Module 2: The Relational Data Model

Relational Model Concepts

The principles of the relational model were first outlined by Dr. E.F. Codd in 1970 in a classic paper called "A relational Model of Data for Large Shared Data Banks". In this paper, Dr. Codd proposed the relational model for the database.

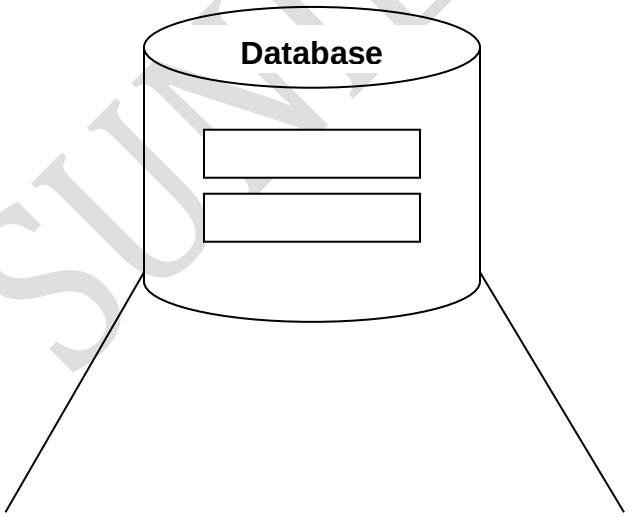
The more popular models used at the time were hierarchical and network, or even simple flat file data structures. Relational database management systems soon became very popular especially for their ease of use and flexibility in structure. In addition, a number of innovative vendors, such as Oracle, supplemented the RDBMS with a suite of powerful application development and user products, providing a total solution.

Components of the Relational Model

- Collections of objects or relations that store the data
- A set of operators that can act on the relations to produce other relations
- Data integrity for accuracy and consistency

Definition of a Relational Database

A relational database is a collection of relations or two-dimensional tables.



EMP

EMPNO	ENAME	DEPTNO
7839	King	10

DEPT

DEPTNO	DNAME	LOC
10	Accounting	New York
20	Research	Dallas

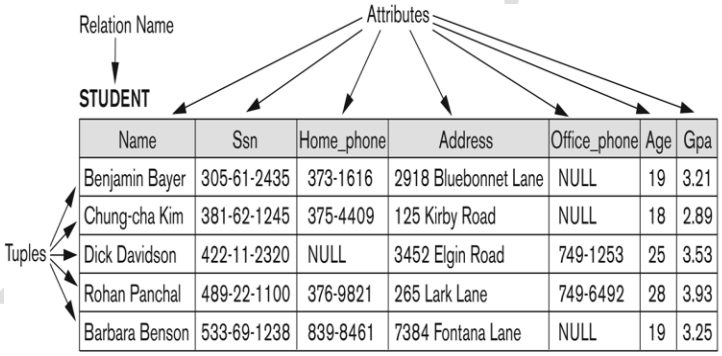
A relational database is a collection of relations or two-dimensional tables to store information.

For Example, you might want to store information about all the employees in your company. In a relational database, you create several tables to store different pieces of information about your employees, such as an employee table, a department table and a salary table

Informal Definitions

- Informally, a relation looks like a table of values.
- A relation typically contains a set of rows.
- The data elements in each row represent certain facts that correspond to a real-world entity or relationship.
    - In the formal model, rows are called tuples
- Each column has a column header that gives an indication of the meaning of the data items in that column.
    - In the formal model, the column header is called an attribute name (or just attribute).

Example of a Relation



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

Figure 5.1
The attributes and tuples of a relation STUDENT.

- The data type describing the types of values that can appear in each columns is called a **domain**.

- Key of a Relation:
    - Each row has a value of a data item (or set of items) that uniquely identifies that row in the table called the **key**
      Eg:- In the STUDENT table, SSN is the key.

    - Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table called *artificial key* or *surrogate key*

Formal Definitions – Schema

- The **Schema** (or description) of a Relation:
  - Denoted by R(A_1, A_2, ...A_n)
  - R is the **name** of the relation
  - The **attributes** of the relation are A_1, A_2, ...A_n
  - The degree of the relation is n (the number of its attributes).
- Example:

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)

  - CUSTOMER is the relation name.
  - Defined over the four attributes:Cust-id, Cust-name, Address, Phone#

- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.
- The **Schema** (or description) of a Relation:
  - Denoted by R(A_1, A_2, ...A_n)
  - R is the **name** of the relation
  - The **attributes** of the relation are A_1, A_2, ...A_n
  - The degree of the relation is n (the number of its attributes).
- Example:

  CUSTOMER (Cust-id, Cust-name, Address, Phone#)

  - CUSTOMER is the relation name.
  - Defined over the four attributes:Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.

- A **tuple** is an ordered set of values (enclosed in angled brackets '< … >').
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  - This is called a 4-tuple as it has 4 values.
  - A tuple (row) in the CUSTOMER relation.
- A relation is a **set** of such tuples (rows).
- A **domain** has a logical definition:
  - Example: "University_Seat_Number" is the set of 10 characters valid in VTU university.
  - "Academic_Dept_Code" is a set of academic department codes such as 'CS', 'IS', 'EC', etc,.
- A domain also has a data-type or a format defined for it.
  - Dates have various formats such as year, month, day formatted as yyyy-mm-dd, or as dd-mmm-yyyy etc.
- The attribute name designates the role played by a domain in a relation:
  - Example: The domain Date may be used to define two attributes named "Invoice-date" and "Payment-date" with different meanings.

Characteristics of Relations

- **Ordering of tuples in a relation r(R):**
  - Tuples ordering is not part of a relation definition because it is defined as a set of tuples.
    - Many logical orders can be specified on the relation.
    - There is no preference for one logical ordering over another.
  - When a relation is implemented as a file, a physical ordering may be specified on the records of the file.
- **Ordering of values within each tuple:**
  - The tuple is an ordered list, so the ordering of values in a tuple is important.
  - At a logical level, the order is not important as long as the correspondence between the attribute and its value is mentioned.

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

STUDENT

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

- **Values and NULLs in the tuple:**
  - All values are considered atomic (indivisible).
  - Each value in a tuple must be from the domain of the attribute for that column.
    - If tuple t = $<v_1, v_2, \ldots, v_n>$ is a tuple (row) in the relation state r of $R(A_1, A_2, \ldots, A_n)$
    - Then each v_i must be a value from $dom(A_i)$
  - A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.
- **Interpretation (Meaning) of a Relation:**

- The relational schema can be interpreted as a declaration or type of assertion.

