Lecture Notes
DATABASE MANAGEMENT SYSTEM (BCS403)

Module – 3

Normalization: Database Design Theory – Introduction to Normalization using Functional and Multivalued Dependencies: Informal design guidelines for relation schema, Functional Dependencies, Normal Forms basedon Primary Keys, Second and Third Normal Forms, Boyce-Codd Normal Form, Multivalued Dependency and Fourth Normal Form, Join Dependencies and Fifth Normal Form.

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.

Textbook 1: Ch 14.1 to 14.7, Ch 6.1 to 6.5 **RBT: L1, L2, L3**

Teaching-Learning Process	Chalk and board, Problem based learning, Demonstration

CHAPTER 1:

Introduction to Normalization using Functional and Multivalued Dependencies

1. Informal design guidelines for relation schema

The four informal guidelines that may be used as measures to determine the quality of relation schema design:

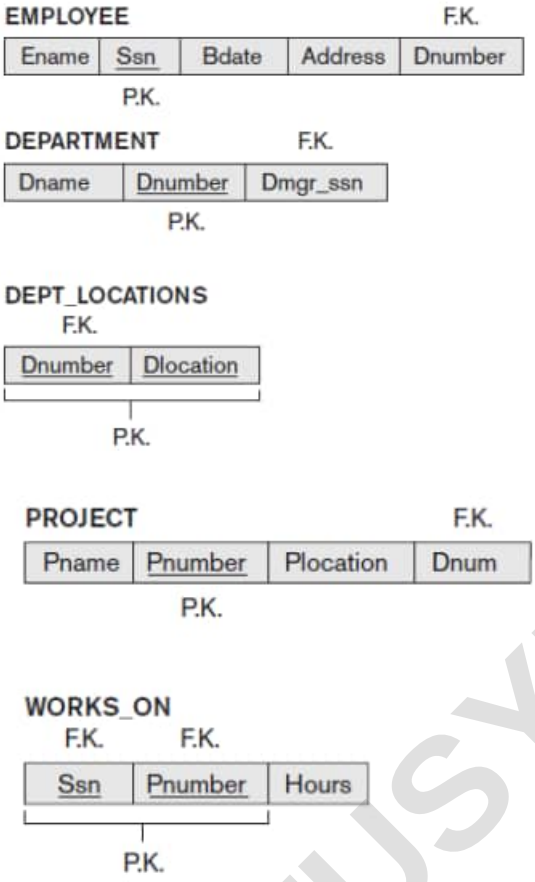
- Making sure that the semantics of the attributes is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples

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 and a proper interpretation associated with them.

The **semantics** of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.

A simplified version of the COMPANY relational database schema considered is:



EMPLOYEE

Ename	Ssn	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	291 Berry, 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	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

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

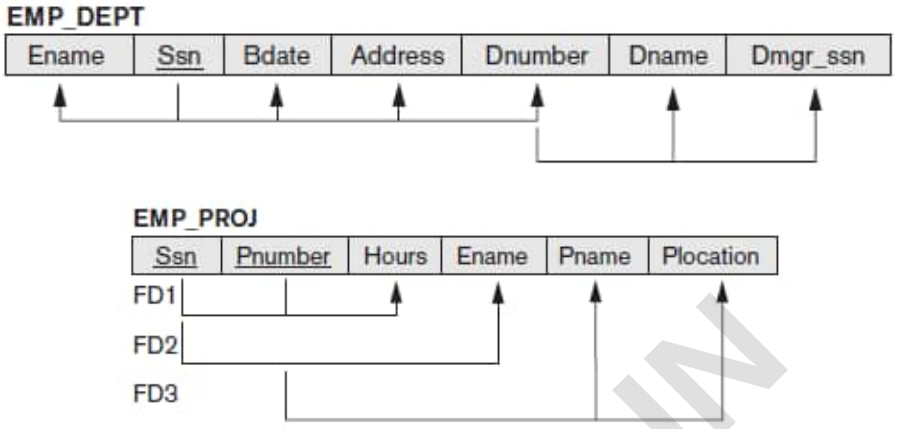
Ssn	Pnumber	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

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. Intuitively, 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.

Examples of Violating Guideline 1



Two relation schemas suffering from update anomalies.

(a) EMP_DEPT and

(b) EMP_PROJ

Redundant Information in Tuples and Update Anomalies

One goal of schema design is to minimize the storage space used by the base relations (and hence the corresponding files). Grouping attributes into relation schemas has a significant effect on storage space.

