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.

Relation Name → STUDENT

Attributes

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

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, $1 \le i \le n$, is an element of dom (A_i) or is a special NULL value. The ith 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 *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

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

Relation Name → STUDENT

Attributes

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.** 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 s*et 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.

→

relational designs and to "normalize"

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.

Chapter 2: Relational Algebra

2.1 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

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

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

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 <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*,

<comparison op> is one of the operators {=, <, ≤, >, ≥, ≠}, 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 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,

$|\sigma_c (R)| \leq |R|$ 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,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<cond\,n>}(R)) \ldots)) = \sigma_{<cond1>} \text{ AND}_{<cond2>} \text{ AND } \ldots \text{ AND}_{<cond\,n>}(R)$$

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

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

would to the following SQL query:

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

2.2.2 The 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

$$\pi_{<attribute\ list>}(R)$$

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>

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 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

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

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

2.2.3 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:

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

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

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

$$R(\text{First_name, Last_name, Salary}) \leftarrow \pi_{\text{Fname, Lname, Salary}}(\text{TEMP})$$

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. $\rho_{S(B1, B2, ..., Bn)}(R)$ — 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 (A_1, A_2, ..., A_n) in that order, then each A_i is renamed as B_i.

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,
```

2.3 Relational Algebra Operations from Set Theory

2.3.1 The UNION, INTERSECTION, and MINUS Operations

- **UNION:** 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.
- **INTERSECTION:** The result of this operation, denoted by $R \cap S$, 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

$$\text{DEP5_EMPS} \leftarrow \sigma_{Dno=5}(\text{EMPLOYEE})$$

