DBMS

UNIT -3

PEARSON
Addison
Wesley

Chapter 8

SQL-99: Schema Definition, Basic Constraints, and Queries

PEARSON
Addison
Wesley

SQL

- **Definition of SQL** : is a standard language used for accessing and manipulating databases. originally SQL was called SEQUEL stands for Structured English QUEry Language was designed & implemented at IBM as an interface for RDBMS called SYSTEM R.

- **What is SQL?**
  - SQL stands for Structured Query Language
  - SQL lets you access and manipulate databases
  - SQL is an ANSI (American National Standards Institute) standard.

    also there different versions of SQL language : SQL-86/SQL-1, SQL-92/SQL-2, SQL-99/SQL-3

- **What Can SQL do?**
  - SQL can **execute queries** against a database
  - SQL can **retrieve data** from a database
  - SQL can **insert records** in a database
  - SQL can **update records** in a database
  - SQL can **delete records** from a database
  - SQL can **create new databases**
  - SQL can **create new tables** in a database
  - SQL can **create stored procedures** in a database
  - SQL can **create views** in a database
  - SQL can **set permissions** on tables, procedures, and views

SQL

- It provides **high level declarative language** interface so that user only specifies what the result is rather than how to execute a query.
- It is **comprehensive** **database language** : It has statements for data definitions, queries and updates.

  - hence, it is both **DDL** and a **DML**.

- **SQL Data Definitions & Data Types**

  - SQL uses the terms **table, row & column** for the formal relational model terms **relation, tuple & attribute** respectively.

  - The main SQL command for data definition is the **CREATE** statement, which can be used to **create schemas, relations (tables) & domains**.

Creation or Definition of Schema :

- An **SQL schema** is identified by the **schema name** & includes an **authorization identifier** to indicate the user who owns the schema as well as **descriptors** for each element in the schema.
- Schema elements includes tables, constraints, views, domains & other constructs such as authorization grants that describe the schema.
- **Example:** statement to create a schema called COMPANY, owned by the user with the authorization identifier ‘abcz’

CREATE SCHEMA COMPANY **AUTHORIZATION** abcz

DDL Commands

SQL command

- CREATE:
- DROP
- ALTER

Used to CREATE, DROP, and ALTER the descriptions of the tables (relations) of a database

CREATE TABLE

Creation or Definition of TABLE :

- CREATE TABLE command is used to 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))

- The attributes are specified first, each attribute is given a name, a data type to specify its domain of values and any attribute constraint such as NOT NULL may be specified on an attribute.

- The key, entity integrity and referential integrity constraints can also be specified after declaring the attributes or can be added later using ALTER TABLE command.

```
CREATE TABLE   DEPARTMENT
     (   DNAME                    VARCHAR(10)   NOT NULL,
              DNUMBER             INTEGER       NOT NULL,
              MGRSSN              CHAR(9),
              MGRSTARTDATE   CHAR(9)    );
```

CREATE TABLE

- **Base Tables (or Base Relations):**

  - is a relations created through CREATE TABLE statements.

  - Tuples are created & stored as file by the DBMS.

  - Tuples are considered to be ordered in the sequence in which they are specified in the CREATE TABLE statement.

- **Virtual Tables(or Virtual Relations):**

  - is a relations created through CREATE VIEW statements.

  - Tuples are created & not stored as physical file by the DBMS.

  - Tuples are not considered to be ordered in the sequence in which they are specified in the CREATE TABLE statement.

Domain Types in SQL

Numeric :

- **Int or Integer** (a finite subset of the integers that is machine-dependent).
- **Smallint.** Small integer (a machine-dependent subset of the integer domain type).
- **Real, double precision.** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **Float(n).** Floating point number, with user-specified precision of at least *n* digits.
- **Numeric(p,d)/Decimal(p,d) or DEC(p,d) : To declare formated numbers**

  - Fixed point number, with user-specified **precision** of *p* digits,

  d- **total number** of decimal digits.

Character-String:

- **char(n).** Fixed length character string, with user-specified length *n.*
- **varchar(n).** Variable length character strings, with user-specified maximum length *n.*

Bit-String:

- **Bit(n) : fixed length n**
- **Bit Varying(n): varying length n, n- max no of bits**
- **BLOB- binary large objects for images**
- **Representation of Bit string constant: B’1010’**

Boolean: Three valued Logic i.e True, False and Unknown , since NULL is used.

