Database Systems

Application Development

H. Turgut Uyar Şule Öğüdücü

2002-2012

License

Topics

Application Development
- Introduction
- Embedded SQL
- ODBC
- JDBC

SQL
- Stored Procedures
- Views
- Permissions
- Tips

Introduction

- using the database language in conjunction with a general-purpose programming language
- general-purpose language: host language
- mismatch between SQL and the host language:
  - SQL operations on sets
  - iteration constructs in programming languages

Program Structure

- connect
  - server, database, username, password
- run statements as needed:
  - update operations return number of affected rows
  - query operations return result sets
    → iterate over rows
- disconnect

Approaches

- application programming interface (API)
- embedded SQL
- ODBC
- language standard interfaces

Application Programming Interface

- using the library functions of the SQL server
- pros: fast
- cons: specific to the SQL product

API Example

Example (PostgreSQL - C)

```
#include <libpq-fe.h>

int main(int argc, char *argv[])
{
    /* connect */
    /* execute query */
    /* disconnect */
}
```

API Example

Example (connecting)

```
/* PGconn *conn; */

conn = PQconnectdb("host=localhost dbname=imdb"
                " user=itucs password=itucs");
if (PQstatus(conn) == CONNECTION_BAD) {
    fprintf(stderr, "Connection failed.\n");
    exit(1);
}
/* execute query */
PQfinish(conn);
```

API Example

Example (executing a query)

```
/* PGresult *result; */

sprintf(query, "SELECT TITLE, SCORE"
        " FROM MOVIE WHERE (YR = %d)", year);
result = PQexec(conn, query);
if (PQresultStatus(result) != PGRES_TUPLES_OK) {
    fprintf(stderr, "Query failed.\n");
    PQclear(result);
    PQfinish(conn);
    exit(1);
}
```

API Example

Example (processing the result set)

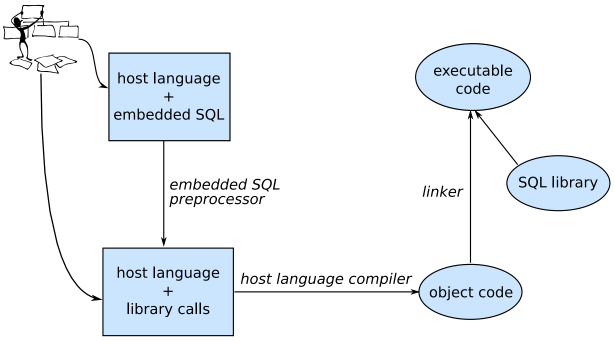
```
for (i = 0; i < PQntuples(result); i++) {
    title = PQgetvalue(result, i, 0);
    score = PQgetvalue(result, i, 1);
    ...
}

PQclear(result);
```

Embedded SQL

- stages:
  1. mark SQL statements within host language code: `EXEC SQL`
  2. embedded SQL preprocessor:
     embedded SQL directives → API calls
  3. host language compiler
- pros: fast, standard
- cons: difficult, does not support most languages

skip Embedded SQL

Embedded SQL



Embedded SQL Standard

- sharing variables with the host language
- error control
- adapting query results

Variable Sharing

Syntax

```
EXEC SQL BEGIN DECLARE SECTION;
shared variables
EXEC SQL END DECLARE SECTION;
```

- ':' in front of host language variables in SQL statements

Error Control

Error Processing

```
EXEC SQL WHENEVER
  { SQLERROR | SQLWARNING | NOT FOUND }
  { STOP | CONTINUE | DO command | GOTO label }
```

Adapting Query Results

Cursors

```
EXEC SQL DECLARE cursor_name CURSOR FOR
    SELECT ...;
EXEC SQL OPEN cursor_name;
EXEC SQL FETCH IN cursor_name INTO variables;
EXEC SQL CLOSE cursor_name;
```

- query is not executed when cursor is defined
- it is executed when cursor is opened
  - cursor points to first element in the result set

Embedded SQL Example

Example (connecting)

```
EXEC SQL BEGIN DECLARE SECTION;
int year;
char *title = NULL, *score = NULL;
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO imdb
    USER itucs IDENTIFIED BY itucs;

/* process query */

EXEC SQL DISCONNECT;
```

