Chapter 3

Spatial Query Languages

Introduction

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 the structured query language (SQL). It is partly based on the formal query language, relational algebra, and it is easy to use, intuitive, and versatile. Since spatial database management systems 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 like 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 object-relational database management system (OR-DBMS).

A corollary to this new-found 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. Since we are dealing with spatial data, we will examine the spatial extensions of SQL.

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 will focus here on the nongraphical 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 relational algebra and SQL respectively. Section 3.4 is devoted to a discussion on the spatial requirements for extending SQL. We also introduce the Open GIS (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, Relational Algebra (RA), is the 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 relational algebra (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.

`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,`
Captial`:char(1), `*Shape*`:char(9))`
`River(`*Name*`: varchar(35), `*Origin*`: varchar(35), `*Length* `: integer,`
Shape`:char(13))`

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). Since, 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.

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 *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 *Shape* attribute is not sufficient. The overloading of Name across tables can be resolved by qualifying attribute with tables using a dot notation table.attribute. County.Name, city.Name, and river.Name uniquely identify Name attribute inside different tables. We also need information about the direction of the river flow. In Chapter 7 we will discuss querying spatial networks where directional information is important.

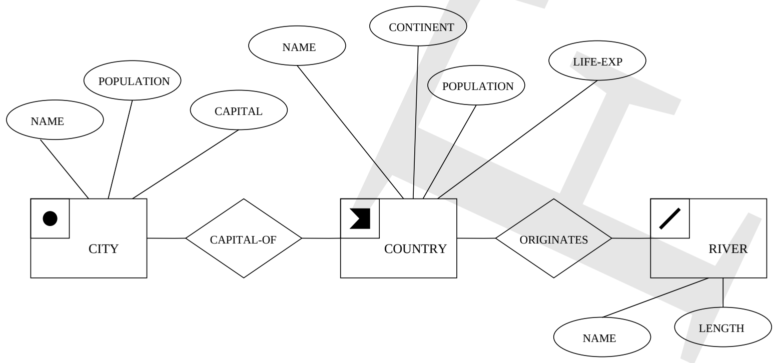

Figure 3.1: The ER diagram of the `World` database

3.2 Relational Algebra

Relational algebra (RA) is a formal query language associated with the relational model. An *algebra* is a mathematical structure consisting of two distinct sets of elements, $(\mathbf{\Omega_a},\mathbf{\Omega_o})$. Ω_a is the set of *operands* and Ω_o is the set of *operations*. An algebra must satisfy many axioms but the most crucial is that the result of an *operation* on an *operand* must remain in Ω_a. A simple example of an algebra is the set of integers. The *operands* are the integers and the *operations* are addition and multiplication. In chapter 8 we will 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 *select, project, union, cross-product, difference, and 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, relational algebra provides two operations: *select* and *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 `Country` table which are in *North-America* (NAM), we use the following relational algebra expression:

$$\sigma_{\text{cont='North-America'}}(Country)$$

The result of this operation is shown in Table 3.2a. The rows retrieved by the select operation σ are specified by the comparison selection operator, which in this example is *cont='North-America'*. The schema of the input relation is not altered by the select operator. The formal the 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 *project* operation, π. For example, to retrieve the names of all countries listed in the `Country` table, we use the following expression:

$$\pi_{\text{Name}}(Country).$$

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 3.2c for the result.

$$\pi_{\text{Name}}(\sigma_{\text{Cont='North-America'}})(Country)$$

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 which are *union-compatible.* Two relations are union-compatible if they have the same number of columns, have 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. $R = \pi_{\text{Name}}(\sigma_{\text{Cont='North-America'}}(\text{Country}))$
  2. $S = \pi_{\text{Origin}}(\text{River})$
  3. $R \cup S$.

  The resulting relation is shown in Table 3.4a. Notice that the attributes *R.Name* and *S.Origin* have the same domain, since *R.Origin* refers to County.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 which have no river (listed in the `River` table) originating in them. The resulting relation is shown in Table 3.43b.

1. $R = \pi_{\text{Name}}(\sigma_{\text{Cont='North-America'}}(\text{Country}))$
2. $S = \pi_{\text{Origin}}(\text{River})$
3. $R - S$.

- **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 which are in South America and also have a river originating in them, we use the intersection operation. The result is shown in Table 3.4c.

  1. $R = \pi_{\text{Name}}(\sigma_{\text{Cont='South America'}}(\text{Country}))$
  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.

