ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



database management system

Lab Assignment #1

**Submitted by:**

Lokendra Puri  
013BSCCSIT023

**Submitted to:**

Er. Sanjay Kr. Yadav

Lecturer  
Department of Computer Science

**Date of Submission: July 24 , 2015**

**Purpose ofDatabase Systems**

Traditionally, file processing system was used to manage information. It stores data in various files of different application programs to extract or insert data to appropriate file.

File processing system has several drawbacks due to which database management system is required. Database management system removes problems found in file processing system. Some major problems of file processing systems are:

* Concurrency: concurrent access (meaning 'at the same time') to the same database by multiple users
* Security: security rules to determine access rights of users
* Backup and recovery: processes to back-up the data regularly and recover data if a problem occurs
* Integrity: database structure and rules improve the integrity of the data
* Data descriptions: a data dictionary provides a description of the data

**View of Data**

A data model is a collection of concepts that can be used to describe the structure of a database. Data models can be broadly distinguished into 3 main categories- 1)high-level or conceptual data models (based on entities & relationships) It provides concepts that are close to the way many users perceive data. 2)lowlevel or physical data models It provides concepts that describe the details of how data is stored in the computer. These concepts are meant for computer specialist, not for typical end users. 3)representational or implementation data models (record-based,object-oriented) It provide concepts that can be understood by end users. These hide some details of data storage but can be implemented on a computer system directly.

**Database Languages**

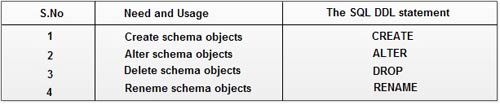
A [DBMS](http://ecomputernotes.com/fundamental/what-is-a-database/advantages-and-disadvantages-of-dbms) must provide appropriate languages and interfaces for each category of users to express database queries and updates. Database Languages are used to create and maintain database on[computer](http://ecomputernotes.com/fundamental/introduction-to-computer/what-is-computer). There are large numbers of database languages like Oracle, MySQL, MS Access, dBase, FoxPro etc. SQL statements commonly used in Oracle and MS Access can be categorized as data definition language (DDL), data control language (DCL) and data manipulation language (DML).

## Data Definition Language (DDL)

It is a language that allows the users to define data and their relationship to other types of data. It is mainly used to create files, databases, data dictionary and tables within databases.

It is also used to specify the structure of each table, set of associated values with each attribute, integrity constraints, security and authorization [information](http://ecomputernotes.com/fundamental/information-technology/what-do-you-mean-by-data-and-information) for each table and physical storage structure of each table on disk.

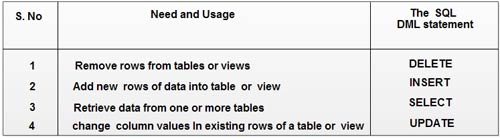
The following table gives an overview about usage of DDL statements in SQL

[](http://ecomputernotes.com/images/DDL%20statement%20in%20SQL.jpg)

## Data Manipulation Language (DML)

It is a language that provides a set of operations to support the basic data manipulation operations on the data held in the databases. It allows users to insert, update, delete and retrieve data from the database. The part of DML that involves data retrieval is called a query language.

The following table gives an overview about the usage of DML statements in SQL:

[](http://ecomputernotes.com/images/DML%20Statement%20in%20SQL.jpg)

**Relational Database**

A relational database (RDB) is a collective set of multiple data sets organized by tables, records and columns. RDBs establish a well-defined relationship between database tables. Tables communicate and share information, which facilitates data searchability, organization and reporting.  
  
RDBs use Structured Query Language (SQL), which is a standard user application that provides an easy programming interface for database interaction.

RDBs organize data in different ways. Each table is known as a relation, which contains one or more data category columns. Each table record (or row) contains a unique data instance defined for a corresponding column category. One or more data or record characteristics relate to one or many records to form functional dependencies. These are classified as follows:

* One to One: One table record relates to another record in another table.
* One to Many: One table record relates to many records in another table.
* Many to One: More than one table record relates to another table record.
* Many to Many: More than one table record relates to more than one record in another table.

