The relational Data model concepts:

The relational model uses a collection of tables to represent both data and the relationships among those data.

A table is a collection of rows and columns. Each column has a unique name.

Each row in a table represent a collection of related data values.

In the relational model, a row is called a tuple & a column is called an attribute and the table is called a relation.

eg. Student (Roll-no, name, city, age)

Roll-NO	Name	city	Age
1	Vijay	Gr. Noida	22
2	Santosh	Delhi	20
3	Gopal	Noida	23

Integrity constraints — Most database application have certain integrity constraints that must hold for the data

Entity Integrity: The entity integrity constraints states that no primary key can be NULL. This is because the primary key value is used to identify individual tuple in a relation, having null values for the primary key implies that we can not identify some tuples.

Referential Integrity: We wish to ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another. This condition is called "Referential Integrity".

→ This constraint establishes a relationship between records across a master and a detail tables.

Relational Algebra (Procedural query language)

Relational algebra is a collection of operations that are used to manipulate the entire set of relations. The output of any relational algebra operation is always a relation.

There are several operations like —
(Set operations) →
- Union, intersection, difference, & cartesian product
- Selection, Projection, and Joining.

Select operation(σ)

Syntax: $\sigma_{<selection\_condition>}(Relation)$

eg. Find the employees whose salary is greater than 10,000 rupees.

$\sigma_{salary > 10,000}(Employee)$

One of the interesting properties of the selection operation is that it is commutative.

Therefore, all the expressions shown below are equivalent;

$\sigma_{condition-1}(\sigma_{condition-2}(R))$

$\sigma_{condition-1 \text{ AND } condition-2}(R)$

$\sigma_{condition-2}(\sigma_{condition-1}(R))$

Projection operation (π)

It is used to select only few columns from a relation.

$\pi_{<attributes>}(Relation)$

eg. List name and salary of all the employees.

$\pi_{Name, salary}(Employee)$

eg. Find the name, address & salary of the employees who earn more than 25000 rupees.

$\pi_{name, salary, addr}(\sigma_{salary > 25000}(Employee))$

Union (U)

The result of this operation, i.e. R U S, is a relation that include all tuples that are either in R or in S or in both.
Duplicate tuples will not appear in the output.

Union Compatibility: Let R & S are two relations with attributes $(A1, A2 \ldots, An)$ and $(B1, B2 \ldots Bn)$
If R & S are to be unioned, it should satisfy the following two rules:

Rule 1: The relation R & S must have same degree. i.e, the number of attributes of R and S must be same.

Rule 2: The domain of ith attribute of R and domain of the ith attribute of S must be same.
$dom(Ai) = dom(Bi)$, where, $1 \leq i \leq n$.

Select operation –

Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D>5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

$\Pi_{A,C}(r)$

A	C
α	1
α	5
β	12
β	23

UNION operation –

Relation r, s

A	B
α	1
α	2
β	1

(r)

A	B
α	2
β	3

(s)

→

A	B
α	1
α	2
β	1
β	3

(r ∪ s)

Set Difference operation –

A	B
α	1
α	2
β	1

(R)

A	B
α	2
β	3

(S)

A	B
α	1
β	1

R – S

Rename: (P) rho

The results of Relational-Algebra expressions do not have a name that we can use to refer to them.

eg. Given a relational-algebra expression E, the expression

$\rho_x(E)$

returns the result of expression E under the name x.

ρ_d(account)

↳ table name.

Cartesian Product

Find the cartesian product instructor x teaches.

→ select * from instructor, teaches.

Find the names of all instructors who have taught some course and the course_id (equijoin, Natural join)

→ select name, course_id from instructor, teaches
where instructor.ID = teaches.ID.

→ Find the names of all instructors in the Art department who have taught some course and the course_id

select name, course_id from instructor, Teaches
where instructor.ID = Teaches.ID and instructor.dept_name = 'Art'

Cartesian Product operation —

Relations

A	B
α	1
β	2

(R)

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

(S)

R x S

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Intersection (∩)

The expression R ∩ S returns all tuples that appear in both the relations R & S.

