Table of Contents for Introduction to Database Systems

CHAPTER 4

RELATIONAL ALGEBRA AND CALCULUS

After reading this chapter, the reader will understand:

- *Different query languages used to extract data from the database*
- *Difference between relational algebra and relational calculus*
- *Various unary relational algebra operations like select, project, and rename*
- *Various binary relational algebra operations like set operations, joins, and division operation*
- *Various set operations like union, intersection, difference, and cartesian product operation*
- *Different types of joins like equijoin, natural, and outer joins (left, right, and full outer join)*
- *The aggregate functions that are used to perform queries, which are mathematical in nature*
- *Two forms of relational calculus, namely, tuple relational calculus and domain relational calculus*
- *The use of tuple variables for defining queries in tuple relational calculus*
- *Transformation of the universal and existential quantifiers*
- *The concept of safe expressions*
- *The use of domain variables for defining queries in domain relational calculus*
- *Expressive power of relational algebra and relational calculus*

The two formal languages associated with relational model that are used to specify the basic retrieval requests are *relational algebra* and *relational calculus*. Relational algebra provides a foundation for relational model

operations and is used as a basis for applying and optimizing queries in relational database management systems. It consists of basic set of operations, which can be used for carrying out basic retrieval operations. Relational calculus, on the other hand, provides declarative notations based on mathematical logic for specifying relational queries. In relational calculus, query is expressed in terms of variables and formulas on these variables. The formula in relational calculus describes the properties of the required resultant.

Relational algebra and relational calculus are logically equivalent; however, they differ on the basis of the approach they follow to retrieve data from a relation. Relational algebra uses procedural approach by providing a step-by-step procedure for carrying out a query, whereas relational calculus uses non-procedural approach as it describes the information to be retrieved without specifying the method for obtaining that information. DBMS automatically transforms these non-procedural queries into equivalent procedural queries.

This chapter discusses various operations of relational algebra. In addition, it discusses two forms of relational calculus, namely, tuple relational calculus and domain relational calculus. The chapter then concludes with the discussion on the expressive power of relational algebra and relational calculus.

4.1 RELATIONAL ALGEBRA

Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input and result into a new relation as an output. These operations are divided into two groups. One group consists of operations developed specifically for relational databases such as *select, project, rename, join*, and *division*. The other group includes the set-oriented operations such as *union, intersection, difference*, and *cartesian product*. Some of the queries, which are mathematical in nature, cannot be expressed using the basic operations of relational algebra. For such queries, additional operations like aggregate functions and grouping are used such as finding sum, average,

etc. This section discusses these operations in detail.

Things to Remember

The relational closure property states that the output generated from any type of relational operation is in the form of relations only.

4.1.1 Unary Operations

The operations operating on a single relation are known as **unary operations**. In relational algebra, *select, project*, and *rename* are unary operations.

Select Operation

The **select operation** retrieves all those tuples from a relation that satisfy a specific condition. It can also be considered as a filter that displays only those tuples that satisfy a given condition. The select operation can be viewed as the horizontal subset of a relation. The Greek letter sigma (σ) is used as a select operator.

In general, the select operation is specified as

$$\sigma_{\texttt{<selection\_condition>}}(R)$$

where,

	`selection_condition` =	the condition on the basis of which subset of tuples is selected
	`R` =	the name of the relation

For example, to retrieve those tuples from `BOOK` relation whose `Category` is *Novel*, the select operation can be specified as

$$\sigma_{\texttt{Category="Novel"}}(\texttt{BOOK})$$

Basically where clause.

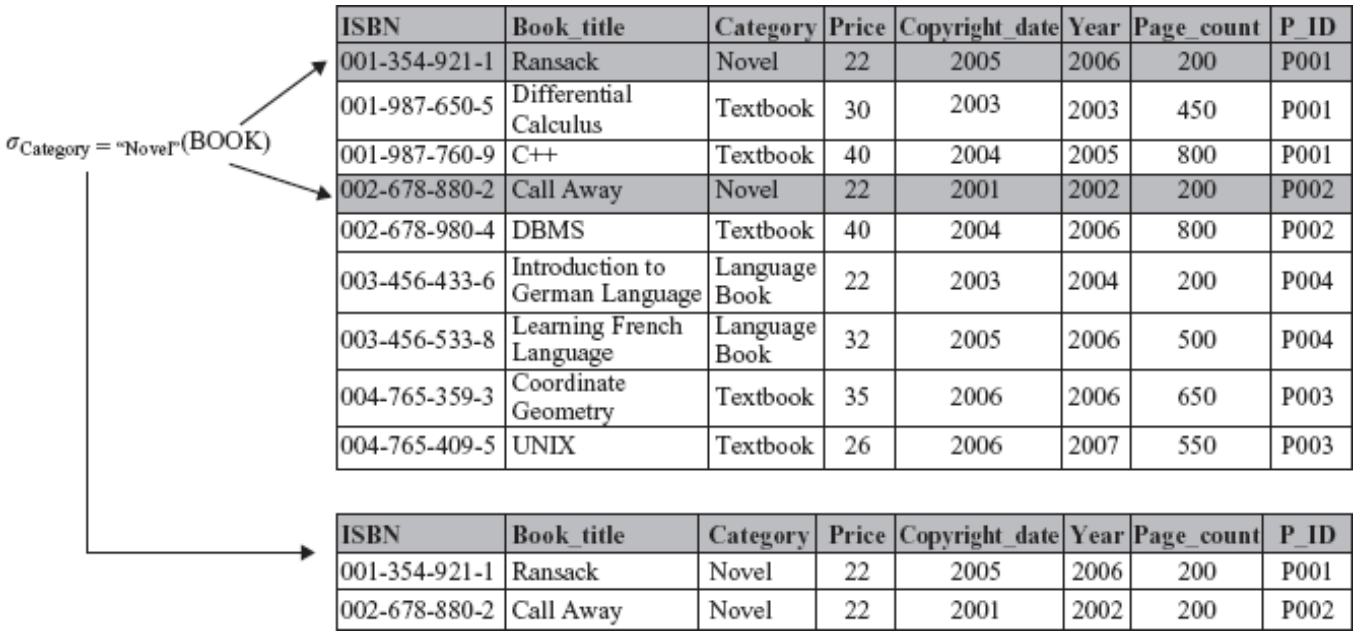
The resultant relation of this operation consists of two tuples with `Category` as *Novel* and it is shown in Figure 4.1.

The comparison operators (=, ≠, <, ≤, >, ≥) can be used to specify the conditions for selecting required tuples from a relation. Moreover, more than one condition can be concatenated to one another to give a more specific condition by using any of the logical operators AND (∧), OR (∨), and NOT (¬). For example, to retrieve those tuples from `BOOK` relation, whose `Category` is *Language Book* and `Page_count` is greater than *400*, the select operation can be specified as

$$\sigma_{\texttt{Category="Language Book"} \wedge \texttt{Page\_count>400}} (\texttt{BOOK})$$

Consider another example, to retrieve those tuples from `PUBLISHER` relation, whose `State` is either *Georgia* or *Hawaii*, the select operation can be specified as

$$\sigma_{\texttt{State="Georgia"} \vee \texttt{State="Hawaii"}} (\texttt{PUBLISHER})$$



ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002

Fig. 4.1 *Select operation*

NOTE The comparison operators are used only with the attributes, whose domain is compatible for comparison.

The degree (number of attributes) of the relation resulting from a select operation is same as the degree of relation `R`. The cardinality (number of tuples) of the resulting relation is always less than or equal to the number of tuples in `R`.

The sequence of select operations can be applied in any order, as it is commutative in nature, that is,

$$\sigma_{\texttt{<condition\_1>}}\ (\sigma_{\texttt{<condition\_2>}}\ R)\ =\ \sigma_{\texttt{<condition\_2>}}\ (\sigma_{\texttt{<condition\_1>}}\ R)$$

In other words, the order in which conditions are applied is not significant.

Project Operation

A relation might consist of a number of attributes, which are not always required. The **project operation** is used to select some required attributes from a relation while discarding the other attributes. It can be viewed as the vertical subset of a relation. The Greek letter pi (π) is used as project operator.

In general, the project operation is specified as

$$\pi_{\texttt{<attribute\_list>}}(R)$$

where,

`attribute_list` = list of required attributes separated by comma
`R` = the name of relation

For example, to retrieve only `ISBN`, `Book_title` and `Price` attributes from `BOOK` relation, the project operation can be specified as

$$\pi_{\texttt{ISBN, Book\_title, Price}}\ (\texttt{BOOK})$$

The resultant relation of this operation consists of three attributes of all the tuples as shown in Figure 4.2.

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

$\pi_{ISBN, Book\_title, Price}(BOOK)$

ISBN	Book_title	Price
001-354-921-1	Ransack	22
001-987-650-5	Differential Calculus	30
001-987-760-9	C++	40
002-678-880-2	Call Away	22
002-678-980-4	DBMS	40
003-456-433-6	Introduction to German Language	22
003-456-533-8	Learning French Language	32
004-765-359-3	Coordinate Geometry	35
004-765-409-5	UNIX	26

Fig. 4.2 *Project operation*

Consider another example to retrieve `P_ID`, `State`, and `Phone` attributes from `PUBLISHER` relation, the project operation can be specified as

$$\pi_{P\_ID,\ State,\ Phone}\ (PUBLISHER)$$

The resultant relation of project operation consists of only those attributes, which are specified in <`attribute_list`> and are in the same

order as they are appearing in the list. Hence, the degree of the resultant relation is equal to the number of attributes in <attribute_list>.

If <attribute_list> contains only non-key attributes of relation R, duplicate tuples are likely to appear in the resultant relation. The project operation removes duplicate tuples from the resultant relation. Therefore, the result of the project operation is a valid relation containing a set of unique tuples. This feature of project operation is known as **duplicate elimination**. For example, consider the project operation given here.

$$\pi_{\text{Category, Page\_count}}\ (\text{BOOK})$$

The resultant relation of this operation consists of only two attributes, namely, Category and Page_count. Since the project operation removes duplicate tuples from the resultant relation, the tuples (*Textbook, 800*) and (*Novel, 200*) appear only once in the resultant relation, although the combination of these values appear repeatedly in BOOK relation as shown in Figure 4.3.

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

$\pi_{Category, Page\_count}(BOOK)$

Category	Page_count
Textbook	450
Novel	200
Textbook	800
Novel	200
Textbook	800
Language Book	200
Language Book	500
Textbook	650
Textbook	550

Fig. 4.3 *Duplicate elimination in project operation*

Unlike select operation, the commutative property does not hold on project operation, that is,

$$\pi_{<list\_1>}(\pi_{<list\_2>}(R)) \neq \pi_{<list\_2>}(\pi_{<list\_1>}(R))$$

In other words, the order in which the list of attributes is specified in project operation is significant.

