SQL in the Real World

Chapter 10

1

Interactive vs. Non-Interactive SQL

- *Interactive SQL*: SQL statements input from terminal; DBMS outputs to screen
  - Inadequate for most applications
    - It may be necessary to process the data before output
    - Amount of data returned not known in advance
    - SQL has very limited expressive power (not Turing-complete)
- *Non-interactive SQL*: SQL statements are included in an application program written in a host language, like C, Java, COBOL

2

Application Program

- *Host language*: A conventional language (*e.g.*, C, Java) that supplies control structures, computational capabilities, interaction with physical devices
- *SQL*: supplies ability to interact with database.
- *Using the facilities of both*: the application program can act as an intermediary between an unsophisticated user at a terminal and the DBMS

3

Preparation

- Before an SQL statement is executed, it must be ***prepared***:
  - What indices can be used?
  - In what order should tables be accessed?
  - What constraints should be checked?
- Decisions based on schema, table sizes, etc.
- Result is a ***query execution plan***
- Preparation is a complex activity, usually done at run time, justified by the complexity of query processing

4

Introducing SQL Into the Application

- SQL statements can be incorporated into an application program in two different ways:
  - ***Statement Level Interface*** (SLI): Application program is a mixture of host language statements and SQL statements and directives
  - ***Call Level Interface*** (CLI): Application program is written entirely in host language
    - SQL statements are values of string variables that are passed as arguments to host language (library) procedures

5

Statement Level Interface

- SQL statements and directives in the application have a *special syntax* that sets them off from host language constructs
  - e.g., `EXEC SQL` *SQL_statement*
- *Precompiler* scans program and translates SQL statements into calls to host language library procedures that communicate with DBMS
- *Host language compiler* then compiles program

6

Statement Level Interface

- SQL constructs in an application take two forms:
  - Standard SQL statements (***static*** or ***embedded*** SQL): Useful when SQL portion of program is known at compile time
  - Directives (***dynamic*** SQL): Useful when SQL portion of program not known at compile time. Application constructs SQL statements *at run time* as values of host language variables that are manipulated by directives
- Precompiler translates statements and directives into arguments of calls to library procedures.

7

Call Level Interface

- Application program written entirely in host language (no precompiler)
  - Examples: JDBC, ODBC
- SQL statements are values of string variables constructed *at run time* using host language
  - Similar to dynamic SQL
- Application uses string variables as arguments of library routines that communicate with DBMS
  - e.g. executeQuery("*SQL query statement*")

8

Static SQL

```
EXEC SQL BEGIN DECLARE SECTION;
  unsigned long num_enrolled;
  char crs_code;
  char SQLSTATE [6];
EXEC SQL END DECLARE SECTION;
     ……….
EXEC SQL SELECT C.NumEnrolled
    INTO :num_enrolled
    FROM Course C
    WHERE C.CrsCode = :crs_code;
```

Variables shared by host and SQL

- Declaration section for host/SQL communication
- Colon convention for value (WHERE) and result (INTO) parameters

9

Status

```
EXEC SQL SELECT C.NumEnrolled
    INTO :num_enrolled
    FROM Course C
    WHERE C.CrsCode = :crs_code;
if ( !strcmp (SQLSTATE, "00000") ) {
     printf ( "statement failed" )
};
```

10

Connections

- To connect to an SQL database, use a connect statement

  CONNECT TO *database_name* AS *connection_name* USING *user_id*

11

Transactions

- No explicit statement is needed to begin a transaction
  - A transaction is initiated when the first SQL statement that accesses the database is executed
- The mode of transaction execution can be set with

  SET TRANSACTION READ ONLY
  ISOLATION LEVEL SERIALIZABLE
- Transactions are terminated with COMMIT or ROLLBACK statements

12

Example: Course Deregistration

```
EXEC SQL CONNECT TO :dbserver;
if ( ! strcmp (SQLSTATE, "000000") ) exit (1);
   …..
EXEC SQL DELETE FROM Transcript T
   WHERE T.StudId = :studid AND T.Semester = 'S2000'
                AND T.CrsCode = :crscode;
if (! strcmp (SQLSTATE, "000000") ) EXEC SQL ROLLBACK;
else {
   EXEC SQL UPDATE Course C
         SET C.Numenrolled = C.Numenrolled - 1
         WHERE C.CrsCode = :crscode;
   if (! strcmp (SQLSTATE, "000000") ) EXEC SQL ROLLBACK;
   else EXEC SQL COMMIT;
}
```

13

Buffer Mismatch

- **Problem**: SQL deals with tables (of arbitrary size); host language program deals with fixed size buffers
  - How is the application to allocate storage for the result of a SELECT statement?
- **Solution**: Fetch a single row at a time
  - Space for a single row (number and type of out parameters) can be determined from schema and allocated in application

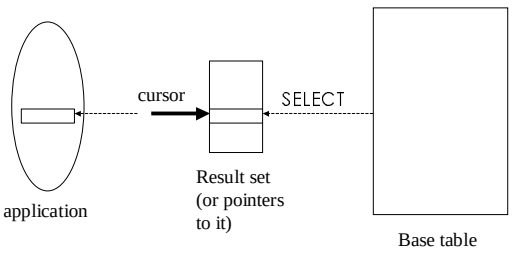
14

Cursor Overview

- ***Result set*** – set of rows produced by a SELECT statement
- ***Cursor*** – pointer to a row in the result set.
- Cursor operations:
  - *Declaration*
  - *Open* – execute SELECT to determine result set and initialize pointer
  - *Fetch* – advance pointer and retrieve next row
  - *Close* – deallocate cursor

