CANARA ENGINEERING COLLEGE

Benjanapadavu – 574219

Program: COMPUTER SCIENCE & ENGINEERING Course code:18CS53
Course Name: DATABASE MANAGEMENT SYSTEM

MODULE 2

Notes prepared by - (Name & Designation) : Mr.LOHIT B and Mr.SANTOSH

OBJECTIVE: Practice SQL programming through a variety of database problems

OUTCOME: Solve broad range of query and data updation problems using relational algebra and SQL.

Contents include:

No. of Weblinks: 3
No. of University Qs and As: 10

Structure of Notes

MODULE 2: The Relational Data Model and Relational Database Constraints

Relational Model Concepts

- The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values.
- 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.
- 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.

2.1 Domains, Attributes, Tuples , and Relations :

- **A domain D** is a set of atomic values. A**tomic** means that each value in the domain is indivisible as far as the formal relational model is concerned.

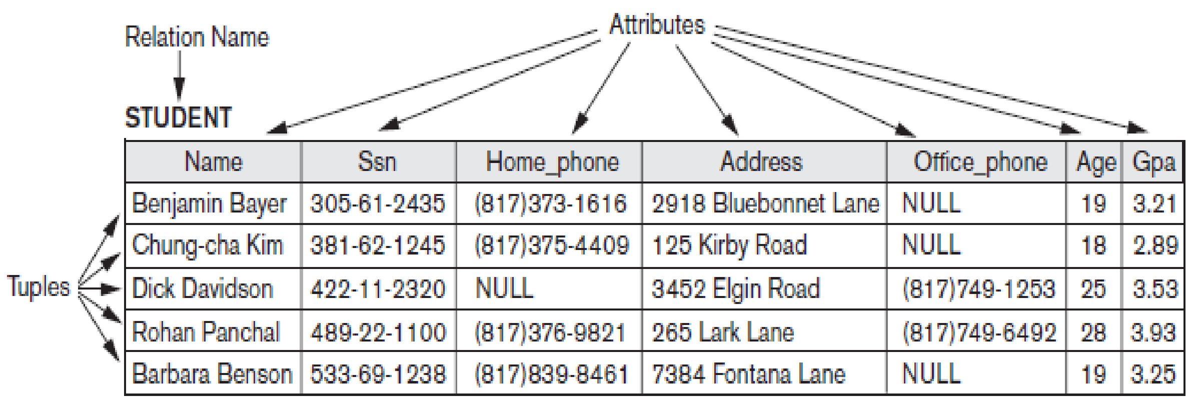
Example: Social_security_numbers: The nine-digit Social Security numbers. Names: The set of character strings that represent names of persons.

- A **data type** or **format** is also specified for each domain.

  Example: The data type for Employee_ages is an integer number between 15 and 80.

- A **relation schema R**, denoted by **R(A_1, A_2, ...,A_n)**is made up of a relation name Rand a list of attributes **A_1, A_2, ...,A_n**. 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 attribute esn of its relation schema. A relation of degree seven, would contain seven attributes describing each student. **STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)**
- A **relation** (or **relation state**) r of the relation schema R(A1, A2, ..., An), also denoted by r(R), is a set of n-tuples r = {t1, t2, ..., tm}. Each **n-tuple** t is an ordered list of n value st =<v1, v2, ..., vn>,The terms **relation intension** for the schema R and **relation extension** for a relation state r(R)are also commonly used.

Figure 3.1 shows an example of a STUDENT relation, which corresponds to the STUDENT schema. Each tuple in the relation represents a particular student . NULL values represent attributeswhose values are unknown or do not exist for some individual STUDENT tuple.



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

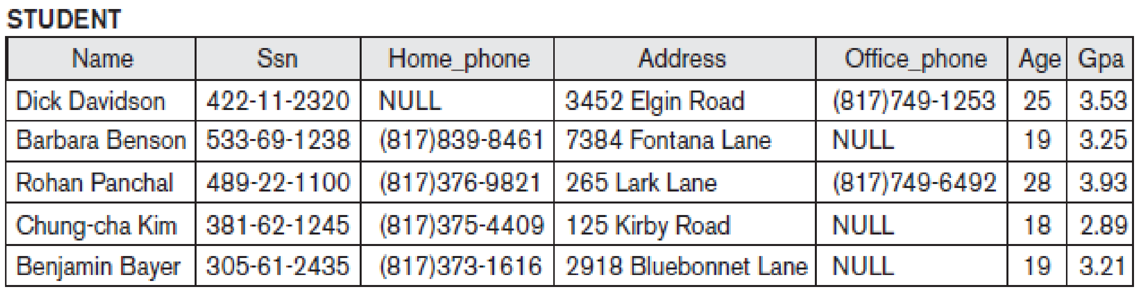
Figure 3.1
The attributes and tuples of a relation STUDENT.

2.1.2 Characteristics of Relations

1. Ordering of Tuples in a Relation.

- A relation is defined as a set of tuples. Elements of a set have no order among them; hence, tuples in a relation do not have any particular order.
- The definition of a relation does not specify any order: There is no preference for one ordering over another. Hence, the relation displayed in ***Figure 3.2 is considered identical to the one shown in Figure 3.1.***

Figure 3.2
The relation STUDENT from Figure 3.1 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	(817)749-1253	25	3.53
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21

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 = {A1, A2, ..., An} is a set of attributes and a relation state r(R) is a finite set of mapping sr = {t1, t2, ..., tm}, where each tuple t_i is a **mapping** from R to D, and D is the **union** (denoted by ∪) of the attribute domains; that is, D = dom(A1) ∪dom(A2) ∪... dom(An). In this definition, t[Ai] must be in dom(Ai) for 1 ≤i ≤n for each mapping t in r. Each mapping t_i is called a tuple.
- According to this definition of tuple as a mapping, a **tuple** can be considered as a **set** of (<attribute>, <value>) pairs. The ordering of attributes is not important, because the attribute name appears with its value. ***By this definition, the two tuples shown in Figure 3.3 are identical.***

Figure 3.3
Two identical tuples when the order of attributes and values is not part of relation definition.

t = < (Name, Dick Davidson),(Ssn, 422-11-2320),(Home_phone, NULL),(Address, 3452 Elgin Road),
(Office_phone, (817)749-1253),(Age, 25),(Gpa, 3.53)>

t = < (Address, 3452 Elgin Road),(Name, Dick Davidson),(Ssn, 422-11-2320),(Age, 25),
(Office_phone, (817)749-1253),(Gpa, 3.53),(Home_phone, NULL)>

3. Values and NULLs in the Tuples.

- Each value in a tuple is an **atomic** value; that is, it is not divisible. Hence, composite and multivalued attributes are not allowed.
- The multivalued attributes must be represented by separate relations, and composite attributes are represented only by their simple component attributes in the basic relational model.
- An important concept is that of NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple.

Example: STUDENT tupleshave NULL for their office phones because they do not have an office . Another student has a NULL for homephone, presumably because either he does not have a home phone or he has one butwe do not know it (value is unknown).

- 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. An example of the last type of NULL will occur if we add an attribute Visa_status to the STUDENT relation that applies only to tuples representing foreign students.
- During database design, it is best to avoid NULL values as much as possible

4. Interpretation (Meaning) of a Relation.

- Each tuple in the relation can be interpreted as a **fact** or a particular instance of the assertion. ***For example, a STUDENT relation whose Name is Benjamin Bayer, Ssn is 305-61-2435, Age is 19, and so on.***
- Relations may represent facts about entities, whereas other relations may represent facts about relationships. For example, a relation schema MAJORS(Student_ssn, Department_code) asserts that students major in academic disciplines. A tuple in this relation relates a student to his or her major discipline. Hence, the relational model represents facts about both entities and relationships uniformly as relations.

2.2 Relational Model Constraints :

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

1. Inherent model-based constraints or **implicit constraints**.

- Are the constraints that are inherent in the data model. The characteristics of relations **(Please refer 3.1.2)** are the inherent constraints of the relational model.

  For example*, the constraint that a relation cannot have duplicate tuples is an inherent constraint*

2. Application-based or **semantic constraints** or **business rules**.

- Are the 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.
- Such constraints can be specified and enforced within the application programs that update the database, or by using a general-purpose **constraint specification language**. Mechanisms called **triggers** and **assertions** can be used. In SQL, CREATE ASSERTION and CREATE TRIGGER statements can be used for this purpose.

3. Schema-based constraints or **explicit constraints**

Are the constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL (data definition language).The schema-based constraints include **domain constraints**, **key constraints**, **constraints on NULLs, entity integrity constraints, and referential integrity constraints.**

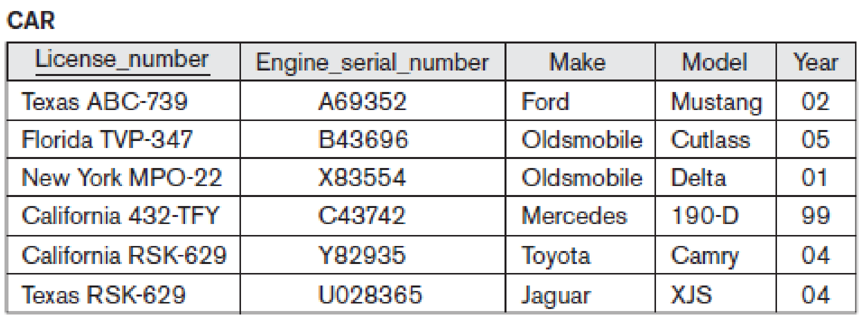
2.2.1 Domain Constraints

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, Characters, Booleans, fixed-length strings, variable-length strings, date, time and timestamp.

2.2.2 Key Constraints and Constraints on NULL Values.

- In the formal relational model, a relation is defined as a set of tuples. By definition, all elements of a set are distinct. This means that no two tuples can have the same combination of values for all their attributes.
- The **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 i.e for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that:**$t1[SK] \neq t2[SK]$**any such set of attributes SK is called a **super key** of the relation schema R.

- A super key can have redundant attributes, however, so a more useful concept is that of a key, which has no redundancy.
- A **key** K of a relation schema R is a super key of R with the additional property that removing any attribute A from K leaves a set of attributes K that is not a super key 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 super key.***
    2. ***It is a minimal super key—that is, a super key from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 hold. This property is not required by a super key.***
- The first property applies to both keys and super keys, the second property is required only for keys. Hence, a key is also a super key but not vice versa.
- Consider the STUDENT relation. The attribute set {Usn} is a **key** of STUDENT because no two student tuples can have the same value for Usn.
- Any set of attributes that includes Usn ***for example, {Ssn, Name, Age}—is a super key. However, 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 super key.***



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

Figure 3.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

- **In general,** 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 in Figure 3.4 has two candidate keys: License_number and Engine_serial_number. 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. The other candidate keys are designated as **unique keys**, and are not underlined.

Example 2: Consider the following relation : Book (BookId, BookName, Author)

- Super Keys : A Super Key is a set of one or more attributes that are taken collectively and can identify all other attributes uniquely.

  For Example,

  (BookId)

  (BookId,BookName)

  (BookId, BookName, Author)

  (BookId, Author)

  (BookName, Author)

- Candidate Keys Candidate keys are a super key which are not having any redundant attributes. In other words candidate keys are minimal super keys.

  For Example,

  (BookId)

  (BookName,Author)

- These two keys can be candidate keys, as remaining keys are having redundant attributes. Means in super key (BookId, BookName) record can be uniquely identify by just BookId and therefore BookName is redundant attribute
- Primary Key: A key which is used to uniquely identify each record is known as primary key. From above Candidate keys any one can be the primary key.

2.2.3 Relational Databases and Relational Database Schemas.

- A **relational database schema** S is a set of relation schemas S = {R1, R2, ..., R_m} and a set of **integrity constraints** IC.
- A **relational database state** DB of S is a set ofrelation states DB = {r1, r2, ..., r_m} such that each ri is a state of Ri and such that theri relation states satisfy the integrity constraints specified in IC.

