MODULE 2

Relational Model: Relational Model Concepts, Relational Model Constraints and relational database schemas, Update operations, transactions, and dealing with constraint violations.

Relational Algebra: Unary and Binary relational operations, additional relational operations (aggregate, grouping, etc.) Examples of Queries in relational algebra.

Mapping Conceptual Design into a Logical Design: Relational Database Design using ER-to-Relational mapping.

SQL:

SQL data definition and data types, specifying constraints in SQL, retrieval queries in SQL, INSERT, DELETE, and UPDATE statements in SQL, Additional features of SQL.

Relational Model:

2.1 Relational Model Concepts

- ✓ The relational model represents the database as a collection of *relations*.

Informally, each relation resembles a table of values or, to some extent, a flat file of records

- ✓ 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 →a tuple, a column header →an attribute, and the table →a relation. The data type describing the types of values that can appear in each column is represented by a domain of possible values.

Domains, Attributes, Tuples, and Relations

- ✓ A domain D is a set of **atomic values.** By atomic means each value in the domain is **indivisible** in formal relational model. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
- ✓ Some examples of domains follow:

USA_phone_number: string of digits of length ten
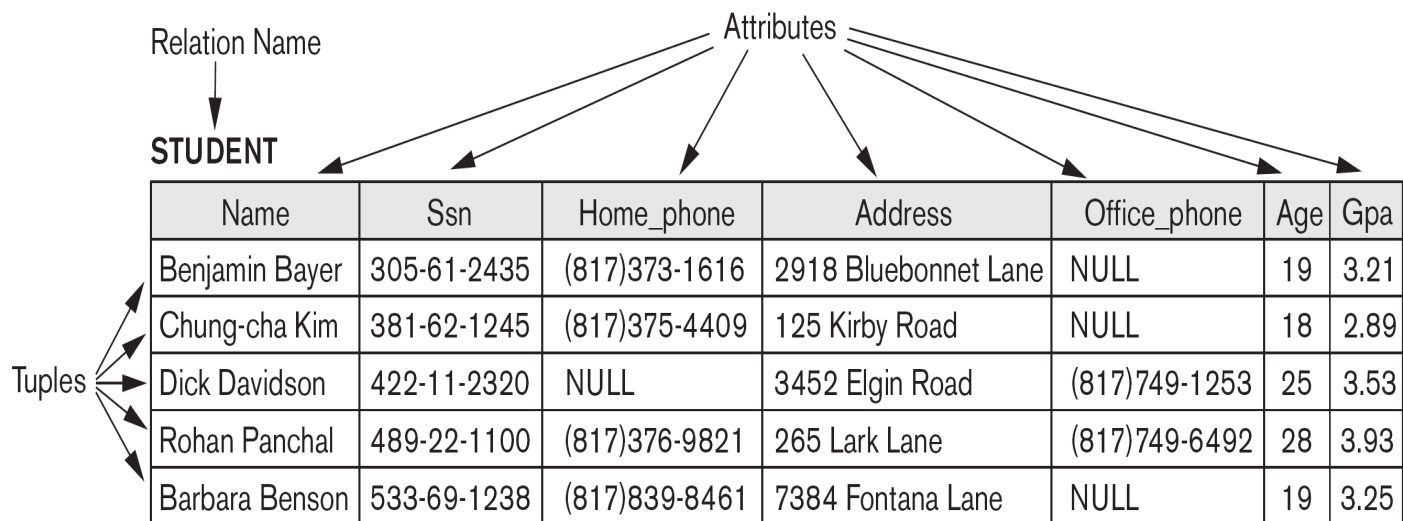SSN: string of digits of length nine
Name: string of characters beginning with an upper case letter
GPA: a real number between 0.0 and 4.0
Sex: a member of the set { female, male }
Dept_Code: a member of the set { CMPS, MATH, ENGL, PHYS, PSYC, ... }

- ✓ A **relation schema R**, denoted by R(A1, A2, … , An), is made up of a relation name R and a list of attributes, A1, A2, … , An.
- ✓ **Attribute:** Ai is the name of a role played by some domain D in the relation schema R. D is called the domain of Ai and is denoted by dom(Ai).
- ✓ **Tuple**: A tuple is a mapping from attributes to values drawn from the respective domains of those attributes. A tuple is intended to describe some entity (or relationship between entities) in the miniworld.
- ✓ R is called the name of this relation.
- ✓ The **degree (or arity) of a relation** is the number of attributes n of its relation schema.
- ✓ A relation of degree seven, which stores information about university students, would contain seven attributes describing each student as follows:
  STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)
- ✓ **Relational Database**: A collection of **relations**, each one consistent with its specified relational schema.
- ✓ 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 values t =<$v_1, v_2 \ldots, v_n$>

| STUDENT | | | | | | |
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

Characteristics of Relations

1. **Ordering of Tuples in a Relation**
   - ✓ A relation is defined as a set of tuples. Mathematically, elements of a set have no order among them; hence, tuples in a relation do not have any particular order.
   - ✓ Similarly, when tuples are represented on a storage device, they must be organized in *some* fashion, and it may be advantageous, from a performance standpoint, to organize them in a way that depends upon their content.

2. **Ordering of Values within a Tuple**
   - ✓ The order of attributes and their values is not that important as long as the correspondence between attributes and values is maintained.
   - ✓ A tuple can be considered as a set of (<attribute>,<value>) pairs, where each pair gives the value of the mapping from an attribute Ai to a value vi from dom(Ai). The ordering of attributes is not important, because the attribute name appears with its value.

3. **Values and NULLs in the Tuples**
   - ✓ Each value in a tuple is an atomic value; that is, it is not divisible into components.
   - ✓ An important concept is NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple.
   - ✓ NULL values has several meanings, such as **value unknown**, **value exists but is not available**, or **attributedoes not apply to this tuple.**

4. **Interpretation (Meaning) of a Relation**
   - ✓ Each tuple in the relation can then be interpreted as a **fact** or a particular instance of the assertion.
   - ✓ Each relation can be viewed as a **predicate** and each tuple in that relation can be viewed as an assertion for which that predicate is satisfied (i.e., has value **true**) for the combination of values in it.
   - ✓ Example:There exists a student having name Benjamin Bayer, having SSN 305-61-2435, having age 19, etc

Relational Model Notation

The following notation are used for presentation:
- ✓ A relation schema R of degree n is denoted by R(A1, A2, … , An).
- ✓ The uppercase letters Q, R, S denote relation names.
- ✓ The lowercase letters q, r, s denote relation states.
- ✓ The letters t, u, v denote tuples.
- ✓ In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation—the current relation state—whereas STUDENT(Name, Ssn, …) refers only to the relation schema.
- ✓ An attribute A can be qualified with the relation name R to which it belongs by using the dot notation R.A—for example, STUDENT.Name or STUDENT.Age. This is because the same name may be used for two attributes in different relations.
- ✓ An n-tuple t in a relation r(R) is denoted by $t = <v1, v2, ….., vn>$, where vi is the value corresponding to attribute Ai. The following notation refers to component values of tuples:
    - • Both t[Ai] and t.Ai (and sometimes t[i]) refer to the value vi in t for attribute Ai.
    - • Both t[Au, Aw, … , Az] and t.(Au, Aw, … , Az), where Au, Aw, … , Az is a list of attributes from R, refer to the subtuple of values from t corresponding to the attributes specified in the list.

