

# Database Management Systems (BCS403) - 2023-24 - Module 3

Dr. Narender M  
Department of CS&E  
The National Institute of Engineering

# Topics

## **Introduction to Normalization using Functional and Multivalued Dependencies**

- Informal design guidelines for relation schema
- Functional Dependencies
- Normal Forms based on Primary Keys
- Second and Third Normal Forms
- Boyce-Codd Normal Form
- Multivalued Dependency and Fourth Normal Form
- Join Dependencies and Fifth Normal Form



# Topics

## **SQL**

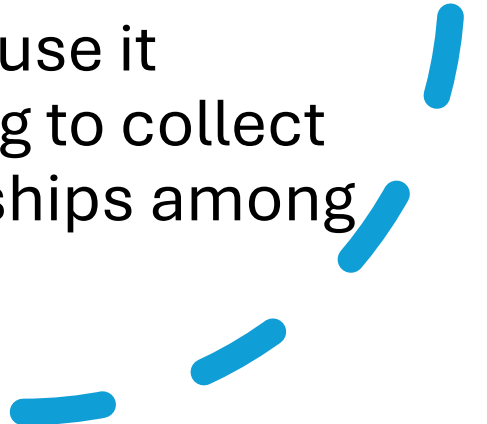
- SQL data definition and data types
- Schema change statements in SQL
- Specifying constraints in SQL
- Retrieval queries in SQL
- INSERT, DELETE, and UPDATE statements in SQL
- Additional features of SQL

# Introduction to Normalization using Functional and Multivalued Dependencies

- 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.
- We need some formal way of analyzing why one grouping of attributes into a relation schema may be better than another.
- Goodness of relation schemas: The first is the logical (or conceptual) level—how users interpret the relation schemas and the meaning of their attributes.

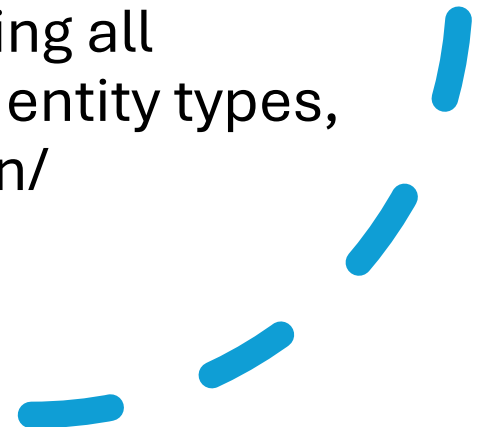


# Introduction to Normalization using Functional and Multivalued Dependencies

- The second is the implementation (or physical storage) level—how the tuples in a base relation are stored and updated.
  - Database design may be performed using two approaches: bottom-up or top-down.
  - A bottom-up design methodology (also called design by synthesis) considers the basic relationships among individual attributes as the starting point and uses those to construct relation schemas.
  - Not very popular in practice because it suffers from the problem of having to collect a large number of binary relationships among attributes as the starting point.
- 

# Introduction to Normalization using Functional and Multivalued Dependencies

- A top-down design methodology (also called design by analysis) starts with a number of groupings of attributes into relations that exist together naturally.
- The relations are then analyzed individually and collectively, leading to further decomposition until all desirable properties are met.
- The implicit goals of the design activity are information preservation and minimum redundancy.
- Information preservation - maintaining all concepts, including attribute types, entity types, relationship types and generalization/specialization relationships.



# Introduction to Normalization using Functional and Multivalued Dependencies

- The relational design must preserve all of these concepts, which are originally captured in the conceptual design.
- Minimizing redundancy implies minimizing redundant storage of the same information and reducing the need for multiple updates to maintain consistency across multiple copies of the same information.



# Informal design guidelines for relation schema

- Four informal guidelines that may be used as measures to determine the quality of relation schema design:
  - Making sure that the semantics of the attributes is clear in the schema
  - Reducing the redundant information in tuples
  - Reducing the NULL values in tuples
  - Disallowing the possibility of generating spurious tuples



# Informal design guidelines for relation schema

## 1. Imparting Clear Semantics to Attributes in Relations

- Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning.
- The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.
- The meaning of the EMPLOYEE relation schema is simple: Each tuple represents an employee, with values for the employee's name (Ename), Social Security number (Ssn), birth date (Bdate), and address (Address), and the number of the department that the employee works for (Dnumber).

# Informal design guidelines for relation schema

- The ease with which the meaning of a relation's attributes can be explained is an informal measure of how well the relation is designed.

EMPLOYEE					F.K.
Ename	<u>Ssn</u>	Bdate	Address	Dnumber	

P.K.

DEPARTMENT			F.K.
Dname	<u>Dnumber</u>	Dmgr_ssn	

P.K.

DEPT_LOCATIONS		F.K.
<u>Dnumber</u>	<u>Dlocation</u>	

P.K.

PROJECT				F.K.
Pname	<u>Pnumber</u>	Plocation	Dnum	

P.K.

WORKS_ON			F.K.	F.K.
<u>Ssn</u>	<u>Pnumber</u>	Hours		

P.K.

# Informal design guidelines for relation schema

## **Guideline 1**

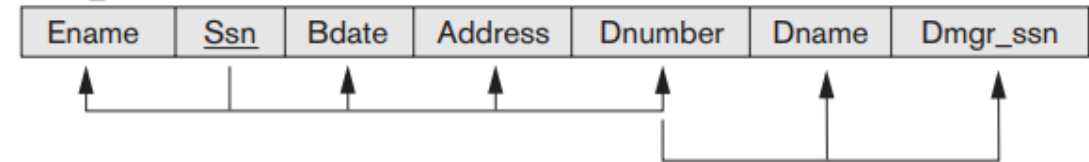
- Design a relation schema so that it is easy to explain its meaning.
- Do not combine attributes from multiple entity types and relationship types into a single relation.
- If a relation schema corresponds to one entity type or one relationship type, it is straightforward to explain its meaning.
- Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result, and the relation cannot be easily explained.

