Three Denerations of DBMS

Maria Skiba

Edith Cowan University

THREE GENERATIONS OF DBMS

by

Maria Skiba BAppSc

A Thesis submitted in Partial Fulfilment of the

Requirements for the Award of Bachelor of

Applied Science with Honours

at the School of Arts and Applied Science,

Edith Cowan University.

Date of Submission 02.12.91

ABSTRACT

This paper describes the evolution of data base technology from early computing to the sophisticated systems of today. It presents an overview of the most popular data base management systems architectures such as hierarchical, network, relational and object-oriented. The last section of this paper presents a view of the factors that will influence the future of data base technology.

Keywords: ***DBMS, IMS, CODASYL DBTG, DB2, OODBMS.***

DECLARATION

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

Signature

Date........2/12/91..................

List of Illustrations

Table of Contents

1.0 Introduction.

Data is a valuable corporate resource, but like any resource, data only becomes a worthwhile asset if marketed, administered and exploited successfully.

The dependency of a company's business on the quality of their information bases becomes ever increasing. Every decision made in today's business world is based on data of some form. As the pace of decision making increases, great attention must be given to the way data is gathered, stored, and made available for the decision makers.

The technological progress made by hardware and software industries offers a range of methods to capture, store and access data. The most popular is the data base management system (DBMS) approach. A variety of DBMSs are widely available at present, they vary in size and functionality, and they can be run on personal computers, medium range computers or mainframes.

The following chapters cover the history of data base technology, an introduction to data base management systems, an overview of four data base management systems (hierarchical, network, relational and object-oriented), and the future outlook for data base technology.

2.0 History of Data Base Technology.

Data base technology has undergone a long evolution during which all the new developments helped to increase the effectiveness of data processing systems.

The "traditional" data management, the file system, took place during the 1950's and most of the 1960s. It consisted of an application program doing sequential processing of master files, commonly called old-master/new-master processing.

This processing was modeled after the physical characteristics of magnetic tapes. The old master file was processed sequentially, record by record, against a file of batched changes (such as a day's orders) that were sorted into the same order as the master file (such as customer account number) to produce a new master file containing the changes.

When a new application using the same master file data was needed, another program to do this processing was written. These applications were written and maintained by data processing specialists. The data was usually stored on tape, and applications were run as batch programs.

Typical applications were orders, inventory management, accounts payable, payroll, and other batch oriented processing, representing the automation of the "back office" of a business establishment.

As much of the data in the "back office" became computerized, the need emerged in the late 1960s and early 1970s for general purpose data management systems.

Such a demand was stimulated by the progress in hardware technology (hardware became faster, cheaper and reliable), the progress in computer science research (formulation of theoretical models and standards), a business requirement for instant, on-line access to data, and changes within the data processing departments which became information systems centers.

The file systems began evolving to data base management systems, ***DBMS***, by centralizing the data previously stored on various magnetic tapes, under the supervision of the enterprise's data processing specialists, and providing a uniform interface to the data.

These systems featured disk storage rather than the usual tape or punched-card storage, and provided on-line, random access to data. Examples of these systems are access methods such as ISAM and VSAM (Fergusson 1983).

These access method systems eliminated one of the problems of the file systems, that of multiple and inconsistent copies of data.

As this evolution of data management progressed, systems were designed to provide a significant increase in the availability, security, integrity, and consistency of the data stored (Fry 1976; Senko 1977; Selinger 1987).

The Third-Generation Database System Manifesto published by the Committee for Advanced DBMS Function in 1989 classifies database systems as follows (Ross 1991, pp. 3-4):

- The network and hierarchical database systems, prevalent in the 1970s and 1980s, are classified as the first-generation of database management systems because they were the first systems to offer substantial DBMS function in a unified system, with a data definition and data manipulation language for collections of records. CODASYL systems and IMS typify such first generation systems.

- In the 1980s first-generation systems were joined by the current collection of relational DBMSs which are termed the second generation database management systems.

  They are widely believed to be a substantial step forward for many applications over the first-generation systems because of their use of a nonprocedural data manipulation language, and their provision of a substantial degree of data independence. Second-generation systems are typified by DB2, INGRES, SQL, ORACLE.

- The third-generation data base systems are being developed. Although the third-generation DBMS technology is in its prototype stage, the Committee expects such a technology to be dominant in the 1990s. The third-generation DBMSs must satisfy following tenets:

  - Besides traditional data management services, third-generation DBMS will provide support for richer object structures and rules.

  - Third-generation DBMS must subsume second-generation DBMS.

  - Third-generation DBMS must be open to other subsystems.

As yet there is not a dominant product on the DBMS market which satisfies the above requirements, but promising work continues in the areas of object-orientation, deductive database systems and distributed systems.

3.0 Introduction to DBMS.

"A database management system (DBMS) is a system for managing stored information and providing protocols and a language interface to define, access, and change that information."(Selinger, 1987, p. 96)

Figure 1 on page 6 shows a typical DBMS configuration.

The following paragraphs describe the most common functions performed by database management systems (DBMS).

Users interact with the DBMS through language subsets. A data definition language (DDL) is used for defining and changing data objects, whereas a data manipulation language (DML) is used for reading or changing instances of the data objects.

These languages can be used statically in programs (known as host language embedding) or used dynamically, via interactive connections to the DBMS (known as query interfaces).

A DBMS stores records as a sequence of fields that take values of a given data type (eg. integer, character string, fixed decimal, etc.). The DBMS may enforce data types, perform data type conversion, support default values, or null values or even perform some arithmetic on data values.

In addition, a DBMS provides a grouping of operations into an atomic unit of work called a transaction. A user specifies the boundaries of a transaction. A successful termination of a set of operations that change a database is known as COMMIT, and an unsuccessful termination is an ABORT or ROLLBACK.