$$\text{RESULT1} \leftarrow \pi_{Ssn}(\text{DEP5_EMPS})$$

$$\text{RESULT2(Ssn)} \leftarrow \pi_{Super_ssn}(\text{DEP5_EMPS})$$

$$\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$$

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:

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

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 $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$ are said to be union compatible (or type compatible) if they have the same degree n and if $dom(A_i) = dom(B_i)$ 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,

$$R - S \neq S - R$$

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

2.3.2 The CARTESIAN PRODUCT (CROSS PRODUCT) Operation

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 $R(A_1, A_2, ..., A_n) \times S(B_1, B_2, ..., B_m)$ is a relation Q with degree $n + m$ attributes $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$, 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 n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then $R \times S$ will have $n_R * n_S$ tuples

Example: suppose that we want to retrieve a list of names of each female 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

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	...

EMPNAMES

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

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

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

2.4 Binary Relational Operations: JOIN and DIVISION

2.4.1 The JOIN Operation

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 $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_m)$ is

$$R \bowtie_{<join\ conditio>} S$$

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

$$\text{DEPT_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr_ssn=Ssn}} \text{EMPLOYEE}$$
$$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT_MGR})$$

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(A_1, A_2, ..., A_n,B_1, B_2, ..., B_m 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 A_i θ B_j, A_i is an attribute of R, B is an attribute of S, Ai and B_j have the same domain, and θ (theta) is one of the comparison operators {=, <, ≤, >, ≥, ≠}.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.

2.4.2 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:

$$\textbf{PROJ_DEPT} \leftarrow \textbf{PROJECT} * \rho_{(\text{Dname, Dnum, Mgr_ssn, Mgr_start_date})}(\textbf{DEPARTMENT})$$

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

$$\textbf{DEPT} \leftarrow \rho_{(\text{Dname, Dnum, Mgr_ssn,Mgr_start_date})}(\textbf{DEPARTMENT})$$

PROJ_DEPT ← PROJECT * 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

DEPT_LOCS ← DEPARTMENT * 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

$$Q \leftarrow R *_{(<list1>),(<list2>)} S$$

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 n_R tuples and S has n_S tuples, the result of a JOIN operation R $\bowtie_{<\text{join condition}>}$ S will have between zero and $n_R * n_S$ tuples. The expected size of the join result divided by the maximum size $n_R * n_S$ 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

