

RELATIONAL MODEL

Module – 2

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.

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.

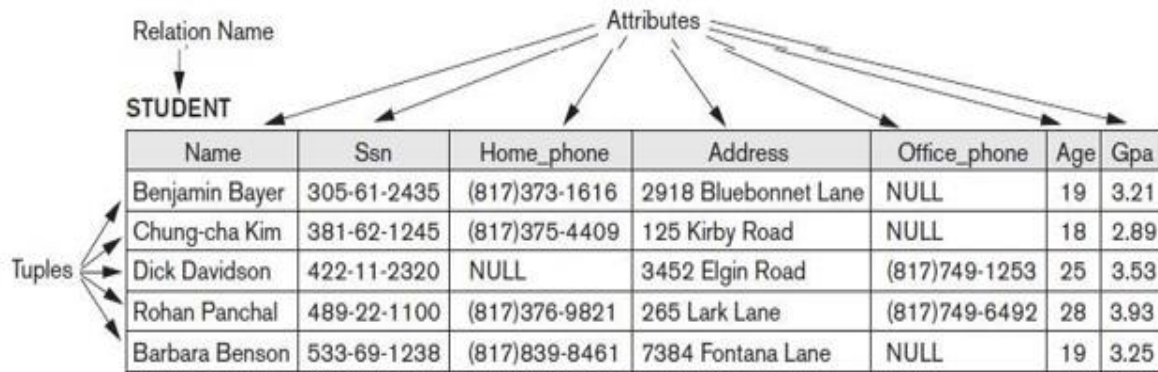
For example, in STUDENT relation because each row represents facts about a particular student entity. The column names—Name, Student_number, Class, and Major—specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type.

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.

Terminologies

Relation

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



Relational Model

Relational Model represents data as a collection of Tables or Relation.

- A table is also called as relation.
- Each row is called as Tuple.
- Each column headers is called as Attributes.

Domain

A domain D is a set of atomic values. By atomic we mean that each value in the domain is indivisible 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 a name for the domain, to help in interpreting its values.

Some examples of domains are given below,

- India_phone_numbers: The set of ten-digit phone numbers valid in India.
- Social_security_numbers (SSN): The set of valid nine-digit Social Security numbers.
- Names: The set of character strings that represent 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 Employee_ages is an integer number between 15 and 80.

Attribute

An 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 $\text{dom}(A_i)$.

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 schema

A relation schema R , denoted by $R(A_1, A_2, \dots, A_n)$, is made up of a relation name R and a list of attributes A_1, A_2, \dots, 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 $\text{dom}(A_i)$. A relation schema is used to describe a relation; R is called the name of this relation.

Example,

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:

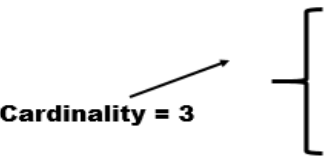
$\text{dom}(\text{Name}) = \text{Names}$; $\text{dom}(\text{Ssn}) = \text{Social_security_numbers}$;
 $\text{dom}(\text{HomePhone}) = \text{phone_numbers}$, $\text{dom}(\text{Office_phone}) = \text{phone_numbers}$,

Degree (or arity) of a 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 shown in the above example.

Cardinality

Total number of tuples present in the relation.

STUDENT			
	Roll_no	Name	Age
Cardinality = 3 	1	Sunil	36
	2	Sonu	31
	3	Liya	25

Relational Database Schema

It is a set of relation schemas and set of integrity constraints. i.e., if we take company database the relational database schema of that company would have the collection of relational schema like,

- Employee relational schema with employee details.
- Department relational schema.
- Project relational schema.

So collection of all these schemas and along with integrity constraints make up relational database.

Relation State (or relation instance)

A relation state (or relation instance) r of the relation schema by $R(A_1, A_2, \dots, A_n)$, also denoted by $r(R)$, is a set of n -tuples $r = \{t_1, t_2, \dots, t_m\}$. Each n -tuple t is an ordered list of n values $t = \langle v_1, v_2, \dots, v_n \rangle$, where each value v_i , $1 \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special NULL value. The i th value in tuple t , which corresponds to the attribute A_i , is referred to as $t[A_i]$ or $t.A_i$.

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

Characteristics of Relations

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.

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 $R(A_1, A_2, \dots, A_n)$, set of attributes and a relation state $r(R)$ is a finite set of mappings $r = \{t_1, t_2, \dots, t_m\}$, where each tuple t_i 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 ($\langle \text{attribute} \rangle, \langle \text{value} \rangle$) pairs, where each pair gives the value of the mapping from an attribute A_i to a value v_i from $\text{dom}(A_i)$. The ordering of attributes is not important, because the attribute name appears with its value.

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).

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 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.

Relational Model Constraints and Relational Database Schemas

Constraints are restrictions or limitations 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:

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.

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.

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's supervisor and the maximum number of hours an employee can work on all projects per week is 56.

But here we mainly focus on schema based constraints the includes,

- Domain Constraints
- Key constraints
- Constraints on Null
- Entity Integrity Constraints
- Referential Integrity constraints

Domain Constraints

Domain Constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain $\text{dom}(A)$. It perform data type check. 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.

STUDENT	Roll_no	Name	Phone Number	Age
	1	Sunil	1234567891	36
	2	Sonu	7894563217	31
	3	Liya	4569871239	A

Violates Domain Constraints →

Key Constraints

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 t_1 and t_2 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

An attribute that can uniquely identify each tuple in a relation is called a **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:

- 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.
- It is a minimal superkey—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 hold. This property is not required by a superkey.

STUDENT	<u>Roll_no</u>	Name	Age	Email
	1	Sunil	36	sunil@gmail.com
	2	Sonu	31	sonu@gmail.com
	2	Liya	25	Liya@gmail.com

Not Possible →

Example: Consider the STUDENT relation

- The attribute set {Roll_no} is a key of STUDENT because no two student tuples can have the same value for Roll_no
- Any set of attributes that includes Roll_no—for example, {Roll_no, Name, Age}—is a superkey.
- The superkey {Roll_no, 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.

