Parul®
University

DIGITAL LEARNING CONTENT

Database Management System

Chapter:4 Relational Algebra

Shaikh Amin Farooqbhai,
Asst. Prof., CSE Dept, PIET

Chapter:4 Relational Data Model

4	**Relational Data Model:** **Relational Data Model:** Introduction, Degree, Cardinality. **Constraints & Keys:** Primary Key, Foreign Key, Super Key, Candidate Key, Not Null Constraint, Check Constraint. **Relational Algebra Operations:** Selection, Projection, Cross-Product, Rename, Joins (Natural & Outer Join), Set Operators (Union, Intersection, Set Difference), Aggregate Functions.	**10**	**4**

Parul®
University

Keys:

- Tuples in a relational model represents individual entities or relationships.
- There must be some way **to distinguished** one tuple from another within a relation.
- Tuples are distinguished based on data values stored in them.
- No any two tuples in relation have **exactly the same value** for **all attributes.**

Parul®
University

Keys:

- To identify each tuple uniquely in relation, several keys have been defined:

- ***Super Key***
  - **"A super key is a set of one or more attributes that allows to identify each tuple uniquely in relation."**
  - For example in Customer relation cid, cname, address, **cid** attribute can distinguish each tuple from another, So, **cid** is a super key for Customer relation

Parul® University

Keys:

- ***Super Key***
  - If K is a super key, then any super set of K is super key. For example, combination of cid with any other attribute say – **cid** and **contact_no** is super key for the relation Customer.
  - Two customers may have same name, so attribute cname cannot identify each and every tuple uniquely.
  - But combination of **cname** and **address** can be considered as super key.

Keys:

- **Relation Key:**
  - **“A super key for which no subset is a super key is called a relation key.”**
  - In simple, relation key is a **minimal super key.**
  - A relation key is also referred as **‘key’** only.
  - A relation key is **sufficient to identify** each and every **tuple uniquely** within a relation.
  - For example combination of **cid** and **contact_no** is super key, but ony cid can alone identify entities uniquely.
  - A super key **cannot have null or duplicate values.**

Keys:

- Some times there can be more than one relation keys for the same relation.
- I.e {cid} as well as {cname, address} are relation key.

Keys:

- **Candidate key:**
  - **“If relation contains more than one relation keys, then they each are called candidate key.”**
  - For example, {cname, address} each key is called candidate key.
- **Primary key:**
  - **“A primary key is candidate key that is chosen by database designers to identify tuple uniquely in relation.”**
  - For example, if **cid key is chosen by database designers to identify tuple uniquely, then** cid **is referred as** primary key.

Keys:

- **Alternate Key**
  - **“An alternate key is candidate key that is not chosen by database designers to identify tuples uniquely in a relation.”**
  - For example if **cid** is chosen as primary key, then {cname, address} are alternate key for the same relation.
- **Foreign Key**
  - **“A foreign key is a set of one or more attributes whose values are derived from the key attribute of another relation.”**
  - A foreign key can have only those values which appear in another relation where it is defined as **candidate key**.

Parul® University

Keys:

Customer:

cid	name	society	city
C01	Riya	Amul Aavas	Anand
C02	Jiya	Sardar Colony	Karamsad
C03	Piya	Marutisadan	VVNagar
C04	Diya	Saral Society	Anand
C05	Tiya	Birla Gruh	VVNagar

Account:

ano	balance	bname*
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Account_Holder:

cid	ano
C01	A01
C02	A02
C03	A03
C04	A04
C05	A05
C02	A04

[*: 'vvn', 'ksad' and 'anand' refers to three cities of Gujarat state namely 'Vallabh Vidyanagar', 'Karamsad', and 'Anand' respectively.]

Constraints:

- Integrity constraints are the set of rules imposed on database to maintain quality of data.
- Integrity constraints ensures that any manipulation on data does not affect the data integrity.
- It secures the data management.
- Various Integrity Constraints are:
  1. Check
  2. Not null
  3. Unique Key
  4. Primary key
  5. Foreign key

Check Constraints:

- Syntax: columnName datatype(size) check (condition);
- Example:
  - Create table Account (ano varchar(10), balance NUMBER(20) check (NOT (balance<0)), bname varchar(10));
- If we try to insert negative values in balance column, it will restrict the query and will generate error.

