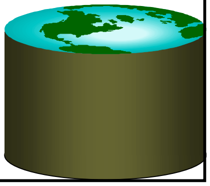
SQL: The Query Language Part 3

15-415, Spring 2003, Lecture 13
R &G - Chapter 6

> It is not every question
> that deserves an answer.
>
> Publius Syrus. 42 B. C.



Two more important topics

- **Constraints**
- **Triggers**

Integrity Constraints (Review)

- **An IC describes conditions that every *legal instance* of a relation must satisfy.**
  - Inserts/deletes/updates that violate IC's are disallowed.
  - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)
- ***Types of IC's*: Domain constraints, primary key constraints, foreign key constraints, general constraints.**
  - *Domain constraints*: Field values must be of right type. Always enforced.
  - *Primary key and foreign key constraints*: you know them.

General Constraints

- **Useful when more general ICs than keys are involved.**
- **Can use queries to express constraint.**
- **Checked on insert or update.**
- **Constraints can be named.**

```
CREATE TABLE  Sailors
    ( sid  INTEGER,
      sname  CHAR(10),
      rating  INTEGER,
      age  REAL,
      PRIMARY KEY (sid),
      CHECK  ( rating >= 1
          AND rating <= 10 ))
```

```
CREATE TABLE  Reserves
    ( sname  CHAR(10),
      bid  INTEGER,
      day  DATE,
      PRIMARY KEY  (bid,day),
      CONSTRAINT  noInterlakeRes
      CHECK  (`Interlake' <>
              ( SELECT  B.bname
                FROM  Boats B
                WHERE  B.bid=bid)))
```

Constraints Over Multiple Relations

- **Awkward and wrong!**
- **Only checks sailors!**
- **Only required to hold if the associated table is non-empty.**

```
CREATE TABLE  Sailors
    ( sid  INTEGER,
      sname  CHAR(10),
      rating  INTEGER,
      age  REAL,
      PRIMARY KEY  (sid),
      CHECK
      ( (SELECT COUNT (S.sid) FROM Sailors S)
      + (SELECT COUNT (B.bid) FROM
             Boats B) < 100 )
```

Number of boats plus number of sailors is < 100

- **ASSERTION is the right solution; not associated with either table.**

```
CREATE ASSERTION  smallClub
CHECK
( (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid)
 FROM Boats B) < 100 )
```

Or, Use a Trigger

- **Trigger: procedure that starts automatically if specified changes occur to the DBMS**
- **Three parts:**
  - Event (activates the trigger)
  - Condition (tests whether the triggers should run)
  - Action (what happens if the trigger runs)
- **Triggers (in some form) are supported by most DBMSs; Assertions are not.**
- **Support for triggers is defined in the SQL:1999 standard.**

Triggers

```
CREATE TRIGGER trigger_name
ON TABLE
{FOR {[INSERT][,][UPDATE][,][DELETE]}
[WITH APPEND]
AS
sql-statements
```

- **Cannot be called directly – initiated by events on the database.**
- **Can be *synchronous* or *asynchronous* with respect to the transaction that causes it to be fired.**

Triggers: Example

```
CREATE TRIGGER member_delete
ON member FOR DELETE
AS
IF (Select COUNT (*) FROM loan INNER JOIN member
     ON loan.member_no = deleted.member_no) > 0
  BEGIN
     PRINT 'ERROR - member has books on loan.'
     ROLLBACK TRANSACTION
   END
ELSE
DELETE reservation WHERE  reservation.member_no =
    deleted.member_no
```

Summary: Triggers, Assertions, Constraints

- **Very vendor-specific (although standard has been developed).**
- **Triggers vs. Contraints and Assertions:**
  - Triggers are “operational”, others are declarative.
- **Triggers can make the system hard to understand if not used with caution.**
  - ordering of multiple triggers
  - recursive/chain triggers
- **Triggers can be hard to optimize.**
- **But, triggers are also very powerful.**
- **Use to create high-performance, “*active”* databases.**

Writing Applications with SQL

- **SQL is not a general purpose programming language.**
  - + Tailored for data retrieval and manipulation
  - + Relatively easy to optimize and parallelize
  - - Can't write entire apps in SQL alone

Options:

Make the query language “turing complete”

Avoids the “impedance mismatch”

but, loses some of the advantages of relational langs.

Allow SQL to be embedded in regular programming languages.

Q: What needs to be solved to make the latter approach work?