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

- 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 (if the employee does not work for a department as yet). 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.

- 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. 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 inadvertently from the database.

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. We can also see the need for a more formal approach to evaluating whether a design meets these guidelines.

NULL Values in Tuples

In some schema designs we may group many attributes together into a "fat" relation. 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 and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level.

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.

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).

Generation of Spurious Tuples

Result of applying NATURAL JOIN to the tuples in EMP_PROJ1 and EMP_LOCS

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.

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.

2. Functional Dependencies

A functional dependency is a constraint between two sets of attributes from the database.

Suppose that our relational database schema has n attributes $A1, A2, \ldots, An$; let us think of the whole database as being described by a single universal relation schema R = {A1, A2, … , An}.

We do not imply that we will actually store the database as a single universal table; we use this concept only in developing the formal theory of data dependencies.

Definition: A functional dependency, denoted by X → 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].

X functionally determines Y in a relation schema R if, and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value. Note the following:

- If a constraint on R states that there cannot be more than one tuple with a given X-value in any relation instance r(R)—that is, X is a candidate key of R—this implies that X → Y for any subset of attributes Y of R (because the key constraint implies that no two tuples in any legal state r(R) will have the same value of X). If X is a candidate key of R, then X → R.
- If X → Y in R, this does not say whether or not Y → X in R.
- Consider the relation schema EMP_PROJ.

The following functional dependencies should hold:

a. Ssn → Ename

b. Pnumber → {Pname, Plocation}

c. {Ssn, Pnumber} → Hours

3.Normal Forms Based on Primary Keys

Normalization of Relations

The normalization process, as first proposed by Codd (1972), 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.*

Initially, Codd proposed three normal forms, which he called first, second, and third normal form.

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, 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

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.

Practical Use of Normal Forms

Most practical design projects in commercial and governmental environment acquire existing designs of databases from previous designs, from designs in legacy models, or from existing files.

They are certainly interested in assuring that the designs are good quality and sustainable over long periods of time.

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.

Definitions of Keys and Attributes Participating in Keys

Definition. A superkey of a relation schema R = {A1, A2, … , An} is a set of attributes S ⊆ R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[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.

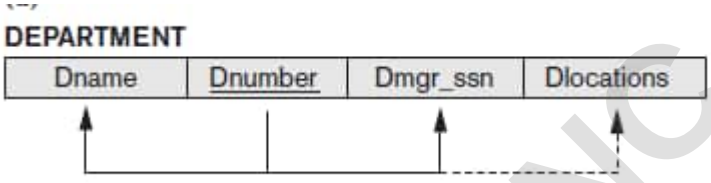
Definition. An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R. An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key.

First Normal Form (1NF)

First normal form (1NF) is now considered to be part of the formal definition of a relation in the basic (flat) relational model.

It was 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.



DEPARTMENT

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

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

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

- Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT.
- Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT.
- 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.

(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, 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

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

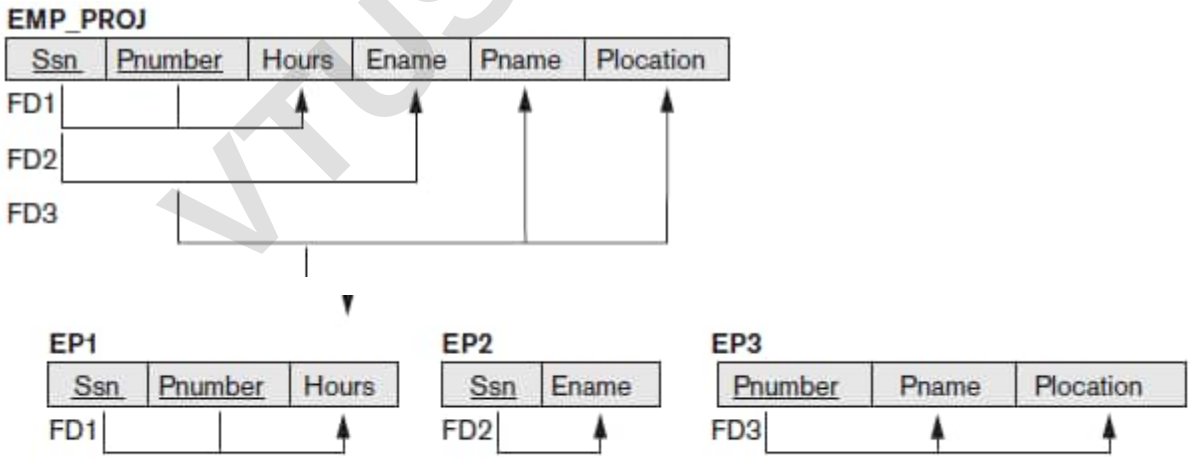
Second Normal Form (2NF)

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 anymore; 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.