Check Constraints:

- It is used to implement business rules.
- It is also referred as business rule constraint.
- Examples:
  - A balance in any account should not be negative value in any situation.
  - An account number must start with “A” and so on.
- **Syntax: columnName datatype(size) check (condition);**

Structure of Relational Database:

- A ***Relational Database Management System (RDBMS)*** is based on the ***relational data model.***

- In relational model, ***tables are represented both data and relationships among those data.***

- So, relational database consists of ***collection of tables.***

Structure of Relational Database:

- Each ***table*** is assigned a ***unique name.***
- A table ***contains fixed number of columns.***
- Each column ***within a table has a unique name, called column header.***
- A table also contains ***variable number of rows, records, which contains data values.***
- A table represents ***an entity set or relationship set.***
- Columns of table ***represents attributes.***

Structure of Relational Database:

Customer:

cid	name	society	city
C01	Riya	Amul Aavas	Anand
C02	Jiya	Sardar Colony	Karamsad
C03	Piya	Marutisadan	VVNagar
C04	Diya	Saral Society	Anand
C05	Tiya	Birla Gruh	VVNagar

Account:

ano	balance	bname*
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Account_Holder:

cid	ano
C01	A01
C02	A02
C03	A03
C04	A04
C05	A05
C02	A04

[*: 'vvn', 'ksad' and 'anand' refers to three cities of Gujarat state namely 'Vallabh Vidyanagar', 'Karamsad', and 'Anand' respectively.]

Structure of Relational Database:

- A customer table contains information about various customers of bank. Each row in a customer table represents one individual customer.
- Account table contains information about various accounts. Each row n account table represents one individual account.
- Each row provides relationship among values it contains.
- For example, account number ‘A01’ has balance 5000 Rs. And it belongs to ‘vvn’ branch.

Concept of Relation:

- The word ***"Relational"*** is derived from ***"Relation".*** And, "***Relation***" is taken from the mathematical concept of ***Relation*****.**
- In mathematics, a relation is defined as subset of a Cartesian product of two or more sets.

  A = {a, b, c}

  B = {1, 2, 3})

So, A×B = {(a,1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3), (c, 1), (c, 2), (c,3)}.

- Any subset of A×B, for example {(a,1),(b,3),(c,2)} is considered as relation of A and B.

Concept of Relation:

- Now, consider some table of banking system, Such as

  Account_Holder.

- This table has two columns,

  cid (Customer ID)

  ano (Account Number)

- This table represents relationship among ***customer*** and their ***accounts.***

Concept of Relation:

- Suppose there are 5 customers, C01, C02, C03, C04, C05 and 5 accounts, A01, A02, A03, A04, A05.
- These values form a ***Domain*** for ***cid*** and ***ano.***
- ***Domains*** are,

  cid = {C01, C02, C03, C04, C05}

  ano = {A01, A02, A03, A04, A05}
- The Cartesian product between these two domains. i.e ***cid×ano,*** will make a set of 25 members, such as,

  cid×ano = {(C01,A01), (C01,A02)… (C01,A05),

  (C02,A01), (C02,A02)… (C02,A05), ….

  (C05,A01), (C05,A02)… (C05,A05) }

Concept of Relation:

- Now, observe a table ***Account_Holder*** given below:

Cid	Ano
C01	A01
C02	A02
C03	A03
C04	A04
C05	A05
C02	A04

- This table represents one subset of Cartesian product between domains of cid and ano (***cid*×*ano*).**

Concept of Relation:

- Similarly, any table can be considered as a subset of Cartesian product among domains of its columns.
- Such kind of subset is considered as ***relation*** in mathematics.
- Due to this, tables are referred as relations also, And, a model which considers database as a collection of tables is referred as ***Relational*** model.

Features of a Relation:

- A table is referred as relation in the relational model.
- Various features of relation are as follows:
- ***Attribute:***
  - An attribute is name of a column in a relation.
  - For example, cid, name, society, city etc.
  - No two attributes in a relation can have the same name.
- ***Arity of Relation:***
  - The total numbers of attributes in a relation is referred as an ***arity/order*** of a relation.
  - ***Arity of relation*** will change infrequently.

Features of a Relation:

- ***Tuple:***
  - A ***row (or record)*** in a relation is referred as a ***Tuple.***
  - So, a ***relation*** can also be considered ***as a set of tuples.***
- ***Cardinality of Relation:***
  - The ***total numbers*** of ***tuples*** in a relation is referred as cardinality of a relation.
  - The cardinality of a relation changes frequently with each insertion and deletion of tuples.

