UNIT - II

Relational Model: Introduction to Relational Model, Concepts of Domain, Attribute, Tuple, Relation, Importance of Null Values, Integrity Constraints and Their Importance, Introduction to Data Types, Use of Basic SQL to Create and Manipulate Tables with Integrity Constraints. Translation of E-R diagrams to Relations

Relational Algebra: Relational Algebra Operators, with Examples.

EF Codd Rules

EF Codd Rules

- **Rule 0:** The *foundation rule*:

  This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

- **Rule 1:** The *information rule*:

  All information(including metadata) is to be represented as stored data in cells of tables.

- **Rule 2:** The *guaranteed access rule*: Each unique piece of data(atomic value) should be accessible by : **Table Name + Primary Key(Row) + Attribute(column)**.

EF Codd Rules

- **Rule 3:** *Systematic treatment of null values*:

  Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever. Expression on NULL must give null.

- **Rule 4:** *Active Online Catlog*:

  Database dictionary(catalog) is the structure description of the complete **Database** and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.

EF Codd Rules

- **Rule 5:** The *comprehensive data sublanguage rule*:

  One well structured language must be there to provide all manners of access to the data stored in the database. Example: **SQL**, etc. If the database allows access to the data without the use of this language, then that is a violation.

- Data definition.
- View definition.
- Data manipulation (interactive and by program).
- Integrity constraints.
- Authorization.
- Transaction boundaries (begin, commit and rollback).

- **Rule 6:** The *view updating rule*:

  All views that are theoretically updatable are also updatable by the system.

EF Codd Rules

- **Rule 7:** Relational level Operations Rule / *Possible for high-level insert, update, and delete*:

  There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.

- **Rule 8:** *Physical data independence*:

  The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not effect the application..

EF Codd Rules

- **Rule 9:** *Logical data independence*:

  Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit un impairment are made to the base tables. If there is change in the logical structure(table structures) of the database the user view of data should not change.

- **Rule 10:** *Integrity independence*:

  Integrity constraints specific to a particular relational data base must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

EF Codd Rules

- **Rule 11:** *Distribution independence*:

  The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

  A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of **distributed database**.

- **Rule 12:** N*on subversion rule*:

  If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of looking or encryption

DBMS

Relational Model

Relational Model in DBMS

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

After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDBMS languages like Oracle SQL, MySQL etc.

What is Relational Model?

- Relational model can represent as a table with columns and rows. Each row is known as a tuple. Each table of the column has a name or attribute.

- Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in Table 1.

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

- **Attribute:** Attributes are the properties that define a relation.
  e.g.; **ROLL_NO**, **NAME**
- **Domain:** It contains a set of atomic values that an attribute can take.

- **Relation Schema:** A relation schema represents name of the relation with its attributes. e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE and AGE) is relation schema for STUDENT. If a schema has more than 1 relation, it is called Relational Schema. A relational schema contains the name of the relation and name of all columns or attributes.

- **Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1 RAM DELHI 9455123451 18

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI		18

- **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of STUDENT at a particular time. It can change whenever there is insertion, deletion or updation in the database. In the relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

- **Degree:** The number of attributes in the relation is known as degree of the relation. The **STUDENT** relation defined above has degree 5. from relation STUDENT.

- **Cardinality:** The number of tuples in a relation is known as cardinality. The **STUDENT** relation defined above has cardinality 4.

