Relational Algebra

- Procedure Query Language
- Operators in relational Algebra
  1) Unary Operators
  2) Binary Operators
  3) From set theory

① Unary Operators
  a) Select (σ)
  b) Project (Π)
  c) Rename (ρ)

② Binary Operator
  a) Join ($\bowtie$)
  b) Cartesian product ($\times$)

③ Set theory ($\cup$, $\cap$, $-$)

a) Select Operator (σ):

It is used to find tuples in a relation, which satisfy the given condition

syntax: σ condition (relation name)

eg: To retrieve entire details from student table:

σ (student)

To display ID, name of student:

$\sigma_{id, name}$ (student)

$\sigma_{dep = 'analysis'}(\sigma_{location = "Newyork"}(Manager))$

(b) Project operator (Π):

It is used to select column or attribute from a table, it avoid repetition

Syntax: Π attribute (Relation)

→ To retrieve id and name from student

$\Pi_{id, name}(Student)$

→ To retrieve id and nam from student where mark >= 75

$\Pi_{id, name}(\sigma_{mark >= 75}(Student))$

ID	Name	Mark	WC
1	Joy	70	A
2	Vishnu	75	A
3	Dusani	60	B

$\Pi_{wc}(Student)$ →

wc
A
B
C

It is commutative

⇒ $\Pi_{id, name}(\sigma_{mark >= 75}(Student))$ and

⇒ $\sigma_{mark >= 75}(\Pi_{wc}(Student))$

(c) Rename operator (ρ)

It is used to rename old relation to new name.

Syntax: ρ (new relation, old relation)

ρ (stu, student)

Cust ID	Cust Name	Cust City
C100	Steve	Agra
C101	Raghu	Delhi
C102	Chaitanya	Noida
C103	Ajeet	Delhi
C104	Gall	Agra

① Retrieve customer name, customer city from relation

→ $\Pi_{custName, custCity}$ (customer)

② Retrieve custname, custcity from relation customer whose id = 'C104'

→ $\Pi_{custName, CustCity}$ ($\sigma_{id=104}$ (customer))

③ Retrieve the city from relation

Π custCity (customer)

Binary Operators :

- It combines information of two different relation into one.

Cartesion product :

- It is also called cross product or cross join.
- AXB

It will result in all attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

A

id	Name
1	A
2	B
3	C

B

Sid	SName
1	C
2	OS

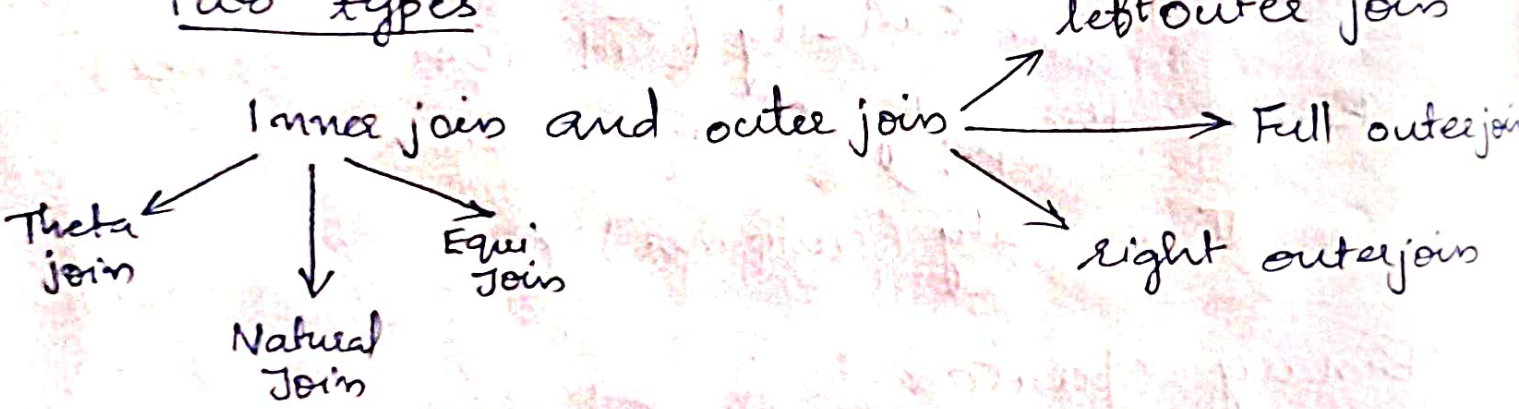
A × B

ID	Name	Sid	BName
1	A	1	C
1	A	2	OS
2	B	1	C
2	B	2	OS
3	C	1	OS
3	C	2	OS

a) Join (⋈)

It is used to combine two tables having same attributes.

