Database Management Systems

What is a DBMS?

Database

- Collection of different types of data
- Most commonly stored in form of tables

DBMS/RDBMS

- Database management system is a software which is used to manage the database. For example: MySQL, Oracle, MS SQL Server, etc. are very popular databases which are used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database along with user management.

Typical Objects in a DBMS

Table

- Storage of data in form of rows and columns

View

- A commonly used query stored for ease of access and also to provide security

Synonym/alias

- A different name given to a table for ease of identity for different purpose

Index

- Sorting of records for quicker access to frequently used data

Procedures/Functions/Triggers

- Database scripts/programs used to perform data manipulation/transformation using a combination of SQL statements

Types of Databases

Relational Databases
- Data stored in forms of tables which are related
- Stored in form of rows and columns
- SQL is the most commonly used language for data operations
- E.g. Oracle, SQL Server, IBM Db2, etc.

NOSQL Databases
- Do not use a defined schema for storage
- Can be used to store semi-structures or un-structured data
- Do not use SQL as common query language
- E.g. MongoDB, Cassandra, etc.

Columnar Databases
- Data stored in columns rather than rows
- Commonly used for data warehousing
- E.g. Google BigQuery, Cassandra, Hbase, etc.

Types of Databases

Document Databases

- Store and manage documents oriented data/semi-structured data
- E.g. MongoDB, Amazon DocumentDB, etc.

Graph Databases

- Data stored as relationships and actual data
- Data represented as nodes and the nodes are related using named relationships
- E.g. Neo4J

Timeseries Databases

- Data stored along with timeseries information
- Most commonly used for event based data resulting from sensors, networks, performance monitoring systems
- E.g. Druid

Emerging Databases

Active Databases

An active database is a database that includes an event-driven architecture which can respond to conditions both inside and outside the database. Possible uses include security monitoring, alerting, statistics gathering and authorization

The Active Database consists of set of triggers.

- Triggers are the rules that specify actions that are automatically triggered by certain events.
- The model that has been used to specify active database rules is referred to as the **Event-Condition-Action** (**ECA)** model.
- Event(E) that triggers the rule (Addition/Modification/Deletion of row).
- Condition(C) that determines whether the action is to be executed (SQL Condition).
- Action(A) to be taken once the event occurs and the condition is satisfied (SQL Statements/Procedures).

Triggers

- A procedure that runs automatically when a certain event occurs in the DBMS
- Implementation of the ECA rules
- The procedure performs some actions, e.g.,
  - Check certain values
  - Fill in some values
  - Inserts/deletes/updates other records
  - Check that some business constraints are satisfied

Types of Triggers

- How many times should the trigger body execute when the triggering event takes place?
  - Per statement: the trigger body executes once for the triggering event. This is the default. (Bulk transation statements)
  - For each row: the trigger body executes once for each row affected by the triggering event.

- When will the trigger is fired
  - Before or after the execution of the DML statement
  - Situation dependent (System Clock) (Time dependent)

Trigger Example

The following trigger checks for temperature value in the Temperature table after insertion of a new record.

```
CREATE OR REPLACE TRIGGER ALERT_TEMP

AFTER INSERT ON TEMP_TABLE

FOR EACH ROW

WHEN NEW.TEMP_ID>0

BEGIN
  IF :NEW.TEMP_VALUE>60 THEN
                DBMS_OUTPUT.PUT_LINE ('High Temperature Alert, Temperature is ' || :NEW.TEMP_VALUE);

END;
```

Trigger Uses in real life

- Auditing database operations
  - Every time a transaction takes place the details (auditing information) are recorded
- Storing of historical values in a record
  - Each time an attribute/record is changed the previous value is stored
- Maintaining Data Integrity
  - When a master record is deleted (E.g. Order Data) the corresponding details to be deleted (Order Items)

Advantages of active database

1. Enhances traditional database functionalities with powerful rule processing capabilities.
2. Enable a uniform and centralized description of the business rules relevant to the information system.
3. Avoids redundancy of checking and repair operations.
4. Suitable platform for building large and efficient knowledge base and expert systems.

Challenges

Active Database is a database consisting of set of triggers. These databases are very difficult to be maintained because of the complexity that arises in understanding the effect of these triggers. In such database, DBMS initially verifies whether the particular trigger specified in the statement that modifies the database) is activated or not, prior to executing the statement.

If the trigger is active then DBMS executes the condition part and then executes the action part only if the specified condition is evaluated to true. It is possible to activate more than one trigger within a single statement.

In such situation, DBMS processes each of the trigger randomly. The execution of an action part of a trigger may either activate other triggers or the same trigger that Initialized this action. Such types of trigger that activates itself is called as ‘recursive trigger’. The DBMS executes such chains of trigger in some pre-defined manner but it effects the concept of understanding.

Deductive Databases

A Deductive Database is a database system that can make deductions or conclusions based on some rules and facts stored in the database.

It is a database system which includes capabilities to define deductive rules, via which we can deduce or infer additional information from the facts stored in a database

Runs on inference engine or a deductive mechanism

The basic foundation for DDBs is mathematical logic

Database + Inference becomes a deductive database

Advantages

Part of the data (which can be derived) can be stored implicitly rather than explicitly storing saving the storage space in the database

Rules allow to store new kind of data, such as recursive data

Uses the power of logic programming

Facts and rules

Facts can be considered as the data stored as relations in a relational database.Facts are specified in a manner similar to the way relations are specified, except that it is not necessary to include the attribute names.

Rules are somewhat similar to relational views. They specify virtual relations that are not actually stored but that can be formed from the facts by applying inference mechanisms based on the rule specifications

Facts and Rules

1. Facts are specified in a manner similar to relations, but not necessarily including attribute names. In a DDB the meaning of an attribute is determined by its position in a tuple

2. Rules are similar to relational views. They specify virtual relations that are not actually stored but rather can be deduced from the facts by applying inference mechanisms based on the rule specifications

Main difference between rules & views is that rules can involve recursion

The evaluation of Prolog programs is based on backward chaining, which involves a top-down evaluation of goals

The evaluation of Datalog programs is roughly based on bottom-up evaluation

In Prolog the order of specification of facts and rules and the order of literals are significant, for the evaluation

in Datalog an effort has been made to circumvent these problems

Predicates

The notation used in Prolog/Datalog is based on providing predicates with unique names

A predicate has an implicit meaning, which is suggested by the predicate name, and a fixed number of arguments

If the arguments are all constant values, the predicate simply states that a certain fact is true

If the predicate has variables as arguments, it is either considered as a query or as part of a rule or constraint

All constant values in a predicate are either numeric or character strings starting with lowercase letters, whereas variable names always start with an uppercase letter

Logical Deductions

All trees have leaves. Banyan is a tree. Conclusion – Banyan tree has leaves

If I have money, I will help you. I have money, Conclusion – I will help you

Mathematical logic

If A=B and B=C then A=C

Negative Inference, if A is even then it is divisible by 2 (B). If 9 is not divisible by 2, then it is not even

All computer science courses are interesting. DBMS is a computer science course. Therefore DBMS is interesting.