Additional Data Types in SQL2 and SQL-99

Has DATE, TIME, and TIMESTAMP data types

- **DATE:**
  - Made up of year-month-day in the format yyyy-mm-dd
- **TIME:**
  - Made up of hour : minute : second in the format hh:mm:ss
- **TIME(i):**
  - Made up of hour:minute:second plus 1 position for additional seperator and additional i digits specifying fractions of a second \
  - **TIMESTAMP:**Has both DATE and TIME components
- **Date:** Dates, containing a (4 digit) year, month and date
  - E.g. **date** ‘2001-7-27’
- **time.** Time of day, in hours, minutes and seconds.
  - E.g. **time** ’09:00:30’ **time** ’09:00:30.75’
- **timestamp**: date plus time of day
  - E.g. **timestamp** ‘2001-7-27 09:00:30.75’

Specifying Constraints in SQL

- Specifying Attribute constraints & Attribute Defaults

  - Allows NULLs as attribute values, constraint NOT NULL can be specified if NULL is not permitted for a particular attribute.

    implicitly – primary key

  - It can be specified for any attributes whose vales are required not to be NULL

  - Example : Consider the DEPARTMENT Table

```
DNAME       VARCHAR(10)     NOT NULL,
DNUMBER     INTEGER         NOT NULL
```

- To define a default value for an attribute by appending the clause **DEFAULT < value >** to an attribute.

  - if no default value is specified, the default value will be NULL for attribute that do not have a NOT NULL constraint.

  - Example : Consider the DEPARTMENT Table

```
Mgr_SSN    CHAR(9)    NOT NULL   DEFAULT ‘123670121’
```

Specifying Constraints in SQL

- Another type of constraint to restrict the attribute or domain values is using CHECK clause following an attribute or domain definition

- Example : Department numbers restricted to integer numbers b/w 1 and 20 then the declaration of DNUMBER attribute is as follows:

  DNUMBER INT NOT NULL CHECK(DNUMBER>0 and DNUMBER<20);

- The CHECK clause can be used along with CREATE DOMAIN statement

  CREATE DOMAIN D_NUM **AS** INTEGER **CHECK**(D_NUM>0 AND D_NUM<20);

  CREATE TABLE DEPT(…, DNUM D_NUM…,…);

  DNUM – attribute

  D_NUM – as attribute type

Specifying Key & Referential Integrity Constraints

- Special clauses within CREATE TABLE statement are used to specify these constraints.
- **PRIMARY KEY Clause :** specify one or more attribute as primary key of a relation.

  Dnumber INT PRIMARY KEY;

- **UNIQUE Clause :** specify alternate(secondary) keys

  Dname VARCHAR(9) NOT NULL UNIQUE;

- **Specifying Constraints on Tuples Using Check:**

  - Tuple based constraints

    CHECK(DEPTSTARTDATE<=MGRSTARTDATE)

Specifying Key & Referential Integrity Constraints

- **Referential Integrity is specified using FOREIGN KEY clause.**
- **A referential Integrity constraint can be violated when tuples are inserted/deleted/ when a FK or PK attribute value is modified.**
- **Default Action – Reject the operation**
- The schema designer can specify an alternative action to be taken if a referential integrity constraint is violated by using **Referential Triggered Action Clause to any FK constraint.**
- **The options : SET NULL, CASCADE & SET DEFAULT**
- **SET NULL/ SET DEFAULT on DELETE: to delete all the**

Specifying Key & Referential Integrity Constraints

- Example : SET NULL – DELETE

  CASCADE – UPDATE for FK Super_ssn of EMPLOYEE.

  - If tuples for supervising employee is deleted , the value of the SUPER_Ssn is automatically set to NULL for all employee tuples that were referencing the deleted employee tuple.

