DATABASE MANAGEMENT SYSTEMS

UNIT - II

Relational Model, Relational Algebra and Calculus

Dr. Shaik Hussain Shaik Ibrahim

School of Engineering

Department of Computer Science & Engineering

Malla Reddy University

Hyderabad

SYLLABUS:

The Relational Model: Introduction to the Relational Model, Integrity Constraints over Relations, Enforcing Integrity Constraints, Querying Relational Data, Logical Database Design, Introduction to Views, Destroying/Altering Tables and Views.

Relational Algebra And Calculus: Relational Algebra- Selection and Projection, Set Operations, Renaming, Joins, Division, Examples of Relational Algebra Queries, Relational Calculus- Tuple Relational Calculus, Domain Relational Calculus.

Relational Model:

Introduction

The relational data model was first introduced by Ted Codd of IBM Research in 1970 in a classic paper (Codd 1970), and it attracted immediate attention due to its simplicity and mathematical foundation. The model uses the concept of a mathematical relation which looks somewhat like a table of values as its basic building block, and has its theoretical basis in set theory and first-order predicate logic.

The first commercial implementations of the relational model became available in the early 1980s, such as the SQL/DS system on the MVS operating system by IBM and the Oracle DBMS. Since then, the model has been implemented in a large number of commercial systems. Current popular relational DBMSs (RDBMSs) include DB2 and Informix Dynamic Server (from IBM), Oracle and Rdb (from Oracle), Sybase DBMS (from Sybase) and SQLServer and Access (from Microsoft). In addition, several open source systems, such as MySQL and PostgreSQL, are available.

The Relational Model Concepts:

The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or, to some extent, a flat file of records. It is called a flat file because each record has a simple linear or flat structure.

When a relation is thought of as a table of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

Example: In STUDENT relation because each row represents facts about a particular student entity. The column names Name, Student_number, Class, and Major specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type.

In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is represented by a domain of possible values.

1.1.1 Domains, Attributes, Tuples, and Relations

Domain:

A **domain** D is a set of atomic values. By **atomic** we mean that each value in the domain is invisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify the name for the domain, to help in interpreting its values.

Some examples of domains follow:

- Usa_phone_numbers: The set of ten-difgit phone numbers valid in United States.
- Social_security_numbers: The set of valid nine-digit social security numbers.
- Names: The set of character strings that represents the names of persons.
- Employee_ages: Possible ages of employees in a company; each must be an integer value between 15 and 80.

The preceding are called logical definitions of domains. A data type or format is also specified for each domain. For example, the data type for the domain Usa_phone_numbers can be declared as a character string of the form (ddd)ddddddd, where each d is a numeric (decimal) digit and the first three digits form a valid telephone area code. The data type for Employee_ages is an integer number between 15 and 80.

Attribute:

An **attribute** Ai is the name of a role played by some domain D in the relation schema R. D is called the domain of Ai and is denoted by dom(Ai).

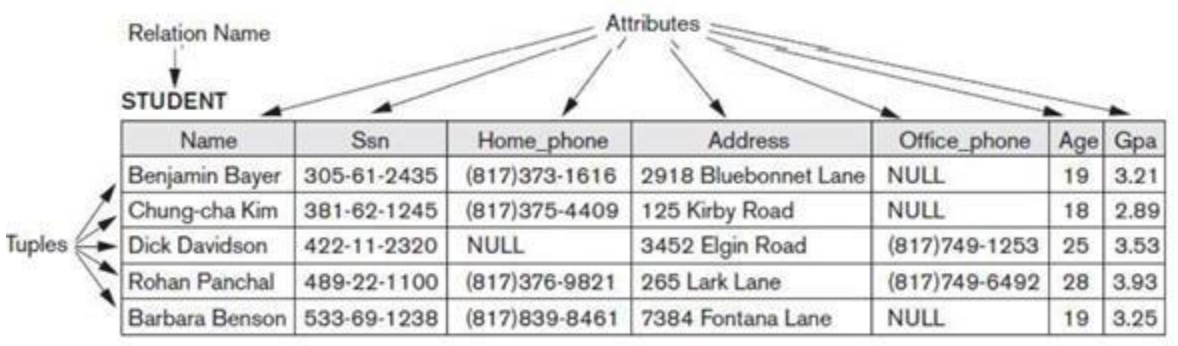
Tuple:

Mapping from attributes to values drawn from the respective domains of those attributes. Tuples are intended to describe some entity (or relationship between entities) in the miniworld
Example: a tuple for a PERSON entity might be

{ Name -->"smith", Gender--> Male, Age --> 25 }

Relation:

A named set of tuples all of the same form i.e., having the same set of attributes.



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Relation schema:

A relation schema R, denoted by R(A_1, A_2, ...,A_n), is made up of a relation name R and a list of attributes A_1, A_2, ...,A_n. Each attribute A_i is the name of a role played by some domain D in the relation schema R. D is called the domain of A_i and is denoted by dom(A_i). A relation schema is used to describe a relation; R is called the name of this relation.

The degree (or arity) of a relation is the number of attributes n of its relation schema. A relation of degree seven, which stores information about university students,would contain seven attributes describing each student as follows:

STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)

Using the data type of each attribute, the definition is sometimes written as:

STUDENT(Name: string, Ssn: string, Home_phone: string, Address: string, Office_phone: string, Age: integer, Gpa: real)

Domains for some of the attributes of the STUDENT relation:

dom(Name) = Names;

dom(Ssn) = Social_security_numbers;

dom(HomePhone) =USA_phone_numbers,

dom(Office_phone) = USA_phone_numbers,

Relation (or relation state):

A relation (or relation state) r of the relation schema by $R(A_1, A_2, ..., A_n)$, also denoted by r(R), is a set of n-tuples r = {t1, t2, ..., tm}. Each n-tuple t is an ordered list of n values t =<v1, v2, ..., vn where each value $v_i \leq i \leq n$ is an element of $dom(A_i)$ or is a special NULL value. The i^{th} value in tuple t, which corresponds to the attribute Ai, is referred to as t[Ai] or t. Ai.

The terms relation intension for the schema R and relation extension for a relation state r(R) are also commonly used.

Informal Terms	Formal Terms
Table	Relation
Column Header	Attribute
All possible Column Values	Domain
Row	Tuple
Table Definition	Schema of a Relation
Populated Table	State of the Relation

1.1.2 Characteristics of Relations

1) Ordering of Tuples in a Relation

A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them; hence, tuples in a relation do not have any particular order. Tuple ordering is not part of a relation definition because a relation attempts to represent facts at a logical or abstract level. Many tuple orders can be specified on the same relation.

2) Ordering of Values within a Tuple and an Alternative Definition of a Relation

The order of attributes and their values is not that important as long as the correspondence between attributes and values is maintained. An alternative definition of a relation can be given, making the ordering of values in a tuple unnecessary. In this definition A relation schema $R(A_1, A_2, ...,A_n)$, set of attributes and a relation state r(R) is a finite set of mappings r = {t1, t2, ..., tm}, where each tuple ti is a mapping from R to D.

According to this definition of tuple as a mapping, a tuple can be considered as a set of (<attribute>, <value>) pairs, where each pair gives the value of the mapping from an attribute A_i to a value vi from dom(A_i) .The ordering of attributes is not important, because the attribute name appears with its value.

Relation Name

Attributes

Tuples

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

Figure 5.2
The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

3) Values and NULLs in the Tuples

Each value in a tuple is atomic. NULL values are used to represent the values of attributes that may be unknown or may not apply to a tuple. For example some STUDENT tuples have NULL for their office phones because they do not have an office .Another student has a NULL for home phone In general, we can have several meanings for NULL values, such as value unknown, value exists but is not available, or attribute does not apply to this tuple (also known as value undefined).

4) Interpretation (Meaning) of a Relation

The relation schema can be interpreted as a declaration or a type of assertion. For example, the schema of the STUDENT relation of asserts that, in general, a student entity has a Name, Ssn, Home_phone, Address, Office_phone, Age, and Gpa. Each tuple in the relation can then be interpreted as a particular instance of the assertion.For example, the first tuple asserts the fact that there is a STUDENT whose Name is Benjamin Bayer, Ssn is 305-61-2435, Age is 19, and so on.

An alternative interpretation of a relation schema is as a predicate; in this case, the values in each tuple are interpreted as values that satisfy the predicate.

1.1.3 Relational Model Notation

- Relation schema R of degree n is denoted by by $R(A_1, A_2, ..., A_n)$
- Uppercase letters Q, R, S denote relation names
- Lowercase letters q, r, s denote relation states
- Letters t, u, v denote tuples
- In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation
- An attribute A can be qualified with the relation name R to which it belongs by using the dot notation R.A for example, STUDENT.Name or STUDENT.Age
- An n-tuple t in a relation r(R) is denoted by $t = <v_1, v_2, ..., v_n>$, where v_i is the value corresponding to attribute A_i. The following notation refers to component values of tuples: Both $t[A_i]$ and $t.A_i$ (and sometimes t[i]) refer to the value v_i in t for attribute A_i.
- Both $t[A_u, A_w, ..., A_z]$ and $t.(A_u, A_w, ..., A_z)$, where $A_u, A_w, ..., A_z$ is a list of attributes from R, refer to the subtuple of values $<v_u, v_w, ..., v_z>$ from t corresponding to the attributes specified in the list.

