7 Application Programming

7.1 Connecting SQL with Programming Languages
7.2 PostgreSQL and JDBC
7.3 Oracle Spatial and JDBC
7.4 Processing of GML-based Data
7.5 Summary

[KGB11]
• Even with spatial extensions core SQL is not computational complete
• However, numerous applications (e.g. shape simplifications) are impossible to solve (or very difficult) without computational completeness
• Connecting SQL with programming languages is needed for geometric/spatial applications
• Current SQL programming interfaces are
  – Embedding
  – Call level interface

• Both interfaces have to bridge the "impedance mismatch" (record oriented programming language vs. set oriented database language)
7.1 Connecting SQL with Programming Languages

- **Embedding (Embedded SQL)**
  - Both languages (SQL and programming language) remain unchanged to a large extent
  - SQL statements are pre-fixed with the keyword `EXEC SQL`
  - A precompiler parses and compiles the SQL statements
  - Variables of the programming language which are occurring in SQL statements are marked by `:`
  - SQL communication area contains status information about results of SQL statement
Example (embedded SQL and C)

/* C-includes for input/output and string processing */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* include for interacting with the database */
#include <sqlenv.h>

/* declaration of the SQL communication area */
EXEC SQL INCLUDE SQLCA;

main()
{
    /* declaration of C variables used in SQL statements */
    EXEC SQL BEGIN DECLARE SECTION;
    char name[20];
    char department[20];
    EXEC SQL END DECLARE SECTION;
7.1 Connecting SQL with Programming Languages

/* cursor declaration for query */
EXEC SQL DECLARE C1 CURSOR FOR
SELECT name, department
FROM professors
ORDER BY name;
/* connect to the database and open the cursor */
EXEC SQL CONNECT TO unidb;
EXEC SQL OPEN C1;

/* while there are tuples available... */
while(strncmp(sqlca.sqlstate, "02000", 5))
{
    /* assign tuple attributes to C variables */
    /* and write out values of the variables */
    EXEC SQL FETCH C1 INTO :name, :department;
    printf("%s %s\n", name, department);
}
7.1 Connecting SQL with Programming Languages

/* close cursor and close database connection */
EXEC SQL CLOSE C1;
EXEC SQL DISCONNECT CURRENT;
return;

Ahn mathematics
Balke computer science
Eick mathematics
Fekete computer science
Kemnitz mathematics
Langemann mathematics
Sonar mathematics
Wolf computer science
Zimmermann mathematics
• Embedded SQL also allows construction of queries at runtime (Dynamic SQL)
  – The query is constructed as value of an application program’s text variable
  – The PREPARE statement translates the query into executable object code and assigns it to an SQL variable of type STATEMENT
  – EXECUTE executes the query
  – At runtime schema information is made available with the DESCRIBE command
7.1 Connecting SQL with Programming Languages

- SQLJ was the direct adaptation of embedded SQL to Java
  - Embedding of (static) SQL statements into Java code
  - Embedded SQL statements can be checked syntactically and semantically at compile time
  - SQL statements are pre-fixed with "#sql"
  - As a result of a query an SQLJ iterator is generated
7.1 Connecting SQL with Programming Languages

• Call Level Interface, CLI
  – In application programs database functions are realized by means of external procedures (methods)
  – Rough classification yields 6 groups
    • Logon/logoff, password/identification
    • Declaration of program variables
    • Compile/execute SQL statements
    • Result processing
    • Error handling
    • Transaction management

