# [DM-01-003] Relational DBMS and their Spatial Extensions

## Abstract

The relational Database Management System (DBMS) is widely used in modern business systems. Entities and relationships from a data model are presented as relational tables. To store data in a relational database, a relation schema should be defined to specify the design and structure of relations. The schema design generally uses database normalization to reduce data redundancy and maintain data integrity. Users can retrieve and manage data in a relational database using Structured Query Language (SQL). To make spatial data fit the relational model, spatial vector geometry or raster data type can be customized by extending basic data types in relational databases. This further helps derive the so-called spatial object-relational DBMS, by manipulating vector geometry and/or raster data types as spatial objects using SQL queries. The performance of queries is improved by adding spatial indexes in relational databases.

*Keywords*: database management systems, Normal Form, RDBMS, Spatial Extension, SQL, Transaction

## Author & citation

Yue, P. and Tan, Z. (2022). Relational Database Management Systems (DBMSs) and their Spatial Extensions. The Geographic Information Science & Technology Body of Knowledge (4th Quarter 2022 Edition), John P. Wilson (ed.). DOI: 10.22224/gistbok/2022.4.2.

The topic is also available in the following editions:

DiBiase, D., DeMers, M., Johnson, A., Kemp, K., Luck, A. T., Plewe, B., and Wentz, E. (2006). Relational DBMS. The Geographic Information Science & Technology Body of Knowledge. Washington, DC: Association of American Geographers. (2nd Quarter 2016, first digital).

and

DiBiase, D., DeMers, M., Johnson, A., Kemp, K., Luck, A. T., Plewe, B., and Wentz, E. (2006). Extensions of the relational model. The Geographic Information Science & Technology Body of Knowledge. Washington, DC: Association of American Geographers. (2nd Quarter 2016, first digital).

## Explanation

1. Definitions
2. Relational DBMS Overview
3. Fundamentals of Relational DBMS
4. Spatial Extensions of Relational DBMS

## 1. Definitions

**Relational DBMS**: A relational Database Management System (DBMS) is a collection of programs that support CRUD (Create, Read, Update, and Delete) operations and administration on relational databases. The data in a relational database are organized in the form of relational tables based on the theory of relational models proposed by E. F. Codd (1980). The relational model is established on rigorous mathematical concepts, and is a kind of logical data models (see Logical Data Models). Both the entity and entity-relationship can be presented in relational tables. The relational database is popular for its simplicity, flexibility, and easy-to-use characteristics.

**Relation**: A relation corresponds to a 2-dimensional data table.

**Tuple**: One row or record of a relational table is a tuple.

**Attribute**: One column of a relational table corresponds to an attribute. Each column has a name for identification.

**Domain**: The value range of an attribute defines its domain.

**Key**: One or a group of attributes that can uniquely identify a tuple is called a **key** or **candidate key**. The attribute(s) of the key is called **prime attribute(s)**, otherwise, they are non-prime attributes. If there is more than one key in a relation, one of them can be designated as the **primary key**. If one attribute in a relation is the primary key of another relation, then the attribute is referred to as the **foreign key**.

**Schema**: Relation schema defines the structure of a relation. It describes how the data is organized in relational tables, formally represented as relationship name (attribute 1, attribute 2, ..., attribute n).

## 2. Relational DBMS Overview

A relational database management system (DBMS) is built on the theory of relational models proposed by E. F. Codd (1980). The data in a relational DBMS are organized in the form of relational tables. Taking the land parcel as an example, the parcel entity has some attributes, such as shape, area, and land use. It can be organized like Table 1, formally presented as Parcel (Parcel_ID, Shape, Area, Landuse). It follows a relation schema with basic data types predefined. In terms of unstructured data such as geometries or images, they can be stored using the binary large object (BLOB) instead of normal field data (See Section 4 below for details). Several fundamental concepts associated with the relational model are defined above.