Note that, the select and project operations can also be combined together to create more complex queries. For example, to retrieve `P_ID`, `Address`, and `Phone` attributes from `PUBLISHER` relation where `State` is

Georgia, the combined operation can be specified as

$$\pi_{P\_ID,\ Address,\ Phone}(\sigma_{State=\text{“Georgia”}}(PUBLISHER))$$

This statement first applies select operation on `PUBLISHER` relation and then applies project operation on the tuples retrieved from select operation.

Rename Operation

When relational algebra operations are performed, the resultant relations are unnamed, hence cannot be used for later reference. In addition, it might be required to apply more operations on the relation obtained from other operations. In such situations, rename operator proves to be useful. **Rename operation** is used to provide name to the relation obtained after applying any relational algebra operation. The Greek letter rho (*r*) is used as a rename operator. Relation obtained from any operation can be renamed by using rename operator as shown here.

> $\rho(R,\ E)$

where,

	ρ =	rename operator
	`E` =	expression representing relational algebra operations
	`R` =	name given to relation obtained by applying relational algebra operations specified in expression `E`

Consider some examples of using rename operator to rename queries discussed earlier.

> $\rho(R_1,\ \sigma_{Category=\text{“Novel”}}(BOOK))$
>
> $\rho(R_2,\ \pi_{P\_ID,\ State,\ Phone}(PUBLISHER))$
>
> $\rho(R_3,\ \pi_{P\_ID,\ Address,\ Phone}(\sigma_{State=\text{“Georgia”}}(PUBLISHER)))$

Here, R_1, R_2, and R_3 are the names given to the relations obtained from the respective relational algebra expressions. In these examples, the name of the attributes in new relation is same as in their corresponding original relations. However, attributes in a new relation can also be renamed using the rename operator as shown here.

> $\rho(R(A_1, A_2, \ldots A_n), E)$

where,

$A_1, A_2, \ldots A_n$ are the new names provided to the attributes of the relation R, obtained after executing relational algebra expression E. For example, consider an expression given here.

> $\rho(R_2(P_1, P_2, P_3), \pi_{P\_ID, State, Phone}(PUBLISHER))$

In this example, P_1, P_2, P_3 are new attribute names in the relation R_2 for the attributes P_ID, $State$, and $Phone$ of the relation $PUBLISHER$, respectively.

Sometimes the situation may arise where multiple relational algebra operations need to be applied in a sequence. One way to achieve this is by grouping together the sequence of operations to form a single relational algebra expression. The second way is to apply one operation at a time to create many intermediate result relations. In such situations, rename operator can be used to provide name to the intermediate relations so that they can be referred easily in the subsequent operations.

For example, consider a query to retrieve P_ID, $Address$, and $Phone$ of publishers located at *Georgia* state. This query can be expressed in a single relational algebra expression as discussed earlier or can be expressed in two steps with the help of rename operator as shown here.

> $\rho(R_1, \sigma_{State=\text{“Georgia”}}(PUBLISHER))$
>
> $\rho(R_2, \pi_{P\_ID, Address, Phone}(R_1))$

In this example, select operation is applied to retrieve tuples where $State$

is *Georgia* and resultant relation is named as R_1. Then, the project operation is applied to retrieve selected attributes, namely, `P_ID`, `Address`, and `Phone` from the intermediate relation R_1 and the resultant relation is named as R_2. The relation R_2 can further be used for more operations.

NOTE The rename operator is optional, it is not always necessary to use it. It can be used as per one's convenience.

4.1.2 Binary Operations

The operations operating on two relations are known as **binary operations**. The set operations, join operations, and division operation are all binary operations.

Set Operations

The standard mathematical operations on sets, namely, *union, intersection, difference*, and *cartesian product* are also available in relational algebra. Of these, the three operations, namely, union, intersection, and difference operations require two relations to be union compatible with each other. The two relations are said to be union compatible, if they satisfy these conditions.

1. The degree of both the operand relations must be same.
2. The domain of *n*th attribute of relation R_1 must be same as that of the *n*th attribute of relation R_2.

However, the cartesian product can be defined on any two relations, that is, they need not be union compatible.

Union Operation The **union operation**, denoted by $\cup$, returns a third relation that contains tuples from both or either of the operand relations. Consider two relations R_1 and R_2, their union is denoted as $R_1 \cup R_2$, which returns a relation containing tuples from both or either of the given relations. The duplicate tuples are automatically removed from the resultant relation.

For example, consider the two union compatible relations PUBLISHER_1 and PUBLISHER_2. The union of these two relations consists of tuples from both relations without any duplicate tuples as shown in Figure 4.4.

PUBLISHER_1

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.
P003	Bright Publications
P004	Paramount Publishing House
P005	Wesley Publications

PUBLISHER_2

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.
P006	ESL Publications
P007	Arizon Publishers Ltd.

PUBLISHER_1 U PUBLISHER_2

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.
P003	Bright Publications
P004	Paramount Publishing House
P005	Wesley Publications
P006	ESL Publications
P007	Arizon Publishers Ltd.

Fig. 4.4 *Union operation*

The union operation is, respectively, commutative and associative in nature, that is,

$$R_1 \cup R_2 = R_2 \cup R_1$$

and

$$R_1 \cup (R_2 \cup R_3) = (R_1 \cup R_2) \cup R_3$$

Intersection Operation The **intersection operation**, denoted by ∩, returns a third relation that contains tuples common to both the operand relations. The intersection of the relations R_1 and R_2 is denoted by $R_1 \cap R_2$, which returns a relation containing tuples common to both the given relations.

For example, consider two union compatible relations PUBLISHER_1 and PUBLISHER_2. The intersection of these two relations consists of tuples

common to both the relations as shown in Figure 4.5.

PUBLISHER_1

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.
P003	Bright Publications
P004	Paramount Publishing House
P005	Wesley Publications

PUBLISHER_2

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.
P006	ESL Publications
P007	Arizon Publishers Ltd.

PUBLISHER_1 ∩ PUBLISHER_2

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.

Fig. 4.5 *Intersection operation*

Like, union operation, intersection operation is, respectively, commutative and associative in nature, that is,

$$R_1 \cap R_2 = R_2 \cap R_1$$

and

$$R_1 \cap (R_2 \cap R_3) = (R_1 \cap R_2) \cap R_3$$

Learn More

Both the union and intersection operations can be performed on any number of relations, as they are associative in nature. Thus, they can be treated as *n*-ary operations.

Difference Operation The **difference operation**, denoted by – (minus), returns a third relation that contains all tuples present in one relation, which are not present in the second relation. The difference of the relations R_1 and R_2 is denoted by $R_1 - R_2$. The expression $R_1 - R_2$ results in a relation containing all tuples from relation R_1, which are not present in relation R_2. Similarly, the expression $R_2 - R_1$ results in a relation containing all tuples from relation R_2, which are not present in relation R_1.

For example, consider the two union compatible relations PUBLISHER_1 and PUBLISHER_2, the difference of these two relations is shown in Figure 4.6.

The difference operation is not commutative in nature, that is,

$$R_1 - R_2 \neq R_2 - R_1$$

Note that any relational algebra expression that uses intersection operation can be rewritten by replacing the intersection operation with a pair of difference operation as shown here.

$$R_1 \cap R_2 = R_1 - (R_1 - R_2)$$

PUBLISHER_1

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.
P003	Bright Publications
P004	Paramount Publishing House
P005	Wesley Publications

PUBLISHER_2

P_ID	Pname
P001	Hills Publications
P002	Sunshine Publishers Ltd.
P006	ESL Publications
P007	Arizon Publishers Ltd.

PUBLISHER_1-PUBLISHER_2

P_ID	Pname
P003	Bright Publications
P004	Paramount Publishing House
P005	Wesley Publications

PUBLISHER_2-PUBLISHER_1

P_ID	Pname
P006	ESL Publications
P007	Arizon Publishers Ltd.

Fig. 4.6 *Difference operation*

Thus, intersection operation is not a fundamental operation. It is easier to write $R_1 \cap R_2$ than to write $R_1 - (R_1 - R_2)$. In addition, the intersection operation can also be expressed in terms of difference and union operations as shown here.

$$R_1 \cap R_2 = (R_1 \cup R_2) - ((R_1 - R_2) \cup (R_2 - R_1))$$

Cartesian Product Operation The **cartesian product**, also known as **cross product** or **cross join**, returns a third relation that contains all

possible combinations of the tuples from the two operand relations. The cartesian product is denoted by the symbol ×. The cartesian product of the relations R_1 and R_2, is denoted by $R_1 \times R_2$.

Each tuple of the first relation is concatenated with all the tuples of the second relation to form the tuples of a new relation. The cartesian product creates tuples with the combined attributes of two relations. Therefore, the degree of a new relation is the sum of the degree of both the operand relations. In addition, the cardinality of the resultant relation is the product of the cardinality of both the operand relations. In other words, if d_1 and d_2 are the degree of the relations R_1 and R_2, respectively, then the degree of the relation $R_1 \times R_2$ is $d_1 + d_2$, and if c_1 and c_2 are the cardinality of relations R_1 and R_2, respectively, then the cardinality of the relation $R_1 \times R_2$ is $c_1 * c_2$.

For example, consider two relations `AUTHOR_1` with attributes `A_ID`, `Aname`, and `City`, and `BOOK_1` with attributes `ISBN`, `Book_title`, and `Price`; the cartesian product of these two relations is shown in Figure 4.7.

In this example, each tuple from relation `AUTHOR_1` is concatenated with every tuple of relation `BOOK_1`. The degree of both the operand relations is 3 and hence, the degree of the resultant relation is 6(3+3). The cardinality of the resultant relation is 12, which is the product of the cardinality of the relation `AUTHOR_1`, that is, 4 and cardinality of the relation `BOOK_1`, that is, 3.

NOTE The cartesian product can be made more useful and meaningful, if it is followed by select and project operations to retrieve meaningful and valuable information from the database.

Joins

The **join** operation is one of the most useful and commonly used operations to extract information from two or more relations. A join can be defined as a cartesian product followed by select and project

operations. The result of a cartesian product is usually much larger than the result of a join.

