SQL Part II
More DML Features
NULLs

• NULL indicates the absence of data. It may mean inapplicable, or applicable but unspecified.

• The system can give NULL as the default value for unmentioned fields

• Find names for students whose major is unspecified: SELECT NAME FROM STUDENT WHERE (MAJOR IS NULL);

• Since NULL stands for value unspecified, all NULLs are considered to be different
Subqueries

- We can make selections that use the value of a relation computed as “an inner loop”
- \textsc{Student}(\textsc{Sname}, \textsc{Major}), \textsc{Enroll}(\textsc{Student}, \textsc{Course}, \textsc{Dept})
- Query: Find name of students who are not taking any courses in their major
- To solve, first suppose we are given values of \textsc{Sname}, \textsc{Major} for a specific student
• Which departments does the student takes classes in?
  \[ \text{depts}(sname) = \pi_{\text{dept}}(\sigma_{\text{enroll.student} = sname}(\text{ENROLL})) \]

• Use this notation to express the query that we want:
  \[ \pi_{sname}(\sigma_{\text{major} \not\in \text{depts}(sname)}(\text{STUDENT})) \]

• Finally, express \( \text{depts}(sname) \) as a subquery:
  \[ \pi_{sname}(\sigma_{\text{major} \not\in \pi_{\text{dept}}(\sigma_{\text{enroll.student} = sname}(\text{ENROLL}))}(\text{STUDENT})) \]

• SELECT SNAME
  FROM STUDENT
  WHERE MAJOR NOT IN
    (SELECT DEPT
     FROM ENROLL
     WHERE ENROLL.STUDENT = SNAME);
**Subqueries: Another Example**

- Given a table `ORDERS(OID,DATE,AMOUNT)`, produce a list of OID for all orders that are bigger than the smallest order placed on the same date:

  - `SELECT OID
    FROM ORDERS AS O
    WHERE AMOUNT >
      (SELECT MIN(AMOUNT)
       FROM ORDERS
       WHERE DATE = O.DATE);`
Subqueries that return multiple rows

• Subqueries may produce relations with multiple rows
• Then, we use operators that compare a single value with a set of values
• For a value v, a set s, and a comparison operator op:
  v op ANY s is true if v op x is true for some x in s
  v op ALL s is true if v op x is true for all x in s
• Examples:
  3 <= ANY {1,2,4}
  3 >= ALL {1,2,3}
**Subqueries with ALL and ANY**

- Given ORDER(OID, AMOUNT, DATE), find all orders that are bigger than the largest order on January 1, 1990

- SELECT OID
  FROM ORDERS
  WHERE AMOUNT > ALL
    (SELECT AMOUNT
     FROM ORDERS
     WHERE DATE = 1990.1.1);
Subqueries with ALL and ANY

• Find all orders that are bigger than the smallest order on January 1, 1990

• SELECT OID
  FROM ORDERS
  WHERE AMOUNT > ANY
    (SELECT AMOUNT
     FROM ORDERS
     WHERE DATE = 1990.1.1);
Testing for Emptiness

- The operator EXISTS tests whether the result of a subquery is empty

- Find all students who are enrolled in some class:
  ```sql
  SELECT NAME
  FROM STUDENT
  WHERE EXISTS
  (SELECT *
   FROM ENROLL
   WHERE ENROLL.STUDENT = NAME);
  ```
Testing for Emptiness (continued)

- Find all students who are not enrolled in any class:

  ```sql
  SELECT NAME
  FROM STUDENT
  WHERE NOT EXISTS
    (SELECT *
     FROM ENROLL
     WHERE ENROLL.STUDENT = NAME);
  ```
Compare the following two queries:

- `SELECT NAME FROM STUDENT WHERE EXISTS (SELECT * FROM ENROLL WHERE ENROLL.STUDENT = NAME);`
- `SELECT NAME FROM STUDENT, ENROLL WHERE ENROLL.STUDENT = STUDENT.NAME`
Modifying the Database

- Until now, no operations were done that modified the database. We were operating in the realm of algebra, that is, expressions were computed from inputs.
- For a real system, we need the ability to modify the relations.
- The three main constructs for modifying the relations are:
  - Insert
  - Delete
  - Update
Insertion of a tuple

• To insert a record into a relation:
  – INSERT INTO STUDENT
    VALUES (‘SMITH’,1115,’BOSTON’);
  – The order of the values is the order of the columns

• To override the default ordering for the attributes:
  – INSERT INTO STUDENT (SID,NAME)
    VALUES (1115,’SMITH’);
  – Here the default value (typically NULL) will be
given to the unmentioned attribute CITY