Key
↓

STUDENT	<u>Roll_no</u>	Name	Age	Grade
	1	Sunil	36	A
	2	Sonu	31	A
	3	Liya	25	B

Candidate key

A relation schema may have more than one key. In this case, each of the keys is called a candidate key. For example, the CAR relation has two candidate keys: License_number and Engine_serial_number.

CAR

<u>License_number</u>	<u>Engine_serial_number</u>	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.

Constraints on Null values

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.

Entity integrity constraint

The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

Key constraints and entity integrity constraints are specified on individual relations.

STUDENT	<u>Roll_no</u>	Name	Age	Email
	1	Sunil	36	sunil@gmail.com
	2	Sonu	31	sonu@gmail.com
	Null	Liya	25	Liya@gmail.com

No Primary key value can be Null →

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.

For 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.

EMPLOYEE	<u>EMPID</u>	Name	DNo
	1001	Sunil	3
	1002	Sonu	1
	1003	Liya	2

Foreign Key →

DEPARTMENT	<u>DNo</u>	DName
	1	CS
	2	IS
	3	DS

The attribute of one relation that refers to primary key of another relation is called **foreign key**.

A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following rules:

- Attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2.
- A value of FK in a tuple t1 of the current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is NULL.

In the former case, we have $t1[FK] = t2[PK]$, and we say that the tuple t1 references or refers to the tuple t2.

In this definition, R1 is called the **referencing relation** and R2 is the **referenced relation**. If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold.

Other Types of Constraints

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.

Functional dependency constraint

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 designs 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.

Update Operations, Transactions and Dealing with Constraint Violations

The operations of the relational model can be categorized into **retrievals and updates**. There are three basic operations that can change the states of relations in the database:

- Insert - used to insert one or more new tuples in a relation.
- Delete - used to delete tuples.
- Update (or Modify) - used to change the values of some attributes in existing tuples.

Whenever these operations are applied, the integrity constraints specified on the relational database schema should not be violated.

The Insert Operation

The Insert operation provides a list of attribute values for a new tuple t that is to be inserted into an relation R . Insert can violate any of the four types of constraints

- **Domain constraints:** if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.
- **Key constraints:** if a key value in the new tuple t already exists in another tuple in the relation $r(R)$.
- **Entity integrity:** if any part of the primary key of the new tuple t is NULL.
- **Referential integrity:** if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

Example:

EMPLOYEE	<u>EMPID</u>	Name	AGE	PHONE	DNo
	1001	Sunil	36	1234567891	2
	1002	Sonu	31	7894563217	1

DEPARTMENT	<u>DNo</u>	DName	Location
	1	HR	Mysore
	2	Finance	Mysore

1. Operation:

Insert <1003, 'Liya', 25, 4567891234, 'A'>

Result: This insertion violates domain constraints as DNo is not of the appropriate data type.

2. Operation:

Insert <1002, 'Liya', 25, 4567891234, '2'>

Result: This insertion violates the key constraint because another tuple with the same EMPID value already exists in the EMPLOYEE relation, and so it is rejected.

3. Operation:

Insert <Null, 'Liya', 25, 4567891234, '2'>

Result: This insertion violates the entity integrity constraint (NULL for the primary key EMPID), so it is rejected.

4. Operation:

Insert <1003, 'Liya', 25, 4567891234, '5'>

Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding referenced tuple exists in DEPARTMENT with DNo = 5.

5. Operation:

Insert <1003, 'Liya', 25, 4567891234, '2'>

Result: This insertion satisfies all constraints, so it is acceptable.

If an insertion violates one or more constraints, the default option is to reject the insertion. It would be useful if the DBMS could provide a reason to the user as to why the insertion was rejected. Another option is to attempt to correct the reason for rejecting the insertion

The delete Operation

The Delete operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted.

Example:

EMPLOYEE	<u>EMPID</u>	Name	AGE	PHONE
	1001	Sunil	36	1234567891
	1002	Sonu	31	7894563217

PROJECT	<u>PNo</u>	PName	DNo
	1	P1	2
	2	P2	1

WORKS_ON	<u>EMPID</u>	PNO	Hours
	1001	1	40
	1002	2	50
	1003	1	60

1. Operation:

Delete the WORKS_ON tuple with EMPID = 1003 and PNo = 1.

Result: This deletion is acceptable and deletes exactly one tuple.

2. Operation:

Delete the EMPLOYEE tuple with EMPID = 1001

Result: This deletion is not acceptable, because there are tuples in WORKS_ON that refer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.

Several options are available if a deletion operation causes a violation

- **Restrict** - is to reject the deletion.
- **Cascade**, is to attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted.
- **Set null or set default** - is to modify the referencing attribute values that cause the violation; each such value is either set to NULL or changed to reference another default valid tuple.

The Update (or Modify) Operation

The Update (or Modify) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified.

Example:

EMPLOYEE	<u>EMPID</u>	Name	AGE	PHONE	DNo
	1001	Sunil	36	1234567891	2
	1002	Sonu	31	7894563217	1

DEPARTMENT	<u>DNo</u>	DName	Location
	1	HR	Mysore
	2	Finance	Mysore

WORKS_ON	<u>EMPID</u>	PNO	Hours
	1001	1	40
	1002	2	50
	1003	1	60

1. Operation:

Update the DNo of the EMPLOYEE tuple with EMPID =1000 to 1.

Result: It is acceptable as it does not violate any of the 4 constraints.

2. Operation:

Update the DNo of the EMPLOYEE tuple with EMPID =1000 to 7.

Result: Unacceptable, because it violates referential integrity.

3. Operation:

Update the DNo of the EMPLOYEE tuple with EMPID =1000 to 1001.

Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of EMPID.

Updating an attribute that is neither part of a primary key nor of a foreign key usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain.

