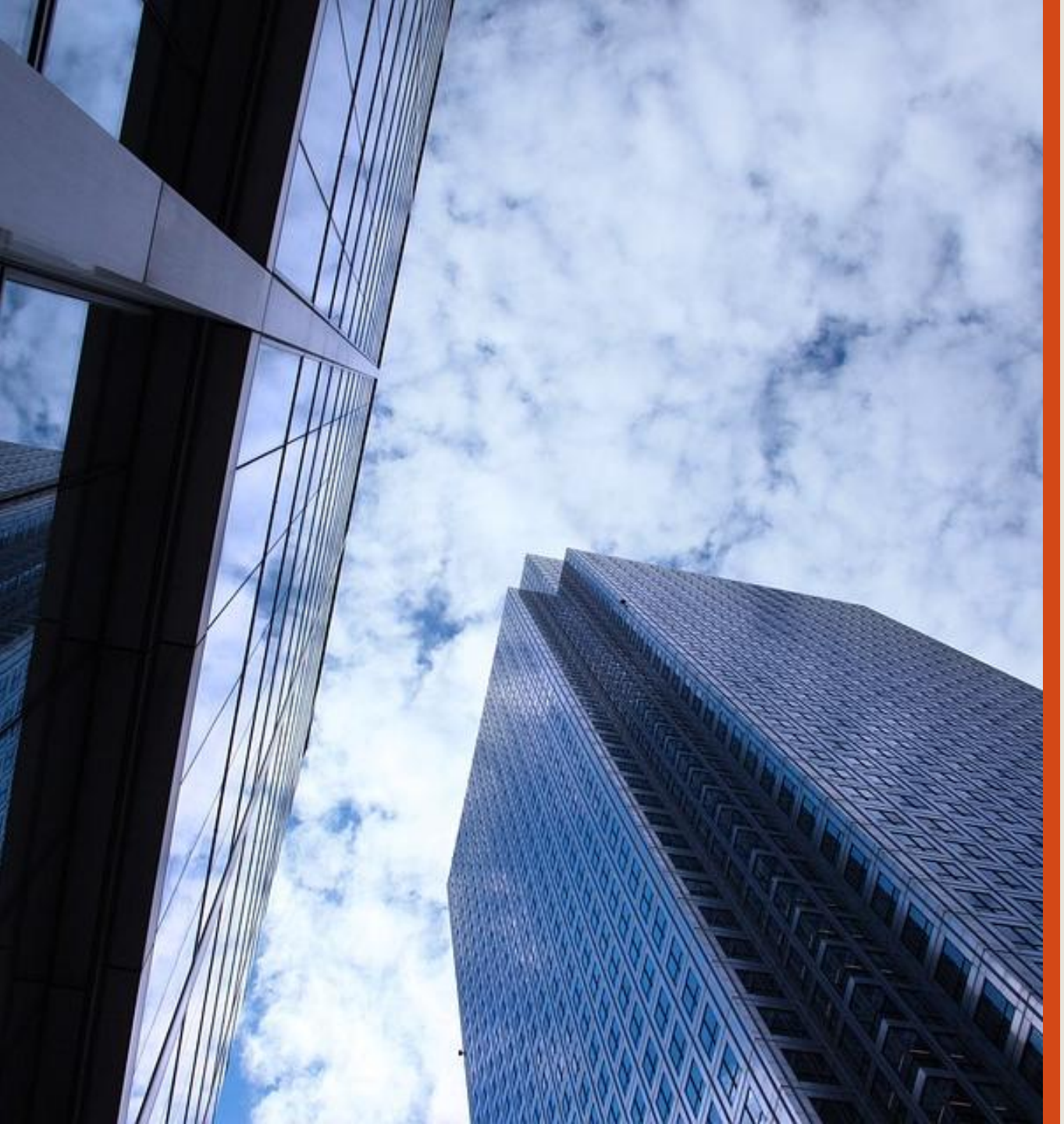


RELATIONAL ALGEBRA

Jayathma Chathurangani

ejc@ucsc.cmb.ac.lk

OUTLINE

- Selection and Projection
- Joins and Division
- Set Operations and Renaming
- Relational Calculus