15

Cursor



16

Cursor

```
EXEC SQL DECLARE GetEnroll INSENSITIVE CURSOR FOR
   SELECT T.StudId, T.Grade        --cursor is not a schema element
    FROM Transcript T
    WHERE T.CrsCode = :crscode  AND T.Semester = 'S2000';
   ……..                          Reference resolved by compile time,
EXEC SQL OPEN GetEnroll;          Value substituted at OPEN time
if ( !strcmp ( SQLSTATE, "00000")) {... fail exit... };
   ……..
EXEC SQL FETCH GetEnroll INTO :studid, :grade;
while ( SQLSTATE = "00000") {
   … process the returned row...
   EXEC SQL FETCH GetEnroll INTO :studid, :grade; }
if ( !strcmp ( SQLSTATE, "02000")) {... fail exit... };
   ……..
EXEC SQL CLOSE GetEnroll;
```

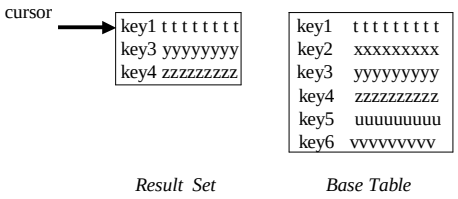
17

Cursor Types

- ***Insensitive cursor***: Result set (effectively) computed and stored in a separate table at OPEN time
  - Changes made to base table subsequent to OPEN (by any transaction) do not affect result set
  - Cursor is read-only
- *Cursors that are not insensitive*: Specification not part of SQL standard
  - Changes made to base table subsequent to OPEN (by any transaction) can affect result set
  - Cursor is updatable

18

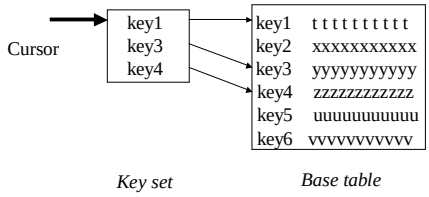
Insensitive Cursor



Keyset-Driven Cursor

- Example of a cursor that is not insensitive
- Primary key of each row in result set is computed at open time
- UPDATE or DELETE of a row in base table by a concurrent transaction between OPEN and FETCH might be seen through cursor
- INSERT into base table, however, not seen through cursor
- Cursor is updatable

Keyset-Driven Cursor



Cursors

DECLARE cursor-name [INSENSITIVE] [SCROLL]
CURSOR FOR table-expr
[ORDER BY column-list]
[FOR {READ ONLY | UPDATE [OF column-list] }]

For updatable (not insensitive, not read-only) cursors

UPDATE *table-name* --*base table*
SET *assignment*
WHERE CURRENT OF *cursor-name*

DELETE FROM *table-name* --*base table*
WHERE CURRENT OF *cursor-name*

Restriction – *table-expr* must satisfy restrictions of updatable view

Scrolling

- If SCROLL option not specified in cursor declaration, FETCH always moves cursor forward one position
- If SCROLL option is included in DECLARE CURSOR section, cursor can be moved in arbitrary ways around result set:

Get previous tuple

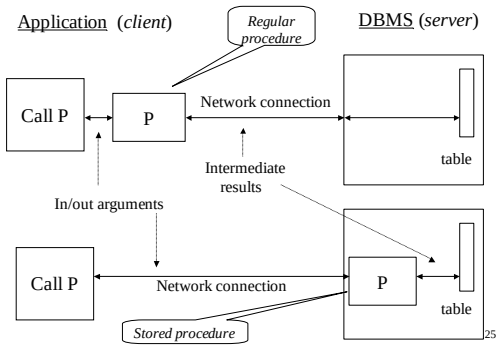
FETCH PRIOR FROM GetEnroll INTO :studid, :grade;

- Also: FIRST, LAST, ABSOLUTE n, RELATIVE n

Stored Procedures

- Procedure – written in a conventional algorithmic language
  - Included as schema element (stored in DBMS)
  - Invoked by the application
- Advantages:
  - Intermediate data need not be communicated to application (time and cost savings)
  - Procedure's SQL statements prepared in advance
  - Authorization can be done at procedure level
  - Added security since procedure resides in server
  - Applications that call the procedure need not know the details of database schema – all database access is encapsulated within the procedure

Stored Procedures



25

Stored Procedures

Schema:

```
CREATE PROCEDURE Register (char :par1, char :par2)
  AS BEGIN
       EXEC SQL SELECT ..... ;
        IF ( .....) THEN ……     -- SQL embedded in
                  ELSE ….       -- Persistent Stored Modules
                                -- (PSM) language
     END
```

Application:

```
EXEC SQL EXECUTE PROCEDURE Register ( :crscode, :studid);
```

26

Integrity Constraint Checking

- Transaction moves database from an initial to a final state, both of which satisfy all integrity constraints but ...
  - Constraints might not be true of intermediate states hence …
  - Constraint checks at statement boundaries might be inappropriate
- SQL (optionally) allows checking to be deferred to transaction COMMIT

27

Deferred Constraint Checking

Schema:

```
CREATE ASSERTION NumberEnrolled
  CHECK ( .....)
  DEFERRABLE;
```

Application:

```
SET CONSTRAINT NumberEnrolled DEFERRED;
```

Transaction is aborted if constraint is false at commit time

28