- For example, schema of the STUDENT relation interprets that, a student entity has Name,

USN, Home_phone, Address, Office_phone, Age, Gpa.

- Alternative interpretation of a relation schema is as a **predicate,** that values in each tuple are interpreted as values that satisfy the predicate.

Relational Model Notation

- $R(A_1, A_2, ..., A_n)$ denotes a relation schema R of degree n (n attributes).
- r(R) is a relation state of relation schema R.
- $t[A_i]$ and $t.A_i$ refer to the value v_i in t for attribute A_i.
- The letters Q, R, S denote relation names.
- The letters q, r, s denote relation states.
- The letters t, u, v denote tuples.
- The dot notation R.A can be used to identify the attributes (e.g. Student.Name or Student.Age).

Relational model constraints

- Constraints on databases can generally be divided into three main categories:

- Inherent model-based or Implicit constraints:- Constraints that are inherent in the data model.

- Schema-based or explicit constraints:- Constraints that can be directly expressed in schema of the data model using DDL.

- Application-based or semantic constraints or business rules:- Constraints expressed and enforced using application programs.

Schema-based constraints

- Schema based constraints in the relational model are:
  - **Domain** constraints
  - **Key** constraints and constraints on NULL values
  - **Entity** integrity constraints
  - **Referential** integrity constraints
- **Domain constraint**
  - Every value in a tuple must be from the *domain of its attribute* and must be atomic (or it could be **null**, if allowed for that attribute).
- **Key Constraints**
  - As a relation is defined as a set of tuple, thus no two tuples can have the same combination of values for all their attributes.
  - Usually, there are other subset of attributes (superkey SK) with this constraint: $t_i[SK] \neq t_j[SK] \quad \forall\ i, j$
  - It is called uniqueness constraint that no two distinct tuples in r can have the same values for SK.
  - A superkey can have redundant attributes.
  - A key K of 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.
  - A key satisfies two constraints:
  - - Two distinct tuples in any state of the relation cannot have identical values for (all) attributes in the key.

- - It is a ***minimal superkey*** – that is , a superkey from which we cannot remove any attributes and still have the uniqueness property in condition 1 hold.
- Key is determined from the meaning of the attributes, and the property is ***time-invariant***.
- It must continue to hold when we insert new tuples in the relation
- Example:
- The attribute {SSN} in the Student relation is a key.
- Any set includes {SSN} is a superkey of the relation student. Eg:- {SSN, Name, Age}
- In general:
- Any *key* is a *superkey* (but not vice versa).
- Any set of attributes that *includes a key* is a *superkey.*
- A *minimal* superkey is also a key.
- A relation schema may have more than one key.
- Such keys are called the candidate keys.
- The primary key is one of the candidate keys which is selected to identify tuples in the relation.
- The attributes that form the primary key are underlined in the schema.
- The primary key value is used to *uniquely identify* each tuple in a relation.

- **Null Constraint:**
  - Not Null constraint specifies that an attribute must have a valid value (e.g. Student name).

Relational Database Schema

- A relational database schema S is a set of relation schemas that belong to the same database.
  - S is the name of the whole **database schema.**
  - S = {R_1, R_2, …, R_m} and a set of integrity constraints IC.
  - R_1, R_2, …, R_m are the names of the individual **relation schemas** within the database S.
- The integrity constraints are specified on a DB schema and are expected to hold on every DB state.
- Example: Company Database

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	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	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Entity Integrity Constraints

- The primary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R).
  - This is because primary key values are used to *identify* the individual tuples.
  - t[PK] ≠ null for any tuple t in r(R).
  - If PK has several attributes, null is not allowed in any of these attributes.
- Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity Constraints

- A constraint involving two relations
  - The previous constraints involve a single relation.
- Used to specify a relationship among tuples in two relations:
  - The referencing relation and the referenced relation.
- Tuples in the referencing relation R_1 have attributes FK (called foreign key attributes) that reference the primary key attributes PK of the referenced relation R_2.
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R_1.FK to R_2.PK.
- Referential integrity constraint is based on foreign key (FK) concept
- 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 two rules:
  - The attributes in FK have the same domain(s) as the primary key attribute PK of R_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 $r_2(R_2)$ ($t_1[FK]=t_2[PK]$) or is null.
- A foreign key can refer to its own relation.
- Referential integrity constraints typically arise from the *relationship among the entities*.

Other Types of Constraints

- **State constraints**

  - define the constraints that a valid state of the database must satisfy.

  Ex: Domain constraints, Key Constraints, Entity Integrity constraints, Referential Integrity constraints

Transition constraints

- defined to deal with state changes in the database.

Ex: the salary of an employee can only increase. Such constraints typically enforced by the application programs or specified using active rules and triggers.

Update Operations on Relations

- The basic update operations of the relational model are:
  - Insert
  - Delete
  - Update (or Modify)
- All integrity constraints specified on the database schema should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may propagate to cause other updates automatically.
  - This may be necessary to maintain integrity constraints.
- The basic update operations of the relational model are:
  - Insert
  - Delete
  - Update (or Modify)
- All integrity constraints specified on the database schema should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may propagate to cause other updates automatically.
  - This may be necessary to maintain integrity constraints.

The Insert Operation

- INSERT may violate any of the four types of constraints:
  - Domain constraint:
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain.
  - Key constraint:
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation.
  - Referential integrity:
    - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation.
  - Entity integrity:

- if the primary key value is null in the new tuple.

- Example:-
- Operation

Insert <'Vinod', 'S', 'Joseph', NULL, '1986-04-05', '#123, Bangalore', F,

28000, NULL, 4> into EMPLOYEE

Result: Violates Entity IC , so it is rejected.

- Operation

Insert <'Alice', 'J', 'Zelaya', '999887777', '1986-04-05', '#334, Bangalore', F,

28000, '98764321', 4> into EMPLOYEE

Result: Violates Key IC , so it is rejected.

- Operation

Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1986-04-05', '#454, Bangalore',

F, 28000, '99876436', 7> into EMPLOYEE

Result: Violates Referential IC , so it is rejected

The Delete Operation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database.
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL .
      - RESTRICT option: reject the deletion.
      - CASCADE option: propagate deletion by deleting tuples that reference the tuple that is deleted.
      - SET NULL option: set the foreign keys of the referencing tuples to NULL.
  - One of the above options must be specified during database design for each foreign key constraint.
  - Example:
    - **Operation**

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

      Result: Acceptable. Deletes exactly one tuple

    - **Operation**

      Delete the EMPLOYEE tuple with SSN= '999887777'.