Conditional Joins

The general conditional join, $\bowtie_c$, between two relations R and S is expressed as follows:

$$R \bowtie_c S = \sigma_c(R \times S).$$

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

$R \times S$	R.Name	R.Pop	S.Name	S.Pop

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

$$U = R \bowtie S = \sigma_{(\text{R.Name} = \text{`Mexico'}) \wedge (\text{R.Pop} > \text{S.Pop})}(R \times S)$$

Natural Join

An important special case of the conditional join is the *natural join.* In a natural join only the *equality* selection condition is applied to the common attributes of the two relations. For example, a natural join can be used to find the populations of countries where rivers originate. The steps follow:

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,
$$C \bowtie_{\text{C.Name} = \text{R.Origin}} R.$$
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:
$$\pi_{\text{Name, Pop}}(C \bowtie R).$$

3.3 Basic SQL Primer

Structured query language (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 of the language only has to specify the answer rather than a procedure to retrieve the answer.

The SQL language has 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. 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 will discuss in Section 3.4. A more detailed and complete exposition of SQL can be found in any standard text on databases [Ullman and Widom, 1999, Elmasri and Navathe, 2000].

3.3.1 Data Definition Language

Creation of the relational schema and addition and deletion of the tables are specified in the data definition language (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.

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

Explanation: The **CREATE TABLE** clause is used to define the 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* 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 give the complete set of datatypes, which can be found in an 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 Data Manipulation Language

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 relavant data from the database. The basic syntax of an SQL query is extremely simple:

```
SELECT   tuples
FROM     relations
WHERE    tuple-constraint
```

This form is equivalent to the relational algebra (RA) expression consisting of π, σ, and $\bowtie$. SQL has more clauses related to aggregation (e.g., `GROUP BY, HAVING`), ordering results (e.g., `ORDER BY`), etc. In addition SQL allows the formulation of nested queries. We will 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 versatality and power of SQL. All the tables queried are from the `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 `CITY` table.

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

   Comments: The SQL expression is equivalent to the project operation in RA. The `WHERE` clause is missing in the SQL expression because there is no 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.

```
SELECT  *
FROM    CITY
WHERE   CAPTIAL='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 names of countries in the `Country` relation where the life-expectancy is less than 70 years.

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

Comments: This expression is equivalent to $\pi \circ \sigma$ 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.

```
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 the standard way of expressing the `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' AND
```

Comments: This query involves a join between 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 non-capital 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. The `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 *Co.cont*: NAM and SAM. Therefore the `Country` table is partitioned into two groups. For each group, the average *GDP* is calculated. The average value is then stored under the attribute *Continent-GDP* as specified in the `SELECT` clause.

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

```
SELECT    R.Origin, MIN(R.length) AS Min-length
FROM      River R
GROUP BY  R.Origin
HAVING    COUNT(*) > 1
```

Comments: This is similar to the previous query. The difference is that the `HAVING` clause allows selection conditions to be enforced on the different groups formed in the `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 Canada's.*

