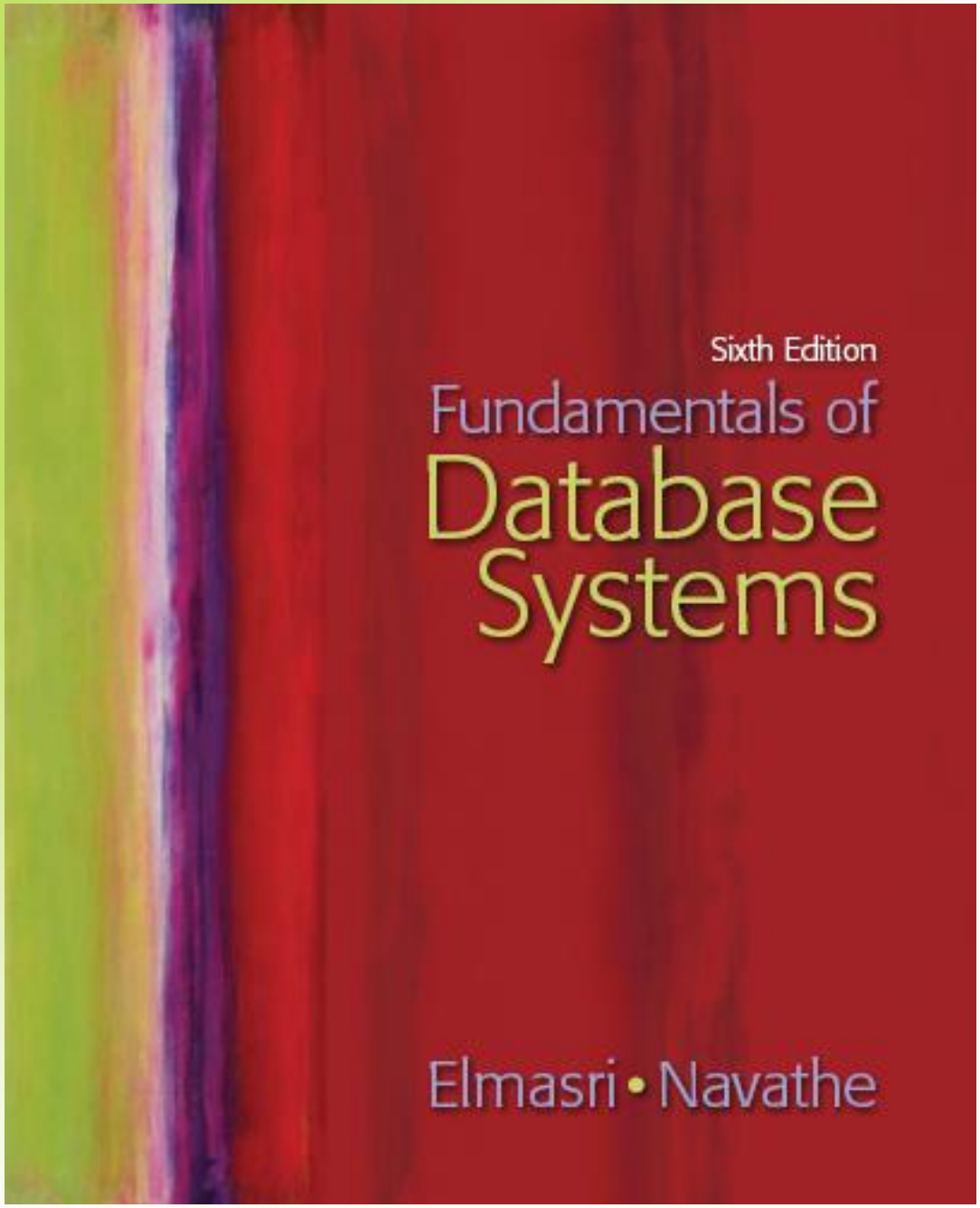
Chapter 4

Basic SQL



Addison-Wesley
is an imprint of
PEARSON

Chapter 4

Basic SQL

Chapter 4 Outline

- Overview of SQL
- SQL Data Definition (DDL) for Specifying a Relational Database Schema
  - CREATE statements
  - Specifying Constraints
- Basic Retrieval Queries in SQL
  - SELECT … FROM … WHERE … statements
- Basic Database Modification in SQL
  - INSERT, DELETE, UPDATE statements