      Result: Not Acceptable. Violates Referential IC.

- **Operation**
  Delete the EMPLOYEE tuple with SSN= '333445555'.

  Result: Not Acceptable. Violates Referential IC.

The Update Operation

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified.
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT.
    - Need to specify similar options to DELETE.
  - Updating a foreign key (FK):
    - May violate referential integrity.
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints.

Operation

- Update the salary of an EMPLOYEE tuple with SSN ='999887777' to 28000.

Result: Acceptable

Operation

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

Result: Violates Referential IC

- Operation
  Update the SSN of the EMPLOYEE tuple with SSN='999887777' to '987654321'.

Result: Violates Primary key constraints

ER-to-Relational Mapping

Step 1

» For each regular entity type E in the ER schema, create a relation R that includes all the simple attributes of E

» For composite attributes, use the simple component attributes.

» Choose one of the key attributes of E as the primary key for R.

» If the chosen key was a composite, the set of simple attributes that form it will together be the primary key of R.

Step 2

» For each weak entity type W in the ER schema with owner entity type E, create a relation R that includes all simple attributes (or simple components of composites) of W.

» Include as foreign key attributes of R the primary key attribute of the relation that corresponds to the owner entity type E.

» The primary key of R is the combination of the primary key of the owner and the partial key of the weak entity type, if any.

Step 3

» 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.

» Choose one of the relations (say S). 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 for the role of S.

» Include the simple attributes of R as attributes of S.

Step 4

» For each binary 1:N relationship type R, identify the relation S that represents the entity type participating at the N-side of the relationship

» Include as a foreign key in S the primary key of the relation T that represents the other entity type participating in R.

» Include the simple attributes of R as attributes of S.

Step 5

» For each binary M:N relationship type R, create a new relation S to represent R.

» Include as foreign key attributes in S the primary keys of the participating entity types.

» Their combination will form the primary key.

» Include any attributes of R as attributes of S.

Step 6

» For each multi-valued attribute A, create a new relation R.

» R will include an attribute corresponding to A, plus the primary key attribute K of the relation that has A as an attribute.

» The primary key of R is the combination of A and K.

Step 7

» For each n-ary relationship type R, where n>2, create a new relation S.

» Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types

» Include any attributes of R.

» The primary key of S is usually a combination of all the foreign keys in S.

ER TO RELATIONAL MAPPING Example

The ER model is convenient for representing an initial, high-level database design. Given an ER diagram describing a databa'3e, a standard approach is taken to generating a relational database schema that closely approximatesthe ER design. We now describe how to translate an ER diagram into a collection of tables with associated constraints, that is, a relational database schema.

3.5.1 Entity Sets to Tables

An entity set is mapped to a relation in a straightforward way: Each attribute of the entity set becomes an attribute of the table. Note that we know both the domain of each attribute and the (primary) key of an entity set.

Consider the Employees entity set with attributes *ssn, name,* and *lot* shown in Figure 3.8. A possible instance of the Employees entity set, containing three

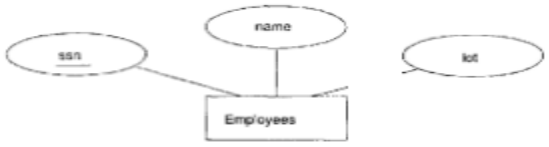


Figure 3.8 The Employees Entity Set

Employees entities, is shown in Figure 3.9 in a tabular format.

ssn	*name*	*lot*
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

Figure 3.9 An Instance of the Employees Entity Set

The following SQL statement captures the preceding information, including the domain constraints and key information:

```
CREATE TABLE Employees ( ssn      CHAR(11),
                         name     CHAR(30),
                         lot      INTEGER,
                         PRIMARY KEY (ssn) )
```

3.5.2 Relationship Sets (without Constraints) to Tables

A relationship set, like an entity set, is mapped to a relation in the relational model.

To represent a relationship, we must be able to identify each participating entity and give values to the descriptive attributes of the relationship. Thus, the attributes of the relation include:

• The primary key attributes of each participating entity set, as foreign key fields.

• The descriptive attributes of the relationship set.

The set of nondescriptive attributes is a superkey for the relation. If there are no key constraints, this set of attributes is a candidate key.

The set of nondescriptive attributes is a superkey for the relation. If there are no key constraints (see Section 2.4.1), this set of attributes is a candidate key.

Consider the Works_In2 relationship set shown in Figure 3.10. Each department has offices in several locations and we want to record the locations at which each employee works.

Figure 3.10 A Ternary Relationship Set

All the available information about the Works_In2 table is captured by the following SQL definition:

CREATE TABLE Works_In2 (ssn CHAR(11), did INTEGER, address CHAR(20) , since DATE, PRIMARY KEY (8sn, did, address), FOREIGN KEY (ssn) REFERENCES Employees, FOREIGN KEY (address) REFERENCES Locations, FOREIGN KEY (did) REFERENCES Departments);

Finally, consider the Reports_To relationship set shown in Figure 3.11. The

Figure 3.11 The Reports_To Relationship Set

role indicators *supervisor* and *subordinate* are used to create meaningful field names in the CREATE statement for the Reports_To table:

```
CREATE TABLE Reports_To (
        supervisor...ssn    CHAR(11),
        subordinate...ssn   CHAR(11),
        PRIMARY KEY (supervisor_ssn, subordinate_ssn),
        FOREIGN KEY (supervisor...ssn) REFERENCES Employees(ssn),
        FOREIGN KEY (subordinate...ssn) REFERENCES Employees(ssn) )
```

3.5.3 Translating Relationship Sets with Key Constraints

If a relationship set involves n entity sets and somem of them are linked via arrows in the ER diagTam, the key for anyone of these m entity sets constitutes a key for the relation to which the relationship set is mapped.

Consider the relationship set Manages shown in Figure 3.12. The table cor-

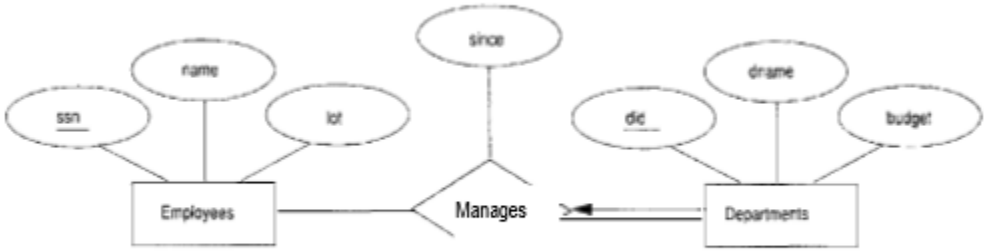


Figure 3.12 Key Constraint on Manages

The following SQL statement, defining a DepLMgr relation that captures the information in both Departments and Manages, illustrates the approach to translating relationship sets with key constraints:

CREATE TABLE DepLMgr(did INTEGER, dname CHAR(20), budget REAL, ssn CHAR (11) , since DATE, PRIMARY KEY (did), FOREIGN KEY (ssn) REFERENCES Employees)

Note that ssn can take on null values.