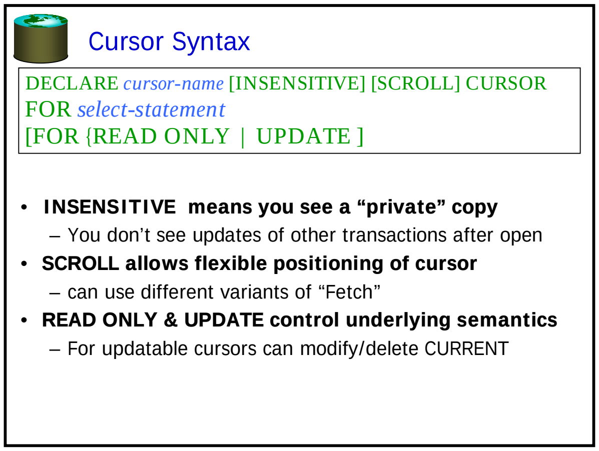
Embedded SQL

- SQL commands can be called from within a host language (e.g., C or COBOL) program.
- SQL statements can refer to host variables (plus special status variables SQLSTATE, SQLERROR).
  - Standard includes mapping of SQL data types to various PL data types.
- Must be able to *connect* to the right DB.
- Need compiler preprocessing or a preprocessor

```
EXEC SQL SELECT S.sname, S.age
     INTO :c_sname,:c_age
     FROM Sailors S
     WHERE S.sid = :c_sid
```

Cursors

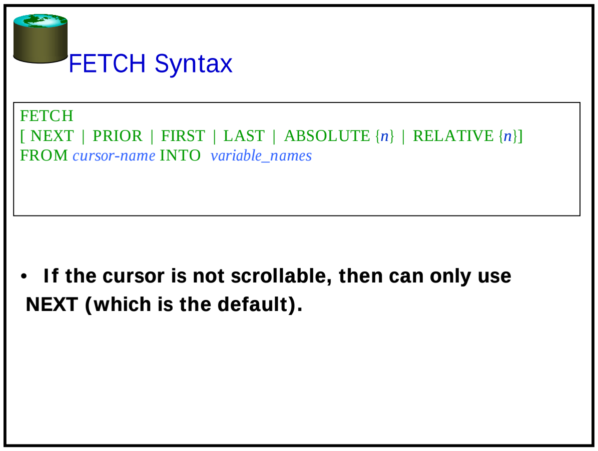
- Previous query worked because SID is a key.
- But, in general, SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records. No such data structure in C.
  - SQL supports ***cursors*** to handle this.
- Can declare a cursor on a relation or query statement (which generates a relation).
- Can *open* a cursor, and repeatedly *fetch* a tuple (which moves the cursor), until all tuples have been retrieved.
- Can also modify/delete tuple pointed to by a cursor.



Cursor Syntax

```
DECLARE cursor-name [INSENSITIVE] [SCROLL] CURSOR
FOR select-statement
[FOR {READ ONLY | UPDATE ]
```

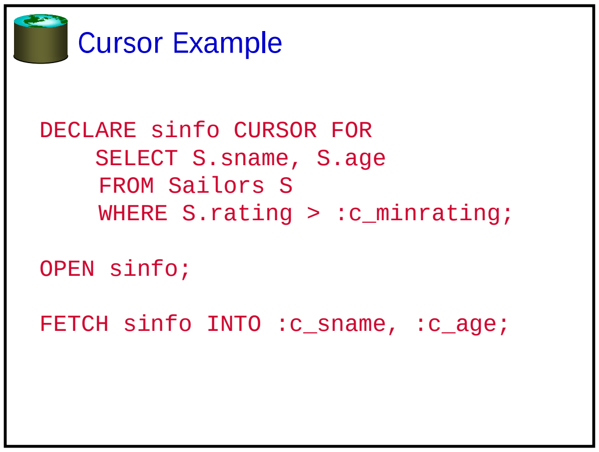
- **INSENSITIVE means you see a “private” copy**
  - You don’t see updates of other transactions after open
- **SCROLL allows flexible positioning of cursor**
  - can use different variants of “Fetch”
- **READ ONLY & UPDATE control underlying semantics**
  - For updatable cursors can modify/delete CURRENT