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*.

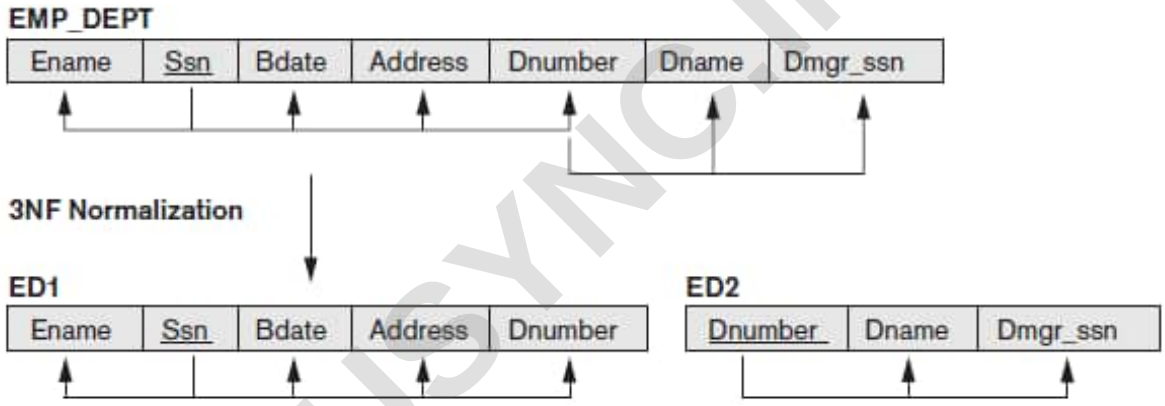


Third Normal Form (3NF)

Third normal form (3NF) is based on the concept of transitive dependency.

A functional dependency X → Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X → Z and Z → Y hold.

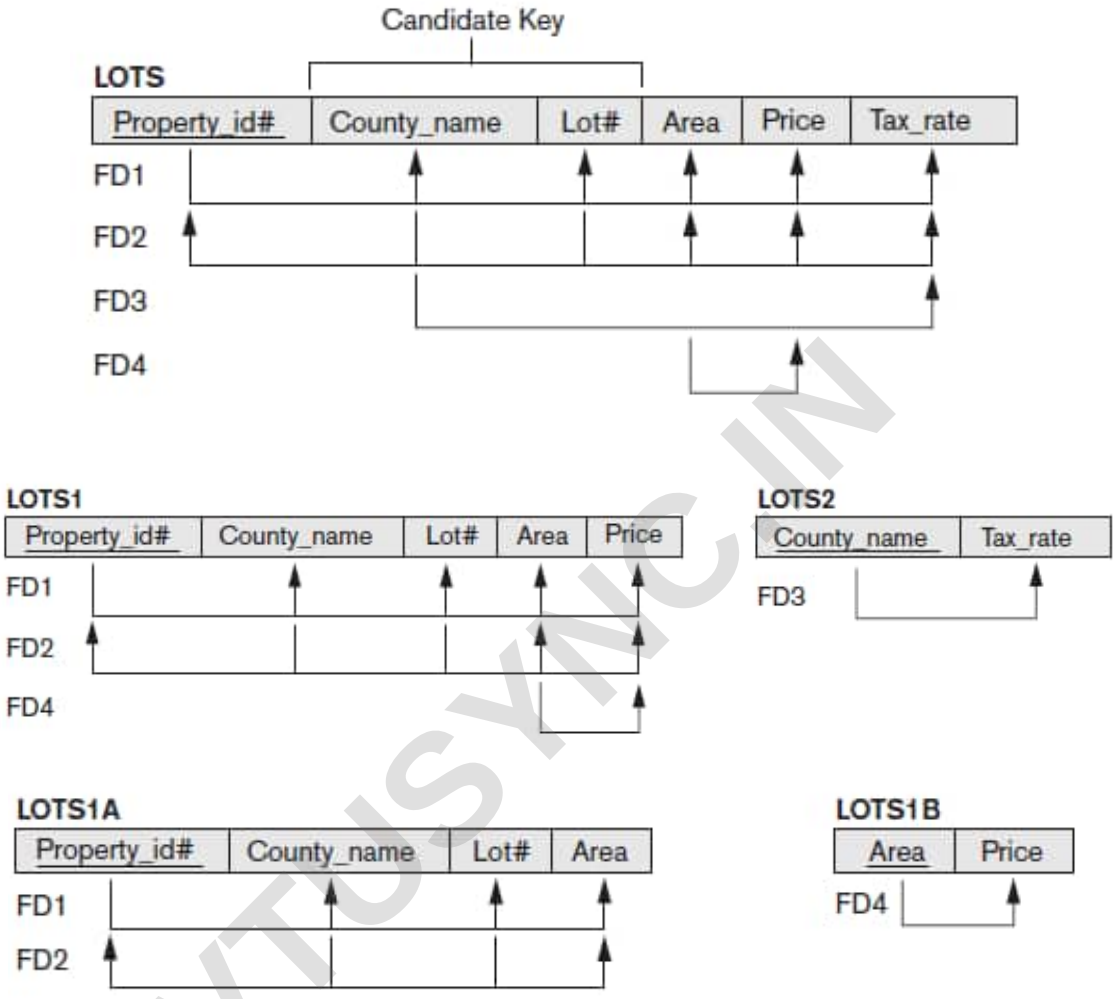
Definition. According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.