1.2 Relational Model Constraints and Relational Database Schemas:

Constraints are restrictions on the actual values in a database state. These constraints are derived from the rules in the miniworld that the database represents. Constraints on databases can generally be divided into three main categories:

1) **Inherent model-based constraints or implicit constraints**
   - Constraints that are inherent in the data model.
   - The characteristics of relations are the inherent constraints of the relational model and belong to the first category. For example, the constraint that a relation cannot have duplicate tuples is an inherent constraint.
2) **Schema-based constraints or explicit constraints**
   - Constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL.
   - The schema-based constraints include domain constraints, key constraints, constraints on NULLs, entity integrity constraints, and referential integrity constraints.
3) **Application-based or semantic constraints or business rules**
   - Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs.
   - Examples of such constraints are the salary of an employee should not exceed the salary of the employee, supervisor and the maximum number of hours an employee can work on all projects per week is 56.

1.2.1 Domain Constraints

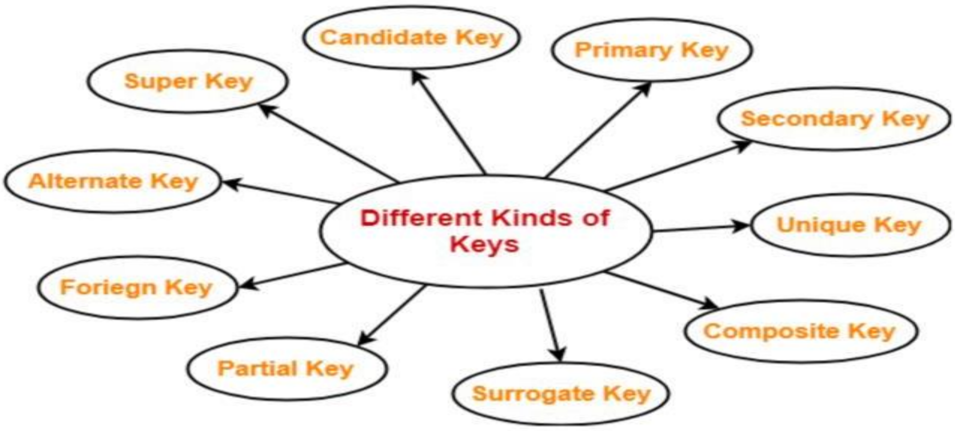
Domain Constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain dom(A). The data types associated with domains typically include standard numeric data types for integers (such as short integer, integer, and long integer) and real numbers (float and doubleprecision float). Characters, Booleans, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and money, or other special data types.

1.2.2 Key Constraints and Constraints on NULL Values

A **key** is a set of one or more attributes that can uniquely identify each row in a table. A key not only identifies the rows of a table but also relates two or more tables.

Different Types of Keys:

1) Super Key
2) Candidate Key
3) Primary Key
4) Foreign Key
5) Secondary Key/Alternate Key
6) Unique Key
7) Composite Key
8) Surrogate Key
9) Partial Key



1) **Super Key:** Super Key is an attribute (or a set of attributes) that uniquely identify a tuple i.e. an entity in entity set.
   It is a superset of Candidate Key, since Candidate Keys are selected from super key.
   Example:
   <Student>

Student_ID	Student_Enroll	Student_Name	Student_Email
S02	4545	Dave	ddd@gmail.com
S34	4541	Jack	jjj@gmail.com
S22	4555	Mark	mmm@gmail.com

Super Keys are:
{Student_ID}
{Student_Enroll}
{Student_Email}
{Student_ID, Student_Enroll}
{Studet_ID, Student_Name}
{Student_ID, Student_Email}
{Student_Name, Student_Enroll}
{Student_ID, Student_Enroll, Student_Name}
{Student_ID, Student_Enroll, Student_Email}
{Student_ID, Student_Enroll, Student_Name, Student_Email}

Candidate Keys are:
{Student_ID}
{Student_Enroll}
{Student_Email}

2) **Candidate Key**: Each table has only a single primary key. Each relation may have one or more candidate key. One of these candidate key is called Primary Key. Each candidate key qualifies for Primary Key. Therefore candidates for Primary Key is called Candidate Key.

Candidate key can be a single column or combination of more than one column. A minimal super key is called a candidate key.

Example:

Student_ID	Student_Enroll	Student_Name	Student_Email
S02	4545	Dave	ddd@gmail.com
S34	4541	Jack	jjj@gmail.com
S22	4555	Mark	mmm@gmail.com

Above, Student_ID, Student_Enroll and Student_Email are the candidate keys. They are considered candidate keys since they can uniquely identify the student record.

3) **Primary Key:** It is an attribute or set of attributes that uniquely identify an entity (row) in the entity set (table). The main difference between the primary key and the candidate key in that is primary key does not contain NULL values.

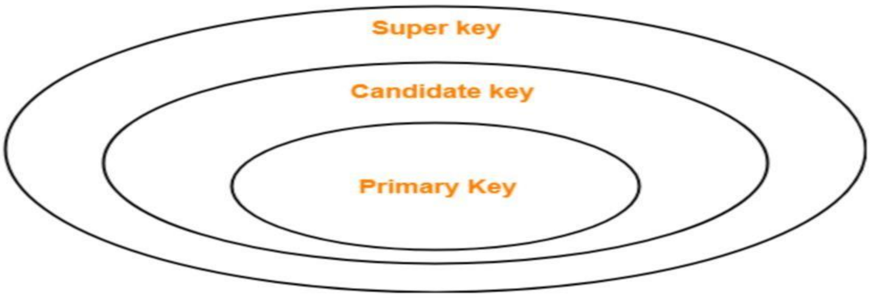
- ✓ Primary Key must be UNIQUE and NOT NULL.

Example:

EMPLOYEE

EID	EMP_FNAME	EMP_LNAME
12	Sai	Keerthi
13	Potluri	Siddhartha
14	Velagapudi	Krishna
15	Rallapalli	Suma

The primary key of the relation can be EID.



4) **Foreign Key:** A foreign key is a set of attributes in a table that refers to the primary key of another table. The foreign key links these two tables.

Example:

Primary Key

STUDENT_STATUS

SROLL_NO	SNAME	DOB
12	Keerthi	10-03-1999
13	Siddhartha	18-06-2001
14	Krishna	21-09-2000
15	Suma	30-01-1998

Foreign Key

STUDENT_DETAILS

SROLL_NO	PROJECT_ID	MARKS	STATUS
12	P1	100	Pass
13	P2	90	Pass
14	P3	80	Pass

5) **Secondary Key**/Alternalte Key: A primary key is the field in a database that is the primary key used to uniquely identify a record in a database. A secondary key is an additional key, or alternate key, which can be use in addition to the primary key to locate specific data.

Secondary Key is the key that has not been selected to be the primary key. However, it is considered a candidate key for the primary key.

Therefore, a candidate key not selected as a primary key is called secondary key. Candidate key is an attribute or set of attributes that you can consider as a Primary key.

Note: Secondary Key is not a Foreign Key.

Example 1:

Student_ID	Student_Enroll	Student_Name	Student_Age	Student_Email
096	9122717	Manish	25	aaa@gmail.com
055	9122655	Manan	23	abc@gmail.com
067	9122699	Shreyas	28	pqr@gmail.com

Above, Student_ID, Student_Enroll and Student_Email are the candidate keys. They are considered candidate keys since they can uniquely identify the student record. Select any one of the candidate key as the primary key. Rest of the two keys would be Secondary Key.

If you selected Student_ID as primary key, therefore Student_Enroll and Student_Email will be Secondary Key (candidates of primary key).

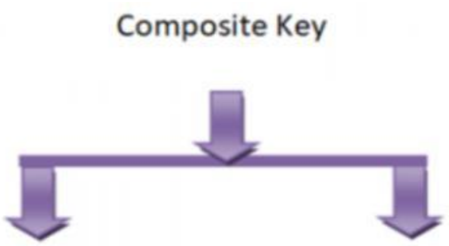
Example 2:

Employee_ID	Employee_No	Employee_Name	Employee_Email	Employee_Dept
0989	E7897	Jacob	jacob@example.com	Finance
0777	E8768	Anna	anna@example.com	HR
0656	E8789	Tom	tom@example.com	Operations

Above, Employee_ID, Employee_No and Employee_Email are the candidate keys. They uniquely identify the Employee record. Select any one of the candidate key as the primary key. Rest of the two keys would be Secondary Key.

6) **Unique Key:** A Unique Key is used to prevent duplicate values in a column. Primary Key provided uniqueness to a table.
A primary key cannot accept NULL values; this makes Primary Key different from Unique Key, since Unique Key allows one value as NULL value.
A table can only have a single Primary Key, whereas a Unique Key can be more than one if you need it in the table.
Unique Key ensures that data is not duplicated in two rows in the database. A row in the database can have null in case of Unique Key.
You cannot modify a Primary Key, but a Unique Key can be modified.

