DBMS

Relational Algebra.

↳ Procedural Query language.

1. Select = Retreive data from tuple.
2. Project = Retreive data from column (attribute)
3. Join = Combine multiple tables into one.
4. Outer Join = Includes null values also.
5. Inner Join = Excludes null values.

Important operations.

1. Select (σ (select condn) (R)
2. Project (Π (Attribute list) (R)

❀ example:- Relational Schema

Professor (emplid, name, Startyear, deptNo, phone)

Q1.) obtain information about a professor with name = " Alan ".

$$\sigma_{name = \text{"Alan"}}(professor)$$

Relational Schema 'R'.

↳ Condition

Q2.) obtain info^n about professors who joined b/w 1990 and 1995.

$$\sigma_{startyear \geq 1990 \wedge startyear < 1995}(professor)$$

↳ Condition.

Example of Project operator.

→ "Π": used to keep only the required attributes of a relation and throw away others.

→ Any duplicate tuples if selected are removed.

$\Pi_{(A1, A2, \ldots Ax)}(r)$ → Relation 'R'

list of Attributes

Student:

Roll No.	Name	Degree
CS034	ABC	M.S
CS676	Mohit	M.E
CS011	Aman	M.S

Q.1) Display names of students holding "M.S" degree.

$\Pi_{Name}(\sigma_{Degree = M.S}(Student))$ → R.

Attribute.

result:

Name
ABC
Aman

Q.2)

A	B	C
α	15	6
β	12	6
α	10	5
α	8	5

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

A	C
α	6
β	6
α	5

Duplicate tuples are removed.

Set operations on Relations

Set operations are applicable if the relations are union compatible.

Union compatibility:

i) The Relation 'R_1' and 'R_2' must be of the same arity (degree). That is, they must have same number of attributes.

ii) The domains of the ith attribute of R_1 and the ith attribute of R_2 must be same for all i.

① Union operation:

denoted by (R ∪ S), is a relation that includes all tuples that are either in R or in S. It eliminates duplicates.

R

id	name	age
25	Punit	20
26	Pooja	25
58	Saurabh	32

∪

S

id	name	age
18	Mohit	36
26	Pooja	25

R ∪ S

25	Punit	20
26	Pooja	25
58	Saurabh	32
18	Mohit	36

Example:

Find names of all bank customers who have either an account or a loan or both

$$\Pi_{\text{customer-name}}(\text{borrower}) \cup \Pi_{\text{customer-name}}(\text{depositor})$$

↳ union operation

② Intersection:

denoted by '∩', 'R ∩ S' is a relation that includes all tuples that are in both R & S.

Q.1 Find all the customers who have both an account and have taken loan.

↳ Intersect operation

$$\Pi_{\text{customer-name}}(\text{borrower}) \cap \Pi_{\text{customer-name}}(\text{depositor})$$

③ Set difference:

denoted by '—'. It allows us to find tuples that are in one relation but are not in another.

(R - S) produces a relation containing those tuples in R but not in S.

Q.1 Find all customers of bank who have an account but not a loan.

$$\Pi_{\text{c-name}}(\text{depositor}) - \Pi_{\text{c-name}}(\text{borrower})$$

↳ Set diff.

④ Cross Product:

It produces all combination of tuples.

R_1 = 's' tuples and 'm' attributes.

R_2 = 't' tuples and 'n' attributes.

$R_1 \times R_2$ = (m+n) attributes and (s*t) tuples.

example:

R

A	B
α	2
α	3
β	1

S

A	B
γ	101
γ	102

R×S →

A	B	C	D
α	2	γ	101
α	3	γ	101
β	1	γ	101
α	2	γ	102
α	3	γ	102
β	1	γ	102

Rename Operation

1. $\rho_S(R)$: Renaming R to S
2. $\rho_S(B_1, B_2, B_3, \ldots B_n)(R)$:

Example:

1. $\rho_{worker(ename, cname, sal, add)}(employee)$
   - worker → new relation name
   - (ename, cname, sal, add) → new attribute name
   - employee → old
   - ↳ Both relation name and attribute names are renamed.
2. $\rho_{worker}(employee)$
   - ↳ Only relation name is renamed.
3. $\rho_{(ename, cname, sal, add)}(employee)$
   - ↳ Only attribute name is renamed.

Join operators.

It is used to combine related tuples from two relations into single tuple.

denoted by '⋈'

$R(A_1, A_2, ----, A_n)$ and $S(B_1, B_2, ----, B_m)$

$R \bowtie_{(\text{JOIN COND}^n)} S$

↳ used to filter the records

Note: In Join, only combinations of tuples satisfying the join condition appear in the result, whereas in the Cartesian product all the combination of tuples are included in the result.

