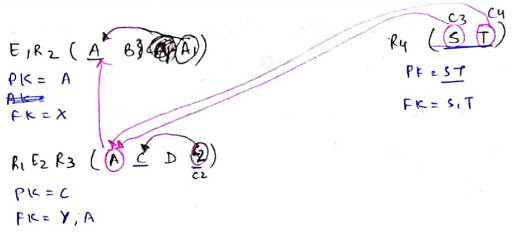
Q. Find min^m relational tables:



(i) Min relational tables. 3

(ii) No. of FKs in RDBMS design. 5

(iii) Total no. of attributes in RDBMS design. 9

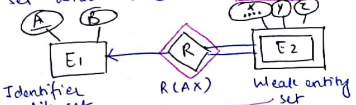


Weak Entity Set:

→ Entity set with no key.
Attributes of entity set not sufficient to differentiate records.

→ Represented by:

→ Entities of weak entity set depends on some strong entity set which is called identifier [OWNER entity set].



X: Partial key

X	Y	Z
...		
X_1	Y_1	Z_1
X_1	Y_1	Z_2

*** • Must be total participation at weak entity set end

*** • Mapping b/w identifier to weak entity set must be 1:M

• To differentiate weak entities partial key combine with identifier key.

⇒ Relationship b/w identifier and weak entity set is also weak relationship.

* RDBMS Design:

• E1 (A B)

• E2 R (A X Y Z)
Primary key (A X)
Foreign key (A)

Min 2 relational tables

→ Weak entity set combines with relationship set in RDBMS design which consist proper candidate key.

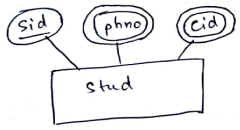
⇒ Multi° valued attribute and weak entity set allowed to represent in ER diagram but not allowed in relational database [RDBMS].

Multivalued Dependency and 4NF:

→ If relation R in BCNF and redundancy exists in Rel R, then, atleast two multivalued attributes converted into single valued attribute in Rel R (forms redundancy over MVDs).

→ If Rel R in BCNF (no redundancy over FDs) (and) atmost one Multivalued attribute converted into single valued attr. in R. (no redundancy over MVD). Then, Rel R free from redundancy. [No ~~chance~~ chance of redundancy over FDs and MVDs)

Eg:



Stud (Sid Pno cid)

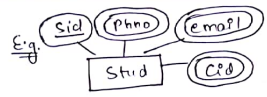
S1	P1/P2	C1/C2/C3
S2	P3	C3

Stud (Sid Pno Cid) → CK [Sid Pno Cid]

S1	P1	C1
S1	P1	C2
S1	P1	C3
S1	P2	C1
S1	P2	C2
S1	P2	C3
S2	P3	C3

- Stud rel in BCNF
- S1 P1, S1 P2 — repeated 3 times
- S1 C1, S1 C2, S1 C3 — repeated 2 times

(redundancy over MVDs)



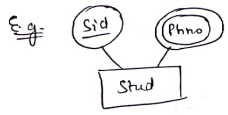
Sid	Pno	email	Cid
S1	P1/P2	e1/e2/e3	C1/C2/C3
S2	P3	e4	C3

Stud

Sid	Pno	email	Cid
S1			
S1			
⋮			
S1			
S2	P3	e4	C3

2·3·3 = 18 tuples for S1

- BCNF
- (S1 P1, S1 P2) repeated 9 times (redundancy)
- (S1 C1, S1 C2, S1 C3) repeated 6 times. (redundancy)



Stud (Sid phno)

Sid	phno
S1	P1/P2
S2	P3

⇓

Stud (Sid phno)

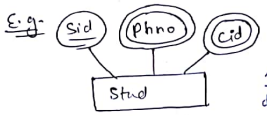
Sid	phno
S1	P1
S1	P2
S2	P3

- BCNF
- No redundancy over MVDs also.

Multivalued Dependency (MVD):

→ Multi° valued attribute (Y) which depends on key attr (X) if converted into single valued attribute in R_2 then, $X \rightarrow\rightarrow Y$ MVD is exists in R.

Eg:

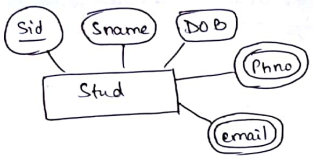


⇒ 1NF design

Stud (Sid Pno Cid)

Sid →→ Pno

Cand. key = { Sid Pno Cid }



{ Sid → Sname
Sid → DOB
Sid →→ Phno
Sid →→ email } FD & MVD set of stud relation.

Properties over MVD : *** [PSUS]

① Complementation Rule :
if $X \rightarrow\rightarrow Y$ is MVD of R
Then $X \rightarrow\rightarrow R-(XUY)$ is also MVD of rel R.

E.g: R(ABCD) : if $A \rightarrow\rightarrow B$ exists in R.
then $A \rightarrow\rightarrow CD$ also must exist in R.

R(A B C D)
4 5 6 7
4 6 7 7
4 5 7 7
4 6 6 7

$A \rightarrow\rightarrow B$, $A \rightarrow\rightarrow CD$

A	B
4	5/6

A	CD
4	67/77

② Trivial MVD :
→ $X \rightarrow\rightarrow Y$ is trivial MVD only if $X \supseteq Y$ (or) $XUY = R$

E.g: ① Stud (Sid Pno)

Sid	Pno
S1	P1
S1	P2
S2	P2

{Sid →→ Pno} (free from redundancy) Trivial MVD
{Sid →→ Pno, Pno →→ Pno} Trivial MVD

② Stud (Sid Pro Cid)

Sid	Pro	Cid
S1	P1	C1
S1	P2	C1
S1	P1	C2
S1	P2	C2

{Sid →→ Pno} Non Trivial MVD
{Sid →→ Cid} Non Trivial MVD

③ R(ABCD) {AB →→ CD} Trivial MVD

NOTE
↳ No non-trivial MVDs possible in rel R(A B) with only two attributes.

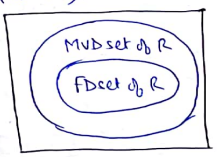
③ MVDs not allowed to split over determiner also.
MVDs not allowed to merge if determiner is same also.

FD: if $X \rightarrow YZ$ implied in R
then, $X \rightarrow Y$, $X \rightarrow Z$
also must in R.

MVD: if $X \rightarrow\rightarrow YZ$ implied in R.
then, $X \rightarrow\rightarrow Y$, $X \rightarrow\rightarrow Z$
may/may not implied in R.

E.g: Stud (Sid Cno Rno Phno)
Sid →→ CnoRno ✓
~~Sid~~ S1 C1R1/C1R2/C2R2
Sid →→ Cno } may not
Sid →→ Rno } in R.

④ Replication Rule :
→ Every FD ($X \rightarrow Y$) can replace as MVD ($X \rightarrow\rightarrow Y$).
but reverse is NOT True.



• If $X \rightarrow Y$, then $X \rightarrow\rightarrow Y$.

Sid → Sname
S1 ---> [A]
single sname for S1
⟹ Sid →→ Sname
S1 ---> [A]
set of names for S1 (can be single also)

Sid →→ Cno
S1 [ ]
↑ set of Cno for S1
⟹ Sid →→ Cno
C1 [ ]
single Cno for S1 (may not true)

DRDO.

Q. Which is correct?

(i) if $A \to BC$, then $A \to B$, $A \to\to C$

(ii) if $A \to\to BC$, then $A \to B$, $A \to\to C$

(iii) if $A \to BC$, then $A \to\to B$, $A \to\to C$

(iv) if $A \to\to BC$, then $A \to\to B$, $A \to C$

(a) i, ii (b) ii, iv (c) i, iii (d) i, ii, iii, iv

4NF Definition:

→ Relational Schema 'R' in 4NF, iff:

(a) every non-trivial FD $X \to Y$ in R with X is superkey.

(b) every non-trivial MVD $X \to\to Y$ in R with X is superkey.

→ Sufficient for 4NF.

** • Attribute Closure
• Candidate keys identification
• Testing of attribute set superkey or not
} should use only FD set of relation.

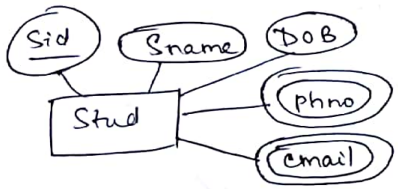
Eg. R(A B C D) $\{A \to B, B \to C, A \to\to D\}$

$(A)^+ = \{A, B, C\}$ $(AD)^+ = \{A, B, C, D\}$

Cand. key = {AD}

# of superkeys in R = {AD, ~~AD~~ ADB, ADC, ADBC}

Q.



