DATA QUERY LANGUAGES

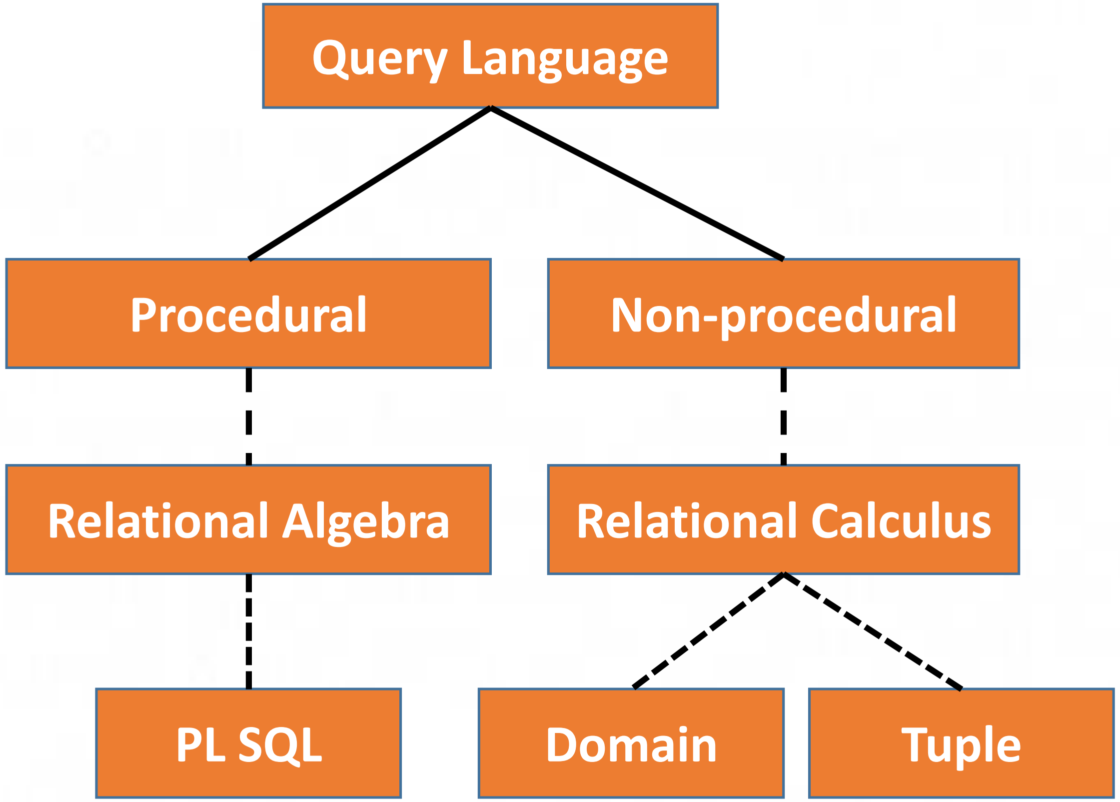
- ❑ Query languages, often known as DQLs or Data Query Languages, are computer languages that are used to make various queries in information systems and databases.
- ❑ A query language is a language in which user requests information from the database.
- ❑ **Procedural Query Language:** User instructs the system to perform a sequence of operations on the database to compute the desired result.

  For Example: Relational algebra

  Structure Query language (SQL) is based on relational algebra.
- ❑ **Non-procedural Query Language:** Information is retrieved from the database without specifying the sequence of operation to be performed. Users only specify what information is to be retrieved.

  For Example: Relational Calculus

  Query by Example (QBE) is based on Relational calculus



RELATIONAL ALGEBRA

- Relational Algebra came in 1970 and was given by Edgar F. Codd (Father of DBMS). It is also known as Procedural Query Language(PQL) as in PQL, a programmer/user has to mention two things, **"What to Do"** and **"How to Do"**.
- **Relational algebra:** It is a collection of operations to manipulate relations.
- Relational Algebra is a procedural query language. It consists of a set of operations that take one or two relations a input and produce a new relation as their result.
- It specifies the operations to be performed on existing relations to derive the result relations.
- Relational Algebra are usually divided into two groups.
  - Mathematical Set Operations e.g. Union, Intersection, Set Difference, Cartesian Product.
  - Relational Database Operations e.g. Select, Project, Rename, Join, Assignment.