7) **Composite Key:** A primary key having two or more attributes is called composite key. It is a combination of two or more columns.
Example 1: Here our composite key is OrderID and ProductID −
{OrderID, ProductID}



OrderID	ProductID	Quantity
22324	99	4
11332	99	9
23467	145	7
22324	129	3

Example 2:
<Student>

StudentID	StudentEnrollNo	StudentMarks	StudentPercentage
S001	0721722	570	90
S002	0721790	490	80
S003	0721766	440	86

Above, our composite keys are StudentID and StudentEnrollNo. The table has two attributes as primary key.
Therefore, the Primary Key consisting of two or more attribute is called Composite Key.

8) **Surrogate Key:** A Surrogate Key's only purpose is to be a unique identifier in a database, for example, incremental key.
Surrogate Key has no actual meaning and is used to represent existence. It has an existence only for data analysis.
Example: The surrogate key is
Key in the <ProductPrice> table.

<ProductPrice>

Key	ProductID	Price
505_92	1987	200
698_56	1256	170
304_57	1898	250
458_66	1666	110

Other examples of a Surrogate Key:

- ✓ Counter
- ✓ System date/time stamp
- ✓ Random alphanumeric string.

9) Partial Key: Partial key is a key using which all the records of the table can not be identified uniquely.

However, a bunch of related tuples can be selected from the table using the partial key.

Example: Consider the following schema-

Department (Emp_no , Dependent_name , Relation)

Emp_no	Dependent_name	Relation
E1	Suman	Mother
E1	Ajay	Father
E2	Vijay	Father
E2	Ankush	Son

Here, using partial key Emp_no, we can not identify a tuple uniquely but we can select a bunch of tuples from the table

Following are the important differences between Primary Key and Candidate key.

Sr. No.	Key	Primary Key	Candidate key
1	Definition	Primary Key is a unique and non-null key which identify a record uniquely in table. A table can have only one primary key.	Candidate key is also a unique key to identify a record uniquely in a table but a table can have multiple candidate keys.
2	Null	Primary key column value can not be null.	Candidate key column can have null value.
3	Objective	Primary key is most important part of any relation or table.	Candidate key signifies as which key can be used as Primary Key.
4	Use	Primary Key is a candidate key.	Candidate key may or may not be a primary key.

Following are the important differences between Super Key and Candidate key.

Sr. No.	Key	Super Key	Candidate key
1	Definition	Super Key is used to identify all the records in a relation.	Candidate key is a subset of Super Key.
2	Use	All super keys can't be candidate keys.	All candidate keys are super keys.
3	Selection	Super keys are combined together to create a candidate key.	Candidate keys are combined together to create a primary key.
4	Count Wise	Super keys are more than Candidate keys.	Candidate keys are less than Super Keys.

Primary Key

- It is used to ensure that the data in the specific column is unique.
- It helps uniquely identify a record in a relational database.
- One primary key only is allowed in a table.
- It is a combination of the 'UNIQUE' and 'Not Null' constraints.
- This means it can't be a NULL value.
- It is the most important part of a table.
- It is a candidate key.
- Its value can't be deleted from parent table.
- The constraint can be implicitly defined for the temporary tables.

Candidate key

- It can have NULL value.
- It may or may not have a primary key.
- It tells about which key can be used as a primary key.
- It is a unique key that helps identify a record uniquely in a table.
- A table can have multiple candidate keys.

All tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes. There are other subsets of attributes of a relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes.

Suppose that we denote one such subset of attributes by SK; then for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that: $t_1[SK] \neq t_2[SK]$. such set of attributes SK is called a superkey of the relation schema R

Superkey

A superkey SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK. Every relation has at least one default superkey the set of all its attributes.

Key

A key K of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K′ that is not a superkey of R anymore. Hence, a key satisfies two properties:

1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This first property also applies to a superkey.
2. It is a minimal superkey that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint in condition will hold.This property is not required by a superkey.

Example: Consider the STUDENT relation

Relation Name

Attributes

STUDENT

Tuples

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

- The attribute set {Ssn} is a key of STUDENT because no two student tuples can have the same value for Ssn.
- Any set of attributes that includes Ssn for example, {Ssn, Name, Age} is a superkey.
- The superkey {Ssn, Name, Age} is not a key of STUDENT because removing Name or Age or both from the set still leaves us with a superkey.

In general, any superkey formed from a single attribute is also a key. A key with multiple attributes must require all its attributes together to have the uniqueness property.

Candidate Key

A relation schema may have more than one key. In this case, each of the keys is called a candidate key.

Example: The CAR relation has two candidate keys: License_number and Engine_serial_number

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Primary Key

It is common to designate one of the candidate keys as the primary key of the relation. This is the candidate key whose values are used to identify tuples in the relation. We use the convention that the attributes that form the primary key of a relation schema are underlined. Other candidate keys are designated as unique keys and are not underlined.

Another constraint on attributes specifies whether NULL values are or are not permitted. For example, if every STUDENT tuple must have a valid, non-NULL value for the Name attribute, then Name of STUDENT is constrained to be NOT NULL.

1.2.3 Relational Databases and Relational Database Schemas

Relational database schema S is a set of relation schemas S = {R_1, R_2, ..., R_m} and a s et of integrity constraints IC.

Example of relational database schema:

COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}

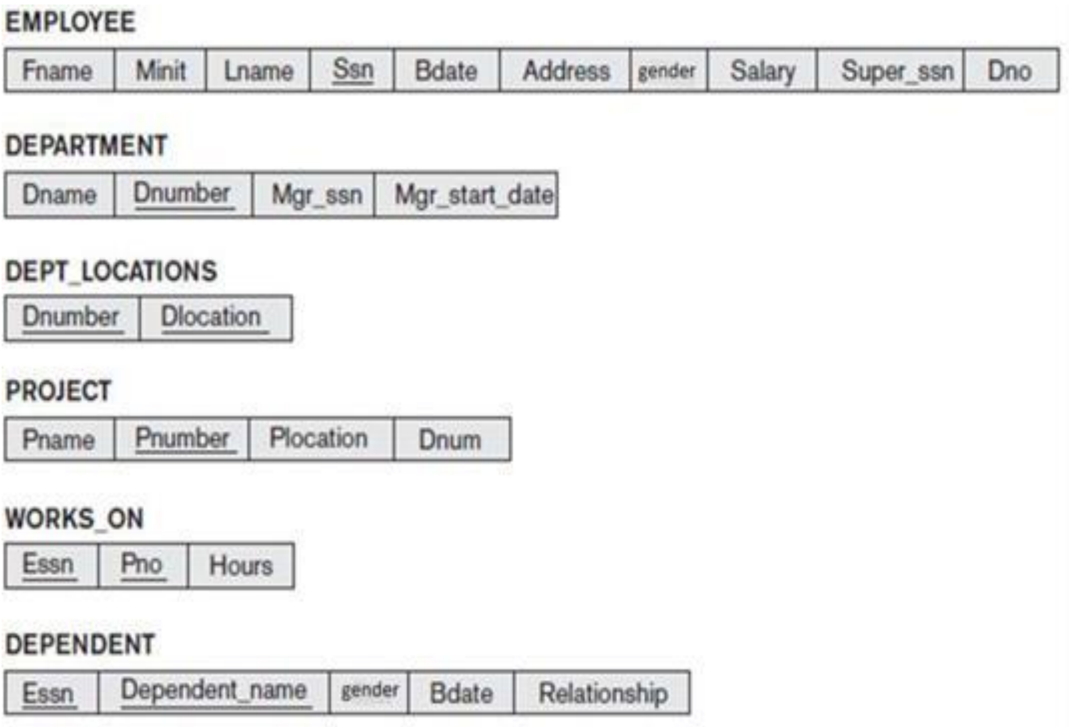


Figure: Schema diagram for the COMPANY relational database schema. The underlined attributes represent primary keys

A Relational database state is a set of relation states DB = {r_1, r_2, ..., r_m}.Each r_i is a state of R and such that the r_i relation states satisfy integrity constraints specified in IC.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Figure: One possible database state for the COMPANY relational database schema.

A database state that does not obey all the integrity constraints is called Invalid state and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a Valid state.

Attributes that represent the same real-world concept may or may not have identical names in different relations.

Example: The Dnumber attribute in both DEPARTMENT and DEPT_LOCATIONS stands for the same real-world concept the number given to a department.

That same concept is called Dno in EMPLOYEE and Dnum in PROJECT.