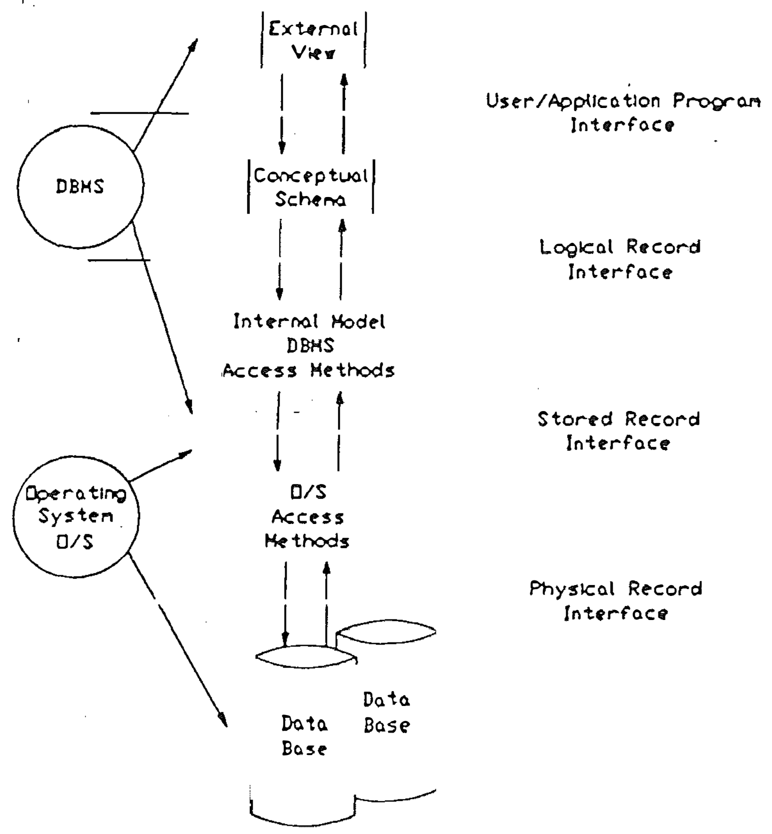

Figure 1. DBMS Three-Level Architecture.

When a transaction aborts, its actions are undone by the DBMS recovery facility, leaving the database in the same state that it was in at the beginning of the transaction.

The recovery facility is also invoked when the entire system (either the DBMS or the computer) crashes. When this happens, the DBMS is restarted (automatically or by an operator) and it ensures that all transactions which were not completed at the time of the crash are undone. The completed transactions have all their effects reapplied to the data if necessary.

The recovery of data after physical media crashes such as disk can be performed in a variety of ways: logs, differential files, time stamps, etc. The most popular technique is logging, that is, recording on a disk or tape the changes made to the database together with the name of the transaction that made them.

Logging techniques may record changes at the physical or logical level, may write these changes before or after the data is changed, may write on two separate media, may record both ″before″ and ″after″ versions of the changed data, and may store the log in a storage hierarchy (keeping only recent log information online, archiving older log records).

One of the major objectives of data base systems is to allow multiple concurrent users to share the same data. Often, users need to access the same data simultaneously, which in case of update requests, can lead to concurrency problems such as lost updates, uncommitted updates and data inconsistency if some form of DBMS concurrency control is not used.

Some of the most common concurrency control techniques are:

1. The use of time stamps as the unique identifiers for transactions.

2. Data base object locking, where objects range from the entire data base to a field within a record.

3. Predeclaration of resources, which is the analysis of queries before they are executed and scheduling their execution so that they will not conflict.

3.1 Advantages of DBMS.

A number of very important benefits of database management systems have been recognised (Senko, 1977; Atre, 1988; Ricardo, 1990):

- *Sharing of data.* A Data Base Administrator manages the data which belongs to an entire organisation and many users can be authorised to access the same information.

- *Redundancy control.* As many users can access an organisation's data there is no need to duplicate data across various departments. Some limited redundancy within the DBMS system is acceptable and mainly used to implement logical connections between data items, or to improve performance.

- *Data consistency.* If a data element is stored in only one location its value needs to be updated only once and all users will have immediate access to the new version of that data. If there are duplicates of a data item within a DBMS they will be updated by the DBMS itself or by an application program.

- *Improved data standards enforcement.* The agreed upon data standards within an organisation will be followed by all the DBMS users.

- *Better data security.* Most of the DBMS available today provide data access security in the form of user authorisation levels, data encryptions, restricted user views of data, and security/audit logs.

- *Improved data integrity.* Some DBMS (eg. DB2) allow for the definition of internal consistency rules that a database must obey. These rules are the intra-record constraints which apply to the data items within a record and the inter-record constraints which are applicable to the relationships between the records.

- *Better data accessibility.* Most data base management systems provide on-line access to data, and query languages which can be used to obtain answers to one-off questions about the data rather than writing application programs.

- *More control over concurrency.* Such control is performed by the DBMS subsystem which makes sure that transactions are not lost or executed incorrectly unlike the file system, which may allow multiple users both read and update access at the same time.

- *Efficient backup and recovery procedures.* In the case of failure, the DBMS will recover the data bases to the state they were just prior to the crash by restoring the latest database backup copies and applying the subsequent log files which contain the "before" and "after" images of the changed data.

3.2 *Disadvantages of DBMS.*

Integration of data within an organisation into a central place may pose some disadvantages, such as (Ricardo 1990):

- *High cost of DBMS.* A complete data base management system is a complex and sophisticated piece of software which is expensive to purchase or lease. Also, it usually requires purchase of additional support tools and software.

- *High conversion costs.* The process of conversion to a database system from a file system or conversion from one type of DBMS to another is difficult, time consuming and effort intensive.

- *Higher hardware costs.* Additional processing power, memory, and DASD (Direct Access Storage Device) may be required to run the DBMS, resulting in the need to upgrade hardware.

- *Increased vulnerability.* The centralised data may become a target of security breaches and may threaten privacy unless stringent security disciplines are maintained. Also, a loss of data can cause disruption to a large number of applications and changes to data structures can result in having to change and recompile a large number of programs.

- *Complexity of recovery.* To perform a database recovery after a system failure the DBMS must determine which transactions were completed and which ones were in progress at the time of failure. Transactions which were not completed will have to be undone (rolled back) and restarted. If a database is damaged it will have to be recovered using the backup copy and the log file.

- *Higher programming costs.* A DBMS is a complex and sophisticated tool with many features which can often only be utilised to their best advantage by highly skilled personnel.

- *Administrative overhead.* A DBMS requires dedicated support staff which usually forms a Data Base Administration group.

- *Complexity of operation.* A DBMS supports concurrent access to data by multiple users, performs a range of operations to keep the data accessible, of high quality and integrity, and provides for data security. Its performance must be constantly monitored, optimised and tuned.

4.0 Standards of Data Base Architecture.

The need for the standardization of data base technology was recognised in the mid 1960's when an informal task group was formed by the Conference on Data Systems and Languages (CODASYL) to study the subject of data bases (Senko, 1977).

An early proposal for standardized vocabulary and architecture was published by the ***CODASYL DBTG*** (Data Base Task Group) in 1971. (Revised versions of this proposal were published in 1973, 1978, 1981, and 1984). A similar proposal was developed and published in 1975 by the Standards Planning and Requirements Committee of the American National Standards Institute Committee on Computers and Information Processing, *ANSI/X3/SPARC* (Ricardo, 1990).

As a result of these reports, data bases can be viewed at three levels of abstraction. Such three-level architecture is composed of three schemas or models which refer to the permanent structure of a database.