RELATIONAL ALGEBRA

Relational Algebra

Set Operations

- $\cup$ Union
- $\cap$ Intersection
- — Set difference
- X Cartesian Product
- $\div$ Division

Relational Database

- σ Select
- ρ Rename
- $\prod$ Project
- $\bowtie$ Join

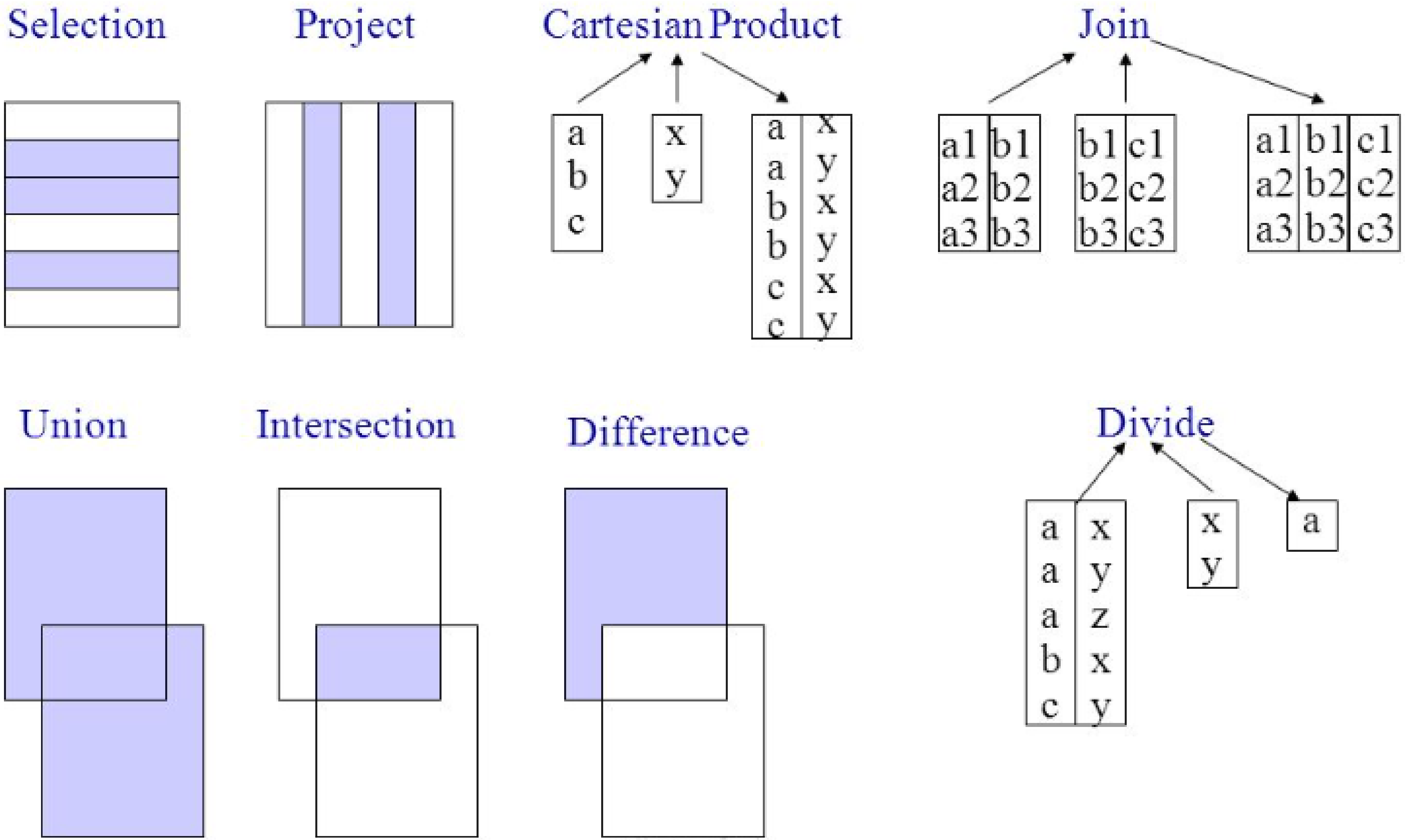
Set Functions

- Sum
- Avg
- Count
- Min
- Max
- Any

RELATIONAL ALGEBRA

- **Select:** *It returns a relation containing all tuples from specified relation that satisfy a condition.*
- **Project:** *It returns a relation containing all tuples that remain in a specified relation after specified attributes have been removed.*
- **Product:** *It returns a new relation that is an outcome of concatenation (that is chaining) of each tuple of one relation with each tuple of another relation.*
- **Join:** *It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations such as the two tuples contributing to a given combination have a common value for the common attributes of the two relations.*
- **Union:** *It returns a relation containing all tuples that appear in either or both of two specified relations.*
- **Intersect**: *It returns a relation containing all tuples that appear in both of two specified relations.*
- **Difference**: *It returns a relation containing all tuples that appear in the first not in second of the two specified relations.*
- **Divide:** *The division operator is used when we have to evaluate queries which contain the keyword 'all'. It permits to find values in an attribute of R that have all values of S in the attribute of the same name.*

RELATIONAL ALGEBRA



RELATIONAL ALGEBRA

- **Select Operator (σ):** *It returns a relation containing all tuples from specified relation that satisfy a condition.* It is denoted by sigma (σ).
- Syntax: $\sigma_p(R)$