- **Column:** Column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted

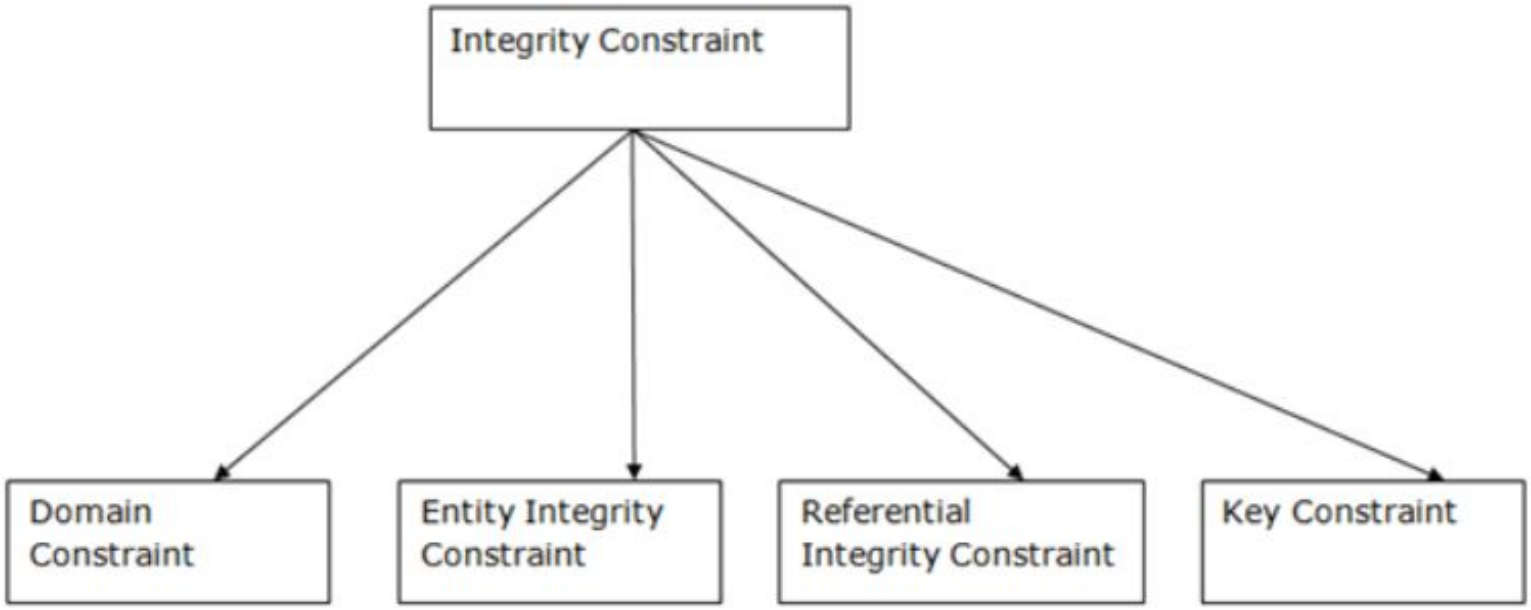
ROLL_NO
1
2
3
4

- **NULL Values:** The value which is not known or unavailable is called NULL value. It is represented by blank space. e.g.; PHONE of STUDENT having ROLL_NO 4 is NULL.

Properties of Relations

- Name of the relation is distinct from all other relations.
- Each relation cell contains exactly one atomic (single) value
- Each attribute contains a distinct name
- Attribute domain has no significance
- Tuple has no duplicate value
- Order of tuple can have a different sequence

Types Integrity Constraints



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.

- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

  Example :

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

Primary Key

A Primary key is the column or columns that contain values that uniquely identify each row in a table.

A database table must have a primary key to insert, update, restore, or delete data from a database table.

2. Entity integrity constraints

- **The entity integrity constraint states that primary key value can't be null.**
- **This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.**
- **A table can contain a null value other than the primary key field.**

Example :