RDB performs "select", "project" and "join" database operations, where select is used for data retrieval, project identifies data attributes, and join combines relations.   
  
RDBs have many other advantages, including:

* Easy extendability, as new data may be added without modifying existing records. This is also known as scalability.
* New technology performance, power and flexibility with multiple data requirement capabilities.
* Data security, which is critical when data sharing is based on privacy. For example, management may share certain data privileges and access and block employees from other data, such as confidential salary or benefit information.

**Database Design**

Database design is the process of specifying the logical and/or physical parts of a database. The goal of database design is to make a representation of some "universe of discourse" - the types of facts, business rules and other requirements that the database is intended to model.

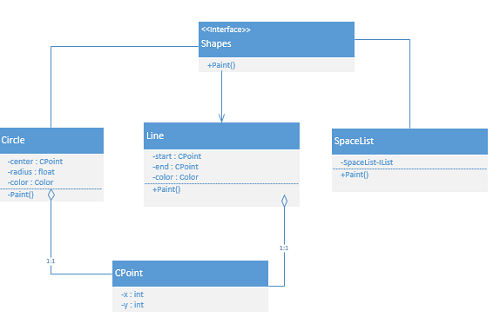
**Object-based and semistructured databases**

**Object DATABASE OR Object oriented database management system** is a database in which the information is represented in form of object as used in object-oriented programming. It is different from rational database. This type of database is used when there is complex data or/and multiple data relationships. It have a many-to-many object relationship. It should not be used when there are few join tables and there are large volume of simple transaction data.

*It works well with the following application:*  
  
**-->** Multimedia Application.  
**-->** CAS Application

### Features of Object Oriented Database:

* It support transactions.
* It supply querying in bulk data.
* Concurrent Access
* Security



### Example of Object Database

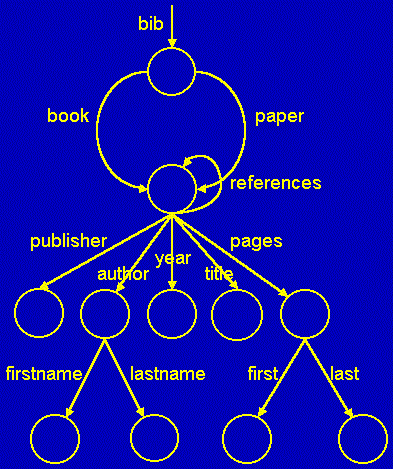
In **Semi-Structured Database** the data are in the form of structured data that edoes not conform with the formal structure of data models associated with rational databases or other form of data. Therefore, it is also known as self-describing structure.

### Types of Semi-Structured Database:

* XML semi-structured database
* JSON (JavaScript Object Notation)semi-structured database

**Advantages of Semi-Structured Database**

* It can show the information of data source that is not constrained by schema.
* It is used to view structured data as semi-structured data.
* The data transfer format may be portable.



### Example of Semi-Structured Database

**Data storage and Querying**

Databases are stored in file formats, which contain records. At physical level, the actual data is stored in electromagnetic format on some device. These storage devices can be broadly categorized into three types –



* **Primary Storage** − The memory storage that is directly accessible to the CPU comes under this category. CPU's internal memory (registers), fast memory (cache), and main memory (RAM) are directly accessible to the CPU, as they are all placed on the motherboard or CPU chipset. This storage is typically very small, ultra-fast, and volatile. Primary storage requires continuous power supply in order to maintain its state. In case of a power failure, all its data is lost.
* **Secondary Storage** − Secondary storage devices are used to store data for future use or as backup. Secondary storage includes memory devices that are not a part of the CPU chipset or motherboard, for example, magnetic disks, optical disks (DVD, CD, etc.), hard disks, flash drives, and magnetic tapes.
* **Tertiary Storage** − Tertiary storage is used to store huge volumes of data. Since such storage devices are external to the computer system, they are the slowest in speed. These storage devices are mostly used to take the back up of an entire system. Optical disks and magnetic tapes are widely used as tertiary storage.