  Figure 3.5 shows a relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT,DEPT_LOCATIONS,PROJECT, WORKS_ON, DEPENDENT}. The underlined attributes represent primary keys. A database state that does not obey all the integrity constraints iscalled an invalid state, and a state that satisfies all the constraints in the defined setof integrity constraints IC is called a valid state.

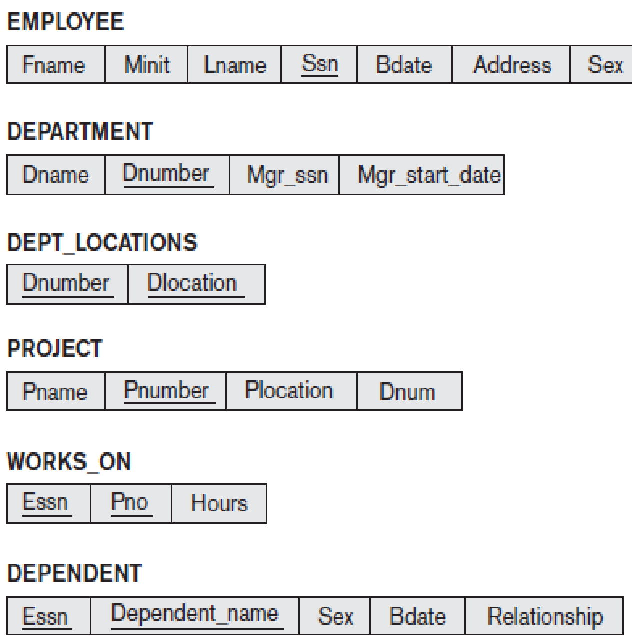

EMPLOYEE
Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno
DEPARTMENT
Dname | Dnumber | Mgr_ssn | Mgr_start_date
DEPT_LOCATIONS
Dnumber | Dlocation
PROJECT
Pname | Pnumber | Plocation | Dnum
WORKS_ON
Essn | Pno | Hours
DEPENDENT
Essn | Dependent_name | Sex | Bdate | Relationship

Figure 3.5
Schema diagram for the COMPANY relational database schema.

2.2.4 Integrity, Referential Integrity, and Foreign Keys

- 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 maynot 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**.**

- 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, The attribute Dno of EMPLOYEE gives the department number for which eachemployee works; hence, its value in every EMPLOYEE tuple must match the Dnumbervalue of some tuple in the DEPARTMENT relation.

- The formal definition of referential integrity is, Consider a set of attributes FK in relation schema R1 is a **foreign key** of R1 that **references** relation R2 if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2.
2. A value of FK in a tuple t1 of the current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is NULL. According to previous case, t1[FK] = t2[PK]. The tuple t1 references orrefers to the tuple t2.

- In this definition, R1 is called the **referencing relation** and R2 is the **referenced relation**. If these two conditions hold, a **referential integrity constraint** from R1 to R2 is said to hold. In a database of many relations, there are usually many referential integrity constraints.

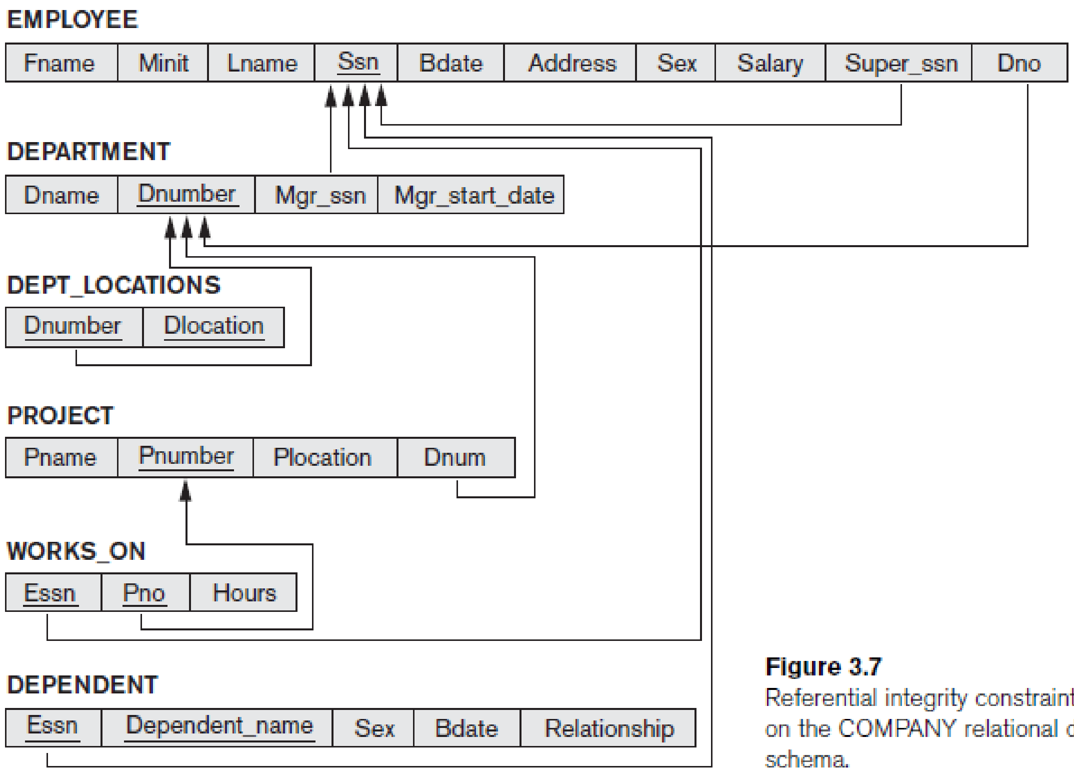

EMPLOYEE
Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno
DEPARTMENT
Dname | Dnumber | Mgr_ssn | Mgr_start_date
DEPT_LOCATIONS
Dnumber | Dlocation
PROJECT
Pname | Pnumber | Plocation | Dnum
WORKS_ON
Essn | Pno | Hours
DEPENDENT
Essn | Dependent_name | Sex | Bdate | Relationship

Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

For example, In the EMPLOYEE relation, the attribute Dno refers to the department for which an employee works; hence, we designate Dno to be a foreign key of EMPLOYEE referencing the DEPARTMENT relation. This means that a value of Dno in any tuple t1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT.

2.3 Update Operations, Transactions & Dealing with Constraint Violations

There are three basic operations that can change the states of relations in the database: **Insert, Delete, and Update.** **Insert** is used to insert one or more new tuples in a relation,

Delete is used to delete tuples, and **Update** is used to change the values of some attributes in existing tuples

2.3.1 The Insert Operation

- The **Insert** operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R.
- Insert can violate any of the four types of constraints.
    1. **Domain constraints** can be violated 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** can be violated if a key value in the new tuple t already exists in another tuple in the relation r(R).
    3. **Entity integrity** can be violated if any part of the primary key of the new tuple t is NULL.
    4. **Referential integrity** can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation.

For Example:

- **Operation 1:**

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

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

- **Operation 2:**

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

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

- **Operation 3:**

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

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

- **Operation 4:**

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

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

2.3.2 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 (ortuples) to be deleted.

Here are some examples.

- **Operation 1:**

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

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

- **Operation 2:**

  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. The first option, called **restrict,** is to reject the deletion. The second option, called **cascade,** is to attempt to cascade (or propagate) the deletion by deleting tuples that reference the tuple that is being deleted. A third option, called **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.

2.3.3 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.

Here are some examples.

- **Operation 1:**

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

  Result: Acceptable.

- **Operation 2:**

  Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 1.

  Result: Acceptable.

- **Operation 3:**

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

  Result: Unacceptable, because it violates referential integrity.

- **Operation 4:**

  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.

2.3.4 The Transaction Concept

- A **transaction** is an executing program that includessome database operations, such as reading from the database, or applying insertions,deletions, or updates to the database. At the end of the transaction, it mustleave the database in a valid or consistent state that satisfies all the constraints specifiedon the database schema.
- For example, a transaction to apply a bank withdrawalwill typically read the user account record, check if there is a sufficient balance,and then update the record by the withdrawal amount.A large number of commercial applications running against relational databases in**online transaction processing (OLTP**) systems are executing transactions at ratesthat reach several hundred per second.

2.4 The Relational Algebra:

- The basic set of operations for the relational model is the **relational algebra.**
- The relational algebra is very important for several reasons. First, it provides a formal Foundation for relational model operations. Second, ,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)

2.4 .1 Unary Relational Operations: SELECT and PROJECT.

2.4.1.1 The SELECT Operation.

- The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a **selection condition**.
- The SELECT operation is visualized as a horizontal partition of the relation into two sets of tuples. Those tuples that satisfy the condition are selected, and those tuples that do not satisfy the condition are discarded.
- In general, the SELECT operation is denoted by σ <selection condition>(R)where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression (condition) specified on the attributes of relation R.

  Example 1: *To select the EMPLOYEE tuples whose department is4, or those whose salary is greater than $30,000, we can individually specify each of these two conditions with a SELECT operation as follows*:

$$\sigma_{Dno=4}(EMPLOYEE)$$

$$\sigma_{Salary>3000}(EMPLOYEE)$$

- The Boolean expression specified in <selection condition> is made up of a numberof **clauses** of the form

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

or

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

Example 2: ***To 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, we can specify the following SELECT operation:***

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

Figure 6.1 shows the result of SELECT operation.

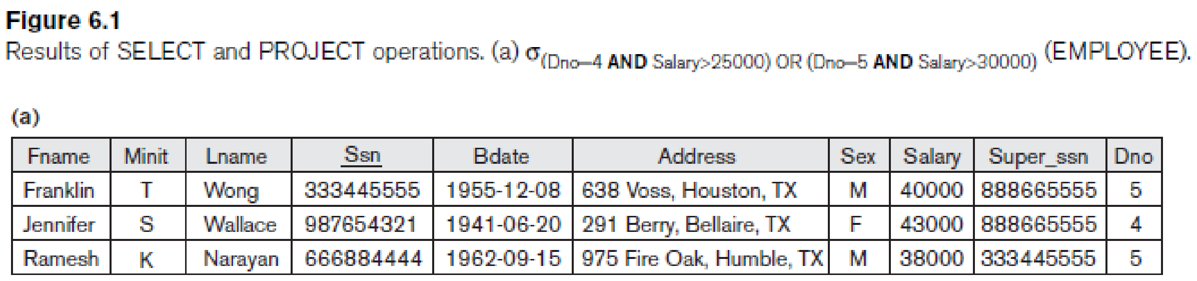


Figure 6.1
Results of SELECT and PROJECT operations. (a) $\sigma_{\text{(Dno=4 AND Salary>25000) OR (Dno=5 AND Salary>30000)}}$(EMPLOYEE).

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

- In SQL, the SELECT condition is typically specified in the WHERE clause of a query.

  For example, the following operation: σDno=4 AND Salary>25000 (EMPLOYEE) Would correspond to the following SQL query:

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

3.4.1.2. The PROJECT Operation

- The SELECT operation chooses some of the rowsfrom the table while discarding other rows. The **PROJECT** operation, on the otherhand, selects certain columns from the table and discards the other columns.

  Example 3: To list each employee's first and last name and salary, we can use the PROJECT operation as follows:

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

- The resulting relation is shown in Figure 6.1(b).

Figure 6.1
Results of SELECT and PROJECT operations.
(b) $\pi_{\text{Lname, Fname, Salary}}$(EMPLOYEE). (c) $\pi_{\text{Sex, Salary}}$(EMPLOYEE).