{Sid → Sname
Sid → DOB
sid →→ Phno
sid →→ email
}

• 1NF design: stud (Sid Phno email Sname DOB)

CK = {Sid Phno Email}

{Sid → Sname DOB, Sid →→ Phno, Sid →→ email, }

Not in 2NF ✓ ✓

* [Min 1 relational tables]

• 2NF Design:

Stud1 (Sid ~~Phno~~ Sname DOB)

[3NF BCNF]

Sid → Sname DOB

Super Key

CK = {Sid}

• 4NF also

Stud2 (Sid Phno email)

Sid →→ Phno
Sid →→ email
} Non Trivial MVDs [3NF BCNF]

CK = {Sid Phno email}

• Not in 4NF

* [Min 2 relational tables]

• 4NF design:

Stud1 (Sid Sname DOB)
Sid → Sname DOB
CK = {Sid}

Stud2 (Sid Phno)
Sid →→ Phno
CK = {Sid Phno}
Trivial MVD

Stud3 (Sid email)
Sid →→ email
CK = {Sid email}
Trivial MVD

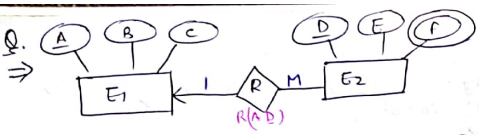
* [Min 3 relational tables]

NOTE

• If BCNF rel R and atleast two MVA stored in R by converting in SVA.
Then, R NOT in 4NF.

• If R is in BCNF and atmost one MVA stored in R by converting in SVA.
Then, R also in 4NF.

Q. ⇒



(i) Min rel tables which satisfy 1NF ? 2

(ii) Min rel tables which satisfy 2NF ? 3

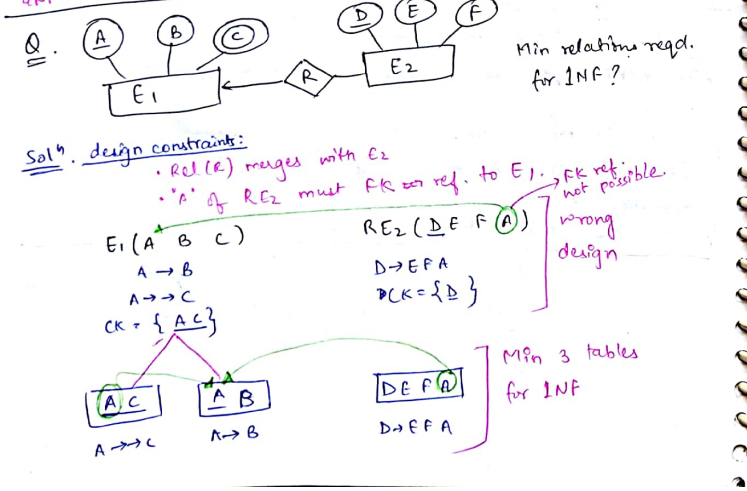
Sol^n. Rel (R) merges with E2

(and) in RE_2 attribute "A" must FK ref. to E1.

E_1 (A B C)
$A \to BC$
CK = {A}
[4NF]

RE_2 (D E F A)
$D \to E$, $D \to A$
CK = {D F}
[Not in 2NF]

Min 3 relational tables for
2NF ✓
3NF ✓
BCNF ✓
4NF ✓



Q.

Min relations reqd. for 1NF ?

Sol^n. design constraints:

- Rel (R) merges with E_2
- "A" of RE_2 must FK ref. to E_1.

FK ref. not possible.

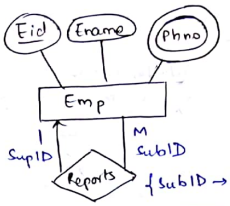
E_1 (A B C)
$A \to B$
$A \to C$
CK = {A C}

RE_2 (D E F A)
$D \to EFA$
PCK = {D}

wrong design

Min 3 tables for 1NF

Q. Eid Ename Phno



. Min^m rel. tables for 1NF.

FK ref not possible

Emp Reports (Eid , Ename, Phno, SupID)
eid → ename
eid → supID
eid →→ phno
CK = {eid phno}

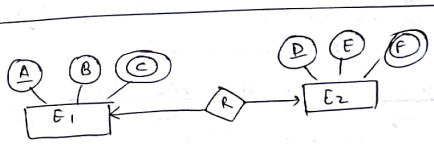
FK ref allowed

Emp$_1$ (Eid Phno)
eid →→ Phno

Emp$_2$ Reports (Eid Ename SupID)
eid → ename supID
CK = {eid}

* Min = 2 rel tables for 1NF, 2NF, 3NF, ~~1NF~~ BCNF, 4NF.

Q.



Take different combinations, and try.

Q ERD → RDBMS
Default NF ?

(a) 1NF ✓ (b) 2NF (c) 3NF (d) ~~1NF~~ BCNF

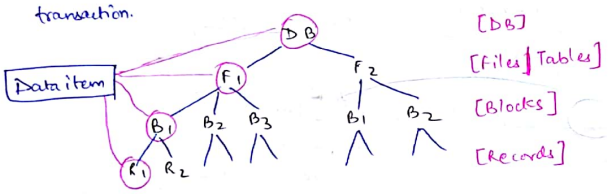
TRANSACTION AND CONCURRENCY CONTROL

→ Transaction: set of logically related R/W operations to perform a unit of work.

OS	DBMS
↓	↓
Process	Transaction

→ Data item: [Shared resource]

- DB element which is required to be accessed by many transaction.

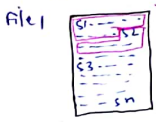


Degree of concurrency:
[# of users that can use DB simultaneously (concurrently).]

Flat File System [OS File System]

⇓

(File: resource)

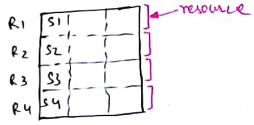


U1: Update S1
U2: Update S2
} Concurrent exe. of U1 U2 requests not allowed.

- less degree of concurrency using flat file system because concurrency control over file level.

DBMS File System

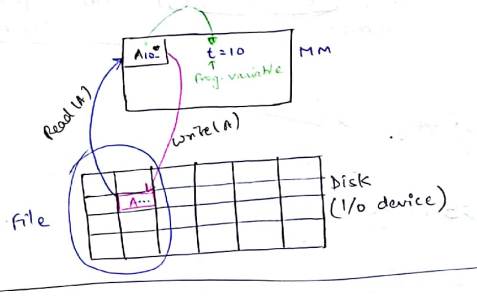
record : resource



- More degree of concurrency using DBMS file system because concurrency control over record level.

Main operations in transaction: A: data item

* Read (A): Access data item 'A' from DB file (disk) to programmed variable (MM) in order to use current value of "A" in transaction logic.
* Write (A): Modification of data item "A" in DB file.



Eg: Account (Cid bal)

Trans (T_1): Trans 500 from cust C_1 to C_2.

SQL Queries:

```
begin Trans T1
    Update Account set bal = bal - 500
    where Cid = C1;
    Update Account set bal = bal + 500
    where Cid = C2;
end Trans T2;
```

⇓ SQL Parser execution

```
Trans (T1)
    Read (bal of C1)        A
    A = A - 500;            A
    write (bal of C1)       A → B
    Read (bal of C2)
    B = B + 500;            B
    write (bal of C2)
commit;
```

Trans T_2 : set bal of C_1 C_2 as 10,000 ;

```
Update Account set bal = 10000 where Cid = C1
Update Account set bal = 10,000 where Cid = C2
```

↓

Trans (T_2)
Write (bal = 10000 of C_1)
write (bal = 10000 of C_2)
commit;

Blind Write
Trans. writes the data irrespective of previous value.

Eg. Trans (T_3)
R(A) ← select query.
R(B)
W(B)
W(C) ← Blind writes
W(D)
commit;

ACID Properties :

→ To preserve integrity (correctness), each transaction of DBMS must satisfy/preserve ACID properties.

DBMS	A : Atomicity	Recovery management component of DBMS s/w takes care of Atomicity & Durability
DBMS	D : Durability	
DBMS	I : Isolation	Concurrency Control Component of DBMS s/w is responsible for Isolation
DBA (Proj)	C : Consistency	USER [DBA or DB developer] is responsible for consistency

* Atomicity :

→ (Execute all operations of transaction including commit)
(or)
(Execute none of the operations of transaction)
by the time of Transaction termination.

Trans (T_1) : Transfer 500 from A to B

Rollback / failure →
✓R(A)
✓A = A − 500
✓W(A)
R(B)
B = B + 500
W(B)
commit;

DB file
A : ~~3000~~ 2500
B : 2000

Because of atomicity failure integrity of data is failure.

⇓

Rollback [undo operations successful]

~~Atomicity~~ [Atomicity satisfied]

Recovery Management Component :

→ should roll back transaction if transaction failed anywhere before commit.

* Rollback / Abort : Undo modifications of DB file which are done by the failed transaction.

* Transaction Log : File maintained by Recovery Mgmt. Component in disk [SM] to record all activities of transaction.

Trans (T_1)
✓R(A)
✓W(A)
R(B)
W(B)
commit;

log file
T1 : begins
T1 : Read, A, 3000
T1 : Write, A, 3000, 2500
T1 : Read, B, 2000
T1 : Write, B, 2000, 2500

DB File

A : 3000 2500	01	B1
B : 2000 2500	01	B2
	0	B3

successful commit

Dirty Block : updated by uncommitted transaction

Dirty Block :

* Redo Operations :
- all modifications of DB file because of transaction committed.
- Clean log entries of committed transaction.