Overview of SQL

- SQL is a standard, comprehensive language, based on the relational model
- SQL includes capabilities for many functions:
  - DDL statements for creating schemas and specifying data types and constraints
  - Statements for specifying database retrievals
  - Statements for modifying the database
  - Statements for specifying views, triggers, and assertions (see Chapter 5)
  - Many other capabilities
- We introduce the first three capabilities in this chapter

SQL Features in This Chapter

- Basic SQL DDL
  - Includes the CREATE statements
  - Has a comprehensive set of SQL data types
  - Can specify key, referential integrity, and other constraints
  - Object-oriented features of SQL DDL will be presented in Chapter 11
- Basic Retrieval Queries in SQL
  - SELECT … FROM … WHERE … statements
- Basic Database Modification in SQL
  - INSERT, DELETE, UPDATE statements

SQL: Origins and History

- Evolution of the SQL Standard
  - First standard approved in 1989 (called SQL-89 or SQL1)
  - Comprehensive revision in 1992 (SQL-92 or SQL2)
  - Third big revision in 1999 (SQL-99, a trimmed-down version of a more ambitious revision known as SQL3)
  - Other revisions known as SQL:2003, SQL:2006, SQL:2008
- Origins of SQL
  - Originally called SEQUEL (Structured English Query Language), then SQL (Structured Query Language)
  - Developed at IBM Research for experimental relational DBMS called *System-R* in the 1970s

SQL Data Definition

- CREATE statement can be used to:
  - Create a named database schema
  - Create individual database tables and specify data types for the table attributes, as well as *key*, *referential integrity*, and *NOT NULL* constraints
  - Create named constraints
- Other commands can modify the tables and constraints
  - DROP and ALTER statements (discussed in Chapter 5)

CREATE TABLE

- In its *simplest form*, specifies a new base relation by giving it a name, and specifying each of its attributes and their data types (INTEGER, FLOAT, DECIMAL(i,j), CHAR(n), VARCHAR(n), DATE, and other data types)
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE DEPARTMENT (
 DNAME   VARCHAR(15)  NOT NULL,
 DNUMBER    INT       NOT NULL,
 MGRSSN     CHAR(9)   NOT NULL,
 MGRSTARTDATE DATE   );
```

CREATE TABLE (cont.)

- CREATE TABLE can also specify the primary key, UNIQUE keys, and referential integrity constraints (foreign keys)
- Via the PRIMARY KEY, UNIQUE, and FOREIGN KEY phrases

