JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

DEPARTMENT OF BCA

DATA ANALYTICS

SUBJECT: RDBMS **SEMESTER: 3rd BCA**

MODULE-2

RELATIONAL ALGEBRA

TRIGGERS

What are Triggers in PL/SQL?

- Triggers are stored programs that are fired automatically when some events occur. The code to be fired can be defined as per the requirement.

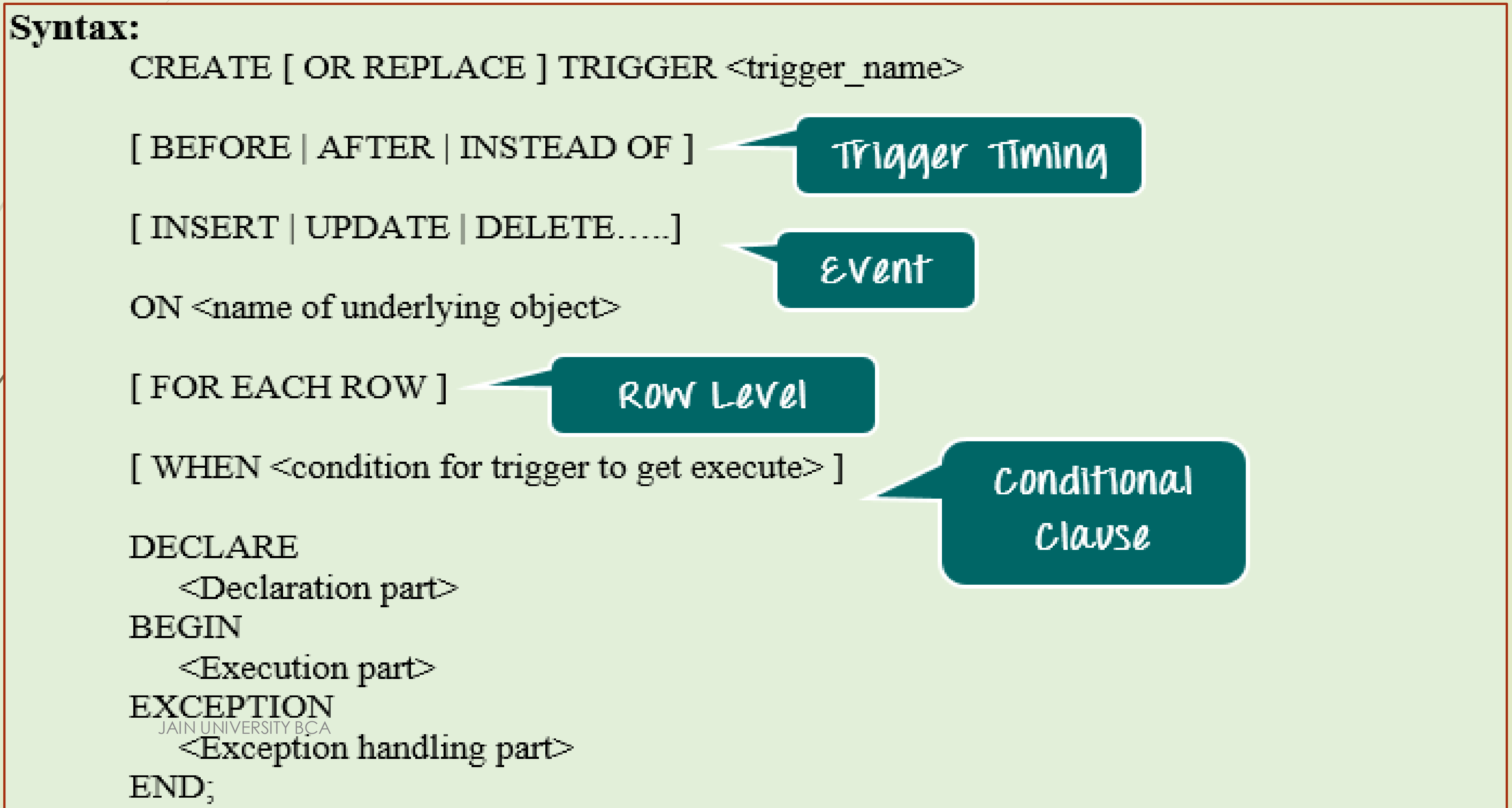
- Oracle has also provided the facility to mention the event upon which the trigger needs to be fire and the timing of the execution.

Types of Triggers in Oracle

- Triggers can be classified based on the following parameters.
- Classification based on the **timing**
  - BEFORE Trigger: It fires before the specified event has occurred.
  - AFTER Trigger: It fires after the specified event has occurred.
- Classification based on the **level**
  - STATEMENT level Trigger: It fires one time for the specified event statement.
  - ROW level Trigger: It fires for each record that got affected in the specified event. (only for DML)
- Classification based on the **Event**
  - DML Trigger: It fires when the DML event is specified (INSERT/UPDATE/DELETE)
  - DDL Trigger: It fires when the DDL event is specified (CREATE/ALTER)
  - DATABASE Trigger: It fires when the database event is specified (LOGON/LOGOFF/STARTUP/SHUTDOWN)
- So each trigger is the combination of above parameters.