  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 (∨), NOT (!). These relational can use as relational operators like =, ≠, ≥, <, >, ≤.
- <u>**Examples-**</u>
  - Select tuples from a relation "Books" where subject is "database"

    $\sigma_{\text{subject = "database"}}$ (Books)

    Select * from Books where subject='database';

  - Select tuples from a relation "Books" where subject is "database" and price is "450"

    $\sigma_{\text{subject = "database"} \wedge \text{ price = "450"}}$ (Books)

    Select * from Books where subject='database' and price=450;

RELATIONAL ALGEBRA

- Select tuples from a relation “Books” where subject is “database” and price is “450” or have a publication year after 2010

$$\sigma_{subject = \text{“database”} \wedge price = \text{“450”} \vee year > \text{”2010″}} (Books)$$

Select * from Books where subject=‘database’ and price=450 or year=2010;

Points to be remembered for Select operator

- We may use logical operators like ∧ , ∨ , ! and relational operators like = , ≠ , > , < , <= , >= with the selection condition.
- Selection operator only selects the required tuples according to the selection condition.
- Selection operator always selects the entire tuple. It can not select a section or part of a tuple.
- Selection operator is commutative in nature i.e.

$$\sigma_{A \wedge B} (R) = \sigma_{B \wedge A} (R)$$

- Degree of the relation from a selection operation is same as degree of the input relation.
- The number of rows returned by a selection operation is obviously less than or equal to the number of rows in the original table.

  Thus,

  Minimum Cardinality = 0, Maximum Cardinality = |R|

RELATIONAL ALGEBRA

- Project Operator (π) is a unary operator in relational algebra that performs a projection operation.
- It displays the columns of a relation or table based on the specified attributes.

    Syntax: $\pi_{<\text{attribute list}>}(R)$

- Example-

    Consider the following Student relation

ID	Name	Subject	Age
100	Ashish	Maths	19
200	Rahul	Science	20
300	Naina	Physics	20
400	Sameer	Chemistry	21

$\pi_{\text{Name, Age}}(\text{Student})$

Select name, age from student

Name	Age
Ashish	19
Rahul	20
Naina	20
Sameer	21

$\pi_{\text{ID , Name}}(\text{Student})$

Select ID, Name from Student

ID	Name
100	Ashish
200	Rahul
300	Naina
400	Sameer

RELATIONAL ALGEBRA

Points to be remembered for Project Operator

- The degree of output relation (number of columns present) is equal to the number of attributes mentioned in the attribute list.
- Projection operator automatically removes all the duplicates while projecting the output relation. So, cardinality of the original relation and output relation may or may not be same. If there are no duplicates in the original relation, then the cardinality will remain same otherwise it will surely reduce.
- If attribute list is a super key on relation R, then we will always get the same number of tuples in the output relation. This is because then there will be no duplicates to filter.
- Projection operator does not obey commutative property i.e.

$$\pi_{<list2>} (\pi_{<list1>} (R)) \neq \pi_{<list1>} (\pi_{<list2>} (R))$$

- Selection Operator performs horizontal partitioning of the relation. Projection operator performs vertical partitioning of the relation.
- There is only one difference between Project and Select operation of SQL. Projection operator does not allow duplicates while SELECT operation allows duplicates. To avoid duplicates in SQL, we use "distinct" keyword and write SELECT distinct. Thus, projection operator of relational algebra is equivalent to SELECT operation of SQL.

RELATIONAL ALGEBRA

- **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.

  Syntax: R X S

- Example-

  Consider the following relations

Employee

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Department

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

Employee X Department

Select Emp_name, Emp_id,dept_name From Employee, department;

Select Emp_name, Emp_id,dept_name from Employee Cross Join Department;

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

RELATIONAL ALGEBRA

- ❑ **Union Operator (∪):** *It returns a relation containing all tuples that appear in either or both of two specified relations.*

  Let R and S be two relations.

  Then-

  - R ∪ S is the set of all tuples belonging to either R or S or both.
  - In R ∪ S, duplicates are automatically removed.
  - Union operation is both commutative and associative.

- ❑ Example-

  Consider the following two relations R and S

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R ∪ S

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science
400	Kajol	French

Select * from R Union Select * from S

RELATIONAL ALGEBRA

- **Intersection Operator (∩):** *It returns a relation containing all tuples that appear in both of two specified relations.*

  Let R and S be two relations.

  Then-

  - R ∩ S is the set of all tuples belonging to both R and S.
  - In R ∩ S, duplicates are automatically removed.
  - Intersection operation is both commutative and associative.

- Example-

  Consider the following two relations R and S

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R ∩ S

ID	Name	Subject
100	Ankit	English

Select * from R Intersect Select * from S

RELATIONAL ALGEBRA

- **Difference Operator (-):** *It returns a relation containing all tuples that appear in the first not in second of the two specified relations.*

  Let R and S be two relations.
  Then-
  - R – S is the set of all tuples belonging to R and not to S.
  - In R – S, duplicates are automatically removed.
  - Difference operation is associative but not commutative.

