

Review: A note on Views

- On creating a view the database can
 - Create a temporary table (preferred)
 - Can answer further queries directly.
 - On an update to base tables, incrementally update Views.
 - Just display results
 - If further queries result apply them on base tables
 - Inefficient
- Updating Views
 - Problems: Can be complicated and ambiguous (if you update a view, there could have been multiple database instances that can generate same view (see book p281))
 - Basically Not Advisable!
 - Some vendors do support them however.

SQL-programming language

1

Lecture 5 Part I

SQL Interfaces

SQL-programming language

2

SQL from a Standard Programming Language: The Problems

- Binding Problem
 - Need to support many programming languages with many different compilers.
- Impedance Problem
 - Programming Languages can only handle one row at a time.
 - SQL processes a relation (table) at a time.

SQL-programming language

3

SQL from a Standard Programming Language: Binding Styles

- Module Language (SQL/86)
- Embedded Static SQL (Precompiler converts to Module Language)
- Embedded Dynamic SQL (SQL/92)
- Call-Level Interface (CLI) (SQL/92)
 - Based on Microsoft's Open Database Connectivity feature (ODBC)

SQL-programming language

4

Binding: **Module** Example (no cursor)

```
PROCEDURE DELETE_PART
  ( SQLSTATE,
    :PNO_PARAM CHAR(6));
  DELETE FROM P WHERE P.PNO = :PNO_PARAM;
```

P & R PNO database names *Module*

```
DECLARE RETCODE CHAR(5);
DECLARE PNO_ARG CHAR(6);
DECLARE DELETE_PART ENTRY (CHAR(5), CHAR(6));
...
PNO_ARG = 'P2';
CALL DELETE_PART (RETCODE, PNO_ARG);
IF RETCODE = '00000'
  THEN ... ; /* delete operation succeeded */
  ELSE ... ; /* delete exception occurred */
```

PL/I

SQL-programming language

5

Binding: **Embedded SQL** (no cursor)

```
EXEC SQL BEGIN DECLARE SECTION;
  DECLARE SQLSTATE CHAR(5);
  DECLARE PNO CHAR(6);
EXEC SQL END DECLARE SECTION;
...
PNO = 'P2';
EXEC SQL DELETE FROM P WHERE P.PNO = :PNO;
IF SQLSTATE = '00000'
  THEN ... ; /* delete operation succeeded */
  ELSE ... ; /* delete exception occurred */
```

PL/I

SQL-programming language

6

SQL from a Standard Programming Language: **Impedance Problem**

- Cursors solve the Impedance Mismatch Problem
 - SQL - table (multiple rows)
 - vs. programming (single row)
- Operations similar to those with a file:
 - OPEN cursor - make rows available
 - FETCH repeat for each row
 - CLOSE cursor - when done

SQL-programming language

7

Cursors

- Solve the Impedance Mismatch Problem
 - SQL - table (multiple rows)
 - vs. programming (single row)
- Operations similar to those with a file:
 - OPEN cursor - make rows available
 - FETCH repeat for each row
 - CLOSE cursor - when done

SQL-programming language

8

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT  SP.SNO, SP.QTY
  FROM    SP
  WHERE   SP.PNO = 'P2';
DECLARE X   CHAR(5);
DECLARE Y   FIXED DECIMAL(5);
DECLARE Z   FIXED DECIMAL(3);

EXEC SQL OPEN C1;
DO for all rows accessible via cursor C1;
  EXEC SQL FETCH C1 INTO :X, :Y;
  process X and Y;
  EXEC SQL UPDATE SP
    SET QTY = QTY + :Z
    WHERE CURRENT OF C1;

END;
EXEC SQL CLOSE C1;
```

SQL-programming language

9

Binding: Embedded Dynamic SQL

