Module-3 Database Design Theory

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables. This module discuss the basic and higher normal forms.

4.1 Objectives

- To study the process of normalization and refine the database design
- To normalize the tables upto 4NF and 5NF
- To study lossless and lossy join operations
- To study inference rules
- To study other dependencies and Normal Forms.

4.2 Introduction to DB design

Each relation schema consists of a number of attributes, and the relational database schema consists of a number of relation schemas. So far, we have assumed that attributes are grouped to form a relation schema by using the common sense of the database designer or by mapping a database schema design from a conceptual data model such as the ER or Enhanced-ER (EER) data model. These models make the designer identify entity types and relationship types and their respective attributes, which leads to a natural and logical grouping of the attributes into relations.

Database Design deals with coming up with a 'good' schema. There are two levels at which we can discuss the goodness of relation schemas:

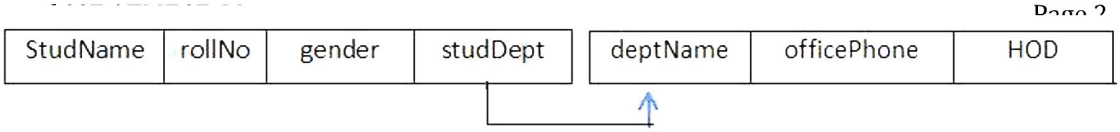
1. The logical (or conceptual) level—how users interpret the relation schemas and the meaning of their attributes.
2. The implementation (or physical storage) level—how the tuples in a base relation are stored and updated. This level applies only to schemas of base relations

An Example

- STUDENT relation with attributes: studName, rollNo, gender, studDept
- DEPARTMENT relation with attributes: deptName, officePhone, hod
- Several students belong to a department
- studDept gives the name of the student's department

Correct schema:

Student Department



Incorrect schema:

Studdept

StudName	rollNo	gender	deptName	officePhone	HOD

Problems with bad schema

- **Redundant storage of data:**
  - Office Phone & HOD info -stored redundantly once with each student that belongs to the department
  - wastage of disk space
- **A program that updates Office Phone of a department**
  - must change it at several places
  - more running time
  - error -prone

4.3 Informal Design Guidelines for Relation Schemas

- Four informal guidelines that may be used as measures to determine the quality of relation schema design:
  1. Making sure that the semantics of the attributes is clear in the schema
  2. Reducing the redundant information in tuples
  3. Reducing the NULL values in tuples
  4. Disallowing the possibility of generating spurious tuples
- These measures are not always independent of one another

4.3.1 Imparting Clear Semantics to Attributes in Relations

- **semantics** of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple
- Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them
- The easier it is to explain the semantics of the relation, the better the relation schema design will be

Guideline 1

- Design a relation schema so that it is easy to explain its meaning
- Do not combine attributes from multiple entity types and relationship types into a single relation

- if a relation schema corresponds to one entity type or one relationship type, it is straightforward to interpret and to explain its meaning
- if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

Examples of Violating Guideline 1

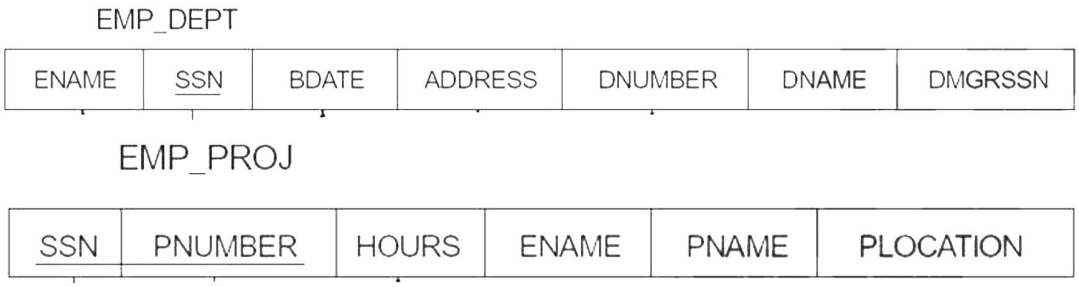


Fig: schema diagram for company database

- Both the relation schemas have clear semantics
- A tuple in the EMP_DEPT relation schema represents a single employee but includes additional information— the name (Dname) of the department for which the employee works and the Social Security number (Dmgr_ssn) of the department manager.
- A tuple in the EMP_PROJ relates an employee to a project but also includes the employee name (Ename), project name (Pname), and project location (Plocation)
- logically correct but they violate Guideline 1 by mixing attributes from distinct real-world entities:
  - EMP_DEPT mixes attributes of employees and departments
  - EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship
- They may be used as views, but they cause problems when used as base relations

4.3.2 Redundant Information in Tuples and Update Anomalies

- One goal of schema design is to minimize the storage space used by the base relations
- Grouping attributes into relation schemas has a significant effect on storage space
- For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT with that for an EMP_DEPT base relation
- In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for every employee who works for that department

- In contrast, each department's information appears only once in the DEPARTMENT relation. Only the department number Dnumber is repeated in the EMPLOYEE relation for each employee who works in that department as a foreign key

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Figure 1: One possible database state for the COMPANY relational database schema

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

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

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

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Figure 1 : One possible database state for the COMPANY relational database schema

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Fig: Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 1

- Storing natural joins of base relations leads to an additional problem referred to as **update anomalies**. These can be classified into:
  - insertion anomalies
  - deletion anomalies,
  - modification anomalies

Insertion Anomalies

- Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

1. To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs
   - For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are *consistent* with the corresponding values for department 5 in other tuples in EMP_DEPT
   - In the design of Employee in fig 1, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation
2. It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee

   - This violates the entity integrity for EMP_DEPT because Ssn is its primary key
   - This problem does not occur in the design of Figure 1 because a department is entered in the DEPARTMENT relation whether or not any employees work for it, and whenever an employee is assigned to that department, a corresponding tuple is inserted in EMPLOYEE.

Deletion Anomalies

- The problem of deletion anomalies is related to the second insertion anomaly situation just discussed
  - If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database
  - This problem does not occur in the database of Figure 2 because DEPARTMENT tuples are stored separately.