The purpose of the three-level architecture is to separate the physical data base representation from its logical view. This is desirable because of the following:

- Users may require different views of the same data.
- Data base complexities should be hidden from the users.
- A Data Base Administrator should be able to change the logical structure of a data base or physical file structure without affecting the users.
- Data bases should be unaffected by changes to physical storage devices.

The levels within the three-level architecture of a data base system are (Ricardo, 1990):

1. The ***external view level*** which is the user view of the data base.

2. The ***logical*** or ***conceptual level*** which is the entire information about the data base structure as seen by the Data Base Administrator (DBA).

3. The ***internal*** or ***physical level*** which is the way the data is actually stored using the data structures and file organisations of a particular DBMS.

   At the physical level a DBMS works with the operating system's access methods to place data on the storage devices, build the indexes, and/or set the pointers that will be used for data retrieval. Therefore, there is actually a physical level below the one that the DBMS is responsible for, one that is managed by the operating system under the direction of the DBMS.

The three-level architecture is shown in Figure 1 on page 6

5.0 The First Generation of DBMS.

The following DBMS structures fall into this category:

- Hierarchical.
- Network.

The hierarchical model is the oldest database model, dating from the 1960s. The network model (often referred to as CODASYL) arrived shortly after the hierarchical model. It basic intention was to improve the hierarchical design, and encourage industry standards for DBMS technology.

5.1 The Hierarchical Data Structure.

The hierarchical data model uses the ***tree*** as its basic structure. A tree consists of a hierarchy of nodes, with a single node, called the root node, at its highest level. The root can have any number of child nodes, but each child node may have only one parent node on which it is dependent. The relationship between the nodes is one-to-one or one-to-many.

A hierarchical data base consists of a collection of occurrences of a single type tree, which is called a ***data base record.*** Each node of a tree is called a ***segment***, which consists of fields or data items that represent attributes of an entity. The entire tree or data base record may be a description of a single entity, or different segments within a data base record may be used to represent different, but related entities.

Multiple occurrences of the same child segment are called *twins*, and all child segments of a different type that share the same parent are called ***siblings***.

The relationships between segments are implemented by the physical placement of a segment or by pointers that simulate such placement, rather than the data itself.

The physical pointers are stored in the segment's prefix area and represent the physical location of the next segment within the tree, as specified by the data base definition.

The tree structure as an underlying data model is used in a number of DBMSs, for instance in IBM's Information Management System (IMS), SAS Institute's System 2000, and Informatic's Mark IV (Senko 1977).

5.2 IMS Architecture.

Although IMS, Information Management System was developed before the abstract data models were defined (Ricardo, 1990), it is possible to discuss its architecture in terms of the three-level interface between a user and the IMS data base, as shown in Figure 2 on page 15.

5.2.1 IMS Components.

5.2.1.1 Physical Data Bases.

An IMS physical data base is a hierarchical arrangement of physical segments of different types where:

1. There is a single type of a root segment.

2. The root may have dependent child segment types.

3. Each child of the root may also have any number of child segment types, up to a maximum of 15 segments in one path.

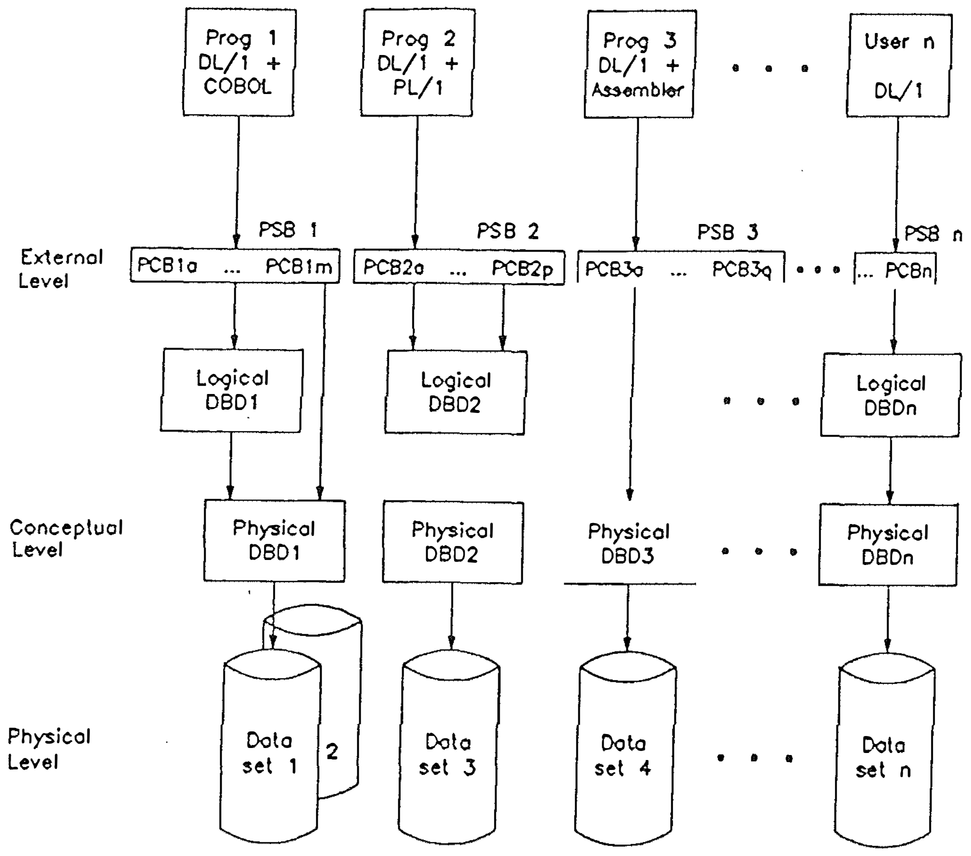

Figure 2. IMS Architecture.