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can’t contain a NULL value

Foreign Key

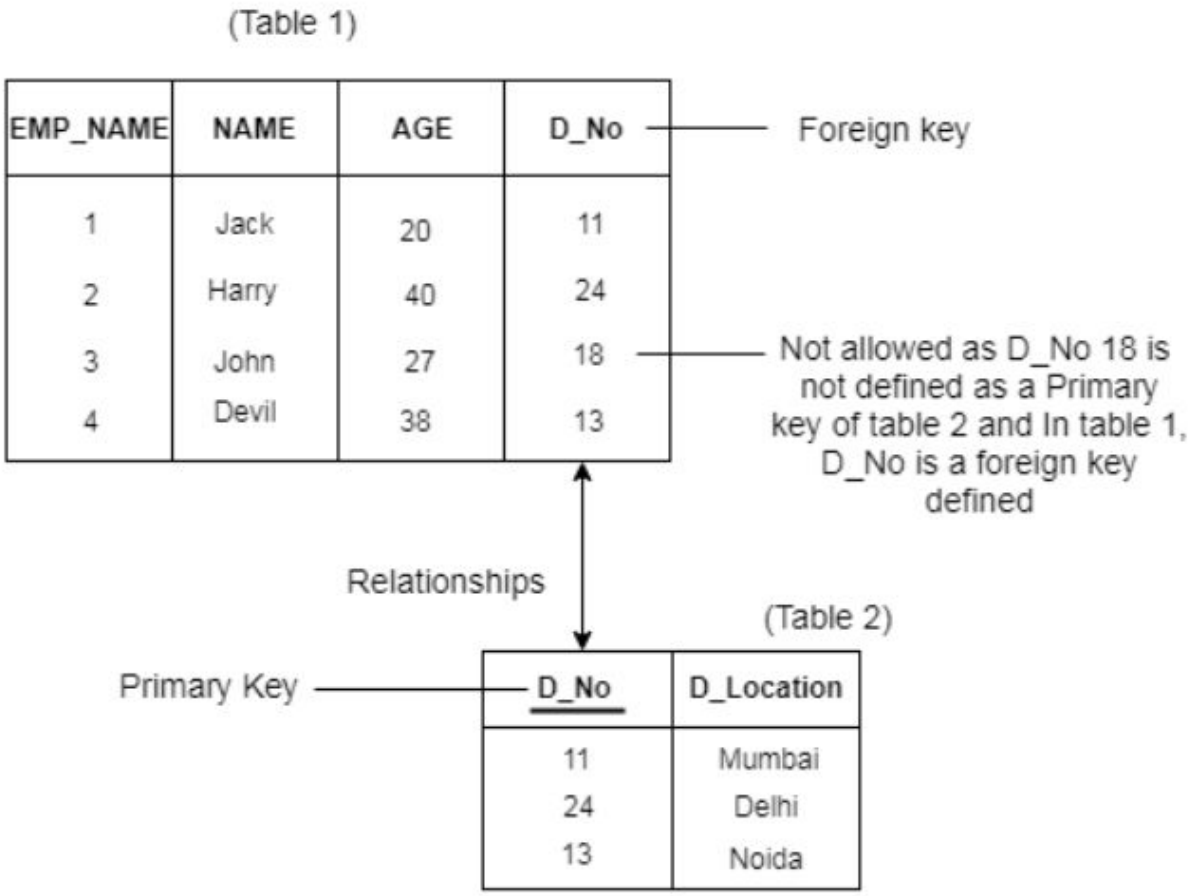
- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

3. Referential integrity constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be available in Table 2.

Example :



4. Key constraints

- Keys are the entity set that is used to identify an entity within its entity set uniquely.

- An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique in the relational table.

Example :

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

Not allowed. Because all row must be unique

Relational Model Example

- **Relational key:** In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

Example: STUDENT Relation

NAME	ROLL_NO	PHONE_NO	ADDRESS	AGE
Ram	14795	7305758992	Noida	24
Shyam	12839	9026288936	Delhi	35
Laxman	33289	8583287182	Gurugram	20
Mahesh	27857	7086819134	Ghaziabad	27
Ganesh	17282	9028 9i3988	Delhi	40

In the given table, NAME, ROLL_NO, PHONE_NO, ADDRESS, and AGE are the attributes.
The instance of schema STUDENT has 5 tuples.
t3 = <Laxman, 33289, 8583287182, Gurugram, 20>

Constraints in Relational Model

While designing Relational Model, we define some conditions which must hold for data present in database are called Constraints.