Model and Languages

In a DDB we specify rules through a declarative language (what, not how, e.g. SQL)

An inference engine or deduction mechanism in the DDB can deduce new facts from the database by interpreting these rules

model used for DDBs is closely related to:

1. Relational Data Model (domain relational calculus)

2. Logic programming, Prolog language

a variation of Prolog, Datalog is used to define rules declaratively.

the existing set of relations is treated as a set of literals in the language

Datalog syntax is similar to Prolog syntax

Datalog semantics <> Prolog semantics

Examples

Example based on the company database

3 predicate names: supervise, superior, subordinate

a) The supervise predicate: defined via a set of facts, each with two arguments: (supervisor name, supervisee name)
supervise expresses the idea of direct supervision

facts correspond to actual data stored in the database

facts correspond to tuples in a relation with two attributes and schema: SUPERVISE(SUPERVISOR, SUPERVISEE)

Note the absence of attribute names

In facts attributes are represented by position. 1st argument is the supervisor, 2nd argument is the supervisee.
supervise(X,Y) states the fact that X supervises Y.

b) The superior, subordinate predicates: defined via a set of rules (superior allows us to express the idea of non-direct supervision)

MAIN IDEA OF DDBs specify rules and a framework to infer (deduce) new information based on these rules

Rule Semantics

A rule means that if a binding of constant values to the variables in the body (RHS) of a rule makes all its predicates TRUE, then the same binding makes the head (LHS) TRUE

A rule provides a way to generate new facts, instantiations of the head of the rule. These new facts are based on already existing facts, corresponding to bindings of predicates in the body of the rule

Note that listing multiple comma-separated predicates in the body or a rule, implicitly means that they are connected by AND Example the superior predicate of the company database (direct/indirect supervision)

superior(X,Y):-supervise(X,Y)

superior(X,Y):-supervise(X,Z),superior(Z,Y)

NOTE 1: a rule may be defined recursively

NOTE 2: when two (or more) rules have the same (LHS) head predicate, the head predicate is true if the 1st body is TRUE OR if the 2nd body is TRUE

Proof Theoretic Interpretation

Facts and rules are true statements or axioms. Ground axioms contain no variables.

Facts are ground axioms that are taken to be true.

Rules are called deductive axioms because they can be used to deduce new facts. The deductive axioms are used to derive new facts from existing facts

Provides a procedural approach to compute the answer to a Datalog query.

Model Theoretic Interpretation

Specify a (usually) finite domain of constant values.

Assign to a predicate every possible combination of values as arguments

Determine if the predicate is true or false

In practice, we specify the combinations of argument values that make the predicate true and state that all other combinations make the predicate false.

When this is done for all predicates, this is called an interpretation of this particular set of predicates. An interpretation for the 2 predicates supervise, superior

An interpretation assigns a truth value (T, F) to every possible combination of argument values (taken from a finite domain) for a set of predicates.

Model Theoretic Interpretation

An interpretation is called a model for a specific set of rules, when these rules are always true in this interpretation. (e.g. for any values assigned in the variables in the rules, the head is true, when the body is true, rules are not violated)

This interpretation is a model for superior Q when is a rule violated?

A particular bindings of the arguments of the predicates in the rule body make it true, and simultaneously make the rule head predicate false.

Example I denotes a certain interpretation Supervise(a, b) is TRUE in I

Superior(b, c) is TRUE in I

Superior(a, c) is FALSE in I

then, I can't be a model for the (recursive) rule

superior(X,Y) :- supervise(X,Z),superior(Z,Y)

Inserting and Querying using Datalog

Facts are stored in tables. If the name of the table is parent, and john is the parent of douglas, store the fact in the database with this:

```
> parent(john, douglas).
```

Each item in the parenthesized list following the name of the table is called a *term*.

A term can be either a logical variable or a constant. Thus far, all the terms shown have been constant terms.

A query can be used to see if a particular row is in a table. Type this to see if john is the parent of douglas:

```
> parent(john, douglas)?
parent(john, douglas).
```

Type this to see if john is the parent of ebbon:

```
> parent(john, ebbon)?
```

The query produced no results because john is not the parent of ebbon. Let's add more rows.

```
> parent(bob, john).
> parent(ebbon, bob).
```

Type the following to list all rows in the parent table:

```
> parent(A, B)?
parent(john, douglas).
parent(bob, john).
parent(ebbon, bob).
```

Inserting and Querying using Datalog

Type the following to list all the children of john:

```
> parent(john, B)?
parent(john, douglas).
```

A term that begins with a capital letter is a logical variable. When producing a set of answers, the Datalog interpreter lists all rows that match the query when each variable in the query is substituted for a constant. The following example produces no answers, as there are no substitutions for the variable A that produce a fact in the database. This is because no one is the parent of oneself.

```
> parent(A, A)?
```

A deductive database can use rules of inference to derive new facts. Consider the following rule:

```
> ancestor(A, B) :- parent(A, B).
```

The rule says that if A is the parent of B, then A is an ancestor of B. The other rule defining an ancestor says that if A is the parent of C, C is an ancestor of B, then A is an ancestor of B.

In the interpreter, DrRacket knows that the clause is not complete, so by pressing Return, it doesn't interpret the line.

Rules are used to answer queries just as is done for facts.

```
> ancestor(A, B)?
ancestor(ebbon, bob).
ancestor(bob, john).
ancestor(john, douglas).
ancestor(bob, douglas).
ancestor(ebbon, john).
ancestor(ebbon, douglas).
```

Inserting and Querying using Datalog

```
> ancestor(X,john)?
ancestor(bob, john).
ancestor(ebbon, john).
```

A fact or a rule can be retracted from the database using tilde syntax:

```
> parent(bob, john)~
> parent(A, B)?
parent(john, douglas).
parent(ebbon, bob).
> ancestor(A, B)?
ancestor(ebbon, bob).
ancestor(john, douglas).
```

Unlike Prolog, the order in which clauses are asserted is irrelevant. All queries terminate, and every possible answer is derived.

```
> q(X) :- p(X).
> q(a).
> p(X) :- q(X).
> q(X)?
q(a).
```

Main Memory Databases

Main Memory Databases

A main memory database system is a DBMS that primarily relies on main memory for computer data storage. In contrast, conventional database management systems typically employ hard disk based persistent storage.