Two types



1. Theta join (Θ):

- $A \bowtie_{\Theta} B$
- It can use any condition in selection criteria

eg: $A \bowtie_{\Theta} A.column2 = B.column2$

2. Equi Join:
It is done when theta join uses only the equivalence condition.

eg: $A \bowtie A.column2 = B.column2$

3. Natural join :

It acts on matching attributes where the value of attributes in both the relations all same.

eg: stud

ID	Name	DNo
1	A	1
2	B	2
3	C	3

Dept

DNo	DName
1	CS
2	EC
4	EE

stud ⋈ Dept

ID	Name	DNo	DName
1	A	1	CS
2	B	2	EC

4. Left outer join (A ⟕ B)

It returns all records from left table to the matched records of right table

eg:

ID	name	DNo	Dname
1	A	1	CS
2	B	2	EC
3	C	3	null

5. Right Outer join (A ⟖ B)

Dno	Dname	ID	Name
1	CS	1	A
2	EC	2	B

6. Outer joint ($A \overline{\bowtie} B$)

ID	Name	Dno	Dname
1	A	1	CS
2	B	2	EC
3	C	3	null
null	null	4	EE

Set Theory:

a) Union (U):-

It includes all tuples that are in table AUB or

A or B $A \cup B = B \cup A$

eg:

Arts

ID	Name
1	A
2	B
3	C

Sports

ID	Name
3	C
2	B
4	D

Arts ∪ Sports

ID	Name
1	A
2	B
3	C
4	D

Q: Retrieve student ID, Name either participant in Arts or sports

Sol Π studentID, Name (Arts) $\cup \Pi_{ID, Name}$ (Sports)

b) Intersection ($\cap$) :-

It is a relation that basically include all tuples that are present in A and B.

$A \cap B = B \cap A$

Retrieve student Name who participate in both arts and sports

Π id, name (Arts) $\cap \Pi_{id, Name}$ (Sports)

ID	Name
2	B
3	C

c) Difference (-) :-

It is a result of set difference operation of tuples which are present in one relation but not in second relation.

It is not commutative.

i.e $A - B \neq B - A$.

eg: Retrieve student name who participate in arts not in sports.

$\Pi_{id, studentname}(Arts) - \Pi_{id, name}(Sports)$

ID	Name
1	A

Q: Consider the following tables:

Passenger (pid, pname, pgender, pcity)
Bus (bid, bdate, time, src, dest)
Agency (aid, aname, acity)
booking (pid, aid, bid, bdate)

a) Given the details of all buses from Dhangadhi to Kathmandu.

Sol. $\Pi_{bid}(\sigma_{src='Dhangadhi' \text{ and } dest='Kathmandu'}(bus))$;

b) Find the name of passengers who booked atleast one bus.

Sol Π_{name} (passenger ⋈ booking)

c) Find the no number of the passengee with pid = 'P04'. For thee does to Bhutwal before 20

Sol $\Pi_{bid}(\sigma_{pid='P04'} \cap dest='Bhutwal' \cap bdate < '20/05/2021'(Guest \bowtie book$

d) Find the name of the passengee who does not booked any bus.

Sol. $\Pi_{pname}(\sigma_{bid=null}(passengee \bowtie booking)$

e) Find the details of all male passengees associated with Migratara Agency.

Sol. Π pname, pid (σ pgendee = 'male' and σ aname = 'migratara' (passenga $\bowtie$ Agency $\bowtie$ booking)

Q: Student (rollno, studentname, Address)
Teachers (TeacherID, Teachee name, Teaching Subject)
college (rollno, teacher ID)

(a) Find the name of student who live in 'Lalitpur'

Sol: $\sigma_{studentname}(\sigma_{address='Lalitpur'}(Student))$

(b) Find the name of teachee who teaches DBMS

Sol $\Pi_{teachername}(\sigma_{teaching\ subject='DBMS'}(Teacher$

(c) Find the name of teacher who teaches 'C' Subject for John Smith.

Ans. Π teachername (σ teaching subject = 'W' σ student name = 'John Smith' (Student $\bowtie$ teachers))

(d) insert a new tuple into relation teaches

Ans. Teacher $\leftarrow$ teacher $\cup$ {"T01", "Sivan", "T0c"}

(e) Delete records of the student whose address is 'pokhara'

Ans. Students $\leftarrow$ Student $-$ σ address = 'pokhara' (student)

Built In functions :

group functions / Multiple functions :

String function : Text Manipulate

1) lower() : all letter lowercase

eg: lower()

select lower ("SQL") result: sql

2) Upper() : all letter uppercase

eg: select upper ("SqL") result: SQL

3) Init Cap() :

4) Ltrim()

eg: Select Ltrim (" Ashi ") result: Ashi

5) Rtrim()

eg: Select Rtrim (" Ashi ") result: Ashi

6) Char()

eg: Select Char ("65")