Types of Joins.

① Equi Join: Join conditions with equality comparison only. Only '=' operator is used.

② Natural Join: It is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.

example: ⊛

Outer Join: A set of operations, called outer joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the Join, regardless of whether or not they have matching tuples in other relation.

① **LEFT Outer Join:** Keeps every tuple in the first or left relation R in $R \bowtie S$

eg:- T → Relation

A	B
a	1
b	2

U → Relation

B	C
1	x
1	y
3	z

$T \overset{C}{\bowtie} U$

A	B	C
a	1	x
a	1	y
b	2	

null Value.

② **RIGHT Outer Join:** Keeps every tuple in the second or right relation S in $R \bowtie S$

$T \bowtie C =$

A	B	C
a	1	x
a	1	y
	3	z

null Value

③ **FULL Outer Join:-**

$T \bowtie U$

A	B	C
a	1	x
a	1	y
b	2	NULL
NULL	3	z

Theta Join:

<cond^n> AND <cond^n> ....

where each cond^n is of the form $A_i \; \theta \; B_j$.

A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and θ (theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$.

SQL JOIN.

used to combine records from two or more tables in a database.

Syntax:

```
Select col1, col2, col3, ....
from table1, table2
where table1.col1 = table2.col2;
```

Types of Joins

1. Inner Join:- Returns rows when there is match in both tables.
2. Left Join
3. Right
4. Full
5. Self:- used to join a table to itself.

Equijoin

uses equal sign as the comparison operator.

Non-equi join

↳ uses comparison operator other than equal sign like >, <, >=, <=

eg:- Select empno, ename, emp.deptno, dname from emp, dept

where emp.deptno = dept.deptno

- Inner join

```
Select Columns
  from -table1
  Inner join -table2
  On -table1.col = -table2.col.
```

Left Outer Join

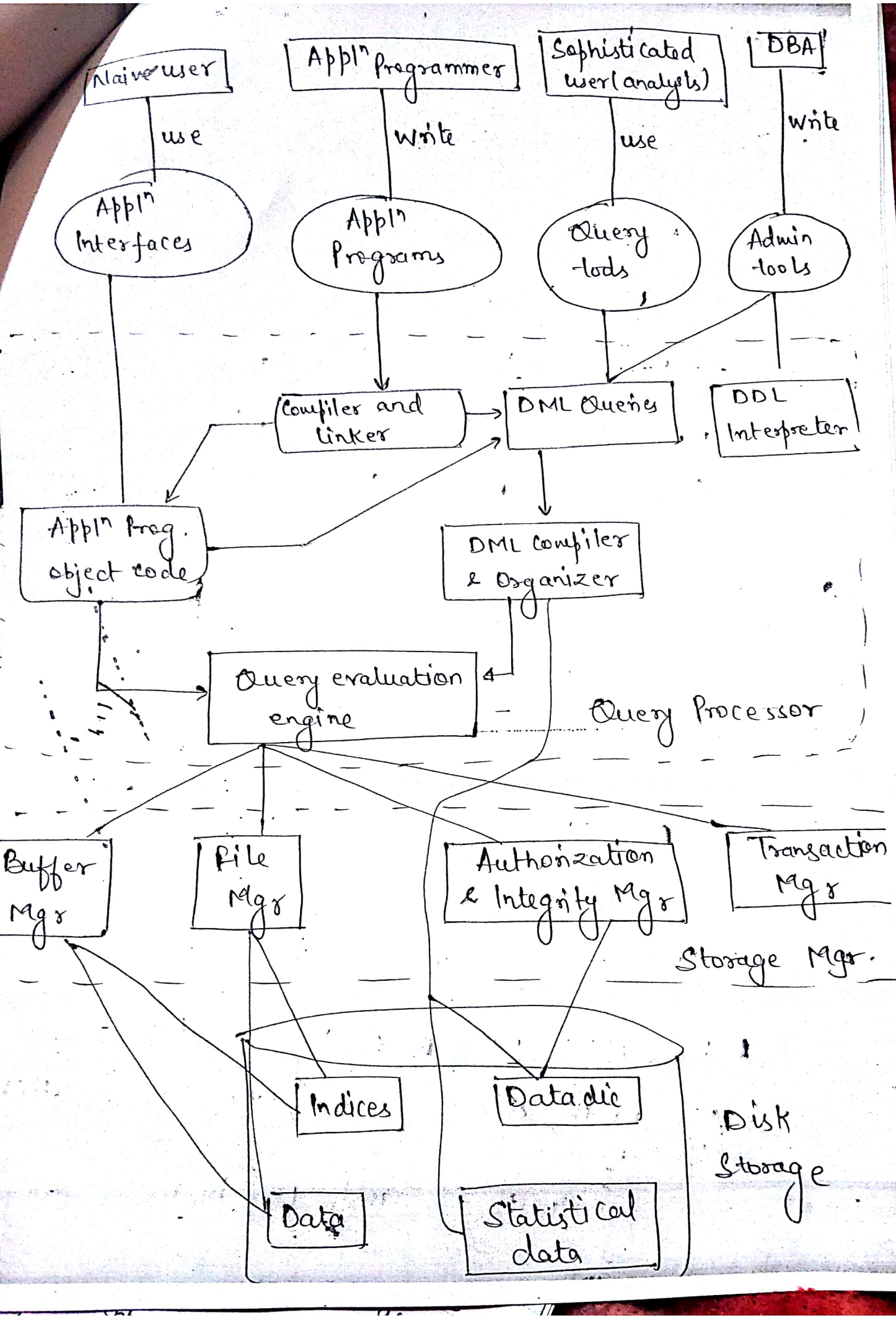
```
Select col
  from -table1
LEFT OUTER JOIN -table2
  on table1.col= table2.col
```

