Relational Model Concepts

Relational Model Concepts:

The relational model was designed by E.F.Codd in 1970. It is based on a simple and uniform data structure- the relation- and has solid theoretical foundation. The relational model represents the database as a collection of relations. A Relation is a mathematical concept based on the ideas of sets. The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations.

Informal Definitions:

RELATION: A table of values. A relation may be thought of as a set of rows. A relation may alternately be thought of as a set of columns. Each row of the relation may be given an identifier. Each column typically is called by its column name or column header or attribute name.

Formal Definitions:

RELATION: A Relation may be defined in multiple ways.

The Schema of a Relation: R (A1, A2...An) Relation R is defined over attributes A1, A2...An.

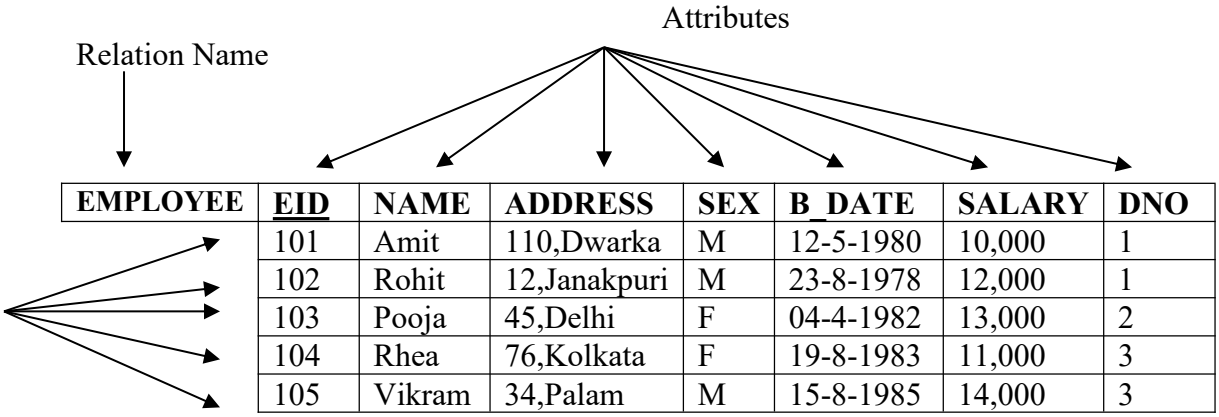
For Example -

EMPLOYEE (EID, NAME, ADDRESS, SEX, SALARY, DNO)

Here, EMPLOYEE is a relation defined over the six attributes EID, NAME, ADDRESS, SEX, SALARY, and DNO each of which has a domain or a set of valid values. For example, the domain of EID is six digit numbers.

Tuple: Each row in the table EMPLOYEE may be called as a tuple. A tuple is an ordered set of values. Each value is derived from an appropriate domain in the table and would consist of six values.

<101, "Amit", "110, Dwarka", "M", 10000,1> is a tuple belonging to the EMPLOYEE relation.



EMPLOYEE	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	DNO
	101	Amit	110,Dwarka	M	12-5-1980	10,000	1
	102	Rohit	12,Janakpuri	M	23-8-1978	12,000	1
	103	Pooja	45,Delhi	F	04-4-1982	13,000	2
	104	Rhea	76,Kolkata	F	19-8-1983	11,000	3
	105	Vikram	34,Palam	M	15-8-1985	14,000	3

Fig.- The attributes and tuples of a relation EMPLOYEE

A relation may be regarded as a set of tuples (rows). Columns in a table are called attributes of the relation.

The relation is formed over the Cartesian product of the sets; each set has values from a domain; that domain is used in a specific role, which is conveyed by the attribute name.

For example, attribute 'Name' is defined over the domain of strings of 25 characters. The role these strings play in the EMPLOYEE relation is that of the name of employee.

Formally, Given R (A1, A2, ... An)

r(R) subset-of dom (A1) X dom (A2) X ....X dom(An)

Definition Summary

Informal Terms	Formal Terms
Table	Relation
Column	Attribute
Row	Tuple
Values in a column	Domain
Table Definition	Schema of Relation

Characteristics of Relations:

1. **Ordering of tuples in a relation r(R):** A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them. Hence, the tuples are not considered to be ordered, even though they appear to be in the tabular form. However, in a file, records are physically stored on disk so there is an order among the records. This ordering indicates 1^{st}, 2^{nd}, i^{th}, and last record in the file. Tuple ordering is not part of a relation definition, because a relation attempts to represent facts at a logical level.
2. **Ordering of attributes in a relation schema R (and of values within each tuple):** We will consider the attributes in R (A1, A2... An) and the values in $t=<v_1, v_2... v_n>$ to be ordered. (However, a more general alternative definition of relation does not require this ordering).
3. **Values in a tuple:** All values are considered atomic (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tuples. Multivalued attributes are represented by separate relations and composite attributes are represented only by their simple component attributes.
4. **Interpretation of a relation:** The relational schema can be interpreted as a declaration. For example, the schema of the EMPLOYEE relation asserts that an employee entity has an Eid, Name, address, Sex, B_Date, Salary, DNO. Each tuple in the relation can then be interpreted as a fact or a particular instance of the assertion.

Relational Model Constraints

Constraints are conditions that must hold on all valid relation instances. There are four main types of constraints:

- Domain constraints
- Key constraints
- Entity integrity constraints
- Referential integrity constraints

Domain Constraints:

It specifies that the value of each attribute A must be an atomic value and from the domain dom (A) for that attributes. The data types associated with domain include standard data types for integers and real numbers, characters, date and so many data types.

Key Constraints:

Super key of R: A set of attributes SK of R such that no two tuples in any valid relation instance r(R) will have the same value for SK. That is, for any distinct tuples t_1 and t_2 in r(R),

$t_1[SK] \neq t_2[SK]$.

Every relation has at least one super key.

Candidate key: An attribute having unique/distinct values can be considered as candidate key.

Primary key: If a relation has several candidate keys, one is chosen arbitrarily to be the primary key. Primary key values are used to **identify** tuples in the relation. The primary key attributes are underlined.

Entity Integrity Constraints:

Entity Integrity: The primary key attributes PK of each relation cannot have null values in any tuple. This is because primary key values are used to identify the individual tuples.

t[PK] <> null for any tuple t in r(R)

Referential Integrity Constraints

The previous constraints (key and entity) are specified on a single relation). The referential integrity constraint is specified between two relations: the referencing relation and the referenced relation. It is used to maintain the consistency among tuples of the two relations.

Foreign key: A set of attributes FK in relational schema R1 is a foreign key of R1 if it satisfies the following two rules:

1. The attributes in FK have the **same domain** as the primary key attribute PK of another relation schema R2.
2. The value of FK in R1 should either be from PK of R2 or null.

Tuples in the referencing relation R1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R2. A tuple t1 in R1 is said to reference a tuple t2 in R2 if t1 [FK] = t2 [PK].

