A query language, the principal means of interaction with the database, is a core requirement of a DBMS. A popular commercial query language for relational database management systems (RDBMS) is SQL. It is partly based on the formal query language, relational algebra (RA) and it is easy to use, intuitive, and versatile. Because SDBMSs are an example of an extensible DBMS and deal with both spatial and nonspatial data, it is natural to seek for an extension of SQL to incorporate spatial data.

As shown in the previous chapter, the relational model has limitations in effectively handling spatial data. Spatial data is “complex,” involving a melange of polygons, lines, and points, and the relational model is geared for dealing with simple data types such as integers, strings, dates, and so forth.

Constructs from object-oriented programming, such as user-defined types and data and functional inheritance, have found immediate applications in the modeling of complex data. The widespread use of the relational model and SQL for applications involving simple datatypes combined with the functionality of the object-oriented model has led to the birth of a new “hybrid” paradigm for database management systems, the OR-DBMS.

A corollary to this newfound interest in OR-DBMS is the desire to extend SQL with object functionality. This effort has materialized into a new OR-DBMS standard for SQL: SQL3. Because we are dealing with spatial data, we examine the spatial extensions and libraries for SQL3.

A unique feature of spatial data is that the “natural” medium of interaction with the user is visual rather than textual. Hence any spatial query language should support a sophisticated graphical-visual component. Having said that, we focus here on the non-graphical spatial extensions of SQL. In Section 3.1, we introduce the World database, which will form the basis of all query examples in the chapter. Sections 3.2 and 3.3 provide a brief overview of RA and SQL, respectively. Section 3.4 is devoted to a discussion on the spatial requirements for extending SQL. We also introduce the OGIS standard for
extending SQL for geospatial data. In Section 3.5, we show how common spatial queries can be posed in OGIS extended SQL. In Section 3.6, we introduce SQL3 and Oracle8’s implementation of a subset of SQL3.

3.1 STANDARD DATABASE QUERY LANGUAGES

Users interact with the data embedded in a DBMS using a query language. Unlike traditional programming languages, database query languages are relatively easy to learn and use. In this section we describe two such query languages. The first, RA, is the more formal of the two and typically not implemented in commercial databases. The importance of RA lies in the fact that it forms the core of SQL, the most popular and widely implemented database query language.

3.1.1 World Database

We introduce RA and SQL with the help of an example database. We introduce a new example database here to provide some diversity in examples and exercises. The World database consists of three entities: Country, City, and River. The pictogram-enhanced ER diagram of the database and the example tables are shown in Figure 3.1 and Table 3.1, respectively. The schema of the database is shown below. Note that an underlined attribute is a primary key. For example, Name is a primary key in Country table, City table, and River table.

```sql
Country(Name: varchar(35), Cont: varchar(35), Pop: integer,
        GDP: Integer, Life-Exp: integer, Shape: char(13))
City(Name: varchar(35), Country: varchar(35), Pop: integer,
     Capital: char(1), Shape: char(9))
River(Name: varchar(35), Origin: varchar(35), Length: integer,
      Shape: char(13))
```

![Figure 3.1. The ER diagram of the World database.](image)
Chapter 3 Spatial Query Languages

TABLE 3.1: The Tables of the World Database with Sample Records

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>Name</th>
<th>Cont</th>
<th>Pop (millions)</th>
<th>GDP (billions)</th>
<th>Life-Exp</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>NAM</td>
<td>30.1</td>
<td>658.0</td>
<td>77.08</td>
<td>69.36</td>
<td>Polygonid-1</td>
</tr>
<tr>
<td>Mexico</td>
<td>NAM</td>
<td>107.5</td>
<td>694.3</td>
<td>66.50</td>
<td></td>
<td>Polygonid-2</td>
</tr>
<tr>
<td>Brazil</td>
<td>SAM</td>
<td>183.3</td>
<td>1004.0</td>
<td>65.60</td>
<td></td>
<td>Polygonid-3</td>
</tr>
<tr>
<td>Cuba</td>
<td>NAM</td>
<td>11.7</td>
<td>16.9</td>
<td>75.95</td>
<td></td>
<td>Polygonid-4</td>
</tr>
<tr>
<td>USA</td>
<td>NAM</td>
<td>270.0</td>
<td>8003.0</td>
<td>75.75</td>
<td></td>
<td>Polygonid-5</td>
</tr>
<tr>
<td>Argentina</td>
<td>SAM</td>
<td>36.3</td>
<td>348.2</td>
<td>70.75</td>
<td></td>
<td>Polygonid-6</td>
</tr>
</tbody>
</table>

(a) Country

<table>
<thead>
<tr>
<th>CITY</th>
<th>Name</th>
<th>Country</th>
<th>Pop (millions)</th>
<th>Capital</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Havana</td>
<td>Cuba</td>
<td>2.1</td>
<td>Y</td>
<td></td>
<td>Poinitnd-1</td>
</tr>
<tr>
<td>Washington, D.C.</td>
<td>USA</td>
<td>3.2</td>
<td>Y</td>
<td></td>
<td>Poinitnd-2</td>
</tr>
<tr>
<td>Monterrey</td>
<td>Mexico</td>
<td>2.0</td>
<td>N</td>
<td></td>
<td>Poinitnd-3</td>
</tr>
<tr>
<td>Toronto</td>
<td>Canada</td>
<td>3.4</td>
<td>N</td>
<td></td>
<td>Poinitnd-4</td>
</tr>
<tr>
<td>Brasilia</td>
<td>Brazil</td>
<td>1.5</td>
<td>Y</td>
<td></td>
<td>Poinitnd-5</td>
</tr>
<tr>
<td>Rosario</td>
<td>Argentina</td>
<td>1.1</td>
<td>N</td>
<td></td>
<td>Poinitnd-6</td>
</tr>
<tr>
<td>Ottawa</td>
<td>Canada</td>
<td>0.8</td>
<td>Y</td>
<td></td>
<td>Poinitnd-7</td>
</tr>
<tr>
<td>Mexico City</td>
<td>Mexico</td>
<td>14.1</td>
<td>Y</td>
<td></td>
<td>Poinitnd-8</td>
</tr>
<tr>
<td>Buenos Aires</td>
<td>Argentina</td>
<td>10.75</td>
<td>Y</td>
<td></td>
<td>Poinitnd-9</td>
</tr>
</tbody>
</table>

(b) City

<table>
<thead>
<tr>
<th>RIVER</th>
<th>Name</th>
<th>Origin</th>
<th>Length (kilometers)</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rio Parana</td>
<td>Brazil</td>
<td>2600</td>
<td>LineStringid-1</td>
<td></td>
</tr>
<tr>
<td>St. Lawrence</td>
<td>USA</td>
<td>1200</td>
<td>LineStringid-2</td>
<td></td>
</tr>
<tr>
<td>Rio Grande</td>
<td>USA</td>
<td>3000</td>
<td>LineStringid-3</td>
<td></td>
</tr>
<tr>
<td>Mississippi</td>
<td>USA</td>
<td>6000</td>
<td>LineStringid-4</td>
<td></td>
</tr>
</tbody>
</table>

(c) River

The Country entity has six attributes. The Name of the country and the continent (Cont) it belongs to are character strings of maximum length thirty-five. The population (Pop) and the gross domestic product (GDP) are integer types. The GDP is the total value of goods and services produced in a country in one fiscal year. Life-Exp attribute represents the life expectancy in years (rounded to the nearest integer) for residents of a country. The Shape attribute needs some explanation. The geometry of a country is represented in the Shape column of Table 3.1. In relational databases, where the datatypes are limited, the Shape attribute is a foreign key to a shape table. In an object-relational or object-oriented database, the Shape attribute will be a polygon abstract datatype (ADT). Because, for the moment, our aim is to introduce basic RA and SQL, we will not query the Shape attribute until Section 3.4.

The City relation has five attributes: Name, Country, Pop, Capital, and Shape. The Country attribute is a foreign key into the Country table. Capital is a fixed character type of length one; a city is a capital of a country, or it is not. The Shape attribute is a foreign key into a point shape table. As for the Country relation, we will not query the Shape column before learning about OGIS data types for SQL3.

The four attributes of the River relation are Name, Origin, Length, and Shape. The Origin attribute is a foreign key into the Country relation and specifies the country where
the river originates. The \textit{Shape} attribute is a foreign key into a line string shape table. To determine the country of origin of a river, the geometric information specified in the \textit{Shape} attribute is not sufficient. The overloading of name across tables can be resolved by a qualifying attribute with tables using a dot notation table.attribute. County.Name, city.Name, and river.Name uniquely identify the Name attribute inside different tables. We also need information about the direction of the river flow. In Chapter 7 we discuss querying spatial networks where directional information is important.