4. General Definitions of Second and Third Normal Forms

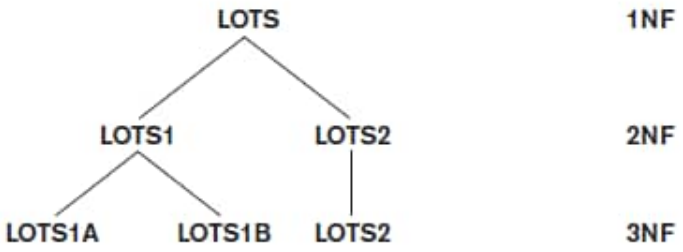
Definition. A relation schema *R* is in **second normal form (2NF)** if every nonprime attribute *A* in *R* is not partially dependent on *any* key of *R*.

Definition. 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.



Alternative Definition. A relation schema *R* is in 3NF if every nonprime attribute of *R* meets both of the following conditions:

- It is fully functionally dependent on every key of *R*.
- It is nontransitively dependent on every key of *R*



5. 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.

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

Most relation schemas that are in 3NF are also in BCNF

The goal is to have each relation in BCNF (or 3NF)

Definition. A relation schema R is in BCNF if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, then X is a superkey of R.

The formal definition of BCNF differs from the definition of 3NF in that clause (b) of 3NF, which allows f.d.'s having the RHS as a prime attribute, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF.

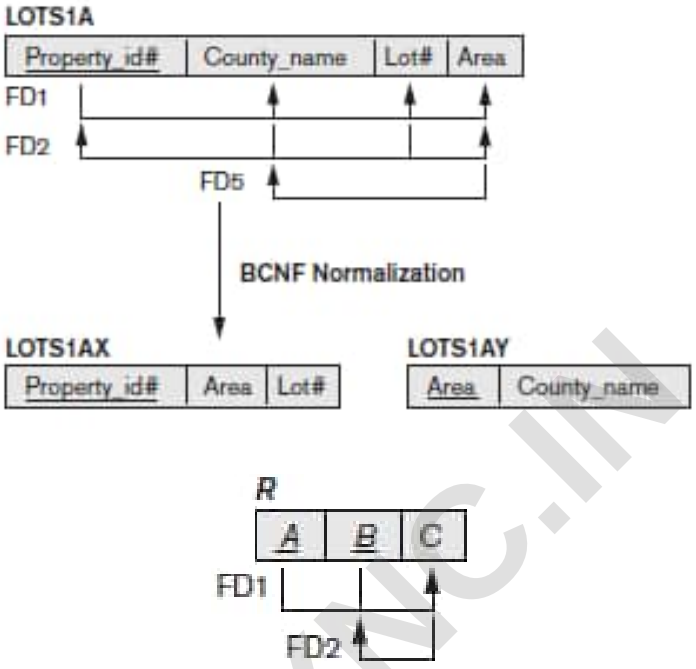


Figure: A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. C → B.

Suppose that we have thousands of lots in the relation and lots are from only two counties: DeKalb and Fulton. Suppose also that lot sizes in DeKalb County are only 0.5 to 1.0 acres, whereas lot sizes in Fulton County are from 1.1 to 2.0 acres. This adds a functional dependency FD5: ***Area->County_name***.