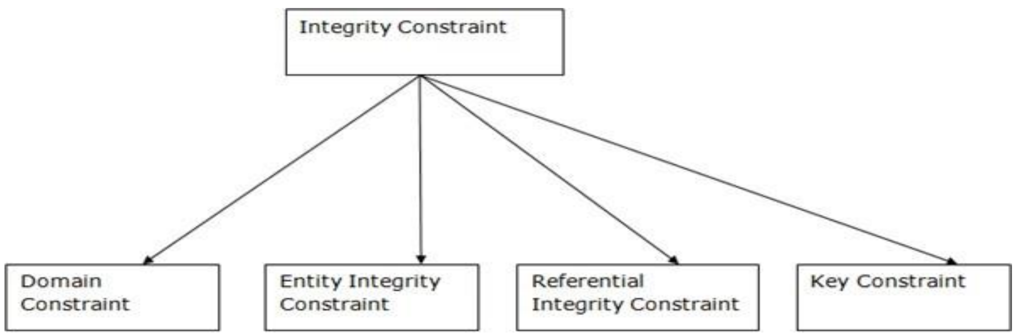
Alternatively, attributes that represent different concepts may have the same name in different relations. For example, we could have used the attribute name Name for both Pname of PROJECT and Dname of DEPARTMENT; in this case, we would have two attributes that share the same name but represent different real world concepts project names and department names.

1.2.4 Integrity, Referential Integrity, and Foreign Keys

Integrity Constraints

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

Types of Integrity Constraint



1) **Domain Constraints:**
   - ✓ Domain constraints can be defined as the definition of a valid set of values for an attribute.
   - ✓ The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

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

Not allowed. Because AGE is an integer attribute

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

Example:

EMPLOYEE

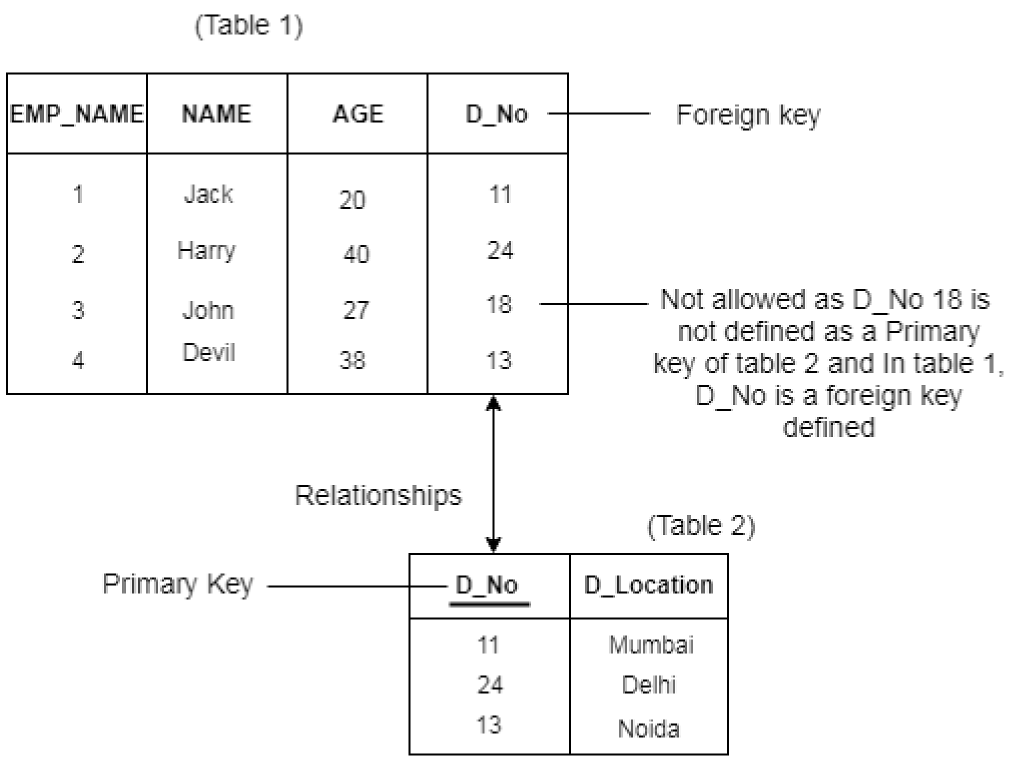
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

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

3) Referential Integrity Constraints

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

Example:



Referential integrity constraint

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Example: COMPANY database, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

To define referential integrity more formally, first we define the concept of a foreign key. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2.

A set of attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following rules:

1. Attributes in FK have the same domain(s) as the primary key attributes PK of R_2; the attributes FK are said to reference or refer to the relation R_2.
2. A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is NULL.

In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple t_1 references or refers to the tuple t_2.

In this definition, R_1 is called the referencing relation and R_2 is the referenced relation. If these two conditions hold, a referential integrity constraint from R_1 to R_2 is said to hold.

4) Key constraints

- ✓ Keys are the entity set that is used to identify an entity within its entity set uniquely.
- ✓ An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

Example:

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

Not allowed. Because all row must be unique

1.2.5 Other Types of Constraints

1) Semantic Integrity Constraints:

Semantic integrity constraints can be specified and enforced within the application programs that update the database, or by using a general-purpose constraint specification language. Examples of such constraints are the salary of an employee should not exceed the salary of the employee's supervisor and the maximum number of hours an employee can work on all projects per week is 56. Mechanisms called triggers and assertions can be used. In SQL, CREATE ASSERTION and CREATE TRIGGER statements can be used for this purpose.

2) Functional Dependency Constraints:

Functional dependency constraint establishes a functional relationship among two sets of attributes X and Y. This constraint specifies that the value of X determines a unique value of Y in all states of a relation; it is denoted as a functional dependency $X \rightarrow Y$. We use functional dependencies and other types of dependencies as tools to analyze the quality of relational design and to normalize relations to improve their quality.

State constraints (static constraints)

Define the constraints that a valid state of the database must satisfy

Transition constraints (dynamic constraints)

Define to deal with state changes in the database

Views:

Creating Views:

We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

Syntax explaination:

view_name: Name for the View
table_name: Name of the table
condition: Condition to select rows

Let us see the data from the Sample Tables:

StudentDetails:

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

StudentMarks:

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Examples for Creating Views:

I. Creating View from a single table:

In this example we will create a View named DetailsView from the table StudentDetails.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM StudentDetails
WHERE S_ID < 5;
```

To see the data in the View, we can query the view in the same manner as we query a table.

SELECT * FROM DetailsView;

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

Examples:

In this example, we will create a view named StudentNames from the table StudentDetails.

Query:

```
CREATE VIEW StudentNames AS
SELECT S_ID, NAME
FROM StudentDetails
ORDER BY NAME;
```

If we now query the view as,

```
SELECT * FROM StudentNames;
```

Output:

S_ID	NAMES
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

II. Creating View from multiple tables:

In this example we will create a View named MarksView from two tables StudentDetails and StudentMarks.

To create a View from multiple tables we can simply include multiple tables in the SELECT statement.

Query:

```
CREATE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS, StudentMarks.MARKS
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

To display data of View MarksView:
SELECT * FROM MarksView;

Output:

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

DELETING VIEWS:

SQL allows us to delete an existing View. We can delete or drop a View using the DROP statement.

Syntax:

DROP VIEW view_name;

view_name: Name of the View which we want to delete.
Example: if we want to delete the View MarksView, we can do this as:
DROP VIEW MarksView;

UPDATING VIEWS:

There are certain conditions needed to be satisfied to update a view. If any one of these conditions is not met, then we will not be allowed to update the view.

- The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.
- The SELECT statement should not have the DISTINCT keyword.
- The View should have all NOT NULL values.
- The view should not be created using nested queries or complex queries.
- The view should be created from a single table.
- If the view is created using multiple tables then we will not be allowed to update the view.

We can use the CREATE OR REPLACE VIEW statement to add or remove fields from a view.
Syntax:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1,coulmn2,..
FROM table_name
WHERE condition;
```

For example, if we want to update the view MarksView and add the field AGE to this View from StudentMarks Table, we can do this as:

```
CREATE OR REPLACE VIEW MarksView AS
SELECT StudentDetails.NAME, StudentDetails.ADDRESS,
StudentMarks.MARKS, StudentMarks.AGE
FROM StudentDetails, StudentMarks
WHERE StudentDetails.NAME = StudentMarks.NAME;
```

If we fetch all the data from MarksView now as:

SELECT * FROM MarksView;

Output:

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

Inserting a row in a view:

We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.

Syntax:

INSERT INTO view_name(column1, column2 , column3,..)
VALUES(value1, value2, value3..);

view_name: Name of the View

Example:

In the below example we will insert a new row in the View DetailsView which we have created above in the example of "creating views from a single table".

INSERT INTO DetailsView(NAME, ADDRESS)
VALUES("Suresh","Gurgaon");

If we fetch all the data from DetailsView now as,

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

Deleting a row from a View:

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view.

Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

Syntax:

DELETE FROM view_name
WHERE condition;

view_name: Name of view from where we want to delete rows
condition: Condition to select rows

Example:

In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.

DELETE FROM DetailsView
WHERE NAME="Suresh";

If we fetch all the data from DetailsView now as,

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

WITH CHECK OPTION:

- ✓ The WITH CHECK OPTION clause in SQL is a very useful clause for views.
- ✓ It is applicable to a updatable view. If the view is not updatable, then there is no meaning of including this clause in the CREATE VIEW statement.
- ✓ The WITH CHECK OPTION clause is used to prevent the insertion of rows in the view where the condition in the WHERE clause in CREATE VIEW statement is not satisfied.
- ✓ If we have used the WITH CHECK OPTION clause in the CREATE VIEW statement, and if the UPDATE or INSERT clause does not satisfy the conditions then they will return an error.

Example:
In the below example we are creating a View SampleView from StudentDetails Table with

WITH CHECK OPTION clause.
CREATE VIEW SampleView AS
SELECT S_ID, NAME
FROM StudentDetails
WHERE NAME IS NOT NULL
WITH CHECK OPTION;

In this View if we now try to insert a new row with null value in the NAME column then it will give an error because the view is created with the condition for NAME column as NOT NULL.

For example,though the View is updatable but then also the below query for this View is not valid:

INSERT INTO SampleView(S_ID)
VALUES(6);

NOTE: The default value of NAME column is null.

Formal Relational Languages:

3.1 Introduction

Relational algebra is the basic set of operations for the relational model. These operations enable a user to specify basic retrieval requests as relational algebra expressions. The result of an operation is a new relation, which may have been formed from one or more input relations. The relational algebra is very important for several reasons

- First, it provides a formal foundation for relational model operations.
- Second, and perhaps more important, it is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs)
- Third, some of its concepts are incorporated into the SQL standard query language for RDBMSs

Relation Algebra is a procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
Six basic operators are:

1. select: σ
2. project: $\prod$
3. union: $\cup$
4. set difference: –
5. Cartesian product: x
6. rename: ρ

2.2 Unary Relational Operations: SELECT and PROJECT

2.2.1 The SELECT Operation

The SELECT operation denoted by σ (sigma) is used to select a subset of the tuples from a relation based on a selection condition. The selection condition acts as a filter that keeps only those tuples that satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to restrict the tuples in a relation to only those tuples that satisfy the condition.

The SELECT operation can also be visualized as a horizontal partition of the relation into two sets of tuples those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.

In general, the select operation is denoted by

$$\sigma_{\text{<select condition>}} (R)$$

where,

- the symbol is used to denote the select operator

- the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
- tuples that make the condition true are selected
  - appear from the result of the operation

- tuples that make the condition false are filtered out

  - discarded from the result of the operation

The Boolean expression specified in <selection condition> is made up of a number of clauses of the form:

<attribute name> <comparison op> <constant value> or

<attribute name> <comparison op> <attribute name>

where

<attribute name> is the name of an attribute of R,

<comparision op> is one of the operators $\{=, <, >, \leq, \geq, \neq\}$

<constant value> is a constant value from the attribute domain

Clauses can be connected by the standard Boolean operators and, or, and not to form a general selection condition

- **The select operation selects tuples that satisfy a given predicate.**

- **Notation:** σ_p (r)
  p is called the selection predicate

- **Example:** select those tuples of the instructor relation where the instructor is in the "Physics" department.

- **Query**:
  σ $_{\text{dept\_name="Physics"}}$ (instructor)

- **Result:**

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Examples:

1) Select the EMPLOYEE tuples whose department number is 4.

$$\sigma_{DNO = 4}(\text{EMPLOYEE})$$

2) Select the employee tuples whose salary is greater than $30,000.

$$\sigma_{SALARY > 30,000}(\text{EMPLOYEE})$$

3) Select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000

$$\sigma_{(Dno=4 \text{ AND } Salary>25000) \text{ OR } (Dno=5 \text{ AND } Salary>30000)}(\text{EMPLOYEE})$$

The result of a SELECT operation can be determined as follows:

- The <selection condition> is applied independently to each individual tuple t in R
- If the condition evaluates to TRUE, then tuple t is selected. All the selected tuples appear in the result of the SELECT operation
- The Boolean conditions AND, OR, and NOT have their normal interpretation, as follows:

- (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise, it is

  FALSE.

- (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is

  FALSE.

- (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE.

The SELECT operator is unary; that is, it is applied to a single relation. The degree of the relation resulting from a SELECT operation is the same as the degree of R. The number of tuples in the resulting relation is always less than or equal to the number of tuples in R. That is,

$$|\sigma_C(R)| \leq \text{ for any condition } C$$

The fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.

The SELECT operation is commutative; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

Hence, a sequence of SELECTs can be applied in any order.we can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<condn>}(R))\ldots)) = \sigma_{<cond1> \text{ AND } <cond2> \text{ AND } \ldots \text{ AND } <condn>}(R)$$

In SQL, the SELECT condition is specified in the WHERE clause of a query.For example, the following operation:

$$\sigma_{\text{Dno=4 AND Salary>25000}}\ (\text{EMPLOYEE})$$

would do the following SQL query:

SELECT * FROM EMPLOYEE WHERE Dno=4 AND Salary>25000;

2.2.2 The PROJECT Operation

The PROJECT operation denoted by selects certain columns from the table and discards the other columns Used when we are interested in only certain attributes of a relation. The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations:

- one has the needed columns (attributes) and contains the result of the operation
- the other contains the discarded columns

The general form of the PROJECT operation is

Π <attribute list>(R)

where

Π (pi) - symbol used to represent the PROJECT operation,

<attributelist> - desired sublist of attributes from the attributes of relation R.

The result of the PROJECT operation has only the attributes specified in <attribute list> in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>

- **Project Operation is a unary operation that returns its argument relation, with certain attributes left out.**
- **Notation:** $\pi_{A1,A2,A3 \ldots Ak}$ (r)

  where A1, A2, …, Ak are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed.
- Duplicate rows removed from result, since relations are sets.

Example: eliminate the dept_name attribute of instructor
Query: π ID, name, salary (instructor)

Result:

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Example:

1) To list each employee's first and last name and salary we can use the PROJECT operation as follows:

$$\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$$

If the attribute list includes only nonkey attributes of R, duplicate tuples are likely to occur. The result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as duplicate elimination.For example, consider the following PROJECT operation:

$$\pi_{\text{gender, Salary}}(\text{EMPLOYEE})$$

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

The tuple <'F', 25000> appears only once in the resulting relation even though this combination of values appears twice in the EMPLOYEE relation.