- Example-

  Consider the following two relations R and S

Relation R

ID	Name	Subject
100	Ankit	English
200	Pooja	Maths
300	Komal	Science

Relation S

ID	Name	Subject
100	Ankit	English
400	Kajol	French

Relation R - S

ID	Name	Subject
200	Pooja	Maths
300	Komal	Science

Select * from R Minus Select * from S

RELATIONAL ALGEBRA

- Division Operation is represented by "division"(÷ or /) operator and is used in queries that involve keywords **"every"**, **"all"**, etc.
  Syntax : R(X,Y)/S(Y)
  Here,
  - R is the first relation from which data is retrieved.
  - S is the second relation that will help to retrieve the data.
  - X and Y are the attributes/columns present in relation. We can have multiple attributes in relation, but keep in mind that attributes of S must be a proper subset of attributes of R.
  - For each corresponding value of Y, the above notation will return us the value of X from tuple<X,Y> which exists **everywhere**.
- It's a bit difficult to understand this in a theoretical way, but you will understand this with an example.
- Let's have two relations, ENROLLED and COURSE. ENROLLED consist of two attributes STUDENT_ID and COURSE_ID. It denotes the map of students who are enrolled in given courses.
- COURSE contains the list of courses available.
- See, here attributes/columns of COURSE relation are a proper subset of attributes/columns of ENROLLED relation. Hence Division operation can be used here.

RELATIONAL ALGEBRA

Query 1: STUDENT_ID of students who are enrolled in **every** course.

ENROLLED(STUDENT_ID, COURSE_ID) ÷ COURSE(COURSE_ID)

ENROLLED

STUDENT_ID	COURSE_ID
Student_1	DBMS
Student_2	DBMS
Student_1	OS

÷

COURSE

COURSE_ID
DBMS
OS

→

ENROLLED ÷ COURSE

STUDENT_ID
Student_1

Query 2: Retrieve the name of subject that is taught in all courses.

SUBJECT(NAME, COURSE) ÷ COURSE(COURSE)