**Table 1. Land Parcel Information in a Relational Table**

| Parcel ID (INT) | Shape (GEOMETRY) | Area (FLOAT) | Landuse (CHAR) |
|---|---|---|---|
| 1 | POLYGON (...) | 20.57 | Commercial |
| 2 | POLYGON (...) | 50.64 | Agriculture |
| ... | ... | ... | ... |

Users can access and manage data in databases using Structured Query Language (SQL). SQL was proposed in 1974 and become the international standard for relational DBMSs in 1987 (Chamberlin, 2012). It is a comprehensive, powerful, and simple language, capable of data query, manipulation, definition, and control. Basic SQL queries and other operations provided by relational DBMSs can be written as statements in the form of commands and are integrated as small programs enabling users to add, modify or retrieve data from database tables (Date, 2011). Listing 1 demonstrates how to create the Parcel table in a database and how to get the number of records where the Area is great than 10 hectares.

```
-- create a relational table
CREATE TABLE Parcel
(Parcel_ID INT PRIMARY KEY,
 Shape GEOMETRY,
 Area FLOAT,
 Landuse CHAR(50)
 );
```

```
-- retrieve records with a condition
SELECT COUNT(*)
FROM Parcel
WHERE Area > 10;
```

Listing 1. Examples of SQL for creating a data table and querying data. Source: authors.

In a relational DBMS, the basic single unit of processing is called a transaction. The transaction possesses four important properties: Atomicity, Consistency, Isolation, and Durability (ACID). ACID guarantee the consistency of the database when performing the read, write, and modification tasks in a database.

- Atomicity: Each statement in a transaction is treated as an atomic unit. Either the entire statement is executed, or none of it is executed.
- Consistency: Data in the database is always in a consistent state when a transaction starts and when it ends.
- Isolation: The execution of one transaction cannot interfere with another transaction. The intermediate state of the transaction is isolated from other transactions.
- Durability: Changes to data are persistent and cannot undo after a transaction is completed.

## 3. Fundamentals of Relational DBMS

### 3.1 Relational Data Model

Data modeling is the process to map entities and relationships into data structures supported by databases. In the conceptual modeling phase (see Conceptual Data Models), the entities, relationships, and attributes should be analyzed based on the business needs. The entity-relationship (ER) diagram (see Conceptual Data Models, Logical Data Models) is a commonly used approach to perform this task. A set of symbols, such as rectangles, ovals, diamonds, and connecting lines, are defined in ER diagrams to represent entities,

associated attributes, and their relationships. For example, a land parcel management system requires the following information for parcels, such as the parcel shape, the owner, and irrigation wells. Figure 1 illustrates the parcel, owner, and well entities, and relationships between them. The 1:n  notation in Figure 1 indicates the two entities have a one-to-many relationship. Specifically, one person can own several land parcels, and several wells can be located in one parcel.