Embedded SQL Example

Example (processing query)

```
scanf("%d", &year);
EXEC SQL DECLARE c_query CURSOR FOR
    SELECT TITLE, SCORE FROM MOVIE
      WHERE (YR = :year);
EXEC SQL OPEN c_query;

/* execute query */

EXEC SQL CLOSE c_query;
EXEC SQL COMMIT;
```

Embedded SQL Example

Example (executing query)

```
EXEC SQL WHENEVER NOT FOUND DO break;
while (1) {
    EXEC SQL FETCH c_query INTO :title, :score;
    ...
}
```

ODBC

- ODBC: Open DataBase Connectivity
  a service layer between the application and the server
- pros: standard
- cons: slow

ODBC Architecture

- application
- driver manager
  - registers the ODBC drivers
  - transfers requests from application to driver
- driver
  - translates and transfers requests to data source
- data source
  - processes instructions from the driver

ODBC Example

Example (PHP)

```
$conn = odbc_connect("imdb", "itucs", "itucs");
$query = "SELECT TITLE, SCORE FROM MOVIE"
       . " WHERE (YR = " . $year . ")";
$result = odbc_exec($conn, $query);

/* process the result set */

odbc_close($conn);
```

ODBC Example

Example (processing the result set)

```
echo "<table>\n";
while (odbc_fetch_row($result)) {
    $title = odbc_result($result, "title");
    $score = odbc_result($result, "score");
    echo "<tr>\n";
    echo "  <td>$title</td>\n";
    echo "  <td>$score</td>\n";
    echo "</tr>\n";
}
echo "</table>\n";
```

JDBC

- JDBC: Java DataBase Connectivity
- same architectural concepts as in ODBC
  - different types of drivers
- JDBC URL for connection
  - `jdbc:<subprotocol>:<parameters>`
- matching Java and SQL data types

JDBC Drivers

- *Type I*: bridges
  - translate into non-native calls (for example ODBC)
- *Type II*: direct translation via non-Java driver
  - translate into API of data source (for example C++)
- *Type III*: network bridges
  - connect to middleware server
for translating into API of data source
- *Type IV*: direct translation via Java driver
  - communicate with DBMS through Java sockets

JDBC Flow

- get a connection object
  - static: `DriverManager.getConnection()` → `Connection`
- create a statement object on the connection
  - `Connection.createStatement()` → `Statement`
- execute the query
  - read: `Statement.executeQuery(query)` → `ResultSet`
  - insert, update, delete: `Statement.executeUpdate(query)`
- process the results
- close resources which are no longer needed
  (result sets, statements, connections)

Processing Results

- `ResultSet` is an iterator
  - whether there are more rows: `ResultSet.hasNext()`
  - proceed to the next row: `ResultSet.next()`
- convert and transfer data in the row to variables
  - by column name: `ResultSet.getXXX(name)`
  - by column order: `ResultSet.getXXX(order)`

Data Type Conversions

SQL type	Java class	ResultSet method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Float	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

JDBC Example

Example (loading the database driver)

```
try {
  Class.forName("org.postgresql.Driver");
} catch (ClassNotFoundException e) {
  // PostgreSQL driver not installed
}
```

JDBC Example

Example (connecting)

```
try {
  Connection conn = DriverManager.getConnection(
    "jdbc:postgresql:imdb", "itucs", "itucs"
  );
} catch (SQLException e) {
  // connection error
}
```

JDBC Example

Example (inserting)

```
String query = "INSERT INTO MOVIE (TITLE, YR)"
    + " VALUES ('Casablanca', 1942)";
Statement stmt = conn.createStatement();
stmt.executeUpdate(query);
stmt.close();
```

Prepared Statements

- prepared statements can be reused by changing parameter values
- creating: `Connection.prepareStatement(query)` → `PreparedStatement`
- placeholder for parameters in query: '?'
  - values must be set before executing: `PreparedStatement.setXXX(order, value)`
- executing: `PreparedStatement.executeQuery()` or `PreparedStatement.executeUpdate()`

