

### Announcements (September 28)

- ✤ Homework #1 graded
- ✤ Homework #2 due today
  - Solution available this weekend
- Midterm in class next Thursday (October 5)
  - Open book, open notes
  - Format similar to the sample midterm
     Solution available this weekend
  - · Covers everything up to next Tuesday's lecture
  - Emphasizes materials exercised in homeworks
- \* Check handout box if you missed any handouts!
- ✤ Project milestone #1 due in 2 weeks

### Motivation

- \* Pros and cons of SQL
  - Very high-level, possible to optimize
  - Not intended for general-purpose computation
- Solutions
  - Augment SQL with constructs from general-purpose programming languages (SQL/PSM)
  - Use SQL together with general-purpose programming languages (JDBC, embedded SQL, etc.)

# Impedance mismatch and a solution

- $\clubsuit$  SQL operates on a set of records at a time
- Typical low-level general-purpose programming languages operates on one record at a time
- Solution: cursor
  - Open (a result table): position the cursor before the first row
  - Get next: move the cursor to the next row and return that row; raise a flag if there is no such row
  - Close: clean up and release DBMS resources
  - Found in virtually every database language/APIWith slightly different syntaxes
  - Some support more positioning and movement options, modification at the current position (analogous to view update), etc.

# Augmenting SQL: SQL/PSM

- ✤ PSM = Persistent Stored Modules
- CREATE PROCEDURE proc\_name ( parameter\_declarations ) local\_declarations procedure\_body;
- CREATE FUNCTION func\_name ( parameter\_declarations ) RETURNS return\_type local\_declarations procedure\_body;
- CALL proc\_name ( parameters );
- \$ Inside procedure body: SET variable = CALL func\_name ( parameters );

SQL/PSM example	6
CREATE FUNCTION SetMaxGPA(IN newMaxGPA FLOAT) RETURNS INT Enforce newMaxGPA; return number of rows modified. BEGIN DECLARE rowsUpdated INT DEFAULT 0; DECLARE studentGursor CURSOR FOR SELECT GPA FROM Student FOR UPDATE; Set a flag whenever there is a "not found" exception: DECLARE noMoreRows INT DEFAULT 0; DECLARE noMoreRows INT DEFAULT 0; DECLARE continue HANDLER FOR NOT FOUND SET noMoreRows = 1; (see next slide) RETURN rowsUpdated; END	

### SQL/PSM example continued

<ul> <li>Fetch the first result row:</li> <li>OPEN studentCursor;</li> <li>FETCH FROM studentCursor INTO thisGPA;</li> <li>Loop over all result rows:</li> <li>WHILE noMoreRows &lt;&gt; 1 D0</li> </ul>
IF thisGPA > newMaxGPA THEN
Enforce newMaxGPA:
UPDATE Student SET Student.GPA = newMaxGPA WHERE CURRENT OF studentCursor;
Update count:
SET rowsUpdated = rowsUpdated + 1;
END IF:
Fetch the next result row:
FETCH FROM studentCursor INTO thisGPA;
END WHILE;
CLOSE studentCursor;

### Other SQL/PSM features

- Assignment using scalar query results
   SELECT INTO
- Other loop constructs
  - FOR, REPEAT UNTIL, LOOP
- Flow control
- GOTO
   Exceptions
  - SIGNAL, RESIGNAL
- ...
- For more DB2-specific information, check out Developing SQL and External Routines
  - Available as part of DB2 v9 manual collection, or directly as
  - ftp://ftp.software.ibm.com/ps/products/db2/info/vr9/pdf/letter/en\_US/db2a3e90.pdf

### Interfacing SQL with another language

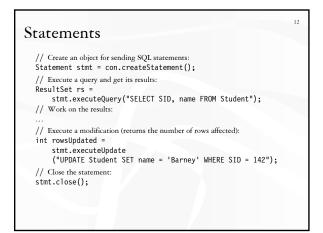
- ♦ API approach
  - SQL commands are sent to the DBMS at runtime
  - Examples: JDBC, ODBC (for C/C++/VB), Perl DBI
  - These API's are all based on the SQL/CLI (Call-Level Interface) standard
- \* Embedded SQL approach
  - SQL commands are embedded in application code
  - A precompiler checks these commands at compile-time and converts them into DBMS-specific API calls
  - Examples: embedded SQL for C/C++, SQLJ (for Java)

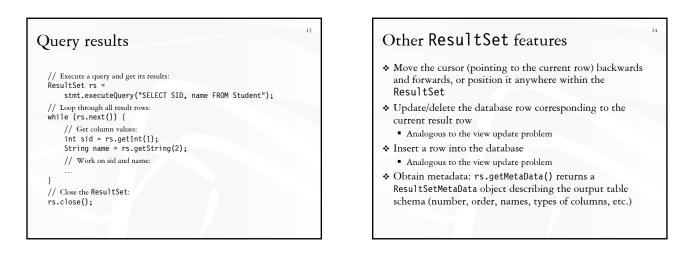
# beta provide the second s

### Connections // Connection URL is a DBMS-specific string: String url = "jdbc:db2://localhost:50000/dbcourse"; // Making a connection: Connection con = DriverManager.getConnection(url, user, password); ... // Closing a connection: con.close(); For clarity we are ignoring

