Lecture 4: Basic SQL (Structured Query Language)

CS3402 Database Systems

Relational Query Languages

- Query languages
  - Data Definition Language (DDL): standard commands for defining the structures in a database. DDL statements create, modify, and remove database objects such as tables, indexes, and users. Common DDL statements are CREATE, ALTER, and DROP.
  - Data Manipulation Language (DML): standard commands for dealing with the manipulation of data present in database. Common DDL statements SELECT, INSERT, UPDATE, and DELETE.
- Each statement in SQL ends with a semicolon (;)

CREATE SCHEMA Statement

- A schema is a way to logically group objects in a single collection and provide a unique namespace for objects.
- The CREATE SCHEMA statement is used to create a schema. A schema name cannot exceed 128 characters. Schema names must be unique within the database.
- Syntax
  - CREATE SCHEMA schema-name AUTHORIZATION user-name
- Example
  - CREATE SCHEMA COMPANY AUTHORIZATION ‘Jsmith’;

Attribute Data Types and Domains: Numeric

- **Numeric** data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION).
- Formatted numbers can be declared by using DECIMAL(i, j) , where i (i.e., the precision) is the total number of decimal digits and j (i.e., the scale) is the number of digits after the decimal point. The default for scale is zero, and the default for precision is implementation-defined.

Attribute Data Types and Domains: Character-string

- **Character-string** data types are either fixed length—CHAR(n) or CHARACTER(n), where n is the number of characters—or varying length—VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.
- When specifying a literal string value, it is placed between single quotation marks (apostrophes), and it is case sensitive (a distinction is made between uppercase and lowercase).
- For fixed length strings, a shorter string is padded with blank characters to the right. For example, if the value ‘Smith’ is for an attribute of type CHAR(10), it is padded with five blank characters to become ‘Smith’ if needed. Padded blanks are generally ignored when strings are compared.

Attribute Data Types and Domains: Date

- 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.
- Only valid dates and times should be allowed by the SQL implementation.
- Literal values are represented by single-quoted strings preceded by the keyword DATE or TIME; for example, DATE '2014-09-27' or TIME '09:12:47'.

Attribute Data Types and Domains: Boolean and Timestamp

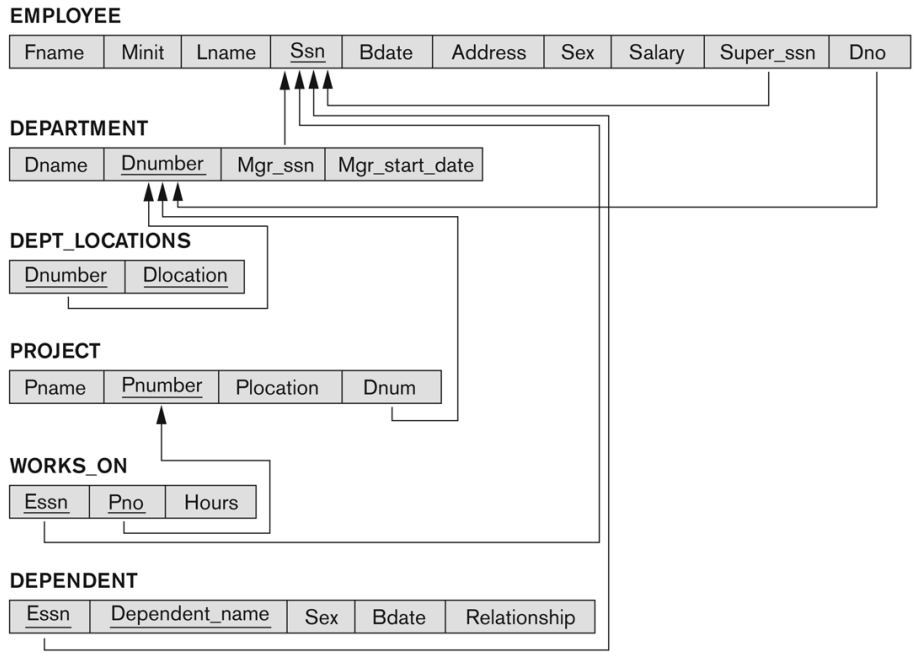
- A **Boolean** data type has the traditional values of TRUE or FALSE. In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is UNKNOWN.
- A **timestamp** data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds.
  - Literal values are represented by single-quoted strings preceded by the keyword TIMESTAMP, with a blank space between data and time; for example, TIMESTAMP '2014-09-27 09:12:47.648302'

Attribute Constraints and Defaults

- Because SQL allows NULLs as attribute values, a constraint NOT NULL may be specified if NULL is not permitted for a particular attribute.
- This is always implicitly specified for the attributes that are part of the primary key of each relation, but it can be specified for any other attributes whose values are required not to be NULL
- It is also possible to define a default value for an attribute by appending the clause DEFAULT <value> to an attribute definition. The default value is included in any new tuple if an explicit value is not provided for that attribute.