RELATIONAL QUERY LANGUAGES

- Two mathematical Query Languages form the basis for "real" query languages (e.g. SQL), and for implementation:
- **Relational Algebra**: More operational, very useful for representing execution plans.
  - Basis for SEQUEL
- **Relational Calculus**: Let's users describe WHAT they want, rather than HOW to compute it. (declarative.)
  - Basis for QUEL

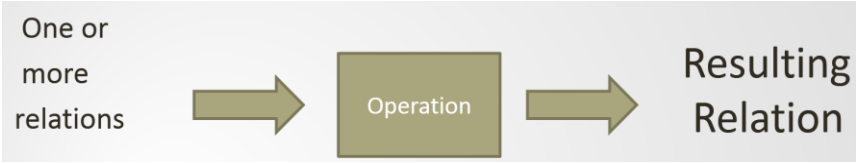
1

RELATIONAL ALGEBRA: INTRODUCTION

1.1 INTRODUCTION

- Relational algebra primarily serves as the theoretical basis for relational databases and SQL.
- Its main goal is to define operators that convert one or more input relations into an output relation.
- Since these operators take relations as input and generate relations as output, they can be combined to form complex queries, which transform multiple input relations (the data stored in the database) into a single output relation (the query results).
- Being a purely mathematical framework, relational algebra does not use English keywords, and its operators are represented through symbols.
- Relational Algebra treats relations as sets

1.2 WHAT IS RELATIONAL ALGEBRA?



- **A collection of operations that users can perform on relations to obtain a desired result**
- Relational algebra comprises a set of operations or rules commonly used to manipulate and query data in a relational database.
- It can be implemented through SQL, allowing users to interact with database tables and query data more effectively and efficiently.
- Relational algebra includes a variety of operations, such as filtering and combining data, which help in organizing and managing data.
- This "algebra" forms the basis for most database queries and enables the extraction of specific information from databases using the SQL query language.

- 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.
- It is also a conceptual language. By this we mean that the queries made in it are not run on the computer so it is not used as a business language. However, this knowledge allows us to understand the optimization and query execution of RDBMS.

1.3 LOGICAL OPERATORS IN RELATIONAL ALGEBRA

- Relational Algebra includes logical operators to filter and manipulate data conditions, similar to how SQL uses AND, OR, and NOT. Below are the main logical operators:

Operator	Symbol	Meaning	Use in Relational Algebra	SQL Equivalent
AND	∧	Both conditions must be true	σ_{condition1 ∧ condition2} (Relation)	WHERE condition1 AND condition2
OR	∨	At least one condition must be true	σ_{condition1 ∨ condition2} (Relation)	WHERE condition1 OR condition2
NOT	¬	Negates a condition	σ_{¬condition} (Relation)	WHERE NOT condition

2

RELATIONAL ALGEBRA: UNARY OPERATIONS: SELECTION AND PROJECTION

2.1 SELECTION(σ)

- Used to filter out rows from a given table based on certain given condition.
- It basically allows you to retrieve only those rows that match the condition as per condition passed during SQL Query.
- It is used to select required tuples of the relations.
- ***Note**: The selection operator only selects the required tuples but does not display them. For display, the data projection operator is used.*
- ***Notation*** – $\sigma_p(r)$

Where σ stands for selection predicate and r stands for relation. p is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like – =, ≠, ≥, < , >, ≤.

- ***Example***

$$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$$

Output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

2.2 PROJECTION(Π)

- While Selection operation works on rows , similarly projection operation of relational algebra works on columns.
- It basically allows you to pick specific columns from a given relational table based on the given condition and ignoring all the other remaining columns.
- It is used to project required column data from a relation.
- ***Note**: By Default, projection removes duplicate data*
- ***Notation*** – $\prod_{A1, A2, An}(r)$

Where A_1, A_2, A_n are attribute names of relation r.

- ***Example***

$$\prod_{subject,\ author}(Books)$$

Output – Selects and projects columns named as subject and author from the relation Books.