How to Create Trigger

- **Below is the syntax for creating a trigger.**



Syntax:

```
CREATE [ OR REPLACE ] TRIGGER <trigger_name>

[ BEFORE | AFTER | INSTEAD OF ]

[ INSERT | UPDATE | DELETE.....]

ON <name of underlying object>

[ FOR EACH ROW ]

[ WHEN <condition for trigger to get execute> ]

DECLARE
    <Declaration part>
BEGIN
    <Execution part>
EXCEPTION
    <Exception handling part>
END;
```

Trigger Timing
Event
Row Level
Conditional Clause

EXAMPLE ON TRIGGER

```
Create or replace trigger trig_error
before insert or update or delete
on emp
begin
    if to_char(sysdate,'DY')  IN ('MON','TUE') then
        raise_application_error(-20111,'No changes can be    made on
        tuesday');
        dbms_output.put_line(‘Cant  insert/delete on Tuesday’);
    end if;
end;
```

EXAMPLE-2: Create a trigger to insert a new record to log table after inserting a record to dept table

```
Create or replace trigger depttrig
AFTER INSERT ON dept
FOR EACH ROW
BEGIN
INSERT INTO dept_log
VALUES(:new.deptno,:new.dname,:new.loc);
END;
```

EXAMPLE-3: Create a trigger to insert aold record to log table after updating or deleting a record to dept table

```
Create or replace trigger dept_trig
BEFORE update or delete on dept
FOR EACH ROW
BEGIN
INSERT INTO dept_log
VALUES(:old.deptno,:old.dname,:old.loc);
END;
```

EXAMPLE-3: Create a trigger to count number of records from emp table after deleting a record to emp table

```
CREATE OR REPLACE TRIGGER emp_count
AFTER DELETE ON EMP
DECLARE
        n INTEGER;
BEGIN
        SELECT COUNT(*) INTO n FROM emp;
        DBMS_OUTPUT.PUT_LINE(' There are now ' || n || ' employees.');
END;
```

Example 4:Create a trigger Convert ename to upper

```
CREATE OR REPLACE TRIGGER before_emp
BEFORE UPDATE OR INSERT ON emp
FOR EACH ROW
begin
    /* convert character values to upper case */
    :new.ename := upper( :new.ename );
    :new.job := upper( :new.job);
end;
```

Dictionary for triggers is user_triggers

SQL>Select trigger_name,trigger_body,table_name,description

from user_triggers

Enabling/Diasabling Triggers

To enable a disabled trigger, use the ALTER TRIGGER statement with the ENABLE clause.

Example:

SQL>ALTER TRIGGER trig_error ENABLE;

To enable all triggers defined for a specific table, use the ALTER TABLE statement with the ENABLE clause and the ALL TRIGGERS option.

SQL>ALTER TABLE emp ENABLE ALL TRIGGERS;

Disabling Triggers

- You might temporarily disable a trigger if:
- You must perform a large data load, and you want it to proceed quickly without firing triggers.
- ALTER TRIGGER trig_error DISABLE;
- To disable all triggers defined for a specific table, use the ALTER TABLE statement with the DISABLE clause and the ALL TRIGGERS option. For example, to disable all triggers defined for the Inventory table, enter the following statement:
- ALTER TABLE emp DISABLE ALL TRIGGERS;

- **Database security** refers to the collective measures used to protect and **secure** a **database** or **database** management software from illegitimate use and malicious threats and attacks.
- It is a broad term that includes a multitude of processes, tools and methodologies that ensure **security** within **database** environment.

Advanced SQL features – (CASE, DECODE, Rollup, Cube) done in SQL

- **Embedded SQL**– Embedded SQL is a method of combining the computing power of a programming language and the database manipulation capabilities of SQL. Embedded SQL statements are SQL statements written inline with the program source code, of the host language.
- **Need of Embed SQL:** Because the host language cannot parse **SQL**, the inserted **SQL** is parsed by an **embedded SQL** preprocessor. **Embedded SQL** is a robust and convenient method of combining the computing power of a programming language with **SQL's** specialized data management and manipulation capabilities

Dynamic SQL:

- Dynamic SQL is a programming methodology for generating and running statements at run-time. It is mainly used to write the general-purpose and flexible programs where the SQL statements will be created and executed at run-time based on the requirement.
- Embedded SQL are SQL statements in an application that do not change at run time and, therefore, can be hard-coded into the application. Dynamic SQL is SQL statements that are constructed at run time; for example, the application may allow users to enter their own queries.

Introduction to Distributed Databases:

- In a distributed database, there are a number of databases that may be geographically distributed all over the world. A distributed DBMS manages the distributed database in a   manner so that it appears as one single database to users.
- **The two types of distributed systems are as follows:**
- Homogeneous distributed databases system: Homogeneous distributed database system is a network of two or more databases (With same type of DBMS software) which can be stored on one or more machines. ...
- Heterogeneous distributed database system.
- Example: Oracle distributed database systems employ a distributed processing architecture to function. For example, an Oracle server acts as a client when it requests data that another Oracle server manages.

Client/Server Databases

- A **client/server application** is a piece of software that runs on a **client** computer and makes requests to a remote **server**. Many such **applications** are written in high-level visual programming languages where UI, forms, and most business logic reside in the **client application**.

