UNIT II

SYLLABUS: Relational Model: Structure of Relational Databases, Fundamental Relational-Algebra Operations, Additional Relational – Algebra Operations, Extended Relational - Algebra Operations, Null Values, Modification of the Databases.

Structured Query Language: Data Definition, Basic Structure of SQL Queries, Set Operations, Aggregate Functions, Null Values, Nested Sub-queries, Complex Queries, Views, Modification of the Database, Joined Relations.

RELATIONAL MODEL

Structure of Relational Database: A relational database consists of a collection of *tables*. Each table is assigned a unique name. A row in a table represents a *relationship* among a set of values. For basic structure, consider the *deposit* table of the following figure:

Branch-Name	Account-Number	Customer-Name	Balance
A	101	Dileep	2500
B	215	Kapil	6200
C	305	Ravi	3400
D	217	Raghu	7400
E	786	Bharath	4000

Table: The *deposit* relation

Domain: A set of permitted values of each attribute of a table is called as domain. The above table has four attributes: *Branch Name, Account Number, Customer Name*, and *Balance*. For each attribute, there is a set of permitted values, called the Domain of that attribute.

For example, for the attribute *Branch Name*, the *domain* is the set of all branch names. Similarly for the *Balance*, the domain is the all balance values.

Relation: A relation is subset of a Cartesian product of a list of domains.

Ex: In the above relation Deposit, there are four attributes. D1 is the domain of branch Names, D2 is the set of all account numbers, D3 is the set of all customer names and D4 is the set of all balance values.

Every row in a deposit relation consists of 4-tuples (v1, v2, v3, v4) where v1 is the branch Name (i.e., v1 is in domain D1), v2 is an account number (i.e., v2 is in domain D2), v3 is the customer name (i.e., v3 is in domain D3) and v4 is the balance (i.e., v4 is in domain D4). In general, deposit will contain only a subset of all possible rows.

i.e., Deposit is a subset of D1 X D2 X D3 X D4

TUPLE: Row of a given flat file (table) is called a tuple of the relation.

Ex: In the above relation, there are five tuples (rows), i.e., the given relation cardinality is five.

Degree of a Relation: The number of the attributes in a given relation is called degree of the relation.

Ex: The degree of the given relation is 4.

KEYS IN A RELATIONAL DATABASE

KEY: A data item (attribute) is used to identify or locate a record is called a key (entity identifier).

VARIOUS_KEYS_IN_A_RELATIONAL_DATABASE

1) Primary key 2) Candidate key 3) Alternate key 4) Secondary key 5) Super key 6) Foreign key

1) **PRIMARY KEY:** The primary key is defined as a data item (attribute) which uniquely identifies a record (one row) in a relation.

Roll-No	Name	Date-of-Birth	Second-Language	Division
95	Swetha	31st Dec	Hindi	First
96	Dhatri	26th Jan	Sanskrit	Second
97	Shivani	15th Aug	Telugu	First
98	Kavya	01st Nov	Hindi	Second
99	Jagruti	29th Feb	Sanskrit	First
100	Shravani	26th Jan	Telugu	First

The *student* Relation

In the above relation *student*, we can choose roll number attribute as primary key because for each row, there is a unique value of Roll number attribute i.e., same roll number is not repeated in another rows.

Therefore *Roll Number* is primary key for given relation.

2) **CANDIDATE KEY:** In a relation in which there is more than one attribute combination possessing the unique identification property, all the various combinations of attributes, which serve as a primary key are called the candidate keys of the given relation.

Note: Subset of Candidate keys is not a Primary Key.

Ex:-In the given relation student:

{Roll number} can identify each row uniquely. So it is one candidate key.

{Second Language, Date of Birth} can identify each row uniquely; it is one more candidate key.

{Date of Birth, Division} can identify each row uniquely so it is one more candidate key.

{Roll Number, Date of Birth} can identify each row uniquely. But subset of this key is {Roll Number} {Date of Birth}.

So Here {Roll Number] is a primary key. As per definition, subset of candidate keys not a primary key. So {Roll Number, Date of Birth} is not a candidate key.

3) **ALTERNATE KEY:** A candidate key that is not the primary key, called as alternate key.

Ex: In a given relation student

{Roll number} is unique for every student. Similarly all student names are unique (no students having the same name). Here we can choose either Roll number or name as a primary key.