The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in R. Commutativity does not hold on PROJECT

$$\pi_{<\text{list1}>}(\pi_{<\text{list2}>}(R)) = \pi_{<\text{list1}>}(R)$$

as long as <list2> contains the attributes in <list1>; otherwise, the left-hand side is an incorrect expression.

In SQL, the PROJECT attribute list is specified in the SELECT clause of a query. For example, the following operation:

$$\pi_{\text{gender, Salary}}(\text{EMPLOYEE})$$

would correspond to the following SQL query:

SELECT DISTINCT gender, Salary FROM EMPLOYEE;

<u>Composition of Relational Operations</u>

- The result of a relational-algebra operation is relation and therefore of relational-algebra operations can be composed together into a relational-algebra expression.
- Consider the query -- Find the names of all instructors in the Physics department.

$$\pi_{\text{name}}(\sigma_{\text{dept\_name} = \text{"Physics"}}(\text{instructor}))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

2.2.3 RENAME Operation

- **The results of relational-algebra expressions do not have a name that we can use to refer to them. The rename operator, ρ ,is provided for that purpose**
- The expression:

  $\rho_x (E)$

  returns the result of expression E under the name x
- Another form of the rename operation:

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

For most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.

For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation. We can write a single relational algebra expression, also known as an in-line expression, as follows:

$$\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno=5}}(\text{EMPLOYEE}))$$

Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, as follows:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{Dno=5}}(\text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{DEP5\_EMPS})$$

We can also use this technique to rename the attributes in the intermediate and result relations. To rename the attributes in a relation, we simply list the new attribute names in parentheses.

$$\text{TEMP} \leftarrow \sigma_{\text{Dno=5}}(\text{EMPLOYEE})$$

$$\text{R(First\_name, Last\_name, Salary)} \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{TEMP})$$

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

If no renaming is applied, the names of the attributes in the resulting relation of a SELECT operation are the same as those in the original relation and in the same order.For a PROJECT operation with no renaming, the resulting relation has the same attribute names as those in the projection list and in the same order in which they appear in the list.

We can also define a formal RENAME operation which can rename either the relation name or the attribute names, or both as a unary operator.

The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms:

1. $\rho_{S(B1, B2, \ldots, Bn)}(R)$	ρ (rho) – RENAME operator
2. $\rho_S(R)$	S – new relation name
3. $\rho_{(B1, B2, \ldots\ldots Bn)}(R)$	$B_1, B_2, \ldots B_n$- new attribute names

The first expression renames both the relation and its attributes. Second renames the relation only and the third renames the attributes only.If the attributes of R are (A1, A2, ..., An) in that order, then each Ai is renamed as Bi.

Renaming in SQL is accomplished by aliasing using AS, as in the following example:

SELECT E.Fname **AS** First_name,

E.Lname **AS** Last_name,

E.Salary **AS** Salary

FROM EMPLOYEE **AS** E

WHERE E.Dno=5,

3.1 Relational Algebra Operations from Set Theory:

3.1.1 The UNION, INTERSECTION and MINUS Operations

- **UNION:** The result of this operation, denoted by R S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- **INTERSECTION:** The result of this operation, denoted by R S, is a relation that includes all tuples that are in both R and S.
- **SET DIFFERENCE (or MINUS):** The result of this operation, denoted by R S, is a relation that includes all tuples that are in R but not in S.

Example: Consider the the following two relations: STUDENT & INSTRUCTOR

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT ∪ INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

STUDENT ∩ INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah

STUDENT-INSTRUCTOR

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR-STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Example: To retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5.

DEPT5_EMP← $\sigma_{Dno=5}$(EMPLOYEE)

RESULT1← π_{Ssn}(DEPT5_EMPS)

RESULT2(Ssn) ← π_{Super_Ssn}(DEPT5_EMPS)

RESULT← RESULT1 ∪ RESULT2

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Single relational algebra expression:

$$\text{Result} \leftarrow \pi_{Ssn}(\sigma_{Dno=5}(\text{EMPLOYEE})) \cup \pi_{SSuper\_ssn}(\sigma_{Dno=5}(\text{EMPLOYEE}))$$

UNION, INTERSECTION and SET DIFFERENCE are binary operations; that is, each is applied to two sets (of tuples). When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called union compatibility or type compatibility.

Two relations R(A1, A2, ..., An) and S(B1, B2, ..., Bn) are said to be union compatible (or type compatible) if they have the same degree n and if dom(Ai) = dom(Bi) for $1 \leq i \leq n$. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

Both UNION and INTERSECTION are commutative operations; that is,

$$R \cup S = S \cup R \text{ and } R \cap S = S \cap R$$

Both UNION and INTERSECTION can be treated as n-ary operations applicable to any number of relations because both are also associative operations; that is,

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

The MINUS operation is not commutative; that is,

in general,

R-S ≠ S-R

INTERSECTION can be expressed in terms of union and set difference as follows,

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

In SQL, there are three operations UNION, INTERSECT, and EXCEPT that correspond to the set operations

Union Operation:

- The union operation allows us to combine two relations
- **Notation:** $r \cup s$

  For $r \cup s$ to be valid.

  1. r, s must have the same arity (same number of attributes)
  2. The attribute domains must be compatible (example: 2nd column of r deals with the same type of values as does the 2nd column of s)

Result of:

$$\Pi_{course\_id} (\sigma_{semester= \text{"Fall"} \wedge year=2017} (section)) \cup$$
$$\Pi_{course\_id} (\sigma_{semester= \text{"Spring"} \wedge year=2018} (section))$$

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set-Intersection Operation:

- The set-intersection operation allows us to find tuples that are in both the input relations.
- **Notation:** $r \cap s$
- **Assume:**

  r, s have the same arity

  attributes of r and s are compatible
- **Example:** Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.

$$\Pi_{course\_id}(\sigma_{semester=\text{"Fall"}} \wedge_{year=2017}(section)) \cap$$
$$\Pi_{course\_id} (\sigma_{semester=\text{"Spring"}} \wedge_{year=2018} (section))$$

- **Result**

course_id
CS-101

Set Difference (Minus) Operation:

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- **Notation**: r – s
- Set differences must be taken between compatible relations.
  r and s must have the same arity
  attribute domains of r and s must be compatible
- **Example:** to find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

  $\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2017}(section)) -$
  $\Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2018}(section))$
- **Result:**

course_id
CS-347
PHY-101

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- **Example:** Find information about courses taught by instructors in the Physics department with salary greater than 90,000

  Query 1

  $\sigma_{dept\_name=\text{“Physics”} \wedge salary > 90,000}(instructor)$

  Query 2

  $\sigma_{dept\_name=\text{“Physics”}}(\sigma_{salary > 90.000}(instructor))$

  The two queries are not identical; they are, however, equivalent -- they give the same result on any database.

3.1.2 The CARTESIAN PRODUCT (CROSS PRODUCT) Operation

Cartesian Product Operation in Relational Algebra

- Applying CARTESIAN PRODUCT on two relations that is on two sets of tuples, it will take every tuple one by one from the left set(relation) and will pair it up with all the tuples in the right set(relation).
- So, the CROSS PRODUCT of two relation A(R1, R2, R3, …, Rp) with degree p, and B(S1, S2, S3, …, Sn) with degree n, is a relation C(R1, R2, R3, …, Rp, S1, S2, S3, …, Sn) with degree p + n attributes.
- **Notation:** A ⨯ S
  where A and S are the relations,
  the symbol '⨯' is used to denote the CROSS PRODUCT operator.
- **Example:**
  Consider two relations STUDENT(SNO, FNAME, LNAME) and DETAIL(ROLLNO, AGE) below:

SNO	FNAME	LNAME
1	Albert	Singh
2	Nora	Fatehi

ROLLNO	AGE
5	18
9	21

- On applying CROSS PRODUCT on STUDENT and DETAIL:
  STUDENT $\times$ DETAILS

SNO	FNAME	LNAME	ROLLNO	AGE
1	Albert	Singh	5	18
1	Albert	Singh	9	21
2	Nora	Fatehi	5	18
2	Nora	Fatehi	9	21

So the number of tuples in the resulting relation on performing CROSS PRODUCT is 2*2 = 4.