2.2 Relational Model Constraints and Relational Database Schemas

Relational ModelConstraints on databases can generally be divided into three main categories:
1. Constraints that are inherent in the data model known as **inherent model-based constraints or implicit constraints**.
2. Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL   known as **schema-based constraints or explicit constraints**.
3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs or in some other way known **as application-based or semantic constraints or business rules.**

The schema-based constraints include **domain constraints, key constraints, constraints on NULLs, entity integrity constraints, and referential integrity constraints.**

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 and real numbers. Characters, Booleans, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and other special data types.

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; hence, all tuples in a relation must also be distinct.
- ✓ This means that no two tuples can have the same combination of values for all their attributes. Usually, there are other subsets of attributes of a relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes.
- ✓ Suppose that we denote one such subset of attributes by SK; then for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that:
    t1[SK] ≠ t2[SK]

- ✓ A superkey SK specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value for SK.
- ✓ A key k of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K′ that is not a superkey of R any more.

  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 uniqueness property also applies to a superkey.

  2. It is a minimal superkey—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold. This minimality property is required for a key but is optional for a superkey.

3. Relational Databases and Relational Database Schemas

- ✓ A relational database is a collection of many relations.
- ✓ A relational database schema S is a set of relation schemas S = {R1, R2, … , Rm} and a set of integrity constraints IC.
- ✓ A relational database state DB of S is a set of relation states DB = {r1, r2, … ,rm} such that each ri is a state of Ri and such that the ri relation states satisfy the integrity constraints specified in IC.
- ✓ A relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}.
- ✓ When we refer to a relational database, we implicitly include both its schema and its current state. A database state that does not obey all the integrity constraints is called **not valid**, and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a **valid state.**

4. Entity 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.
- ✓ 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 each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.
- ✓ The conditions for a **foreign key**, given below, specify a referential integrity constraint between the two relation schemas R1 and R2.
- ✓ 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. In the former case, we have t1[FK] = t2[PK], and we say that the tuple t1 references or refers to the tuple t2.
- ✓ In this definition, R1 is called the referencing relation and R2 is the referenced relation. If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold.
- ✓ In the EMPLOYEE relation, the attribute Dno refers to the department for which an employee works; hence, it is designated 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 Constraints the primary key of DEPARTMENT—the Dnumber attribute—in some tuple t2 of the DEPARTMENT relation, or the value of Dno can be NULL if the employee does not belong to a department or will be assigned to a department later.

Referential integrity constraints displayed on the COMPANY relational database schema.
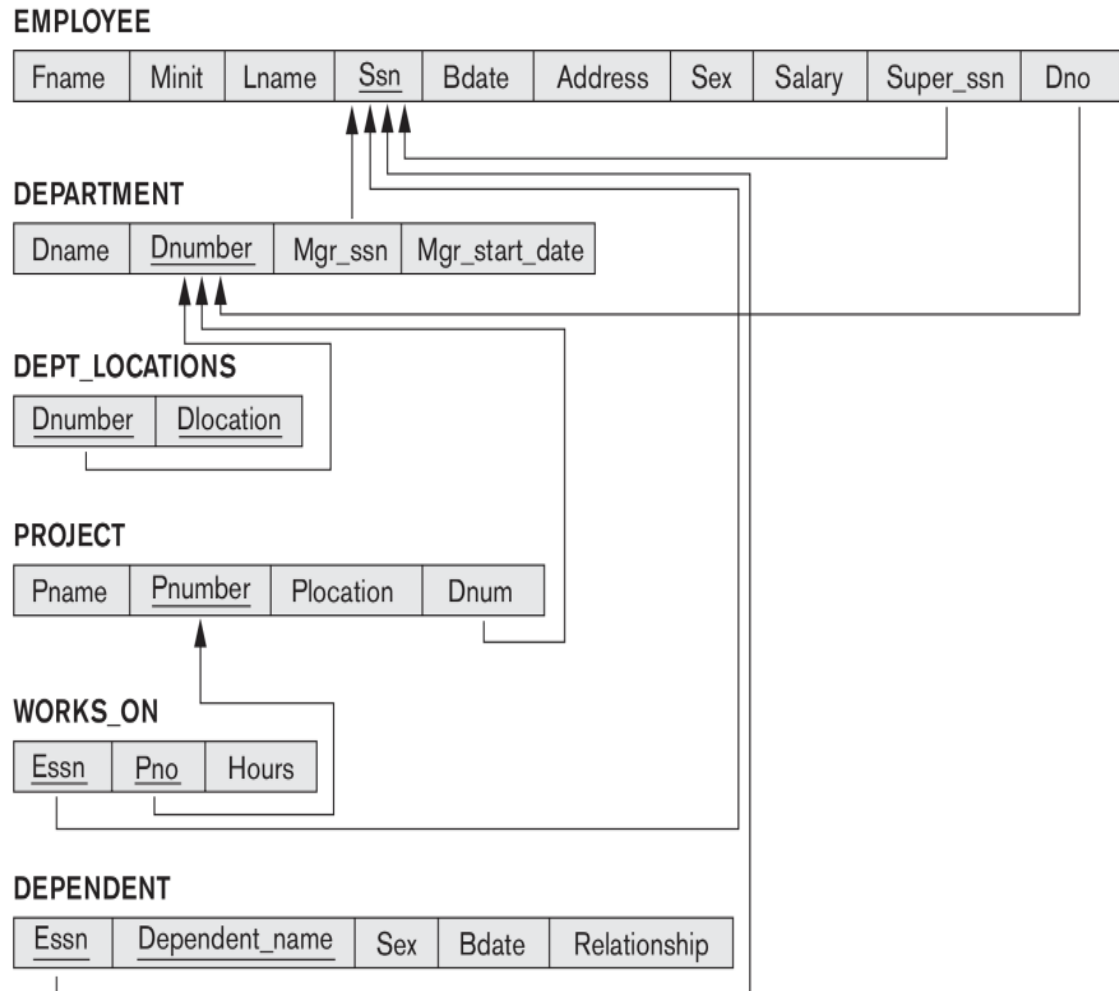
Other types of constraints

✓ 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. Sometimes called as **Semantic Integrity constraint**.

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

2.3 Update Operations, Transactions, and Dealing with Constraint Violations

✓ There are three basic operations that can change the states of relations in the database: Insert, Delete, and Update (or Modify).
✓ Insert is used to insert one or more new tuples in a relation.
✓ Delete is used to delete tuples.
✓ Update (or Modify) is used to change the values of some attributes in existing tuples.

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.

▨ *Operation:*
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 the primary key Ssn), so it is rejected.

▨ *Operation:*
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 with the same Ssn value already exists in the EMPLOYEE relation, and so it is rejected.

▨ *Operation:*
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 on Dno in EMPLOYEE because no corresponding tuple exists in DEPARTMENT with Dnumber = 7.

