SQL -
A Relational Database Language

Data Definition

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

CREATE TABLE:

- 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))
- A constraint NOT NULL may be specified on an attribute

Example

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

Data Definition (Cont.)

- One important constraint missing from the CREATE TABLE command is that of specifying the primary key attributes, secondary keys, and referential integrity constraints (foreign keys).
- Key attributes can be specified via the CREATE UNIQUE INDEX command.
- More recent SQL systems can specify primary keys and referential integrity constraints.

Data Definition (Cont.)

For keys and foreign keys include following within cerate table statement.

Primary Key (DNumber),
Unique (DNAME),
Foreign Key (MGRSSN) References DEPARTMENT (DNUMBER)

In SQL 92- On update or On delete | set null ; cascade; set default

MGRSSN CHAR(9) NOT NULL DEFAULT `88866555'
CONSTRAINT DEPTMGRFK
Foreign key (MGRSSN) references Employee (SSN)
On delete set default ON UPDATE cascade;

(if employee MGRSSN is deleted – a default manager is assigned; an UPDATE of SSN of MGRSSN is cascaded to all tuples of DEPT having the same manager)

Data Definition (Cont.)

DROP TABLE:

- Used to remove a relation (base table) and its definition
- The relation can no longer be used in queries, updates, or any other command since its description no longer exists

Example: DROP TABLE DEPARTMENT;

Data Definition (Cont.)

ALTER TABLE

- Used to add or drop (with cascade) an attribute to one of the base relations
- The newly added attribute will have NULLs (or default values) 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

Example:

ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);

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

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 (σ) 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 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 using the CREATE UNIQUE INDEX command, or by using the DISTINCT option.

Retrieval Queries in SQL

Basic form of the SQL SELECT statement is called a mapping or a SELECT-FROM-WHERE block

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. (*) to get all attributes.
- <table list> is a list of relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query. Omitted to retrieve all tuples.

Basic 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

Retrieve the birthdate and address of the employee whose name is `John Smith';

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

Similar to a (σ-Π) pair of relational algebra operations; the SELECT-clause specifies the Π attributes and the WHERE-clause specifies the σ condition

However, the result of the query may contain duplicate tuples

Basic SQL Queries (Cont.)

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;
```

Similar to a (σ-Π-$\bowtie$) sequence of relational algebra operations (DNAME = 'Research') is a selection condition (corresponds to a σ operation in relational algebra).

(DNUMBER=DNO) is a JOIN condition (corresponds to a $\bowtie$ operation in relational algebra)

Aliases

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 or DEPARTMENT.DNAME

Some queries need to refer to the same relation twice

In this case, aliases are given to the relation name

Aliases (Cont.)

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

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

In the above query, the alternate relation names E and S are called aliases for the EMPLOYEE relation

We can think of E and S as two different copies of the EMPLOYEE relation; E represents employees in the role of supervises and S represents employees in the role of supervisor

Aliasing can also be used in any SQL query for convenience

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.

Example: Retrieve the SSN values for all employees.

```
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. For example:

```
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

```
SELECT *
FROM   EMPLOYEE;
```

```
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, the keyword DISTINCT is used

For example, the result may have duplicate SALARY values whereas the second below does not have any duplicate values

```
SELECT      SALARY
FROM        EMPLOYEE;

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) 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: Example

Make a list of all project names 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.

```
(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');
```

QUIZ

Retrieve name of employee who is a manager of a department that controls a project, in which the employee has worked on for more than 40 hours.

```
SELECT    FNAME, LNAME
FROM      EMPLOYEE, DEPARTMENT, PROJECT, WORKS_ON
WHERE     SSN = MGRSSN          AND
          DNUM = DNUMBER        AND
          PNUMBER = PNO         AND
          ESSN = SSN            AND
          HOURS > 40;
```

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

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

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

The nested query selects the number of the 'Research' department

The outer query selects an EMPLOYEE tuple if its DNO value is in the result of nested query.

Nesting of Queries (Cont.)

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

Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT      T.FNAME, T.LNAME
FROM        EMPLOYEE T
WHERE       T.SSN IN      ( SELECT ESSN
                            FROM   DEPENDENT
                            WHERE  ESSN=T.SSN AND
                            T.FNAME=DEPENDENT-NAME);
```

CORRELATED NESTED QUERIES (Cont.)