# Informal design guidelines for relation schema

- A tuple in the EMP\_DEPT relation schema represents a single employee but includes, along with the Dnumber and additional information.
- They violate Guideline 1 by mixing attributes from distinct real-world entities: EMP\_DEPT mixes attributes of employees and departments, and EMP\_PROJ mixes attributes of employees and projects and the WORKS\_ON relationship.

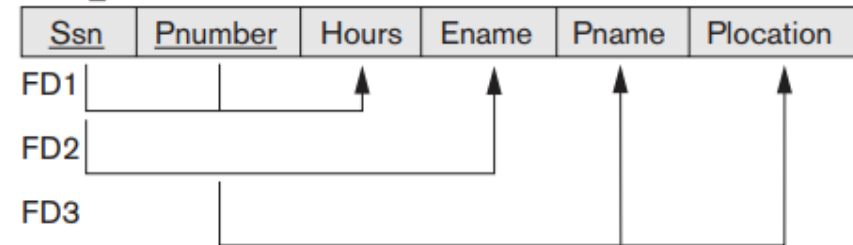
(a)

EMP\_DEPT



(b)

EMP\_PROJ

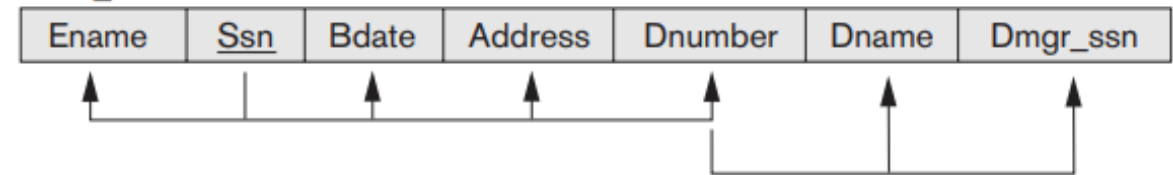


# Informal design guidelines for relation schema

- Hence, they fare poorly against the above measure of design quality.
- They may be used as views, but they cause problems when used as base relations

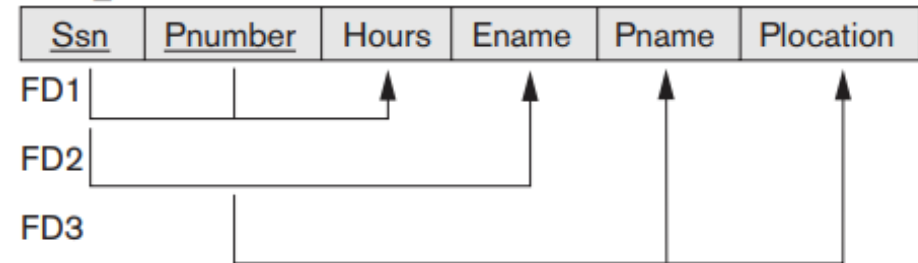
(a)

EMP\_DEPT



(b)

EMP\_PROJ



# Informal design guidelines for relation schema

## **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.
- 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, and modification anomalies.

					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					
<u>Ssn</u>	<u>Pnumber</u>	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



**EMPLOYEE**

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

**DEPARTMENT**

Dname	<u>Dnumber</u>	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

**DEPT\_LOCATIONS**

<u>Dnumber</u>	<u>Dlocation</u>
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston



# Informal design guidelines for relation schema

- Insertion Anomalies
  - 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
  - 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 its primary key Ssn cannot be null.
- Deletion Anomalies
  - 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 inadvertently from the database.

# Informal design guidelines for relation schema

- 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.
- Three anomalies are undesirable and cause difficulties to maintain consistency of data as well as require unnecessary updates.

# Informal design guidelines for relation schema

## **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.
- These guidelines may sometimes have to be violated in order to improve the performance of certain queries.

# Informal design guidelines for relation schema

## 3. NULL Values in Tuples

- If many of the attributes in a ‘fat’ relation 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 and may also lead to problems with understanding the meaning of the attributes.
- Another problem with NULLs is 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.

# Informal design guidelines for relation schema

- 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.

# Informal design guidelines for relation schema

## 4. Generation of Spurious Tuples

- Because of bad schema design we might not be able to recover the information that was originally in relations.
- 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.



**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

Informal design guidelines for  
relation schema

# Informal design guidelines for relation schema

	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.

\*  
\*  
\*



# Informal design guidelines for relation schema

- This is because in this case Plocation happens to be 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.

# Functional Dependencies

---

- A formal tool for analysis of relational schemas that enables us to detect and describe some of the above-mentioned problems in precise terms.
- The single most important concept in relational schema design theory is that of a functional dependency.

## **Definition of Functional Dependency**

- A functional dependency, denoted by  $X \rightarrow Y$ , between two sets of attributes  $X$  and  $Y$  that are subsets of  $R$  specifies a constraint on the possible tuples that can form a relation state  $r$  of  $R$ . The constraint is that, for any two tuples  $t1$  and  $t2$  in  $r$  that have  $t1[X] = t2[X]$ , they must also have  $t1[Y] = t2[Y]$ .

# Functional Dependencies

---

- This means that the values of the Y component of a tuple in  $r$  depend on, or are determined by, the values of the X component.
- Alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component.
- We also say that there is a functional dependency from X to Y, or that Y is functionally dependent on X.
- Relation extensions  $r(R)$  that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R.

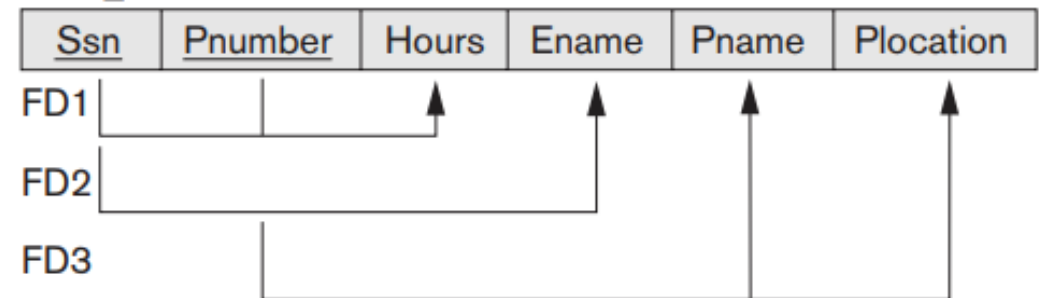
# Functional Dependencies

---

- Consider the relation schema EMP\_PROJ; from the semantics of the attributes and the relation, we know that the following functional dependencies should hold:
  - $Ssn \rightarrow Ename$
  - $Pnumber \rightarrow \{Pname, Plocation\}$
  - $\{Ssn, Pnumber\} \rightarrow Hours$