FETCH Syntax

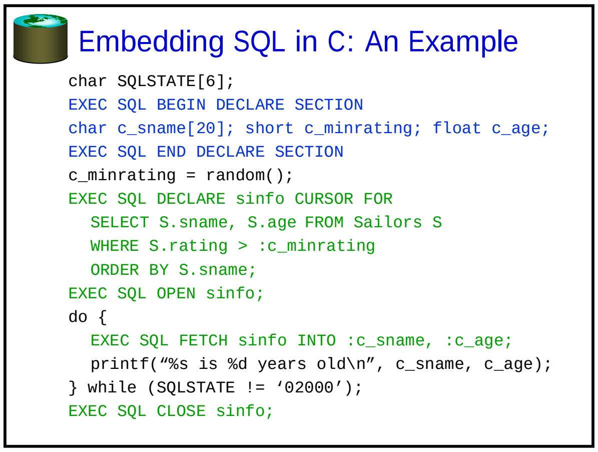
```
FETCH
[ NEXT | PRIOR | FIRST | LAST | ABSOLUTE {n} | RELATIVE {n}]
FROM cursor-name INTO  variable_names
```

- **If the cursor is not scrollable, then can only use NEXT (which is the default).**



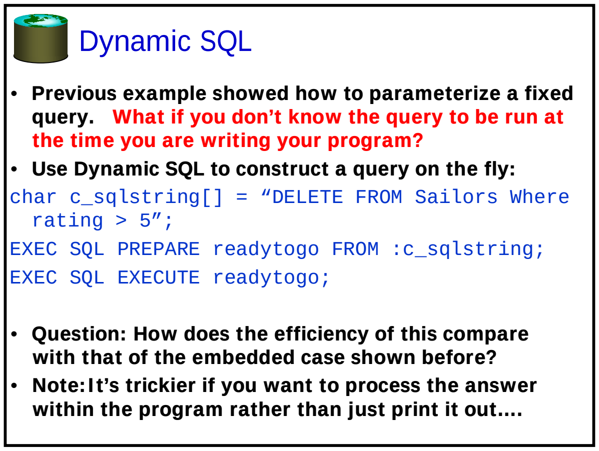
Cursor Example

```
DECLARE sinfo CURSOR FOR
    SELECT S.sname, S.age
    FROM Sailors S
    WHERE S.rating > :c_minrating;

OPEN sinfo;

FETCH sinfo INTO :c_sname, :c_age;
```



Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR
  SELECT S.sname, S.age FROM Sailors S
  WHERE S.rating > :c_minrating
  ORDER BY S.sname;
EXEC SQL OPEN sinfo;
do {
  EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
  printf(“%s is %d years old\n”, c_sname, c_age);
} while (SQLSTATE != ‘02000’);
EXEC SQL CLOSE sinfo;
```

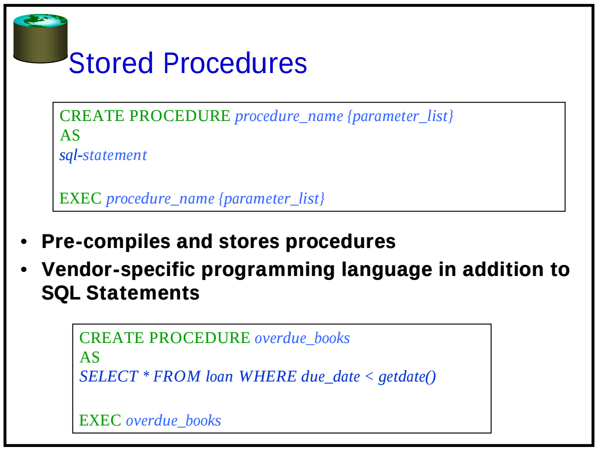


Dynamic SQL

- **Previous example showed how to parameterize a fixed query.   What if you don’t know the query to be run at the time you are writing your program?**
- **Use Dynamic SQL to construct a query on the fly:**

```
char c_sqlstring[] = “DELETE FROM Sailors Where
  rating > 5”;
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo;
```

- **Question: How does the efficiency of this compare with that of the embedded case shown before?**
- **Note:It’s trickier if you want to process the answer within the program rather than just print it out….**



Stored Procedures

```
CREATE PROCEDURE procedure_name {parameter_list}
AS
sql-statement

EXEC procedure_name {parameter_list}
```

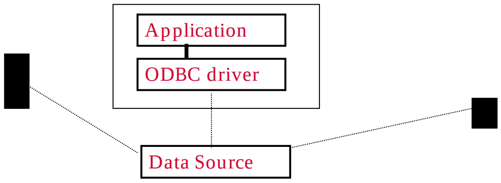
- **Pre-compiles and stores procedures**
- **Vendor-specific programming language in addition to SQL Statements**

```
CREATE PROCEDURE overdue_books
AS
SELECT * FROM loan WHERE due_date < getdate()