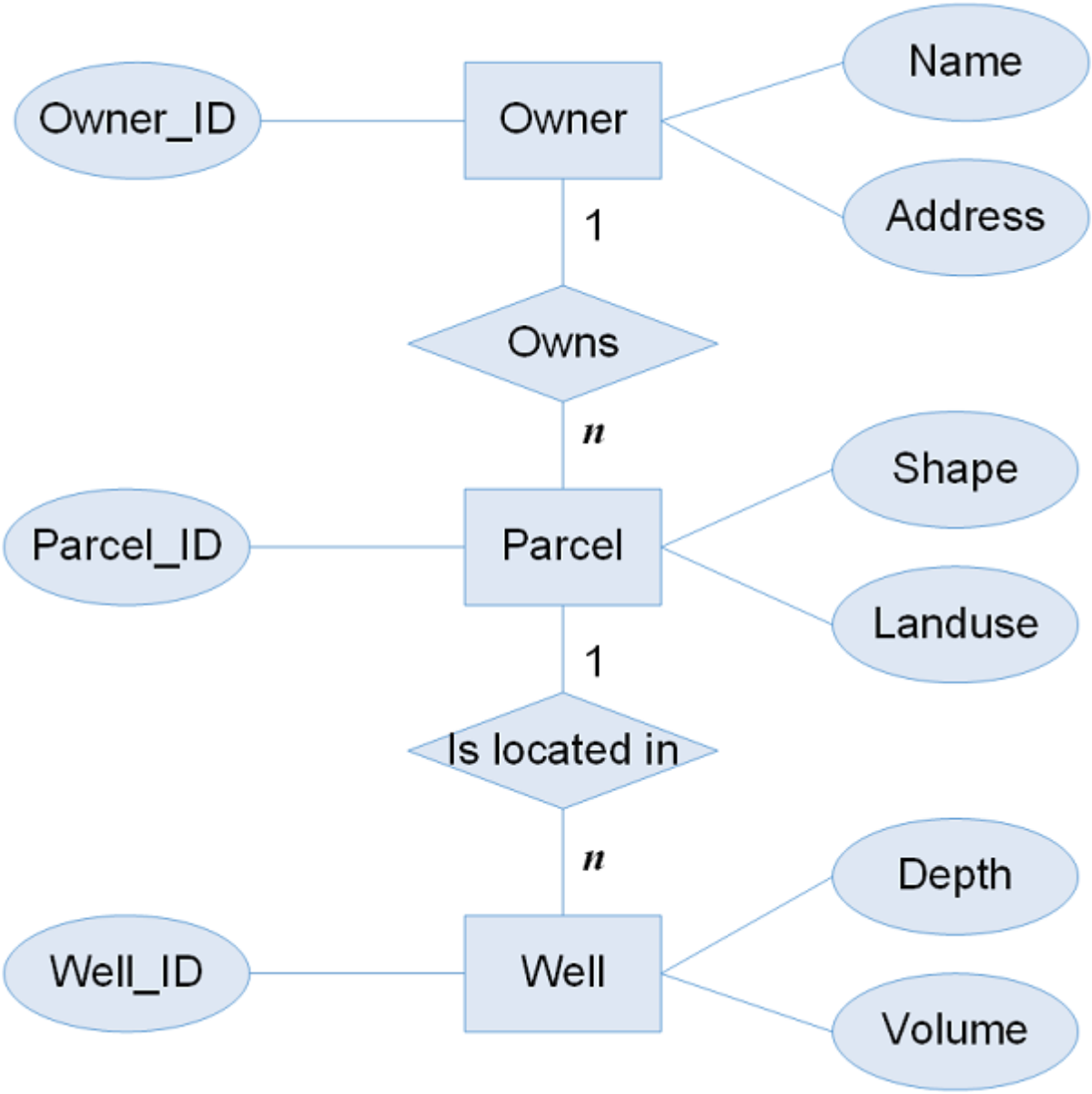


Figure 1. The ER diagram of the land parcel. Source: authors.

In the logical modeling phase (see Logical Data Models), the ER diagram should be translated into specific data structures supported by the underlying database. For a relational DBMS, the ER diagram needs to be translated into relational tables.  Usually, an entity is converted into a relational table. For example, the Parcel entity can be formally presented as a Parcel table: Parcel (Parcel_ID, Shape, Landuse). The primary key in the relation is underlined. A relationship between two entities can be converted into a new table or merged into existing tables as new columns. For example, the relationship between

the Parcel and Owner entities can be converted to a new table: Parcel-Owner (Parcel_ID, Owner_ID), or the Owner can be attached to the Parcel table: Parcel (Parcel_ID, Shape, Landuse, Owner_ID). The conversion can be fine-tuned using database normalization. After finishing the logical modeling, the physical modeling (see Physical Data Models) should be conducted including the design of the data index and storage structure.

### 3.2 Relational operation

The relational operations include query, insert, delete, and update operations. The query operation employs relational algebra to perform queries and produce results as a new relation. Some primary operations of relational algebra are **selection**, **projection**, **cartesian product**, **union**, **intersection**, and **difference** (Figure 2). The selection and projection operations are used to select rows and columns respectively in a table. The cartesian product, union, intersection, and difference operations are defined from the set theory to operate on two tables. Additional operations can be derived from these operations. For example, the join operation can be implemented with the cartesian product and selection operations. The **inner join** results in the intersection of two tables (Figure 3a). The union of two tables is the outer join, which can be classified into the **left**, **right,** and **full join** (Figures 3b-d). In addition, indexing is often employed in databases to improve the performance of operations (see Physical Data Models). For example, in a relational DBMS the B-tree and B+ tree are commonly used as indexing methods on data.