Advantages

The main advantage of MMDBMS over normal DBMS technology is superior performance, as I/O cost is no more a performance cost factor. With I/O as main optimization focus eliminated, the architecture of main memory database systems typically aims at optimizing CPU cost and CPU cache usage, leading to different data layout strategies (avoiding complex tuple representations) as well as indexing structures (e.g., B-trees with lower-fan-outs with nodes of one or a few CPU cache lines).

While built on top of volatile storage, most MMDB products offer ACID properties, via the following mechanisms: (i) Transaction Logging, which records changes to the database in a journal file and facilitates automatic recovery of an in-memory…

ACID properties of Database

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

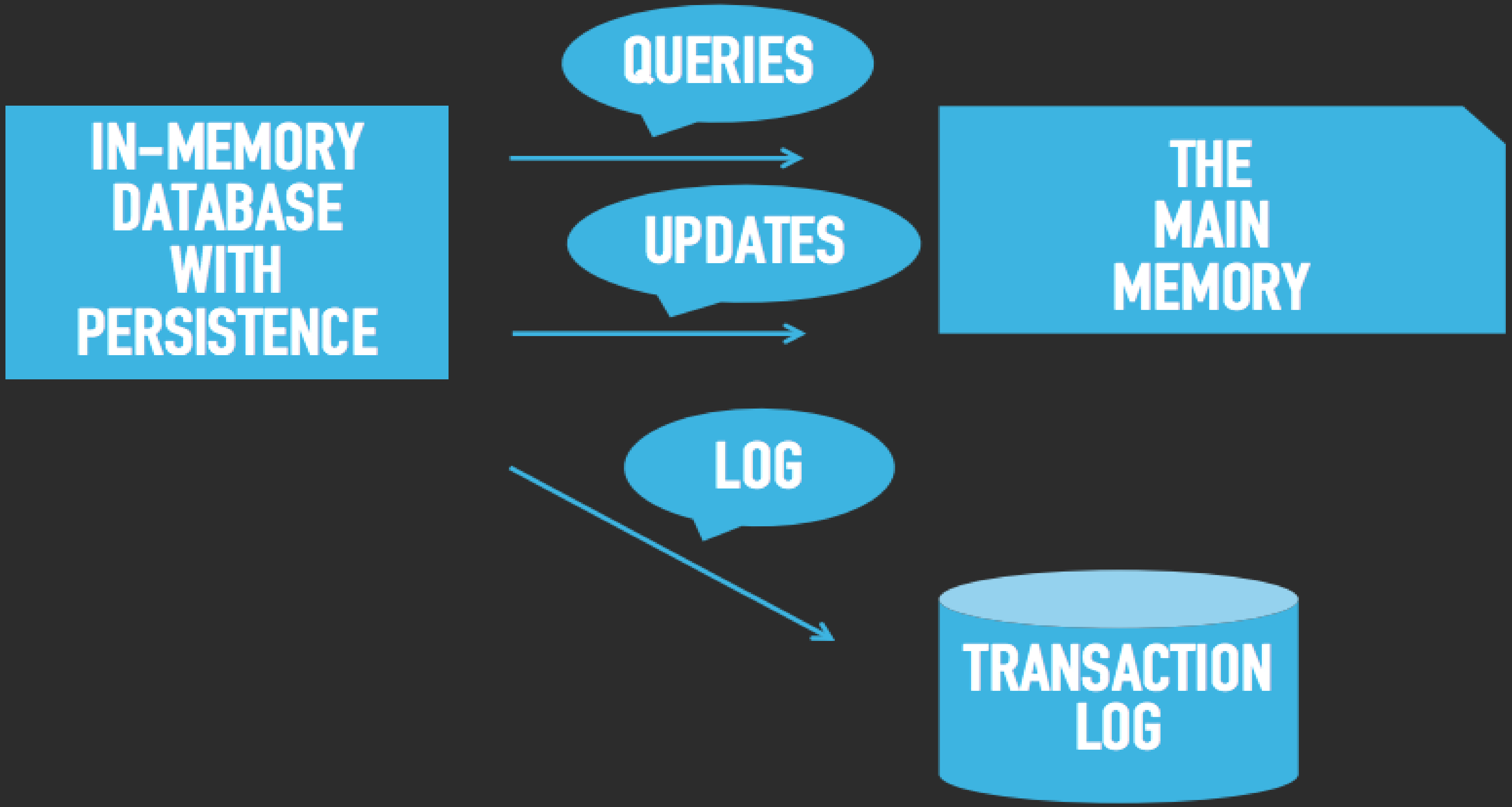
Working of IMDB

In-memory databases work faster than databases with disk storage. This is because they use "internal" optimization algorithms, which are simpler and faster, and this type of system requires fewer CPU instructions than a disk storage system. Additionally, accessing data that has been stored "in-memory" eliminates the need for seek time while searching for data. As a consequence, several data warehouse vendors are switching to in-memory technology to speed up the processing of data. The cloud also presents an opportunity for using in-memory databases.

Traditionally, data has been stored on disk drives, with RAM used for short-term memory while the computer is in use. in-memory database architecture uses a database management system that relies primarily on a computer's main memory (RAM), and is organized by an In-Memory Database Management System (IMDBMS). In-memory database (IMDB) architecture requires a management system designed to use the computer's main memory as the primary location to store and access data, rather than a disk drive.

Working of IMDB

An in-memory database has advanced in-memory data structures optimized for working with memory. Direct pointers are used to manage interrelationships between different parts of the database. Algorithms are simpler and transactions require fewer copy operations and CPU instructions. The data is used as-is, without any translating or caching required. Sharing data between applications and tasks is also simple and quick because the data is stored in one place and accessed through a single process with multiple threads.


IN-MEMORY DATABASE WITH PERSISTENCE
QUERIES
UPDATES
THE MAIN MEMORY
LOG
TRANSACTION LOG

Uses of IMDB

Though in-memory database systems do have broad uses, they are used primarily for real-time applications requiring high performance technology. The use cases for these systems include applications for real-time responses, such as with the finance, defense, telecom, and intelligence industries. Applications requiring real-time data access such as streaming apps, call center apps, reservations apps, and travel apps also work well with IMDBMS.

The two primary reasons in-memory databases have not historically been popular have to do with costs and a lack of ACID (atomicity, consistency, isolation, and durability) compliance. The lack of "durability," refers to the IMBDs loss of memory, should the electricity be cut. Also, RAM has, historically, been fairly expensive, and this has stunted the growth and evolution of in-memory databases. Recently, the cost of RAM has begun to drop, making IMBDs more affordable.

Comparison with traditional DB

Memory vs. Storage

Storage is for data that is currently not being used, but has been recorded on a hard disk, can be saved indefinitely and be recalled as needed. Data stored on a disk is permanent unless erased. Hard drive storage is generally used for long-term storage purposes. Traditionally, hard drives were designed to save much larger amounts of data than RAM. That situation is changing.