(b)

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

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

- The general form of the PROJECT operation is **π<attribute list>(R)** where π (pi) is the symbol used to represent the PROJECT operation, and <attributelist> is the desired sub list of attributes from the attributes of relation R
- The PROJECT operation removes any duplicate tuples, so 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:* $\pi_{\text{Sex, Salary}}$*(EMPLOYEE). In SQL, the PROJECT attribute list is specified in the SELECT clause of a query.*

SELECT DISTINCT Sex, Salary **FROM** EMPLOYEE

2.4.1.3. Sequences of Operations and the RENAME Operation

- 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.***

πFname, Lname, Salary (σDno=5(EMPLOYEE))

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

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

RESULT ← $\pi_{\text{Fname, Lname, Salary}}$ (DEP5_EMPS)

- To rename the attributes in a relation, list the new attribute names in parentheses, as in the following example:

TEMP ← σDno=5(EMPLOYEE)

R(First_name, Last_name, Salary) ← π$_{\text{Fname, Lname, Salary}}$(TEMP)

Figure 6.2
Results of a sequence of operations. (a) $\pi_{\text{Fname, Lname, Salary}}$ ($\sigma_{\text{Dno=5}}$(EMPLOYEE)). (b) Using intermediate relations and renaming of attributes.

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)
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

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

  ρS(B1, B2, ..., Bn)(R) or ρS(R) orρ(B1, B2, ..., Bn)(R)

  Where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name, and B1, B2, ..., Bn are the new attribute names.

SELECT E.Fname **AS** First_name, E.Lname **AS** Last_name, E.Salary **AS** Salary
FROM EMPLOYEE **AS** E **WHERE** E.Dno=5.

2.5 Relational Algebra Operations from Set Theory

2.5.1 The UNION, INTERSECTION, and MINUS Operations

- **For 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.***

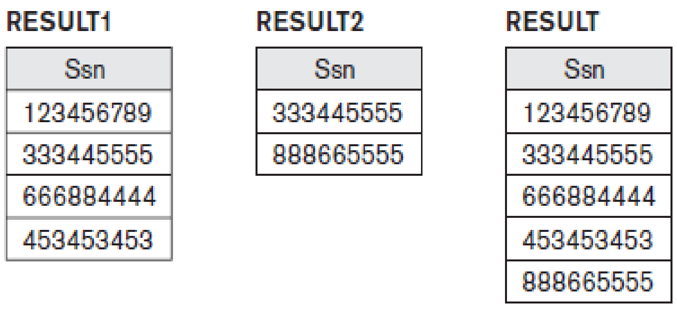
  DEP5_EMPS ← σDno=5(EMPLOYEE)

  RESULT1 ← πSsn(DEP5_EMPS)

RESULT2(Ssn) ← πSuper_ssn(DEP5_EMPS)

RESULT ← RESULT1 ∪ RESULT2

- The relation RESULT1 has the Ssn of all employees who work in department 5,where as RESULT2 has the Ssn of all employees who directly supervise an employee who works in department 5. The UNION operation produces the tuples that are in either RESULT1 or RESULT2 or both (see Figure 6.3), while eliminating any duplicates. Thus, the Ssn value '333445555' appears only once in the result.



RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

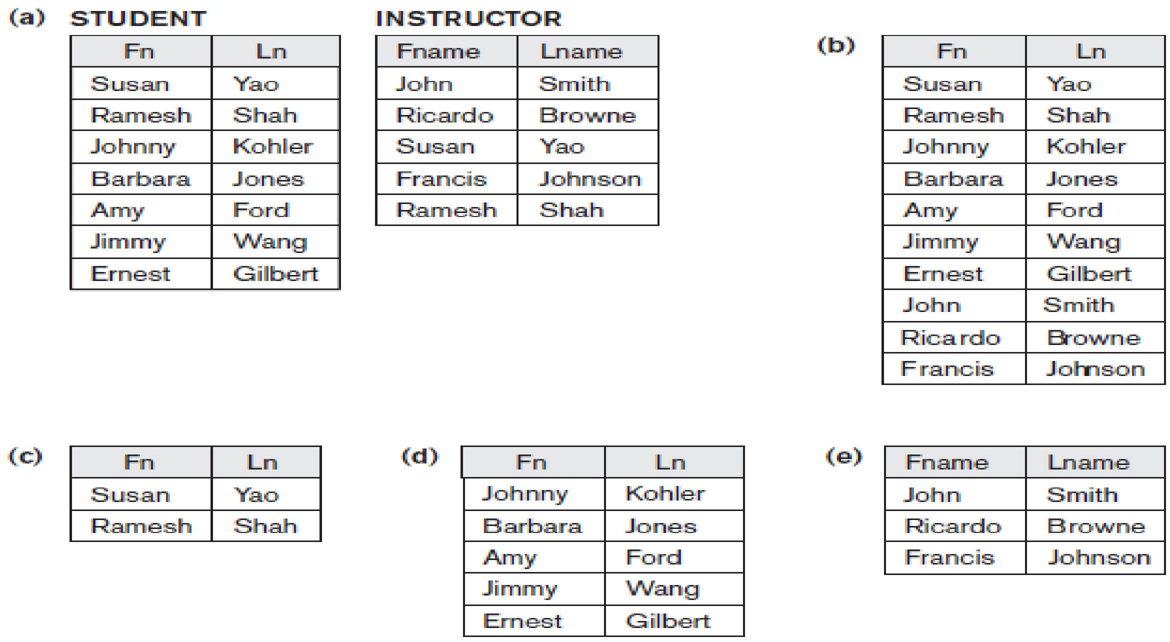
Ssn
123456789
333445555
666884444
453453453
888665555

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

We can define the **three operations UNION, INTERSECTION, and SET DIFFERENCE** on two union-compatible relations R and S as follows:

- UNION: The result of this operation, denoted by R∪S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.
- INTERSECTION: The result of this operation, denoted by R ∩S, is a relation that includes all tuples that are in both R and S.
- SET DIFFERENCE (or MINUS): The result of this operation, denoted byR – S, is a relation that includes all tuples that are in R but not in S.
- The relations STUDENT and INSTRUCTOR in Figure 6.4(a) are union compatible and their tuples represent the names of students and the names of instructors, respectively.
- The result of the UNION operation in Figure 6.4(b) shows the names of all students and instructors. Note that duplicate tuples appear only once in the result.
- The result of the INTERSECTION operation, Figure 6.4(c) includes only those who are both students and instructors.
- Figure 6.4(d) shows the names of students who are not instructors.
- Figure 6.4(e) shows the names of instructors who are not students.

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.



(a) 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

(b)

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

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

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

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

- UNION and INTERSECTION are commutative operations; that is,

  R ∪ S = S ∪ R and R ∩ S = S ∩ R

- 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 ∪ (S ∪ T) = (R ∪ S) V T and (R ∩ S) ∩ T = R ∩ (S ∩ T)

- The MINUS operation is not commutative; that is, in general, **R − S ≠ S – R**
- INTERSECTION can be expressed in terms of union and set difference as follows:

  R ∩ S = ((R ∪ S) − (R − S)) − (S − R).

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

2.5.2 The CARTESIAN PRODUCT (CROSS PRODUCT)Operation.

- The **CARTESIAN PRODUCT** operation—also known as **CROSSPRODUCT** or **CROSS JOIN**—which is denoted by **X.**
- Cartesian Product produces a new element by combining every member (tuple) of one relation (set) with every member (tuple) from the other relation (set).

- The result of R(A1, A2, ..., A_n) X S(B1, B2, ..., B_m) is a relation Q with degree n + m attributes Q(A1, A2, ..., An, B1, B2, ..., Bm).
- The CARTESIAN PRODUCT creates tuples with the combined attributes of two relations. We can SELECT related tuples from the two relations by specifying an appropriate selection condition after the Cartesian product,

  For example, suppose that we want to retrieve a list of names of each female employee's dependents.

FEMALE_EMPS ← σSex='F'(EMPLOYEE)

EMPNAMES← πFname, Lname, Ssn(FEMALE_EMPS)

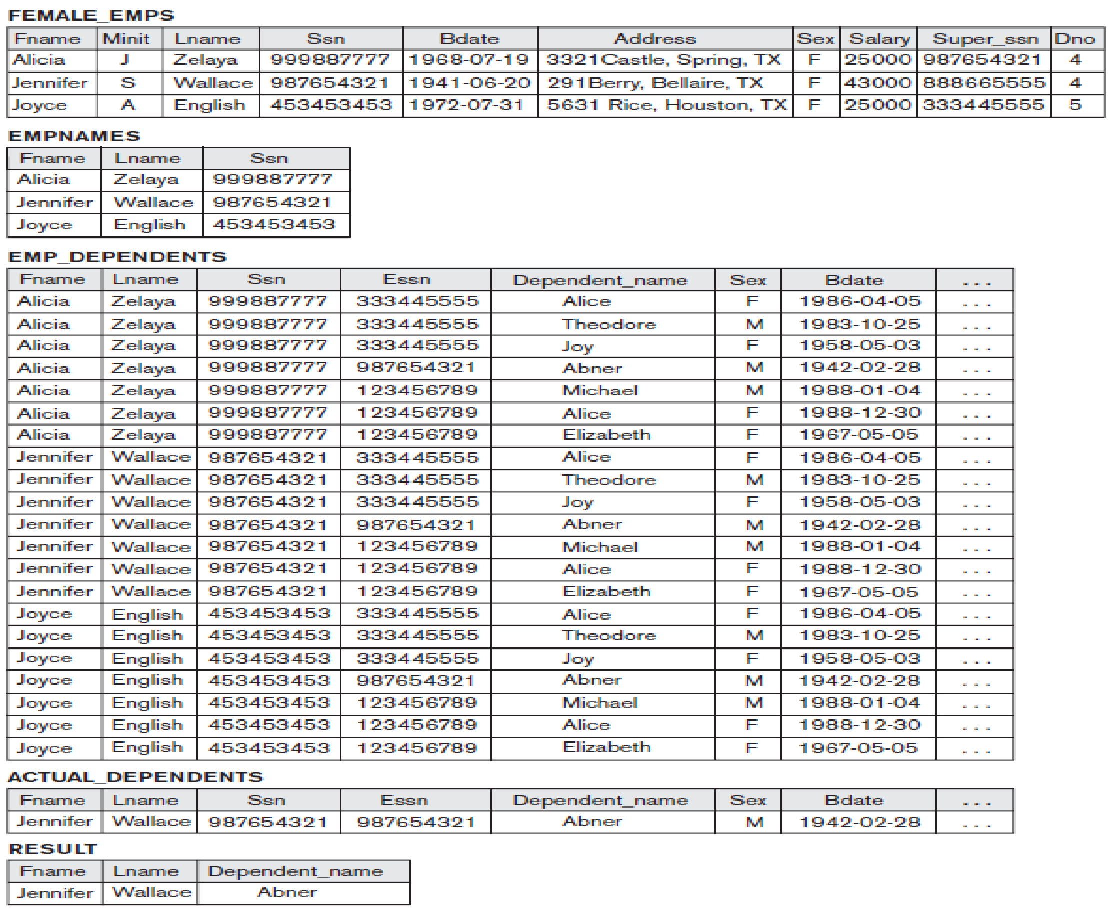
EMP_DEPENDENTS ← EMPNAMES × DEPENDENT

ACTUAL_DEPENDENTS ← σSsn=Essn(EMP_DEPENDENTS)

RESULT ← πFname, Lname, Dependent_name(ACTUAL_DEPENDENTS)

- The resulting relations from this sequence of operations are shown in Figure 6.5.

Figure 6.5
The Cartesian Product (Cross Product) operation.



FEMALE_EMPS

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

EMPNAMES

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

EMP_DEPENDENTS

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

ACTUAL_DEPENDENTS

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

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

2.6 Binary Relational Operations: JOIN and DIVISION

2.6.1 The JOIN Operation

- The JOIN operation, denoted by $\bowtie$, is used to combine related tuples from two relations into single “longer” tuples.
- **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 $R \bowtie_{<\text{join condition}>} S$.**
- 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.

To illustrate JOIN, suppose that we want to retrieve the nameof the manager of each department. To get the manager’s name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr_ssn value in the department tuple.