```
CREATE TABLE DEPARTMENT (
  DNAME       VARCHAR(15)  NOT NULL,
  DNUMBER     INT          NOT NULL,
  MGRSSN      CHAR(9)      NOT NULL,
  MGRSTARTDATE  DATE,
  PRIMARY KEY (DNUMBER),
  UNIQUE (DNAME),
  FOREIGN KEY(MGRSSN) REFERENCES EMPLOYEE);
```

Example: The COMPANY Database

- Figure 3.7 (next slide) shows the COMPANY database *schema diagram* introduced in Chapter 3
- Referential integrity constraints shown as directed edges from *foreign key* to *referenced relation*
- Primary key attributes of each table *underlined*

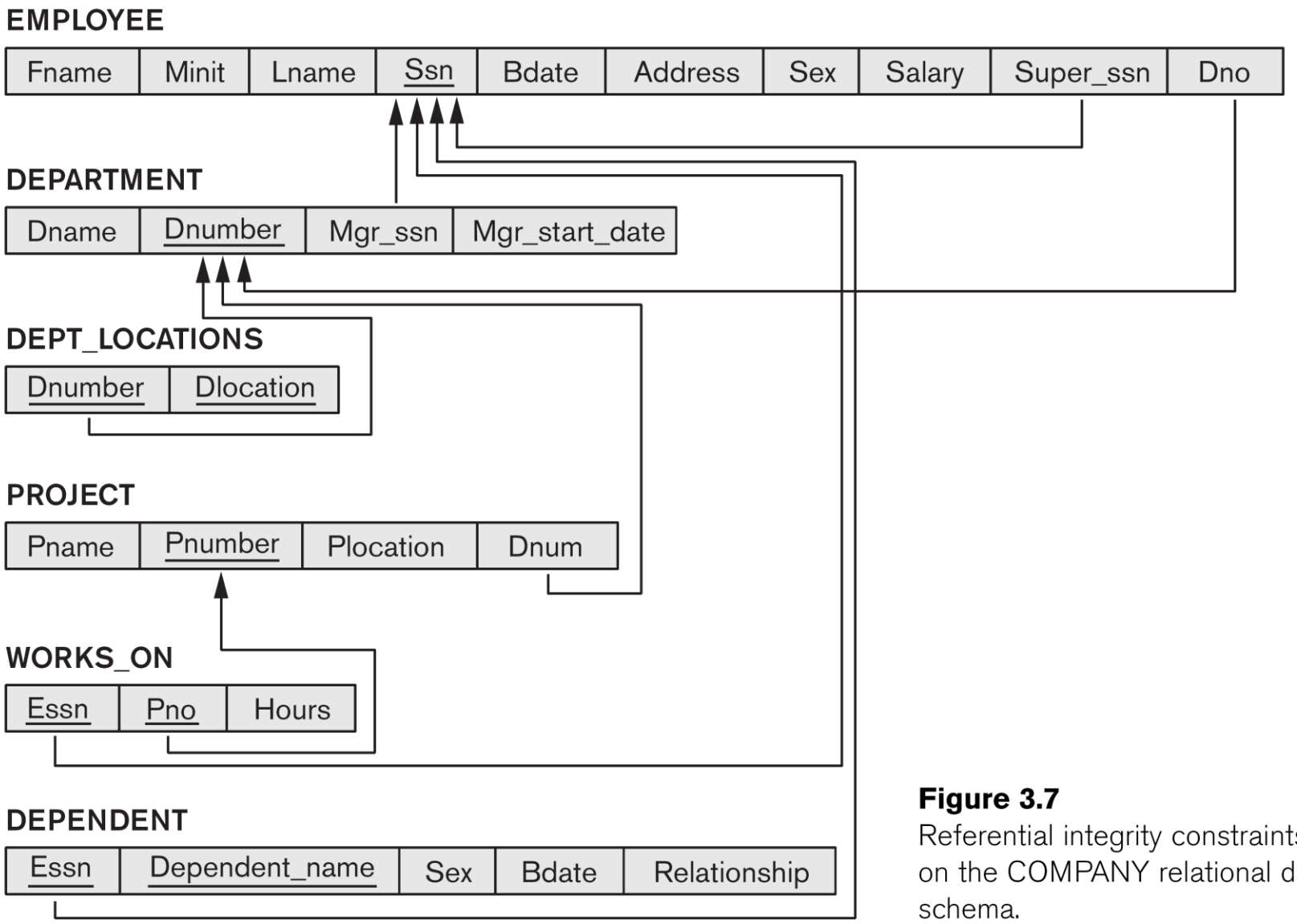


Figure 3.7
Referential integrity constraints displayed on the COMPANY relational database schema.

Example: The COMPANY DDL

- Figure 4.1 (following two slides) shows *example DDL* for creating the tables of the COMPANY database
- Circular reference problem:
  - In Figure 3.7, some foreign keys cause *circular references*:
    - EMPLOYEE.Dno → DEPARTMENT.Dnumber
    - DEPARTMENT.Mgr_ssn → EMPLOYEE.Ssn
  - One of the tables must be created first without the FOREIGN KEY in the CREATE TABLE statement
    - The missing FOREIGN KEY can be added later using ALTER TABLE (see Chapter 5)

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

Figure 4.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 3.7.

Continued next page…

```
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) );
```

The CREATE SCHEMA Statement

- A DBMS can manage multiple databases
- DBA (or authorized users) can use CREATE SCHEMA to have distinct databases; for example:

```
CREATE SCHEMA COMPANY AUTHORIZATION 'Smith';
```

- Each database has a schema name (e.g. COMPANY)
- User 'Smith' will be “owner” of schema, can create tables, other constructs in that schema
- Table names can be prefixed by schema name if multiple schemas exist (e.g. COMPANY.EMPLOYEE)
- Prefix not needed if there is a “default schema” for the user

Basic SQL Data Types

- **Basic numeric data types**:
  - Integers: INTEGER (or INT), SMALLINT
  - Real (floating point): FLOAT (or REAL), DOUBLE PRECISION
  - Formatted: DECIMAL(i,j) (or DEC (i,j) or NUMERIC(i,j)) specify i total decimal digits, j after decimal point
    - i called ***precision***, j called ***scale***
- **Basic character string data types:**
  - Fixed-length: CHAR(n) or CHARACTER(n)
  - Variable-length: VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n)
  - Placed in single quotation marks, case sensitive
  - Fixed length strings padded with blanks to the right
  - Alphabeticaly ordered
  - Concatenation operator ||
  - CLOB

SQL Data Types (cont)

- Bit – string data types:
  - BIT(n), BIT VARYING (n), default length of n is 1
  - Literal bit strings – B'0101'
  - Variable length bit strings – BLOB in Kb/Mb/Gb
- Boolean data types:
  - TRUE/FALSE/UNKNOWN
- Large object data types:
  - Binary large objects: BLOB(n)
  - Can be used to store attributes that represent images, audio, video, or other large binary objects
  - Character large objects: CLOB(n)
  - Can be used attributes that store articles, news stories, text files, and other large character objects

SQL Data Types (cont)

- DATE data type:
  - Standard DATE formatted as yyyy-mm-dd
  - For example, DATE '2010-01-22'
  - Older formats still used in some systems, such as 'JAN-22-2010'
  - Values are ordered, with later dates larger than earlier ones
- TIME data type:
  - Standard TIME formatted as hh:mm:ss
  - E.g., TIME '15:20:22' represents 3.20pm and 22 seconds
  - TIME(i) includes and additional i-1 decimal digits for fractions of a second
  - E.g., TIME(5) value could be '15:20:22.1234‘
  - < comparison works for DATE and TIME
  - Literal values – DATE’2008-09-27’ or TIME’09:10:12’

SQL Data Types (cont)

- TIMESTAMP data type:
  - A DATE combined with a TIME(i), where the default i=7
  - For example, TIMESTAMP '2010-01-22 15:20:22.123456'
  - A different i>7 can be specified if needed
- INTERVAL
  - represents a relative YEAR/MONTH or DAY/TIME (e.g. 2 years and 5 months, or 5 days 20 hours 3 minutes 22 seconds)
  - Can be used to increment/decrement an absolute value of DATE/TIME/TIMESTAMP
- Other SQL data types exist; we presented the most common ones

Specifying Constraints in SQL

Basic constraints in SQL as part of table creation (Key constraints, Referential integrity constraints, Restrictions on attribute domains & NULLs)

Attribute constraints & attribute defaults

- NOT NULL constraints, implicit for key attributes, can be defined for others.
- Default value for an attribute –
- Restrict attribute or domain value using CHECK

Key and Referential Integrity constraints

- PRIMARY key – PRIMARY KEY clause
- UNIQUE key –
- FOREIGN KEY clause

Specifying Referential Integrity Options and Naming of Constraints

- We can specify RESTRICT (the default), CASCADE, SET NULL or SET DEFAULT on referential integrity constraints (foreign keys)
- Separate options for ON DELETE, ON UPDATE
- Figure 4.2 (next slide) gives some examples
- A constraint can be given a *constraint name*; this allows to DROP the constraint later (see Chapter 5)
- Constraint names should be unique in a schema
- Figure 4.2 illustrates naming of constraints