3.2 RA

RA is a formal query language associated with the relational model. An \textit{algebra} is a mathematical structure consisting of two distinct sets of elements, \((\Omega_a, \Omega_o)\). \(\Omega_a\) is the set of \textit{operands} and \(\Omega_o\) is the set of \textit{operations}. An algebra must satisfy many axioms, but the most crucial is that the result of an \textit{operation} on an \textit{operand} must remain in \(\Omega_a\). A simple example of an algebra is the set of integers. The \textit{operands} are the integers, and the \textit{operations} are addition and multiplication. In Chapter 8 we discuss other kinds of algebra associated with raster and image objects.

In RA there is only one type of operand and six basic operations. The operand is a relation (table), and the six operations are \textit{select}, \textit{project}, \textit{union}, \textit{cross-product}, \textit{difference}, and \textit{intersection}. We now introduce some of the basic operations in detail.

3.2.1 The Select and Project Operations

To manipulate data in a single relation, RA provides two operations: \textit{select} and \textit{project}. The select operation retrieves a subset of rows of the relational table, and the project operation extracts a subset of the columns. For example, to list all the countries in the \textit{Country} table which are in \textit{North-America} (NAM), we use the following relational algebra expression:

\[ \sigma_{\text{cont}=\text{NAM}}(\text{Country}). \]

The result of this operation is shown in Table 3.2(a). The rows retrieved by the select operation \(\sigma\) are specified by the comparison selection operator, which in this example is \(\text{cont}=\text{'North-America'}.\) The schema of the input relation is not altered by the select operator. The formal syntax of the select operation is

\[ \sigma_{<\text{selection operator}>}(\text{Relation}). \]

Subsets of columns for all rows in a relation are extracted by applying the \textit{project} operation, \(\pi\). For example, to retrieve the names of all countries listed in the \textit{Country} table, we use the following expression:

\[ \pi_{\text{Name}}(\text{Country}). \]

The formal syntax of the project operation is

\[ \pi_{<\text{list of attributes}>}(\text{Relation}) \]

We can combine the select and the project operations. The following expression yields the names of countries in North America. See Table 3.2(c) for the result.

\[ \pi_{\text{Name}}(\sigma_{\text{Cont}=\text{NAM}}(\text{Country})) \]
Chapter 3  Spatial Query Languages

TABLE 3.2: Results of Two Basic Operations in RA Select and Project

<table>
<thead>
<tr>
<th>Name</th>
<th>Cont</th>
<th>Pop (millions)</th>
<th>GDP (billions)</th>
<th>Life-Exp</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>NAM</td>
<td>30.1</td>
<td>658.0</td>
<td>77.08</td>
<td>Polygonid-1</td>
</tr>
<tr>
<td>Mexico</td>
<td>NAM</td>
<td>107.5</td>
<td>694.3</td>
<td>69.36</td>
<td>Polygonid-2</td>
</tr>
<tr>
<td>Cuba</td>
<td>NAM</td>
<td>11.7</td>
<td>16.9</td>
<td>75.95</td>
<td>Polygonid-4</td>
</tr>
<tr>
<td>USA</td>
<td>NAM</td>
<td>270.0</td>
<td>8003.0</td>
<td>75.75</td>
<td>Polygonid-5</td>
</tr>
</tbody>
</table>

(a) Select

(b) Project

3.2.2 Set Operations

At its most fundamental level a relation is a set. Thus all set operations are valid operations in the relational algebra. Set operations are applied to relations that are union-compatible. Two relations are union-compatible if they have the same number of columns, share the same domain, and if the columns appear in the same order from left to right.

- **Union:** If $R$ and $S$ are relations, then $R \cup S$ returns all tuples which are either in $R$ or $S$. For example, we can use the union operation to list the countries which are either in North America or have a river originating in them:

  1. $R = \pi_{\text{Name}}(\sigma_{\text{Cont}=\text{NAM}}(\text{Country}))$
  2. $S = \pi_{\text{Origin}}(\text{River})$
  3. $R \cup S$.

The resulting relation is shown in Table 3.4(a). Notice that the attributes $R.\text{Name}$ and $S.\text{Origin}$ have the same domain, as $R.\text{Origin}$ refers to County.\text{Name}. This is sufficient for $R$ and $S$ to be union-compatible.

- **Difference:** $R - S$ returns all tuples in $R$ that are not in $S$. The difference operation can be used, for example, to list all countries in North America that have no river (listed in the River table) originating in them. The resulting relation is shown in Table 3.4(b).

  1. $R = \pi_{\text{Name}}(\sigma_{\text{Cont}=\text{NAM}}(\text{Country}))$
  2. $S = \pi_{\text{Origin}}(\text{River})$
  3. $R - S$. 
### TABLE 3.3: The Cross-Product of Relations $R$ and $S$

<table>
<thead>
<tr>
<th></th>
<th>$R$</th>
<th></th>
<th>$S$</th>
<th></th>
<th>$R \times S$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>(a) Relation $R$</td>
<td>$A_1$</td>
<td>$B_1$</td>
<td>$C_1$</td>
<td>$D_1$</td>
<td>$A_1$</td>
<td>$B_1$</td>
<td>$C_1$</td>
</tr>
<tr>
<td></td>
<td>$A_2$</td>
<td>$B_2$</td>
<td>$C_2$</td>
<td>$D_2$</td>
<td>$A_1$</td>
<td>$B_1$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>(b) Relation $S$</td>
<td></td>
<td></td>
<td>$C_1$</td>
<td>$D_1$</td>
<td>$A_2$</td>
<td>$B_2$</td>
<td>$C_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$A_2$</td>
<td>$B_2$</td>
<td>$C_2$</td>
</tr>
<tr>
<td>(c) $R \times S$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Intersection**: For two union-compatible relations $R$ and $S$, the intersection operation $R \cap S$ returns all tuples which occur both in $R$ and $S$. Note that this operation, though convenient, is redundant: it can be derived from the difference operation, $R \cap S = R - (R - S)$. To list countries that are in South America and also have an originating river, we use the intersection operation. The result is shown in Table 3.4(c).

1. $R = \pi_{\text{Name}}(\sigma_{\text{Cont=SAM}}(\text{Country}))$
2. $R = \pi_{\text{Origin}}(\text{River})$
3. $R \cap S$.

- **Cross-Product**: This operation applies to any pair of relations, not just those that are union-compatible. $R \times S$ returns a relation whose schema contains all the attributes of $R$ followed by those of $S$. For simplicity, an abstract example is shown in Table 3.3. Notice the use of the cascading dot notation to distinguish the attributes of the two relations.

#### 3.2.3 Join Operation

The select and project operations are useful for extracting information from a single relation. The **join** operation is used to query across different relational tables. A join operation can be thought of as a cross-product followed by the select operation. The general join operation is called the **conditional join**. An important and special case of the conditional join is called the **natural join**.

### TABLE 3.4: The Results of Set Operations

<table>
<thead>
<tr>
<th>NAME</th>
<th>NAME</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canada</td>
<td>Canada</td>
<td>Brazil</td>
</tr>
<tr>
<td>Mexico</td>
<td>Mexico</td>
<td>Cuba</td>
</tr>
<tr>
<td>Brazil</td>
<td>Cuba</td>
<td>USA</td>
</tr>
<tr>
<td></td>
<td>(a) Union</td>
<td>(b) Difference</td>
</tr>
</tbody>
</table>
Chapter 3 Spatial Query Languages

Conditional Joins

The general conditional join, \( \Join_c \), between two relations \( R \) and \( S \) is expressed as follows:

\[
R \Join_c S = \sigma_c(R \times S).
\]

The \( c \) condition usually refers to the attributes of both \( R \) and \( S \). For example, we can use the join operation to query for the names of the countries whose population is greater than Mexico’s (see Table 3.5):

1. \( R = \pi_{\text{Name, Pop}}(\text{Country}) \)
2. \( S = R. \) (\( S \) is duplicate copy of \( R \))
3. Form the cross-product \( R \times S \). The schema of the \( R \times S \) relation is

<table>
<thead>
<tr>
<th>( R \times S )</th>
<th>R.Name</th>
<th>R.Pop</th>
<th>S.Name</th>
<th>S.Pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Canada</td>
<td>30.1</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Mexico</td>
<td>107.5</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Brazil</td>
<td>183.3</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Cuba</td>
<td>11.7</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>USA</td>
<td>270.0</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Argentina</td>
<td>36.3</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

4. Apply condition, that is, the population of a country in relation \( S \) is greater than the population of Mexico.

\[
U = R \Join S = \sigma(R.\text{Name} = \text{‘Mexico’}) \land (R.\text{Pop} > S.\text{Pop})(R \times S)
\]

Natural Join

An important special case of the conditional join is the natural join. In a natural join, only the \textit{equality} selection condition is applied to the common attributes of the two relations and only one column in result represents common equi-join attribute. For example, a natural join can be used to find the populations of countries where rivers originate. The steps follow:

**TABLE 3.5:** Steps of the Conditional Join Operation

<table>
<thead>
<tr>
<th>( R \times S )</th>
<th>R.Name</th>
<th>R.Pop</th>
<th>S.Name</th>
<th>S.Pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Canada</td>
<td>30.1</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Mexico</td>
<td>107.5</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Brazil</td>
<td>183.3</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Cuba</td>
<td>11.7</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>USA</td>
<td>270.0</td>
<td></td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Argentina</td>
<td>36.3</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(a) A portion of \( R \times S \)

<table>
<thead>
<tr>
<th>R.Name</th>
<th>R.Pop</th>
<th>S.Name</th>
<th>S.Pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Canada</td>
<td>30.1</td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Cuba</td>
<td>11.7</td>
</tr>
<tr>
<td>Mexico</td>
<td>107.5</td>
<td>Argentina</td>
<td>36.3</td>
</tr>
</tbody>
</table>

(b) The select operation on \( R \times S \)
1. Rename the Country relation \( C \) and the River relation \( R \).
2. Form the cross-product \( C \times R \).
3. Join the two relations on the attributes \( C.Name \) and \( R.Origin \). The domains of these two attributes are identical,
   \[ C \times_{C.Name} R.Origin \]
4. In a natural join, the selection condition is unambiguous; therefore, it does not have to be explicitly subscripted in the join formula.
5. The final result is obtained by projecting onto the \( Name \) and \( Pop \) attributes:
   \[ \pi_{Name, Pop}(C \times R) \]

### 3.3 BASIC SQL PRIMER

SQL is a commercial query language first developed at IBM. Since then, it has become the standard query language for RDBMS. SQL is a declarative language, that is, the user only has to specify the answer rather than a procedure to retrieve the answer.

The SQL language has at least two separate components: the data definition language (DDL) and the data modification language (DML). The DDL is used to create, delete, and modify the definition of the tables in the database. In the DML, queries are posed and rows inserted and deleted from tables specified in the DDL. SQL also have other statements for data control language. We now provide a brief introduction to SQL. Our aim is to provide enough understanding of the language so that readers can appreciate the spatial extensions that we discuss in Section 3.4. A more detailed and complete exposition of SQL can be found in any standard text on databases [Elmasri and Navathe, 2000; Ullman and Widom, 1999].

#### 3.3.1 DDL

The creation of the relational schema and the addition and deletion of the tables are specified in the DDL component of SQL. For example, the City schema introduced in Section 3.2 is defined below in SQL. The Country and River tables are defined in Table 3.6.

```sql
CREATE TABLE CITY {
    Name VARCHAR(35),
    Country VARCHAR(35),
    Pop INT,
    Capital CHAR(1),
    Shape CHAR(13),
    PRIMARY KEY Name
}
```

The **CREATE TABLE** statement is used to define the relations in a relational schema. The name of the table is **CITY**. The table has four columns, and the name of each column and its corresponding datatype must be specified. The **Name** and **Country** attributes must be ASCII character strings of less than thirty-five characters. **Population**
TABLE 3.6: The Country and River Schema in SQL

| CREATE TABLE Country { Name VARCHAR(35), Cont VARCHAR(35), Pop INT, GDP INT, Shape CHAR(15), PRIMARY KEY (Name) } |
| CREATE TABLE River { Name VARCHAR(35), Origin VARCHAR(35), Length INT, Shape CHAR(15), PRIMARY KEY (Name) } |

(a) Country schema (b) River schema

is of the type integer, and Capital is an attribute which is a single character Y or N. In SQL92 the possible datatypes are fixed and cannot be user defined. We do not list the complete set of datatypes, which can be found in any text on standard databases. Finally, the Name attribute is the primary key of the relation. Thus each row in the table must have a unique value for the Name attribute. Tables no longer in use can be removed from the database using the DROP TABLE command. Another important command in DDL is ALTER TABLE for modifying the schema of the relation.

3.3.2 DML

After the table has been created as specified in DDL, it is ready to accept data. This task, which is often called “populating the table,” is done in the DML component of SQL. For example, the following statement adds one row to the table River:

```
INSERT INTO River(Name, Origin, Length)
VALUES('Mississippi', 'USA', 6000)
```

If all the attributes of the relation are not specified, then default values are automatically substituted. The most often used default value is NULL. An attempt to add another row in the River table with Name = ‘Mississippi’ will be rejected by the DBMS because of the primary key constraint specified in the DDL.

The basic form to remove rows from the table is as follows:

```
DELETE FROM TABLE WHERE < CONDITIONS >
```

For example, the following statement removes the row from the table River that we inserted above.

```
DELETE FROM River
WHERE Name = 'Mississippi'
```

3.3.3 Basic Form of an SQL Query

Once the database schema has been defined in the DDL component and the tables populated, queries can be expressed in SQL to extract relevant subsets of data from the database. The basic syntax of an SQL query is extremely simple:
Section 3.3  Basic SQL Primer  61

SELECT  column-names
FROM  relations
WHERE  tuple-constraint

This form is equivalent to the RA expression consisting of \( \pi, \sigma, \text{ and } \bowtie \). SQL SELECT statement has more clauses related to aggregation (e.g., \textit{GROUP BY}, \textit{HAVING}), ordering results (e.g., \textit{ORDER BY}), and so forth. In addition, SQL allows the formulation of nested queries. We illustrate these with a set of examples.

### 3.3.4 Example Queries in SQL

We now give examples of how to pose different types of queries in SQL. Our purpose is to give a flavor of the versatility and power of SELECT statement. All the tables queried are from the \textit{WORLD} example introduced in Section 3.1.1. The results of the different queries can be found in Tables 3.7 and 3.8.

1. **Query:** List all the cities and the country they belong to in the \textit{CITY} table.

   ```sql
   SELECT Ci.Name, Ci.Country
   FROM CITY Ci
   ```

   **Comments:** The SQL expression is equivalent to the \textit{project} operation in RA. The \textit{WHERE} clause is missing in the SQL expression because there is no

   \textbf{TABLE 3.7: Tables from the Select, Project, and Select and Project Operations}

<table>
<thead>
<tr>
<th>Name</th>
<th>Country</th>
<th>Pop(millions)</th>
<th>Capital</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Havana</td>
<td>Cuba</td>
<td>2.1</td>
<td>Y</td>
<td>Point</td>
</tr>
<tr>
<td>Washington, D.C.</td>
<td>USA</td>
<td>3.2</td>
<td>Y</td>
<td>Point</td>
</tr>
<tr>
<td>Brasilia</td>
<td>Brazil</td>
<td>1.5</td>
<td>Y</td>
<td>Point</td>
</tr>
<tr>
<td>Ottawa</td>
<td>Canada</td>
<td>0.8</td>
<td>Y</td>
<td>Point</td>
</tr>
<tr>
<td>Mexico City</td>
<td>Mexico</td>
<td>14.1</td>
<td>Y</td>
<td>Point</td>
</tr>
<tr>
<td>Buenos Aires</td>
<td>Argentina</td>
<td>10.75</td>
<td>Y</td>
<td>Point</td>
</tr>
</tbody>
</table>

   (a) Query 2 Select

<table>
<thead>
<tr>
<th>Name</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Havana</td>
<td>Cuba</td>
</tr>
<tr>
<td>Washington, D.C.</td>
<td>USA</td>
</tr>
<tr>
<td>Monterrey</td>
<td>Mexico</td>
</tr>
<tr>
<td>Toronto</td>
<td>Canada</td>
</tr>
<tr>
<td>Brasilia</td>
<td>Brazil</td>
</tr>
<tr>
<td>Rosario</td>
<td>Argentina</td>
</tr>
<tr>
<td>Ottawa</td>
<td>Canada</td>
</tr>
<tr>
<td>Mexico City</td>
<td>Mexico</td>
</tr>
<tr>
<td>Buenos Aires</td>
<td>Argentina</td>
</tr>
</tbody>
</table>

   (b) Query 1 Project

<table>
<thead>
<tr>
<th>Name</th>
<th>Life-exp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mexico</td>
<td>69.36</td>
</tr>
<tr>
<td>Brazil</td>
<td>65.60</td>
</tr>
</tbody>
</table>

   (c) Query 3: Select and project
equivalent of the select operation in RA required in this query. Also notice
the optional cascading dot notation. The CITY table is renamed Ci, and its
attributes are referenced as Ci.Name and Ci.Country.

2. Query: List the names of the capital cities in the CITY table.

```sql
SELECT * 
FROM CITY 
WHERE CAPITAL='Y'
```

**Comments:** This SQL expression is equivalent to the select operation in RA. It is
unfortunate that in SQL the select operation of RA is specified in the WHERE
and not the SELECT clause! The * in SELECT means that all the attributes in
the CITY table must be listed.

3. Query: List the attributes of countries in the Country relation where the life-
expectancy is less than seventy years.

```sql
SELECT Co.Name, Co.Life-Exp
FROM Country Co
WHERE Co.Life-Exp < 70
```

**Comments:** This expression is equivalent to \( \pi \sigma \) in RA. The projected attributes,
Co.Name and Co.Life-Exp in this example are specified in the SELECT clause. The selection condition is specified in the WHERE clause.