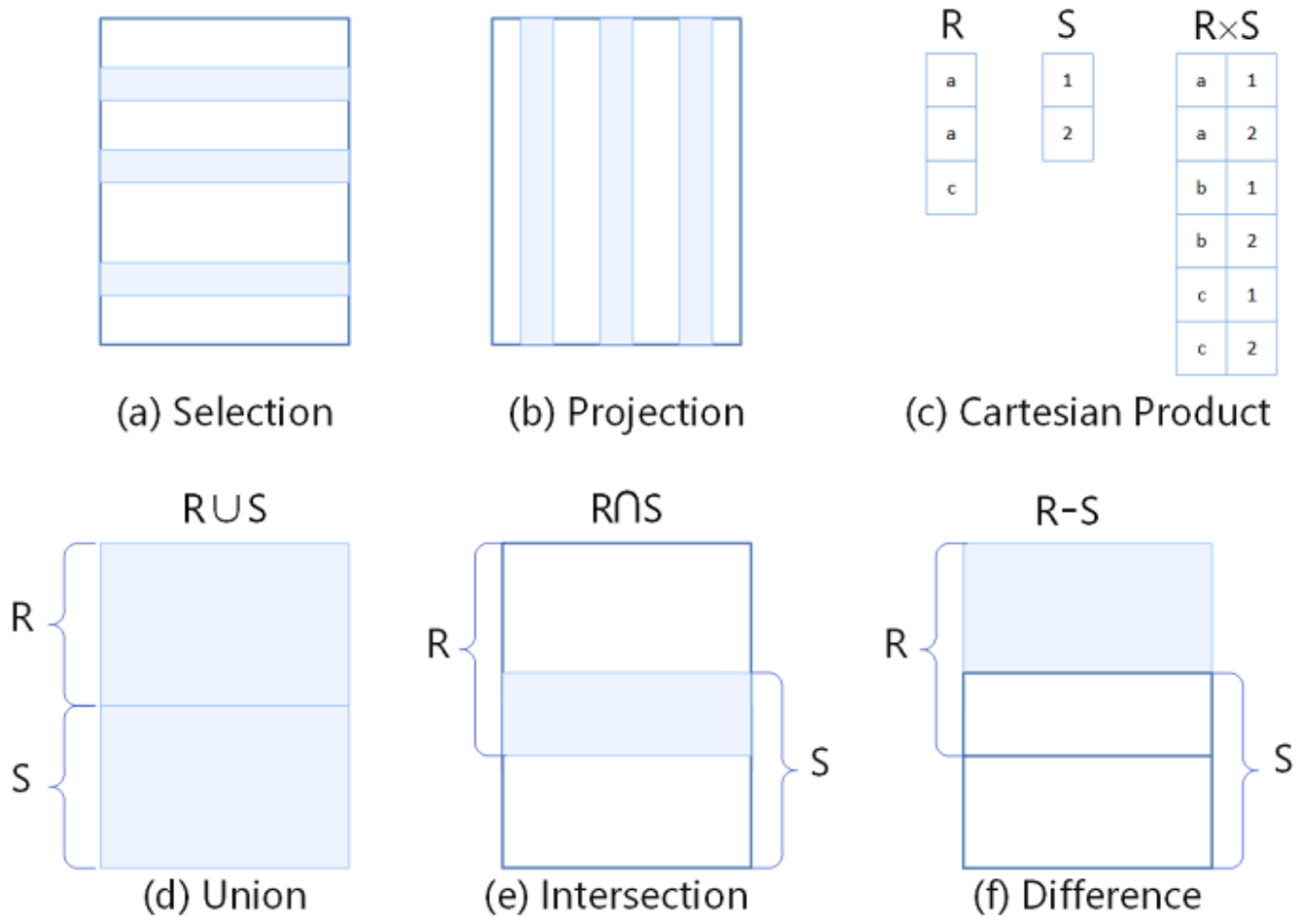

Figure 2. Some operations in relational algebra. Source: authors.