4. There can be up to 255 segment types in one physical data base.

5. For any occurrence of a given segment type, there may be any number of occurrences of each of its children.

6. No child segment can exist without a parent.

The hierarchy of the segment types is defined as top to bottom and left to right, as in Figure 3 on page 16

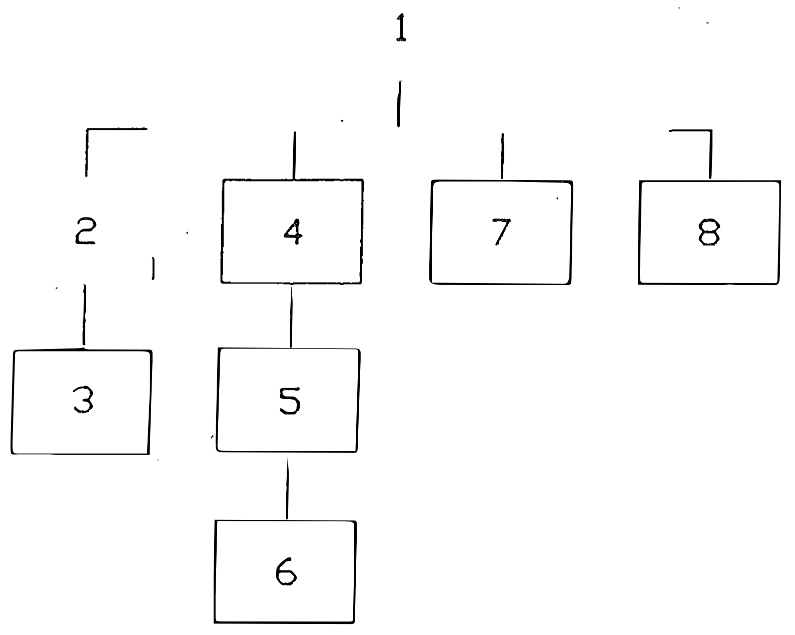

Figure 3. IMS Segment Hierarchy.

5.2.1.2 Logical Data Bases.

Logical data bases are the data views used by application programmers. These views are strictly hierarchical, but they can be constructed with the help of one or a number of physical data bases using logical relationship pointers.

Logical relationships connect segments from two or more physical data bases, or connect the segments from different branches of the same physical hierarchy, creating a new logical hierarchy.

The capability of logical relationships in IMS provides some of the network data model properties which will be explained in the later section.

The rules concerning logical data bases are as follows: (Ricardo, 1990)

- The root of a logical data base must be the root of some physical data base.
- A segment cannot be both a logical child and a logical parent.
- Any segment type in the physical hierarchy, together with all its dependants, can be omitted from the logical structure.
- Within a segment, fields can be omitted or rearranged.

5.2.1.3 *Control Blocks.*

An IMS data base definition is implemented by means of control blocks, i.e. DDL descriptions that create physical data bases, logical data bases, and program specification blocks. These descriptions are created by a DBA, compiled and stored in object code form for use by the IMS control program.

- *Physical DBD* (Data Base Description) block is used to define the physical hierarchy of the IMS data base and the physical access method. The physical DBD is the IMS conceptual schema.
- *Logical DBD* block defines the logical view which can be based on one or more physical data bases. In the three-level model, the Logical DBD can be placed between the conceptual level and the external level.
- *PSB* (Program Specification Block) specifies the data bases that an application program uses. The PSB consists of one or more *PCBs* (Program Communication Blocks) and they are the application program's view of a data base.

5.2.1.4 The Data Manipulation Language.

The DML for IMS is called Data Language One, ***DL/1*** and it is basically an embedded language, invoked from a host language such as PL1, COBOL or Assembler. DL/1 supports the following data manipulation functions:

- Retrieval operation is a ***GET*** command which comes in several forms (Ricardo, 1990). For example, a GU (Get Unique) will retrieve the first segment that satisfies a condition. Such a condition can be specified using DL/1 segment search arguments, ***SSA***.

  An unqualified SSA is the one that specifies only the segment name, a qualified SSA consists of the segment name and possibly, a set of brackets in which a condition may be given. The condition may be a comparison of a data element (field) using the relational operators (<, >, =, < =, > =) and/or the Boolean operators (AND, OR).

- Insertion, ***ISRT***.

- Update, ***REPL***.

- Deletion, ***DLET***. When executing this command, for the root segment in particular, extra care must be taken because IMS will delete the whole occurrence of the tree.

The segments are retrieved into a template or a segment layout defined in a host language. Each operation returns a ***status code*** which is a blank for success, or a two letter code for a warning or an error (IBM, 1986).

5.2.1.5 Internal Level of IMS.

At the physical level an IMS data base consists of one or more physical data files which can be organised according to the IMS access methods (Atre, 1988), such as:

- ***HSAM***, Hierarchical Sequential Access Method, is a basic physical sequential access method.

- *HISAM*, Hierarchical Indexed Sequential Access Method, uses a combination of indexed access to the root segment and sequential access to its dependents.

- *HDAM*, Hierarchical Direct Access Method uses a direct access method for relating segments within the hierarchy. The root address is calculated using a hashing algorithm.

- *HIDAM*, Hierarchical Indexed Direct Access Method uses indexed access to the root and direct access to the first occurrence of each dependent segment.

5.2.1.6 Secondary Indexing with IMS.

Because of the hierarchical nature of IMS data bases, certain types of searches are difficult to perform. For instance, dependent segments are normally accessed via the root segment, so it is difficult to find a dependent record on the basis of its value. To overcome this problem, IMS allows secondary indexes to provide direct access in any of the following ways (IBM, 1986):

- Access to a root, using a field other than the sequence field (primary key).

- Access to a root using a field in a dependent segment.

- Access to a dependent segment, using one of its field, .

- Access to a dependent using a field in a lower level d ,pendent segment.

Secondary indexes can be created for HISAM, HDAM or HIDAM data bases. An index data base uses the VSAM access method, it requires its own DBD and changes to the original data base DBD must be made to indicate that such a secondary index exists.

5.3 The Network Model.

A network model is a directed graph consisting of ***nodes*** connected by ***links*** or directed arcs. The nodes correspond to the record types (entities) and the links to pointers (relationships). The possible relationships in the network model are one-to-one and one-to-many. The network model looks like the hierarchical model, except that a dependent node, called a child or a ***member***, may have more than one parent or ***owner*** node.

A network data base consists of any number of named record types which consist of any number of fields. The relationships between the entities (records) are implemented using ***sets***.

A set type consists of a single owner record and one or more dependent records called members. An occurrence of a set is a collection of records having one owner occurrence and any number of member record occurrences.

A many-to-many relationship within the network structure is implemented using an ***intersection record*** or link record which consists of the key fields of the associated records and any other fields that are dependent on such an association.

In the network model, the sets are implemented using pointers. A linked list is created which is headed by an owner occurrence. The owner points to the first member, which points to the next, and so on, until the last member points back to the owner.

A member record can have the following pointers:

- The ***owner pointer*** which points back to the set owner.
- The ***prior pointer*** which points back to the previous member record.
- The ***next pointer*** which points to the next member record.

The network model or CODASYL DBTG model is used as an underlying structure in products such as ***IDMS/R*** from Computer Associates , ***PRIME DBMS*** from PRIME Computers, ***IDS II*** from Honeywell, ***TOTAL*** from CINCOM and ***IMAGE*** from Hewlett-Packard (Ricardo 1990).

5.4 *CODASYL DBTG Architecture.*

The fundamentals of the DBTG network model were first described in the CODASYL DBTG report in 1971. Figure 4 on page 22 represents the DBTG system architecture.