3.5.4 Translating Relationship Sets with Participation Constraints

Consider the ER diagram in Figure 3.13, which shows two relationship sets, Manages and "Works_In.

Every department is required to have a manager, due to the participation constraint, and at most one manager, due to the key constraint.

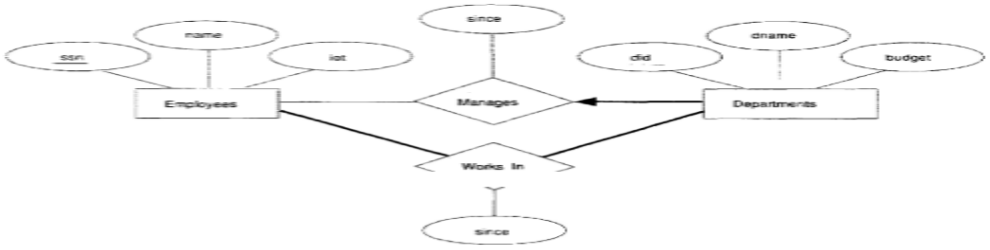


Figure 3.13 Manages and WorksJn

```
CREATE TABLE Dept_Mgr ( did       INTEGER,
                        dname     CHAR(20),
                        budget    REAL,
                        ssn       CHAR(11) NOT NULL,
                        since     DATE,
                        PRIMARY KEY (did),
                        FOREIGN KEY (ssn) REFERENCES Employees
                                  ON DELETE NO ACTION)
```

It also captures the participation constraint that every department must have a manager: Because ssn cannot take on null values, each tuple of Dep-Mgr identifies a tuple in Employees (who is the manager). The NO ACTION specification, which is the default and need not be explicitly specified, ensures that an Employees tuple cannot be deleted while it is pointed to by a Dept-Mgr tuple.

3.5.5 Translating Weak Entity Sets

A weak entity set always participates in a one-to-many binary relationship and has a key constraint and total participation.we must take into account that the weak entity has only a partial key. Also, when an owner entity is deleted, we want all owned weak entities to be deleted.

Consider the Dependents weak entity set shown in Figure 3.14, with partial key *pname*. A Dependents entity can be identified uniquely only if we take the key of the *owning* Employees entity and the *pname* of the Dependents entity, and the Dependents entity must be deleted if the owning Employees entity is deleted.

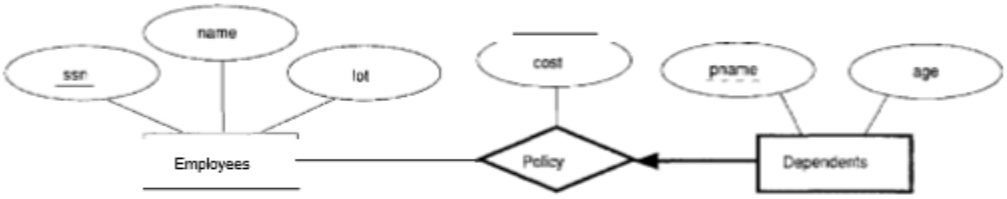


Figure 3.14 The Dependents Weak Entity Set

We can capture the desired semantics with the following definition of the Dep_Policy relation:

```
CREATE TABLE Dep_Policy (pname   CHAR(20),
                         age     INTEGER,
                         cost    REAL,
                         ssn     CHAR(11),
                         PRIMARY KEY (pname, ssn),
                         FOREIGN KEY (ssn) REFERENCES Employees
                                 ON DELETE CASCADE )
```

3.5.6 Translating Class Hierarchies

We present the two basic approaches to handling ISA hierarchies by applying them to the ER diagram shown in Figure 3.15:

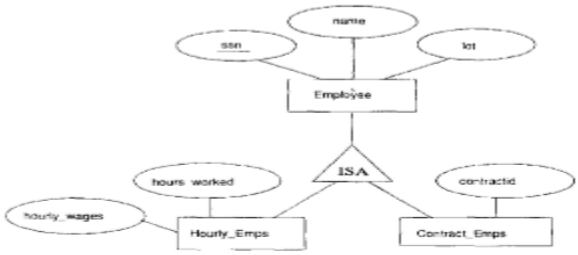


Figure 3.15 Class Hierarchy

1. We can map each of the entity sets Employees, Hourly_Emps, and ContracLEmps to a distinct relation. The Employees relation is created as in Section 2.2. We discuss Hourly_Emps here; ContracLEmps is handled similarly. The relation for Hourly_Emps includes the *hourly_wages* and *hours_worked* attributes of Hourly_Emps. It also contains the key attributes of the superclass *(ssn,* in this example), which serve as the primary key for Hourly_Emps, as well as a foreign key referencing the superclass (Employees). For each Hourly_Emps entity, the value of the *name* and *lot* attributes are stored in the corresponding row of the superclass (Employees). Note that if the superclass tuple is deleted, the delete must be cascaded to Hourly_Emps.
2. Alternatively, we can create just two relations, corresponding to Hourly_Emps and ContracLEmps. The relation for Hourly_Emps includes all the attributes of Hourly_Emps as well as all the attributes of Employees (i.e., *ssn, name, lot, hourly_wages, hours_worked*).