Exercise Problems based on Relational Model

1. Suppose that each of the following update operations is applied directly to the database state as shown below. Discuss all integrity constraints violated by each operation, if any and the different ways of enforcing these constraints.

STUDENT	<u>Reg_No</u>	FName	LName	DNo
	1000	Sunil	Sonu	1
	1001	Lashika	Liya	1
	1002	Supi	Sathi	3

DEPARTMENT	<u>DNo</u>	DName	Building
	1	CS	B1
	2	IS	B2
	3	DS	B3
	4	AI	B4

INSTRUCTOR	<u>TID</u>	FName	LName	Salary	DNo
	001	Sunil	Sonu	50000	4
	002	Lashika	Liya	50000	1
	003	Supi	Sathi	50000	1

1. Operation:

Insert (1004,'smith','jones', 2) into student relation.

1004	Smith	Jones	2
------	-------	-------	---

- **Domain Constraints:** It is not violated as it matches all the attributes.
- **Key constraints:** it is not violated as 1004 does exist of primary key and it is unique.
- **Entity integrity constraint:** no primary key value is Null, so it is not violated.
- **Referential integrity constraint:** it is not violated as value of the foreign key should refer to an existing tuple in the relation it refers to or it should be Null.

Solution: Therefore no constraints violations.

2. Operation:

Insert (004,'smith','jones', 55000, 5) into instructor relation.

004	Smith	Jones	55000	5
-----	-------	-------	-------	---

- **Domain Constraints:** It is not violated as it matches all the attributes.
- **Key constraints:** it is not violated as there is no duplication of key value.
- **Entity integrity constraint:** no primary key value is Null, so it is not violated.
- **Referential integrity constraint:** it is violated, it does not match the foreign key DNo = 5 to refer primary key in department relation.

Solution: we may enforce the constraints by following,

- Rejecting the insertion.
- Changing the value of DNo in the new tuple any of the value that is already in attribute that is referred to.
- Insert the new tuple in department relation with DNo= 5.

3. Operation:

Insert (4,'ML', B5) into Department relation.

4	ML	B4
----------	-----------	-----------

- **Domain Constraints:** It is not violated as it matches all the attributes.
- **Key constraints:** it is violated as DNo=4 is already existing of primary key.
- **Entity integrity constraint:** no primary key value is Null, so it is not violated.
- **Referential integrity constraint:** it is not violated as value of the foreign key should refer to an existing tuple in the relation it refers to or it should be Null.

Solution: we may enforce the constraints by following,

- Rejecting the insertion.(default option)
- Changing the value of DNo in the Relation department.

4. Operation:

Insert (Null, 'sammy', 'somu', 7) into student relation.

Null	Sammy	Somu	7
-------------	--------------	-------------	----------

- **Domain Constraints:** It is not violated as it matches all the attributes.
- **Key constraints:** it is not violated.
- **Entity integrity constraint:** primary key value is Null, so it is violated.
- **Referential integrity constraint:** it is violated.

Solution: for entity integrity violation we may enforce the constraints by following,

- Rejecting the insertion.(default option)
- Changing the value of Reg_no in the Relation student.

For Referential integrity violation we may enforce the constraints by following,

- Rejecting the insertion.(default option)
- Changing the value of DNo into any other value which already exist in referred relation.
- Insert new tuple in department relation with DNo=7.

5. Operation:

Delete the instructor tuples with TID=002.

In delete operation only referential integrity constraints can be violated and no other constraints can be violated.

Solution: No constraints violations.

6. Operation:

Delete the Department tuples with DNo=1.

Solution: violates referential integrity constraints and we may enforce this constraint by,

- Rejecting the insertion. (Default option).
- Deleting all the tuples that reference this tuple that is being deleted.

7. Operation:

Modify the DNo of Student tuple with Reg_No= 1002 to 4.

1002	Supi	Sathi	4
-------------	-------------	--------------	----------

Solution: No constraints violations.

8. Operation:

Modify the DNo of attribute of instructor tuple with TID=003to 8.

003	Supi	Sathi	50000	8
------------	-------------	--------------	--------------	----------

Solution: violates referential integrity constraints and we may enforce this constraint by,

- Rejecting the insertion. (Default option).
- Insert new tuple in department relation with DNo=8.

Relational Algebra

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.

Relational algebra operations: Unary

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.

General Syntax,

$$\sigma \langle \text{selection condition} \rangle (R)$$

Where,

- the symbol σ 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 in 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 $\langle \text{selection condition} \rangle$ is made up of a number of clauses of the form:

$\langle \text{attribute name} \rangle \langle \text{comparison op} \rangle \langle \text{constant value} \rangle$

or

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

Where,

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

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

<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.

Examples:

1. Select the employee tuples where department number is 2.

EMPLOYEE	FName	LName	DNo
	Sunil	Sonu	3
	Supi	Sathi	2
	Abhi	Bhat	2

Solution: $\sigma_{DNo = 2} (EMPLOYEE)$

FName	LName	DNo
Supi	Sathi	2
Abhi	Bhat	2

2. Select the employee tuples where department number is 2 and salary > 30000.

EMPLOYEE	FName	LName	Salary	DNo
	Sunil	Sonu	30000	3
	Supi	Sathi	20000	2
	Abhi	Bhat	40000	2
	Ajay	Kumar	38000	3

Solution: $\sigma_{DNo = 2 \text{ AND } Salary > 30000} (EMPLOYEE)$

FName	LName	Salary	DNo
Abhi	Bhat	40000	2

3. Select the employee tuples who work in department number is 3 and with salary > 35000 or who work in DNo = 2 and with salary > 25000.

