In the Lecture Series Introduction to Database Systems

SQL and Programming Languages

Presented by Stéphane Bressan and Gillian Dobbie

SQL and programming languages

- Procedural SQL,
- Database Connectivity, and
- Embedded SQL
- PL SQL,
- JDBC, and
- SQLj

SQL alone is not enough

- SQL is not **computationally complete**

  Computational completeness is the ability to express all possible computations

- SQL is not **system complete**

  System completeness is the ability to access and control the resources of the computer

Incomplete by design

SQL incompleteness makes it a simpler language

- Easy to use (for the programmer)
- Easy to optimize (for the DBMS)

Example: flight connections

flight

number	departure	arrival	origin	destination
AF235	10:15:00 AM	12:10:00 PM	CDG	VCE
GA826	1:00:00 PM	2:00:00 PM	CGK	SIN
MH364	2:30:00 PM	3:30:00 PM	SIN	LGK
NZ4319	11:00:00 PM	2:15:00 PM	LHR	SIN
SQ5051	11:55:00 AM	1:00:00 PM	HKT	SIN
SQ866	1:30:00 PM	5:00:00 PM	SIN	HKG
TG401	3:00:00 PM	5:25:00 PM	SIN	BKK
TG952	11:55:00 PM	7:00:00 AM	BKK	CDG

Example: flight connections

```
SELECT flight.destination
  FROM flight
  WHERE flight.origin='CGK'
```

number	departure	arrival	origin	destination
AF235	10:15:00 AM	12:10:00 PM	CDG	VCE
GA826	1:00:00 PM	2:00:00 PM	CGK	SIN
MH364	2:30:00 PM	3:30:00 PM	SIN	LGK
NZ4319	11:00:00 PM	2:15:00 PM	LHR	SIN
SQ5051	11:55:00 AM	1:00:00 PM	HKT	SIN
SQ866	1:30:00 PM	5:00:00 PM	SIN	HKG
TG401	3:00:00 PM	5:25:00 PM	SIN	BKK
TG952	11:55:00 PM	7:00:00 AM	BKK	CDG

Example: flight connections

```
SELECT f2.destination
  FROM flight f1, flight f2
  WHERE f1.origin='CGK'
  AND f2.origin=f1.destination
```

number	departure	arrival	origin	destination
AF235	10:15:00 AM	12:10:00 PM	CDG	VCE
GA826	1:00:00 PM	2:00:00 PM	CGK	SIN
MH364	2:30:00 PM	3:30:00 PM	SIN	LGK
NZ4319	11:00:00 PM	2:15:00 PM	LHR	SIN
SQ5051	11:55:00 AM	1:00:00 PM	HKT	SIN
SQ866	1:30:00 PM	5:00:00 PM	SIN	HKG
TG401	3:00:00 PM	5:25:00 PM	SIN	BKK
TG952	11:55:00 PM	7:00:00 AM	BKK	CDG

Example: flight connections

can be expressed in SQL 99, not SQL 92

```
WITH RECURSIVE
  connection ( origin, destination ) AS
  ( ( SELECT origin, destination
      FROM flight )
  UNION
    ( SELECT f.origin AS origin,
             c.destination AS destination
      FROM flight f, connection c
      WHERE c.origin = f.destination ) )

SELECT destination
  FROM connection
  WHERE origin = 'CGK'
```


Example: flight connections

number	departure	arrival	origin	destination
AF235	10:15:00 AM	12:10:00 PM	CDG	VCE
GA826	1:00:00 PM	2:00:00 PM	CGK	SIN
MH364	2:30:00 PM	3:30:00 PM	SIN	LGK
NZ4319	11:00:00 PM	2:15:00 PM	LHR	SIN
SQ5051	11:55:00 AM	1:00:00 PM	HKT	SIN
SQ866	1:30:00 PM	5:00:00 PM	SIN	HKG
TG401	3:00:00 PM	5:25:00 PM	SIN	BKK
TG952	11:55:00 PM	7:00:00 AM	BKK	CDG

In Summary

We need to connect SQL and programming languages

Coupling approaches

- External coupling → Code runs outside DBMS
- Internal coupling → Code runs inside DBMS

Three solutions

- Procedural SQL
- Database Connectivity
- Embedded SQL

Program structure

- Connection (external coupling) Open
- Statement preparation
- Statement execution
- Collection of results
- Exception handling
- Disconnection (external coupling) Close

Statement preparation

- **Prepare data structures** for communication between program and DBMS
- Request DBMS to **Compile SQL statement** (if possible)