In the previous query, 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.

```
SELECT   E.FNAME, E.LNAME
FROM     EMPLOYEE E, DEPENDENT D
WHERE    E.SSN=D.ESSN AND
         E.FNAME=D.DEPENDENT-NAME;
```

Explicit Sets

It is also possible to use an explicit set of values in the WHERE-clause rather than a nested query

Retrieve the social security numbers of all employees who work on project number 1, 2, or 3

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

THE EXISTS FUNCTION

EXISTS used to check whether the result of a correlated nested query is empty (contains no tuples) or not.

Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT    T.FNAME, T.LNAME
FROM      EMPLOYEE T
WHERE     EXISTS ( SELECT   *
                   FROM     DEPENDENT
                   WHERE    T.SSN = ESSN AND
                            FNAME = DEPENDENT_NAME );
```

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

Retrieve the names of all employees who do not have supervisors.

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    SUPERSSN IS NULL;
```

Aggregate Functions and Grouping

Include COUNT, SUM, MAX, MIN, and AVG

Find the maximum salary, the minimum salary, and the average salary among all employees.

```
SELECT   MAX (SALARY), MIN (SALARY), AVG (SALARY)
FROM     EMPLOYEE;
```

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

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

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

Aggregate Functions and 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 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.

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.

Aggregate Functions and Grouping

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

```
SELECT      PNUMBER, PNAME, COUNT(*)
FROM        PROJECT, WORKS-ON
WHERE       PNUMBER = PNO
GROUP BY    PNUMBER, PNAME;
```

The grouping and functions are applied after 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).

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.

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKS-ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
HAVING      COUNT(*) > 2;
```

SUBSTRING COMPARISON

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.

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

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

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.

Show the effect of giving all employees who work on 'ProductX' project a 10% raise.

```
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 attributes. Default is ascending order. Use DESC to specify descending order after attribute name.

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

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

Summary of SQL Queries

A query in SQL can consist of up to six clauses, but only the first two are mandatory

```
SELECT       <ATTRIBUTE LIST>
FROM         <TABLE LIST>
[WHERE       <CONDITION>]
[GROUP BY    <GROUPING ATTRIBUTES>]
[HAVING      <GROUP CONDITION>]
[ORDER BY    <ATTRIBUTE LIST>]
```

A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

Data Manipulation in SQL

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

INSERT:

In its simplest form, it is used to add a single tuple to a relation

Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command

Example:

INSERT INTO EMPLOYEE VALUES
('Richard','K','Marini','653298653' '30-DEC-521, '98 Oak Forest,KatyTX', 'M',37000,'987654321', 4)

Insert (Cont.)

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

Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard','Marini','653298653')
```

Important Note: Only the constraints specified in the DDL commands are automatically enforced by the DBMS when updates are applied to the database

Insert (Cont.)

Another variation of INSERT allows insertion of multiple tuples in a relation in a single command

Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department.

U3A: CREATE DEPTS-INFO

```
(D-NAME          VARCHAR(10),
 NO-OF-EMPS      INTEGER,
 T-SAL           INTEGER);
INSERT INTO DEPTS-INFO (D-NAME, NO_OF-EMPS, T_SAL)
        SELECT DNAME, COUNT (*), SUM (SALARY)
        FROM DEPARTMENT, EMPLOYEE
        WHERE DNUMBER=DNO
        GROUP BY DNAME;
```

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.

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 of the relation that satisfy the WHERE-clause condition.

DELETE

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.

Change the location and controlling department number of project number IO to 'Bellaire' and 5, respectively.