LOTS1A still is in 3NF because ***County_name*** is a prime attribute, but not in BCNF as ***Area*** is not a superkey of LOTS1A

6. Decomposition of Relations not in BCNF

Another example shows a relation TEACH with the

following dependencies:

FD1: {Student, Course} → Instructor

FD2: Instructor → Course

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)

All three decompositions *lose* the functional dependency FD1. The question then becomes: Which of the above three is a *desirable decomposition*?

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

we strive to meet two properties of decomposition during the normalization process: the nonadditive join property and the functional dependency preservation property. We are not able to meet the functional dependency preservation for any of the above BCNF decompositions as seen above; but we must meet the nonadditive join property. A simple test comes in handy to test the binary decomposition of a relation into two relations:

NJB (Nonadditive Join Test for Binary Decompositions). A decomposition D = {R1, R2} of R has the lossless (nonadditive) join property with respect to a set of functional dependencies F on R if and only if either

- The FD $((R1 \cap R2) \rightarrow (R1 - R2))$ is in F^+, or
- The FD $((R1 \cap R2) \rightarrow (R2 - R1))$ is in F^+

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 property by the following procedure.It decomposes *R* successively into a 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 a 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.

6. Multivalued Dependency and Fourth Normal Form

Consider the relation EMP. 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, and the employee's projects and dependents are independent of one another.

As illustrated by the EMP relation, some relations have constraints that cannot be specified as functional dependencies and hence are not in violation of BCNF. To address this situation, the concept of *multivalued dependency* (MVD) was proposed and, based on this dependency, the *fourth normal form* was defined.

EMP

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

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

The EMP relation with two MVDs:

Ename →→ Pname and

Ename →→ Dname.

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]

Whenever X →→ Y holds, we say that X multidetermines Y. Because of the symmetry in the definition, whenever X →→ Y holds in R, so does X →→ Z. Hence, X →→ Y implies X →→ Z and therefore it is sometimes written as X →→ 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.

For example, the relation EMP_PROJECTS in Figure has the trivial

MVD Ename →→ Pname and the relation EMP_DEPENDENTS has the trivial MVD Ename →→ Dname.

An MVD that satisfies neither (a) nor (b) is called a nontrivial MVD.

A trivial MVD will hold in any relation state r of R; it is called trivial because it does not specify any significant or meaningful constraint on R.

7. Fourth normal form (4NF)

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 →→ Y in F^+, X is a superkey for R.

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 relation in Figure, 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.

8. Join Dependencies and Fifth Normal Form

Definition. 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

$$* (\pi R1(r), \pi R2(r), \dots, \pi Rn(r)) = r$$

Notice that an MVD is a special case of a JD where n = 2.

That is, a JD denoted as JD(R1, R2) implies an MVD (R1 ∩ R2) →→ (R1 − R2)(or, by symmetry, (R1 ∩ R2) →→ (R2 − R1)).

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.

Such a dependency is called trivial because it has the nonadditive join property for any relation state r of R and thus does not specify any constraint on R.

Definition. 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(R1, R2, … , Rn) in F+ (that is, implied by F), every Ri is a superkey of R.

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

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

Suppose that the following additional constraint always holds:

Whenever a supplier *s* supplies part *p*, *and* a project *j* uses part *p*, *and* the supplier *s* supplies *at least one* part to project *j*, *then* supplier *s* will also be supplying part *p* to project *j*.

This constraint can be restated in other ways and specifies a join dependency JD(*R*1, *R*2, *R*3) among the three projections *R*1 (Sname, Part_name), *R*2 (Sname, Proj_name), and *R*3 (Part_name, Proj_name) of SUPPLY.

If this constraint holds, the tuples below the dashed line in Figure must exist in any legal state of the SUPPLY relation that also contains the tuples above the dashed line.

CHAPTER 2

SQL: SQL data definition and data types

The SQL language is one popular language to work on relational databases. Because it became a standard for relational databases, users were less concerned about migrating their database applications from other types of database systems—for example, older network or hierarchical systems—to relational systems.

SQL is presently expanded as Structured Query Language.

Originally, SQL was called SEQUEL (Structured English QUEry Language) and was designed and implemented at IBM Research as the interface for an experimental relational database system called SYSTEM R.

SQL 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), types, and domains, as well as other constructs such as views, assertions, and triggers.

Schema and Catalog Concepts in SQL

The concept of an SQL schema was incorporated starting with SQL2 in order to group together tables and other constructs that belong to the same database application (in some systems, a *schema* is called a *database*)

An **SQL schema** is identified by a **schema name** and includes an **authorization identifier** to indicate the user or account who owns the schema, as well as **descriptors** for *each element* in the schema.