(b)

EMP\_PROJ



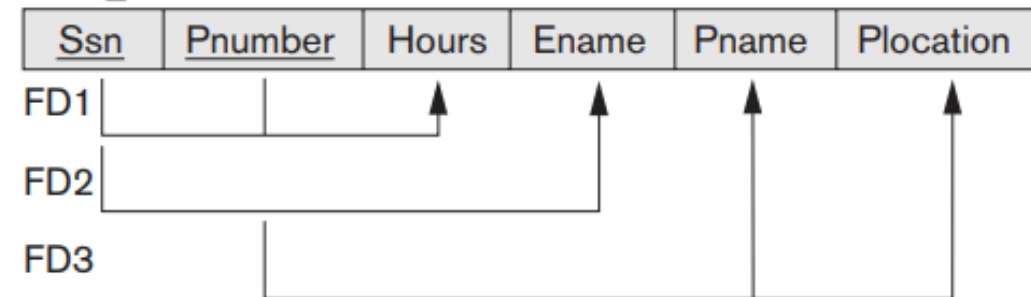
# Functional Dependencies

---

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

(b)

EMP\_PROJ



# Functional Dependencies

**TEACH**

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

**Figure 14.7**

A relation state of TEACH with a *possible* functional dependency  $\text{TEXT} \rightarrow \text{COURSE}$ . However,  $\text{TEACHER} \rightarrow \text{COURSE}$ ,  $\text{TEXT} \rightarrow \text{TEACHER}$  and  $\text{COURSE} \rightarrow \text{TEXT}$  are ruled out.

- We may think that  $\text{Text} \rightarrow \text{Course}$ , we cannot confirm this unless we know that it is true for all possible legal states of TEACH.
- It is sufficient to demonstrate a single counterexample to disprove a functional dependency.
- For example, because 'Smith' teaches both 'Data Structures' and 'Database Systems,' we can conclude that Teacher does not functionally determine Course.

# Functional Dependencies

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:  $B \rightarrow C$ ;  $C \rightarrow B$ ;  $\{A, B\} \rightarrow C$ ;  $\{A, B\} \rightarrow D$ ; and  $\{C, D\} \rightarrow B$ .
- 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

---

## Normalization of Relations

- The normalization process, as first proposed by Codd, takes a relation schema through a series of tests to certify whether it satisfies a certain normal form.
- The process, which proceeds in a top-down fashion by evaluating each relation against the criteria for normal forms and decomposing relations as necessary, can thus be considered as relational design by analysis.
- Codd proposed three normal forms, which he called first, second, and third normal form.



# Normal Forms based on Primary Keys

---

- A stronger definition of 3NF—called Boyce-Codd normal form (BCNF)—was proposed later by Boyce and Codd.
- All these normal forms are based on a single analytical tool: the functional dependencies among the attributes of a relation.
- Later, a fourth normal form (4NF) and a fifth normal form (5NF) were proposed, based on the concepts of multivalued dependencies and join dependencies.

# Normal Forms based on Primary Keys

---

- 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
  - (2) minimizing the insertion, deletion, and update anomalies
- A relation schema that does not meet the condition for a normal form is decomposed into smaller relation schemas that contain a subset of the attributes and meet the test that was otherwise not met by the original relation.

# Normal Forms based on Primary Keys

---

- 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.
- The process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas.
- 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.

# Normal Forms based on Primary Keys

---

## **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.
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are hard to understand or to detect.
- The database designers need not normalize to the highest possible normal form (usually up to 3NF, BCNF or 4NF)
- Denormalization: The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form.

# Normal Forms based on Primary Keys

---

## Definitions of Keys and Attributes Participating in Keys

- A superkey of a relation schema  $R = \{A_1, A_2, \dots, A_n\}$  is a set of attributes  $S$  subset-of  $R$  with the property that no two tuples  $t_1$  and  $t_2$  in any legal relation state  $r$  of  $R$  will have  $t_1[S] = t_2[S]$ .
- A key  $K$  is a superkey with the additional property that removal of any attribute from  $K$  will cause  $K$  not to be a superkey anymore.
- 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.

# Normal Forms based on Primary Keys

---

- A Prime attribute must be a member of some candidate key
- A Nonprime attribute is not a prime attribute—that is, it is not a member of any candidate key.

## **First Normal Form**

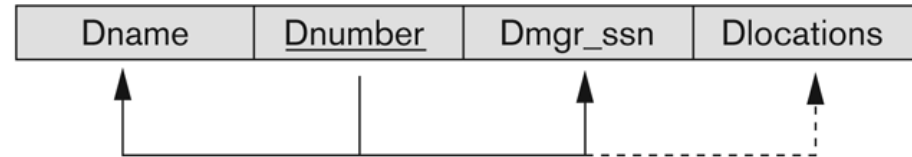
- Disallows
  - composite attributes
  - multivalued attributes
  - nested relations; attributes whose values for an individual tuple are non-atomic
- Considered to be part of the definition of relation and all relations in a relational schema are in 1NF by default.

# Normal Forms based on Primary Keys

---

(a)

DEPARTMENT



(b)

DEPARTMENT

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

(c)

DEPARTMENT

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

**Figure 10.8**

Normalization into 1NF.

(a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

# Normal Forms based on Primary Keys

(a)

EMP_PROJ		Projs	
Ssn	Ename	Pnumber	Hours

(b)

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, AliciaJ.	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

(c)

EMP\_PROJ1

<u>Ssn</u>	Ename
------------	-------

EMP\_PROJ2

<u>Ssn</u>	<u>Pnumber</u>	Hours
------------	----------------	-------

**Figure 10.9**

Normalizing nested relations into 1NF. (a) Schema of the EMP\_PROJ relation with a *nested relation* attribute PROJS. (b) Example extension of the EMP\_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP\_PROJ into relations EMP\_PROJ1 and EMP\_PROJ2 by propagating the primary key.