Statement execution

- **Static** SQL statements
  Written in the program source
- **Dynamic** SQL statements
  Build at runtime

Collection of results

- INSERT, DELETE, UPDATE statements return single values
- SELECT statements return a collection of t-uples

Cursor

If result is too big or its size unknown, a **cursor** is used to fetch individual t-uples, one at the time, while the result is kept in the database

↳ Cursor is a pointer to one of the tuples in a result held by the DBMS
↳ Associated with evaluation of query

↳ Can be fetched and cast into main program's data type

Transactions and exception handling

- Transactions:
  - Begin
  - Commit
  - Rollback
- Exception
  - Catch

Running example

```
CREATE TABLE employee
 (name VARCHAR(24) PRIMARY KEY,
  address VARCHAR(36)
            DEFAULT 'company address',
  department VARCHAR(24)
            REFERENCES department(name),
  salary NUMERIC);

CREATE TABLE department
 (name VARCHAR(24) PRIMARY KEY,
  location VARCHAR(36),
  budget NUMERIC);
```


Coupling

Procedural SQL

Most DBMS support a procedural extension of SQL

PL/SQL is the procedural extension of Oracle 9i

Stored procedures are written in PL/SQL

Structure of a PL/SQL program

```
DECLARE
   /* Declarative section: variables, types, and local subprograms. */
BEGIN
   /* Executable section: procedural and SQL statements go here. */
   /* This is the only section of the block that is required. */
   EXCEPTION
   /* Exception handling section: error handling statements go here. */
END;
```


Preparation

Declaration part.

```
DECLARE
 salary NUMERIC NOT NULL;
 credit_bound  NUMERIC := 3000000;
 date DATE;
 employee_number NUMERIC := 0;
 manager BOOLEAN;
```


Execution

```
DECLARE
my_department_name VARCHAR;
BEGIN
 my_department_name := 'sales';
 SELECT name
 FROM employee
 WHERE department = my_department_name;
END;
```

Variable

Hold name of dept 'sales'

Execution: variables

```
DECLARE
 my_salary NUMERIC;
BEGIN
 SELECT salary INTO my_salary WHERE name = 'Nancy
  Santi';
END;
```

Assigns salary of Nancy Santi into my_salary

Execution: control

```
DECLARE
  sales NUMERIC;
  sal NUMERIC;
BEGIN
    [...]
    IF sales > 50000 THEN sal := 1500;
          ELSIF sales > 35000 THEN sal := 1200;
          ELSE sal := 1000;
    END IF;
    INSERT INTO employee
    VALUES ('Tiziana Dezza', '132, via Dellatti',
                                                'research', sal);
END;
```


Preparation: cursor

Cursor name

```
CURSOR high_budget IS
  SELECT name, budget
  FROM department
  WHERE budget > 10000;
```

SQL query attached to cursor

Execution: cursor

- Open

  Open cursor.

  ```
  OPEN high_budget;
  ```

- Fetch

  ```
  FETCH [orientation] FROM high_budget
  INTO name, budget;
  ```

  (orientation is one of NEXT, PRIOR, FIRST, LAST,…)

- Close

  Close cursor

  ```
  CLOSE high_budget;
  ```


Execution: cursor

```
DECLARE
name VARCHAR(24);
budget NUMERIC;

CURSOR high_budget IS
  SELECT name, budget
  FROM department
  WHERE budget > 10000;

BEGIN
    OPEN high_budget;

    LOOP
          FETCH NEXT FROM high_budget INTO name, budget;
          EXIT WHEN high_budget%NOTFOUND;
          dbmsoutput.put_line('Name: '|| name);
          dbmsoutput.put_line('  Budget: '|| budget);
    END LOOP;

    CLOSE high_budget;
END;
```


Execution: dynamic statements

```
EXECUTE IMMEDIATE dm;
```

can be executed @ runtime

Transactions

A transaction

- Starts implicitly
- Ends with execution, or
- COMMIT, or
- ROLLBACK

Exception Handling

Handle integrity constraint violations

```
DECLARE
  sum_bud NUMERIC;
  avg_bud NUMERIC;
BEGIN
  SELECT SUM(budget) INTO sum_bud FROM
    department;
  IF sum_bud = 0  THEN RAISE ZERO_BUDGET END IF;
  SELECT sum_bud/COUNT(*) INTO avg_bud FROM
    employee;
 EXCEPTION
    WHEN ZERO_DIVIDE
        THEN PRINT "Division by Zero";
    WHEN ZERO_BUDGET
        THEN PRINT "No budget has been assigned yet";
END;
```