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

DEPT_MGR

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

Figure 6.6
Result of the JOIN operation $\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn}=\text{Ssn}} \text{EMPLOYEE}$.

Consider the earlier example illustrating CARTESIAN PRODUCT, which included the following sequence of operations:

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

- **These two operations can be replaced with a single JOIN operation as follows:**

  ACTUAL_DEPENDENTS ← EMPNAMES $\bowtie_{Ssn=Essn}$DEPENDENT

- **A general join condition is of the form** :

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

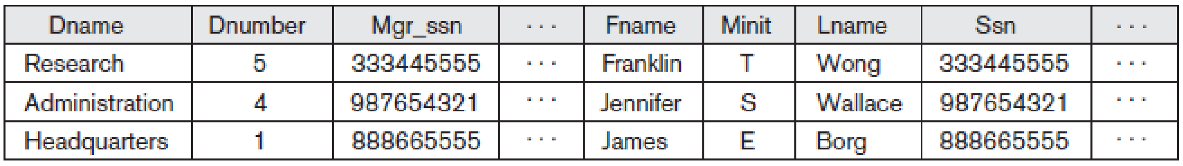
  where each <condition> is of the form Ai θ Bj, Ai is an attribute of R, Bj is an attribute of S, A JOIN operation with such a general join condition is called a **THETA JOIN.**

2.6.2 Variations of JOIN: The EQUI JOIN and NATURAL JOIN

- The JOIN operation where the only comparison operator used is = in join condition is called an **EQUIJOIN**.
- EQUIJOIN always have one or more pairs of attributes that have identical valuesin every tuple.

  For example, in Figure 6.6, the values of the attributes Mgr_ssn and Ssn are identical in every tuple of DEPT_MGR (the EQUIJOIN result) because the equality join condition specified on these two attributes requires the values to be identical in every tuple in the result.

DEPT_MGR



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	. . .

Figure 6.6
Result of the JOIN operation DEPT_MGR ← DEPARTMENT $\bowtie_{Mgr_ssn=Ssn}$EMPLOYEE.

- Because one of each pair of attributes with identical values is superfluous, a new operation called **NATURAL JOIN** denoted by * was created to get rid of the second (**superfluous**) attribute in an EQUIJOIN condition.
- The standard definition of NATURAL JOIN requires that the two 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. In the following example, 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:

PROJ_DEPT←PROJECT*ρ $_{\text{(Dname, Dnum, Mgr\_ssn, Mgr\_start\_date)}}$(DEPARTMENT)

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

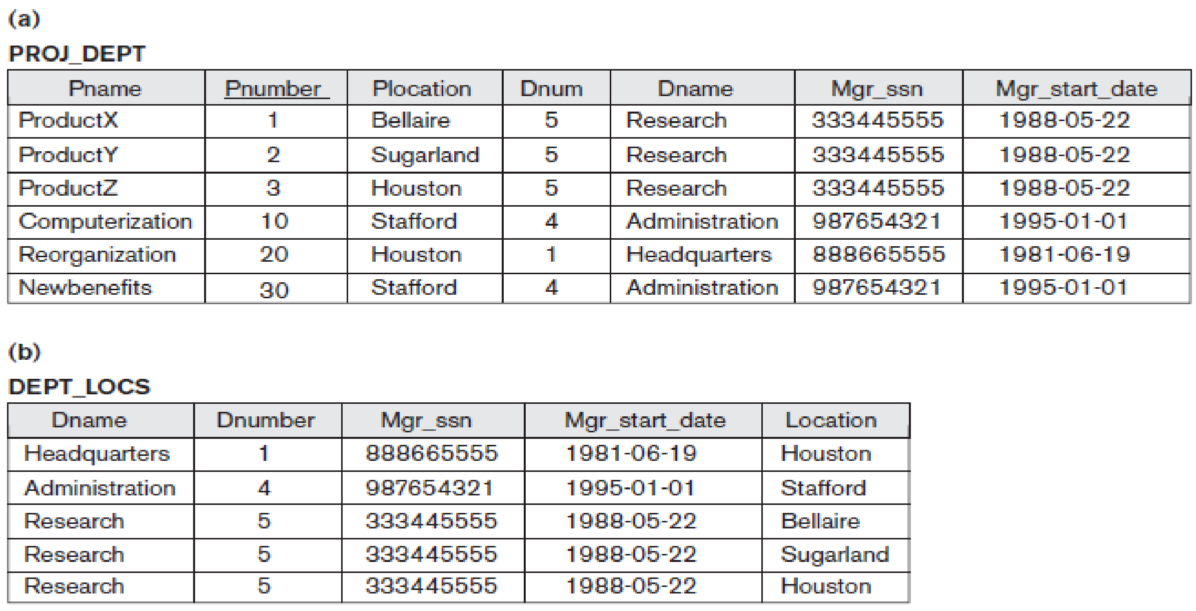
DEPT ←$\rho_{\text{(Dname, Dnum, Mgr\_ssn, Mgr\_start\_date)}}$(DEPARTMENT)

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. The resulting relation is illustrated in Figure 6.7(a).

- 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. The resulting relation is shown in Figure 6.7(b),***

DEPT_LOCS ←DEPARTMENT * DEPT_LOCATIONS



(a)
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

(b)
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

Figure 6.7
Results of two NATURAL JOIN operations. (a) PROJ_DEPT ← PROJECT * DEPT.
(b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

- 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.** An inner join is a type of match and combine operation defined formally as a combination of CARTESIAN PRODUCT and SELECTION.

2.6.3 The DIVISION Operation

- The division operator is an interesting operator that is useful in answering queries that involve "for all" statements.
- 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 works for 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:

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

- The preceding operations are shown in Figure 6.8(a).

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

(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

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(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

- In the formulation of the DIVISION operation, the tuples in the denominator relation S restrict the numerator relation R by selecting those tuples in the result that match all values present in the denominator.
- As illustrated in the SMITH_PNOS relation in the above example. Figure 6.8(b) illustrates a DIVISION operation where X = {A}, Y = {B}, and Z = {A,B}. Notice that the tuples (values) b1 and b4 appear in R in combination with all three tuples in S; that is why they appear in the resulting relation T. All other values of B in R do not appear with all the tuples in S and are not selected: b2 does not appear with a2, and b3 does not appear with a1.

List of the various basic relational algebra operations:

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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

2.7 Notation for Query Trees

- 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.
- Figure 6.9 shows a query tree for 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.

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

- In Figure 6.9, the three 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 ofexecution in the following sense. In order to execute Q2, the node marked (1) in Figure 6.9 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) canstart execution, and so on.

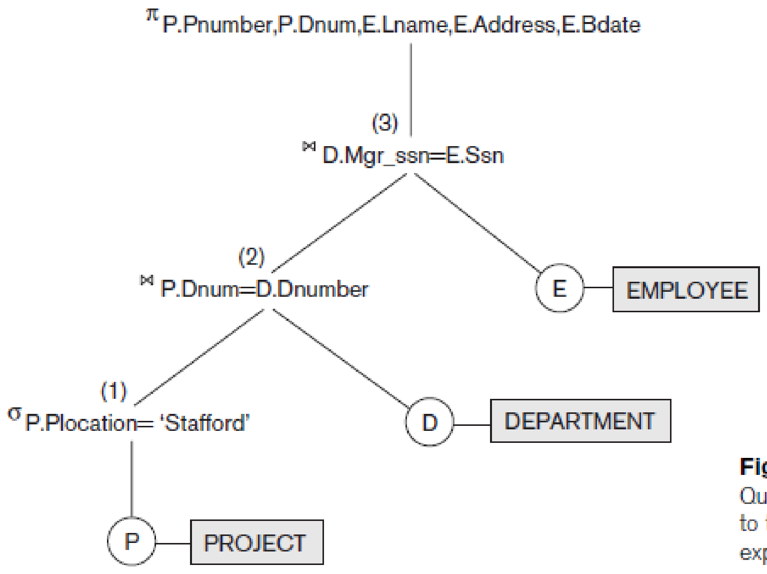


Figure 6.9
Query tree corresponding to the relational algebra expression for Q2.

2.8 Additional Relational Operations

2.8.1 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:$\boldsymbol{\pi}_{\mathbf{F1,\ F2,\ ...,\ Fn}}$ **(R)**where F1, F2, ..., Fn are functions over the attributes in relation R and may involve arithmetic operations and constant values.

  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.
- Then a generalized projection combined with renaming may be used as follows:

A ← π ($_{\textbf{Ssn, Salary – Deduction, 2000 * Years\_service,0.25 * Salary)}}$**(EMPLOYEE)).**

REPORT←ρ $_{\textbf{(Ssn, Net\_salary, Bonus, Tax)}}$ **(A).**

2.8.2 Aggregate Functions and Grouping

- Aggregate functions are used in simple statistical queries that summarize information from the database tuples. Some of the aggregate functions are **SUM, AVERAGE, MAXIMUM, and MINIMUM**. The **COUNT** function is used for counting tuples or values.
- AGGREGATE FUNCTION operation can be defined using the symbol ℑ(pronounced script F).

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

Where <**grouping attributes**> is a list of attributes of the relation specified in R,<**function list**> is a list of **(<function><attribute>)** pairs. In each such pair,<**function**> is one of the functions such as SUM, AVERAGE, MAXIMUM,MINIMUM, COUNT.

For example, to retrieve each department number, the number of employees in the department, and their average salary,

ρR(Dno,No_of_employees,Average_sal)(DnoℑCOUNTSsn,AVERAGESalary(EMPLOYEE))

- The result of this operation on the EMPLOYEE is shown in Figure 6.10(a).

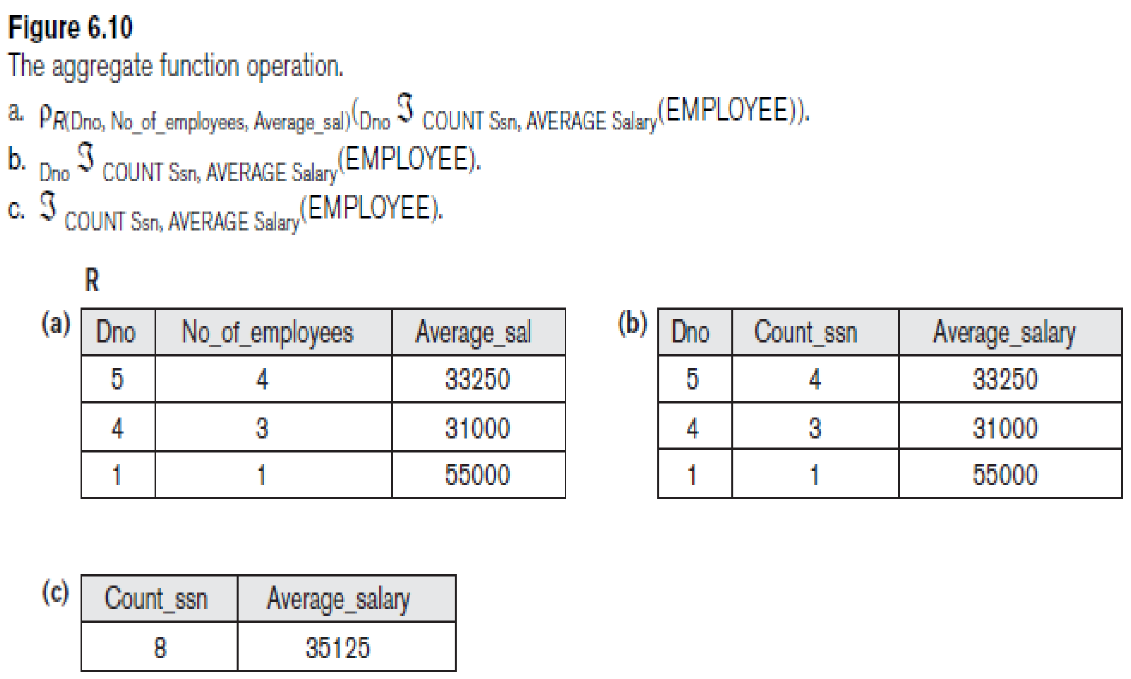


Figure 6.10
The aggregate function operation.

