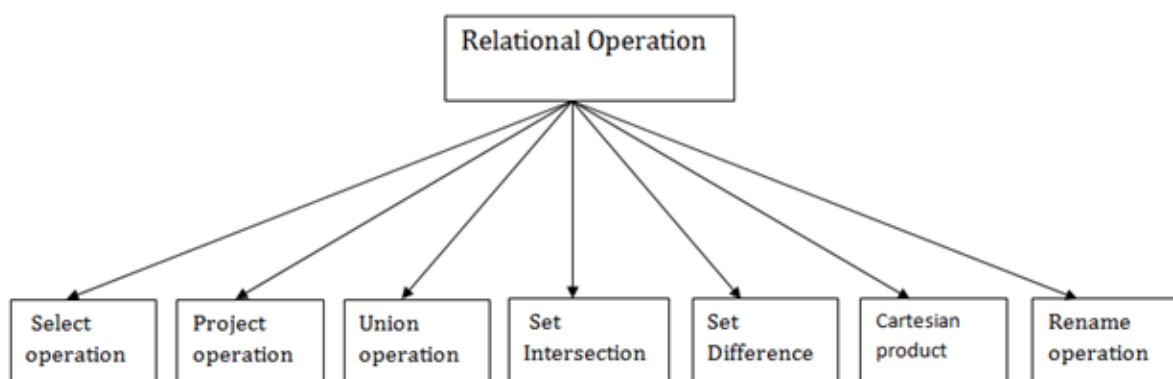


Relational Algebra

Relational algebra is a procedural query language. It gives a step-by-step process to obtain the result of the query. It uses operators to perform queries.

Types of Relational operation



1. Select Operation:

- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

1. Notation: $\sigma p(r)$

Where:

σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like $=, \neq, \geq, <, >, \leq$.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

1. σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

The SELECTION operation:

Ex.2: Select the EMPLOYEE tuples whose Department Number is 2 and Salary > 30000.

Sol:

$\sigma_{DNo=2 \text{ AND } Salary > 30000}$ (EMPLOYEE)

EMPLOYEE	FName	LName	Salary	DNo
	Tom	Ford	30000	3
	Amy	Jones	20000	2
	Alicia	Smith	40000	2

2. Project Operation:

- This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.
- It is denoted by π .

1. Notation: $\pi A_1, A_2, A_n (r)$

Where

A1, A2, A3 is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison

Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input:

1. π NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

The PROJECTION operation:

Syntax: $\pi_{\langle \text{attribute_list} \rangle} (R)$

$\pi \Rightarrow$ PROJECTION operator

Ex.1: To list the Employee's First name, Last name and Salary, we can use the PROJECTION operation.

Sol: $\pi_{\text{FName, LName, Salary}} (\text{EMPLOYEE})$

Result {	FName	LName	Salary	EMPLOYEE	FName	LName	Salary	DNo
	Tom	Ford	30000		Tom	Ford	30000	1
	Amy	Jones	30000		Amy	Jones	30000	2

3. Union Operation:

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by \cup .

1. Notation: $R \cup S$

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101

Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input:

- π CUSTOMER_NAME (BORROW) \cup π CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson

Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

4. Set Intersection:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection \cap .

1. Notation: $R \cap S$

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. \cap CUSTOMER_NAME (BORROW) \cap \cap CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME

Smith
Jones

5. Set Difference:

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

1. Notation: $R - S$

Example: Using the above DEPOSITOR table and BORROW table

Input:

1. π CUSTOMER_NAME (BORROW) - π CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

6. Cartesian product

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.

- It is denoted by X.

1. Notation: E X D

Example:

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:

1. EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

Degree of $R1 \times R2$ = degree of $R1$ + degree of $R2$

{degree = total no of columns}

Example

Consider $R1$ table –

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B

Table $R2$

Name	RegNo
Bhanu	2
Priya	4

R1 X R2

RegNo	Branch	Section	Name	RegNo
1	CSE	A	Bhanu	2
1	CSE	A	Priya	4
2	ECE	B	Bhanu	2
2	ECE	B	Priya	4
3	CIVIL	A	Bhanu	2
3	CIVIL	A	Priya	4
4	IT	B	Bhanu	2
4	IT	B	Priya	4

7. Rename Operation:

The rename operation is used to rename the output relation. It is denoted by **ρ** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENT1.