In such a case we may arbitrarily choose one of the candidates. Say *Roll Number* as the primary key for the relation. A candidate key that is not the primary key, such *Name* in the relation student is called alternate key.

i.e., if Roll number is primary key, then Name is alternate key. If Name is primary key, then Roll Number is alternate key.

4) **SECONDARY KEY:** System may also use a key which does not identify a unique record or tuple but which identifies all those which have certain property. This is referred to as a Secondary Key.

Ex: In a given relation STUDENT the value of the attribute "second language" may be used as a secondary key. This key could be used to identify those entities (students), who belong to second language Hindi, Telugu or Sanskrit.

Second-Language	Roll-No
Hindi	95
	98
Sanskrit	96
	99
Telugu	97
	100

Secondary Keys

5) **SUPER_KEY:** More than one attribute combined together for unique identification of the record is defined as a Super Key As shown in figure below, neither supplier no. (S#), nor product no. (P#) are enough to identify the each row. To get unique information for each row, we need combined attributes s#, p#. i.e. {s# + p#} is a Super key (or) concatenate key.

S#	P#	Qty
S1	P1	500
S1	P2	700
S1	P3	450
S2	P4	920
S3	P1	650
S3	P5	400

Super Keys

6) **FOREIGN KEY:** A foreign key is an attribute or group of fields in a database record that points to a key field or group of fields forming a key of another database record in a different table. Usually a foreign key in one table refers to the primary key of another table. This way references can be made to link. For Ex: For an Account relation Customer_ID is considered as the foreign key.

QUERY LANGUAGES

A query language is a language in which a user requests information from the database. These languages are typically of a higher level than standard programming languages. Query languages can be categorized as being either

a) Procedural

b) Non-procedural.

In a procedural language, the user instructs the system to perform a sequence of operations on the database to compute the desired result. Relational algebra is called as procedural query language. In a nonprocedural language, the user describes the information desired without giving a specific procedure for obtaining the information. Tuple relational calculus and domain relational calculus is belong to nonprocedural query language. The relational algebra is procedural while the relational calculus and the domain relational calculus are nonprocedural.

RELATIONAL ALGEBRA

The relational algebra is a *procedural* query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. The fundamental operations in the relational algebra are *select, project, union, set difference, Cartesian product,* and *rename*. In addition to the fundamental operations, there are several other operations—namely, *set intersection, natural join,* and *assignment*. We shall define these operations in terms of the fundamental operations.

Fundamental Operations

The select, project, and rename operations are called *unary* operations, because they operate on one relation. The other three operations operate on pairs of relations and are, therefore, called *binary* operations.

Consider the Two relations STUDENT, QUARTERLY. The STUDENT relation is used to describe the complete personal information about student, his roll no, name, date of birth, 2nd language. Another relation QUARTERLY used to describe the students marks in 3 subjects with roll no's.

STUDENT

Roll-No	Name	Date-of-Birth	Second-Language
1	Sunny	01-07-70	Hindi
2	Rashni	15-08-72	Sanskrit
3	Anthra	29-01-71	Hindi
4	Nasreen	31-12-70	Telugu

QUARTERLY

Roll-No	Maths	Physics	Computers
1	72	85	90
2	65	74	68
3	97	94	96
4	87	93	72

The following are queries based on relational algebra to obtain required information from stored relational database.

1) **The SELECT Operation:** The Select operation selects tuples that satisfy a given predicate. A lower case Greek letter sigma (σ) is used to denote Select operation. Predicate appears as subscript to r.

The argument relation is given in parentheses. The General form of selection operation is:

$$\sigma_{\text{predicate}} (\text{relation})$$

All comparisons =, #, <, >, <=, >= were allowed in the select operation predicate. Furthermore, several predicates may be combined into a large predicate using the connectives *and* (∧) and *or* (V).

Ex: 1) List out the complete information about all students whose 2nd language is Hindi

$$\sigma_{\text{2nd-language = Hindi}} (\text{STUDENT})$$

Result of the above Query is:

Roll-No	Name	Date-of-Birth	Second-Language
1	Sunny	01-07-70	Hindi
3	Anthra	29-01-71	Hindi

Ex: 2) Display all students with Roll no with their marks who secured more than 90 in all the three subjects

$$\sigma_{((\text{Maths} > 90) \wedge (\text{Physics} > 90) \wedge (\text{Computers} > 90))} (\text{QUARTERLY})$$

Result of the above Query is:

Roll-No	Maths	Physics	Computers
3	97	94	96

2) The PROJECT Operation: The projection of a relation is defined as a projection of all its tuples over some set of attributes. i.e., it yields a "vertical subset" of the relation. The projection operation is used to either reduce the number of attributes in the resultant relation or to reorder attributes. Projection is denoted by Greek letter pi (Π). We list these attributes that we wish to appear in the result as a subscript to. The argument relation follows in parenthesis.