Modification Anomalies

- In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all* employees who work in that department; otherwise, the database will become inconsistent
- If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

Guideline 2

- Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations
- If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly
- The second guideline is consistent with and, in a way, a restatement of the first guideline
- These guidelines may sometimes have to be violated in order to improve the performance of certain queries.

4.3.3 NULL Values in Tuples

- If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples
  - this can waste space at the storage level
  - may lead to problems with understanding the meaning of the attributes
  - may also lead to problems with specifying JOIN operations
  - how to account for them when aggregate operations such as COUNT or SUM are applied
- SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable.
- Moreover, NULLs can have multiple interpretations, such as the following:
  - The attribute *does not apply* to this tuple. For example, Visa_status may not apply to U.S. students.
  - The attribute value for this tuple is *unknown*. For example, the Date_of_birth may be unknown for an employee.
  - The value is *known but absent*; that is, it has not been recorded yet. For example, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

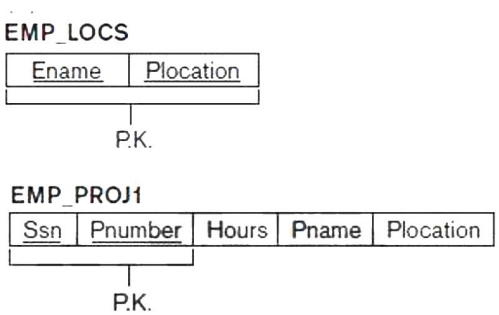
Guideline 3

- As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL
- If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation
- Using space efficiently and avoiding joins with NULL values are the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns with the appropriate key columns

- For example, if only 15 percent of employees have individual offices,there is little justification for including an attribute Office_number in the EMPLOYEE relation; rather, a relation EMP_OFFICES(Essn, Office_number) can be created to include tuples for only the employees with individual offices.

4.3.4 Generation of Spurious Tuples

- Consider the two relation schemas EMP_LOCS and EMP_PROJ1 which can be used instead of the single EMP_PROJ



- A tuple in EMP_LOCS means that the employee whose name is Ename works on *some project* whose location is Plocation
- A tuple in EMP_PROJ1 refers to the fact that the employee whose Social Security number is Ssn works Hours per week on the project whose name, number, and location are Pname, Pnumber, and Plocation.

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

- Suppose that we used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ. This produces a particularly bad schema design because we cannot recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS
- If we attempt a NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the result produces many more tuples than the original set of tuples in EMP_PROJ
- Additional tuples that were not in EMP_PROJ are called **spurious tuples** because they represent spurious information that is not valid.
- The spurious tuples are marked by asterisks (*)

	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

⋮

- Decomposing EMP_PROJ into EMP_LOCS and EMP_PROJ1 is undesirable because when we JOIN them back using NATURAL JOIN, we do not get the correct original information

- This is because in this case Plocation is the attribute that relates EMP_LOCS and EMP_PROJ1, and Plocation is neither a primary key nor a foreign key in either EMP_LOCS or EMP_PROJ1.

Guideline 4

- Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated
- Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

4.4 Functional Dependencies

- Formal tool for analysis of relational schemas that enables us to detect and describe some of the problems in precise terms

Definition of Functional Dependency

- A functional dependency is a constraint between two sets of attributes from the database.
- Given a relation R, a set of attributes X in R is said to **functionally determine** another attribute Y, also in R, (written X → Y) if and only if each X value is associated with at most one Y value.
- X is the determinant set and Y is the dependent attribute. Thus, given a tuple and the values of the attributes in X, one can determine the corresponding value of the Y attribute.
- The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.
- A functional dependency is a property of the semantics or meaning of the attributes.
- The database designers will use their understanding of the semantics of the attributes of R to specify the functional dependencies that should hold on all relation states (extensions) r of R.

- Consider the relation schema EMP_PROJ;

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

- From the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

a. Ssn→Ename

b. Pnumber →{Pname, Plocation}

c. {Ssn, Pnumber}→Hours

- These functional dependencies specify that

  (a) the value of an employee's Social Security number (Ssn) uniquely determines the employee name (Ename)

  (b) the value of a project's number (Pnumber) uniquely determines the project name (Pname) and location (Plocation), and

  (c) a combination of Ssn and Pnumber values uniquely determines the number of hours the employee currently works on the project per week (Hours).

- Alternatively,we say that Ename is functionally determined by (or functionally dependent on) Ssn, or *given a value of Ssn, we know the value of Ename,* and so on.
- Relation extensions r(R) that satisfy the functional dependency constraints are called **legal relation states** (or **legal extensions**) of R
- A functional dependency is a property of the relation schema R, not of a particular legal relation state r of R
- Therefore, an FD cannot be inferred automatically from a given relation extension r but must be defined explicitly by someone who knows the semantics of the attributes of R

Diagrammatic notation for displaying FDs

- Each FD is displayed as a horizontal line
- The left-hand-side attributes of the FD are connected by vertical lines to the line representing the FD
- The right-hand-side attributes are connected by the lines with arrows pointing toward the attributes.

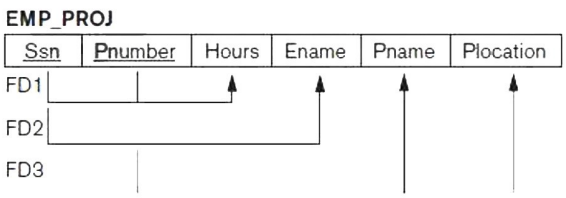


Fig: diagrammatic notation for displaying FDs

Example:

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

- The following FDs *may hold* because the four tuples in the current extension have no violation of these constraints:
  - $B \rightarrow C$
  - $C \rightarrow B$
  - $\{A, B\} \rightarrow C$
  - $\{A, B\} \rightarrow D$
  - $\{C, D\} \rightarrow B$.
- The following *do not* hold because we already have violations of them in the given extension:
  - $A \rightarrow B$ (tuples 1 and 2 violate this constraint)
  - $B \rightarrow A$ (tuples 2 and 3 violate this constraint)
  - $D \rightarrow C$ (tuples 3 and 4 violate it)