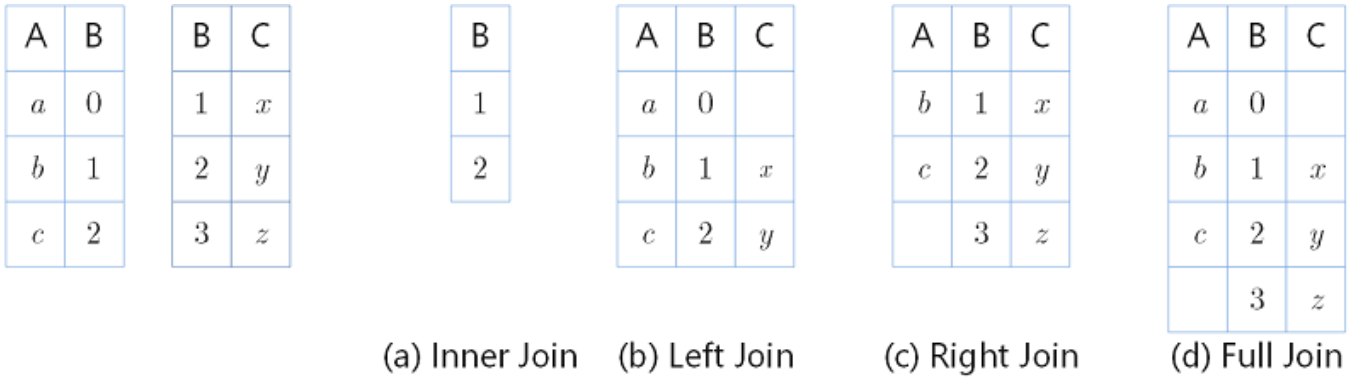


Figure 3. Join operations in relational algebra. Source: authors.

### 3.3 Normalization

Database normalization is used to guide the design of relational tables to reduce data redundancy and maintain data integrity (Kent, W., 1983). There are various levels of normalization, among which the first normal form (**1NF**), second normal form (**2NF**), and third normal form (**3NF**) are main forms. Take a land parcel database as an example, the following demonstrates how to employ the normal form to optimize relational tables. **1NF** states that each column contains atomic values that cannot be divided. In this aspect, a single table Parcel-Well-Owner (Parcel_ID, Shape, Landuse, Well_ID, Depth, Volume, Owner_ID, Name, Address) containing all entities and relationships can be constructed. The attributes (Parcel_ID, Well_ID) can uniquely identify each tuple, and they serve as a composite key in the relation.

There are some issues regarding the Parcel-Well-Owner relation in 1NF. For example, if there is no well in a specific parcel, the parcel record cannot be inserted into the database because this violates the entity integrity constraint that prime attributes in a relation cannot be empty. In this case, **2NF** can be used to solve this problem caused by composite keys based on the concept of function dependency. In the relational theory, a functional dependency is a relationship between two or two groups of attributes. Specifically, if a non-prime attribute can be uniquely determined by attribute subset of the candidate key, the dependency between the key and the non-prime attribute is called **partial functional dependency.** Otherwise, it is referred to as **full functional dependency**. As shown in Figure 4a. Prime attributes are marked in the orange color and functional dependencies are marked with arrows. The key is composed of two attributes (Parcel_ID, Well_ID). The attributes Shape and Landuse are fully dependent on the Parcel_ID. That makes them partially dependent on the composite key (Parcel_ID, Well_ID), and so do the attributes Depth and Length. The 2NF can then be employed to remove the partial functional dependencies in the relation. As shown in Figure 4b, the Parcel-Well-Owner relation is decomposed into two relations: the Parcel-Owner and Well. There is no partial functional dependency in the two relations and the insertion anomaly can be solved.

The 3NF is based on the concept of transitive functional dependency. If a non-prime

attribute r1 can uniquely determine another non-prime attribute r2 in a relation, the dependency between the prime attribute(s) and non-prime attribute r2 is called **transitive functional dependency**. For example, the non-prime attribute Name is functionally dependent on the Owner_ID, and the Owner_ID is functional dependent on the prime attribute Parcel_ID, then the Name is transitively dependent on the Parcel_ID. If one person owns five parcels, the owner information needs to be altered in five tuples once the address is updated. The **3NF** is used to remove the transitive functional dependency in a relation. As shown in Figure 4b, the attributes Name and Address are transitively dependent on the Parcel_ID in the Parcel-Owner relation. To eliminate this transitive functional dependency, the Parcel-Owner relation is decomposed into another two relations (Figure 4c): the Parcel and Owner.

