Database Systems

Application Development

H. Turgut Uyar Şule Öğüdücü

2002-2010

License

Topics

Application Development
- Introduction
- Embedded SQL
- ODBC
- JDBC

SQL Facilities
- Stored Procedures
- Views
- Permissions
- Performance

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
- as necessary:
  - execute a query
  - iterate over the result set
- disconnect

Approaches

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

Application Programming Interface

- using the library functions of the SQL server
- pros and cons:
  - fast
  - not standard

API Example

Example (PostgreSQL - C)

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

int main(void)
{
    /* connect */
    /* execute query */
    /* disconnect */
}
```


API Example

Example (connecting)

```c
/* 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)

```c
/* 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)

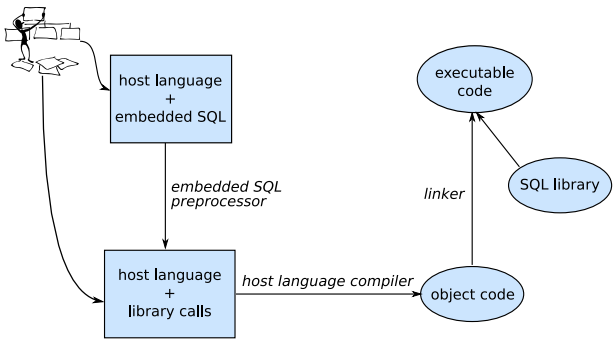
```c
for (i = 0; i < PQntuples(result); i++)
    printf("%s %s\n",
           PQgetvalue(result, i, 0),
           PQgetvalue(result, i, 1));

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 and cons:
  - fast, standard
  - 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 SQL statements: ':' in front of host language variables

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 yr;
char *title = NULL, *score = NULL;
EXEC SQL END DECLARE SECTION;

EXEC SQL CONNECT TO itucs
    USER itucs IDENTIFIED BY itucs;

/* process query */

EXEC SQL DISCONNECT;
```

Embedded SQL Example

Example (processing query)

```
scanf("%d", &yr);
EXEC SQL DECLARE c_query CURSOR FOR
    SELECT TITLE, SCORE FROM MOVIE
      WHERE (YR = :yr);
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;
    printf("%s %s\n", title, score);
    title = score = NULL;
}
free(title);
free(score);
```


ODBC

- ODBC: Open DataBase Connectivity:
  a service layer between the application and the server
- pros and cons:
  - standard
  - very 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>:<otherParameters>
- 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 Package

- import the JDBC package:
  `import java.sql.*`
- exception about SQL operations:
  SQLException

JDBC Connection

- DriverManager: connection manager
  - getConnection (static)
- Connection: database connection
  - createStatement

JDBC Queries

- Statement: SQL statement
  - executeQuery: for query operations
  - executeUpdate: for insert/update/delete operations
- PreparedStatement: precompiled SQL statement
  - can be used multiple times
  - placeholder for parameters: ?
  - set value before invoking query
- ResultSet: set of query results
  - next: next element in the result set
  - methods for getting the data in appropriate type

Java and SQL Data Types

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 (checking 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 (executing query)

```
Statement stmt = conn.createStatement();
String query = "SELECT TITLE, SCORE FROM MOVIE"
    + " WHERE (YR = 1990)";
ResultSet result = stmt.executeQuery(query);
while (result.next())
    System.out.println(result.getString(1)
            + " " + result.getFloat(2));
```


JDBC Example

Example (prepared statement)

```
String query = "DELETE FROM MOVIE" +
    " WHERE (SCORE < ?)";
PreparedStatement stmt =
    conn.prepareStatement(query);
stmt.setFloat(1, score);
stmt.executeUpdate();
```


References

Required text: Ramakrishnan, Gehrke

- Chapter 6: Database Application Development
  - 6.1. Accessing Databases from Applications
  - 6.2. An Introduction to JDBC
  - 6.3. JDBC Classes and Interfaces

Supplementary text: Date

- Chapter 4: An Introduction to SQL
  - 4.6. Embedded SQL

Stored Procedures

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

  → implement business logic on the application server

Creating Functions

SQL Statement

```
CREATE FUNCTION function_name([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

SQL 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 (keep a "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 (keep a "points" column)

```
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 application programs from changes in database structure
  - denormalizing after database refactoring

Creating Views

SQL 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 are needed

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)
```


Snapshots

- similar to view but creates a new base table
  - the query is executed only once when snapshot is created
- changes to base tables do not effect the snapshot
- no updates on the snapshot

Creating Snapshots

SQL Statement

```
CREATE SNAPSHOT snapshot_name AS
  SELECT ...
```


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 (granting permissions on a view)

```
GRANT SELECT ON NEW_MOVIES 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 can speed up queries
  - slows and insert and update operations
  - every key definition creates an index

Indexes

SQL Statement

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

References

Required text: Date

- Chapter 9: Integrity
  - 9.11. Triggers (a Digression)
- Chapter 10: Views
- Chapter 17: Security
  - 17.1. Introduction
  - 17.6. SQL Facilities