General form of projection operation is

$$\Pi_{\text{List-of-attributes (Predicate)}} (\text{relation})$$

Ex: 1) List out all Roll Nos. and their Computer Marks

$$\Pi_{\text{Roll No, Computers}} (\text{QUARTERLY})$$

Result of the above Query is:

Roll-No	Computers
1	90
2	68
3	96
4	72

2) Display all the student names with their date of birth whose 2nd language is Hindi.

$$\Pi_{\text{Name, Date-of-birth}} (\sigma_{\text{2nd-language = Hindi}}) (\text{STUDENT})$$

Result of the above query is:

Name	Date-of-Birth
Sunny	01-07-70
Anthra	29-01-71

3) What is the Date of Birth of Rashni?

$$\Pi_{\text{Date-of-Birth}} (\sigma_{\text{Name = Rashni}}) (\text{STUDENT})$$

Result of the above query is:

Date-of-Birth
15-08-72

4) Find all Roll Nos who secured more than 90 marks in Computers

$$\Pi_{\text{Roll-No}} (\sigma_{\text{computers} > 90}) (\text{QUARTERLY})$$

Result of the above query is:

Roll-No
1
3

3) **The RENAME Operation:** Unlike relations in the DB, the results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful to be able to give them names; the **rename** operator, denoted by the lowercase Greek letter rho (ρ), lets us do this. Given a relational-algebra expression *E*, the following expression returns the result of expression *E* under the name *x*.

$$\rho_x (E)$$

Ex: $\rho_{\text{teacher}} (instructor)$

A relation *r* by itself is considered a (trivial) relational-algebra expression. Thus, we can also apply the rename operation to a relation *r* to get the same relation under a new name.

A Second form of the rename operation is as follows: Assume that a relational algebra expression *E* has arity *n*. Then, the following expression returns the result of expression *E* under the name *x*, and with the attributes renamed to *A*1, *A*2, . . . , *An*.

$$\rho_{x(A1,A2,\ldots,An)} (E)$$

Ex: $\rho_{\text{teacher (id, name, sal)}} (instructor)$

4) **CARTESIAN PRODUCT Operation:** This operation allows us to combine information from several relations. Thus operation is denoted by a cross(X). Thus operation is a binary. Suppose r1 and r2 are two relations, Cartesian product of these two relations can be written are r1 X r2.

In other words, Cartesian product of two relations is the concatenation of tuples belonging to the two relations. A new resultant relation scheme is created consisting of all possible combinations of tuples.

If there are m tuples in relation r1, and n tuples in relation r2, then there is m x n ways of choosing a pair of tuples. One tuple from each relation is chosen, so there are n1 x n2 tuples in r.

Ex: (i) Find student names and their Computer marks.

To list out the Name, Computer Marks, we have to refer both the relations, STUDENT & QUARTERLY. Student name is an attribute from STUDENT relation and Computers is an attribute from QUARTERLY relation. Referring 2 relations is denoted by "X"

STUDENT X QUARTERLY

Roll-No	Name	Date-of-Birth	Second-Language	Roll-No	Maths	Physics	Computers
1	Sunny	01-07-70	Hindi	1	72	85	90
1	Sunny	01-07-70	Hindi	2	65	74	68
1	Sunny	01-07-70	Hindi	3	97	94	96
1	Sunny	01-07-70	Hindi	4	87	93	72
2	Rashni	15-08-72	Sanskrit	1	72	85	90
2	Rashni	15-08-72	Sanskrit	2	65	74	68
2	Rashni	15-08-72	Sanskrit	3	97	94	96
2	Rashni	15-08-72	Sanskrit	4	87	93	72
3	Anthra	29-01-71	Hindi	1	72	85	90
3	Anthra	29-01-71	Hindi	2	65	74	68
3	Anthra	29-01-71	Hindi	3	97	94	96
3	Anthra	29-01-71	Hindi	4	87	93	72
4	Nasreen	31-12-70	Telugu	1	72	85	90
4	Nasreen	31-12-70	Telugu	2	65	74	68
4	Nasreen	31-12-70	Telugu	3	97	94	96
4	Nasreen	31-12-70	Telugu	4	87	93	72

Information is retrieved from the above relation STUDENT X QUARTERLY. By selecting the common attribute in the same relation i.e., in given two relations, Roll No is the common attribute.