Schema **elements** include tables, types, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema.

A schema is created via the CREATE SCHEMA statement.

A schema called COMPANY owned by the user with authorization identifier 'Jsmith'.

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

SQL uses the concept of a **catalog**—a named collection of schemas.

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.

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.

CREATE TABLE COMPANY.EMPLOYEE

Rather than

CREATE TABLE EMPLOYEE

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

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

CREATE TABLE DEPT_LOCATIONS
        ( Dnumber        INT                NOT NULL,
          Dlocation      VARCHAR(15)        NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE PROJECT
        ( Pname          VARCHAR(15)        NOT NULL,
          Pnumber        INT                NOT NULL,
          Plocation      VARCHAR(15),
          Dnum           INT                NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE 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) );
```

```
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) );
```

Attribute Data Types and Domains in SQL

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

Numeric data types include

- Integer numbers of various sizes- INTEGER or INT, and SMALLINT
- floating-point (real) numbers - FLOAT or REAL, and DOUBLE PRECISION
- Formatted numbers- DECIMAL (*i*, *j*)—or DEC (*i*, *j*) or NUMERIC (*i*, *j*)—where *i*, the *precision*, is the total number of decimal digits and *j*, the *scale*, is the number of digits after the decimal point.

Character-string data types

- 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.

Bit-string data types

- Fixed length *n*—BIT(*n*)
- Varying length— BIT VARYING(*n*), where *n* is the maximum number of bits. Another variable-length bitstring data type called BINARY LARGE OBJECT or BLOB is also available to specify columns that have large binary values, such as images.

A **Boolean** data type has the traditional values of TRUE or FALSE.

In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN.

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

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

Some additional data types

A **timestamp** data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.

TIMESTAMP '2014-09-27 09:12:47.648302'.

INTERVAL data type. This specifies an **interval**—a *relative value* that can be used to increment or decrement an absolute value of a date, time, or timestamp.

Intervals are qualified to be either YEAR/MONTH intervals or DAY/TIME intervals.

we can create a domain SSN_TYPE by the following statement:

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

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.

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

If no default clause is specified, the default *default value* is NULL for attributes *that do not have* the NOT.

NULL constraint.

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

For example, suppose that department numbers are restricted to integer numbers between 1 and 20.

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

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

CREATE DOMAIN D_NUM **AS** INTEGER

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

Keys and referential integrity constraints are very important, there are special clauses within the CREATE TABLE statement to specify them.

CREATE TABLE EMPLOYEE

(… ,

Dno INT **NOT NULL DEFAULT** 1,

CONSTRAINT EMPPK

PRIMARY KEY (Ssn),

CONSTRAINT EMPSUPERFK

FOREIGN KEY (Super_ssn) **REFERENCES** EMPLOYEE(Ssn)

ON DELETE SET NULL **ON UPDATE** CASCADE,

CONSTRAINT EMPDEPTFK

FOREIGN KEY(Dno) **REFERENCES** DEPARTMENT(Dnumber)

ON DELETE SET DEFAULT **ON UPDATE** CASCADE);

CREATE TABLE DEPARTMENT
(… ,
Mgr_ssn CHAR(9) **NOT NULL DEFAULT** '888665555',
… ,
CONSTRAINT DEPTPK
PRIMARY KEY(Dnumber),
CONSTRAINT DEPTSK
UNIQUE (Dname),
CONSTRAINT DEPTMGRFK
FOREIGN KEY (Mgr_ssn) **REFERENCES** EMPLOYEE(Ssn)
ON DELETE SET DEFAULT **ON UPDATE** CASCADE);

CREATE TABLE DEPT_LOCATIONS
(… ,
PRIMARY KEY (Dnumber, Dlocation),
FOREIGN KEY (Dnumber) **REFERENCES** DEPARTMENT(Dnumber)
ON DELETE CASCADE **ON UPDATE** CASCADE);

Specifying Key and Referential Integrity Constraints

The **PRIMARY KEY** clause specifies one or more attributes that make up the primary key of a relation.

If a primary key has a *single* attribute, the clause can follow the attribute directly. For example, the primary key of DEPARTMENT can be specified as follows

Dnumber INT **PRIMARY KEY**,

If a primary key has more than one attribute then it is declared as

PRIMARY KEY (Dnumber, Dlocation),

The **UNIQUE** clause specifies alternate (unique) keys, also known as candidate keys as shown in the DEPARTMENT and PROJECT table declarations.

The **UNIQUE** clause can also be specified directly for a unique key if it is a single attribute.

Dname VARCHAR(15) **UNIQUE**,

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

A referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is updated.