```
SELECT  Co.Name
FROM    Country Co
WHERE   Co.GDP      > ANY  ( SELECT  Co1.GDP
                             FROM     Country Co1
                             WHERE    Co1.Name = 'Canada' )
```

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 `WHERE` clause, though it can appear, albeit rarely, in the `FROM` and the `SELECT` clauses. The `ANY` is a set comparison operator. Consult a standard database text for a complete overview of nested queries.

3.3.5 Summary of Relational Algebra and SQL

Relational algebra is a formal database query language. While it is typically not implemented in any commercial database management system, it forms an important core of SQL. Structured query language (SQL) is the most widely implemented database language. SQL has two components: the data definition language (DDL) and data manipulation language (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.

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

(a) Country

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

(b) City

RIVER	Name	Origin	Length (kilometers)	Shape
	Rio Parana	Brazil	2600	LineStringid-1
	St. Lawrence	USA	1200	LineStringid-2
	Rio Grande	USA	3000	LineStringid-3
	Mississippi	USA	6000	LineStringid-4

(c) River

Table 3.1: The tables of the World database

Name	Cont	Pop (millions)	GDP (billions)	Life-Exp	Shape
Canada	NAM	30.1	658.0	77.08	Polygonid-1
Mexico	NAM	107.5	694.3	69.36	Polygonid-2
Cuba	NAM	11.7	16.9	75.95	Polygonid-4
USA	NAM	270.0	8003.0	75.75	Polygonid-5

(a) Select

Name
Canada
Mexico
Brazil
Cuba
USA
Argentina

(b) Project

Name
Canada
Mexico
Cuba
USA

(c) Select and project

Table 3.2: Results of two basic operations in relational algebra: select and project

R	R.A	R.B
	A_1	B_1
	A_2	B_2

(a) Relation R

S	S.C	S.D
	C_1	D_1
	C_2	D_2

(b) Relation S

$R \times S$	R.A	R.B	S.C	S.D
	A_1	B_1	C_1	D_1
	A_1	B_1	C_2	D_2
	A_2	B_2	C_1	D_1
	A_2	B_2	C_2	D_2

(c) $R \times S$

Table 3.3: The cross-product of relations R and S

NAME
Canada
Mexico
Brazil
Cuba
USA

(a) Union

NAME
Canada
Mexico
Cuba

(b) Difference

NAME
Brazil

(c) Intersection

Table 3.4: The results of set operations

$R \times S$	R.Name	R.Pop	S.Name	S.Pop
	⋮	⋮	⋮	⋮
	Mexico	107.5	Canada	30.1
	Mexico	107.5	Mexico	107.5
	Mexico	107.5	Brazil	183.3
	Mexico	107.5	Cuba	11.7
	Mexico	107.5	USA	270.0
	Mexico	107.5	Argentina	36.3
	⋮	⋮	⋮	⋮

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

R.Name	R.Pop	S.Name	S.Pop
Mexico	107.5	Canada	30.1
Mexico	107.5	Cuba	11.7
Mexico	107.5	Argentina	36.3

(b) The select operation on $R \times S$

Table 3.5: Steps of the conditional join operation

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

(a) `Country` schema

```
CREATE   TABLE Country {
         Name     VARCHAR(35),
         Origin   VARCHAR(35),
         Length   INT,
         Shape    CHAR(15)
         PRIMARY KEY   Name }
```

(b) `River` schema

Table 3.6: The `Country` and `River` schema in SQL

Name	Country	Pop(millions)	Capital	Shape
Havana	Cuba	2.1	Y	Point
Washington, D.C.	USA	3.2	Y	Point
Brasilia	Brazil	1.5	Y	Point
Ottawa	Canada	0.8	Y	Point
Mexico City	Mexico	14.1	Y	Point
Buenos Aires	Argentina	10.75	Y	Point

(a) Query1: Select

Name	Counry
Havana	Cuba
Washington, D.C.	USA
Monterrey	Mexico
Toronto	Canada
Brasilia	Brazil
Rosario	Argentina
Ottawa	Canada
Mexico City	Mexico
Buenos Aires	Argentina

(b) Query2: Project

Name	Life-exp
Mexico	69.36
Brazil	65.60

(c) Query3: Select and project

Table 3.7: Tables from the select, project, and select and project operations

Ci.Name	Co.Pop
Brassilia	183.3
Washington, D.C.	270.0

(a) Query 4

Ci.Name	Ci.Pop
Washington, D.C.	3.2

(b) Query 5

Average-Pop
2.2

(c) Query 6

Cont	Continent-Pop
NAM	2343.05
SAM	676.1

(d) Query 7

Origin	Min-length
USA	1200

(e) Query 8

Co.Name
Mexico
Brazil
USA

(f) Query 9

Table 3.8: Results of example queries

3.4 Extending SQL for spatial data

Although they are powerful query-processing languages, relational algebra and SQL have their shortcomings. The main one is that these languages can handle only simple datatypes: for example, integers, dates, and strings. Spatial database applications must handle complex datatypes like 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 upon 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 like 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 object-relational database management systems (OR-DBMS). The key feature of OR-DBMS is that they support 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 abstract data types (ADT). 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 open GIS (OGIS) consortium was formed by major software vendors to formulate an industry-wide standard related to GIS interoperability. OGIS spatial data model can be embeded in a variety of programming languages, e.g., C, Java, SQL etc. We will 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 non-instantiable (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 set of operations

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

Table 3.9: A sample of operations listed in the OGIS standard for SQL [OGIS, 1999]

which 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 applicable 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 which test for topological relationships between spatial objects. For example, `intersect` tests whether the interior (see Chapter 2) of two objects has a non-empty set intersection.

3. General operations 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.

3.5 Example Queries Which 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.