Difference

This operation, written as R−S (set difference) returns all tuples that are in R but not in S.

Cartesian Product (×)

The cartesian product or cross-produ
binary operation that is used to c
two relations. Assuming R & S as relations
with n & m attributes respectively, the
cartesian product R×S can be written as —

R (A1, A2, ... An) × S (B1, B2 ..., Bm)

The result of the above set operation is
Q (A1, A2 ... An, B1, B2 ..., Bm)

where, Degree (Q) = n+m

Count (Q) = Number of tuples in R * Number of tuples in S.

10/40

Division

Consider two relations R & S. Assume that R has two attributes a_1 & a_2 & S has only one attribute a_2 with the same domain as in R. This is to ensure that the degree of the numerator is more than the degree of denominator. Now we shall define R/S as

a_1 is the result.

Divison ÷

R

A	B
α_1	β_1
α_2	β_1
α_1	β_2
α_3	β_2

S

B
β_1
β_2

A(A,B) / S(B)

It results A values for that there should be tuple (A,B) for every y value of relation B.

Used in queries where every / all is used

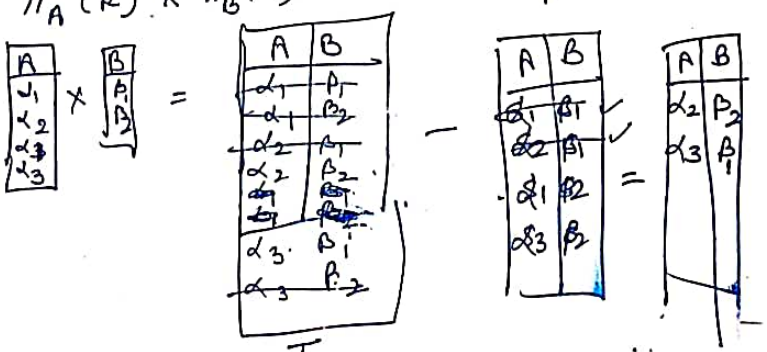
eg. Select A students id who enrolled for all / every courses.

output

A
α_1

(1) calculate × (cross Product)