4. Query: List the capital cities and populations of countries whose GDP exceeds
one trillion dollars.
Section 3.3 Basic SQL Primer 63

```
SELECT Ci.Name, Co.Pop
FROM City Ci, Country Co
WHERE Ci.Country = Co.Name AND
    Co.GDP > 1000.0 AND
    Ci.Capital = 'Y'
```

**Comments:** This is an implicit way of expressing the join operation. SQL2 and SQL3 also support an explicit JOIN operation. In this case the two tables City and Country are matched on their common attributes Ci.country and Co.name. Furthermore, two selection conditions are specified separately on the City and Country table. Notice how the cascading dot notation alleviated the potential confusion that might have arisen as a result of the attribute names in the two relations.

5. **Query:** What is the name and population of the capital city in the country where the St. Lawrence River originates?

```
SELECT Ci.Name, Ci.Pop
FROM City Ci, Country Co, River R
WHERE R.Origin = Co.Name AND
    Co.Name = Ci.Country AND
    R.Name = 'St. Lawrence' AND
    Ci.Capital = 'Y'
```

**Comments:** This query involves a join among three tables. The River and Country tables are joined on the attributes Origin and Name. The Country and the City tables are joined on the attributes Name and Country. There are two selection conditions on the River and the City tables respectively.

6. **Query:** What is the average population of the noncapital cities listed in the City table?

```
SELECT AVG(Ci.Pop)
FROM City Ci
WHERE Ci.Capital = 'N'
```

**Comments:** The AVG (Average) is an example of an aggregate operation. These operations are not available in RA. Besides AVG, other aggregate operations are COUNT, MAX, MIN, and SUM. The aggregate operations expand the functionality of SQL because they allow computations to be performed on the retrieved data.

7. **Query:** For each continent, find the average GDP.

```
SELECT Co.Cont, Avg(Co.GDP) AS Continent-GDP
FROM Country Co
GROUP BY Co.Cont
```

**Comments:** This query expression represents a major departure from the basic SQL query format. This is because of the presence of the GROUP BY clause.
Chapter 3 Spatial Query Languages

The \texttt{GROUP BY} clause partitions the table on the basis of the attribute listed in the clause. In this example there are two possible values of \texttt{Co.cont}: NAM and SAM. Therefore the \texttt{Country} table is partitioned into two groups. For each group, the average \texttt{GDP} is calculated. The average value is then stored under the attribute \texttt{Continent-GDP} as specified in the \texttt{SELECT} clause.

8. Query: For each country in which at least two rivers originate, find the length of the smallest river.

\begin{verbatim}
SELECT R.Origin, MIN(R.length) AS Min-length
FROM River R
GROUP BY R.Origin
HAVING COUNT(*) > 1
\end{verbatim}

Comments: This is similar to the previous query. The difference is that the \texttt{HAVING} clause allows selection conditions to be enforced on the different groups formed in the \texttt{GROUP BY} clause. Thus only those groups are considered which have more than one member.

9. Query: List the countries whose GDP is greater than that of Canada.

\begin{verbatim}
SELECT Co.Name
FROM Country Co
WHERE Co.GDP > ANY ( SELECT Co1.GDP
FROM Country Co1
WHERE Co1.Name = 'Canada'
)
\end{verbatim}

Comments: This is an example of a nested query. These are queries which have other queries embedded in them. A nested query becomes mandatory when an intermediate table, which does not exist, is required before a query can be evaluated. The embedded query typically appears in the \texttt{WHERE} clause, though it can appear, albeit rarely, in the \texttt{FROM} and the \texttt{SELECT} clauses. The \texttt{ANY} is a set comparison operator. Consult a standard database text for a complete overview of nested queries.

3.3.5 Summary of RA and SQL

RA is a formal database query language. Although it is typically not implemented in any commercial DBMS, it forms an important core of SQL. SQL is the most widely implemented database language. SQL has two components: the DDL and DML. The schema of the database tables are specified and populated in the DDL. The actual queries are posed in DML. We have given a brief overview of SQL. More information can be found in any standard text on databases.

3.4 EXTENDING SQL FOR SPATIAL DATA

Although they are powerful query-processing languages, RA and SQL have their shortcomings. The main one is that these languages traditionally provided only simple datatypes, for example, integers, dates, and strings. SDB applications must handle complex datatypes such as points, lines, and polygons. Database vendors have responded in
two ways: They have either used blobs to store spatial information, or they have created a hybrid system in which spatial attributes are stored in operating-system files via a GIS. SQL cannot process data stored as blobs, and it is the responsibility of the application techniques to handle data in blob form [Stonebraker and Moore, 1997]. This solution is neither efficient nor aesthetic because the data depends on the host-language application code. In a hybrid system, spatial attributes are stored in a separate operating-system file and thus are unable to take advantage of traditional database services such as query language, concurrency control, and indexing support.

Object-oriented systems have had a major influence on expanding the capabilities of DBMS to support spatial (complex) objects. The program to extend a relational database with object-oriented features falls under the general framework of OR-DBMS. The key feature of OR-DBMS is that it supports a version of SQL, SQL3/SQL99, which supports the notion of user-defined types (as in Java or C++). Our goal is to study SQL3/SQL99 enough so that we can use it as a tool to manipulate and retrieve spatial data.

The principle demand of spatial SQL is to provide a higher abstraction of spatial data by incorporating concepts closer to our perception of space [Egenhofer, 1994]. This is accomplished by incorporating the object-oriented concept of user-defined ADTs. An ADT is a user-defined type and its associated functions. For example, if we have land parcels stored as polygons in a database, then a useful ADT may be a combination of the type polygon and some associated function (method), say, adjacent. The adjacent function may be applied to land parcels to determine if they share a common boundary. The term abstract is used because the end user need not know the implementation details of the associated functions. All end users need to know is the interface, that is, the available functions and the data types for the input parameters and output results.

### 3.4.1 The OGIS Standard for Extending SQL

The OGIS consortium was formed by major software vendors to formulate an industry wide standard related to GIS interoperability. The OGIS spatial data model can be embedded in a variety of programming languages, for example, C, Java, SQL, and so on. We focus on SQL embedding in this section.

The OGIS is based on a geometry data model shown in Figure 2.2. Recall that the data model consists of a base-class, GEOMETRY, which is noninstantiable (i.e., objects cannot be defined as instances of GEOMETRY), but specifies a spatial reference system applicable to all its subclasses. The four major subclasses derived from the GEOMETRY superclass are Point, Curve Surface and GeometryCollection. Associated with each class is a set of operations that acts on instances of the classes. A subset of important operations and their definitions are listed in Table 3.9.

The operations specified in the OGIS standard fall into three categories:

1. Basic operations apply to all geometry datatypes. For example, SpatialReference returns the underlying coordinate system where the geometry of the object was defined. Examples of common reference systems include the well-known latitude and longitude system and the often-used Universal Traversal Mercator (UTM).

2. Operations test for topological relationships between spatial objects. For example, overlap tests whether the interior (see Chapter 2) of two objects has a nonempty set intersection.
### Chapter 3: Spatial Query Languages

#### TABLE 3.9: A Sample of Operations Listed in the OGIS Standard for SQL [OGIS, 1999]

<table>
<thead>
<tr>
<th>Basic Functions</th>
<th>SpatialReference()</th>
<th>Returns the underlying coordinate system of the geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Envelope()</td>
<td>Returns the minimum orthogonal bounding rectangle of the geometry</td>
</tr>
<tr>
<td></td>
<td>Export()</td>
<td>Returns the geometry in a different representation</td>
</tr>
<tr>
<td></td>
<td>IsEmpty()</td>
<td>Returns true if the geometry is a null set</td>
</tr>
<tr>
<td></td>
<td>IsSimple()</td>
<td>Returns true if the geometry is simple (no self-intersection)</td>
</tr>
<tr>
<td></td>
<td>Boundary()</td>
<td>Returns the boundary of the geometry</td>
</tr>
<tr>
<td>Topological/ Set</td>
<td>Equal</td>
<td>Returns true if the interior and boundary of the two geometries are spatially equal</td>
</tr>
<tr>
<td>Operators</td>
<td>Disjoint</td>
<td>Returns true if the boundaries and interior do not intersect</td>
</tr>
<tr>
<td></td>
<td>Intersect</td>
<td>Returns true if the geometries are not disjoint</td>
</tr>
<tr>
<td></td>
<td>Touch</td>
<td>Returns true if the boundaries of two surfaces intersect but the interiors do not</td>
</tr>
<tr>
<td></td>
<td>Cross</td>
<td>Returns true if the interior of a surface intersects with a curve</td>
</tr>
<tr>
<td></td>
<td>Within</td>
<td>Returns true if the interior of the given geometry does not intersect with the exterior of another geometry</td>
</tr>
<tr>
<td></td>
<td>Contains</td>
<td>Tests if the given geometry contains another given geometry</td>
</tr>
<tr>
<td></td>
<td>Overlap</td>
<td>Returns true if the interiors of two geometries have nonempty intersection</td>
</tr>
<tr>
<td>Spatial Analysis</td>
<td>Distance</td>
<td>Returns the shortest distance between two geometries</td>
</tr>
<tr>
<td></td>
<td>Buffer</td>
<td>Returns a geometry that consists of all points whose distance from the given geometry is less than or equal to the specified distance</td>
</tr>
<tr>
<td></td>
<td>ConvexHull</td>
<td>Returns the smallest convex geometric set enclosing the geometry</td>
</tr>
<tr>
<td></td>
<td>Intersection</td>
<td>Returns the geometric intersection of two geometries</td>
</tr>
<tr>
<td></td>
<td>Union</td>
<td>Returns the geometric union of two geometries</td>
</tr>
<tr>
<td></td>
<td>Difference</td>
<td>Returns the portion of a geometry that does not intersect with another given geometry</td>
</tr>
<tr>
<td></td>
<td>SymmDiff</td>
<td>Returns the portions of two geometries that do not intersect with each other</td>
</tr>
</tbody>
</table>

3. General operations are for spatial analysis. For example, distance returns the shortest distance between two spatial objects.