EMPLOYEE	FName	LName	Salary	DNo
	Sunil	Sonu	30000	3
	Supi	Sathi	20000	2
	Abhi	Bhat	40000	2
	Ajay	Kumar	38000	3

Solution: $\sigma_{(DNo = 3 \text{ AND } Salary > 35000) \text{ OR } (DNo = 2 \text{ AND } Salary > 25000)}(EMPLOYEE)$

FName	LName	Salary	DNo
Abhi	Bhat	40000	2
Ajay	Kumar	38000	3

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,

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

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,

$$\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(R)) = \sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond1} \rangle}(R))$$

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,

$$\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(\dots(\sigma_{\langle \text{condn} \rangle}(R)) \dots)) = \sigma_{\langle \text{cond1} \rangle \text{ AND } \langle \text{cond2} \rangle \text{ AND } \dots \text{ AND } \langle \text{condn} \rangle}(R)$$

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

σ_{Dno=4 AND Salary>25000}(EMPLOYEE)

Would do the following SQL query:

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

2. The PROJECT Operation OR PROJECT Operation

The PROJECT operation denoted by π (pi) 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

$\pi_{\langle \text{attribute list} \rangle}(\mathbf{R})$

Where,

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

$\langle \text{attributelist} \rangle$ - desired sublist of attributes from the attributes of relation R.

The result of the PROJECT operation has only the attributes specified in $\langle \text{attribute list} \rangle$ in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in $\langle \text{attribute list} \rangle$.

Examples:

1. To list each employee's first name, last name, and salary we can use the projection operation.

EMPLOYEE	FName	LName	Salary	DNo
	Sunil	Sonu	30000	3
	Supi	Sathi	30000	2

Solution:

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

FName	LName	Salary
Sunil	Sonu	30000
Supi	Sathi	30000

3. Sequences of Operations and the RENAME Operation

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:

EMPLOYEE	FName	LName	Salary	DNo
	Sunil	Sonu	30000	3
	Supi	Sathi	20000	2
	Abhi	Bhat	40000	2
	Ajay	Kumar	38000	3

$$\pi_{\text{FName, LName}}(\sigma_{\text{Dno}=3}(\text{EMPLOYEE}))$$

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.

$$\text{TEMP} \leftarrow \sigma_{\text{Dno}=3}(\text{EMPLOYEE})$$

TEMP	FName	LName	Salary	DNo
	Sunil	Sonu	30000	3
	Ajay	Kumar	38000	3

$$\text{R} \leftarrow \pi_{\text{FName, LName}}(\text{TEMP})$$

R	FName	LName
	Sunil	Sonu
	Ajay	Kumar

Now we can rename,

$$\text{R}(\text{First name, Last name}) \leftarrow \pi_{\text{FName, LName}}(\text{TEMP})$$

R	First Name	Last Name
	Sunil	Sonu
	Ajay	Kumar

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

1. $\rho_{(B_1, B_2, \dots, B_n)}(R)$ ρ (rho) – RENAME operator
2. $\rho_{S(R)}$ S – new relation name
3. $\rho_{(B_1, B_2, \dots, B_n)}(R)$ B₁, B₂, ..., 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, ..., A_n) in that order, then each A_i is renamed as B_i.

Relational Algebra Operations from Set Theory

1. The UNION Operations ($R \cup S$)

The result of this operation, denoted by $R \cup S$, 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.

Example: Consider the following two relations: STUDENT & INSTRUCTOR relation table.

STUDENT	FN	LN	INSTRUCTOR	FName	LName
↑ R	Sunil	Sonu	↑ S	Supi	Sathi
	Supi	Sathi		Somu	Ramu
	Abhi	Bhat		Abhi	Bhat
	Ajay	Kumar			

Result: $R \cup S$

FN	LN
Sunil	Sonu
Supi	Sathi
Abhi	Bhat
Ajay	Kumar
Somu	Ramu

2. The INTERSECTION Operations ($R \cap S$)

The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S.

Example: Consider the following two relations: STUDENT & INSTRUCTOR relation table.

STUDENT	FN	LN	INSTRUCTOR	FName	LName
	Sunil	Sonu		Supi	Sathi
	Supi	Sathi		Somu	Ramu
	Abhi	Bhat		Abhi	Bhat
	Ajay	Kumar			

Result: $R \cap S$

FN	LN
Supi	Sathi
Abhi	Bhat

3. The MINUS Operations OR Set Difference ($R - S$)

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 following two relations: STUDENT & INSTRUCTOR relation table.

STUDENT	FN	LN	INSTRUCTOR	FName	LName
	Sunil	Sonu		Supi	Sathi
	Supi	Sathi		Somu	Ramu
	Abhi	Bhat		Abhi	Bhat
	Ajay	Kumar			

Result: $R - S$

FN	LN
Sunil	Sonu
Ajay	Kumar

Result: $S - R$

FN	LN
Somu	Ramu

4. The CARTESIAN Product OR Cross Product OR Cross Joint Operation ($R \times S$)

The CARTESIAN PRODUCT operation—also known as CROSS PRODUCT or CROSS JOIN denoted by \times 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 $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$, 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 $R \times S$ will have $nR * nS$ tuples.

Example: Consider given two relations below and perform Cartesian product operation.

R	A	B	S	C	D	E
	101	104		P	A	X
	102	105		Q	B	Y
	103	106				

Result: $R \times S$

M	A	B	C	D	E
	101	104	P	A	X
	101	104	Q	B	Y
	102	105	P	A	X
	102	105	Q	B	Y
	103	106	P	A	X
	103	106	Q	B	Y

- **R – Degree – $n = 2 = 2$ attributes**
- **S- Degree – $m = 3 = 3$ attributes**
- **So, $n+m = 5$ attributes.**
- **R – Cardinality – 3**
- **S - Cardinality – 2**
- **So, $(R \times S) = (3 \times 2) = 6$ Tuples.**

Relational Algebra Operations – Binary

1. The JOIN Operation (\bowtie)

The JOIN operation, denoted by \bowtie , 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 $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_m)$ is

$$R \bowtie_{\langle \text{join condition} \rangle} S$$

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

Department	DName	DNo	Mgr_SSN
	Research	2	123456789
	Finance	5	789456123

Employee	SSN	FName	LName	DNo
	456789123	Sunil	Sonu	2
	123456789	Supi	Sathi	5
	789456123	Abhi	Bhat	5
	321654987	Ajay	Kumar	2

$$\text{Dept_Mgr} \leftarrow \text{Department} \bowtie_{\text{Mgr_SSN} = \text{SSN}} \text{Employee}$$