$\Pi_A(R) \times \Pi_B(S)$



T

(2) T − R

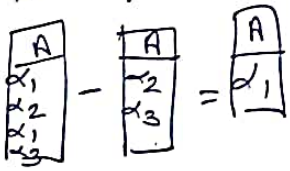
(3) $\Pi_A(T-R)$ → Project A from T−R

A
α_2
α_3

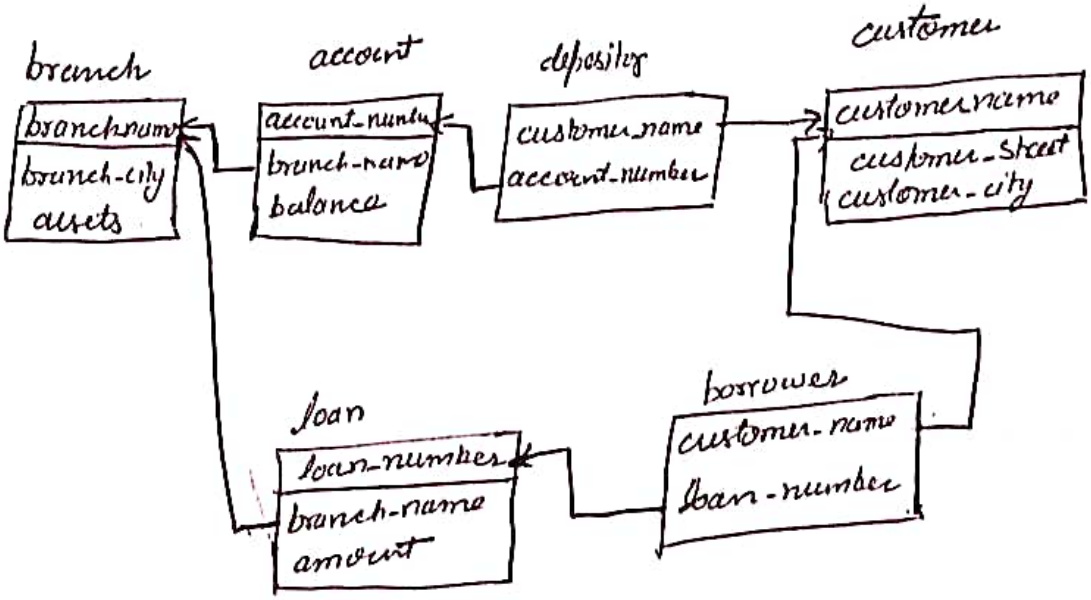
= W

(4)

$\Pi_A(R) - W$



$$\Pi_A(R) - \left(\Pi_A\left(\left(\left(\Pi_A(R)\right) \times \Pi_B(S)\right) - (R)\right)\right)$$



Schema diagram for the banking enterprise.

Q. Find all loan made at "Perryridge" branch.

$$\sigma_{branch\_name = \text{"Perryridge"}}(loan).$$

Q. Find all loans of over $1200

$$\sigma_{amount > 1200}(loan).$$

Q. Find all tuples who have been taken loans of more than $1200 made by the "Perryridge" branch.

$$\sigma_{branch = \text{"Perryridge"} \wedge amount > 1200}(loan)$$

15/40

Q. Find all loan numbers and the amount of loans.

$$\Pi_{loan\_number, amount}(loan)$$

Q. Find the loan number of each loan of an amount greater than $1200.

$$\Pi_{loan\_number}(\sigma_{amount > 1200}(loan))$$

Q. Find those customers who live in "Harrison".

$$\Pi_{customer\_name}(\sigma_{customer\_city = \text{"Harrison"}}(customer))$$

→ Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

$\Pi_{customer\_name}(\sigma_{branch\_name = "Perryridge"}(\sigma_{borrower.loannumber = loan.loan\_number}(borrower \times loan))) - \Pi_{customer\_name}(depositor)$.

→ Rename

Find the largest account balance in the bank.

$\Pi_{balance}^{(account)} - \Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_{d}^{(account)}))$

Step ② Step ①

Project balance that are not largest (as a temporary relation)

Rename account table as d for Cartesian Product with self.

Q Suppliers (S_id, S_name, S_addd)

Parts (P_id, P_name, color)

catalog (S_id, P_id, cost)

write the following query

(a) Find the name of suppliers who supply yellow parts

(b) Find the name of suppliers who supply both blue and green parts.

(c) Find the name of suppliers who supply all parts.

parts.

Relational Algebra query:

(a) π_{sname} (Supplier $\bowtie$ catalog $\bowtie$ ($\sigma_{color = 'yellow'}$ (Parts)))

(b) $\pi_{s\_name, color}$ (Supplier $\bowtie$ catalog $\bowtie$ Parts) $\div$ π_{color} ($\sigma_{color = 'blue' \text{ or } color = 'green'}$ (Parts))

(c) $\pi_{s\_name, color}$ (Supplier $\bowtie$ catalog $\bowtie$ Parts) $\div$ π_{color} (Parts).

Relational calculus

In contrast to Relational Algebra, Relational calculus is a non-procedural query language i.e, it tells what to do but never explains how to do it.

Tuple Relational Calculus (TRC)

- filtering variable ranges over tuples
- Notation — $\{T \mid \text{condition}\}$ i.e, $\{t \mid P(t)\}$ → condition. ↳ Resulting tuple.
- Return all tuples T that satisfies a condition.
- $P(t)$. → Predicate.
- eg. $\{T.name \mid Author(T) \text{ AND } Tarticle = 'database'\}$
- output - Returns tuples with 'name' from Authors who has written article on 'database'.
- TRC can be quantified. We can use Existential($\exists$) and Universal Qualifiers ($\forall$).
- $\exists \in r(Q(t))$ = "there exists a tuple in t in relation r such that Q(t) predicate is true
- $\forall t \in r(Q(t))$ = Q(t) is true "for all" tuples in relation r.