▨ *Operation:*
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.

✓ If an insertion violates one or more constraints, the default option is to reject the insertion.
✓ Another option is to attempt to correct the reason for rejecting the insertion, but this is typically not used for violations caused by Insert; rather, it is used more often in correcting violations for Delete and Update.

The Delete Operation

✓ The Delete operation can violate only referential integrity.

✓ This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database.

✓ To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted.

▪ *Operation:*
Delete the WORKS_ON tuple with Essn = '999887777' and Pno = 10.
Result: This deletion is acceptable and deletes exactly one tuple.

▪ *Operation:*
Delete the EMPLOYEE tuple with Ssn = '999887777'.
Result: This deletion is not acceptable, because there are tuples in WORKS_ON that refer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.

▪ *Operation:*
Delete the EMPLOYEE tuple with Ssn = '333445555'.
Result: This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

➢ Several options are available if a deletion operation causes a violation. 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. For example, in operation 2, the DBMS could automatically delete the offending tuples from WORKS_ON with Essn = '999887777'.

➢ 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.

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.

▪ *Operation:*
Update the salary of the EMPLOYEE tuple with Ssn = '999887777' to 28000.
Result: Acceptable.

▪ *Operation:*
Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 1.
Result: Acceptable.

▪ *Operation:*
Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 7.
Result: Unacceptable, because it violates referential integrity.

▪ *Operation:*
Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to '987654321'.
Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn.

A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must

leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

A large number of commercial applications running against relational databases in **online transaction processing** (OLTP) systems are executing transactions at rates that reach several hundred per second.

RELATIONAL ALGEBRA

2.4 Unary and Binary relational operations

SELECT and PROJECT

The SELECT Operation

- ✓ The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a¬ selection condition.
- ✓ It restricts the tuples in a relation to only those tuples that¬ satisfy the condition.
- ✓ It can also be visualized as a horizontal partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.
- ✓ For example, to select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000

$$\sigma Dno=4(EMPLOYEE)$$
$$\sigma Salary>30000(EMPLOYEE)$$

- ✓ 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.
- ✓ The Boolean expression specified in is made up of a number of clauses of the form :
  <attribute name><comparison op><constant value>
  Or
  <attribute name><comparison op><attribute name>
- ✓ Clauses can be connected by the standard Boolean operators and, or, and not to form a general selection condition.
- ✓ For example, 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:

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

- ✓ The Boolean conditions AND, OR, and NOT have their normal interpretation, as follows:
  - ■ (cond1 AND cond2) is TRUE if both (cond1) and (cond2) are TRUE; otherwise, it is FALSE.
  - ■ (cond1 OR cond2) is TRUE if either (cond1) or (cond2) or both are TRUE; otherwise, it is FALSE.
  - ■ (NOT cond) is TRUE if cond is FALSE; otherwise, it is FALSE.
- ✓ The SELECT operator is unary; that is, it is applied to a single relation. Hence, selection conditions cannot involve more than one tuple.
- ✓ The degree of the relation resulting from a SELECT operation—its number of attributes—is the¬ same as the degree of R.
- ✓ The SELECT operation is commutative; that is,

$$\sigma_{(cond1)}(\sigma_{(cond2)}(R)) = \sigma_{(cond2)}(\sigma_{(cond1)}(R))$$

The PROJECT Operation

- ✓ The PROJECT operation, selects certain columns from the table and discards the other columns.
- ✓ The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations: one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.
- ✓ For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

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

- ✓ The general form of the PROJECT operation is

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

  where (pi) is the symbol used to represent the PROJECT operation, and is the desired sublist of attributes from the attributes of relation R.
- ✓ The result of the PROJECT operation has only the attributes specified in in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>.
- ✓ 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.

Sequences of Operations and the RENAME Operation

- ✓ The relations shown above depict operation results do not have any names.
- ✓ Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations.
- ✓ In the latter case, we must give names to the relations that hold the intermediate results.
- ✓ For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, apply a SELECT and a PROJECT operation.
- ✓

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

- ✓ Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, and using the assignment operation, denoted by ← (left arrow), as follows:

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Fname, Lname, Salary}(DEP5\_EMPS)$$

✓ It is sometimes simpler to break down a complex sequence of operations by specifying intermediate result relations than to write a single relational algebra expression.

✓ We can also use this technique to rename the attributes in the intermediate and result relations.

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

> TEMP ← σDno=5(EMPLOYEE)
> R(First_name, Last_name, Salary) ← πFname, Lname, Salary(TEMP)

✓ The formal RENAME operation—which can rename either the relation name or the attribute names, or both—as a unary operator.

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

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

✓ The first expression renames both the relation and its attributes, the second renames the relation only, and the third renames the attributes only.

Relational Algebra Operations from Set Theory

The UNION, INTERSECTION, and MINUS Operations

■ 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 by R − S, is a relation that includes all tuples that are in R but not in S.

✓ These are binary operations; that is, each is applied to two sets (of tuples).

✓ When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called union compatibility or type compatibility.

✓ Two relations R(A1, A2, … , An) and S(B1, B2, … , Bn) are said to be union compatible (or type compatible) if they have the same degree n and if dom(Ai) = dom(Bi) for 1 ≤ i ≤ n. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

✓ 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

✓ Both UNION and INTERSECTION are commutative operations; that is,

R ∪ S = S ∪ R and R ∩ S = S ∩ R

- ✓ Both UNION and INTERSECTION can be treated as n-ary operations applicable to any number of relations because both are also associative operations; that is,

$$R \cup (S \cup T) = (R \cup S) \cup T \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

- ✓ The MINUS operation is not commutative; that is, in general, $R - S \neq S - R$
- ✓ The INTERSECTION can be expressed in terms of union and set difference as follows:

$$R \cap S = ((R \cup S) - (R - S)) - (S - R)$$

The CARTESIAN PRODUCT (CROSS PRODUCT) Operation

- ✓ CARTESIAN PRODUCT operation—also known as CROSS PRODUCT or CROSS JOIN—• which is denoted by ×.
- ✓ This is also a binary set operation, but the relations on which it is applied do not have to be union compatible.
- ✓ This set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set).
- ✓ In general, the result of R(A1, A2, ..., An) × S(B1, B2, ..., Bm) is a relation Q with degree n + m• attributes Q(A1, A2, ..., An, B1, B2, ..., Bm), in that order.
- ✓ The resulting relation Q has one tuple for each combination of tuples—one from R and one from S.
- ✓ Hence, if R has m tuples and S has n tuples, then R × S will have m*n tuples.
- ✓ Example, suppose that we want to retrieve a list of names of each female employee's dependents. We can do this as follows:

> 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)

Binary Relational Operations: JOIN and DIVISION
The JOIN Operation

- ✓ The JOIN operation, denoted by ⋈, is used to combine related tuples from two relations into single "longer" tuples.
- ✓ This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.
- ✓ To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department, as follows:

> DEPT_MGR ← DEPARTMENT ⋈ Mgr_ssn=Ssn
> EMPLOYEE
> RESULT ← πDname, Lname, Fname(DEPT_MGR)

- ✓ The general form of a JOIN operation on two relations R(A1, A2, ... , An) and S(B1, B2, ... , Bm) is

$$R ⋈ \text{<join condition>} S$$

- ✓ The result of the JOIN is a relation Q with n + m attributes Q(A1, A2, ... , An, B1, B2, ... , Bm) 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.