This join operation is nothing but Cartesian operation followed by selection operation.

$$\text{TEMP} \leftarrow \text{Department} \times \text{Employee}$$

$$\text{Dept_Mgr} \leftarrow \sigma_{\text{Mgr_SSN} = \text{SSN}} \text{TEMP}$$

Result: of the JOIN operation is,

Dept_Mgr	DName	DNo	Mgr_SSN	SSN	FName	LName	DNo
	Research	2	123456789	123456789	Supi	Sathi	5
	Finance	5	789456123	789456123	Abhi	Bhat	5

2. The Theta JOIN (θ)

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 $A_i \theta B_j$, A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and θ (theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$. A JOIN operation with such a general join condition is called a THETA JOIN. Tuples whose join attributes are NULL or for which the join condition is FALSE do not appear in the result.

<JOIN Condition>: $A_i \theta B_j$

Syntax,

$R \bowtie_{A_i \theta B_j} S$

Example:

R	A ₁	A ₂	S	B ₁
	20	25		50
	80	40		35

Operations:

1. $R \bowtie_{A_2 = B_1} S$
2. $R \bowtie_{A_2 < B_1} S$
3. $R \bowtie_{A_2 \leq B_1} S$
4. $R \bowtie_{A_2 > B_1} S$
5. $R \bowtie_{A_2 \geq B_1} S$
6. $R \bowtie_{A_2 \neq B_1} S$

We consider one operation example, $R \bowtie_{A_2 > B_1} S$ we should first perform Cartesian product for R and S that is stored in Result X as displayed below.

X	A ₁	A ₂	B ₁
	20	25	50
	20	25	35
	80	40	50
	80	40	35

Result of JOIN $\bowtie_{A_2 > B_1}$ operation is,

A ₁	A ₂	B ₁
80	40	35

Variations of JOIN: The EQUIJOIN and NATURAL JOIN

1. The EQUIJOIN Operation

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.

Dept_Mgr \leftarrow **Department** \bowtie **Mgr_SSN = SSN** **Employee**

2. NATURAL JOIN Operation

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 DNo attribute of DEPARTMENT to Dnum—so that it has the same name as the Dnum attribute in PROJECT—and then we apply NATURAL JOIN:

Example:

Project	PID	PName	DNum
	101	P1	1
	102	P2	2
	103	P3	2

Department	DNo	Mgr_SSN
	1	123456789
	2	789456321

Project_Dept \leftarrow **Project** * ρ (Dnum, Mgr_SSN) (**DEPARTMENT**)

DEPT \leftarrow ρ (Dnum, Mgr_SSN) (**DEPARTMENT**)

Project_Dept \leftarrow **Project** * **DEPT**

Result of * (Natural JOIN) is given below,

Project_Dept	PID	PName	DNum	Mgr_SSN
	101	P1	1	123456789
	102	P2	2	789456321
	103	P3	2	789456321

3. DIVISION Operation (\div)

The DIVISION operation, denoted by \div , 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.

Example:

Employee	EID	PID
	1001	1
	1002	1
	1002	2
	1003	2

Project	PID
	1
	2

Result \leftarrow Employee \div Project

So, result of the Division operation is as shown below,

Result	EID
	1002

Series of Operations or Steps i.e., projection, cross product and minus operation.

Example: Operations

1. $T1 \leftarrow \pi_x (R)$
2. $T2 \leftarrow \pi_x (\pi \times S) - R$
3. **Result $\leftarrow T1 - T2$**

	x	y
	\downarrow	\downarrow
Employee	EID	PID
	1001	1
	1002	1
	1002	2
	1003	2

Project	PID
	1
	2

\uparrow
R

\uparrow
S

Solution:

T1		T1 X S		$(\pi \times S) - R$		T2	
T1	EID	EID	PID	EID	PID	T2	EID
	1001	1001	1	1001	2		1001
	1002	1001	2				1003
	1003	1002	1	1003	1		
		1002	2				
		1003	1				
		1003	2				

Result	EID
	1002

Table 6.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R .	$\sigma_{\langle \text{selection condition} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of R , and removes duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$, OR $R_1 \bowtie_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$
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.	$R_1 \star_{\langle \text{join condition} \rangle} R_2$, OR $R_1 \star_{(\langle \text{join attributes 1} \rangle), (\langle \text{join attributes 2} \rangle)} R_2$ OR $R_1 \star R_2$
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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Additional Relational Algebra Operations

Some database request specified by database users cannot be performed with the original relational algebra operations which have learnt so far. i.e., these request cannot be performed with unary, set theory and relational algebra operations.

Aggregate Functions and Grouping

Aggregate functions are used in simple statistical queries that summarize information from the database tuples. Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values. For example, retrieving the average or total salary of all employees or the total number of employee tuples.

Grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group. For example, group EMPLOYEE tuples by Dno, so that each group includes the tuples for employees working in the same department. We can then list each Dno value along with, say, the average salary of employees within the department, or the number of employees who work in the department.

Aggregate function operation can be defined by using the symbol Σ (script F):

<grouping attributes> Σ <function list> (R)

Where,

<grouping attributes> : list of attributes of the relation specified in R

<function list>: list of (<function> <attribute>) pairs.

<function> - such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT

<attribute> is an attribute of the relation specified by R

The resulting relation has the grouping attributes plus one attribute for each element in the function list.

Example:

Employee	FName	SSN	Salary	DNo
	Sunil	123456789	40000	2
	Supi	789654123	30000	2
	Abhi	456123789	30000	1
	Ajay	321654987	20000	2

1. To retrieve the number of employees and their average salary.

COUNT_SSN	Average_Salary
4	30000

- To retrieve the number of employees and their average salary in each department.