AUTHOR_1

A_ID	Aname	City
A001	Thomas William	New York
A002	James Erin	Atlanta
A003	Charles Smith	Los Angeles
A004	Lewis Ian	Seattle

BOOK_1

ISBN	Book_title	Price
001-987-760-9	C++	40
001-354-921-1	Ransack	22
002-678-980-4	DBMS	40

AUTHOR_1 × BOOK_1

A_ID	Aname	City	ISBN	Book_title	Price
A001	Thomas William	New York	001-987-760-9	C++	40
A001	Thomas William	New York	001-354-921-1	Ransack	22
A001	Thomas William	New York	002-678-980-4	DBMS	40
A002	James Erin	Atlanta	001-987-760-9	C++	40
A002	James Erin	Atlanta	001-354-921-1	Ransack	22
A002	James Erin	Atlanta	002-678-980-4	DBMS	40
A003	Charles Smith	Los Angeles	001-987-760-9	C++	40
A003	Charles Smith	Los Angeles	001-354-921-1	Ransack	22
A003	Charles Smith	Los Angeles	002-678-980-4	DBMS	40
A004	Lewis Ian	Seattle	001-987-760-9	C++	40
A004	Lewis Ian	Seattle	001-354-921-1	Ransack	22
A004	Lewis Ian	Seattle	002-678-980-4	DBMS	40

Fig. 4.7 *Cartesian product*

The join operation joins two relations to form a new relation on the basis of one common attribute present in the two operand relations. That is, if both the operand relations have one common attribute defined over some common domain, then they can be joined over this common attribute. The join results in a new wider relation in which each tuple is formed by concatenating the two tuples, one from each of the operand relations, such that two tuples have the same value for that common attribute.

The join is denoted by the symbol ⋈. The most general form of the join operation is based on a condition and a pair of operand relations.

That is, the join operation is specified as

$$R_1 \bowtie_{cond} R_2 = \sigma_{cond}(R_1 \times R_2)$$

Note that the condition *cond* refers to the join condition involving attributes of both relations R_1 and R_2. There are different types of join operations in relational algebra, namely, *equijoin, natural join, outer join*, etc.

Equijoin A common case of the join operation $R_1 \bowtie R_2$ is one in which the join condition consists only of equality condition. This type of join where the only comparison operator used is, = is known as **equijoin**. The resultant relation of an equijoin always has one or more pairs of attributes that have identical values in every tuple.

For example, the relations BOOK and PUBLISHER can be joined for the tuples where BOOK.P_ID is same as PUBLISHER.P_ID. This is an example of equijoin operation and it is specified as shown here.

$$BOOK \bowtie_{BOOK.P\_ID = PUBLISHER.P\_ID} PUBLISHER$$

The resultant relation of such a join is shown in Figure 4.8.

ISBN	Book_title	...	BOOK.P_ID	PUBLISHER.P_ID	Pname	...	Email_id
001-987-760-9	C++	...	P001	P001	Hills Publications	...	h_pub@hills.com
001-354-921-1	Ransack	...	P001	P001	Hills Publications	...	h_pub@hills.com
001-987-650-5	Differential Calculus	...	P001	P001	Hills Publications	...	h_pub@hills.com
002-678-980-4	DBMS	...	P002	P002	Sunshine Publishers Ltd.	...	null
002-678-880-2	Call Away	...	P002	P002	Sunshine Publishers Ltd.	...	null
004-765-409-5	UNIX	...	P003	P003	Bright Publications	...	bright@bp.com
004-765-359-3	Coordinate Geometry	...	P003	P003	Bright Publications	...	bright@bp.com
003-456-433-6	Introduction to German Language	...	P004	P004	Paramount Publishing House	...	param_house@paramount.com
003-456-533-8	Learning French Language	...	P004	P004	Paramount Publishing House	...	param_house@paramount.com

Fig. 4.8 *Equijoin operation*

In the resultant relation of equijoin operation, the values of the attributes `PUBLISHER.P_ID` and `BOOK.P_ID` are same for every tuple as equality condition is specified on these attributes.

The attribute publisher ID is appearing with the same name `P_ID` in both the relations, thus, in order to differentiate these two, the attribute names are qualified by their corresponding relation name using the dot (.) operator.

The select operation can be applied on the resultant relation to retrieve only selected tuples. For example, if only those tuples are to be displayed where the `Category` of book is *Textbook* and it is published by publishers located in *Georgia*, then the join operation along with the select operation can be specified as

$$\sigma_{\text{Category="Textbook"} \wedge \text{State="Georgia"}}(\text{BOOK} \bowtie_{\text{BOOK.P\_ID} = \text{PUBLISHER.P\_ID}} \text{PUBLISHER})$$

The result of this expression consists of a set of tuples from the resultant relation of join operation satisfying the conditions specified.

Natural Join The joins having equality condition as their join condition result in two attributes in the resulting relation having exactly the same value. However, if one of the two identical attributes is removed from the result of equijoin, it is known as **natural join**.

The natural join of two relations R_1 and R_2 is obtained by applying a project operation to the equijoin of these two relations in a sequence given here.

1. Find the equijoin of two relations R_1 and R_2.
2. For each attribute A that is common to both relations R_1 and R_2, project operation is applied to remove the column $R_1.A$ or $R_2.A$. Therefore, if there are *m* attributes common in both the relations, then *m* duplicate columns are removed from the resultant relation of equijoin.

Learn More

The expression to obtain natural join of relations BOOK and PUBLISHER can be specified as

$$\text{ISBN, Book\_title, Category, BOOK.P\_ID, Pname, Address, Email\_ID}(\text{BOOK} \bowtie_{\text{BOOK.P\_ID} = \text{PUBLISHER.P\_ID}} \text{PUBLISHER})$$

The resultant relation of such a natural join is shown in Figure 4.9.

ISBN	Book_title	Category	BOOK.P_ID	Pname	Address	Email_id
001-987-760-9	C++	Textbook	P001	Hills Publications	12, Park street, Atlanta	h_pub@hills.com
001-354-921-1	Ransack	Novel	P001	Hills Publications	12, Park street, Atlanta	h_pub@hills.com
002-678-980-4	DBMS	Textbook	P002	Sunshine Publishers Ltd.	45, Second street, Newark	null
004-765-409-5	UNIX	Textbook	P003	Bright Publications	123, Main street, Honolulu	bright@bp.com
004-765-359-3	Coordinate Geometry	Textbook	P003	Bright Publications	123, Main street, Honolulu	bright@bp.com
003-456-433-6	Introduction to German Language	Language Book	P004	Paramount Publishing House	789, Oak street, New York	param_house @param.com
001-987-650-5	Differential Calculus	Textbook	P001	Hills Publishers Ltd.	12, Park street, Atlanta	h_pub@hills.com
002-678-880-2	Call Away	Novel	P002	Sunshine Publishers Ltd.	45, Second street, Newark	null
003-456-533-8	Learning French Language	Language Book	P004	Paramount Publishing House	789, Oak street, New York	param_house @param.com

Fig. 4.9 *Natural join operation*

Note that, the attribute `P_ID` appears only once in the resultant relation of natural join. The pre-requisite of natural join is that the two attributes involved in the join condition must have the same name. If they have different names then the renaming operation is applied prior to the join operation.

Outer join The joins discussed so far are simple and are known as **inner joins**. An inner join selects only those tuples from both the joining relations that satisfy the joining condition. The **outer join**, on the other hand, selects all the tuples satisfying the join condition along with the tuples for which no tuples from the other relation satisfy the join condition.

An outer join is a type of equijoin that can be used to display all the tuples from one relation, even if there are no corresponding tuples in the second relation, thus preventing information loss. It is commonly used with relations having one-to-many relationship. The three types of outer joins are

- **Left outer join:** The **left outer join** includes all the tuples from both

the relations satisfying the join condition along with all the tuples in the left relation that do not have a corresponding tuple in the right relation. The tuples from the left relation not satisfying the join condition are concatenated with the tuples having null values for all the attributes from the right relation. The left outer join of relations R_1 and R_2 is specified as $R_1 ⟕_{cond} R_2$.

- **Right outer join:** The **right outer join** includes all the tuples from both the relations satisfying the join condition along with all the tuples in the right relation that do not have a corresponding tuple in the left relation. The tuples from the right relation not satisfying the join condition are concatenated with the tuples having null values for all the attributes from the left relation. The right outer join of relations R_1 and R_2 is specified as $R_1 ⟖_{cond} R_2$.
- **Full outer join:** A **full outer join** combines the results of both the left and the right outer joins. The resultant relation contains all records from both the relations, with null values for the missing corresponding tuples on either side. The full outer join of relations R_1 and R_2 is specified as $R_1 ⟗_{cond} R_2$.

Division

The **division** operation, denoted by ÷ is useful for queries of the form '*for all objects having all the specified properties*'. To understand the division operation, consider a relation R_1 having exactly two attributes A and B, and a relation R_2 having just one attribute B with the same domain as in R_1. The division operation $R_1 \div R_2$ is defined as the set of all values of attribute A, such that for every value of attribute B in R_2, there is a tuple (A, B) in R_1. In general, for the relations R_1 and R_2, $R_1 \div R_2$ results in the relation R_3 such that $R_3 \times R_2 \subseteq R_1$.

For example, consider the relations R_1 and R_2 with the sample data as shown in Figure 4.10.

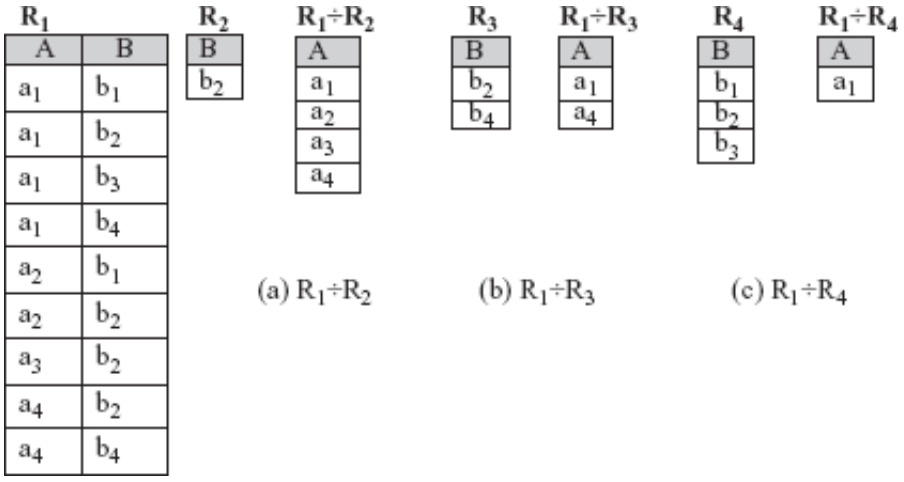


Fig. 4.10 *Division operation*

In this example, consider the case of division operation on the relations R_1 and R_2. The relation $R_1 \div R_2$ is the set of values that forms a tuple with every value in the relation R_2, appearing in a relation R_1. To elaborate further, the tuples (a_1, b_2), (a_2, b_2), (a_3, b_2) and (a_4, b_2) are appearing in the relation R_1 where, b_2 is a member of the relation R_2 and (a_1, a_2, a_3, a_4) are members of the relation $R_1 \div R_2$. Similarly, the results of operations $R_1 \div R_2$ and $R_1 \div R_4$ are shown in Figure 4.10 (b) and 4.10 (c), respectively.