3.5.7 Translating ER Diagrams with Aggregation

Consider the ER diagram shown in Figure 3.16. The Employees, Projects,

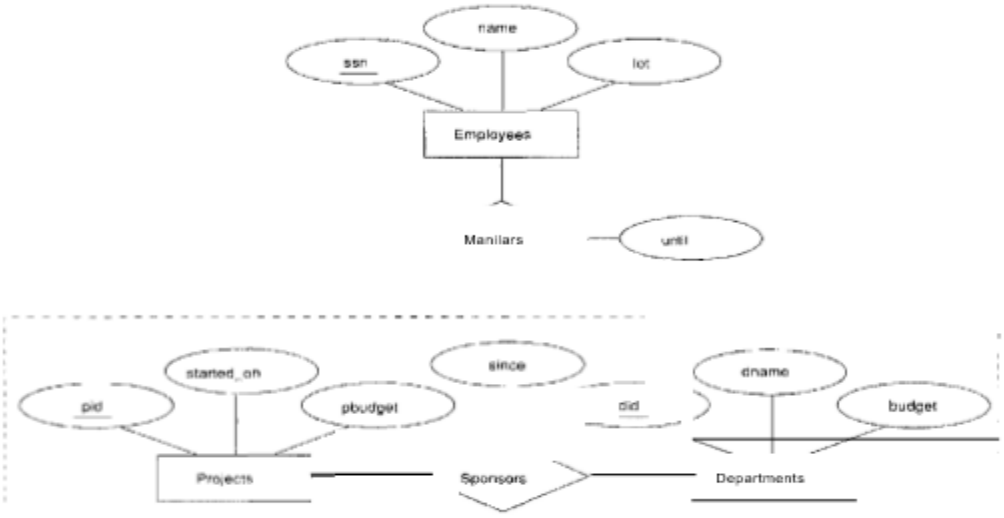


Figure 3.16 Aggregation

and Departments entity sets and the Sponsors relationship set are mapped as described in previous sections. For the Monitors relationship set, we create a relation with the following attributes: the key attributes of Employees *(88n),* the key attributes of Sponsors (*did, pid*), and the descriptive attributes of Monitors (*until*). This translation is essentially the standard mapping for a relationship set, as described in Section 3.5.2.

3.5.8 ER to Relational: Additional Examples

Consider the ER diagram shown in Figure 3.17. We can use the key constraints

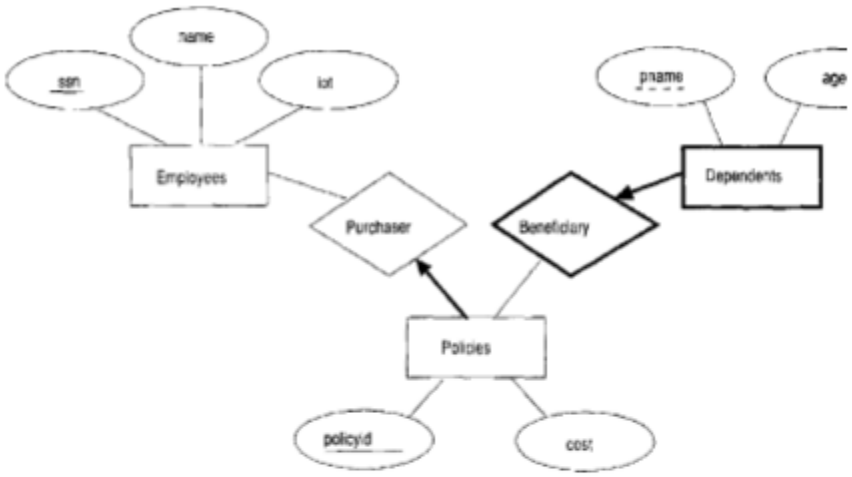


Figure 3.17 Policy Revisited

to combine Purchaser information with Policies and Beneficiary information with Dependents, and translate it into the relational model as follows:

```
CREATE TABLE Policies ( policyid INTEGER,
                        cost     REAL,
                        ssn      CHAR(11) NOT NULL,
                        PRIMARY KEY (policyid),
                        FOREIGN KEY (ssn) REFERENCES Employees
                                ON DELETE CASCADE )
```

CREATE TABLE Dependents (pnameCHAR(20) , age INTEGER, policyid INTEGER, PRIMARY KEY (pname, policyid), FOREIGN KEY (policyid) REFERENCES Policies ON DELETE CASCADE);

SQL

The SQL language may be considered one of the major reasons for the commercial success of relational databases. The name SQL is presently expanded as Structured Query Language. Originally, SQL was called SEQUEL (Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R. SQL is now the standard language for commercial relational DBMSs.

SQL is a comprehensive database language: It has statements for data definitions, queries, and updates. Hence, it is both a DDL and a DML. It also has rules for embedding SQL statements into a general-purpose programming language such as Java,COBOL,or C/C++.

4.1 SQL Data Definition and Data Types

SQL uses the terms table, row, and column for the formal relational model terms relation,tuple,andattribute,respectively.We will use the corresponding terms interchangeably. The main SQL command for data definition is the CREATE statement, which can be used to create schemas, tables (relations), and domains (as well as other constructs such as views, assertions, and triggers).

4.1.1 Schema and Catalog Concepts in SQL

The concept of an SQL schema was incorporated starting with SQL2 in order to group together tables and other constructs that belong to the same database application. An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema,as well as descriptors for each element in the schema.Schema elements include tables,constraints,views,domains,and other constructs (such as authorization grants) that describe the schema.

A schema is created via the CREATE SCHEMA statement,which can include all the schema elements' definitions. Alternatively, the schema can be assigned a name and authorization identifier, and the elements can be defined later. For example, the following statement creates a schema called COMPANY,owned by the user with authorization identifier 'Jsmith'.Note that each statement in SQL ends with a semicolon.

CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';

In general,not all users are authorized to create schemas and schema elements.The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

4.1.2 The CREATE TABLE Command in SQL

The CREATE TABLE command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and any attribute constraints, such as NOT NULL. The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared, or they can be added later using the ALTER TABLE command. we can explicitly attach the schema name to the relation name, separated by a period.

For example,by writing

CREATE TABLE COMPANY.EMPLOYEE ...

rather than

CREATE TABLE EMPLOYEE ...

we can explicitly (rather than implicitly) make the EMPLOYEEtable part of the COMPANY schema.

The relations declared through **CREATE TABLE** statements are called base tables (or base relations); this means that the relation and its tuples are actually created and stored as a file by the DBMS.Base relations are distinguished from virtual relations, created through the **CREATE VIEW**statement which may or may not correspond to an actual physical file.

Example:

CREATE TABLE EMPLOYEE (Fname VARCHAR(15) NOT NULL, Minit CHAR, Lname VARCHAR(15) NOT NULL, Ssn CHAR(9) NOT NULL, Bdate DATE, Address VARCHAR(30), Sex CHAR, Salary DECIMAL(10,2), Super_ssn CHAR(9), Dno INT NOT NULL, PRIMARY KEY (Ssn), FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn), FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber)) ;

CREATE TABLE DEPARTMENT (Dname VARCHAR(15) NOT NULL, Dnumber INT NOT NULL, Mgr_ssn CHAR(9) NOT NULL, Mgr_start_date DATE, PRIMARY KEY (Dnumber), UNIQUE (Dname), FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn));

4.1.3 Attribute Data Types and Domains in SQL

The basic data types available for attributes include numeric, character string, bit string,Boolean,date,and time.

