Unit-2

Relational Model

The relational model was proposed by E.F. Codd to model data in the form of relations or tables.

After designing the conceptual model of the database using ER Diagram, we need to convert the conceptual model into a relational model which can be implemented using any RDBMS language like SQL etc.

Structure of Relational Database

A relational database consists of a collection of tables, each of which is assigned a unique name.

A row in a table represents a relationship among a set of values.

Basic Structure

Consider the account table. It has three column header - acc-no, branch-name and balance

acc-no	branch-name	balance
A-101	A	500
A-102	B	400
A-103	C	900
A-104	D	700
A-105	E	750
A-106	F	300
A-107	G	350

* Following the terminology of the relational model, we refer to these headers as attributes.

* For each attribute, there is a set of permitted values, called the domain of that attribute. For the attribute branch-name, for example, the domain is the set of all branch names.

* Because tables are essentially relations, we shall use the mathematical terms relation and tuples in terms of place of the terms table and row.

* A tuple variable is a variable that stands for a tuple, in other words, a tuple variable is a variable whose domain is the set of all tuples.

In the account relation, there are seven tuples. Let the tuple variable t refer to the first tuple of the relation. We use the notation t[acc_no] to denote the value of t on the acc-no attribute. Thus t[acc_no] = "A-101" and t[branch-name] = "A".

* Since a relation is a set of tuples, we use the mathematical notation of $t \in R$ to denote that tuple t is in relation R.

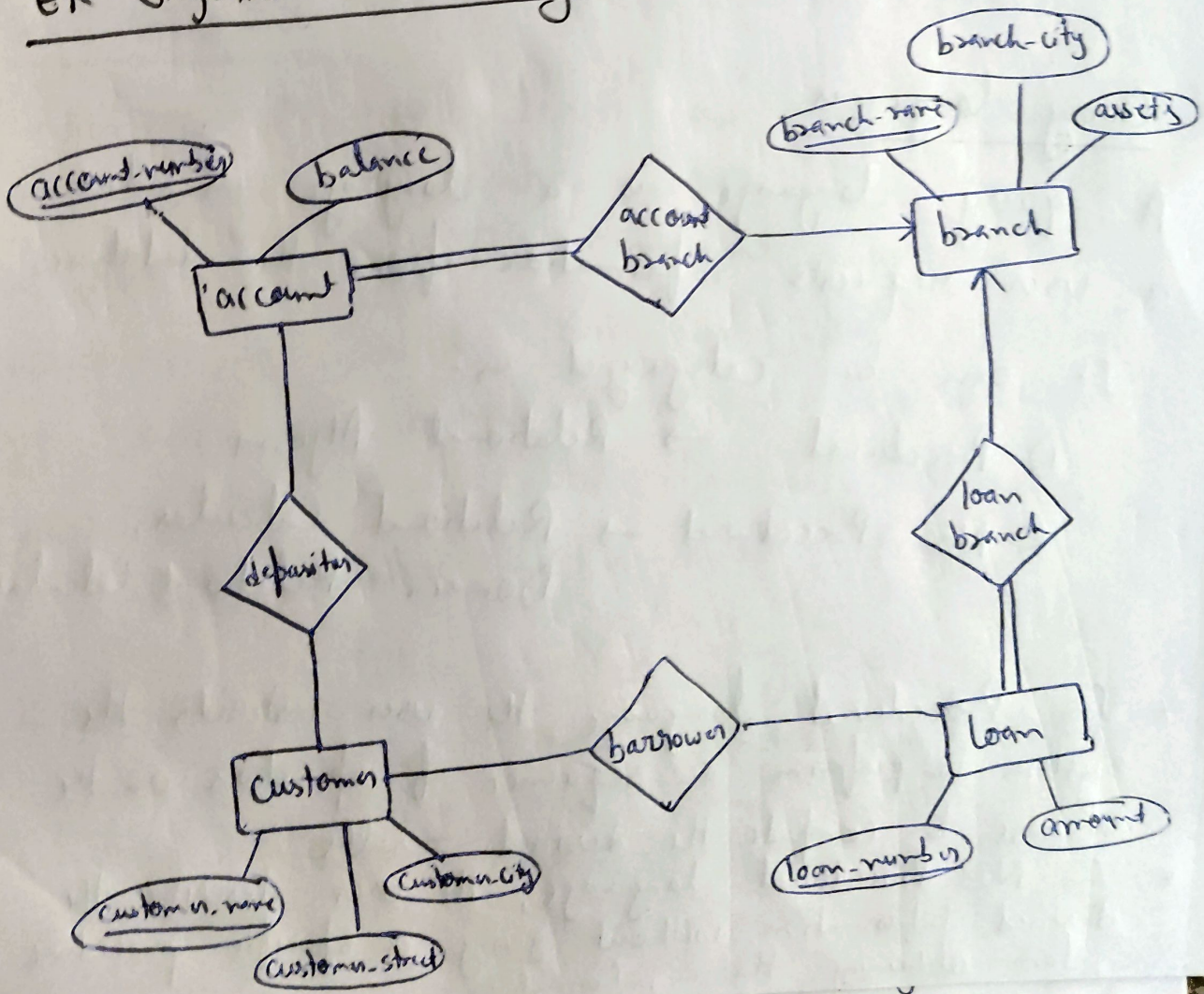
Database Schema

* Database Schema, is a logical design of the database and
* database instance, is a snapshot of the data in the database, at a given instant in time.

