

Relational Model

Learning Objectives. This chapter is dedicated to relational model which is in use since late 1970s. Various operations in relational algebra and relational calculus are given in this chapter. After completing this chapter the reader should be familiar with the following concepts:

- Evolution and importance of relational model
- Terms in relational model like tuple, domain, cardinality, and degree of a relation
- Operations in relational algebra and relational calculus
- Relational algebra vs relational calculus
- QBE and various operations in QBE

3.1 Introduction

E.F. Codd (Edgar Frank Codd) of IBM had written an article “A relational model for large shared data banks” in June 1970 in the Association of Computer Machinery (ACM) Journal, Communications of the ACM. His work triggered people to work in relational model. One of the most significant implementations of the relational model was “System R,” which was developed by IBM during the late 1970s. System R was intended as a “proof of concept” to show that relational database systems could really build and work efficiently. It gave rise to major developments such as a structured query language called SQL which has since become an ISO standard and de facto standard relational language. Various commercial relational DBMS products were developed during the 1980s such as DB2, SQL/DS, and Oracle. In relational data model the data are stored in the form of tables.

3.2 CODD’S Rules

In 1985, Codd published a list of rules that became a standard way of evaluating a relational system. After publishing the original article Codd stated that there are no systems that will satisfy every rule. Nevertheless the rules represent relational ideal and remain a goal for relational database designers.

Note: The rules are numbered from 1 to 12 whereas the statements preceded by the *bullet mark* are interpretations of the Codd's rule:

1. *The Information Rule.* All information in a relational database is represented explicitly at the logical level and in exactly one way-by values in tables:
 - Data should be presented to the user in the tabular form.
2. *Guaranteed Access Rule.* Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name:
 - Every data element should be unambiguously accessible.
3. *Systematic Treatment of Null Values.* Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.
4. *Dynamic On-line Catalog Based on the Relational Model.* The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data:
 - The database description should be accessible to the users.
5. *Comprehensive Data Sublanguage Rule.* A relational system may support several languages and various modes of terminal use (for example the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and whose ability to support all the following is comprehensive: data definition, view definition, data manipulation (interactive and by program), integrity constraints, and transaction boundaries:
 - A database supports a clearly defined language to define the database, view the definition, manipulate the data, and restrict some data values to maintain integrity.
6. *View Updating Rule.* All views that are theoretically updatable are also updatable by the system:
 - Data should be able to be changed through any view available to the user.
7. *High-level Insert, Update, and Delete.* The capacity of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data:
 - All records in a file must be able to be added, deleted, or updated with singular commands
8. *Physical Data Independence.* Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods:

- Changes in how data are stored or retrieved should not affect how a user accesses the data.
9. *Logical Data Independence.* Application programs and terminal activities remain logically unimpaired whenever information-preserving changes of any kind that theoretically permit unimpairment are made to the base tables:
 - A user's view of data should be unaffected by its actual form in files.
 10. *Integrity Independence.* Integrity constraints specific to a particular relational database must be definable in a relational data sublanguage and storable in the catalog, not in the application programs.
 - Constraints on user input should exist to maintain data integrity.
 11. *Distribution Independence.* A relational DBMS has distribution independence. Distribution independence implies that users should not have to be aware of whether a database is distributed.
 - A database design should allow for distribution of data over several computer sites.
 12. *Nonsubversion Rule.* If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time):
 - Data fields that affect the organization of the database cannot be changed.

There is one more rule called Rule Zero which states that “For any system that is claimed to be a relational database management system, that system must be able to manage data entirely through capabilities.”

3.3 Relational Data Model

The relational model uses a collection of tables to represent both data and the relationships among those data. Tables are logical structures maintained by the database manager. The relational model is a combination of three components, such as Structural, Integrity, and Manipulative parts.

3.3.1 Structural Part

The structural part defines the database as a collection of relations.

3.3.2 Integrity Part

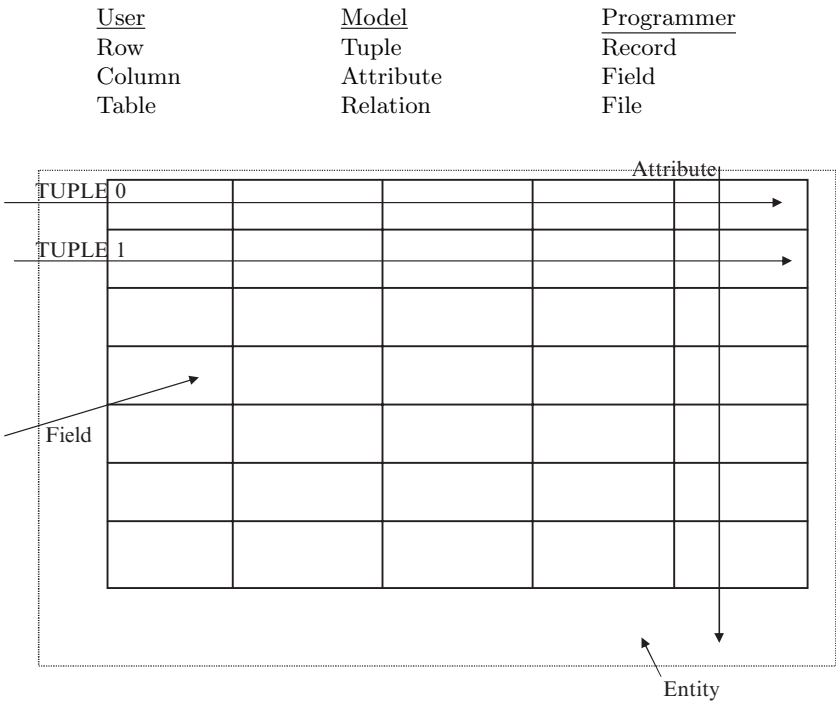
The database integrity is maintained in the relational model using primary and foreign keys.

3.3.3 Manipulative Part

The relational algebra and relational calculus are the tools used to manipulate data in the database. Thus relational model has a strong mathematical background. The key features of relational data model are as follows:

- Each row in the table is called tuple.
- Each column in the table is called attribute.
- The intersection of row with the column will have data value.
- In relational model rows can be in any order.
- In relational model attributes can be in any order.
- By definition, all rows in a relation are distinct. No two rows can be exactly the same.
- Relations must have a key. Keys can be a set of attributes.
- For each column of a table there is a set of possible values called its domain. The domain contains all possible values that can appear under that column.
- Domain is the set of valid values for an attribute.
- Degree of the relation is the number of attributes (columns) in the relation.
- Cardinality of the relation is the number of tuples (rows) in the relation.

The terms commonly used by user, model, and programmers are given later.



3.3.4 Table and Relation

The general doubt that will rise when one reads the relational model is the difference between table and relation. For a table to be relation, the following rules holds good:

- The intersection row with the column should contain single value (atomic value).
- All entries in a column are of same type.
- Each column has a unique name (column order not significant).
- No two rows are identical (row order not significant).

Example of Relational Model

Representation of Movie data in tabular form is shown later.

MOVIE			
Movie Name	Director	Actor	Actress
Titanic	James Cameron	Leonardo DiCapiro	Kate Winslet
Autograph	Cheran	Cheran	Gopika
Roja	Maniratnam	AravindSwamy	Madubala

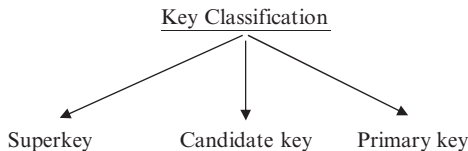
In the earlier relation:

The degree of the relation (i.e., is the number of column in the relation) = 4.

The cardinality of the relation (i.e., the number of rows in the relation) = 3.

3.4 Concept of Key

Key is an attribute or group of attributes, which is used to identify a row in a relation. Key can be broadly classified into (1) Superkey (2) Candidate key, and (3) Primary key



3.4.1 Superkey

A superkey is a subset of attributes of an entity-set that uniquely identifies the entities. Superkeys represent a constraint that prevents two entities from ever having the same value for those attributes.

3.4.2 Candidate Key

Candidate key is a minimal superkey. A candidate key for a relation schema is a minimal set of attributes whose values uniquely identify tuples in the corresponding relation.

Primary Key

The primary key is a designated candidate key. It is to be noted that the primary key should not be null.

Example

Consider the employee relation, which is characterized by the attributes, employee ID, employee name, employee age, employee experience, employee salary, etc. In this employee relation:

Superkeys can be employee ID, employee name, employee age, employee experience, etc.

Candidate keys can be employee ID, employee name, employee age.

Primary key is employee ID.

Note: If we declare a particular attribute as the primary key, then that attribute value cannot be NULL. Also it has to be distinct.

3.4.3 Foreign Key

Foreign key is set of fields or attributes in one relation that is used to “refer” to a tuple in another relation.

3.5 Relational Integrity

Data integrity constraints refer to the accuracy and correctness of data in the database. Data integrity provides a mechanism to maintain data consistency for operations like INSERT, UPDATE, and DELETE. The different types of data integrity constraints are Entity, NULL, Domain, and Referential integrity.

3.5.1 Entity Integrity

Entity integrity implies that a primary key cannot accept null value. The primary key of the relation uniquely identifies a row in a relation. Entity integrity means that in order to represent an entity in the database it is necessary to have a complete identification of the entity’s key attributes.