Domain:

- **Domain** of an **attribute** is a **set of permitted values** for that attribute.

- For example, if bank is organized in three branches, named as ‘vvn’, ‘ksad’ and ‘anand’, then the domain of attribute ***bname*** *is* ***{‘vvn’,’kasad’,’anand’}.***

- Similarly, a domain for ***ano*** in Account table is a set of strings starting from ‘A’ followed by unique numerical value, such as ***{A01, A02, A03, A04 }.***

Domain:

- **Domain** may contain ***domain name, domain type, domain description*** and ***domain definition.***

ATTRIBUTE	DOMAIN NAME	TYPE	DESCRIPTION	DEFINITION
ANO	ACCOUNT NUMBER	STRING	SET OF STRINGS STARTING FROM ‘A’ FOLLOWED BY UNIQUE NUMBER	{STR| STR IS A STRING AS PER DESCRIPTION}
BAL	BALANCE	NUMBER	POSSIBLE VALUES FOR ALANCE FOR A BANK ACCOUNT.	{N| ANY POSITIVE NUMBER}
BNAME	BRANCH NAME	STRING	SET OF ALL POSSIBLE BRANCH NAMES	{‘ANAND’, ‘KSAD,’VVN’}

Domain:

- A ***domain*** is ***atomic*** if elements are considered to be ***indivisible*** units.
- For example, a domain ***bname*** is an atomic but ***address*** may not be atomic, as it can be divided into sub-parts.
- ***Domain*** of an attribute should be ***atomic*** (preferable).

Domain:

- Different attributes can have same ***domain.***
- For example, domains of two different attributes- ***Customer_name*** and ***Employee_name*** are ***same***, as they belong ***to name of some persons***.
- One possible element for any domain is the ***null*** value. It indicates that the ***value is not applicable, missing or not known.***

Database Schema and Database Instance:

- A ***database schema*** is an ***overall logical design of the database,*** while ***database instance*** is ***a collection of information stored in a database at a particular moment.***
- In relational database, ***a collection of relations form a database.***
- Here, ***a relation schema refers to a logical design of relation.***
- It is simply ***a list of attributes in a specific order.***

Parul® University

Database Schema and Database Instance:

- Consider a ***relation R with attributes A_1, A_2, A_3 and A_4. Schema for this relation*** can be represented by using various notations.
- 1. ***List out all attributes of a relation in brackets along with relation name.***

  Notation: **R{A_1,A_2,A_3, A_4}**

  Example: Account(ano, balance, bname)

  2. ***List out all attributes in form of A tuple along with relation name as given below:***

  RELATION: R

A_1	A_2	A_3	A_4

  ACCOUNT:

ANO	BALANCE	BNAME

- 3. ***List out all attributes in form of a column along with relation name as given below:***

ACCOUNT:

R:

A_1
A_2
A_3
A_4

Account:

Ano
Balance
bname

- ***Along all these three notations, any notation can be used according to convenience.***

Relational Algebra:

- **Relational algebra** is a **language** for **expressing relational database queries.**

- A ***query*** is a **statement requesting the retrival of information.**

- Relational algebra is a procedural query language. It requires to **specify what data to retrieve as well as how to retrieve those data.**

Relational Algebra:

- It requires user to instruct the system to **perform a sequence of operations on the database to retrieve desired data.**
- Relational algebra provides the base for commercial query language.
- It provides a way to extract data from the database.

Relational Algebra:

- Relational algebra consists of a set of different operations.
- These operations takes one or two relations as input and produce a new relation as output.
- These functions can be divided into two categories:
- Fundamental operations: Such as select, project, union, self-difference, Cartesian product, rename etc.

Relational Algebra:

- Secondary operations: Such as set-intersection, Natural join, Division and assignment.

- Based on **number of input relations,** these operations can also be categorized as given below:

  - **Unary Operations: Which does the operation on single relation, i.e Select, Project, rename etc.**

  - **Binary Operations: Which does the operation on pair of relations, i.e Union, Cartesian product etc.**

Relational Algebra:

- **Based on the number of i/p relations, these operations can also be categorized as:**
- **Unary Operations:**
  - Operates on a single relation.
  - Examples: Select, Project, Rename etc.
- **Binary Operations:**
  - Operates on pair of relation.
  - Examples: Union, Cartesian Product etc.