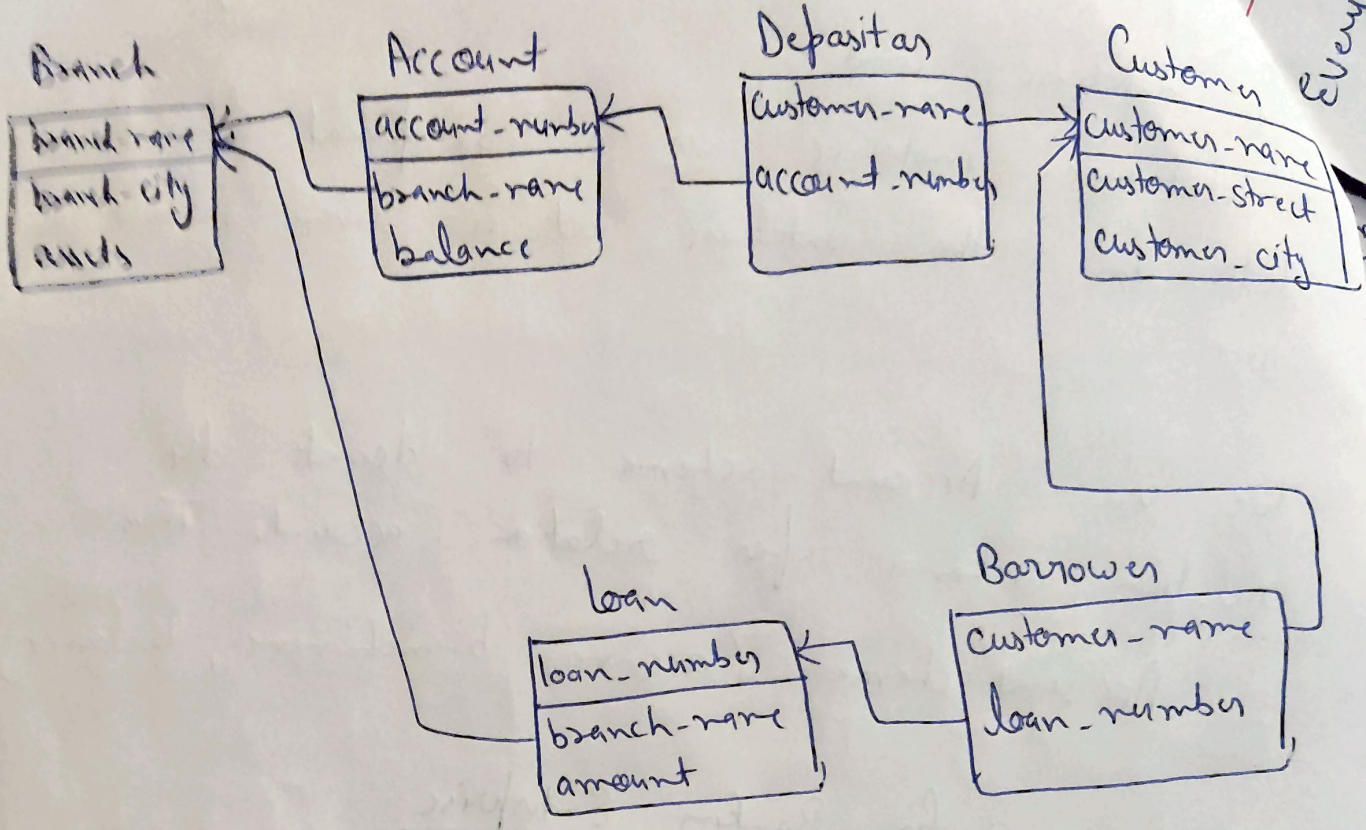
We use Account-schema to denote the relation schema for relation account. Thus,

Account-schema = (acc-no, branch-name, balance)

ER Diagram for Banking Enterprise



Schema Diagram for the Banking Enterprise



Query Languages

A query language is a language in which a user requests information from the database.

It can be categorized as -

1) Procedural → Relational Algebra
2) Non Procedural → Relational Calculus, Domain Relational Calculus

* In Procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired results.
* In Non Procedural language, the user describe the desired information without giving a specific procedure for obtaining that information.

Constraints / Relational Integrity Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called Relational Integrity Constraints.

There are three main integrity constraints -

* Key Constraints
* Domain Constraints
* Referential Integrity Constraints

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called Key for that relation. If there are more than one such minimal subsets, these are called Candidate Keys.

Key Constraints forces that -

* In a relation with a key attribute, no two tuples can have identical values for key attribute.
* A key attribute can not have NULL values.

Key Constraints also referred to as Entity Constraints.

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer., telephone numbers cannot contain a digit outside 0-9.

Referential Integrity Constraints

Referential Integrity Constraints work on the concept of Foreign Keys. A Foreign Key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Advantages of Relational Model

* Simple Model
* flexible
* Secure
* Data Accuracy
* Data Integrity
* operations can be applied easily

ational Model Concepts in DBMS

* **Attribute:** Each Column in a Table. Attributes are the properties which define a relation, eg. - Student-Roll no, Name, etc.

* **Tables:** In the Relational Model, the relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.

* **Tuple:** It is nothing but a single row of a table, which contains a single record.

* **Relation Schema:** A relation schema represents the name of the relation with its attributes.

* **Degree:** The total number of attributes which in the relation is called the degree of the relation.

* **Cardinality:** Total number of rows present in the table.

* **Column** : The column represents the values for a specific attribute.

* **Relation Instance** : Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate keys.

* **Attribute Domain** : Every attribute has some pre-defined value and scope which is known as attribute domain.

