CHAPTER - 11

Advance Database Concepts

11.1 Object-Oriented Model

RDBMSs were originally designed for mainframe computer and business data processing applications. Moreover, relational systems were optimized for environments with large number of users, who issue short queries. But today's application has moved from centralized computer aided design (CAD), multimedia system, software engineering, knowledge database. These operations require complex operations and data structure representation. For example, a multimedia database may contain variable length text, graphics and images, audio and video data. Finally a knowledge base system requires data rich in semantics.

Existing commercial DBMS, both small and large has proven inadequate for these applications. The traditional database notion of storing data in two-dimensional tables, or in flat files, breaks down quickly in the face of complex data structures and data types used in today's applications. Research in model and process complex data has gone in two directions:

- Extending the functionality of RDBMS and remove the difficulty in accessing database from high level programming languages.
- Developing and implementing OODBMS that is based on object oriented programming paradigm. OODBMSs are designed for use in today's application areas such as multimedia, CAD, office automation, etc.

Object Oriented Database Management System

Object-oriented technologies in use today include object-oriented programming languages (e.g., C++ and small talk), object-oriented database systems, object-oriented user interfaces (e.g., Macintosh and Microsoft Windows systems) and so on. An object-oriented technology is a technology that makes available to the users facilities that are based on object-oriented concepts.

Features of Object Oriented System: Object-oriented systems make these promises:

- **Reduced maintenance:** The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs. Because most of the processes within the system are encapsulated, the behaviors may be reused and incorporated into new behaviors.
- **Real-world modelling:** Object-oriented systems tend to model the real world in a more complete fashion than do traditional methods. Objects are organized into classes of objects, and, objects are associated with behaviors. The model is based on objects rather than on data and processing.
- **Improved reliability:** Object-oriented systems promise to be far more reliable than traditional systems, primarily because new behaviors can be built from existing objects.
- **High code reusability:** When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned. The new object will also inherit the data and behaviors from all super classes in which it participates.

The object model is made up of the following structures, connections, and some of these commonly used terms are given below:

- **Class:** A class is the equivalent of a relational entity or table. It is important to understand that a class is not the same as an object. An object is the instantiation or iteration of a class during execution. Since object structures allow inheritance, the structure and content of an object can be completely different to that of its constructor class. This is because inheritance allows passing of attribute structures, values, and methods up and down through a hierarchy.

- **Attribute:** An attribute is equivalent to a column in a relational entity column or field.
- **Method:** A method is a chunk of code or program executed exclusively on the contents of the object to which it is attached. The relational model does not allow use of methods, using stored procedures instead. Stored procedures are simply stored in the database as database procedures but have no link with specific entities.
- **Inheritance:** Classes are linked together through an inheritance hierarchy; some object databases and SDKs allow multiple inheritance. Multiple inheritance allows a class to inherit from more than one class at a time. Effectively, multiple inheritance allows a hierarchical structure to go in both directions.
- **Multiple Inheritance:** Multiple inheritance allows a class to inherit details from more than one class. Thus, the hierarchy goes both upwards and downwards, such that a class can be both a parent and a child of other classes. In other words a class can inherit properties from parent classes, as well as allowing child classes to inherit attributes from itself.
- **Specialization and Abstraction:** The result of inheritance is that classes can be both specialized and abstracted. Syntactically these two terms could be construed to mean the same thing, but semantically there is a difference. Specialization implies that an inherited class inherits part of a parent class or adds to the parent class, making a special case of the parent class. Abstraction implies an abstraction, or generic form, of a child or more likely a group of child classes.
- **Collection:** A collection is the term applied to a repetition of elements of one object contained within another object. A collection is similar to a relational entity on the many side of a one-to-many relationship.

Difference between RDBMS and OODBMS

Criteria	RDBMS	ODBMS
Defining standard	SQL2 (ANSI X3H2)	ODMG-V2.0
Support for object-oriented programming	Poor; programmers spend 25% of coding time mapping the program object to the database	Direct and extensive
Simplicity of use	Table structures easy to understand; many end-user tools available	OK for programmers; some SQL access for end users
Simplicity of development	Provides independence of data from application, good for simple relationships	Objects are a natural way to model; can accommodate a wide variety of types and relationships
Extensibility and content	None	Can handle arbitrary complexity; users can write methods and on any structure
Complex data relationships	Difficult to model	Can handle arbitrary complexity; users can write methods and on any structure
Performance versus interoperability	Level of safety varies with vendor, must be traded off; achieving both requires extensive testing	Level of safety varies with vendor; most ODBMSs allow programmers to extend DBMS functionality by defining new classes
Distribution, replication, and federated databases	Extensive	Varies with vendor; a few provide extensive support
Product maturity	Very mature	Relatively mature
Legacy people and the universality of SQL	Extensive supply of tools and trained developers	SQL accommodated, but intended for object-oriented programmers
Software ecosystems	Provided by major RDBMS companies	ODBMS vendors beginning to emulate RDBMS vendors, but none offers large markets to other ISVs