7) Concat():

eg: Selcct concat('Somu', 'Soble', 'Ashi');

result: SomuSoble Ashi

8) Substr():

eg: Select Substr("HelloWorld", 1, 4)

result: Hello

9) Instr():

eg: Select instr('HelloWorld', 'Woeld')

result:

10) length():

eg: Select length('Somu');

result: 4

11) mid():

eg: Select mid("Hello", 4)

result: lo

12) left():

eg: Select left("HelloWorld", 2);

result: He

13) Right()

eg: Selects right('HelloWorld', 2);

result: ld

14) ascii()

eg: Select ascii('A');

result: 65

Mathematical Functions:

1) Round():

eg: Select round (45.926, 2)

result : 45.93

2) Truncate ()

eg: select truncate (49.532, 2)

result : 49.53

3) Mode ()

eg: select mode (10, 5)

result : 0

4) power (3, 2)

eg: select power (5, 2);

result : 25

Data and time functions

1) sysdate ():

Current system Date and time

eg: select sysdate ();

2) curdate

Only current be

eg: selects curdate ()

result: 2022-11-17 8:59:03

3) Now ()

eg: select Now ();

4) Date() : select date ('2022-11-17');

5) Dayname 8) Dayofmonth

6) Day 9) Dayofweek

7) Month 10) Day of years

Views in SQL :

View is a virtual table that contain rows and column just like a real table.

Virtual table

1) Create View view_name as
   select column1, column2 from tablename
   where condition;
2) select * from VW_emp;

Q: Develop SQL Queries for creating and dropping views

a) Create a view VW_emp on employee table

b) Create another view VW_SSN contains SuperSSN and DNO of female employee

c) Update the address of employee to Chennai whose id is e100 in view VM_emp.

d) Delete the view VW_emp

Sol a) • Create View VW_EMP as select * from employee;
• select * from VW_EMP;

b) • Create view VW_SSN As select SuperSSN,
• dno from employee where sex = 'female';
select * from VW_SSN;

c) • update VW_EMP set address = 'Chennai'
where SSN = "P100";

d) drop view vw_emp

Assertion:

An assertion is a predicate expressing a condition we wish the database to always satisfy.

Example.

The salary of managers in staff table must not less than salary of project manager.

⇒ Create assertion <assertionname> check <predicate>;

⇒ drop assertion <assertionname>

example:

Sno.	Name	Sal.	position	Dno.
S000	John	30000	Manager	D001
S001	Susan	40000	Manager	D002
S002	David	12000	Proj Manage	D003
S003	Ann	5000	Proj Manager	D004

(a) Create an assertion to check the salary of managers in staff table must not less than salary of project manager.

Sol: Create assertion Salary_check check (not exists (select * from staff where position = 'manager' and sal < (select Sal from

staff where position = 'Programmer')));

example:

Booktable:

Bookid	Bnames	price	Category
B014	Sherlock		Textbook
B015	Merchant		Textbook
B016	Rukmake		Novel
B017	Joke		Novel.

Q1. consider book table write an assertion to check whether the price of text book category is not less than minimum price of novel.

Sol: Create assertion Book_check check (not exists (select * from book where category = 'TextBook and price < (select price from book where category = 'novel')));

(nested queries)

Note

Not exists: It indicate that the result of the query must be empty.

Assertion is violated if the result of que

inside not exist clause is not empty.

Q9 Employee Table:

eid	Name	Salary

Attendence table:

eid	Punching-in	Punching-out

(a) Create assertion to check whether any staff has punched in after 10 am.

Sol. Create assertion Latepunch_check check (not exists (Select name from Employee where eid in (select eid from attendence where employee.eid = attendence.eid and attendence.punching_in > 10)));

Trigger:

Triggers are SQL code that are automatically executed in response to a ~~particular~~ certain events on a particular table

Modules of trigger

- Event
- Condition
- Action

(a) Event - g. insert, delete, update

(b) Condition - Is to evaluate the event.

(c) Action - Takes place based on condition.

Syntax:

→ Create trigger <trigger_Name> (After | before) trigger event on tablename for each row when condition trigger action.

→ Select * from <tablename>;

Q. Given student report database in which student marks assessment is recorded create a trigger so that the total and average of specified map is automatically inserted before a record is inserted.