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

- If no **renaming** operation is applied, then the resulting relation will be in the form <**function_attribute**>. *For example, Figure 6.10(b) shows the result of the following operation:*

DnoℑCOUNT Ssn, AVERAGE Salary (EMPLOYEE)

- If no grouping attributes are specified, the functions are applied to all the tuples in the relation, so the resulting relation has a single tuple only. *For example, Figure6.10(c) shows the result of the following operation:*

ℑCOUNT Ssn, AVERAGE Salary(EMPLOYEE)

2.8.3 Recursive Closure Operations

- **Recursive closure** operation cannot be specified in the relational algebra**.** This operation is applied to a **recursive relationship**between tuples of the same type, such as the relationship between anemployee and a supervisor.

- This relationship is described by the foreign key Super_ssnof the EMPLOYEE relation.
- An example of a recursive operation is to retrieve all SUPERVISEES of an EMPLOYEE e at all levels—that is, all EMPLOYEE e' directly supervised by e; all employees e'' directly supervised by each employee e'; all employees e''' directly supervised by each employee e''; and so on

For example, To specify the Ssns of all employees directly supervised by James Borg' at level one.

$$\text{BORG\_SSN} \leftarrow \pi_{\text{Ssn}}(\sigma_{\text{Fname='James' AND Lname='Borg'}}(\text{EMPLOYEE}))$$

$$\text{SUPERVISION (Ssn1, Ssn2)} \longleftarrow \pi_{\text{Ssn,Super\_ssn}}(\text{EMPLOYEE})$$

$$\text{RESULT1(Ssn)} \longleftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}} \text{BORG\_SSN})$$

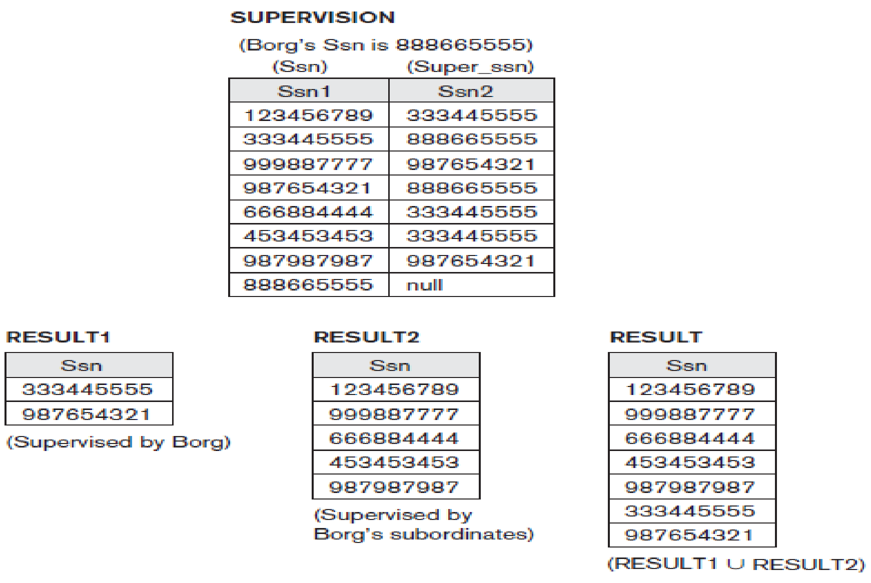
For example. To retrieve all employees supervised by Borg at level 2 is as follows:

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

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

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

- The results of these queries are illustrated in Figure 6.11.



SUPERVISION
(Borg's Ssn is 888665555)

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

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by Borg's subordinates)

RESULT

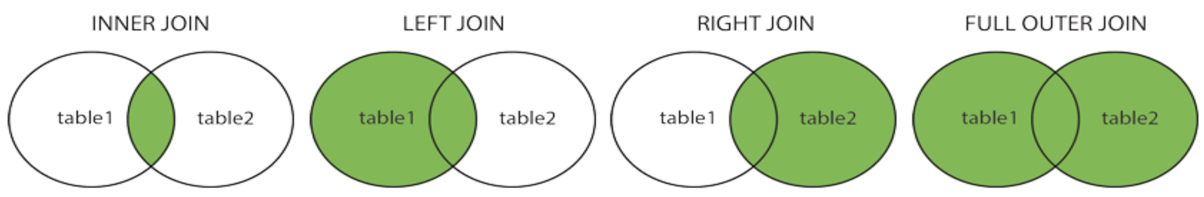
Ssn
123456789
999887777
666884444
453453453
987987987
333445555
987654321

(RESULT1 ∪ RESULT2)

Figure 6.11
A two-level recursive query.

2.8.4 OUTER JOIN Operations.

- 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 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**, is 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,
- There are three types of outer join operations**: left outer join, right outer join, and full outer join.**



Left Outer Join Operation

The left outer join is written as R ⟕ S where R and S are <u>relations</u>. The result of the left outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in R that have no matching tuples in 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

For an example considers the tables Employee and Dept and their left outer join:

Employee

Name	**EmpId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales

Dept

DeptName	**Manager**
Sales	Harriet
Production	Charles

Employee ⟕ Dept

Name	**EmpId**	**DeptName**	**Manager**
Harry	3415	Finance	NULL
Sally	2241	Sales	Harriet

George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

George	3401	Finance	NULL
Harriet	2202	Sales	Harriet
Tim	1123	Executive	NULL

Right Outer Join Operation

The right outer join of relations *R* and *S* is written as *R ⟖ S.* The result of the right outer join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names, in addition to tuples in *S* that have no matching tuples in *R*.

For example, consider the tables Employee and Dept and their right outer join:

Employee

Name	**EmpId**	**Dept Name**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	**Manager**
Sales	Harriet
Production	Charles

Employee ⟖ Dept

Name	**Emp Id**	**DeptName**	**Manager**
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
NULL	**NULL**	Production	Charles

- In the resulting relation, tuples in R which have no common values in common attribute names with tuples in S take a null value.

FULL Outer join Operation.

- The **outer join** or **full outer join** in effect combines the results of the left and right outer joins.
- The full outer join is written as $R ⟗ S$ where R and S are relations. The result of the full outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R and tuples in R that have no matching tuples in S in their common attribute names.

For an example considers the tables Employee and Dept and their full outer join:

Employee

Name	**EmpId**	**DeptName**
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	**Manager**
Sales	Harriet
Production	Charles

Employee ⟗ Dept

Name	**EmpId**	**DeptName**	**Manager**
Harry	3415	Finance	NULL
Sally	2241	Sales	Harriet
George	3401	Finance	NULL
Harriet	2202	Sales	Harriet
Tim	1123	Executive	NULL
NULL	NULL	Production	Charles

- In the resulting relation, tuples in R which have no common values in common attribute names with tuples in S take a *null* value. Tuples in S which have no common values in common attribute names with tuples in R also take a *null* value

2.8.5 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, areunion compatible.
- 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) and 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. The resulting relation, STUDENT_OR_INSTRUCTOR, will have the following attributes:

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

- 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.

2.9 Examples of Queries in Relational Algebra

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

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

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

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

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.

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

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.

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

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

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

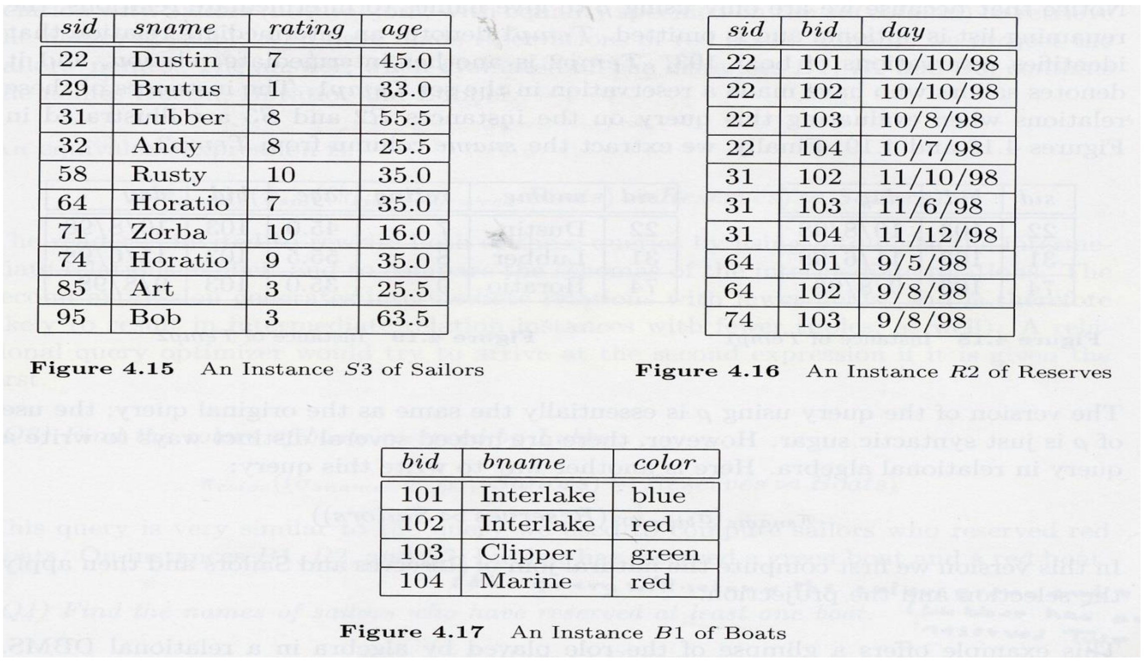
Query 6. Retrieve the names of employees who have no dependents.

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

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

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

Illustrate queries using the following new instances S3 of sailors, R2 of Reserves and B1 of boats.



sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance *S3* of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance *R2* of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance *B1* of Boats

(Q1) Find the names of sailors who have reserved boat 103

$$\pi_{sname}(\sigma_{bid=103}(\text{Reserves* Sailors}))$$

Q2) Find the names of sailors who have reserved a red boat.

$$\text{Sailres} \leftarrow \text{Sailors} * \text{Reserves}$$
$$\pi_{sname}((\sigma_{color='red'}\text{Boats}) * \text{Sailres})$$

(Q3) Find the colors of boats reserved by Lubber.

$$\text{Resboat} \leftarrow \text{Reserves} * \text{Boats}$$
$$\pi_{color}(\sigma_{sname='Lubber'}(\text{Sailors} * \text{Resboat}))$$

(Q4) Find the names of Sailors who have reserved at least one boat

$$\pi_{sname}(\text{Sailors} * \text{Reserves})$$

(Q5) Find the names of sailors who have reserved a red or a green boat.

$$\text{Tempboats} \leftarrow (\sigma_{color='red'}\text{Boats}) \cup (\sigma_{color='green'}\text{Boats}))$$
$$\pi_{sname}(\text{Tempboats}*\text{Reserves}*\text{Sailors})$$

Q6) Find the names of Sailors who have reserved a red and a green boat.

$$\text{Tempred} \leftarrow \pi_{sid}(\sigma_{color='red'}(\text{Boats}*\ \text{Reserves}))$$
$$\text{Tempgreen} \leftarrow \pi_{sid}(\sigma_{color='green'}(\text{Boats} * \text{Reserves}))$$
$$\pi_{sname}\ ((\text{Tempred} \cap \text{Tempgreen})\ *\text{Sailors})$$

Q7) Find the sids of sailors with age over 20 who have not reserved

$$\text{sailortwenty} \leftarrow \pi_{sid}(\sigma_{age>20}\text{Sailors})$$
$$\text{Result} \leftarrow \text{sailortwenty} - \text{sailredboat}$$

(Q8) Find the names of sailors who have reserved all boats.

$$\text{Tempsids} \leftarrow (\pi_{sid,bid}\text{Reserves})\ \div (\pi_{bid}\text{Boats}))$$
$$\text{Result} \leftarrow \pi_{sname}(\text{Tempsids}*\text{Sailors})$$

(Q9) Find the names of sailors who have reserved all boats called Interlake.

$$\text{Tempsids} \leftarrow (\pi_{sid,bid}\text{Reserves}) \div (\pi_{bid}(\sigma_{bname='Interlake'}\text{Boats}))$$
$$\text{Result} \leftarrow \pi_{sname}(\text{Tempsids}*\ \text{Sailors})$$

2.10 LOGICAL DATABASE DESIGN: ER TO RELATIONAL

The ER model is convenient for representing an initial, high level database design. The following steps show how to translate an ER diagram into a collection of tables with associated constraints.