RAM is a physical component, not a software program. It uses computer chips (integrated circuits) that are soldered to the main logic board, or, as is true of many personal computers, uses a plug-in system for the easy upgrading of memory modules (also known as DRAM modules). Using an IMDB instead of a disk drive system provides the following benefits:

- RAM can be increased to improve performance with relative ease.
- Additional RAM allows a computer to do more at once (but does not actually make it faster).
- Additional RAM improves the switching between different applications and allows multiple applications to be open without causing the system to become sluggish.
- It uses less power than disk drives.

Types of RAM

There are two basic types of RAM: DRAM (Dynamic Random Access Memory) and SRAM (Static Random Access Memory). RAM has been used as a form of short-term memory for computer use. The word used to describe RAM’s loss of memory when the electricity is cut off is “volatile.”

- **DRAM:** The term “dynamic” indicates the memories must constantly be refreshed. DRAM is generally used as the main memory in computers. RAM must be refreshed thousands of times each second.
- **SRAM:** Typically used as a system cache. (A smaller, faster memory that is closer to a processor core.) It stores copies of regularly used data from its main memory, and is described as “static” because it does not need to be refreshed. However, SRAM is also volatile, and loses its memories when the power is cut.
- **NVRAM:** Non-volatile RAM

Benefit of Cloud

The cloud provides an excellent environment for getting the most out of in-memory computing. A cloud environment offers organizations the ability to access large amounts of RAM at will. This approach can help organizations avoid the expense of an on-premises in-memory computer.

The cloud can also provide an environment that makes in-memory storage more reliable through the use of redundant hosts and virtual machines using automatic failover. With these measures, the disruption of RAM will not lead to a data loss. These protective measures are more difficult to develop in an on-premises computer system. Combining the cloud and in-memory computing provides an excellent way to maximize the benefits an in-memory system.

Semi-Structured Data

Structured Data

What is Structured Data?

Structured data is information that is formatted and stored into a well-defined data model. The raw data is mapped into predesigned fields that can then be extracted and read through SQL easily.

SQL relational databases, consisting of tables with rows and columns, are the perfect example of structured data.

Structured data is more inter-dependent and less flexible.

Semi-Structured Data

Semi-structured data is the data which does not conform to a data model but has some structure. It lacks a fixed or rigid schema. It is the data that does not reside in a rational database but that has some organizational properties that make it easier to analyze. With some process, we can store them in the relational database.

Characteristics of semi-structured Data:

- Data does not conform to a data model but has some structure.
- Data can not be stored in the form of rows and columns as in Databases
- Semi-structured data contains tags and elements (Metadata) which is used to group data and describe how the data is stored
- Similar entities are grouped together and organized in a hierarchy

Evolution of Semi-Structured Data

The increase in digitization of almost everything we interact with, along with multiple transactions has resulted in a massive amount of data. The tremendous increase in the speed of digital information has led the global data to double in very short time intervals. As per Gartner, around 80% of data with organization is **unstructured data/semi-structured data** which is comprised of data from emails, social media feeds and customer calls.

This is in addition to information logged by the user devices. It has been increasingly tough to make proper sense of this unstructured data.

Need of Analysis

Organizations need to analyze semi- structured and **unstructured data sets to extract structured data insights** to make improved business decisions. These decisions include shaping customer sentiment, finding customer needs and identifying the offerings that will relate more to the customer requirements.

While filtering big amounts of data can look like a tedious work, there are benefits. By **analyzing large data sets of unstructured data**, you can categorize connections from unconnected data sources and find specific patterns. And this analysis enables the discovery of business as well market trends.

Extraction of information

Identify Sources for the data to be analyzed

Decide on the ways and means for data extraction and loads

Create a structured data from the information

Maintain both the original as well as the converted/transformed data

Use of tools for proper analysis of data

Characteristics

- Data has some structure which, however, does not conform to the structure of a data model.
- A hierarchy is defined wherein all similar entities form a group, and such groups are organised into the hierarchy of semi structured data examples.
- It is not storable as table columns and rows like data in a relational database.
- The data, which is semi-structured, has metadata/elements and tags to help group it and describe its storage.
- The attributes in any group of items typically are different.
- The group of entities in a group may not or may have the same properties and attributes.
- Semi structured data is hard to manage or automate as its metadata is insufficient and hence cannot be put into a table with rows & columns.
- Programming such data is difficult as it lacks a sufficient defined structure.

Examples/Sources

- Sources of semi-structured Data:
  - E-mails
  - XML and other markup languages
  - Binary executables
  - TCP/IP packets
  - Zipped files
  - Integration of data from different sources
  - Web pages

Advantages

- The data is not constrained by a fixed schema
- Flexible i.e Schema can be easily changed.
- Data is portable
- It is possible to view structured data as semi-structured data
- Its supports users who can not express their need in SQL
- It can deal easily with the heterogeneity of sources

Disadvantages

- Lack of fixed, rigid schema make it difficult in storage of the data
- Interpreting the relationship between data is difficult as there is no separation of the schema and the data.
- Queries are less efficient as compared to structured data.

Storage limitations

- In comparison to structured data, the costs of storage with semi-structured data is always higher.
- As data and its schema are dependent on each other and tightly coupled, queries on such data cause frequent updates to the schema as well as the data. The same query can update the original data schema more frequently than is desirable.
- Semi-structured data typically has a partially-structured structure. Some of it may have a definite data structure, while other sets may not have a conforming structure making it difficult to interpret and understand the data relationships.
- Due to the lack of distinction between the data and its schema, designing the data structure can often become a complicated issue.

Nested Data Types

Nested Data Types

Nested data types are structured data types for some common data patterns. Nested data types support structs, arrays, and maps.

A *struct* is similar to a relational table. It groups object properties together

As a general definition, nested data exists whenever multiple records are sampled from a single record. The data then consists of two, with header information populated at Level 1 and details or itemized information populated at Level 2

Nested Data Types – XML

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C (World Wide Web Consortium) Recommendation

Difference between HTML and XML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

Advantages of XML

- It simplifies data sharing
- It simplifies data transport
- It simplifies platform changes
- It simplifies data availability

Many computer systems contain data in incompatible formats. Exchanging data between incompatible systems (or upgraded systems) is a time-consuming task for web developers. Large amounts of data must be converted, and incompatible data is often lost.

XML stores data in plain text format. This provides a software- and hardware-independent way of storing, transporting, and sharing data.

XML also makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

With XML, data can be available to all kinds of "reading machines" like people, computers, voice machines, news feeds, etc.

Examples of XML data

SampleXML1.txt

NestedXML1.txt

Nested Data Types – JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

JSON Syntax

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

The JSON format is syntactically identical to the code for creating JavaScript objects.

Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

JSON Data

JSON Objects

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/value pairs:

```
'{"name":"John", "age":30, "car":null}’
```

JSON Arrays

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain objects:

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
```

Jason Data Examples

JasonDataSet.txt

JasonDataSetCurrency.txt

NestedJason.txt

Advantages of JSON

- **Less Verbose**: JSON has a more compact style than XML, and it is often more readable. The lightweight approach of JSON can make significant improvements in RESTful APIs working with complex systems.
- **Faster**: The XML software parsing process can take a long time. One reason for this problem is the DOM manipulation libraries that require more memory to handle large XML files. JSON uses less data overall, so you reduce the cost and increase the parsing speed.
- **Readable**: The JSON structure is straightforward and readable. You have an easier time mapping to domain objects, no matter what programming language you're working with.
- **Structure Matches the Data**: JSON uses a map data structure rather than XML's tree. In some situations, key/value pairs can limit what you can do, but you get a predictable and easy-to-understand data model.
- **Objects Align in Code:** JSON objects and code objects match, which is beneficial when quickly creating domain objects in dynamic languages.

Semantic Databases

- Semantics is the study of meaning
- It focuses on the relationship between:
  - Signifiers: words, phrases, signs and symbols
  - Denotation: what they stand for

- Semantic Database is typically used in conjunction with the Semantic Data Model
- By exposing the semantics of the data, machines can then utilize the information in more interesting ways than just storing it or displaying it
  - Semantics are useful for understanding the structure of a "thing", however we need ontologies to relate things with other things. Therefore, one would expect that a semantic database be relational -- it should be able to relate structured data into ontologies.

Semantic Databases

In a semantic database (going back to the very early definition of semantics), the schema:

- describes denotations
- describes relationships between denotations

The job of the database then is to associate signifiers (values) to those denotations. Therefore:

- Structure resolves to concrete properties to which instance values can be associated

Semantic Data Model

Semantic data model (SDM) is **a high-level semantics-based database description and structuring formalism** (database model) for databases. ... It is a conceptual data model in which semantic information is included. This means that the model describes the meaning of its instances

A method of organizing data that reflects the basic meaning of data items and the relationships among them. This organization makes it easier to develop application programs and to maintain the consistency of data when it is updated.

Semantic Repository

A semantic repository is a database management system (DBMS). They use a semantic data schema paradigm, called an ontology, which is stored and managed independently from the data. As a result, semantic repositories offer easier data integration of diverse sources as well as more analytical power.

Ontology is the branch of philosophy that studies concepts such as existence, being, becoming, and reality. It includes the questions of how entities are grouped into basic categories and which of these entities exist on the most fundamental level.

Semantic Repository