11.2 Object-relational mapping (ORM or O/RM or O/R mapping)

Object-relational mapping (ORM or O/RM or O/R mapping) is a programming technique for converting data between two different paradigms (a relational database and an object-oriented programming language) where data representations, data manipulation, modelling techniques and other entities related to it are different. It helps to create a "virtual object database" that can be used from within the programming language so that the objects can be translated into forms which can be stored in the database for easy retrieval, while preserving the properties of the objects and their relationships. While dealing with ORM you have to know what a ***persistent object*** is. It is the object that can automatically store and retrieve itself in database while preserving the properties of objects and their relationships.

Although the advantage of ORM is that it often reduces the amount of code needed to be written, the disadvantage would be due to some O/R mapping tools that do not perform well during bulk deletions of data. Hence ORM software has been pointed to as a major factor in producing poorly designed databases.

Significance of mapping: When we write an application, for example in Java, which stores data in a RDBMS by relying on data-aware controls (for e.g. data-aware GUI components) to interface directly with the database then such applications are not object oriented and by using such two-layer (GUI/Database) applications, we lose the benefits of object oriented design principle i.e. encapsulation. Applications that use data-aware widgets or controls, the client and the database are very tightly coupled where GUI code, *business logic*, and SQL statements are all interwoven throughout the application source code. So, any changes in the database schema will surely cascade into unexpected failures resulting in maintenance nightmare. This type of problem where technical difficulties are often encountered when a relational database management system (RDBMS) is being used by a program written in an object-oriented programming language or style is known as object-relational impedance mismatch. This type of problem occurs when objects or class definitions are mapped in a straightforward way to database tables or relational schema. Object-relational impedance mismatch is due to the following reasons:

- Objects can't be directly saved to and retrieved from relational databases
- Objects have identity, state, and behavior in addition to data whereas RDBMS stores data only.
- Even data alone can present a problem, since there is often no direct mapping between Java and RDBMS data types.
- Objects are traversed using direct references while RDBMS tables are related via like values in foreign and primary keys.
- Current RDBMS have no parallel to Java's object inheritance for data and behavior.
- The goal of relational modelling is to normalize data (i.e., eliminate redundant data from tables), whereas the goal of object-oriented design is to model a business process by creating real-world objects with data and behavior.

One of the main objectives of Object-Relational mapping is to solve the problem stated above (i.e. the object-relational impedance mismatch).

We can see that classes of OOP language can be mapped to RDBMS tables. For e.g. JAVA classes can be mapped to RDBMS tables. We can map a persistent class and a table in so many ways. The simplest mapping between a persistent class and a table is one-to-one. In this case, all attributes in the persistent class are represented by all columns of a table. Each instance of a business class is then stored in a row of the table.

Note: Class = Relation, Class attribute = Column and Object = Row or Tuple

11.3 Distributed Database

A distributed database as a collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system (distributed DBMS) is then defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users. The execution of global transactions requires communication among the each sites, which maintains a local databases system.