1. $\rho(\text{STUDENT1}, \text{STUDENT})$

Note: Apart from these common operations Relational algebra can be used in Join operations.

Division operation

The division operator is used for queries which involve the 'all'.

$R1 \div R2$ = tuples of R1 associated with all tuples of R2.

Example

Retrieve the name of the subject that is taught in all courses.

Name	Course
System	Btech
Database	Mtech
Database	Btech
Algebra	Btech

÷

Course
Btech
Mtech

=

Name
Database

The resulting operation must have all combinations of tuples of relation S that are present in the first relation or R.

Example 2

Retrieve names of employees who work on all the projects that John Smith works on.

Consider the Employee table given below –

Name	Eno	Pno
John	123	P1
Smith	123	P2
A	121	P3

÷

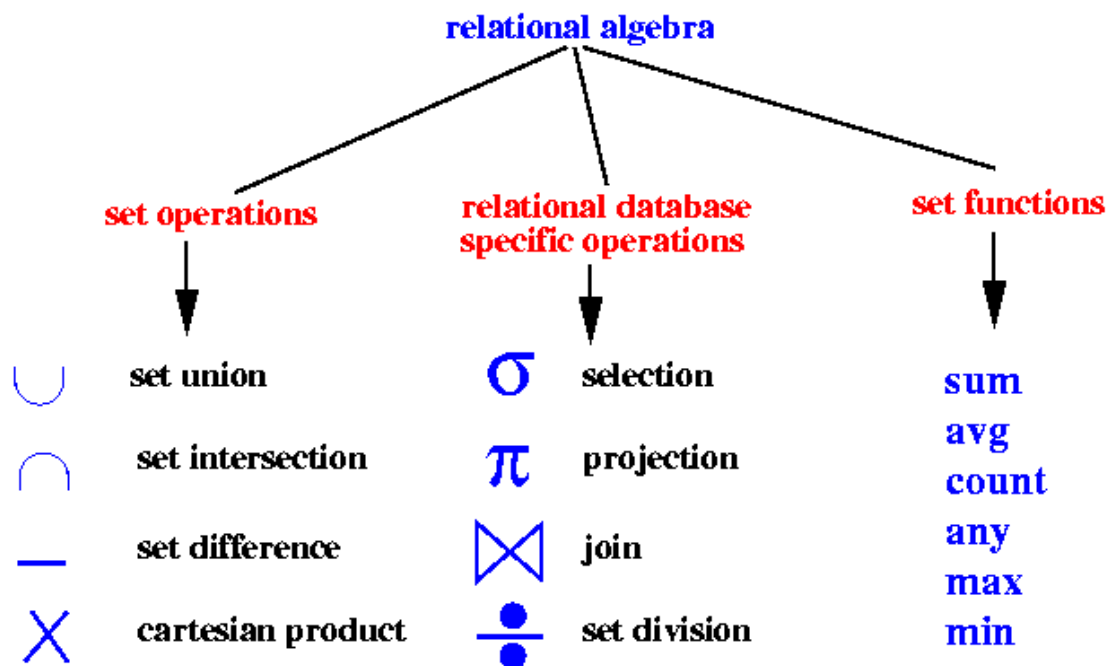
Works on the following –

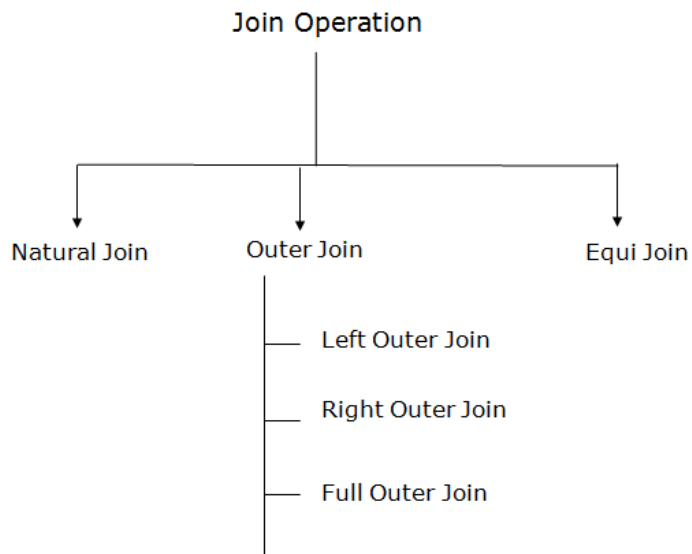
Eno	Pno	Pname
123	P1	Market
123	P2	Sales