- ✓ The main difference between CARTESIAN PRODUCT and JOIN are ,In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.
- ✓ The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples. Each tuple combination for which the join condition evaluates to TRUE is included in the resulting relation Q as a single combined tuple.

> - ✓ A general join condition is of the form
>   <condition>AND<condition> AND … AND<condition>
>   where each <condition> is of the form Ai θ Bj , Ai is an attribute of R, Bj is an attribute of S, Ai and Bj have the same domain, and θ (theta) is one of the comparison operators {=, <,>,< , ≥,≠}.
>   A JOIN operation with such a general join condition is called a THETA JOIN.

Variations of JOIN:

The EQUIJOIN and NATURAL JOIN

- ✓ The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUIJOIN.
- ✓ Notice that in the result of an EQUIJOIN we always have one or more pairs of attributes that haveidentical values in every tuple.
- ✓ For example, the values of the attributes Mgr_ssn and Ssn are identical in every• tuple of DEPT_MGR (the EQUIJOIN result) because the equality join condition specified on these two attributes requires the values to be identical in every tuple in the result.
- ✓ 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 (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first.
- ✓ Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project. 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(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 ← ρ(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.
- ✓ In general, the join condition for NATURAL JOIN is constructed by equating each pair of join attributes that have the same name in the two relations and combining these conditions with AND.
- ✓ 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.
- ✓ A more general, but nonstandard definition for NATURAL JOIN is Q ← R *(list1),(list2)S
- ✓ In this case,<list1> specifies a list of i attributes from R, and <list2>specifies a list of i attributes from S.
- ✓ The NATURAL JOIN or EQUIJOIN operation can also be specified among multiple tables, leading to an n-way join. For example, consider the following three-way join:
  ((PROJECT ⋈ Dnum=DnumberDEPARTMENT) ⋈ Mgr_ssn=SsnEMPLOYEE)

A Complete Set of Relational Algebra Operations

- ✓ It has been shown that the set of relational algebra operations $\{\sigma, \pi, \cup, \rho, -, \times\}$ is a complete set;• that is, any of the other original relational algebra operations can be expressed as a sequence of operations from this set.
- ✓ For example, the INTERSECTION operation can be expressed by using UNION and MINUS as follows: $R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$
- ✓ JOIN operation can be specified as a CARTESIAN PRODUCT followed by a SELECT operation:
  $R \bowtie_{<condition>} S \equiv \sigma_{<condition>}(R \times S)$
- ✓ A NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME and followed by SELECT and PROJECT operations.

The DIVISION Operation

- ✓ The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimesoccurs in database applications.
- ✓ example is Retrieve the names of employees who work on all the projects that 'John Smith' works on.
  To express this query using the DIVISION operation, proceed as follows. First, retrieve the list of project numbers that 'John Smith' works on in the intermediate relation SMITH_PNOS:

  SMITH ← σFname='John' AND Lname='Smith'(EMPLOYEE)

  SMITH_PNOS ← πPno(WORKS_ON $\bowtie$ Essn=SsnSMITH)

- ✓ Next, create a relation that includes a tuple whenever the employee whose Ssn is Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

  SSN_PNOS ← πEssn, Pno(WORKS_ON)

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

  SSNS(Ssn) ← SSN_PNOS ÷ SMITH_PNOS

  RESULT ← πFname, Lname(SSNS * EMPLOYEE)

- ✓ In general, the DIVISION operation is applied to two relations $R(Z) \div S(X)$, where the attributes of R are a subset of the attributes of S; that is, $X \subseteq Z$. Let Y be the set of attributes of R that are not attributes of S.
- ✓ The DIVISION operation is defined for convenience for dealing with queries that involve universal quantification or the all condition.

Operations of Relational Algebra

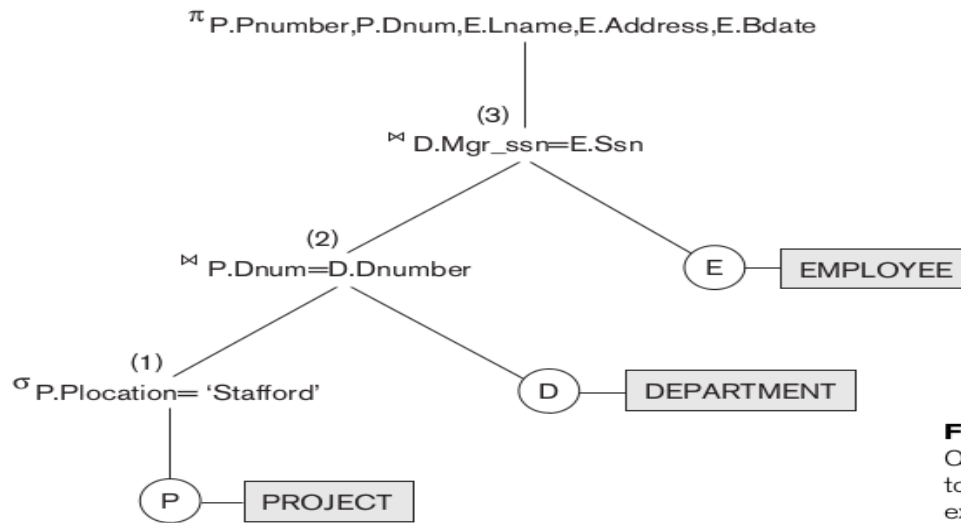| Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	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.	$R_1 \cap R_2$
SET 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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Notation for Query Trees

- ✓ Describes about the notation typically used in relational systems to represent queries internally.
- ✓ The notation is called a query tree or sometimes it is known as a query evaluation tree or query execution tree.
- ✓ It includes the relational algebra operations being executed and is used as a possible data structure for the internal representation of the query in an RDBMS.
- ✓ A query tree is a tree data structure that corresponds to a relational algebra expression.
- ✓ It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes.
- ✓ An execution of the query tree consists of executing an internal node operation whenever its operands (represented by its child nodes) are available, and then replacing that internal node by the relation that results from executing the operation.
- ✓ The execution terminates when the root node is executed and produces the result relation for the query.

πPnumber, Dnum, Lname, Address, Bdate$((($\sigma$Plocation='Stafford'(PROJECT)) $\bowtie$ Dnum=Dnumber(DEPARTMENT)) $\bowtie$ Mgr_ssn=Ssn(EMPLOYEE))

- ✓ Query tree for the abobe query.In this, the three leaf nodes P, D, and E represent the three relations PROJECT, DEPARTMENT, and EMPLOYEE.