DNo	COUNT_SSN	Average_Salary
2	3	30000
1	1	30000

To rename the same we write query as below,

$\rho_{R(Dno, No_of_employees, Average_sal)}(Dno \bowtie COUNT_{Ssn, AVERAGE Salary}(EMPLOYEE))$

Recursive closure operation

Recursive closure operation is applied to a recursive relationship between tuples of the same type, such as the relationship between an employee and a supervisor.

Example:

- To Retrieve SSN of all employees directly supervised by 'Ajay' 'kumar' (at level 1).

Employee	FName	LName	SSN	Super_SSN
	Sunil	Sonu	123456789	321654987
	Supi	Sathi	789654123	123456789
	Abhi	Bhat	456123789	321654987
	Ajay	Kumar	321654987	Null

$D_SSN \leftarrow \pi_{SSN}(\sigma_{FName = 'Ajay' \text{ AND } LName = 'kumar'}(Employee))$

$SUPERVISION(SSN1, SSN2) \leftarrow \pi_{SSN, Super_SSN}(Employee)$

$RESULT\ 1 \leftarrow \pi_{SSN1}(SUPERVISION \bowtie_{SSN2=SSN} D_SSN)$

D_SSN	SSN
	321654987

SUPERVISION	SSN1	SSN2
	123456789	321654987
	789654123	123456789
	456123789	321654987
	321654987	Null

Result of \bowtie Operation

SSN1	SSN2	SSN
123456789	321654987	321654987
456123789	321654987	321654987

RESULT 1	SSN1
	123456789
	456123789

2. To Retrieve SSN of all employees directly supervised by 'Ajay' 'kumar' (at level 2).

RESULT 2 $\leftarrow \pi_{SSN1}(\text{SUPERVISION} \bowtie_{SSN2=SSN} \text{RESULT 1})$

RESULT 1	SSN1
	123456789
	456123789

SUPERVISION	SSN1	SSN2
	123456789	321654987
	789654123	123456789
	456123789	321654987
	321654987	Null

Result of \bowtie Operation

SSN1	SSN2	SSN
789654123	123456789	123456789

RESULT 2	SSN1
	789654123

3. To Retrieve SSN of all employees supervised by 'Ajay' 'kumar' (at level 1 AND at level 2).

RESULT $\leftarrow \text{RESULT 2} \cup \text{RESULT 1}$

OUTER JOIN Operations

A set of operations, called outer joins, were developed for the case where the user wants to keep all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.

There are three types of OUTER JOIN Operations.

1. Left OUTER JOIN

Suppose that we want a list of all employee names as well as the name of the departments they manage if they happen to manage a department; if they do not manage one, we can indicate it with a NULL value. We can apply an operation LEFT OUTER JOIN, denoted by \bowtie_{Left} to retrieve the result as follows:

Example:

Res $\leftarrow R \bowtie_{\text{Left}} S$

R	A	B
	1	a
	2	c

S	C	D
	1	b
	3	d

Res	A	B	C	D
	1	a	1	b
	2	c	Null	Null

2. Right OUTER JOIN

A similar operation, RIGHT OUTER JOIN, denoted by \bowtie_{\rightarrow} keeps every tuple in the second, or right, relation S in the result of $R \bowtie_{\rightarrow} S$.

Example:

$\text{Res} \leftarrow R \bowtie_{A=C} S$

R	A	B
	1	a
	2	c

S	C	D
	1	b
	3	d

Res	A	B	C	D
	1	a	1	b
	Null	Null	3	d

3. Full OUTER JOIN

A third operation, FULL OUTER JOIN, denoted by $\bowtie_{\leftrightarrow}$ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with NULL values as needed.

Example:

$\text{Res} \leftarrow R \bowtie_{A=C} S$

R	A	B
	1	a
	2	c

S	C	D
	1	b
	3	d

Res	A	B	C	D
	1	a	1	b
	2	c	Null	Null
	Null	Null	3	d

OUTER UNION Operations

The OUTER UNION operation was developed to take the union of tuples from two relations that have some common attributes, but are not union (type) compatible. This operation will take the UNION of tuples in two relations $R(X, Y)$ and $S(X, Z)$ that are partially compatible, meaning that only some of their attributes, say X, are union compatible.

Example:

R	A	B
	1	a
	2	c

S	A	D
	1	10
	3	d30

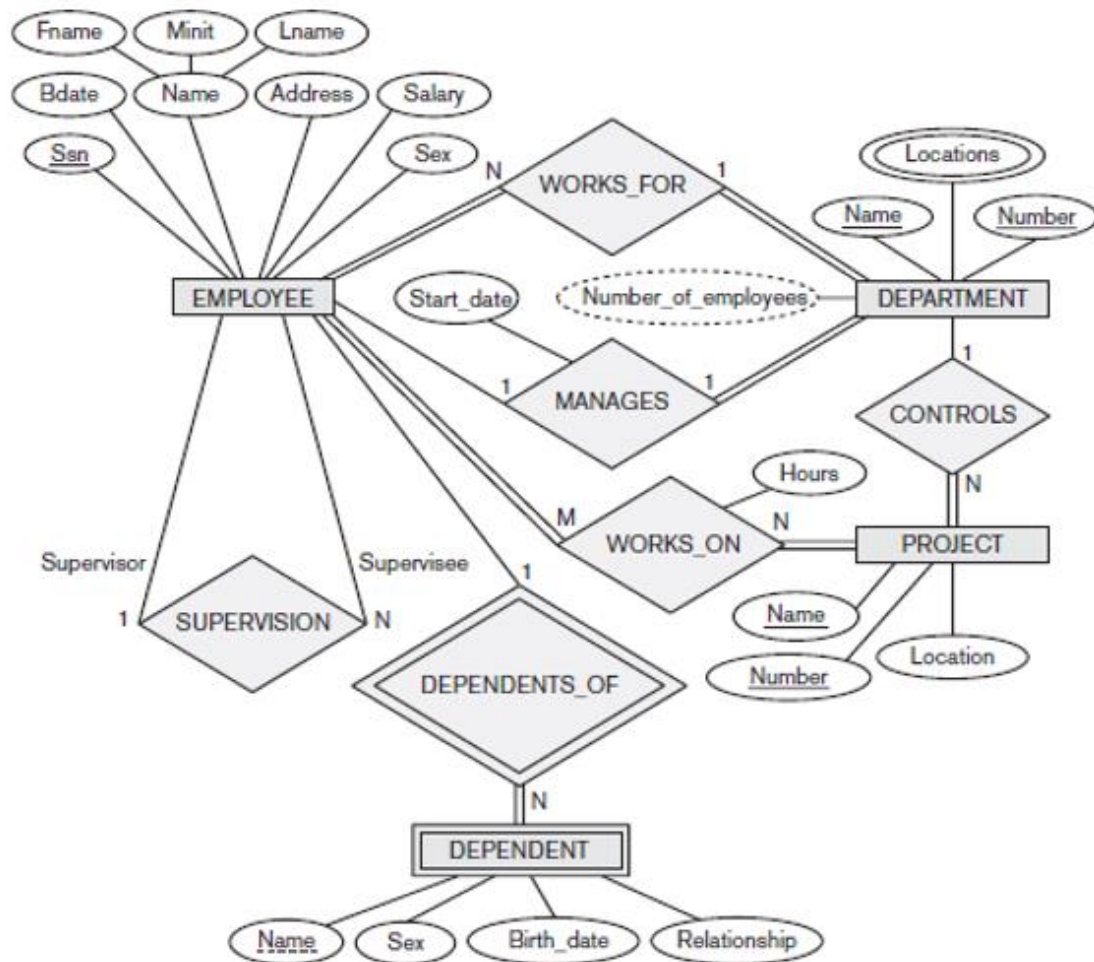
Res	A	B	D
	1	A	10
	2	C	Null
	3	Null	30

Mapping Conceptual Design into a Logical Design

Relational Database Design using ER-to-Relational mapping

Procedure to create a relational schema from an Entity-Relationship (ER)

Consider the following ER diagram of company database below,



Step 1: Mapping of Regular Entity Types

- For each regular entity type, create a relation R that includes all the simple attributes of E
- Include only the simple component attributes of a composite attribute
- Choose one of the key attributes of E as the primary key for R
- If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R.
- If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R

- In our example-COMPANY database, we create the relations EMPLOYEE, DEPARTMENT, and PROJECT
- We choose Ssn, Dnumber, and Pnumber as primary keys for the relations EMPLOYEE, DEPARTMENT, and PROJECT, respectively
- The relations that are created from the mapping of entity types are called entity relations because each tuple represents an entity instance.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary
-------	-------	-------	------------	-------	---------	-----	--------

DEPARTMENT

Dname	<u>Dnumber</u>
-------	----------------

PROJECT

Pname	<u>Pnumber</u>	Plocation
-------	----------------	-----------

Step 2: Mapping of Weak Entity Types

- For each weak entity type, create a relation R and include all simple attributes of the entity type as attributes of R
- Include primary key attribute of owner as foreign key attributes of R
- In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT
- We include the primary key Ssn of the EMPLOYEE relation—which corresponds to the owner entity type—as a foreign key attribute of DEPENDENT; we rename it as Essn
- The primary key of the DEPENDENT relation is the combination {Essn, Dependent_name}, because Dependent_name is the partial key of DEPENDENT
- It is common to choose the propagate (CASCADE) option for the referential triggered action on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity.
- This can be used for both ON UPDATE and ON DELETE.

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary
-------	-------	-------	------------	-------	---------	-----	--------

DEPARTMENT

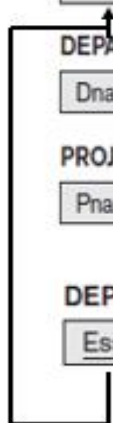
Dname	<u>Dnumber</u>
-------	----------------

PROJECT

Pname	<u>Pnumber</u>	Plocation
-------	----------------	-----------

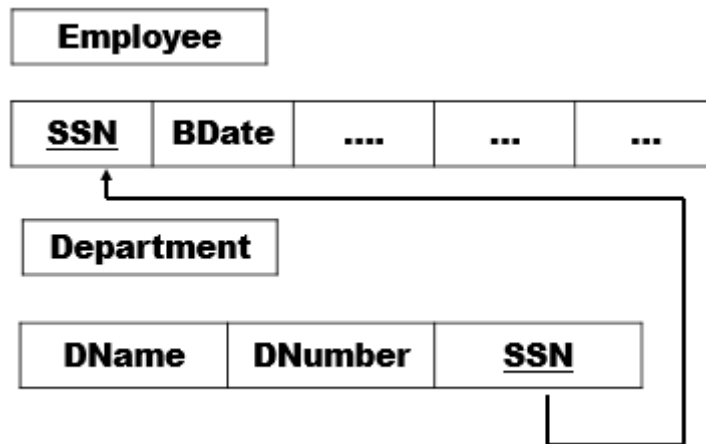
DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------



Step 3: Mapping of Binary 1:1 Relationship Types

- In ER Diagram there is 1:1 relation between Employee and Department.
- Department and managers has total participation therefore Employee manages department.
- Department should have employee SSN since employee manages department.