Normal Forms Based on Primary Keys

We assume that a

- Set of functional dependencies is given for each relation
- Each relation has a designated primary key
- This information combined with the tests (conditions) for normal forms drives the normalization process for relational schema design
- First three normal forms for relation takes into account all candidate keys of a relation rather than the primary key

4.4.1 Normalization of Relations

- The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to *certify* whether it satisfies a certain **normal form**.
- Initially, Codd proposed three normal forms, which he called first, second, and third normal form
- All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation
- A fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies, respectively
- **Normalization of data** can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of
  (1) minimizing redundancy and
  (2) minimizing the insertion, deletion, and update anomalies

- It can be considered as a "filtering" or "purification" process to make the design have successively better quality
- Unsatisfactory relation schemas that do not meet certain conditions—the **normal form tests**—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.
- Thus, the normalization procedure provides database designers with the following:
    - A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
    - A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree
- **Definition:** The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized

4.4.2 Practical Use of Normal Forms

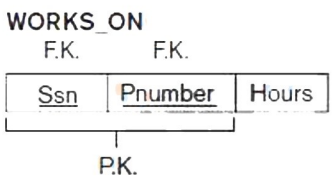
- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- Database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.
- The database designers *need not* normalize to the highest possible normal form
- Relations may be left in a lower normalization status, such as 2NF, for performance reasons
- **Definition: Denormalization** is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

4.4.3 Definitions of Keys and Attributes Participating in Keys

- **Superkey:** specifies a uniqueness constraint that no two distinct tuples in any state r of R can have the same value
- **key** K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey any more
- Example:
    - The attribute set {Ssn} is a key because no two employees tuples can have the same value for Ssn

•Any set of attributes that includes Ssn—for example, {Ssn, Name, Address}—is a superkey

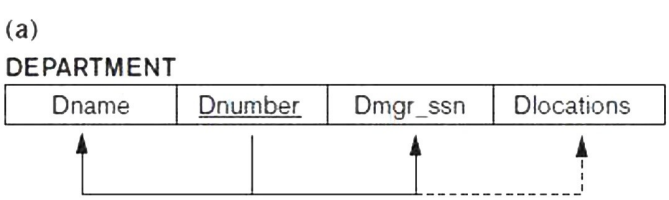
- If a relation schema has more than one key, each is called a **candidate key**
- One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called **secondary keys**
- In a practical relational database, each relation schema must have a primary key
- If no candidate key is known for a relation, the entire relation can be treated as a default superkey
- For example {Ssn} is the only candidate key for EMPLOYEE, so it is also the primary key
- **Definition.** An attribute of relation schema *R* is called a **prime attribute** of *R* if it is a member of *some candidate key* of *R*. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key



- In WORKS_ON relation Both Ssn and Pnumber are prime attributes whereas other attributes are nonprime.

4.4.4 First Normal Form

- Defined to disallow multivalued attributes, composite attributes, and their combinations
- It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute
- 1NF disallows relations within relations or relations as attribute values within tuples
- The only attribute values permitted by 1NF are single **atomic** (or **indivisible**) **values**.
- Consider the DEPARTMENT relation schema shown in Figure below



- Primary key is Dnumber
- We assume that each department can have a number of locations

- The DEPARTMENT schema and a sample relation state are shown in Figure below

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

- As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure
- There are two ways we can look at the Dlocations attribute:
  - The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is not functionally dependent on the primary key Dnumber
  - The domain of Dlocations contains sets of values and hence is nonatomic. In this case, Dnumber→Dlocations because each set is considered a single member of the attribute domain
- In either case, the DEPARTMENT relation is not in 1NF

There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination {Dnumber, Dlocation}. A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing *redundancy* in the relation

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

3. If a maximum number of values is known for the attribute—for example, if it is known that at most three locations can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing NULL values if most departments have fewer than three locations. Querying on this attribute becomes more difficult; forexample, consider how you would write the query: List the departments that have 'Bellaire' as one of their locations in this design.

- Of the three solutions, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values
- First normal form also disallows multivalued attributes that are themselves composite.
- These are called **nested relations** because each tuple can have a relation within it.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

- Figure above shows how the EMP_PROJ relation could appear if nesting is allowed
- Each tuple represents an employee entity, and a relation PROJS(Pnumber, Hours) *within each tuple* represents the employee's projects and the hours per week that employee works on each project.
- The schema of this EMP_PROJ relation can be represented as follows:

  EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})
- Ssn is the primary key of the EMP_PROJ relation and Pnumber is the **partial** key of the nested relation; that is, within each tuple, the nested relation must have unique values of Pnumber
- To normalize this into 1NF, we remove the nested relation attributes into a new relation and *propagate the primary key* into it; the primary key of the new relation will combine the partial key with the primary key of the original relation
- Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2,

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

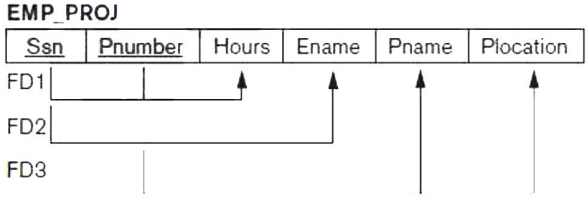
EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

4.4.5 Second Normal Form

- **Second normal form (2NF)** is based on the concept of full functional dependency
- A functional dependency X → Y is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A ε X, (X – {A}) does not functionally determine Y
- A functional dependency X→Y is a **partial dependency** if some attribute A ε X can be removed from X and the dependency still holds; that is, for some A ε X, (X – {A}) → Y

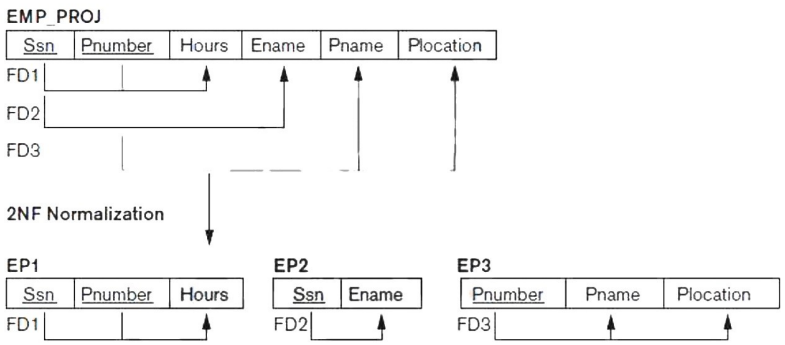
EMP_PROJ



- In the above figure , {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber→Hours holds)
- {Ssn, Pnumber}→Ename is partial because Ssn→Ename holds
- **Definition.** A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R