```
CREATE  TABLE   Country(
        Name    varchar(30),
        Cont    varchar(30),
        Pop     Integer,
        GDP     Number,
        Shape   Polygon);
```

(a)

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

(b)

```
CREATE  TABLE   City (
        Name    varchar(30),
        Country varchar(30),
        Pop     integer,
        Shape   Point );
```

(c)

Table 3.10: Basic datatypes

1. **Query:** Find the names of all countries which are neighbors of *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 and set predicates specified in the OGIS Standard. One of the nice properties 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 DBMS do not support object-oriented notions of class inheritance and operation overloading. Thus for all practial purposes, these operations must 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.

```
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?

```
SELECT C1.Name, R1.Name
FROM   City C1, River R1
WHERE  Distance (C1.Shape, R1.Shape)   <
                ( SELECT Distance(C2.Shape, R2.Shape)
                FROM     City C2, River R2
                WHERE    C1.Name <> C2.Name
                         AND R1.Name <> R2.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 which are within 300 km. List the cities which can use water from the St. Lawrence.

```
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 flood-plain management and urban and rural zoning laws. A graphical depiction of the buffer operation 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, while City `C` will remain unaffected.

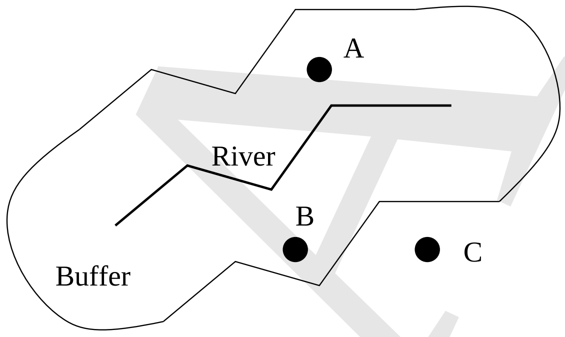

Figure 3.2: The Buffer Of a River and Points Within and Outside

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

```
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 be performed before the `Area` can be calcuated. 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.

```
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 the 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.

```
SELECT  Co.GDP, Distance(Point(0,Ci.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 DBMS are geared toward online transaction processing (OLTP), while this query, as posed, is in the realm of online 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.y)` is a point on the equator which has the same longtiude as that of the current capital instantiated in `Ci.Name`.

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.

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
```

```
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 which have only one neighbor. The `HAVING` clause plays a role similar to the `WHERE` clause with the exception that it must include aggregate functions like `count`, `sum`, `max`, and `min`.

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)
```

Co.Name	Co.GDP	Dist-to-Eq(in Km).
Havana	16.9	2562
Washington D.C.	8003	4324
Brasilia	1004	1756
Ottawa	658	5005
Mexico City	694.3	2161
Buenos Aires	348.2	3854

Table 3.11: Results of query 7

3.6 Trends: Object-Relational SQL

The OGIS standard specifies the datatypes and their associated operations which are considered essential for spatial applications like 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 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. While this feature is clearly an advantage, especially when dealing with non-traditional database applications like 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 which 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. *Abstact Datatype (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`:

```
CREATE TYPE  Point   (
          x  NUMBER,
          y  NUMBER,

   FUNCTION  Distance(:u Point,:v Point)
             RETURNS NUMBER
                    );
```

   The colons before `u` and `v` signifies 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`.

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

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

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

In this text we emphasise 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 object-relational DBMS introduced by the Oracle Corporation. Similar products are available from other database companies, e.g., IBM. It 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.

```
 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:

```
    CREATE  TABLE    City (
            Name     varchar(30),
            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

```
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`:

```
   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`

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

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

```
 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.

```
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.

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

(a)

```
CREATE  TYPE Polygon AS OBJECT (
        Num_of_Points INT,
        Geometry PolyType ,
        MEMBER FUNCTION Area(SELF IN) RETURN NUMBER,
        PRAGMA RESTRICT_REFERENCES(Length, WNDS);
```

(b)

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

(c)

```
INSERT INTO  Country('Mexico', 'NAM', 107.5, 694.3, 1004.0,
             Polygon(23, Polytype(Point(1,1), ..., Point(1,1)))
```

(d)

Table 3.12: The sequence of creation of the Country table

3.6.3 Example Queries

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

```
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 predicate 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.