→ Right outer join

→ Full Outer join

→ emp.deptno(+) = dept.deptno [left outer join]

→ emp.deptno = dept.deptno(+) [Right outer join]



Reduction of ER diagram to tables

① Tabular Representation of Strong Entity Sets

E = Strong entity set

$a_1, a_2, \ldots a_n$ = attributes

→ Represent this entity by a table called (E) with (n) distinct columns, each of which corresponds to one of the attributes of E.

② Tabular Rep^n of Weak Entity Sets

A = weak Entity Set

$a_1, a_2, \ldots a_n$ attributes.

B = Strong entity set on which A depends on.

Primary Key of B = $b_1, b_2, \ldots b_n$

Represent entity A by a table called 'A' with one column for each attribute of set.

$$\{a_1, a_2, \ldots a_m\} \cup \{b_1, b_2, \ldots b_n\}$$

eg:- Pr.

loan_no	Pay_no	Pay_date	Pay_amt

③ Tabular Repⁿ of Relationship sets

R = relation

$a_1, a_2 \ldots a_m$ = be the set of attributes formed by the union of the primary keys of each of the entity sets participating in R, and let the descriptive attributes of R be $b_1, b_2, \ldots b_n$.

We represent this relationship set by a table called 'R' with one column for each attribute of the set:

$$\{a_1, a_2, \ldots a_m\} \cup \{b_1, b_2, \ldots b_n\}$$

eg:- Relationship set (Borrower)

↳ Customer → Pr. key cust_id

↳ loan → Pr. key loan_no.

Borrower table

cust_id	loan_no

Composite attributes

↳ Create separate attribute for each of the component attribute.

↳ do not create separate column for composite attribute.

eg:- address → address street, address city

table would contain two columns [address street] and [address city]

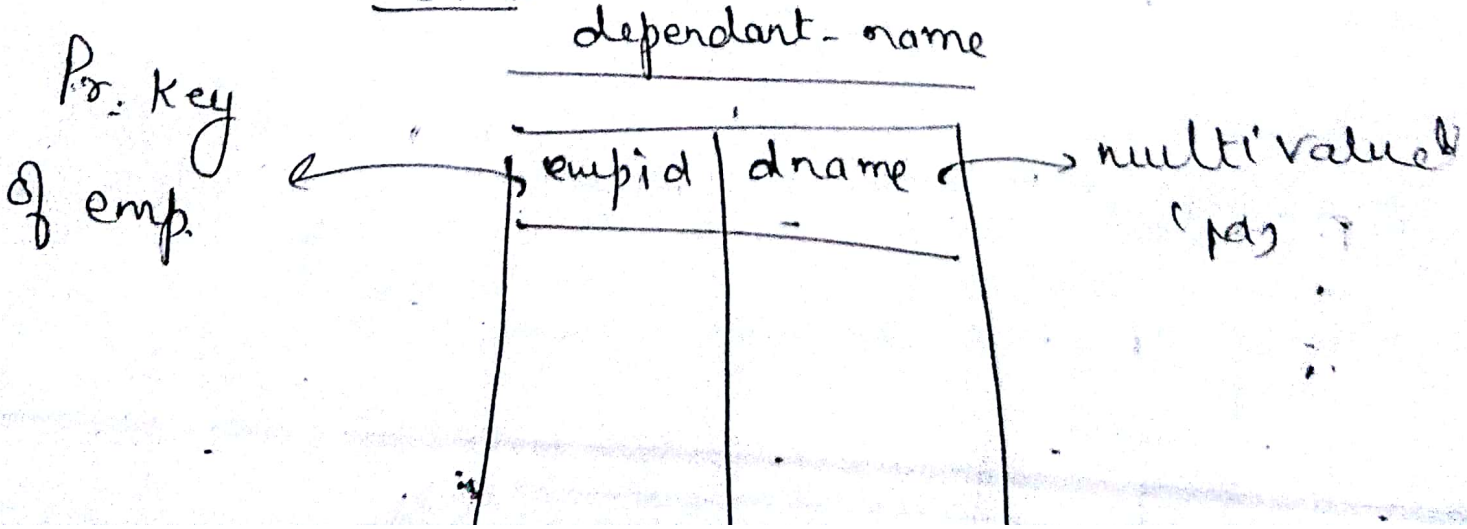
Multivalued attribute (M)