Relational Algebra

- Relational Algebra
- Fundamental operations
- Additional Operations
- Domain Relational Calculus
- Tuple Relational Calculus

Relational Query Languages

- Languages for describing queries on a relational database
- *Structured Query Language* (SQL)
  - Predominant application-level query language
  - Declarative
- *Relational Algebra*
  - Intermediate language used within DBMS
  - Procedural

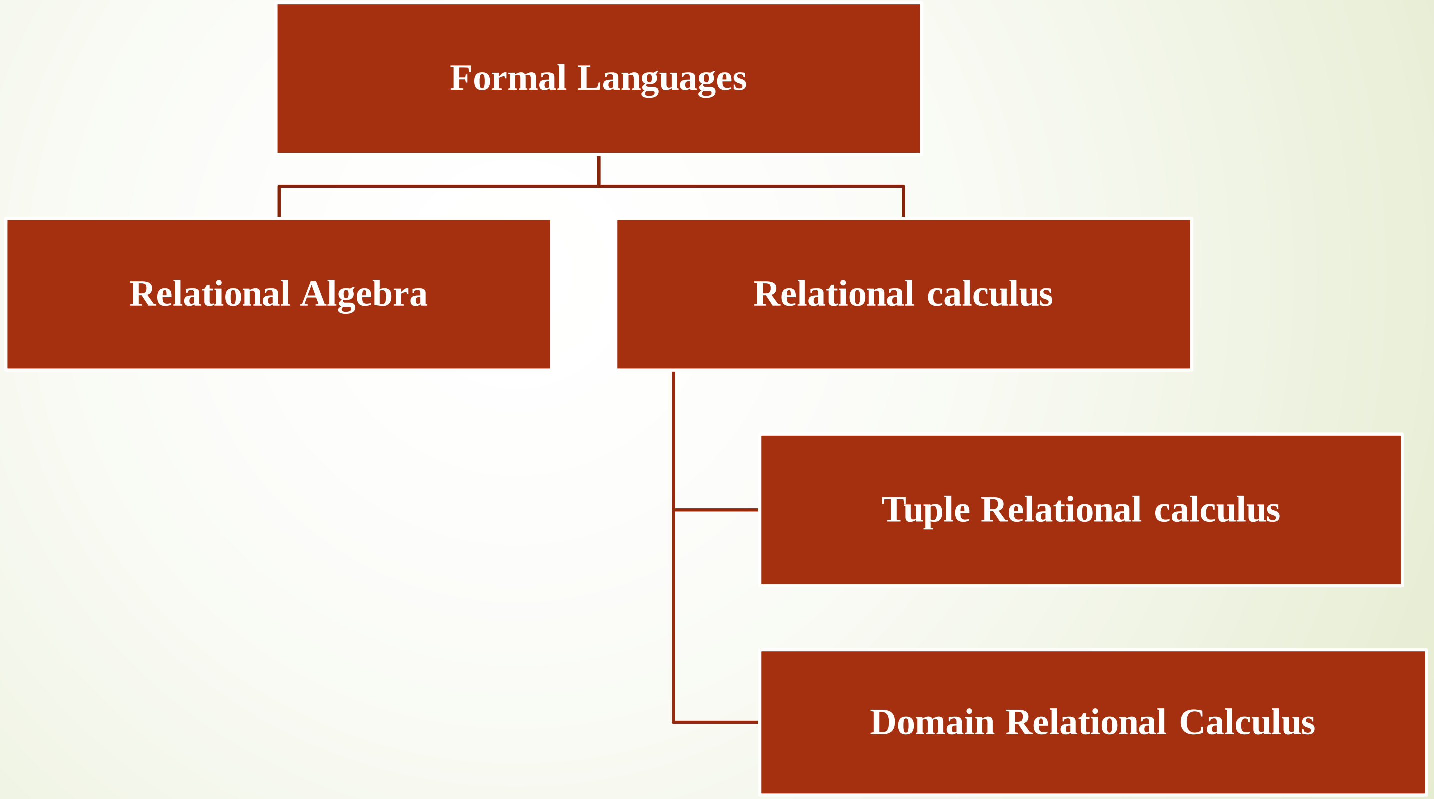
What is an Algebra?

- A language based on operators and a domain of values.
- Operators map values taken from the domain into other domain values
- Hence, an expression involving operators and arguments produces a value in the domain
- When the domain is a set of all relations (and the operators are as described later), we get the *relational algebra*
- We refer to the expression as a *query* and the value produced as the *query result*

Relational Algebra:

- Relational algebra is a procedural query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data. When I say that relational algebra is a procedural query language, it means that it tells what data to be retrieved and how to be retrieved.
- On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn’t tell how to retrieve it.
- **Types of operations in relational algebra:**
- We have divided these operations in two categories:
  1. Basic Operations
  2. Derived Operations

Formal Languages



Relational Algebra

- *Domain*: set of relations
- *Basic operators*: select, project, union, set difference, Cartesian product
- *Derived operators*: set intersection, division, join
- *Procedural*: Relational expression specifies query by describing an algorithm (the sequence in which operators are applied) for determining the result of an expression

Basic/Fundamental Operations:

- 1. Select (σ)
- 2. Project ($\prod$)
- 3. Union ($\cup$)
- 4. Set Difference (-)
- 5. Cartesian product (X)
- 6. Rename (ρ)

Derived Operations:

- 1. Natural Join ($\bowtie$)
- 2. Left, Right, Full outer join (⟕, ⟖, ⟗)
- 3. Intersection ($\cap$)
- 4. Division ($\div$)

Select Operator

- Produce table containing subset of rows of argument table satisfying condition

$\sigma_{condition}\ relation$

- Example:

Person

Id	Name	Address	Hobby
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\sigma_{Hobby=\text{‘stamps’}}(\text{Person})$

Id	Name	Address	Hobby
1123	John	123 Main	stamps
9876	Bart	5 Pine St	stamps

Selection Condition

- Operators: <, ≤, ≥, >, =, ≠
- Simple selection condition:
  - *<attribute> operator <constant>*
  - *<attribute> operator <attribute>*
- *<condition>* AND *<condition>*
- *<condition>* OR *<condition>*
- NOT *<condition>*

Selection Condition - Examples

- $\sigma_{\text{Id>3000 Or Hobby='hiking'}}$ (Person)
- $\sigma_{\text{Id>3000 AND Id <3999}}$ (Person)
- $\sigma_{\text{NOT(Hobby='hiking')}}$ (Person)
- $\sigma_{\text{Hobby}\neq\text{'hiking'}}$ (Person)