The CARTESIAN PRODUCT operation also known as CROSS PRODUCT or CROSS JOIN denoted by × is a binary set operation, but the relations on which it is applied do not have to be union compatible. This set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set).

In general, the result of R(A1, A2, ..., An) × S(B1, B2, ..., Bm) is a relation Q with degree n+m attributes Q(A1, A2, ..., An, B1, B2, ..., Bm), in that order. The resulting relation Q has one tuple for each combination of tuples one from R and one from S. Hence, if R has nR tuples (denoted as |R| = nR), and S has nS tuples, then R × S will have nR * nS tuples

Example: Suppose that we want to retrieve a list of names of each employee’s dependents.

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{gender='F'}}(\text{EMPLOYEE})$$

$$\text{EMPNAMES} \leftarrow \pi_{\text{Fname, Lname, Ssn}}(\text{FEMALE\_EMPS})$$

$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$$

$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{Ssn=Essn}}(\text{EMP\_DEPENDENTS})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname, Dependent\_name}}(\text{AC TUAL\_DEPENDENTS})$$

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	gen	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

The CARTESIAN PRODUCT creates tuples with the combined attributes of two relations. We can SELECT related tuples only from the two relations by specifying an appropriate selection condition after the Cartesian product.

In SQL, CARTESIAN PRODUCT can be realized by using the CROSS JOIN option in joined tables.

Cartesian Product Operation in Relational Algebra

- Applying CARTESIAN PRODUCT on two relations that is on two sets of tuples, it will take every tuple one by one from the left set (relation) and will pair it up with all the tuples in the right set (relation).
- So, the CROSS PRODUCT of two relation A(R1, R2, R3, …, Rp) with degree p, and B(S1, S2, S3, …, Sn) with degree n, is a relation C(R1, R2, R3, …, Rp, S1, S2, S3, …, Sn) with degree p + n attributes.
- **Notation:** A ⨉ S
  where A and S are the relations,
  the symbol '⨉' is used to denote the CROSS PRODUCT operator.
- **Example:**
  Consider two relations STUDENT(SNO, FNAME, LNAME) and DETAIL(ROLLNO, AGE) below:

SNO	FNAME	LNAME
1	Albert	Singh
2	Nora	Fatehi

ROLLNO	AGE
5	18
9	21

On applying CROSS PRODUCT on STUDENT and DETAIL:

STUDENT ⨯ DETAILS

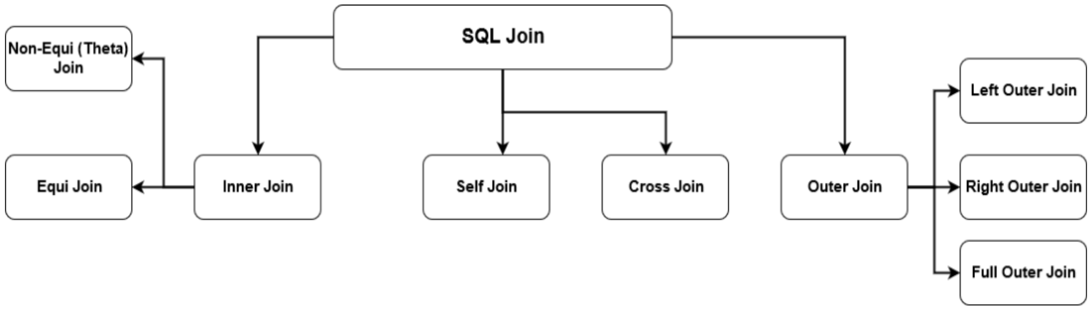
SNO	FNAME	LNAME	ROLLNO	AGE
1	Albert	Singh	5	18
1	Albert	Singh	9	21
2	Nora	Fatehi	5	18
2	Nora	Fatehi	9	21

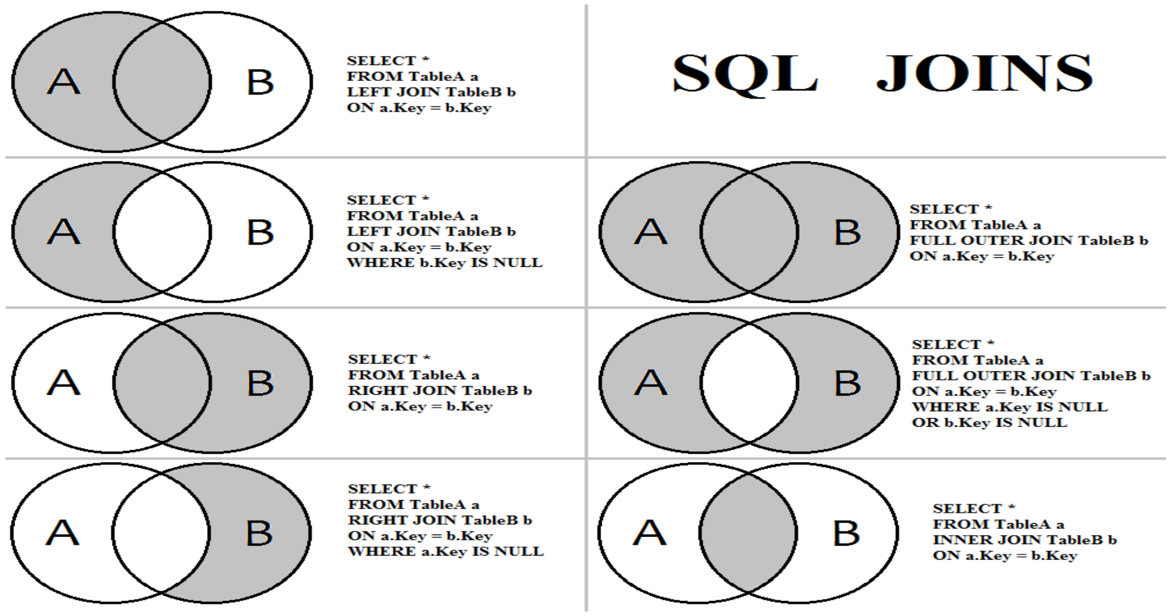
So the number of tuples in the resulting relation on performing CROSS PRODUCT is 2*2 = 4.

3.5 Binary Relational Operations: JOIN and DIVISION:

3.5.1 The JOIN Operation

- **JOIN Operation:** Join in DBMS is a binary operation which allows you to combine join product and selection in one single statement.
- The goal of creating a join condition is that it helps you to combine the data from two or more DBMS tables.
- The tables in DBMS are associated using the primary key and foreign keys.
- **Types of Join**





1. Natural Join:

A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names. It is denoted by ⋈.

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

Input: ∏ EMP_NAME, SALARY (EMPLOYEE ⋈ SALARY)

Output:

EMP_NAME	SALARY
Stephan	50000
Jack	30000
Harry	25000

2. Outer Join:

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

Example:

EMPLOYEE

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

FACT_WORKERS

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

Input: (EMPLOYEE ⋈ FACT_WORKERS)

Output:

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

An outer join is basically of three types:

a) Left outer join
b) Right outer join
c) Full outer join

a) Left Outer Join: Left outer join contains the set of tuples of all combinations in R and S that are equal on their common attribute names.

- In the left outer join, tuples in R have no matching tuples in S.
- It is denoted by ⟕.
- **Example:** Using the above EMPLOYEE table and FACT_WORKERS table.
- **Input:** EMPLOYEE ⟕ FACT_WORKERS
- **Output:**

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

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

- In right outer join, tuples in S have no matching tuples in R.
- It is denoted by ⟖.
- **Example:** Using the above EMPLOYEE table and FACT_WORKERS Relation
- **Input:** EMPLOYEE ⟖ FACT_WORKERS
- **Output:**

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

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

- In full outer join, tuples in R that have no matching tuples in S and tuples in S that have no matching tuples in R in their common attribute name.
- It is denoted by ⟗.
- **Example:** Using the above EMPLOYEE table and FACT_WORKERS table
- **Input:** EMPLOYEE ⟗ FACT_WORKERS
- **Output:**

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

1) **Equi join:** It is also known as an inner join. It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

Example:

CUSTOMER RELATION

CLASS_ID	NAME
1	John
2	Harry
3	Jackson

PRODUCT

PRODUCT_ID	CITY
1	Delhi
2	Mumbai
3	Noida

Input: CUSTOMER ⋈ PRODUCT
Output:

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

The JOIN operation, denoted by ⋈ is used to combine related tuples from two relations into single longer tuples. It allows us to process relationships among relations.The general form of a JOIN operation on two relations R(A1, A2, ..., An) and S(B1, B2, ..., Bm) is

R ⋈ <join condition>S

Example: Retrieve the name of the manager of each department.

To get the manager's name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr_ssn value in the department tuple

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn}=\text{Ssn}} \text{EMPLOYEE}$$
$$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT\_MGR})$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

The result of the JOIN is a relation Q with n + m attributes Q(A1, A2, ..., An,B1, B2, ..., Bm in that order. Q has one tuple for each combination of tuples one from R and one from S whenever the combination satisfies the join condition. This is the main difference between CARTESIAN PRODUCT and JOIN. In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result. The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples.

Each tuple combination for which the join condition evaluates to TRUE is included in the resulting relation Q as a single combined tuple. A general join condition is of the form

<condition> AND <condition> AND...AND <condition>

where each <condition> is of the form Ai Bj, Ai is an attribute of R, B is an attribute of S, Ai and Bj have the same domain, and θ (theta) is one of the comparision operators {=, <, >, ≤, ≥, ≠}. A JOIN operation with such a general join condition is called as **THETA JOIN**. Tuples whose join attributes are NULL or for which the join condition is FALSE do not appear in the result.

3.5.2 Variations of JOIN: The EQUIJOIN and NATURAL JOIN

The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUIJOIN.In the result of an EQUIJOIN we always have one or more pairs of attributes that have identical values in every tuple.

For example the values of the attributes Mgr_ssn and Ssn are identical in every tuple of DEPT_MGR (the EQUIJOIN result) because the equality join condition specified on these two attributes requires the values to be identical in every tuple in the result.

The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first. Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project.first we rename the Dnumber attribute of DEPARTMENT to Dnum so that it has the same name as the Dnum attribute in PROJECT and then we apply NATURAL JOIN:

$$\textbf{PROJ\_DEPT} \leftarrow \textbf{PROJECT} * _{\textbf{(Dname, Dnum, Mgr\_ssn, Mgr\_start\_date)}}\textbf{(DEPARTMENT)}$$

The same query can be done in two steps by creating an intermediate table DEPT as follows:

$$\textbf{DE PT} \leftarrow \rho_{\textbf{(Dname, Dnum, Mgr\_ssn,Mgr\_start\_date)}}\textbf{(DEPARTMENT)}$$

$$\textbf{PROJ\_DEPT} \leftarrow \textbf{PROJECT} * \textbf{DEPT}$$

The attribute Dnum is called the join attribute for the NATURAL JOIN operation, because it is the only attribute with the same name in both relations.

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

If the attributes on which the natural join is specified already have the same names in both relations, renaming is unnecessary. For example, to apply a natural join on the Dnumber attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write

$$\textbf{DEPT\_LOCS} \leftarrow \textbf{DEPARTMENT} * \textbf{DEPT\_LOCATIONS}$$

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

In general, the join condition for NATURAL JOIN is constructed by equating each pair of join attributes that have the same name in the two relations and combining these conditions with AND. If no combination of tuples satisfies the join condition, the result of a JOIN is an empty relation with zero tuples.

A more general, but nonstandard definition for NATURAL JOIN is

$$Q \leftarrow R *_{(\langle list1 \rangle),(\langle list2 \rangle)} S$$

where,

<list1> : list of i attributes from R,

<list2> : list of i attributes from S