#### 3.4.2 Limitations of the Standard

The OGIS specification is limited to the *object* model of space. As shown in the previous chapter, spatial information is sometimes most naturally mapped onto a field-based model. OGIS is developing consensus models for field datatypes and operations. In Chapter 8 we...
introduce some relevant operations for the field-based model which may be incorporated into a future OGIS standard.

Even within the object model, the OGIS operations are limited for simple SELECT–PROJECT–JOIN queries. Support for spatial aggregate queries with the GROUP BY and HAVING clauses does pose problems (see Exercise 4). Finally, the focus in the OGIS standard is exclusively on basic topological and metric spatial relationships. Support for a whole class of metric operations, namely, those based on the direction predicate (e.g., north, south, left, front), is missing. It also does not support dynamic, shape-based, and visibility-based operations discussed in Section 2.1.5.

3.5 Example Queries that Emphasize Spatial Aspects

Using the OGIS datatypes and operations, we formulate SQL queries in the World database which highlight the spatial relationships between the three entities: Country, City, and River. We first redefine the relational schema, assuming that the OGIS datatypes and operations are available in SQL. Revised schema is shown in Table 3.10.

1. Query: Find the names of all countries which are neighbors of the United States (USA) in the Country table.

   SELECT C1.Name AS "Neighbors of USA"
   FROM Country C1, Country C2
   WHERE Touch(C1.Shape, C2.Shape) = 1 AND
      C2.Name = 'USA'

   Comments: The Touch predicate checks if any two geometric objects are adjacent to each other without overlapping. It is a useful operation to determine neighboring geometric objects. The Touch operation is one of the eight topological predicates specified in the OGIS standard. One of the nice properties

<table>
<thead>
<tr>
<th>TABLE 3.10: Basic Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE Country(</td>
</tr>
<tr>
<td>Name varchar(30),</td>
</tr>
<tr>
<td>Cont varchar(30),</td>
</tr>
<tr>
<td>Pop Integer,</td>
</tr>
<tr>
<td>GDP Number,</td>
</tr>
<tr>
<td>Shape Polygon);</td>
</tr>
<tr>
<td>(a)</td>
</tr>
<tr>
<td>CREATE TABLE River(</td>
</tr>
<tr>
<td>Name varchar(30),</td>
</tr>
<tr>
<td>Origin varchar(30),</td>
</tr>
<tr>
<td>Length Number,</td>
</tr>
<tr>
<td>Shape LineString);</td>
</tr>
<tr>
<td>(b)</td>
</tr>
<tr>
<td>CREATE TABLE City (</td>
</tr>
<tr>
<td>Name varchar(30),</td>
</tr>
<tr>
<td>Country varchar(30),</td>
</tr>
<tr>
<td>Pop integer,</td>
</tr>
<tr>
<td>Shape Point );</td>
</tr>
<tr>
<td>(c)</td>
</tr>
</tbody>
</table>
Chapter 3 Spatial Query Languages

of topological operations is that they are invariant under many geometric transformations. In particular the choice of the coordinate system for the World database will not affect the results of topological operations.

Topological operations apply to many different combinations of geometric types. Therefore, in an ideal situation these operations should be defined in an “overloaded” fashion. Unfortunately, many object-relational DBMSs do not support object-oriented notions of class inheritance and operation overloading. Thus, for all practical purposes these operations may be defined individually for each combination of applicable geometric types.

2. Query: For all the rivers listed in the River table, find the countries through which they pass.

```sql
SELECT R.Name, C.Name
FROM River R, Country C
WHERE Cross(R.Shape, C.Shape) = 1
```

Comments: The Cross is also a topological predicate. It is most often used to check for the intersection between a LineString and Polygon objects, as in this example, or a pair of LineString objects.

3. Query: Which city listed in the City table is closest to each river listed in the River table?

```sql
SELECT C1.Name, R1.Name
FROM City C1, River R1
WHERE Distance (C1.Shape, R1.Shape) < ALL (SELECT Distance(C2.Shape, R1.Shape)
FROM City C2
WHERE C1.Name <> C2.Name
)
```

Comments: The Distance is a real-valued binary operation. It is being used once in the WHERE clause and again in the SELECT clause of the subquery. The Distance function is defined for any combination of geometric objects.

4. Query: The St. Lawrence River can supply water to cities that are within 300 km. List the cities that can use water from the St. Lawrence.

```sql
SELECT Ci.Name
FROM City Ci, River R
WHERE Overlap(Ci.Shape, Buffer(R.Shape, 300)) = 1 AND R.Name = 'St. Lawrence'
```

Comments: The Buffer of a geometric object is a geometric region centered at the object whose size is determined by a parameter in the Buffer operation. In the example the query dictates the size of the buffer region. The buffer operation is used in many GIS applications, including floodplain management and urban and rural zoning laws. A graphical depiction of the buffer operation
Section 3.5 Example Queries that Emphasize Spatial Aspects

FIGURE 3.2. The buffer of a river and points within and outside.

is shown in Figure 3.2. In the figure, Cities A and B are likely to be affected if there is a flood on the river, whereas City C will remain unaffected.

5. **Query**: List the name, population, and area of each country listed in the Country table.

   ```sql
   SELECT C.Name, C.Pop, Area(C.Shape) AS "Area"
   FROM Country C
   
   Comments: This query illustrates the use of the Area function. This function is only applicable for Polygon and MultiPolygon geometry types. Calculating the Area clearly depends upon the underlying coordinate system of the World database. For example, if the shape of the Country tuples is given in terms of latitude and longitude, then an intermediate coordinate transformation must be performed before the Area can be calculated. The same care must be taken for Distance and the Length function.

6. **Query**: List the length of the rivers in each of the countries they pass through.

   ```sql
   SELECT R.Name, C.Name, Length(Intersection(R.Shape, C.Shape)) AS "Length"
   FROM River R, Country C
   WHERE Cross(R.Shape, C.Shape) = 1
   
   Comments: The return value of the Intersection binary operation is a geometry type. The Intersection operation is different from the Intersects function, which is a topological predicate to determine if two geometries intersect. The Intersection of a LineString and Polygon can either be a Point or LineString type. If a river does pass through a country, then the result will be a LineString. In that case, the Length function will return non-zero length of the river in each country it passes through.

7. **Query**: List the GDP and the distance of a country’s capital city to the equator for all countries.
Chapter 3 Spatial Query Languages

TABLE 3.11: Results of Query 7

<table>
<thead>
<tr>
<th>Co. Name</th>
<th>Co. GDP</th>
<th>Dist-to-Eq (in Km).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Havana</td>
<td>16.9</td>
<td>2562</td>
</tr>
<tr>
<td>Washington, D.C.</td>
<td>8003</td>
<td>4324</td>
</tr>
<tr>
<td>Brasilia</td>
<td>1004</td>
<td>1756</td>
</tr>
<tr>
<td>Ottawa</td>
<td>658</td>
<td>5005</td>
</tr>
<tr>
<td>Mexico City</td>
<td>694.3</td>
<td>2161</td>
</tr>
<tr>
<td>Buenos Aires</td>
<td>348.2</td>
<td>3854</td>
</tr>
</tbody>
</table>

SELECT Co.GDP, Distance(Point(0, Ci.Shape.y), Ci.Shape) AS "Distance"
FROM Country Co, City Ci
WHERE Co.Name = Ci.Country AND Ci.Capital = 'Y'

Comments: Searching for implicit relationships between datasets stored in a database is outside the scope of standard database functionality. Current DBMSs are geared toward on-line transaction processing (OLTP), while this query, as posed, is in the realm of on-line analytical processing (OLAP). OLAP itself falls under the label of data mining, and we explore this topic in Chapter 8. At the moment the best we can do is list each capital and its distance to the equator.