Project Operator

- Produces table containing subset of columns of argument table

$\Pi_{\textit{attribute list}}(\textit{relation})$

- Example:

Person

Id	Name	Address	Hobby
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\Pi_{\textit{Name,Hobby}}$**(Person)**

Name	Hobby
John	stamps
John	coins
Mary	hiking
Bart	stamps

Project Operator Example:

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$\Pi_{Name,Address}$(Person)

Name	*Address*
John	123 Main
Mary	7 Lake Dr
Bart	5 Pine St

Result is a table (no duplicates)

Expressions

$$\Pi_{Id,\ Name}\ (\sigma_{Hobby='stamps'\ \mathrm{OR}\ Hobby='coins'}\ (\mathrm{Person})\ )$$

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

Result

Id	*Name*
1123	John
9876	Bart

Set Operators

- Relation is a set of tuples => set operations should apply
- Result of combining two relations with a set operator is a relation => all its elements must be tuples having same structure
- Hence, scope of set operations limited to *union compatible relations*

Union Compatible Relations

- Two relations are *union compatible* if
  - Both have same number of columns
  - Names of attributes are the same in both
  - Attributes with the same name in both relations have the same domain
- Union compatible relations can be combined using *union*, *intersection*, and *set difference*

Union Operation (∪)

- It performs binary union between two given relations and is defined as −
- $r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$ **Notation** − r U s
- Where **r** and **s** are either database relations or relation result set .
- For a union operation to be valid, the following conditions must hold −
- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.
- $\prod_{\text{author}} (\text{Books}) \cup \prod_{\text{author}} (\text{Articles})$

Example

Tables:

Person (SSN, Name, Address, Hobby)

Professor (Id, Name, Office, Phone)

are not union compatible. However

Π_{Name} (Person) and Π_{Name} (Professor)

are union compatible and

Π_{Name} (Person) - Π_{Name} (Professor)

makes sense.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation – **r – s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author} (Books) - \prod_{author} (Articles)$

Output – Provides the name of authors who have written books but not articles.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − ρ_{x} (E)

Where the result of expression **E** is saved with name of **x**.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where r and s are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

σ author = 'Martin Seligmen'(Books X Articles)

Output − Yields a relation, which shows all the books and articles written by Martin Seligmen.

Derived Operation: Join

The expression :

$$\sigma_{\textit{join-condition}'} (R \times S)$$

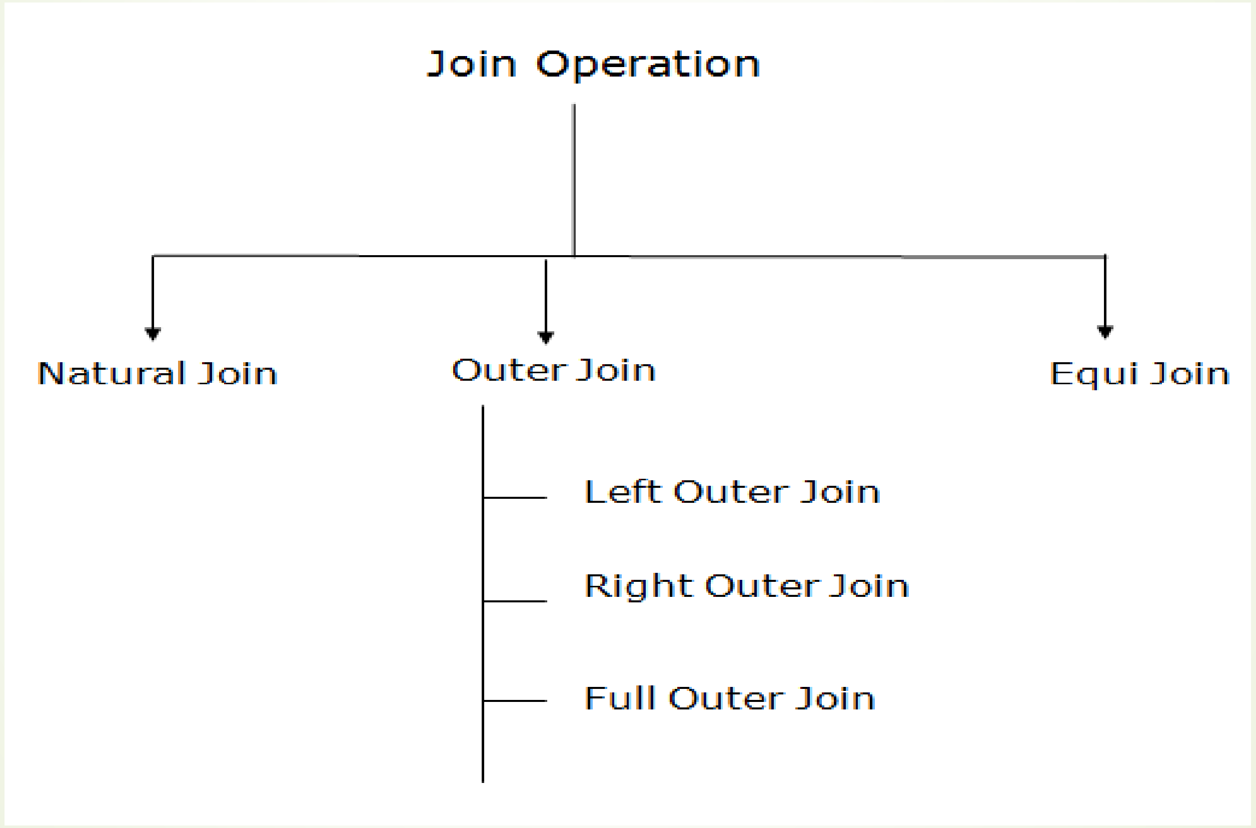
where *join-condition´* is a *conjunction* of terms:

$$A_i \;\; oper \; B_i$$

in which A_i is an attribute of R, B_i is an attribute of S, and *oper* is one of =, <, >, ≥ ≠, ≤, is referred to as the (theta) join of R and S and denoted:

$$R \bowtie_{\textit{join-condition}} S$$

Types of join operations



Join Types

`[INNER] JOIN` $A \bowtie B$	Row must exist in <u>both</u> tables
`LEFT [OUTER] JOIN` $A ⟕ B$	Row must *at least* exist in the table to the left (padded with **`NULL`**)
`RIGHT [OUTER] JOIN` $A ⟖ B$	Row must exist *at least* in the table to the right (padded with **`NULL`**)
`FULL OUTER JOIN` $A ⟗ B$	Row exists in <u>either</u> table (padded with **`NULL`**)

Join Type Example (1)

ALPHA

a	b
x	1
y	2
z	3

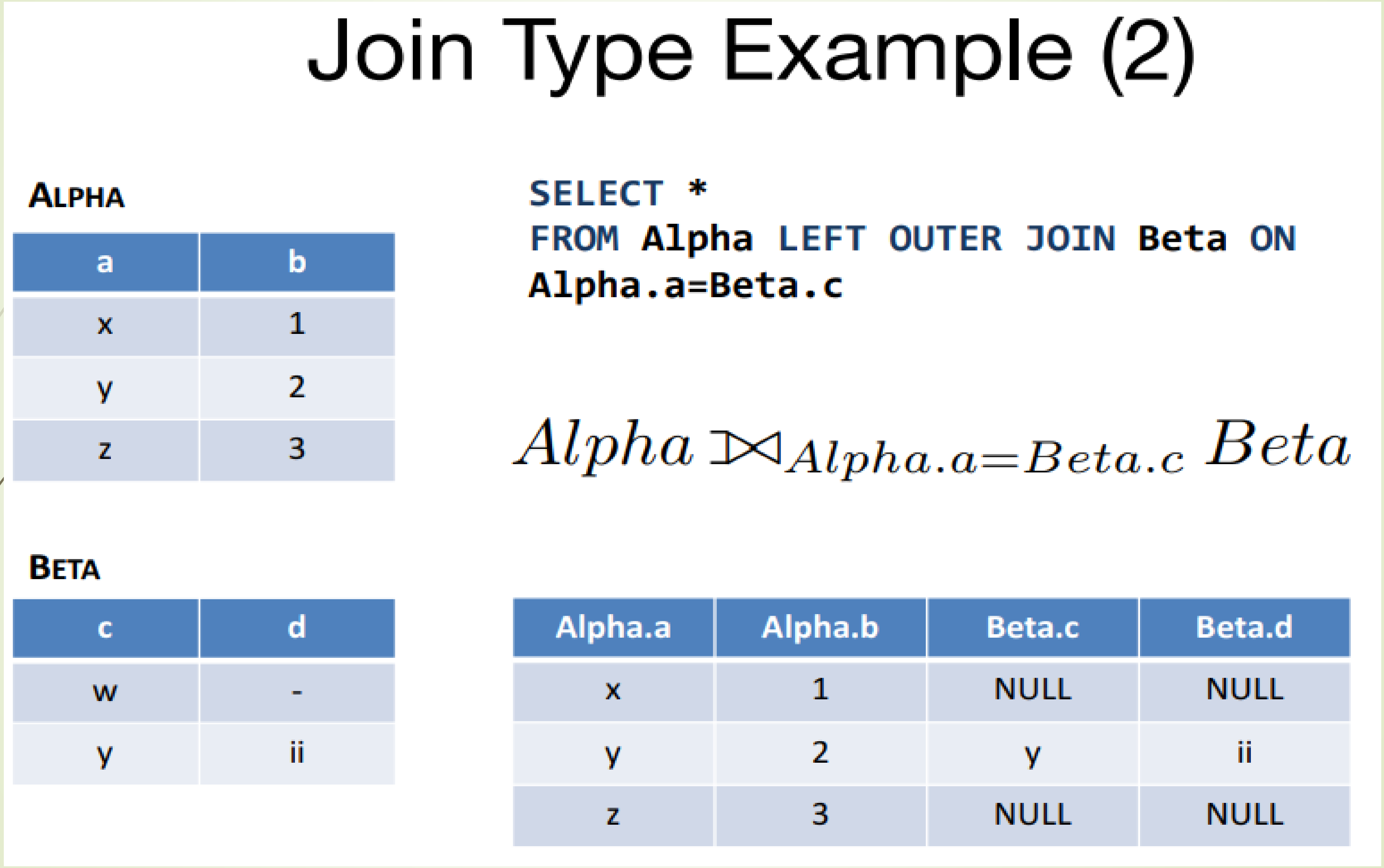
BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha INNER JOIN Beta ON
Alpha.a=Beta.c
```

$$Alpha \bowtie_{Alpha.a=Beta.c} Beta$$

Alpha.a	Alpha.b	Beta.c	Beta.d
y	2	y	ii



Join Type Example (2)

ALPHA

a	b
x	1
y	2
z	3

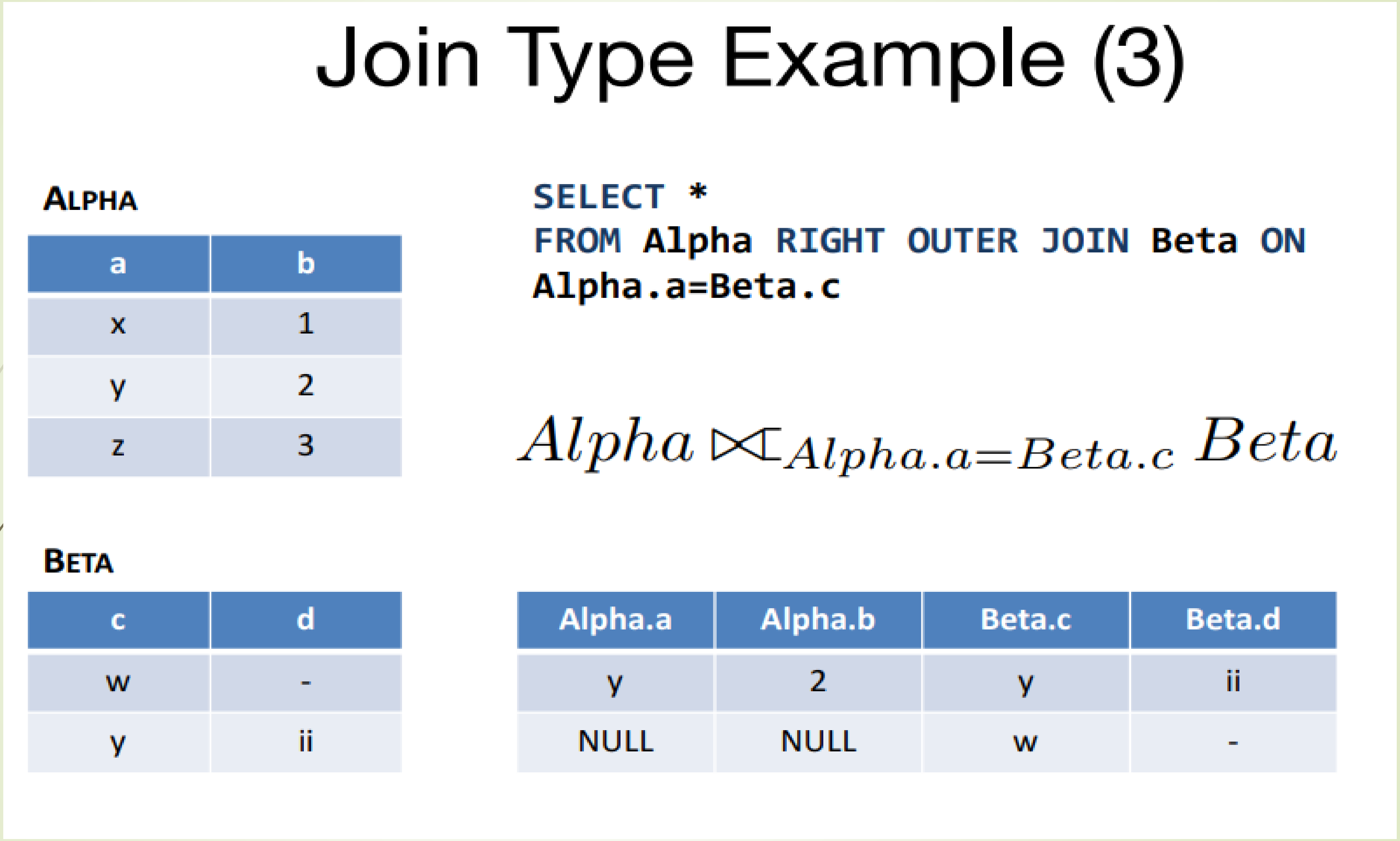
BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha LEFT OUTER JOIN Beta ON
Alpha.a=Beta.c
```