2.4.3 A Complete Set of Relational Algebra Operations

The set of relational algebra operations {σ, π, ∪, ρ, –, ×} 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.

2.4.4 The DIVISION Operation

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:

$$\text{SSN_PNOS} \leftarrow \pi_{\text{Essn, Pno}}(\text{WORKS_ON})$$

- 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:

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

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

$$\text{SSNS(Ssn)} \leftarrow \text{SSN_PNOS} \div \text{SMITH_PNOS}$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname}}(\text{SSNS} * \text{EMPLOYEE})$$

(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 t_R appear in R with t_R [Y] = t, and with t_R [X] = t_S for every tuple t_S 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) *b*1 and *b*4 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: *b*2 does not appear with *a*2, and *b*3 does not appear with *a*1.

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

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

Table 6.1 Operations of Relational Algebra

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

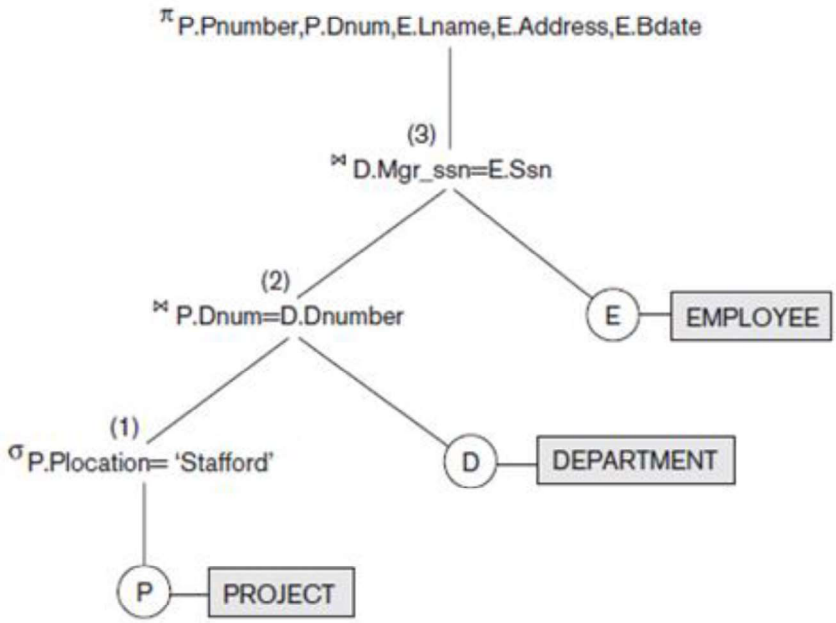
2.4.5 Notation for Query Trees

Query tree (query evaluation tree or query execution tree) is used in relational systems to represent queries internally. A query tree is a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes.

An execution of the query tree consists of executing an internal node operation whenever its operands represented by its child nodes are available, and then replacing that internal node by the relation that results from executing the operation. The execution terminates when the root node is executed and produces the result relation for the query.

Example: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

$$\pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(((\sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})) \bowtie_{\text{Dnum=Dnumber}}(\text{DEPARTMENT})) \bowtie_{\text{Mgr_ssn=Ssn}}(\text{EMPLOYEE}))$$



Leaf nodes P, D, and E represent the three relations PROJECT, DEPARTMENT, and EMPLOYEE. The relational algebra operations in the expression are represented by internal tree nodes. The query tree signifies an explicit order of execution in the following sense. The node marked (1) must begin execution before node (2) because some resulting tuples of operation (1) must be available before we can begin to execute operation (2). Similarly, node (2) must begin to execute and produce results before node (3) can start execution, and so on.

A query tree gives a good visual representation and understanding of the query in terms of the relational operations it uses and is recommended as an additional means for expressing queries in relational algebra.