A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.PK

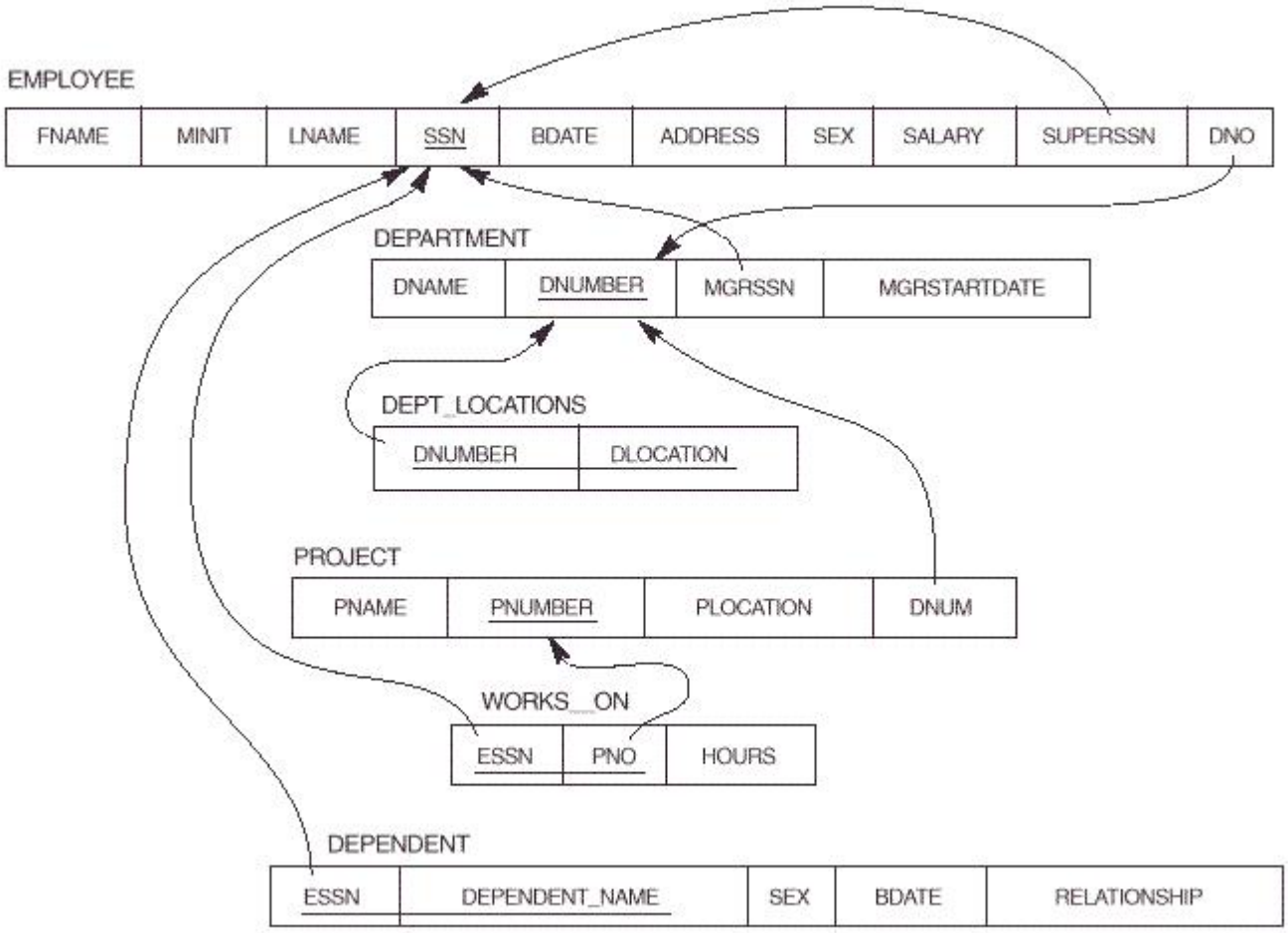


Fig.- Primary Key & Referential integrity constraints on the COMPANY relational database schema

ER -to -Relational Mapping Algorithm

STEP 1: For each regular entity type E in the ER schema, Create a relation R that includes all the simple attributes of E. Include only the simple attributes and only the simple components attribute of a composite attribute of E. Choose one of the key attributes of E as primary key for R.

Step 2: For each weak entity type W in the ER schema with owner entity type E, create a relation R, and include all simple attributes or simple components of composite attributes of W as attributes of R, In addition include as foreign key attributes of R the primary key attributes of the relation that correspond to the owner entity type; this takes care of identifying relationship type of W. The primary key of R is the combination of the primary key of the owner and the partial key of the weak entity type W, if any.

STEP 3: For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations - S and include as foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S. Include all the simple attributes of the 1:1 relationship type R as attributes of S.

STEP 4: For each regular binary 1:N relationship type R; identify the relation S that represents the participating entity type at the N- side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.

STEP 5: For each binary M: N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relation that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M: N relationship type as attributes of S.

STEP 6: For each multivalued attribute A, create a new relation R, This relation R will include an attribute corresponding to A, plus the primary key attribute K- as a foreign key in R - of the relation that represents the entity type of relationship type that has A as an attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, include only its simple components.

STEP 7: For each n-ary relationship type R, n>2, create a new relation S to represent R. Includes as a foreign key attributes in S the primary key of the relations that represent the participating entity types. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity type.

Update Operations And Constraint Violations on Relations

There are three basic update operations on relations:

Insert a tuple.
Delete a tuple.
Modify a tuple.

Insert Operation:

Insert is used to insert a new tuple or tuples in a relation. Insert can violate any of the four types of constraint.

-Domain constraint can be violated if an attribute value is not from the corresponding domain.

-Key constraint can be violated if a key value in the new tuple already exists in another tuple in the relation.

-Entity integrity can be violated if the primary key of the new tuple is null.

-Referential integrity can be violated if the value of foreign key refers to a tuple that does not exit in the referenced relation.

If an insertion violates one or more constraints, two options are available.

First, reject the insertion. Second, correct the reason for rejecting the relation.

Delete operation:

Delete is used to delete tuples. Delete operation can violate only referential integrity constraint.

Referential integrity constraint can be violated if the tuple being deleted is referenced by the foreign key from other tuples in the database.
If a delete operation causes a violation, three options are available:
First, reject the deletion. Second, delete the tuple that reference the tuple that is being deleted. Third, modify the referencing attribute values that cause the violation; each such value is either set to null or changed to reference another valid tuple.