**Query**

A query is a request for information from a [database](http://www.webopedia.com/TERM/D/database.html). There are three general methods for posing queries:

* Choosing [parameters](http://www.webopedia.com/TERM/P/parameter.html) from a [menu](http://www.webopedia.com/TERM/M/menu.html): In this method, the[database system](http://www.webopedia.com/TERM/D/database_management_system_DBMS.html) presents a list of parameters from which you can choose. This is perhaps the easiest way to pose a query because the menus guide you, but it is also the least flexible.
* [Query by example](http://www.webopedia.com/TERM/Q/query_by_example.html) (QBE): In this method, the [system](http://www.webopedia.com/TERM/S/system.html)presents a blank [record](http://www.webopedia.com/TERM/R/record.html) and lets you specify the [fields](http://www.webopedia.com/TERM/F/field.html) and values that define the query.
* [Query language](http://www.webopedia.com/TERM/Q/query_language.html): Many database systems require you to make requests for information in the form of a stylized query that must be written in a special [*query language*](http://www.webopedia.com/TERM/Q/query_language.html). This is the most complex method because it forces you to learn a specialized [language](http://www.webopedia.com/TERM/L/language.html), but it is also the most powerful.

**Transaction Management**

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

**A’s Account**

Open\_Account(A)

Old\_Balance = A.balance

New\_Balance = Old\_Balance - 500

A.balance = New\_Balance

Close\_Account(A)

**B’s Account**

Open\_Account(B)

Old\_Balance = B.balance

New\_Balance = Old\_Balance + 500

B.balance = New\_Balance

Close\_Account(B)

**Database Architecture**

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent **n** modules, which can be independently modified, altered, changed, or replaced.

In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



* **Database (Data) Tier** − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
* **Application (Middle) Tier** − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
* **User (Presentation) Tier** − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

**Database Users and Administrators**

Database users are the one who really use and take the benefits of database. There will be different types of users depending on their need and way of accessing the database.

1. **Application Programmers -** They are the developers who interact with the database by means of DML queries. These DML queries are written in the application programs like C, C++, JAVA, Pascal etc. These queries are converted into object code to communicate with the database. For example, writing a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database. It will include a embedded SQL query in the C Program.
2. **Sophisticated Users -** They are database developers, who write SQL queries to select/insert/delete/update data. They do not use any application or programs to request the database. They directly interact with the database by means of query language like SQL. These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirement. In short, we can say this category includes designers and developers of DBMS and SQL.
3. **Specialized Users -** These are also sophisticated users, but they write special database application programs. They are the developers who develop the complex programs to the requirement.
4. **Stand-alone Users -** These users will have stand –alone database for their personal use. These kinds of database will have readymade database packages which will have menus and graphical interfaces.
5. **Native Users -** these are the users who use the existing application to interact with the database. For example, online library system, ticket booking systems, ATMs etc which has existing application and users use them to interact with the database to fulfill their requests.

DATABASE ADMINISTRATORS

The life cycle of database starts from designing, implementing to administration of it. A database for any kind of requirement needs to be designed perfectly so that it should work without any issues. Once all the design is complete, it needs to be installed. Once this step is complete, users start using the database. The database grows as the data grows in the database. When the database becomes huge, its performance comes down. Also accessing the data from the database becomes challenge. There will be unused memory in database, making the memory inevitably huge. These administration and maintenance of database is taken care by database Administrator – DBA.  
A DBA has many responsibilities. A good performing database is in the hands of DBA.