/* Start the transaction */
new.operation = SQL_TXN_CREATE
rtnc=SQLSetConnectAttr(hdbc,SQL_TXN_CREATE,4,0,0,0,0,
                         SQL_TXN_READ_COMMITTED,SQL_TXN_DEFAULT,SQL_TXN_DEFAULT,0); /* Do some CLI work */
rtnc=SQLAllocHandle(SQL_HANDLE_STMT,stmt,hdbc,SQL_HANDLE_STMT); /* set the integer */
rtnc=SQLAllocHandle(SQL_HANDLE_STMT,stmt,hdbc,SQL_HANDLE_STMT);
rtnc=SQLPrepare(hstmt,sqlstr,SQL_HANDLE_STMT,SQL_HANDLE_STMT,0);
rtnc=SQLBindParameter(hstmt,1,1,SQL_BLOB,SQL_HANDLE_STMT,8192,SQL_PARAM_INPUT,0,0,0,
                      0,0,0); /* set the integer */
rtnc=SQLExecute(hstmt);
if (rtnc!=SQL_SUCCESS)
{
    printf("SQLExecute failed\n"%i is the SQLCODE\n",rtnc);
    printf("%i is the length of error\n",rtnc);
    printf("%s is the state\n",SQLSTATE); buffer);
}
else
    printf("SQLExecute success\n"%i is the SQLCODE\n",rtnc);
• Current call interface for Java: JDBC (Java Database Connectivity)
• Relies on Java base classes
• Provides methods for accessing any (relational) database from Java applications

http://www.developersbook.com
7.1 Connecting SQL with Programming Languages

• **Typical sequence**
  
  – **Establishing a connection to the database**
    
    • Specifying the connection information
    
    • Selecting and loading of the driver
  
  – **Sending an SQL statement**
    
    • Definition of the statement
    
    • Assignment of parameters
  
  – **Processing the query results**
    
    • Navigating in the resultset
    
    • Accessing attributes
7.1 Connecting SQL with Programming Languages

- **JDBC example**

```java
import java.sql.*;

public class Sample {

    public static void main(String args[]) {
        ResultSet rs=null;
        Statement stmt=null;
        Connection con=null;
        try {
            // load driver for postgresql:
            Class.forName("org.postgresql.Driver");

            // connect to the database 'sample' on postgresql
            // at the site it177.idb.cs.tm-cu.de with port 4742
            con=DriverManager.getConnection(
                "jdbc:postgresql://it177.idb.cs.tm-cu.de:4742/sample",
                "mueller", // username
                "pwml1er"); // password
        }
    }
}
```
// create a new statement
stmt=con.createStatement();

// select name, zip code, and cities of all persons (ordered by name)
rs=stmt.executeQuery("SELECT name, zipCode, city"+
    "FROM person"+
    "ORDER BY name");

// display result on screen
if (!rs.next()) { // no result
    System.out.println("Who deleted the data?\n");
} else { // normal case ...
    System.out.println("name    zip    city");
    do {
        String name=rs.getString("name")+" ";
        System.out.println(name.substring(0,25)+" | "+
            rs.getInt("zipCode")+" " + rs.getString(3));
    } while (rs.next());
}
7.1 Connecting SQL with Programming Languages

```java
} catch (ClassNotFoundException c) {
    System.err.println("A class could not be found" +
    "(Error message:" + c.getMessage() + "")
} catch (SQLException se) {
    System.err.println("There was an (unexpected?) SQL Error!" +
    se.getMessage() + "");
}

// close of ResultSet, Statement, Connection
try { rs.close(); } catch (Exception ex) { }
try { stmt.close(); } catch (Exception ex) { }
try { con.close(); } catch (Exception ex) { }
```

PostgreSQL is a relational database system with spatial data types and corresponding SQL extensions (see chapter 4.5)

Following data types are offered

- Point (float, float)
- Box (point, point)
- Lseg (point, point)
- Path (point₁, point₂, ..., pointₙ)
- Polygon (point₁, point₂, ..., pointₙ)
- Circle (point, float)
7.2 PostgreSQL and JDBC

• Numerous geometric predicates and functions
• Spatial data types can be used for attributes in the usual way
• Values of spatial data types are listed as appropriate character strings