Consider the entity PLAYER; the attributes of the entity PLAYER are Name, Age, Nation, and Rank. In this example, let us consider PLAYER's name as the primary key even though two players can have same name. We cannot insert any data in the relation PLAYER without entering the name of the player. This implies that primary key cannot be null.

3.5.2 Null Integrity

Null implies that the data value is not known temporarily. Consider the relation PERSON. The attributes of the relation PERSON are name, age, and salary. The age of the person cannot be NULL.

3.5.3 Domain Integrity Constraint

Domains are used in the relational model to define the characteristics of the columns of a table. Domain refers to the set of all possible values that attribute can take. The domain specifies its own name, data type, and logical size. The logical size represents the size as perceived by the user, not how it is implemented internally. For example, for an integer, the logical size represents the number of digits used to display the integer, not the number of bytes used to store it. The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values. Strictly speaking, only values from the same domain should ever be compared or be integrated through a union operator. The domain integrity constraint specifies that each attribute must have values derived from a valid range.

Example 1

The age of the person cannot have any letter from the alphabet. The age should be a numerical value.

Example 2

Consider the relation APPLICANT. Here APPLICANT refers to the person who is applying for job. The sex of the applicant should be either male (M) or female (F). Any entry other than M or F violates the domain constraint.

3.5.4 Referential Integrity

In the relational data model, associations between tables are defined through the use of foreign keys. The referential integrity rule states that a database

must not contain any unmatched foreign key values. It is to be noted that referential integrity rule does not imply a foreign key cannot be null. There can be situations where a relationship does not exist for a particular instance, in which case the foreign key is null. A referential integrity is a rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null.

3.6 Relational Algebra

The relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation. Thus, both the operands and the results are relations; hence the output from one operation can become the input to another operation. This allows expressions to be nested in the relational algebra. This property is called closure. Relational algebra is an abstract language, which means that the queries formulated in relational algebra are not intended to be executed on a computer. Relational algebra consists of group of relational operators that can be used to manipulate relations to obtain a desired result. Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system.

3.6.1 Role of Relational Algebra in DBMS

Knowledge about relational algebra allows us to understand query execution and optimization in relational database management system. The role of relational algebra in DBMS is shown in Fig. 3.1. From the figure it is evident that when a SQL query has to be converted into an executable code, first it has to be parsed to a valid relational algebraic expression, then there should be a proper query execution plan to speed up the data retrieval. The query execution plan is given by query optimizer.

3.7 Relational Algebra Operations

Operations in relational algebra can be broadly classified into set operation and database operations.

3.7.1 Unary and Binary Operations

Unary operation involves one operand, whereas binary operation involves two operands. The selection and projection are unary operations. Union, difference, Cartesian product, and Join operations are binary operations:

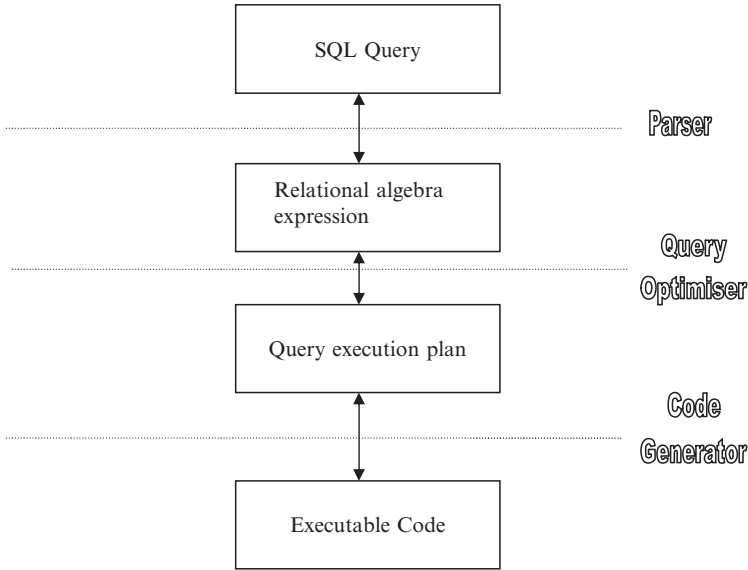
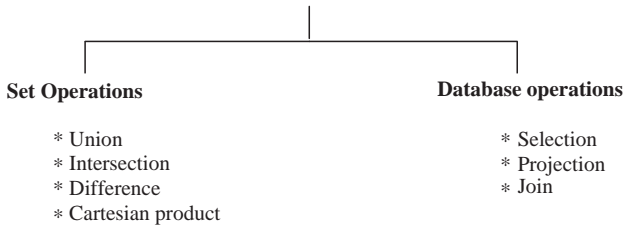


Fig. 3.1. Relational algebra in DBMS

- Unary operation operate on one relation
- Binary operation operate on more than one relation

Relational algebra operations



Three main database operations are SELECTION, PROJECTION, and JOIN.

Selection Operation

The selection operation works on a single relation R and defines a relation that contains only those tuples of R that satisfy the specified condition (Predicate). Selection operation can be considered as row wise filtering. This is pictorially represented in Fig. 3.2

Syntax of Selection Operation

The syntax of selection operation is: $\sigma_{\text{Predicate}}(R)$. Here R refers to relation and predicate refers to condition.



Fig. 3.2. Pictorial representation of SELECTION operation

Illustration of Selection Operation

To illustrate the SELECTION operation consider the STUDENT relation with the attributes Roll number, Name, and GPA (Grade Point Average).

Example

Consider the relation STUDENT shown later:

STUDENT		
Student Roll. No	Name	GPA
001	Aravind	7.2
002	Anand	7.5
003	Balu	8.2
004	Chitra	8.0
005	Deepa	8.5
006	Govind	7.2
007	Hari	6.5

Query 1: List the Roll. No, Name, and GPA of those students who are having GPA of above 8.0

Query expressed in relational algebra as $\sigma_{\text{GPA} > 8}(\text{Student})$.
The result of the earlier query is:

Student Roll. No	Name	GPA
003	Balu	8.2
005	Deepa	8.5

Query 2: Give the details of first four students in the class.

Relational algebra expression is $\sigma_{\text{Roll. No} \leq (\text{student})}$.

Table as a result of query 2 is

Student Roll. No	Name	GPA
001	Aravind	7.2
002	Anand	7.5
003	Balu	8.2
004	Chitra	8.0

Projection Operation

The projection operation works on a single relation R and defines a relation that contains a vertical subject of R, extracting the values of specified attributes and elimination duplicates. The projection operation can be considered as column wise filtering. The projection operation is pictorially represented in Fig. 3.3.

Syntax of Projection Operation

The syntax of projection operation is given by: $\prod_{a_1, a_2, \dots, a_n} (R)$.

Where a_1, a_2, \dots, a_n are attributes and R stands for relation.

STAFF				
Staff No	Name	Gender	Date of birth	Salary
SL21	Raghavan	M	1-5-76	15,000
SL22	Raghu	M	1-5-77	12,000
SL55	Babu	M	1-6-76	12,500
SL66	Kingsly	M	1-8-78	10,000



Fig. 3.3. Pictorial representation of Projection operation

Illustration of Projection Operation

To illustrate projection operation consider the relation STAFF, with the attributes Staff number, Name, Gender, Date of birth, and Salary.

Query 1: Produce the list of salaries for all staff showing only the Name and salary detail. Relational algebra expression: $\Pi_{\text{Name.salary}}(\text{staff})$

Output for the Query 1

Name	Salary
Raghavan	15,000
Raghu	12,000
Babu	12,500
Kingsly	10,000

Query 2: Give the name and Date of birth of the all the staff in the STAFF relation.

Relational algebra expression for query 2: $\Pi_{\text{Name, date of birth}}(\text{staff})$

Name	Date of birth
Raghavan	1-5-76
Raghu	1-5-77
Babu	1-6-76
Kingsly	1-8-78

3.7.2 Rename operation (ρ)

The rename operator returns an existing relation under a new name. $\rho_A(B)$ is the relation B with its name changed to A. The results of operation in the relational algebra do not have names. It is often useful to name such results for use in further expressions later on. The rename operator can be used to name the result of relational algebra operation.

Example of Rename Operation

Consider the relation BATSMAN with the attributes name, nation, and BA.

BATSMAN		
Name	Nation	BA
Sachin Tendulkar	India	45.5
Brian Lara	West Indies	43.5
Inzamamulhaq	Pakistan	42.5

The attributes of the relation BATSMAN can be renamed as name, nation and batting average as name, nation, batting average (BATSMAN) so that the relation BATSMAN after rename operation as shown later.

BATSMAN		
Name	Nation	Batting average
Sachin Tendulkar	India	45.5
Brian Lara	West Indies	43.5
Inzamamulhaq	Pakistan	42.5

From the earlier operation it is clear that rename operation changes the schema of the database and it does not change the instance of the database.

Union Compatibility

In order to perform the Union, Intersection, and the Difference operations on two relations, the two relations should be union compatible. Two relations are union compatible if they have same number of attributes and belong to the same domain. Mathematically UNION COMPATIBILITY it is given as:

Let $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_n)$ be the two relations. The relation R has the attributes A_1, A_2, \dots, A_n and the relation S has the attributes B_1, B_2, \dots, B_n . The two relations R and S are union compatible if $\text{dom}(A_i) = \text{dom}(B_i)$ for $i = 1$ to n .