Relational Algebra:

- **On the basis of concept of set theory, operations can also be categorized as:**
- **Set-theoretic Operations:**
  - Considers that, relations are essentially set of rows.
  - Example: Union, Cartesian Product etc.
- **Native relational Operations:**
  - Focus on structure of rows
  - Example: Select, Project, Natural Join etc.

Parul® University

Relational Algebra:

Account:

Ano	balance	Bname
A01	5000	Vvn
A02	6000	Ksad
A03	7000	Anand
A04	8000	Ksad
A05	6000	vvn

Branch:

Bname	Baddress
Vvn	Mota bazar, VVNagar
Ksad	Chhota bazar, Karamsad
Anand	Nana bazar, Anand

Customer:

cid	Name
C01	Riya
C02	Jiya
C03	Piya
C04	Diya
C05	Tiya
C06	Palak
C07	Taral

Employee:

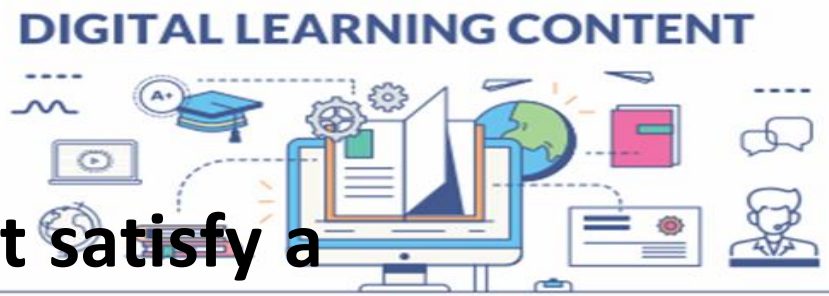
Eid	Name	Mngr_id
E01	Palak	E03
E02	Taral	E03
E03	Jiya	Null
E04	Saral	E03
E05	Zalak	E03
E06	Kamal	E03
E07	Riya	E03

Relational Algebra:

- Here the "**Account**" relation contains five tuples representing five different accounts.
- The **"Branch"** relation represents three different branches of bank.
- The customer relation has only two attributes – **cid and name-** representing customer id and name.
- **'Employee"** relation has seven diff. employees along with their employee id and their manager.
- An employee E03 has no any manager as he is manager himself.
- Also the same person can be **customer** as well as **employee.**

Parul® University

The Select operation:



- Operation: **Selects tuples from a relation that satisfy a given condition.**
- **Symbol: σ $(Sigma)$ Notation: $\sigma_{\text{condition}}$**<Relation>
- Operations: **=, ≠, <, >, ≤, ≥, AND, OR (V)**
- **For example;**

$\sigma_{(bname)}$ = "vvn" (Account)

Ano	Balance	Bname
A01	5000	Vvn
A02	6000	Vvn

The Project operation:

Parul University

- Operation: **Selects specified attributes of a relation.**
- **Symbol: Π (Pi) Notation:** $\Pi_{\text{attribute set}}$<Relation>
- **For example;**

$$\Pi_{(ano)} (\sigma_{(balance < 7000)} (\text{Account}))$$

Ano	balance	Bname
A01	5000	Vvn
A02	6000	Ksad
A03	7000	Anand
A04	8000	Ksad
A05	6000	vvn

Ano
A01
A02
A05

The Union operation:

Parul® University

- Operation: **Selects specified attributes of a relation.**
- **Symbol: $U(Union)$ Notation: $Relation\ U\ Relation2$**
- **Requirements: For relation R and S;**
  - Both have the same Arity, i.e total numbers of attributes
  - Domains of i^{th} attribute of R and S are same

$\Pi_{(name)}$ (Customer) U $\Pi_{(name)}$ (Employee)

The Set-Difference (Minus) operation:

- Operation: **Selects tuples those are in one relation but not in another relation.**
- **Symbol: -$(Minus)$Notation:$Relation1 - Relation2$**
- **Requirements: For relation R and S;**
  - Both have the same Arity, i.e total numbers of attributes
  - Domains of i^{th} attribute of R and S are same

$\Pi_{(name)}$ (Customer) - $\Pi_{(name)}$ (Employee)

The Set-Difference (Minus) operation:

Parul® University

cid	Name
C01	Riya
C02	Jiya
C03	Piya
C04	Diya
C05	Tiya
C06	Palak
C07	Taral

–

Eid	Name	Mngr_id
E01	Palak	E03
E02	Taral	E03
E03	Jiya	Null
E04	Saral	E03
E05	Zalak	E03
E06	Kamal	E03
E07	Riya	E03

Name
Piya
Diya
Tiya

The Set-Intersection operation:

- Operation: **Selects those tuples who are in both relation.**
- **Symbol:** ∩ $(\boldsymbol{Intersection})$**Notation:**$\boldsymbol{Relation1} \cap \boldsymbol{Relation2}$
- **Requirements: For relation R and S;**
  - Both have the same Arity, i.e total numbers of attributes
  - Domains of i^{th} attribute of R and S are same

Π$_{(\boldsymbol{name})}$ **(Customer) ∩ Π**$_{(\boldsymbol{name})}$ **(Employee)**

Parul® University

The Set-Intersection operation:

cid	Name
C01	Riya
C02	Jiya
C03	Piya
C04	Diya
C05	Tiya
C06	Palak
C07	Taral

∩

Eid	Name	Mngr_id
E01	Palak	E03
E02	Taral	E03
E03	Jiya	Null
E04	Saral	E03
E05	Zalak	E03
E06	Kamal	E03
E07	Riya	E03

Name
Riya
Jiya
Palak
Taral

The Cartesian-Product operation:

- Operation: **Selects those tuples who are in both relation.**
- **Symbol:** $\times$ $(cross)$ **Notation:** $Relation1 \times Relation2$
- **Resultant Relation:**
  - If Relation1 and Relation2 have n1 and n2 attributes, then resultant will have n1+n2, combining both attributes from both relations.
  - If both relation have same attributes having same name, it can be distinguished by combining ‘Relation-name, Attribute-name”
  - If Relation1 and Relation2 have n1 and n2 tuples, then relation will have n1*n2 attributes, combining each possible pair of tuples from both relations.

The Cartesian-Product operation:

- **Operation:**

$$Account \times Branch$$

Query: Account X Branch

ano	balance	Account. bname	Branch. bname	address
A01	**5000**	**vvn**	**vvn**	**Mota bazaar, VVNagar**
A01	5000	vvn	ksad	Chhota bazaar, Karamsad
A01	5000	vvn	anand	Nana bazaar, Anand
A02	6000	ksad	vvn	Mota bazaar, VVNagar
A02	**6000**	**ksad**	**ksad**	**Chhota bazaar, Karamsad**
A02	6000	ksad	anand	Nana bazaar, Anand
A03	7000	anand	vvn	Mota bazaar, VVNagar
A03	7000	anand	ksad	Chhota bazaar, Karamsad
A03	**7000**	**anand**	**anand**	**Nana bazaar, Anand**
A04	8000	ksad	vvn	Mota bazaar, VVNagar
A04	**8000**	**ksad**	**ksad**	**Chhota bazaar, Karamsad**
A04	8000	ksad	anand	Nana bazaar, Anand
A05	**6000**	**vvn**	**vvn**	**Mota bazaar, VVNagar**
A05	6000	vvn	ksad	Chhota bazaar, Karamsad
A05	6000	vvn	anand	Nana bazaar, Anand

Figure 5.10: Result of Account X Branch

The Cartesian-Product operation:

- As you can see here for many tuples, values of ranch name are no same.
- For example, second row combines account of ‘vvn’ with ‘ksad’, such tuples provide inconsistent data and they should be removed from the resultant relation.
- To defeat such kind of problems, Cartesian product can be combined with “Select” and “Project” statement.

Query: $\sigma_{\text{Account.bname} = \text{Branch.bname}}$ (Account X Branch)

utput

ano	balance	Account. bname	Branch. bname	address
A01	5000	vvn	vvn	Mota bazaar, VVNagar
A02	6000	ksad	ksad	Chhota bazaar, Karamsad
A03	7000	anand	anand	Nana bazaar, Anand
A04	8000	ksad	ksad	Chhota bazaar, Karamsad
A05	6000	vvn	vvn	Mota bazaar, VVNagar

Figure 5.11: Only consistent data from Account X Branch

The Cartesian-Product operation:

- Combination of Relational Algebra operations can yield quite useful information.
- As shown Cartesian product alone has almost no use, but **in combination with select and project operations, it can be retrieved information spread over multiple relations.**

Example 9: Find branch address for account having account number 'A01'.