CREATE TABLE buildings
  (id CHAR(25),
   typeOfUse CHAR(25),
   groundPlan POLYGON(50),
   PRIMARY KEY(id));

INSERT INTO buildings VALUES
  ('B4211','police',
   '((12125,1333),(13430,1560),(13260,2497),
    (14111,2695),(14111,2638),(16040,3092),
    (15303,6468),(13345,5958),(13771,3943),
    (12948,3773),(12948,3887),(11671,3631))');
• For processing geometric attributes special Java classes and methods are offered

• They are subclasses of class "PGobject"

• Subclasses are PGpoint, PGbox, PGlseg, PGline, PGpolygon, PGcircle

• Each subclass includes the following methods
  – Constructor
  – equals compares two objects of the same type
  – getValue returns the object as a string
Example: constructor for circle

```java
/**
 * @param c PGpoint describing the circle's center
 * @param r radius of circle
 */
public PGcircle(PGpoint c, double r)
{
    this();
    this.center = c;
    this.radius = r;
}
```
Example: equals for polygon

```java
/**
 * @param obj Object to compare with
 * @return true if the two polygons are identical
 */
public boolean equals(Object obj) {
    if (obj instanceof PGpolygon) {
        PGpolygon p = (PGpolygon)obj;
        if (p.points.length != points.length)
            return false;
        for (int i = 0; i < points.length; i++)
            if (!points[i].equals(p.points[i]))
                return false;
        return true;
    }
    return false;
}
```
• A given (abstract) polygon with n points has $2^n$ representations.

• A normal form is desirable.
7.2 Polygon Normal Form

• A normal form (canonical form, standard form) of an object is a standard way of representing that object as an expression.

• Every object has a unique representation.

• To test whether two objects are equivalent, it suffices to test their normal forms for equality.
7.2 Polygon Normal Form

• Examples for normal forms
  – Boolean logic:
    disjunctive normal form, conjunctive normal form
  – Formal language theory:
    Chomsky normal form for context-free grammars
  – Predicate calculus:
    prenex normal form
  – Data base design:
    several relational normal forms
For polygons the starting point and the direction of rotation have to be specified.

Starting point

- Choose point $P_{xl}$ with lowest x-coordinate.
- If multiple $P_{xl}$ exist, choose the one with lowest y-coordinate.
7.2 Polygon Normal Form

• Direction of rotation
  – Choose clockwise direction
    • Determine the two slopes between the point $P_{xl}$ and its two possible successors
    • Choose the point with the largest slope
7.2 Polygon Normal Form

• Examples

\[(2430, 1490)\]
\[(2610, 2930)\]
\[(4680, 2525)\]
\[(7920, 1940)\]
\[(7695, 635)\]
\[(6030, 905)\]
\[(761, 3151)\]
\[(2733, 3221)\]
\[(4032, 3292)\]
\[(5212, 2362)\]
\[(5550, 1475)\]
\[(4212, 1207)\]
\[(3212, 954)\]
\[(2874, 1644)\]
\[(1156, 1292)\]
\[(973, 2193)\]
• Example: getValue for polygon

```java
/**
 * @return the PGpolygon in the syntax expected
 * by org.postgresql
 */
public String getValue()
{
    StringBuffer b = new StringBuffer();
b.append("(");
    for (int p = 0;p < points.length;p++)
    {
        if (p > 0)
            b.append(",");
b.append(points[p].toString());
    }
b.append(")");
    return b.toString();
}
```
• For transferring non spatial tuple attributes into Java variables there are specific methods (of the class ResultSet), including:
  – getBoolean, getByte, getDate, getDouble, getFloat, getInt, getObject, getString, getTime

• However, there are no methods
  – getPoint, getBox, getLseg, getLine, getPolygon, getCircle