Point(0, Ci.Shape.y) is a point on the equator which has the same longitude as that of the current capital instantiated in Ci.Name. Results are shown in Table 3.11.

8. Query: List all countries, ordered by number of neighboring countries.

SELECT Co.Name, Count(Co1.Name)
FROM Country Co, Country Co1
WHERE Touch(Co.Shape, Co1.Shape)
GROUP BY Co.Name
ORDER BY Count(Co1.Name)

Comments: In this query all the countries with at least one neighbor are sorted on the basis of number of neighbors.

9. Query: List the countries with only one neighboring country. A country is a neighbor of another country if their land masses share a boundary. According to this definition, island countries, like Iceland, have no neighbors.

SELECT Co.Name
FROM Country Co, Country Co1
WHERE Touch(Co.Shape, Co1.Shape)
GROUP BY Co.Name
HAVING Count(Co1.Name) = 1
Section 3.6  Trends: Object-Relational SQL  71

SELECT Co.Name
FROM Country Co
WHERE Co.Name IN
  (SELECT Co.Name
   FROM Country Co, Country Co1
   WHERE Touch(Co.Shape, Co1.Shape)
   GROUP BY Co.Name
   HAVING Count(*) = 1)

Comments: Here we have a nested query in the FROM clause. The result of the query within the FROM clause is a table consisting of pairs of countries which are neighbors. The GROUP BY clause partitions the new table on the basis of the names of the countries. Finally the HAVING clause forces the selection to be paired to those countries that have only one neighbor. The HAVING clause plays a role similar to the WHERE clause with the exception that it must include such aggregate functions as count, sum, max, and min.

10. Query: Which country has the maximum number of neighbors?

CREATE VIEW Neighbor AS
  SELECT Co.Name, Count(Co1.Name) AS num_neighbors
  FROM Country Co, Country Co1
  WHERE Touch(Co.Shape, Co1.Shape)
  GROUP BY Co.Name

SELECT Co.Name, num_neighbors
FROM Neighbor
WHERE num_neighbor = (SELECT Max(num_neighbors)
  FROM Neighbor)

Comments: This query demonstrates the use of views in simplifying complex queries. First query (view) computes the number of neighbors for each country. This view creates a virtual table which can be used as a normal table in subsequent queries. The second query selects the country with the largest number of neighbors from the Neighbor view.

3.6 TRENDS: OBJECT-RELATIONAL SQL

The OGIS standard specifies the datatypes and their associated operations which are considered essential for spatial applications such as GIS. For example, for the Point datatype an important operation is Distance, which computes the distance between two points. The length operation is not a semantically correct operation on a Point datatype. This is similar to the argument that the concatenation operation makes more sense for Character datatype than for, say, the Integer type.

In relational databases the set of datatypes is fixed. In object-relational and object-oriented databases, this limitation has been relaxed, and there is built in support for user-defined datatypes. Even though this feature is clearly an advantage, especially when dealing with nontraditional database applications such as GIS, the burden of
constructing syntactically and semantically correct datatypes is now on the database application developer. To share some of the burden, commercial database vendors have introduced application-specific “packages” which provide a seamless interface to the database user. For example, Oracle markets a GIS specific package called the Spatial Data Cartridge.

SQL3/SQL99, the proposed SQL standard for OR-DBMS allows user-defined datatypes within the overall framework of a relational database. Two features of the SQL3 standard that may be beneficial for defining user-defined spatial datatypes are described below.

### 3.6.1 A Glance at SQL3

The SQL3/SQL99 proposes two major extensions to SQL2/SQL92, the current accepted SQL draft.

1. **ADT**: An ADT can be defined using a `CREATE TYPE` statement. Like classes in object-oriented technology, an ADT consists of attributes and member functions to access the values of the attributes. Member functions can potentially modify the value of the attributes in the datatype and thus can also change the database state.

   An ADT can appear as a column type in a relational schema. To access the value that the ADT encapsulates, a member function specified in the `CREATE TYPE` must be used. For example, the following script creates a type `Point` with the definition of one member function `Distance`:

   ```sql
   CREATE TYPE Point (
     x NUMBER,
     y NUMBER,
     FUNCTION Distance(:u Point,:v Point)
       RETURNS NUMBER
   );
   ```

   The colons before `u` and `v` signify that these are local variables.

2. **Row Type**: A row type is a type for a relation. A row type specifies the schema of a relation. For example the following statement creates a row type `Point`.

   ```sql
   CREATE ROW TYPE Point ( 
     x NUMBER,
     y NUMBER );
   ```

   We can now create a table that instantiates the row type. For example:

   ```sql
   CREATE TABLE Pointtable of TYPE Point;
   ```

   In this text we emphasize the use of ADT instead of row type. This is because the ADT as a column type naturally harmonizes the definition of an OR-DBMS as an extended relational database.
3.6.2 Object-Relational Schema

Oracle8 is an OR-DBMS introduced by the Oracle Corporation. Similar products are available from other database companies, for example, IBM. OR-DBMS implements a part of the SQL3 Standard. The ADT is called the "object type" in this system.

Below we describe how the three basic spatial datatypes: Point, LineString, and Polygon are constructed in Oracle8.

```sql
CREATE TYPE Point AS OBJECT (  
x NUMBER,  
y NUMBER,  
MEMBER FUNCTION Distance(P2 IN Point) RETURN NUMBER,  
PRAGMA RESTRICT_REFERENCES(Distance, WNDS));
```

The Point type has two attributes, x and y, and one member function, Distance. PRAGMA alludes to the fact that the Distance function will not modify the state of the database: WNDS (Write No Database State). Of course in the OGIS standard many other operations related to the Point type are specified, but for simplicity we have shown only one. After its creation the Point type can be used in a relation as an attribute type. For example, the schema of the relation City can be defined as follows:

```sql
CREATE TABLE City (  
Name varchar(30),  
Country varchar(35),  
Pop int,  
Capital char(1),  
Shape Point);
```

Once the relation schema has been defined, the table can be populated in the usual way. For example, the following statement adds information related to Brasilia, the capital of Brazil, into the database

```sql
INSERT INTO CITY(¯’Brasilia’, ‘Brazil’, 1.5, ¯’Y’,  
Point(-55.4,-23.2));
```

The construction of the LineString datatype is slightly more involved than that of the Point type. We begin by creating an intermediate type, LineType:

```sql
CREATE TYPE LineType AS VARRAY(500) OF Point;
```

Thus LineType is a variable array of Point datatype with a maximum length of 500. Type specific member functions cannot be defined if the type is defined as a Varray. Therefore we create another type LineString

```sql
CREATE TYPE LineString AS OBJECT (  
Num_of_Points INT,  
Geometry LineType,  
MEMBER FUNCTION Length(SELF IN) RETURN NUMBER,  
PRAGMA RESTRICT_REFERENCES(Length, WNDS));
```
Chapter 3 Spatial Query Languages

The attribute Num of Points stores the size (in terms of points) of each instance of the LineString type. We are now ready to define the schema of the River table.

```sql
CREATE TABLE River(
    Name varchar(30),
    Origin varchar(30),
    Length number,
    Shape LineString
);
```

While inserting data into the River table, we have to keep track of the different datatypes involved.

```sql
INSERT INTO RIVER('Mississippi', 'USA', 6000,
    LineString(3, LineType(Point(1,1),Point(1,2),
    Point(2,3)))
```

The Polygon type is similar to LineString. The sequence of type and table creation and data insertion is given in Table 3.12.

**TABLE 3.12: The Sequence of Creation of the Country Table**

(a) CREATE TYPE PolyType AS VARRAY(500) OF Point

(b) CREATE TYPE Polygon AS OBJECT (
    Num of Points INT,
    Geometry PolyType,
    MEMBER FUNCTION Area(SELF IN) RETURN NUMBER,
    PRAGMA RESTRICT REFERENCES(Length, WD)));

(c) CREATE TABLE Country(
    Name varchar(30),
    Cont varchar(30),
    Pop int,
    GDP number,
    Life-Exp number,
    Shape LineString
);

(d) INSERT INTO Country('Mexico', 'NAM', 107.5, 694.3, 69.36,
    Polygon(23, Polytype(Point(1,1), ..., Point(1,1)))
### 3.6.3 Example Queries

1. **Query:** List all the pairs of cities in the City table and the distances between them.

   ```sql
   SELECT C1.Name, C1.Distance(C2.Shape) AS 'Distance'
   FROM City C1, City C2
   WHERE C1.Name <> C2.Name
   ``

   **Comments:** Notice the object-oriented notation for the `Distance` function in the `SELECT` clause. Contrast it with the test notation used in Section 3.5: `Distance(C1.Shape, C2.Shape)`. The predicate in the `WHERE` clause ensures that the `Distance` function is not applied between two copies of the same city.

2. **Query:** Validate the length of the rivers given in the River table, using the geometric information encoded in the `Shape` attribute.

   ```sql
   SELECT R.Name, R.Length, R.Length() AS 'Derived Length'
   FROM River R
   ``

   **Comments:** This query is being used for data validation. The length of the rivers is already available in the `Length` attribute of the River table. Using the `Length()` function we can check the integrity of the data in the table.