2.5 Additional Relational Operations

2.5.1 Generalized Projection

The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list. The generalized form can be expressed as:

$$\pi_{F1, F2, ..., Fn}(R)$$

where F_1, F_2, ..., F_n are functions over the attributes in relation R and may involve arithmetic operations and constant values.

The generalized projection helpful when developing reports where computed values have to be produced in the columns of a query result. For example,consider the relation EMPLOYEE (Ssn, Salary,Deduction, Years_service). A report may be required to show

Net Salary = Salary – Deduction,

Bonus = 2000 * Years_service, and

Tax = 0.25 * Salary.

generalized projection combined with renaming :

$$\text{REPORT} \leftarrow \rho_{(Ssn,\ Net_salary,\ Bonus,\ Tax)}(\pi_{Ssn,\ Salary - Deduction,\ 2000 * Years_service,\ 0.25 * Salary}(\text{EMPLOYEE})).$$

2.5.2 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 $\Im$ (script F) :

$$\text{<grouping attributes>}\ \Im\ \text{<function list>}\ (R)$$

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: To retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes

$$\rho_{R(\text{Dno, No_of_employees, Average_sal})}(\text{Dno}\ \Im\ \text{COUNT Ssn, AVERAGE Salary}\ (\text{EMPLOYEE}))$$

The aggregate function operation.

a. $\rho_{R(\text{Dno, No_of_employees, Average_sal})}(\text{Dno}\ \Im\ \text{COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE}))$.

b. $\text{Dno}\ \Im\ \text{COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$.

c. $\Im\ \text{COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

2.5.3 Recursive Closure Operations

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 : Retrieve all supervisees of an employee *e* at all levels—that is, all employees *e*' directly supervised by *e*, all employees *e*'$\Im$ directly supervised by each employee *e*', all employees *e*''' directly supervised by each employee *e*'' and so on.

$$\text{BORG_SSN} \leftarrow \pi_{\text{Ssn}}(\sigma_{\text{Fname='James' AND Lname='Borg'}}(\text{EMPLOYEE}))$$
$$\text{SUPERVISION(Ssn1, Ssn2)} \leftarrow \pi_{\text{Ssn,Super_ssn}}(\text{EMPLOYEE})$$
$$\text{RESULT1(Ssn)} \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}} \text{BORG_SSN})$$

SUPERVISION

(Borg's Ssn is 888665555)

(Ssn)	(Super_ssn)
Ssn1	Ssn2
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

To retrieve all employees supervised by Borg at level 2—that is, all employees e'' supervised by some employee e' who is directly supervised by Borg—we can apply another JOIN to the result of the first query, as follows:

$$\text{RESULT2(Ssn)} \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}} \text{RESULT1})$$

RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by Borg's subordinates)

To get both sets of employees supervised at levels 1 and 2 by 'James Borg', we can apply the UNION operation to the two results, as follows:

$$\text{RESULT} \leftarrow \text{RESULT2} \cup \text{RESULT1}$$

2.5.4 OUTER JOIN Operations