A. ~~Create trigger <stud_marks)~~

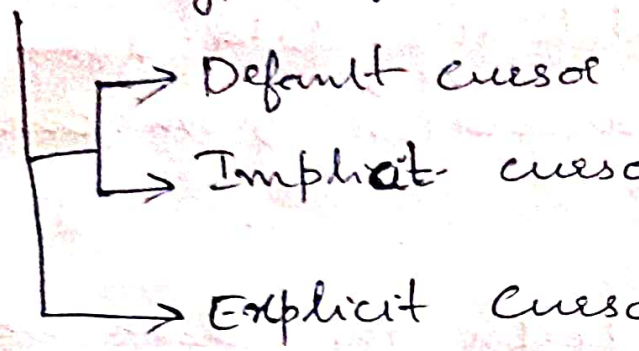
Create trigger Stud_marks before insert on student for each row when Student. total = student.subj1 + Student.subj2 + Student.subj3, Student.avg = Student. total / 3

Cursor : Temporary Memory
Temporary Workstation

It is used to store database table.

It is allocated by database server at the time of performing DML operations by the users.

Two types of Cursor:



It is used for fetching data from table.

Creating Explicit Cursor:

① Declare cursor object

- declare <cursorname> cursor for select * from <tablename>

eg: declare S1 cursor for select * from StudDetails

② Open cursor connection

- open <cursor_name>

eg: open S1

③ fetch Data from cursor

fetching data can be done by 6 types

1) first - firstrow of cursor table
used for fetching firstrow from the table

2) last - fetch only the last row from the cursor table.

3) Next - used to fetch the data ~~next~~ in forward direction from cursor table.

4) prior - used to fetch the data in backward direction in cursor table

5) Absolute n - used to fetch the exact data from the nth row from the cursor table.

6) Relative n :- used to fetch data in incremental way as well as decremental way from the cursor table.

Syntax:

Fetch first/last/next/prior/absolute n/relative n from <cursor name>

eg: fetch first from S1
fetch prior from S1
fetch last from S1
fetch absolute 7 from S1
fetch relative -7 from S1

④ Close cursor connection.

- close <cursorname>

⑤ Deallocate cursor memory

- deallocate <cursor_name>

eg: deallocate S1

Normalization

⇒ Data redundancy Issues
(duplication of data)

* More memory space
* More access time
* Data Inconsistency - Modification of redundent data.
* Anomalies - errors
  - Insertion
  - Deletion
  - Updation

ⓐ Insertion Anomaly:

Anomaly caused during insertion of

(b) Deletion Anomaly: Deletion of Row as entirely the error occured

(c) Updation Anomaly: Caused during updation of an entirely row.

⇒ Normalization technique

- Normal forms
  - 1NF, 2NF, 3NF — Edger F Codd
  - BCNF, 4NF, 5NF — Raymond F Boyce

⇒ Functional Dependencies: (→)

Example: $x \rightarrow y$ functional dependency

- determinant (x) → dependent (y)
- unique value

- x must be able to determine y
- x and y are attributes of the table

eg: Parent-child relationship

Functional Dependency

Roll No	Name	Age
1	AA	20
2	AB	18

Roll No → Name

Roll No → Age

Types of functional dependency:

1) Trivial Functional dependency: $x \cap y \neq \phi$

A dependent is always a subset of determinant. Composite key - prime attribute

eg: $X \rightarrow y$ (Roll No, Name) → Name

2) Non Trivial functional dependency: $x \cap y = \phi$

A dependent is strictly not a subset

- Prime attribute
- Non-prime attribute

of determinant.

eg: $x \rightarrow y$ RollNo → name

RollNo → Name, age

3) Multivalued functional Dependency:

The entities of dependent set are not dependent on each other.

eg: RollNo → Name, age.

4) Transitive functional Dependency:

Dependent is indirectly dependent on determinant.

eg: a b c if $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$

Properties or Rules of function Dependency:

Amstrong Axioms (6 rules)

(a) Reflexive rule:

If y is a subset of x then $x \rightarrow y$

(b) Augmentation rule (Partial dependency Rule):

If $x \rightarrow y$, $xz \rightarrow yz$

(c) Transitive rule:

If $x \rightarrow y$ and $y \rightarrow z$ then $x \rightarrow z$

(d) Union rule

If $x \rightarrow y$ and $x \rightarrow z$ then $x \rightarrow yz$

Given $x \rightarrow y$ —①

$x \rightarrow z$ —②

Doing augmentation in ①

$xx \longrightarrow xy$

$x \rightarrow xy$ —③

Doing augmentation in ②

$xy \rightarrow yz$ —④

As per transitive rule

$x \rightarrow xy$ and $xy \rightarrow yz$

$\Rightarrow$ $x \rightarrow yz$ hence proved.