■**Numeric data types** include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION). Formatted numbers can be declared by using DECIMAL(i,j)—or DEC(i,j) or NUMERIC(i,j)—where i, the precision, is the total number of decimal digits and j, the scale, is the number of digits after the decimal point.

■**Character-string data types** are either fixed length—CHAR(n) or CHARACTER(n), where n is the number of characters—or varying length— VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n),where nis the maximum number of characters.When specifying a literal string value,it is placed between single quotation marks (apostrophes), and it is case sensitive (a distinction is made between uppercase and lowercase).3 For fixedlength strings, a shorter string is padded with blank characters to the right. For example, if the value 'Smith' is for an attribute of type CHAR(10), it is padded with five blank characters to become 'Smith ' if needed.

■**Bit-string data types** are either of fixed length n—BIT(n)—or varying length—BIT VARYING(n), where n is the maximum number of bits. The default for n, the length of a character string or bit string, is 1. Literal bit strings are placed between single quotes but preceded by a B to distinguishthem from character strings;

for example,B'10101'

■A **Boolean data type** has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN.

■ The **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD.

The **TIME** data type has at least eight positions,with the components HOUR,MINUTE,and SECOND in the form HH:MM:SS.

■ A **timestamp** data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.Literal values are represented by singlequoted strings preceded by the keyword TIMESTAMP, with a blank space between data and time;

for example, TIMESTAMP '2008-09-27 09:12:47.648302'.

■Another data type related to DATE, TIME, and TIMESTAMP is the **INTERVAL** data type. This specifies an interval—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp. Intervals are qualified to be either YEAR/MONTH intervals or DAY/TIME intervals.

It is possible to specify the data type of each attribute directly,as in Figure 4.1;alternatively, a domain can be declared, and the domain name used with the attribute specification. This makes it easier to change the data type for a domain that is used by numerous attributes in a schema,and improves schema readability.For example, we can create a domain SSN_TYPE by the following statement:

CREATE DOMAIN SSN_TYPE AS CHAR(9);

We can use SSN_TYPE in place of CHAR(9) in the above create table for the attributes Ssn and Super_ssn of EMPLOYEE, Mgr_ssn of DEPARTMENT, Essn of WORKS_ON, and Essn of DEPENDENT.

4.2 Specifying Constraints in SQL

Now will describe the basic constraints that can be specified in SQL as part of table creation. These include key and referential integrity constraints, restrictions on attribute domains and NULLs,and constraints on individual tuples within a relation.

4.2.1 Specifying Attribute Constraints and Attribute Defaults

Because SQL allows NULLs as attribute values,a constraint **NOT NULL**may be specified if NULLis not permitted for a particular attribute.This is always implicitly specified for the attributes that are part of the primary key of each relation,but it can be specified for any other attributes whose values are required not to be NULL.

It is also possible to define a default value for an attribute by appending the clause **DEFAULT**<value> to an attribute definition.

Another type of constraint can restrict attribute or domain values using the **CHECK** clause following an attribute or domain definition.

For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then, we can change the attribute declaration of Dnumber in the DEPARTMENT table to the following: **Dnumber INT NOT NULL CHECK (Dnumber> 0 AND Dnumber< 21);**

Ex:

CREATE TABLE EMPLOYEE (Dno INT NOT NULL **DEFAULT** 1, CONSTRAINT EMPPK PRIMARY KEY (Ssn), CONSTRAINT EMPDEPTFK FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber) **ON DELETE SET DEFAULT** ON UPDATE CASCADE);

TheCHECKclause can also be used in conjunction with the CREATE DOMAINstatement.

For example,we can write the following statement:

CREATE DOMAIN D_NUM AS INTEGER CHECK (D_NUM > 0 AND D_NUM <21);

We can then use the created domain D_NUM as the attribute type for all attributes that refer to department numbers In create table , such as Dnumber of DEPARTMENT, Dnum of PROJECT,Dno of EMPLOYEE,and so on.

4.2.2 Specifying Key and Referential Integrity Constraints

The PRIMARY KEY clause specifies one or more attributes that make up the primary key of a relation. If a primary key has a single attribute, the clause can follow the attribute directly.

Ex: Dnumber INT PRIMARY KEY;

The UNIQUE clause specifies alternate (secondary) keys, as illustrated in the DEPARTMENT and PROJECT table declarations.

Ex: DnameVARCHAR(15) UNIQUE;

Referential integrity is specified via the FOREIGN KEY clause. A referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is modified. The default action that SQL takes for an integrity violation is to reject the update operation that will cause a violation, which is known as the RESTRICT option. However, the schema designer can specify an alternative action to be taken by attaching a referential triggered action clause to any foreign key constraint. The options include SET NULL, CASCADE, and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE. We illustrate this with the examples shown in create table Here, the database designer chooses ON DELETE SET NULL and ON UPDATE CASCADE for the foreign key Super_ssn of EMPLOYEE. This means that if the tuple for a supervising employee is deleted, the value of Super_ssnis automatically set to NULLfor all employee tuples that were referencing the deleted employee tuple.On the other hand,if the Ssnvalue for a supervising employee is updated (say,because it was entered incorrectly),the new value is cascaded to Super_ssn for all employee tuples referencing the updated employee tuple.8

4.2.3 Giving Names to Constraints

A constraint may be given a constraint name,following the keyword CONSTRAINT. The names of all constraints within a particular schema must be unique. A constraint name is used to identify a particular constraint in case the constraint must be dropped later and replaced with another constraint. Giving names to constraints is optional.

Ex: CONSTRAINT **EMPPK** PRIMARY KEY (Ssn).

4.2.4 Specifying Constraints on Tuples Using CHECK

In addition to key and referential integrity constraints, which are specified by special keywords, other table constraints can be specified through additional CHECK clauses at the end of a CREATE TABLE statement. These can be called tuple-based constraints because they apply to each tuple individually and are checked whenever a tuple is inserted or modified.Forexample,suppose that the DEPARTMENTtable in Figure 4.1 had an additional attribute Dept_create_date, which stores the date when the department was created. Then we could add the following CHECK clause at the end of the CREATE TABLE statement for the DEPARTMENT table to make sure that a manager's start date is later than the department creation date.