The JOIN operations match tuples that satisfy the join condition. For example, for a NATURAL JOIN operation R * S, only tuples from R that have matching tuples in S—and vice versa—appear in the result. Hence, tuples without a matching (or related) tuple are eliminated from the

JOIN result. Tuples with NULL values in the join attributes are also eliminated.This type of join, where tuples with no match are eliminated, is known as an inner join.

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.

For example, 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 ⟕ to retrieve the result as follows:

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} ⟕_{\text{Ssn=Mgr_ssn}} \text{DEPARTMENT})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

The LEFT OUTER JOIN operation keeps every tuple in the first, or left, relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or padded with NULL values.

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

A similar operation, RIGHT OUTER JOIN, denoted by ⟖, keeps every tuple in the *second*, or right, relation *S* in the result of *R* ⟖ *S*.

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

2.5.5 The OUTER UNION Operation

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.

The attributes that are union compatible are represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation T(X, Y, Z). Two tuples t_1 in R and t_2 in S are said to match if $t_1[X]= t_2[X]$. These will be combined (unioned) into a single tuple in t. Tuples in either relation that have no matching tuple in the other relation are padded with NULL values.

For example, an OUTER UNION can be applied to two relations whose schemas are:

STUDENT(Name, Ssn, Department, Advisor)

INSTRUCTOR(Name, Ssn, Department, Rank)

Tuples from the two relations are matched based on having the same combination of values of the shared attributes—Name, Ssn, Department. All the tuples from both relations are included in the result, but tuples with the same (Name, Ssn, Department) combination will appear only once in the result. Tuples appearing only in STUDENT will have a NULL for the Rank attribute, whereas tuples appearing only in INSTRUCTOR will have a NULL for the Advisor attribute.

A tuple that exists in both relations, which represent a student who is also an instructor, will have values for all its attributes The resulting relation, STUDENT_OR_INSTRUCTOR, will have the following attributes:

STUDENT_OR_INSTRUCTOR(Name, Ssn, Department, Advisor, Rank)

2.6 Examples of Queries in Relational Algebra

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

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

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

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

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

$$\text{STAFFORD_PROJS} \leftarrow \sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})$$
$$\text{CONTR_DEPTS} \leftarrow (\text{STAFFORD_PROJS} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT})$$