```
UPDATE PROJECT
SET PLOCATION ='Bellaire', DNUM = 5
WHERE PNUMBER=10;
```

UPDATE

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

```
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.

Relational Views in SQL

A view is a single virtual table that is derived from other tables.

The other tables could be base tables or previously defined views.

A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.

There are no limitations on querying a view.

The CREATE VIEW command is used to specify a view by specifying a (virtual) table name and a defining query.

The view attribute names can be inherited from the attribute names of the tables in the defining query.

VIEWS (Cont.)

```
V1: CREATE VIEW WORKS-ON1
      AS SELECT FNAME, LNAME, PNAME, HOURS
           FROM EMPLOYEE, PROJECT, WORKS-ON
           WHERE SSN=ESSN AND PNO=PNUMBER;
V2: CREATE VIEW DEPT-INFO
      (DEPT-NAME, NO-OF-EMPS, TOTAL-SAL)
      AS SELECT DNAME, COUNT (*), SUM (SALARY)
           FROM DEPARTMENT, EMPLOYEE
           WHERE DNUMBER=DNO
           GROUP BY DNAME;
```

In V1 the names of the view attribute names are inherited

In V2. the view attribute names are listed using a one-to-one correspondence with the entries in the SELECT-clause of the defining query

QUERIES ON VIEWS

Retrieve the last name and first name of all employees who work on 'ProjectX'.

```
SELECT PNAME, FNAME, LNAME
FROM WORKS-ON1
WHERE PNAME='ProjectX';
```

Without the view WORKS-ON1, this query specification would require two join conditions.

A view can be defined to simplify frequently occurring queries.

The DBMS is responsible for keeping the view always up-to-date if the base tables on which the view is defined are modified.

Hence, the view is not realized at the time of view definition, but rather at the time we specify a query on the view.

A view is removed using the DROP VIEW command.

```
DROP VIEW WORKS-ON1;
```

Views can also be used as a security and authorization mechanism

UPDATING OF VIEWS

A view update operation may be mapped in multiple ways to update operations on the defining base relations

The topic of updating views is still an active research area

Suppose we update the WORKS-ON1 view by modifying the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'.

```
UPDATE WORKS-ON1
SET PNAME ='ProductY'
WHERE LNAME='Smith AND FNAME='John' AND
PNAME='ProdudX';
```

UPDATING OF VIEWS (Cont.)

This can be mapped into several updates on the base relations to give the desired update on the view. Two possibilities are:

1. Change the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'

It is quite unlikely that the user who specified the view update wants the update to be interpreted this way

(1): UPDATE PROJECT

SET PNAME ='ProductY'

WHERE PNAME ='ProductX';

UPDATING OF VIEWS (Cont.)

2. Relate 'John Smith' to the 'ProductY' PROJECT tuple in place of the 'ProductX' PROJECT tuple

This is most likely the update the user means

(2): UPDATE WORKS-ON
SET PNO = (SELECT PNUMBER FROM PROJECT
WHERE PNAME='ProductY')
WHERE ESSN = (SELECT SSN FROM EMPLOYEE
WHERE LNAME='Smith' AND FNAME='John')
AND PNO = (SELECT PNUMBER FROM PROJECT
WHERE PNAME='ProductX');

Some view updates may not make much sense; for example, modifying the TOTAL-SAL attribute of DEPT-INFO.

UV2: UPDATE DEPT-INFO
SET TOTAL-SAL=100000
WHERE DNAME='Research';

UPDATING OF VIEWS (Cont.)

In general, we cannot guarantee that any view can be updated.

A view update is unambiguous only if one update on the base relations can accomplish the desired update effect on the view.

If a view update can be mapped to more than one update on the underlying base relations, we must have a certain procedure to choose the desired update.

We can make the following general observations:

- A view with a single defining table is updatable if the view attributes contain the primary key.
- Views defined on multiple tables using joins are generally not updatable.
- Views defined aggregate functions are not updatable.