π P.Pnumber,P.Dnum,E.Lname,E.Address,E.Bdate

(3)
⋈ D.Mgr_ssn=E.Ssn

(2)
⋈ P.Dnum=D.Dnumber

E — EMPLOYEE

(1)
σ P.Plocation= 'Stafford'

D — DEPARTMENT

P — PROJECT

Fig
Qu
to
ex

- ✓ In order to execute query , the node marked (1) in Figure must begin execution before node(2) because some resulting tuples of operation (1) must be available before we can begin to execute operation (2). Similarly, node (2) must begin to execute and produce results before node (3) can start execution, and so on.
- ✓ In general, a query tree gives a good visual representation and understanding of the query in terms of the relational operations it uses and is recommended as an additional means for expressing queries in relational algebra.

2.5 Additional Relational Operations (aggregate, grouping, etc.)

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:

  πF1, F2, ..., Fn (R)

  where F1, F2, … , Fn are functions over the attributes in relation R and may involve arithmetic operations and constant values.
- ✓ This operation is helpful when developing reports where computed values have to be produced in the columns of a query result.
- ✓ Example: 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:

  REPORT ← ρ(Ssn, Net_salary, Bonus, Tax)(πSsn, Salary – Deduction, 2000 * Years_service, 0.25 * Salary(EMPLOYEE))

Aggregate Functions and Grouping

- ✓ Mathematical aggregate functions on collections of values from the database. Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
- ✓ Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM.
  The COUNT function is used for counting tuples or values.
- ✓ AGGREGATE FUNCTION operation defined, using the symbol $\mathfrak{I}$, to specify these types of requests as follows:
  <grouping attribute>$\mathfrak{I}$<function list> (R)
  Where<grouping attribute> is a list of attributes of the relation specified in R, and <function list>is a list of (<function><attribute>) pairs.
- ✓ In each such pair, is one of the allowed functions—such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT—and is an attribute of the relation specified by R.
- ✓ The resulting relation has the grouping attributes plus one attribute for each element in the function list.
- ✓ For example, to retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes as indicated below, we write:
  ρR(Dno, No_of_employees, Average_sal) (Dno $\mathfrak{I}$ COUNT Ssn, AVERAGE Salary (EMPLOYEE))
- ✓ If we do not want to rename the attributes then the above query we can write it as,
  Dno $\mathfrak{I}$ COUNT Ssn, AVERAGE Salary(EMPLOYEE)
- ✓ Note: 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.

Recursive Closure Operations

- ✓ Recursive closure operation in relational algebra is applied to a recursive relationship between tuples of the same type, such as the relationship between an employee and a supervisor.
- ✓ This relationship is described by the foreign key Super_ssn of the EMPLOYEE relation and it relates each employee tuple (in the role of supervisee) to another employee tuple (in the role of supervisor).
- ✓ An example of a recursive operation is to retrieve all supervisees of an employee e at all levels—that is, all employees 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 e_ directly supervised—at level one—by the employee e whose name is 'James Borg' we can apply the following operation:
  BORG_SSN ← πSsn(σFname='James' AND Lname='Borg'(EMPLOYEE))
  SUPERVISION(Ssn1, Ssn2) ← πSsn,Super_ssn(EMPLOYEE)
  RESULT1(Ssn) ← πSsn1(SUPERVISION $\bowtie$ Ssn2=SsnBORG_SSN)

- ✓ To retrieve all employees supervised by Borg at level 2—that is, all employees e'' supervised by some employee e' who is directly supervised by Borg—we can apply another JOIN to the result of the first query, as follows:
  RESULT2(Ssn) ← πSsn1(SUPERVISION $\bowtie$ Ssn2=SsnRESULT1)
- ✓ To get both sets of employees supervised at levels 1 and 2 by 'James Borg', we can apply the UNION operation to the two results, as follows:

  RESULT ← RESULT2 U RESULT1

OUTER JOIN Operations

- ✓ For a NATURAL JOIN operation R * S, only tuples from R that have matching tuples in S—andvice versa—appear in the result. Hence, tuples without a matching (or related) tuple are eliminated from the JOIN result.
- ✓ Tuples with NULL values in the join attributes are also eliminated. This type of join, where tuples with no match are eliminated, is known as an inner join.
- ✓ This amounts to the loss of information if the user wants the result of the JOIN to include all thetuples in one or more of the component relations.
- ✓ A set of operations, called outer joins, were developed for the case where the user wants to keep all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.
- ✓ This satisfies the need of queries in which tuples from two tables are to be combined by matching corresponding rows, but without losing any tuples for lack of matching values.
- ✓ For example, suppose that we want a list of all employee names as well as the name of the departments they manage if they happen to manage a department; if they do not manage one, we can indicate it with a NULL value. We can apply an operation LEFT OUTER JOIN, denoted by ⋈, to retrieve the result as follows:

TEMP ← (EMPLOYEE ⋈ Ssn=Mgr_ssnDEPARTMENT)
RESULT ← πFname, Minit, Lname, Dname(TEMP)

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

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

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

The OUTER UNION Operation

- ✓ The OUTER UNION operation was developed to take the union of tuples from two relations that have some common attributes, but are not union (type) compatible.
- ✓ This operation will take the UNION of tuples in two relations R(X, Y) and S(X, Z) that are partially compatible, meaning that only some of their attributes, say X, are union compatible.
- ✓ The attributes that are union compatible are represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation T(X,Y, Z). It is therefore the same as a FULL OUTER JOIN on the common attributes.
- ✓ Two tuples t1 in R and t2 in S are said to match if t1[X]=t2[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.

2.6 Examples of Queries in Relational Algebra

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

RESEARCH_DEPT ← $\sigma_{Dname='Research'}$(DEPARTMENT)

RESEARCH_EMPS ← (RESEARCH_DEPT $\bowtie_{Dnumber=Dno}$EMPLOYEE)

RESULT ← $\pi_{Fname, Lname, Address}$(RESEARCH_EMPS)

As a single expression, this query becomes

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

This query could be specified in other ways; for example, the order of the JOIN and SELECT operations could be reversed, or the JOIN could be replaced by a NATURAL JOIN after renaming one of the join attributes.

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.

STAFFORD_PROJS ← $\sigma_{Plocation='Stafford'}$ (PROJECT)

CONTR_DEPT ← (STAFFORD_PROJS $\bowtie_{Dnum=Dnumber}$ DEPARTMENT)

PROJ_DEPT_MGR ← (CONTR_DEPT $\bowtie_{Mgr\_ssn=Ssn}$ EMPLOYEE)

RESULT ← $\pi_{Pnumber, Dnum, Lname, Address, Bdate}$ (PROJ_DEPT_MGR)

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

DEPT5_PROJS(Pno) ← $\pi_{Pnumber}$($\sigma_{Dnum=5}$(PROJECT))

EMP_PROJ(Ssn, Pno) ← $\pi_{Essn, Pno}$(WORKS_ON)

RESULT_EMP_SSNS ← EMP_PROJ ÷ DEPT5_PROJS

RESULT ← $\pi_{Lname, Fname}$ (RESULT_EMP_SSNS * 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.

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

As a single expression, this query becomes

$$\pi_{Pno} (WORKS\_ON \bowtie_{Essn=Ssn} (\pi_{Ssn} (\sigma_{Lname='Smith'} (EMPLOYEE)))$$
$$\cup \; \pi_{Pno} ((\pi_{Dnumber} (\sigma_{Lname='Smith'} (\pi_{Lname, \, Dnumber} (EMPLOYEE)))$$
$$\bowtie_{Ssn=Mgr\_ssn} DEPARTMENT)) \bowtie_{Dnumber=Dnum} PROJECT)$$

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

Strictly speaking, this query cannot be done in the *basic (original)* relational algebra. We have to use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the *same* employee have *distinct* DEPENDENT_NAME values.