5.4.1 The DBTG DDL: The Schema.

The conceptual level of the DBTG architecture is specified in the ***schema***, which describes a data base and consists of:

- The ***Schema Section*** which gives the name of the data base.
- The ***Area Section*** which provides a list of the storage areas for a data base.

  The records and sets of a database are split into areas that are allocated to different physical files or devices. The major benefit of this technique is to allow the data base designer to place the most frequently accessed records on a faster device, and group together the logically related records in the same physical file for efficient retrieval.

  The physical storage details were removed by DBTG from the schema in the later versions of the DBTG model, but some commercial products are still based on the original model that was proposed in 1971 (Ricardo 1990).

- The ***Record Section.*** This section gives a complete description of each record structure and the record location mode.

  The record location mode is a method used by the DBMS to place the record in storage. There are three location modes:

  1. ***CALC***, which means that the DBMS must use a hashing function on a key field to determine a physical address for that record.
  2. ***VIA*** option places a record close to its owner..

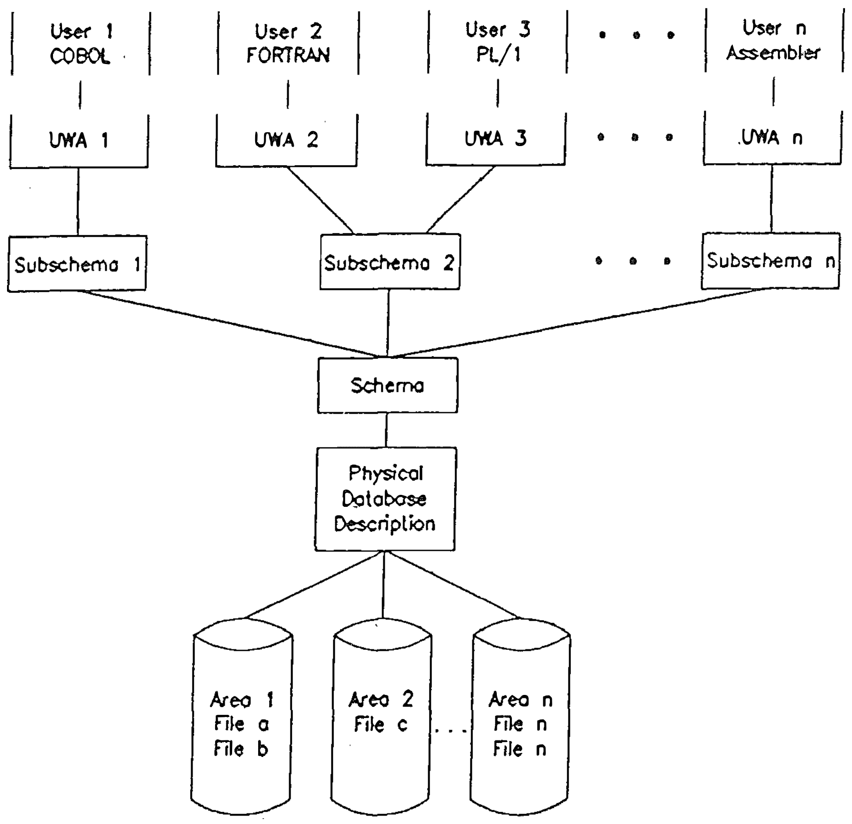

Figure 4. Architecture of DBTG System.

3. *DIRECT* option means that the address will be specified by the prog[redacted]er when a record is to be inserted. This option is usually avoided because the addre the record, stored using DIRECT option, must be specified to retrieve it.

The Set Section. This section identifies all the sets, the owner and the member record types, the logical order of records within the set, insertion class and retention class of member records.

Dependent records within a set can be ordered chronologically, i.e. their position is determined by the time they were put in a set. Such order can be FIRST, LAST, NEXT, PRIOR or IMMATERIAL (the DBMS decides where the member record should be placed in a set).

The DBTG schema also allows the definition of a SORTED order. The member records can be sorted by a record name or by the record key.

Set insertion class specifies how the member records will be placed in a set occurrence. There are two ways of placing the records:

1. MANUAL - the member record will be placed by an application program using the CONNECT command to set the required link.

2. AUTOMATIC - the member record will be connected to its set by the DBMS.

Set retention class determines whether a member record can be removed from a set once it is placed there. There are the following choices for set retention:

1. FIXED - a member record cannot be removed from a set occurrence unless this record is deleted from the data base.

2. MANDATORY - a member record must remain in a set occurrence, but not necessarily in the same occurrence.

3. OPTIONAL - a member record can be disconnected from a set.

5.4.2 The DBTG DDL: The Subschema.

The subschema is a description of the external view. Its function is similar to the IMS Logical DBD. The exact format of the subschema is dependent on the host language, but usually contains the following divisions (Ricardo 1990):

1. *Title Division*, which gives the subschema name and the associated schema.

2. *Mapping Division*, which specifies the changes in data items (records, sets, fields) from the schema to subschema.

3. *Structure Division*, which gives the names of data areas to be included in the subschema, records and sets, possibly with new names.

5.4.3 Currency Indicators.

Currency indicators are the pointers that identify the records most recently accessed by an application program. The currency indicators are contained in the *User Work Area (UWA)* or program work area which is a buffer assigned to a program.

The following types of currency indicators are used (Atre 1988):

1. *Current of run unit* is a pointer to the last record occurrence accessed by an execution of a program.

2. *Current of record.* For each record type, a pointer to the last record occurrence accessed is present in the UWA.

3. *Current of set.* For each set type, a pointer to the last occurrence of the record (member or owner) accessed is kept.

4. *Current of the area.* For each area, a pointer to the la.t record occurrence accessed is returned, regardless of its type.

5.4.4 DBTG DML.

A number of commands for data storage, retrieval and modification is provided by the DBTG DML. Some of them are considered here:

- *OPEN*: opens a data area for processing.

- *CLOSE*: closes a data area after processing.

- *FIND*: locates the record in a data base based on a specified key value.

- *GET*: retrieves the previously located record.

- *MODIFY*: updates a record with the changes.

- *ERASE*: deletes a record from the data base.

- *STORE*: adds a record to a data base.

- *CONNECT*: places a new member in a set occurrence.

- *DISCONNECT*: removes a member from a set occurrence.

- *RECONNECT*: moves a member record from one set occurrence to another.

For each record type there is a template defined in the host language and status flags are used to indicate successful, or otherwise, completion of the above operations.

6.0 The second generation of DBMS.

Data Base Management Systems based on the relational model belong to this category. For example, IBM's DB2 and SQL/DS, Oracle Corporation's ORACLE, Cincom's SUPRA and Relational Technology's INGRES (Atre, 1988).