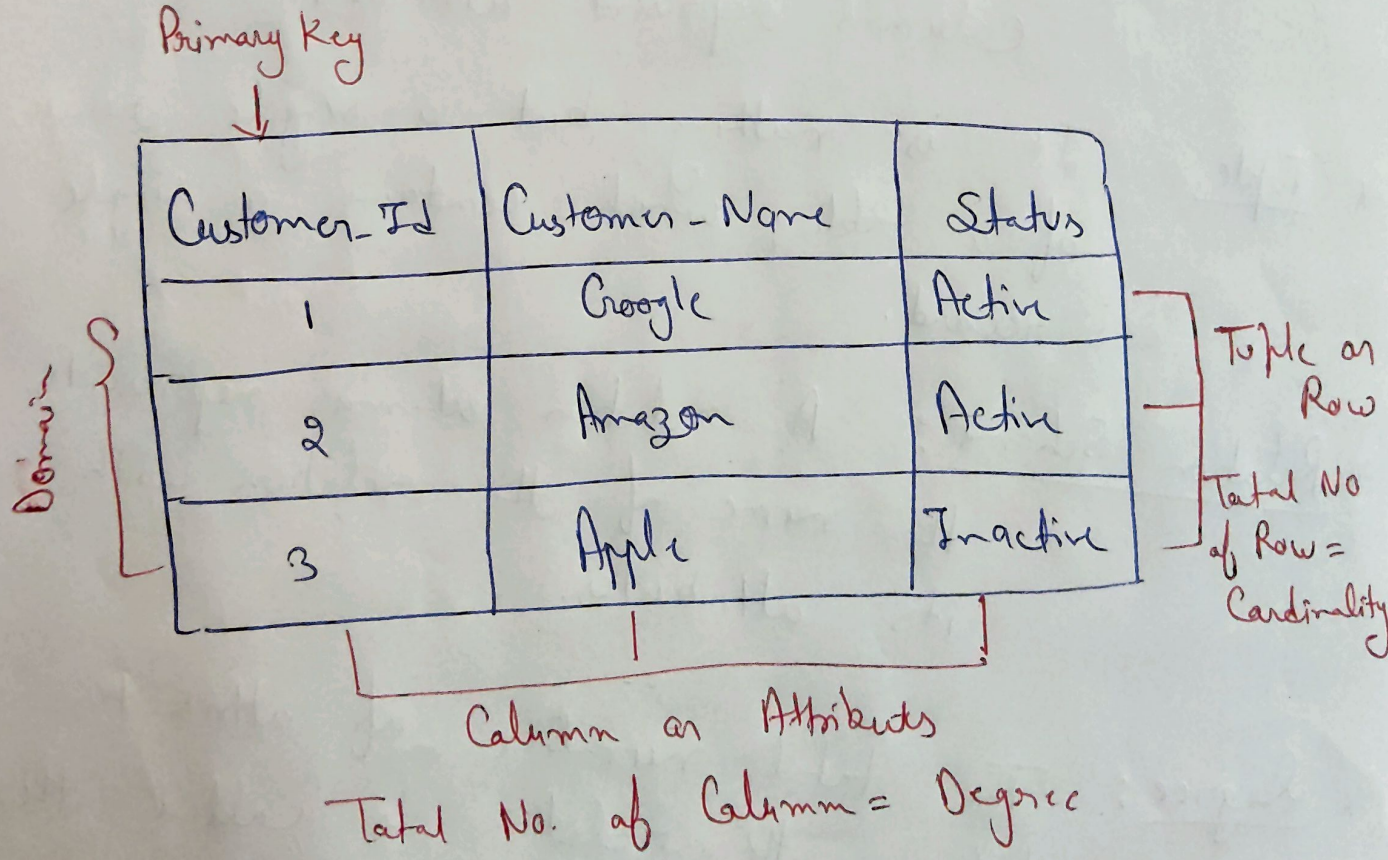


Table also called Relation

Relational Algebra

Relational Algebra is a procedural query language. It mainly provides a theoretical foundation for relational databases and SQL.

The main purpose of using relational Algebra is to define operations that transform one or more relations into an output relation.

Fundamental Operations

1) Selection (σ)
2) Projection (Π)
3) Union ($\cup$)
4) Set Difference ($-$)
5) ~~Set Intersection~~ (Cut)
6) Rename (ρ)
7) Cartesian Product ($\times$)

Additional Operations

1) Set Intersection operation ($\cap$)
2) Natural Join ($\bowtie$)
3) Division Operation ($\div$)
4) Assignment Operation ($\leftarrow$)

Disadvantages of Relational Model

* Not good for large Database
* Relation between tables become difficult some

Characteristics of Relational Model

* Data is represented into rows and columns called as Relation.
* Data is stored in tables having relationship between them called Relational Model.
* Relational model supports the operations like Data Definition, Data Manipulation, Transaction management.
* Each Column have distinct name and they are representing attribute.
* Each row represents the single entity.

Extended Relational Algebra operations

* Generalized Projection
* Aggregate Functions
* Outer Join

Fundamental operators

1) Select Operation (σ)

σ → sigma

It selects tuples that satisfy the given predicate from a relation

Notation $\sigma_p(r)$

where: σ = Stands for selection Predicate
r = relation
p = propositional logic formula which may use connectors like and, or and not.

These terms may use relational operators like $=, \neq, \geq, \leq, >, <$

Note:- The selection operator only selects the required tuple but does not display them. For display, the data projection operation is used.

Sir please see here

loan_number	branch_name	Amount
L-11	Alwar	900
L-14	Jaipur	1500
L-15	Kota	1500
L-16	Delhi	1300
L-17	Alwar	1000
L-23	Jodhpur	2000
L-93	Sikar	500

The loan Relation

loan_schema = (loan_number, branch_name, amount)

Query:- Select those tuples of the loan relation where the branch is "Alwar", we write

$$\sigma_{branch\_name = "Alwar"}(loan)$$

Answer:-
Output

loan_number	branch_name	Amount
L-11	Alwar	900
L-17	Alwar	1000

Generalized

Extended

Query: Find all tuples in which the amount is more than 1300, we write

$$\tau_{amount > 1300}(loan)$$

Query

We can combine several predicates into a larger predicate by using the connectives and ($\wedge$), or ($\vee$) and not ($\neg$).

Query: Find those tuples which have loan of more than 1300 made by Perryridge branch, we write

$$\tau_{branch\_name = \text{"Perryridge"} \wedge amount > 1300}(loan)$$

2) **Projection** : It is used to project required column data from a relation.