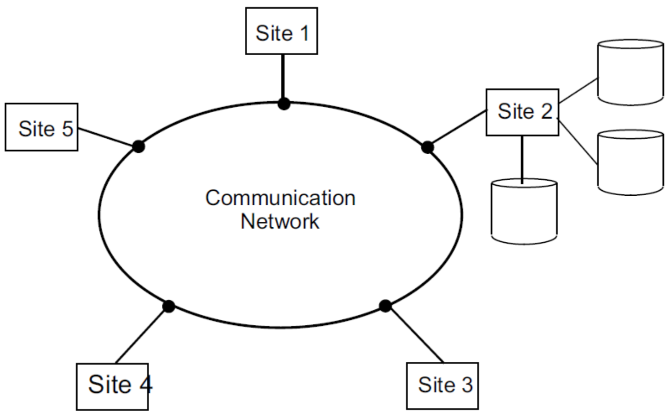

Fig. Central Database on a Network

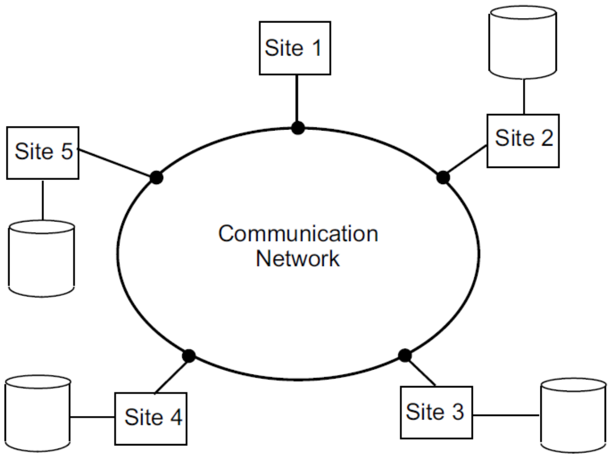

Fig. DDBS Environment

All the distribution transparency is maintained by the DDBMS by distributing its components in any of following ways:

- **Replication or Duplication:** Multiple copy of same database is placed in different place or geographical area of the earth. This may be fully or partially replicated.
- **Fragmentation or Partition:** In this method, first database is fragmented and then dispersed to different computer in different location.

A distributed database system consists of loosely coupled sites that shares no physical components. Here "loosely coupled" indicates the multicomputer homogenous or heterogeneous system. The database system that runs on each site are independent of each other. The transaction may access data at one or more sites.

There are several reasons for building distributed database systems, including sharing of data, reliability and availability, and speed of query processing. However, along with these advantages come several disadvantages, including software development cost, greater potential for bugs, and increased processing overhead. The primary disadvantage of distributed database systems is the added complexity required to ensure proper co-ordination among the sites.

Advantages

- Increase reliability, availability and easier expansion
- Reflects organizational structure — database fragments potentially stored within the departments they relate to
- Local autonomy or site autonomy — a department can control the data about them
- Protection of valuable data — if there were ever a catastrophic event such as a fire, all of the data would not be in one place, but distributed in multiple locations
- Improved performance — data is located near the site of greatest demand, and the database systems themselves are parallelized, allowing load on the databases to be balanced among servers.
- Modularity — systems can be modified, added and removed from the distributed database without affecting other modules (systems)
- Reliable transactions - due to replication of the database
- Continuous operation, even if some nodes go offline (depending on design)
- Distributed query processing can improve performance
- Distributed transaction management. All transactions follow A.C.I.D. property
- Single-site failure does not affect performance of system.

Disadvantages

- Complexity — DBAs may have to do extra work to ensure that the distributed nature of the system is transparent.
- Economics — increased complexity and a more extensive infrastructure means extra labor costs
- Security — remote database fragments must be secured, and they are not centralized so the remote sites may have security problems.
- Difficult to maintain integrity — but in a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible
- Inexperience — distributed databases are difficult to work with, and in such a young field there is not much readily available experience in "proper" practice
- Lack of standards — there are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS
- Database design more complex — In addition to traditional database design challenges, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication
- Additional software is required
- Operating system should support distributed environment
- Concurrency control poses a major issue. It can be solved by locking and time stamping.
- Distributed access to data
- Analysis of distributed data

11.4 Data warehouse

A data warehouse is a collection of information gathered from many different operational databases used to create business intelligence that supports business analysis activities and decision making tasks. So, data warehouses support OLAP.

- A decision support database that is maintained separately from the organization’s operational database.
- Support information processing by providing a solid platform of consolidated, historical data for analysis.
- “A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s decision-making process.”

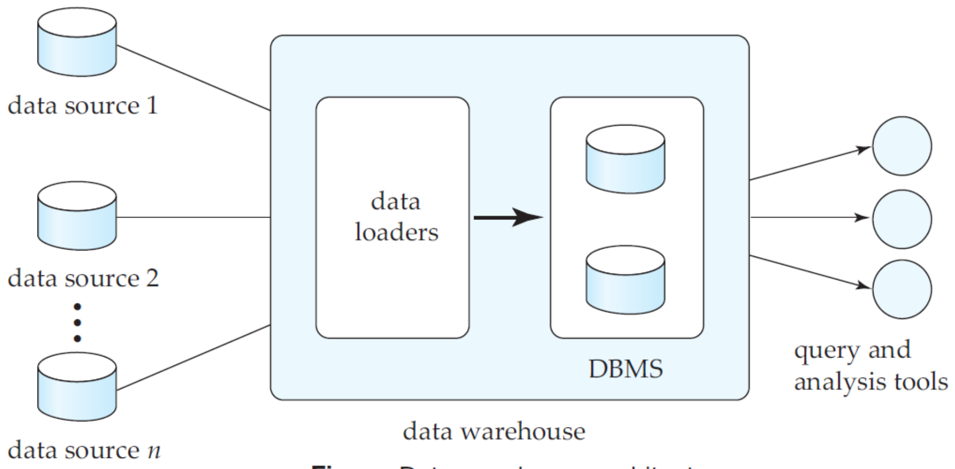

Figure Data-warehouse architecture.