3.7.3 Union Operation

The union of two relations R and S defines a relation that contains all the tuples of R or S or both R and S, duplicate tuples being eliminated.

Relational Algebra Expression

The union of two relations R and S are denoted by $R \cup S$. $R \cup S$ is pictorially represented in the Fig. 3.4.

Illustration of UNION Operation

To illustrate the UNION operation consider the two relations Customer 1 and Customer 2 with the attributes Name and city.

Customer 1		Customer 2	
Name	City	Name	City
Anand	Coimbatore	Gopu	Tirunelveli
Aravind	Chennai	Balu	Kumbakonam
Gopu	Tirunelveli	Rahu	Chidambaram
Helan	Palayankottai	Helan	Palayamkottai

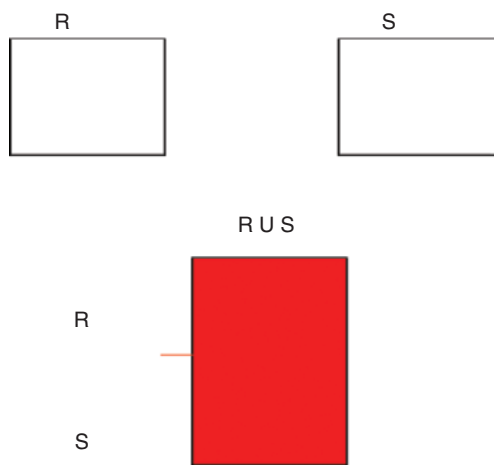


Fig. 3.4. Union of two relations R and S

Example

Query Determine Customer 1 \cup Customer 2
Result of Customer 1 \cup Customer 2

Customer 1 \cup Customer 2	
Name	City
Anand	Coimbatore
Aravind	Chennai
Balu	Kumbakonam
Gopu	Tirunelveli
Rahu	Chidambaram
Helan	Palayamkottai

3.7.4 Intersection Operation

The intersection operation defines a relation consisting of the set of all tuples that are in both R and S.

Relational Algebra Expression

The intersection of two relations R and S is denoted by $R \cap S$.

Illustration of Intersection Operation

The intersection between the two relations R and S is pictorially shown in Fig.3.5.

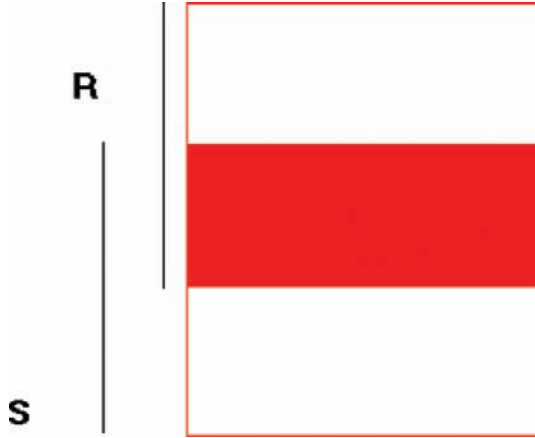


Fig. 3.5. Intersection of two relations R and S

Example

Find the intersection of Customer 1 with Customer 2 in the following table.

Customer 1 \cap Customer 2	
Name	City
Gopu	Tirunelveli
Helan	Palayamkottai

3.7.5 Difference Operation

The set difference operation defines a relation consisting of the tuples that are in relation R but not in S.

Relational Algebra Expression

The difference between two relations R and S is denoted by **$R - S$** .

Illustration of Difference Operation

The difference between two relations R and S is pictorially shown in Fig. 3.6.

Example

Compute $R - S$ for the relation shown in the following table.



Fig. 3.6. Difference between two relations R and S

Customer 1 – Customer 2	
Name	City
Anand	Coimbatore
Aravind	Chennai

3.7.6 Division Operation

The division of the relation R by the relation S is denoted by $R \div S$, where $R \div S$ is given by:

$$R \div S = \Pi_{R-S(r)} - \Pi_{R-S}((\Pi_{R-S(r)} \times s) - r)$$

To illustrate division operations consider two relations STUDENT and MARK. The STUDENT relation has the attributes Student Name and the mark in particular subject say mathematics. The MARK relation consists of only one column mark and only one row.

Student		Mark
Name	Mark	Mark
Arul	97	100
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

Case (1)

If we divide the STUDENT relation by the MARK relation, the resultant relation is shown as:

Case (2)

Now modify the relation MARK that is change the mark to be 98. So that the entry in the MARK relation is modified as 98.

<u>Answer</u>		
<u>Name</u>		
Banu		
Dinesh		

<u>Student</u>		<u>Mark</u>
<u>Name</u>	<u>Mark</u>	<u>Mark</u>
Arul	97	98
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

If we divide the relation STUDENT by MARK relation then the resultant relation is given by ANSWER

<u>Answer</u>
<u>Name</u>
Christi
Lakshmi

Case (3)

Now the MARK relation is modified in such a way that the entry in the MARK relation is 99. If we divide the STUDENT relation with the MARK relation, the result is NULL. Because there is no student in the STUDENT relation with the mark 99.

<u>Student</u>		<u>Mark</u>
<u>Name</u>	<u>Mark</u>	<u>Mark</u>
Arul	97	99
Banu	100	
Christi	98	
Dinesh	100	
Krishna	95	
Ravi	95	
Lakshmi	98	

The division of the STUDENT relation with the MARK relation is given by the ANSWER relation.

The division operation extracts records and fields from one table on the basis of data in the second table.

<u>Answer</u>
<u>Name</u>
NULL

3.7.7 Cartesian Product Operation

The Cartesian product operation defines a relation that is the concatenation of every tuples of relation R with every tuples of relation S. The result of Cartesian product contains all attributes from both relations R and S.

Relational Algebra Symbol for Cartesian Product:

The Cartesian product between the two relations R and S is denoted by **$R \times S$** .

Note: If there are n1 tuples in relation R and n2 tuples in S, then the number of tuples in $R \times S$ is $n1 * n2$.

Example

If there are 5 tuples in relation “R” and 2 tuples in relation “S” then the number of tuples in $R \times S$ is $5 * 2 = 10$.

Illustration of Cartesian Product

To illustrate Cartesian product operation, consider two relations R and S as given later:

R	S
a	1
	2
b	3

Determine $R \times S$:

R	S
a	1
a	2
a	3
b	1
b	2
b	3

Note:

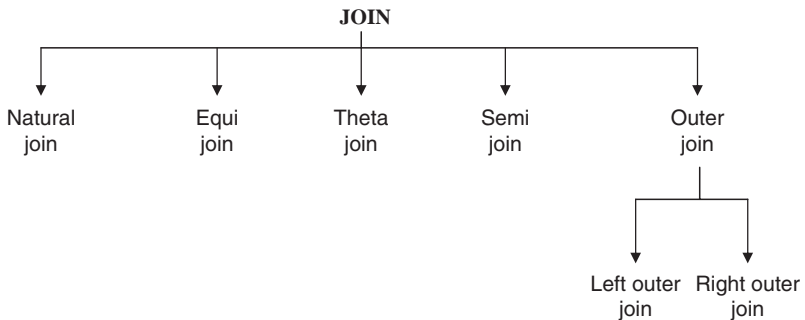
No. of tuples in $R \times S = 2 * 3 = 6$

No. of attributes in $R \times S = 2$

3.7.8 Join Operations

Join operation combines two relations to form a new relation. The tables should be joined based on a common column. The common column should be compatible in terms of domain.

Types of Join Operation



Natural Join

The natural join performs an equi join of the two relations R and S over all common attributes. One occurrence of each common attribute is eliminated from the result. In other words a natural join will remove duplicate attribute. In most systems a natural join will require that the attributes have the same name to identity the attributes to be used in the join. This may require a renaming mechanism. Even if the attributes do not have same name, we can perform the natural join provided that the attributes should be of same domain.

Input: Two relations (tables) R and S
Notation: $R \bowtie S$
Purpose: Relate rows from second table and

- Enforce equality on all column attributes
- Eliminate one copy of common attribute
- * Short hand for $\prod_{L}(R \times S)$:
 - L is the union of all attributes from R and S with duplicate removed
 - P equates all attributes common to R and S

Example of Natural Join Operation

Consider two relations EMPLOYEE and DEPARTMENT. Let the common attribute to the two relations be DEPTNUMBER. The two relations are shown later:

It is worth to note that Natural join operation is associative. (i.e.,) If R, S, and T are three relations then

$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$

Employee				Department	
Employee ID	Designation	Dept Number		Dept name	Dept Number
C100	Lecturer	E1	\bowtie	Electrical	E1
C101	Assistant Professor	E2		Computer	C1
C102	Professor	C1			

Employee		\bowtie	Department	
Employee ID	Designation	Dept Number	Dept name	
C100	Lecturer	E1	Electrical	
C102	Professor	C1	Computer	

Equi Join

A special case of condition joins where the condition C contains only equality.

Example of Equi Join

Given the two relations STAFF and DEPT, produce a list of staff and the departments they work in.