3. **Query:** List the names, populations, and areas of all countries adjacent to the USA.

   ```sql
   SELECT C2.Name, C2.Pop, C2.Area() AS 'Area'
   FROM Country C1, Country C2
   WHERE C1.Name = 'USA' AND
   C1.Touch(C2.Shape) = 1
   ``

   **Comments:** The `Area()` function is a natural function for the Polygon ADT to support. Along with `Area()`, the query also invokes the `Touch` topological predicate.

### 3.7 SUMMARY

In this chapter we discussed database query languages, covering the following topics.

- **RA** is the formal query language associated with the relational model. It is rarely, if ever, implemented in a commercial system but forms the core of SQL.

- **SQL** is the most widely implemented query language. It is a declarative language, in that the user only has to specify the result of the query rather than means of a arriving at the result. SQL extends RA with many other important functions, including aggregate functions to analytically process queried data.

- The **OGIS** standard recommends a set of spatial datatypes and functions that are considered crucial for spatial data querying.

- **SQL3/SQL 1999** is the standardization platform for the object-relational extension of SQL. It is not specific to GIS or spatial databases but covers general object-relational databases. The most natural scenario is that the OGIS standard recommendations will be implemented in a subset of SQL3.
BIBLIOGRAPHIC NOTES

3.1, 3.2, 3.3 A complete exposition of relational algebra and SQL can be found in any introductory text in databases, including [Elmasri and Navathe, 2000; Ramakrishnan, 1998; Ullman and Widom, 1999].

3.4, 3.5 Extensions of SQL for spatial applications are explored in [Egenhofer, 1994]. The OGIS document [OpenGIS, 1998] is an attempt to harmonize the different spatial extensions of SQL. For an example of query languages in supporting spatial data analysis, see [Lin and Huang, 2001].

3.6 SQL 1999/SQL3 is the adopted standard for the object-relational extension of SQL. Subsets of the standard have already been implemented in commercial products, including Oracle’s Oracle8 and IBM’s DB2.

EXERCISES

For all queries in Exercises 1 and 2 refer to Table 3.1.

1. Express the following queries in relational algebra.
   (a) Find all countries whose GDP is greater than $500 billion but less than $1 trillion.
   (b) List the life expectancy in countries that have rivers originating in them.
   (c) Find all cities that are either in South America or whose population is less than two million.
   (d) List all cities which are not in South America.

2. Express in SQL the queries listed in Exercise 1.

3. Express the following queries in SQL.
   (a) Count the number of countries whose population is less than 100 million.
   (b) Find the country in North America with the smallest GDP. Do not use the MIN function. Hint: nested query.
   (c) List all countries that are in North America or whose capital cities have a population of less than 5 million.
   (d) Find the country with the second highest GDP.

4. The Reclassify (see Section 2.1.5) is an aggregate function that combines spatial geometries on the basis of nonspatial attributes. It creates new objects from the existing ones, generally by removing the internal boundaries of the adjacent polygons whose chosen attribute is same. Can we express the Reclassify operation using OGIS operations and SQL92 with spatial datatypes? Explain.

5. Discuss the geometry data model of Figure 2.2. Given that on a “world” scale, cities are represented as point datatypes, what datatype should be used to represent the countries of the world. Note: Singapore, the Vatican, and Monaco are countries. What are the implementation implications for the spatial functions recommended by the OGIS standard.

6. [Egenhofer, 1994] proposes a list of requirements for extending SQL for spatial applications. The requirements are shown below. Which of these the recommendations have been accepted in the OGIS SQL standard? Discuss possible reasons for postponing the others.

7. The OGIS standard includes a set of topological spatial predicates. How should the standard be extended to include directional predicates such as East, North, North-East, and so forth. Note that the directional predicates may be fuzzy: “Where does North-East end and East begin?”

8. This exercise surveys the dimension-extended nine-intersection model: DE-9IM. The DE-9IM extends Egenhofer’s nine-intersection model introduced in Chapter 2. The
Spatial ADT | An abstract data type spatial hierarchy with associated operations
--- | ---
Graphical presentation | Natural medium of interaction with spatial data
Result combination | Combining the results of a sequence of queries
Context | Place result in context by including information not explicitly requested
Content examination | Provide mechanisms to guide the evolution of map drawing
Selection by pointing | Pose and constraints by pointing to maps
Display manipulations | Varying graphical presentation of spatial objects and their parts
Legend | Descriptive legend
Labels | Labels for understanding of drawings
Selection of map scale | Produced map should allow user to continue applying their skills on interpreting actual size of objects drawn and the selection of a specific scale of rendering
Area of interest | Tools to restrict the area of interest to a particular geography

Template matrix of DE-9IM is shown below.

\[
\Gamma_9(A, B) = \begin{pmatrix}
\dim(A \cap B) & \dim(A \cap B^c) & \dim(A \cap B^-) \\
\dim(A^\circ \cap B^c) & \dim(A^\circ \cap B^-) & \dim(A^\circ \cap \partial B^-) \\
\dim(A^- \cap B^c) & \dim(A^- \cap B^-) & \dim(A^- \cap \partial B^-)
\end{pmatrix}
\]

The key difference between 9IM and DE-9IM is that instead of testing whether each entry in the matrix is empty or nonempty; in the DE-9IM only the dimension of the geometric object is required. The dimension of planar two-dimensional objects can take four values: \(-1\) for empty-set, 0 for points, 1 for lines, and 2 for nonzero area objects. In many instances the value of the matrix entry does not matter. The following is the list of values that the matrix entries can span.

\[T: X \text{ and } Y \text{ must intersect. } \dim(X \cap Y) = 0, 1, 2. X \text{ and } Y \text{ are either the interior, exterior, or boundary of } A \text{ and } B \text{ respectively.}\]

\[F: \dim(X \cap Y) = -1. X \text{ and } Y \text{ must not intersect.}\]

\[*: \text{It does not matter if the intersection exists. } \dim(X \cap Y) = \{-1, 0, 1, 2\}\]

\[0: \dim(X \cap Y) = 0\]

\[1: \dim(X \cap Y) = 1\]

\[2: \dim(X \cap Y) = 2\]

Below is the signature matrix of two equal objects.

\[
\begin{pmatrix}
T & * & F \\
* & * & F \\
* & * & *
\end{pmatrix}
\]
Chapter 3  Spatial Query Languages

(a) What is the signature matrix (matrices) of the touch and cross topological operations? Note that the signature matrix depends on the combination of the datatypes. The signature matrix of a point/point combination is different from that of a multipolygon/multipolygon combination.

(b) What operation (and combination of datatypes) does the following signature matrix represent?

\[
\begin{pmatrix}
1 & * & T \\
* & * & F \\
T & * & *
\end{pmatrix}
\]

(c) Consider the sample figures shown Figure 3.3. What are signature matrices in 9IM and DE-9IM. Is DE-9IM superior to 9IM? Discuss.

9. Express the following queries in SQL, using the OGIS extended datatype and functions.

(a) List all cities in the City table which are within five thousand miles of Washington, D.C.

(b) What is the length of Rio Paranas in Argentina and Brazil?

(c) Do Argentina and Brazil share a border?

(d) List the countries that lie completely south of the equator.

10. Given the schema:

    RIVER(NAME:char, FLOOD-PLAIN:polygon, GEOMETRY:linestring)
    ROAD(ID:char, NAME:char, TYPE:char, GEOMETRY:linestring)
    FOREST(NAME:char, GEOMETRY:polygon)
    LAND-PARCELS(ID:integer, GEOMETRY:polygon, county:char)

Transform the following queries into SQL using the OGIS specified datatypes and operations.

(a) Name all the rivers that cross Itasca State Forest.

(b) Name all the tar roads that intersect Francis Forest.

(c) All roads with stretches within the floodplain of the river Montana are susceptible to flooding. Identify all these roads.

(d) No urban development is allowed within two miles of the Red River and five miles of the Big Tree State Park. Identify the landparcels and the county they are in that cannot be developed.

(e) A river defines part of boundary of a country.

---

FIGURE 3.3. Sample objects [Clementini and Felice, 1995]
Section 3.8  Appendix: State Park Database

11. Study the compiler tools such as YACC (Yet Another Compiler Compiler). Develop a syntax scheme to generate SQL3 data definition statements from an Entity Relationship Diagram (ERD) annotated with pictograms.

12. How would one model the following spatial relationships using 9-intersection model or OGIS topological operations?
   (a) A river (LineString) originates in a country (Polygon)