3

RELATIONAL ALGEBRA: JOINS AND DIVISIONS

3.1 WHAT?

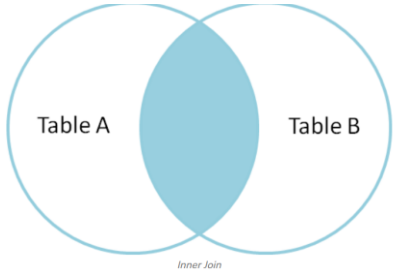
- Join is an operation in DBMS(Database Management System) that combines the row of two or more tables based on related columns between them.
- The main purpose of Join is to retrieve the data from multiple tables in other words Join is used to perform multi-table queries.
- It is denoted by ⋈.
- ***Notation*** – *R3 <- ⋈ (R1) <join_condition> (R2)*

where R1 and R2 are two relations to be joined and R3 is a relation that will hold the result of the join operation.

Example

Temp <- ⋈ (student) S.roll=E.roll(Exam)

3.2.1 INNER JOIN: CONDITIONAL JOIN



- Inner Join is a join operation in DBMS that combines two or more tables based on related columns and returns only rows that have matching values among tables.
- ***Inner join types: Conditional, Equi and Natural***
- **Theta Join = Conditional Join** (both names are used interchangeably) is a type of inner join in which tables are combined based on the specified condition.

$R \bowtie_{\theta} S$ Where:

- R and S are two relations (tables).
- θ (theta condition) represents a comparison condition, which can include <, >, <=, >=, ≠ operators in addition to the = operator.

Table A

R	S
10	5
7	20

Table B

T	U
10	12
17	6

Example

- Joins the table A, B and projects attributes R, S, T, U were condition S

SELECT R, S, T, U FROM TableA JOIN TableB ON S < T;

$A \bowtie_{S<T} B$

Output

R	S	T	U
10	5	10	12

3.2.2 INNER JOIN: EQUI JOIN

- Equi Join is a type of Inner join in which we use equivalence('=') condition in the join condition
- It joins tables using a specific equality condition (=) in the ON or WHERE clause.
- ***It does not automatically remove duplicate columns.***

Example

- The data value "a" is available in both tables Hence we write that "a" is the table in the given output.
- SELECT * FROM A JOIN B ON A.ColumnB = B. ColumnB;

Table A

Column A	Column B
a	a
a	b

Table B

Column A	Column B
a	a
a	c

$$A \bowtie_{A.Column\ B = B.Column\ B} (B)$$

Output

Column A	Column B
a	a

3.2.3 INNER JOIN: NATURAL JOIN

- Natural join is a type of inner join in which we do not need any comparison operators.
- In natural join, columns should have the same name and domain. There should be at least one common attribute between the two tables.
- It automatically joins tables based on columns with the same name and data type in both tables.
- No need to specify the joining condition explicitly.
- ***It eliminates duplicate columns from the result.***

Example

- SELECT * FROM TableA NATURAL JOIN TableB;

Table A

Number	Square
2	4
3	9

Table B

Number	Cube
2	8
3	27

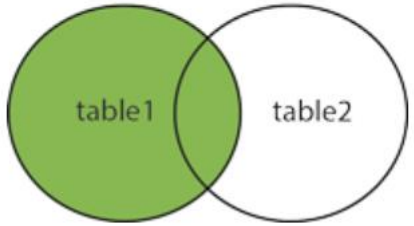
Output

Number	Square	Cube
2	4	8
3	9	27

A ⋈ B

Feature	Natural Join	Equi Join
Join Condition	Implicit (based on common column names)	Explicit (specified in `ON` clause)
Duplicate Columns	Eliminated	Retained
Flexibility	Less control over join condition	More control (can specify conditions)

3.3.1 OUTER JOIN: LEFT OUTER JOIN



- Outer join is a type of join that retrieves matching as well as non-matching records from related tables.
- Outer join types: Left outer join, Right outer join, Full outer join
- Left Outer Join also called left join. This type of outer join retrieves all records from the left table and retrieves matching records from the right table.

Example

- SELECT * FROM TableA LEFT OUTER JOIN TableB ON TableA.Number = TableB.Number;

Table A

Number	Square
2	4
3	9
4	16

Table B

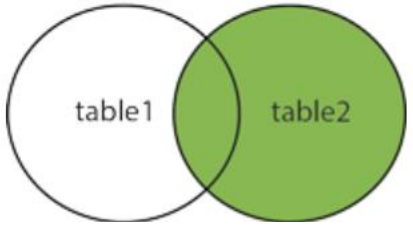
Number	Cube
2	8
3	27
5	125

$A \bowtie B$

Output

Number	Square	Cube
2	4	8
3	9	27
4	16	NULL

3.3.2 OUTER JOIN: RIGHT OUTER JOIN



- Right outer join is also called a right join.
- This type of outer join retrieves all records from the right table and retrieves matching records from the left table. And for the record which doesn't lies in Left table will be marked as NULL in result Set.