* Undo Operations :
- all modifications of DB file because of transaction failure (rollback).
- Clean log entries of transaction.

commit T_1 ⇒ redo of T_1
commit T_2 ⇒ redo of T_2
commit T_3 ⇒ redo of T_3
commit T_4 ⇒ redo of T_4

⇒ Because of many redo operations, the throughput of the system reduces,
To avoid this problem consistency checkpoints introduced.

E.g.

9:00 AM Checkpoint
9:05 AM Checkpoint
9:10 AM Checkpoint
All committed transactions until previous checkpoint is performs redo.
• All uncommitted transactions remains as it is.

9:14 AM failure ⇒ ① ~~All committed transactions~~
②

⇒ If checkpoint issued, all committed transactions until previous checkpoint should perform redo operation and clean log entries of committed transaction.

⇒ If system failure happens, required operations to recover are:
① All committed transactions until previous checkpoint will perform redo.
② All uncommitted transactions in entire system will perform undo.
③ Clean log entries.

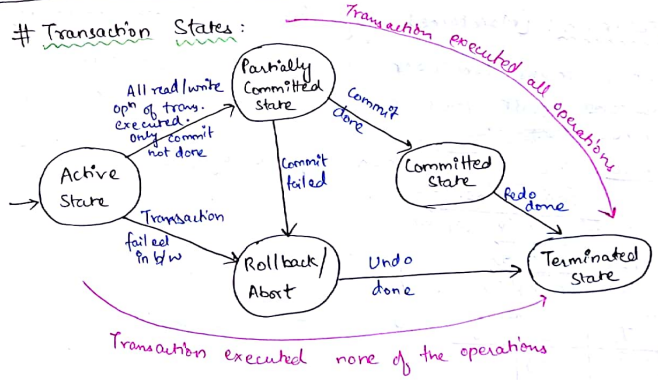
Q. Consider log file entries:

1. T_1 begin
2. T_1 W, A, 10, 20
3. T_2 begin
4. T_2 W, B, 0, 5
5. T_2 commit
6. T_3 begin
7. T_3 W, B, 5, 10
8. checkpoint
9. T_4 begin
10. T_5 begin
11. T_4 W, C, 0, 100
12. T_4 commit
13. failure occurs. T_4: redo
T_1 T_3 T_5 : undo

what are the required operations performed to recover from failure.

(a) T_2 T_4 redo, T_1, T_3 T_5 undo
(b) T_4 redo, T_1, T_3, T_5 undo
(c) T_1 T_2 T_3 T_4 T_5 undo
(d) T_2 redo, T_1 T_3 T_4 T_5 undo

Transaction States:



Active State: Transaction begins execution.

* Durability :

→ Transaction should be able to recover under any case of failure.

→ Transaction failure because of :

1. Power failure
2. s/w crash
   - OS restarted
   - DBMS restarted
3. OS/DBMS concurrency controller may kill transaction.
4. H/w Crash [E.g. disk crash]

if log file and states of each transaction maintained in disk → should be able rollback Durable

RAID Architecture of disk design must be used

RAID : 0 (PC)

RAID : 1 Mirror image of disk

*** Isolation : [Maintained by Concurrency Control Component]

→ Isolation means concurrent execution of two or more transactions result must be equal to result of some serial schedule.

* Schedule : Time order execution sequence of two or more transactions.

T_1 : $r_1(A)$ $r_1(B)$ $w_1(B)$
T_2 : $r_2(A)$ $r_2(B)$
T_3 : $w_3(A)$ $w_3(B)$

S:	T_1	T_2	T_3
1.		$r_2(A)$	
2.	$r_1(A)$		
3.			$w_3(A)$
4.	$r_1(B)$		
5.	$w_1(B)$		
6.		$r_2(B)$	
7.			$w_3(B)$

* Serial Schedule :

→ Transactions should execute one after another.
[After commit of current executing transaction allowed to begin other transaction]

E.g. T_1 : Transfer 500 from A to B.
T_2 : Display total balance of A, B.

A : 3000
B : 2000

S1 :

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
commit	
	$r_2(A)$ 2500
	$r_2(B)$ 2500
	display(A+B) 5000
	commit

S2 :

T_1	T_2
	$r_2(A)$ 3000
	$r_2(B)$ 2000
	display(A+B) 5000
	commit
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	
commit	

Guaranteed Correct

* Advantage of Serial Schedule :

→ Serial schedule guarantees correct result. [Integrity guaranteed].

* Disadvantage of serial schedule [If only transactions allowed to execute serially]

→ less degree of concurrency.
→ throughput of system also very less.

* Concurrent Schedule :

→ Transactions can execute concurrently / simultaneously / interleaved order.

E.g. S3 :

T_1	T_2
3000 $r_1(A)$	
2500 $w_1(A)$	
	$r_2(A)$ 2500
	$r_2(B)$ 2000
	display(A+B) 4500
	commit
$r_1(B)$	
$w_1(B)$	
commit	

Incorrect result [Incorrect schedule]

S4 :

T_1	T_2
3000 $r_1(A)$	
2500 $w_1(A)$	
	$r_2(A)$ 2500
2000 $r_1(B)$	
2500 $w_1(B)$	
commit	
	$r_2(B)$ 2500
	display(A+B) 5000
	commit

Correct result / Correct schedule

* both serial & non serial schedules are called concurrent schedules.

S_3: concurrent schedule

- Schedule NOT equal to $T_1 \rightarrow T_2$ serial schedule.
- Schedule NOT equal to $T_2 \rightarrow T_1$ serial schedule.

Isolation failed

S_4 : concurrent schedule

- Schedule is equal to $T_1 \rightarrow T_2$ schedule.

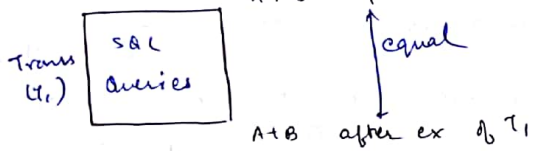
[Isolation satisfied]

14/11/2017

* Consistency: [USER responsible for consistency]

→ DB operations requested by user [SQL Queries | Transaction operations] must be logically correct.

T_1 : Trans 500 from A to B



A, D] Recovery Manager
I] Concurrency Controller
C] USER

} Responsible for integrity [correctness of Trans. result].

Concurrency Control: Method to avoid incorrect result because of concurrent execution.

Schedules Classification

Serializeability
[Isolation must satisfy]

(a) Conflict serializable schedule
(b) View serializable schedule

Serializability Testing

Recoverable Schedule
[Atomicity & Durability must satisfy]

other problem of concurrent execution.

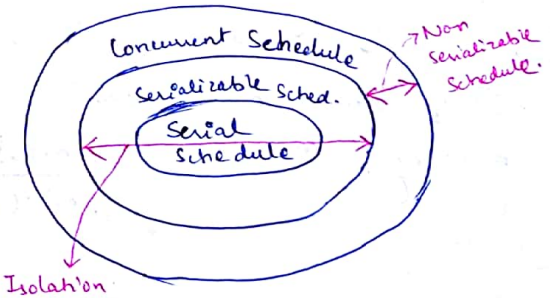
1. Irrecoverable schedule
2. Cascading roll back problem
3. Lost update problem

To avoid above problem:

1. Recoverable schedule
2. Cascadeless Rollback Schedule
3. Strict recoverable schedule.

Serializable Schedule:

Schedule(s) is/are serializable iff some serial schedule(s) equal to schedule(s).



Q. $T_1, T_2, \ldots, T_n$.

# of serial schedule possible : $n!$

Q. T_1: ~~r~~ $r_1(A)$ $w_1(A)$ $r_1(B)$ $w_1(B)$

T_2: $r_2(A)$ $r_2(B)$

① # of concurrent schedules b/w T_1 & T_2.

	T_1	T_2
1.		
2.		
3.		
4.		
5.		
6.		

$^6C_4 . {}^2C_2$

$= 15$ concurrent schedules

② # of non serial schedules b/w T_1 & T_2.

[# of concurrent schedules] − [# of serial schedules] $= 15 - 2! = 13$.

Q3 Transactions T_1, T_2 with n, m operations each.

# of concurrent schedules $= {}^{n+m}C_{n \text{ or } m} = \frac{(n+m)!}{n!\,m!}$

Q4. T_1 T_2 T_3 Transactions with n, m, p operations each.

# of concurrent schedules.

$${}^{(n+m+p)}C_n \cdot {}^{(m+p)}C_m \equiv \frac{(n+m+p)!}{n!\,m!\,p!}$$

Conflict Serializable Schedule:

* ① Topological order
② Conflict pair
③ Precedence graph
* ④ Conflict equal schedule.

* Topological order: [Greedy Method Algo]

→ Graph Traversal algo only for Directed Acyclic Graph (DAG).

① Visit vertex (v) whose In-degree "0"

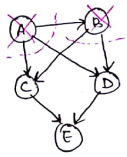
(and)

Delete visited vertex (v) from graph.

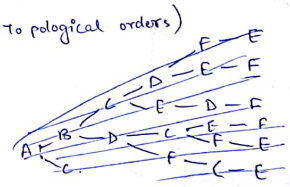
② Repeat ① for all vertices of graph.

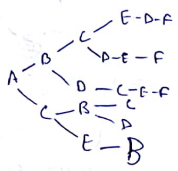
Topological Order: visited order of vertex

E.g.1.