# Second Normal Form

---

- Uses the concepts of FDs and primary key.
- Definition:
  - Full functional dependency: a FD  $Y \rightarrow Z$  where removal of any attribute from  $Y$  means the FD does not hold any more.
  - A functional dependency  $X \rightarrow Y$  is a partial dependency if some attribute  $A \in X$  can be removed from  $X$  and the dependency still holds; that is, for some  $A \in X$ ,  $(X - \{A\}) \rightarrow Y$ .
- Examples:
  - $\{SSN, PNUMBER\} \rightarrow HOURS$  is a full FD since neither  $SSN \rightarrow HOURS$  nor  $PNUMBER \rightarrow HOURS$  hold
  - $\{SSN, PNUMBER\} \rightarrow ENAME$  is not a full FD (it is called a partial dependency) since  $SSN \rightarrow ENAME$  also holds.

# Second Normal Form

---

- A relation schema  $R$  is in second normal form (2NF) if every non-prime attribute  $A$  in  $R$  is fully functionally dependent on the primary key.
- $R$  can be decomposed into 2NF relations via the process of 2NF normalization.

# Third Normal Form

---

- Definition:
  - Transitive functional dependency: a FD  $X \rightarrow Z$  that can be derived from two FDs  $X \rightarrow Y$  and  $Y \rightarrow Z$
- Examples:
  - $SSN \rightarrow DMGRSSN$  is a transitive FD
    - Since  $SSN \rightarrow DNUMBER$  and  $DNUMBER \rightarrow DMGRSSN$  hold
  - $SSN \rightarrow ENAME$  is non-transitive
    - Since there is no set of attributes  $X$  where  $SSN \rightarrow X$  and  $X \rightarrow ENAME$

# Third Normal Form

---

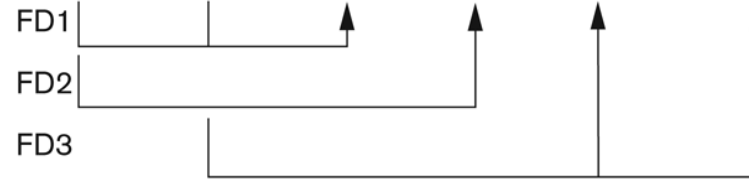
- A relation schema  $R$  is in third normal form (3NF) if it is in 2NF, and no non-prime attribute  $A$  in  $R$  is transitively dependent on the primary key
- $R$  can be decomposed into 3NF relations via the process of 3NF normalization
- NOTE:
  - In  $X \rightarrow Y$  and  $Y \rightarrow Z$ , with  $X$  as the primary key, we consider this a problem only if  $Y$  is not a candidate key.
  - When  $Y$  is a candidate key, there is no problem with the transitive dependency .
  - E.g., Consider EMP (SSN, Emp#, Salary ).
    - Here,  $SSN \rightarrow Emp\# \rightarrow Salary$  and Emp# is a candidate key.

# Third Normal Form

(a)

**EMP\_PROJ**

<u>Ssn</u>	<u>Pnumber</u>	Hours	Ename	Pname	Plocation
------------	----------------	-------	-------	-------	-----------



2NF Normalization

**EP1**

<u>Ssn</u>	<u>Pnumber</u>	Hours
------------	----------------	-------



**EP2**

<u>Ssn</u>	Ename
------------	-------



**EP3**

<u>Pnumber</u>	Pname	Plocation
----------------	-------	-----------



(b)

**EMP\_DEPT**

Ename	<u>Ssn</u>	Bdate	Address	Dnumber	Dname	Dmgr_ssn
-------	------------	-------	---------	---------	-------	----------



3NF Normalization

**ED1**

Ename	<u>Ssn</u>	Bdate	Address	Dnumber
-------	------------	-------	---------	---------



**ED2**

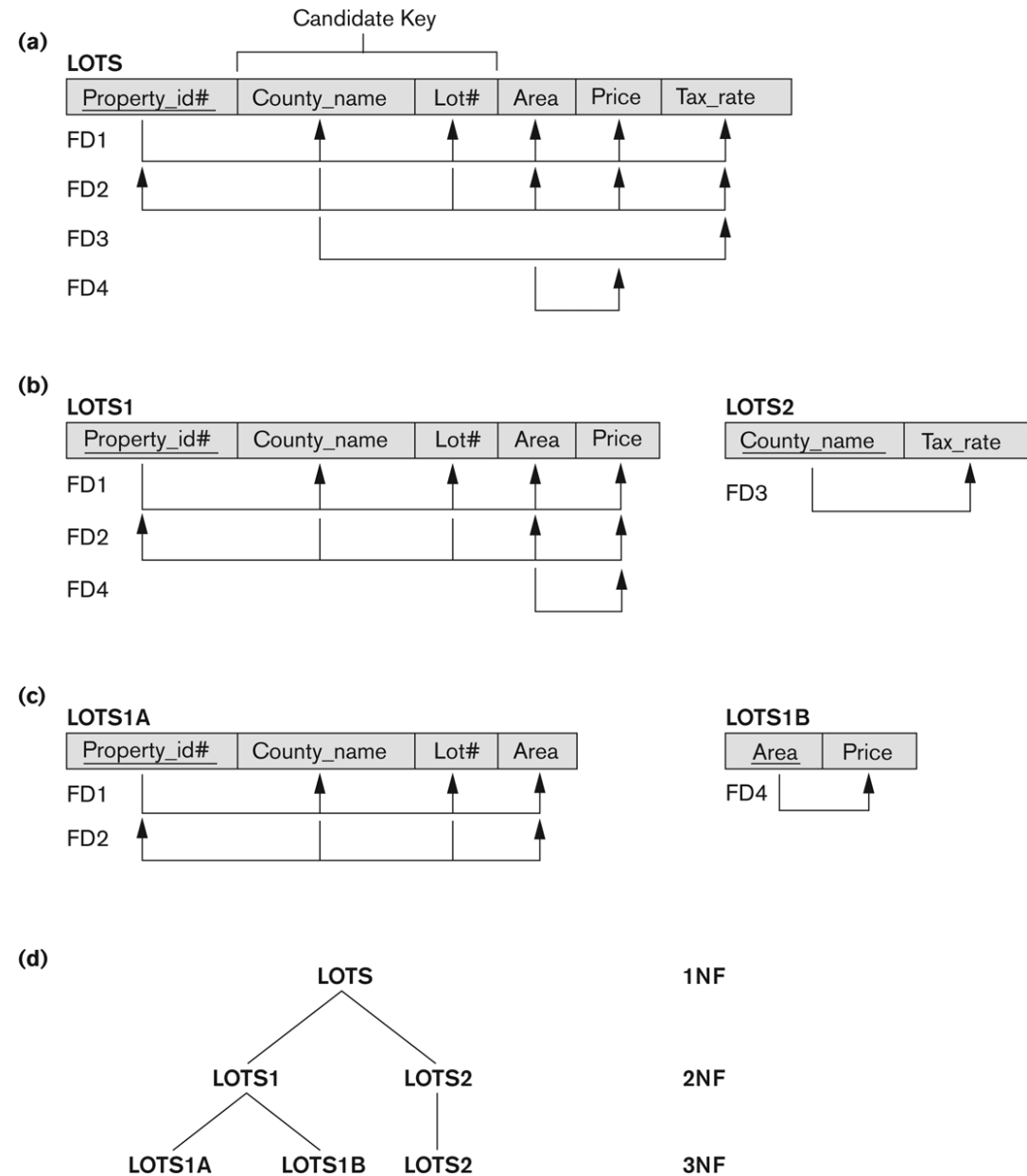
<u>Dnumber</u>	Dname	Dmgr_ssn
----------------	-------	----------