Example

- SELECT * FROM TableA RIGHT OUTER JOIN TableB ON TableA.Number= TableB.Number;

Table A

Number	Square
2	4
3	9
4	16

Table B

Number	Cube
2	8
3	27
5	125

$A \rtimes B$

Output:

Number	Square	Cube
2	4	8
3	9	27
5	NULL	125

3.4 SEMI JOIN

- Semi Join returns only the tuples from the first relation (R) that have a matching tuple in the second relation (S), but it does not include attributes from S in the result.

  Notation: $R \triangleright S$

Example: Suppose we have:

- Employee (EmpID, Name, DeptID)
- Department (DeptID, DeptName)

Query: Find employees who work in a department.

Semi Join:

$$Employee \triangleright_{Employee.DeptID=Department.DeptID} Department$$

- Key Difference: Semi Join only filters R based on matching records in S, but does not return attributes from S.

Feature	Semi Join (▷)	Inner Join (⋈)	Outer Join (⟗, ⟕, ⟖)
Returns only tuples from first relation (R)?	✅ Yes	❌ No (returns both)	❌ No (returns both)
Includes attributes from second relation (S)?	❌ No	✅ Yes	✅ Yes
Filters based on matching records?	✅ Yes	✅ Yes	✅ Yes + includes NULLs
Includes unmatched tuples?	❌ No	❌ No	✅ Yes (NULLs for missing values)

3.5 DIVISION

- Given two relations:

  R(A, B) → A relation containing two attributes A and B.

  S(B) → A relation containing only attribute B.

- The result:

R ÷ S gives all A values from R that are associated with all B values from S.

- Example: Find students who have taken **all mandatory courses**.

Relations:

•StudentCourse(Student, Course)

- (Alice, DB)
- (Alice, OS)
- (Bob, DB)
- (Charlie, DB)
- (Charlie, OS)

•MandatoryCourses(Course)

- (DB)
- (OS)

Query:

•StudentCourse ÷ MandatoryCourses

•Result:

- ✅ Alice
- ✅ Charlie
- ❌ Bob (because he didn't take OS)

3.5 DIVISION (CONTD)

- Since SQL doesn't directly support division, we use a combination of COUNT and GROUP BY.

```
SELECT Student
FROM StudentCourse
WHERE Course IN (SELECT Course FROM MandatoryCourses)
GROUP BY Student
HAVING COUNT(DISTINCT Course) = (SELECTCOUNT(*) FROM MandatoryCourses);
```

- Explanation: Selects students who have taken all mandatory courses.
- Uses HAVING COUNT() to ensure all courses are covered.

- Limitations of Division

•Only Works on Two Attributes (A, B)

- The second relation must have a single attribute.

•Strict Matching Requirement

- Partial matches are not included (e.g., if a student misses just one course, they are excluded).

•Performance Issues

- Can be slow on large datasets unless indexed properly.

3

RELATIONAL ALGEBRA: SET OPERATIONS AND RENAMING

4.1 WHAT?

- Used to combine results from multiple relations.
- Work only on union-compatible relations (same number & type of attributes).
- Main Set Operations:
  - Union (∪)
  - Intersection (∩)
  - Difference (–)

4.2 UNION

- Combines tuples from both relations without duplicates.
- Works only on union-compatible relations.

$R \cup S$

Example:

- **Student1 (ID, Name)**
  - (1, Alice)
  - (2, Bob)
- **Student2 (ID, Name)**
  - (3, Charlie)
  - (2, Bob)
- **Result (Student1 ∪ Student2):**
  - (1, Alice)
  - (2, Bob)
  - (3, Charlie)

4.3 INTERSECTION

- Returns only common tuples in both relations.

$R \cap S$

Example:

- **Student1 (ID, Name)**
  - (1, Alice)
  - (2, Bob)
- **Student2 (ID, Name)**
  - (3, Charlie)
  - (2, Bob)
- **Result (Student1 ∩ Student2):**
  - (2, Bob)

4.3 SET DIFFERENCE

- Returns tuples in first relation but not in second.