These constraints are checked before performing any operation (insertion, deletion and updation) in database. If there is a violation in any of constrains, operation will fail.

- **Domain Constraints:** These are attribute level constraints. An attribute can only take values which lie inside the domain range. e.g,; If a constrains AGE>0 is applied on STUDENT relation, inserting negative value of AGE will result in failure.

Constraints in Relational Model

- **Key Integrity:** Every relation in the database should have atleast one set of attributes which defines a tuple uniquely. Those set of attributes is called key. e.g.; ROLL_NO in STUDENT is a key. No two students can have same roll number. So a key has two properties:

  - It should be unique for all tuples.
  - It can’t have NULL values.

- **Referential Integrity:** When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. Let us suppose we have 2 relations

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

BRANCH_CODE of STUDENT can only take the values which are present in BRANCH_CODE of BRANCH which is called referential integrity constraint.

The relation which is referencing to other relation is called REFERENCING RELATION (STUDENT in this case) and the relation to which other relations refer is called REFERENCED RELATION (BRANCH in this case).

ANOMALIES

- An anomaly is an irregularity, or something which deviates from the expected or normal state. When designing databases, we identify three types of anomalies: Insert, Update and Delete.

- **Insertion Anomaly in Referencing Relation:**

  We can't insert a row in REFERENCING RELATION if referencing attribute's value is not present in referenced attribute value. e.g.; Insertion of a student with BRANCH_CODE 'ME' in STUDENT relation will result in error because 'ME' is not present in BRANCH_CODE of BRANCH.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

Inserting BRANCH_CODE 'ME' in STUDENT relation will result in error because 'ME' is not present in BRANCH_CODE of BRANCH.

5	KRISHNA	CHENNAI	7768545634	21	ME

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

ANOMALIES

- **Deletion/ Updation Anomaly in Referenced Relation:**

We can't delete or update a row from REFERENCED RELATION if value of REFERENCED ATTRIBUTE is used in value of REFERENCING ATTRIBUTE.

e.g; if we try to delete tuple from BRANCH having BRANCH_CODE 'CS', it will result in error because 'CS' is referenced by BRANCH_CODE of STUDENT, but if we try to delete the row from BRANCH with BRANCH_CODE CV, it will be deleted as the value is not been used by referencing relation.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

Deleting tuple from BRANCH having BRANCH_CODE 'CS', it will result in error because 'CS' is referenced by BRANCH_CODE of STUDENT

CS	COMPUTER SCIENCE

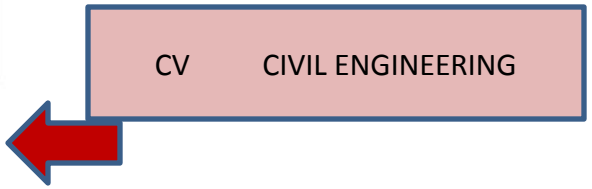
STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

delete the row from BRANCH with BRANCH_CODE CV, it will be deleted as the value is not been used by referencing relation.



ANOMALIES

- **Deletion/ Updation Anomaly in Referenced Relation:**

It can be handled by following method:

- **ON DELETE CASCADE:** It will delete the tuples from REFERENCING RELATION if value used by REFERENCING ATTRIBUTE is deleted from REFERENCED RELATION. e.g;, if we delete a row from BRANCH with BRANCH_CODE 'CS', the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be deleted.

- ON DELETE CASCADE constraint is used in MySQL to delete the rows from the child table automatically, when the rows from the parent table are deleted.

ON DELETE CASCADEON DELETE CASCADE ensures that when a record in the parent table is deleted, the corresponding records in the child table automatically get deleted.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

REFERENCING RELATION (STUDENT)

Rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be deleted.

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

REFERENCED RELATION (BRANCH).