$$Alpha \mathbin{⟕}_{Alpha.a=Beta.c} Beta$$

Alpha.a	Alpha.b	Beta.c	Beta.d
x	1	NULL	NULL
y	2	y	ii
z	3	NULL	NULL



Join Type Example (3)

ALPHA

a	b
x	1
y	2
z	3

BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha RIGHT OUTER JOIN Beta ON
Alpha.a=Beta.c
```

$$Alpha ⟖_{Alpha.a=Beta.c} Beta$$

Alpha.a	Alpha.b	Beta.c	Beta.d
y	2	y	ii
NULL	NULL	w	-

Join Type Example (4)

ALPHA

a	b
x	1
y	2
z	3

BETA

c	d
w	-
y	ii

```
SELECT *
FROM Alpha FULL OUTER JOIN Beta ON
Alpha.a=Beta.c
```

$$Alpha \bowtie_{Alpha.a=Beta.c} Beta$$

Alpha.a	Alpha.b	Beta.c	Beta.d
x	1	NULL	NULL
y	2	y	ii
z	3	NULL	NULL
NULL	NULL	w	-

Join and Renaming

- **Problem**: *R* and *S* might have attributes with the same name
  .
- **Solution**:
  - Rename attributes prior to forming the product and use new names in *join-condition*´.
  - Common attribute names are qualified with relation names in the result of the join

Theta Join – Example

Output the names of all employees that earn more than their managers.

$\Pi_{\text{Employee}.Name}$ (**Employee** $\bowtie_{MngrId=Id \text{ AND } Salary>Salary}$ **Manager**)

The join yields a table with attributes: Employee.*Name*

Equijoin Join - Example

Equijoin: Join condition is a conjunction of *equalities*.

$$\Pi_{Name,CrsCode}(\text{Student} \bowtie_{Id=StudId} \sigma_{Grade=\text{'A'}}(\text{Transcript}))$$

Student

Id	*Name*	*Addr*	*Status*
111	John	…..	…..
222	Mary	…..	…..
333	Bill	…..	…..
444	Joe	…..	…..

Transcript

StudId	*CrsCode*	*Sem*	*Grade*
111	CSE305	S00	B
222	CSE306	S99	A
333	CSE304	F99	A

Mary	CSE306
Bill	CSE304

The equijoin is used very frequently since it combines related data in different relations.

Schema for Student Registration System

Student (*Id*, *Name*, *Addr*, *Status*)
Professor (*Id*, *Name*, *DeptId*)
Course (*DeptId*, *CrsCode*, *CrsName*, *Descr*)
Transcript (*StudId*, *CrsCode*, *Semester*, *Grade*)
Teaching (*ProfId*, *CrsCode*, *Semester*)
Department (*DeptId*, *Name*)

Self-join Queries

Find Ids of all professors who taught at least two courses in the same semester:

```
SELECT  T1.ProfId
FROM  Teaching T1, Teaching T2
WHERE  T1.ProfId = T2.ProfId
    AND  T1.Semester = T2.Semester
    AND  T1.CrsCode <> T2.CrsCode
```

Tuple variables essential in this query!

Equivalent to:

$$\pi_{ProfId}(\sigma_{T1.CrsCode \neq T2.CrsCode\ and}(\text{Teaching T1} \bowtie_{T1.Semester=T2.semester\ and\ T1.profId=T2.profid} \text{Teaching T2})$$

EXAMPLES

```
SELECT City AS city
FROM customer
WHERE Country = 'Canada';
```

$$\rho_{(city)}(\pi_{City}(\sigma_{Country='Canada'}(customer)))$$

```
SELECT * FROM artist
WHERE Name LIKE 'Black%' ORDER BY Name ASC;
```

$$\tau_{Name}(\sigma_{Name\ LIKE\ 'Black\%'}(artist))$$

```
SELECT * FROM artist art INNER JOIN album alb
ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name ASC, alb.Title ASC;
```

$$J \leftarrow \rho_{art}(artist) \bowtie_{art.ArtistId=alb.ArtistId} \rho_{alb}(album)$$

$$S \leftarrow \sigma_{Name\ LIKE\ 'Black\%'}(J)$$

$$RES \leftarrow \tau_{art.Name,alb.Title}(S)$$

```
SELECT art.ArtistId, art.Name, alb.AlbumId, alb.Title
FROM artist art LEFT OUTER JOIN album alb
ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name, alb.Title;
```

$$J \leftarrow \rho_{art}(artist) \text{⟕}_{art.ArtistId=alb.ArtistId}\ \rho_{alb}(album)$$

$$S \leftarrow \sigma_{Name\ LIKE\ 'Black\%'}(J)$$

$$P \leftarrow \pi_{art.ArtistId,art.Name,alb.AlbumId,alb.Title}(S)$$

$$RES \leftarrow \tau_{art.Name,alb.Title}(P)$$