Notation : $\Pi_{A_1, A_2, \ldots A_n}(r)$

where $A_1, A_2, \ldots A_n$ are attribute names of relation r.

Note:- Duplicate rows are automatically Eliminated

Query:- List all loan Numbers and the amount of the loan

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

Output :

loan_number	amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Composition of Relational operations

Relational Algebra operations can be composed together into a relational-algebra expression.

Composing relational-algebra operations into relational-algebra expression is just like composing arithmetic operations (such as +, −, *, ÷) into arithmetic expressions.

Query: find those customers name who live in Rajasthan.

If we have table customer_detail.

Customer-Name	State
A	Rajasthan
B	Haryana
C	Rajasthan
D	Delhi

Query ⇒ $\Pi_{\text{customer-name}}\left(\sigma_{\text{State} = \text{'Rajasthan'}}(\text{customer\_detail})\right)$

3) Union operation:

* The union operation contains all the tuples that are either in R or S or both in R & S.
* It eliminates the duplicate tuples.
* It is denoted by ∪.

Notation: $R \cup S$

Note: A Union operation must hold the following condition:

→ R and S must have the attribute of the same number. (No. of Columns)

→ Duplicate tuples are eliminated automatically.

Points to Remember:
① No. of Columns should be same
② Domain of every Columns must be same

Example:-

Rollno	Name
1	A
2	B
3	C

Student

Emp.no	Name
7	E
1	A

Employee

General Union

(Student) ∪ (Employee)

Rollno	Name
1	A
2	B
3	C
7	E

$(\pi_{name}(student) \cup \pi_{name}(Employee))$

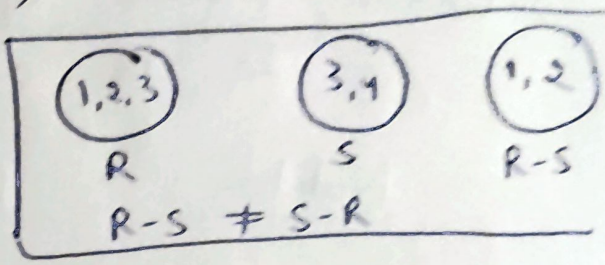
Name
A
B
C
E

Set Difference

The set Difference operations contains all tuples that are in R but not in S.

* It is denoted by minus (-)

Notation: R - S
= R ∩ S'



Example:-

RNo	Name
1	A
2	B
3	C

(Student)

EmpNO	Name
7	E
1	A

Points to Remember
* No. of Columns must be same
* Domain of Every Column must be same

General Difference

R-NO	Name
2	B
3	C

(Student) - (Employee)

Query: Find the name of Person who is student but not a Employee

$(\pi_{name}(Student) - \pi_{name}(Employee))$

Name
B
C

5) Rename operation

It is used to give the name of our choice to a new relation, obtaining from after using any relational algebra operation.

It is an unary operation

Notation : $\rho\ (R, E)$

R = New Relation Name that we want

E = Relational Algebra operation Expression

* It is also used to Rename Attribute.

Example

Student → old Relation

S_id	S_Name	S_Age	S_state	S_course
1				
2				
3				
4				
5				

New Name → Student_Detail in which only S_Age & S_state will Exists

$$\rho\left(\text{Student\_detail}, \pi_{\text{S\_age, S\_state}}(\text{Student})\right)$$

New Attribute Name in Student Detail as S_age = age & S_state = state

$$\rho\left(\text{Student\_detail}(\text{age}, \text{state}), \pi_{\text{S\_age, S\_state}}(\text{Student})\right)$$

Cartesian 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.

Notation: E X D

Example:

R_1

A	B	C
1	2	3
2	1	4

No. of Col (m) = 3
No. of tuples (x) = 2

R_2

C	D	E
3	4	5
2	1	2

No. of Col (n) = 3
No. of tuples (y) = 2

Cross Product $(R_1 \times R_2)$

A	B	C	C	D	E
1	2	3	3	4	5
1	2	3	2	1	2
2	1	4	3	4	5
2	1	4	2	1	2

No. of Col = m + n = 3 + 3 = 6
No. of tuples = x * y = 2 * 2 = 4

Points to Remember

It is required for Join.
for Join, at least one column should be same in both tables.

Additional Relational Algebra Operations

* Set Intersection

The set Intersection operation contains all tuples that are in both R and S. (Means Common tuples)
It is denoted by $\cap$.

Notation: $R \cap S$

Points to Remember

* No. of Columns should be same
* Domain of Every Column must be same.

Example:-

Rollno	Name
1	A
2	B
3	C

Student

Emp-No	Name
7	G
1	A

Employee

$$((\pi_{Rollno}(Student)) \cap (\pi_{EmpNo}(Employee)))$$

Roll No
1

* Division operation (Derived operator)

The Division operator is used for queries which involve the "all".

Denoted by ÷.

It is derived operator, in this we uses other relational operators.

Ex:

S-Name	C-Name
A	B.Tech
B	B.Tech
A	M.Tech
D	B.Tech

S-Detail

C-Name
B.Tech
M.Tech

C-Detail

Query:- Retrieve Name of Student who Enrolled in all Course.

S-Detail ÷ C-Detail

S-Detail (S-Name, C-Name) ÷ C-Detail (C-Name)

To solve this query

1) We required all Student Name
   We assume that Every Student is Enrolled in Each Course. (S-Detail × C-Detail)
2) Then Applying to S-Detail we will get

2) Then we will minus Actual table from Cross Product, then we will get those ds who are not Enrolled in at least One Course.

$$\left(\pi\left(S\text{-}Detail\ (S\text{-}Name)\right) \times \pi\left(C\text{-}Detail\ (C\text{-}Name)\right)\right)$$

S-Name
A
B
D

X

C-Name
B-Tech
M-Tech

=

S-Name	C-Name
A	B-Tech
A	M-Tech
B	B-Tech
B	M-Tech
D	B-Tech
D	M-Tech

According to this we find that Every student is Enrolled in Every Course.