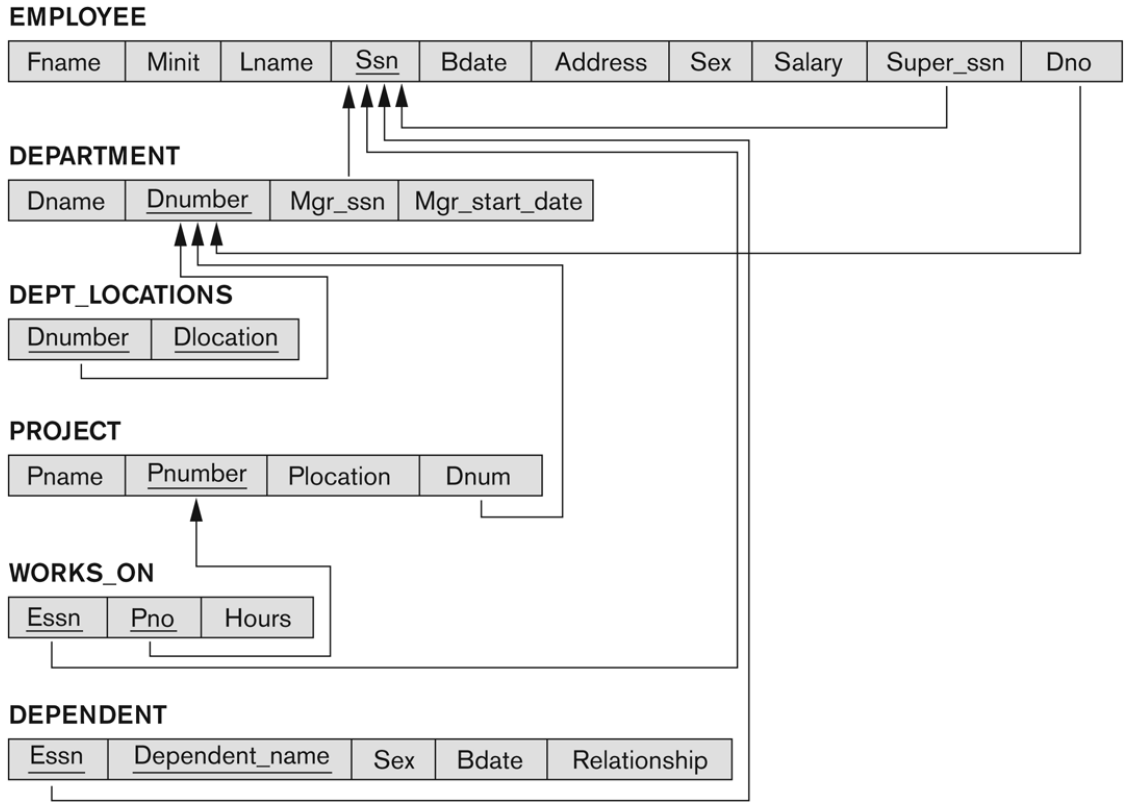
CREATE TABLE Statement (1/4)

- The relations declared through CREATE TABLE statements are called **base tables** (or base relations). Tables contain columns and constraints, rules to which data must conform. Table-level constraints specify a column or columns. Columns have a data type and can specify column constraints (column-level constraints).
- Create 6 tables
  - EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, & DEPENDENT