If we delete a row from BRANCH with BRANCH_CODE ‘CS’,

ANOMALIES

Updation Anomaly in Referenced Relation:

- **ON UPDATE CASCADE:** It will update the REFERENCING ATTRIBUTE in REFERENCING RELATION if attribute value used by REFERENCING ATTRIBUTE is updated in REFERENCED RELATION.

  e.g;, if we update a row from BRANCH with BRANCH_CODE ‘CS’ to ‘CSE’, the rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be updated with BRANCH_CODE ‘CSE’.

ON UPDATE CASCADE : This type of cascade when a primary key in the parent table is updated. In such cases, the corresponding foreign key values in the child table are automatically updated.

STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE	BRANCH_CODE
1	RAM	DELHI	9455123451	18	CS
2	RAMESH	GURGAON	9652431543	18	CS
3	SUJIT	ROHTAK	9156253131	20	ECE
4	SURESH	DELHI		18	IT

REFERENCING RELATION (STUDENT)

Rows in STUDENT relation with BRANCH_CODE CS (ROLL_NO 1 and 2 in this case) will be updated with BRANCH_CODE 'CSE'.

BRANCH

BRANCH_CODE	BRANCH_NAME
CS	COMPUTER SCIENCE
IT	INFORMATION TECHNOLOGY
ECE	ELECTRONICS AND COMMUNICATION ENGINEERING
CV	CIVIL ENGINEERING

REFERENCED RELATION (BRANCH).

Update a row from BRANCH with BRANCH_CODE 'CS' to 'CSE',

More about Integrity Constraints

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Difference between Relational Algebra and SQL

- *Relational Algebra is a procedural query language that specifies how to retrieve dat.,*

- *SQL is a declarative language that specifies what data to retrieve. SQL implementations are influenced by the principles of Relational Algebra.*

Why is Relational Algebra is important

- *It provides a formal foundation for relational databases and is used for query optimization, ensuring that database operations are performed efficiently.*

Relational Algebra

- Relational algebra is a widely used procedural query language.
- It collects instances of relations as input and gives occurrences of relations as output.
- It uses various operation to perform this action.
- Relational algebra operations are performed recursively on a relation.
- The output of these operations is a new relation, which might be formed from one or more input relations.

Fundamental Operations in Relational Algebra

- Selection Operator (σ)
- Projection Operator (∏)
- Union Operator (∪)
- Intersection Operator (∩)
- Cartesian Product (**X**)
- Join Operator (⋈)

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN

Selection Operator (σ)

- The SELECT operation is used for selecting a subset of the tuples according to a given selection condition.
- Sigma(σ)Symbol denotes it.
- It is used as an expression to choose tuples which meet the selection condition.
- Select operation selects tuples that satisfy a given predicate.

Select Operation - σ

- Select Operation:
- The select operation selects tuples that satisfy a given predicate.
- It is denoted by sigma (σ).

Notation: σ p(r)

Where:
σ is used for selection prediction

r is used for relation

p is used as a propositional logic formula which may use connectors like: AND OR and NOT. These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

For example: LOAN Relation

BRANCH_NAME	LOAN_NO	AMOUNT
Downtown	L-17	1000
Redwood	L-23	2000
Perryride	L-15	1500
Downtown	L-14	1500
Mianus	L-13	500
Roundhill	L-11	900
Perryride	L-16	1300

Input:

σ BRANCH_NAME="perryride" (LOAN)

Output:

BRANCH_NAME	LOAN_NO	AMOUNT
Perryride	L-15	1500
Perryride	L-16	1300

Projection Operator (∏)

- The projection eliminates all attributes of the input relation but those mentioned in the projection list.
- The projection method defines a relation that contains a vertical subset of Relation.
- This helps to extract the values of specified attributes to eliminates duplicate values.
- (∏) The symbol used to choose attributes from a relation.
- This operation helps you to keep specific columns from a relation and discards the other columns.