EXEC overdue_books
```

Database APIs: alternative to embedding

- **Rather than modify compiler, add a library with database calls (API)**
  - special procedures/objects
  - passes SQL strings from language, presents result sets in a language-friendly way
  - Microsoft's *ODBC* becoming C/C++ standard on Windows
  - Sun's *JDBC* a Java equivalent
  - For Perl there is DBI or "oraPerl"
  - Mostly DBMS-neutral (or at least they try to hide the complexities of dealing with different database systems).

Architecture



- a "driver" traps the calls and translates them into DBMS-specific code
  - Different levels of drivers provide functionality/performance tradeoffs
- database can be across a network
- Same program can be used (in theory) to access multiple database systems – by using different drivers.
- Data source may not even be an SQL database!

Visual C++ ODBC *(from microsoft.com)*

Visual C++ provides ODBC drivers for the following databases:

- **SQL Server**
- **Microsoft Access**
- **Microsoft FoxPro**
- **Microsoft Excel**
- **dBASE**
- **Paradox**
- **Oracle**
- **Text files**

SQL API in Java (JDBC)

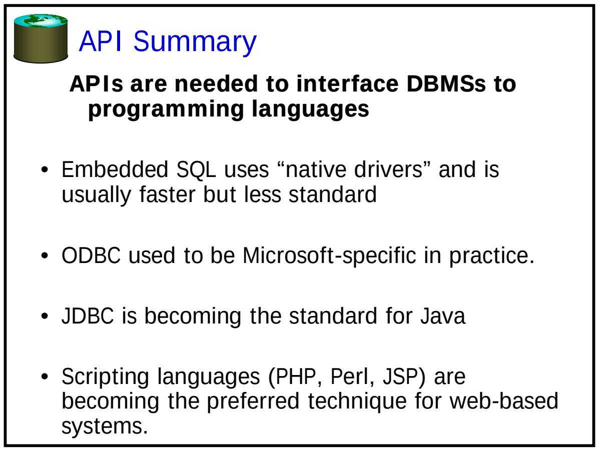
```
Connection con = // connect
    DriverManager.getConnection(url, ”login", ”pass");
Statement stmt = con.createStatement(); // set up stmt
String query = "SELECT COF_NAME, PRICE FROM COFFEES";
ResultSet rs = stmt.executeQuery(query);
try { // handle exceptions
    // loop through result tuples
    while (rs.next()) {
        String s = rs.getString("COF_NAME");
        Float n = rs.getFloat("PRICE");
        System.out.println(s + "   " + n);
    }
} catch(SQLException ex) {
    System.out.println(ex.getMessage ()
        + ex.getSQLState () + ex.getErrorCode ());
}
```

ODBC Code Sample

```
TRY
    {
            AllocStatusArrays();

            // call the ODBC catalog function with data member params
            AFX_SQL_ASYNC(this, (::SQLTables)(m_hstmt,
                        (m_strQualifierParam.IsEmpty()? (UCHAR FAR *)NULL:(UCHAR FAR *)(const
    char*)m_strTypeParam), SQL_NTS));
            if (!Check(nRetCode))
                        ThrowDBException(nRetCode, m_hstmt);

            // Allocate memory and cache info
            AllocAndCacheFieldInfo();
            AllocRowset();

            // Fetch the first row of data
            MoveNext();

            // If EOF, result set is empty, set BOF as well
            m_bBOF = m_bEOF;
    }
```

Perl DBI Sample (we use PHP instead)

```
use DBI;
my $dbh = DBI->connect('DBI:Oracle:payroll')
 or die "Couldn't connect to database: " . DBI->errstr;
 my $sth = $dbh->prepare('SELECT * FROM people WHERE lastname = ?')
      or die "Couldn't prepare statement: " . $dbh->errstr;
  print "Enter name> ";
  while ($lastname = <>) {          # Read input from the user
    my @data;
    chomp $lastname;
    $sth->execute($lastname)          # Execute the query
     or die "Couldn't execute statement: " . $sth->errstr;
    # Read the matching records and print them out
    while (@data = $sth->fetchrow_array()) {
     my $firstname = $data[1];
     my $id = $data[2];
     print "\t$id: $firstname $lastname\n";
    }
    if ($sth->rows == 0) {
     print "No names matched `$lastname'.\n\n";
    }
    print "\n";
    print "Enter name> ";
   }
```



API Summary

APIs are needed to interface DBMSs to programming languages

- Embedded SQL uses “native drivers” and is usually faster but less standard
- ODBC used to be Microsoft-specific in practice.
- JDBC is becoming the standard for Java
- Scripting languages (PHP, Perl, JSP) are becoming the preferred technique for web-based systems.