=

The result is as follows

Eno
123





Natural Join:

- A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names.
- It is denoted by \bowtie .
- Is cross product.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

1. $\pi_{EMP_NAME, SALARY} (EMPLOYEE \bowtie SALARY)$

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

Equi join:

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

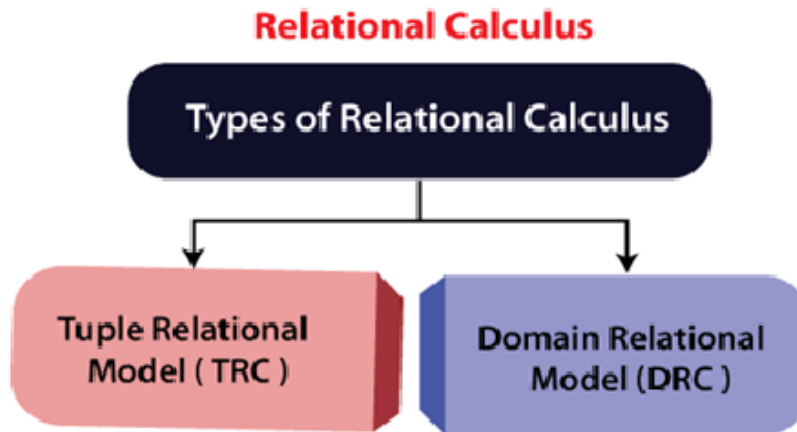
Input:

1. CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Relational calculus:



- Tuple relational calculus (TRC)
- Domain relational calculus (DRC)

1. Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describes the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation. The result of the relation can have one or more tuples.

Notation:

A Query in the tuple relational calculus is expressed as following notation

1. $\{T \mid P(T)\}$ or $\{T \mid \text{Condition}(T)\}$

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

1. $\{T.\text{name} \mid \text{Author}(T) \text{ AND } T.\text{article} = \text{'database'}\}$

Output: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (\exists) and Universal Quantifiers (\forall).

For example:

1. $\{ R \mid \exists T \in \text{Authors}(T.\text{article} = \text{'database'} \text{ AND } R.\text{name} = T.\text{name}) \}$

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives \wedge (and), \vee (or) and \neg (not). It uses Existential (\exists) and Universal Quantifiers (\forall) to bind the variable. The QBE or Query by example is a query language related to domain relational calculus.

Notation:

1. $\{ a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n) \}$

Where

a1, a2 are attributes

P stands for formula built by inner attributes

For example:

1. $\{ \langle \text{article}, \text{page}, \text{subject} \rangle \mid \in \text{javatpoint} \wedge \text{subject} = \text{'database'} \}$

Output: This query will yield the article, page, and subject from the relational javatpoint, where the subject is a database.

DRC

The domain regional calculus works based on the filtering of the domain and the related attributes.

DRC is the variable range over the domain elements or the filed values. It is a type of simple subset

of first-order logic. It is domain-dependent compared to TRC is tuple dependent. In DRC the formal

variables are explicit for the relational calculus representations. The domain attributes in DRC can be

represented as C_1, C_2, \dots, C_n and the condition related to the attributes can be denoted as the

formula defining the condition for fetching the $F(C_1, C_2, \dots, C_n)$

Syntax of DRC in DBMS

$\{c_1, c_2, \dots, c_n \mid F(c_1, c_2, \dots, c_n)\}$

Let us assume the same Product table in the database as follows:

Product_id	Product Category	Product Name	Product Unit Price
8	New	TV Unit 1	\$100
10	New	TV Unit 2	\$120
12	Existing	TV Cabinet	\$77

DRC for the product name attribute from the Product table needs where the product id is 10, It will be demoted as:

$\{ \langle \text{Product Name}, \text{Product_id} \rangle \mid \in \text{Product} \wedge \text{Product_id} > 10 \}$

The result of the domain relational calculus for the Product table will be

Product_id	Product Name
10	TV Unit 2

Some of the commonly used logical operator notations for DRC are \wedge for AND, \vee for OR, and \neg for NOT. imilarly, the mathematical symbol \in refers to the relation “is an element of” or known as the set membership.