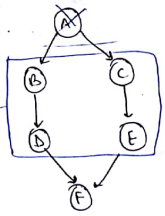


A; B — C:D:E / D:C:E (2 Topological orders)



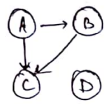


E.g.2.



A B _ D _ F

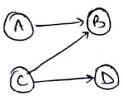
$^4C_2 \cdot {}^2C_2 = 6$

Eg. 

- A : B : C -
D

$^4C_1 = 4$

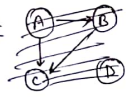
# of Topological orders :

E.g. 

A : C — B — D
D — B

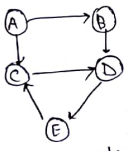
C — A — B — D
D — A — B

# of topological orders:

Eg 

# of Topological order of null graph with n vertices. [0 edges]
n!

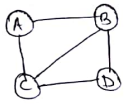
E.g. Cyclic Graph



# of topological orders 0

[No possible topological order for cyclic graph/ undirected graph]

Undirected Graph



# of topological orders = 0

Conflict Pair :

→ Pair of operations in schedule(s) are called conflict pairs iff (i) Atleast one write operation &
(ii) over same data item &
(iii) from different transactions

S: $r_i(A) \ldots\ldots w_j(A)$
S: $w_i(A) \ldots\ldots r_j(A)$
S: $w_i(A) \ldots\ldots w_j(A)$
} Conflict pairs

S: $r_i(A) \ldots\ldots r_j(A)$
S: $r_i(A)/w_i(A) \ldots\ldots r_j(B)/w_j(B)$
} Non Conflict pairs

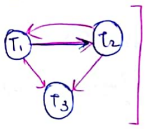
S: $r_1(B)\ r_1(A)\ w_1(A)\ w_2(B)\ w_3(B)$

— Non conflict pair
— Conflict pair

* Precedence Graph :

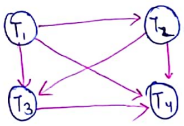
Vertices (V) : Transactions of stud
Edges (E) : Conflict pair precedences.

① S: $r_1(A)\ r_2(B)\ w_2(A)\ w_1(A)\ w_2(B)\ w_3(B)\ w_3(A)$



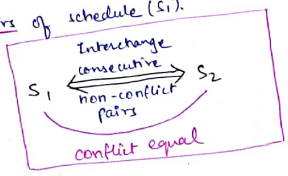
Precedence Graph of the schedule.

② S: $r_1(A)\ r_2(A)\ r_3(A)\ r_4(A)\ w_1(B)\ w_2(B)\ w_3(B)\ w_4(B)$



Conflict Equal Schedules :

→ S_1 S_2 schedules are conflict equal iff S_2 derived by interchange of consecutive non-conflict pairs of schedule (S_1).



Eg. S_1:

T_1	T_2
$r_1(A)$	
$w_1(A)$	
	$r_2(A)$
$r_1(B)$	
$w_1(B)$	
	$r_2(B)$

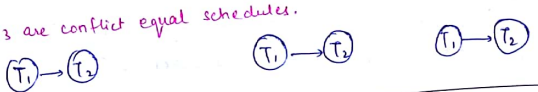
S_2:

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(A)$	
~~$r_1(B)$~~	
~~$w_1(A)$~~	$r_2(A)$
$w_1(B)$	
	$r_2(B)$

S_3:

T_1	T_2
$r_1(A)$	
$w_1(A)$	
$r_1(A)$	
$w_1(B)$	
	$r_2(A)$
	$r_2(B)$

S_1, S_2, S_3 are conflict equal schedules.



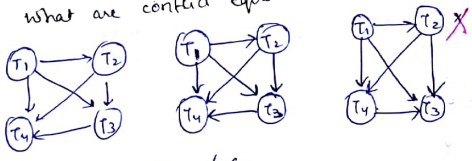
Eg. S_1: $r_1(A)$ $r_2(A)$ $r_3(A)$ $r_4(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$ $w_4(B)$

~~S_2: $r_2(A)$ $w_2(B)$ $r_1(A)$ $r_4(A)$~~

S_2: $r_2(A)$ $r_1(A)$ $w_1(B)$ $w_2(B)$ $r_3(A)$ $w_3(B)$ $r_4(A)$ $w_4(B)$

S_3: $r_3(A)$ $r_4(A)$ $r_1(A)$ $w_1(B)$ $r_2(A)$ $w_2(B)$ $r_4(A)$ $w_4(B)$ $w_3(B)$

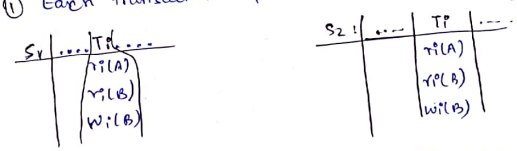
What are conflict equal schedules?



$S_1 = S_2 \neq S_3$

* S_1, S_2 schedules are conflict equal, iff:

① Each transaction of S_1 must be exactly same in S_2.



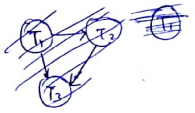
② Every conflict pair precedence of S_1 S_2 must be same precedence.

Pg.68.

Q.7. S_1: $r_2(A)$ $w_2(A)$ $r_3(C)$ $w_2(B)$ $w_3(A)$ $r_1(A)$ $r_1(B)$ $w_1(A)$ $w_1(B)$ $w_3(C)$

S_2: $r_3(C)$ $r_2(A)$ $w_2(A)$ $w_2(B)$ $w_3(A)$ $r_1(A)$ $r_1(B)$ $w_1(A)$ $w_1(B)$ $w_3(C)$

S_3: $r_2(A)$ $r_3(C)$ $w_3(A)$ $w_2(A)$ $w_2(B)$ $w_3(C)$ $r_1(A)$ $r_1(B)$ $w_1(A)$ $w_1(B)$



* If S_1, S_2 are conflict equal schedules, the precedence graph of S_1, S_2 must be same.

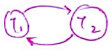
* But if S_1, S_2 schedules with same precedence graph, then, S_1, S_2 may/may not be conflict equal.

Eg. S_1:

T_1	T_2
$r_1(A)$	
	$w_2(A)$
$w_1(A)$	
	$r_2(A)$

S_2:

T_1	T_2
$r_1(A)$	
	$w_2(A)$
	$r_2(A)$
$w_1(A)$	



S_1, S_2 not conflict equal.

T_1	T_2
$r_1(A)$	
$r_1(B)$	
	$w_2(B)$

$T_1 \rightarrow T_2$

T_1	T_2
$r_1(B)$	
$r_1(A)$	
	$w_2(B)$

$T_1 \rightarrow T_2$

S_1, S_2 not conflict equal.

Conflict Serializable Schedule:

→ ~~Such~~ Schedule "s" is conflict serializable iff:

some serial schedule (s') conflict equal to schedule (s)

[S-schedule] ⟺ (conflict equal) [s': serial schedule]

S is conflict serializable

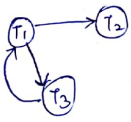
[Precedence Graph of schedule (S) also must be "Acyclic"] ⟺ [Precedence graph of serial schedule (s') is Acyclic.]

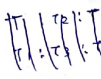
****

* Testing Condition of conflict serializable schedule:

→ Schedule(S) is conflict serializable schedule iff precedence graph of schedule "s" is Acyclic. & conflict equal serial schedules are topological orders of precedence graph.

E.g. S: $r_1(A)\ w_1(B)\ r_2(B)\ r_3(C)\ w_3(C)\ w_3(A)\ w_1(C)$

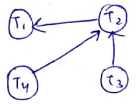




Schedule (S) is not conflict serializable schedule.

[No serial schedule conflict equal to S]

Q. S: $r_2(A)\ w_1(A)\ r_3(B)\ w_2(B)\ r_4(C)\ w_2(C)$



$T4 : T3 : T2 : T1$
$T3 : T4 : T2 : T1$

~~Schedule "s" conflict~~ ~~seri~~ conflict equal serial Schedules.

Schedule "s" is conflict serializable schedule.

{# of serial schedules conflict equal to schedule(s)} = {No. of topological order of ~~#~~ schedule "s" precedence graph}

View Serializable Schedule:

→ Schedule "s" view serializable iff:

some serial schedule "s'" view equal to schedule "s".

⇒ S_1, S_2 view equal iff:

① Initial Read:

S_1: .... (T_i .... R(A)

S_2: .... (T_i .... R(A)

→ Every initial read of S_1 must be also initial read in S_2.

E.g. S_1:

T_1	T_2	T_3
$r_1(A)$		
	$r_2(A)$	
	$r_2(B)$	
		$w_3(B)$

S_2:

T_1	T_2	T_3
$r_1(A)$		
	$r_2(B)$	
	$r_2(C)$	
		$w_3(B)$

S_1 & S_2 are not view equal.

② Updated Read:

S_1: .... T_i .... T_j ....

W(A) → R(A)

S_2: .... T_i .... T_j ....

W(A) → R(A)

→ Every updated read of S_1 must be same updated ~~read~~ read in S_2.

E.g: S_1:

T_1	T_2	T_3
W(A)		
	W(A)	
		R(A)

S_2:

T_1	T_2	T_3
W(A)		
		R(A)
	W(A)	

S_1 & S_2 are not view equal.