SUBJECT

NAME	COURSE
Systems	BCS
Database	BCS
Database	MCS
Algebra	MCS

÷

COURSE

COURSE
BCS
MCS

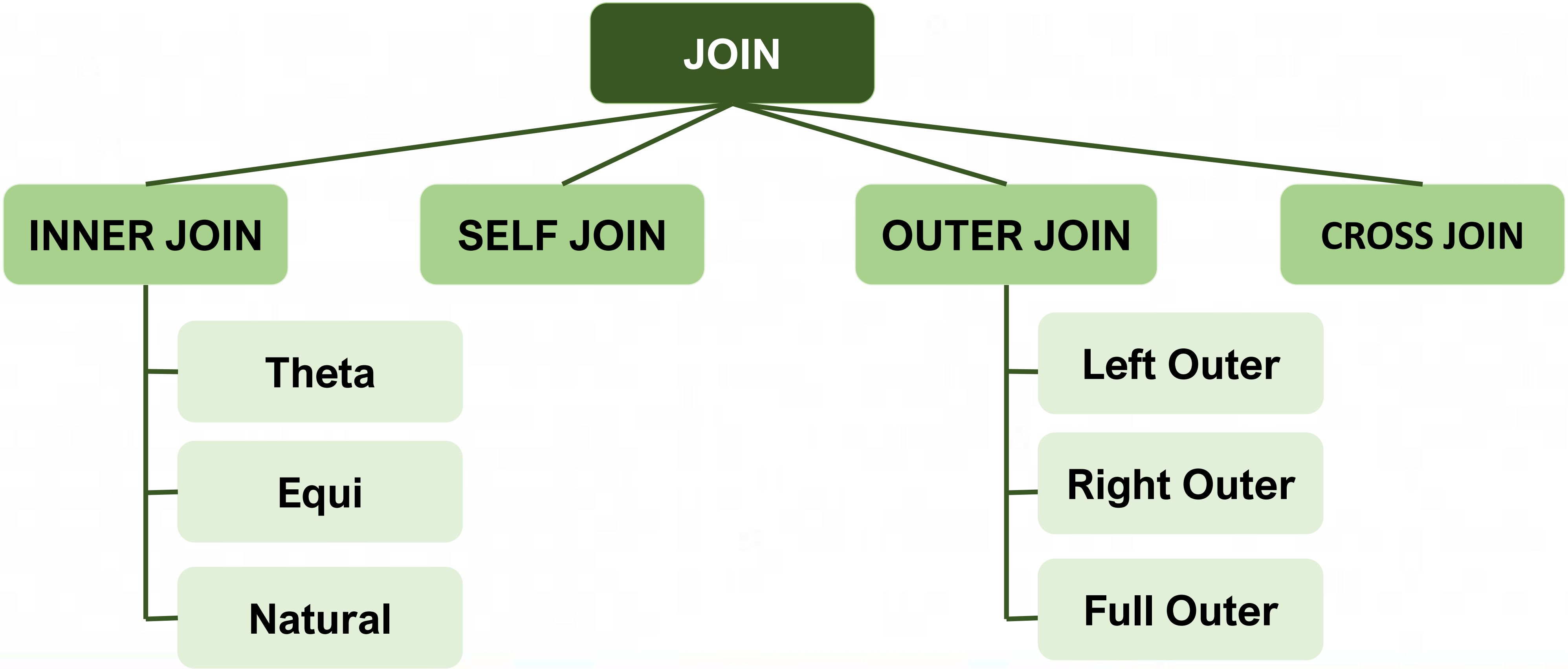
→

SUBJECT ÷ COURSE

NAME
Database

RELATIONAL ALGEBRA

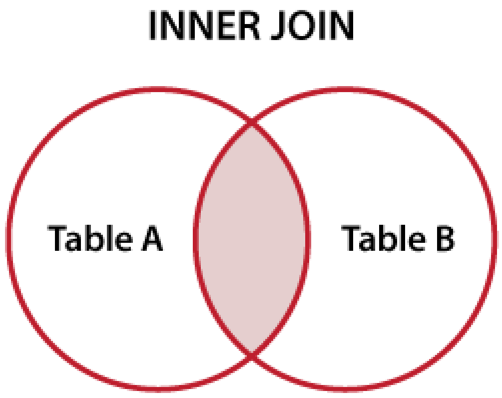
- **Join Operation:** *It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations such as the two tuples contributing to a given combination have a common value for the common attributes of the two relations.*
- Join Operation in DBMS are binary operations that allow us to combine two or more relations.
- They are further classified into two types: Inner Join, and Outer Join.



RELATIONAL ALGEBRA

- **Inner Join:** When we perform Inner Join, only those tuples returned that satisfy the certain condition. It is also classified into three types: Theta Join, Equi Join and Natural Join.
- **Theta Join (θ):** Theta Join combines two relations using a condition. This condition is represented by the symbol "theta"(θ). Here conditions can be inequality conditions such as >,<,>=,<=, etc. Notation : $R \bowtie_\theta S$, Where R is the first relation, S is the second relation, and θ is the condition.

Let there be a database of all the class 12th boys students in a school. Let's understand Theta Join with the Boys and Interest tables used above :



Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

RELATIONAL ALGEBRA

Theta Join -

Boys $\bowtie_{(\text{Boys.ID = Interest.ID and Interest.Sport = Chess and Boys.Percentage > 70})}$ Interest

So the condition here is
Boys.ID = Interest.ID and Interest.Sport = Chess and Boys.Percentage > 70

so while performing join, we will have to check this condition every time two rows are joined.

Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⋈θ Interest

ID	Name	Percentage	Gender	Sport
2	Rohit	85	M	Cricket
3	Amit	75	M	Cricket
6	Tejan	84	M	Chess

Select * from Boys Join Interest On Boys.ID=Interest.ID and Interest.Sport='Chess' and Boys.Percentage>70;

RELATIONAL ALGEBRA

Equi join is same as Theta Join, but the only condition is it only uses equivalence condition while performing join between two tables.

A ⋈(... = ...) B, where (... = ...) is the equivalence condition on any of the attributes of the joining table.

In the above example, what if we are told to find out all the students of class 12th who have interest in chess only?

We can perform Equi join as :

Equi join: Boys ⋈(Boys.ID = Interset.ID and Interest.Sport = Chess) Interest

Result after performing Equi join:

Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⋈(... = ...) Interest

ID	Name	Percentage	Gender	Sport
6	Tejan	84	M	Chess

Select * from Boys Join Interest
On Boys.ID=Interest.ID;

RELATIONAL ALGEBRA

Natural Join is also considered a type of inner join but it does not use any comparison operator for join condition. *It joins the table only when the two tables have at least one common attribute with same name and domain*.

In the result of the Natural Join the common attribute only appears once.

It will be more clear with help of an example :

What if we are told to find all the Students of class 12th and their sports interest we can apply Natural Join as : Boys ⋈ Interest

So when we perform Natural Join on table Boys and table Interest they both have a common attribute ID and have the same domain. So, the Result of Natural Join will be:

Boys

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

Interest

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⋈ Interest

ID	Name	Percentage	Gender	Sport
2	Rohit	85	M	Cricket
3	Amit	75	M	Chess
5	Saiz	65	M	Cricket
6	Tejan	84	M	Chess

Select * from Boys Natural Join Interest ;

RELATIONAL ALGEBRA

Outer Join

Outer Join in Relational algebra returns all the attributes of both the table depending on the condition. If some attribute value is not present for any one of the tables it returns NULL in the respective row of the table attribute.