**Figure 10.10**

Normalizing into 2NF and 3NF.  
(a) Normalizing EMP\_PROJ into 2NF relations.  
(b) Normalizing EMP\_DEPT into 3NF relations.

# Third Normal Form



**Figure 10.11**

Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Summary of the progressive normalization of LOTS.

# Informally saying

## 1<sup>st</sup> normal form

- All attributes depend on **the key**

## 2<sup>nd</sup> normal form

- All attributes depend on **the whole key**

## 3<sup>rd</sup> normal form

- All attributes depend on **nothing but the key**

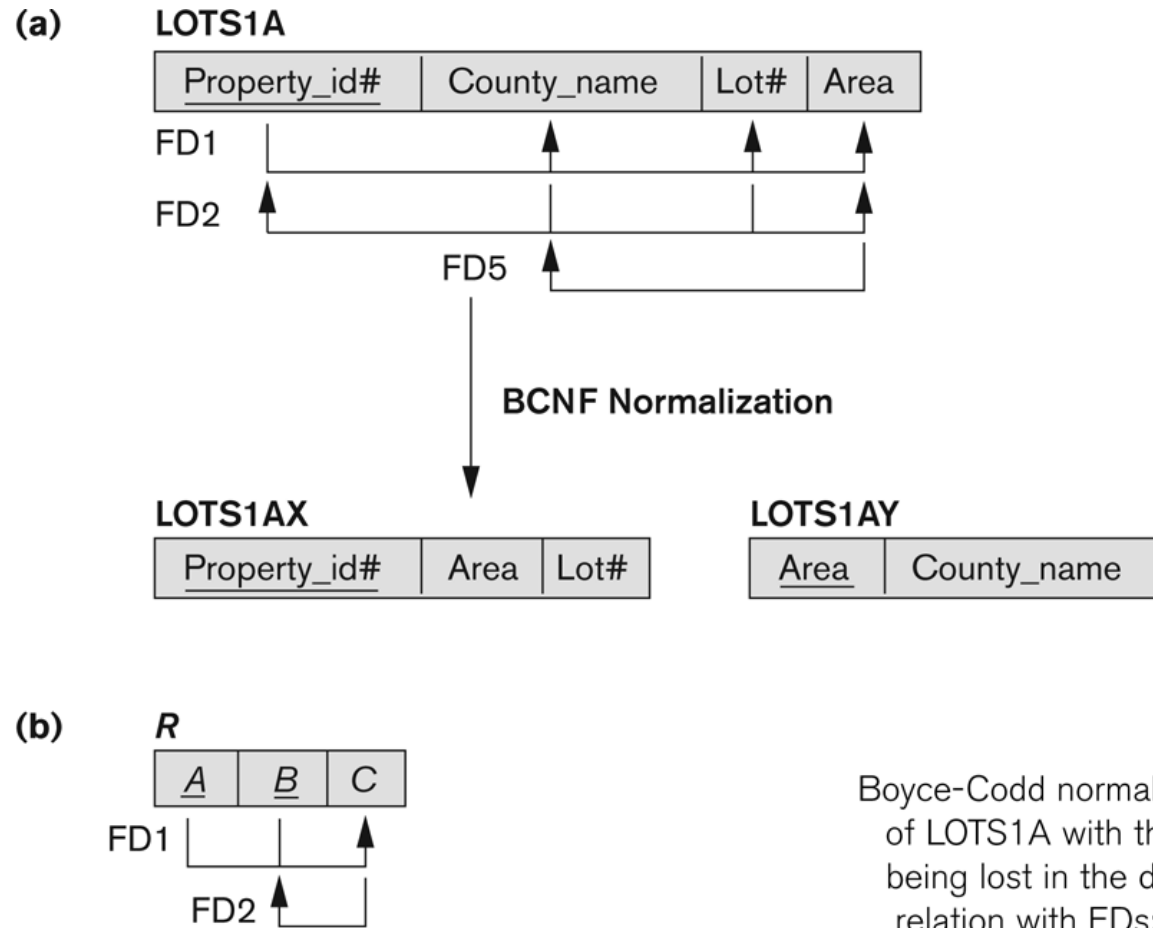
# Boyce-Codd Normal Form

---

- A relation schema  $R$  is in Boyce-Codd Normal Form (BCNF) if whenever an FD  $X \rightarrow A$  holds in  $R$ , then  $X$  is a superkey of  $R$
- Each normal form is strictly stronger than the previous one
  - Every 2NF relation is in 1NF
  - Every 3NF relation is in 2NF
  - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- The goal is to have each relation in BCNF (or 3NF)



# Boyce-Codd Normal Form



**Figure 10.12**  
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

# Boyce- Codd Normal Form

---

**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

**Figure 10.13**

A relation TEACH that  
is in 3NF but not  
BCNF.