```
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);
```

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 options include SET NULL, CASCADE, and SET DEFAULT.

An option must be qualified with either ON DELETE or ON UPDATE

Giving Names to Constraints

A constraint may be given a **constraint name**, following the keyword **CONSTRAINT.**

The names of all constraints within a particular schema must be unique.

A constraint name is used to identify a particular constraint in case the constraint

must be dropped later and replaced with another constraint.

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 **row-based** constraints because they apply to each row *individually* and are checked whenever a row is inserted or modified.

CHECK (Dept_create_date <= Mgr_start_date);

Basic Retrieval Queries in SQL

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

The SELECT-FROM-WHERE Structure of Basic SQL Queries

The SELECT statement *is not the same as* the SELECT operation of relational algebra.

There are many options and flavors to the SELECT statement in SQL.

The SELECT-FROM-WHERE Structure of Basic SQL Queries

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.

Query 0. 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 basic logical comparison operators - are =, <, <=, >, >=, and <>.

The SELECT clause of SQL specifies the attributes whose values are to be retrieved, which are called the **projection attributes** in relational algebra.

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

SELECT Fname, Lname, Address **FROM** EMPLOYEE, DEPARTMENT

WHERE Dname = 'Research' **AND** Dnumber = Dno;

A query that involves only selection and join conditions plus projection attributes is known as a **select-project-join** query.

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

SELECT Pnumber, Dnum, Lname, Address, Bdate

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE Dnum = Dnumber **AND** Mgr_ssn = Ssn **AND**

Plocation = 'Stafford'

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

The same name can be used for two (or more) attributes as long as the attributes are in *different tables.*

If this is the case, and a multitable query refers to two or more attributes with the same name, we *must* **qualify** the attribute name with the relation name to prevent ambiguity.

This is done by *prefixing* the relation name to the attribute name and separating the two by a period.

SELECT Fname, EMPLOYEE.Name, Address **FROM** EMPLOYEE, DEPARTMENT **WHERE**

DEPARTMENT.Name = 'Research' **AND**

DEPARTMENT.Dnumber = EMPLOYEE.Dnumber;

SELECT EMPLOYEE.Fname, EMPLOYEE.LName, EMPLOYEE.Address

FROM EMPLOYEE, DEPARTMENT

WHERE DEPARTMENT.DName = 'Research' **AND**

DEPARTMENT.Dnumber = EMPLOYEE.Dno;

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;

E and S is called **aliases** or **tuple variables**, for the EMPLOYEE relation.

It is also possible to **rename** the relation attributes within the query in SQL by giving them aliases.

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

We can use this alias-naming or **renaming** mechanism in any SQL query to specify tuple variables for every table in the WHERE clause, whether or not the same relation needs to be referenced more than once.

SELECT E.Fname, E.LName, E.Address

FROM EMPLOYEE **AS** E, DEPARTMENT **AS** D

WHERE D.DName = 'Research' **AND** D.Dnumber = E.Dno;

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.

Queries 1 and 2. Select all EMPLOYEE Ssns (Q1) and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname (Q2) in the database.

Q1: SELECT Ssn **FROM** EMPLOYEE;

Q2: 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.* The * can also be prefixed by the relation name or alias;

Retrieve all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5

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

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

SELECT * **FROM** EMPLOYEE, DEPARTMENT

WHERE Dname = 'Research' **AND** Dno = Dnumber;

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.

The reasons why SQL does not automatically eliminate duplicate tuples in the results of queries are:

- 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.

An SQL table with a key is restricted to being a set, since the key value must be distinct in each tuple.

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

Query: Retrieve the salary of every employee and all distinct salary Values.

SELECT ALL Salary **FROM** EMPLOYEE;

SELECT DISTINCT Salary **FROM** EMPLOYEE;

Query: 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');

The set union (**UNION**), set difference (**EXCEPT**), and set intersection (**INTERSECT**) operations are directly included in SQL.

Substring Pattern Matching and Arithmetic Operators

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

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

SELECT Fname, Lname **FROM** EMPLOYEE

WHERE Address **LIKE** '%Houston,TX%';

Query: Find all employees who were born during the 1950s.

SELECT Fname, Lname **FROM** EMPLOYEE

WHERE Bdate **LIKE** '_ _ 5_____';

If an underscore or % is needed as a literal character in the string, the character should be preceded by an *escape character*, which is specified after the string using the