```
strcpy( prep_string, "SELECT SP.SNO, SP.QTY
                    FROM SP
                    WHERE SP.PNO = ?");
EXEC SQL PREPARE s1 FROM :prep_string;
EXEC SQL DECLARE c1 CURSOR FOR s1;
host_var = P2;
EXEC SQL OPEN c1 USING :host_var;
::: do stuff with tuples, close cursor
host_var = P3;
EXEC SQL OPEN c1 USING :host_var;
SQL-programming language
::: do stuff with new set of tuples
```

10

Embedded SQL: Static vs Dynamic

- | | |
|---|--|
| <ul style="list-style-type: none">• Static+ less effort to create+ no need for runtime compilation+ less initial overhead so short programs are faster | <ul style="list-style-type: none">• Dynamic+ flexibility+ useful when query formation is not possible at compile-time.+ preferred for longer programs (performance reasons) |
|---|--|

SQL-programming language

11

Binding: Call-Level Interface (CLI)

- Essentially Dynamic SQL with function-level interface as opposed to an embedded SQL application.
 - Embedded SQL uses an embedded SQL interface requires a precompiler to convert the SQL statements into code, which is then compiled, bound to the database, and executed.
 - CLI simply invokes function calls, passing manipulation operations and queries as arguments
- Advantages:
 - Portability (Open Database Connectivity)
 - Elimination of precompiling & binding (no need for host variables)
 - Can do more than dynamic sql
- Disadvantages:
 - Overheads: start-up, function-calls

SQL-programming language

12

SQL

- Non-Programming Interface

- Direct SQL (isql)
 - Definition and Manipulation done interactively
 - Stored procedures can be loaded and executed
 - Basic Interaction
 - Load/Create Query Buffer
 - Go (execute)
- Read and follow tutorial notes (online) before starting next assignment.

SQL-programming language

13

Lecture 5 Part II

Relational Calculus

SQL-programming language

14

Introduction

- Relational algebra
 - required base for computing queries of SQL
 - a procedural way for stating queries--the 'how'
- Calculus is interested in relationships between quantities, and in deducing quantities through the relationships.
- Relational calculus
 - employs declarative expressions--the "what"
 - TRC is equivalent to relational algebra in its expressive power
- Relational calculus languages are based on first order predicate calculus

SQL-programming language

15

Propositional Logic Back to Basics

- A proposition is a statement which might be T/F
 - “students like 670”, “students don’t like 570”
- Propositional logic is concerned with operators which create new propositions from given ones.
 - “students like 670” and “students don’t like 570”
- Expressions of propositional logic rely on:
 - Atoms (p,q)
 - Logical connectives: OR, AND, NOT, IMPLIES
- Propositions can be true or false, dependent on the interpretation given to their atoms
 - p=“some students like 670”
 - q=“all students don’t like 570”

SQL-programming language

16

Predicate Logic

- Predicates are parameterized propositions, allowing references to classes of objects.
- Example Propositions
 - P = John likes 670
 - Q = James likes 570
 - R = Dan likes 677
- Predicate Versions of the above: likes(x,y)
- P = likes(John,670)
- Q = likes(James,570)
- R = likes(Dan,677)

SQL-programming language

17

Predicate Logic (contd)

- Predicate logic is an extension of propositional logic interested both in the sentential connectives of the atomic propositions, and in the internal structure of the atomic propositions.
- Atoms allow functions and relations on variables
 - E.g. likes(s2, course)
- The variables may be quantified: (there exists), (for all). Those that aren’t are **free variables**
 - For all courses there exist both a student that likes that course and a student that does not like that course.
 - $\forall c \exists s1, s2$ (likes (s1, c) AND ~likes(s2,c))

SQL-programming language

18

Predicate Calculus

- The domain of a query consists of the tuples which may be assigned to the free variables of the formula
 - {James,John,Dan} × {James, John, Dan}
- An assignment of values to the free variables of a formula is a tuple which provides a true or false value to the formula
 - Try all 9 tuples from the above cross product
- The selection of a query is the set of assignments to free variables that satisfy the query

SQL-programming language

19

TRC: Tuple Relational Calculus

- The queries employ formulas of predicate logic on tuple variables and with free variables
 - E.g. List all employee who earn > 50K
 - {t | EMP(t) and t.Salary > 50K}
- The only free tuple variables allowed in a relational calculus expression are those that appear to the left of the bar |
 - E.g. Retrieve Name and Address of all employees who work in columbus
 - {t.fname, t.lname, t.address | EMP(t) and ∃d.(DEPT(d) and d.LOC = "columbus" and d.DNO = EMP.DNO)}

SQL-programming language

20

Q11. Find the numbers of those departments that have employees who can do *some* job that is done by an employee in department D3.

{p.DNO |
DEPT(p) AND
∃ e ∃ j (EMP(e) and EMP(j) and
e.DNO = D3 and
j.DNO = p.DNO and e.JOB = j.JOB)}

SQL-programming language

21

Examples

Q11. Find the numbers of those departments that have employees who can do *some* job that is done by an employee in department D3.

```
{p.DNO |  
  DEPT(p) AND  
  ∃ e ∃ j (EMP(e) and EMP(j) and  
    e.DNO = D3 and  
    j.DNO = p.DNO and e.JOB = j.JOB )}
```

SQL-programming language

22

Examples

Find the numbers of those departments that have employees who can do *all* the jobs that are done by an employee in department D3.

```
• {p.DNO |  
  DEPT(p) AND  
  ∀ x_( not EMP(x) or not  
  (x.DNO=D3) or ∃ y( EMP(y)  
  AND y.JOB=x.JOB AND  
  y.DNO= p.DNO ) ) }
```

SQL-programming language

23

SQL-programming language

24