   (b) A country (e.g., Vatican city) is completely surrounded by another (e.g., Italy) country
   (c) A river (e.g., Missouri) falls into another (e.g., Mississippi) river
   (d) Forest stands partition a forest

13. Review the example RA queries provided for state park database in Appendix. Write SQL expressions for each RA query.

14. Redraw the ER diagram provided in Figure 3.4 using pictograms. How will one represent Fishing-Opener and Distance attributes in the new ER diagram by translating the resulting ER diagram using SQL3/OGIS constructs?

15. Consider the table-designs in Figure 1.3 and 1.4. Describe SQL queries to compute spatial properties (e.g. area, perimeter) of census blocks using each representation. Which representation lead to simple queries?

16. Revisit the Java program in Section 2.1.6. Design Java programs to carry out the spatial queries listed in Section 3.6.3. Compare and contrast querying spatial dataset using Java with querying using SQL3/OGIS.

17. Define user defined data types for geometry aggregation data types in OGIS using SQL3.

18. Revisit relational schema for state park example in Section 2.2.3. Outline SQL DML statements to create relevant tables using OGIS spatial data type.

19. Consider shape-based queries, for example, list countries shaped like ladies boot or list squarish census blocks. Propose extensions to SQL3/OGIS to support such queries.

20. Consider visibility-based queries, for example, list objects visible (not occluded) from a vista-point and viewer orientation. Propose a set of data types and operations for extending SQL3/OGIS to support such queries.

3.8 APPENDIX: STATE PARK DATABASE

The State Park database consists of two entities: Park and Lake. The attributes of these two entities and their relationships are shown in Figure 3.4. The ER diagram is mapped into the relational schema shown below. The entities and their relationships are materialized in Table 3.13.

StatePark(Sid: integer, Sname: string, Area: float, Distance: float)
Lake(Lid: integer, Lname: string, Depth: float, Main-Catch: string)
ParkLake(Lid: integer, Sid: integer, Fishing-Opener: date)

The above schema represents three entities: StatePark, Lake, and ParkLake. StatePark represents all the state parks in Minnesota, and its attributes are a unique national identity number, Sid; the name of the park, Sname; its area in sq. km., Area; and the distance of the park from Minneapolis, Distance. The Lake entity also has a unique id, Lid, a name, Lname; the average depth of the lake, Depth; and the primary fish in the lake, Main-catch. The ParkLake entity is used to integrate queries across the two entities StatePark and Lake. ParkLake identifies the lakes that are in the state parks. Its attributes are Lid, Sid, and the date the fishing season commences on the
given lake, Fishing-Opener. Here we are assuming that different lakes have different Fishing-Openers.

### 3.8.1 Example Queries in RA

We now give examples that show how the relational operators defined earlier can be used to retrieve and manipulate the data in a database. Our format is as follows: We first list the query in plain English; then we give the equivalent expression in RA, and finally we make comments about the algebraic expression, including an alternate form of the algebraic expression.

**TABLE 3.13: Tables for the StatePark Database**

<table>
<thead>
<tr>
<th>Park</th>
<th>Sid</th>
<th>Sname</th>
<th>Area</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Itasca</td>
<td>150.0</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>Woodbury</td>
<td>255.0</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>S3</td>
<td>Brighton</td>
<td>175.0</td>
<td>300</td>
<td></td>
</tr>
</tbody>
</table>

(a) Park

<table>
<thead>
<tr>
<th>Lake</th>
<th>Lid</th>
<th>Lname</th>
<th>Depth</th>
<th>Main-Catch</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Lino</td>
<td>20.0</td>
<td>Walleye</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>Chaska</td>
<td>30.0</td>
<td>Trout</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>Sussex</td>
<td>45.0</td>
<td>Walleye</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td>Todd</td>
<td>28.0</td>
<td>Bass</td>
<td></td>
</tr>
</tbody>
</table>

(b) Lake

<table>
<thead>
<tr>
<th>ParkLake</th>
<th>Lid</th>
<th>Sid</th>
<th>Fishing-Opener</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>S1</td>
<td>05/15</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>S1</td>
<td>05/15</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>S3</td>
<td>06/01</td>
<td></td>
</tr>
</tbody>
</table>

(c) ParkLake
Section 3.8  Appendix: State Park Database  81

Query: Find the name of the StatePark which contains the Lake with Lid number 100.

\[ \pi_{\text{Spname}}(\text{StatePark} \bowtie \sigma_{\text{Lid} = 100}((\text{ParkLake})) \]

Comments: We begin by selecting the set of tuples in ParkLake with Lid 100. The resultant set is naturally joined with the relation StatePark on the key Sid. The result is projected onto the StatePark name, Spname. This query can be broken into parts using the renaming operator \( \rho \). The renaming operator is used to name the intermediate relations that arise during the evaluation of a complex query. It can also be used to rename the attributes of a relation. For example,

\[ \rho(\text{Newname}(1 \rightarrow \text{Att1}), \text{Oldname}) \]

renames the relation Oldname to the Newname. Also the first attribute, counting from left to right, of the Newname is called Att1.

With this naming convention, we can break up this query into parts as follows:

\[ \rho(\text{Temp1}, \sigma_{\text{Lid}=100}(\text{ParkLake})) \]
\[ \rho(\text{Temp2}, \text{Temp1} \bowtie \text{StatePark}) \]
\[ \pi_{\text{Spname}}(\text{Temp2}) \]

An alternate formulation of the query is

\[ \pi_{\text{Spname}}(\sigma_{\text{Lid}=100}(\text{ParkLake} \bowtie \text{StatePark})) \]

From the point of view of implementation, this query is more expensive than the previous one because it is performing a join on a larger set, and join is the most expensive of all the five operators in relational algebra.

1. Query: Find the names of the StateParks with Lakes where the MainCatch is Trout.

\[ \pi_{\text{Spname}}(\text{StatePark} \bowtie (\text{ParkLake} \bowtie \sigma_{\text{Main-Catch} = 'Trout'}(\text{Lake}))) \]

Comments: Here we are applying two join operators in succession. But first we reduce the set size by first selecting all Lakes with Main-Catch of Trout. Then we join the resultant on the Lid key with ParkLake. This is followed by another join with StatePark on Sid. Finally we project the answer on the StatePark name.

2. Query: Find the Main-Catch of the lakes that are in Itasca State Park

\[ \pi_{\text{Main-Catch}}(\text{Lake} \bowtie (\text{ParkLake} \bowtie \sigma_{\text{Spname} = 'Itasca'}(\text{StatePark}))) \]

Comments: This query is very similar to the one above.

Query: Find the names of StateParks with at least one lake.

\[ \pi_{\text{Spname}}(\text{StatePark} \bowtie \text{ParkLake}) \]

Comment: The join on Sid creates an intermediate relation in which tuples from the StatePark relation are attached to the tuples from ParkLake. The result is then projected onto Spname.
3. **Query:** List the names of StateParks with lakes whose main catch is either bass or walleye.

\[ \rho(\text{TempLake}, \sigma_{\text{Main-Catch} = \text{Bass}}(\text{Lake}) \cup \sigma_{\text{Main-Catch} = \text{Walleye}}(\text{Lake})) \]

\[ \pi_{\text{spname}}(\text{TempLake} \bowtie \text{ParkLake} \bowtie \text{StatePark}) \]

**Comments:** Here we use the union operator for the first time. We first select lakes with Main-Catch of bass or walleye. We then join on Lid with ParkLake and join again on Sid with StatePark. We get the result by projecting on Spname.

4. **Query:** Find the names of StateParks that have both bass and walleye as the main-catch in their lakes.

\[ \rho(\text{TempBass}, \pi_{\text{Spname}}(\sigma_{\text{Main-Catch} = \text{Bass}}(\text{Lake}) \bowtie \text{ParkLake} \bowtie \text{StatePark})) \]

\[ \rho(\text{TempWall}, \pi_{\text{Spname}}(\sigma_{\text{Main-Catch} = \text{Walleye}}(\text{Lake}) \bowtie \text{ParkLake} \bowtie \text{StatePark})) \]

\[ \text{TempBass} \cap \text{TempWall} \]

**Comment:** This query formulation is barely right!

5. **Query:** Find the names of the StateParks that have at least two lakes.

\[ \rho(\text{Temp}, \pi_{\text{Sid}, \text{Spname}, \text{Lid}}(\text{StatePark} \bowtie \text{ParkLake})) \]

\[ \pi_{\text{Spname}}(\sigma_{\text{Sid}1 = \text{Sid}2} \land (\text{Lid}1 \neq \text{Lid}2)) \]

6. **Query:** Find the identification number, Sid, of the StateParks that are at least fifty miles away from Minneapolis with lakes where the Main-Catch is not trout.

\[ \pi_{\text{Sid}}(\sigma_{\text{distance} > 50}(\text{StatePark})) \]

\[ \pi_{\text{Sid}}((\sigma_{\text{main-catch} \neq \text{Trout}}(\text{Lake}) \bowtie \text{ParkLake} \bowtie \text{StatePark}) \]