keyword ESCAPE.
For example, 'AB_CD\%EF' ESCAPE '\' represents the literal string 'AB_CD%EF'
Another feature allows the use of arithmetic in queries. The standard arithmetic operators for addition (+), subtraction (−), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains.

Query: Show the resulting salaries if every employee working on the
'ProductX' project is given a 10% 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';
For string data types, the concatenate operator || can be used in a query to append two string values.
For date, time, timestamp, and interval data types, operators include incrementing (+) or decrementing (−) a date, time, or timestamp by an
interval.
In addition, an interval value is the result of the difference between two date, time, or timestamp values.
Another comparison operator, which can be used for convenience, is **BETWEEN.**

Query: 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;

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.

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

SELECT D.Dname, E.Lname, E.Fname, P.Pname
FROM DEPARTMENT **AS** D, EMPLOYEE **AS** E, WORKS_ON **AS** W,
PROJECT **AS** 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** f we want to see the result in a descending order of values. The keyword **ASC** can be used to specify ascending order explicitly.

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

Discussion and Summary of Basic SQL Retrieval Queries

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

SELECT <attribute list>

FROM <table list>

[**WHERE** <condition>]

[**ORDER BY** <attribute list>];

INSERT, DELETE, and UPDATE Statements in SQL

The INSERT Command

INSERT is used to add a single tuple (row) to a relation (table).

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.

INSERT INTO EMPLOYEE

VALUES ('Richard', 'K', 'Marini', '653298653', '1962-12-30', '98

Oak Forest, Katy, TX', 'M', 37000, '653298653', 4);

A second form of the INSERT statement allows the user to specify explicit attribute

names that correspond to the values provided in the INSERT command.

INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn)

VALUES ('Richard', 'Marini', 4, '653298653');

Attributes not specified in statement are set to their DEFAULT or to NULL

A DBMS that fully implements SQL should support and enforce all the integrity constraints that can be specified in the DDL.

INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno)

VALUES ('Robert', 'Hatcher', '980760540', 2);

(U2 is rejected if referential integrity checking is provided by DBMS.)

INSERT INTO EMPLOYEE (Fname, Lname, Dno)

VALUES ('Robert', 'Hatcher', 5);

(U2A is rejected if NOT NULL checking is provided by DBMS.)

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

CREATE TABLE WORKS_ON_INFO

(Emp_name VARCHAR(15), Proj_name VARCHAR(15), Hours_per_week DECIMAL(3,1));

INSERT INTO WORKS_ON_INFO (Emp_name, Proj_name,

Hours_per_week)

SELECT E.Lname, P.Pname, W.Hours

FROM PROJECT P, WORKS_ON W, EMPLOYEE E

WHERE P.Pnumber = W.Pno **AND** W.Essn = E.Ssn;

Most DBMSs have *bulk loading* tools that allow a user to load formatted data from a file into a table without having to write a large number of INSERT commands.

Another variation for loading data is to create a new table TNEW that has the same attributes as an existing table T, and load some of the data currently in T into TNEW. The syntax for doing this uses the LIKE clause.

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

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

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

DELETE FROM EMPLOYEE **WHERE** Ssn = '123456789';

DELETE FROM EMPLOYEE **WHERE** Dno = 5;

DELETE FROM EMPLOYEE;

The UPDATE Command

The **UPDATE** command is used to modify attribute values of one or more selected tuples. As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation.

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

UPDATE PROJECT **SET** Plocation = 'Bellaire', Dnum = 5

WHERE Pnumber = 10;

UPDATE EMPLOYEE **SET** Salary = Salary * 1.1

WHERE Dno = 5;

Additional Features of SQL

1. There are techniques for specifying complex retrieval queries, including nested queries, aggregate functions, grouping, joined tables, outer joins, case statements, and recursive queries; SQL views, triggers, and assertions; and commands for schema modification.
2. SQL has various techniques for writing programs in various programming languages that include SQL statements to access one or more databases.
3. Set of commands for specifying physical database design parameters, file structures for relations, and access paths such as indexes. We called these commands a *storage definition language* (*SDL*).
4. SQL has transaction control commands. These are used to specify units of database processing for concurrency control and recovery purposes.
5. SQL has language constructs for specifying the *granting and revoking of privileges* to users.
6. SQL has language constructs for creating triggers.
7. SQL has incorporated many features from object-oriented models to have more powerful capabilities, leading to enhanced relational systems known as **object-relational.**
8. SQL and relational databases can interact with new technologies such as XML.