It is further classified as:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Let's see how these Joins are performed.

Left Outer Join

It returns all the rows of the left table even if there is no matching row for it in the right table performing Left Outer Join.

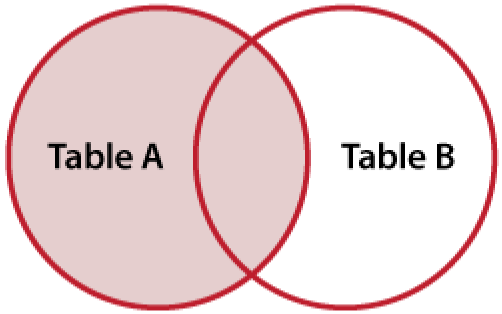
$$A ⟕ B$$

Let's perform Left Outer Join on table Boys and Interest and find out all the boys of class 12th and their sports interest.

RELATIONAL ALGEBRA

If we perform Left Outer Join on table Boys and table Interest such that Boys.ID = Interest.ID . Then Result of the Join will be:

LEFT OUTER JOIN



Boys ⟕ Interest

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys.ID	Boys.Name	Boys.Percentage	Interest.ID	Interest.Name	Interest.Gender	Interest.Sport
1	Rohan	56	NULL	NULL	NULL	NULL
2	Rohit	85	2	Rohit	M	Cricket
3	Amit	75	3	Amit	M	Cricket
4	Ravi	79	NULL	NULL	NULL	NULL
5	Saiz	65	5	Saiz	M	Cricket
6	Tejan	84	6	Tejan	M	Chess
7	Rishabh	75	NULL	NULL	NULL	NULL

Select * from Boys Left Outer Join Interest On Boys.ID=Interest.ID;

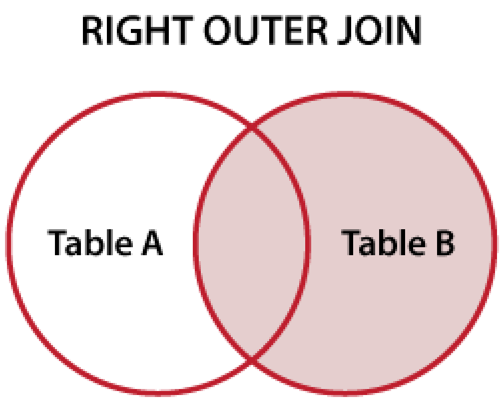
RELATIONAL ALGEBRA

Right Outer Join

It returns all the rows of the second table even if there is no matching row for it in the first table performing Right Outer Join.

A ⟖ B

Let's perform Right Outer Join on table Boys and Interest and find out all the boys of class 12th and their sports interest. If we perform Right Outer Join on table Boys and table Interest such that Boys.ID = Interest.ID . Then Result of the join will be:



RELATIONAL ALGEBRA

If we perform Right Outer Join on table Boys and table Interest such that Boys.ID = Interest.ID . Then Result of the join will be:

Clearly, we can observe that all the rows of the right table, i.e., table Interest is present in the result.

ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⟖ Interest

Boys.ID	Boys.Name	Boys.Percentage	Interest.ID	Interest.Name	Interest.Gender	Interest.Sport
3	Amit	75	3	Amit	M	Cricket
NULL	NULL	NULL	23	Aman	M	Chess
5	Saiz	65	5	Saiz	M	Cricket
NULL	NULL	NULL	10	Shreya	F	Badminton
6	Tejan	84	6	Tejan	M	Chess
NULL	NULL	NULL	15	Sakshi	F	Chess
2	Rohit	85	2	Rohit	M	Cricket

Select * from Boys Right Outer Join Interest On Boys.ID=Interest.ID;

RELATIONAL ALGEBRA

Full Outer Join

It returns all the rows of the first and second Table.

A ⟗ B

Clearly, we can observe that all the rows of the right table and left Table, i.e., Table B and A are present in the result.

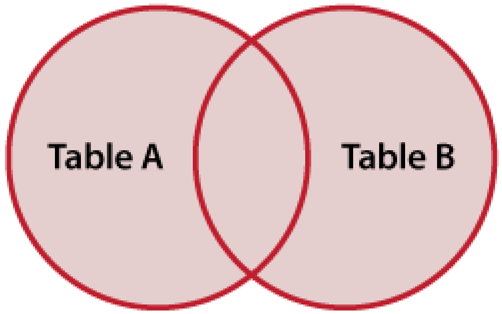
ID	Name	Percentage %
1	Rohan	56
2	Rohit	85
3	Amit	75
4	Ravi	79
5	Saiz	65
6	Tejan	84
7	Rishabh	75

ID	Name	Gender	Sport
3	Amit	M	Cricket
23	Aman	M	Chess
5	Saiz	M	Cricket
10	Shreya	F	Badminton
6	Tejan	M	Chess
15	Sakshi	F	Chess
2	Rohit	M	Cricket