Creating Indexes in SQL

An SQL base relation generally corresponds to a stored file.

SQL has statements to create and drop indexes on base relations.

One or more indexing attributes are specified for each index.

The CREATE INDEX command is used to specify an index.

Each index is given an index name.

Example:

```
CREATE INDEX LNAME-INDEX ON EMPLOYEE (LNAME);
```

The index entries are in ascending (ASC) order of the indexing attributes; for descending order, the keyword DESC is added.

Creating Indexes in SQL (Cont.)

An index can be created on a combination of attributes.

Example:

```
CREATE INDEX NAMES-INDEX
 ON EMPLOYEE (LNAME ASC, FNAME DESC, MINIT);
```

Two options on indexes in SQL are UNIQUE and CLUSTER.

To specify the key constraint on the indexing attribute or combination of attributes, the keyword UNIQUE is used.

Example:

```
CREATE UNIQUE INDEX SSN-INDEX ON EMPLOYEE
(SSN);
```

This is best done before any tuples are inserted in the relation.

Creating Indexes in SQL (Cont.)

An attempt to create a unique index on an existing base table W fails if the current tuples in the table do not obey the constraint.

A second option on index creation is to specify that the index is a clustering index using the keyword CLUSTER.

A base relation can have at most one clustering index, but any number of non-clustering indexes.

Example: CREATE INDEX DNO-INDEX
ON EMPLOYEE (DNO) CLUSTER;

Each DBMS will have its own index implementation technique; in most cases, some variation of the B+-tree data structure is used.

To drop an index, we issue the DROP INDEX command.

The index name is needed to refer to the index when it is to be dropped.

DROP INDEX DNO-INDEX;

Embedding SQL in a Programming Language

SQL can also be used in conjunction with a general purpose programming language, such as PASCAL, COBOL, or PL/I

The programming language is called the host language

The embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor can extract the SQL statements

In PL/1 the keywords EXEC SQL precede any SQL statement

In some implementations, SQL statements are passed as parameters in procedure calls

Embedding SQL in a Programming Language

We will use PASCAL as the host programming language, and a "$" sign to identify SQL statements in the program

Within an embedded SQL command, we may refer to program variables, which are prefixed by a "%" sign

The programmer should declare program variables to match the data types of the database attributes that the program will process

These program variables may or may not have names that are identical to their corresponding attributes

Example: Write a program segment (loop) that reads a social security number and prints out some information from the corresponding EMPLOYEE tuple.

Embedding SQL in a Programming Language

```
while LOOP ='Y' do
  begin
  writeln('input social security number:');
  readln(SOC-SEC-NUM);
  $SELECT FNAME, MINIT, LNAME, SSN, BDATE,
                ADDRESS,   SALARY
    INTO %E.FNAME, %E.MINIT,, %E.LNAME, %E.SSN,
            %E.BDATE, %E.ADDRESS, %E.SALARY
    FROM EMPLOYEE
    WHERE SSN=%SOC-SEC-NUM;
   writeln(E.FNAME, E.MINIT, E.LNAME, E.SSN, E.BDATE,
              E.ADDRESS, E.SALARY);
  writeln('more social security numbers (Y or N)?  ');
  readln(LOOP)
 end;
```

Embedding SQL in a Programming Language

A single tuple is selected by the embedded SQL query; that is why we are able to assign its attribute values directly to program variables.

In general, an SQL query can retrieve many tuples.

The concept of a cursor is used to allow tuple-at-a-time processing by the PASCAL program.

CURSORS

We can think of a cursor as a pointer that points to a single tuple (row) from the result of a query.
The cursor is declared when the SQL query command is specified.
A subsequent OPEN cursor command fetches the query result and sets the cursor to a position before the first row in the result of the query; this becomes the current row for the cursor.
Subsequent FETCH commands in the program advance the cursor to the next row and copy its attribute values into PASCAL program variables specified in the FETCH command.
An implicit variable SQLCODE communicates to the program the status of SQL embedded commands.
An SQLCODE of 0 (zero) indicates successful execution.