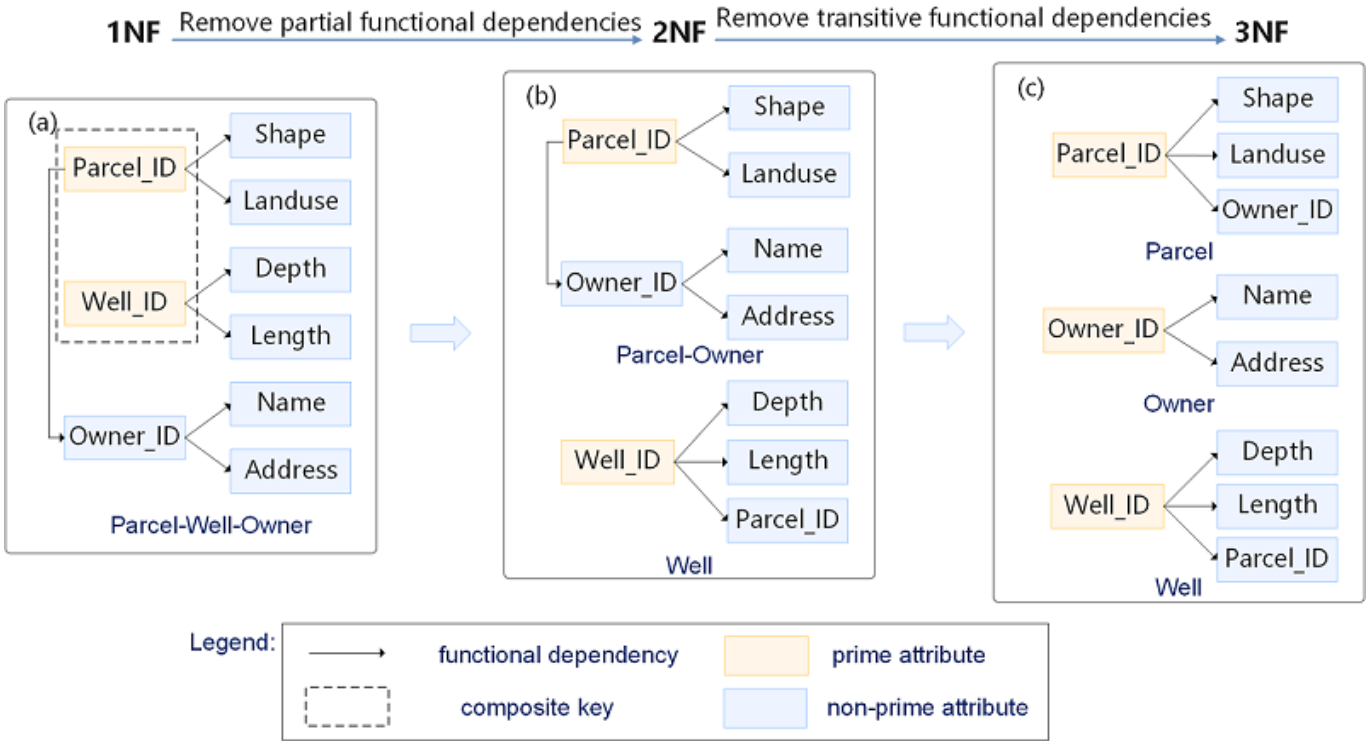


Figure 4. Normalization of relations. Source: authors.

## 4. Spatial Extensions of Relational DBMS

### 4.1 Spatial Data in Relational DBMS

Spatial data are often represented using vector or raster data structures. To accommodate them in a relational data model, specific strategies are often needed (Yue & Tan, 2018). With regards to vector data, spatial entities such as points, lines, and polygons can be converted into new relational tables. The images as unstructured data are often stored using the BLOB column in a relational table. It is also possible to store spatial entities as a new BLOB column in existing tables, since the geometries are also a kind of unstructured data composed by a list of coordinates. Modern relational DBMS borrow ideas from object-oriented paradigm, and defines spatial data types (e.g. point, lines, and polygons) as

objects on top of BLOB, thus making a relation database capable of operating on spatial objects. This is the so-called object-relational database.