A semantic repository is a database management system. It allows storing, querying, and managing structured data. Semantic repositories is still not a largely adopted term and is often referred to by synonyms such as semantic graph database, reasoner, ontology server, semantic store, metadata store, RDF database, RDF triplestore and more. Different wording often emphasizes the particular features and usages, rather than a difference in the implementation and performance.

The major benefit of semantic repositories, compared to traditional DBMSs such as relational databases (RDBMSs), is the usage of semantic data schema paradigm, called ontology, which is stored and managed independently from the data.

Semantic and Object-Oriented Models

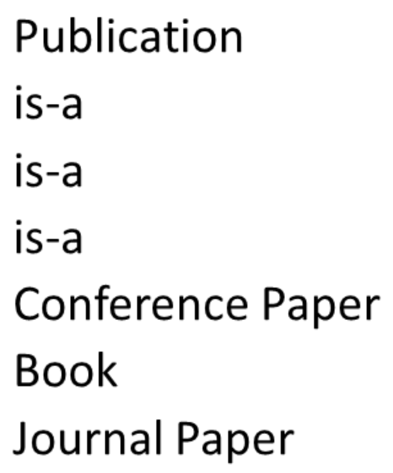
- Entity Relationship Model is entity-based.
- ER model entities are static entities identified by a key.
- Insert, delete, and modify plus graph-oriented query languages.
- Keys, weak and strong entity types, etc.
- Semantic and Object Oriented Models are object-based.
- Attempt to capture the meaning of data not only in the structure of the objects, but also in the operations upon the objects.

Semantic Modeling Primitives

- Aggregation
  - Lower level types can be aggregated into a higher level type. This construct is used group attributes or properties of a similar object into the object type.
  - The relational model aggregates attributes into relations.
  - Title and Author types can be aggregated into the type Publication
  - Street, City, State, Zip can be aggregated into Address.
  - Is-part-of or conversely Is-composed-of relationship.
- The original E/R model does not allow relationships involving other relationships.
- Aggregation is a abstraction by which a relationship (and its associated entities) are considered as high-level entities.

Semantic Modeling Primitives

- Generalization/Specialization
  - Different object types may be differentiated by a property (predicate) but when viewed by the similarities, can be generalized to a higher level type.
  - For example, the concepts of Journal_Paper, Book, and Conference_Paper can be generalized to Publication.
  - Journal_Paper IS-A Publication
  - Book IS-A Publication
  - Conference_Paper IS-A Publication
  - Publication would have attributes common to all publications while the subtypes journal paper, book and conference paper would have specialized attributes.



Semantic Modeling Primitives

- Generalization
  - Entity sets may be specialized based on properties,
  - Accounts may be saving-accounts and checking-accounts.
  - This allows each sub-type entity sets to have attributes particular to that type.
  - Conversely, we may generalize to a super-type entity set, for example Student, Secretary, Mechanic may be generalized to Employee which in turn may be generalized to Person.

Graph DB

GraphDB is one of the most popular semantic repositories.

Unlike a relational database, a graph database is **structured entirely around data relationships**. Graph databases treat relationships not as a schema structure but as data, like other values. ... The best way to understand the difference between relational databases and graph databases is to walk through a sample use case

Very simply, a graph database is a database designed to treat the relationships between data as equally important to the data itself. It is intended to hold data without constricting it to a pre-defined model. Instead, the data is stored like we first draw it out - showing how each individual entity connects with or is related to others.

Graph DB Example

Graph Database presents data as entities, or *nodes*. Nodes can have properties that have further information. Nodes are connected to other nodes with *edges*. Each connection between two nodes can be labeled with properties.

Here is a very simple Graph Database example:

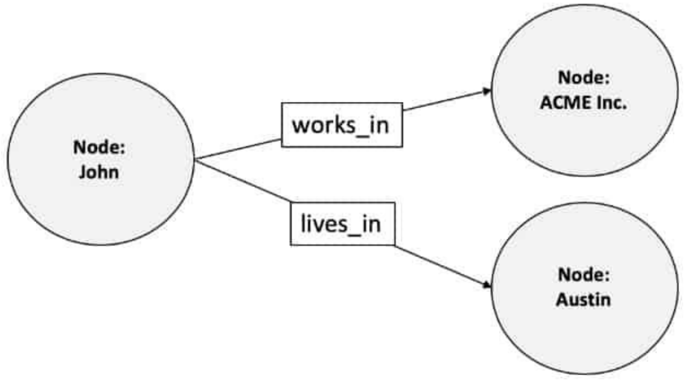
Node A: John,

Node B: ACME Inc.,

Node C: Austin,