6.1 The Relational Model.

This model was first proposed by Codd in 1970, and it was based on the mathematical notion of a *set* relation.

Sets consist of entities (records), or attributes (fields), or domains (allowable values for attributes), or relations.

A relation is physically represented as a ***table***. Tables are two-dimensional, made up of rows and columns, where each row represents entities and each column represents their attributes. The relationships between the entities are represented by common columns containing values from a common domain.

A table that represents a relation has the following characteristics:

- Each cell of the table contains only one value.
- Each column has a different name and represents an attribute.
- The values in a column come from the same domain.

- The order of columns is immaterial.
- There are no duplicate rows in a table.
- The order of rows is immaterial.

The number of columns in a table is called the ***degree*** of the relation. It does not change, and it can be unary, binary or n-ary. The number of rows in a table is called the ***cardinality***, and it changes as new rows are added. (Ricardo, 1990).

Rows in a table are uniquely identified by ***primary keys***, which consist of one or more attributes.

The relationships within the relational model are governed by ***integrity rules***, such as:

- The ***entity integrity*** which states that the key attributes must have a value, and they cannot be null (no value).
- The ***referential integrity*** which applies to the foreign keys.

  A ***foreign key*** of a relation is an attribute, or combination of attributes, which is the primary key of another relation.

  The referential integrity rule states that a foreign key value must be the same as the primary key value of another relation.

Two data manipulation languages were developed for the relational model, known as Relational Algebra and Relational Calculus.

Relational algebra is the procedural language of the relational model. There are several operators in relational algebra, all of which have the ability to manipulate tables to create other tables. Both the operands and the operators are tables. Some of the basic operators are (IBM, 1983):

- *SELECT* which takes a single table, and takes rows that meet specified conditions, and creates another table.

- *PROJECT* which also operates on a single table, but produces a vertical susbset of the table, extracting the values of specified columns, and placing the values in a new table.

- *JOIN* which comes in a variety of forms. Two of them are:

    - *EQUI-JOIN* which takes two tables and creates a third one. The third table consists of tuples from the first, concatenated with tuples from the second. This concatenation is limited to values in which specified attributes of the first table match specified attribute values in the second.

    - *NATURAL JOIN* differs from the EQUI-JOIN only by removing repeated columns.

- Other relational algebra operators are: *UNION*, *INTERSECTION*, *DIFFERENCE*, *CARTESIAN PRODUCT*, and *DIVIDE*.

Relational Calculus is a nonprocedural data manipulation language, in which the user specifies what data should be retrieved and not how.

6.2 DB2 Architecture.

Data Base 2, ***DB2***, was announced by IBM in 1983. It was developed on the basis of a prototype relational data base management system, called System R (Fry, 1976) which was developed by Codd and his associates at the IBM Research Laboratory during the late 1970s.

DB2 supports the standard three-level architecture for data bases as shown in Figure 5 on page 29.

The conceptual level consists of base tables which are physically stored in table spaces, which in turn, can be simple (store one or more tables) or partitioned (store one table). Tables can have any number of indexes, and are usually created by a DBA but maintained by DB2.

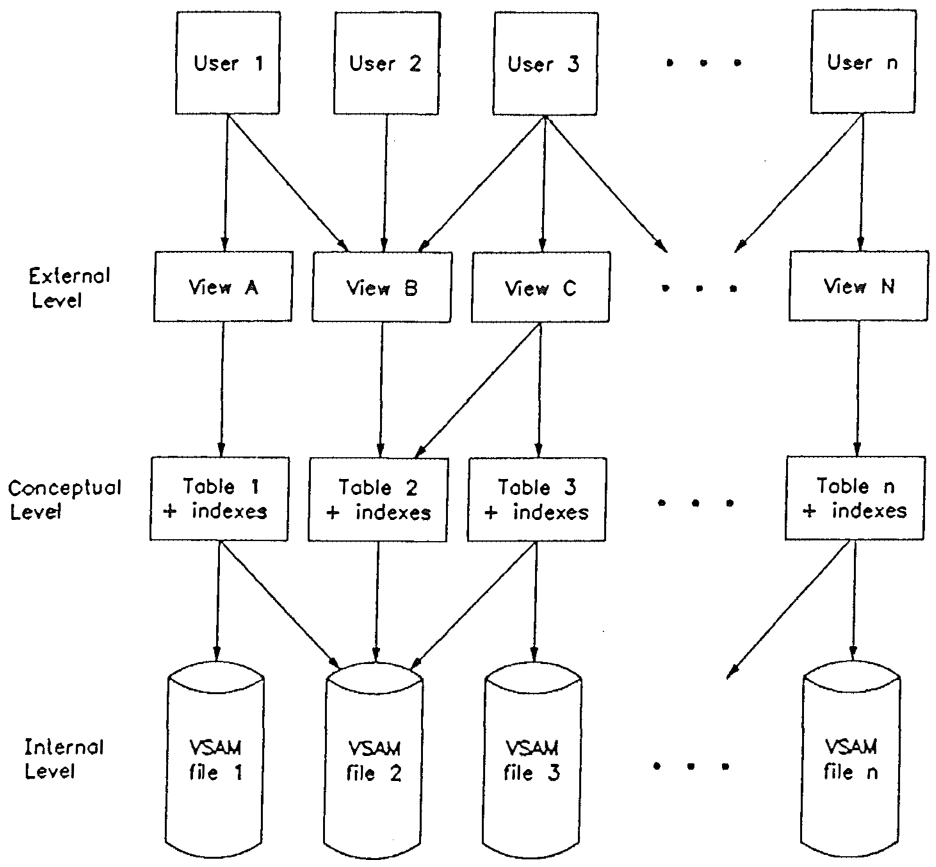

Figure 5. DB2 Architecture.

The external level consists of a number of user views. A view is a virtual table which is not stored permanently and is created when a user needs to access it. Views are subsets of one or more base tables, and they are useful for the following reasons:

- All[redacted]rent users to see the same data in different forms.
- Provide a simple authorisation control device.
- Can simplify complex DML operations, especially where several level JOIN operations are involved.
- Provide program independence from the conceptual data base definition.

At the physical level, base tables and indexes are stored in VSAM files. Rows of a table correspond to physically stored records, but their order and other details of storage may be changed. VSAM is used only as a disk manager, while DB2 controls the internal structure of both the data files and indexes. Users are usually not aware of what indexes exist and have no control over which index will be used by DB2 to locate a record. The access path is determined by the automatic query optimiser which chooses the optimum path.

The optimiser prepares an SQL statement for execution. accepts a parsed version of that statement and performs the following (IBM, 1989, p. 32):

- Authorisation checking.

- Symbol resolution, ie. checks if the objects named in the statement exist.