# Boyce-Codd Normal Form

---

- Two FDs exist in the relation TEACH:
  - fd1: { student, course} -> instructor
  - fd2: instructor -> course
- {student, course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 10.12 (b).
  - So this relation is in 3NF but not in BCNF
- A relation NOT in BCNF should be decomposed so as to meet this property, while possibly forgoing the preservation of all functional dependencies in the decomposed relations.

# Multivalued Dependency and Fourth Normal Form

---

- The Fourth Normal Form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key.
- It builds on the first three normal forms (1NF, 2NF, and 3NF) and the Boyce-Codd Normal Form (BCNF).
- It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

# Multivalued Dependency and Fourth Normal Form

---

- Multivalued dependency would occur whenever two separate attributes in a given table happen to be independent of each other.
- Both depend on another third attribute.
- The multivalued dependency contains at least two of the attributes dependent on the third attribute.
- This is the reason why it always consists of at least three of the attributes.
- MVD between a and b is denoted by  $a \twoheadrightarrow \twoheadrightarrow b$  or  $a \twoheadrightarrow\!\!\rightarrow b$ .

# Multivalued Dependency and Fourth Normal Form

---

- Condition For MVD
  1.  $R1[a], R2[a], R3[a], R4[a]$  should contain the same value.
    - $R1[a] = R2[a] = R3[a] = R4[a]$
  2. The value of  $R1[b]$  should be equal to the value  $R3[b]$  and the value  $R2[b]$  should be equal to the value of  $R4[b]$ .
    - $R1[b] = R3[b], R2[b] = R4[b]$
- Name --> > Course work and  
Name --> > Hobby

Name	Course_work	Hobby
Rahul	C++	Painting
Rahul	Python	Music
Rahul	C++	Music
Rahul	Python	Painting

# Multivalued Dependency and Fourth Normal Form

Name	Course_work	Hobby
Rahul	C++	Painting
Rahul	Python	Music
Rahul	C++	Music
Rahul	Python	Painting

Name	Course_work
Rahul	C++
Rahul	Python

Name	Hobby
Rahul	Painting
Rahul	Music

# Join Dependencies and Fifth Normal Form

---

- A given relation 'R' is decomposed into 3 relations after 4NF resulting into R1, R2 and R3.
- Now we perform a join between R1, R2 and R3 in any order. If we get back original R without loss of data and without spurious tuples, we say it is in 5NF.
- A relation R is in 5NF if and only if it satisfies the following conditions:
  1. R should be already in 4NF.
  2. It cannot be further non loss decomposed (join dependency).



## Join Dependencies and Fifth Normal Form

---

- Join dependency exists if we are not able to get original table R when decomposed tables are joined.

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

# Join Dependencies and Fifth Normal Form

---

Agent	Company
A1	PQR
A1	XYZ
A2	PQR

Agent	Product
A1	Nut
A1	Bolt
A2	Nut

Company	Product
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

# Join Dependencies and Fifth Normal Form

---

When joined, we get the original table R. Hence it is in 5NF.

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

# Module 3 Chapter 2

---

# SQL Introduction

---

- **SQL language**
  - Considered one of the major reasons for the commercial success of relational databases
- **SQL**
  - The origin of SQL is relational predicate calculus called tuple calculus which was proposed initially as the language SQUARE.
  - SQL Actually comes from the word “SEQUEL” which was the original term used in the paper: “SEQUEL TO SQUARE” by Chamberlin and Boyce. IBM could not copyright that term, so they abbreviated to SQL and copyrighted the term SQL.
  - Now popularly known as “Structured Query language”.
  - SQL is an informal or practical rendering of the relational data model with syntax.

# SQL data definition and data types

---

- **Terminology:**
  - Table, row, and column used for relational model terms relation, tuple, and attribute
- **CREATE statement**
  - Main SQL command for data definition.
- **SQL schema**
  - Identified by a schema name
  - Includes an authorization identifier and descriptors for each element
- **Schema elements include**
  - Tables, constraints, views, domains, and other constructs
  - Each statement in SQL ends with a semicolon

# SQL data definition and data types

---

- **CREATE SCHEMA statement**
  - CREATE SCHEMA COMPANY AUTHORIZATION 'Jsmith';
- **Catalog**
  - Named collection of schemas in an SQL environment
  - SQL also has the concept of a cluster of catalogs.

## **CREATE TABLE Command**

- Specifying a new relation
  - Provide name of table
  - Specify attributes, their types and initial constraints

# SQL data definition and data types

---

- Can optionally specify schema:
  - CREATE TABLE COMPANY.EMPLOYEE ...
  - or
  - CREATE TABLE EMPLOYEE ...
- **Base tables (base relations)**
  - Relation and its tuples are actually created and stored as a file by the DBMS
- **Virtual relations (views)**
  - Created through the CREATE VIEW statement. Do not correspond to any physical file.



# SQL data definition and data types

---

```
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) );
```

# SQL data definition and data types

---

```
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) );
```

# SQL data definition and data types

---

## Attribute Data Types and Domains in SQL

- **Numeric data types**
  - Integer numbers: INTEGER, INT, and SMALLINT
  - Floating-point (real) numbers: FLOAT or REAL, and DOUBLE PRECISION
- **Character-string data types**
  - Fixed length: CHAR(n), CHARACTER(n)
  - Varying length: VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)
- **Bit-string data types**
  - Fixed length: BIT(n)
  - Varying length: BIT VARYING(n)

# SQL data definition and data types

---

- **Boolean data type**
  - Values of TRUE or FALSE or NULL
- **DATE data type**
  - Ten positions
  - Components are YEAR, MONTH, and DAY in the form YYYY-MM-DD
  - Multiple mapping functions available in RDBMSs to change date formats

# SQL data definition and data types

---

## Additional data types

- **Timestamp data type**

- Includes the DATE and TIME fields
- Plus a minimum of six positions for decimal fractions of seconds
- Optional WITH TIME ZONE qualifier

- **INTERVAL data type**

- Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp
- DATE, TIME, Timestamp, INTERVAL data types can be cast or converted to string formats for comparison.