Now if we minus the Actual S-Detail table from it then we will get.

$$\left(\left(\pi_{S\text{-}Name}(S\text{-}Detail) \times \pi_{C\text{-}Name}(C\text{-}Detail)\right) - \pi_{S\text{-}Name, C\text{-}Name}(S\text{-}Detail)\right)$$

S-Name	C-Name
A	B-Tech
A	M-Tech
B	B-Tech
B	M-Tech
D	B-Tech
D	M-Tech

−

S-Name	C-Name
A	B-Tech
B	B-Tech
A	M-Tech
D	B-Tech

=

S-Name	C-Name
B	M-Tech
D	M-Tech

from this we get those students who are not Enrolled in At least One Course

Division operation (Derived operator)

Division operator is used for queries which involve the "all".

Denoted by ÷

Ex.

S-Name	C-Name
A	B.Tech
B	M.Tech
A	M.Tech
D	B.Tech

S-Detail

C-Name
B.Tech
M.Tech

C-Detail

Query: Retrieve Name of Student who Enrolled in all Courses.

S_Detail ÷ C_Detail

S_Detail(S-Name, C-Name) ÷ C_Detail(C-Name)

Result

S-Name
A

* Natural Join / operation

Join operation [Cross Product + Some Condition]

Join operation combines the Relation R1 and R2 w.r. respect to a condition. It is denoted by ⋈.

The different types of Join operation are as-

- → Theta Join
- → Natural Join
- → Outer Join
  - ↳ Left Outer Join
  - ↳ Right Outer Join
  - ↳ Full Outer Join

* Natural Join operation

A natural Join is the set of tuples of all combinations in R and S that are equal on their common attribute names.

It is denoted by ⋈

Ex.

R_NO	S_Name	Age
1	A	20
2	B	18
3	C	15
4	D	19

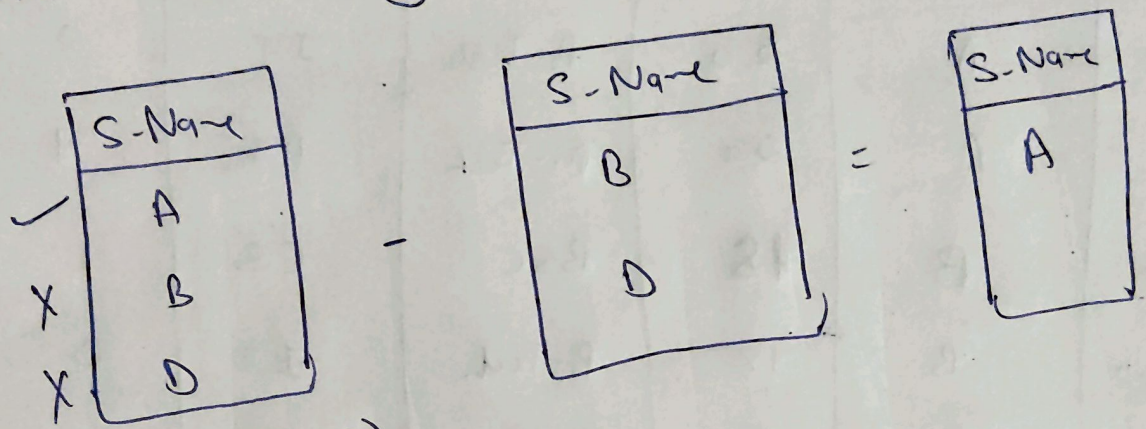
Student

Course	Branch	R_NO
BSC	CS	1
B.Tech	IT	2
M.Tech	AI	4

Branch

we get disqualified Students.

So Now if we will minus Disqualified Students from all Students then we will get Qualified Student, that mean we get those Name of Students who are Enrolled in Each and Every Course



$\pi_{S\text{-}Name}(S\_Detail) - \left(\pi_{S\text{-}Name}\left(\left(\pi_{S\text{-}Name}(S\_Detail) \times \pi_{C\text{-}Name}(C\_Detail)\right) - \pi_{S\text{-}Name, C\text{-}Name}(S\_Detail)\right)\right)$

Query :- find S_Name who is having a ~~course~~ Bra...

π_{S_Name} (σ student.rollno = branch.rollno (Student X

Student			Branch		
R_NO	S_Name	Age	Course	Branch	R_NO
1	A	20	BSC	CS	1
1	A	20	B.Tech	IT	2
1	A	20	M.Tech	AI	4
2	B	18	BSC	CS	1
2	B	18	B.Tech	IT	2
2	B	18	M.Tech	AI	4
3	C	15	BSC	CS	1
3	C	15	B.Tech	IT	2
3	C	15	M.Tech	AI	4
4	D	19	BSC	CS	1
4	D	19	B.Tech	IT	2
4	D	19	M.Tech	AI	4

(Rows marked: ✓, X, X, X, ✓, X, X, X, X, X, X, ✓)

Output =

S_Name
A
B
D

Actual Query

π ~~Emp_name~~ (~~S_Na~~

π_{S_name} (Student ⋈ Branch)

Join

we join condition then equi Join

Join (θ)

- we join R1 and R2, other than the equal condition then it is called theta Join / non-equi Join.

Notation:- $R1 \bowtie_{\theta} R2$

Example:

R1

RegNo	Branch	Section
1	CS	A
2	EC	B
3	CE	A
4	IT	B
5	IT	A

R2

Name	Reg NO
abc	2
xyz	4

R1 ⋈ R2 with Condition $R1_{regNo} > R2_{regNo}$

$\Pi_{regNo, Branch, Name}\left(\sigma_{R1.regNo > R2.regNo}(R1 \times R2)\right)$

RegNo	Branch	Section	Name	Reg NO
3	CE	A	abc	2
4	IT	B	abc	2
5	IT	A	abc	2
5	IT	A	xyz	4

Outer Join

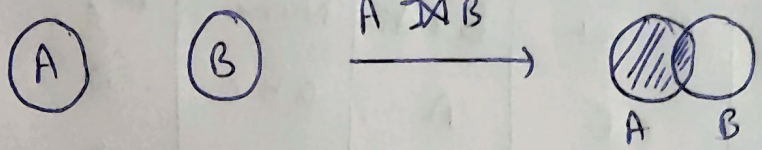
It is an extension of natural Join to deal missing values of relation.