Modify operation:

Modify operation is used to change the values of one or more attributes in a tuple(s). It is necessary to specify a condition on the attributes of a relation to select the tuple(s) to be modified.
Modifying an attribute that is neither a primary key nor a foreign key usually causes no problem. Modifying a primary key value is similar to deleting one tuple and inserting another in its place.
Hence, the issue discussed earlier under both 'Insert' and 'Delete' come into play.

DEPARTMENT	DNO	DNAME	MGRID	MGRJD
	1	Research	102	11-5-1998
	4	Administration	104	10-3-1997
	5	Head office	108	01-1-2000

DEPT_LOCATION	DNO	DLOCATION
	1	A-Block
	4	B-Block
	5	C-Block
	5	D-Block
	5	A-Block

EMPLOYEE	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	DNO
	101	Amit	110,Dwarka	M	12-5-1980	30,000	5
	102	Rohit	12,Janakpuri	M	23-8-1978	40,000	5
	103	Pooja	45,Delhi	F	04-4-1982	25,000	4
	104	Rhea	76,Kolkata	F	19-8-1983	43,000	4
	105	Vikram	34,Palam	M	15-8-1985	38,000	5
	106	Charu	12, Banaras	F	21-6-1981	25,000	5
	107	Karan	9,Mumbai	M	10-5-1985	25,000	4
	108	Vishal	34,Channai	M	14-9-1979	55,000	1

PROJECT	PNO	PNAME	PLOCATION	DNUM
	10	Computerization	B-Block	4
	20	Reorganization	A-Block	1
	30	Marketing	B-Block	4
	40	Accounts	C-Block	5
	50	HRD	D-Block	5

WORKS_ON	EID	PNO	HOURS
	101	10	11
	101	30	08
	102	10	23
	103	20	16
	104	40	19
	108	10	20
	105	50	17
	106	40	13
	107	20	21

DEPENDENT	EID	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	101	Monu	M	10-5-94	Son
	101	Sarita	F	15-8-79	Wife
	102	Pooja	F	23-7-80	Wife
	102	Neetu	F	09-4-99	Daughter
	103	Chintu	M	14-9-00	Son
	105	Neha	F	17-2-82	Wife
	107	Annu	F	11-1-83	Wife

Fig.- A relational database instance (state) of the COMPANY schema.

The Relational Algebra

Relational algebra is a **collection of operations** that are used to manipulate entire relations. These operation are used:

-To select tuples and attributes from relations.

-To specify retrieval requests (queries).

The result of each operation is a new relation, which can be further manipulated. The relational algebra operation are divided into two group:

-Relational database operations: SELECT, PROJECT, And JOIN.

-Set theory operations: UNION, INTERSECTION, DIFFERENCE, and CARTESIAN PRODUCT.

SELECT Operation:

The SELECT operation is used to select a **subset of the tuples** in a relation that satisfy a **selection condition**.

SELECT operation (denoted by σ):

-Selects the tuples (rows) from a relation R that satisfy a certain selection condition c

-The condition c is an arbitrary Boolean expression on the attributes of R

-Resulting relation has the same attributes as R

-Resulting relation includes each tuple in r(R) whose attribute values satisfy the condition c.

-The SELECT operator is **unary**; it is applied on one relation.

Generally, the SELECT operation is denoted as:

$\sigma_{\text{<selection condition>}}$(<relation name>)

Examples:

$\sigma_{DNO=5}$ (EMPLOYEE)

DEPT5_EMPS	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	DNO
	101	Amit	110,Dwarka	M	12-5-1980	30,000	5
	102	Rohit	12,Janakpuri	M	23-8-1978	40,000	5
	105	Vikram	34,Palam	M	15-8-1985	38,000	5
	106	Charu	12, Banaras	F	21-6-1981	25,000	5

$\sigma_{SALARY>30000}$ (EMPLOYEE)

EMPLOYEE	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	DNO
	102	Rohit	12,Janakpuri	M	23-8-1978	40,000	5
	104	Rhea	76,Kolkata	F	19-8-1983	43,000	4
	105	Vikram	34,Palam	M	15-8-1985	38,000	5

$\sigma_{SEX=F}$(EMPLOYEE)

FEMALE_EMPS	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	DNO
	103	Pooja	45,Delhi	F	04-4-1982	25,000	4
	104	Rhea	76,Kolkata	F	19-8-1983	43,000	4
	106	Charu	12, Banaras	F	21-6-1981	25,000	5

PROJECT Operation:

The PROJECT operation selects certain columns/attributes from the table/relation.