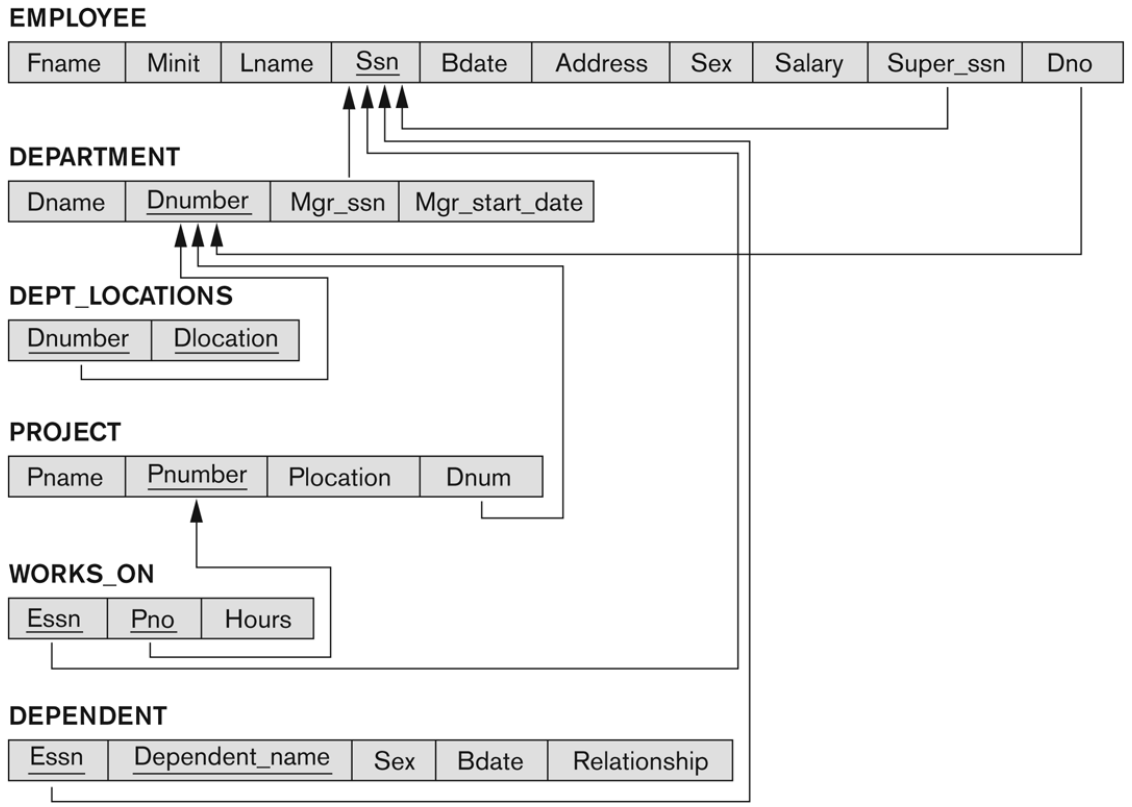
CREATE TABLE Statement (2/4)

```
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 Statement (3/4)

```
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) );
```



CREATE TABLE Statement (4/4)

- The PRIMARY KEY clause specifies one or more attributes that make up the primary key of a relation.
- The UNIQUE clause specifies alternate (unique) keys, also known as candidate keys, as in the DEPARTMENT and PROJECT table declarations.
- Referential integrity is specified via the FOREIGN KEY clause.

Specifying Key and Referential Integrity Constraints (1/2)

- Referential integrity is specified via the FOREIGN KEY clause.
- The schema designer can specify an alternative action to be taken by attaching a **referential triggered action** clause to any foreign key constraint. The options include SET NULL, CASCADE, and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE.
- In general, the action taken by the DBMS for SET NULL or SET DEFAULT is the same for both ON DELETE and ON UPDATE: The value of the affected referencing attributes is changed to NULL for SET NULL and to the specified default value of the referencing attribute for SET DEFAULT.
- The action for CASCADE ON DELETE is to delete all the referencing tuples, whereas the action for CASCADE ON UPDATE is to change the value of the referencing foreign key attribute(s) to the updated (new) primary key value for all the referencing tuples.

Specifying Key and Referential Integrity Constraints (2/2)

- For example, the database designer chooses ON DELETE SET NULL and ON UPDATE CASCADE for the foreign key Super_ssn of EMPLOYEE.
- This means that if the tuple for a supervising employee is deleted, the value of Super_ssn is automatically set to NULL for all employee tuples that were referencing the deleted employee tuple.
- On the other hand, if the Ssn value for a supervising employee is updated (say, because it was entered incorrectly), the new value is cascaded to Super_ssn for all employee tuples referencing the updated employee tuple.