STUDENT.ROLLNO = QUARTERLY.ROLLNO

$\Pi_{\text{Name, Computers}} (\sigma_{\text{STUDENT.ROLL-NO = QUARTERLY.ROLL-NO}})$ (STUDENT X QUARTERLY)

Name	Computers
Sunny	90
Rashni	68
Anthra	96
Nasreen	72

(ii) Find the Student Roll No, Date of Birth, 2nd Language, Maths, Physics and Computer Marks.

$\sigma_{\text{STUDENT.ROLL NO = QUARTERLY.ROLL NO.}}$ (STUDENT X QUARTERLY)

Result of the query is:

Roll-No	Name	Date-of-Birth	Second-Language	Roll-No	Maths	Physics	Computers
1	Sunny	01-07-70	Hindi	1	72	85	90
2	Rashni	15-08-72	Sanskrit	2	65	74	68
3	Anthra	29-01-71	Hindi	3	97	94	96
4	Nasreen	31-12-70	Telugu	4	87	93	72

5) **UNION Operation:** The union of two relations r and s is denoted by r ∪ s. The output relation Z= r ∪ s has tuples drawn from r and s. The result relation Z contains tuples that are in either r or s or in both of them. The duplicate tuples are eliminated.

For a union operation $r \cup s$ to be valid, we require that two conditions hold:

- The relations *r* and *s* must be of the same arity. i.e., they must have the same number of attributes.
- The domains of the *i*th attribute of *r* and the *i*th attribute of *s* must be the same, for all *i*.

Note that *r* and *s* can be either database relations or temporary relations that are the result of relational algebra expressions.

As an example, consider the relations CULTURAL (name, class) and SPORTS (name, class). These two relations represent information about all cultural competition winners & sports winners separately.

CULTURAL

Name	Class
Kavya	MPC III
Lahari	MSC II
Bhanu	MCA II
Zeba	MPC III
Nisha	MBA II
Hima	MPC III

SPORTS

Name	Class
Deepta	MCA III
Lahari	MSC II
Hima	MPC III
Archana	MBA II
Sheela	MPC III

Ex: Find all the student Names of MPC III who won cultural competition or sports competition or both competitions.

$$\Pi_{\text{Name}} (\sigma_{\text{Class = 'MPC III'}}) (\text{CULTURAL}) \cup \Pi_{\text{Name}} (\sigma_{\text{Class = MPC III}}) (\text{SPORTS})$$

NAME
Kavya
Zeba
Hima

NAME
Hima
Sheela

NAME
Kavya
Zeba
Hima
Sheela

6) **SET-DIFFERENCE Operation:** The difference between two relations r and s is r - s. The result relation contains the set of tuples belonging to r and not in s.

Ex: Find Student Names of MPC III who won cultural prizes but not sports.

$$\Pi_{\text{Name}} (\sigma_{\text{Class = 'MPC III'}}) (\text{CULTURAL}) - \Pi_{\text{Name}} (\sigma_{\text{Class = MPC III}}) (\text{SPORTS})$$

NAME
Kavya
Zeba
Hima

NAME
Hima
Sheela

NAME
Kavya
Zeba

Formal Definition of the Relational Algebra: The fundamental operations of relational algebra allow us to give a complete definition of an expression in the relational algebra. A basic expression in the relational algebra consists of either one of the following:

- A relation in the database
- A constant relation

A constant relation is written by listing its tuples within { }, for example
{(22222, Einstein, Physics, 95000), (76543, Singh, Finance, 80000)}.
A general expression in the relational algebra is constructed out of smaller sub expressions. Let $E1$ and $E2$ be relational-algebra expressions. Then, the following are all relational-algebra expressions:

- ✓ $E1 \cup E2$
- ✓ $E1 - E2$
- ✓ $E1 \times E2$
- ✓ $\sigma_P(E1)$, where P is a predicate on attributes in $E1$
- ✓ $\Pi_S(E1)$, where S is a list consisting of some of the attributes in $E1$
- ✓ $\rho_x(E1)$, where x is the new name for the result of $E1$

Additional Relational-Algebra Operations

We define additional operations that do not add any power to the algebra, but simplify common queries.