CURSORS

Different codes are returned to indicate exceptions and errors
A special END-OF-CURSOR code is used to terminate a loop over the tuples in a query result
A CLOSE cursor command is issued to indicate that, we are done with the result of the query
When a cursor is defined for rows that are to be updated the clause FOR UPDATE OF must be in the cursor declaration, and a list of the names of any attributes that will be updated follows
The condition WHERE CURRENT OF cursor specifies that the current tuple is the one to be updated (or deleted)
Example: Write a program segment that reads (inputs) a department name, then lists the names of employees who work in that department, one at a time. The program reads a raise amount for each employee and updates the employee's salary by that amount.

Cursors (Cont.)

```
readln(DNAME);
 $SELECT DNUMBER INTO
   %DNUMBER
  FROM DEPARTMENT
  WHERE DNAME= % DNAME;
 $DECLARE EMP CURSOR FOR
       SELECT SSN, FNAME, MINIT,
    LNAME, SALARY
       FROM EMPLOYEE
       WHERE DNO=%DNUMBER
    FOR UPDATE OF SALARY;
  $OPEN EMP;
  $FETCH EMP INTO %E.SSN,
   %E.FNAME, %E.MINIT,
          %E.LNAME, %E.SAL;
```

```
while SQLCODE = 0 do
       begin
        writeln('employee name:', E.FNAME,
        E.MINIT, E.LNAME);
       writeln('enter raise amount)
       readln(RAISE);
       $UPDATE EMPLOYEE
        SET SALARY = SALARY + %RAISE
        WHERE CURRENT OF EMP;
       $FETCH EMP INTO %E.SSN,
        %E.FNAME, %E.MINIT,
        %E.LNAME, %E.SAL;
      end;
$CLOSE CURSOR EMP;
```

QUIZ

For each department having more than five employees, retrieve the department name and the number of employees making more than $40,000.

```
SELECT      DNAME, COUNT(*)
FROM        DEPARTMENT, EMPLOYEE
WHERE       DNO=DNUMBER AND SALARY > 40000 AND
            DNO IN (    SELECT DNO
                        FROM EMPLOYEE
                        GROUP BY DNO
                        HAVING COUNT(*) > 5 )
GROUP BY    DNAME;
```

QUIZ

For every student who secured grade ‘A’, retrieve the student name and the course name.

```
SELECT   Name, CourseName
FROM     STUDENT S, GRADE-REPORT G
         SECTION SC, COURSE C
WHERE    Grade = ‘A’ AND
         S.Student-Number = G.Student-Number AND
         G.Section-Identifier = SC.Section-Identifier AND
         SC.Course-Number = C.Course-Number;
```

There are three join conditions. To get the course information for a student one needs to join STUDENT with GRADE-REPORT, GRADE-REPORT with SECTION, and then SECTION with COURSE.

Example Database

STUDENT			
Name	Student Number	Class	Major
Smith	17	1	COSC
Brown	8	2	COSC

GRADE REPORT		
Student Number	Section-Identifier	Grade
17	85	A
18	102	B+

PREREQUISITE	
Course Number	Prerequisite Number
COSC3380	COSC3320
COSC3320	COSC1310

SECTION				
Section-Identifier	Course Number	Semester	Year	Instructor
85	MATH2410	Fall	91	King
92	COSC1310	Fall	91	Anderson
102	COSC3320	Spring	92	Knuth
135	COSC3380	Fall	92	Stone

COURSE			
Course Name	Course Number	Credit Hours	Department
Intro to CS	COSC1310	4	COSC
Data Structures	COSC3320	4	COSC
Discrete Mathematics	MATH2410	3	MATH
Data Base	COSC3380	3	COSC

COMPANY DATABASE