• For transferring spatial tuple attributes therefore the method "GetObject" is used with casting the result to an appropriate sub-class of "PGobject"
• Example of an application program with spatial data types, PostgreSQL, and JDBC

```java
import java.sql.*;
// the spatial data types point and circle are used
import org.postgresql.geometric.PGpoint;
import org.postgresql.geometric.PGcircle;

public class GeometricTest {
    // the main method with
    // - load driver for postgresql
    // - connect to the database 'test' on postgres on server localhost port 5432
    // - create table geomtest
    // - methods insertCircle and retrieveCircle are called
    // - disconnect and close the database
    public static void main(String args[]) throws Exception {
        Class.forName("org.postgresql.Driver");
        String url = "jdbc:postgresql://localhost:5432/test";
```
Connection conn = DriverManager.getConnection(url,"test","");
Statement stmt = conn.createStatement();

// Table geomtest only has the single attribute mycirc
// with the data type circle
stmt.execute("CREATE TEMP TABLE geomtest(mycirc circle)");
stmt.close();
insertCircle(conn);
retrieveCircle(conn);
conn.close();
}

// method insertCircle inserts a circle into the
// table mycirc (one tuple with one attribute value)
private static void insertCircle(Connection conn) throws SQLException {
    // construction of the circle with center and radius
    PGpoint center = new PGpoint(1, 2.5);
double radius = 4;
Pgcircle circle = new Pgcircle(center, radius);
// prepare of insert with "?" as parameter
PreparedStatement ps = conn.prepareStatement("INSERT INTO geomtest(mycirc) VALUES (?)");
// set the parameter and execute the statement
ps.setObject(1, circle);
ps.executeUpdate();
ps.close();
}

// method retrieveCircle retrieves a circle with
// its calculated area from the table mycirc
private static void retrieveCircle(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    // query for the circle and its area
    ResultSet rs = stmt.executeQuery("SELECT mycirc, area(mycirc) FROM geomtest");
    rs.next();
    // fetch data of circle via GetObject and cast on PGcircle
    PGcircle circle = (PGcircle)rs.getObject(1);
// fetch calculated area of circle via getDouble
double area = rs.getDouble(2);

// extract center and radius
PGpoint center = circle.center;
double radius = circle.radius;

// output of all data
System.out.println("Center (X,Y) = (" + center.x + ", " + center.y + ")");
System.out.println("Radius = " + radius);
System.out.println("Area = " + area);
• Extension of the Oracle database system
• One single geometric data type, which is very specific (see chapter 4.5)

```sql
CREATE TYPE sdo_geometry AS OBJECT (
  SDO_GTYPE NUMBER,
  SDO_SRID NUMBER,
  SDO_POINT SDO_POINT_TYPE,
  SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,
  SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```
7.3 Oracle Spatial and JDBC

• Overview of the components of Oracle Spatial
7.3 Oracle Spatial and JDBC

• Spatial data type can be used for attributes in the usual way

• Values of spatial data type are listed as appropriate character strings

CREATE TABLE buildings
(id CHAR(25),
typeOfUse CHAR(25),
groundPlan SDO_GEOMETRY,
PRIMARY KEY(id));

INSERT INTO buildings
VALUES ('B4211', 'police',
SDO_GEOMETRY(SDO_GTYPE=1003,
SDO_SRID=NULL, SDO_POINT=NULL,
SDO_ELEM INFO = (1,1003,1),
SDO_ORDINATES=(12125,1333,13430,
1560,13260,2497,14111,2695,14111,
2638,16040,3092,15303,6468,13345,
5958,13771,3943,12948,3773,12948,
3887,11671,3631)));

Spatial Databases and GIS – Karl Neumann, Sarah Tauscher – Ifis – TU Braunschweig
632
• For processing geometric attributes the special Java class "JGeometry" is provided
• JGeometry maps the SDO_GEOMETRY data type (STRUCT) to a more convenient class

[KGB11]
Numerous basic methods for accessing spatial information are offered, including

- `createCircle`, `createPoint`, `createLinearPolygon`
- `equals`
- `getDimensions`, `getElemInfo`, `getFirstPoint`, `getJavaPoint`, `getJavaPoints`, `getLastPoint`, `getNumPoints`, `getOrdinatesArray`, `getSize`, `getType`
- `isCircle`, `isPoint`, `isRectangle`
- `load`, `store`
- `setType`, `toString`
7.3 Oracle Spatial and JDBC

• Generic example