```
SELECT * FROM artist art LEFT OUTER JOIN album alb
ON art.ArtistId=alb.ArtistId
WHERE Name LIKE 'Black%'
ORDER BY art.Name, alb.Title;
```

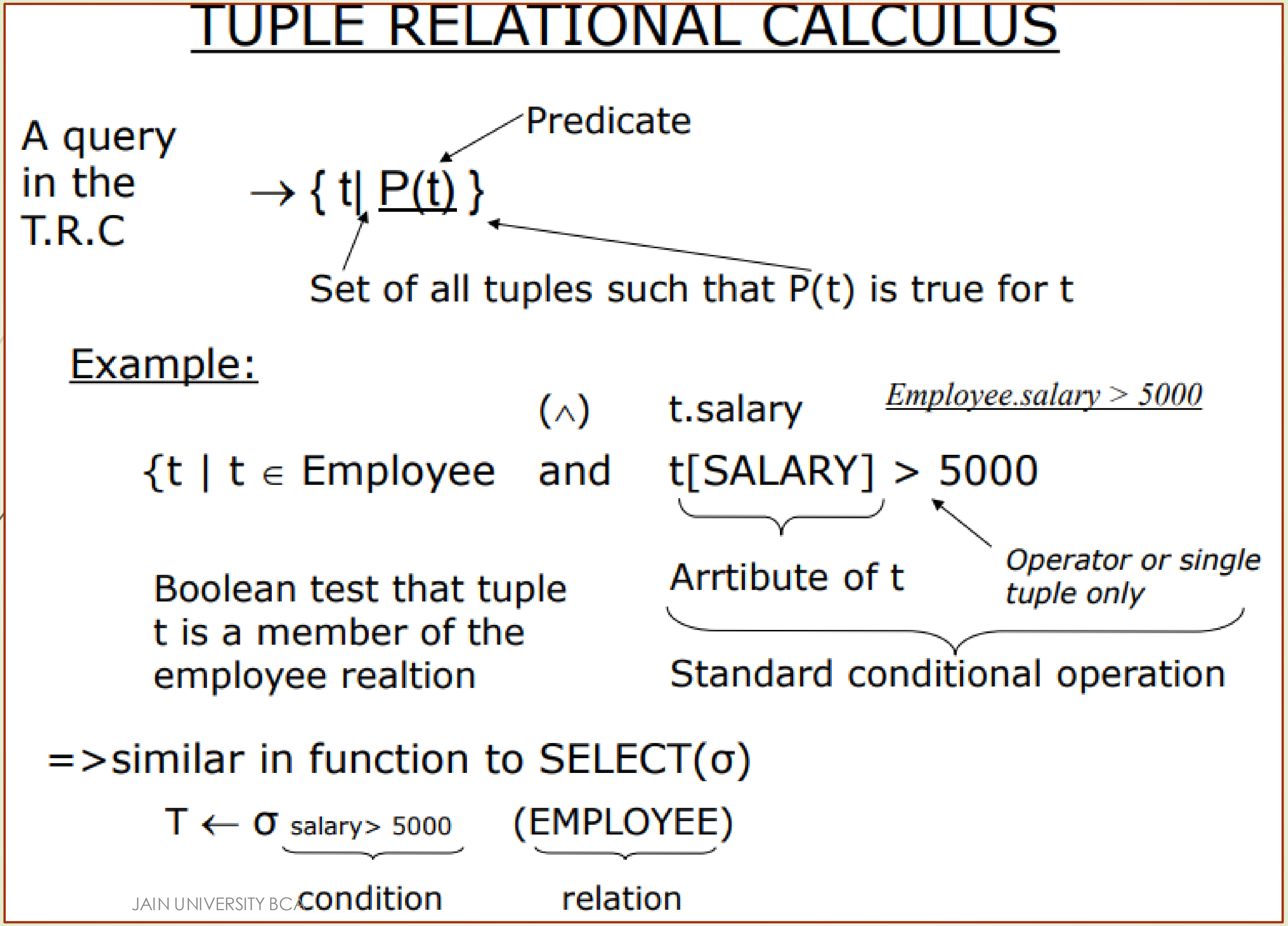
$$J \leftarrow \rho_{art}(artist) \text{⟕}_{art.ArtistId=alb.ArtistId}\ \rho_{alb}(album)$$

$$S \leftarrow \sigma_{Name\ LIKE\ 'Black\%'}(J)$$

$$RES \leftarrow \tau_{art.Name,alb.Title}(S)$$

TUPLE RELATIONAL CALUCULUS

- Relational calculus is a query language which is non-procedural, and instead of algebra, it uses mathematical predicate calculus.
- When applied to databases, it is found in two TRC,DRC.



TUPLE RELATIONAL CALCULUS

Predicate

A query in the T.R.C $\rightarrow \{ t \mid \underline{P(t)} \}$

Set of all tuples such that P(t) is true for t

Example:

(∧) t.salary *Employee.salary > 5000*

$\{t \mid t \in \text{Employee} \quad \text{and} \quad \underbrace{t[\text{SALARY}]}_{\text{Arrtibute of t}} > 5000$

Operator or single tuple only

Boolean test that tuple t is a member of the employee realtion

Standard conditional operation

=>similar in function to SELECT(σ)

$$T \leftarrow \sigma_{\underbrace{\text{salary} > 5000}_{\text{condition}}} \underbrace{(\text{EMPLOYEE})}_{\text{relation}}$$

- Let us define tuple variable (there exist t)£t

First Name	Last Name	Marks
James	Jhonson	90
Jack	Jill	20
Jones	Jim	30

- {t | student(t) AND t.marks>50}

Display all

{t.fname t.lname | student(t) AND t.marks>50}

For example, to specify the range of a tuple variable S as the Staff relation, we write:
Staff(S)

To express the query 'Find the set of all tuples S such that F(S) is true,' we can write:
{S | F(S)}

Here, F is called a formula (well-formed formula, or wff in mathematical logic).
For example, to express the query to 'Find the staffNo, fName, lName, position, sex, DOB, salary, and branchNo of all staff earning more than £10,000',

we can write:

$\{S \mid Staff(S) \wedge S.salary > 10000\}$

{t | TEACHER (t) and t.SALARY>20000}

- It implies that it selects the tuples from the TEACHER in such a way that the resulting teacher tuples will have the salary greater than 20000. This is an example of selecting a range of values.

{t | TEACHER (t) AND t.DEPT_ID = 8}

- T select all the tuples of teachers name who work under Department 8.

Domain Relational Calculus

In domain relational calculus, filtering is done based on the domain of the attributes and not based on the tuple values.
{x1,x2,x3,... | condition(x1,x2,..}

Example:
emp(empno, ename, job, sal)
let us take domain variable as a, b,c,d
dept(deptno, dname, loc)

let us take domain variable x, y,z
Query:
List the name and job of the employee whose name of SCOTT and JOB CLERK
a, b, c, d domain variables
{a,c| there exist b, d(emp(abcd) and (a=‘SCOTT') and (c='CLERK'))}

Example -2

List all the loan number of all the loans with an amount greater than 1000

Loan_no	branch	amount
100	BLORE	1000
200	MYS	1500

{l | b,a there exists(loan(l,b,a) and a>1000)}

List all the branch where loan amount less than 1500

{b | b,a there exists(loan(a,b,a) and a>1000)}

List the employees work for dname Accounting.

{t.ename|emp(t)
AND(∃ d)dept(d) ^ dname='SALES'
AND emp.deptno=dept.deptno}

Obtain the names of courses enrolled by student named Mahesh

{c.name | course(c)
  ^ (∃s) (∃e) (student(s)
  ^ enrollment(e)
  ^ s.name = "Mahesh"
  ^ s.rollNo = e.rollNo
  ^ **c.courseId = e.courseId }**

Get the names of students who have scored ‘S’ in all subjects they have enrolled. Assume that every student is enrolled in at least one course.

{s.name | student(s)
 ∧ (∀e)((enrollment(e)
 ∧ e.rollNo = s.rollNo)
 ∧e.grade =‘S’)}

Determine the departments that do not have any girl students

{d.dname|department(d)
 ∧ ¬(∃ s)(student(s)
 ∧ s.sex =‘F’
 ∧ s.deptNo = d.deptId)