(e) Decomposition / Project Rule:

If $x \rightarrow yz$ then $x \rightarrow y$ and $x \rightarrow z$

(f) Pseudo transitive Rule:

If $x \rightarrow y$ and $yz \rightarrow w$ then

$xz \rightarrow w$

$\Rightarrow$ Normal forms:

1) INF

- It should have only single valued attributes
- It do not allow multi valued attributes

example

1st sol

Data redundancy

Cid	CName	Pbrand
1	aaa	Nike
1	aaa	Woodland
2	bbb	Adidas
2	bbb	Woodland

2nd sol

Cid	CName	PB1	PB2	PB3

free space

3rd sol

Cid	CName
1	aaa
2	bbb

Cid	PBrand
1	Nike
1	woodland
2	Adidas
2	woodland

Data redundancy

2) 2NF :-

- All non keys are fully functional dependent on primary key
- 1NF
- No partial dependency

Example :

Cid	CName	Pbrand	Ptype	Pcategory	price
1	aaa	Nike	Cloths	Tshirt	2000
1	aaa	WL	FW	shoes	2500
2	bbb	Adidas	Cloths	Jacket	2000
2	bbb	WL	FW	shoes	3000

P(Cid, CName)
A B

A B
Candidate key
Composite

AB → CD - fully functional Dependency

B → C
B → D
A → C
A → D
} partial fuctional Dependency

Cid	CName
1	aaa
2	bbb

Cid	pbrand	ptype	pcategory	price
1	Nike	Cloth	Tshirt	2000
1	WL	FW	shoes	2500
2	Adidas	Cloths	Jacket	2000
2	WL	FW	Shoes	3000

1st eq.

Cid	CName

Cid	pbrand	price

Cid, pbrand

pbrand	ptype	pategory

Here also partial dependency exist

pbrand ⟶ ptype / pcategory

3) 3NF

→ 2 NF Format table
→ There must not be any transitive dependency

eg: Here

(Cid, CName)

Cid	CName

(Cid, pbrand), prize)

cid	pbrand	prize

(Pbrand (ptype, pcategory))

Pbrand	ptype	pcategory

Pbrand → pcategory, ptype
ptype → pcategory
pbrand → pcategory

eg:

①

Cid	CName

Cid	pbrand	price

Pbrand	ptype

Pbrand	pcatego

4) BCNF Boyle Codd NF:

→ 3NF format table
→ Determinant should be a super key or Candidate key

eg:

cid	brand	Seller
1	Nike	S1
2	Adidas	S2
2	WL	S2
2	Nike	S3

Cadidate key (cid, brand)
(Cid, brand) → seller
LHS Candidate

If brand → Seller is there brand is not the Candidate key / Super key therefore split the table as (Cid, brand)

cid	brand

(Cid, Seller)

cid	seller

Q: R(ABCD), $AB \rightarrow CD$ $C \rightarrow A$ $AB \rightarrow D$ $D \rightarrow B$ is in INF

steps to check:

2NF : LHS should be a proper subset of Candidate key and RHS should be a non prime of A

3NF : LHS should be a Candidate key or RHS should be a primary attribute

BCNF : LHS should be a Candidate key or Super key

Closure Property:

It is a set of all those attribute which can be functionally determined from an attribute or set of attribute are called closure of that attribute or set of attribute

eg: Covid Contact list:

Infected one → contacted people

They Contacted.

A A^+

eg: R(ABCD) with functional dependencies are $A \rightarrow B$, $B \rightarrow C$, $AB \rightarrow C$

Sol.

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

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

$C^+ = \{C\}$

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

Q: R(A B C) the fundamental dependencies are

$A \rightarrow B$ $B \rightarrow C$ $AB \rightarrow C$

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

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

$C^+ = \{C\}$

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

To find Candidate key
→ All values included
→ Minimal key

CK - A

SK - AB

Candidate key it is the minimal key and is obtained from a Super key.

Prime attribute - A

Non prime attribute - {B C}

Q: R(A B C D) values are atomic

$AB \rightarrow D$ $BC \rightarrow D$ $A \rightarrow C$ $C \rightarrow A$ find out the highest normal form.

Sol. closure

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

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

Candidate key = {AB, BC} because proper subset of {A, B, C, D}

Prime attribute = {A, B C}

Non prime attribute = {D}

a) Partial dependency Checking for 2NF

- $AB \rightarrow D$

condition ①: False

②: True

False

$AB \rightarrow D$ is in 2NF

Conditions:

① LHS should be a proper subset of Candidate key

② RHS should be a non-prime attribute

- $BC \rightarrow D$ condition ①: False
  False ②: True
  $BC \rightarrow D$ is in 2NF
- $A \rightarrow C$ condition ①: True
  False ②: False
  $A \rightarrow C$ is in 2NF
- $C \rightarrow A$ condition ①: True
  False ②: False

Hence the R(ABCD) is in 2NF.

b) for 3NF

Conditions:

① LHS should be a candidate key

OR

② RHS should be a prime attribute

- $AB \rightarrow D$ condition ①: True
  True ②: False
- $BC \rightarrow D$ Condition ①: True
  True ②: False
- $A \rightarrow C$ Condition ①: False
  True ②: True
- $C \rightarrow A$ condition ①: False
  True ②: True

Hence the R(ABCD) is in 3NF

c) Check for BCNF

Conditions:

① LHS = Candidate key / Super key

- $AB \rightarrow B$ LHS - Candidate key True
- $BC \rightarrow D$ LHS - Candidate key True
- $A \rightarrow C$ LHS - Candidate key False
- $C \rightarrow A$ LHS - Candidate key False

Hence the R(ABCD) is not in BCNF.