R-S

Example

- **Student1 (ID, Name)**
  - (1, Alice)
  - (2, Bob)

- **Student2 (ID, Name)**
  - (3, Charlie)
  - (2, Bob)

- **Result (Student1 – Student2):**
  - (1, Alice)

EXTRA: FYI

Feature	Set Difference (−)	Left Outer Join (⋈)	Right Outer Join (⋈)
Purpose	Finds elements in R but not in S	Keeps all records from R, adding matches from S	Keeps all records from S, adding matches from R
Output Includes	Only from R (excluding common elements)	All from R, plus matching S values	All from S, plus matching R values
NULLs Used?	❌ No	✅ Yes (for missing matches in S)	✅ Yes (for missing matches in R)
Union-Compatible?	✅ Yes	❌ No	❌ No

When to use?

✅ **Use Set Difference (−)**

- When you want to **remove common elements** and keep only unique ones.
- Example: Find students **only in Course_A but not in Course_B.**
- (removes common elements.)

✅ **Use Left Outer Join (⋈)**

- When you want to **keep all elements from R** and see if they exist in S.
- Example: List all customers and **show orders if available.**

✅ **Use Right Outer Join (⋈)**

- When you want to **keep all elements from S** and see if they exist in R.
- Example: Show all **departments**, including ones without employees.

4.4 RENAMING

- Used to **rename** relation or attributes.
- Helpful in **self-joins & disambiguating** column names.

$$\rho_{new\_name}(R)$$

- **Example**
- Original Relation: Employee(ID, Name, Salary)
- Renamed:

$$\rho_{Emp(ID,EName,Sal)}(Employee)$$

4.5 FOR YOUR KNOWLEDGE

Checking for NULL Values in Relational Algebra

- Relational Algebra **does not have direct support for NULL values** as it follows the **relational model**, where every tuple has a definite value. However, databases that support **NULL values (like SQL)** handle this differently.
- To simulate NULL checks in **Relational Algebra**, we typically use **Outer Joins (⟕, ⟖)** and **Set Difference (−)**.

What is " * " in Relational Algebra?

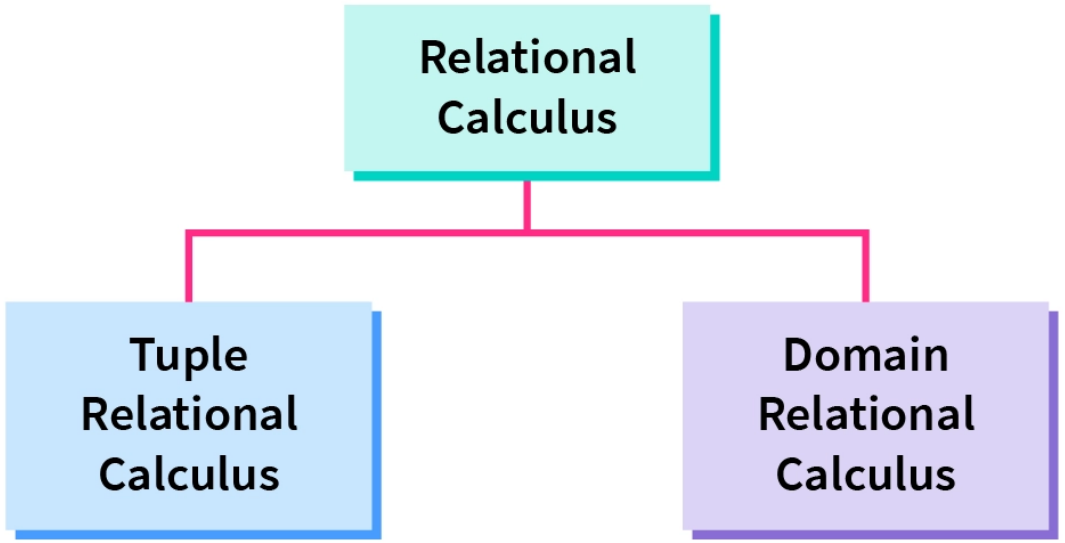
- In **Relational Algebra**, the " * " symbol represents the **Cartesian Product (×)** operation.