Now, take another example of *Online Book* database, 'retrieve the list of all authors writing books for publishers located in *New Jersey* and *Hawaii*'. To achieve this, first the join operation is applied to create a relation R_1(P_ID, A_ID), which contains the list of publisher ID for whom authors have written the books. Now create a relation R_2(P_ID) having the list of publisher ID belonging to *New Jersey* or *Hawaii* state. The relations R_1 and R_2 can be obtained using these algebraic expressions.

$$\rho\ (R_1,\ \pi_{P\_ID,\ A\_ID}(BOOK \bowtie_{BOOK.ISBN=AUTHOR\_BOOK.ISBN} AUTHOR\_BOOK))$$

$$\rho\ (R_2,\ \pi_{P\_ID}(\sigma_{State=\text{“New Jersey” OR State=“Hawaii”}}(PUBLISHER))$$

The resultant relation of division operation $R_1 \div R_2$ is shown in Figure 4.11.

R_1

P_ID	A_ID
P001	A001
P002	A002
P001	A003
P004	A004
P001	A005
P002	A006
P001	A007
P002	A008
P003	A008
P003	A009
P004	A010

R_2

P_ID
P002
P003

$R_1 \div R_2$

A_ID
A008

Fig. 4.11 *Example of division operation*

In this example, the relation $R_1 \div R_2$ is the set of values that forms a tuple with every value in relation R_2, which also appears in relation R_1. To elaborate further, the tuples (*P002, A008*) and (*P003, A008*) appear in the relation R_1 where *P002* and *P003* are from relation R_2.

4.1.3 Aggregate Functions and Grouping

The queries discussed so far are sufficient enough to satisfy most of the requirements of an organization. However, some of the requirements like finding total amount, average price, which are mathematical in nature cannot be expressed by basic relational algebra operations. For such type of queries, aggregate functions are used. Aggregate functions take a collection of values as input, process them, and return a single value as the result. For example, the aggregate function `SUM` returns a sum of the collection of numeric values taken as input. Similarly, aggregate functions, namely, `AVG`, `MAX`, and `MIN` are used to find average, maximum, and minimum of the collection of numeric values, respectively. The `COUNT` function is used for counting tuples or values in a relation.

Sometimes, situation may arise where tuples in a relation are required to be grouped, based on the values of an attribute. For example, the tuples of relation `BOOK` can be divided into groups based on the values of attribute `Category`. In other words, the books belonging to textbook

category form one group, books belonging to novel category form another group, and so on.

The aggregate functions can be applied on each group individually, thus, returning a result value for each group separately. For example, for calculating average price for each category of book, AVG function is applied on each category of book separately. In other words, if there are three categories of books, then AVG function is applied to each of the three categories separately, which returns three average values for each category. The grouping of tuples of a relation can be based on one or more attributes. For example, tuples in relation BOOK can be grouped on two attributes, that is, first on the basis of category and then on the basis of publisher ID.

The aggregate function operation can be specified using the symbol $\mathfrak{F}$ (called 'script F') as shown here.

$$<\text{attribute\_list}>\mathfrak{F}_{<\text{function(attribute)\_list}>}(R)$$

where,

<attribute_list> = list of attributes on the basis of which tuples of a relation are to be grouped

<function(attribute)_list> = list of functions to be applied on different attributes.

For example, consider a relation BOOK, where average, maximum, and minimum price is to be calculated for each category of book separately. The expression to represent this query can be specified as

$$\text{Category}\mathfrak{F}_{\text{AVG(Price), MAX(Price), MIN(Price)}}(\text{BOOK})$$

The list of attributes on the basis of which tuples are grouped can be omitted from an expression. If grouping attribute is not specified in the expression, then the function is applied individually on all the tuples in the relation. Consider an example where the number of tuples in a relation

has to be calculated. The required expression can be specified as

$$\Im_{COUNT(ISBN)}(BOOK)$$

Whenever a function is applied on an attribute, the duplicate values are also included while performing the calculations. However, concatenating the function name with the string `_DISTINCT` can eliminate duplicate values. For example, consider a relation `BOOK` where the number of categories is to be calculated. If `COUNT` function is applied on attribute Category, it counts all the values including duplicate values. However, if `COUNT_DISTINCT` function is applied, it counts only the unique values.

4.1.4 Example Queries in Relational Algebra

Various operations of relational algebra can be combined together to create more advanced queries to extract required data from the database. Some of the queries are discussed here to extract information from *Online Book* database using operations of relational algebra.

Query 1: Retrieve city, phone and URL of the author whose name is *Lewis Ian*.

$$\pi_{City,Phone,URL}(\sigma_{Aname=\text{“Lewis Ian”}}(AUTHOR))$$

In this query, the select operation is used to retrieve all the tuples where `Aname` is *Lewis Ian*, and then the project operation is used to display the required attributes, namely, `City`, `Phone` and `URL` for the tuples retrieved.

Query 2: Retrieve name, address, and phone of all the publishers located in *New York* state.

$$\pi_{Pname,\ Address,\ Phone}(\sigma_{State=\text{“New York”}}(PUBLISHER))$$

In this query, the select operation is used to retrieve all the tuples where `State` is *New York*, and then the project operation is used to display the

required attributes, namely, `Pname`, `Address`, and `Phone` for the tuples retrieved.

Query 3: Retrieve title and price of all the textbooks with page count greater than *600*.

$$\pi_{\text{Book\_title, Price}}(\sigma_{\text{Category=“Textbook” } \wedge \text{ Page\_count>600}}(\text{BOOK}))$$

In this query, the select operation is used to retrieve all the tuples where `Category` is *Textbook* and `Page_count` is greater than *600* and then the project operation is used to display the required attributes, namely, `Book_title` and `Price` for the tuples retrieved.

Query 4: Retrieve ISBN, title, and price of the books belonging to either novel or language book category.

$$\pi_{\text{ISBN, Book\_title, Price}}(\sigma_{\text{Category=“Novel” } \vee \text{ Category=“Language Book”}}(\text{BOOK})$$

In this query, the select operation is used to retrieve all the tuples where `Category` is either *Textbook* or *Novel* and then the project operation is used to display the required attributes, namely, `ISBN`, `Book_title`, and `Price` for the tuples retrieved.

Query 5: Retrieve ID, name, address, and phone of publishers publishing novels.

$$\pi_{\text{P\_ID,Pname,Address,Phone}}(\sigma_{\text{Category=“Novel”}}(\text{BOOK} \bowtie_{\text{BOOK.P\_ID=PUBLISHER.P\_ID}} \text{PUBLISHER}))$$

In this query, the join of relations `BOOK` and `PUBLISHER` is created on the basis of common attribute `P_ID`. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where `Category` is *Novel*, and then the project operation is used to display the required attributes, namely, `P_ID`, `Pname`, `Address`, and `Phone` for the tuples

retrieved.

Query 6: Retrieve title and price of all the books published by *Hills Publications*.

$$\pi_{\texttt{Book\_title,Price}}(\sigma_{\texttt{Pname="Hills Publications"}}(\texttt{BOOK} \bowtie_{\texttt{BOOK.P\_ID=PUBLISHER.P\_ID}} \texttt{PUBLISHER}))$$

In this query, the join of relations `BOOK` and `PUBLISHER` is created on the basis of common attribute `P_ID`. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where `Pname` is *Hills Publications*, and then the project operation is used to display the required attributes, namely, `Book_title` and `Price` for the tuples retrieved.

Query 7: Retrieve book title, reviewers ID, and rating of all the textbooks.

$$\pi_{\texttt{Book\_title,R\_ID,Rating}}(\sigma_{\texttt{Category="Textbook"}}(\texttt{BOOK} \bowtie_{\texttt{BOOK.ISBN=REVIEW.ISBN}} \texttt{REVIEW}))$$

In this query, the join of relations `BOOK` and `REVIEW` is created on the basis of common attribute `ISBN`. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where `Category` is *Textbook*, and then the project operation is used to display the required attributes, namely, `Book_title`, `R_ID`, and `Rating` for the tuples retrieved.

Query 8: Retrieve title, category, and price of all the books written by *Charles Smith*.

$$\pi_{\texttt{Book\_title,Category,Price}}(\sigma_{\texttt{Aname="Charles Smith"}} ((\texttt{BOOK} \bowtie_{\texttt{BOOK.ISBN=AUTHOR\_BOOK.ISBN}} \texttt{AUTHOR\_BOOK}) \bowtie_{\texttt{AUTHOR\_BOOK.A\_ID=AUTHOR.A\_ID}} \texttt{AUTHOR}))$$

In this query, the join of relations `BOOK` and `AUTHOR_BOOK` is created on the basis of common attribute `ISBN` and the resultant relation is joined with the relation `AUTHOR` on the basis of common attribute `A_ID`. After creating

a join, the select operation is applied on the resultant relation to retrieve all the tuples where `Aname` is *Charles Smith*, and then the project operation is used to display the required attributes, namely, `Book_title`, `Category` and `Price` for the tuples retrieved.

Query 9: Retrieve ID, name, URL of author, and category of the book *C ++*.

$$\pi_{A\_ID,Aname,URL,Category}(\sigma_{Book\_title=\text{“C++”}}((AUTHOR \bowtie_{AUTHOR.A\_ID=AUTHOR\_BOOK.A\_ID} AUTHOR\_BOOK) \bowtie_{AUTHOR\_BOOK.ISBN=BOOK.ISBN} BOOK))$$

In this query, the join of relations `AUTHOR` and `AUTHOR_BOOK` is created on the basis of common attribute `A_ID` and the resultant relation is joined with the relation `BOOK` on the basis of common attribute `ISBN`. After creating a join, the select operation is applied on the resultant relation to retrieve all the tuples where `Book_title` is *C ++*, and then the project operation is used to display the required attributes, namely, `A_ID`, `Aname`, `URL`, and `Category` for the tuples retrieved.

Query 10: Retrieve book title, price, author name, and URL for the publishers *Bright Publications*.

$$\pi_{Book\_title,Price,Aname,URL}(\sigma_{Pname=\text{“Bright Publications”}}(((BOOK \bowtie_{BOOK.ISBN=AUTHOR\_BOOK.ISBN} AUTHOR\_BOOK) \bowtie_{AUTHOR\_BOOK.A\_ID=AUTHOR.A\_ID} AUTHOR) \bowtie_{BOOK.P\_ID=PUBLISHER.P\_ID} PUBLISHER))$$

In this query, four relations are joined on the basis of some common attribute. First the join of the relations `BOOK` and `AUTHOR_BOOK` is created based on the common attribute `ISBN`, then the resultant relation is joined with the relation `AUTHOR` on the basis of a common attribute `A_ID`. After this, the resultant relation is joined with the fourth relation `PUBLISHER` on the basis of a common attribute `P_ID`.