Edge 1: works_in, Edge 2: lives_in. This database tells you that John works

in ACME Inc and he lives in Austin



Graph DB Use Cases

There is a good reason why the world's forerunner-businesses are increasingly using Graph databases. This modern technology offers unprecedented agility, scalability, and performance for managing vast amounts of highly dynamic and exponentially growing data for various use-cases — this is precisely what today's applications require.

And, the Graph database is adopted for ever more use-cases and applications as organizations continue implementing the Graph technology.

The Graph database reveals the complex and hidden relationships between separate data sets, allows you to analyze them, to further improve your business processes, and make smarter business decisions, faster.

Words are just words until put to practice.

All these use-cases have been successfully implemented in a real business environment

Comparison with RDBMS

Compared with relational databases, graph databases are often faster for associative data sets and map more directly to the structure of object-oriented applications. They can scale more naturally to large datasets as they do not typically need join operations, which can often be expensive. As they depend less on a rigid schema, they are marketed as more suitable to manage ad hoc and changing data with evolving schemas.

Conversely, relational database management systems are typically faster at performing the same operation on large numbers of data elements, permitting the manipulation of the data in its natural structure. Despite the graph databases' advantages and recent popularity over relational databases, it is recommended the graph model itself should not be the sole reason to replace an existing relational database. A graph database may become relevant if there is an evidence for performance improvement by orders of magnitude and lower latency.

Object Oriented Databases

Object Oriented DB and Programming

The object-oriented database model ties related packages together. A data set and all its attributes are combined with an object. In this way, all of the information is directly available. Instead of distributing everything across different tables, then, the data can be retrieved in one package. Alongside the attributes, methods are also stored in the objects. This is where the databases' proximity to object-oriented programming languages becomes clear. As in programming, each object has certain activities that it can carry out.

Object Oriented Database

ODBMS is a database management system that is implemented based on object-oriented data model

Object Oriented Programming Concepts

- Class
- Object
- Abstraction
- Encapsulation
- Interface
- Methods and their Signatures
- Attributes

Object Oriented Database

Main Features:

- Powerful type system
- Classes
- Object Identity
- Inheritance

Powerful Type System

Regular Types – Integer, String, Date, Float, Boolean, etc

Structure Types – Attributes of record/struct type

Collection Type – Set, List, Array, etc.

Reference Type – Reference to other object

Classes, Objects and Inheritance

A class is like a blueprint of properties and functions

All objects belonging to a same class share the same properties and behavior

Object Identity is the unique identifier for the Object, and is not visible to the end user

It is used for referencing objects

Classes inherit the properties and functions of the main/super class. E.g. If Car is a super class and Nexon is a sub-class, then Nexon will inherit the attributes and functions of class Car

Illustration of DB Class and Subclasses

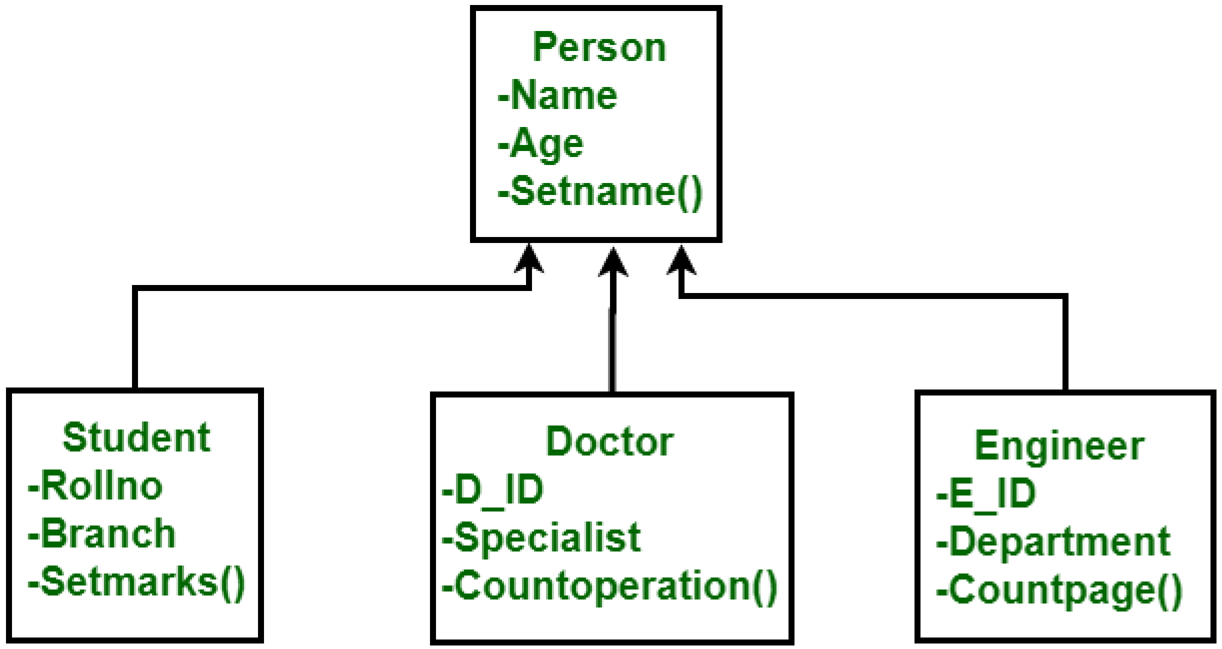
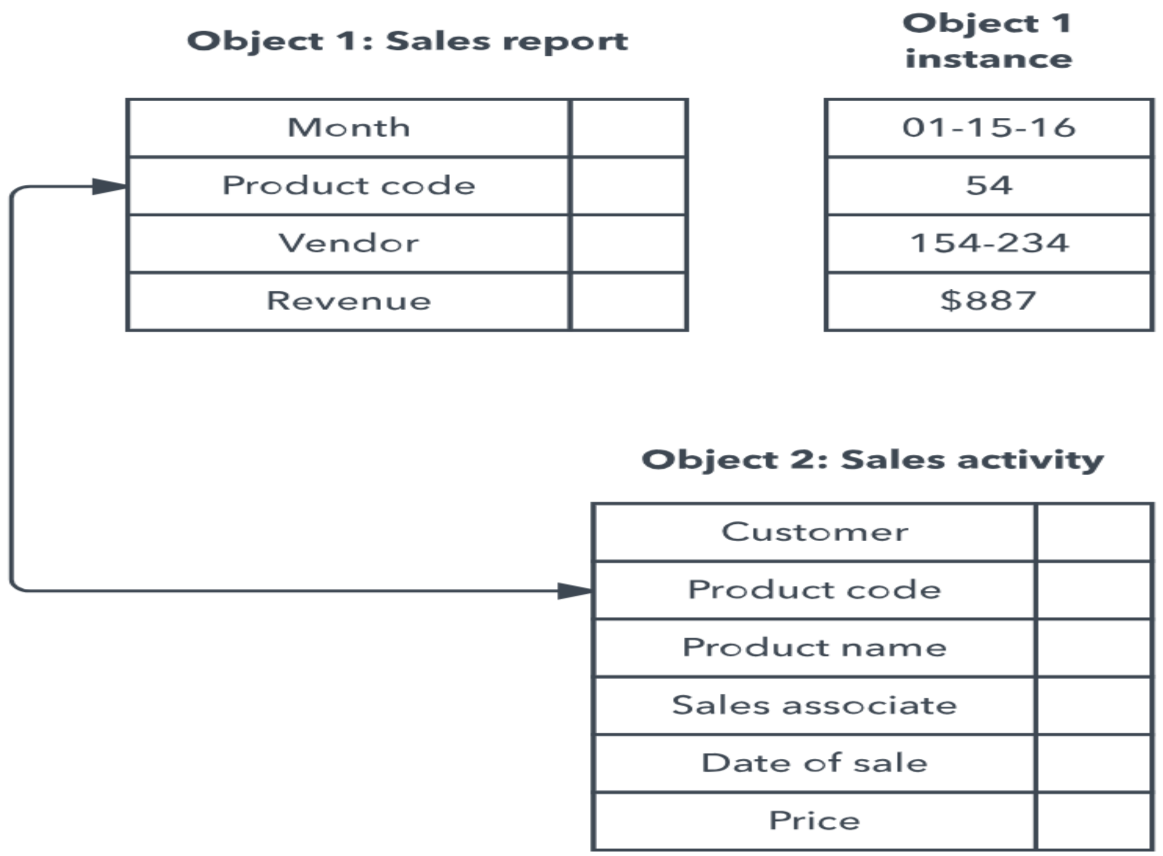


Illustration of Object Instance

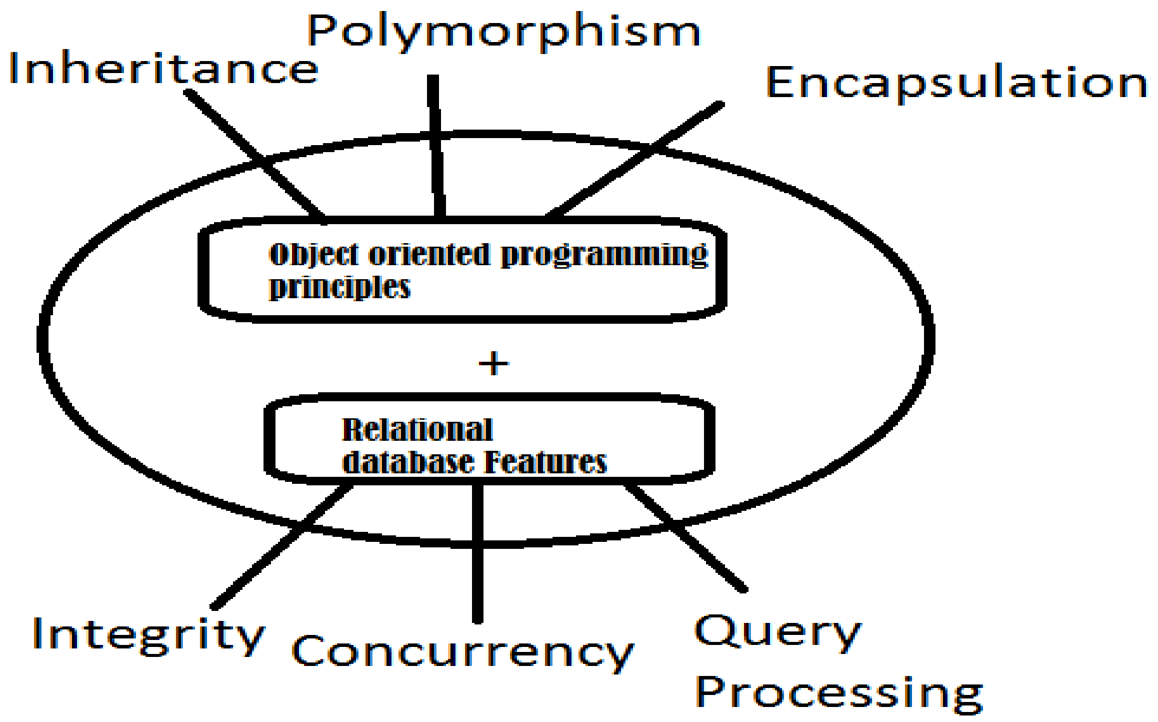


Code Example