Boys ⟗ Interest

Boys.ID	Boys.Name	Boys.Percentage	Interest.ID	Interest.Name	Interest.Gender	Interest.Sport
1	Rohan	56	NULL	NULL	NULL	NULL
2	Rohit	85	2	Rohit	M	Cricket
3	Amit	75	3	Amit	M	Cricket
4	Ravi	79	NULL	NULL	NULL	NULL
5	Saiz	65	5	Saiz	M	Cricket
6	Tejan	84	6	Tejan	M	Chess
7	Rishabh	75	NULL	NULL	NULL	NULL
NULL	NULL	NULL	23	Aman	M	Chess
NULL	NULL	NULL	10	Shreya	F	Badminton
NULL	NULL	NULL	15	Sakshi	F	Chess

FULL OUTER JOIN



Select * from Boys Full Outer Join Interest On Boys.ID=Interest.ID;

RELATIONAL CALCULUS

- Relational Calculus is a **non-procedural query language** used in database management systems (DBMS) to specify queries.

- Unlike relational algebra, which focuses on operations, **relational calculus specifies what to retrieve rather than how to retrieve it**.

- It is based on **predicate logic** and consists of formulas that define the required result set without specifying a step-by-step execution method.

- **Types of Relational Calculus**

  Relational Calculus is mainly divided into two types:
  - **Tuple Relational Calculus (TRC)** – Works with **tuples (rows)** in a relation.
  - **Domain Relational Calculus (DRC)** – Works with **domains (column values)** instead of tuples.

RELATIONAL CALCULUS

Tuple Relational Calculus (TRC)

- In TRC, queries are expressed using **tuple variables**.
- The result is a set of tuples that satisfy a given condition.
- The general form of TRC query: {t | P(t)}

  where:

  t is a **tuple variable** (representing a row in the table).

  P(t) is a **predicate (condition)** that must be true for t to be included in the result.

Example

Find the employees who work in the "HR" department:

$$\{t \mid t \in Employees \text{ AND } t.dept =' HR'\}$$

This retrieves all tuples t from the **Employees** table where the department is "HR".

Operators in TRC

- **Logical Operators:** AND (∧), OR (∨), NOT (¬)
- **Comparison Operators:** =, ≠, >, <, ≥, ≤
- **Existential (∃) and Universal (∀) Quantifiers**

RELATIONAL CALCULUS

Operators in TRC

- **Logical Operators:** AND (∧), OR (∨), NOT (¬)
- **Comparison Operators:** =, ≠, >, <, ≥, ≤
- **Existential (∃) and Universal (∀) Quantifiers**

Example with Existential Quantifier (∃)

Find employees who work in at least one department:

$$\{t \mid t \in Employees \text{ AND } \exists d(d \in Departments \text{ AND } t.dept\_id = d.dept\_id)\}$$

Here, **∃ d** ensures that an employee is included only if a matching department exists.

RELATIONAL CALCULUS

Domain Relational Calculus (DRC)

- In DRC, queries are expressed in terms of **column values (domains)** instead of tuples.
- The result is a set of values rather than complete tuples.
- The general form of DRC query:

  {(x1,x2,...,xn) | P(x1,x2,...,xn)}

  where:

  x1,x2,...,xn represent attribute values (domains).

  P(x1,x2,...,xn) is a condition that must be true for the values to be included in the result.

Example

Find the names of employees who work in the "HR" department:

$$\{e\_name \mid \exists d(emp\_id, e\_name, d\_id) \in Employees \text{ AND } (d\_id, dept\_name) \in Departments \text{ AND } dept\_name =' HR'\}$$

This retrieves only the **employee names** instead of entire tuples.

RELATIONAL CALCULUS

Operators in TRC

- **Logical Operators:** AND (∧), OR (∨), NOT (¬)
- **Comparison Operators:** =, ≠, >, <, ≥, ≤
- **Existential (∃) and Universal (∀) Quantifiers**

- **Example with Universal (∀) Quantifier**

  Find employees who work in every department:

$$\{e\_name \mid \forall d(d \in Departments \rightarrow \exists emp(emp \in Employees \text{ AND } emp.dept\_id = d.dept\_id))\}$$

Here, **∀ d** ensures that an employee works in every department.

RELATIONAL CALCULUS

Relational Algebra and Relational Calculus

Feature	Relational Algebra	Relational Calculus
Type	Procedural (specifies how to retrieve data)	Declarative (specifies what to retrieve)
Operators Used	Uses operators like SELECT (σ), PROJECT (π), JOIN (⋈), UNION (∪), etc.	Uses predicate logic, quantifiers (∃, ∀)
Flexibility	Less flexible, requires sequence of operations	More flexible, does not specify sequence
Implementation	Easier to implement in DBMS	More theoretical, used for query formulation

Relational Calculus

- **Declarative Query Language:** Focuses on describing the desired result rather than the retrieval process.
- **Based on Predicate Logic:** Uses conditions and quantifiers to filter data.
- **Supports Expressive Queries:** Can handle complex queries using existential and universal quantifiers.
- **Forms the Basis for SQL:** SQL is influenced by relational calculus, particularly in its use of predicates and logical expressions.