Finally, the select operation is applied on the resultant relation to retrieve all the tuples where `Pname` is *Bright Publications*, and then the project

operation is used to display the required attributes, namely, `Book_title`, `Price`, `Aname`, and `URL` for the tuples retrieved.

Query 11: Retrieve the name and address of publishers who have not published any books.

$$\rho(R_1, \pi_{P\_ID}(\text{PUBLISHER}))$$

$$\rho(R_2, \pi_{P\_ID}(\text{BOOK}))$$

$$\rho(R_3, R_1 - R_2)$$

$$\rho(R_4, \pi_{\text{Pname, Address}} (R_3 \bowtie_{R_3.P\_ID=\text{PUBLISHER}.P\_ID} \text{PUBLISHER}))$$

This query uses the difference operation to retrieve ID of publishers who have not published any books. In the first step, the list of `P_ID` is retrieved from the relation `PUBLISHER` and stored in the relation R_1. In the second step, the list of `P_ID` is retrieved from the relation `BOOK` and stored in the relation R_2. The relation R_3 consists of a list of `P_ID` in the relation R_1, which is not present in the relation R_2. In other words, R_3 stores the `P_ID` of the publishers having a record in relation `PUBLISHER` who have not published any book; hence, there is no corresponding entry in relation `BOOK`. Finally, after creating join of relations `PUBLISHER` and R_3, `Pname` and `Address` for publishers who have not published any books are retrieved in relation R_4.

Query 12: Retrieve the name of all the publishers and the ISBN and title of books published by them (if any).

$$\pi_{\text{Pname, ISBN, Book\_title}}(\text{BOOK} \bowtie_{\text{BOOK.P\_ID=PUBLISHER.P\_ID}} \text{PUBLISHER})$$

In this query, the right outer join of relations `BOOK` and `PUBLISHER` is created to retrieve the name of all the publishers and the `ISBN` and title of the books published by them along with those publishers who have not published any book. In that case, the values for attributes `ISBN` and `Book_title` will have null values in the resultant relation.

Query 13: Retrieve ID and the number of books written by each author.

$$_{A\_ID}\Im_{COUNT(ISBN)}(AUTHOR\_BOOK)$$

In this query, firstly the tuples of relation AUTHOR_BOOK are grouped on the basis of A_ID and then, the aggregate function COUNT is used to count the number of occurrences of ISBN in each group.

Query 14: Retrieve ISBN and the average rating given to each book.

$$_{ISBN}\Im_{AVG(Rating)}(REVIEW)$$

In this query, firstly the tuples of relation REVIEW are grouped on the basis of ISBN and then the aggregate function AVG is used to find the average rating in each group.

Query 15: Retrieve the name of the publishers who have published all categories of books.

$$\rho(R_1, \pi_{Pname, Category}(BOOK \bowtie_{BOOK.P\_ID = PUBLISHER.P\_ID} PUBLISHER))$$

$$\rho(R_2, \pi_{Category}(BOOK))$$

$$\rho(R_3, R_1 \div R_2)$$

In this query, firstly the join of relations BOOK and PUBLISHER is created and Pname and Category are retrieved in a relation R_1. Then the unique values for Category are retrieved from the BOOK relation in another relation R_2. Finally, the relation R_1 is divided by the relation R_2 to retrieve the name of all those publishers who have published the books of all categories in the relation R_3.

4.2 RELATIONAL CALCULUS

Relational calculus, an alternative to relational algebra, is a non-procedural or declarative query language as it specifies what is to be retrieved rather than how to retrieve it. In relational calculus, queries are

expressed in the form of variables and formulas consisting of these variables. The formula specifies the properties of the resultant relation without giving specific procedure for evaluating it. There are two forms of relational calculus, namely, *tuple relational calculus (TRC)* and *domain relational calculus (DRC)*. In tuple relational calculus, the variable ranges over tuples from a specified relation and in domain relational calculus, the variable ranges over attributes from a specified relation. The Structured Query Language (SQL) is influenced by TRC; however, the Query-By-Example (QBE) is influenced by DRC.

4.2.1 Tuple Relational Calculus

The tuple relational calculus is based on **tuple variables**, which takes tuples of a specific relation as its values. A simple tuple relational calculus query is of the form

```
{T|P(T)}
```

where, `T` is a tuple variable and `P(T)` is a condition or formula for retrieving required tuples from a relation. The resultant relation of this query is the set of all tuples `T` for which the condition `P(T)` evaluates to *true*. For example, the query to retrieve all books having price greater than *$30* can be specified in the form of tuple calculus expression as

```
{T | BOOK(T) ∧ T.Price>30}
```

The condition `BOOK(T)` specifies that the tuple variable `T` is defined for the relation `BOOK`. The query retrieves the set of all the tuples `T` satisfying the condition `T.Price>30`. Note that `T.Price` refers to the attribute `Price` of tuple variable `T`. This query extracts all the attributes of the relation `BOOK`. However, if only selected attributes have to be retrieved, then the attribute list can be specified in the tuple calculus expression as

```
{T.ISBN, T.Book_title, T.Price | BOOK(T) ∧ T.Price>30}
```

This expression extracts only three attributes, namely, `ISBN`, `Book_title`, and `Price` of the tuple set satisfying the given condition.

The tuple calculus expression consists basically of three components, which are

- The relation `R` for which tuple variable `T` is defined
- A condition `P(T)` on the basis of which set of tuples is to be retrieved.
- An attribute list specifying the required attributes to be retrieved for the tuples satisfying the given condition.

Expressions and Formulas in Tuple Relational Calculus

A general expression of the tuple relational calculus consists of a tuple variable `T` and a formula `P(T)`. A formula can consist of more than one tuple variable. A formula is made up of atoms and atoms can be of any of the forms given here.

1. `R(T)`, where `T` is a tuple variable and `R` is a relation.
2. `T1.A1 opr T2.A2`, where `opr` is a comparison operator (=, ≠, <, ≤, >, ≥), T_1 and T_2 are tuple variables. A_1 is an attribute of the relation on which T_1 ranges and A_2 is an attribute of the relation on which T_2 ranges.
3. `T.A opr c` or `c opr T.A`, where `opr` is any comparison operator, `T` is a tuple variable, `A` is an attribute of the relation on which `T` ranges, and `c` is a constant in the domain of attribute `A`.

Each of the atoms evaluates to either *true* or *false* for a specific combination of tuples known as the **truth value** of an atom. A formula is built from one or more atoms concatenated with the help of the logical operators (¬, ∧,∨) by using these rules

1. An atom is a formula.

2. If F is a formula, then so is $\neg F$.
3. If F_1 and F_2 are formulae, then so are F 1 $\wedge F_2$, $F_1 \vee F$ 2 and $F_1 \Rightarrow F_2$, where $F_1 \Rightarrow F_2$ is also equivalent to $(\neg(F_1) \vee F_2)$.
4. If F is a formula, then so is $\exists T(F)$, where T is a tuple variable.
5. If F is a formula, then so is $\forall T(F)$, where T is a tuple variable.

Note that in the last two clauses, special symbols called **quantifiers**, namely, *existential quantifier* ($\exists$) and *universal quantifier* ($\forall$) are used to quantify the tuple variable T. The expression, $\forall T$, means 'for every occurrence of T' and the expression $\exists T$, means 'for some occurrence of T'. A tuple variable T is said to be **bound,** if it is quantified, that is, it appears in ($\exists T$) or ($\forall T$) clause, otherwise, it is **free**. A tuple variable T can be said to be free or bound in a formula, on the basis of the rules given here.

- A tuple variable T is free in a formula F, which is an atom.
- A tuple variable T is free or bound in a formula of the forms $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$ and $\neg F$ depending on whether it is free or bound in F_1 or F_2 (if it occurs in either of the formulas). Note that in a formula of the form $F=(F_1 \vee F_2)$, a tuple variable may be free in F_1 and bound in F_2, or vice versa. In such cases, one occurrence of the tuple variable is bound and the other is free in F.
- All free occurrences of a tuple variable T in F are bound in a formula F' of the form $F' = (\exists T)\ (F)$ or $F' = (\forall T)(F)$. The tuple variable is bound to the quantifier specified in F'.

A query can be evaluated on the basis of a given instance of the database. The truth value of a formula can be derived as given here.

1. An atom (formula) is *true* if tuple variable T is assigned a tuple from a relation R, otherwise it is *false*.
2. $\neg F$ is *true* if F is *false*, and it is *false* if F is *true*.
3. $F_1 \wedge F_2$ is *true* if both F_1 and F_2 are *true*, otherwise it is *false*.
4. $F_1 \vee F_2$ is *false* if both F_1 and F_2 are *false*, otherwise it is *true*.
5. In $F_1 \Rightarrow F_2$, F_2 is *true* whenever F_1 is *true*, otherwise it is *false*.

6. $(\exists T)(F)$ is *true* if F is *true* for some (at least one) tuple assigned to free occurrences of T in F, otherwise it is *false*.
7. $(\forall T)(F)$ is *true* if F is *true* for every tuple (universal) assigned to free occurrences of T in F, otherwise it is *false*.

Transforming the Universal and Existential Quantifiers

An expression containing a universal quantifier can be transformed into an equivalent expression containing an existential quantifier and vice versa. The steps to perform this type of transformation are

1. Replace one type of quantifier into the other and precede the quantifier by NOT (¬) operator.
2. Replace AND($\wedge$) with OR($\vee$) operator and vice versa.
3. Formula is preceded by NOT (¬) operator, as a result negated formula becomes affirmed and vice versa.

Things to Remember

In a formula, when multiple tuple variables are quantified, the evaluation takes place from inside to outside. For example, in the formula $\exists T \forall S(F)$, first $\forall S$ will evaluate and then $\exists T$ will be evaluated. The sequence of identical quantifiers can be altered with no other type of quantifier appearing in between.

Consider some examples of this transformation given here.

$$(\exists T)(F) \equiv \neg(\forall T)(\neg F)$$

$$(\forall T)(F) \equiv \neg(\exists T)(\neg F)$$

$$(\exists T)(F_1 \wedge F_2) \equiv \neg(\forall T)(\neg F_1 \vee \neg F_2)$$

$$(\forall T)(F_1 \vee F_2) \equiv \neg(\exists T)(\neg F_1 \wedge \neg F_2)$$

$$(\exists T)(\neg F_1 \vee \neg F_2) \equiv \neg(\forall T)(F_1 \wedge F_2)$$

$$(\forall T)(\neg F_1 \wedge \neg F_2) \equiv \neg(\exists T)(F_1 \vee F_2)$$

Note that the symbol ≡ represents equivalent to.