STAFF			DEPT	
Staff No	Job	Dept	Dept	Name
1	salesman	100	100	marketing
2	draftsman	101	101	civil

Answer for the earlier query is equi-join of STAFF and DEPT:

STAFF EQUI JOIN DEPARTMENT				
Staff No	Job	dept	dept	Name
1	salesman	100	100	marketing
2	draftsman	101	101	civil

Theta Join

A conditional join in which we impose condition other than equality condition. If equality condition is imposed then theta join become equi join. The symbol θ stands for the comparison operator which could be $>$, $<$, $>=$, $<=$.

Expression of Theta Join

$$\sigma_{\theta}(R \times S)$$

Illustration of Theta Join

To illustrate theta join consider two relations FRIENDS and OTHERS with the attributes Name and age.

FRIENDS		OTHERS	
Name	Age	Alias	Size
Joe	4	Bob	8
Sam	9	Gim	10
Sue	10		

Result of theta join

Name	Age	Alias	Size
Joe	4	Bob	8
Sam	9	Gim	10
Sue	10		

Outer Join

In outer join, matched pairs are retained unmatched values in other tables are left null.

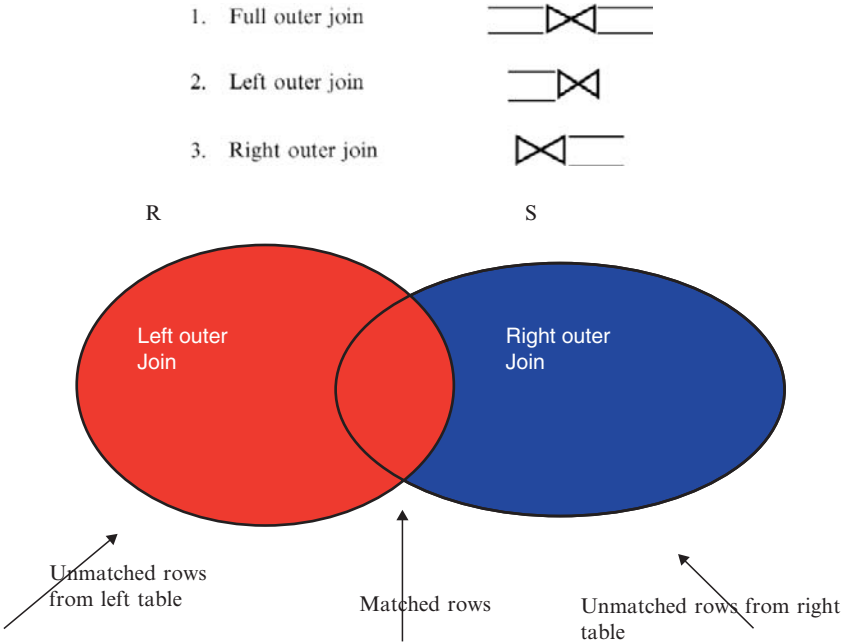


Fig. 3.7. Representation of left and right outer join

Types of Outer Join


The pictorial representation of the left and the right outer join of two relations R and S are shown in Fig. 3.7:

1. *Left Outer Join.* Left outer joins is a join in which tuples from R that do not have matching values in the common column of S are also included in the result relation.
2. *Right Outer Join.* Right outer join is a join in which tuples from S that do not have matching values in the common column of R are also included in the result relation.
3. *Full Outer Join.* Full outer join is a join in which tuples from R that do not have matching values in the common columns of S still appear and tuples in S that do not have matching values in the common columns of R still appear in the resulting relation.

Example of Full Outer Left Outer and Right Outer Join


Consider two relations PEOPLE and MENU determine the full outer, left outer, and right outer join.

Table 3.1. Left outer join of PEOPLE and MENU relation

PEOPLE  **PEOPLE.Food = MENU.Food** **MENU**

Name	Age	People.Food	Menu.Food	Day
Raja	21	Idly	Idly	Tuesday
Ravi	22	Dosa	Dosa	Wednesday
Rani	20	Pizza	NULL	NULL
Devi	21	Pongal	Pongal	Monday

Table 3.2. Right outer join of PEOPLE and MENU relation

PEOPLE  **PEOPLE.Food = Menu.Food** **MENU**

Name	Age	People.Food	Menu.Food	Day
Devi	21	Pongal	Pongal	Monday
Raja	21	Idly	Idly	Tuesday
Ravi	22	Dosa	Dosa	Wednesday
NULL	NULL	NULL	Fried rice	Thursday
NULL	NULL	NULL	Parotta	Friday


PEOPLE

Name	Age	Food
Raja	21	Idly
Ravi	22	Dosa
Rani	20	Pizza
Devi	21	Pongal

MENU

Food	Day
Pongal	Monday
Idly	Tuesday
Dosa	Wednesday
Fried rice	Thursday
Parotta	Friday

1. The left outer join of PEOPLE and MENU on Food is represented as

PEOPLE  **PEOPLE.Food = MENU.Food** **MENU**. The result of the left outer join is shown in Table 3.1.

From this table, it is to be noted that all the tuples from the left table (in our case it is PEOPLE relation) appears in the result. If there is any unmatched value then a NULL value is returned.

2. The right outer join of PEOPLE and MENU on Food is represented in

the relational algebra as **PEOPLE**  **PEOPLE.Food = Menu.Food** **MENU**. The result of the right outer join is shown in Table 3.2.

Table 3.3. Full outer join of PEOPLE and MENU relation

Name	Age	People.Food	Menu.Food	Day
Raja	21	Idly	Idly	Tuesday
Ravi	22	Dosa	Dosa	Wednesday
Rani	20	Pizza	NULL	NULL
Devi	21	Pongal	Pongal	Monday
NULL	NULL	NULL	Fried rice	Thursday
NULL	NULL	NULL	Parotta	Friday

From this table, it is clear that all tuples from the right-hand side relation (in our case the right hand relation is MENU) appears in the result.

3. The full outer join of PEOPLE and MENU on Food is represented in the relational algebra as $PEOPLE \bowtie_{PEOPLE.Food = MENU.Food} MENU$. The result of the full outer join is shown in Table 3.3.

From this table, it is clear that tuples from both the PEOPLE and the MENU relation appears in the result.

Semi-Join

The semi-join of a relation R, defined over the set of attributes A, by relation S, defined over the set of attributes B, is the subset of the tuples of R that participate in the join of R with S. The advantage of semi-join is that it decreases the number of tuples that need to be handled to form the join. In centralized database system, this is important because it usually results in a decreased number of secondary storage accesses by making better use of the memory. It is even more important in distributed databases, since it usually reduces the amount of data that needs to be transmitted between sites in order to evaluate a query.

Expression for Semi-Join

$$R \ltimes_F S = \prod_A (R \ltimes_F S) \quad \text{where } F \text{ is the predicate.}$$

Example of Semi-Join

In order to understand semi-join consider two relations EMPLOYEE and PAY

EMPLOYEE			PAY	
Employee Number	Employee Name	Designation	Designation	Salary
E1	Rajan	Programmer	Programmer	25,000
E2	Krishnan	System Analyst	Consultant	70,000
E3	Devi	Database Administrator		
E4	Vidhya	Consultant		

The semi-join of EMPLOYEE with the PAY is denoted by:
 $EMPLOYEE \bowtie_{EMPLOYEE.DESIGNATION=PAY.DESIGNATION} PAY$. The result of this semi-join is given later:

Employee Number	Employee Name	Designation
E1	Rajan	Programmer
E4	Vidhya	Consultant

From the result of the semi-join it is clear that a semi-join is half of a join: the rows of one table that match with at least one row of another table. Only the rows of the first table appear in the result.

3.8 Advantages of Relational Algebra

The relational algebra has solid mathematical background. The mathematical background of relational algebra is the basis of many interesting developments and theorems. If we have two expressions for the same operation and if the expressions are proved to be equivalent, then a query optimizer can automatically substitute the more efficient form. Moreover, the relational algebra is a high level language which talks in terms of properties of sets of tuples and not in terms of for-loops.

3.9 Limitations of Relational Algebra

The relational algebra cannot do arithmetic. For example, if we want to know the price of 10l of petrol, by assuming a 10% increase in the price of the petrol, which cannot be done using relational algebra.

The relational algebra cannot sort or print results in various formats. For example we want to arrange the product name in the increasing order of their price. It cannot be done using relational algebra.

Relational algebra cannot perform aggregates. For example we want to know how many staff are working in a particular department. This query cannot be performed using relational algebra.

The relational algebra cannot modify the database. For example we want to increase the salary of all employees by 10%. This cannot be done using relational algebra.

The relational algebra cannot compute “transitive closure.” In order to understand the term transitive closure consider the relation RELATIONSHIP, which describes the relationship between persons.

Consider the query, Find all direct and indirect relatives of Gopal? It is not possible to express such kind of query in relational algebra. Here transitive means, if the person A is related to the person B and if the person B is related to the person C means indirectly the person A is related to the person C. But relational algebra cannot express the transitive closure.