Prepared Statement Example

Example (inserting - using prepared statement)

```
String query = "INSERT INTO MOVIE (TITLE, YR)"
    + " VALUES (?, ?)";
PreparedStatement stmt =
    conn.prepareStatement(query);
for (Movie movie : getMovies()) {
    stmt.setString(1, movie.getTitle());
    stmt.setInt(2, movie.getYear());
    stmt.executeUpdate();
}
stmt.close();
```

Result Set Example

Example (listing movie titles in a year)

```
String query = String.format(
  "SELECT TITLE FROM MOVIE WHERE (YR = %d)",
  year);
Statement stmt = conn.createStatement();
ResultSet results = stmt.executeQuery(query);
while (results.next()) {
    String title = results.getString("TITLE");
    System.out.println("Title: " + title),
}
results.close();
stmt.close();
```

JDBC Example

Example (deleting)

```
String query = "DELETE FROM MOVIE" +
    " WHERE (ID = ?)";
PreparedStatement stmt =
    conn.prepareStatement(query);
stmt.setInt(1, movie.getId());
stmt.executeUpdate();
stmt.close();
```

JDBC Example

Example (updating)

```
String query = "UPDATE MOVIE SET YR = ?" +
    " WHERE (ID = ?)";
PreparedStatement stmt =
    conn.prepareStatement(query);
stmt.setInt(1, movie.getYear());
stmt.setInt(2, movie.getId());
stmt.executeUpdate();
stmt.close();
```

Closing Resources

- it is recommended to close resources
  such as results sets and statements
  in the `finally` part of `try` - `catch` - `finally` blocks

Resource Closing Example

```
Statement stmt = conn.createStatement();
ResultSet results = null;
try {
    results = stmt.executeQuery(query);
    ...
} catch (SQLException e) {
    ...
} finally {
    results.close();
    stmt.close();
}
```

Auto-Generated Identity Values

- to obtain auto-generated identity values,
  use the `RETURN_GENERATED_KEYS` flag
  when creating the statement
- after executing the statement:
  `Statement.getGeneratedKeys()` → `ResultSet`

Auto-Generated Keys Example

```
String query = "INSERT INTO ... ";
PreparedStatement stmt =
    connection.prepareStatement(query,
        Statement.RETURN_GENERATED_KEYS);
stmt.executeUpdate();

ResultSet ids = statement.getGeneratedKeys();
// assuming there is one and only one result
ids.next();
int id = ids.getInt(1);
```

Batch Mode

- accumulating statements to run them in batches
  might be faster
- get the connection out of "auto commit" mode
- accumulate: `Statement.addBatch(query)`
- execute accumulated statements:
  `Statement.executeBatch()`
- finalizing changes: `Connection.commit()`

Batch Mode Example

```
stmt = conn.createStatement( ... );
conn.setAutoCommit(false);
int queryCount = 0;
int batchSize = 100;
for ( ... ) {
    stmt.addBatch(query);
    queryCount++;
    if (queryCount % batchSize == 0) {
        stmt.executeBatch();
        conn.commit();
    }
}
```

Fetch Size

- the JDBC drivers prefetch rows from the query
- increases performance, but also increases memory usage
- setting a smaller fetch size decreases memory usage: `Statement.setFetchSize(count)`

Fetch Size Example

```
stmt.setFetchSize(1);

// MySQL
stmt.setFetchSize(Integer.MIN_VALUE)
```

Stored Procedures

- implementing functionality in the database server
  - languages: SQL, PL/SQL, C, ...
- not recommended
  - not portable
  - not scalable
  - database servers are not optimized for business logic

  → implement business logic on the application server

Creating Functions

Statement

```
CREATE FUNCTION
  function_name([parameter_type [, ...]])
  RETURNS return_type
  AS function_body
  LANGUAGE language_name
```

- first parameter $1, second parameter $2, ...