  - If the SSN value for a supervising employee is updated , the new value is cascaded to super_ssn for all employee tuples referencing the updated tuple

-
  ```
  CREATE TABLE   DEPT
  (       DNAME     VARCHAR(10)           NOT NULL,
          DNUMBER                 INTEGER NOT NULL,
          MGRSSN   CHAR(9),
          MGRSTARTDATE       CHAR(9),
          PRIMARY KEY (DNUMBER),
          UNIQUE (DNAME),
          FOREIGN KEY (MGRSSN) REFERENCES EMP
  ON DELETE SET DEFAULT ON UPDATE CASCADE  );
  ```

Specifying Key & Referential Integrity Constraints

```
CREATE TABLE   EMP
     (           ENAME            VARCHAR(30)  NOT NULL,
                 ESSN    CHAR(9),
                 BDATE   DATE,
                 DNO     INTEGER  DEFAULT 1,
                 SUPERSSN         CHAR(9),
                 PRIMARY KEY (ESSN),
                 FOREIGN KEY (DNO) REFERENCES DEPT
              ON DELETE SET DEFAULT ON UPDATE CASCADE,
                 FOREIGN KEY (SUPERSSN) REFERENCES EMP
              ON DELETE SET NULL ON UPDATE CASCADE  );
```

CREATE TABLE

- In SQL2, can use the CREATE TABLE command for specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the PRIMARY KEY and UNIQUE phrases

```
CREATE TABLE   DEPT
 (   DNAME    VARCHAR(10)          NOT NULL,
      DNUMBER          INTEGER        NOT NULL
      CHECK(DNUMBER>0 and DNUMBER<20),
      DEPTSTARTDATE CHAR(9),
     MGRSSN              CHAR(9),
     MGRSTARTDATE           CHAR(9),
     PRIMARY KEY (DNUMBER),
     UNIQUE (DNAME),
     FOREIGN KEY (MGRSSN) REFERENCES EMP,
     CHECK(DEPTSTARTDATE<=MGRSTARTDATE)
   );
```

DROP TABLE

- DROP command can be used to:

  - remove a schema and its elements : tables, domains or constraints.
  - remove a base relation within a schema which is no longer needed, the relation and its definitions. (means deletes all the records in the table and also deletes the table definition from the catlog.)

- There are two drop behavior options :

  - CASCADE : remove db schema & all its tables, domains & other elements.
  - RESTRICT : remove db schema only if it has no elements in it.

- If a whole schema is no longer needed, the DROP SCHEMA command is used as follows:

Example:

DROP SCHEMA COMPANY CASCADE/RESTRICT;

DROP TABLE

- If a a base relation within a schema which is no longer needed, the relation and its definitions can be removed by using the DROP TABLE command as follows:
- Relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

  DROP TABLE DEPENDENT; // cascade

  DROP TABLE DEPENDENT CASCADE/RESTRICT;

- CASCADE: all the constraints and views that references the table are automatically removed(dropped) from schema.
- RESTRICT: table is dropped only if it is not referenced in any constraints(Referential integrity constraints:foreign key)

ALTER TABLE

- Used to
  - add/delete an attribute(column),
  - add/delete table constraint (constraints to/from one of the base relations.)
  - changing the column definition.
- The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is *not allowed* for such an attribute
- Syntax:

  _ALTER TABLE table_name ADD column_name column-definition;

Example:

ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);

- The database users must still enter a value for the new attribute JOB for each EMPLOYEE tuple. This can be done using the UPDATE command.

ALTER TABLE…

- ALTER TABLE table_name ADD (column_1 column-definition, column_2 column-definition, ... column_n column_definition);
- ALTER TABLE table_name MODIFY column_name column_type;
- ALTER TABLE table_name MODIFY (column_1 column_type, column_2 column_type, ... column_n column_type);
- ALTER TABLE table_name DROP COLUMN column_name;
- ALTER TABLE table_name RENAME COLUMN old_name to new_name;
- ALTER TABLE table_name RENAME TO new_table_name;
- ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

ALTER TABLE…

- ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
- ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
- ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
- ALTER TABLE table_name DROP CONSTRAINTMyUniqueConstraint;
- ALTER TABLE table_name DROP PRIMARY KEY;
- ALTER TABLE table_name ALTER COLUMN column_name DROP NOT NULL;
- ALTER TABLE table_name DROP COLUMN column_name CASCADE/RESTRICT;
- ALTER TABLE table_name DROP PRIMARY KEY CASCADE/RESTRICT;

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database : the SELECT statement
- This is *not the same as* the SELECT operation of the relational algebra