EXPRESSIVE POWER OF ALGEBRA AND CALCULUS (2)

- A query language is said to be relationally complete if it can express all the queries that can be expressed in relational algebra.
- SQL is relationally complete.
- Every query that can be expressed using a safe relational calculus query can be also be expressed as a relational algebra query.
- SQL provides additional expressive power beyond relational algebra.

48

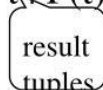


Tuple Relational Calculus

What ? represent tuple calculus expression to specify user's request

Basic Concepts

A query in tuple relational calculus is expressed as

$\{ t \mid P(t) \}$ t : tuple variable
 $P(t)$: well-formed formula (WFF) = condition
 t in $P(t)$: bound variable

⇒ A well-formed formula produces True or False given an interpretation

- Well-Formed Formula (WFF)

Atomic formulas connected by logical connectives, AND, OR, NOT and quantified by quantifiers, \exists (existential quantifier), \forall (universal quantifier)



Unsafe Queries, Expressive Power

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called unsafe.
 - e.g.,
$$\{S \mid \neg(S \in Sailors)\}$$
- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- Relational Completeness: Query language (e.g., SQL) can express every query that is expressible in relational algebra/calculus.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry

25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds at least one of the following conditions for every non-trivial function dependency $X \rightarrow Y$.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich

666	John	462007	MP	Bhopal
-----	------	--------	----	--------

Super key in the table above:

1. {EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
---------	-----------	----------

201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency $X \rightarrow Y$, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
--------	----------

D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. $EMP_ID \rightarrow EMP_COUNTRY$
2. $EMP_DEPT \rightarrow \{DEPT_TYPE, EMP_DEPT_NO\}$

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: $\{EMP_ID, EMP_DEPT\}$

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency $A \twoheadrightarrow B$, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing

21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

SEMESTER	SUBJECT
----------	---------

Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. $X \rightarrow Y$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

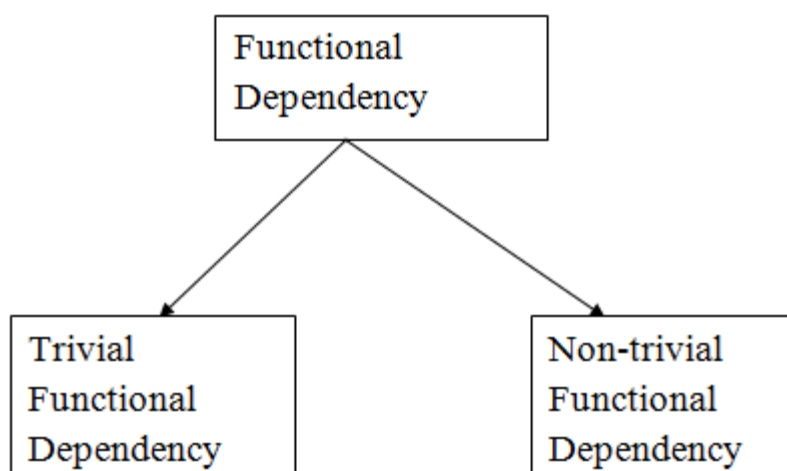
Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

1. $\text{Emp_Id} \rightarrow \text{Emp_Name}$

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency



1. Trivial functional dependency

- $A \rightarrow B$ has trivial functional dependency if B is a subset of A .
- The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. $\{Employee_id, Employee_Name\} \rightarrow Employee_Id$ is a trivial functional dependency as
3. Employee_Id is a subset of $\{Employee_Id, Employee_Name\}$.
4. Also, $Employee_Id \rightarrow Employee_Id$ and $Employee_Name \rightarrow Employee_Name$ are trivial dependencies too.

2. Non-trivial functional dependency

- $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A .
- When $A \cap B = \text{NULL}$, then $A \rightarrow B$ is called as complete non-trivial.