Example Queries in Tuple Relational Calculus

The queries expressed in relational algebra can also be expressed in tuple relational calculus. The difference is only in the notation used to express the queries. The queries discussed in relational algebra can be expressed in tuple relational calculus as shown here.

Query 1: Retrieve city, phone, and URL of the author whose name is *Lewis Ian*.

```
{T.City, T.Phone, T.URL | AUTHOR(T) ∧ T.Aname="Lewis Ian"}
```

In this query, tuple variable `T` for the relation `AUTHOR` and the condition to retrieve required tuples from this relation is specified after the bar (|). The tuples satisfying the given condition are assigned to `T`, and the attribute values for `City`, `Phone`, and `URL` are displayed for the tuples retrieved.

Query 2: Retrieve name, address, and phone of all the publishers located in *New York* state.

```
{T.Pname, T.Address, T.Phone | PUBLISHER(T) ∧ T.State="New Y
```

In this query, the tuple variable `T` for the relation `PUBLISHER` is defined, and the tuples satisfying the given condition are assigned to `T`. Then, the attribute values for `Pname`, `Address`, and `Phone` are displayed for the tuples retrieved.

Query 3: Retrieve title and price of all the textbooks with page count greater than *600*.

```
{T.Book_title, T.Price | BOOK(T) ∧ T.Category="Textbook" ∧ T
```

In this query, the tuple variable `T` for the relation `BOOK` is defined, and the tuples satisfying the given condition are assigned to `T`. Then, the attribute values for `Book_title` and `Price` are displayed for the tuples retrieved.

Query 4: Retrieve ISBN, title and price of the books belonging to either novel or language book category.

```
{T.ISBN, T.Book_title, T.Price | BOOK(T) ∧ (T.Category="Nove
```

In this query, the tuple variable `T` for the relation `BOOK` is defined, and the tuples satisfying the given condition are assigned to `T`. Then, the attribute values for `ISBN`, `Book_title`, and `Price` are displayed for the tuples retrieved.

Query 5: Retrieve ID, name, address, and phone of publishers publishing novels.

```
{T.P_ID, T.Pname, T.Address, T.Phone | PUBLISHER(T) ∧ (∃S)(B
```

In this query, tuple variables `T` and `S` for the relations `PUBLISHER` and `BOOK`, respectively, are defined. Of these, `T` is a free tuple variable and `S` is bound to the existential quantifier. Note that the condition `S.P_ID=T.P_ID` is a join condition for `PUBLISHER` and `BOOK` relations. The attribute values of `P_ID`, `Pname`, `Address`, and `Phone` of all the tuples satisfying the specified condition are displayed.

Query 6: Retrieve title and price of all the books published by *Hills Publications*.

```
{T.Book_title, T.Price | BOOK(T) ∧ (∃S)(PUBLISHERS(S) ∧ S.P_
```

In this query, the tuple variables `T` and `S` for the relations `BOOK` and `PUBLISHER`, respectively, are defined. Here, `T` is a free tuple variable and `S`

is bound to the existential quantifier. The two relations are joined on the basis of common attribute `P_ID`. The attribute values of `Book_title` and `Price` of all the tuples satisfying the specified condition are displayed.

Query 7: Retrieve book title, reviewers ID, and rating of all the textbooks.

```
{T.Book_title, S.R_ID, S.Rating | BOOK(T) ∧ REVIEW(S) ∧ T.Ca
```

In this query, the tuple variables `T` and `S` for the relations `BOOK` and `REVIEW`, respectively, are defined. The two relations are joined on the basis of common attribute `ISBN`. The attribute values of `Book_title`, `R_ID`, and `Rating` of all the tuples satisfying the specified condition are displayed.

Query 8: Retrieve title, category, and price of all the books written by *Charles Smith*.

```
{T.Book_title, T.Category, T.Price | BOOK(T) ∧ ((∃S)(∃P)(AUT
```

In this query, the tuple variables `T`, `S`, and `P` for the relations `BOOK`, `AUTHOR`, and `AUTHOR_BOOK`, respectively, are defined. Here, `T` is a free tuple variable, whereas `S` and `P` are bound to the existential quantifiers. The relations `BOOK` and `AUTHOR_BOOK` are joined on the basis of common attribute `ISBN` and the resultant relation is joined with the relation `AUTHOR` on the basis of common attribute `A_ID`. After this, the attribute values for `Book_title`, `Category`, and `Price` of all the tuples where `Aname` is *Charles Smith*, are displayed.

Query 9: Retrieve ID, name, URL of author, and category of the book *C ++*.

```
{T.A_ID, T.Aname, T.URL, S.Category | AUTHOR(T) ∧ BOOK(S) ∧
```

In this query, the tuple variables `T`, `S`, and `P` for the relations `AUTHOR`, `BOOK`,

and AUTHOR_BOOK, respectively, are defined. Here, T and S are free tuple variables whereas P is bound to the existential quantifier. The relations AUTHOR and AUTHOR_BOOK are joined on the basis of common attribute A_ID, and the resultant relation is joined with the relation BOOK on the basis of common attribute ISBN. After this, the attribute values for A_ID, Aname, URL, and Category of all the tuples where Book_title is *C ++* are displayed.

Query 10: Retrieve book title, price, author name, and URL for the publishers *Bright Publications*.

```
{T.Book_title, T.Price, S.Aname, S.URL | BOOK(T) ∧ AUTHOR(S)
```

In this query, the tuple variables T, S, P, and R for the relations BOOK, AUTHOR, PUBLISHER, and AUTHOR_BOOK, respectively, are defined. Among these, T and S are free tuple variables, whereas P and R are bound to existential quantifiers. Here, four relations are joined on the basis of some common attribute. First the join of the relations BOOK and AUTHOR_BOOK is created on the basis of the attribute ISBN, then the resultant relation is joined with the relation AUTHOR on the basis of the attribute A_ID. After this, the resultant relation is joined with the fourth relation PUBLISHER on the basis of the attribute P_ID. Finally, the attribute values for Book_title, Price, Aname, and URL of all tuples where Pname is *Bright Publications*, are displayed.

Query 11: Retrieve name and address of publishers who have not published any books.

```
{T.Pname, T.Address | PUBLISHER(T) ∧ (¬(∃S)(BOOK(S) ∧ T.P_ID
```

In this query, the tuple variables T and S for the relations PUBLISHER and BOOK, respectively, are defined. Here, T is a free tuple variable, whereas S is bound to existential quantifier. Note that the negation operator (¬) is

used to retrieve the tuples, which do not satisfy the join condition, that is, it retrieves the attributes `Pname` and `Address` of the publishers who have no corresponding books in relation `BOOK`.

This query can also be expressed by using universal quantifier instead of existential quantifier by following general transformation rules as shown here.

```
{T.Pname, T.Address | PUBLISHER(T) ∧ ((∀S)(¬(BOOK(S)) ∨ ¬(T.
```

Safe Expressions

An expression of tuple relational calculus containing universal quantifier, existential quantifier or negation condition may lead to a relation of infinite number of tuples. For example, consider an expression given here.

```
{T | ¬BOOK(T)}
```

This expression appears to be syntactically correct but it refers to all the tuples in the universe not belonging to `BOOK` relation, which are infinite in number. These types of expressions are known as **unsafe expressions**. Therefore, care must be taken while using quantifier symbols and negation operator in a relational calculus expression to ensure safe expressions. A **safe expression** is an expression, which results in a relation with finite number of tuples.

Safe expressions can be defined more precisely by introducing the concept of the domain of a tuple relational calculus expression, `E`. The domain of expression `E`, denoted by `Dom(E)`, is the set of all values appearing either as constant values in the expression `E`, or appearing in any of the tuples of relations referred by `E`. In other words, the domain of `E` is the set of all values that appear as constant in `E`, or appear in one or more relations whose names appear in `E`. Therefore, the domain of the

above expression includes all the values appearing in the relation BOOK. Similarly, the domain for the includes all the values appearing in the relations BOOK and PUBLISHER. Hence, an expression E, is said to be safe if all the values appearing in the resultant relation are from the domain of E, Dom(E).

The expression {T | ¬BOOK(T)} is said to be safe if all the values appearing in the resultant relation are from the domain, Dom(BOOK). However, this expression is unsafe as the resultant relation of this expression includes all the tuples in the universe not appearing in the BOOK relation, hence not belonging to the domain Dom(BOOK). All other expressions discussed in tuple relational calculus are safe expressions.

4.2.2 The Domain Relational Calculus

The **domain relational calculus** is based on domain variables, which unlike tuple relational calculus, range over the values from the domain of an attribute, rather than values for an entire tuple. A simple domain relational calculus query is of the form

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

where, T represents a set of domain variables $(x_1, x_2, \ldots, x_n)$ that ranges over domains of attributes $(A_1, A_2, \ldots, A_n)$ and P represents the formula on T. Like, tuple relational calculus, formulae in domain relational calculus are built up from atoms and an atom can be of any of the forms given here.

1. $R(x_1, x_2, \ldots, x_n)$, where R is a relation with *n* attributes and $x_1, x_2, \ldots, x_n$ are domain variables or domain constants for the corresponding *n* attributes.
2. x_1 opr x_2, where opr is a comparison operator $(=, \neq, <, \leq, >, \geq)$ and x_1 and x_2 are domain variables. Note that the domain of attributes x_1 and x_2 are compatible for comparison.

3. $x\ opr\ c$ or $c\ opr\ x$, where opr is a comparison operator, x is a domain variable and c is a constant in the domain of the attribute for which x is a domain variable.

Formulas are built from atoms by using these rules

1. An atom is a formula.
2. If F is a formula, then so is $\neg F$.
3. If F_1 and F_2 are formulae, then so are $F_1 \wedge F_2$, $F_1 \vee F_2$ and $F_1 \Rightarrow F_2$.
4. If F is a formula, then so is $\exists x(F)$, where x is a domain variable.
5. If F is a formula, then so is $\forall x(F)$, where x is a domain variable.

Like in tuple relational calculus, atoms evaluate to either *true* or *false* for a specific set of values. In rule 1, an atom (formula) is *true*, if the domain variable is assigned values corresponding to a tuple of the specified relation R. Similarly, in the remaining rules, if the domain variables are assigned values that satisfy the condition, then the atom is *true*, otherwise it is *false*.