CHECK (Dept_create_date<= Mgr_start_date);

4.3 Basic Retrieval Queries in SQL

SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values.Hence,in general, an SQL table is not a set of tuples, because a set does not allow two identical members; rather,it is a multiset (sometimes called a bag) of tuples.

Figure 3.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

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
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	Sex	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

4.3.1 The SELECT-FROM-WHERE Structure of Basic SQL Queries

The basic form of the SELECT statement, sometimes called a mapping or a select-from-where block, isformed of the three clauses SELECT, FROM, and WHERE and has the following form:

SELECT <attribute list>
FROM <table list>
WHERE <condition>;
where
■<attribute list> is a list of attribute names whose values are to be retrieved by the query.
■<table list> is a list of the relation names required to process the query.
■<condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.
In SQL,the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <, <=, >, >=, and <>(not equal operator).

Query 1.

Ex:**Retrieve the birth date and address of the employee(s) whose name is 'John B.Smith'.**

SELECTBdate, Address **FROM** EMPLOYEE **WHERE**Fname='John' AND Minit='B' AND Lname='Smith';

(a)

Bdate	Address
1965-01-09	731Fondren, Houston, TX

Query 1.a.

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

SELECT Fname, Lname, Address FROM EMPLOYEE, DEPARTMENT WHERE Dname='Research' AND Dnumber=Dno;

(b)

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

In the WHERE clause of query, the condition Dname = 'Research'is a selection condition that chooses the particular tuple of interest in the DEPARTMENT table,becauseDname is an attribute of DEPARTMENT. The condition Dnumber = Dno is called a join condition, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to the value of Dno in EMPLOYEE.

A query that involves only selection and join conditions plus projection attributes is known as a select-project-join query. The next example is a select-project-join query with two join conditions.

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.

SELECT Pnumber, Dnum, Lname, Address, Bdate FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford';

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

The join condition Dnum = Dnumber relates a project tuple to its controlling department tuple,whereas the join condition Mgr_ssn=Ssnrelates the controlling department tuple to the employee tuple who manages that department.

4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

In SQL,the same name can be used for two (or more) attributes as long as the attributes are in different relations.If this is the case,and a multitable query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name to prevent ambiguity. This is done by **prefixing the relation name to the attribute name and separating the two by a period**.

SELECT Fname, EMPLOYEE.Name, Address FROM EMPLOYEE, DEPARTMENT WHERE DEPARTMENT.Name='Research' AND DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;

We can also create an **alias** for each table name to avoid repeated typing of long table names.

For each employee**, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.**

SELECT E.Fname, E.Lname, S.Fname, S.Lname

FROM EMPLOYEE AS E, EMPLOYEE AS S WHERE E.Super_ssn=S.Ssn;

(d)

E.Fname	E.Lname	S.Fname	S.Lname
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Alicia	Zelaya	Jennifer	Wallace
Jennifer	Wallace	James	Borg
Ramesh	Narayan	Franklin	Wong
Joyce	English	Franklin	Wong
Ahmad	Jabbar	Jennifer	Wallace

In this case, we are required to declare alternative relation names E and S, called aliases or tuple variables, for the EMPLOYEE relation. An alias can follow the keyword AS, as shown in Query,or it can directly follow the relation name—for example, by writing EMPLOYEE E, EMPLOYEE S in the FROM clause of Query.

It is also possible to rename the relation attributes within the query in SQL by giving them aliases. For example,if we write EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno) in the FROMclause,Fn becomes an alias for Fname,Mi for Minit,Ln for Lname,and so on.

In Query,we can think of E and S as two different copies of the EMPLOYEErelation;the first, E, represents employees in the role of supervisees or subordinates; the second, S, represents employees in the role of supervisors.

4.3.3 Unspecified WHERE Clause and Use of the Asterisk

A missing WHERE clause indicates no condition on tuple selection; hence, all tuples of the relation specified in the FROM clause qualify and are selected for the query result.If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT—all possible tuple combinations—of these relations is selected.

Queries 9 and 10.

Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

Q9: SELECT Ssn FROM EMPLOYEE;

123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555

Q10: SELECT Ssn, Dname FROM EMPLOYEE, DEPARTMENT;

(f)

Ssn	Dname
123456789	Research
333445555	Research
999887777	Research
987654321	Research
666884444	Research
453453453	Research
987987987	Research
888665555	Research
123456789	Administration
333445555	Administration
999887777	Administration
987654321	Administration
666884444	Administration
453453453	Administration
987987987	Administration
888665555	Administration
123456789	Headquarters
333445555	Headquarters
999887777	Headquarters
987654321	Headquarters
666884444	Headquarters
453453453	Headquarters
987987987	Headquarters
888665555	Headquarters

It is extremely important to specify every selection and join condition in the WHERE clause; if any such condition is overlooked, incorrect and very large relations may result.

4.3.4 Tables as Sets in SQL

SQL usually treats a table not as a set but rather as a multiset; duplicate tuples can appear more than once in a table,and in the result of a query. SQL does not automatically eliminate duplicate tuples in the results of queries.

If we do want to eliminate duplicate tuples from the result of an SQL query, we use the keyword DISTINCT in the SELECT clause, meaning that only distinct tuples should remain in the result.

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values

(Q11A).

Q11: SELECT ALL Salary FROM EMPLOYEE;

Q11A: SELECT **DISTINCT** Salary FROM EMPLOYEE;

In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.

(a)

Salary
30000
40000
25000
43000
38000
25000
25000
55000

(b)

Salary
30000
40000
25000
43000
38000
55000

Figure 4.4
Results of additional SQL queries when applied to the COMPANY database state shown in Figure 3.6. (a) Q11. (b) Q11A. (c) Q16. (d) Q18.

SQL has directly incorporated some of the set operations from mathematical set theory,which are also part of relational algebra (see Chapter 6).There are set union **(UNION**), set difference (**EXCEPT**) and set intersection **(INTERSECT**) operations. The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminated from the result. These set operations apply only to union-compatible relations, so we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations.The next example illustrates the use of UNION.

Query 4. Make a list of all 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.

(SELECT DISTINCT Pnumber FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Lname='Smith')

UNION

(SELECT DISTINCT Pnumber FROM PROJECT, WORKS_ON, EMPLOYEE WHERE Pnumber=Pno AND Essn=Ssn AND Lname='Smith');

(a) R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

(b) T

A
a1
a1
a2
a2
a2
a3
a4
a5

(c) T

A
a2
a3

(d) T

A
a1
a2

Figure 4.5
The results of SQL multiset operations. (a) Two tables, R(A) and S(A). (b) R(A) UNION ALL S(A). (c) R(A) EXCEPT ALL S(A). (d) R(A) INTERSECT ALL S(A).