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.

Steps to convert Entity Sets to Tables.

1. Create a table for the entity set.
2. Make each attribute of the entity set a field of the table, with an appropriate data type.
3. Declare the field or fields comprising the primary key

- Consider the Employees entity set with attributes ssn, name, and lot.

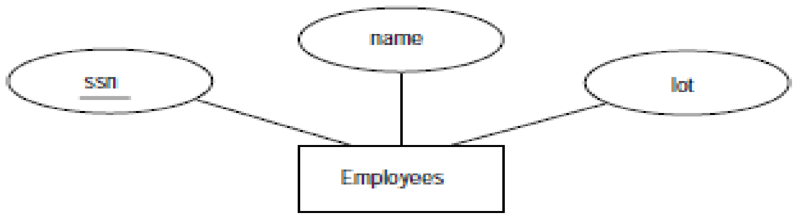


Figure 3.8 The Employees Entity Set

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

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 attributes of the relationship.

Steps to convert Relationship sets to tables

1. Create a table for the relationship set.
2. Add all primary keys of the participating entity sets as fields of the table.

3. Add a field for each attribute of the relationship.
4. Declare a primary key using all key fields from the entity sets.
5. Declare foreign key constraints for all these fields from the entity sets.

- 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

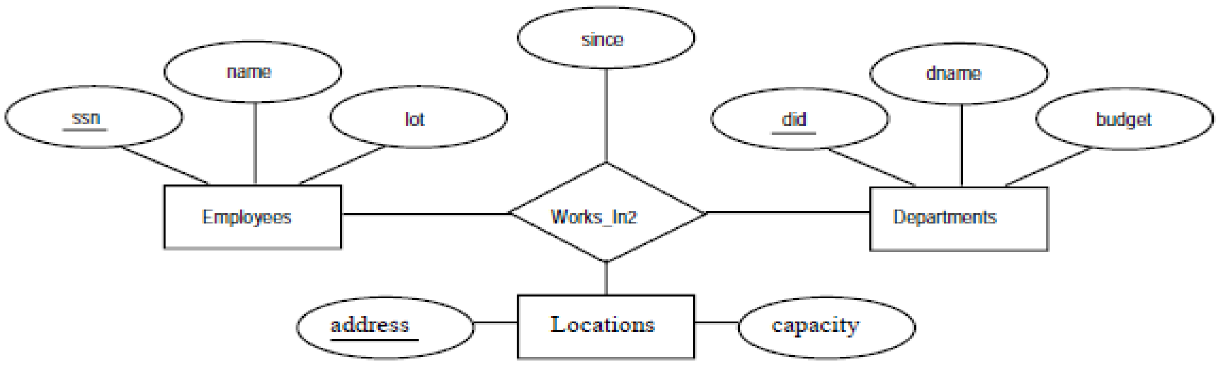


Figure 3.10 A Ternary Relationship Set

- All the available information about the Works-ln2 table is captured by the following SQL definition:

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

- The address, did. and ssn fields cannot take on null values. Because these fields are part of the primary key for Works_In2, a NOT NULL constraint is implicit for each of these fields. This constraint ensures that these fields uniquely identify a department, an employee, and a location in each tuple of Works_In.

3. Translating Relationship Sets with Key Constraints.

- If a relationship set involves n entity sets and some m of them are linked via arrows in the ER diagram, the key of these m entity sets constitutes a key for the relation to which the relationship set is mapped. Hence we have m candidate keys, and one of these should be designated as the primary key.

- Consider the relationship set Manages. The table corresponding to Manages has the attributes ssn, did, since. However, because each department has at most one manager, no two tuples can have the same did value but differ on the ssn value. A consequence of this observation is that did is itself a key for Manages; indeed, the set did, ssn is not a key (because it is not minimal).
- The Manages relation can be defined using the following SQL statement:

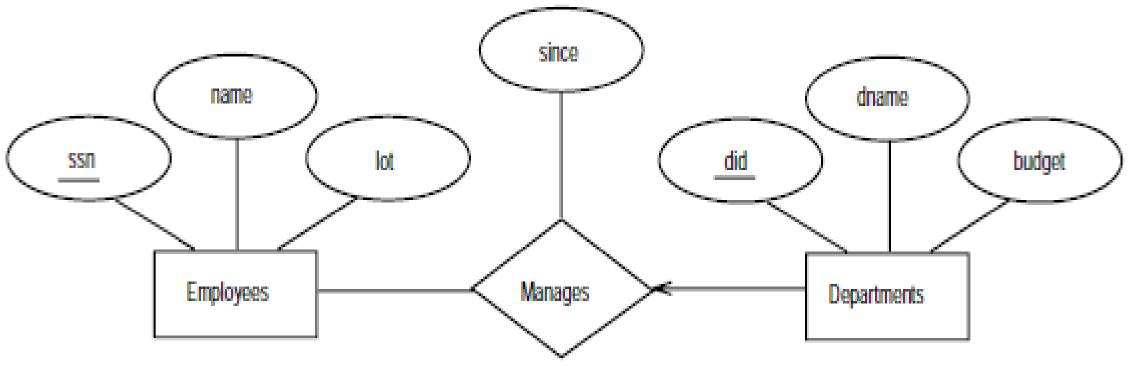


Figure 3.12 Key Constraint on Manages

```
CREATE TABLE Manages ( ssn      CHAR(11),
                       did      INTEGER,
                       since    DATE,
                       PRIMARY KEY (did),
                       FOREIGN KEY (ssn) REFERENCES Employees,
                       FOREIGN KEY (did) REFERENCES Departments )
```

Steps to translate Translating Relationship Sets with Key Constraints to tables.

1. Create a table for the relationship set.
2. Add all primary keys of the participating entity sets as fields of the table.
3. Add a field for each attribute of the relationship.
4. Declare a primary key using the key fields from the source entity set only.
5. Declare foreign key constraints for all the fields from the source and target entity sets.

- The following SQL statement, defining a Dept_Mgr relation that captures the information in both Departments and Manages, illustrates the second approach to translating relationship sets with key constraints. The only drawback to this approach is that space could be wasted if several departments have no managers. In this case the added fields would have to be filled with *null* values

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

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.

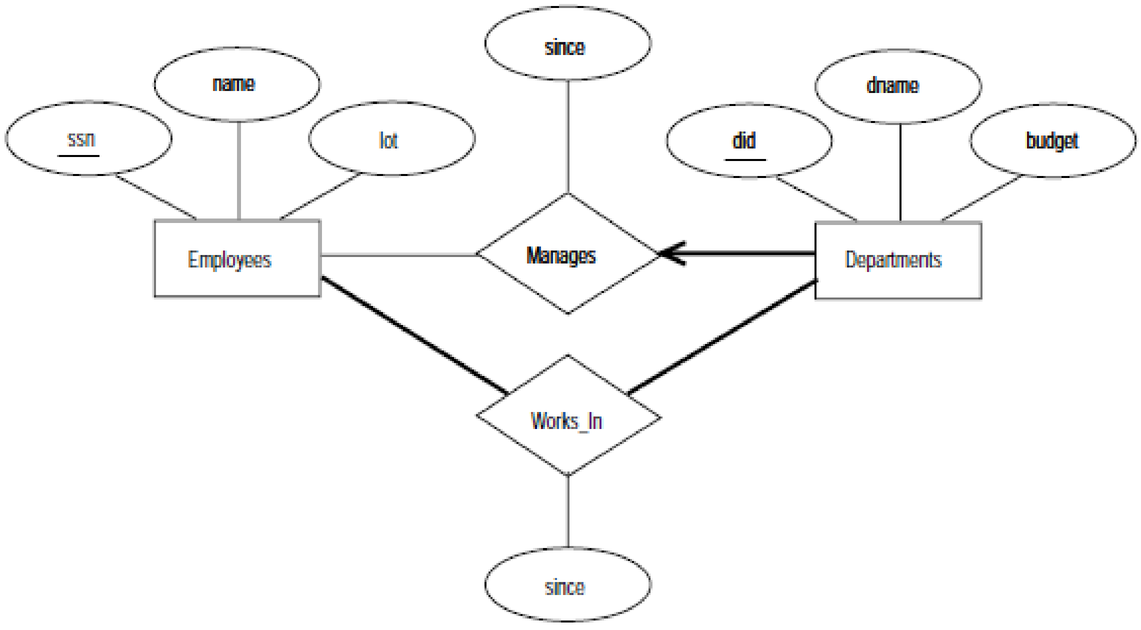


Figure 3.13 Manages and Works_In

Steps to translate relationship sets with participation constraints

1. Create a table for the source and target entity sets as usual.
2. Add every primary key field of the target as a field in the source.

3. Declare these fields as not null.
4. Declare these fields as foreign keys.

```
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 )
```

- According to participation constraint every department must have a manager: because ssn cannot take on null values, each tuple of Dept_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. If we wish to delete such an Employees tuple, we must first change the Dept_Mgr tuple to have a new employee as manager.

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.
- The weak entity has only a partial key, when an owner entity is deleted, all the weak entities should be deleted.
- Consider the Dependents weak entity set shown in Figure 3.14, with partial key pname. A Dependents entity can be identified uniquely by 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.

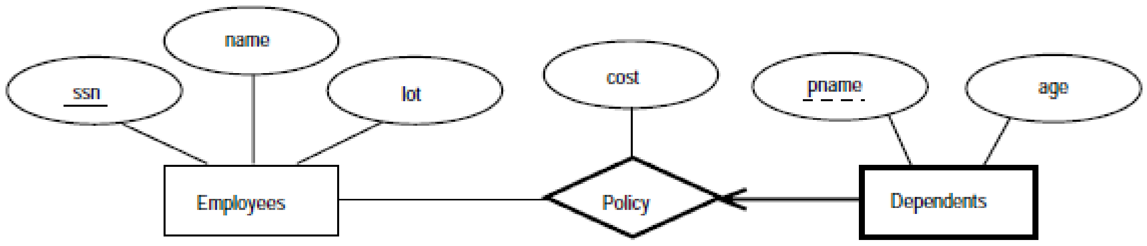


Figure 3.14 The Dependents Weak Entity Set

```
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 )
```

Steps to translate weak entity set to tables.

1. Create a table for the weak entity set.
2. Make each attribute of the weak entity set a field of the table.
3. Add fields for the primary key attributes of the identifying owner.
4. Declare a foreign key constraint on these identifying owner fields.
5. Instruct the system to automatically delete any tuples in the table for which there are no owners.

- The primary key is (pname, ssn), since Dependents is a weak entity. The value of 'ssn' cannot be null because the Dependents entity is associated with an Employees entity (the owner), as per the total participation constraint on Dependents.
- The ON DELETE CASCADE option ensures that information about an employee's policy and dependents is deleted if the corresponding Employees tuple is deleted.