PROJECT Operation (denoted by π):

-Keeps only certain attributes (columns) from a relation R specified in an attribute list L

-Resulting relation has only those attributes of R specified in L.

-If the L includes only non-key attributes of R, the PROJECT operation eliminates duplicate tuples in the resulting relation so that it remains a mathematical set (no duplicate elements).

-If L includes a key of the relation, the resulting relation has the same number of tuples as the original one.

-The number of tuples in resulting relation is always less than or equal to the number of tuples in the original relation.

- The PROJECT operator is **unary**; it is applied on one relation.

The general form of the project operation is:

$\pi_{<attribute\ list>}(<relation\ name>)$

Example:

$\pi_{NAME, SALARY}$ (EMPLOYEE)

EMPLOYEE	NAME	SALARY
	Amit	30,000
	Rohit	40,000
	Pooja	25,000
	Rhea	43,000
	Vikram	38,000
	Charu	25,000
	Karan	25,000
	Vishal	55,000

$\pi_{SEX, SALARY}$ (EMPLOYEE)

EMPLOYEE	SEX	SALARY
	M	30,000
	M	40,000
	F	25,000
	F	43,000
	M	38,000
	M	25,000
	M	55,000

If several male employees have salary 30000, only a single tuple <M, 30000> is kept in the resulting relation. Duplicate tuples are eliminated by the π operation.

Sequences of operations:

Several operations can be combined to form a relational algebra expression (query).

Example: Retrieve the names and salaries of employees who work in department 5:

$\pi_{NAME, SALARY}$ ($\sigma_{DNO=5}$(EMPLOYEE))

EMPLOYEE	NAME	SALARY
	Amit	30,000
	Rohit	40,000
	Vikram	38,000
	Charu	25,000

Alternatively, we specify explicit intermediate relations for each step:

DEPT5_EMPS $\leftarrow \sigma_{DNO=5}$(EMPLOYEE)

RESULT $\leftarrow \pi_{NAME, SALARY}$ (DEPT5_EMPS)

DEPT5_EMPS	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	DNO
	101	Amit	110,Dwarka	M	12-5-1980	30,000	5
	102	Rohit	12,Janakpuri	M	23-8-1978	40,000	5
	105	Vikram	34,Palam	M	15-8-1985	38,000	5
	106	Charu	12, Banaras	F	21-6-1981	25,000	5

RESULT	NAME	SALARY
	Amit	30,000
	Rohit	40,000
	Vikram	38,000
	Charu	25,000

Note: Attributes can also optionally be renamed in the resulting relation:

Set Operations:

These operations are binary; they can be applied to two sets/relations/tables. When these operations are performed on relational database, we must make sure that the operation can be applied to two relations so that result is also a valid relation.

Union Compatibility: Two relations R (A_1, A2, …, A_n) and S(B_1,B_2,…,B_n) are said to be union compatible if they have the same degree n, and dom (A_i) = dom (B_i) for $1 \leq i \leq n$. This means that two relations have the same number of attributes and that each pair of corresponding attributes has the same domain.

Note: Set operations are performed on R & S only if they are union compatible.

We can define these operations on two union compatible relations R and S as

UNION (R ∪ S): The result of this operation is a relation that includes all tuple that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

INTERSECTION (R ∩ S): The result of this operation is a relation that includes all tuples that are in both R and S.

DIFFERENCE (R – S): The result of this operation is a relation that includes all tuples that are in R but not in S.

Let us consider two union compatible relations STUDENT & INSTRUCTOR as shown in fig. The result of three set operations described above is shown in fig. below.

STUDENT	NAME
	Amit
	Rohit
	Pooja
	Rhea
	Vikram
	Charu
	Karan
	Vishal

INSTUCTOR	NAME
	Ashish
	Sikha
	Amit
	Juhi
	Rohit

S ∪ I	NAME
	Amit
	Rohit
	Pooja
	Rhea
	Vikram
	Charu
	Karan
	Vishal
	Ashish
	Sikha
	Juhi

S ∩ I	NAME
	Amit
	Rohit

S – I	NAME
	Pooja
	Rhea
	Vikram
	Charu
	Karan
	Vishal

I – S	NAME
	Ashish
	Sikha
	Juhi

Note: 1) Both UNION and INTERSECTION are commutative operations.
i.e. $R \cup S = S \cup R$ and $R \cap S = S \cap R$
2) The DIFFERENCE operation is not commutative in general.
i.e. $R - S \neq S - R$

CARTESIAN PRODUCT (R X S): This is also a binary set operation, but the relation on which it is applied do not have to be union compatible. This operation is used to combine tuples from two relations so that related tuples can be identified. In general, the result of R (A_1, A_2,…, A_n) X S(B_1,B_2,…,B_m) is a relation Q with n + m attributes Q(A_1,A_2,…,A_n,B_1,B_2,…,B_m). Hence, if R has n_R tuples and S has n_S tuples, then R X S will have n_R*n_S tuples.