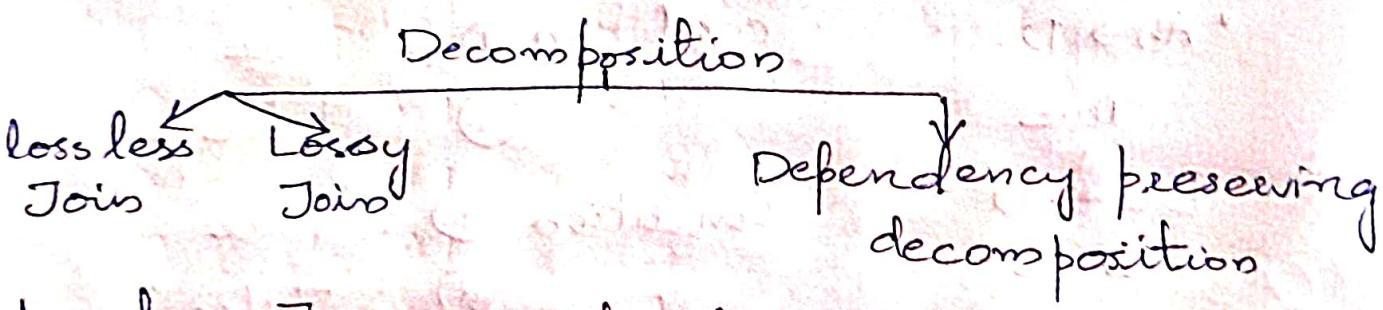
The highest Normal Form of R(ABCD) is 3NF

⇒ Decomposition:

The process of breaking up or dividing a single relation into 2 or more subrelation is termed as decomposition of a relation.

Conditions for decomposition.

It must preserve its attributes and dependences.

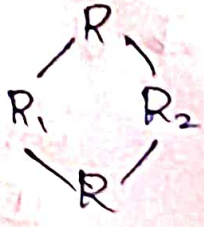


1). Lossless Join Decomposition:

No information is lost from the original relation during decomposition. When the subrelations are joined back using natural join it should get the same exact parent relation

$R \longrightarrow R_1\ R_2\ R_3 \ldots$

$R_1 \bowtie R2 \bowtie R3 \bowtie \ldots \bowtie R_n = R$

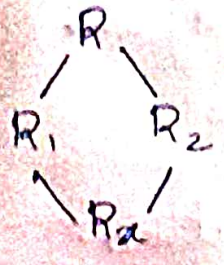


2). Lossy Join Decomposition:

The Join of sub relation does not results in the exact parent relation. The natural join of subrelation results some extraineous interversions.

$R1 \bowtie R2 \bowtie R3 \bowtie \ldots \bowtie Rn \supset R$ superset of R

$R_x \supset R$



Rx is a superset of R.

Normalize – Decomposition:

* example for Lossy Join Decomposition:

Roll No	Name	Subject
1	A	C
2	B	C
3	A	OS

A

Roll No	Name
1	A
2	B
3	A

B

Name	Subject
A	C
B	C
A	OS

Cartision product of A and B:

Roll No	Name	Name	Subject
1	A	A	C
1	A	B	C
1	A	A	OS
2	B	A	C
2	B	B	C
2	B	A	OS
3	A	A	C
3	A	B	C
3	A	A	OS

Matching Name

Final table with matching name in Cartesian P...

RollNo	Name	Subject
✓ 1	A	C
1	A	OS
✓ 2	B	C
3	A	C
✓ 3	A	OS

There are extra colum... which include extra... information. Therefore it is an example for Lossy Join.

* Example for Lossless Join Decomposition:

RollNo	Name	Subject
1	A	C
2	B	C
3	A	OS

A

RollNo	Name
1	A
2	B
3	A

B

RollNo	Subject
1	C
2	C
3	OS

Cartesian Product of A and B

Name	RollNo	RollNo	Subject
A	1	1	C
A	1	2	C
A	1	3	OS
B	2	1	C
B	2	2	C
B	2	3	OS
A	3	1	C
A	3	2	C
A	3	3	OS

Final table with matching RollNo in CP

RollNo	Name	Subject
1	A	C
2	B	C
3	A	OS

Here the values of the table are same as the table used for decomposition therefore its an example for losseless join Decomposition.