```
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);
```

Figure 4.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

Specifying DOMAIN Constraints Using the CHECK Clause

- Another type of constraint can be specified using CHECK
- E.g., CHECK (Dnumber > 0 AND Dnumber < 21) can be specified after the Dnumber attribute
- Alternatively, a special domain can be created with a domain name using CREATE DOMAIN
- E.g. CREATE DOMAIN D_NUM AS INTEGER CHECK (Dnumber > 0 AND Dnumber < 21);
- D_NUM can now be used as the data type for the Dnumber attribute of the DEPARTMENT table, as well as for Dnum of PROJECT, Dno of EMPLOYEE, and so on
- CHECK can also be used to specify more general constraints (see Chapter 5)

Basic Retrieval Queries in SQL

- SQL basic statement for retrieving information from a database is the **SELECT** statement
  - NOTE: This is *not the same as* the SELECT operation of the relational algebra (see Chapter 6)
- Important distinction between *practical SQL model* and the *formal relational model* (see Chapter 3):
  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
  - Hence, an SQL relation (table) is a **multi-set** (sometimes called a **bag**) of tuples; it is *not* a set of tuples
- SQL relations can be *constrained to be sets* by specifying PRIMARY KEY or UNIQUE attributes, or by using the DISTINCT option in a query

Bag (Multiset) versus Set

- A **bag** or **multi-set** is like a set, but an element may appear more than once
  - Example: {A, B, C, A} is a bag (but *not* a set). {A, B, C} is a bag (and also a set).
  - Bags also resemble lists, but the order is irrelevant in a bag.
- Example:
  - {A, B, A} = {B, A, A} as bags
  - However, [A, B, A] is not equal to [B, A, A] as lists

Basic Retrieval Queries in SQL (cont.)

- Simplest form of the SQL SELECT statement is called a *mapping* or a SELECT-FROM-WHERE *block*
- Our examples use the COMPANY database schema

- **SELECT** <attribute list>
  FROM <table list>
  WHERE <condition>

  - <attribute list> is a list of attribute names whose values are to be retrieved by the query, also called *projection attributes*
  - <table list> is a list of the table (relation) names required to process the query
  - <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query, also known as *selection condition*

Simple SQL Queries (cont.)

- Query text ends with a semi-colon
- First example of a simple query on one table (relation)
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'
- Use the EMPLOYEE table only

```
Q0:SELECT   BDATE, ADDRESS
   FROM     EMPLOYEE
   WHERE    FNAME='John' AND MINIT='B'
            AND LNAME='Smith' ;
```

Simple SQL Queries (cont.)

- Example of a query that uses two tables
- 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 ;
```

  - (DNAME='Research') is called a *selection condition*
  - (DNUMBER=DNO) is called a *join condition* (it *joins* two tuples from EMPLOYEE and DEPARTMENT tables whenever EMPLOYEE.DNO=DEPARTMENT.DNUMBER)
  - *Select-project-join* query

Simple SQL Queries (cont.)

- A *selection condition* refers to attributes from a single table, and selects (chooses) those records that satisfy the condition
- A *join condition generally* refers to attributes from two tables, and joins (or combines) pairs of records that satisfy the condition
- In the previous query:
  - (DNAME='Research') chooses the DEPARTMENT record
  - (DNUMBER=DNO) joins the record with each EMPLOYEE record that works for that department

Simple SQL Queries (cont.)

- Query 2 (needs 3 tables): For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```
Q2: SELECT   PNUMBER, DNUM, LNAME, BDATE, ADDRESS
    FROM     PROJECT, DEPARTMENT, EMPLOYEE
    WHERE    DNUM=DNUMBER AND MGRSSN=SSN
             AND PLOCATION='Stafford' ;
```

  - In Q2, there are two join conditions
  - The join condition DNUM=DNUMBER relates a PROJECT record to its controlling DEPARTMENT record
  - The join condition MGRSSN=SSN relates the controlling DEPARTMENT to the EMPLOYEE who manages that department

Qualifying Attribute Name with Table Name

- An attribute name in an SQL query can be *prefixed* (or *qualified*) with the table name
- Examples:

  EMPLOYEE.LNAME

  DEPARTMENT.DNAME

Query Q1 can be rewritten as:

```
SELECT   EMPLOYEE.FNAME, EMPLOYEE.LNAME,
   EMPLOYEE. ADDRESS
FROM          EMPLOYEE, DEPARTMENT
WHERE    DEPARTMENT.DNAME='Research' AND
   DEPARTMENT.DNUMBER=EMPLOYEE.DNO ;
```

Aliasing Table Names Using Tuple Variables

- An *alias* (or *tuple variable*) can be used instead of the table name when prefixing attribute names
- Example:

Query Q1 can be rewritten as follows using the aliases D for DEPARTMENT and E for EMPLOYEE:

```
SELECT  E.FNAME, E.LNAME, E. ADDRESS
FROM        EMPLOYEE AS E, DEPARTMENT AS D
WHERE   D.DNAME='Research' AND D.DNUMBER=E.DNO ;
```

Renaming of Attributes

- It is also possible to **rename** the attributes of a table within a query; new attribute names are listed in the same order that the attributes where specified in CREATE TABLE
- This is sometimes necessary for certain joins (see Chapter 5)
- Example: Query Q1 can be rewritten as follows:

```
SELECT   E.FN, E.LN, E. ADR
FROM     DEPARTMENT AS D(DNM, DNO, MSSN, STRDATE),
         EMPLOYEE AS E(FN,MI,LN,S,BD,ADR,SX,SAL,SU,DNO)
WHERE    D.DNM='Research' AND D.DNO=E.DNO ;
```

Queries that Require Qualifying of Attributes Names

- In SQL, all attribute names in a particular table must be different
  - However, the same attribute name can be used in *different tables*
  - In this case, it is required to use aliases if both tables are used in the same query
  - Example: **Suppose** that the attribute name NAME was used for both DEPARTMENT name and PROJECT name

Query: For each project, retrieve the project's name, and the name of its controlling department.

```
Q:  SELECT   P.NAME, D.NAME
    FROM     PROJECT AS P, DEPARTMENT AS D
    WHERE    P.DNUM=D.DNUMBER ;
```

In Q, P.NAME refers to the NAME attribute in the PROJECT table, and D.NAME refers to the NAME attribute in the DEPARTMENTtable,

Queries that Require Qualifying of Attributes (cont.)

- Some queries need to refer to the same relation twice
  - In this case, *aliases* are also required

Query 8: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
Q8:  SELECT   E.FNAME, E.LNAME, S.FNAME, S.LNAME
     FROM     EMPLOYEE AS E , EMPLOYEE AS S
     WHERE    E.SUPERSSN=S.SSN ;
```

  - In Q8, E and S are two different *aliases* or *tuple variables* for the EMPLOYEE relation
  - We can think of E and S as two different *copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*
  - The join condition joins two different employee records together (a supervisor S and a subordinate E)

Missing WHERE-clause

- The *WHERE-clause* is **optional** in an SQL query
- A *missing WHERE-clause* indicates no condition; hence, all tuples of the relations in the FROM-clause are selected
  - This is equivalent to the condition WHERE TRUE
- Example: Retrieve the SSN values for all employees.

```
Q9:   SELECT   SSN
      FROM     EMPLOYEE ;
```

- If more than one relation is specified in the FROM-clause *and* there is no WHERE-clause (hence no join conditions), then *all possible combinations* of tuples are joined together (known as *CARTESIAN PRODUCT* of the relations)

Missing WHERE-clause (cont.)

- Example:

```
Q10:  SELECT SSN, DNAME
      FROM   EMPLOYEE, DEPARTMENT
```

  - In this query, every EMPLOYEE is joined with every DEPARTMENT
  - It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large query results are produced

Retrieving all the Attributes Using Asterisk (*)

- To retrieve all the attribute values of the selected tuples, a * (asterisk) is used, which stands for *all the attributes* Examples:

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

Q1D:SELECT   *
     FROM    EMPLOYEE, DEPARTMENT
     WHERE   DNAME='Research' AND
                              DNO=DNUMBER ;
```

Eliminating Duplicates Using DISTINCT

- As mentioned earlier, SQL does not treat a relation as a set but a multiset (or bag); duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- Example: Result of Q11 may have duplicate SALARY values; result of Q11A does not have any duplicate values

```
Q11:  SELECT    SALARY
      FROM      EMPLOYEE
Q11A: SELECT    DISTINCT SALARY
      FROM      EMPLOYEE
```

Set and Multiset Operations in SQL

- SQL has directly incorporated some set operations
- The set operations are: union (UNION), set difference (EXCEPT or MINUS) and intersection (INTERSECT)
- Results of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- Set operations apply only to *type compatible* relations (also called *union compatible*); the two relations must have the same attribute types and in the same order
- Set operations typically applied to the results of two separate queries; e.g Q1 UNION Q2

Set Operations (cont.)

- Example: Query 4: Make a list of all project names for projects that involve an employee whose last name is 'Smith' as a worker on the project or as a manager of the department that controls the project.