Example Queries in Domain Relational Calculus

The queries expressed in relational algebra or tuple relational calculus can also be expressed in domain relational calculus. The difference is only in the notation used to express the queries. In domain relational calculus, the queries based on the relation AUTHOR require seven domain variables $a_1, a_2, \ldots, a_7$ to range over the domain of each attribute of the relation in order. Similarly, for the relations BOOK, PUBLISHER, AUTHOR_BOOK, and REVIEW, the number of domain variables required is eight $(b_1, b_2, \ldots, b_8)$, six $(p_1, p_2, \ldots, p_6)$, two (c_1, c_2) and three (r_1, r_2, r_3), respectively.

In domain relational calculus, the domain variables for the attributes to be displayed are specified before the bar (|) and the relations and conditions to retrieve the required tuples are specified after the bar (|). The queries discussed earlier can be expressed in domain relational calculus as

shown here.

Query 1: Retrieve city, phone, and URL of the author whose name is *Lewis Ian*.

$$\{a_4, a_6, a_7 \mid (\exists a_2)\ (\text{AUTHOR}(a_1, a_2, a_3, a_4, a_5, a_6, a_7) \wedge a_2 = \text{“L}$$

The required attributes `City`, `Phone`, and `URL` to be displayed are specified by free domain variables a_4, a_6, and a_7, respectively, for the relation `AUTHOR`. Existential quantifier quantifies the variable a_2 for which condition is specified.

Query 2: Retrieve name, address, and phone of all the publishers located in *New York* state.

$$\{p_2, p_3, p_5 \mid (\exists p_4)\ (\text{PUBLISHER}(p_1, p_2, p_3, p_4, p_5, p_6) \wedge p_4 = \text{“N}$$

The required attributes `Pname`, `Address`, and `Phone` to be displayed are specified by free domain variables p_2, p_3, and p_5, respectively, for the relation `PUBLISHER`. Existential quantifier quantifies the variable p_4 for which condition is specified.

Query 3: Retrieve title and price of all the textbooks with page count greater than *600*.

$$\{b_2, b_4 \mid (\exists b_3)\ (\exists b_7)\ (\text{BOOK}(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8) \wedge b$$

The required attributes `Book_title` and `Price` to be displayed are specified by free domain variables b_2 and b_4, respectively, for the relation `BOOK`. Existential quantifiers quantify the variables b_3 and b_7 for which conditions are specified.

Query 4: Retrieve ISBN, title and price of the books belonging to either novel or language book category.

$$\{b_1, b_2, b_4 \mid (\exists b_3)\ (\text{BOOK}(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8) \wedge (b_3$$

The required attributes `ISBN`, `Book_title`, and `Price` to be displayed are specified by free domain variables b_1, b_2, and b_4, respectively, for the relation `BOOK`. Existential quantifier quantifies the variable b_3 for which conditions are specified.

Query 5: Retrieve ID, name, address, and phone of publishers publishing novels.

$$\{p_1, p_2, p_3, p_5 \mid (\exists b_3)\ (\exists b_8)\ (\text{BOOK}(b_1, b_2, b_3, b_4, b_5, b_6, b_7,$$

The required attributes `P_ID`, `Pname`, `Address`, and `Phone` to be displayed are specified by free domain variables p_1, p_2, p_3, and p_5, respectively, for the relation `PUBLISHER`. Existential quantifiers quantify the variables b_3 and b_8 for which conditions are specified. The variable p_1 appearing in a condition is not quantified as it is appearing as a free domain variable before the bar (|).

Note that the condition $b_8 = p_1$ is a join condition as it relates two domain variables ranging over common attributes from two different relations. This condition creates join of two relations `BOOK` and `PUBLISHER`.

Query 6: Retrieve title and price of all the books published by *Hills Publications*.

$$\{b_2, b_4 \mid (\exists p_2)\ (\exists p_1)\ (\exists b_8)\ (\text{BOOK}(b_1, b_2, b_3, b_4, b_5, b_6, b_7,$$

The required attributes `Book_title` and `Price` to be displayed are specified by free domain variables b_2 and b_4, respectively, for the relation `BOOK`. Existential quantifiers quantify the variables p_2, p_1, and b_8 for which conditions are specified. Note that the condition $p_1 = b_8$ is a join condition as it creates a join of two relations `PUBLISHER` and `BOOK`.

Query 7: Retrieve book title, reviewers ID, and rating of all the textbooks.

$$\{b_2, r_1, r_3 \mid (\exists b_3)\ (\exists b_1)\ (\exists r_2)\ (\text{BOOK}(b_1, b_2, b_3, b_4, b_5, b_6,$$

The required attributes `Book_title`, `R_ID`, and `Rating` to be displayed are specified by free domain variables b_2 for the relation BOOK, r_1 and r_3 for the relation REVIEW, respectively. Existential quantifiers quantify the variables b_3, b_1, and r_2 for which conditions are specified. Note that, the condition $b_1 = r_2$ is a join condition as it creates a join of two relations BOOK and REVIEW.

Query 8: Retrieve title, category, and price of all the books written by *Charles Smith*.

$$\{b_2, b_3, b_4 \mid (\exists b_1)\ (\exists c_2)\ (\exists c_1)\ (\exists a_1)\ (\exists a_2)\ (\text{BOOK}(b_1, b_2, b_3,$$

The required attributes `Book_title`, `Category`, and `Price` to be displayed are specified by free domain variables b_2, b_3, and b_4, respectively, for the relation BOOK. Existential quantifiers quantify the variables b_1, c_2, c_1, a_1, and a_2 for which conditions are specified. The conditions $b_1 = c_2$ and $c_1 = a_1$ creates a join of relations BOOK, AUTHOR_BOOK, and AUTHOR.

Query 9: Retrieve ID, name, URL of author, and category of the book *C++*.

$$\{a_1, a_2, a_7, b_3 \mid (\exists c_1)\ (\exists b_1)\ (\exists c_2)\ (\exists b_2)\ (\text{BOOK}(b_1, b_2, b_3, b_4,$$

The required attributes `A_ID`, `Aname`, `URL`, and `Category` to be displayed are specified by free domain variables a_1, a_2, a_7 for the relation AUTHOR and b_3 for the relation BOOK, respectively. Existential quantifiers quantify the variables c_1, b_1, c_2, and b_2 for which conditions are specified. Note that variable a_1 appearing in a condition is not quantified as it is appearing as free domain variable before the bar (|). The conditions $a_1 = c_1$ and $c_2 =$

b_1 creates a join of relations AUTHOR, AUTHOR_BOOK, and BOOK.

Query 10: Retrieve book title, price, author name, and URL for the publishers *Bright Publications*.

$$\{b_2, b_4, a_2, a_7 \mid (\exists a_1)\ (\exists c_1)\ (\exists c_2)\ (\exists b_1)\ (\exists b_8)\ (\exists p_1)\ (\exists p_2)\ (BC$$

The required attributes Book_title, Price, Aname, and URL to be displayed are specified by free domain variables b_2, b_4 for the relation BOOK and a_2, a_7 for the relation AUTHOR, respectively. Existential quantifiers quantify the variables a_1, c_1, c_2, b_1, b_8, p_1, and p_2 for which conditions are specified. The conditions $a_1 = c_1$, $c_2 = b_1$, and $b_8 = p_1$ creates a join of relations AUTHOR, AUTHOR_BOOK, BOOK, and PUBLISHER.

Query 11: Retrieve name and address of publishers who have not published any books.

$$\{p_2, p_3 \mid (\exists p_1)\ (\text{PUBLISHER}(p_1, p_2, p_3, p_4, p_5, p_6) \wedge (\neg(\exists b_8)\ (b$$

The required attributes Pname and Address to be displayed are specified by free domain variables p_2, and p_3, respectively for the relation PUBLISHER. Existential quantifiers quantify the variables p_1 and b_8 for which conditions are specified. The negation operator (¬) is used to retrieve those publisher IDs who have not published any book.

This query can also be expressed by using universal quantifier instead of existential quantifier by following general transformation rules as shown here.

$$\{p_2, p_3 \mid (\exists p_1)\ (\text{PUBLISHER}(p_1, p_2, p_3, p_4, p_5, p_6) \wedge ((\forall b_7)\ (\neg$$

4.3 EXPRESSIVE POWER OF RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

As discussed earlier, the relational algebra provides a procedure for solving a problem, whereas relational calculus simply describes what the requirement is. However, relational algebra and relational calculus are logically equivalent. That is, for every safe relational calculus expression there exists equivalent relational algebra expression and vice versa. In other words, there is one-to-one mapping between the two and the difference lies only in the way the query is expressed.

The expressive power of relational algebra is frequently used as a metric to measure the power of any relational database query language. A query language is said to be **relationally complete** if it is at least as powerful as relational algebra, that is, if it can express any query expressed in relational algebra. Relational completeness has become an important basis for comparing the expressive power of various relational database query languages. Most of the relational database query languages are relationally complete. In addition, they have more expressive power than relational algebra or relational calculus with the inclusion of various advanced operations such as aggregate functions, grouping, and ordering.

SUMMARY

1. Relational algebra is a procedural query language that consists of a set of operations that take one or two relations as input, and result into a new relation as an output.
2. Relational algebra operations are divided into two groups. One group consists of operations developed specifically for relational databases such as select, project, rename, join, and division. The other group includes the set-oriented operations such as union, intersection, difference, and cartesian product.
3. The select operation retrieves all those tuples from a relation that satisfy a specific condition. The Greek letter sigma (σ) is used as a select operator.
4. The project operation is used to select some required attributes from a relation while discarding the other attributes. The Greek letter pi

(π) is used as project operator.

5. Rename operation is used to provide name to the relation obtained after applying any relational algebra operation. The Greek letter rho (ρ) is used as a rename operator.
6. The union operation, denoted by ∪, returns a third relation that contains tuples from both or either of the operand relations.
7. The intersection operation, denoted by ∩, returns a third relation that contains tuples common to both the operand relations.
8. The difference operation, denoted by – (minus), returns a third relation that contains all tuples present in one relation, which are not present in the second relation.
9. The cartesian product, also known as cross product or cross join, returns a third relation that contains all possible combinations of the tuples from the two operand relations. The cartesian product is denoted by the symbol ×.
10. A join can be defined as a cartesian product followed by select and project operations. The join operation joins two relations to form a new relation on the basis of one common attribute present in the two operand relations.
11. A common case of the join operation in which the join condition consists only of equality condition is known as equijoin.
12. The equijoin operation results in two attributes in the resulting relation having exactly the same value. If one of the two identical attributes is removed from the result of equijoin, it is known as natural join.
13. An inner join selects only those tuples from both the joining relations that satisfy the joining condition. The outer join, on the other hand, selects all the tuples satisfying the join condition along with the tuples for which no tuples from the other relation satisfy the join condition.
14. There are three types of outer joins, namely, left outer join, right outer join, and full outer join.
15. Some of the requirements like finding total amount, average price, which are mathematical in nature, cannot be expressed by basic

relational algebra operations. For such type of queries, aggregate functions are used. Aggregate functions take a collection of values as input, process them, and return a single value as the result.