```
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);
```

Table Manipulation

- The ALTER TABLE statement allows you to
  - Add a column to a table
  - Add a constraint to a table
  - Drop a column from a table
  - Drop an existing constraint from a table
  - Increase the width of a VARCHAR column
  - Change the default value for a column
  - …
- DROP TABLE statement removes the specified table.

The Database State for the COMPANY Relational Database Schema

EMPLOYEE

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

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

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

WORKS_ON

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

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

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

SELECT Statement (1/7)

- Queries in SQL can be very complex. We will start with simple queries.
- The basic form of the SELECT statement formed of the three clauses SELECT, FROM, and WHERE and has the following form:

```
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.
- <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.

SELECT Statement (2/7)

- In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <, <=, >, >=, and <>.
- These correspond to the relational algebra operators =, <, ≤, >, ≥, and ≠, respectively, and to the C/C++ programming language operators =, <, <=, >, >=, and !=.
- The main syntactic difference is the not equal operator.
- SQL has additional comparison operators.

SELECT Statement (3/7)

- Example 1: Retrieve the birth date and address of the employee(s) whose name is ‘John B. Smith’.

```
SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname='John' AND Minit='B' AND Lname='Smith';
```

- This query involves only the EMPLOYEE relation listed in the FROM clause. The query selects the individual EMPLOYEE tuples that satisfy the condition of the WHERE clause, then projects the result on the Bdate and Address attributes listed in the SELECT clause.

SELECT Statement (4/7)

- Example 1: Retrieve the birth date and address of the employee(s) whose name is ‘John B. Smith’.

```
SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname='John' AND Minit='B' AND Lname='Smith';
```

- This query involves only the EMPLOYEE relation listed in the FROM clause. The query selects the individual EMPLOYEE tuples that satisfy the condition of the WHERE clause, then projects the result on the Bdate and Address attributes listed in the SELECT clause.

SELECT Statement (5/7)

- Example 2: Retrieve the name and address of all employees who work for the 'Research' department.

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

Result:

Fname	Lname	Address
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

- In the WHERE clause
  - The condition Dname='Research' is a **selection condition** that chooses the particular tuple of interest in the DEPARTMENT table, because Dname is an attribute of DEPARTMENT.
  - The condition Dnumber=Dno is called a **join condition**, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to the value of Dno in EMPLOYEE.

SELECT Statement (6/7)

- Example 3: For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birth date.

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford'
```

- The join condition Dnum=Dnumber relates a project tuple to its controlling department tuple.
- The join condition Mgr_ssn=Ssn relates the controlling department tuple to the employee tuple who manages that department.
- Each tuple in the result will be a combination of one project, one department (that controls the project), and one employee (that manages the department).

SELECT Statement (7/7)

- Example 3 (continued):

```
SELECT Pnumber, Dnum, Lname, Address, Bdate
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Plocation='Stafford'
```

Result:

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

Ambiguous Attribute Names

- In SQL, 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, e.g.,

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

Aliasing, Renaming, and Tuple Variables (1/2)

- The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice.
- For example, for each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT E.Fname, E.Lname, S.Fname, S.Lname
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.Super_ssn = S.Ssn;
```

- In this case, we are required to declare alternative relation names E and S, called **aliases** or **tuple variables**, for the EMPLOYEE relation.

Result:

E.Fname	E.Lname	S.Fname	S.Lname
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Alicia	Zelaya	Jennifer	Wallace
Jennifer	Wallace	James	Borg
Ramesh	Narayan	Franklin	Wong
Joyce	English	Franklin	Wong
Ahmad	Jabbar	Jennifer	Wallace

Aliasing, Renaming, and Tuple Variables (2/2)

- 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.
- In fact, this practice is recommended since it results in queries that are easier to comprehend.
- For example, we could specify query in Example 2 as:

```
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 (1/2)

- 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.
- For example, Select all EMPLOYEE Ssns:

```
SELECT Ssn
FROM EMPLOYEE;
```

Result:

Ssn
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555

Unspecified WHERE Clause (2/2)

- If more than one relation is specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT—all possible tuple combinations—of these relations is selected.
- For example, all combinations of EMPLOYEE Ssn and DEPARTMENT Dname:

```
SELECT Ssn, Dname
FROM EMPLOYEE, DEPARTMENT;
```

Result:

Ssn	Dname
123456789	Research
333445555	Research
999887777	Research
987654321	Research
666884444	Research
453453453	Research
987987987	Research
888665555	Research
123456789	Administration
333445555	Administration
999887777	Administration
987654321	Administration
666884444	Administration
453453453	Administration
987987987	Administration
888665555	Administration
123456789	Headquarters
333445555	Headquarters
999887777	Headquarters
987654321	Headquarters
666884444	Headquarters
453453453	Headquarters
987987987	Headquarters
888665555	Headquarters

Use of the Asterisk (*) (1/2)

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

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

Use of the Asterisk (*) (2/2)

- 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;
```

- Specify the CROSS PRODUCT of the EMPLOYEE and DEPARTMENT relations

```
SELECT *
FROM EMPLOYEE, DEPARTMENT;
```

Substring Pattern Matching (1/2)

- The **LIKE** comparison allows comparison conditions on only parts of a character string, using 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.
- For example: Retrieve all employees whose address is in Houston, Texas

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

Substring Pattern Matching (2/2)

- Another example: To retrieve all employees who were born during the 1970s, ‘7’ must be the third character of the string (according to our format for date), so we use the value ‘_ _ 7 _ _ _ _ _ _ _ _’, with each underscore serving as a placeholder for an arbitrary character.

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE Bdate LIKE '_ _ 7 _ _ _ _ _ _ _ _';
```

Ordering of Query Results (1/2)

- 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. The default order is in ascending order of values.
- For example, 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;
```

Ordering of Query Results (2/2)

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

```
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 DESC, E.Lname ASC, E.Fname ASC;
```

Tables as Sets in SQL (1/5)

- 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.
- If we do want to eliminate duplicate tuples from the result of an SQL query, we use the keyword **DISTINCT** in the SELECT clause, meaning that only distinct tuples should remain in the result.
- A query with SELECT DISTINCT eliminates duplicates, whereas a query with SELECT ALL does not.
- Specifying SELECT with neither ALL nor DISTINCT is equivalent to SELECT ALL.

Tables as Sets in SQL (2/5)

- For example, retrieve the salary of every employee; if several employees have the same salary, that salary value will appear as many times in the result of the query:

```
SELECT ALL Salary
FROM EMPLOYEE;
```

- If we are interested only in distinct salary values, we want each value to appear only once, regardless of how many employees earn that salary. By using the keyword DISTINCT:

```
SELECT DISTINCT Salary
FROM EMPLOYEE;
```

Result:

Salary
30000
40000
25000
43000
38000
25000
25000
55000

Result:

Salary
30000
40000
25000
43000
38000
55000

Tables as Sets in SQL (3/5)

- SQL has directly incorporated some of the set operations from mathematical set theory. There are set union (UNION), set difference (EXCEPT), and set intersection (INTERSECT) operations
- The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminated from the result. These set operations apply only to type-compatible relations, so we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations.

Tables as Sets in SQL (4/5)