Coupling 2

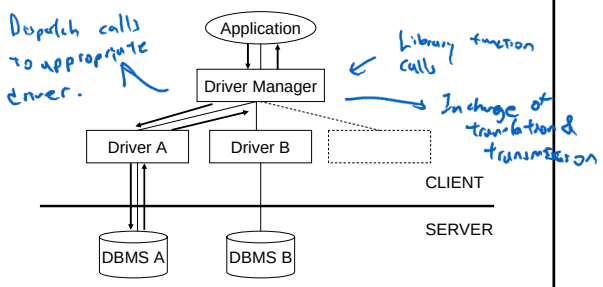
Database connectivity

- Database connectivity is a programming interface called "Call Level Interface"
- The CLI is implemented as a library for the host language
- SQL-99 specifies how CLIs must be defined

Database connectivity

- ODBC:
  Open Database Connectivity
- JDBC:
  Java Database Connectivity

Database connectivity architecture



Application comm with driver manager.

Connection

Driver specification:

Type 1 Driver

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Connection:

```
Connection con =
DriverManager.getConnection("jdbc:odbc:myDB",
                            "myLogin", "myPassword");
```

Open connections must always close in the end

Preparation and execution

SQL statements are strings

- They are dynamic statements

```
Statement stmt = con.createStatement();
stmt.executeQuery("SELECT * FROM employee");
```

- They can be 'prepared'

```
PreparedStatement pstmt =
con.prepareStatement("SELECT * FROM employee");
pstmt.executeQuery();
```

Preparation and execution

Prepared Statements can be parameterized

```
PreparedStatement pstmt =
  con.prepareStatement("SELECT *
                        FROM employee
                        WHERE department = ?");
pstmt.setString(1, 'research');
pstmt.executeQuery();
```

- *Parameters identifed by '?'*
- *Values assigned to parameters by 'setXXX' methods*
- *Parameters identified by their position in the string*

Execution: cursor

```
Statement selstmt = con.createStatement();

String stmt =  "SELECT *
                FROM employee
                WHERE salary > 1000";
ResultSet  res = selstmt.executeQuery (stmt);

while ( res.next() )
 {
   System.out.println (res.getString ("salary"));
 }
```

Move alot within results.

Coupling 3

Transactions and exception handling

- Auto commit:
  every connection is a transaction
- Auto commit can be turned off:
  explicit commit or abort
- Exceptions can be caught:
  java.sql.SQLException

Embedded SQL

Embedded SQL is a form of external coupling

Programming languages like C, Pascal, Fortran or Cobol are extended to the execution of SQL statements

- prefix is keyword **EXEC SQL**
- suffix is the semi-colon

SQLj is an ANSI/ISO for embedding SQL into Java

- prefix is **#sql**
- suffix is the semi-colon

Connection

```
oracle.connect("jdbc:odbc:my_DB",
               "my_login",
               "my_password");
```

↳ Connection established.

Execution

```
#sql { INSERT INTO employee VALUES (
     'Stefano Olivaro', 'Piazzale Roma',
     'research',1500) };
```

Execution: variables

Java inside SQL.

```
String department = 'research';
 BigDecimal my_budget;
 #sql { SELECT budget INTO :my_budget
        FROM department
        WHERE name = :department };
```


Execution: result and cursor

↳ if result returns > 1 tuple.

```
#sql iterator EmpIter (String name, Real salary);
EmpIter my_empiter = null;
#sql my_empiter = {SELECT name, salary FROM
   employee};
while (my_empiter.next())
   {
  System.out.println("Name: " + my_empiter.name());
  System.out.println("Salary: " + my_empiter.salary());
   }
my_empiter.close();
```


Transactions and exception handling

- Explicit COMMIT and ABORT
- Auto commit can be turned on
- Exceptions can be caught

In conclusion

- Procedural SQL, → PL SQL,
- Database Connectivity, and → JDBC, and
- Embedded SQL → SQLj

Credits

The content of this lecture is based on chapter 6 of the book "Introduction to database Management Systems" By S. Bressan and B. Catania, McGraw Hill publisher

Animated characters are animated using VocaliseTTS under license from Digital Curiosty

Clipart and media are licensed from Microsoft Office Online Clipart and Media

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc.

JDBC is a trademark of Sun Microsystems, Inc.

PL/SQL is a trademark of Oracle Corporation

Oracle 9i is a trademark of Oracle Corporation