4

RELATIONAL CALCULUS

4.1 WHAT?

- **What is Relational Calculus?**
  - A **non-procedural** query language (unlike Relational Algebra). i.e. Declarative.
  - Focuses on **what** data is required, not how to retrieve it.
  - **Based on logic & predicates** (i.e., conditions).



4.2 TUPLE RELATIONAL CALCULUS (TRC)

- Uses variables that represent tuples.
- Queries are expressed using logical predicates.
- **Notation:** {t|P(t)}
  - t represents a tuple.
  - P(t) is a condition that must be satisfied.
- **Example:**
  - Problem: Find employees who earn more than $50,000.
  - Relation: Employee (EmpID, Name, Salary, DeptID)
  - TRC Query: {t|t∈Employee ∧ t[Salary]>50000}
  - Explanation: t represents tuples from Employee. The query filters employees with Salary > 50000.

4.2 TUPLE RELATIONAL CALCULUS (TRC) - CONTD

- **TRC uses logical operators:**
  - ∧ **(AND)** – Both conditions must be true.
  - ∨ **(OR)** – At least one condition must be true.
  - ¬ **(NOT)** – Negates a condition.
  - ∃ **(Exists)** – At least one tuple satisfies the condition.
  - ∀ **(For all)** – All tuples satisfy the condition.
- **Example**

Problem: Find employees who work in the Sales department.

Relation: Employee (EmpID, Name, Salary, DeptID), Department (DeptID, DeptName)

TRC Query: **{t|t∈Employee ∧ ∃ d ∈ Department (d[DeptID]=t[DeptID] ∧ d[DeptName]='Sales')}**

Explanation: Checks if a department exists where DeptName = 'Sales'. Returns employees working in the Sales department.

4.3 DOMAIN RELATIONAL CALCULUS (DRC)

- Uses variables that represent values (domains) instead of tuples.
- Queries are written using quantifiers and logical predicates.
- Notation: {<d1,d2,...,dn>|P(d1,d2,...,dn)}
  - d1,d2,...d1,d2,... are domain variables.
  - P is a predicate condition.
- **Example:**

Problem: Find employees who earn more than $50,000.

Relation: Employee(EmpID, Name, Salary, DeptID)

DRC Query: **{<E,N,S,D>|Employee(E,N,S,D) ∧ S>50000}**

Explanation: E,N,S,DE,N,S,D are domain variables representing Employee attributes. Condition S > 50000 filters employees by salary.

4.4 TRC VS DRC

Feature	TRC	DRC
Representation	Uses **tuple variables**	Uses **domain variables**
Focus	Works with entire tuples	Works with Specific Values
Example Query	$\{ t \mid t \in Employee \land t.Salary > 50000 \}$ Retrieves all tuples **t** from Employee where Salary > 50000.	$\{ <E, N, S, D> \mid Employee(E, N, S, D) \land S > 50000 \}$ Retrieves specific **domain values** (E = EmpID, N = Name, S = Salary, D = DeptID) where Salary > 50000.
More Readable	Easier	Not easier
Usecases	Used in **relational databases** (like SQL)	Theoretical & **less commonly implemented**

4

RELATIONAL ALGEBRA VS RELATIONAL CALCULUS

- **Relational Algebra and Relational Calculus are not the same.** They are two different formal query languages in database theory, but they serve similar purposes in defining queries over relational databases.

Feature	Relational Algebra	Relational Calculus
Variants	Only **one type**: **Relational Algebra**.	**Two types:** Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC).
Type of Language	**Procedural** – Specifies **how** to retrieve data.	**Declarative** – Specifies **what** data is required.
Operations	Uses **mathematical operations** like Selection (σ), Projection (π), Join (⋈), Set Operations (∪, ∩, −).	Uses **logical formulas** with tuple or domain variables.
Approach	Uses a **step-by-step process** to obtain results.	Uses **logic & predicates** to describe the query result.
Example Query (Find employees earning > 50K)	π Name (σ Salary > 50000 (Employee))	{ t | t ∈ Employee ∧ t.Salary > 50000 }
Ease of Understanding	**More intuitive** (step-by-step process).	**More abstract** (logic-based).
Implementation	Used as the foundation for **query execution engines** (SQL processing).	Used in **query optimization and theoretical database models**.

WRAP UP
THANK YOU!