Figure 5a shows a relational approach without explicit support of spatial data types. The spatial entities and relationships can be organized into different new relational tables. A parcel has boundaries and the boundaries are defined by road segments in order. Each segment has start and end nodes, and each node has x and y coordinates to identify its position. Thus, the spatial information of a parcel can be represented using three relations: Node (ID, X, Y, Name), Segment (ID, Start, End, Length), and Parcel (ID, Segment, Area, Landuse). A complex spatial entity can be assembled by joining multiple data tables. Although this approach fits the relation data model, it stays at the theoretical level and does not perform well in practice. The multi-table join operation significantly lowers the performance of queries and make the vector data update difficult. Thus at the beginning of this century, some GIS vendors such as ArcGIS and GeoStar, employ BLOB, a built-in type in relational DBMS, to host the spatial data, shown in Figure 5b. Specific GIS middleware for spatial data storage such as ArcSDE is needed to process the BLOB to allow spatial indexing and operations. The middleware is a mediator between GIS software and relational DBMS data storage such as SQL Server or Oracle databases. It includes the definition of spatial abstract data types and their operations. These data types and operations are standardized by the Open Geospatial Consortium (OGC) through the OGC Simple Feature Access specification (Herring, 2011). The OGC Well-Known Binary (WKB) representation for geometry can also be used in BLOB.

Some database vendors also realize the aforementioned issue and provide spatial extensions in their own database products (Date & Darwen, 1998). They follow the object-relational model by extending the relational model to support the storage and management of unstructured data with customized object data types. The spatial object type in the object-relational DBMS is used to store spatial data, shown in Figure 5c. One typical example is the Oracle Spatial software. Moreover, a set of built-in functions operating on the spatial objects are provided. Spatial indexing is also supported on the spatial objects to improve the performance of queries for spatial data. Since the extensions are provided by database vendors, the solutions are often more effective and efficient than the relational approach or GIS middleware for spatial data storage.

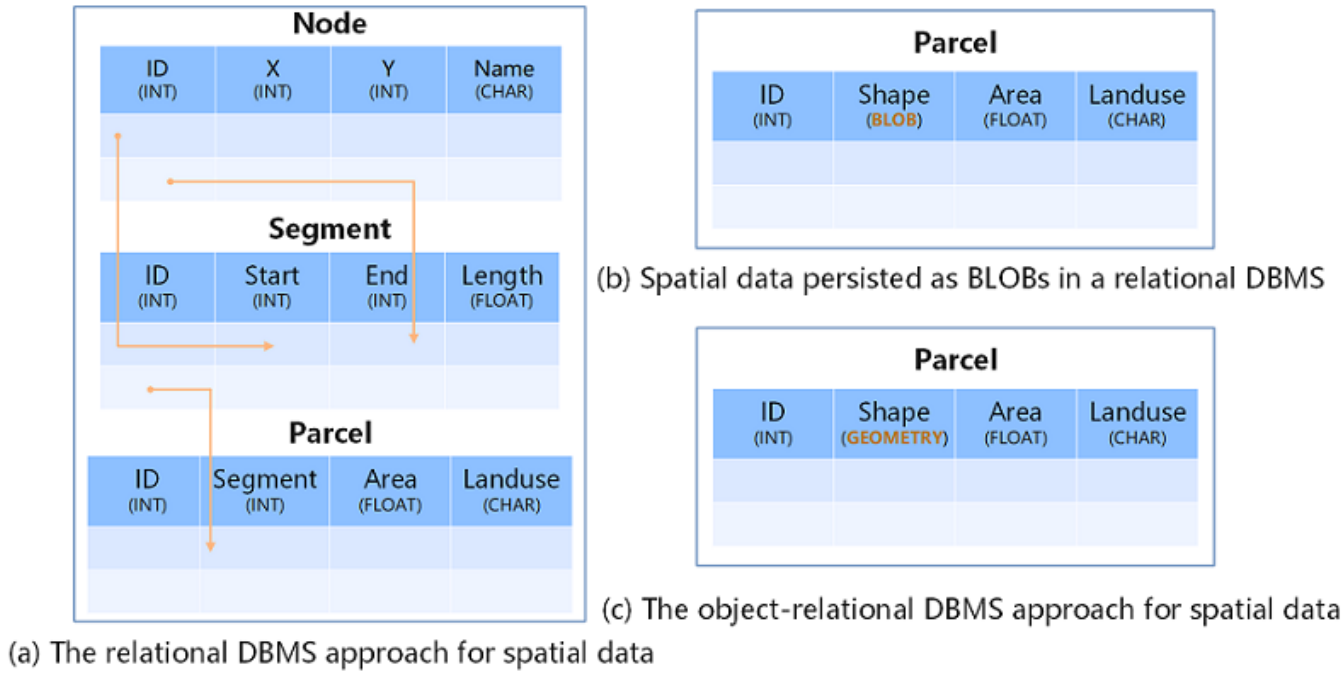


Figure 5. Different approaches to store spatial data in a relational DBMS. Source: authors.