Example:

1. $ID \rightarrow Name$,
2. $Name \rightarrow DOB$

Inference Rule (IR):

- The Armstrong's axioms are the basic inference rule.
- Armstrong's axioms are used to conclude functional dependencies on a relational database.
- The inference rule is a type of assertion. It can apply to a set of FD(functional dependency) to derive other FD.
- Using the inference rule, we can derive additional functional dependency from the initial set.

The Functional dependency has 6 types of inference rule:

1. Reflexive Rule (IR_1)

In the reflexive rule, if Y is a subset of X , then X determines Y .

1. If $X \supseteq Y$ then $X \rightarrow Y$

Example:

1. $X = \{a, b, c, d, e\}$

2. $Y = \{a, b, c\}$

2. Augmentation Rule (IR_2)

The augmentation is also called as a partial dependency. In augmentation, if X determines Y, then XZ determines YZ for any Z.

1. If $X \rightarrow Y$ then $XZ \rightarrow YZ$

Example:

1. For $R(ABCD)$, if $A \rightarrow B$ then $AC \rightarrow BC$

3. Transitive Rule (IR_3)

In the transitive rule, if X determines Y and Y determine Z, then X must also determine Z.

1. If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

4. Union Rule (IR_4)

Union rule says, if X determines Y and X determines Z, then X must also determine Y and Z.

1. If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$

Proof:

1. $X \rightarrow Y$ (given)

2. $X \rightarrow Z$ (given)

3. $X \rightarrow XY$ (using IR_2 on 1 by augmentation with X. Where $XX = X$)

4. $XY \rightarrow YZ$ (using IR_2 on 2 by augmentation with Y)

5. $X \rightarrow YZ$ (using IR_3 on 3 and 4)

5. Decomposition Rule (IR_5)

Decomposition rule is also known as project rule. It is the reverse of union rule.

This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

1. If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$

Proof:

1. $X \rightarrow YZ$ (given)

2. $YZ \rightarrow Y$ (using IR_1 Rule)

3. $X \rightarrow Y$ (using IR_3 on 1 and 2)

6. Pseudo transitive Rule (IR_6)

In Pseudo transitive Rule, if X determines Y and YZ determines W, then XZ determines W.

1. If $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$

Proof:

1. $X \rightarrow Y$ (given)

2. $WY \rightarrow Z$ (given)

3. $WX \rightarrow WY$ (using IR_2 on 1 by augmenting with W)

4. $WX \rightarrow Z$ (using IR_3 on 3 and 2)

Lossless Decomposition in DBMS

Lossless join decomposition is a decomposition of a relation R into relations R1, R2 such that if we perform a natural join of relation R1 and R2, it will return the original relation R. This is effective in removing redundancy from databases while preserving the original data...

In other words by lossless decomposition, it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.

In Lossless Decomposition, we select the common attribute and the criteria for selecting a common attribute is that the common attribute must be a candidate key or super key in either relation R1, R2, or both.

Decomposition of a relation R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F^+ (Closure of functional dependencies)

$R1 \cap R2 \rightarrow R1$

OR

$R1 \cap R2 \rightarrow R2$

Let R (A, B, C, D) be a relational schema with the following functional dependencies:

$A \rightarrow B, B \rightarrow C,$

$C \rightarrow D$ and $D \rightarrow B.$

The decomposition of R into

(A, B), (B, C), (B, D)

(C, D)

$f: R(A, B, C, D)$

Lossless Join and Dependency Preserving Decomposition

Decomposition of a relation is done when a relation in relational model is not in appropriate normal form. Relation R is decomposed into two or more relations if decomposition is lossless join as well as dependency preserving.