- The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key
- If the primary key contains a single attribute, the test need not be applied at all

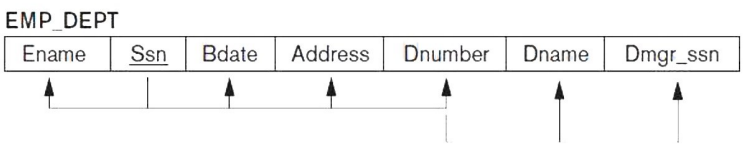
- The EMP_PROJ relation is in 1NF but is not in 2NF.
- The nonprime attribute Ename violates 2NF because of FD2, as do the nonprime attributes Pname and Plocation because of FD3
- The functional dependencies FD2 and FD3 make Ename, Pname, and Plocation partially dependent on the primary key {Ssn, Pnumber} of EMP_PROJ, thus violating the 2NF test.
- If a relation schema is not in 2NF, it can be *second normalized* or *2NF normalized* into a number of 2NF relations in which **nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.**
- Therefore, the functional dependencies FD1, FD2, and FD3 lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure below, each of which is in 2NF.



4.4.6 Third Normal Form

- **Transitive functional dependency**

  A functional dependency $X \rightarrow Y$ in a relation schema R is a **transitive dependency** if there exists a set of attribute Z that are neither a primary nor a subset of any key of R(candidate key) and both X → Z and Y → Z holds
- **Example:**

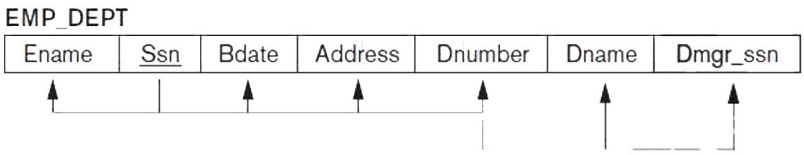


- **SSN → DMGRSSN is a transitive FD** since SSN → DNUMBER and DNUMBER → DMGRSSN hold

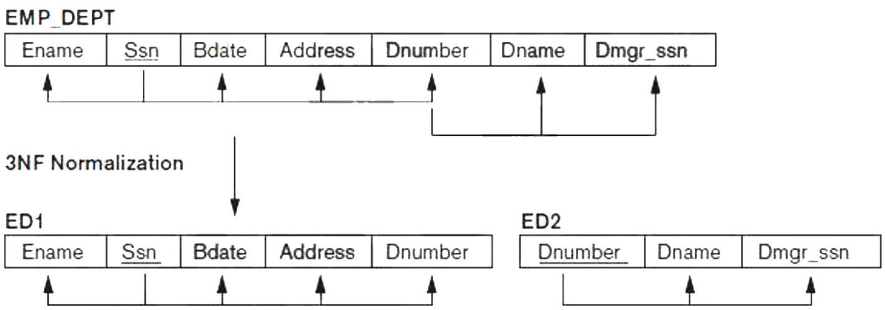
Dnumber is neither a key itself nor a subset of the key of EMP_DEPT

- **SSN → ENAME is non-transitive** since there is no set of attributes X where SSN → X and X → ENAME

- **Definition: A relation schema R is in third normal form (3NF) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key**
- The relation schema EMP_DEPT is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (and also Dname) on Ssn via Dnumber



- We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2



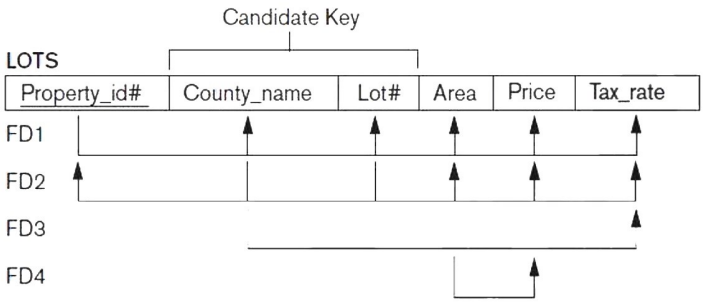
- ED1 and ED2 represent independent entity facts about employees and departments
- A NATURAL JOIN operation on ED1 and ED2 will recover the original relation EMP_DEPT without generating spurious tuples
- Problematic FD
  - Left-hand side is part of primary key
  - Left-hand side is a non-key attribute
- 2NF and 3NF normalization remove these problem FDs by decomposing the original relation into new relations
- In general, we want to design our relation schemas so that they have neither partial nor transitive dependencies because these types of dependencies cause the update anomalies

Table 15.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

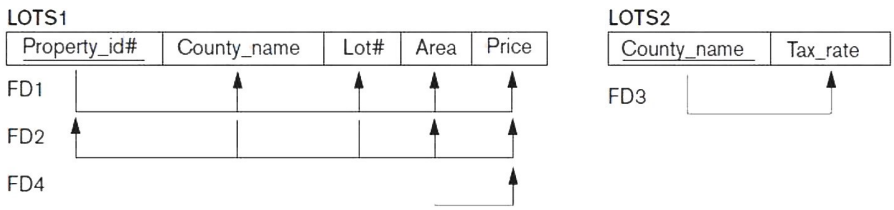
Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

4.5 General Definition of Second and Third Normal Form

- Takes into account all candidate keys of a relation into account
- **Definition of 2NF:** A relation schema R is in **second normal form (2NF)** if every nonprime attribute A in R is not partially dependent on any key of R
- Consider the relation schema LOTS which describes parcels of land for sale in various counties of a state
- Suppose that there are two candidate keys: Property_id# and {County_name, Lot#}; that is, lot numbers are unique only within each county, but Property_id# numbers are unique across counties for the entire state.