- For example, make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

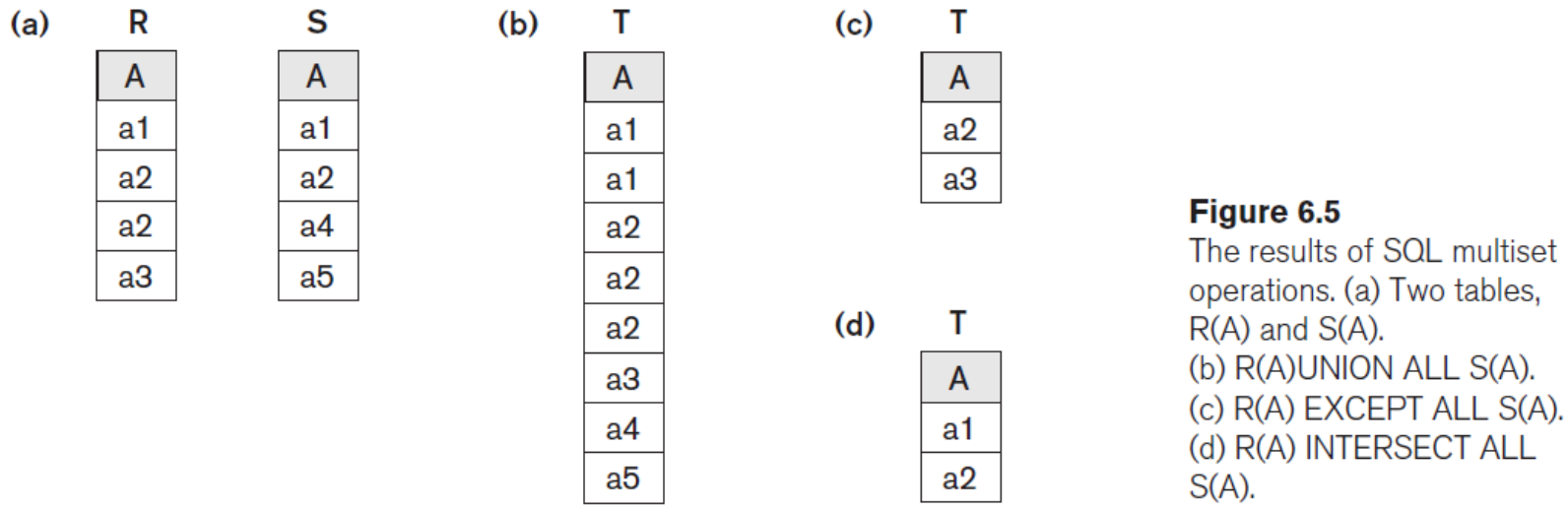
```
(SELECT DISTINCT Pnumber
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE Dnum=Dnumber AND Mgr_ssn=Ssn AND Lname='Smith')
UNION
(SELECT DISTINCT Pnumber
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE Pnumber=Pno AND Essn=Ssn AND Lname='Smith');
```

Essn = ssn OR MRg_ssn = ssn

- The first SELECT query retrieves the projects that involve a 'Smith' as manager of the department that controls the project, and the second retrieves the projects that involve a 'Smith' as a worker on the project. Applying the UNION operation to the two SELECT queries gives the desired result.

Tables as Sets in SQL (5/5)

- SQL also has corresponding multiset operations, which are followed by the keyword ALL (UNION ALL, EXCEPT ALL, INTERSECT ALL). Their results are multisets (duplicates are not eliminated).



(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 6.5
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).

INSERT Statement (1/4)

- 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.
- For example, to add a new tuple to the EMPLOYEE relation:

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

INSERT Statement (2/4)

- 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.
- This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple. However, the values must include all attributes with NOT NULL specification and no default value. Attributes with NULL allowed or DEFAULT values are the ones that can be left out.
- For example, to enter a tuple for a new EMPLOYEE for whom we know only the Fname, Lname, Dno, and Ssn attributes:

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

- Attributes not specified in this command are set to their DEFAULT or to NULL.

INSERT Statement (3/4)

- A DBMS that fully implements SQL should support and enforce all the integrity constraints that can be specified in the DDL.
- For example, if we issue the command on the database shown in, the DBMS should reject the operation because no DEPARTMENT tuple exists in the database with Dnumber = 2.

```
INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno)
VALUES ('Robert', 'Hatcher', '980760540', 2);
```

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

INSERT Statement (4/4)

- Similarly, the following insert command would be rejected because no Ssn value is provided and it is the primary key, which cannot be NULL.

```
INSERT INTO EMPLOYEE (Fname, Lname, Dno)
VALUES ('Robert', 'Hatcher', 5);
```

DELETE Statement (1/2)

- The DELETE command removes tuples from a relation. It includes a WHERE clause to select the tuples to be deleted.
- Tuples are explicitly deleted from only one table at a time. However, the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential integrity constraints of the DDL.
- A missing WHERE clause specifies that all tuples in the relation are to be deleted; however, the table remains in the database as an empty table.

DELETE Statement (2/2)

- For examples, delete zero tuple from the EMPLOYEE relation:

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

- Delete one tuple from EMPLOYEE:

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

- Delete four tuples from EMPLOYEE:

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

- Delete all tuples from EMPLOYEE:

```
DELETE FROM EMPLOYEE;
```

EMPLOYEE

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

UPDATE Statement (1/2)

- The UPDATE command is used to modify attribute values of one or more selected tuples.
- The WHERE clause in the UPDATE command selects the tuples to be modified from a single relation. However, updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL.
- An additional SET clause in the UPDATE command specifies the attributes to be modified and their new values.
- Each UPDATE command explicitly refers to a single relation only. To modify multiple relations, we must issue several UPDATE commands.

UPDATE Statement (2/2)

- For example, to 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;
```