- Semantic checking. For example, the operands in compare operations are checked for compatibility.

- Access path selection. The optimiser considers all the available paths (including indexes) to data and, hopefully, selects the best one.

One of the most useful features of DB2 is ***dynamic data base definition.*** Tables can be added to or deleted from a data base any time. This is in contrast to previously presented DBMSs, which require the entire data base structure to be defined at creation time.

6.2.1 DB2 SQL.

Structured Query Language, ***SQL***, consists of a complete data definition language (DDL), data manipulation language (DML) and an authorisation language. was developed by Codd and associates as a data sublanguage for the relational model.

The DB2 SQL sublanguage can be embedded in a host programming language or can be used interactively.

The most important SQL DDL commands are:

- *CREATE TABLE* which takes as parameters the following: table name, column names, data types, foreign key name, primary key name, reference to other table names.

- *CREATE INDEX* requires index name, base table name, and column name. An index can be unique (eg. in primary key order) or nonunique (any field order). Indexes greatly improve performance and it is the system, not user, that decides which index should be used to search for records. However, indexes require external storage space and maintenance so over indexing can lead to a significant performance degradation.

- *ALTER TABLE* allows the addition of new columns to the right of a table. It takes a table name, column name and data type as parameters.

- *DROP TABLE* deletes a table and all indexes and views that depend on that table, and all the security authorisations.

- *DROP INDEX* deletes a specified index table.

SQL DML commands come in a variety of forms. The basic statements are:

- The *SELECT* statement is used for retrieval of data. It is a powerful command, performing the equivalent of relational algebra's SELECT, PRO'ECT and JOIN.

- The *UPDATE* statement is used to update values already stored in a table.

- The *INSERT* operator is used to add new rows to an existing table. It is useful for a small number of records. An entire table with a large amount of records can be loaded using the DB2 LOAD utility.

- The *DELETE* operator deletes rows from an existing table. The number of deleted rows depends on how many satisfy a given predicate. If no predicate is given all rows can be deleted from a table.

Each of the above commands has the capability to operate on a collection of records or return a collection of records, unlike in the previously discussed DBMSs, where DDL calls can only action one record at time.

DB2 SQL DML uses a combination of relational algebra operators and relational calculus predicates.

6.2.2 DB2 Catalog.

The DB2 catalog is a system data base which contains "data about all the data DB2 manages" (IBM, 1983a, p 121). The information is kept in table form. Whenever a table or a view is created or updated, the DBMS stores the information in the catalog. Some of the details include table names, column names, creator id, authorisation information, backup information, etc.

The DB2 catalog can be regarded as a "system-oriented data dictionary" (Ricardo, 1990, p.301) and it is managed using SQL DDL, data authorisation language (eg.GRANT capability TO, REVOKE capability FROM), and the BIND process, which will be described in the following section.

6.2.3 DB2 Internals and Related Products.

Some of the important DB2 DBMS components are as follows:

- The *DB2 Precompiler* is a preprocessor that examines application programs for SQL statements, replaces them with host language CALL statements, and generates a data base request module, *DBRM*, which consists of the parsed query conditions. The DBRM is then used as input to the BIND process.

- The *Bind* component takes the DBRM and creates optimised machine code, called the application plan, which is stored in the system catalog and contains information about the program, the program's data and modules that will be called to access the data base. The SQL optimiser is executed as part of the BIND process.

- The ***Runtime Supervisor*** controls the application program execution. When a program reaches a CALL, the Runtime Supervisor gets control and transfers it to the application plan, which calls the Stored Data Manager to perform the required I/O operation.

- The ***Stored Data Manager*** acts as the access method for a data base. It controls the interface to the operating system access method.

- ***DB2 Interactive, DB2I,*** is an interactive interface that allows users to execute SQL queries interactively. It also allows the on-line preparation and execution of programs, DB2 commands, and a range of utilities (eg. recovery, image copy).

- ***QMF, Query Management Facility*** is a DB2 related product that provides interactive capabilities for end-user query and report writing facilities.

- ***Data Extract, DXT*** is a utility program designed to move data from IMS data bases, VSAM files or Sequential Access Methods (SAM) files to DB2 data bases.

7.0 Relational DBMS vs First Generation DBMS.

A number of problems were recognised with the first generation of DBMS (Codd, 1982; Snell, 1985; Kim, 1991) over the years, such as:

- Complexity of data base structure. Application programmers must know how to navigate through a data base to get to the required data elements.
- Minimal data independence. That is, a distinction between a program's view of data and the physical storage of data.
- The installation and maintenance of data bases and applications is a complex and difficult task to perform.
- No support for set operations on data, that is, the ability to manipulate large amounts of data with one statement.
- Minimum provision for end-user computing. Users of the first generation DBMS are applications programmers or data base specialists whose main task is to create transactions which will be used by the end-users.

The relational DBMS solves most of the first generation DBMS's problems (Codd, 1982) and offers additional benefits, such as:

- Data can be accessed through data values rather than user visible links such as pointers. SQL is a common access language for all user levels.

- The Query optimiser determines access path to data, rather than a DBA or programmer.

- Improved data independence. There is a clear distinction between the physical and logical aspects of database management.

- The Data Base Administrator's task to ensure optimum data base performance has been facilitated by a number of SQL commands and relational DBMS utilities. Generally, the system decides how data base maintenance will be performed, the DBA decides what needs to be done.

- SQL supports set processing and provides nonprocedural access to data.

- End-users can execute simple SQL queries with minimum training, using, for instance, DB2 QMF.

- There is provision for data integrity controls via the integrity rules as described previously.

- The relational model is based on a sound theoretical foundation.

Although the relational systems provide many services, which are considered superior to those of the first-generation systems, there is concern about the relational system's performance, which is seen as a drawback by some (Atre, 1988; Ross, 1987).

The first generation systems were designed to take the best possible advantage of the available hardware which used to be slow, making the I/O operation costly. Today's hardware technology boosts the performance of the first generation systems even more, but does nothing about their complexity.

The relational DBMS performance can be affected by the following:

- The flexibility of DML, usually classified as an advantage, can result in constructing complex and inefficient SQL calls, such as multileveled joins, or executing very simple calls such as

SELECT * (all) from a table which consists of millions of rows. There is no built in mechanism to control inefficient end user queries.

- A larger number of objects (tables and indexes) to store, maintain and manipulate.
- A larger number of functions to perform which requires more CPU time.
- The Automatic query optimiser may or may not determine the optimum access path. In either case, it is not possible or is extremely difficult to change it.

According to Lees (1991) a relational DBMS is fast at some tasks, and a hierarchical or network DBMS is an excellent performer in some other tasks. It is up to the user to decide which DBMS will be better given the requirements.