```
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 avaliable 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.

```
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.

Relational algebra (RA) is the formal query language language associated with the relational model. It is rarely, if ever, implemented in a commercial system but forms the core of structured query language(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 funtions, including aggregate functions to analytically process queried data.

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

SQL3/SQL 1999 is the standardization platform for the object-relational extension of SQL. The draft 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, Ullman and Widom, 1999, Ramakrishnan, 1998].

3.4, 3.5 Extensions of SQL for spatial applications are explored Egenhofer 1994; and Roussopolulos et al., 1987. The OGIS document [OpenGIS, 1998], is an attempt to harmonize the different versions of SQL for geospatial For an example of query languages in supporting spatial data analysis, see [Lin and Huang, 2001].

3.6 SQL3 is the prospposed standard for the object-relational extension of SQL. Though still in draft form, subsets of the draft have already been implemented in commercial products, including Oracle's Oracle8 and IBM's DB2. A copy of the current draft is available at http://www.nssn.org.

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 five hundred billion dollars but less than one trillion dollars.
   (b) List the life-expectancy in countries which have rivers originating in them.
   (c) Find all cities which 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 one hundred 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 which are in North America or whose capital cities have a population of less than five million.
   (d) Find the country with the second highest GDP.
4. The `Reclassify` is an aggreate function which 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?
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 ignoring the others.

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 constranintsby 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

7. The OGIS standard includes a set of *topological* spatial predicates. How should the standard be extended to include `directional predicates` like `East, North, North-East,` and so forth. Note that the directional predicates are inherently 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 template matrix of `DE-9IM` is shown below.

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

The key difference between `9IM` and `DE-9IM` is that instead of testing whether each entry in the matrix is empty or non-empty, 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 it does not matter what the value of the matrix entry is. The following is the list of values that the matrix entries can span.

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

F: $dim(X \cap Y) = -1$. X and Y must not intersect.

$*$: 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.

$$\begin{pmatrix} T & * & F \\ * & * & F \\ * & * & * \end{pmatrix}$$

(a) What is the signature matrix (matrices) of the `touch` and `cross` topological operations. Note that 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.

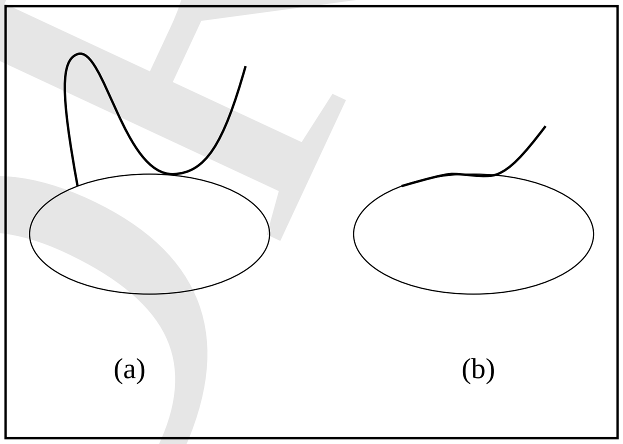

Figure 3.3: Sample objects [Clementini and Felice, 1995]

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 which lie completely south of the equator.

10. Given the schema:

RIVER(NAME:char, FLOOD-PLAIN:polgon, GEOMETRY:linstring)
ROAD(ID:char, NAME:char, TYPE:char, GEOMETRY:linstring);
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 which cross Itasca State Forest.

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