For clarity we are ignoring exception handling for now

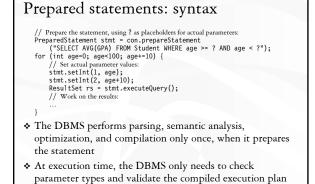
11





### Prepared statements: motivation Statement stmt = con.createStatement(); for (int age=0; age<100; age+=10) { ResultSet rs = stmt.executeQuery ("SELECT AVG(GPA) FROM Student" + " WHERE age >= " + age + " AND age < " + (age+10));</pre> // Work on the results: \* Every time an SQL string is sent to the DBMS, the DBMS

- must perform parsing, semantic analysis, optimization, compilation, and then finally execution
- \* These costs are incurred 10 times in the above example
- \* A typical application issues many queries with a small number of patterns (with different parameter values)



Transaction processing Set isolation level for the current transaction con.setTransactionIsolationLevel(/); Where l is one of TRANSACTION SERIALIZABLE (default), TRANSACTION REPEATABLE READ, TRANSACTION READ COMITTED, and TRANSACTION READ UNCOMMITTED Set the transaction to be read-only or read/write (default) con.setReadOnly(true|false); Turn on/off AUTOCOMMIT (commits every single statement) con.setAutoCommit(true|false); Commit/rollback the current transaction (when AUTOCOMMIT is off) con.commit(); • con.rollback();

### Odds and ends of JDBC Most methods can throw SQLException Make sure your code catches them Remember to close Statement, ResultSet, etc., in finally block getSQLState() returns the standard SQL error code getMessage() returns the error message DataSource interface for establishing connections Better than through DriverManager

18

- Methods for examining metadata in databases
- \* Methods to retrieve the value of a column for all result rows into an array without calling ResultSet.next() in a loop
- \* Methods to construct/execute a batch of SQL statements

# JDBC drivers - Types I, II

- Type I (bridge): translate JDBC calls to a standard API not native to the DBMS (e.g., JDBC-ODBC bridge)
  - Driver is easy to build using existing standard API's
  - Extra layer of API adds overhead
- Type II (native API, partly Java): translates JDBC calls to DBMS-specific client API calls
  - DBMS-specific non-Java client library needs to be installed on each client
  - Good performance

# JDBC drivers – Types III, IV

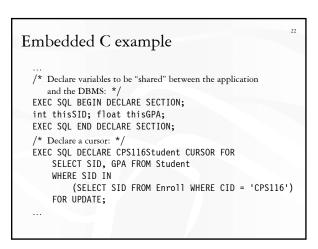
- Type III (network bridge): sends JDBC requests to a middleware server which in turn communicates with a database
  - Client JDBC driver is completely Java, easy to build, and does not need to be DBMS-specific
  - Middleware adds translation overhead
- Type IV (native protocol, full Java): converts JDBC requests directly to native network protocol of the DBMS
  - Client JDBC driver is completely Java but is also DBMS-specific
  - Good performance
  - Supported by, e.g., com.ibm.db2.jcc.DB2Driver

### Additional Information

Documentation for JDBC and API docs for java.sql.\*

21

- For DB2-specific information, check out Developing Java Applications
  - Available as part of DB2 v9 manual collection, or directly as ftp://ftp.software.ibm.com/ps/products/db2/info/vr9/pdf/letter/en\_US/db2a3e90.pdf
- Example code on rack040
  - Web-db-beers: To obtain a copy of the source code, follow instructions on course Web site under Programming Notes / Tomcat Notes
  - RA (less documented): /home/dbcourse/software/ra-2.0b/



### Embedded C example continued

/\* Open the cursor: \*/
EXEC SQL OPEN CPS1165tudent;
/\* Specify exit condition: \*/
EXEC SQL WHENEVER NOT FOUND D0 break;
/\* Loop through result rows: \*/
while (1) {
 /\* Get column values for the current row: \*/
 EXEC SQL FETCH CPS116Student INTO :thisSID, :thisGPA;
 printf("SID %d: current GPA is %f\n", thisSID, thisGPA);
 /\* Updare GPA: \*/
 printf("Enter new GPA: ");
 scanf("%f", %thisGPA);
 EXEC SQL UPDATE Student SET GPA = :thisGPA
 WHERE CURRENT OF CPS116Student;
}
/\* Close the cursor: \*/
EXEC SQL CLOSE CPS116Student;

### Pros and cons of embedded SQL

### \* Pros

- More compile-time checking (syntax, type, schema, ...)
- Code could be more efficient (if the embedded SQL statements do not need to checked and recompiled at run-time)
- Cons
  - DBMS-specific
    - Vendors have different precompilers which translate code into different native API's
    - Application executable is not portable (although code is)
    - Application cannot talk to different DBMS at the same time

# Pros and cons of augmenting SQL

### & Cons

- Already too many programming languages
- SQL is already too big
- General-purpose programming constructs complicate optimization, and make it difficult to tell if code running inside the DBMS is safe

25

- At some point, one must recognize that SQL and the DBMS engine are not for everything!
- \* Pros
  - More sophisticated stored procedures and triggers
  - More application logic can be pushed closer to data