16. Relational calculus, an alternative to relational algebra, is a non-procedural or declarative query language as it specifies what is to be retrieved rather than how to retrieve it. In relational calculus, queries are expressed in the form of variables and formulas consisting of these variables.
17. There are two forms of relational calculus, namely, tuple relational calculus (TRC) and domain relational calculus (DRC).
18. The tuple relational calculus is based on tuple variables, which takes tuples of a specific relation as its values.
19. An expression of tuple relational calculus containing universal quantifier, existential quantifier or negation condition may lead to a relation of infinite number of tuples. These types of expressions are known as unsafe expressions. A safe expression, on the other hand, results in a relation with finite number of tuples.
20. The domain relational calculus is based on domain variables, which unlike tuple relational calculus, range over the values from the domain of an attribute, rather than values for an entire tuple.
21. The expressive power of relational algebra is frequently used as a metric to measure the power of any relational database query language. A query language is said to be relationally complete if it is at least as powerful as relational algebra, that is, if it can express any query expressed in relational algebra.

KEY TERMS

- Select operation
- Project operation
- Rename operation
- Union operation
- Intersection operation
- Difference operation
- Cartesian product

- Join operation
- Equijoin
- Natural join
- Outer join
- Left outer join
- Right outer join
- Full outer join
- Division
- Aggregate functions
- Tuple relational calculus
- Tuple variable
- Atom
- Formula
- Existential quantifier
- Universal quantifier
- Safe expression
- Domain relational calculus
- Domain variable

EXERCISES

A. Multiple Choice Questions

1. Which of the following is the operation developed specifically for the relational databases?
    1. Union
    2. Cartesian product
    3. Division
    4. Difference
2. Which of the following is the set-oriented operation?
    1. Select
    2. Difference
    3. Division
    4. Project
3. Which of the following is the binary operation?

    1. Union
    2. Join
    3. Cartesian product
    4. All of these
4. Which of the following is true for right outer join?
    1. It includes all the tuples from both the relations satisfying the join condition along with all the tuples in the right relation that do not have a corresponding tuple in the left relation.
    2. It includes all the tuples from both the relations satisfying the join condition along with all the tuples in the left relation that do not have a corresponding tuple in the right relation.
    3. It selects all the tuples satisfying the join condition along with the tuples for which no rows from the other relation satisfy the join condition.
    4. None of these
5. Which of the following aggregate functions is used for counting tuples or values in a relation?
    1. `Avg`
    2. `Count`
    3. `Max`
    4. `Min`
6. Which of the following is the component of tuple calculus expression?
    1. The relation `R` for which tuple variable `T` is defined.
    2. A condition `P(T)` on the basis of which set of tuples is to be retrieved.
    3. An attribute list specifying the required attributes to be retrieved for the tuples satisfying the given condition.
    4. All of these
7. The truth value of a formula can be derived as
    1. $\neg F$ is *true* if F is *false*, and it is *false* if F is *true*.
    2. $F_1 \wedge F_2$ is *true* if both F_1 and F_1 and F_2 are *true*, otherwise it is *false*.
    3. $F_1 \vee F_1$ and F_{52} are *false*, otherwise it is *true*.

4. All of these

8. Which of the following is symbol for existential quantifier?
    1. $\forall$
    2. $\exists$
    3. $\neg$
    4. $\equiv$
9. Which of the following statements is true?
    1. $(\forall T)(F) \equiv \neg(\exists T)(F)$
    2. $(\exists T)(F_1 \wedge F_2) \equiv \neg(\forall T)(\neg F_1 \vee \neg F_2)$
    3. $(\forall T)(F_1 \vee F_2) \equiv (\exists T)(\neg F_1 \wedge \neg F_2)$
    4. $(\forall T)(F_1 \wedge F_2) \equiv \neg(\exists T)(F_1 \vee F_2)$
10. Which of the following is not a rule for forming formula in domain relational calculus?
    1. An atom is a formula.
    2. If F is a formula, then so is $\neg F$.
    3. If F_1 and F_2 are formulae, then so are $F_1 \wedge F_2$, $F_1 \vee F_2$ and $F_1 \Rightarrow F_2$.
    4. If F is a formula, then so is $\exists T(F)$, where T is a tuple variable.

B. Fill in the Blanks

1. The _____ is used to select some required attributes from a relation while discarding the other attributes.
2. _____ is used to provide name to the relation obtained after applying any relational algebra operation.
3. The _____ can be defined on any two relations, that is, they need not be union compatible.
4. Relational algebra expression that uses intersection operation can be rewritten by replacing the intersection operation with a pair of _____ operation.
5. A common case of the join operation $R_1 \bowtie R_2$ is one in which the join condition consists only of equality condition, also known as _____.
6. The _____ is useful for queries of the form ‘for all objects having all the specified properties’.

7. A tuple variable `T` is said _____, if it is quantified, that is, it appears in `(∃T)` or `(∀T)` clause, otherwise, it is _____.
8. A _____ is an expression, which results in a relation with a finite number of tuples.
9. The _____ is based on domain variables, which unlike tuple relational calculus, range over the values from the domain of an attribute, rather than values for an entire tuple.
10. A query language is said to be _____ if it is at least as powerful as relational algebra, that is, if it can express any query expressed in relational algebra.

C. Answer the Questions

1. Give similarities and differences between relational algebra and relational calculus.
2. What are the various unary and binary operations in relational algebra? Why are they called so?
3. Discuss various unary operations of relational algebra with example.
4. What do you mean by union compatibility of relations? Explain with example.
5. Discuss various set operations of relational algebra with example.
6. What is the purpose of rename operator in relational algebra?
7. What is the role of join operations in relational algebra? Differentiate between equijoin and natural join.
8. What are outer join operations and how are they different from inner join operations? Discuss various types of outer join operations.
9. What is cartesian product? How is it different from join operation?
10. Discuss division operation in detail with example.
11. What is relational calculus? What are its two forms? Differentiate between them.
12. What information is required to be specified in tuple calculus expression? Explain with example.
13. What are quantifiers and what are its types? How is existential quantifier transformed into universal quantifier and vice versa? Give suitable examples.

14. When is an expression in tuple relational calculus said to be unsafe? How can safe expressions be defined? Explain with example.
15. Differentiate between the following:
    1. Tuple variable and domain variable
    2. Atom and formula
16. What are the different forms of atom in tuple relational calculus and domain relational calculus? Give rules for forming formulas in tuple relational calculus and domain relational calculus.
17. Convert the following domain calculus expression into corresponding, English statement, relational algebra expression and tuple calculus expression.

    $\{b_2, r_1, r_3 \mid (\exists b_3)\ (\exists b_1)\ (\exists r_2)\ (Rel\_1(b_1, b_1, r_2, r_3) \wedge b_1 = r_2 \wedge b_3 = \text{“ABC”})\}$

18. What do you understand by the expressive power of relational algebra? How are relational algebra and relational calculus logically equivalent? When is a query language said to be relationally complete?

D. Practical Questions

1. 1. Consider the following relational schema for the SALES database:

   ```
   CUSTOMER(CustNo, CName, City)
   ORDER(OrderNo, OrderDate, CustNo, Amount)
   ORDER_ITEM(OrderNo, ItemNo, Qty)
   ITEM(ItemNo, UnitPrice)
   ```

   On the basis of this relational schema, write the following queries in relational algebra, tuple calculus, and domain calculus.

   1. Retrieve number and date of orders placed by customers residing at *Atlanta* city.
   2. Retrieve number and unit price of items for which order of quantity greater than *50* is placed.
   3. Retrieve order number, date, and item number for the order of

items having unit price greater than *20*.
4. Retrieve details of customers who have placed order for the item number *I010*.
5. Retrieve number and unit price of items for which order is placed by the customer number *C001*.

2. Given the relational schemes:

   `ENROL(S#, C#, Section)`—S# represents student number

   `TEACH(Prof, C#, Section)`—C# represents course number

   `ADVISE(Prof, S#)`—Prof is thesis advisor of S#

   `PRE_REQ(C#, Pre_C#)`—Pre_C# is prerequisite course

   `GRADES(S#, C#, Grade, Year)`

   `STUDENT(S#, Sname)`—Sname is student name

   Express the following queries in relational algebra, tuple calculus, and domain calculus.

   1. List all students taking courses with *Zeba*.
   2. List all students taking at least one course that their advisor teaches.
   3. List those professors who teach more than one section of the same course.

3. Consider the relational database:

   ```
   EMPLOYEE(Empname, Street, City)

   WORKS(Empname, Companyname, Salary)

   COMPANY(Companyname, City)

   MANAGES(Empname, Managername)
   ```

   Give an expression in the relational algebra for each request.

1. Find the names of all employees who work for *First Bank Corporation*.
2. Find the names, street addresses, and cities of residence of all employees who work for *First Bank Corporation* and earn more than *200,000* per annum.
3. Find the names of all employees in this database who live in the same city as the company for which they work.
4. Find the names of all the employees who earn more than every employee of *Small Bank Corporation*.
5. Find the number of employees working in each company.
6. Find average, maximum, and minimum salary for each company.

4. Consider the following relational schema for the `BANK` database:

```
BRANCH(BranchID, Bname, City, Phone)

ACCOUNT(AccountNo, Aname, AType, BranchID, Balance)

TRANSACTION(TID, T_Date, T_Type, AccountNo, Amount)
```

   On the basis of this relational schema, write the following queries in relational algebra.

   1. Retrieve ID and name of all the branches located in *Seattle* city.
   2. Retrieve ID, type, and amount of all the transactions of withdrawal type.
   3. List number and type of all accounts opened in the branch having ID *B010*.
   4. List number and name of account holders withdrawing amount greater than 10,000 on 31st March, 2007.
   5. List number and name of account holders having savings account in the city *Los Angeles*.
   6. Calculate average balance of accounts present in the *AX International Bank* in *New York*.
   7. Calculate maximum and minimum balance of accounts in each city.

5. Express queries (a) to (d) of question 3 in tuple calculus and domain

calculus.

6. Express queries (a) to (e) of question 4 in tuple calculus and domain calculus.
7. R_1 and R_2 are two given relations:

   R_1

A	**B**
A1	B1
A2	B2
A3	B3
A4	B4

X	**Y**
A1	B1
A7	B7
A2	B2
A4	B4

   Find Union, Intersection, and Set difference.