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

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

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

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

As a single expression, this query becomes

$$\pi_{Lname, \, Fname}((\pi_{Ssn}(EMPLOYEE) - \rho_{Ssn}(\pi_{Essn}(DEPENDENT))) * EMPLOYEE)$$

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

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

Mapping Conceptual Design into a Logical Design

2.7 Relational Database Design using ER-to-Relational mapping.

Step 1: For each **regular (strong) entity type** E in the ER schema, create a relation R that includes all the simple attributes of E.
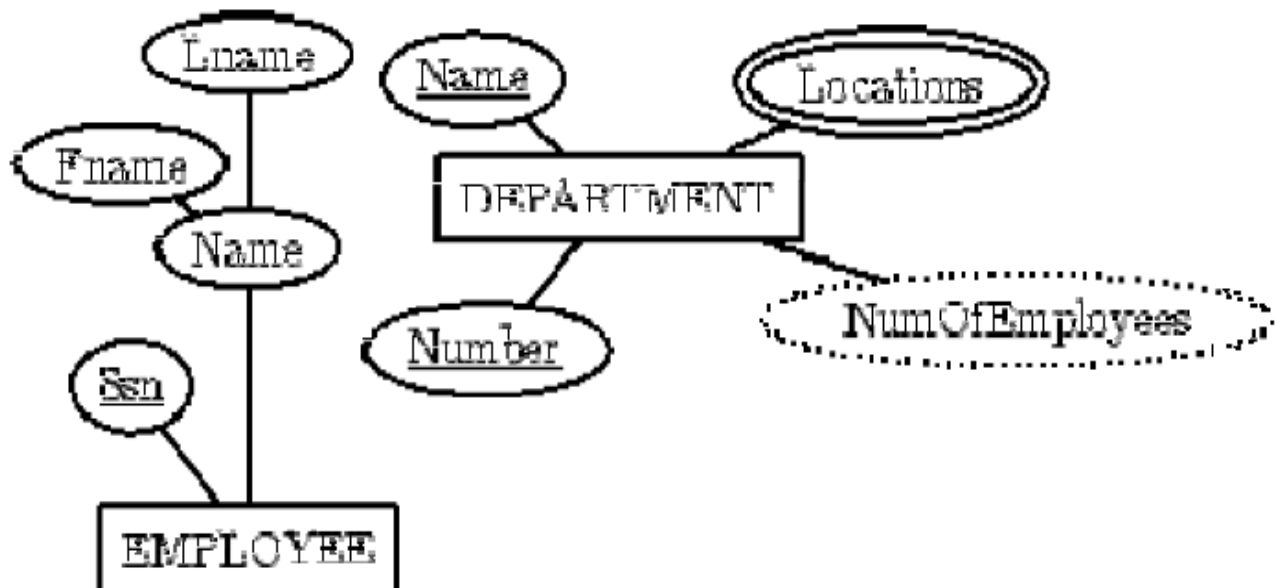
EMPLOYEE

SSN	Lname	Fname

DEPARTMENT

NUMBER	NAME

Step 2: For each **weak entity type** W in the ER schema with owner entity type E, create a relation R, and include all simple attributes (or simple components of composite attributes) of W as attributes. In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
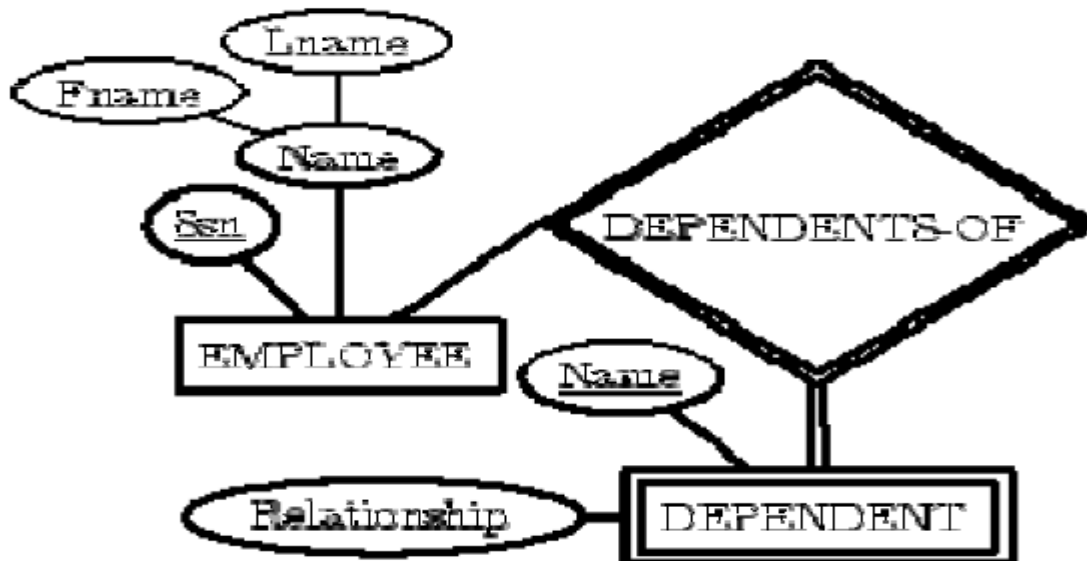
DEPENDENT

EMPL-SSN	NAME	Relationship

Step 3: For each **binary 1:1 relationship type** R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations, say S, and include the primary key of T as a foreign key in S. Include all the simple attributes of R as attributes of S.
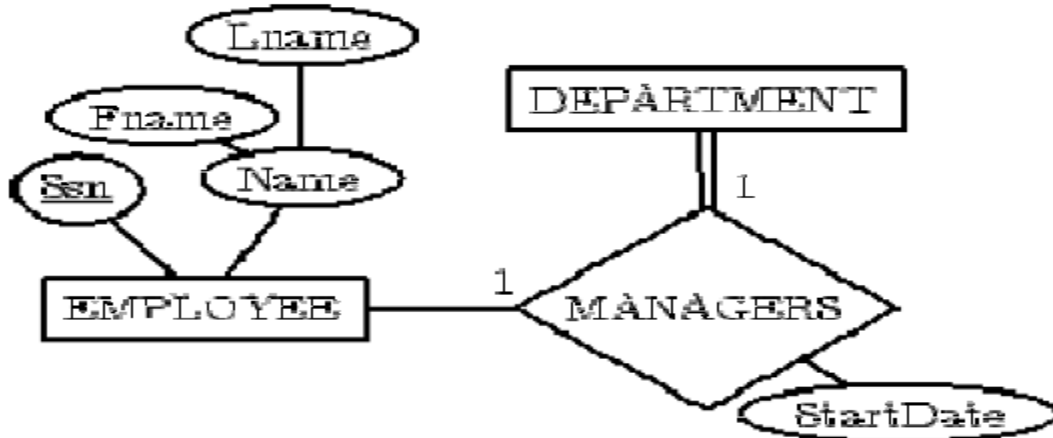
DEPARTMENT

MANAGER-SSN	StartDate

Step 4: For each regular **binary 1:N relationship type** R identify the relation (N) relation S. the primary key of T as a foreign key of S. Simple attributes of R map to attributes of S.