1) **SET-INTERSECTION Operation:** The intersection of two relations r and s is denoted by r ∩ s. The output relation contains the set of all tuples belonging to both r and s.

Ex: List out all the names belonging to MPC III who won both the cultural & sports competition.

$$\Pi_{\text{Name}}(\sigma_{\text{Class = 'MPC III'}})(\text{CULTURAL}) \cap \Pi_{\text{Name}}(\sigma_{\text{Class = MPC III}})(\text{SPORTS})$$

NAME
Kavya
Zeba
Hima

NAME
Hima
Sheela

NAME
Hima

2) **NATURAL JOIN Operation:** Natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation. It is denoted by the "join" symbol. Natural join operation forms a Cartesian product of its two arguments, performs a selection forcing equalities on those attributes that appear in both relation schemas, and finally removes duplicate columns.

Ex: STUDENT ⋈ QUARTERLY

This operation performs a Cartesian product (X) of two relations, performs selection equality on those attributes that appear in both the relation schemas and finally removes duplicate columns. In STUDENT X QUARTERLY relations ROLLNO is common in both relations.

i.e. STUDENT ⋈ QUARTERLY becomes:

Roll-No	Name	Date-of-Birth	Second-Language	Roll-No	Maths	Physics	Computers
1	Sunny	01-07-70	Hindi	1	72	85	90
2	Rashni	15-08-72	Sanskrit	2	65	74	68
3	Anthra	29-01-71	Hindi	3	97	94	96
4	Nasreen	31-12-70	Telugu	4	87	93	72

Query can be written as

$$\Pi_{\text{Roll No, Name, Date of Birth, 2nd language, Maths, Physics, Computers}} (\text{STUDENT} \bowtie \text{QUARTERLY})$$

Now, to find student names and Computer marks, the query will be:

$$\Pi_{\text{Name, Computers}} (\text{STUDENT} \bowtie \text{QUARTERLY})$$

Name	Computers
Sunny	90
Rashni	68
Anthra	96
Nasreen	72

3) **The ASSIGNMENT Operation:** It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables. The **assignment** operation, denoted by ←, works like assignment in a programming language. To illustrate this operation, consider the definition of the natural-join operation. We could write $r \cup s$ as:

$$result \leftarrow r \cup s$$

The evaluation of an assignment does not result in any relation being displayed to the user. Rather, the result of the expression to the right of the ← is assigned to the relation variable on the left of the ←. This relation variable may be used in subsequent expressions.

For relational algebra queries, assignment must always be made to a temporary relation variable. Note that the assignment operation does not provide any additional power to the algebra. It is, however, a convenient way to express complex queries.

Extended Relational-Algebra Operations

The relational algebra operations that provide the ability to write queries that cannot be expressed using the basic relational-algebra operations are called **extended relational-algebra** operations.

1) **GENERALIZED PROJECTION:** The first operation is the **generalized-projection** operation, which extends the projection operation by allowing operations such as arithmetic and string functions to be used in the projection list. The generalized-projection operation has the form:

$$\Pi_{F1,F2,\ldots,Fn} (E)$$

where E is any relational-algebra expression, and each of $F1, F2, \ldots, Fn$ is an arithmetic expression involving constants and attributes in the schema of E. As a base case, the expression may be simply an attribute or a constant. In general, an expression can use arithmetic operations such as $+, -, *$, and $\div$ on numeric valued attributes, numeric constants, and on expressions that generate a numeric result. Generalized projection also permits operations on other data types, such as concatenation of strings.
For example, the expression:

$$\Pi_{ID,\ name,\ deptname,\ salary \div 12}(instructor)$$

gives the *ID, name, deptname,* and the monthly salary of each instructor.

2) **AGGREGATION:** The second extended relational-algebra operation is the aggregate operation G, which permits the use of aggregate functions such as min or average, on sets of values.