- Important distinction between SQL and the formal relational model :

  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values. Hence, an SQL (table) is a *multi-set* (sometimes called a bag) of tuples; it *is not* a set of tuples

  - where as formal relational model doesn’t allow a table to have two identical tuples.Hence, it is 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

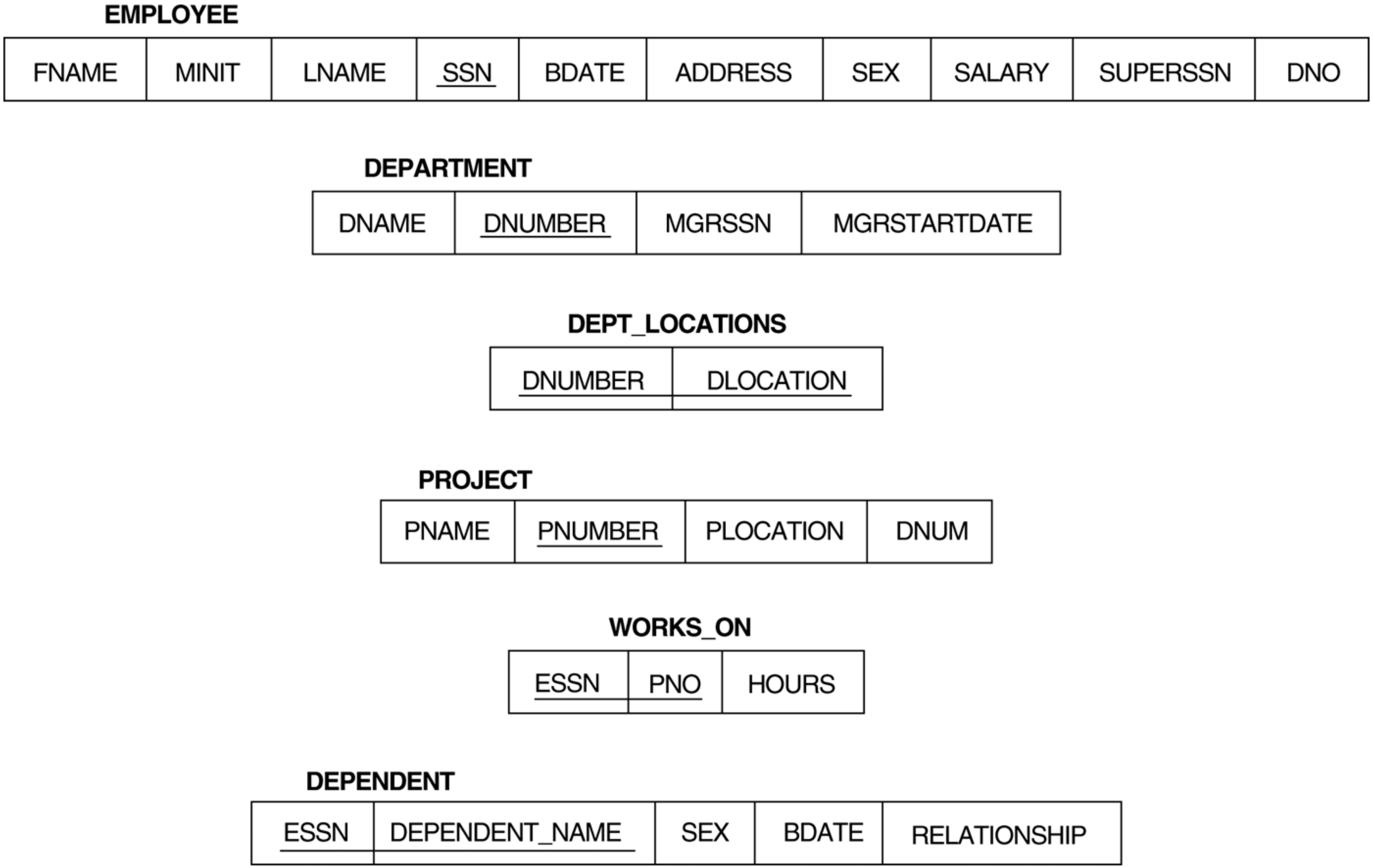
Retrieval Queries in SQL

- Basic form of the SQL SELECT statement is called a *mapping* or a *SELECT-FROM-WHERE block is formed of three clauses :*

```
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

Relational Database Schema--Figure 5.5



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

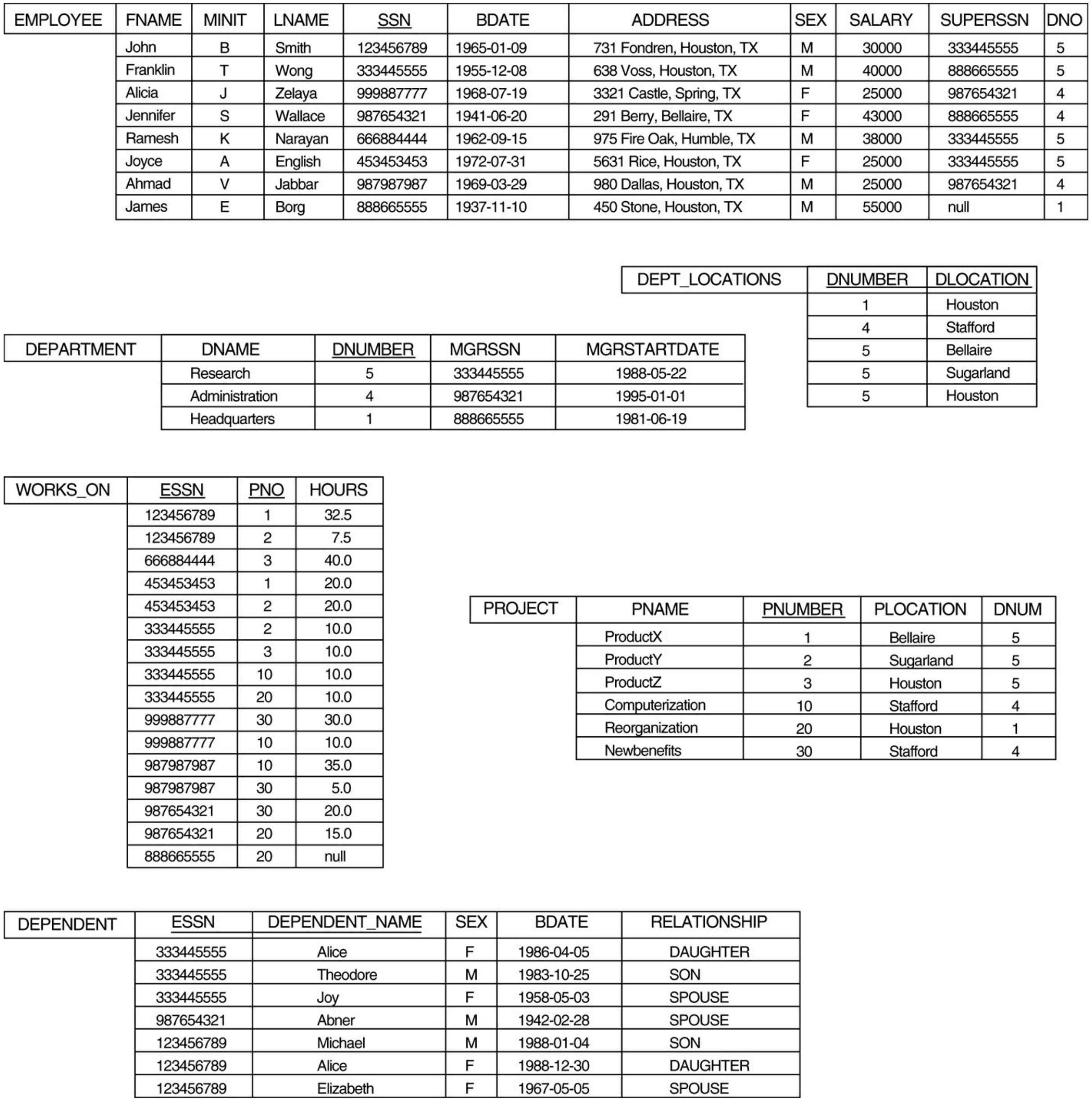
WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Populated Database--Fig.5.6



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	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	1968-07-19	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

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

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

Simple SQL Queries

- Basic SQL queries correspond to using the SELECT, PROJECT, and JOIN operations of the relational algebra
- All subsequent examples use the COMPANY database
- Example of a simple query on *one* relation
- Query 0: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

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

- SQL query with a single relation name in the FROM clause is similar to a SELECT-PROJECT pair of relational algebra operations;
- the SELECT-clause specifies the *projection attributes* and
- the WHERE-clause specifies the *selection condition*
- However, the result of the query *may contain* duplicate tuples – main difference

Simple SQL Queries (cont.)

- <u>Query 1:</u> 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
  ;
```

  - Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a *selection condition* (corresponds to a SELECT operation in relational algebra)
  - (DNUMBER=DNO) is a *join condition* (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries (cont.)

- 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 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 to its controlling department
  - The join condition MGRSSN=SSN relates the controlling department to the employee who manages that department

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
  A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name

Example: EMPLOYEE.LNAME, DEPARTMENT.DNAME

Suppose Dno & Lname --- Dnumber & Name --- Employee Relation

Dname ---Name ----Department Relation then the query

Retrieve the name and address of all employees who work for the 'Research' department would be repharased as

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

ALIASES

- Some queries need to refer to the same relation twice
- In this case, *aliases* are given to the relation name
- 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 E, EMPLOYEE S
    WHERE       E.SUPERSSN=S.SSN;
```

  - In Q8, the alternate relation names E and S are called *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*

ALIASES (cont.)

- Aliasing can also be used in any SQL query for convenience
  Can also use the AS keyword to specify aliases

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

ALIASING ATTRIBUTES

EMPLOYEE AS E(FN,LN,MN,SAL…….)

UNSPECIFIED WHERE-clause

- 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
- Query 9: 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 join condition, then the *CARTESIAN PRODUCT* of tuples is selected

UNSPECIFIED WHERE-clause (cont.)

- Example:

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

  - It is extremely important not to overlook specifying any selection and join conditions in the WHERE-clause; otherwise, incorrect and very large relations may result

USE OF *

- To retrieve all the attribute values of the selected tuples, a * 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;
```

USE OF DISTINCT

- SQL does not treat a relation as a set; *duplicate tuples can appear*
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used
- For example, the result of Q11 may have duplicate SALARY values whereas Q11A does not have any duplicate values

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

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (**UNION**), and in *some versions* of SQL there are set difference (**MINUS/EXCEPT**) and intersection (**INTERSECT**) operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations* ; the two relations must have the same attributes and the attributes must appear in the same order

SET OPERATIONS (cont.)

- Query 4: Make a list of all projects for projects that involve an employee whose last name is 'Smith' as a worker 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');
```

UNION ALL,EXCEPT ALL,INTERSECT ALL

SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used: '%' replaces an arbitrary number of characters, and '_' replaces a single arbitrary character

SUBSTRING COMPARISON (cont.)

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

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

SUBSTRING COMPARISON (cont.)

- Query 26: Retrieve all employees who were born during the 1950s. Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '_______5_', with each underscore as a place holder for a single arbitrary character.
  Ex: 08-NOV-52

```
Q26:    SELECT    FNAME, LNAME
        FROM      EMPLOYEE
        WHERE     BDATE LIKE       '_______5_’;
```

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

ARITHMETIC OPERATIONS

- The standard arithmetic operators '+', '-'. '*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result
- 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';
```

ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- 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

- ORDER BY DNAME DESC

NESTING OF QUERIES

- A complete SELECT query, called a *nested query* , can be specified within the WHERE-clause of another query, called the *outer query*
- Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:  SELECT      FNAME, LNAME, ADDRESS
     FROM        EMPLOYEE
     WHERE       DNO IN  (SELECT  DNUMBER
                          FROM    DEPARTMENT
                          WHERE   DNAME='Research' );
```

NESTING OF QUERIES (cont.)

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its Dno value is in the result of either nested query
- The comparison operator **IN** compares a value v with a set (or multi-set) of values V, and evaluates to **TRUE** if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be *correlated*
- The result of a correlated nested query is *different for each tuple (or combination of tuples) of the relation(s) the outer query*
- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12: SELECT   E.FNAME, E.LNAME
     FROM     EMPLOYEE AS E
     WHERE    E.SSN IN ( SELECT   ESSN
                         FROM     DEPENDENT
                         WHERE    ESSN=E.SSN AND
                               E.FNAME=DEPENDENT_NAME)
```

CORRELATED NESTED QUERIES (cont.)