Outer Join is of three types -

1) Left Outer Join $R1 ⟕ R2$
2) Right Outer Join $R1 ⟖ R2$
3) Full Outer Join $R1 ⟗ R2$

1) Left Outer Join

In the left Outer Join, operation allows keeping all tuple in the left relation. However if there is no matching tuple found in right relation, then the attributes of right relation in the join are filled with null values.

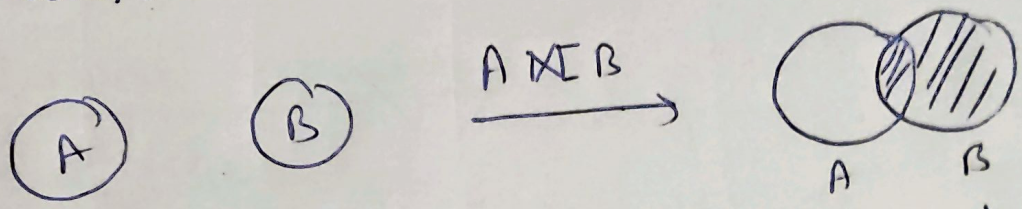


All rows from left table

R NO	S.Name	Age	Course	Branch
1	A	20	BSC	CS
2	B	18	BTech	IT
4	D	19	M.Tech	AI
3	C	15	-	-

Right Outer Join

The right Outer Join, operation allows keeping all tuple in the right relation, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the Join result are filled with null values.



All rows from Right Tables

R.NO	S.Name	Age
1	A	20
2	B	18
3	C	15
4	D	19

Student

Course	Branch	R.NO
BSC	CS	1
B.Tech	IT	2
M.Tech	AI	4
B.Tech	EC	5

Branch

R.NO	S.Name	Age	Course	Branch
1	A	20	BSC	CS
2	B	18	B.Tech	IT
4	C	15	M.Tech	AI
5	-	-	B.Tech	EC

Student ⟖ Branch

3) Full Outer Join

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

$$A \text{ FOJ } B = (A \bowtie B) \cup (A \bowtie B)$$

Operators

metic operators

+ - Add
- - Subtract
* - Multiply
/ - Divide
% - Modula

SQL Bitwise operators

& - Bitwise AND
| - Bitwise OR
^ - Bitwise Exclusive OR

Comparison Operators

= - Equal to
> - Greater than
< - Less than
>= - Greater than or Equal to
<= - less than or Equal to
<> - Not Equal to

SQL logical operators

ALL - True if all of the Subquery values meet the condition

AND - True if all the conditions separated by AND is true

ANY - True if any of the Subquery values meet the condition

Between - True if the operand is within the range of comparisons

Exists - True if the Subquery returns one or more records

IN - True if the operand is equal to one of a list of expressions

Like - True if the operand matches a pattern

NOT - Display a record if the condition is not true

OR - True if any of the conditions separated by OR is true

SOME - True if any of the Subquery values meet the condition.

Aggregate Functions in SQL

In database Management, an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

1) Count ()
2) Sum ()
3) Avg ()
4) Min ()
5) Max ()

Eg.

ID	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	NULL

1) **Count ()** :-

Count (*) - Returns total no. of records i.e. 6

Count (salary) - Return number of Non Null Values over the Column salary i.e. 5

Count (Distinct Salary) - Return number of Non Null Values over the column i.e. 4

2) Sum ()

→ Sum (Salary) - Sum of all Non Null Values of Column salary - 310

→ Sum (Distinct Salary) - Sum of all distinct Non-Null values, i.e. 250

3) Avg ()

→ Avg (Salary) - Sum (Salary) / Count (Salary)
= 310 / 5

→ Avg (Distinct Salary) - Sum (Distinct salary) / Count (Distinct Salary)
= 250 / 4

4) Min () → gives minimum value except NULL

5) Max() → gives maximum value

Clauses

1) Group by
2) Having
3) Order by

1) Group by

* SQL Group by Statement is used to arrange identical data into groups. The Group by Statement is used with the SQL Select Statement.
* Group by statement is used with aggregation function.

Query ①:- Select Salary from Employee group by Salary;

o/p Query ①

Salary
80
40
60
70

o/p Query ②

Salary	count(*)
80	1
70	1
60	2
40	1

Query ②:- Write a query to display all the Salary along with no of Employee having that Salary.

=> Select salary, count(*) from Employee group by Salary;

2) Having

* Having clause is used to specify a condition for a group or an aggregate.
* Having is used in a Group by clause.

Query:-

```
Select salary, count(*) from Employee
        group by salary having count(*)>1;
```

o/p

Salary	Count (*)
60	2

3) Order by

* The Order by clause sorts the result-set in ascending or descending order.
* It sorts the records in ascending order

ASC:- It is used to sort the result set in ascending order by expression

DESC:- It sort the result set in descending order by expression.

select * from Employee order by Name;

Ans:- select * from Employee order by
ID desc;

Set operation

SQL Set operation is used to combine the two or more SQL statements.

Types of Set operation

1) Union
2) UnionAll
3) Intersect
4) Minus

1) Union

* The SQL Union operation is used to combine the result of two or more SQL select queries.
* In the Union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
* The Union operation eliminates the duplicate rows from its result set.

Syntax:-

```
select col-name from table1
Union
select col-name from table2;
```

Eg:

first

ID	Name
1	A
2	B
3	C

Second

ID	Name
3	C
4	D
5	E

⇒ Select * from first
Union
Select * from Second;

ID	Name
1	A
2	B
3	C
4	D
5	E

2) Union All

It is equal to the Union operation. It returns the set without removing duplication and sorting the data.