### 4.2 Spatial Index and Operations in Relational DBMS

The indexing on spatial data differs from the non-spatial data because queries on spatial data are usually based on its location or topology among spatial objects. Spatial index (see Spatial Indexing) is an elaborately designed data structure allowing for efficient access to spatial objects. Examples of spatial indexes include grid index, quadtree, R-tree and R+ tree index.

The spatial operations are extensions in a relation DBMS for querying and manipulating spatial objects (Güting, 1994). Many spatial relational DBMSs support spatial operations according to the OGC Simple Feature Access specification (SFA) for SQL (Herring, 2010). The spatial operations can be classified into three categories: basic methods on geometry, such as Dimension() and Boundary(); methods for testing spatial relations between geometric objects, such as Equals() and Within(); methods that support spatial analysis, such as Buffer() and Intersection(). There is also a standard from ISO, ISO/IEC 13249-3 SQL Multimedia - Spatial standard (SQL/MM), which extends the OGC SFA with more Geometry subtypes such as curves and topology-network primitives.

## 4.3 Cases of spatial relational DBMSs

At present, many relational DBMSs have provided geometry types for spatial features. Some even support spatial raster data. For example, Oracle Spatial provides customized data types and sophisticated storage schema for spatial vector and raster data (Kothuri, 2004). PostGIS is an object-relational spatial database extension of PostgreSQL in the open source community, and has built-in geometry and raster object types for spatial data (Hsu, 2021). Table 2 provides some examples of relational DBMS approaches for managing spatial data.

**Table 2. Examples of Relational DBMS Approaches for Managing Spatial Data**

| Database Product | Characteristics |
|---|---|
| Oracle Spatial | Oracle Spatial provides a SQL schema and functions that facilitate the storage and retrieval of spatial data in an Oracle database. A spatial feature is stored as a geometry type in a column of a table, which is called a customized spatial type SDO_GEOMETRY. Spatial raster data are managed with SDO_GEORASTER and SDO_RASTER types in tables. |
| PostGIS | PostGIS is an open-source and OGC-compliant spatial database. PostGIS provides built-in geometry and raster object types to support spatial vector and raster following an object-relational approach. |
| Esri ArcSDE | ArcSDE is a middleware developed by Esri to extend existing relational DBMSs to store, retrieve and manage spatial data. ArcSDE provides a common interface for users regardless of underlying relational DBMSs. |
| SpatiaLite | SpatiaLite extends the SQLite core to support spatial vector data and enable spatial SQL capabilities. It is simple and lightweight. A whole database is wrapped into a single file that can be transferred simply. |

## References


Chamberlin, D. D. (2012). Early History of SQL. IEEE Annals of the History of Computing, vol. 34, no. 4, pp. 78-82.

Codd, E. F. (1980). Data models in database management. ACM SIGMOD Record - Proceedings of the workshop on Data abstraction, databases and conceptual modelling, 11(2), 112-114.

Date, C. J. (2011). SQL and Relational Theory: How to Write Accurate SQL Code, 2nd Edition. O'Reilly Media, Inc.

Date, C. J., & Darwen, H. (1998). Foundation for Object/Relational Databases: the Third Manifesto. Addison Wesley Longman Publishing Co., Inc.

Güting, R. H. (1994). An introduction to spatial database systems. The VLDB Journal, 3(4), 357-399.

Kent, W. (1983). A simple guide to five normal forms in relational database theory. Communications of the ACM, 26(2), 120-125.

Kothuri, R., Beinat, E., & Godfrind, A. (2004). Pro Oracle Spatial. Apress.

Obe, R. and Hsu, L. S. (2021). PostGIS in Action. Simon and Schuster.

Open Geospatial Consortium (OGC) (2010). OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option. OGC 06-104r4, Open Geospatial Consortium Inc.

Open Geospatial Consortium (OGC). (2011). OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture. OGC

06-103r4, Open Geospatial Consortium Inc.

Yue, P., & Tan, Z. (2018). GIS Databases and NoSQL Databases. In: Huang, B. (Ed.), Comprehensive Geographic Information Systems. Vol. 1, pp. 50–79. Oxford: Elsevier.