6 Translating class Hierarchies.

The two basic approaches to handle hierarchies by applying them to the ER diagram are:

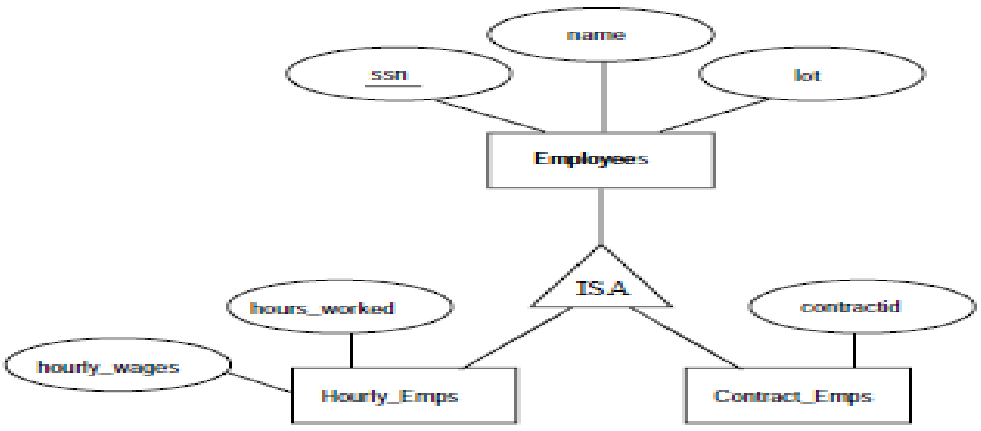


Figure 3.15 Class Hierarchy

- Map each entity sets Employees, Hourly_Emps, and Contract_Emps to a distinct relation. 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.
- Alternatively, two relations can be created, corresponding to Hourly_Emps and ContractEmps. The relation for Hourly_mps includes all the attributes of Hourly_Emps as well as all the attributes of Employees (i.e., ssn, name, lot, hourly_ wages, hours_worked:).

7 Translating ER Diagrams with Aggregation

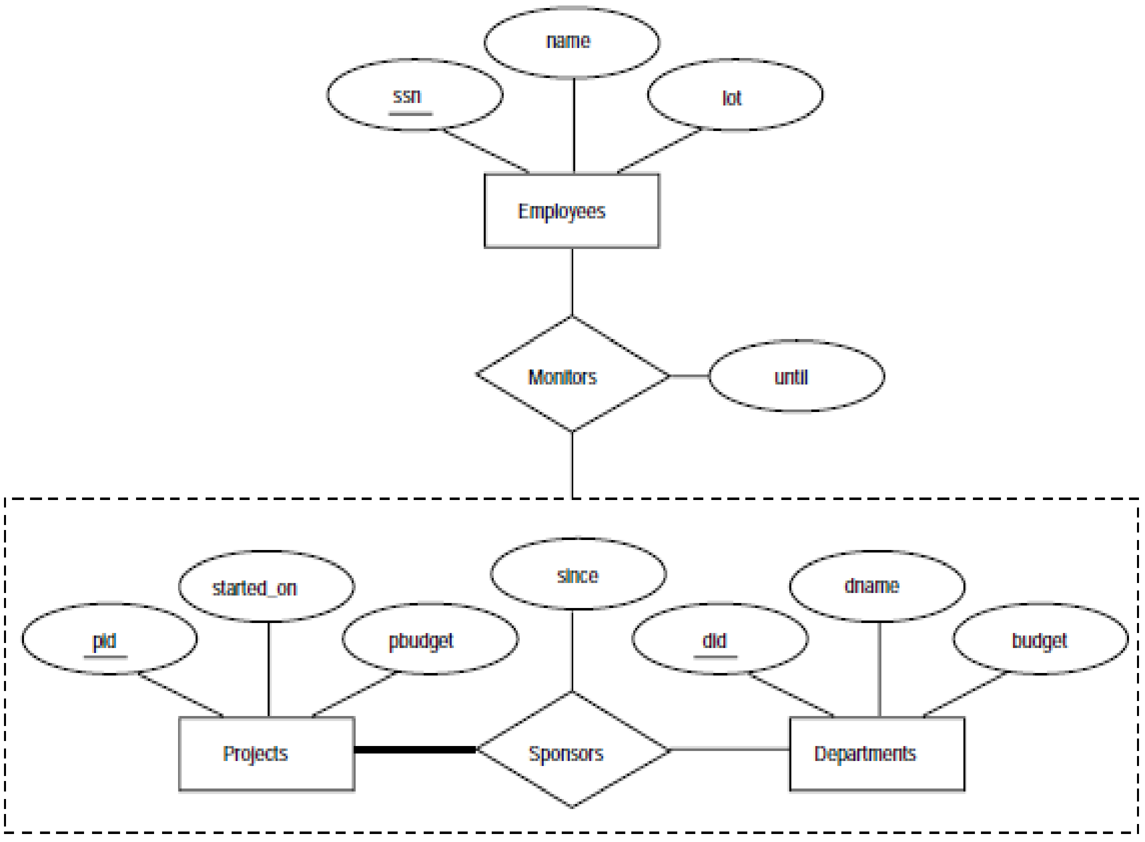


Figure 3.16 Aggregation

- The Employees, Projects, and Departments entity sets and the Sponsors relationship set are mapped to relations. For the Monitors relationship set, we create a relation with the following attributes: the key attributes of Employees (ssn), the key attributes of Sponsors (did, pid), and the descriptive attributes of Monitors (until).

- Consider the Sponsors relation. It has attributes pid, did, and since; and in general we need it (in addition to Monitors) for two reasons:
    1. Record the descriptive attributes (since) of the Sponsors relationship.
    2. Not every sponsorship has a monitor, and thus some (pid, did) pairs in the Sponsors relation may not appear in the Monitors relation.

2.11 Basic SQL

2.11.1 SQL Data Definition and Data Types

1 Schema in SQL

- 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.

For example, the following statement creates a schema called COMPANY, owned by the user with authorization identifier 'Jsmith'.

CREATE SCHEMA COMPANY **AUTHORIZATION** 'Jsmith';

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.

CREATE TABLE COMPANY.EMPLOYEE
rather than
CREATE TABLE EMPLOYEE

3 Attribute Data Types and Domains in SQL

1. Numeric :

- **The** data types include integer numbers of various sizes INTEGER or INT, and SMALLINT
- Floating-point numbers of various precision FLOAT or REAL.
- 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.

2. Character-string

- The data types with fixed length characters—CHAR(n) or CHARACTER(n), where n is the number of characters
- The data types with varying length characters VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.
- 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. Padded blanks are generally ignored when strings are compared.
- Another variable-length string data type called CHARACTER LARGE OBJECT or CLOB is also available to specify columns that have large text values, such as documents. The CLOB maximum length can be specified in kilobytes (K), megabytes (M), or gigabytes (G). For example, CLOB(20M) specifies a maximum length of 20 megabytes

3. Bit-string

- The data types of fixed length n is BIT(n).
- The varying length is BIT VARYING(n), where n is the maximum number of bits.
- Literal bit strings are placed between single quotes but preceded by a B to distinguish them from character strings; for example, B'10101'.
- Another variable-length bitstring data type called BINARY LARGE OBJECT or BLOB is also available to specify columns that have large binary values, such as images

4. **Boolean**

   The 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.

5. **DATE**

   - The 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. for example, DATE '2008-09- 27' or TIME '09:12:47'

6. **Timestamp**

   - **The** data type (TIMESTAMP) includes the DATE and TIME fields,
   - Example, TIMESTAMP '2008-09-27 09:12:47.648302'.

2.11.2 Specifying Constraints in SQL

1 Specifying Attribute Constraints and Attribute Defaults

- SQL allows NULLs as attribute values, a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute.
- It is also possible to define a default value for an attribute by appending the clause **DEFAULT** <value> to an attribute definition.

  Example: Mgr_ssn CHAR (9) NOT NULL DEFAULT '888665555',
- The default value is included in any new tuple if an explicit value is not provided for that attribute.
- If no default clause is specified, the default default value is NULL for attributes that do not have the NOT NULL constraint.
- Another type of constraint that can restrict attribute or domain values is by 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);

The CHECK clause can also be used in conjunction with the CREATE DOMAIN statement.For example,

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

2 Specifying Key and Referential Integrity Constraints

- The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain unique values, and cannot contain NULL values. A table can have only one primary key, which may consist of single or multiple fields.

  For example, the primary key of DEPARTMENT can be specified as

  ```
  Dnumber  INT  PRIMARY KEY;
  ```

- The **UNIQUE** clause specifies alternate (secondary) keys, The UNIQUE constraint ensures that all values in a column are different. Both the UNIQUE and PRIMARY KEY constraints are similar .A table can have many UNIQUE constraints , but only one PRIMARY KEY constraint per table.
- The **UNIQUE** clause can also be specified directly for a secondary key if the secondary key is a single attribute, as in the following ***example: Dname VARCHAR(15) UNIQUE;***

  Example:

```
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)
);
```

- Referential integrity is specified via the **FOREIGN KEY.**
- A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.
- 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.
- 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**.

FOREIGN KEY (Super_ssn) **REFERENCES** EMPLOYEE(Ssn) **ON DELETE** SET NULL **ON UPDATE** CASCADE.

- 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_ssn is automatically set to NULL for all employee tuples that were referencing the deleted employee tuple.
- On the other hand, if the Ssn value for a supervising employee is updated the new value is cascaded to Super_ssn for all employee tuples referencing the updated employee tuple.
- In general, the action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE: The value of the affected referencing attributes is changed to NULL for SET NULL and to the specified default value of the referencing attribute for SET DEFAULT. The action for CASCADE ON DELETE is to delete all the referencing tuples, whereas the action for CASCADE ON UPDATE is to change the value of the referencing foreign key attribute(s) to the updated (new) primary key value for all the referencing tuples.

3 Giving Names to Constraints

- 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,

Example: CONSTRAINT DEPTMGRFK FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE (Ssn)

DEPTMGRFK is a constraint name

4 Specifying Constraints on Tuples Using CHECK

- 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.

For example, suppose that the DEPARTMENT table has 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);

Note : SQL Comparison Operators

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.

Note : SQL Logical Operators

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list as per the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.

The following table has a few examples showing the WHERE part having different LIKE clause with **'%'** and **'_'** operators:

Statement	**Description**
WHERE SALARY LIKE '200%'	Finds any values that start with 200.
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position.
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions.
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length.
WHERE SALARY LIKE '%2'	Finds any values that end with 2.
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3.
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3.