Project Operation - ∏

- Project Operation:

This operation shows the list of those attributes that we wish to appear in the result. Rest of the attributes are eliminated from the table.

- It is denoted by ∏

- Notation: ∏ A1, A2, An (r)

Where

A1, **A2**, **A3** is used as an attribute name of relation **r**.

Example: CUSTOMER RELATION

NAME	STREET	CITY
Jones	Main	Harrison
Smith	North	Rye
Hays	Main	Harrison
Curry	North	Rye
Johnson	Alma	Brooklyn
Brooks	Senator	Brooklyn

Input : ∏ NAME, CITY (CUSTOMER)

Output:

NAME	CITY
Jones	Harrison
Smith	Rye
Hays	Harrison
Curry	Rye
Johnson	Brooklyn
Brooks	Brooklyn

SELECTION & PROJECTION 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

σ Hobby ='stamps'(Person)

ID	Name	Address	Hobby
1123	John	123 Main	Stamps
9876	Bart	5 Pine St	Stamps

$\prod$ Name, Hobby(Person)

Name	Hobby
John	Stamps
John	Coins
Mary	Hiking
Bart	Stamps

UNION Operator (U)

- UNION is symbolized by ∪ symbol.
- It includes all tuples that are in tables A or in B.
- It also eliminates duplicate tuples.
- So, set A UNION set B would be expressed as:

  The result <- A ∪ B
- For a union operation to be valid, the following conditions must hold -R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Union Operation - ∪

- **Union Operation:**

Suppose there are two tuples R and S. 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 ∪ S

- A union operation must hold the following condition:
- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

UNION

Input: ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

UNION Example

A

A	1
B	2
D	3
F	4
E	5

B

A	1
C	2
D	3
E	4

A U B

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

Intersection Operator (∩)

- An intersection is defined by the symbol ∩
- For Ex. A ∩ B
- It defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.

Set Intersection - ∩

- **Set Intersection:**

Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.

- It is denoted by intersection ∩.

Notation: R ∩ S

- **Example:** Using the above DEPOSITOR table and BORROW table

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input : **∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)**

Set Intersection - ∩

Input : **∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)**

Output:

CUSTOMER_NAME
Smith
Jones

Intersection Example

A

A	1
B	2
D	3
F	4
E	5

B

A	1
C	2
D	3
E	4

$A \cap B$

A	1
D	3

Set Difference (-)

- Set Difference:

Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.

- It is denoted by intersection minus (-).

Notation: R - S

- **Example:** Using the above DEPOSITOR table and BORROW table

Example:

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Input : ∏ CUSTOMER_NAME (BORROW) − ∏ CUSTOMER_NAME (DEPOSITOR)

Set Difference (-)

Input :

∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

Output:

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

Cartesian Product(X)

- This type of operation is helpful to merge columns from two relations.
- Generally, a Cartesian product is never a meaningful operation when it performs alone.
- However, it becomes meaningful when it is followed by other operations.
- Cartesian Product operation denoted by **X**.

Cartesian product X

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

Cartesian product X Example

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Input:
EMPLOYEE X DEPARTMENT

Cartesian product X Example

Input:
EMPLOYEE X DEPARTMENT

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

Cartesian Product Example

A

A	1
B	2
D	3
F	4
E	5

B

A	1
C	2
D	3
E	4

A X B

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

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

Rename Operation(ρ)

- Rename Operation:
- The rename operation is used to rename the output relation. It is denoted by **rho** (ρ).

Example: We can use the rename operator to rename STUDENT relation to STUDENTS

ρ(STUDENTS, STUDENT)

Join Operation ⋈

- Join Operations:

A Join operation combines related tuples from different relations, if and only if a given join condition is satisfied. It is denoted by ⋈.