# SQL data definition and data types

---

- **Domain**

- Name used with the attribute specification
- Makes it easier to change the data type for a domain that is used by numerous attributes
- Improves schema readability
- Example: `CREATE DOMAIN SSN_TYPE AS CHAR(9);`

- **TYPE**

- User Defined Types (UDTs) are supported for object-oriented applications. Uses the command `CREATE TYPE`

# Schema change statements in SQL

## The DROP Command

- The DROP command can be used to drop named schema elements, such as tables, domains, or constraints.
- One can also drop a schema. For example, if a whole schema is no longer needed, the DROP SCHEMA command can be used.
- There are two drop behavior options: CASCADE and RESTRICT.
- For example, to remove the COMPANY database schema and all its tables, domains, and other elements, the CASCADE option is used as follows:

# Schema change statements in SQL

- DROP SCHEMA COMPANY CASCADE;
- If the RESTRICT option is chosen in place of CASCADE, the schema is dropped only if it has no elements in it; otherwise, the DROP command will not be executed.
- To use the RESTRICT option, the user must first individually drop each element in the schema, then drop the schema itself.
- The DROP TABLE command not only deletes all the records in the table if successful, but also removes the table definition from the catalog.



# Schema change statements in SQL

## The ALTER Command

- The definition of a base table or of other named schema elements can be changed by using the ALTER command.
- For base tables, the possible alter table actions include adding or dropping a column (attribute), changing a column definition, and adding or dropping table constraints.
- ALTER TABLE COMPANY.EMPLOYEE ADD COLUMN Job VARCHAR(12);

# Schema change statements in SQL

- To drop a column, we must choose either CASCADE or RESTRICT for drop behavior.
- If CASCADE is chosen, all constraints and views that reference the column are dropped automatically from the schema, along with the column.
- If RESTRICT is chosen, the command is successful only if no views or constraints (or other schema elements) reference the column.
- ALTER TABLE COMPANY.EMPLOYEE DROP COLUMN Address CASCADE;

# Specifying constraints in SQL

## Basic constraints:

- Relational Model has 3 basic constraint types that are supported in SQL:
  - Key constraint: A primary key value cannot be duplicated
  - Entity Integrity Constraint: A primary key value cannot be null
  - Referential integrity constraints : The “foreign key “ must have a value that is already present as a primary key or may be null.

# Specifying constraints in SQL

## Attribute Constraints

- **Default value of an attribute**
  - DEFAULT <value>
  - NULL is not permitted for a particular attribute (NOT NULL)
- **CHECK clause**
  - Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

# Specifying constraints in SQL

## Specifying Key and Referential Integrity Constraints

- **PRIMARY KEY clause**

- Specifies one or more attributes that make up the primary key of a relation
- Dnumber INT PRIMARY KEY;

- **UNIQUE clause**

- Specifies alternate (secondary) keys (called CANDIDATE keys in the relational model).
- Dname VARCHAR(15) UNIQUE;

# Specifying constraints in SQL

- **FOREIGN KEY clause**

- Default operation: reject update on violation
- Attach referential triggered action clause
- Options include SET NULL, CASCADE, and SET DEFAULT
- Action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE CASCADE option suitable for “relationship” relations

## **Giving Names to Constraints**

- Using the Keyword CONSTRAINT
  - Name a constraint
  - Useful for later altering

```

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 constraints in SQL



# Specifying constraints in SQL

- Additional Constraints on individual tuples within a relation are also possible using CHECK
- CHECK clauses at the end of a CREATE TABLE statement
  - Apply to each tuple individually
  - `CHECK (Dept_create_date <= Mgr_start_date);`



# Retrieval queries in SQL

---

- SELECT statement
  - One basic statement for retrieving information from a database
- SQL allows a table to have two or more tuples that are identical in all their attribute values
  - Unlike relational model (relational model is strictly set-theory based)
  - Multiset or bag behavior
  - Tuple-id may be used as a key

# Retrieval queries in SQL

Basic form of the SELECT statement:

```
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.

# Retrieval queries in SQL

---

- Logical comparison operators
  - =, <, <=, >, >=, and <>
- Projection attributes
  - Attributes whose values are to be retrieved
- Selection condition
  - Boolean condition that must be true for any retrieved tuple. Selection conditions include join conditions when multiple relations are involved.

# Retrieval queries in SQL

---

**Query 0.** Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:  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.

```
Q1:  SELECT  Fname, Lname, Address
      FROM    EMPLOYEE, DEPARTMENT
      WHERE   Dname='Research' AND Dnumber=Dno;
```

# Retrieval queries in SQL

---

## Ambiguous Attribute Names

- Same name can be used for two (or more) attributes in different relations
  - As long as the attributes are in different relations
  - Must qualify the attribute name with the relation name to prevent ambiguity

```
Q1A:  SELECT  Fname, EMPLOYEE.Name, Address
        FROM    EMPLOYEE, DEPARTMENT
        WHERE   DEPARTMENT.Name='Research' AND
                DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

# Retrieval queries in SQL

---

## Aliasing and Renaming

- Aliases or tuple variables
  - Declare alternative relation names E and S to refer to the EMPLOYEE relation twice in a 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;
```
  - Recommended practice to abbreviate names and to prefix same or similar attribute from multiple tables.

# Retrieval queries in SQL

---

- The attribute names can also be renamed
  - `EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)`
- Note that the relation `EMPLOYEE` now has a variable name `E` which corresponds to a tuple variable
- The “AS” may be dropped in most SQL implementations

# Retrieval queries in SQL

---

## Unspecified WHERE Clause and Use of the Asterisk

- Missing WHERE clause
  - Indicates no condition on tuple selection
- Effect is a CROSS PRODUCT
  - Result is all possible tuple combinations (or the Algebra operation of Cartesian Product)

Queries 9 and 10. Select all EMPLOYEE Ssns (Q9) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q10) in the database.