```
Q4:     (SELECT   PNAME
        FROM      PROJECT, DEPARTMENT, EMPLOYEE
        WHERE     DNUM=DNUMBER AND
    MGRSSN=SSN AND LNAME='Smith')
        UNION
        (SELECT   PNAME
        FROM      PROJECT, WORKS_ON, EMPLOYEE
        WHERE     PNUMBER=PNO AND
                  ESSN=SSN AND LNAME='Smith') ;
```

Multiset Operations in SQL

- SQL has multiset operations when the user does not want to eliminate duplicates from the query results
- These are: UNION ALL, EXCEPT ALL, and INTERSECT ALL; see examples in Figure 4.5 (next slide)
- Results of these operations are multisets of tuples; *all tuples and duplicates in the input tables are considered* when computing the result
- Multiset operations also apply only to *type compatible* relations
- Typically applied to the results of two separate queries; e.g Q1 UNION ALL Q2 ;

(a)

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

(b)

T

A
a1
a1
a2
a2
a2
a3
a4
a5

(c)

T

A
a2
a3

(d)

T

A
a1
a2

Figure 4.5
The results of SQL multiset operations. (a) Two tables, R(A) and S(A). (b) R(A) UNION ALL S(A). (c) R(A) EXCEPT ALL S(A). (d) R(A) INTERSECT ALL S(A).

Substring Comparison Conditions

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used: '*****' (or '**%**' in some implementations) replaces an arbitrary number of characters, and '_' replaces a single arbitrary character
- Conditions can be used in WHERE-clause

Substring Comparison Conditions (cont.)

- Example: Query 25: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX‘ in it.

```
Q25: SELECT     FNAME, LNAME
     FROM    EMPLOYEE
     WHERE ADDRESS LIKE '*Houston,TX*' ;
```

Substring Comparison Conditions (cont.)

- Example: Query 26: Retrieve all employees who were born during the 1950s.
  - Here, '5' must be the 3rd character of the string (according to the standard format for DATE yyyy-mm-dd), so the BDATE value is '__5_____', with each underscore as a place holder for a single arbitrary character.

```
Q26:  SELECT  FNAME, LNAME
      FROM    EMPLOYEE
      WHERE   BDATE LIKE  '__5_____';
```

- The LIKE operator allows users to get around the fact that each value is considered atomic and indivisible
  - Hence, in SQL, character string attribute values are not atomic

Applying Arithmetic in SQL Queries

- The standard arithmetic operators **'+', '-'. '*',** and **'/'** (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric attributes and values in an SQL query
- Example: Query 27: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q27:  SELECT   FNAME, LNAME, 1.1*SALARY
      FROM     EMPLOYEE, WORKS_ON, PROJECT
      WHERE    SSN=ESSN AND PNO=PNUMBER
                        AND PNAME='ProductX' ;
```