Note: A - subclass

B - subclass

Lossless Join Decomposition conditions :

① $A \cup B = R$

② $A \cap B \neq \phi$

③ $A \cap B$ = Superkey of any of the subrelation or both.

Q: $R(ABCDE)$ $R_1(ABD)$ and $R_2(ACDE)$ where functional dependency exsist b/w $AD \rightarrow B$, $C \rightarrow DE$, $B \rightarrow AE$, $AE \rightarrow C$

Sol ① $R_1 \cup R_2 = \{A\ B\ C\ D\ E\} = R$

② $R_1 \cap R_2 = \{AD\}$

③ $AD^+ = \{A, D, B\ E\ C\}$

Its a lossless join decomposition since we are able to have all the conditions satisfied and the resulting is all the relations.

Q: $R(ABCD)$ $R_1(AB)$ $R_2(CD)$ where functional dependency exsist $A \rightarrow C$ $C \rightarrow D$

Sol. ① $R_1 \cup R_2 = \{A\ B\ C\ D\} = R$

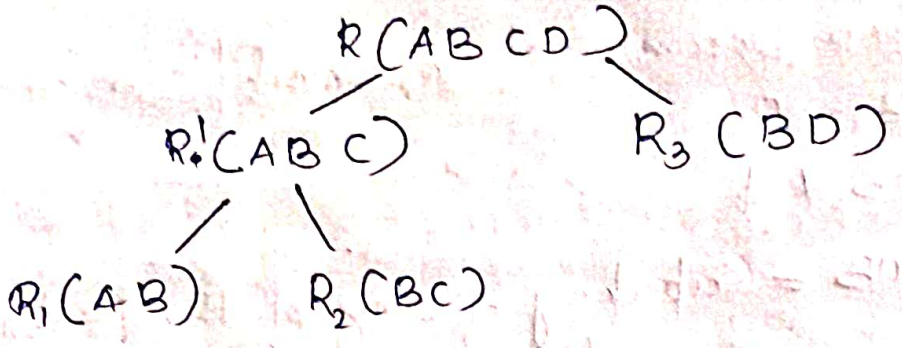
② $R_1 \cap R_2 = \phi$

The decomposition is Lossy Join decomposition Since it does not satisfies the condition for

lossless Join Decomposition.

Q: R(A B C D) where R_1(A B) R_2(B C) and R_3(B D) functional dependency exist A → B B → C C → D D → B

Sol



① R_1(AB) R_2(BC)

① $R_1 \cup R_2 = \{A B C\} = R'$

② $R_1 \cap R_2 = \{B\} \neq \phi$

$B^+ = \{B C D\}$.

Hence it is lossless join Decomposition

② R' (ABC) R_3 (BD)

① $R' \cup R_3 = \{A B C D\} = R$

② $R' \cap R_3 = \{B\}$

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

Hence it is lossless join Decomposition

⇒ Functional Dependency:

$(F_1 \cup F_2 \cup F_3 \cup \ldots F_n)^+ = F$

Using closure of the functional dependencies joined.

Q. R(A B C D) Function Dependencies of
A → B B → C C → D D → A
R_1(A B) R_2(B C) and R_3(C D) check for functional dependency.

Sol. 1) Removing trivial dependency - eg: AB → AB

R_1(A B) = {A → B, B → A}

R_2(B C) = {B → C, C → B}

R_3(C D) = {C → D, D → C}

B^+ = {B C D A}

C^+ = {D A B C}

A U B U C U D = {A → B, B → A, B → C, C → B, C → D, D → C}

D^+ = {D A B C}

The function is dependency preserving.

Q: R(A B C D E) Functional Dependency
A → BC C → DE D → E where R_1(A B C D)
R_2(D E)

Sol. R_1(A B C D) = {A → BC C → D}

R_2(D E) = {D → E E → D}

$R_1 \cup R_2$ = {A → BC C → D D → E E → D}

This function is dependency satisfied.

⇒ Equivalence of Functional Dependencies:

Q. α = (A → B, B → C, AB → D)

γ = (A → B, B → C, A → C, A → D)

If (x covers y) $x \supseteq y$ and if (y covers x) $y \supseteq x$ we can say x is equivalent to y ($x \equiv y$).

evaluation :

- Closure of y is in x
- functional Dependency of y in y are checked for equivalence.

x and y are in equivalence relation if

Step ① (x covers y)

Checking by LHS in y and finds the closure in x. Also checks for functional dependency of y in y

Step ② (y covers x)

Checking by LHS in x and finds the closure in y. Also checks for the functional dependency of x in x

If both step ① and step ② satisfied the equivalence of the function is true

Q: $x = \{A \to B \quad B \to C \quad A \to D\}$

$y = \{A \to B \quad B \to C\}$ Check for equivalence.