RELATIONSHIP		
Person1	Person2	Relationship
Gopal	Nandini	Father
Siva	Raja	Brother
Gopal	Neena	Husband
Deepa	Lakshmi	Sister

3.10 Relational Calculus

The purpose of relational calculus is to provide a formal basis for defining declarative query languages appropriate for relational databases. Relational Calculus comes in two flavors (1) Tuple Relational Calculus (TRC) and (2) Domain Relational Calculus (DRC). The basic difference between relational algebra and relational calculus is that the former gives the procedure of how to evaluate the query whereas the latter gives only the query without giving the procedure of how to evaluate the query:

- The variable in tuple relational calculus formulae range over tuples.
- The variable in domain relational calculus formulae range over individual values in the domains of the attributes of the relations.
- Relational calculus is nonoperational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)

Relational Calculus and Relational Algebra:

The major difference between relational calculus and relational algebra is summarized later:

- A relational calculus query specifies *what* information is retrieved
- A relational algebra query specifies *how* information is retrieved

3.10.1 Tuple Relational Calculus

Tuple relational calculus is a logical language with variables ranging over tuples. The general form of tuple relational calculus is given by:

$$\{\langle \text{tuple variable list} \rangle \mid \langle \text{conditions} \rangle\}$$

$$\{t \mid \text{COND}(t)\}$$

Here t is the tuple variable, which stands for tuples of relation. $\text{COND}(t)$ is a formula that describes t . The meaning of the earlier expression is to return all tuples T that satisfy the condition COND :

- $T/R(T)$ means return all tuples T such that T is a tuple in relation R .
- For example, $\{T.\text{name}/\text{FACULTY}(T)\}$ means return all the names of faculty in the relation FACULTY .
- $\{T.\text{name}/\text{FACULTY}(T) \text{ AND } T.\text{deptid} = \text{'EEE'}\}$ means return the value of the name of the faculty who are working in EEE department.

Quantifiers

Quantifiers are words that refer to quantities such as “some” or “all” and tell for how many elements a given predicate is true. A predicate is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables. Quantifiers can be broadly classified into two types (1) Universal Quantifier and (2) Existential Quantifier.

Existential Quantifier

symbol: \exists

$$\exists T \in \text{Cond}(R)$$

It will succeed if the condition succeeds for at least one tuple in T .

- $(\exists t)(C)$ – Existential operator – True if there exists a tuple t such that the condition(s) C are true.
- Example of existential quantifier is $\exists(m)$ such that $m^2 = m$. (i.e., $m = 1$).

Universal Quantifier

symbol: \forall

- $(\forall t)(C)$ – Universal operator – True if C is true for every tuple t .
- Example of universal quantifier is $\forall(2), \sin^2(2) + \cos^2(2) = 1$.
The example refers to the fact that for all values of $2 \sin^2(2) + \cos^2(2) = 1$.

Free Variable

Any variable that is not bound by a quantifier is said to be free.

Bound Variable

Any variable which is bounded by universal or existential quantifier is called bound variable.

Example of *selection* operation in TRC:

1. To find details of all staff earning more than Rs. 10,000:

$$\{S \mid \text{Staff}(S) \wedge S.\text{salary} > 10000\}$$

Example of *projection* operation in TRC:

2. To find a particular attribute, such as salary, write:

$$\{S.\text{salary} \mid \text{Staff}(S) \wedge S.\text{salary} > 10000\}$$

Quantifier Example

Client(ID, fName, lName, Age)

Matches(Client1, Client2, Type)

- List the first and last names of clients that appear as client1 in a match of any type.

RAlg: $p(\text{fName}, \text{lName})(\text{Client} \mid (\text{ID} = \text{Client1}) \text{ Matches})$

RCalc: $\{c.\text{fName}, c.\text{lName} \mid \text{CLIENT}(c) \text{ AND } (\exists m)(\text{MATCHES}(m) \text{ AND } c.\text{ID} = m.\text{Client1})\}$

Joins in Relational Calculus

Consider the two relations Client and Matches as

Client(ID, fName, lName, Age)

Matches(Client1, Client2, Type)

- **List all information about clients and the corresponding matches that appear as client1 in a match of any type.**

The earlier query can be expressed both in Relational Algebra and Tuple relational Calculus as:

- **RAlg:** $\text{Client} \mid (\text{ID} = \text{Client1}) \text{ Matches}$

- **RCalc:** $\{c, m \mid \text{CLIENT}(c) \text{ AND } \text{MATCHES}(m) \text{ AND } c.\text{ID} = m.\text{Client1}\}$

3.10.2 Set Operators in Relational Calculus

The set operations like Union, Intersection, difference, and Cartesian Product can be expressed in Tuple Relational Calculus as:

Union

- $R1(A,B,C) \cup R2(A, B, C)$
- $\{r \mid R1(r) \text{ OR } R2(r)\}$

Intersection

- $R1(A,B,C) \cap R2(A, B, C)$
- $\{r \mid R1(r) \text{ AND } R2(r)\}$

Cartesian Product

- $R(A, B, C) \times S(D, E, F)$
- $\{r, s \mid R(r) \text{ AND } S(s)\}$ // same as join without the select condition

Subtraction

- $R1(A,B,C) - R2(A, B, C)$
- $\{r \mid R1(r) \text{ AND NOT } R2(r)\}$

Queries and Tuple Relational Calculus Expressions

Some of the queries and the corresponding relational calculus and their explanations are given later. Here we have given set of queries like SET 1, SET 2, and SET 3.

- Query set 1 deals with Railway Reservation Management
- Query set 2 deals with Library Database Management
- Query set 3 deals with Hostel Database Management

Query Set1: Query set 1 deals with railway reservation system.

Query 1: Find all the train details for the trains where starting place is “Chennai.”

Relational calculus expression: $\{t \mid t \in \text{train_details} \wedge \text{start place} = \text{“Chennai”}\}$

Explanation: Set of all tuples “t” that belong to the relation “train details” and also the starting place is “Chennai” is found by the query.

Query 2: Find all train names whose destination is “Salem.”

Relational calculus expression

$\{t \mid \exists s \in \text{train_details} (t[\text{train_no}] = s[\text{train_no}] \wedge s[\text{destination}] = \text{“Salem”})\}$

Explanation: There exist a tuple “t” in the relation “r” such that the predicate is true.

The set of all tuples “t” such that, there exists a tuple “s” in relation train details for which the values of “t” and “s” for the train_no attribute are equal and the value of “s” for the destination is “Salem.”

Query 3: Find the names of all passengers who have canceled the ticket and whose age is above 40.

Relational calculus expression $\{t \mid \exists s \in \text{cancel} (t [\text{train_no}] = s [\text{train_no}] \wedge \exists u \in \text{passen_details} (u [\text{name}] = s [\text{name}] \wedge u [\text{age}] > 40))\}$

Explanation: Set of all passenger names tuples for which the age is above 40 and the ticket is canceled. The tuple variable “s” ensures that the passenger canceled the ticket. The tuple “u” is restricted to having the same passenger name as “s.”

Query 4: List the train numbers of all trains which has no cancelation and only reservation.

Relational Calculus Expression

Relational calculus expression $\{t \mid \exists s \in \text{reserve} (t [\text{train_no}] = s [\text{train_no}]) \wedge \neg \exists u \in \text{cancel} (t [\text{train_no}] = u [\text{train_no}])\}$

Explanation: Set of all tuples “t” such that there exists a tuple “s” that belongs to reserve such that the train_no attribute is equal for “t” and “s” and there exists a tuple “u” that belongs to cancel where the values of “t” and “u” for the train_no attribute is the same.

Query 5: List all female passengers name who are traveling by the train “Blue Mountain.”

Relational Calculus Expression

Relational calculus expression $\{t \mid \exists s \in \text{passen_details} (t [\text{p_name}] = s [\text{p_name}] \wedge s [\text{sex}] = \text{“female”} \wedge s [\text{train_name}] = \text{“Blue mountain”})\}$.

Explanation: Set of all tuples “t” such that there exists a tuple “s” that belongs to passen_details for which the values of “t” and “s” for the p_name attribute is same and the sex attribute = “female” and train_name attribute = “Blue mountain.”

Query Set 2: Query set 2 deals with frequent queries in library database management.

Query 1: Find the acc.no/- for each book whose price >1000.

Relational Calculus Expression

Relational calculus expression $\{t \mid \exists s \in \text{book} (t [\text{acc_no/-}] = s [\text{acc_no/-}] \wedge s [\text{price}] > 1000)\}$

Explanation: The set of all tuples “t” such that there exists a tuple “s” in relation book for which the values “t” and “s” for the acc_no/- attribute are equal and the value of the s for the price attribute is greater than 1000.

Query 2: Find the name of all the students who have borrowed a book and price of those book is greater than 1000.

Relational Calculus Expression

$\{t \mid \exists s \in \text{books_borrowed} (t[\text{std_name}] = s[\text{std_name}] \wedge \exists u \in \text{book} (u[\text{acc_no/-}] = s[\text{acc_no/-}] \wedge u[\text{price}] > 1000))\}$

Explanation: The set of all tuples “t” such that there exists a tuple “s” in relation books.borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on book relation for which “u” and “s” for the acc_no/- attribute are equal and the value of “u” for the price attribute is greater than 1000.

Query 3: Find the name of the students who borrowed book, have book in his account or both.

Relational Calculus Expression

$\{t \mid \exists s \in \text{books_borrowed} (t[\text{stud_name}] = s[\text{std_name}]) \vee \exists u \in \text{books_remaining} (t[\text{std_name}] = su[\text{std_name}])\}$

Explanation: The set of all tuples “t” such that there exists a tuple “s” in relation books_ borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on books_remaining relation for which “u” and “s” for the stud_name attribute are equal.

Query 4: Find only those students’ names who are having both the books in their account as well as the books borrowed from their account.