Q9:     SELECT     Ssn  
          FROM     EMPLOYEE;

Q10:    SELECT     Ssn, Dname  
          FROM     EMPLOYEE, DEPARTMENT;



# Retrieval queries in SQL

---

- Specify an asterisk (\*)
  - Retrieve all the attribute values of the selected tuples
  - The \* can be prefixed by the relation name

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

Q1D:    SELECT    \*  
         FROM     EMPLOYEE, DEPARTMENT  
         WHERE    Dname='Research' AND Dno=Dnumber;

Q10A:   SELECT    \*  
         FROM     EMPLOYEE, DEPARTMENT;

# Retrieval queries in SQL

---

## Tables as Sets in SQL

- SQL does not automatically eliminate duplicate tuples in query results
- For aggregate operations duplicates must be accounted for
- Use the keyword DISTINCT in the SELECT clause
  - Only distinct tuples should remain in the result

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

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

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

# Retrieval queries in SQL

---

## Set operations

- UNION, EXCEPT (difference), INTERSECT
- Corresponding multiset operations: UNION ALL, EXCEPT ALL, INTERSECT ALL)
- Type compatibility is needed for these operations to be valid

**Query 4.** Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A: ( 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' );
```

# Retrieval queries in SQL

---

## Substring Pattern Matching and Arithmetic Operators

- LIKE comparison operator
  - Used for string pattern matching
  - % replaces an arbitrary number of zero or more characters
  - underscore (\_) replaces a single character
  - Examples: WHERE Address LIKE '%Houston,TX%';
  - WHERE Ssn LIKE '\_\_ 1\_\_ 8901';
- BETWEEN comparison operator
  - WHERE(Salary BETWEEN 30000 AND 40000) AND Dno = 5;

# Retrieval queries in SQL

---

- Standard arithmetic operators:
  - Addition (+), subtraction (−), multiplication (\*), and division (/) may be included as a part of SELECT
- Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

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

# Retrieval queries in SQL

---

## Ordering of Query Results

- Use ORDER BY clause
  - Keyword DESC to see result in a descending order of values
  - Keyword ASC to specify ascending order explicitly
  - Typically placed at the end of the query
  - ORDER BY D.Dname DESC, E.Lname ASC, E.Fname ASC

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

# Retrieval queries in SQL

## **Complete syntax of SELECT**

SELECT attribute list

FROM table list

WHERE conditions

GROUP BY grouping attribute

HAVING condition

ORDER BY attribute

# INSERT, DELETE, and UPDATE statements in SQL

---

- Three commands used to modify the database:
  - INSERT, DELETE, and UPDATE
- INSERT typically inserts a tuple (row) in a relation (table)
- UPDATE may update a number of tuples (rows) in a relation (table) that satisfy the condition
- DELETE may also update a number of tuples (rows) in a relation (table) that satisfy the condition



# INSERT, DELETE, and UPDATE statements in SQL

---

## INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command
- Constraints on data types are observed automatically
- Any integrity constraints as a part of the DDL specification are enforced

# INSERT, DELETE, and UPDATE statements in SQL

---

- Specify the relation name and a list of values for the tuple. All values including nulls are supplied.
- The variation below inserts multiple tuples where a new table is loaded values from the result of a query.

```
U1:  INSERT INTO  EMPLOYEE  
      VALUES      ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98  
                    Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

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

# INSERT, DELETE, and UPDATE statements in SQL

---

## BULK LOADING OF TABLES

- Another variation of INSERT is used for bulk-loading of several tuples into tables
- A new table TNEW can be created with the same attributes as T and using LIKE and DATA in the syntax, it can be loaded with entire data.

- EXAMPLE:

```
CREATE TABLE D5EMPS LIKE EMPLOYEE
(SELECT E.*
FROM EMPLOYEE AS E
WHERE E.Dno = 5) WITH DATA;
```

# INSERT, DELETE, and UPDATE statements in SQL

---

## DELETE

- Removes tuples from a relation
  - Includes a WHERE-clause to select the tuples to be deleted
  - Referential integrity should be enforced
  - Tuples are deleted from only one table at a time (unless CASCADE is specified on a referential integrity constraint)
  - A missing WHERE-clause specifies that all tuples in the relation are to be deleted; the table then becomes an empty table
  - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

# INSERT, DELETE, and UPDATE statements in SQL

---

- Includes a WHERE clause to select the tuples to be deleted. The number of tuples deleted will vary.

U4A:	DELETE FROM WHERE	EMPLOYEE Lname='Brown';
U4B:	DELETE FROM WHERE	EMPLOYEE Ssn='123456789';
U4C:	DELETE FROM WHERE	EMPLOYEE Dno=5;
U4D:	DELETE FROM	EMPLOYEE;

# INSERT, DELETE, and UPDATE statements in SQL

---

## **UPDATE**

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples in the same relation
- Referential integrity specified as part of DDL specification is enforced

# INSERT, DELETE, and UPDATE statements in SQL

---

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively

```
UPDATE PROJECT
```

```
SET Plocation = 'Bellaire', Dnum = 5
```

```
WHERE Pnumber = 10;
```

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

```
UPDATE EMPLOYEE
```

```
SET Salary = Salary * 1.1
```

```
WHERE Dno = 5;
```

# Additional features of SQL

---

- Techniques for specifying complex retrieval queries
- Writing programs in various programming languages that include SQL statements: Embedded and dynamic SQL, SQL/CLI (Call Level Interface) and its predecessor ODBC, SQL/PSM (Persistent Stored Module)
- Set of commands for specifying physical database design parameters, file structures for relations, and access paths, e.g., CREATE INDEX



# Additional features of SQL

---

- Transaction control commands
- Specifying the granting and revoking of privileges to users
- Constructs for creating triggers
- Enhanced relational systems known as object- relational define relations as classes. Abstract data types (called User Defined Types- UDTs) are supported with CREATE TYPE
- New technologies such as XML and OLAP are added to versions of SQL

# End of Module 3