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

(d) No urban development is allowed within 2 miles of the Red river and 5 miles of the Big Tree State Park. Identify the land-parcels and the county they are in which cannot be developed.

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

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(<u>S</u>id: integer, Sname: string, Area: float, Distance: float)

Lake(<u>L</u>id: integer, Lname: string, Depth: float, Main-Catch: string)

ParkLake(<u>L</u>id: 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 entites `StatePark` and `Lake`. `ParkLake` identifies lakes which 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`.

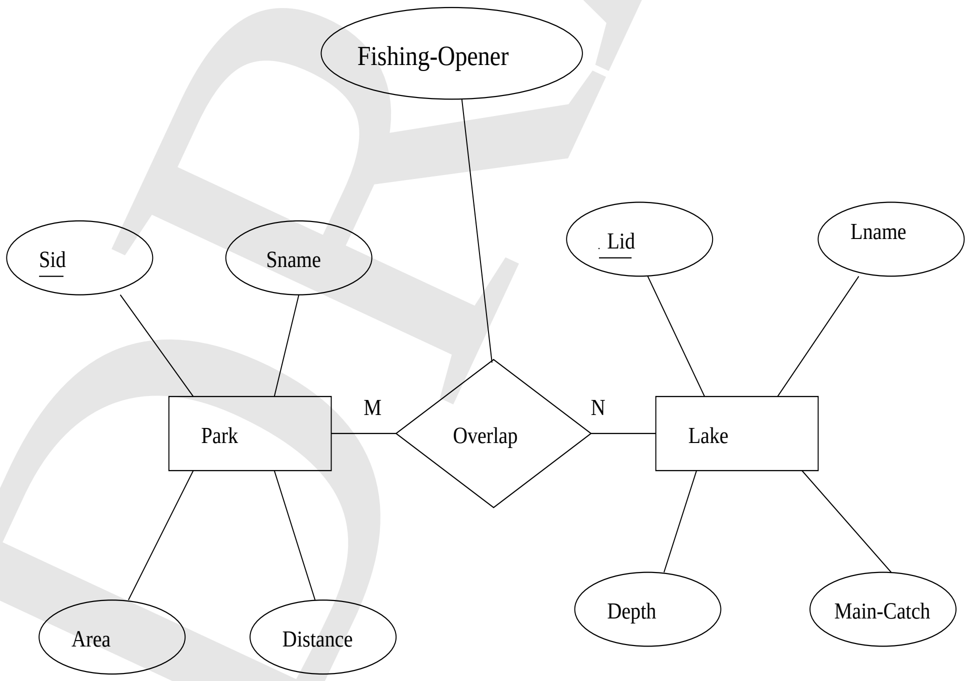

Figure 3.4: The ER diagram of the `StatePark` database

Park	Sid	Sname	Area	Distance
	S1	Itasca	150.0	52
	S2	Woodbury	255.0	75
	S3	Brighton	175.0	300

(a) Park

Lake	Lid	Lname	Depth	Main-Catch
	100	Lino	20.0	Walleye
	200	Chaska	30.0	Trout
	300	Sussex	45.0	Walleye
	400	Todd	28.0	Bass

(b) Lake

ParkLake	Lid	Sid	Fishing-Opener
	100	S1	05/15
	200	S1	05/15
	300	S3	06/01

(c) ParkLake

Table 3.13: Tables for the `StatePark` database

3.8.1 Example Queries in Relational Algebra

We now give examples that show how the relational operators defined above 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 relational algebra, and finally we make comments about the algebraic expression, including an alternate form of the algebraic expression.

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

$$\pi_{\text{Spname}}(\text{StatePark} \bowtie \sigma_{\texttt{Lid}=100}(\text{ ParkLake}))$$

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 ρ. 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,

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

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 the above query into parts as follows:

$$\begin{aligned} \rho(Temp1, \sigma_{Lid=100}(ParkLake)) \\ \rho(Temp2, Temp1 \bowtie StatePark) \\ \pi_{Spname}(Temp2) \end{aligned}$$

An alternate formulation of the query is

$$\pi_{spname}(\sigma_{Lid=100}(ParkLake \bowtie StatePark)$$

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 `Main-Catch` is `Trout`.

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

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 which are in Itasca State Park*

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

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

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

$$\pi_{\text{Spname}}(\texttt{StatePark} \bowtie \text{ ParkLake})$$

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.*

$$\rho(\text{TempLake}, \sigma_{\text{Main-Catch = Bass'}}(Lake) \cup \sigma_{\text{Main-Catch = 'Walleye'}}(Lake)$$
$$\pi_{\text{spname}}(\text{TempLake} \bowtie \text{ParkLake} \bowtie StatePark)$$

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`.

Query: *Find the names of* `StateParks` *which have both bass and walleye as the* `Main-Catch` *in their lakes.*

$$\rho(TempBass, \pi_{Spname}(\sigma_{Main-Catch=\text{`}Bass'} \bowtie ParkLake \bowtie StatePark))$$
$$\rho(TempWall, \pi_{Spname}(\sigma_{Main-Catch=\text{`}Walleye'} \bowtie ParkLake \bowtie StatePark))$$
$$TempBass \cap TempWall$$

Comment: This query formulation is barely right!

Query: *Find the names of the* `StateParks` *which have at least two lakes.*

$$c\rho(Temp, \pi_{Sid,Spname,Lid}(StatePark \bowtie ParkLake))$$
$$\rho(Temppair, Temp \times Temp)$$
$$\pi_{Spname}\sigma_{(Sid1=Sid2)\wedge(Lid1\neq Lid2)}Temppair.$$

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

$$\pi_{sid}(\sigma_{distance>50}StatePark) \quad -$$
$$\pi_{sid}((\sigma_{main-catch='Trout'}Lake \quad \bowtie \quad ParkLake \bowtie StatePark$$