Relational Calculus Expression

$\{t \mid \exists s \in \text{books_borrowed} (t[\text{std_name}] = s[\text{std_name}]) \wedge \exists u \in \text{books_remaining} (t[\text{std_name}] = s[\text{std_name}])\}$

Explanation: The set of all tuples “t” such that there exists a tuple “s” such that in relation books.borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on books_remaining relation for which “u” and “s” for the student name attribute are equal.

Query 5: Query that uses implication symbol $p \Rightarrow q$ find all students belongs to EEE department who borrowed the books.

Relational Calculus Expression

$$\{t \mid \exists r \in \text{books_borrowed} (r[\text{std_name}] = t[\text{std_name}] \wedge (\forall u \in \text{department} (u[\text{dept_name}] = \text{"EEE"})))\} \Rightarrow \{t \mid \exists r \in \text{books_borrowed} (r[\text{std_name}] = t[\text{std_name}] \wedge \exists w \in \text{student} (w[\text{roll_no}/-] = r[\text{roll_no}/-] \wedge w[\text{dept_name}] = u[\text{dept_name}]))\}$$

Explanation: The set of all tuples “t” such that there exists a tuple “s” such that in relation books_borrowed for which the values “t” and “s” for the student name attribute are equal and “u” tuple variable on department relation must be equal to “EEE.” And this must be equal to the set of all tuple “t” such that there exists a tuple “r” in relation books_borrowed for which the values “r” and “t” for the student name attribute are equal and “w” the variable on relation student for which “w” and “r” are equal for the roll.no/- attribute and “w” and “u” are equal for the dept_name.

Query Set 3: Query set 3 deals with hostel management.

Query 1: Find all the students id who are staying in hostel.

Tuple Relational Calculus Expression

$$\{t \mid \exists s \in \text{student_detail} (t[\text{roll no}] = s[\text{rollno}])\}$$

Explanation: Here t is the set of tuples in the relation student_detail such that there exists a tuple s which consists of students ID who are staying in the hostel.

Query 2: Find all the details of the student who are belonging to EEE branch.

Tuple Relational Calculus Expression

$$\{t \mid t \in \text{student_detail} \wedge t[\text{course name}] = \text{"EEE"}\}$$

Explanation: Here t is the set of tuples in the relation student_detail such that it consists of all the details of the student who are belonging to the “EEE” branch.

Query 3: Find all the third semester BE-EEE students.

Tuple Relational Calculus Expression

$$\{t \mid t \in \text{student_detail} \wedge t[\text{coursename}] = \text{"EEE"} \wedge t[\text{semester}] = 3\}$$

Explanation: Here t is the set of tuples in the relation student_detail such that it consists of all the details of the student who belongs to the third semester BE-EEE branch.

Query 4: Find all the lecturers name belonging to the EEE department.

Tuple Relational Calculus Expression

$\{t \mid \exists s \in \text{staff_detail } (t[\text{staffname}] = s[\text{staffname}])\}$

Explanation: Here t is the set of tuples in the relation `staff_detail` and there exists a tuple s which consists of lecturers name who belongs to the “EEE” department.

Query 5: Find all the staff who are having leisure period at third hour on Monday.

Tuple Relational Calculus Expression

$\{t \mid \exists s \in \text{staff_detail } (t[\text{staffname}] = s[\text{staffname}] \wedge \exists u \in \text{lecturerschedule_monday } (s[\text{staffid}] = u[\text{staffid}] \wedge u[\text{third hour}] = \text{“EEE”}))\}$

Explanation: Here t is the set of tuples in the relation `staff_detail` and there exists a tuple s which consists of staff name who are all having leisure period at third hour on Monday for every week.

Safety of Expression

It is possible to write tuple calculus expressions that generate infinite relations. For example $\{t/\sim t \in R\}$ results in an infinite relation if the domain of any attribute of relation R is infinite. To guard against the problem, we restrict the set of allowable expressions to safe expressions. An expression $\{t/P(t)\}$ in the tuple relational calculus is safe if every component of t appears in one of the relations, tuples, or constants that appear in P (Here P refers to Predicate or condition).

Limitations of TRC

TRC cannot express queries involving:

- Aggregations.
- Groupings.
- Orderings.

3.11 Domain Relational Calculus (DRC)

Domain relational calculus is a nonprocedural query language equivalent in power to tuple relational calculus. In domain relational calculus each query is an expression of the form:

$\{ \langle X_1, X_2, \dots, X_n \rangle / P(X_1, X_2, \dots, X_n) \}$ where

- X_1, X_2, \dots, X_n represent domain variables
- P represents a formula similar to that of the predicate calculus.

Domain variable: A domain variable is a variable whose value is drawn from the domain of an attribute.

3.11.1 Queries in Domain Relational Calculus:

Consider the ER diagram:



<u>STUDENT</u>	<u>CLASS</u>	<u>TAKES</u>
ID Name Address	CID CNAME location	ID CID GRADE
123 Anbu		
456 Anu		

Query 1:

Get the details of all students?
This query can be expressed in DRC as
 $\{ \langle I, n, a \rangle / \langle I, n, a \rangle \in \text{STUDENT} \}$

Query 2: (Selection operation)

Find the details of the student whose roll no (or) ID is 123?
 $\{ \langle 123, n, a \rangle / \langle 123, n, a \rangle \in \text{STUDENT} \}$
(OR)
 $\{ \langle I, n, a \rangle / \langle I, n, a \rangle \in \text{STUDENT} \wedge I = 123 \}$
(Here I, n, a are referred to as domain variables)

Query 3: (Projection)

Find the name of the student whose roll no. is 456?
 $\{ \langle I \rangle / \langle I, n, a \rangle \in \text{STUDENT} \wedge I = 456 \}$

3.11.2 Queries and Domain Relational Calculus Expressions

Some of the queries and the corresponding relational calculus and their explanations are given later. Here we have given set of queries like SET 1, SET 2, and SET 3:

- Query set 1 deals with Railway Reservation Management
- Query set 2 deals with Library Database Management
- Query set 3 deals with Department Database Management

Query Set 1: Query set 1 deals with railway reservation system.

Query 1: List the details of the passengers traveling by the train “Intercity express.”

Domain Relational Calculus Expression

$\{ \langle \text{name, age, sex, train_no, "blue mountain"} \rangle \mid \langle \text{name, age, sex, train_no, train_name} \rangle \in \text{passen_details} \}$

Explanation: The attributes of the passen_details are listed where the train_name attribute = “Intercity express.”

Query 2: Select names of passengers whose sex = “female” and age > 20.

Domain Relational Calculus Expression

$\{ \langle \text{p_name} \rangle \mid \exists \text{p_age, p_sex, p_trainno. } (\langle \text{p_name, p_age, p_sex, p_trainno} \rangle \in \text{passen_details} \wedge \text{p_sex} = \text{"female"} \wedge \text{p_age} > 20) \}$

Explanation: Lists the names of passengers from the relation passenger_details where there are two constraints which are sex = female and age > 20.

Query 3: Find all the names of passengers who have “Salem” as start place and find their train names.

Domain Relational Calculus Expression

$\{ \langle \text{p_name, train_name} \rangle \mid \exists \text{p_name} > \text{p_name, p_age, p_trainno, } (\langle \text{p_name, p_age, p_sex, p_train_no, p_trainname} \rangle \in \text{passen_details} \wedge \exists \text{t_start, t_dest, t_route, t_no } (\langle \text{t_name, t_no, t_start, t_dest, t_route} \rangle \in \text{train_details} \wedge \text{t_start} = \text{"salem"})) \}$

Explanation: Two relations – passen_details and train_details are involved in this query. The train names and the passenger names whose start place = Salem is displayed.

Query 4: Find all train names which has reservation and no cancellation.

Domain Relational Calculus Expression

$\{ \langle \text{t_name} \rangle \mid \exists \text{t_name, p_name, p_source, p_dest} (\langle \text{t_name, t_no, p_name, p_source, p_dest} \rangle \in \text{reserve} \wedge \exists \text{ticket_no, t_no, s_no, p_name } (\langle \text{t_name, t_no, tick_no, p_name, s_no} \rangle \in \text{cancel})) \}$

Explanation: The reserve and cancel relations are involved here. The train names which satisfies both the conditions are displayed.

Query 5: Find names of all trains whose destination is “CHENNAI” and source is “COIMBATORE.”

Domain Relational Calculus Expression

$\{ \langle \text{t_name} \rangle \mid \exists \text{t_no, t_start, t_dest, t_route } (\langle \text{t_name, t_no, t_start, t_dest, t_route} \rangle \in \text{train_details} \wedge \text{t_source} = \text{"coimbatore"} \wedge \text{t_dest} = \text{"chennai"}) \}$

Explanation: The name of the trains that start from Coimbatore and reach Chennai are listed from the relations train_details.

Query Set 2:

Query set 2 deals with Library Management.

Query 1: Find the student name, roll_no. for those belongs to “EEE” department.

Domain Relational Calculus Expression

$$\{ \langle \text{std_name}, \text{std_roll_no} \rangle \mid \text{dept_name}(\langle \text{std_name}, \text{roll_no}, \text{depart_name} \rangle \in \text{student} \wedge \text{depart_name} = \text{“EEE”}) \}$$

Explanation: Student relation is involved in this. Std_name, roll_no are the attribute belongs to the student relation whose department name is “EEE.”