- In Q12, the nested query has a different result *for each tuple* in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query. For example, Q12 may be written as in Q12A

```
Q12A:   SELECT      E.FNAME, E.LNAME
        FROM        EMPLOYEE E, DEPENDENT D
        WHERE       E.SSN=D.ESSN AND
                    E.FNAME=D.DEPENDENT_NAME
```

- The original SQL as specified for SYSTEM R also had a **CONTAINS** comparison operator, which is used in conjunction with nested correlated queries
- This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently

CORRELATED NESTED QUERIES (cont.)

- Most implementations of SQL *do not* have this operator
- The CONTAINS operator compares two *sets of values* , and returns TRUE if one set contains all values in the other set
  (reminiscent of the *division* operation of algebra).
  - Query 3: Retrieve the name of each employee who works on *all* the projects controlled by department number 5.

```
Q3:    SELECT  FNAME, LNAME
       FROM    EMPLOYEE
       WHERE  ( (SELECT          PNO
                 FROM   WORKS_ON
                 WHERE           SSN=ESSN)
                 CONTAINS
                (SELECT          PNUMBER
                 FROM   PROJECT
                 WHERE           DNUM=5) )
```

CORRELATED NESTED QUERIES (cont.)

- In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5
- The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different *for each employee tuple* because of the correlation

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
- We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below

THE EXISTS FUNCTION (cont.)

- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
Q12B:   SELECT   FNAME, LNAME
        FROM     EMPLOYEE
        WHERE    EXISTS    (SELECT   *
                 FROM      DEPENDENT
                 WHERE     SSN=ESSN AND
                 FNAME=DEPENDENT_NAME)
```

THE EXISTS FUNCTION (cont.)

- Query 6: Retrieve the names of employees who have no dependents.

```
Q6:     SELECT    FNAME, LNAME
        FROM      EMPLOYEE
        WHERE     NOT EXISTS  (SELECT    *
                  FROM  DEPENDENT
                  WHERE SSN=ESSN)
```

  - In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist* , the EMPLOYEE tuple is selected
  - EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
Q13:    SELECT    DISTINCT ESSN
        FROM      WORKS_ON
        WHERE     PNO IN  (1, 2, 3)
```

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is NULL (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .
- Query 14: Retrieve the names of all employees who do not have supervisors.

```
Q14:     SELECT    FNAME, LNAME
         FROM      EMPLOYEE
         WHERE     SUPERSSN  IS  NULL
```

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

Joined Relations Feature in SQL2

- Can specify a "joined relation" in the FROM-clause
- Looks like any other relation but is the result of a join
- Allows the user to specify different types of joins (regular "theta" JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Joined Relations Feature in SQL2 (cont.)

- Examples:

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

can be written as:

```
Q8:  SELECT      E.FNAME, E.LNAME, S.FNAME, S.LNAME
     FROM  (EMPLOYEE E LEFT OUTER JOIN EMPLOYEES
           ON  E.SUPERSSN=S.SSN)

Q1:  SELECT      FNAME, LNAME, ADDRESS
     FROM  EMPLOYEE, DEPARTMENT
     WHERE       DNAME='Research' AND DNUMBER=DNO
```

Joined Relations Feature in SQL2 (cont.)

- could be written as:

```
Q1:  SELECT        FNAME, LNAME, ADDRESS
     FROM  (EMPLOYEE JOIN DEPARTMENT
                ON DNUMBER=DNO)
     WHERE         DNAME='Research’
```

or as:

```
Q1:  SELECT        FNAME, LNAME, ADDRESS
     FROM  (EMPLOYEE NATURAL JOIN DEPARTMENT
                AS DEPT(DNAME, DNO, MSSN, MSDATE)
     WHERE         DNAME='Research’
```

Joined Relations Feature in SQL2 (cont.)