Syntax:-

Select col-name from table1
Union All
Select col-name from table2;

from first

Union All

select * from Second;

ID	Name
1	A
2	B
3	C
3	C
4	D
5	E

3) Intersect

* It is used to combine two Select statements. The intersect operation returns the common rows from both the select statements.
* In the intersect operation, the number of datatype and columns must be the same.
* It has no duplicates and it arranges the data in ascending order by default.

Syntax:-

select col-name from table1

Intersect

select col-name from table2;

Eg: Select * from First
Intersect
select * from Second;

ID	Name
3	C

4) Minus

* It combines the result of two Select statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
* It has no duplicates and data arranged in ascending order by default.

Syntax: select column from table
minus
select column from table;

Eg: select * from first
minus
select * from second;

ID	Name
1	A
2	B

Same

SQL, except
used by the
et returned by

Clause

SQL, Except returns those tuples that are returned by the first Select operation and not returned by the second Select operation.

Eg.

first

ID	Name	Course
1	Rahan	DBMS
2	Kevin	OS
3	Mansi	DBMS
4	Mansi	ADA
5	Rekha	ADA
6	Megha	OS

Second

ID	Name	Course
1	Kevin	TOC
2	Sita	IP
3	Manik	SE
4	Rekha	DSA

```
Select Name from first
    Except
Select Name from Second;
```

O/P

Name
Rahan
Mansi
Megha

Note:- To Retain duplicate, we must explicitly write EXCEPTALL instead of Except.

ggers in SQL

A trigger in SQL is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database.

Triggers are used to protect the data integrity in the database.

Syntax:-

```
create Trigger  Trigger_Name
[ Before | After ] [ Insert | Update | Delete ]
On [ Table_Name ]
[ For Each Row | For Each Column ]
As
set of SQL Statement
```

Query:- Xxxxxx

Example:-

To understand the concept of trigger in SQL, first, we have to create the table on which trigger is to be executed.

The following query creates the Student_Trigger table in the SQL database:

```
create table Student_Trigger
(
  Student_Rollno  INT,
  Student_FirstName  varchar (100),
  Student_EnglishMarks  INT,
  Student_PhysicsMarks  INT,
  Student_ChemistryMarks  INT,
  Student_MathsMarks  INT,
  Student_TotalMarks  INT,
  Student_Percentage);
```

The following query fires a trigger before the insertion of the student record in the table

```
create  Trigger  Student_Table_Marks
Before  Insert
on
Student_Trigger
for  Each Row
SET  new. Student_Total Marks =
new. student_EnglishMarks + new. Student_PhysicsMarks +
new. Student_ChemistryMarks + new. student_MathsMarks,

new. Student_Percentage = (new. Student_Total Marks/400)*100;
```

The following query inserts the record into Student_Trigger table:

⇒ Insert into Stud

t into Student_Trigger (Student_RollNo, Student_firstName, Student_EnglishMarks, Student_PhysicsMarks, Student_ChemistryMarks, Student_MathsMarks, Student_TotalMarks, Student_Percentage) values (101, "A", 80, 70, 60, 80, 0, 0);

To check the output of the above INSERT statement, you have to type the following Select statement :-

```
select * from Student_Trigger;
```

O/P

						Total	Percentage
					Percenta	Percentage	
101	A	80	70	60	80		

Active Databases

An active database is a database consisting of a set of triggers. These triggers databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such Databases, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database is activated or not.

If the trigger is active then DBMS

executes the condition part and then executes the action part only if the specified condition is evaluated to true.

It is possible to activate more than one trigger within a single statement. In such situation, DBMS processes each of the trigger randomly.

The DBMS execution of an action part of a trigger may either activate other triggers, or the same trigger that initialized this action. Such types of triggers that activates itself is called as "recursive trigger."

Feature of Active Database

⇒ It passes all the concepts of a conventional database, i.e. data modelling facilities, query lang. etc.

⇒ It supports all the functions of a traditional database like data definition, data manipulation, storage management etc.

⇒ It detects event occurrence.

⇒ It must be able to evaluate conditions and to execute actions.

⇒

Relational Calculus

It is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results.

The relational calculus tells what to do but never explains how to do.

There are two types of Quantifiers.

1) Universal Quantifiers: (∀) for all

The Universal quantifier denoted by ∀ is read as for all which means that in a given set of tuples exactly all tuples satisfy a given condition.

2) Existential Quantifiers: (∃) There Exist

The existential quantifier denoted by ∃, is read for all which means that in a given set of tuples there is at least one occurrences whose value satisfy a given condition.

Note: Quantifier is used to quantify the variable of predicates.

$\exists_T$ => for some occurence of T

$\forall_T$ => for every occurence of T

Types of Relational Calculus

1) Tuple Relational Calculus (TRC)
2) Domain Relational Calculus (DRC)

1) Tuple Relational Calculus (TRC)

It is a non-procedural query language which is based on finding a number of tuple variables also known as range variable for which predicate holds true. It describe the desired information without giving a specific procedure for obtaining that information. The tuple relational calculus is specified to select the tuples in a relation. The result of the relation can have one or more tuples.

Notation:

$$\{ T \mid P(T) \}$$

or

$$\{ T \mid \text{Condition}(T) \}$$

where T = resulting tuple

P(T) = Condition used to fetch (T)

Query:- find all student, who having age >21

$$\{ t \mid \underbrace{t \in \text{student}}_{t \in R} \wedge \underbrace{t[\text{age}] > 21}_{t[A]} \}$$

or

$$\{ t \mid \text{student}(t) \wedge t[\text{age}] > 21 \}$$

or is basically $t.age > 21$ }

Ex:- find all student, who having age >21 & <18

$\{t \mid t \in \text{student} \wedge t[age] > 21 \wedge t[age] < 18\}$

Query: find all student Name, Age, Address, having age >21.