Domain Calculus.

In DRC, the filtering variables uses the domain of attributes instead of entire tuple values —

Notation

→ resulting domain

$$\{ a_1, a_2, a_3, \ldots, a_n \mid P(a_1, a_2, a_3, \ldots, a_n) \}$$

where a_1, a_2 are attributes and P stands for formulate built inner attributes.

eg:

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

$\langle a_1, a_2, \ldots, a_n \rangle \in r$

output – Yields Article, page, and subject from the relation Tutorialspoint, where subject is database.

Just like TRC, DRC can also be written using existential and universal quantifiers.

Q. Find loan number, branchname and amount for loans of greater than 1200.

$$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$$

Q. Find all loan number for loans with an amount greater than $1200.

$$\{ \langle l \rangle \mid \exists b, a \, (\langle l, b, a \rangle \in \text{loan} \wedge a > 1200) \}$$

tuple calculus

Q. Write a query in SQL to display Sname of suppliers.

$$\{ S.\text{Sname} \mid \text{supplier}(S) \}$$

↳ S is a variable of supplier table, will check all rows.

Q. Write a query to display Pname of parts whose color is red.

$$\{ P.\text{name} \mid \text{Parts}(P) \wedge P.\text{color} = \text{'Red'} \}$$

Q. Write a query to display SID of suppliers whose name is 'John' and address is 'Delhi'.

$$\{ S.\text{sid} \mid \text{supplier}(S) \wedge S.\text{name} = \text{'John'} \wedge S.\text{add} = \text{'Delhi'} \}$$

Q. Write a query sir to display SID of suppliers who supplied some parts.

Q. Write a query in SQL to display Sname of suppliers who supplied red color parts.

$$\{ s.sname \mid supplier(s) \wedge \exists c(catalog(c)) \wedge \exists p(parts(p) \wedge s.sid = c.sid \wedge p.pid = c.pid \wedge p.color = 'Red') \}$$

s is free variable
c & p is bounded variable.

Scanned with CamScanner

Domain calculus

Find the names of all customers who have loan from "pennyridge" branch and find the loan amount.

$$\{ \langle c, a \rangle \mid \exists l (\langle c, l \rangle \in borrower \wedge \exists b (\langle l, b, a \rangle \in loan \wedge b = "Pennyridge")) \}$$

Find the names of all customers who have a loan an account or both at Pennyridge branch.

$$\{ \langle c \rangle \mid \exists l (\langle c, l \rangle \in borrower \wedge \exists b, a (\langle l, b, a \rangle \in loan \wedge b = "Pennyridge")) \vee \exists a (\langle c, a \rangle \in depositor \wedge \exists b, n (\langle a, b, n \rangle \in account \wedge b = "Pennyridge")) \}$$

→

Q. Find the loan no., branchname and amount for loans of over $1200.

$$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in loan \wedge a > 1200 \}$$

Tuple Relational calculus.

Unsafe expression:

$\{ S.name \mid \neg Supplier(S) \}$

name the tuple supplier which does not exist in supplier. It will stuck in infinite loop.

SQL (Structured Query language)

is a language that provides an interface to relational database systems.

Advantages of SQL

(1) Standard independent language:
The universal rules of the SQL have been established by ANSI and ISO. Therefore, the SQL is an open language implying it is not owned or controlled by any single company.

(2) Speed: The intense competition among database vendors has resulted in faster, more robust database management systems that works at lower cost per transaction.

(3) Easy to learn & use:- SQL statements resemble simple English sentences making SQL easy to learn & understand.

(4) Less programming: SQL allows extraction, manipulation and organisation of data with less programming as compared to traditional methods.