Example: Suppose we want to retrieve for each female employee, a list of the names of her dependents.

FEMALE_EMPS ← $\sigma_{SEX='F'}$ (EMPLOYEE)

FEMALE_EMPS	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	DNO
	103	Pooja	45,Delhi	F	04-4-1982	25,000	4
	104	Rhea	76,Kolkata	F	19-8-1983	43,000	4
	106	Charu	12, Banaras	F	21-6-1981	25,000	5

EMPNAMES ← $\pi_{EID, NAME}$ (FEMALE_EMPS)

EMPNAMES	EID	NAME
	103	Pooja
	104	Rhea
	106	Charu

EMP_DEPENDENTS ← EMPNAMES X DEPENDENTS

EMP_DEPENDENTS	EID	NAME	EID	DEPENDENT_NAME	SEX	…
	103	Pooja	101	Monu	M	…
	103	Pooja	101	Sarita	F	…
	103	Pooja	102	Pooja	F	…
	103	Pooja	102	Neetu	F	…
	103	Pooja	103	Chintu	M	…
	103	Pooja	105	Neha	F	…
	103	Pooja	107	Annu	F	…
	104	Rhea	101	Monu	M	…
	…	…	…	…	…	…

ACTUAL_DEPENDENTS ← $\sigma_{EID=EID}$ (EMP_DEPENDENTS)

ACTUAL_DEPENDENTS	EID	NAME	EID	DEPENDENT_NAME	…
	103	Pooja	103	Chintu	…

RESULT ← $\pi_{NAME, DEPENDENT\_NAME}$ (ACTUAL_DEPENDENTS)

RESULT	NAME	DEPENDENT_NAME
	Pooja	Chintu

Fig.- The CARTESIAN PRODUCT operation

Note: 1) The CARTESIAN PRODUCT is rarely used. We will prefer JOIN operation.

2) CARTESIAN PRODUCT is a *meaningless operation* on its own. It can *combine related tuples* from two relations *if followed by the appropriate SELECT operation.*

JOIN Operations (×):

The JOIN operation is used to combine related tuples from two relations into single tuple. The general form of a JOIN operation on two relation R (A_1, A_2,…,A_n) and S(B_1,B_2,…,B_m) is

$$R \times_{\text{<join condition>}} S$$

The result of the JOIN is a relation Q with n + m attribute Q (A_1, A2,…,A_n, B_1, B_2,…, B_m). Q has one tuple for each combination of tuples--one from R and one from S –whenever the combination satisfies the **join condition**.

This is the main difference between CARTESIAN PRODUCT and JOIN: in JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.

Example: we want to retrieve the name of the manager of each department.

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \times_{\text{MGRID=EID}} \text{EMPLOYEE}$$

$$\text{RESULT} \leftarrow \pi_{\text{NAME}} (\text{DEPT\_MGR})$$

DEPT_MGR	DNO	DNAME	MGRID	…	EID	NAME	…
	1	Research	102		102	Rohit	
	4	Administration	104		104	Rhea	
	5	Head Office	108		108	Vishal	

Fig.-the JOIN operation

A join condition is of the form:

<Condition> AND <condition> AND … AND <condition>

Where each condition is of the form $A_i \theta B_j$, A_i is an attribute of R, B_j is an attribute of S. Ai and Bj have the same domain and θ is one of the comparison operator $\{=, <, \leq, >, \geq, \neq\}$.

THETA JOIN: A join operation with a general join condition discussed above is called theta join. It is Similar to a CARTESIAN PRODUCT followed by a SELECT. **EQUIJOIN**: It is the most common join operation having join condition with equality comparisons (=) only. The join condition includes one or more *equality comparisons* involving attributes from R and S. Join condition is of the form:

$(A_i=B_j)$ AND ... AND $(A_h=B_k)$; $1 \leq i, h \leq m, 1 \leq j, k \leq n$

In the above EQUIJOIN operation:

A_i... A_h are called the **join attributes** of R

B_j... B_k are called the **join attributes** of S

Note: In the result of EQUIJOIN we always have one or more pair of attributes that have **identical values (superfluous)** in every tuple. In above fig. Value of EID and MGRID are identical.