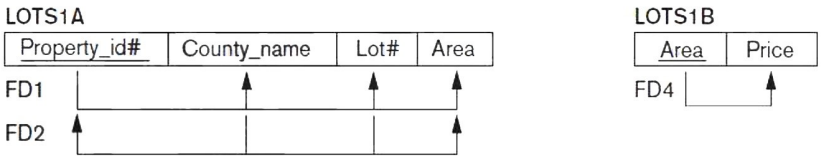


- Based on the two candidate keys Property_id# and {County_name, Lot#}, the functional dependencies FD1 and FD2 hold
  - FD1: Property_id → { County_name,Lot#,Area,Price,Tax_rate}
  - FD2:{County_name,Lot#} →{Property_id, Area,Price,Tax_rate}
  - FD3: County_name → Tax_rate
  - FD4: Area → Price
- We choose Property_id# as the primary key, but no special consideration will be given to this key over the other candidate key
- FD3 says that the tax rate is fixed for a given county (does not vary lot by lot within the same county)
- FD4 says that the price of a lot is determined by its area regardless of which county it is in.
- The LOTS relation schema violates the general definition of 2NF because Tax_rate is partially dependent on the candidate key {County_name, Lot#}, due to FD3
- To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2



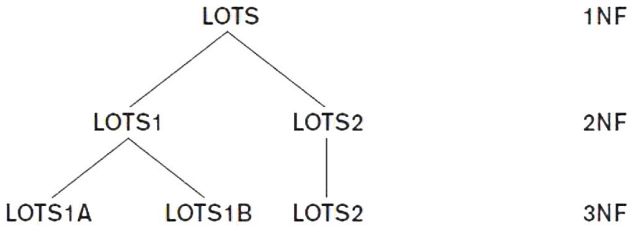
- We construct LOTS1 by removing the attribute Tax_rate that violates 2NF from LOTS and placing it with County_name (the left-hand side of FD3 that causes the partial dependency) into another relation LOTS2.
- Both LOTS1 and LOTS2 are in 2NF.

- **Definition of 3NF:** A relation schema R is in **third normal form (3NF)** if, whenever a nontrivial functional dependency X→A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R
- According to this definition, LOTS2 is in 3NF
- FD4 in LOTS1 violates 3NF because Area is not a superkey and Price is not a prime attribute in LOTS1
- To normalize LOTS1 into 3NF, we decompose it into the relation schemas LOTS1A and LOTS1B



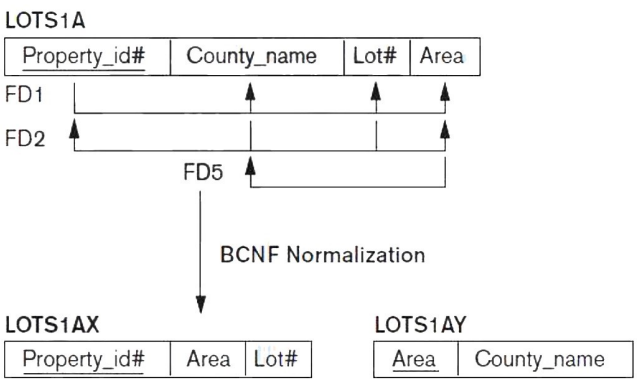
- We construct LOTS1A by removing the attribute Price that violates 3NF from LOTS1 and placing it with Area (the lefthand side of FD4 that causes the transitive dependency) into another relation LOTS1B.

- Both LOTS1A and LOTS1B are in 3NF



4.6 Boyce-Codd Normal Form

- **Boyce-Codd normal form (BCNF)** was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF
- Every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF
- **Definition.** A relation schema R is in **BCNF** if whenever a nontrivial functional dependency X→A holds in R, then X is a superkey of R
- The formal definition of BCNF differs from the definition of 3NF in that condition (b) of 3NF, which allows *A* to be prime, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF
- In our example, FD5 violates BCNF in LOTS1A because AREA is not a superkey of LOTS1A
- FD5 satisfies 3NF in LOTS1A because County_name is a prime attribute (condition b), but this condition does not exist in the definition of BCNF
- We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.



- In practice, most relation schemas that are in 3NF are also in BCNF
- Only if *X*→*A* holds in a relation schema *R* with *X* not being a superkey *and A* being a prime attribute will *R* be in 3NF but not in BCNF
- Example: consider the relation TEACH with the following dependencies:

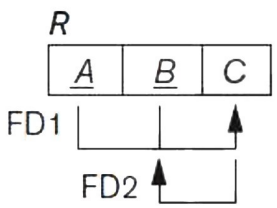
TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

FD1: {Student, Course} → Instructor

FD2: Instructor → Course -- means that each instructor teaches one course

- {Student, Course} is a candidate key for this relation
- The dependencies shown follow the pattern in Figure below with Student as *A*, Course as *B*, and Instructor as *C*



- Hence this relation is in 3NF but not BCNF
- Decomposition of this relation schema into two schemas is not straightforward because it may be decomposed into one of the three following possible pairs:

  1. R1(Student, Instructor) and R2(Student, Course)

  2. R1(Course, Instructor) and R2(Course, Student)

  3. R1(Instructor, Course)and R2(Instructor, Student)
- It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF
- Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:
  - The **nonadditive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
  - The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

- We are not able to meet the functional dependency preservation ,but we must meet the non additive join property
- **Nonadditive Join Test for Binary Decomposition:**

  A decomposition D={R_1, R_2} of R has the lossless join property with respect to a set of functional dependencies F on R if and only if either

  - The FD (($R_1 \cap R_2$) →(R_1-R_2)is in F^+ or
  - The FD (($R_1 \cap R_2$) →(R_2-R_1)is in F^+
- The third decomposition meets the test

  $R_1 \cap R_2$ is Instructor

  R_1-R_2 is Course
- Hence, the proper decomposition of TEACH into BCNF relations is:

  TEACH1(Instructor,Course) and TEACH2(Instructor,Student)
- In general, a relation R not in BCNF can be decomposed so as to meet the nonadditive join prorperty by the following procedure. It decomposes R successively into set of relations that are in BCNF:

  Let R be the relation not in BCNF, let X R, and let X → A be the FD that causes violation of BCNF. R may be decomposed into two relations:

  R – A

  XA

  If either R-A or XA is not in BCNF, repeat the process

4.7 Multivalued Dependency and Fourth Normal Form

- For example, consider the relation EMP shown in Figure below:

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

- A tuple in this EMP relation represents the fact that an employee whose name is Ename works on the project whose name is Pname and has a dependent whose name is Dname
- An employee may work on several projects and may have several dependents
- The employee's projects and dependents are independent of one another

- To keep the relation state consistent, and to avoid any spurious relationship between the two independent attributes, we must have a separate tuple to represent every combination of an employee's dependent and an employee's project
- In the relation state shown in the EMP, the employee Smith works on two projects 'X' and 'Y' and has two dependents 'John' and 'Anna' and therefore there are 4 tuples to represent these facts together
- The relation EMP is an **all-key relation** (with key made up of all attributes) and therefore no f.d's and as such qualifies to be a BCNF relation
- There is a redundancy in the relation EMP-the dependent information is repeated for every project and project information is repeated for every dependent
- To address this situation, the concept of multivalued dependency(MVD) was proposed and based on this dependency, the fourth normal form was defined
- **Multivalued dependencies** are a consequence of 1NF which disallows an attribute in a tuple to have a set of values, and the accompanying process of converting an unnormalized relation into 1NF
- Informally, whenever two independent 1:N relationships are mixed in the same relation, R(A, B, C), an MVD may arise

4.7.1 Formal Definition of Multivalued Dependency

Definition. A multivalued dependency X→→Y specified on relation schema R, where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t1 and t2 exist in r such that t1[X] = t2[X], then two tuples t3 and t4 should also exist in r with the following properties where we use Z to denote (R – (X ∪ Y))

- t3[X] = t4[X] = t1[X] = t2[X].
- t3[Y] = t1[Y] and t4[Y] = t2[Y].
- t3[Z] = t2[Z] and t4[Z] = t1[Z].

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Let X= Ename, Y=Pname
t1[Ename]=t2[ename]=Smith
Z= (EMP-(Ename u Pname))
= Dname

- t3(Ename)=t4(Ename)=t1(Ename)=t2(Ename)=Smith

- t3(Pname)=t1(Pname)=X and t4(Pname)=t2(Pname)=Y
- t3(Dname)=t2(Dname)=Anna and t4(Dname)=t1(Dname)=John
- Whenever $X \rightarrow\rightarrow Y$ holds, we say that X **multidetermines** Y. Because of the symmetry in the definition, whenever $X \rightarrow\rightarrow Y$ holds in R, so does $X \rightarrow\rightarrow Z$. Hence, $X \rightarrow\rightarrow Y$ implies $X \rightarrow\rightarrow Z$, and therefore it is sometimes written as $X \rightarrow\rightarrow Y|Z$.
- An MVD X →→ Y in R is called a **trivial MVD** if
  (a) Y is a subset of X, or
  (b) X ∪ Y = R

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

- For example, the relation EMP_PROJECTS has the trivial MVD
  Ename →→ Pname
- An MVD that satisfies neither (a) nor (b) is called a **nontrivial MVD**
- If we have a nontrivial MVD in a relation, we may have to repeat values redundantly in the tuples
- In the EMP relation the values 'X' and 'Y' of Pname are repeated with each value of Dname (or, by symmetry, the values 'John' and 'Anna' of Dname are repeated with each value of Pname)
- This redundancy is clearly undesirable.
- We now present the definition of **fourth normal form (4NF)**, which is violated when a relation has undesirable multivalued dependencies, and hence can be used to identify and decompose such relations
- **Definition: A relation schema *R* is in 4NF with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \rightarrow\rightarrow Y$ in F^+ *X* is a superkey for *R***
- The process of normalizing a relation involving the nontrivial MVDs that is not in 4NF consists of decomposing it so that each MVD is represented by a separate relation where it becomes a trivial MVD

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

- We decompose EMP into EMP_PROJECTS and EMP_DEPENDENTS
- Both EMP_PROJECTS and EMP_DEPENDENTS are in 4NF, because the MVDs Ename →→ Pname in EMP_PROJECTS and Ename →→ Dname in EMP_DEPENDENTS are trivial MVDs
- No other nontrivial MVDs hold in either EMP_PROJECTS or EMP_DEPENDENTS. No FDs hold in these relation schemas either

- We can state the following points:
  - An all-key relation is always in BCNF since it has no FDs
  - An all-key relation such as the EMP, which has no FDs but has the MVD Ename→→ Pname | Dname, is not in 4NF
  - A relation that is not in 4NF due to a nontrivial MVD must be decomposed to convert it into a set of relations in 4NF
  - The decomposition removes the redundancy caused by the MVD

4.8 Join Dependencies and Fifth Normal Form

- A **join dependency** (**JD**), denoted by JD(R1, R2, ..., Rn), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a nonadditive join decomposition into R1, R2, ..., Rn. Hence, for every such r we have

$$* (\pi_{R_1}(r), \pi_{R_2}(r), ..., \pi_{R_n}(r)) = r$$

- A join dependency JD(R1, R2, ..., Rn), specified on relation schema R, is a **trivial** JD if one of the relation schemas Ri in JD(R1, R2, ..., Rn) is equal to R.

Fifth normal form (project-join normal form)

- A relation schema *R* is in **fifth normal form (5NF)** (or **project-join normal form (PJNF)**) with respect to a set *F* of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(*R*1, *R*2, ..., *Rn*) in F^+ every R_i is a superkey of *R*.
- A database is said to be in 5NF, if and only if,
  - It's in 4NF

- If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

Fig: The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(*R*1, *R*2, *R*3)

R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Fig: Decomposing the relation SUPPLY into the 5NF relations *R*1, *R*2, *R*3.

SQL

4.1 Introduction

SQL was called SEQUEL (Structured English Query Language) and was designed and implemented at IBM Research.The SQL language may be considered one of the major reasons for the commercial success of relational databases. SQL is a comprehensive database language. It has statements for data definitions, queries, and updates. Hence, it is both a DDL *and* a DML. In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls. It also has rules for embedding SQL statements into a general-purpose programming language such as Java, COBOL, or C/C++.