2.12 Basic Retrieval Queries in SQL

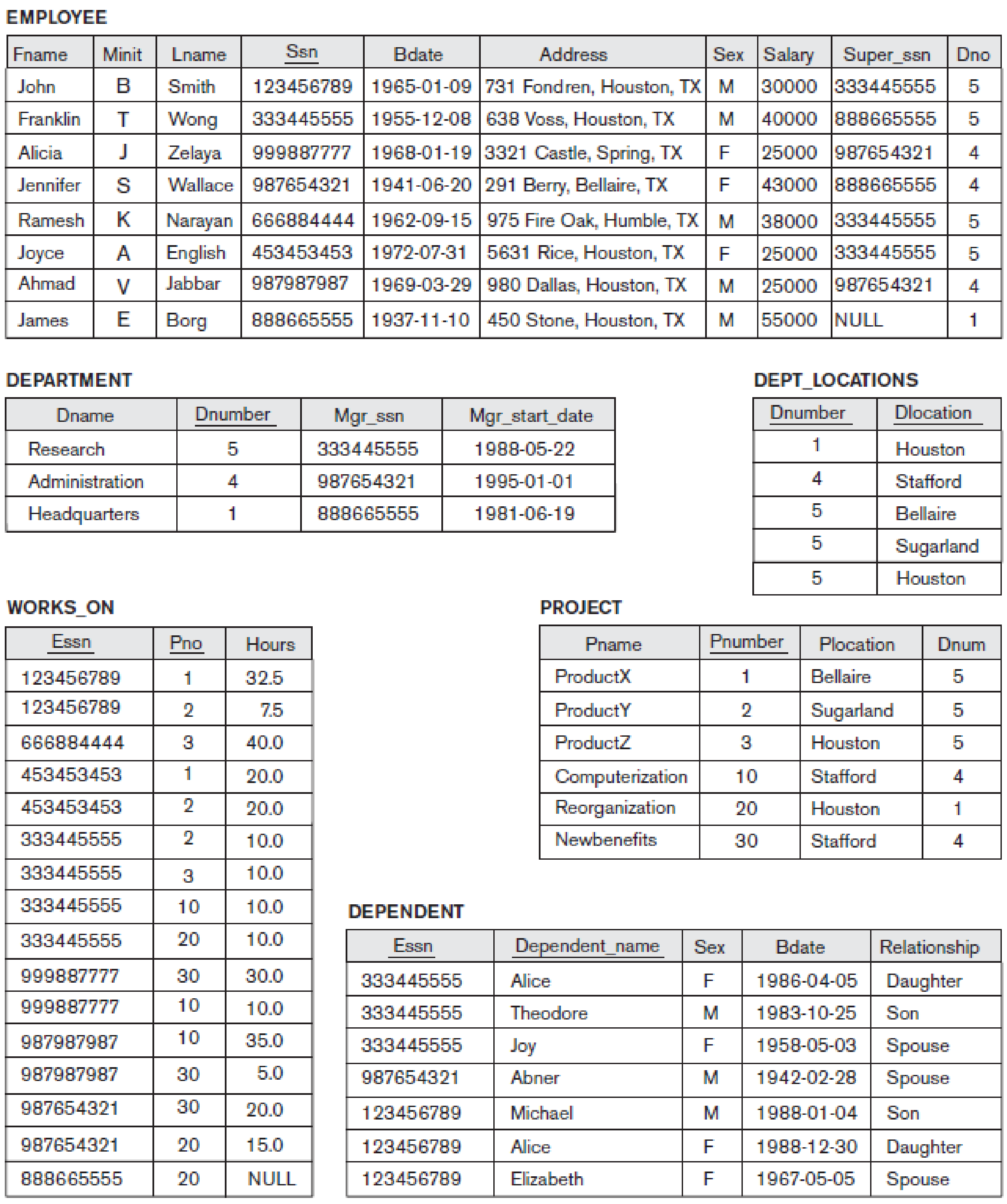
1 The SELECT-FROM-WHERE Structure of Basic SQL Queries

- The basic form of the SELECT statement is formed of the three clauses SELECT, FROM, and WHERE .

  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 <>. These correspond to the relational algebra operators =, <, ≤, >, ≥, and ≠, respectively,



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

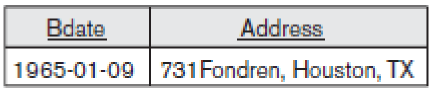
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

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

Query 1. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

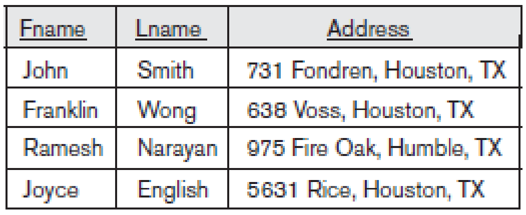
```
SELECT    Bdate, Addres   FROM      EMPLOYEE
WHERE     Fname='John' AND Minit='B' AND Lname='Smith';
```



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

Query 2. Retrieve the name and address of all employees who work for the 'Research' department.

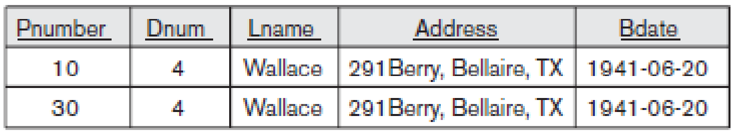
```
SELECT    Fname, Lname, Address FROM EMPLOYEE, DEPARTMENT
WHERE     Dname='Research' AND Dnumber=Dno;
```



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

Query 3. 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  num=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford';
```



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

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

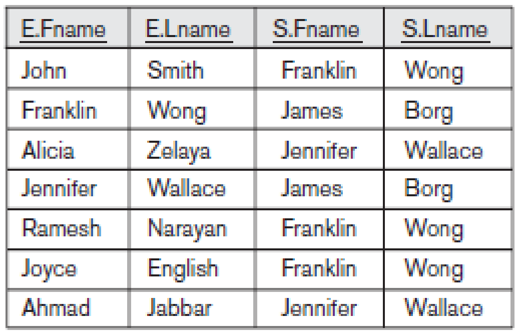
- In SQL, the same name can be used for two (or more) attributes as long as the attributes are in *different relations.*

- When 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.
- To illustrate consider the Dno and Lname attributes of the EMPLOYEE relation were called Dnumber and Name, and the Dname attribute of DEPARTMENT was also called Name; then, to prevent ambiguity we need to prefix the attribute with relation name.

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

- 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;
```



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

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.

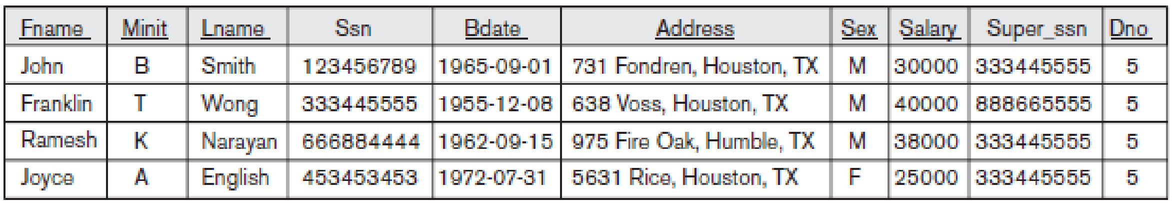
- For example, ***Query 9 selects all EMPLOYEE Ssns (Figure 4.3(e)), and Query 10 selects all combinations of an EMPLOYEE Ssn and a DEPARTMENT Dname, regardless of whether the employee works for the department or not***

```
(Q9)   SELECT   Ssn  FROM      EMPLOYEE;

(Q10)  SELECT Ssn, Dname FROM EMPLOYEE, DEPARTMENT;
```

- To retrieve all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; we just specify an asterisk (*), which stands for all the attributes. ***For example, query Q1C retrieves all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5.***

```
Q1C:   SELECT   * FROM     EMPLOYEE
                WHERE    Dno=5;
```



Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-09-01	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

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, for the following reasons:
  1. Duplicate elimination is an expensive operation. One way to implement it is to sort the tuples first and then eliminate duplicates.
  2. The user may want to see duplicate tuples in the result of a query.
  3. When an aggregate function is applied to tuples, in most cases we do not want to eliminate duplicates.
- 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.

- In general, a query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not. Do not specify SELECT with neither ALL nor DISTINCT as in our previous examples is equivalent to SELECT ALL.

  For example, Q11 retrieves the salary of every employee; if several employees have the same salary, that salary value will appear as many times in the result of the query,

  Q11: **SELECT ALL** Salary **FROM** EMPLOYEE;

Query 11. Retrieve the salary of every employee (Q11) and all distinct salary values (Q11A).

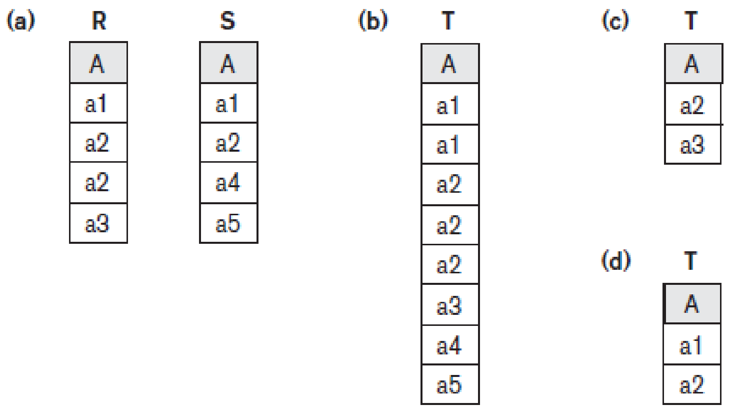
Q11A: **SELECT DISTINCT** Salary **FROM** EMPLOYEE;

- SQL has some of the set operations, They 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.

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'
);
```

- SQL also has corresponding multiset operations, which are followed by the keyword **ALL** (UNION ALL, EXCEPT ALL, INTERSECT ALL). Their results are multisets (duplicates are not eliminated).
- The behavior of these operations is illustrated by the examples in Figure 4.5.



(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).

5 Substring Pattern Matching and Arithmetic Operators

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards used in conjunction with the LIKE operator:
  1. **% The percent sign represents zero, one, or multiple characters**
  2. **_ The underscore represents a single character.**

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 _ _ _ _ _ _ _' ;
```

- If an underscore or % is needed as a literal character in the string, the character should be preceded by an escape character('\'), which is specified after the string . For example, 'AB_CD\%EF' is used to represent the literal string 'AB_CD%EF' .
- The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains. For example, suppose that we want to see the effect of giving all employees who work on the 'ProductX' project a 10 percent raise;

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';
```

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;
```

6 Ordering of Query Results

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword. The keyword **ASC** can be used to specify ascending order.

Query 15. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

```
SELECT   D.Dname, E.Lname, E.Fname, P.Pname
FROM     DEPARTMENT D, EMPLOYEE E, WORKS_ON W, PROJECT P
WHERE    D.Dnumber= E.Dno AND E.Ssn= W.Essn AND W.Pno= P.Pnumber
ORDER BY D.Dname, E.Lname, E.Fname;
```

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> ];
```

2.13 INSERT, DELETE, and UPDATE Statements in SQL

In SQL, three commands can be used to modify the database: INSERT, DELETE, and UPDATE.

1 The INSERT Command

- INSERT is used to add a single tuple to a relation. We must specify the relation name and a list of values for the tuple.
- While adding values for all the columns of the table, need not specify the column names in the SQL query The values should be listed *in the same order* in which the corresponding attributes were specified in the CREATE TABLE command.
- The INSERT INTO syntax would be as follows:

INSERT INTO *table_name* VALUES (*value1*, *value2*, *value3*, ...);

For example, to add a new tuple to the EMPLOYEE relation shown.

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

- The second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command.

  For example, to enter a tuple for a new EMPLOYEE for whom we know only the Fname, Lname, Dno, and Ssn attributes.

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

Example:

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

2 The DELETE Command

- The DELETE command removes tuples from a relation. The WHERE clause, selects the tuples to be deleted. Tuples are explicitly deleted from only one table at a time.
- The deletion propagates to tuples in other relations if *referential triggered actions(on delete cascade)* are specified in the referential integrity constraints of the DDL.
- 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. The DROP TABLE command to remove the table definition**.**

Example1:

```
DELETE FROM EMPLOYEE
WHERE Lname=‘Brown’;
```

Example 2

```
DELETE FROM EMPLOYEE
```

3 The UPDATE Command

- The **UPDATE** command is used to modify attribute values of one or more selected Tuples, the WHERE clause in the UPDATE command selects the tuples to be modified from a single relation.
- Updating a primary key value may propagate to the foreign key values of tuples in other relations if such a *referential triggered action(on update cascade)* 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.

```
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;
```

2.14 Additional Features of SQL

- SQL has various techniques for specifying complex retrieval queries, including nested queries, aggregate functions, grouping, joined tables, outer joins, 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)..
- SQL has set of commands for specifying physical database design parameters, file structures for relations, and access paths such as indexes. We called these commands a *storage definition language* (*SDL*).
- 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.

2.15 References:

Text Books

1. **Fundamentals of Database System, Elmasri and Navathe , 7th edition.**
2. **Database Management System, Raghu Ramakrishnan and Johannes Gehrke, 3rd edition.**

Web Resource

1. **https://www.geeksforgeeks.org/introduction-of-er-model/**
2. **https://beginnersbook.com/2015/04/e-r-model-in-dbms/**
3. **https://www.tutorialspoint.com/dbms/er_model_basic_concepts.htm**
4. **https://www.gatevidyalay.com/relationship-sets/**
5. **https://tutorialwing.com/mapping-constraints-in-dbms-for-relationship-types/**