• Insertion of duplicates is allowed
Insertion of a Relation

• In general, we can insert any result of a query, as long as compatible, into a relation

• Suppose we have a relation
  NEWSTUDENT(SID, NAME, CITY, STATUS)

• Insert the admitted new students into the main table:
  INSERT INTO STUDENT(SID, NAME, CITY)
  SELECT SID, NAME, CITY
  FROM NEWSTUDENT
  WHERE STATUS = 'admitted'
Deletion

- **DELETE FROM STUDENT**
  WHERE CITY = 'DAYTON';
  - This removes rows that satisfy the specified condition

- **DELETE FROM STUDENT**;
  - This removes all rows from the relation, leaving an empty relation
  - But the relation attributes, along with their associated information, still remain
Implementing Difference with DELETE

- If an implementation doesn’t provide a difference operator, it can still be done with DELETE

- To compute the difference $R(A,B) - S(A,B)$, we can delete from $R$ all of the rows that appear in $S$:

  - `DELETE FROM R
    WHERE EXISTS
      (SELECT *
       FROM S
       WHERE (S.A = R.A AND S.B = R.B));`
To update information in a tuple:

- \textbf{UPDATE STUDENT}
  
  \textbf{SET NAME = ’JOE’, CITY = ’BOSTON’}
  
  \textbf{WHERE SID = 119;}

To update information in a batch of tuples:

- \textbf{UPDATE EMPLOYEE}
  
  \textbf{SET MANAGER=’Aho’, SALARY = SALARY * 2}
  
  \textbf{WHERE JOB=’programmer’}
SQL Embedded in a Host Language

- Sometimes we want to write programs that will interact with the database system.
- The advantage is that we can use the structure of the database, its layers, indices, etc.
- The disadvantage is that the host language may not understand the concepts of relations, tuples, etc.
- For such interactions, we can use a version of SQL called Embedded SQL.
SQL Commands as Procedure Calls

- SQL commands in host languages can, at a gross level, be viewed as procedure calls
- ANSI standard specifies Embedded SQL for some programming languages only
- A preprocessor converts Embedded SQL statements into the appropriate function calls in the host language
**Shared Variables**

- Variables in the host language are used to communicate with the SQL module
- They are declared in a special section
- To act on the STUDENT table, using C as a host language:
  
  ```c
  EXEC SQL BEGIN DECLARE SECTION;
  int sid;
  char name[10];
  char city[10];
  char SQLSTATE[6];
  EXEC SQL END DECLARE SECTION;
  ```
Examples of Embedded SQL Calls

- To get the city for a student, the host program could set sid to a value and then make the call:
  
  EXEC SQL SELECT CITY
  FROM STUDENT
  INTO :city
  WHERE SID = :sid

- To insert a record into the student table:
  
  EXEC SQL INSERT INTO STUDENT
  VALUES(:sid, :name, :city)
Queries that Produce Sets of Tuples

For queries that may return multiple tuples, we use a looping mechanism called a CURSOR.

1. Define a CURSOR, which resembles a query. This defines a relation without computing it.

2. OPEN the CURSOR. The relation is now computed, but not accessible.

3. FETCH CURSOR is repeatedly executed, to get each tuple. SQLSTATE is used to check whether the tuples are all processed.

4. Close the CURSOR
Example with CURSOR

Use up to $100,000 to increase the salaries of some employees in the sales department by 10 percent.

depth = 'sales'
EXEC SQL DECLARE CURSOR EMP FOR
SELECT SALARY
FROM EMPLOYEE
WHERE DEPT = :dept;

EXEC SQL OPEN CURSOR EMP;
amount = 0;
#define EOD strcmp(SQLSTATE,"02000")==0

23
Example with CURSOR (cont’d)

LOOP
  EXEC SQL FETCH EMP INTO :salary;
  IF (EOD) break;
  amount = amount + salary / 10;
  if (amount < 100,000)
    EXEC UPDATE EMPLOYEE
    SET SALARY = SALARY * 1.10;
    WHERE CURRENT OF EMP
  endif
ENDLOOP
EXEC SQL CLOSE CURSOR EMP;
Dynamic Embedded SQL

- What we have covered so far is static embedded SQL, because the queries were defined in the program text.
- With dynamic embedded SQL, a string containing a query is passed to the query module.
- LOOP
  
  Input a value for queryString from the user;
  EXEC SQL PREPARE qry FROM :queryString;
  EXEC SQL EXECUTE qry;
ENDLOOP
Topics covered in this unit

• NULLs
• Subqueries
• Database modifications:
  – Insert
  – Delete
  – Update
• Embedded SQL