Scanned with CamScanner

SQL Commands

- create table

syntax: create table tablename (column_name datatype(size), column_name datatype(size));

- Insertion of Data into tables:

Insert into tablename (column name 1, column_name 2) values (expression, expression);

- Select command

Select * from table name

- Delete

Delete from table_name

Delete from table_name where search condition

Group by

Select <column(s)> from <table>
where <condition>
group by <group_column>
[having <group condition>]

select deptno, min(sal), max(sal)
from emp
group by deptno

SQL Queries

Set operations: union, intersect, difference (except)
(∪) (∩) (−)

④

① Union Operation

To find all customers having a loan, an account, or both at the bank

(Select customer_name from depositor)
union
(Select customer_name from borrower)

$\Pi_{customer\_name}(depositor) \cup \Pi_{customer\_name}(borrower)$

If we want to retain all duplicates, we must write union all in place of union.

select customer_name from depositor
union all
select customer_name from borrower

② The Intersect Operation:

$R \cap S = (R \cup S) - ((R - S) \cup (S - R))$

To find all customers who have both a loan and an account at the bank.

select distinct customer_name from depositor
intersect
select distinct customer_name from borrower

$\Pi_{customer\_name}(depositor) - \Pi_{customer\_name}(borrower)$

③ Difference (except) Operation

To find all customers who have an account but no loan at the bank

select distinct customer_name from depositor
except
select customer_name from borrower

If we want to retain all duplicates, we must write except all in place of except.

select customer_name from depositor
except all
select customer_name from borrower.

Union

Q→ Find the names of all customers who have a loan, an account, or both from the bank:

$$\Pi_{\underset{name}{customer}}(borrower) \cup \Pi_{customer\,name}(depositor)$$

Intersection

Q → Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\_name}(borrower) \cap \Pi_{custom\_name}(depositor)$$

difference

Q → Find the names of all customers who have an account but no loan from the bank.

$$\Pi_{customer\_name}(depositor) - \Pi_{customer\_name}(borrower)$$

cartesian Product ×

Find the names of all customers who have a loan at Perryridge branch.

① borrower × loan

$$\Pi_{customer\,name}(\sigma_{branch = "Perryridge"} \; \sigma_{borrower.loan\,number = loan.\underset{number}{loan}} (borrower \times loan))$$

29/40

acno	balance
1	500
2	700
3	900
4	900

×

1	500
2	700
3	900
4	900

ano	balance
1	500
2	700
3	900
4	800

×

1	500
2	700
3	900
4	800

no	balance	no	balance
1	500	1	500
1	500	2	700
1	500	3	900
1	500	4	800
2	700	1	500
2	700	2	700
2	700	3	900
2	700	4	800
3	900	1	500
3	900	2	700
3	900	3	900
3	900	4	800
4	800	1	500
4	800	2	700
4	800	3	900
4	800	4	800

sal

500
~~500~~
700
800

→ ✓

Aggregate function

Aggregate function are functions that take a collection of values as input and return a single value. SQL offers five suit in aggregate function.

1) AVG — select Avg(salary) from Emp.
2) MIN } select min(salary), max(salary) from emp
   MAX } where Deptno = 30.

Aggregate Function

Aggregate function are functions that take a collection of values as input and return a single value. SQL offers five suit in aggregate function.

(1) AVG — select Avg(salary) from Emp.

(2) MIN } select min(salary), max(salary) from emp
(3) Max }

(4) SUM select sum(salary) from emp where Deptno = 30.

(5) COUNT → how many tuples are in relation

select count(*) from Emp

select count(distinct Job) from Emp

Index

An index is an ordered list of content of a column or group of column in a table.

Creating an Index:

- Syntax:

```
create index index_name
ON tablename (columnname);
```

Dropping indexes:

An index can be dropped by using

- Drop index

Syntax:

```
Drop INDEX indexfilename.
```

Order by

→ list in alphabetic order the names of all instructors

select distinct name from instructor order by name.

Between

find the names of all instructors with salary between 90,000 and 100,000

select name from instructor where salary between 90,000 and 100,000

Tuple comparison

select name, course_id from instructor, teaches where (instructor.ID, dept_name) = (teaches.ID, 'Biology')

Views and indexes.