$$\text{PROJ_DEPT_MGRS} \leftarrow (\text{CONTR_DEPTS} \bowtie_{\text{Mgr_ssn=Ssn}} \text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(\text{PROJ_DEPT_MGRS})$$

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

$$\text{DEPT5_PROJS} \leftarrow \rho_{(Pno)}(\pi_{Pnumber}(\sigma_{Dnum=5}(\text{PROJECT})))$$
$$\text{EMP_PROJ} \leftarrow \rho_{(Ssn,\ Pno)}(\pi_{Essn,\ Pno}(\text{WORKS_ON}))$$
$$\text{RESULT_EMP_SSNS} \leftarrow \text{EMP_PROJ} \div \text{DEPT5_PROJS}$$
$$\text{RESULT} \leftarrow \pi_{Lname,\ Fname}(\text{RESULT_EMP_SSNS} * \text{EMPLOYEE})$$

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 of the department that controls the project.

$$\text{SMITHS(Essn)} \leftarrow \pi_{Ssn}(\sigma_{Lname='Smith'}(\text{EMPLOYEE}))$$
$$\text{SMITH_WORKER_PROJS} \leftarrow \pi_{Pno}(\text{WORKS_ON} * \text{SMITHS})$$
$$\text{MGRS} \leftarrow \pi_{Lname,\ Dnumber}(\text{EMPLOYEE} \bowtie_{Ssn=Mgr_ssn} \text{DEPARTMENT})$$
$$\text{SMITH_MANAGED_DEPTS(Dnum)} \leftarrow \pi_{Dnumber}(\sigma_{Lname='Smith'}(\text{MGRS}))$$
$$\text{SMITH_MGR_PROJS(Pno)} \leftarrow \pi_{Pnumber}(\text{SMITH_MANAGED_DEPTS} * \text{PROJECT})$$
$$\text{RESULT} \leftarrow (\text{SMITH_WORKER_PROJS} \cup \text{SMITH_MGR_PROJS})$$

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

$$T1(\text{Ssn, No_of_dependents}) \leftarrow {}_{Essn}\Im_{\text{COUNT Dependent_name}}(\text{DEPENDENT})$$
$$T2 \leftarrow \sigma_{No_of_dependents>2}(T1)$$
$$\text{RESULT} \leftarrow \pi_{Lname,\ Fname}(T2 * \text{EMPLOYEE})$$

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

$$\text{ALL_EMPS} \leftarrow \pi_{Ssn}(\text{EMPLOYEE})$$
$$\text{EMPS_WITH_DEPS(Ssn)} \leftarrow \pi_{Essn}(\text{DEPENDENT})$$
$$\text{EMPS_WITHOUT_DEPS} \leftarrow (\text{ALL_EMPS} - \text{EMPS_WITH_DEPS})$$
$$\text{RESULT} \leftarrow \pi_{Lname,\ Fname}(\text{EMPS_WITHOUT_DEPS} * \text{EMPLOYEE})$$

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

$$\text{MGRS(Ssn)} \leftarrow \pi_{Mgr_ssn}(\text{DEPARTMENT})$$
$$\text{EMPS_WITH_DEPS(Ssn)} \leftarrow \pi_{Essn}(\text{DEPENDENT})$$
$$\text{MGRS_WITH_DEPS} \leftarrow (\text{MGRS} \cap \text{EMPS_WITH_DEPS})$$
$$\text{RESULT} \leftarrow \pi_{Lname,\ Fname}(\text{MGRS_WITH_DEPS} * \text{EMPLOYEE})$$

Chapter 3: Mapping Conceptual Design into a Logical Design

3.1 Relational Database Design using ER-to-Relational mapping

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

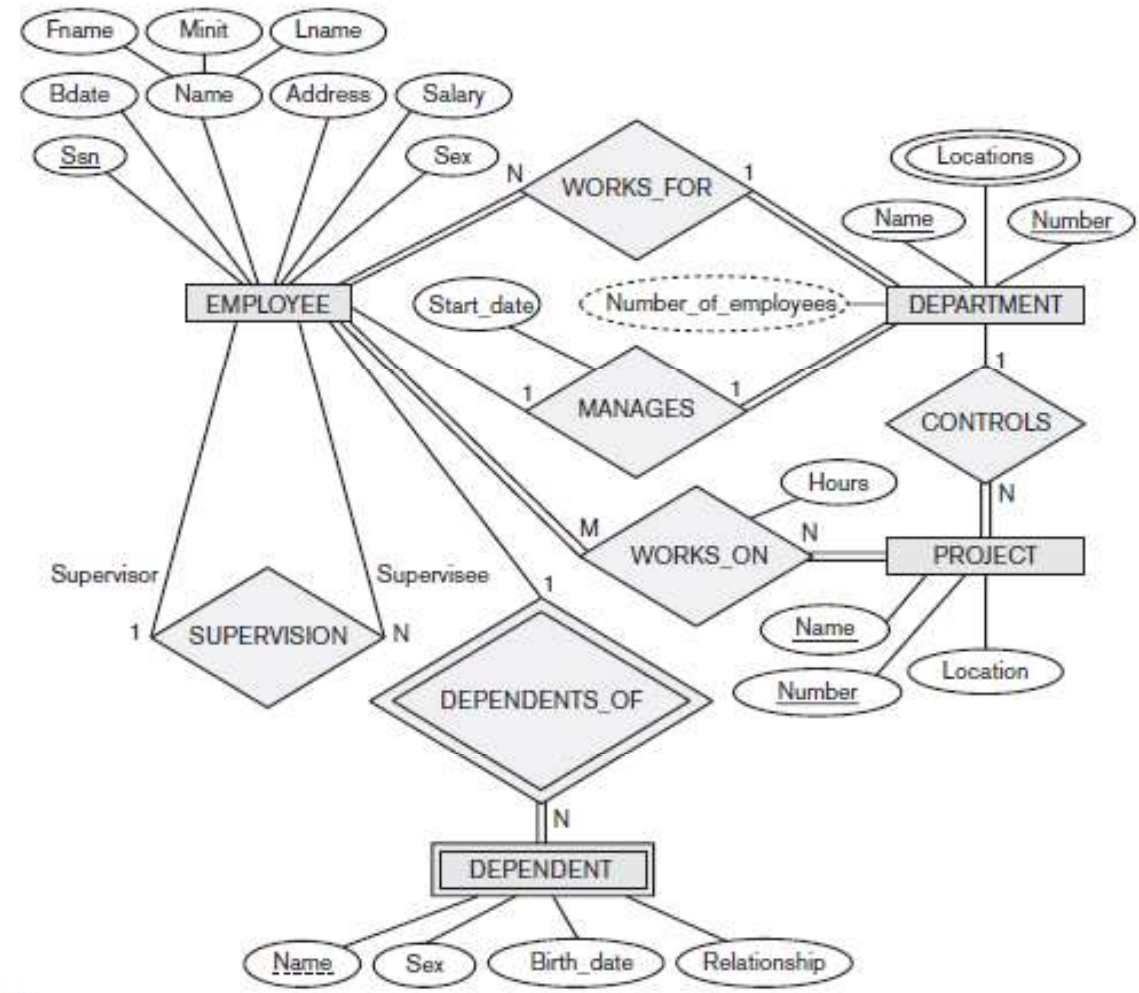


Fig 3.1: ER diagram of company database

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	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

PROJECT

Pname	Pnumber	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}, becau partial key of DEPENDENT
- It is co o choose the prop (CASCADE) option e referential triggered n on the f key in the relation c nding to the ntity type, since a weak enti has an e e dependency on its entity.
- This ca ed for both ON U and ON D

ENDENT