Query 2: Find the acc_no, books_cal_no, and author name for the books of price >120.

Domain Relational Calculus Expression

$$\{ \langle \text{acc_no}, \text{book_call_no}, \text{author_name} \rangle \mid \exists \text{book_name}, \text{price}(\langle \text{book_name}, \text{acc_no}, \text{call_no}, \text{author_name}, \text{price} \rangle \in \text{books} \wedge \text{price} > 120) \}$$

Explanation: Books relation is involved here. In this expression acc_no, book_call_no, and author name are selected for the book for which the price is greater than 120.

Query 3: Find the roll_no of all the students who have borrowed book from library and find the no/- of books they borrowed and that books belongs to “EEE” department.

Domain Relational Calculus Expression

$$\{ \langle \text{roll_no/-} \rangle \mid \exists \text{std_name}, \text{book_acc_no}(\langle \text{std_name}, \text{roll_no}, \text{book_acc_no}, \text{number of books borrowed} \rangle \in \text{books_borrowed} \wedge \exists \text{name}, \text{dept_name}(\langle \text{name}, \text{roll_no}, \text{dept_name} \rangle \in \text{student} \wedge \text{dept_name} = \text{“EEE”})) \}$$

Explanation: Here two relations are involved (1) books_borrowed and (2) student. The roll_no/- of the students who borrowed “EEE” department book involves both the earlier relations. Roll_no/- are selected from the both the relation of the student who borrowed book from library which belongs to “EEE” department.

Query 4: Find the std_name and their depart_name who have borrowed a book which is less than 2 in number.

Domain Relational Calculus Expression

$$\{ \langle \text{dept_name}, \text{name} \rangle \mid \exists \text{ roll_no} / -, \text{book_acc_no} / -, \text{no_of_books_borrowed} \langle \text{roll_no} / -, \text{book_acc_no} / -, \text{no_of_books_borrowed}, \text{std_name} \rangle \in \text{books_borrowed} \wedge \text{no_of_books_borrowed} < 2 \wedge \exists \text{ roll_no} / - (\text{roll_no} / -, \text{name}, \text{dept_name}) \in \text{student} \} \}$$

Explanation: Here two relations are involved (1) books_borrowed and (2) student. For student name the relation involved is books_borrowed and for depart_name the relation involved is student and the constraint is no/- of books_borrowed is less than two.

Query 5: Find the name of all the students who have borrowed, having books in his account or both in the department EEE.

Domain Relational Calculus Expression

$$\{ \langle \text{name} \rangle \mid \exists \text{ roll_no} / -, \text{book_acc_no} / -, \text{no_of_books_borrowed} \langle \text{name}, \text{roll_no} / -, \text{book_acc_no} / -, \text{no_of_books_borrowed} \rangle \in \text{books_borrowed} \wedge \exists \text{ roll_no} / -, \text{depart_name} \langle \text{name}, \text{roll_no} / -, \text{dept_name} \rangle \in \text{student} \wedge \text{dept_name} = \text{"eee"} \} \vee \exists \text{ roll_no} / -, \text{no_of_books_remaining} \langle \text{name}, \text{roll_no} / -, \text{no_of_books_remaining} \rangle \in \text{books_remaining} \wedge \exists \text{ roll_no} / -, \text{dept_name} \langle \text{name}, \text{roll_no} / -, \text{dept_name} \rangle \in \text{student} \wedge \text{dept_name} = \text{"EEE"} \} \}$$

Explanation: Here three relations are involved (1) books_remaining, (2) books_borrowed, and (3) student. Name is an attribute belonging to books_borrowed and books_remaining relations, dept_name belongs to student relation. The student borrowed books or having books in his account or both which belongs to “EEE” department is selected.

Query Set 3: Query set 2 deals with Department Database Management system.

Query 1: Find all the student name belongs to fifth sem ECE branch.

Domain Relational Calculus Expression

$$\{ \langle \text{stud_name} \rangle \mid \exists \langle \text{r}, \text{cn}, \text{s}, \text{h}, \text{dob}, \text{pn}, \text{b} \rangle \in \text{student_detail} \wedge \text{s} = \text{"V"} \wedge \text{b} = \text{"ECE"} \}$$

Explanation: Students name domain is formed from relation V semester “ECE” branch.

Domain variables used:

r - roll no.; cn - course name; s - semester; h - hosteller
dob - date of birth; pn - phone no.; b - branch name

Query 2: Find all the details of students belonging to CSE branch.

Domain Relational Calculus Expression

$$\{ \langle \text{sn}, \text{r}, \text{cn}, \text{s}, \text{h}, \text{dob}, \text{pn}, \text{b} \rangle \mid \langle \text{sn}, \text{r}, \text{cn}, \text{s}, \text{h}, \text{dob}, \text{pn}, \text{b} \rangle \in \text{student_detail} \wedge \text{b} = \text{"CSE"} \}$$

Explanation: All domain variables included from student-detail table which consists of all details about students belonging to the CSE branch.

Query 3: Find all the students id whose date of birth is above 1985.

Domain Relational Calculus Expression

$$\{ \langle \text{r} \rangle \mid \exists \text{ sn}, \text{cn}, \text{s}, \text{h}, \text{dob}, \text{pn}, \text{b} \ (\langle \text{r}, \text{sn}, \text{cn}, \text{b}, \text{s}, \text{h}, \text{dob}, \text{pn} \rangle \in \text{student_detail} \mid \wedge \text{dob} > \text{"1985"}) \}$$

Explanation: Domain variable r (roll no) is included from student_detail relation, which consists of students ID whose date of birth is above 1985.

Query 4: Find all the lecturers id belonging to production dept.

Domain Relational Calculus Expression

$$\{ \langle \text{sid} \rangle \mid \exists \text{ sn}, \text{dob}, \text{desg}, \text{y}, \text{foi}, \text{e}, \text{d} \mid \langle \text{sid}, \text{sn}, \text{dob}, \text{desg}, \text{y}, \text{foi}, \text{e}, \text{d} \rangle \in \text{staff_detail} \wedge \text{d} = \text{"prod"} \}$$

Explanation: Domain variables from staff_detail:

sid - staff_ID; dob - date of birth; sn - staff name; desg - designation
y - year since serving; foi - field of interest; e - email id; d - department

The sid (staff id) from staff detail belonging to production department.

Query 5: Find all the lecturers' names who are having fifth period as leisure period on Friday.

Domain Relational Calculus Expression

$$\{ \langle \text{sn} \rangle \mid \exists \text{ sed}, \text{dob}, \text{desg}, \text{y}, \text{foi}, \text{e}, \text{d} \mid \langle \text{sn}, \text{sid}, \text{dob}, \text{desg}, \text{y}, \text{foi}, \text{e}, \text{d} \rangle \in \text{staff_detail} \wedge \exists \langle \text{sid}, \text{i}, \text{ii}, \text{iii}, \text{iv}, \text{v}, \text{vi}, \text{vii} \rangle (\langle \text{sid}, \text{sn}, \text{i}, \text{ii}, \text{iii}, \text{iv}, \text{v}, \text{vi}, \text{vii} \rangle \in \text{rev_schedul_friday} \wedge \text{v} = \text{"free"}) \}$$

Explanation: Staff name domain variable from staff detail relation with fifth period as leisure which is checked using lecture schedule relation on Friday. Thus, in this, we have used two relations: staff detail and lecture schedule for Friday.

3.12 QBE

QBE stands for **Q**uery **B**y **E**xample. QBE uses a terminal display with attribute names as table headings for queries. This looks a little strange in textbooks, but people like it when they have worked with it for a while on a

terminal screen. It is very easy to list the entire schema, simply by scrolling information on the screen. QBE was developed originally by IBM in the 1970s to help users in their retrieval of data from a database. QBE represents a visual approach for accessing data in a database through the use of query templates. QBE can be considered as GUI (Graphical User Interface) based on domain calculus. QBE allows users to key in their input requests by filling in empty tables on the screen, and the system will also display its response in tabular form. QBE is user-friendly because the users are not required to formulate sentences for query requests with rigid query-language syntax. In QBE the request is entered in the form of tables whose skeletons are initially constructed by QBE.

Some of the QBE query template examples:

Example 1. Projection operation

In this template **P.** implies “Print.” The meaning is: Print the **PLAYER_ADDRESS** who belong to the country **INDIA**. To make a projection only put **P.** in any column of the projection. QBE will enforce uniqueness of projections automatically.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
P.			INDIA

Example 2. Selection operation

To make a selection, put quantifiers in the columns of the attributes in the question. To print a whole record, put **P.** in the column with the name of the record.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
P.			INDIA

The meaning is to print the **PLAYER_ADDRESS** who belong to the country **INDIA**.

Example 3. AND condition

To understand the AND condition consider the following template.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
P.		CHENNAI	INDIA

The meaning of the earlier template is: Print the **PLAYER_ADDRESS** who live in **INDIA** and belong to the city **CHENNAI**.

Example 4. OR condition

To understand the OR condition consider the following template:

PLAYER_ADDRESS	NAME	CITY	COUNTRY
	P.	CHENNAI	INDIA
	P.	DELHI	INDIA

The meaning of the earlier template is “Print the name of the Player who belongs to the country INDIA and city either CHENNAI or DELHI”.