```
class CLERK
{
//variables char name;
string address;
int id;
int salary;

//methods
char get_name();
string get_address();
int annual_salary(); };
```

RDBMS + Object Oriented



Advantages

- Code can be reused due to inheritance.
- Easily understandable.
- Cost of maintenance can reduced due to reusability of attributes and functions because of inheritance.

Disadvantages

Object databases are not widely adopted

In some situations, the high complexity can cause performance problems

Examples

Common examples are Smalltalk is used in GemStone, LISP is used in Gbase, and COP is used in Vbase. Object databases are commonly used in applications that require high performance, calculations, and faster results

Persistent Objects

Persistent objects are the objects that are created and stored in database and exist even after the program termination

Spatial Databases

Spatial Databases

Spatial data are data types (files, databases, web services) that encode geographic information for use in location-aware applications.

What is GIS?

Cartography is as old as civilization itself. From crossing oceans to planning cities, information was collected and transcribed onto paper.*

Just as we digitized text and numbers, cartographers started putting point, line, and polygon data into digital systems.

This created the **Geographic Information Systems (GIS)** domain. Spatial data could be drawn as a graphical layer on a map but also analyzed geographic characteristics and location relationships. Forest management, land use planning, transportation, surveying, and more, all took advantage of these systems.

Spatial Databases

A spatial database — also known as a "geospatial database" — is built to capture and store the points, lines, and areas of cartographic information that we refer to as spatial data. Often these databases were pretty ordinary but had extensions to handle binary objects (BLOBs) in one of their fields – SQL Server, Oracle, PostgreSQL, Ingres, SQLite all have spatial addons.

Issues grew more complex as data volumes grew and workflows morphed. Instead of just producing and sharing data, developers were being pressured to produce more information and knowledge from the data instead. There was also a drive for interoperability and efficiency.

This presented new opportunities to create standards for web-optimized geospatial services – for sharing images of maps or raw tabular data to be used in another location-aware application. While ESRI's file-based formats (**shapefiles, geodatabase**) were popular for desktop users, many web developers wanted data in an enterprise database like **PostgreSQL** which has been the leading open source spatial database for years.

Once the data was in a database, developers were able to take more control over the workflow. Likewise, an increasing number of **spatial analysts** started to use the spatial functionality of the database systems.

Spatial Database Functions

Geospatial Database Functions

Drawing Maps

The most common function for developers accessing spatial data is to make a map – whether online, in a mobile app, or on a desktop. This can take a few forms but is mainly a request for a specific set of records of documents that the application then renders for the user.

Rendering maps is a domain in its own right, but most online APIs that help do this (e.g. MapBox, Leaflet) have some basic defaults that get users up and running quickly. To be honest, most online maps just show some point markers on top of an image. So this is one use case for map data but is not really what GIS users would choose when performing analysis.

Beyond just requesting a set of documents, a spatial database needs to allow filtering for specific features/documents based on a query. Naturally, the use of SQL languages that give a WHERE clause helps make this possible. There are more advanced geographic ways of filtering data – e.g., for a specific region/area – we'll cover that as well.

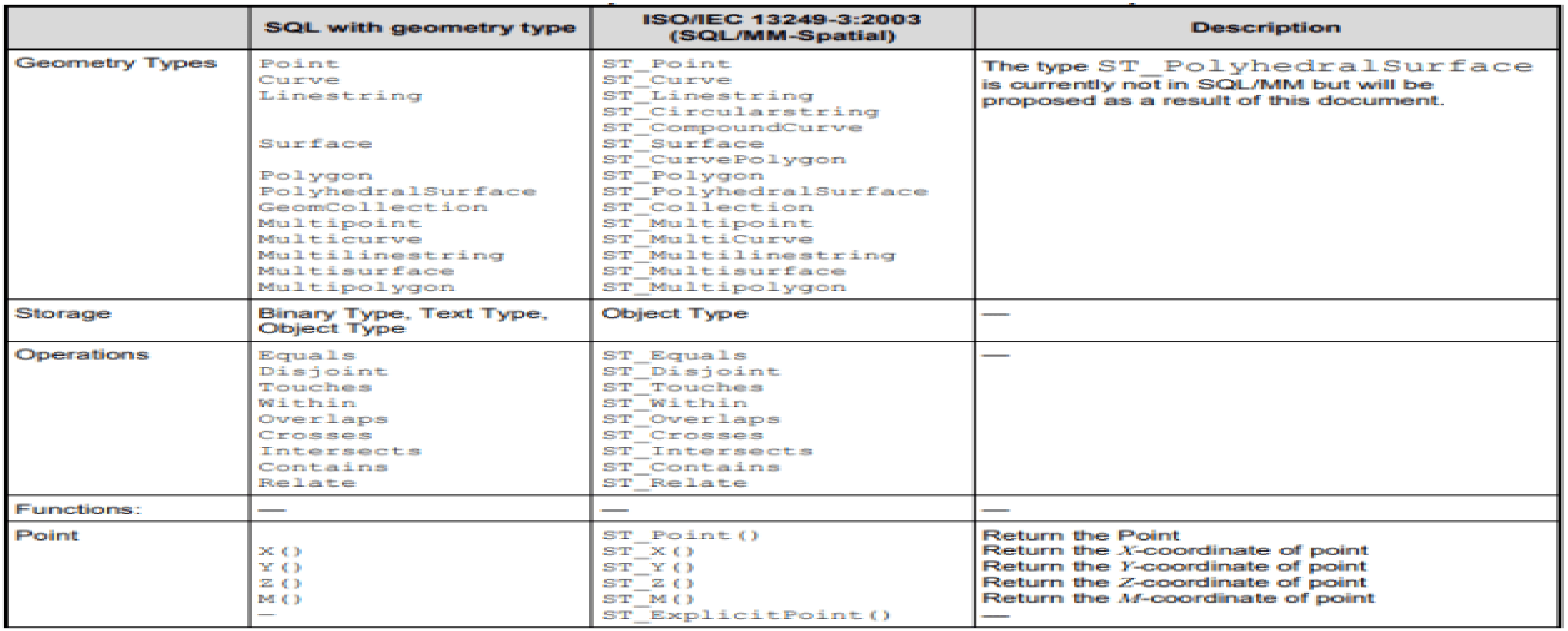
Indexing

Spatial Indexing

To do more advanced spatial filtering to draw maps, or to even do what is called spatial joins (joining records based on their proximate location, like a store within a region), you need to have the geographic data indexed. Just like you would index other fields in a database, there are special indexing methods for geospatial field types.

Spatial databases typically just compute a rectangle around each of the features in a dataset and use that as a rough index for queries. This is also known as a minimum bounding rectangle (MBR) and a type of R-Tree index is used too.

Data Types



	SQL with geometry type	ISO/IEC 13249-3:2003 (SQL/MM-Spatial)	Description
Geometry Types	Point	ST_Point	The type ST_PolyhedralSurface is currently not in SQL/MM but will be proposed as a result of this document.
	Curve	ST_Curve	
	Linestring	ST_Linestring	
		ST_Circularstring	
		ST_CompoundCurve	
	Surface	ST_Surface	
		ST_CurvePolygon	
	Polygon	ST_Polygon	
	PolyhedralSurface	ST_PolyhedralSurface	
	GeomCollection	ST_Collection	
	Multipoint	ST_Multipoint	
	Multicurve	ST_MultiCurve	
	Multilinestring	ST_Multilinestring	
	Multisurface	ST_Multisurface	
	Multipolygon	ST_Multipolygon	
Storage	Binary Type, Text Type, Object Type	Object Type	—
Operations	Equals	ST_Equals	—
	Disjoint	ST_Disjoint	
	Touches	ST_Touches	
	Within	ST_Within	
	Overlaps	ST_Overlaps	
	Crosses	ST_Crosses	
	Intersects	ST_Intersects	
	Contains	ST_Contains	
	Relate	ST_Relate	
Functions:	—	—	—
Point		ST_Point()	Return the Point
	X()	ST_X()	Return the X-coordinate of point
	Y()	ST_Y()	Return the Y-coordinate of point
	Z()	ST_Z()	Return the Z-coordinate of point
	M()	ST_M()	Return the M-coordinate of point
	—	ST_ExplicitPoint()	—

References

http://web.cs.wpi.edu/~cs561/s12/Lectures/activeDB/ActiveDB.pdf

https://www.w3schools.com/xml/xml_whatis.asp

https://www.profium.com/