NATURAL JOIN (*): In an EQUIJOIN Q ← R × c S, the join attributes of S appear *redundantly* in the result relation Q. In a NATURAL JOIN, the *redundant join attributes* of S are *eliminated* from Q. The equality condition is *implied* and need not be specified. A general notation for NATURAL JOIN is:

$$Q \leftarrow R *_{\text{(join attributes of R),(join attributes of S)}} S$$

Example: Retrieve each Employee's name and the name of the Department he/she works for:

$$Q \leftarrow \text{EMPLOYEE} *_{\text{(DNO),(DNUMBER)}} \text{DEPARTMENT}$$

$$\text{RESULT} \leftarrow \pi_{\text{NAME, DNAME}}(Q)$$

If the join attributes have the *same names* in both relations, they *need not be specified* and we can write Q ← R * S.

Complete Set of Relational Algebra Operations:
All the operations discussed so far can be described as a sequence of **only** the operations SELECT, PROJECT, UNION, SET DIFFERENCE, and CARTESIAN PRODUCT.
Hence, the set {σ, π, ∪, −, X} is called a ***complete set*** of relational algebra operations. Any of the other relational algebra operations can be expressed as a sequence of operation from this set.
Example: 1) The INTERSECTION operation can be expressed by using UNION and DIFFERENCE as follows:

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

2) A JOIN operation can be specified as a CARTESIAN PRODUCT followed by a SELECT operation:

$R \times_{<condition>} S = \sigma_{<condition>} (R \text{ X } S)$

3) A NATURAL JOIN can be specified as a CARTESIAN PRODUCT followed by SELECT and PROJECT operations.

DIVISION Operation:
The DIVISION operation is useful for a special kind of query that sometimes occurs in database applications. For example, 'retrieve the name of employees who works on all the projects that 'Amit' works on.'

AMIT ← $\sigma_{NAME='Amit'}$ (EMPLOYEE)
AMIT_PNOS ← π_{PNO} (WORKS_ON * AMIT)
EID_PNOS ← $\pi_{EID, PNO}$ (WORKS_ON)
EID ← EID_PNOS ÷ AMIT_PNOS
RESULT ← π_{NAME} (EID * EMPLOYEE)

Generally, DIVISION operation is applied to relations R (Z) ÷ S (X), where $X \subseteq Z$
Example: Let R and S be two relations

R	A	B
	a_1	**b_1**
	a_2	**b_1**
	a_3	**b_1**
	a_4	b_1
	a_1	b_2
	a_3	b_2
	a_2	b_3
	a_3	b_3
	a_4	b_3
	a_1	**b_4**
	a_2	**b_4**
	a_3	**b_4**

S	A
	a_1
	a_2
	a_3

T	B
	b_1
	b_4

Fig.-5.11 The division operation. T ← R ÷ S

Additional Relational Operations

AGGREGATE FUNCTIONS

Functions such as SUM, COUNT, AVERAGE, MIN, and MAX are often applied to sets of values or sets of tuples in database applications.

In general, a FUNCTION operation is expressed as:

$$<\text{grouping attributes}>\ f\ <\text{function list}>\ (R)$$

The grouping attributes are optional

Example: 1) Retrieve the average salary of all employees (no grouping needed):

$$R\ (AVGSAL) \leftarrow f_{\text{AVERAGE SALARY}}\ (EMPLOYEE)$$

Example: 2) For each department, retrieve the department number, the number of employees, and the average salary (in the department):

$$R\ (DNO, NUMEMPS, AVGSAL) \leftarrow {}_{DNO}f_{\text{COUNT EID, AVERAGE SALARY}}\ (EMPLOYEE)$$

DNO is called the *grouping attribute* in the above example.

Recursive Closure Operation:

This operation is applied to a recursive relationship between tuples of the same type, such as relationship between an employee and supervisor. This relationship is described by the foreign key SID of the EMPLOYEE relation as given below:

EMPLOYEE	EID	NAME	ADDRESS	SEX	B_DATE	SALARY	SID	DNO
	101	Amit	110,Dwarka	M	12-5-1980	30,000	102	5
	102	Rohit	12,Janakpuri	M	23-8-1978	40,000	108	5
	103	Pooja	45,Delhi	F	04-4-1982	25,000	104	4
	104	Rhea	76,Kolkata	F	19-8-1983	43,000	108	4
	105	Vikram	34,Palam	M	15-8-1985	38,000	102	5
	106	Charu	12, Banaras	F	21-6-1981	25,000	102	5
	107	Karan	9,Mumbai	M	10-5-1985	25,000	104	4
	108	Vishal	34,Channai	M	14-9-1979	55,000	null	1

Fig.- Recursive relationship

It relates each employee tuple (in the role of supervisee) to another employee tuple (in the role of supervisor).