* **Installing and upgrading the DBMS Servers: -** DBA is responsible for installing a new DBMS server for the new projects. He is also responsible for upgrading these servers as there are new versions comes in the market or requirement. If there is any failure in upgradation of the existing servers, he should be able revert the new changes back to the older version, thus maintaining the DBMS working. He is also responsible for updating the service packs/ hot fixes/ patches to the DBMS servers.
* **Design and implementation: -** Designing the database and implementing is also DBA’s responsibility. He should be able to decide proper memory management, file organizations, error handling, log maintenance etc for the database.
* **Performance tuning: -** Since database is huge and it will have lots of tables, data, constraints and indices, there will be variations in the performance from time to time. Also, because of some designing issues or data growth, the database will not work as expected. It is responsibility of the DBA to tune the database performance. He is responsible to make sure all the queries and programs works in fraction of seconds.
* **Migrate database servers: -** Sometimes, users using oracle would like to shift to SQL server or Netezza. It is the responsibility of DBA to make sure that migration happens without any failure, and there is no data loss.
* **Backup and Recovery: -** Proper backup and recovery programs needs to be developed by DBA and has to be maintained him. This is one of the main responsibilities of DBA. Data/objects should be backed up regularly so that if there is any crash, it should be recovered without much effort and data loss.
* **Security: -** DBA is responsible for creating various database users and roles, and giving them different levels of access rights.
* **Documentation: -** DBA should be properly documenting all his activities so that if he quits or any new DBA comes in, he should be able to understand the database without any effort. He should basically maintain all his installation, backup, recovery, security methods. He should keep various reports about database performance.

In order to perform his entire task, he should have very good command over DBMS.

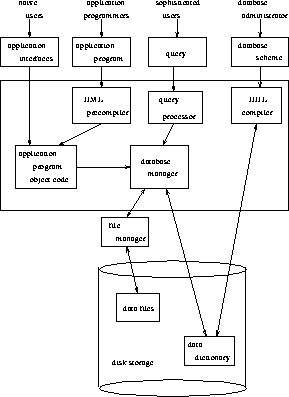
**Overall structure**

1. Database systems are partitioned into modules for different functions. Some functions (e.g. file systems) may be provided by the operating system.
2. Components include:
   * **File manager** manages allocation of disk space and data structures used to represent information on disk.
   * **Database manager**: The interface between low-level data and application programs and queries.
   * **Query processor** translates statements in a query language into low-level instructions the database manager understands. (May also attempt to find an equivalent but more efficient form.)
   * **DML precompiler** converts DML statements embedded in an application program to normal procedure calls in a host language. The precompiler interacts with the query processor.
   * **DDL compiler** converts DDL statements to a set of tables containing metadata stored in a data dictionary.

In addition, several data structures are required for physical system implementation:

* + **Data files:** store the database itself.
  + **Data dictionary:** stores information about the structure of the database. It is used **heavily**. Great emphasis should be placed on developing a good design and efficient implementation of the dictionary.
  + **Indices:** provide fast access to data items holding particular values.

1. Figure [1.6](http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter1/node21.html#fig16arch) shows these components.

     
**Figure 1.6:** Database system structure.

**History of Database Systems**

Database management systems (DBMSs) have played an outsized role in the history of software development and in the creation and growth of the software products industry. Recognizing the major role played by these products, the Annals is publishing two special issues on the subject. These two issues will be the fourth and fifth sponsored by the Software Industry Special Interest Group of the Computer History Museum (formerly the Software History Center). This issue (the first) is focused on the products, companies, and people who designed, programmed, and sold mainframe DBMS software products beginning in the 1960s and 1970s. The second issue will be devoted to the relational DBMS products, which were developed during the 1970s and came to prominence (and some say dominance) during the 1980s and 1990s.

What was so important about these DBMS products? Why did they have such a major impact on the growth of the software products industry and, more importantly, on the way that almost all major commercial applications were built from the 1970s on? It is a complex story, part of which is told in this issue. Thomas Haigh begins this issue by describing the world prior to DBMSs and some of the early DBMS products. Tim Bergin and Thomas Haigh then examine the database management products that dominated the IBM environment and other major computer platforms in the 1970s and 1980s.

This issue tells the rest of the story through a series of pioneer recollections, principally from people who founded the major DBMS companies or were heavily involved in the growth and development of these products and companies. These eight recollections cover the principal DBMS software products for IBM mainframe computers.1IBM itself was a significant player in this marketplace with its IMS product, but all the other products were produced and marketed by independent software companies. Many historians and industry analysts believe that these products and these companies formed the foundation on which the mainframe software products industry was built.