SQL Function Example

Example (calculating new score)

$1: old score, $2: old votes, $3: new vote

```
CREATE FUNCTION NEW_SCORE(float, int, int)
  RETURNS float
  AS 'SELECT ($1*$2+$3) / ($2+1);'
  LANGUAGE 'sql'
```

Triggers

Definition

trigger: a function that will be automatically activated on an event

- can be useful for maintaining integrity

Creating Triggers

Statement

```
CREATE TRIGGER trigger_name
  { BEFORE | AFTER } { event [ OR ... ] }
  ON table_name
  [ FOR [ EACH ] { ROW | STATEMENT } ]
  EXECUTE PROCEDURE function_name(...)
```

- PL/pgSQL:
  - `old`: tuple before the operation
  - `new`: tuple after the operation

Trigger Example

Example (let SCORE * VOTES be kept in the POINTS column)

```
CREATE FUNCTION UPDATE_MOVIE_POINTS()
  RETURNS opaque
  AS 'BEGIN
      new.POINTS = new.SCORE * new.VOTES;
      RETURN new;
      END;'
  LANGUAGE 'plpgsql'
```

Trigger Example

Example (calculate POINTS automatically on updates)

```
CREATE TRIGGER UPDATE_MOVIE
  BEFORE INSERT OR UPDATE ON MOVIE
  FOR EACH ROW
  EXECUTE PROCEDURE UPDATE_MOVIE_POINTS()
```

Views

- presenting a derived table like a base table
- isolating users and application programs from changes in database structure

Creating Views

Statement

```
CREATE VIEW view_name AS
  SELECT ...
```

- the SELECT query will be executed every time the view is used

View Example

Example

```
CREATE VIEW NEW_MOVIE AS
  SELECT ID, TITLE, YR FROM MOVIE
    WHERE (YR > 1995)

SELECT * FROM NEW_MOVIE
```

Updating Views

- changes have to performed on the base tables
  - rules need to be defined

Creating Rules

```
CREATE RULE rule_name AS
  ON event TO view_name
  [ WHERE condition ]
  DO [ INSTEAD ] sql_statement
```

View Rule Example

Example

```
UPDATE NEW_MOVIE SET TITLE='..'
  WHERE (ID = 1)

CREATE RULE UPDATE_TITLE AS
  ON UPDATE TO NEW_MOVIE
  DO INSTEAD
    UPDATE MOVIE SET TITLE = new.TITLE
      WHERE (ID = old.ID)
```

Permissions

- subject: active entities (user, group)
- object: passive entities (table, column, view, ...)
- owner of object determines permissions of other subjects: *discretionary access control*

SQL Permissions

Granting Permissions

```
GRANT permission_name [, ...]
  ON object_name TO subject_name
  [ WITH GRANT OPTION ]
```

Revoking Permissions

```
REVOKE permission_name
  ON object_name FROM subject_name
```

Permission Examples

Example (granting permissions on a table)

```
GRANT SELECT, INSERT, UPDATE ON MOVIE
  TO 'itucs'
```

Example (revoking permissions on a table)

```
REVOKE INSERT ON MOVIE
  FROM 'itucs'
```

Indexes

- some operations require sorting:
  `ORDER BY, DISTINCT, GROUP BY, UNION, ...`
- creating indexes speeds up queries
  - slows down insert and update operations

Statement

```
CREATE [ UNIQUE ] INDEX index_name
  ON table_name(column_name [, ...])
```

Bulk Data Transfer

- use vendor-specific bulk data transfer commands instead of inserting or deleting one-by-one
- export to / import from tab separated value files

Bulk Data Transfer

PostgreSQL

```
COPY table_name (column_name [, ...])
  TO 'output_file_path'

COPY table_name (column_name [, ...])
  FROM 'input_file_path'
```

Bulk Data Transfer

MySQL

```
SELECT column_name [, ...] FROM table_name
  INTO OUTFILE 'output_file_path'

LOAD DATA INFILE 'input_file_path'
  INTO TABLE table_name (column_name [, ...])
```

References

Required Reading: Date

- Chapter 4: An Introduction to SQL
  - 4.6. Embedded SQL
- Chapter 9: Integrity
  - 9.11. Triggers (a Digression)
- Chapter 10: Views

Supplementary Reference: Ramakrishnan, Gehrke

- Chapter 6: Database Application Development