```java
// read geometry from database
ResultSet rs = statement.executeQuery(
    "SELECT geometry FROM states WHERE name = 'Florida'";
STRUCT st = (oracle.sql.STRUCT) rs.getObject(1);
// convert STRUCT to JGeometry
    JGeometry j_geom = JGeometry.load(st);

// ... manipulate geometry, or create new geometry...

// prepare to write back geometry to database
PreparedStatement ps = connection.prepareStatement(
    "UPDATE states SET geometry=? WHERE name='Florida'";
// convert JGeometry to STRUCT
    STRUCT obj = JGeometry.store(j_geom, connection);
    ps.setObject(1, obj);
    ps.execute();
```
7.4 Processing of GML-based Data

- GML is an important XML-based standard for exchanging spatial data (see chapter 6.3)
- Reading, processing, and output of GML data is a typical sequence of tasks in GIS context
- Example: placement of symbol signatures in GML-coded polygons of buildings (see chapter 3.4)
7.4 Processing of GML-based Data

- Processing of XML documents with Java is frequently done via SAX or DOM
  - **SAX (Simple API for XML)**
    - Event based programming interface
    - Provides methods for detecting of "document events" via event handler (e.g. startDocument, startElement, characters)
  - **DOM (Document Object Model)**
    - Object based programming interface
    - Provides classes for handling documents (including Node, NodeList, Element, Text)

http://www.xml.com/
• Using the Document Object Model is reasonable when
  – Randomly accessing the document
  – Transforming the document
  – Navigating in the document tree

• Processing in two steps
  – Parsing the document and building an appropriately structured object
  – Navigating in the object and modifying the object
7.4 Processing of GML-based Data

• Methods of the DOM API offer
  – Navigation between the nodes of a document
  – Create, move, delete nodes
  – Read, modify, delete texts

• Central methods for navigating
  – boolean node.hasChildNodes indicates whether a node has children
  – Node node.getFirstChild gets first node of the list of children
7.4 Processing of GML-based Data

- String node.getNodeName gets the name of the node
- short node.getNodeType gets the type of the node

- **Central methods for expanding documents**
  - node.appendChild(Node newChild) appends new node
  - node.removeChild(Node oldChild) removes node
7.4 Processing of GML-based Data

- `node.replaceChild(Node newChild, Node oldChild)` replaces node
- `node.insertBefore(Node newChild, Node refChild)` inserts new node
- `node.setNodeValue(String nodeValue)` sets value of node
Xerces is a typical system for parsing XML documents and to build DOM representations

To work with Xerces following packages are needed

- import org.w3c.dom.*;
- import org.xml.sax.SAXException;
- import org.apache.xerces.parsers.DOMParser;

Next steps include creating an instance of the DOM parser, parsing the document of the input file, generating a parsed DOM object

- DOMParser parser = new DOMParser();
- parser.parse(name of the XML source file);
- Document doc = parser.getDocument();
The following (simplified) example shows a (drastically shortened) Java program for processing of GML-based buildings data:

- ALKIS' extracts of the land register primary data are available in GML format.
- Typically each extract contains about 10,000 objects (buildings, parcels, border points, etc.).
- When derivating the real estate map, for some buildings an additional placement of symbol signatures is needed.
7.4 Processing of GML-based Data

- Needed steps for placement of signatures
  - Parsing of buildings
  - Detect missing presentation object (see chapter 3.4)
  - Computing the optimal position
  - Create a new presentation object