A data warehouse is a repository of information collected from multiple sources, stored under a unified schema, and that usually resides at a single site. Data warehouses are constructed via a process of data cleaning, data integration, data transformation, data loading, and periodic data refreshing.

Suppose that *AllElectronics* is a successful international company, with branches around the world. Each branch has its own set of databases. The president of *AllElectronics* has asked you to provide an analysis of the company's sales per item type per branch for the third quarter. This is a difficult task, particularly since the relevant data are spread out over several databases, physically located at numerous sites. Figure shows the typical framework for construction and use of a data warehouse for *AllElectronics*.

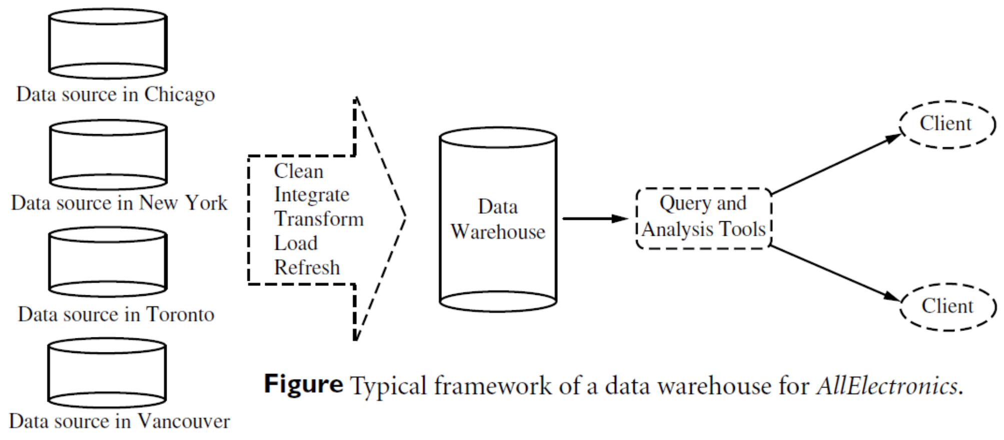

Figure Typical framework of a data warehouse for *AllElectronics*.

Warehouse Schemas

Data warehouses typically have schemas that are designed for data analysis, using tools such as online analytical processing (OLAP) tools. Thus, the data are usually multidimensional data, with dimension attributes and measure attributes. Tables containing multidimensional data are called **fact tables** and are usually very large. To minimize storage requirements, dimension attributes are usually short identifiers that are foreign keys into other tables called **dimension tables**.

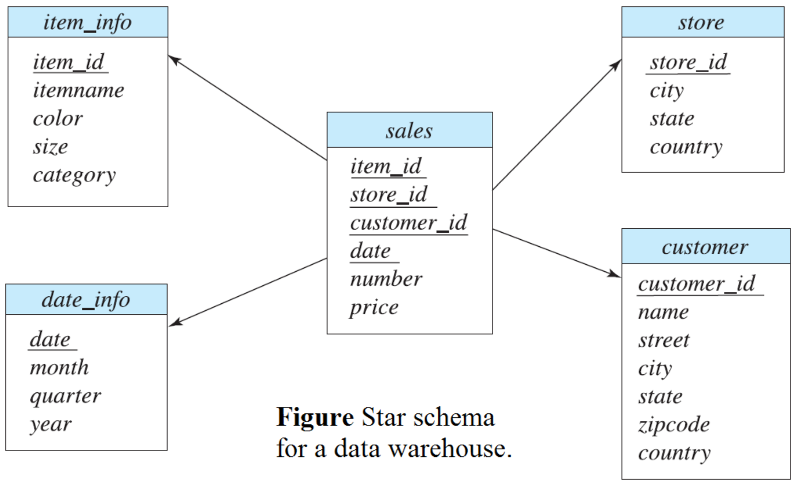

Figure Star schema for a data warehouse.