RELATIONAL CALCULUS

Exercises on Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC)

Given Database Schema

Employee(Emp_id, Emp_Name, Dept_id, Salary)

Department(Dept_id, Dept_Name)

Employee Table

emp_id	emp_name	dept_id	salary
1	Alice	10	50000
2	Bob	20	60000
3	Charlie	NULL	55000
4	David	10	65000
5	Emma	30	70000

Department Table

dept_id	dept_name
10	HR
20	Finance
30	IT
40	Marketing

1. Retrieve the names of all employees who earn more than 55,000.

$$\{t.emp\_name \mid t \in Employee \text{ AND } t.salary > 55000\}$$

$$\{e\_name \mid \exists emp\_id, dept\_id, salary\ (emp\_id, e\_name, dept\_id, salary) \in Employee \text{ AND } salary > 55000\}$$

RELATIONAL CALCULUS

2. Find employees who work in the HR department.

$$\{t.emp\_name \mid t \in Employee \text{ AND } \exists d(d \in Department \text{ AND } d.dept\_id = t.dept\_id \text{ AND } d.dept\_name =' HR')\}$$

$$\{e\_name \mid \exists emp\_id, dept\_id, salary, dept\_name\ (emp\_id, e\_name, dept\_id, salary) \in Employee \text{ AND } (dept\_id, dept\_name) \in Department \text{ AND } dept\_name =' HR'\}$$

3. Find the employees who do not belong to any department (i.e., dept_id is NULL).

$$\{t.emp\_name \mid t \in Employee \text{ AND } t.dept\_id = \text{NULL}\}$$

$$\{e\_name \mid \exists emp\_id, salary\ (emp\_id, e\_name, NULL, salary) \in Employee\}$$

4. Find all departments that have at least one employee.

$$\{d.dept\_name \mid d \in Department \text{ AND } \exists t(t \in Employee \text{ AND } t.dept\_id = d.dept\_id)\}$$

$$\{dept\_name \mid \exists dept\_id\ (dept\_id, dept\_name) \in Department \text{ AND } \exists emp\_id, e\_name, salary\ (emp\_id, e\_name, dept\_id, salary) \in Employee\}$$

5. Retrieve employees who work in every department.

$$\{t.emp\_name \mid t \in Employee \text{ AND } \forall d(d \in Department \rightarrow \exists e(e \in Employee \text{ AND } e.emp\_id = t.emp\_id \text{ AND } e.dept\_id = d.dept\_id))\}$$

$$\{e\_name \mid \forall dept\_id, dept\_name((dept\_id, dept\_name) \in Department \rightarrow \exists emp\_id, salary((emp\_id, e\_name, dept\_id, salary) \in Employee))\}$$

RELATIONAL CALCULUS

6. Retrieve the department names that have no employees.

$$\{d.dept\_name \mid d \in Department \text{ AND } \neg\exists t(t \in Employee \text{ AND } t.dept\_id = d.dept\_id)\}$$

$$\{dept\_name \mid \exists dept\_id(dept\_id, dept\_name) \in Department \text{ AND } \neg\exists emp\_id, e\_name, salary((emp\_id, e\_name, dept\_id, salary) \in Employee)\}$$

7. Find employees who work in Finance and earn more than ₹70,000.

$$\{t.Name \mid t \in Employee \text{ AND } t.Salary > 70000 \text{ AND } \exists d(d \in Department \text{ AND } d.Dept\_ID = t.Dept\_ID \text{ AND } d.Dept\_Name =' Finance')\}$$

$$\{Name \mid \exists Emp\_ID, Age, Gender, Salary\ (Emp\_ID, Name, Age, Gender, Salary, Dept\_ID) \in Employee \text{ AND } Salary > 70000 \text{ AND }$$
$$(Dept\_ID, Dept\_Name, Location) \in Department \text{ AND } Dept\_Name =' Finance'\}$$

8. Find the highest-paid employee in the company.

$$\{t.Name \mid t \in Employee \text{ AND } \neg\exists e(e \in Employee \text{ AND } e.Salary > t.Salary)\}$$

$$\{Name \mid \exists Emp\_ID, Age, Gender, Dept\_ID\ (Emp\_ID, Name, Age, Gender, Salary, Dept\_ID) \in Employee \text{ AND }$$
$$\neg\exists Emp\_ID1, Name1, Age1, Salary1\ (Emp\_ID1, Name1, Age1, Gender1, Salary1, Dept\_ID1) \in Employee \text{ AND } Salary1 > Salary\}$$