The lists are used to form equality comparison conditions between pairs of corresponding attributes and then the conditions are then ANDed together. Only the list corresponding to attributes of the first relation R <list1> is kept in the result Q.

In general, if R has nR tuples and S has nS tuples, the result of a JOIN operation R $\bowtie$ <join condition> S will have between zero and nR * nS tuples. The expected size of the join result divided by the maximum size nR * nS leads to a ratio called join selectivity, which is a property of each join condition. If there is no join condition, all combinations of tuples qualify and the JOIN degenerates into a CARTESIAN PRODUCT, also called CROSS PRODUCT or CROSS JOIN.

A single JOIN operation is used to combine data from two relations so that related information can be presented in a single table. These operations are also known as inner joins. Informally, an inner join is a type of match and combine operation defined formally as a combination of CARTESIAN PRODUCT and SELECTION. The NATURAL JOIN or EQUIJOIN operation can also be specified among multiple tables, leading to an n-way join. For example, consider the following three-way join:

$$((\text{PROJECT} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT}) \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$$

This combines each project tuple with its controlling department tuple into a single tuple, and then combines that tuple with an employee tuple that is the department manager. The net result is a consolidated relation in which each tuple contains this project-department-manager combined information.

In SQL, JOIN can be realized in several different ways

- The first method is to specify the <join conditions> in the WHERE clause, along with any other selection conditions.
- The second way is to use a nested relation
- Another way is to use the concept of joined tables

3.5.3 A Complete Set of Relational Algebra Operations

The set of relational algebra operations $\{\sigma, \pi, \cup, \rho, -, \times\}$ is a complete set; that is, any of the other original relational algebra operations can be expressed as a sequence of operations from this set. For example, the INTERSECTION operation can be expressed by using UNION and MINUS as follows:

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

As another example, a JOIN operation can be specified as a CARTESIAN PRODUCT followed by a SELECT operation,

$$R \bowtie_{<\text{condition}>} S \equiv \sigma_{<\text{condition}>}(R \times S)$$

Similarly, a NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME and followed by SELECT and PROJECT operations. Hence, the various JOIN operations are also not strictly necessary for the expressive power of the relational algebra.

3.5.4 The DIVISION Operation

- **Division operation:** The division operator is used for queries which involve the 'all'. R1 ÷ R2 = tuples of R1 associated with all tuples of R2.
- **Example:** Retrieve the name of the subject that is taught in all courses.

Name	Course
System	Btech
Database	Mtech
Database	Btech
Algebra	Btech

÷

Course
Btech
Btech

=

Name
database

- The expression:
  Smith ← $\Pi_{Pno}(\sigma_{Ename = \text{'john smith'}}$ (employee * works on Pno=Eno))

Consider the Employee table given below −

Name	Eno	Pno
John	123	P1
Smith	123	P2
A	121	P3

÷

Works on the following −

Eno	Pno	Pname
123	P1	Market
123	P2	Sales

=

The result is as follows

Eno
123

The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications. An example is Retrieve the names of employees who work on all the projects that 'John Smith' works on. To express this query using the DIVISION operation, proceed as follows.

- First, retrieve the list of project numbers that'John Smith' works on in the intermediate relation SMITH_PNOS:

$$SSN\_PNOS \leftarrow \pi_{Essn,\ Pno}(WORKS\_ON)$$

- Next, create a relation that includes a tuple <Pno, Essn> whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

$$SMITH \leftarrow \sigma_{Fname=\text{'John'}\ AND\ Lname=\text{'Smith'}}(EMPLOYEE)$$
$$SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$$

- Finally, apply the DIVISION operation to the two relations, which gives the desired employees' Social Security numbers:

$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$
$$RESULT \leftarrow \pi_{Fname,\ Lname}(SSNS * EMPLOYEE)$$

(a)
SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SSNS

Ssn
123456789
453453453

SMITH_PNOS

Pno
1
2

In general, the DIVISION operation is applied to two relations R(Z) ÷ S(X), where the attributes of R are a subset of the attributes of S; that is, X Z.Let Y be the set of attributes of R that are not attributes of S; that is, Y = Z X (and hence Z = X Y). The result of DIVISION is a relation T(Y) that includes a tuple t if tuples tR appear in R with tR [Y] = t, and with tR [X] = tS for every tuple tS in S. This means that, for a tuple t to appear in the result T of t

Figure below illustrates a DIVISION operation where X = {A}, Y = {B}, and Z = {A, B}.

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

The tuples (values) b1 and b4 appear in R in combination with all three tuples in S; that is why they appear in the resulting relation T. All other values of B in R do not appear with all the tuples in S and are not selected: b2 does not appear with a2, and b3 does not appear with a1.

The DIVISION operation can be expressed as a sequence of π, x and – operations are as follows:

$$T1 \leftarrow \pi_Y(R)$$
$$T2 \leftarrow \pi_Y((S \times T1) - R)$$
$$T \leftarrow T1 - T2$$

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<selection\ condition>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<attribute\ list>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<join\ condition>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<join\ condition>} R_2$, OR $R_1 \bowtie_{(<join\ attributes\ 1>),(<join\ attributes\ 2>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<join\ condition>} R_2$, OR $R_1 *_{(<join\ attributes\ 1>),(<join\ attributes\ 2>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Table: Operations of Relational Algebra

3.7 Examples of Queries in Relational Algebra:

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

$$RESEARCH\_DEPT \leftarrow \sigma_{Dname='Research'}(DEPARTMENT)$$
$$RESEARCH\_EMPS \leftarrow (RESEARCH\_DEPT \bowtie_{Dnumber=Dno} EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Fname,\ Lname,\ Address}(RESEARCH\_EMPS)$$

As a single in-line expression, this query becomes:

$$\pi_{Fname,\ Lname,\ Address}(\sigma_{Dname='Research'}(DEPARTMENT \bowtie_{Dnumber=Dno}(EMPLOYEE))$$

Query 2. For every project located in 'Stafford', list the project number, the controllong department number, and the department manager's last name, address and birth date.

$$
\begin{aligned}
&STAFFORD\_PROJS \leftarrow \sigma_{Plocation=\text{'Stafford'}}(PROJECT)\\
&CONTR\_DEPTS \leftarrow (STAFFORD\_PROJS \bowtie_{Dnum=Dnumber} DEPARTMENT)\\
&PROJ\_DEPT\_MGRS \leftarrow (CONTR\_DEPTS \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE)\\
&RESULT \leftarrow \pi_{Pnumber,\ Dnum,\ Lname,\ Address,\ Bdate}(PROJ\_DEPT\_MGRS)
\end{aligned}
$$

Query 3. Find the names of employees who work on all the projects controlled by department number 5.

$$
\begin{aligned}
&DEPT5\_PROJS \leftarrow \rho_{(Pno)}(\pi_{Pnumber}(\sigma_{Dnum=5}(PROJECT)))\\
&EMP\_PROJ \leftarrow \rho_{(Ssn,\ Pno)}(\pi_{Essn,\ Pno}(WORKS\_ON))\\
&RESULT\_EMP\_SSNS \leftarrow EMP\_PROJ \div DEPT5\_PROJS\\
&RESULT \leftarrow \pi_{Lname,\ Fname}(RESULT\_EMP\_SSNS * EMPLOYEE)
\end{aligned}
$$

Query 4. Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager as a department that controls the project.

$$
\begin{aligned}
&SMITHS(Essn) \leftarrow \pi_{Ssn}(\sigma_{Lname=\text{'Smith'}}(EMPLOYEE))\\
&SMITH\_WORKER\_PROJS \leftarrow \pi_{Pno}(WORKS\_ON * SMITHS)\\
&MGRS \leftarrow \pi_{Lname,\ Dnumber}(EMPLOYEE \bowtie_{Ssn=Mgr\_ssn} DEPARTMENT)\\
&SMITH\_MANAGED\_DEPTS(Dnum) \leftarrow \pi_{Dnumber}(\sigma_{Lname=\text{'Smith'}}(MGRS))\\
&SMITH\_MGR\_PROJS(Pno) \leftarrow \pi_{Pnumber}(SMITH\_MANAGED\_DEPTS * PROJECT)\\
&RESULT \leftarrow (SMITH\_WORKER\_PROJS \cup SMITH\_MGR\_PROJS)
\end{aligned}
$$

Query 5. List the names of all employees with two or more dependents.

$$
\begin{aligned}
&T1(Ssn,\ No\_of\_dependents) \leftarrow {}_{Essn}\Im_{COUNT\ Dependent\_name}(DEPENDENT)\\
&T2 \leftarrow \sigma_{No\_of\_dependents>2}(T1)\\
&RESULT \leftarrow \pi_{Lname,\ Fname}(T2 * EMPLOYEE)
\end{aligned}
$$

Query 6. Retrieve the names of employees who have no dependents.

$$
\begin{aligned}
&ALL\_EMPS \leftarrow \pi_{Ssn}(EMPLOYEE)\\
&EMPS\_WITH\_DEPS(Ssn) \leftarrow \pi_{Essn}(DEPENDENT)\\
&EMPS\_WITHOUT\_DEPS \leftarrow (ALL\_EMPS - EMPS\_WITH\_DEPS)\\
&RESULT \leftarrow \pi_{Lname,\ Fname}(EMPS\_WITHOUT\_DEPS * EMPLOYEE)
\end{aligned}
$$

Query 7. List the names of managers who have at least one dependent.

$$
\begin{aligned}
&MGRS(Ssn) \leftarrow \pi_{Mgr\_ssn}(DEPARTMENT)\\
&EMPS\_WITH\_DEPS(Ssn) \leftarrow \pi_{Essn}(DEPENDENT)\\
&MGRS\_WITH\_DEPS \leftarrow (MGRS \cap EMPS\_WITH\_DEPS)\\
&RESULT \leftarrow \pi_{Lname,\ Fname}(MGRS\_WITH\_DEPS * EMPLOYEE)
\end{aligned}
$$