sn	Dependent	Sex		Relationship

Step 3: Mapping of Binary 1:1 Relationship Types

- For each binary 1:1 relationship type *R* in the ER schema, identify the relations *S* and *T* that correspond to the entity types participating in *R*
- There are three possible approaches:
  - foreign key approach
  - merged relationship approach
  - crossreference or relationship relation approach

1. The foreign key approach

- Choose one of the relations—*S*, say—and include as a foreign key in *S* the primary key of *T*.
- It is better to choose an entity type with *total participation* in *R* in the role of *S*
- Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type *R* as attributes of *S*.
- In our example, we map the 1:1 relationship type by choosing the participating entity type DEPARTMENT to serve in the role of *S* because its participation in the MANAGES relationship type is total
- We include the primary key of the EMPLOYEE relation as foreign key in the DEPARTMENT relation and rename it Mgr_ssn.
- We also include the simple attribute Start_date of the MANAGES relationship type in the DEPARTMENT relation and rename it Mgr_start_date

2. Merged relation approach:

- merge the two entity types and the relationship into a single relation
- This is possible when *both participations are total,* as this would indicate that the two tables will have the exact same number of tuples at all times.

3.Cross-reference or relationship relation approach:

- set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.
- [illegible] ionships
- The [illegible] R is called a rela[illegible] relation (or somet[illegible] lookup table), becaus[illegible]h tuple [illegible] presents a relationsh[illegible]nce that relat[illegible] tuple from S with one tuple from [illegible]
- The re[illegible] R will include the [illegible]y key attrib[illegible]S and T as foreign keys to S and T.
- The p[illegible]e of the two fore[illegible], and the other foreign key will be a uniq[illegible] of R.

- The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type.
- Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*
- Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of *S*
- In our example, we now map the 1:N relationship types WORKS_FOR, CONTROLS, and SUPERVISION
- For WORKS_FOR we include the primary key Dnumber of the DEPARTMENT relation as foreign key in the EMPLOYEE relation and call it Dno.
- For SUPERVISION we include the primary key of the EMPLOYEE relation as foreign key in the EMPLOYEE relation itself—because the relationship is recursive—and call it Super_ssn.
- The CONTROLS relationship is mapped to the foreign key attribute Dnum of PROJECT, which references the primary key Dnumber of the DEPARTMENT relation.

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}.

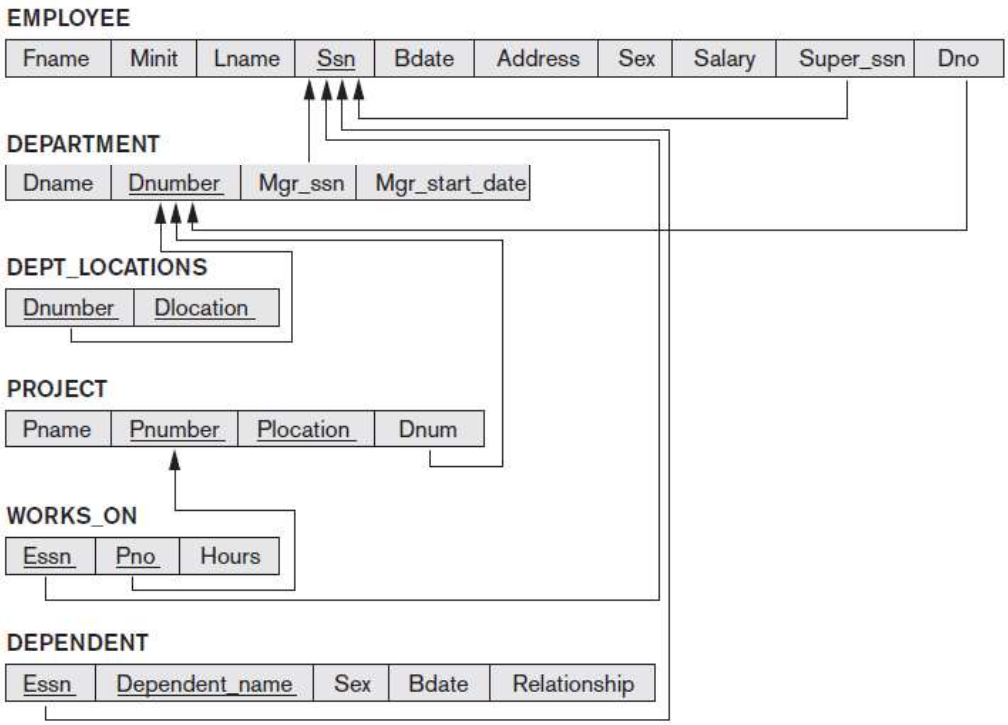
WORKS_ON

Essn	Pno	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}.

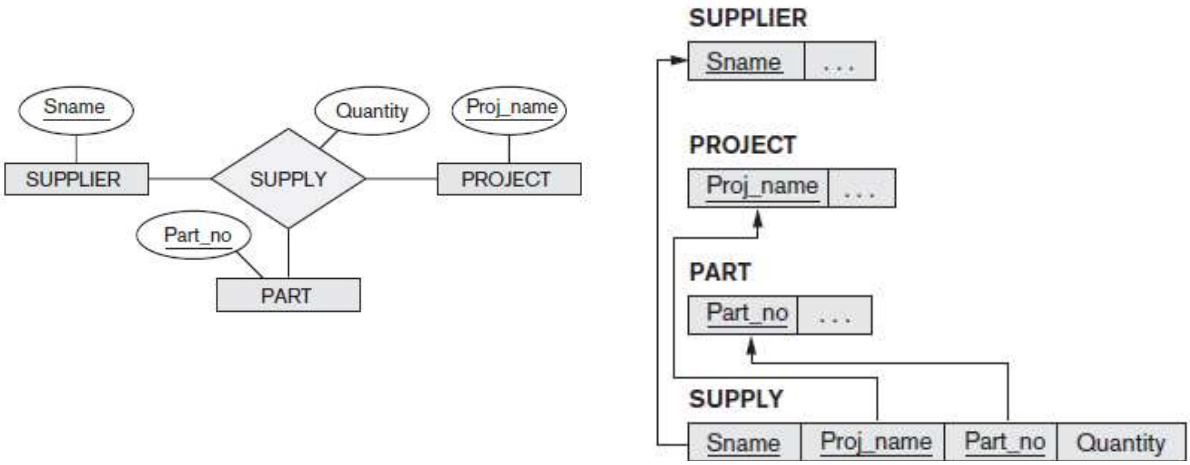


Figure 3.2: Mapping the *n*-ary relationship type SUPPLY