Join operation

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

EMP_CODE	SALARY
101	50000
102	30000
103	25000

Operation: (EMPLOYEE ⋈ SALARY)

Result:

EMP_CODE	EMP_NAME	SALARY
101	Stephan	50000
102	Jack	30000
103	Harry	25000

Join Operation

Join Operation

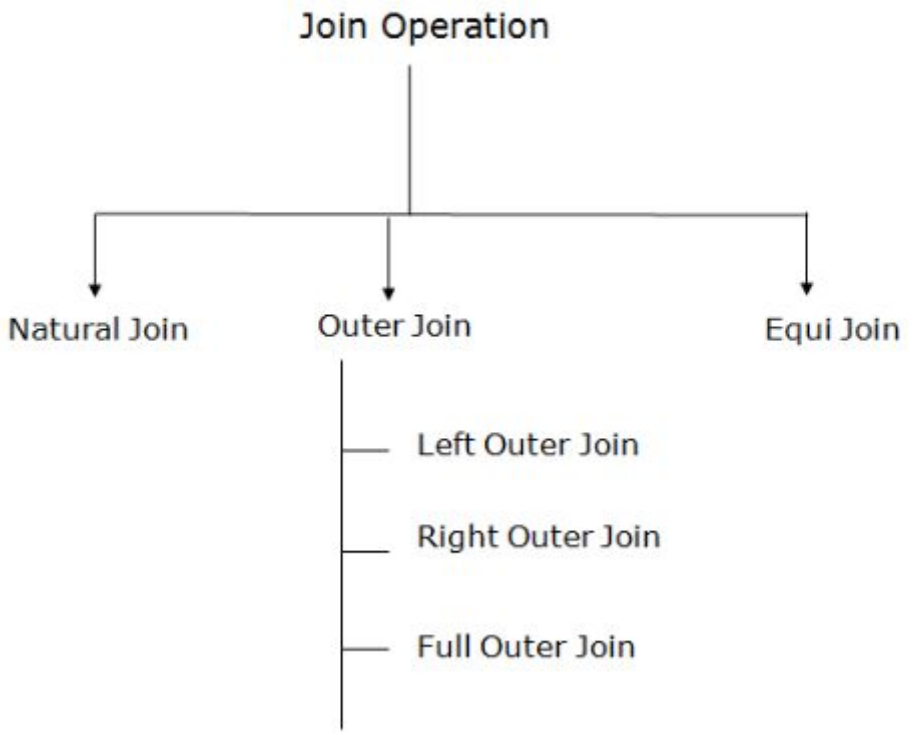
- Join operation is essentially a Cartesian product followed by a selection criterion.
- Join operation denoted by ⋈.
- JOIN operation also allows joining variously related tuples from different relations.
- Types of JOIN:

  Various forms of join operation are:

  - Inner Joins:
    - Theta join
    - EQUI join
    - Natural join
  - Outer join:
    - Left Outer Join
    - Right Outer Join
    - Full Outer Join

Types of Join operations

Types of Join operations:



Natural Join

Natural Join:

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 ⋈.

Example: Let's use the above EMPLOYEE table and SALARY table:

Input:

∏EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Join operation

Example:

EMPLOYEE

EMP_CODE	EMP_NAME
101	Stephan
102	Jack
103	Harry

SALARY

EMP_CODE	SALARY
101	50000
102	30000
103	25000

Input : $\Pi_{EMP\_NAME, SALARY}$ (EMPLOYEE $\bowtie$ SALARY)

Result:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

Natural Join

Example: Natural Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells ⋈ Bars

Note: Bars.name has become Bars.bar to make the natural join "work."

BarInfo(

bar,	beer,	price,	addr
Joe's	Bud	2.50	Maple St.
Joe's	Milller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

)

Outer Join

- Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Outer Join

Input: (EMPLOYEE ⋈ FACT_WORKERS)

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru nagar	Hyderabad	TCS	50000

Outer Join

- Outer Join:

The outer join operation is an extension of the join operation. It is used to deal with missing information.