- Another Example;
  - Q2 could be written as follows; this illustrates multiple joins in the joined tables

```
Q2:     SELECT      PNUMBER, DNUM,
LNAME,                          BDATE,
ADDRESS
        FROM        (PROJECT JOIN
                    DEPARTMENT ON
                    DNUM=DNUMBER) JOIN
                    EMPLOYEE ON
                    MGRSSN=SSN) )
        WHERE        PLOCATION='Stafford’
```

AGGREGATE FUNCTIONS

- Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**
- Query 15: Find the maximum salary, the minimum salary, and the average salary among all employees.

```
Q15:    SELECT    MAX(SALARY),
                  MIN(SALARY), AVG(SALARY)
        FROM      EMPLOYEE
```

– Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTIONS (cont.)

- Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16: SELECT     MAX(SALARY), MIN(SALARY),
                AVG(SALARY)
       FROM     EMPLOYEE, DEPARTMENT
       WHERE    DNO=DNUMBER AND
                DNAME='Research'
```

AGGREGATE FUNCTIONS (cont.)

- Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:    SELECT   COUNT (*)
        FROM     EMPLOYEE

Q18:    SELECT   COUNT (*)
        FROM     EMPLOYEE,
                 DEPARTMENT
        WHERE    DNO=DNUMBER AND
                 DNAME='Research’
```

GROUPING

- In many cases, we want to apply the aggregate functions *to subgroups of tuples in a relation*
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

GROUPING (cont.)

- Query 20: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q20: SELECT      DNO, COUNT (*), AVG (SALARY)
     FROM  EMPLOYEE
     GROUP BY    DNO
```

  - In Q20, the EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO
  - The COUNT and AVG functions are applied to each such group of tuples separately
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
  - A join condition can be used in conjunction with grouping

GROUPING (cont.)

- Query 21: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q21:    SELECT      PNUMBER, PNAME, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       PNUMBER=PNO
        GROUP BY    PNUMBER, PNAME
```

  – In this case, the grouping and functions are applied *after* the joining of the two relations

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples)

THE HAVING-CLAUSE (cont.)

- Query 22: For each project *on which more than two employees work* , retrieve the project number, project name, and the number of employees who work on that project.

```
Q22:      SELECT      PNUMBER, PNAME, COUNT
                      (*)
          FROM        PROJECT, WORKS_ON
          WHERE       PNUMBER=PNO
          GROUP BY    PNUMBER, PNAME
          HAVING      COUNT (*) > 2
```

Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:

```
SELECT    <attribute list>
FROM      <table list>
[WHERE    <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING   <group condition>]
[ORDER BY <attribute list>]
```

Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

Specifying Updates in SQL

- There are three SQL commands to modify the database; INSERT, DELETE, and UPDATE

INSERT

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

INSERT (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 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 (cont.)

- Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database
- Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation

INSERT (cont.)

– Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department. A table DEPTS_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A:     CREATE TABLE  DEPTS_INFO
                   (DEPT_NAME     VARCHAR(10),
                    NO_OF_EMPS    INTEGER,
                    TOTAL_SAL     INTEGER);

U3B:     INSERT INTO    DEPTS_INFO (DEPT_NAME,
                        NO_OF_EMPS, TOTAL_SAL)
         SELECT         DNAME, COUNT (*), SUM (SALARY)
         FROM           DEPARTMENT, EMPLOYEE
         WHERE          DNUMBER=DNO
         GROUP BY       DNAME ;
```

INSERT (cont.)

- Note: The DEPTS_INFO table may not be up-to-date if we change the tuples in either the DEPARTMENT or the EMPLOYEE relations *after* issuing U3B. We have to create a view (see later) to keep such a table up to date.

DELETE

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

DELETE (cont.)

- Examples:

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

U4B:      DELETE FROM      EMPLOYEE
          WHERE            SSN='123456789’

U4C:      DELETE FROM      EMPLOYEE
          WHERE            DNO  IN
                (SELECT    DNUMBER
                FROM       DEPARTMENT
                WHERE      DNAME='Research')

U4D:      DELETE FROM      EMPLOYEE
```

UPDATE

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

UPDATE (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
    WHERE       PNUMBER=10
```

UPDATE (cont.)

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6:  UPDATE        EMPLOYEE
     SET           SALARY = SALARY *1.1
     WHERE         DNO  IN (SELECT       DNUMBER
                      FROM          DEPARTMENT
                      WHERE         DNAME='Research')
```

- 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