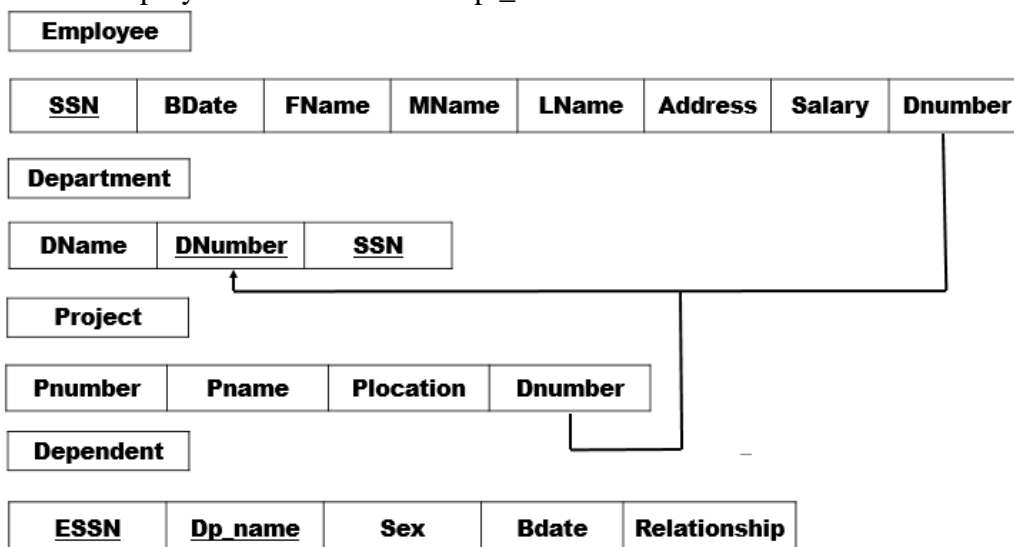
Step 4: Mapping of Binary 1: N Relationship Types

- In ER diagram N : 1 or 1 : N has 3 relations.
 - Department to Employee
 - Department to Project
 - Employee to Dependent

i.e., we will move from “1” to “N”.

Since Department: Employee

- Each department has many employees there.
- So for employee there should be Dept_no.



Step 5: Mapping of Binary M: N Relationship Types

- For each binary M:N relationship type
 - Create a new relation S
 - Include primary key of participating entity types as foreign key attributes in S
 - Include any simple attributes of M:N relationship type
- In our example, we map the M:N relationship type WORKS_ON by creating the relation WORKS_ON. We include the primary keys of the PROJECT and EMPLOYEE relations as foreign keys in WORKS_ON and rename them Pno and Essn, respectively.
- We also include an attribute Hours in WORKS_ON to represent the Hours attribute of the relationship type.
- The primary key of the WORKS_ON relation is the combination of the foreign key attributes {Essn, Pno}.

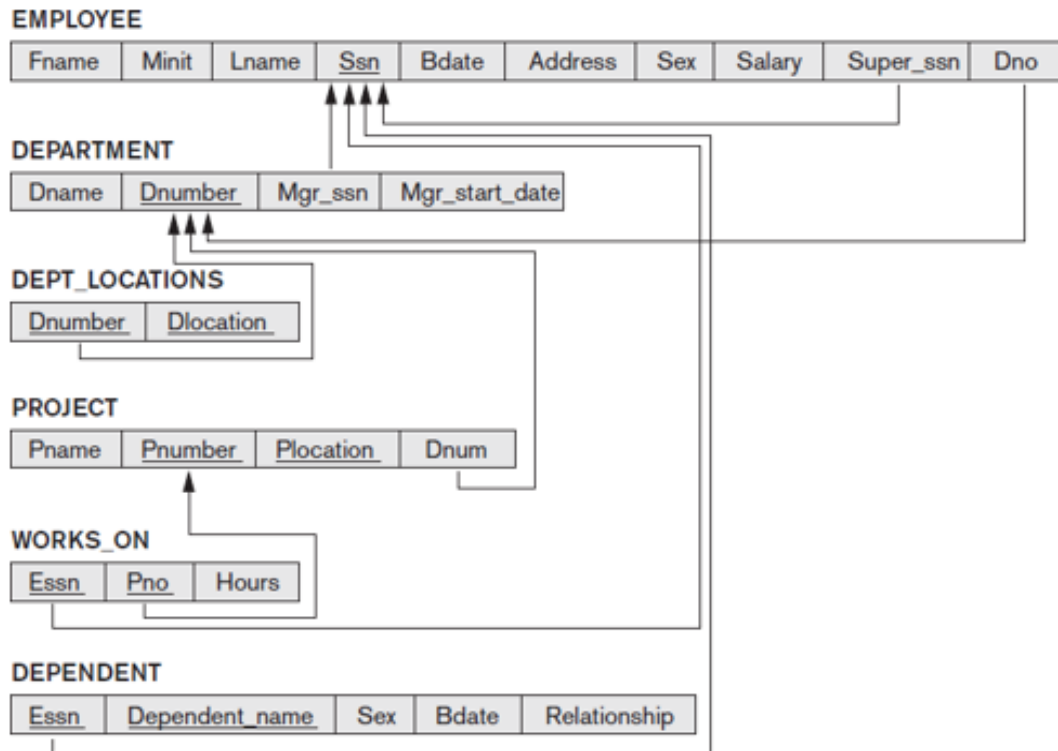
WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

- The propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the relationship R, since each relationship instance has an existence dependency on each of the entities it relates. This can be used for both ON UPDATE and ON DELETE.

Step 6: Mapping of Multivalued Attributes

- For each multivalued attribute
 - Create a new relation
 - Primary key of R is the combination of A and K
 - If the multivalued attribute is composite, include its simple components
- In our example, we create a relation DEPT_LOCATIONS
- The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while Dnumber—as foreign key—represents the primary key of the DEPARTMENT relation.
- The primary key of DEPT_LOCATIONS is the combination of {Dnumber, Dlocation}
- A separate tuple will exist in DEPT_LOCATIONS for each location that a department has
- The propagate (CASCADE) option for the referential triggered action should be specified on the foreign key in the relation R corresponding to the multivalued attribute for both ON UPDATE and ON DELETE.



Step 7: Mapping of N-ary Relationship Types

- For each n-ary relationship type R
 - Create a new relation S to represent R
 - Include primary keys of participating entity types as foreign keys
 - Include any simple attributes as attributes
- The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types.
- For example, consider the relationship type SUPPLY. This can be mapped to the relation SUPPLY whose primary key is the combination of the three foreign keys {Sname, Part_no, Proj_name}.