③ Final Write:

S_1: .... T_i ....

Final write of A → W(A)

S_2: .... T_i ....

Final write of A → W(A)

→ Every final write of S_1 must be same in S_2

E.g: S_1:

T_1	T_2	T_3
W(B)		
	W(B)	
		W(B) ←

S_2:

T_1	T_2	T_3
	W(B)	
W(B)		
		W(B) ←

Final write condition satisfied.

E.g① S_1:

T_1	T_2	T_3
R(A)		
R(B)		
	W(A)	
W(A)		
W(B)		
		R(A)
		W(A)

S_2:

T_1	T_2	T_3
	R(A)	
	R(B)	
	W(A)	
	W(B)	
W(A)		
		R(A)
W(B)		
		W(B)

~~View~~

S_1 & S_2 are view equal.

S_1 & S_2 are not conflict equal

E.g② S_1: $W_1(A)\ W_2(A)\ W_3(A)\ R_4(A)$

S_2: $W_2(A)\ W_1(A)\ W_3(A)\ R_4(A)$

~~view equal~~ S_1 & S_2 are not conflict equal.

S_1 & S_2 are view equal.

NOTE

→ If S_1, S_2 are conflict equal, then, S_1, S_2 are also view equal.

→ If S_1, S_2 schedules are not conflict equal, then, S_1, S_2 schedules may/may not be view equal.

→ If schedule "s" is conflict serializable schedule, then, "s" is also view serializable schedule.

→ If schedule "s" is not conflict serializable schedule, then, schedule "s" may/may not be view serializable.

→ Conflict serializable schedule testing condition is only sufficient but not necessary for serializeability testing.

→ View seriable schedule testing condition is both sufficient and necessary for serializability testing.

[Sufficient but not necessary]
[Only if] [⟶ implied]

Testing condition
of conflict serializable

Precedence graph of schedule "S" is acyclic.

⟶ Implied

Schedule (S) Ps serializable schedule

P. Problem $O(n^2)$

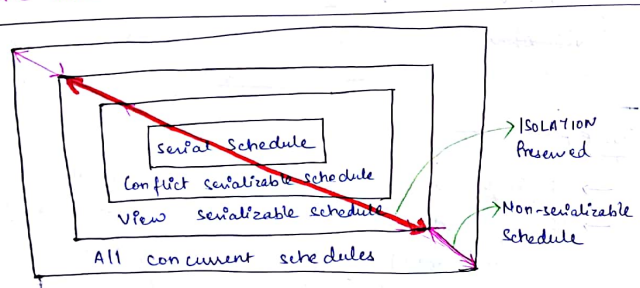
[Sufficient and necessary]
[If and only if] [⟷ bi implication]

Stmt 1
Schedule "S" satisfied view serializable testing condition

⟷

Stmt 2
Schedule (S) serializable schedule

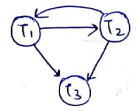
NPC Problem $O(2^n)$



Q Test given schedule is view serializable or not:

① S:

T_1	T_2	T_3
$r_1(A)$		
	$w_2(A)$	
$w_1(A)$		
		$w_3(A)$

② 

"S" is not conflict serializable.

③ View serializable:

Given schedule

Final write A: T_1 T_2 T_3

Initial read:

DATA item	Initial Read	write
A	T_1	T_1 T_2 T_3

Serial Schedule

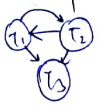
$(T_1, T_2) \rightarrow T_3$

$T_1 \rightarrow (T_2\ T_3)$

Schedule "S" view equal to $T_1 : T_2 : T_3$ serial schedule.

②

T_1	T_2	T_3
	$r(A)$	
	$r(B)$	
	$w(A)$	
$w(A)$		
$w(B)$		
	$w(B)$	
		$R(A)$
		$w(B)$



✓ View serializable:

	Given schedule		Serial Schedule
① Final write	A: T_1 T_2 T_1		$T_2 \rightarrow T_1$
	B: T_1 T_2 T_3		$(T_1, T_2) \rightarrow T_3$
② Initial Read	D.I.	I.R.	write
	A	T_2	T_2 T_1
	B	T_2	T_1 T_2 T_3
			$T_2 \rightarrow T_1$
			$T_2 \rightarrow (T_1, T_3)$
③ Updated-write read	$W_1(A) \rightarrow R_3(A)$ & T_2 other trans. also writes "A"		$T_1 \rightarrow T_2 \rightarrow T_3$

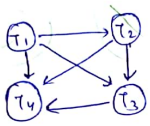
$T_2 : T_1 : T_3$ serial schedule view ≠ S

(3) S: $r_1(A)\, r_2(A)\, r_3(A)\, r_4(A)\, w_1(B)\, w_2(B)\, w_3(B)\, w_4(B)$

(i) # of serial schedules conflict equal to schedule "S". 1

(ii) # of ~~view~~ serial schedules view equal to schedule "S". 6

(i)



$T_1 : T_2 : T_3 : T_4$

# of conflict equal serial schedules = 1.

(ii)

	Given schedule	Serial schedule
Final write ⇒	B: $T_1\, T_2\, T_3\, T_4$	$(T_1\, T_2\, T_3) \rightarrow T_4$
Initial Read ⇒	DI: A — $T_1\, T_2\, T_3\, T_4$; B — $T_1\, T_2\, T_3\, T_4$	$(T_1\, T_2\, T_3\, T_4)$
Updated Read	—	$(T_1\, T_2\, T_3) \rightarrow T_4$
		3!. 2 6 ⇒ view equal serial schedules.

Pg. 61.

(T1) S3: $r_1(A)\, r_3(D)\, w_1(B)\, r_2(B)\, w_3(B)\, r_4(B)\, w_2(C)\, r_3(C)\, w_4(E)\, r_5(E)\, w_5(B)$

Given Schedule

Final write ⇒ A: ~~B~~ —; B: $T_1\ T_3$ (T_5); C: (T_2); D: —

Initial Read ⇒

DI	IR	write
A	T_1	—
B	~~B~~ —	$T_1\, T_3\, T_5$
C	—	T_2
D	T_3	—

Updated Read ⇒

W → R	other Transactions writes DI.
$W_1(B) \rightarrow r_2(B)$	$(T_3\, T_5)$
$W_2(C) \rightarrow r_5(C)$	
$W_4(E) \rightarrow r_5(E)$	
$W_3(B) \rightarrow r_4(B)$	$T_1\, T_5$

Serial Schedule

$(T_1\, T_3) \rightarrow T_5$

—

DI:	~~T_3~~ T_5
$T_1 \rightarrow T_2$	
$T_2 \rightarrow T_5$	
$T_4 \rightarrow T_5$	
$T_3 \rightarrow T_4$	

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5$

Updated Read	write → Read	other trans. writes DI.
	$W_1(B) \rightarrow r_2(B)$	$T_3\, T_5$
	$W_3(B) \rightarrow r_4(B)$	$T_1\, T_5$
	$W_2(C) \rightarrow r_5(C)$	—
	$W_4(E) \rightarrow r_5(E)$	—

$T_1 \rightarrow T_2$ ($T_3\, T_5$); $T_3 \rightarrow T_4$ ($T_1\, T_5$); $T_2 \rightarrow T_5$; $T_4 \rightarrow T_5$

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5$

$T_3 \rightarrow T_4 \rightarrow T_1 \rightarrow T_2 \rightarrow T_5$

Choose max possible trans. in one seq.

(T_3): T_1 ✓, T_2 ×, T_5 ✓, T_4 ×

(ii) ~~Find # of conflict equal serial schedules.~~

(T_4): T_3 ×, T_1 ✓, T_2 ×, T_5 ×

$T_3\, T_4\, T_1\, T_2\, T_5$

$T_1\, T_2\, T_3\, T_5$ ×

$T_1\, T_2\, T_3\, T_4\, T_5$

(2)

(ii') Find # of conflict equal serial schedules. 1

Classification based on Recoverability:

→ ~~for~~ Concurrent execution may lead to
① IRRECOVERABLE Schedule
② Cascading Roll back problem
③ Lost Update Problem
} These problems can occur even if the schedule is serializable.

IRRECOVERABLE Schedule:
[Rollbacking committed transaction]

→ Schedule "s" is irrecoverable iff:
Transaction T_j reads data item "x" which is updated by T_i
Commit of T_j before commit/rollback of T_i.

S:

T_i	T_j
:	
W(x)	
	R(x)
	commit
commit/ rollback	

Irrecoverable Schedule

T_1	T_2
1. W(A)	
2.	R(A)
3.	C2
4. C1	

Basic structure of irrecoverable schedule.

E.g: T_1: Withdraw 5000 from A [$r_1(A)$ A:=A−5000 $w(A)$ commit]
T_2: Check balance of A [$r_2(A)$ commit]

T_1	T_2
r(A)	
A=A−5000	
w(A)	
	r(A)
	commit
failed	

A: 10000 → 5000

IRRECOVERABLE Schedule

* Trans. T_2 must rollback because of T_1 failure but rollback of T_2 not possible because T_2 already committed.