View → It is how we want to see the current data in our database.

Syntax [or replace]

create view viewname AS

select columnname, columnname

from tablename

where column = expression-list

Renaming the coolumn viewname AS follows

Select columnname, columnname

From tablename

where column = expression-list

Renaming the coolumn viewname AS follows

Syntax :

Create view viewname AS

Select newcolumnname. . . .

from tablename

where columnname = expression-list

Procedures in PL/SQL

Procedure is a logical group of SQL and PL/SQL statements that performs a specific action.

Types of PL/SQL Procedures:

(i) Local or Anonymous procedure : Part of an anonymous block which is not stored as an object in the Oracle database. called from the executable section of that block. It cannot be called from inside of any other procedure or function.

(ii) Stored Procedure: Stored in the database as database objects (in the compiled form called P-code). It can be invoked from various environments that have access to the databases (eg, other procedures).

Trigger

Trigger is a PL/SQL block structure which is fired when a DML statements like Insert, delete, update is executed, on a database table.

Syntax create [or Replace] Trigger Trigger name

[Before / After / Instead of]

{ Insert [OR] / Update [OR] / Delete } on table name

[referencing OLD as a new]

[for each row]

when (condition)

BEGIN

---SQL Statements

END;

Cursor

The oracle Engine uses a work area for its internal processing in order to execute an SQL statement. This work area is called cursor.

The data stored in the cursor is called the "Active Data Set".

eg: cursor <cursor name> IS SQL statement

Types of cursor

Implicit cursor:- If the oracle engine for its internal processing has opened a cursor, they are known as implicit cursor. That is a cursor which opens oracle engine for its internal processing is ~~internal~~ implicit cursor.

Explicit Cursor: A user can also open a cursor for processing data as required, such user defined cursors are known as Explicit cursor.

Scanned with CamScanner

Outer joins.

Dangling tuples in join.

Usually, only a subset of tuple of each relation will actually participate in a join. i.e. only tuple that match with the joining attributes. Tuples of a relation not participating in a join are called dangling tuples.

- → left outer join
- → right outer join
- → full outer join

Join

The Join operation, denoted by ⋈, used to combine related tuples from two relations

The Join operation can be stated in terms of a cartesian Product followed by a select operation.

The general form of a Join operation on two relations R & S

$$R \bowtie_{\text{<Join condition>}} S$$

In Join only combination of tuples satisfying the join condition appear in the result, where as in CARTESIAN PRODUCT all combination of tuples are included in the result.

Join = Cartesian Product + Selection

Types of Join

1. Inner Join (Join)
   - → Theta Join
   - → Equi Join
   - → Natural Join
2. Outer Join.
   - → Left Outer Join
   - → Right Outer Join
   - → Full Outer Join.

Inner Join:

An Inner join includes only those tuples that satisfy the matching criteria,

1) Theta join / conditional join: It joins two or more relation based on some conditions.

It uses all kinds of comparison operators like $<$, $>$, $<=$, $>=$, $=$, $\neq$.

Syntax: $R \bowtie_\theta S$ θ is a predicate / condition

$$R \bowtie_\theta S = \sigma_\theta (A \times B)$$

R

Sid	Name	age
1	A	21
2	B	20
5	C	21

S

Sid	C-Name	C-code
2	DBMS	301
4	C++	101
6	OS	201

$$R \bowtie_{R.sid <= S.sid} S$$

equivalent to

$$\sigma_{R.sid <= S.sid} (R \times S)$$

Natural Join.

Natural Join can only be performed if there is at least one common attribute (column) in two relations. In addition, the attributes must have the same name & domain.

→ Natural Join does not use any comparison operators.

→ It is same as equijoin which occurs implicitly by comparing all the common attributes (columns) in both relation, but difference is that in Natural join the common attribute appears only once.