Create table 'T' with column 'C' that corresponds to 'M' and columns corresponding to pr. keys ~~of the entity~~

eg:- dependant_name is multivalued

table.

dependant_name



- Natural Join :- It is a binary operation that allows combine certain selections & a Cartesian product one operation.

denoted by "join" ⋈ Symbol.

→ The natural join - operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas.

→ It removes duplicate attributes.

Division operation:-

denoted by '÷'. (for all)

'R ÷ S' → Result consists of the restrictions of tuples in R to the attribute name unique to R, i.e. in the header of R but not in the header of S, for which it holds that all their combinations with tuples in S are present in R.

eg:-

R

Student	Task
A	DB
A	DB2
B	Comp
C	DB
D	Compl-

S

Task
DB
DB2

R ÷ S

Student
A
C

Assignment operation

denoted by '$\leftarrow$'

$temp \leftarrow \Pi_R(S)$ → Can be any relational algebra expression.

$temp_1 \leftarrow \Pi_A(S)$ →

$result \leftarrow temp - temp_1$

Generalized Projection

extends the projection operation by allowing arithmetic functions to be used in the projection list.

eg:-

$$\Pi_{c\text{-name}, \underbrace{(limit - balance)}_{\text{Arithmetic expression}} \text{ as } c\text{-balance}}(R)$$

Aggregate Functions

↳ Takes a collection of values and returns a single value as a result.

eg:- $G_{sum(salary)}(R)$

↳ this implies that aggregation is to be applied.

$G_1, G_2, \ldots, G_n \; \mathcal{G} \; F_1(A_1), F_2(A_2), \ldots F_m(A_m)$

$G_1, G_2, \ldots G_n$ → List of attributes on which to group.

$F_1(A_1), F_2(A_2) \ldots$ → Aggregate function.

Modification of the database:

① Deletion:

$r \leftarrow r - E$ → Relational-algebra Query.

(Relation

eg:- Delete all of 'ABC' account records.

$$R \leftarrow R - \sigma_{\text{cust-name} = \text{"ABC"}}(R)$$

② Insertion:

$$r \leftarrow r \cup E$$

$$R \leftarrow R \cup \{(101, \text{"A1"}, 100)\}.$$

③ Updation: use Generalized projection.

$$R \leftarrow \Pi_{\text{a-no, bal} * 5}(R)$$

→ Increments "balance value".

Ques) update Rel^n 's' → (i) Increment sal by 6% if balance is more than 1000 and increment sal by 5% if balance is less than 1000.

$S \leftarrow \Pi_{name,\ (sal * 1.06)} (\sigma_{bal > 1000}(S))$

$\cup\ \Pi_{name,\ (sal * 1.05)} (\sigma_{bal < 1000}(S))$

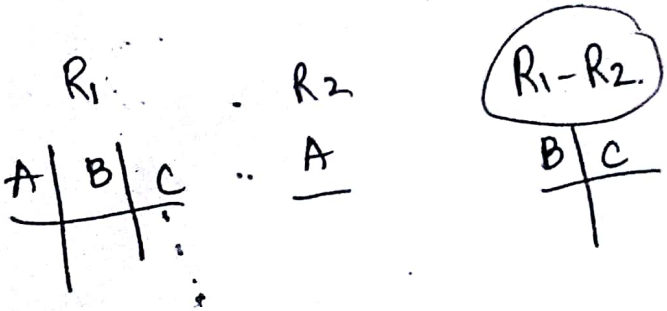
Division operator

$R_1(x) = R_1(Z) \div R_2(y)$

if Result contains all tuples that appear in $R_1(z)$ in combination with every tuple from $R_2(y)$, where $z_2 (x \cup y)$

→ Contains $(R_1 - R_2)$ attributes.

Best suited to Queries that include "for-all".



Student

name	Sub-failed
A	DBMS
A	Java
B	DBMS
C	Java

Subjects

Sub
DBMS
Java

find name of students who failed in all the Subjects.

Π_{name} (Student ÷ Subject)

(A)