Module 2

Chapter 1: The Relational Data 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.

1.1 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.

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 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 follow:

- Usa_phone_numbers: The set of ten-digit phone numbers valid in the United States.
- Social_security_numbers: 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 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 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).

Tuple

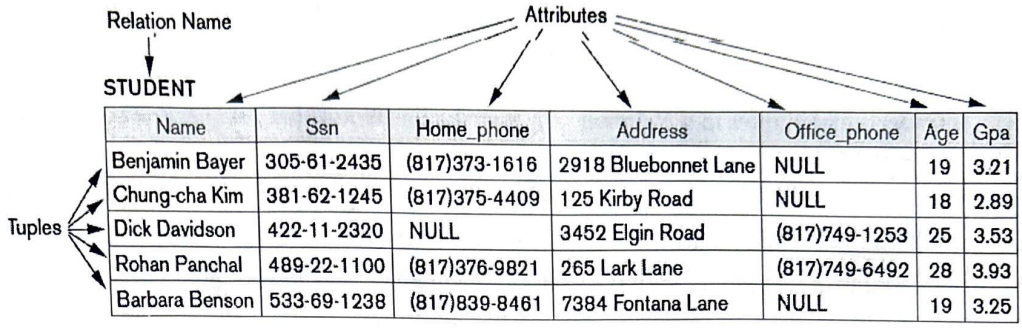
Mapping from attributes to values drawn from the respective domains of those attributes. Tuples are intended to describe some entity (or relationship between entities) in the miniworld

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 $r = \{t_1, t_2, ..., t_m\}$. Each n-tuple t is an ordered list of n values $t = <v_1, v_2, ..., v_n>$, where each value vi, $1 \leq i \leq n$, 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 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.

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.

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's 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

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.

Primary

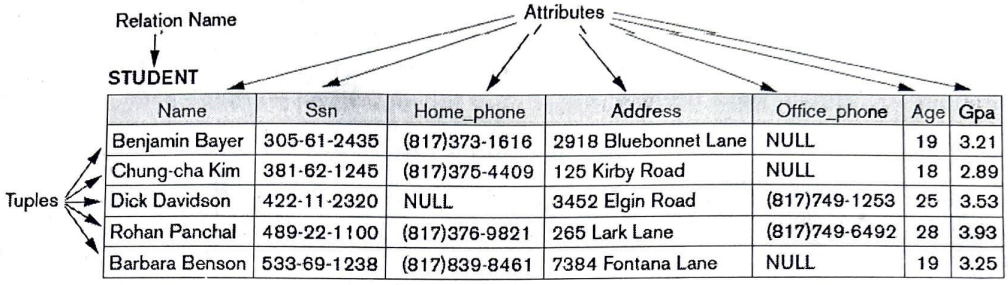
Key

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 1 hold.This property is not required by a superkey.

Example: Consider the STUDENT relation



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.** For 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

A 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}

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship

Figure1.2.3 (a): 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 1.2.3(b) : 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. For 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 realworld concepts—project names and department names.

1.2.4 Integrity, Referential Integrity, and Foreign Keys

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.

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.

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.

1.2.5 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 → 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

1.3 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:

1. Insert - used to insert one or more new tuples in a relation
2. Delete- used to delete tuples
3. 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.

1.3.1 The Insert Operation

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

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

Examples:

1. Operation:

   Insert <'Cecilia', 'F', 'Kolonsky', NULL, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4>

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

2. Operation:

   Insert <'Alicia', 'J', 'Zelaya', '999887777', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, '987654321', 4>

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

3.Operation:

   Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windswept, Katy, TX', F, 28000, '987654321', 7>

Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding referenced tuple exists in DEPARTMENT with Dnumber = 7.

4. Operation:

Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windy Lane,Katy, TX', F, 28000, NULL, 4>

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 an attempt to correct the reason for rejecting the insertion

1.3.2 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.

Examples:

1. Operation:

   Delete the WORKS_ON tuple with Essn = '999887777' and Pno =10.

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

2. Operation:

   Delete the EMPLOYEE tuple with Ssn = '999887777'.

   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.

3. Operation:

   Delete the EMPLOYEE tuple with Ssn = '333445555'

   Result: This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

Several options are available if a deletion operation causes a violation

1. restrict - is to reject the deletion
2. cascade, is to attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted

3. 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.

1.3.3 The Update 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.

Examples:

1. Operation:

   Update the salary of the EMPLOYEE tuple with Ssn = '999887777' to 28000.

   Result: Acceptable.

2. Operation:

   Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 7.

   Result: Unacceptable, because it violates referential integrity.

3. Operation:

   Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to '987654321'.

   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 Ssn

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.

1.3.4 The Transaction Concept

A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema A single transaction may involve any number of retrieval operations and any number of update operations. These retrievals and updates will together form an atomic unit of work against the database.For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.