Recoverable Schedule:

→ Schedule "s" is recoverable iff:
① No uncommitted read (dirty read) in "s".
(or) ② If transaction T_j reads data item "x" which is updated by transaction T_i then, commit of T_j must delay until, ~~com~~ commit/rollbacks of transaction T_i.

T_i	T_j
1. W(x)	
2.	R(x) ← Dirty Read
3. commit/	
~~4.~~ rollback	
4.	commit

* Uncommitted Read [Dirty Read]:
→ Transaction T_j reads data item "x" which is updated by uncommitted transaction transaction T_i.

T_i	T_j
W(x)	
	R(x) ← Dirty Read.
C/R	

Q. Test which of the schedules are recoverable?

① Recoverable

T_1	T_2
W(A)	
	R(A)
	Rollback
Commit	

- Recoverable
- Not cascadeless Rollback
- Not strict recoverable

② Not strict recoverable R

T_1	T_2
R(A)	
	R(A)
	W(A)
W(A)	
	C2
C1	

Recoverable
No dirty reads.
Cascadeless Rollback

③ Not SR.

T_1	T_2
W(A)	
	W(A)
	W(B)
W(B)	
	C2
C1	

Recoverable
No dirty reads.
C.L. Rollback

④ Recoverable

T_1	T_2
W(A)	
	R(A)
	R(B)
W(B)	
C1	
	C2

NOT C.L. Rollback
Not SR.

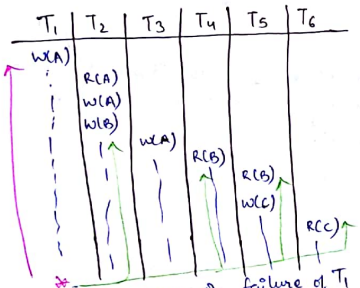
⑤

T_1	T_2
W(A)	
	W(A)
	R(A)
	C2
C1	

- Recoverable
- No dirty read.
- CL Rollback
- Not S.R.

Cascading Rollback Schedule (Problem)

→ Because of failure of some transaction forced to rollback some set of other transactions.



* Disadvantage of cascading rollback:

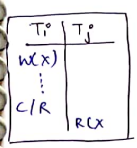
① Wastage of CPU execution time.

② Wastage of I/O access cost.

Solution ⇒ Cascadeless Rollback Recoverable Schedule

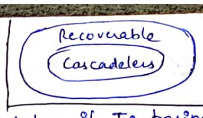
Cascadeless Rollback recoverable Schedule:

[No dirty reads in schedule]

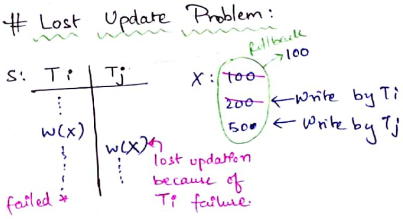


→ Transaction T_j read(x) should delay if T_j trying to read "x" which is updated by T_i until T_i commit/rollback.

[Dirty Reads are not allowed]



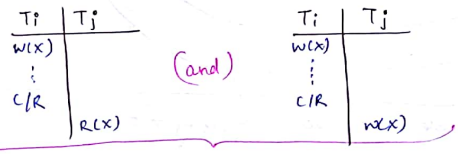
Lost Update Problem:

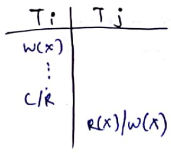


→ Can occur if T_j writes "X" which is already written by uncommitted transaction T_i.

Strict Recoverable Schedule:

{Cascadeless Rollback, Recoverable Schedule} and {If T_i writes "X", then other transaction T_j write of "X" must delay until T_i commit/rollback} → No lost update problem





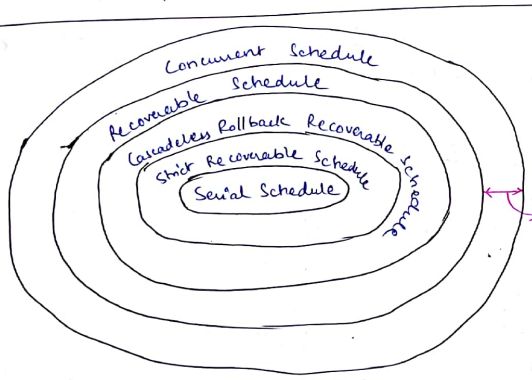
Strict recoverable Schedule:

→ If T_i writes data item "x", then Read(x) / write(x) of Trans T_j must delay until commit / rollback of T_i

Eg:

T_1	T_2	T_3
R(A)		
	W(A)	
R(B)		
	W(B)	
	commit	
W(A)		
commit		
		W(A)
		commit

Strict recoverable schedule.



* Strict recoverable schedule may/may not be serializable.
* ~~Strict~~ Serializable schedule may/may not be recoverable.

Pg-61.

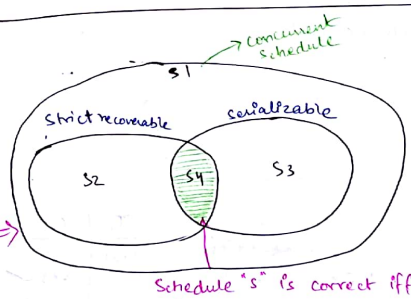
T_2. S2: Initial read no validation

S: $r_1(x)\ r_2(z)\ r_1(z)\ r_3(x)\ r_3(y)\ w_1(x)\ w_3(y)\ r_2(y)\ w_2(z)\ w_2(y)\ c_1 c_2 c_3$

Which is true?

(a) Irrecoverable
(b) Recoverable but not CL Rec.
(c) CL Rec. but not strict recoverable
(d) Strict rec.

T_1	T_2	T_3
r(x)		
	r(z)	
r(z)		
		r(x)
		r(y)
w(x)		
		w(y)
	r(y)	→ Dirty read
	w(z)	
	w(y)	
c_1		
	c_2	
		c_3



Schedule "S" is correct iff:
"S" is strict recoverable
& "S" is serializable.

15/11/2017:

Concurrency Control Protocols:

→ Concurrency control protocol should not allow to execute
. non-serializable schedule (or) non-strict recoverable schedule.

Locking Protocol:

→ Lock is a variable used to identify status of data item.

Trans (T_1)
lock(A) ← granted by concurrency controller
R(A)
W(A)
lock(B) ← denied by C.C.
⋮
lock(B) ← ~~denied~~ granted by C.C.
R(B)
unlock(A)
unlock(B)

* Shared Lock [S]: Read only locks.
* Exclusive Locks [X]: Read/write locks

Trans (T_1)
S(A) ← granted
R(A)
X(B) ← granted
R(B)
W(B)

x Lock Compatible Table:

data item "A"	Held by T_i: S	X
Requested by T_j: S	Yes	No
X	No	No

T_1	T_2
S(A)	S(A) ✓
R(B)	R(A)
S(B)	X(B) ✗
R(B)	

T_1	T_2
X(A)	
W(A)	
U(A)	
	S(A)
	R(A)
	U(A)
	S(B)
	R(B)
	U(B)
X(B)	
W(B)	
U(B)	
	C2
C1	

Serializable (2PL)

- Non serializable schedule
- Not recoverable schedule
- s/x locks not sufficient to avoid non-serializable (or) non-strict recoverable

$W_1(A) \rightarrow R_2(A)$: $T_1 \rightarrow T_2$
i.e. T_2 reads, T_1 writes
B: $T_2 \rightarrow T_1$ ∴ $T_2 \rightarrow T_1$ → Non serializable.

Two-phase locking (2PL): [Guaranteed Serializability]

Definition: Transaction 'T' allowed to request locks on any data item in any mode (S/X) until first unlock of transaction (T).

Trans (T).

S(A) — Growing Phase
X(B)
S(C)
X(D) * ← lock point of Trans(T) [position of last lock / first unlock of Trans T]
First unlock of T → U(C) *
U(D) — Shrinking Phase
U(A)
U(B)

Eg.

T_1: Transfer 500 from A to B
T_1
X(A)
R(A)
W(A)
X(B)
R(B)
W(B)
U(A)
U(B)
commit

T_2: display total A,B balance
T_2
S(A)
R(A)
S(B)
R(B)
U(A)
U(B)
disp(A+B)
commit

T_3: Set A,B as 2000
T_3
X(A)
W(A)
X(B)
W(B)
U(A)
U(B)
commit

Executes concurrently
concurrent ex of T_1, T_2, T_3 guaranteed serializable

T_1	T_2
X(A)	
R(A)	
W(A)	
X(B)	
U(A)	
	S(A)
	R(A)
	S(B) ← denied
	R(B)
R(B)	
W(A)	
U(B)	

Not allowed by 2PL

⇒

T_1	T_2
X(A)	
R(A)	
W(A)	
W(B)	
U(A)	
	S(A)
	R(A)
R(B)	
W(B)	
U(B)	
	S(B)
	R(B)

$T_1 \rightarrow T_2$ conflict serializable allowed by 2PL

⇒ If schedule executed by 2PL, the schedule is guaranteed to be conflict serializable schedule and

- conflict equal serial schedule is based on lockpoints order of transactions.