Example:

EMPLOYEE

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Left Outer Join

An outer join is basically of three types:

a. **Left outer join**
b. **Right outer join**
c. **Full outer join**

Left outer join:

Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names. In the left outer join, tuples in R have no matching tuples in S.

It is denoted by ⟕.

Example: Using the above EMPLOYEE table and FACT_WORKERS table

EMP_NAME	STREET	CITY
Ram	Civil line	Mumbai
Shyam	Park street	Kolkata
Ravi	M.G. Street	Delhi
Hari	Nehru nagar	Hyderabad

FACT_WORKERS

EMP_NAME	BRANCH	SALARY
Ram	Infosys	10000
Shyam	Wipro	20000
Kuber	HCL	30000
Hari	TCS	50000

Left Outer Join ⟕

Input:

EMPLOYEE ⟕ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL

Right Outer Join ⟖

- **Right outer join:**

Right outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.

In right outer join, tuples in S have no matching tuples in R.
It is denoted by ⟖.

Example: Using the above EMPLOYEE table and FACT_WORKERS Relation
Input:
EMPLOYEE ⟖ FACT_WORKERS

Output:

EMP_NAME	BRANCH	SALARY	STREET	CITY
Ram	Infosys	10000	Civil line	Mumbai
Shyam	Wipro	20000	Park street	Kolkata
Hari	TCS	50000	Nehru street	Hyderabad
Kuber	HCL	30000	NULL	NULL

Full Outer Join ⟗

Full outer join:

Full outer join is like a left or right join except that it contains all rows from both tables.

In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.

It is denoted by ⟗

Example: Using the above EMPLOYEE table and FACT_WORKERS table

Input:

EMPLOYEE ⟗ FACT_WORKERS

Output:

EMP_NAME	STREET	CITY	BRANCH	SALARY
Ram	Civil line	Mumbai	Infosys	10000
Shyam	Park street	Kolkata	Wipro	20000
Hari	Nehru street	Hyderabad	TCS	50000
Ravi	M.G. Street	Delhi	NULL	NULL
Kuber	NULL	NULL	HCL	30000

Equi Join ⋈

- **Equi join:**

It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition.

The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Equi Join

Input:

CUSTOMER ⋈ PRODUCT

Output:

CLASS_ID	NAME	PRODUCT_ID	CITY
1	John	1	Delhi
2	Harry	2	Mumbai
3	Harry	3	Noida

Equi Join

Equi join(⋈)

- Equijoin is a **special case of conditional join**
- When a theta join uses only equivalence condition, it becomes a equi join.
- As values of two attributes will be equal in result of equijoin, only one attribute will be appeared in result.

Example

$P \bowtie Q$

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21

Order

Oid	Oname
101	Pizza
101	Noodles
103	Burger

Customer $\bowtie_{Customer.cid=Order.oid}$ Order

Cid	Cname	Age	Oname
101	Ajay	20	Pizza
101	Ajay	20	Noodles
103	Sita	21	Burger

Theta Join

Theta-Join

- $R3 := R1 \bowtie_C R2$
  - Take the product R1 X R2.
  - Then apply σ_C to the result.

Theta Join

Example: Theta Join

Sells(

bar,	beer,	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

)

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

)

Theta Join

Theta join(θ)

- The general case of JOIN operation is called a Theta join.
- It is denoted by symbol **θ.**
- Also Known as Conditional Join.
- Used when you want to join two or more relation based on some conditions.

Example

$$P \bowtie_{\theta} Q$$

Customer

Cid	Cname	Age
101	Ajay	20
102	Vijay	19
103	Sita	21

Order

Oid	Oname
101	Pizza
101	Noodles
103	Burger

$$\text{Customer} \bowtie_{\text{Customer.cid}>\text{Order.oid}} \text{Order}$$

Cid	Cname	Age	Oid	Oname
102	Vijay	19	101	Pizza
102	Vijay	19	101	Noodles
103	Sita	21	101	Pizza
103	Sita	21	101	Noodles