$\{t.name, t.age, t.address \mid \text{student}(t) \wedge t[age] > 21\}$

Quantifiers : which Quantify the Tuple Variable 'T'.

$\exists_T \rightarrow$ for some occurences of T

$\forall_T \rightarrow$ for every occurences of T

Query:- $\{t \mid \exists_T \in \text{Employee}\ (s[salary] = t[salary] \wedge t[salary] > 7000\}$

↳ if yes → True otherwise false

$\{t \mid \forall_T \in \text{Employee}\ (s[salary] = t[salary] \wedge t[salary] > 7000\}$

↳ if yes → True otherwise false

2) Domain Relational Calculus (DRC)

In domain relational calculus, filtring variables uses the domain of attributes. Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ($\wedge$) and, ($\vee$) or, and $\neg$ (not). It uses ($\exists$) and ($\forall$) to bind the variable.

Notation:-

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

where

a_1, a_2 are attributes

P stands for formula built by inner attributes.

Query:- find all student having age > 21

S_ID Name Age class

$$\{ S, N, A, C \mid \langle S, N, A, C \rangle \in \text{Student} \wedge A > 21 \}$$

By Using $\exists$ Quantifier:-

Query:- Name of student having age > 21

* $\{ N \mid \langle \exists S, A, C \rangle \in \text{Student} \wedge A > 21 \}$
* $\{ N \mid (\exists A)(\text{Student}(S, N, A, C) \wedge A > 21 \}$

Safe Expressions

A safe Expression is one that is guaranteed to yield a finite number of tuples as its results.

Expressive Power of Languages

When the domain relational calculus is restricted to safe expressions, it is equivalent in expressive to the tuple relational calculus restricted to safe expressions.

The restricted tuple relational calculus is equivalent to the relational algebra, all three of the following are equivalent:

* The Basic relational algebra (without the extended relational algebra operations)
* The tuple relational calculus restricted to safe expressions
* The domain relational calculus restricted to safe expressions.

...eries in SQL

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query.

There are mainly two types of nested queries-

1) Independent Nested queries

In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query.

Various operator like IN, NOT, ANY, ALL etc. are used in writing independent nested queries.

Student

S_ID	S_Name	S_address	S_Phone	S_Age
S1	A	Delhi	123	18
S2	B	Noida	234	18
S3	C	Jaipur	368	20
S4	D	Kota	193	18

Course

C_ID	C_Name
C1	DSA
C2	C
C3	DBMS

Student_Course

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

Query:- find out S_ID who are enrolled in C_Name DSA or DBMS.

Step1:- finding C_ID for C_Name = DSA or DBMS

Select C_ID from Course where C_Name = "DSA" or C_Name = "DBMS".

Step2:- finding S_ID using C_ID of step1

select S_ID from Student_Course where C_ID IN (Select C_ID from Course where C_Name = "DSA" or C_Name = "DBMS");

The inner query will return a set with numbers C1 and C3 and outer query will return those S_ID's for which C_ID is equal to any numbers of set (C1 and C3).

All sections S1, S2, S4.

S-ID
S1
S2
S4

Query:- find out Names of students who have either enrolled in 'DSA' or 'DBMS'.

=> select S_Name from Student where S-ID in
(Select S-ID from Student_Course where C-ID in
(Select C-ID from Course where C_Name = 'DSA'
or C_Name = 'DBMS'));

Subquery (Synchronized Query)

It is a subquery that uses values from outer query.

O_Q (I_Q)

↳

Note:- In co-related nested queries, the o/p of inner query depends on the row which is being currently executed in outer query

* It is a Tab-Down Approach

Emp

Eid	Name	address
1	A	Jaipur
2	B	Delhi
3	C	Alwar
4	D	Kota
5	E	Ajmer
6	F	Noida

Dept

Did	Dname	Eid
D1	CS	1
D2	ME	2
D3	EE	3
D4	CE	4

Query:- find all Employee detail who work in a department

=> select * from Emp where exists (select * from Dept where Dept. Eid = Emp. Eid)

o/p

Eid	Name	address
1	A	Jaipur
2	B	Delhi
3	C	Alwar
4	D	Kota

SQL

SQL is known as the Structured Query language. It is the language that we use to perform operations and transactions on the databases.

When we talk about industry-level applications we need properly connected systems which could draw data from the database and present to the users. In such cases, Embedded SQL comes.

We Embed SQL queries into high-level languages such that they can easily perform the logic part of our analysis.

Some of the prominent examples of languages with which we embed SQL are as -

- → C++
- → JAVA
- → Python, etc.

Embedded SQL gives us the freedom to use databases as and when required.

With the help of the Embedding of queries, we can easily use the database without creating any bulky code. With the Embedded SQL, we can create API's which can easily fetch and feed data as and when required.

For using Embedded SQL, we need tools in each high-level language. In some cases, we have inbuilt libraries which provide us with the basic building blocks.

While in some cases we need to import or use some packages to perform some desired tasks.

Advantages of Embedded SQL

* Helps to access databases from anywhere.
* Allows integrating authentication service for large scale application.
* Provides extra security to database transaction.
* Avoids logical errors while performing transactions on our database.
* Makes it easy to integrate the frontend and the backend of our application.

SQL

SQL is the process that we follow programming SQL queries in such a way the queries are built dynamically with the application operations.

It helps us to manage big industrial applications and manage the transactions without any added overhead.

With dynamic SQL we are free to create flexible SQL queries and names of the variables or any other parameters are passed when the application runs.

We need to use Dynamic SQL for the following use cases -

=) When we need to run dynamic queries on our database, mainly DML queries.

=) When we need to access an object which is not in existance during the compile time.

=) Whenever we need to optimize the run time of our queries.

=) When we need to instantiate the created logic blocks.

=) When we need to perform operations on application fed data using Invoker rights.