Example 5. Query involving more than one table

Let us consider a query which involves data from more than one table. Let us consider two tables PLAYER_ADDRESS and PLAYER_RANK. Here we have two tables PLAYER_ADDRESS and PLAYER_RANK, the template meaning is: Print the name of the player who belong to the country INDIA and rank less than 50. The clue for understanding the query is the fact the variable_NAME is the same in all rows of the display.

PLAYER_ADDRESS	NAME	CITY	COUNTRY
	P._NAME		INDIA

PLAYER_RANK	NAME	RANK	COUNTRY
	P._NAME	<50	INDIA

Example 6. Comparison operation

Consider the EMPLOYEE table with the columns EMPLOYEE_ID, EMPLOYEE_NAME, SALARY, and MANAGER_ID. If one wants to know the name of the employees who make more money than their managers, it can be shown in QBE as:

EMPLOYEE	EMPLOYEE_ID	EMPLOYEE_NAME	SALARY	MANAGER_ID
		P._N	X _Y <_Y	–X

Example 7. Ordering of records

The records can be arranged either in the ascending order or in the descending order using the operator **AO.** and **DO.**, respectively.

- **AO.** implies arrange the records in ascending order.
- **DO.** implies arrange the records in descending order.

- **AO.ALL.** implies arrange the records in ascending order by preserving duplicate records.
- **DO.ALL.** implies arrange the records in descending order by preserving duplicate records.

Both **AO.** and **DO.** operators automatically eliminates duplicate responses. However, if one wishes to have all duplicate records displayed, an **ALL.** Operator must be added.

Consider the relation VEGETABLE which has three attributes VEGETABLENAME, QUANTITY, and PRICE.

VEGETABLE		
VEGETABLENAME	QUANTITY(in Kg)	PRICE(in Rs)
Brinjal	1	13
Potato	1	17
Ladies Finger	1	12
Carrot	1	16
Tomato	1	14

The QBE template to print the VEGETABLE in the increasing order of price is given later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
P.AO.			

The QBE template to print the VEGETABLE in the decreasing order of price is given later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
P.DO.			

Example 8. Retrieval using Negation

The symbol used for negation is +. For example print the quantity and price of the VEGETABLE that do not belong to Brinjal is given by:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
+	Brinjal	P.	P.

Condition Box:

The condition box is used to store logical conditions that are not easily expressed in the table skeleton. A condition box can be obtained by pressing a special function key.

Example 9. Retrieval using condition box:

For example, if we want to print the quantity and price of the VEGE-
TABLE, which is either Ladies Finger or Carrot, the condition box is used.

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
	VN	P.	P.

CONDITIONS
VN = Ladies Finger OR Carrot

Example 10. QBE Built-In Functions

QBE provides **MIN**, **MAX**, **CNT**, **SUM**, and **AVG** built-in functions:

- **MIN.ALL** implies the computation of minimum value of an attribute.
- **MAX.ALL** implies the computation of maximum value of an attribute.
- **CNT.ALL** implies COUNT the number of tuples in the relation.
- **SUM.ALL** implies the computation of sum of an attribute.
- **AVG.ALL** implies the computation of average value of an attribute.

Note: UNQ. which stands for unique operator is used to eliminate duplicates.
For example, **CNT.UNQ.ALL** computes the number of tuples in the relation
by eliminating duplicate values.

Example 10.1. MIN and MAX command

The QBE template to get the minimum and maximum vegetable price is
given later:

VEGETABLE	VEGETABLE	QUANTITY	PRICE
	NAME		
			P.MIN.ALL.CX
			P.MAX.ALL.CY

Example 10.2. AVG command

The QBE template to get the average price of the vegetable is given later.

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
			P.AVG.CX

Example 10.3. CNT command

The QBE template to count the number of unique vegetables in the VEGE-
TABLE relation is shown later.

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
P.CNT.UNQ.ALL			

Example 11. Update operation

The QBE template to increase the price of all vegetables by 10% is given as:

Here U. implies Update. The price UX of the vegetable is increased by 10% which is denoted by $1.1 * \underline{UX}$

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
U.			<u>UX</u> 1.1 * <u>UX</u>

Example 12. Record deletion

The QBE template to delete the record of all vegetables is shown later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
D.			

Here D. implies deletion of the entire relation.

Single Record Deletion

The QBE form to delete the record of the vegetable “Brinjal” is shown later:

VEGETABLE	VEGETABLENAME	QUANTITY	PRICE
D.	Brinjal		

Summary

In relational model, the data are stored in the form of tables or relations. Each table or relation has a unique name. Tables consist of a given number of columns or attributes. Every column of a table must have a name and no two columns of the same table may have identical names. The rows of the table are called tuples. The total number of columns or attributes that comprises a table is known as the *degree* of the table. The chapter has introduced the basic terminology used in relational model. Specific importance is given to E.F. Codd’s rule.

This chapter also introduced different integrity rules. Relational algebra concepts, different operators like SELECTION, PROJECTION, UNION, INTERSECTION, and JOIN operators were discussed with suitable examples. Relational calculus and its two branches, tuple relational calculus and domain relational calculus, were discussed in this chapter.

Finally, graphical user interface QBE, its relative advantage, different operations in QBE, concept of *condition box* in QBE, and aggregate functions in QBE were explained with suitable examples.

Review Questions

3.1. What is the degree and cardinality of the “Tennis Player” relation shown later:

Position	Player	Points	Nation
1	Federer	1117	Switzerland
2	Roddick, A.	671	USA
3	Hewitt, L.	638	Australia
4	Safin, M	497	Russia
5	Moya, C.	484	Spain

Hint: Degree of the relation = Number of columns in the relation.
 Cardinality of the relation = Number of rows in the relation.

3.2. A relation has a degree of 5 and cardinality of 7. How many attributes and tuples does the relation have?

3.3. A relation R has a degree of 3 and cardinality of 2 and the relation S has a degree of 2 and cardinality of 3, then what will be the degree and cardinality of the Cartesian product of R and S?

Ans: Cardinality = 6, Degree = 5.

3.4. What is the key of the following EMPLOYEE table?

EMPLOYEE				
EMPLOYEE NUMBER	EMPLOYEE NAME	DEPARTMENT	AGE	DESIGNATION
C100	Dr. Vijayarangan	Mechanical	51	Principal
C202	Dr. S. Jayaraman	ECE	50	Head
C203	Dr. Murugesh	EEE	50	Head
C204	Dr. Sivanandam	ComputerScience	53	Head
C208	Dr. Selvan	IT	51	Head

Ans: In the earlier table, EMPLOYEE NUMBER is the primary key. Because keys are used to enforce that no two rows are identical.

3.5. Define the operators in the core relational algebra?

3.6. Explain the following concepts in relational databases:

- (a) Entity integrity constraint
- (b) Foreign key and how it can specify a referential integrity constraint between two relations
- (c) Semantic integrity constraint

3.7. Mention the pros and cons of relational data model?

Pros of relational data model:

1. The relational data model is a well formed and data independent model which is easy to use for applications which fit well into the model.
2. The data used by most business applications fits this model, and that business applications were the first large customers of database system explains the popularity of the model.

Cons of relational data model:

1. The simplicity of the model restricts the amount of semantics, which can be expressed directly by the database.
2. Different groups of information, or tables, must be joined in many cases to retrieve data.

3.8. Bring out the reasons, why relational model became more popular?

1. Relational model was based on strong mathematical background.
2. Relational model used the power of mathematical abstraction. Operations do not require user to know storage structures used.
3. Strong mathematical theory provides tool for improving design.
4. Basic structure of the relation is simple, easy to understand and implement.

3.9. A union, intersection or difference can only be performed between two relations if they are type compatible. What is meant by type compatibility? Give an example of two type compatible and two nontype compatible relations?

Two relations are type compatible if they have same set of attributes. Example of two type compatible relations is:

```
Men {<name:varchar>, <dob:date>, <address:varchar>}
Women {<name:varchar>, <dob:date>, <address:varchar>}
Example of two relations which are nontype compatible is:
Husband {<name:varchar>, <dob:date>, <salary: number>}
Wife {<name:varchar>, <dob:date>, <address:varchar>}
```

3.10. What are the advantages of QBE?

QBE can be considered as GUI (Graphical User Interface) based on domain calculus. QBE allows users to key in their input requests by filling in empty tables on the screen, and the system will also display its response in tabular form. QBE is user-friendly because the users are not required to formulate sentences for query requests with rigid query-language syntax.

3.11. What do you understand by domain integrity constraint?

The domain integrity constraints are used to specify the valid values that a column defined over the domain can take. We can define the valid values by listing them as a set of values (such as an enumerated data type in a strongly typed programming language), a range of values, or an expression that accepts the valid values.

3.12. What do you understand by “safety of expressions”?

It is possible to write tuple calculus expressions that generate infinite relations. For example $\{t/\sim t \in R\}$ results in an infinite relation if the domain of any attribute of relation R is infinite. To guard against the problem, we restrict the set of allowable expressions to safe expressions.

3.13. What are “quantifiers”? How will you classify them?

Quantifiers are words that refer to quantities such as “some” or “all” and tell for how many elements a given predicate is true. A predicate is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables. Quantifiers can be broadly classified into two types (1) Universal Quantifier and (2) Existential Quantifier.