4.2 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. The main SQL command for data definition is the CREATE statement, which can be used to create schemas, tables (relations), domains, views, assertions and triggers.

4.2.1 Schema and Catalog Concepts in SQL

An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for *each element* in the schema. Schema elements include tables, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema. A schema is created via the CREATE SCHEMA statement .

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

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

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

SQL uses the concept of a **catalog**—a named collection of schemas in an SQL environment. A catalog always contains a special schema called INFORMATION_SCHEMA, which provides information on all the schemas in the catalog and all the element descriptors in these

schemas. Integrity constraints such as referential integrity can be defined between relations only if they exist in schemas within the same catalog. Schemas within the same catalog can also share certain elements, such as domain definitions.

4.2.2 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.

Typically, the SQL schema in which the relations are declared is implicitly specified in the environment in which the CREATE TABLE statements are executed. Alternatively, we can explicitly attach the schema name to the relation name, separated by a period. For example, by writing

CREATE TABLE COMPANY.EMPLOYEE ...

rather than

CREATE TABLE EMPLOYEE ...

The relations declared through CREATE TABLE statements are called **base tables.**

Examples:

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

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

4.2.3 Attribute Data Types and Domains in SQL

Basic data types

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

2. **Character-string data types**
   - fixed length—CHAR(*n*) or CHARACTER(*n*), where *n* is the number of characters
   - varying length—VARCHAR(*n*) or CHAR VARYING(*n*) or CHARACTER VARYING(*n*), where *n* is the maximum number of characters
   - When specifying a literal string value, it is placed between single quotation marks (apostrophes), and it is *case sensitive*
   - For fixed length strings, a shorter string is padded with blank characters to the right
   - For example, if the value 'Smith' is for an attribute of type CHAR(10), it is padded with five blank characters to become 'Smith ' if needed
   - 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** data types are either of
   - fixed length *n*—BIT(*n*)—or varying length—BIT VARYING(*n*), where *n* is the maximum number of bits.
   - The default for *n*, the length of a character string or bit string, is 1.

- Literal bit strings are placed between single quotes but preceded by a B to 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.
- The maximum length of a BLOB can be specified in kilobits (K), megabits (M), or gigabits (G)
- For example, BLOB(30G) specifies a maximum length of 30 gigabits.

4. **A Boolean** data type has the traditional values of TRUE or FALSE.In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN
5. The **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD
6. The **TIME data** type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS.
   Only valid dates and times should be allowed by the SQL implementation.
7. **TIME WITH TIME ZONE** data type includes an additional six positions for specifying the displacement from the standard universal time zone, which is in the range +13:00 to –12:59 in units of HOURS:MINUTES. If WITH TIME ZONE is not included, the default is the local time zone for the SQL session.

Additional data types

1. **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.
2. **INTERVA**L data type. This specifies an **interval**—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp. Intervals are qualified to be either YEAR/MONTH intervals or DAY/TIME intervals.

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

CREATE DOMAIN SSN_TYPE **AS** CHAR(9);

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

4.3 Specifying Constraints in SQL

Basic constraints that can be specified in SQL as part of table creation:

- key and referential integrity constraints
- Restrictions on attribute domains and NULLs
- constraints on individual tuples within a relation

4.3.1 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. This is always implicitly specified for the attributes that are part of the primary key of each relation, but it can be specified for any other attributes whose values are required not to be NULL.

It is also possible to define a default value for an attribute by appending the clause **DEFAULT** <value> to an attribute definition. The default value is included in any new tuple if an explicit value is not provided for that attribute.

```
CREATE TABLE DEPARTMENT
( . . . ,
    Mgr_ssn CHAR(9) NOT NULL DEFAULT '888665555',
    ----------------
    ----------------
)
```

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);

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

CREATE DOMAIN D_NUM **AS** INTEGER

CHECK (D_NUM > 0 **AND** D_NUM < 21);

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

4.3.2 Specifying Key and Referential Integrity Constraints

The **PRIMARY KEY** clause specifies one or more attributes that make up the primary key of a relation. If a primary key has a single attribute, the clause can follow the attribute directly. For example, the primary key of DEPARTMENT can be specified as:

Dnumber INT **PRIMARY KEY**;

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**;

Referential integrity is specified via the **FOREIGN KEY** clause

FOREIGN KEY (Super_ssn) **REFERENCES** EMPLOYEE(Ssn),
FOREIGN KEY (Dno) **REFERENCES** DEPARTMENT(Dnumber

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**(Dno) **REFERENCES** DEPARTMENT(Dnumber) **ON DELETE** SET DEFAULT **ON UPDATE** CASCADE
- **FOREIGN KEY** (Super_ssn) **REFERENCES** EMPLOYEE(Ssn) **ON DELETE** SET NULL **ON UPDATE** CASCADE
- **FOREIGN KEY** (Dnumber) **REFERENCES** DEPARTMENT(Dnumber) **ON DELETE** CASCADE **ON UPDATE** CASCADE

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 . It is the responsibility of the database designer to choose the appropriate action and to specify it in the database schema. As a general rule, the CASCADE option is suitable for "relationship" relations such as WORKS_ON; for relations that represent multivalued attributes, such as DEPT_LOCATIONS; and for relations that represent weak entity types, such as DEPENDENT.

4.3.3 Giving Names to Constraints

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.

4.3.4 Specifying Constraints on Tuples Using CHECK

In addition to key and referential integrity constraints, which are specified by special keywords, other *table constraints* can be specified through additional CHECK clauses at the end of a CREATE TABLE statement. These can be called **tuple-based** constraints because they apply to each tuple individually and are checked whenever a tuple is inserted or modified

For example, suppose that the DEPARTMENT table had an additional attribute Dept_create_date, which stores the date when the department was created. Then we could add the following CHECK clause at the end of the CREATE TABLE statement for the DEPARTMENT table to make sure that a manager's start date is later than the department creation date

CHECK (Dept_create_date <= Mgr_start_date);

4.4 Basic Retrieval Queries in SQL

SQL has one basic statement for retrieving information from a database: the **SELECT** statement.

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

The basic form of the SELECT statement, sometimes called a **mapping** or a **select-from-where block**, 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.