Query:

$\Pi_{(baddress)} (\sigma_{ano="A01"} (\sigma_{Account.bname=Branch.bname} (Account \times Branch)))$

Output:

baddress
Mota bazaar, VVNagar

Figure 5.12: Branch address for account "A01"

The Rename Operation

Parul® University

DIGITAL LEARNING CONTENT

- This is a unary operator which changes attribute names for a relation without changing any values.
- Renames relation as well as its attributes.

Paternity

Father	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael

$\rho_{Father \rightarrow Parent}$(Paternity)

Parent	Child
Adam	Cain
Adam	Abel
Abraham	Isaac
Abraham	Ishmael

Operation: Renames relation as well as its attributes.

Symbol: ρ (rho)

Notation: ρ_x (R) : Renames relation R to x.

$\rho_{x(A1,A2,...An)}$ (R) : Renames relation R to x and its attributes to A1, A2,…,An.

The following examples clarify the use of rename operation.

The Rename Operation

Parul® University

DIGITAL LEARNING CONTENT

- Find largest balance from Account relation.
- Step 1: Compare all branches with each other and find balances which are not highest one.
- For this, take Cartesian product of Account relation with itself and compare balances.
- Rename is useful here to distinguish two balances:

Query:

$$\Pi_{(a1.balance)} (\sigma_{a1.balance < a2.balance} (\rho_{a1} (Account) \times \rho_{a2} (Account)))$$

Result of Step-1:

a1. balance
5000
6000
7000

Figure 5.13: Result of step-1.

The Rename Operation

Parul® University

DIGITAL LEARNING CONTENT

- Find largest balance from Account relation.
- Step 2: Now use the set-difference operation to find out the largest balance.

Query: $\Pi_{(balance)}$ (Account) –

$\Pi_{(a1.balance)}$ ($\sigma_{a1.balance < a2.balance}$ (ρ_{a1} (Account) X ρ_{a2} (Accou

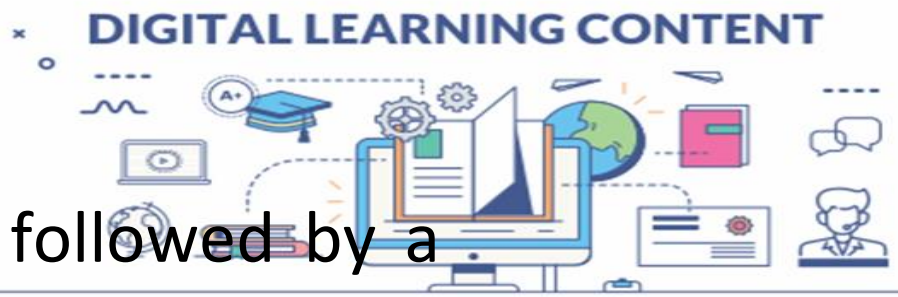
Result of Step-2:

balance
8000

Figure 5.14: The largest balance from Account relation.

Parul® University

Join



- Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

  - Theta (θ) Join
  - Natural Join (⋈)
  - Left Outer Join(R Left Outer Join S)
  - Right Outer Join: (R Right Outer Join S)
  - Full Outer Join: (R Full Outer Join S)

Parul® University

Join

- **Theta Join, Equijoin, and Natural Join are called inner joins.** An inner join includes **only those tuples with matching attributes** and the **rest are discarded** in the resulting relation.

- Therefore, we need to use **outer joins** to **include all the tuples** from the participating relations in the resulting relation. There are three kinds of outer joins – **left outer join, right outer join, and full outer join.**

Parul® University

Natural Join

- To retrieve consistent and useful information from multiple relations, it was necessary to combine Cartesian production operation with select operation for equality check on common attributes.
- Natural join does not use any comparison operator.
- It does not concatenate the way a Cartesian product does.
- We can perform a Natural **Join only if there is at least one common attribute** that exists **between two relations.**
- In addition, the **attributes must have the same name and domain**.
- Natural join acts on those **matching attributes** where the **values of attributes in both the relations are same**.