4.3.5 Substring Pattern Matching and Arithmetic Operators

In this section we discuss several more features of SQL.The first feature allows comparison conditions on only parts of a character string, using the **LIKE** comparison operator.This can be used for string pattern matching. Partial strings are specified using two reserved characters: **%** replaces an arbitrary number of zero or more characters,and the underscore **(_)** replaces a single character.Forexample,consider the following query.

Query 12.**Retrieve all employees whose address is in Houston,Texas.**

SELECT Fname, Lname FROM EMPLOYEE WHERE Address LIKE '%Houston,TX%';

Query 12A.**Find all employees who were born during the 1950s.**

SELECT Fname, Lname FROM EMPLOYEE WHERE Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';

Another feature allows the use of arithmetic in queries. The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains.

Query 13.**Show the resulting salaries if every employee working on the 'ProductX'project is given a 10 percent raise.**

SELECT E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P WHERE E.Ssn=W.Essn AND W.Pno=P.Pnumber AND P.Pname='ProductX';

Another comparison operator, which can be used for convenience,is**BETWEEN**, which is illustrated in Query 14.

Query 14.**Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.**

SELECT * FROM EMPLOYEE WHERE (Salary BETWEEN 30000 AND 40000) AND Dno = 5;

The condition (Salary BETWEEN30000AND40000) in Q14is equivalent to the condition

((Salary >= 30000) AND (Salary <= 40000)).

4.3.6 Ordering of Query Results

SQL allows the user to order the tuples in the result of a query by the values of one or more of the attributes that appear in the query result, by using the ORDER BY clause.

Ex: SELECT Salary FROM EMPLOYEE ORDER BY Salary ;

The default order is in ascending order of values.We can specify the keyword DESC if we want to see the result in a descending order of values.The keyword ASCcan be used to specify ascending order explicitly.

4.3.7 Discussion and Summary of Basic SQL Retrieval Queries

A simple retrieval query in SQL can consist of up to four clauses, but only the first two—SELECT and FROM—are mandatory. The clauses are specified in the following order,with the clauses between square brackets [...] being optional:

SELECT <attribute list> FROM <table list>[WHERE<condition>] [ORDER BY <attribute list>];

The SELECTclause lists the attributes to be retrieved,and the FROMclause specifies all relations (tables) needed in the simple query. The WHERE clause identifies the conditions for selecting the tuples from these relations, including join conditions if needed.ORDERBYspecifiesan order for displaying the results of a query.

4.4 INSERT, DELETE, and UPDATE Statements in SQL

4.4.1 The INSERT Command

In its simplest form,INSERTis used to add a single tuple to a relation.We must specify the relation name and a list of values for the tuple.The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.

Ex**: INSERT INTO EMPLOYEE VALUES ('Richard','K','Marini','653298653','1962-12-30','98 Oak Forest, Katy, TX','M', 37000,'653298653', 4);**

A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERTcommand.This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.

INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn) VALUES ('Richard','Marini', 4,'653298653');

Attributes not specified in the above query are set to their DEFAULT or to NULL, and the values are listed in the same order as the attributes are listed in the INSERTcommand itself.

A variation of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the result of a query.

CREATE TABLE WORKS_ON_INFO (Emp_nameVARCHAR(15), Proj_name VARCHAR(15), Hours_per_week DECIMAL(3,1));

INSERT INTO WORKS_ON_INFO (Emp_name, Proj_name, Hours_per_week)
SELECT E.Lname, P.Pname, W.Hours FROM PROJECT P, WORKS_ON W, EMPLOYEE E WHERE P.Pnumber=W.Pno AND W.Essn=E.Ssn;

4.4.2 The DELETE Command

The DELETE command removes tuples from a relation.It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted. Tuples are explicitly deleted from only one table at a time. However, the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential integrity constraints of the DDL. Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command. A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table.

Ex: DELETE FROM EMPLOYEE WHERE Lname='Brown';

DELETE FROM EMPLOYEE WHERE Ssn='123456789';

DELETE FROM EMPLOYEE WHERE Dno=5;

DELETE FROM EMPLOYEE;

The above DELETE commands if applied independently to the database in Figure 3.6,will delete zero,one,four,and all tuples,respectively,from the EMPLOYEErelation:

4.4.3 The UPDATE Command

The UPDATE command is used to modify attribute values of one or more selected tuples. As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation. However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL. An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values.

For example**, to change the location and controlling department number of project number 10 to 'Bellaire'and 5,respectively,we use the below query**

UPDATE PROJECT SET Plocation = 'Bellaire',Dnum =5 WHERE Pnumber=10;

Several tuples can be modified with a single UPDATE command. An example is to give all employees in the 'Research'department a 10 percent raise in salary.

UPDATE EMPLOYEE SET Salary = Salary * 1.1 WHERE Dno = 5;

It is also possible to specify NULLor DEFAULTas the new attribute value.

4.5 Additional Features of SQL

SQL has a number of additional features .These are as follows:

SQL has various techniques for specifying complex retrieval queries, including nested queries, aggregate functions, grouping, joined tables,outerjoins,and recursive queries; SQL views,triggers,and assertions; and commands for schema modification.

■ SQL has various techniques for writing programs in various programming languages that include SQL statements to access one or more databases. These include embedded (and dynamic) SQL, SQL/CLI (Call Level Interface) and its predecessor ODBC (Open Data Base Connectivity), and SQL/PSM (Persistent Stored Modules).

■ Each commercial RDBMS will have,in addition to the SQL commands,a set of commands for specifying physical database design parameters, file structures for relations, and access paths such as indexes.

SQL has transaction control commands. These are used to specify units of database processing for concurrency control and recovery purposes.

. ■ SQL has language constructs for specifying the granting and revoking of privileges to users.Privileges typically correspond to the right to use certain SQL commands to access certain relations. Each relation is assigned an owner, and either the owner or the DBA staff can grant to selected users the privilege to use an SQL statement—such as SELECT, INSERT, DELETE, or UPDATE—to access the relation.

In addition, the DBA staff can grant the privileges to create schemas, tables, or views to certain users. These SQL commands—called GRANT and REVOKE—

■ SQL has language constructs for creating triggers. These are generally referred to as active database techniques, since they specify actions that are automatically triggered by events such as database updates.

■ SQL has incorporated many features from object-oriented models to have more powerful capabilities,leading to enhanced relational systems known as object-relational.

■ SQL and relational databases can interact with new technologies such as XML .