Examples:

1. Retrieve the birth date 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';
```

The SELECT clause of SQL specifies the attributes whose values are to be retrieved, which are called the **projection attributes.** The WHERE clause specifies the Boolean condition that must be true for any retrieved tuple, which is known as the **selection condition.**

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;
```

In the WHERE clause, the condition Dname = 'Research' is a **selection condition** that chooses the particular tuple of interest in the DEPARTMENT table, because Dname is an attribute of DEPARTMENT. The condition Dnumber = Dno is called a **join condition**, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to thevalue of Dno in EMPLOYEE.A query that involves only selection and join conditions plus projection attributes is known as a **select-project-join** 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 Dnum=Dnumber **AND** Mgr_ssn=Ssn **AND** Plocation='Stafford';

The join condition Dnum = Dnumber relates a project tuple to its controlling department tuple, whereas the join condition Mgr_ssn = Ssn relates the controlling department tuple to the employee tuple who manages that department. Each tuple in the result will be a *combination* of one project, one department, and one employee that satisfies the join conditions. The projection attributes are used to choose the attributes to be displayed from each combined tuple.

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

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

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

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

The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice. For example consider the query: 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;

In this case, we are required to declare alternative relation names E and S, called **aliases** or **tuple variables**, for the EMPLOYEE relation. An alias can follow the keyword **AS**, or it can directly follow the relation name—for example, by writing EMPLOYEE E, EMPLOYEE S. It is also possible to **rename** the relation attributes within the query in SQL by giving them aliases. For example, if we write

EMPLOYEE **AS** E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)

in the FROM clause, Fn becomes an alias for Fname, Mi for Minit, Ln for Lname, and so on

4.4.3 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.

Example: Select all EMPLOYEE Ssns and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname in the database.

```
SELECT Ssn
FROM EMPLOYEE;
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, the following query retrieves all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5

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

SELECT * FROM EMPLOYEE, DEPARTMENT WHERE Dname='Research'
AND Dno=Dnumber;

SELECT * FROM EMPLOYEE, DEPARTMENT;
```

4.4.4 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:

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

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

Example : Retrieve the salary of every employee and all distinct salary values

(a) SELECT ALL Salary **FROM** EMPLOYEE;

(b) SELECT DISTINCT Salary **FROM** EMPLOYEE;

(a)

Salary
30000
40000
25000
43000
38000
25000
25000
55000

(b)

Salary
30000
40000
25000
43000
38000
55000

SQL has directly incorporated some of the set operations from mathematical *set theory*, which are also part of relational algebra. There are

- set union (**UNION**)
- set difference (**EXCEPT**) and
- set intersection (**INTERSECT**)

The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminated from the result. These set operations apply only to union-compatible relations, so we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations.

Example: 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' );
```

4.4.5 Substring Pattern Matching and Arithmetic Operators

Several more features of SQL

The first feature allows comparison conditions on only parts of a character string, using the **LIKE** comparison operator. This can be used for string **pattern matching**. Partial strings are specified using two reserved characters:

- % replaces an arbitrary number of zero or more characters
- _ (underscore) replaces a single character

For example, consider the following query: Retrieve all employees whose address is in Houston, Texas

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

To retrieve all employees who were born during the 1950s, we can use Query

```
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 using the keyword ESCAPE. For example, 'AB_CD\%EF' ESCAPE '\' represents the literal string 'AB_CD%EF' because \ is specified as the escape character.Also, we need a rule to specify apostrophes or single quotation marks (' ') if they are to be included in a string because they are used to begin and end strings. If an apostrophe (') is needed, it is represented as two consecutive apostrophes (") so that it will not be interpreted as ending the string.

Another feature allows the use of arithmetic in queries.The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains. For example,suppose that we want to see the effect of giving all employees who work on the 'ProductX' project a 10 percent raise; we can issue the following query:

```
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';
```

Example: Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

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

The condition (Salary **BETWEEN** 30000 **AND** 40000) is equivalent to the condition((Salary >= 30000) **AND** (Salary <= 40000)).

4.4.6 Ordering of Query Results

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

Example:Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically bylast 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;

The default order is in ascending order of values.We can specify the keyword **DESC** if we want to see the result in a descending order of values. The keyword ASC can be used to specify ascending order explicitly. For example, if we want descending alphabetical order on Dname and ascending order on Lname, Fname, the ORDER BY clause can be written as

ORDER BY D.Dname DESC, E.Lname ASC, E.Fname **ASC**

4.5 INSERT, DELETE, and UPDATE Statements in SQL

4.5.1The 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. The values should be listed *in the same order* in which the corresponding attributes were specified in the CREATE TABLE command.

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

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

A second form of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command. The values must include all attributes with NOT NULL specification and no default value. Attributes with NULL allowed or DEFAULT values are the ones that can be left out.

A variation of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the *result of a query.* For example, to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project, we can write the statements in U3A and U3B:

U3A: CREATE TABLE WORKS_ON_INFO(
Emp_name VARCHAR(15),
Proj_name VARCHAR(15),
Hours_per_week DECIMAL(3,1));

U3B: INSERT INTO WORKS_ON_INFO
(Emp_name, Proj_name,Hours_per_week)
SELECT E.Lname, P.Pname, W.Hours
FROM PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE P.Pnumber=W.Pno **AND** W.Essn=E.Ssn;

A table WORKS_ON_INFO is created by U3A and is loaded with the joined information retrieved from the database by the query in U3B. We can now query WORKS_ON_INFO as we would any other relation;

4.5.2 The DELETE Command

The DELETE command removes tuples from a relation. It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted. Tuples are explicitly deleted from only one table at a time. The deletion may propagate to tuples in other relations if *referential triggered actions* are specified in the referential integrity constraints of the DDL.

Example:

DELETE FROM EMPLOYEE **WHERE** Lname='Brown';

Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command. A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table.

4.5.3 The UPDATE Command

The UPDATE command is used to modify attribute values of one or more selected Tuples.An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values. For example, to change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively, we use

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

As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation. However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL.

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, as shown by the following query

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

Each UPDATE command explicitly refers to a single relation only. To modify multiple relations, we must issue several UPDATE commands.

4.6 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.
- 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.
- 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