Recursive closure operation means to retrieve the all supervisees of an employee e at all level—that is,

All employee e_1 directly supervised by e,
All employee e_2 directly supervised by e_1,
All employee e_3 directly supervised by e_2, and so on.

Example: To retrieve the EIDs of all employees e_1 directly supervised –at one level-by the employee e whose name is 'Vishal'.

$$VISHAL\_EID \leftarrow \pi_{\text{EID}}\ (\sigma_{\text{NAME='Vishal'}}\ (EMPLOYEE)$$

$$SUPERVISION\ (EID1, EID2) \leftarrow \pi_{\text{EID, SID}}\ (EMPLOYEE)$$

$$RESULT1\ (EID) \leftarrow \pi_{\text{EID1}}\ (SUPERVISION \times_{\text{EID2=EID}} VISHAL\_EID)$$

To retrieve the EIDs of all employees e_2 directly supervised –at one level-by the employee e_1 whose name is 'Vishal'.

$$RESULT2\ (EID) \leftarrow \pi_{\text{EID1}}\ (SUPERVISION \times_{\text{EID2=EID}} RESULT1)$$

To get both sets of employees supervised at level one and two by 'Vishal', we can apply the UNION operation to the two results.

$$RESULT3 \leftarrow (RESULT1 \cup RESULT2)$$

SUPERVISION	EID1	EID2
	101	102
	102	108
	103	104
	104	108
	105	102
	106	102
	107	104
	108	null

RESULT1	EID
	102
	104

RESULT2	EID
	101
	103
	105
	106
	107

RESULT	EID
	101
	103
	105
	106
	107
	102
	104

Fig.- Two-level recursion.

Although it is possible to retrieve employees at each level and then take their UNION but we cannot. We have to specify the level explicitly. Specifying a query such as "retrieve the supervisees of 'Vishal' at all levels" is not logical. If we do not know the maximum number of level, we would need a looping mechanism.

OUTER JOIN

In a regular EQUIJOIN or NATURAL JOIN operation, tuples in R or S that do not have matching tuples in the other relation *do not appear in the result.* Tuples without a 'related tuple' are eliminated from the result. Tuples with null in the join attributes are also eliminated.

But Some queries require all tuples in R (or S or both) to appear in the result.

A set of operation OUTER JOIN can be used when we want to keep all tuples in R or S or both in the result—whether or not they have matching tuples in the other relation. When no matching tuples are found, **null**s are placed for the missing attributes.

LEFT OUTER JOIN(R × S): The LEFT OUTER JOIN operation keeps every tuple in the first or left relation R in R × S. If no matching tuples are found in S, then attributes of S are filled with null values.

Example: Retrieve a list of all employee names and also the name of the departments they manage.

$$TEMP \leftarrow (EMPLOYEE \times_{EID=MGRID} DEPARTMENT)$$

$$RESULT \leftarrow \pi_{NAME, DNAME} (TEMP)$$

RESULT	NAME	DNAME
	Amit	null
	Rohit	Research
	Pooja	null
	Rhea	Administration
	Vikram	null
	Charu	null
	Karan	null
	Vishal	Head Office

Fig.-: The LEFT OUTER JOIN operation

RIGHT OUTER JOIN(R × S): It keeps every tuples in the second or right relation S in the result of R × S. If no matching tuples are found in R, then attributes of R are filled with null values.
FULL OUTER JOIN(R × S): It keeps all tuples in both left and right relation. When no matching tuples are founds, null values are filled up.
OUTER UNION:
The OUTER UNION operation is used to take the union of tuples from two relations that are **not union compatible**. This operation will take the UNION of tuples in two relations that are **partially compatible** (only some of their attributes are union compatible).
The attributes that are not union compatible from either relation are kept in the result, and tuples that have no values for these attributes are filled with null values.

References:
1. Fundamentals of Database Systems: Ramez Elmasri, Shamkant B. Navathe, Pearson.
2. Database System Concepts: Avi Silberschatz · Henry F. Korth · S. Sudarshan, McGraw Hill