Syntax $R \bowtie S$

→ Natural Join of two relations can be obtained by applying a Projection operation to equijoin of two relation.

Natural Join = Cartesian product + selection + Projection.

$R(A\ B\ C\ D)$ $S(B\ D\ E)$

$R \bowtie S = (A, B, C, D, E)$

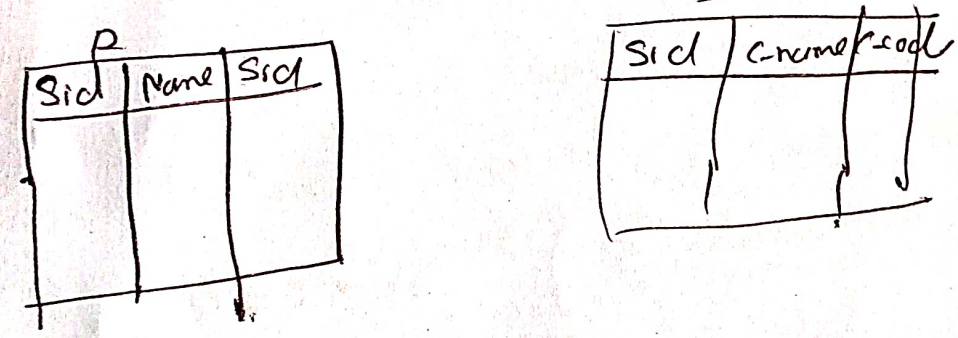
$\Pi_{R.A, R.B, R.C, R.D, S.E}\left(\sigma_{R.B = S.B \wedge R.D = S.D}(R \times S)\right)$

Equi Join

→ When we use = condition in theta join, it becomes a Equi join.

→ Equi join is a special case of theta join where condition contains equalities (=).

$$R \bowtie_{R.a1 = S.b1 \wedge \ldots\ldots \wedge R.an = S.bn} S$$



$$R \bowtie_{R.sid = S.sid} S$$

equivalent to

$$\sigma_{R.sid = S.sid} (R \times S).$$

Course

C-id	CourseName	Dept
CS101	DBMS	CS
ME101	DS	ME
EC101	IOT	EC

HOD

Dept	Head
CS	A
ME	B
EC	C

Course ⟗ HOD

$\Pi_{CID, coursename, course.dept, head}(\sigma_{course.dept = HOD.dept}(course \times HOD))$

Join operations

- Natural join (*)
- Equijoin ($\bowtie$)
- Theta join (θ)
- outer join ($\overline{\bowtie}$)

Natural Join : When we omit the condition during joining of two relations, then it is called as natural join (*).

eg.

Employee (SSN, Name, Addr, DNo)

Department (DNo, DName)

Theta Join(θ)

Let R & S are two relations. Consider an attribute x in R and an attribute y in S. The thetajoin of these two relations can be written as,

$R \times S$ where $x \theta y$

where θ indicates a valid relational operator.

(iii) Full outer Join :- ⟗

EId	Name	Age	D.No.	D-Name
1	A	20	101	D_1
5	NULL	NULL	102	D_3
5	C	25	103	D_4
2	B	23	NULL	NULL

DBMS

11/Oct/2021

Join ($\bowtie$) :- There are 2 types of Join :-

(i) Outer Join
- @ Left outer Join
- (b) Right outer Join
- (c) Full outer Join

(ii) Inner Join
- @ Theta Join
- (b) Equi Join
- (c) Natural Join.

@ Left outer Join :- $\bowtie$

Employee :-

E-Id	Name	Age
1	A	20
2	B	28
3	C	25

Department

D-No	D-Name	E-Id
101	D_1	1
102	D_3	5
103	D_4	3

E-Id	Name	Age	D-No	D-Name
1	A	20	101	D_1
2	B	28	NULL	NULL
3	C	25	103	B_4

(b) Right Outer Join :- $\bowtie$

E Id	D-Name	D-No	Name	Age
1	D_1	101	A	20
5	D_3	102	NULL	NULL
3	D_4	103	C	25