Ordering a query result using the ORDER BY clause

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- Example: Query 28: Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
Q28:  SELECT   DNAME, LNAME, FNAME, PNAME
      FROM     DEPARTMENT, EMPLOYEE,
                                WORKS_ON, PROJECT
      WHERE    DNUMBER=DNO AND SSN=ESSN
                    AND PNO=PNUMBER
      ORDER BY    DNAME, LNAME ;
```

ORDER BY (cont.)

- The default order is in ascending order of values
- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default
- Without ORDER BY, the rows in a query result appear in some system-determined (random) order

Summary of Basic SQL Queries

- A basic query in SQL can consist of up to four clauses, but only the first two, SELECT and FROM, are mandatory
- Two additional clauses (GROUP BY and HAVING) will be discussed in Chapter 5
- The four basic clauses are specified in the following order:

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

SQL Statements for Updating the Database

- There are three SQL commands to modify the database: **INSERT**, **DELETE**, and **UPDATE**
- INSERT is used for adding one or more records to a table
- DELETE is for removing records
- UPDATE is for modifying existing records
- Some operations may be rejected if they violate integrity constraints; others may propagate additional updating automatically if specified in the database schema

INSERT statement

- Used to add one or more tuples (records) to a relation (table)
- Values for the attributes should be listed in the same order as the attributes were specified in the **CREATE TABLE** command
- Attributes that have defaults values can be omitted in the new record(s)

INSERT statement (cont.)

- Example:

  U1: INSERT INTO EMPLOYEE
  VALUES ('Richard','K','Marini', '653298653', '30-DEC-52',
  '98 Oak Forest,Katy,TX', 'M', 37000,'987654321', 4) ;

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
  - Attributes with NULL or default values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

  U1A: INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
  VALUES ('Richard', 'Marini', '653298653') ;

INSERT statement (cont.)

- Constraints specified in the DDL are automatically enforced by the DBMS when updates are applied to the database
- Can insert multiple tuples in one INSERT statement
  - The values in each tuple are enclosed in parentheses

INSERT statement (cont.)

- Can also insert tuples from a query result into a table
- Example: Suppose we want to create a temporary table that has the employee last name, project name, and hours per week for each employee.
  - A table WORKS_ON_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A:  CREATE TABLE WORKS_ON_INFO
          (EMP_NAME          VARCHAR(15),
           PROJ_NAME         VARCHAR(15),
           HOURS_PER_WEEK    DECIMAL(3,1));

U3B:  INSERT INTO  WORKS_ON_INFO (EMP_NAME,
                   PROJ_NAME, HOURS_PER_WEEK)
      SELECT       E.LNAME, P.PNAME, W.HOURS
      FROM         EMPLOYEE E, PROJECT P, WORKS_ON W
      WHERE        E.SSN=W.ESSN AND W.PNO=P.PNUMBER ;
```

INSERT statement (cont.)

- Note: The WORKS_ON_INFO table may not be up-to-date if we change the tuples in the WORKS_ON, PROJECT, or EMPLOYEE relations *after* executing U3B.
- We have to use CREATE VIEW (see Chapter 5) to keep such a table up to date.

DELETE Statement

- Removes tuples from a relation
  - Includes a WHERE-clause to select the tuples to be deleted
  - Referential integrity should be enforced (via REJECT, CASCADE, SET NULL, or SET DEFAULT)
  - Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
  - Missing WHERE-clause deletes *all tuples* in the relation; the table then becomes an empty table
  - Number of tuples deleted is the number of tuples selected by the WHERE-clause

DELETE Statement (cont.)

- Examples:

```
U4A:  DELETE FROM   EMPLOYEE
      WHERE         LNAME='Brown' ;

U4B:  DELETE FROM   EMPLOYEE
      WHERE         SSN='123456789' ;

U4C:  DELETE FROM   EMPLOYEE
      WHERE         DNO  = 5 ;

U4D:  DELETE FROM   EMPLOYEE ;
```

UPDATE Statement

- 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 should be enforced (via REJECT, CASCADE, SET NULL, or SET DEFAULT)

UPDATE Statement (cont.)

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

```
U5:   UPDATE    PROJECT
      SET  PLOCATION = 'Bellaire', DNUM = 5
      WHEREPNUMBER=10 ;
```

UPDATE Statement (cont.)

- Example: Give all employees in department number 5 a 10% raise in salary

```
U6:    UPDATE   EMPLOYEE
       SET      SALARY = SALARY *1.1
       WHERE    DNO = 5 ;
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
  - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
  - The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification

Some Other Features of SQL

- More complex query features: nested queries, aggregate functions, GROUP BY, HAVING, EXISTS function (see Chapter 5)
- Views, assertions, schema modification (Chapter 5)
- Triggers (Chapter 5 and Section 26.1)
- Object-relational features (Chapter 11, Section 11.2)
- Programming techniques (Chapter 13)
- Transaction features (Chapter 21, Section 21.6)
- Database Security (Chapter 24, Section 24.2)

Chapter 4 Summary

- Overview of SQL
- SQL Data Definition (DDL) for Specifying a Relational Database Schema
  - CREATE statements
  - Specifying Constraints
- Basic Retrieval Queries in SQL
  - SELECT … FROM … WHERE … statements
- Basic Database Modification in SQL
  - INSERT, DELETE, UPDATE statements