Aggregate Functions: Aggregate Functions take a collection of values and return a single value as a result. Aggregate operation in relational algebra is expressed as:

$$G_1, G_2, \ldots, Gn\ \ G_{F1(A1),\ F2(A2),\ldots,\ Fn(An)}\ (E)$$

Where, E is any relational algebra expression

Each F_i is an aggregate function

Each A_i is an attribute name

$G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)

The Aggregate Functions include:

1. **sum:** It is used to find the sum of values of an attribute in a relation.
   Ex: $G_{\mathbf{sum}(salary)}(instructor)$
2. **avg**: It is used to find the average value of an attribute in a relation.
   Ex: $G_{\mathbf{avg}(salary)}(instructor)$
3. **count**: It is used to find the number of values in an attribute in a relation.
   Ex: $G_{\mathbf{count}(salary)}(instructor)$

There are cases where we must eliminate multiple occurrences of a value before computing an aggregate function. If we do want to eliminate duplicates, we use the same function names as before, with the addition of the key word "**distinct**" appended to the end of the function name (for ex. **count-distinct**). Now, the above example can also be written as:

$G_{\mathbf{count\text{-}distnict}(salary)}(instructor)$

4. **min:** It is used to find the minimum value in an attribute in a relation.
   Ex: $G_{\mathbf{min}(salary)}(instructor)$
5. **max:** It is used to find the maximum value in an attribute in a relation.
   Ex: $G_{\mathbf{max}(salary)}(instructor)$

❖ Result of aggregation does not have a name.

- Can use rename operation to give it a name.
- For convenience, we permit renaming as part of aggregate operation

NULL VALUES: It is possible for tuples to have a null value, denoted by *null*, for some of their attributes. *null* signifies an unknown value or that a value does not exist.

- ✓ The result of any arithmetic expression involving *null* is *null.*
- ✓ Aggregate functions simply ignore null values
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values.
- ✓ For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same. As an alternative, assume each null is different from each other like in SQL.

Arithmetic Operations and Comparisons with null values return the special truth value *unknown or null.* For logical operators with an input as null:

OR: (*unknown* **or** *true*) = *true*,
(*unknown* **or** *false*) = *unknown*
(*unknown* **or** *unknown) = unknown*

AND: *(true* **and** *unknown) = unknown,*
(false **and** *unknown) = false,*
(unknown **and** *unknown) = unknown*

NOT*:* *(***not** *unknown) = unknown*

MODIFYING THE DATABASE: The content of the database may be modified using the following operations:

- ❖ Deletion
- ❖ Insertion
- ❖ Updating

- All these operations are expressed using the assignment operator.

❖ **Deletion**: A delete request is expressed in much the same way as a query. However, instead of displaying tuples to the user, we remove the selected tuples from the database. We may delete only whole tuples; we cannot delete values on only particular attributes. In relational algebra, a deletion is expressed by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra query.

Ex: (i) Delete the information of the students whose second language is Hindi

STUDENT ← STUDENT – ($\boldsymbol{\sigma}_{\text{second –language = Hindi}}$) (STUDENT)

(ii) Delete the marks of the students who got less than 75 marks in COMPUTERS.

QUARTERLY ← QUARTERLY – ($\boldsymbol{\sigma}_{\text{COMPUTERS} < 75}$) (QUARTERLY)

- **Insertion**: To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuples must be members of the attribute's domain. In relational algebra, an insertion is expressed by :

$$r \leftarrow r \cup E$$

where r is a relation and E is a relational algebra expression.

Ex: To insert the information of Mathew with Roll. No. 5, Date of Birth 15-9-1971 and Second Language is Telugu, we write:

$$STUDENT \leftarrow STUDENT \cup \{(5, \text{“15-9-71”}, \text{“Telugu”})\}$$

Updating: In certain situations we may wish to change a value in a tuple without changing all values in the tuple. It can be done by using the generalized projection operator.

$$r \leftarrow \Pi_{F1, F2, \ldots, Fn}(r)$$

Each *Fi* is either the *i*th attribute of *r*, if the *i*th attribute is not updated, or, which gives the new value for the attribute.

Ex: To increase the salary of each instructor by 10%, we write:

$$instructor \leftarrow \Pi_{sal * 1.10}(instructor)$$

SQL VERSUS RELATIONAL ALGEBRA

The term *select* in relational algebra has a different meaning than the one used in SQL. In relational algebra, the term *select* corresponds to what we refer to in SQL as *where*. We emphasize the different interpretations here to minimize potential confusion.