EMPLOYEE

SupervisorSSN

Step 5: For each **binary M:N relationship type** R, create a relation S. Include the primary keys of participant relations as foreign keys in S. Their combination will be the primary key for S. Simple attributes of R become attributes of S.

WORKS-FOR

EmployeeSSN	DeptNumber

Step 6: For each **multi-valued attribute A**, create a new relation R. This relation will include an attribute corresponding to A, plus the primary key K of the parent relation (entity type or relationship type) as a foreign key in R. The primary key of R is the combination of A and K.

DEP-LOCATION

Location	DEP-NUMBER
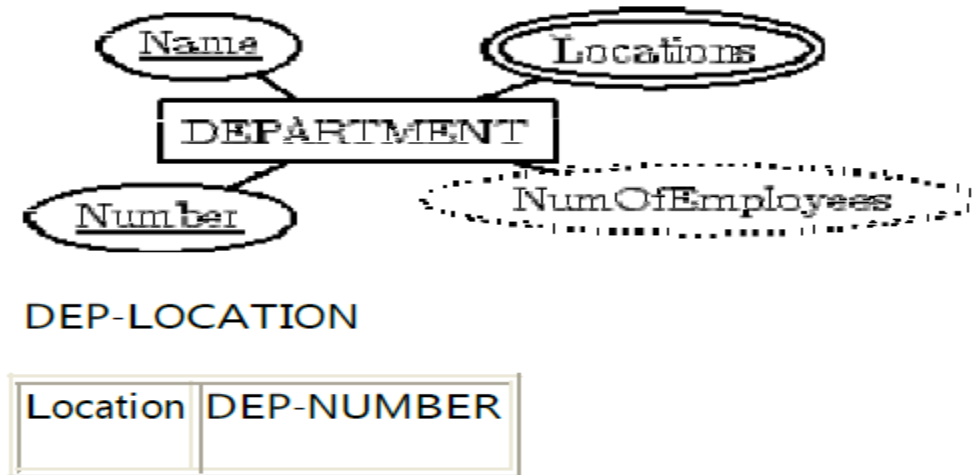
Step 7: For each **n-ary relationship type R**, where n>2, create a new relation S to represent R. Include the primary keys of the relations participating in R as foreign keys in S. Simple attributes of R map to attributes of S. The primary key of S is a combination of all the foreign keys that reference the participants that have cardinality constraint > 1. For a recursive relationship, we will need a new relation.

SQL

2.8 SQL Data Definition and Data Types

- ✓ SQL uses the terms table, row, and column for the formal relational model terms relation, tuple, and attribute, respectively.
- ✓ SQL command for data definition is the CREATE statement, which can be used to create schemas, tables (relations), types, and domains, as well as other constructs such as views, assertions, and triggers.
- ✓ An SQL schema is identified by a **schema name** and includes **an authorization identifier** to indicate the user or account who owns the schema, as well as descriptors for each element in the schema.
- ✓ Schema elements include tables, types, constraints, views, domains, and other constructs.
- ✓ A schema is created via the CREATE SCHEMA statement, which can include all the schema elements' definitions.

  Eg: CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
- ✓ In SQL**catalog**—a named collection of schemas.

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 possibly 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.

✓ Eg: CREATE TABLE EMPLOYEE

```
CREATE TABLE EMPLOYEE
        ( Fname                          VARCHAR(15)              NOT NULL,
          Minit                          CHAR,
          Lname                          VARCHAR(15)              NOT NULL,
          Ssn                            CHAR(9)                  NOT NULL,
          Bdate                          DATE,
          Address                        VARCHAR(30),
          Sex                            CHAR,
          Salary                         DECIMAL(10,2),
          Super_ssn                      CHAR(9),
          Dno                            INT                      NOT NULL,
        PRIMARY KEY (Ssn),
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) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber                        INT                      NOT NULL,
          Dlocation                      VARCHAR(15)              NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname                          VARCHAR(15)              NOT NULL,
          Pnumber                        INT                      NOT NULL,
          Plocation                      VARCHAR(15),
          Dnum                           INT                      NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                           CHAR(9)                  NOT NULL,
          Pno                            INT                      NOT NULL,
          Hours                          DECIMAL(3,1)             NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                           CHAR(9)                  NOT NULL,
          Dependent_name                 VARCHAR(15)              NOT NULL,
          Sex                            CHAR,
          Bdate                          DATE,
          Relationship                   VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Attribute Data Types and Domains in SQL

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

Numeric data types

▪ Integer numbers of various sizes like INTEGER or INT, and SMALLINT

- Floating-point (real) numbers of various precision like FLOAT or REAL, and DOUBLE PRECISION
- Formatted numbers using DECIMAL(i, j)—or DEC(i, j) or NUMERIC(i, j)—where i, the precision, is the total number of decimal digits and j, the scale, is the number of digits after the decimal point.

Character-string datatypes
- Fixed length—CHAR(n) or CHARACTER(n), where n is the number of character
- Varying length— VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.

Bit-string data types
- Fixed length n—BIT(n)—or varying length— BIT VARYING(n), where n is the maximum number of bits.
- 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.

Boolean data types
- 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.

DATE data types
- Date 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.

Timestamp data type (TIMESTAMP)
- Includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.

Interval
- Another data type related to DATE, TIME, and TIMESTAMP is the INTERVAL data type.
- This specifies an interval—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp.

2.9 Specifying Constraints in SQL

These include key and referential integrity constraints, restrictions on attribute domains and NULLs, and constraints on individual tuples within a relation using the CHECK clause.

Specifying Attribute Constraints and Attribute Defaults
- ✓ Because SQL allows NULLs as attribute values, a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute which are part of the primary key of each relation.
- ✓ It is also possible to define a default value for an attribute by appending the clause DEFAULT to an attribute definition.
- ✓ The default value is included in any new tuple if an explicit value is not provided for that attribute.
- ✓ Figure illustrates an example of specifying a default manager for a new department and a default department for a new employee.
- ✓ 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 can restrict attribute or domain values using the **CHECK clause** following an attribute or domain definition.
- ✓ For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then, we can change the attribute declaration of Dnumber in the DEPARTMENT table to the following:

  Dnumber INT NOT NULL CHECK (Dnumber> 0 AND Dnumber< 21);