```xml
<gm1:featureMember>
  <AP_PPO gml:id="DEBWL00100000fAW">
    <lebenszeitintervall> ... </lebenszeitintervall>
    <anlass>000000</anlass>
    <position>
      <gml:Point><gml:pos>3540847.175 5805897.864</gml:pos></gml:Point>
    </position>
    <signaturnummer>3316</signaturnummer>
    <drehwinkel>67.000</drehwinkel>
  </AP_PPO>
</gm1:featureMember>
```
7.4 Processing of GML-based Data

- The following example shows an excerpt from the method for parsing of buildings that are already converted into a DOM structure
  – An instance of the class "AX_Gebaeude" (simplified)

```xml
<AX_Gebaeude gml:id="DEHHSERV00001FN1">
  ...
  <position>
    <gml:Polygon>
      <gml:exterior>
        <gml:Ring>
          <gml:pos>3567807.047 5930017.550</gml:pos>
          ...
          <gml:pos>3567807.047 5930017.550</gml:pos>
        </gml:Ring>
      </gml:exterior>
    </gml:Polygon>
  </position>
  <gebaeudefunktion>2000</gebaeudefunktion>
  <weitereGebaeudefunktion>1170</weitereGebaeudefunktion>
  <bauweise>2100</bauweise>
  <anzahlDerOberirdischenGeschosse>1</anzahlDerOberirdischenGeschosse>
  <dachform>3100</dachform>
</AX_Gebaeude>
```
static public void parseBuildings ()throws SAXException, IOException, ParserConfigurationException {

    // get all buildings into a list;
    // an extract of the land register primary data already is available as "Root"
    NodeList BuildingsList =
    Root.getElementsByTagName("AX_Gebaeude");

    // processing of all buildings
    for(int k=0; k< BuildingsList.getLength();k++){
        Element Building = (Element) BuildingsList.item(k);

        // initialize function of building
        int buildfct = 0;

        // initialize further function of building (there may be several further functions)
        Vector<Integer> ffct = new Vector<Integer>();

        // read building-Id
        String ID = Building.getAttribute("gml:id");
    }
7.4 Processing of GML-based Data

// read function of building (may be empty)
NodeList fnNodeList = Building.getElementsByTagName("gebaeufunktion");
if(fnNodeList.getLength() != 0){
    Text fnText = (Text)((Element)fnNodeList.item(0)).getFirstChild();
    buildfct = Integer.parseInt(fnText.getNodeValue());
}

// read further function of building
fnNodeList = Building.getElementsByTagName("weitereGebaeufunktion");
if(fnNodeList.getLength() != 0){
    for(int l = 0; l<fnNodeList.getLength(); l++){
        Text fnText = (Text)((Element)fnNodeList.item(l)).getFirstChild();
        int fct = Integer.parseInt(fnText.getNodeValue());
        ffct.addElement(fct);
    }
}

// read further attributes of building;
// e.g. coordinates of the position node
...

} // end: all buildings
7.5 Summary

• Connecting SQL with programming languages
  – Embedded SQL
  – Call level interface
  – JDBC

• PostgreSQL and JDBC
  – Class “PGobject”
  – Subclasses “PGpoint”, “PGbox”
7.5 Summary

- **Oracle Spatial and JDBC**
  - Class “JGeometry”

- **Processing of GML-based data**
  - SAX (Simple API for XML)
  - DOM (Document Object Model)
  - Navigating the document tree
  - Example: GML-based data of buildings

```sql
CREATE TYPE sdo_geometry AS OBJECT ( 
  SDO_GTYPE NUMBER, 
  SDO_SRID NUMBER, 
  SDO_POINT SDO_POINT_TYPE, 
  SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY, 
  SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```
7.6 Summary

- GIS
  - collect
  - manage
  - analyse
  - display

- GML
  - objects
  - thematic
  - geometry
  - vector

- SQL + programming language
  - SQLJ
  - JDBC
  - embedded SQL