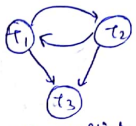
S: T_1 T_2 T_3

$T_2 \rightarrow T_1 \rightarrow T_3$

⇒ If schedule is not conflict serializable schedule [cyclic precedence graph schedule], then, schedule not allowed to execute by 2PL.

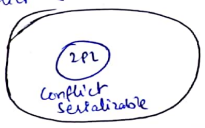
S: T_1 T_2 T_3

R(A) — X(A) denied — W(A) — W(B) — U(A) — W(A)



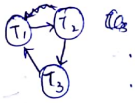
① Not conflict serializable
② Not allowed by 2PL
③ View serializable

⇒ Every schedule which is allowed by 2PL is always conflict serializable schedule but not every conflict serializable schedule is allowed by 2PL.



Q. S: T_1 T_2 T_3

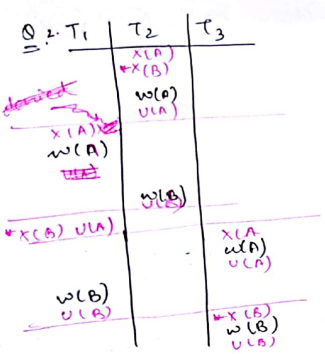
S(A) R(A) X(C) U(A) U(C) — X(A) denied — W(A) X(B) W(B) U(A) U(B) — S(A) R(B) S(C) denied R(C) — W(C) U(C)

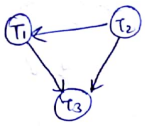


- Not CSS
- Not allowed by 2PL

(a) conflict ss && allowed by 2PL
(b) conflict ss but not allowed by 2PL
(c) not conflict ss and not allowed by 2PL
(d) not CSS but view serializable.

Q2. T_1 | T_2 | T_3



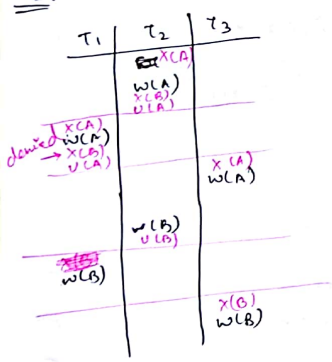


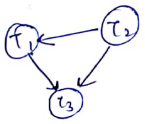
(b) ~~Not conflict but view serializable~~

(a) CSS & allowed by 2PL

Equal serial schedule = $T_2 \rightarrow T_1 \rightarrow T_3$

Q3. $W_2(A)\ W_1(A)\ W_3(A)\ W_2(B)\ W_1(B)\ W_3(B)$





(b) CSS but not allowed by 2PL.

Lock upgrading 2PL Vs. without lock upgrade 2PL:

* In exam, take lock upgrading by default.

2PL w/o lock upgrade

S: T_1

X(A)
R(A)

No other transaction allowed to use "A" in shared mode

W(A)
U(A)

→ Less degree of concurrency.

T_1	T_2
X(A)	
R(A)	
	S(A) ← denied
	R(A)
W(A)	
U(A)	

* Serializable
* Not allowed by 2PL w/o lock upgrade

2PL with lock upgrading

S: T_1

S(A)
R(A) — other trans. can use "A" in shared mode

→ X(A) (update W(A))
U(A)

* lock upgrade of trans(T) must complete in growing phase

→ More degree of concurrency.

T_1	T_2
S(A)	
R(A)	
	S(A) ← granted.
	R(A)
	U(A)
X(A)	
W(A)	
U(A)	

* Allowed to execute by 2PL.

Q. Test, given schedule allowed by 2PL (or) not.

(1) S: $r_1(A)\ w_1(A)\ r_2(A)\ r_2(B)\ r_1(B)\ w_1(B)$

T_1	T_2
S(A)	
R(A)	
X(A)	
W(A)	
S(B) X(B)	
U(A)	
	S(A)
	R(A)
	S(B) ← denied
	R(B)
r(B)	
w(B)	

NOT allowed

② S: $r_1(A)\ r_2(A)\ r_2(B)\ w_1(B)\ w_1(A)$

T_1	T_2
S(A) r(A)	
	S(A) r(A)
	S(B) r(B) U(B)
X(B) w(B) U(A) w(A) X(A)	

$T_1 \leftarrow T_2$

allowed.

③ S: $r_1(A)\ r_2(A)\ w_1(A)\ w_2(A)$

T_1	T_2
S(A) r(A)	
	S(A) r(A) ~~rate denied~~ ~~read~~ U(A)
X(A) w(A) U(A)	
	~~X(A) ← denied~~ ~~w(A) ← denied~~ w(A) U(A)

$T_1 \rightleftarrows T_2$

NOT allowed

Limitations of 2PL:

① 2PL restriction may lead to deadlock.

② 2PL restriction may lead to starvation.

③ 2PL condition not sufficient to avoid:

① Irrecoverable schedule

② Cascading Rollback problem

③ ~~Strict recoverable problem~~ Lost update problem

E.g. ① 2PL restriction may lead to deadlock.

T_1: $r_1(A)\ r_1(B)$

[S(A) $r_1(A)$ S(B) | U(A) $r_1(B)$ U(B)]

T_2: $w_2(B)\ w_2(A)$

[X(B) w_2 X(A) U(B) w_2 U(A)]

T_1	T_2
S(A) r(A)	
	X(B) w(B)
denied ↳ S(B) r(B)	
	X(A) w(A)

forms deadlock

Q. data items

$A_1\ A_2\ A_3 \ldots\ A_n$

Protocol:

① Trans should lock all required data item in proper sequence of $A_1, A_2 \ldots A_n$

② Trans. should perform read/write.

③ Trans. unlocks data

Which is true?

Protocol

(a) Guaranteed serializable and no deadlocks.

(b) Guaranteed serializable ~~and~~ but may cause deadlock.

(c) Not guaranteed serializable and free from deadlock.

(d) none.

E.g.

X(A_2)
X(A_3)
X(A_6)
⋮
w(A_3)
⋮
w(A_2)
w(A_6)
⋮
U(A_1)
U(A_2)
U(A_6)

→ 2PL
- guaranteed serializable
- No deadlock

Eg: (2) 2PL may form starvation.

T_1	T_2	T_3	T_4
	S(A)		
X(A) (denied because of T_2)	⋮	S(A)	
X(A) (denied because of T_3)	U(A)		SCA)
X(A) (denied because of T_4)		U(A)	

Eg: (3) 2 PL restriction not sufficient to avoid :
- irrecovable schedule
- Cascading RB
- Lost update problem.

T_1	T_2
X(A)	
W(A)	
X(B)	
U(A)	S(A)
	R(A)
W(B)	
U(B)	
	X(B)
	W(B)
	U(A)
	U(B)
	commit
commit	

- IRRECOVERABLE schedule
- Cascading RB possible
- Lost update problem possible.
- Allowed to execute be 2PL
- Conflict serializable

Strict recoverable condition ~~schedule~~ using locks:

T_1	T_2
X(A)	
W(A)	
⋮	
C/R	
U(A)	S(A)/X(A) R(A)/W(B)

All exclusive locks of Trans (T_1) should hold until C/R of transaction T_1.

Strict 2PL Protocol:

* Basic 2PL: Lock requests of transaction (T) not allowed in shrinking phase of transaction (T).

and

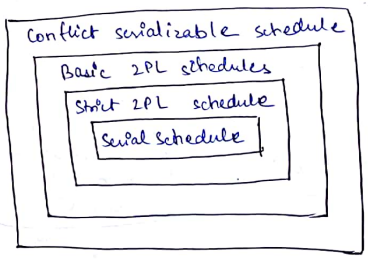
* Strict recoverable: All exclusive locks of transaction (T_j) must hold until C/R of transaction (T_i).

* Strict 2PL protocol:
  - guaranteed serializable
  and - guaranteed strict recoverable.

* Strict 2PL also not free from deadlock and starvation.

Trans (T):

S(A)	Growing Phase
X(B)	
S(C)	
X(D)	
U(A)	Shrinking Phase
U(C)	
commit	
U(B)	
U(D)	



Pg 61.
Q.23. T_1: $r_1(A)\ r_1(B)\ w_1(B)$
T_2: $r_2(A)\ r_2(B)\ w_2(B)$
} $^6C_3 = 20$ # of total schedules
# of non-serial serial schedules which are serializable ?