```
CREATE TABLE EMPLOYEE
    ( ...,
        Dno        INT           NOT NULL        DEFAULT 1,
    CONSTRAINT EMPPK
        PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET NULL        ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
        FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE SET DEFAULT     ON UPDATE CASCADE);
CREATE TABLE DEPARTMENT
    ( ...,
        Mgr_ssn    CHAR(9)       NOT NULL        DEFAULT '888665555',
        ...,
    CONSTRAINT DEPTPK
        PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
        UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET DEFAULT   ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
    ( ...,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE CASCADE         ON UPDATE CASCADE);
```

Specifying Key and Referential Integrity Constraints

- ✓ The PRIMARY KEY clause specifies one or more attributes that make up the primary key of a relation. If a primary key has a single attribute, the clause can follow the attribute directly.
- ✓ Ex: the primary key of DEPARTMENT can be specified as follows:
  Dnumber INT PRIMARY KEY,
- ✓ The UNIQUE clause specifies alternate (unique) keys, also known as candidate keys
- ✓ The UNIQUE clause can also be specified directly for a unique key if it is a single attribute, as in the following example: Dname VARCHAR(15) UNIQUE,
- ✓ Referential integrity is specified via the FOREIGN KEY clause,constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is updated.
- ✓ 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.
- ✓ 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 Key and referential integrity constraints were not included in early versions of SQL.

- ✓ 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.

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

Specifying Constraints on Tuples Using CHECK
- ✓ other table constraints can be specified through additional CHECK clauses at the end of a CREATE TABLE statement.
- ✓ These can be called row-based constraints because they apply to each row individually and are checked whenever a row is inserted or modified.
- ✓ For example, suppose that the DEPARTMENT table in Figure had an additional attribute Dept_create_date, which stores the date when the department was created.
- ✓ In 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);

2.10 Basic Retrieval Queries in SQL
- ✓ The basic form of the SELECT statement, sometimes called a mapping or a select-from-where block, is formed of the three clauses SELECT, FROM, and WHERE and has the following form:

    SELECT <attribute list>
    FROM <table list>
    WHERE<condition> ;

where <attribute list> is a list of attribute names whose values are to be retrieved by the query.

<table list> is a list of the relation names required to process the query.

<condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query, the basic logical comparison operators are =, <=, >, >=, and <>

Query 0

Retrieve the birthdate and address of the employee(s) whose name is 'John B Smith'

SELECT   BDATE, ADDRESS
FROM   EMPLOYEE
WHERE   FNAME = 'John' **AND** MINIT = 'B' **AND** LNAME = 'Smith';

Query 1

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;

Query 1A

Ambiguous attribute names

SELECT	FNAME, EMPLOYEE.NAME, ADDRESS
FROM	EMPLOYEE, DEPARTMENT
WHERE	DEPARTMENT.NAME = 'Research' **AND**
	DEPARTMENT.DNUMBER = EMPLOYEE.DNUMBER;

Query 1B

Aliasing

SELECT	E.FNAME, E.NAME, E.ADDRESS
FROM	EMPLOYEE E, DEPARTMENT D
WHERE	D.NAME = 'Research' **AND** D.DNUMBER = E.DNUMBER;

Query 1C

Retrieve all the attribute values of EMPLOYEE tuples who work in department number 5

SELECT	*
FROM	EMPLOYEE
WHERE	DNO = 5;

Query 1D

Retrieve all the attributes of an EMPLOYEE and the attributes of the DEPARTMENT he or she works in for every employee of the 'Research' department

SELECT	*
FROM	EMPLOYEE, DEPARTMENT
WHERE	DNAME = 'Research' **AND** DNO = DNUMBER;

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 birthdate

SELECT	PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM	PROJECT, DEPARTMENT, EMPLOYEE
WHERE	DNUM = DNUMBER **AND** MGRSSN = SSN **AND** PLOCATION = 'Stafford';

Query 8

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.SUPERSSN = S.SSN;

Query 8A

Reformulation of query 8 to retrieve the last name of each employee and his or her supervisor, while renaming the resulting attribute names as EMPLOYEE_NAME and SUPERVISOR_NAME

SELECT	E.LNAME **AS** EMPLOYEE_NAME, S.LNAME **AS** SUPERVISOR_NAME
FROM	EMPLOYEE **AS** E, EMPLOYEE **AS** S
WHERE	E.SUPERSSN = S.SSN;

Query 9

Select all EMPLOYEE SSNs in the database

SELECT SSN
FROM EMPLOYEE;

Query 10

Select all combination of EMPLOYEE SSN and DEPARTMENT DNAME in the database

SELECT SSN, DNAME
FROM EMPLOYEE, DEPARTMENT;

Query 10A

Select the CROSS PRODUCT of the EMPLOYEE and DEPARTMENT relations

SELECT *
FROM EMPLOYEE, DEPARTMENT;

Query 11

Retrieve the salary of every employee

SELECT ALL SALARY
FROM EMPLOYEE;

Query 11A

Retrieve all distinct salary values

SELECT DISTINCT SALARY
FROM EMPLOYEE;

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** '_ _ _ _ _ _ _ 70';

Query 13

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

SELECT FNAME, LNAME, 1.1*SALARY
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN = ESSN **AND** PNO = PNUMBER **AND** 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;
```

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, first name

```
SELECT      DNAME, LNAME, FNAME, PNAME
FROM        DEPARTMENT, EMPLOYEE,WORKS_ON, PROJECT
WHERE DNUMBER = DNO AND SSN = ESSN AND PNO = PNUMBER
ORDER BY DNAME DESC, LNAME ASC, FNAME ASC;
```

2.11 INSERT, DELETE, and UPDATE Statements in SQL

The INSERT Command

- ✓ INSERT is used to add a single tuple (row) to a relation (table).
- ✓ must specify the relation name and a list of values for the tuple.
- ✓ The values should be listed in the same order in which the corresponding attributes were specified in the CREATE TABLE command.
- ✓ INSERT INTO EMPLOYEE VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98 Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);
- ✓ For example, to enter a tuple for a new EMPLOYEE for whom we know only the Fname, Lname, Dno, and Ssn attributes,
  INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn) VALUES ('Richard', 'Marini', 4, '653298653');

The DELETE Command

- ✓ The DELETE command removes tuples from a relation.
- ✓ It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted.
- ✓ Tuples are explicitly deleted from only one table at a time.
- ✓ Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command.
- ✓ A missing WHERE clause specifies that all tuples in the relation are to be deleted.
- ✓ DELETE FROM EMPLOYEE WHERE Ssn = '123456789';
- ✓ DELETE FROM EMPLOYEE;

The UPDATE Command

- ✓ The UPDATE command is used to modify attribute values of one or more selected tuples.
- ✓ As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation.
- ✓ UPDATE PROJECT,
  SET Plocation = 'Bellaire', Dnum = 5
  WHERE Pnumber = 10;

2.12 Additional Features of SQL

- ✓ various techniques for specifying complex retrieval queries, including nested queries, aggregate functions, grouping, joined tables, outer joins, case statements, and recursive queries; SQL **views, triggers, and assertions**; and commands for schema modification.

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

- ✓ Each commercial RDBMS will have, in addition to the SQL commands, a set of commands for specifying physical database design parameters, file structures for relations, and access paths such as indexes. 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. Capabilities such as creating complex-structured attributes, specifying abstract data types (called UDTs or user-defined types) for attributes and tables, creating object identifiers for referencing tuples, and specifying operations on types.

- ✓ SQL and relational databases can interact with new technologies such as **XML** and OLAP/data warehouses.