Sol • Check for $x \supseteq y$

→ $A^+ = \{A\ B\ C\ D\}$ $A \to B$ True

→ $B^+ = \{B\ C\}$ $B \to C$ True

Functional dependency satisfied

hence $x \supseteq y$ is True

- Check for $y \supseteq x$

→ $A^+ = \{A\ B\ C\}$ — $A \to B$ True, $B \to C$ True, $A \to D$ False

→ $B^+ = \{B\ C\}$

Functional Dependency not satisfied

hence $xy \not\supseteq x$ and $x \not\equiv y$

x and y are not equivalent relations.

Q. R(A B C D)

$FD_1 = (A \to B \quad B \to C \quad A \to C)$

$FD_2 = (A \to B \quad B \to C \quad A \to D)$

Sol • Check for FD1 ⊇ FD2

→ $A^+ = \{A\ B\ C\}$ — $A \to B$ True, $B \to C$ True, $A \to D$ False

→ $B^+ = \{B\ C\}$

Funtional Dependency not satisfied

hence $FD_1 \not\supseteq FD_2$ and $FD_1 \not\equiv FD_2$

Q: Check whether two functional dependencies are functionally dependent or not.

R(A C D E H)

$F_1 = \{A \to C \quad AC \to D \quad E \to AD \quad E \to H\}$

$F_2 = \{A \to CD \quad E \to AH\}$

Sol. • F1 ⊇ F2

$A^+ = \{A\ C\ D\}$ — $A \to CD$ True

$E^+ = \{E\ A\ H\ C\ D\}$ — $E \to AH$ True

Functional Dependency Satisfied

F1 ⊇ F2 is True

- F2 ⊇ F1

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

$E^+ = \{E\ A\ H\ C\ D\}$

$AC^+ = \{A\ C\ D\}$

$A \to C$ True
$AC \to D$ True
$E \to AD$ True
$E \to H$ True

Functional Dependency Satisfied

Hence F2 ⊇ F1 and F1 ≡ F2

F1 is equivalent to F2

Q: $x = (A \to B\ \ B \to C\ \ AB \to D)$

$y = (A \to B\ \ B \to C\ \ A \to C\ \ A \to D)$

Check for functional equivalence

Sol. $x \supseteq y$

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

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

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

Functional Dependency Satisfied

$y \supseteq x$

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

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

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

$A \to B$ True
$B \to C$ True
$AB \to D$ True

Functional Dependency Satisfied

Hence $x \supseteq y$ and $y \supseteq x$ ∴ $x \equiv y$

Q: R(A B C D E F) with the functional dependency

$F1 = \{A \to BC\ \ B \to CDE\ \ AE \to F\}$

$F2 = \{A \to BCF,\ B \to DE\ \ E \to AB\}$

Sol. Check for F1 ⊇ F2

$A^+ = \{A\ B\ C\ D\ E\ F\}$

$B^+ = \{B\ C\ D\ E\}$

$E^+ = \{E\}$

$A \to BCF$ True

$B \to DE$ True

$E \to AB$ False

Functional Dependency Not satisfied

$\therefore$ $F1 \not\equiv F2$ and $F1 \not\equiv F2$.

⇒ Minimal Cover / canonical Cover / ~~Parsi~~ Irreducible set

Steps:

1. Split the Functional Dependency or Decomposition

   $A \to BC \Rightarrow A \to B$ & $A \to C$

2. Remove extraneous Functional dependencies

3. $BC \to A$: split the two attributes into single ones

Q: $w \to x$ $y \to x$ $z \to wxy$ $wy \to z$

Sol ① $w \to x$ $y \to x$ $z \to w$ $z \to x$ $z \to y$ $wy \to z$

② Here $z \to x$ not possible $z \to y$

$z^+ \to \{z, w, y, x\}$ but $z^+ = \{z, w, x, y\}$

Can be removed (x) $y \to x$ hence cannot remove y

$y \to x$

③ $w \to x$ $y \to x$ $z \to w$ $z \to y$ $wy \to z$

$w^+ = \{w\ x\}$

$y^+ = \{y\ x\}$ $\therefore$

Minimal form -

$w \to x$ $y \to x$ $z \to wy$ $wy \to z$

Q: $\{A \to C \quad AC \to D \quad E \to H \quad E \to A \quad E \to D\}$

Sol ① $A \to C$, $AC \to D$ $E \to H$ $E \to A$ $E \to D$

② $A \to C$ $AC \to D$ $E \to H$ $E \to A$

③ $A^+ = \{A\ C\ D\}$ $C^+ = \{C\}$

A closure contains C, then C is extraneous and it can be removed $\{A \to D\}$

$A \to C$ $E \to H$ $E \to A$ $A \to D$