Lossless Join Decomposition

If we decompose a relation R into relations R1 and R2,

- Decomposition is lossy if $R1 \bowtie R2 \supset R$
- Decomposition is lossless if $R1 \bowtie R2 = R$

To check for lossless join decomposition using FD set, following conditions must hold:

1. Union of Attributes of R1 and R2 must be equal to attribute of R.
Each attribute of R must be either in R1 or in R2.
 $Att(R1) \cup Att(R2) = Att(R)$
2. Intersection of Attributes of R1 and R2 must not be NULL.
 $Att(R1) \cap Att(R2) \neq \emptyset$
3. Common attribute must be a key for at least one relation (R1 or R2)
 $Att(R1) \cap Att(R2) \rightarrow Att(R1) \text{ or } Att(R1) \cap Att(R2) \rightarrow Att(R2)$

For Example, A relation R (A, B, C, D) with FD set{A→BC} is decomposed into R1(ABC) and R2(AD) which is a lossless join decomposition as:

1. First condition holds true as $Att(R1) \cup Att(R2) = (ABC) \cup (AD) = (ABCD) = Att(R)$.
2. Second condition holds true as $Att(R1) \cap Att(R2) = (ABC) \cap (AD) \neq \emptyset$

3. Third condition holds true as $\text{Att}(R1) \cap \text{Att}(R2) = A$ is a key of $R1(ABC)$ because $A \rightarrow BC$ is given.

Dependency Preserving Decomposition

GATE Question:

Consider a schema $R(A,B,C,D)$ and functional dependencies $A \rightarrow B$ and $C \rightarrow D$. Then the decomposition of R into $R1(AB)$ and $R2(CD)$ is [GATE-CS-2001]

- A. dependency preserving and lossless join
- B. lossless join but not dependency preserving
- C. dependency preserving but not lossless join
- D. not dependency preserving and not lossless join

Answer: For lossless join decomposition, these three conditions must hold true:

- 1. $\text{Att}(R1) \cup \text{Att}(R2) = ABCD = \text{Att}(R)$
- 2. $\text{Att}(R1) \cap \text{Att}(R2) = \Phi$, which violates the condition of lossless join decomposition. Hence the decomposition is not lossless.

For dependency preserving decomposition,

$A \rightarrow B$ can be ensured in $R1(AB)$ and $C \rightarrow D$ can be ensured in $R2(CD)$. Hence it is dependency preserving decomposition.

So, the correct option is C.

Lossless-join decomposition is a process in which a relation is decomposed into two or more relations. This property guarantees that the extra or less tuple generation problem does not occur and no information is lost from the original relation during the decomposition. It is also known as non-additive join decomposition.

When the sub relations combine again then the new relation must be the same as the original relation was before decomposition.

Consider a relation R if we decomposed it into sub-parts relation $R1$ and relation $R2$.

The decomposition is lossless when it satisfies the following statement –

- If we union the sub-Relation R1 and R2 then it must contain all the attributes that are available in the original relation R before decomposition.
- Intersections of R1 and R2 cannot be Null. The sub relation must contain a common attribute. The common attribute must contain unique data.

The common attribute must be a super key of sub relations either R1 or R2.

Here,

$R = (A, B, C)$

$R1 = (A, B)$

$R2 = (B, C)$

The relation R has three attributes A, B, and C. The relation R is decomposed into two relation R1 and R2.

R1 and R2 both have 2-2 attributes.

The common attributes are B.

The Value in Column B must be unique. if it contains a duplicate value then the Lossless-join decomposition is not possible.

Draw a table of Relation R with Raw Data –

R (A, B, C)

A	B	C
12	25	34
10	36	09
12	42	30

It decomposes into the two sub relations –

R1 (A, B)

A	B
12	25
10	36

A	B
12	42

R₂ (B, C)

B	C
25	34
36	09
42	30

Now, we can check the first condition for Lossless-join decomposition.

The union of sub relation R₁ and R₂ is the same as relation R.

R₁ ∪ R₂ = R

We get the following result –

A	B	C
12	25	34
10	36	09
12	42	30

The relation is the same as the original relation R. Hence, the above decomposition is Lossless-join decomposition.

What is Schema Refinement?

- **Schema Refinement** is the study of what should go where in a DBMS, or, which schemas are best to describe an application.
- For example, consider this schema

EmpDept

<u>EID</u>	Name	DeptID	DeptName
A01	Ali	12	Wing
A12	Eric	10	Tail
A13	Eric	12	Wing
A03	Tyler	12	Wing

- Versus this one:

Emp

<u>EID</u>	Name	DeptID
A01	Ali	12
A12	Eric	10
A13	Eric	12
A03	Tyler	12

Dept

<u>DeptID</u>	DeptName
12	Wing
10	Tail

- Which schema do you think is best? Why?