Parul® University

Natural Join

Course:

EID	Course	Dept.
E01	DBMS	CE
E05	MT	ME
E09	BE	EE

⋈

Faculty:

Dept.	Faculty
CE	AMIT
ME	HARSHIL
EE	KRINA

Course ⋈ Faculty

Dept.	EID	Course	Faculty
CE	E01	DBMS	AMIT
ME	E05	MT	HARSHIL
EE	E09	BE	KRINA

Equi Join/Theta (θ) Join

- Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol θ.
- R1 ⋈ θ R2
- R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is R1 ∩ R2 = Φ.

Student		
SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects	
Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Parul® University

Equi Join/Theta (θ) Join

Student

SID	Name	Std
101	Alex	10
102	Maria	11

Subjects

Class	Subject
10	Math
10	English
11	Music
11	Sports

$$STUDENT \bowtie_{Student.Std = Subject.Class} SUBJECT$$

Student_detail

SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Left Outer⟕Join(R S)

- All the tuples from the Left relation, R, are included in the resulting relation.

- If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Parul® University

Left Outer ⋈ Join(R S)

Left	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right	
A	**B**
100	Alex
102	Maya
104	Mira

Courses ⋈ HoD			
A	**B**	**C**	
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer ⟕ Join(R S)

- All the tuples from the Right relation, S, are included in the resulting relation.

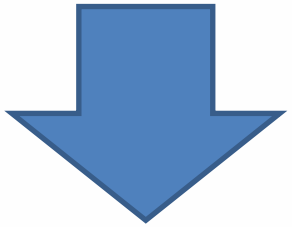
- If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Parul® University

Right Outer ⋈ Join(R S)

Left	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right	
A	**B**
100	Alex
102	Maya
104	Mira



Courses ⋈ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer ⟗ Join(R S)

- All the tuples from both participating relations are included in the resulting relation.
- If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

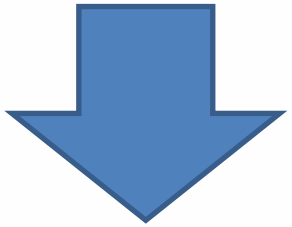
Parul® University

DIGITAL LEARNING CONTENT

Full Outer ⟗ Join(R ⟗ S)

Left	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right	
A	**B**
100	Alex
102	Maya
104	Mira



Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

Constraints:

- SQL constraints are used to specify **rules** for the data in a table.
- Constraints are used to **limit the type of data that can go into a table**.
- This ensures the **accuracy and reliability** of the data in the table.
- If there is any violation between the constraint and the data action, **the action is aborted.**
- Constraints can be **column level or table level.**
- Column level constraints apply to a column, and table level constraints apply to the whole table.

Constraints in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition

Constraints in SQL:

- NOT NULL
- By default, a column can hold **NULL** values.
- The NOT NULL constraint enforces a **column to NOT accept NULL values.**
- This enforces a field to **always contain a value,** which means that you cannot insert a new record, or update a record without adding a value to this field.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

Constraints in SQL:

- Unique
- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

Constraints in SQL:

- PRIMARY KEY
- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

Constraints in SQL:

- FOREIGN KEY
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

Constraints in SQL:

- CHECK
- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a column it will allow only certain values for this column.
- If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

Aggregate Functions:

- An aggregate function is a function that performs a calculation on **a set of values**, and returns a single value.

- Aggregate functions are often used with the **GROUP BY clause of the SELECT statement.** The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

Aggregate Functions:

- The most commonly used SQL aggregate functions are:

- MIN() - returns the smallest value within the selected column
- MAX() - returns the largest value within the selected column
- COUNT() - returns the number of rows in a set
- SUM() - returns the total sum of a numerical column
- AVG() - returns the average value of a numerical column

Aggregate Functions:

- The SQL MIN() and MAX() Functions

SELECT MIN(column_name)
FROM table_name
WHERE condition;

SELECT MAX(column_name)
FROM table_name
WHERE condition;

Aggregate Functions:

- COUNT()

- Find the total number of rows in the Products table:

SELECT COUNT(*) FROM Products;

Aggregate Functions:

- SUM()
- Find the total number of rows in the Products table:
- Syntax

SELECT SUM(column_name) FROM table_name WHERE condition;

SELECT SUM(Quantity) FROM OrderDetails;

Aggregate Functions:

- AVG()
- Syntax

  SELECT AVG(column_name) FROM table_name WHERE condition;
- Example

  SELECT AVG(Price) FROM Products;