(i) Concurrent ex. of T_1 & T_2 equal to $T_1 \longrightarrow T_2$

S:	T_1	T_2
	$r_1(A)$	
	$r_1(B)$	
	$w_1(B)$	
		$r_2(A)$
		$r_2(B)$
		$w_2(B)$

$r_2(A)$

S: $r_1(A)\ r_1(B)\ w_1(B)\ r_2(B)\ w_2(A)$ — $r_2(A)$

4

Concurrent ex. of T_1 & T_2 equal to $T_2 \longrightarrow T_1$

S:	T_1	T_2
		$r_2(A)$
		$r_2(B)$
		$w_2(B)$
	$r_1(A)$	
	$r_1(B)$	
	$w_1(B)$	

S: $r_2(A)\ r_2(B)\ w_2(B)\ r_1(B)\ w_1(B)$ — $r_1(A)$

4

20 concurrent schedules
- 8 serializable
  - 2 serial sched
  - 6 non-serial schedules which are serializable
- 12 non-serializable

Q.24. S: $r_1(A)\ w_1(B)\ r_2(A)\ w_2(B)\ r_3(A)\ w_3(B)$

(i) # of schedules conflict equal to given schedule S:

[S: $r_1(A)\ w_1(B)\ r_2(A)\ w_2(B)\ r_3(A)\ w_3(B)$] $\Longleftrightarrow$ conflict equal

$r_1(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$

$r_2(A)$

$r_3(A)$

$=3$ $\quad =5$

$\therefore 3 * 5 = 15$

(ii) # of schedules view equal to schedule S:

S: $r_1(A)$ $w_1(B)$ $r_2(A)$ $w_2(B)$ $r_3(A)$ $w_3(B)$

$\Updownarrow$ view equal

*final write condition

$w_1(B)\ w_2(B)\ w_3(B)$ (or) $w_2(B)\ w_1(B)\ w_3(B)$

15 + 15

$= 30$

H.W.
Q. S: $w_1(A)\ w_1(B)\ r_2(A)\ w_2(B)\ r_3(A)\ w_3(B)$
(i) # of conflict equal schedules.
(ii) # of view equal schedules.

TUPLE RELATIONAL CALCULUS:

* First order logic and predicate calculus:

Q. Write FOL stmts for given specification using fn:

C(x): x studied in class.

S(x): x studied maths.

(i) Some student in class studied maths.

$\exists_x (C(x) \wedge S(x))$

Some 'x' such that x student in class && x studied maths.

(ii) Every student in class studied maths.

Correct $\forall x (C(x) \rightarrow S(x))$

for every 'x' if x student in class, then x must study maths.

wrong $\exists \forall x (C(x) \wedge S(x))$

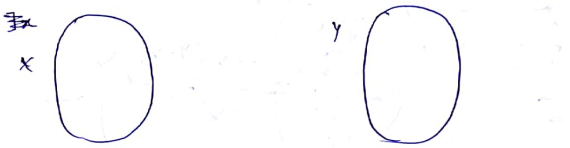
every 'x' must be student in class.

Eg: X = { A ✓ B ✓ C ✓ D ✗ E ✗ }

← Students in class → ← Not students in class →

A, B, C studied maths.

(iii) Atleast two students in class studied maths.



Some (x) student in class && studied maths

Some (y) student in class && studied maths.

$\neq$

$\exists x \left(C(x) \wedge S(x) \wedge \exists y (C(y) \wedge S(y) \wedge (x \neq y)) \right)$

$\exists x \exists y \left(C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y) \right)$

(iv) Only one student in class studied maths.

Method!

~~$\exists x (C(x) \wedge S(x) \wedge \forall y (\neg C(y) \vee \neg S(y) \vee (x = y)))$~~

$P \rightarrow Q \equiv P \wedge \neg Q$

= [atleast one student in class studied maths] $\exists x (C(x) \wedge S(x))$

but not

[atleast two students in class studied maths]

$\exists x \exists y (C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y))$

$= \exists x (C(x) \wedge S(x)) \wedge \neg \exists x \exists y (C(x) \wedge S(x) \wedge C(y) \wedge S(y) \wedge (x \neq y))$

Method 2

{ some x student in class and studied maths } (and) (every "y" who are students in class and studied maths) → { (y=x) }

$$\exists x \left(C(x) \wedge S(x) \wedge \forall y \left((C(y) \wedge S(y)) \rightarrow (y = x) \right) \right)$$

Format of TRC Query:

→ $\{ T \mid P(T) \}$

T: Tuple variable

P(T): formula over tuple variable.

Results set of tuples (T) those are satisfied formula P(T)

→ $\exists T_1 \in stud\ (P(T_1))$ and $\forall T_2 \in course\ (P(T_2))$ } T_1 T_2 are bounded tuple variable.

⇒ Tuple variable used for result of TRC query must be free tuple variable.

E.g. stud (Sid, age)
course (Cid, Instructor)
Enroll (Sid, Cid, Fee)

① Retrieve Sid's enrolled some course taught by Korth.

Soln. Enroll (Sid Cid ...) course (Cid Inst = Korth)

$\exists T_1$ { S_1 C_1 $\exists T_2$ { C_1 Korth

$T_1.sid$

⇒ $\exists T_1 \in Enroll\ \exists T_2 \in Course$
$(T_1.cid = T_2.cid \wedge T_2.Inst = Korth \wedge T.sid = T_1.sid)$

T: Free variable to store result.

② Retrieve Sid's enrolled some course taught by "Korth" and some course taught by "Navathe".

$P \cap Q \equiv \{ x \mid x \in P \wedge x \in Q \}$
$P \cup Q \equiv \{ x \mid x \in P \vee x \in Q \}$
$P - Q \equiv \{ x \mid x \in P \wedge x \notin Q \}$

$\pi_{sid}(E \bowtie \sigma_{Inst=Korth}(C)) \cap \pi_{sid}(E \bowtie \sigma_{Inst:Navathe}(C))$

$T.sid$

$= \{ \exists T_1 \in Enroll\ \exists T_2 \in course (T_1.cid = T_2.cid \wedge T_2.Inst = Korth \wedge T.sid = T_1.sid)$

$\wedge \exists T_1 \in Enroll\ \exists T_2 \in Course (T_1.cid = T_2.cid \wedge T_2.Inst = Navathe \wedge T.sid = T_1.sid) \}$

Q. $\{ T.Sid \mid \exists T_1 \in Enroll (T.sid = T_1.sid) \wedge$ — Sid's enrolled atleast one course

$\rightarrow \exists T_1 \in Enroll \exists T_2 \in Enroll (T_1.Sid = T_2.Sid \wedge T_1.cid \neq T_2.cid \wedge T.sid = T_1.sid) \}$ — Sid's enrolled atleast 2 courses.

$\equiv$ Sid's enrolled only one course.

Q. $\{ T.sid \mid \exists T_1 \in Stud (T.sid = T_1.sid) \wedge$

$\rightarrow \exists T_1 \in Stud \nexists T_2 \in Stud (T_1.age < T_2.age \wedge T.sid = T_1.sid) \}$

$\equiv$ Retrieve Sid's whose age maximum.

Q. $\Rightarrow$ Sid's enrolled every course

$* \pi_{sid,cid}(Enroll) / \pi_{cid}(course)$

* Sid of Enroll(T_1) such that
  Every course of course(C)
  There exists some records in Enroll(T_2)
  with $T_2.cid = C.cid$ and
  $T_2.sid = T_1.sid$

Enroll (Sid, Cid)

Sid
S1

Course (Cid)
∀C
C1
C2
C3

Enroll (Sid, Cid)
S1, C1
S1, C2
S1, C3

$\{ T.sid \mid \exists T_1 \in Enroll (T.sid = T_1.sid) \wedge \forall C \in course \exists T_2 \in Enroll (T_2.cid = C.cid \wedge T_2.sid = T.sid) \}$

$\Rightarrow$ Sid's enrolled ~~the~~ every course taught by Korth

$\pi_{sid,cid}(Enroll) / \pi_{cid}(\sigma_{Inst = Korth}(course))$

$\forall(x)(C(x) \rightarrow S(x))$

x is Korth course — x should enroll. by $T_1.sid$

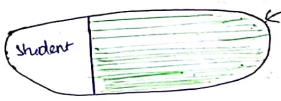
$\{ T.sid \mid \exists T_1 \in Enroll (T.sid = T_1.sid) \wedge \forall C \in course \exists T_2 \in Enroll (C.Instructor = Korth \rightarrow (T_2.cid = C.cid \wedge T_2.sid = T.sid)) \}$

Unsafe TRC Query:

→ TRC Query with infinite set of records in result.

$\{ T \mid T \notin Student \}$

↑ Set of Tuples (T) those not belonging to Student relation [Infinite records]



Expressive Power Comparison:

[Safe TRC Query Expressive Power] equal to [Basic RA Queries expressive power]

Basic RA Queries:

→ RA Queries which can express using

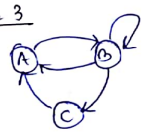
$\{\pi, \sigma, \times, \cup, -, \rho, \cap, \bowtie_c, \bowtie, \overline{\bowtie}, \overline{\bowtie}, \overline{\bowtie}, /\}$

⇒ Queries failed to express using Basic RA:

- Sum of attribute val
- Avg of attribute val
- count of records / counts of attr val.
- Ordering of resulted records of Query.

Chapter 3

Q.9.



Adj

X	Y
A	B
B	B
B	C
B	A
C	A

(a) Find adj of given vertex "C".

$\pi_y(\sigma_{x=C}(Adj))$

(b) Find vertices of self loop.

$\pi_x(\sigma_{x=y}(Adj))$

(c) Retrieve vertices with loop less than 3 vertices.

T_1

X	Y
A	B
B	A
A	C
B	C

T_2

X	Y
A	B
B	A
A	A
B	C

$\equiv \pi_X\left(Adj \underset{X=Y1 \wedge Y=X1}{\bowtie} \rho_{X1,Y1}(Adj)\right)$

(d) Reachability:

retrieve all vertices reachable from given vertex "A".

$\sigma(T_1 X \ldots X T_{n-1})$

* Cannot be expressed using relational algebra.