At present, many organisations around the world run two DBMSs. The most common combination is IMS/DB2. Such a situation will not change for sometime until the IMS applications become obsolete and it will be possible to replace them with new ones, developed using DB2 technology. DB2 has the ability to communicate with IMS, so the application programs can contain both SQL and IMS calls.

8.0 The Third Generation of DBMS.

As stated by the Committee for Advanced DBMS Function, third-generation systems must support a large range of advanced features, some of them are (Ross 1991):

- Support for richer, abstract data structures.
- Rules about data elements.
- Inheritance hierarchy of data elements.
- Unique identifiers assigned by the DBMS if the primary key is not available.
- Updatable data views.
- Encapsulation of data and function.
- Capture semantics of data.
- Nonprocedural, high level access to data.
- Performance issues must be hidden from users.
- Support persistent programming languages, i.e., languages with permanent storage added to objects defined in an object programming language.
- Support access from multiple high level languages.

The most prominent requirement for third-generation DBMSs is that they will combine the data base and programming languages into one integrated system for programming and data base management.

Currently, data base management facilities are separated from programming environments. A data base management system only manages the declarative semantics of the shared, persistent data; the application programs support most of an application's operational definition, DML provides some operational capabilities as well.(Atkinson, 1987)

The work on the third-generation systems continues, and significant achievements come from the area of object-orientation.

According to Kim (1991, pp. 21-22), the two major reasons why the object-oriented paradigm is a sound basis for a new generation of data base technology are:

1. The object-oriented theory can be used as a basis for a data model that subsumes the data models of conventional data base systems. Using the object-oriented model, it is not only possible to represent data, relationships and constraints on the data, but also it is possible to encapsulate data and programs that operate on the data, and provide a uniform framework for the treatment of any user defined data types.

2. The notions of encapsulation and inheritance (reuse), are designed to reduce the difficulty of developing and evolving complex software systems and designs.

8.1 Object-Oriented Systems.

"An object oriented data base system is a persistent and sharable repository and manager of an object-oriented database, and an object-oriented data base is a collection of objects defined by an object-oriented data model"(Kim, 1991, p. 22)

There is no clear definition of an object oriented data model, but according to Kim (1991) it can be defined as follows:

"An object-oriented data model is data model that consists of a set of core object-oriented concepts found in most object oriented programming languages and systems" (p. 22).

The most significant object-oriented concepts are (Davis, 1988; Wirfs-Brock, 1990; Jordan, 1990; Mayer, 1990; Kim, 1991):

- An ***object*** is a data structure that is encapsulated with its state and behaviour, where the ***state*** of an object is a set of values for the attributes, and the ***behaviour*** of an object is a set of methods that operate on the state of that object. In relational terms, an object corresponds to an entity and encapsulation is a form of information hiding.

- A ***method*** is a special purpose function (code).

- A ***class*** is a collection of objects that share the same attributes and methods. A class corresponds to an entity type.

- An ***instance*** is an occurrence of an object described by a particular class.

- The classes are organised as rooted, directed acyclic graph, called a ***class hierarchy***. A class inherits all the attributes and methods from its direct and indirect ancestors in the class hierarchy. Simple class inheritance is shown in Figure 6 on page 40

  Semantically, a class is a specialization, ***subclass***, of the classes from which it inherits attributes and methods or a class is a generalization, ***superclass***, of the classes that inherit attributes and methods from it. The class hierarchy must be dynamically extensible, that is, new subclasses can be derived from one or more existing classes.

- The ***domain*** (type) of an attribute of a class can be any class. It may be a primitive class such as an integer or string, or it may a general class with its own set of attributes and methods. The value of an attribute may also be an object that belongs to a class.

- Objects communicate with each other or can be invoked via external ***messages***. A message is a request for a service that a particular object provides. Messages can change the state of an

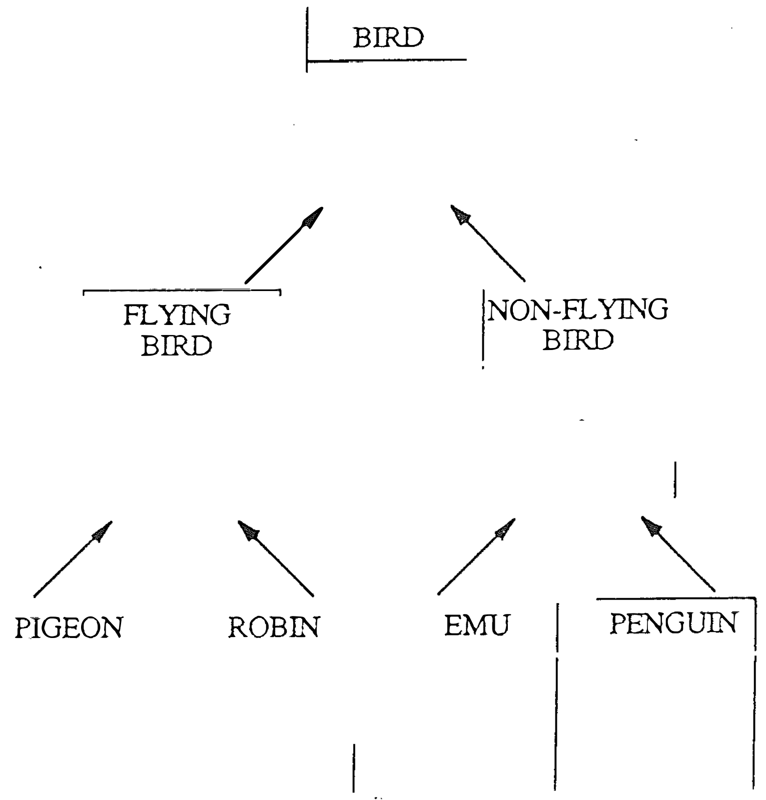

Figure 6. Simple Class Inheritance: The classes of flying and non-flying bird inherit the majority of their attributes from the parent class bird, but the flying attribute is defined in these classes. Individual bird types then inherit the appropriate set of characteristics from a bird class + flying or non-flying bird.

object, that is, modify its values. The recipient object will always respond to a message that has been sent to it. If an object does not understand the message, it signals an error which then triggers the debugging facilities to restore the communication.

- The inheritance of methods within the class hierarchy gives rise to ***polymorphism*** which is the ability of the same method to be invoked for objects of different classes. It also enhances the reuse and extensibility of application programs, since new classes may be defined by inheriting the methods associated with existing classes.

An object DBMS differs from the classic hierarchical, network and relational DBMS in three fundamental ways (Loomis, 1990, p. 11):

1. It supports user defined abstract data types and is not restricted to notions of records.

2. It supports arbitrarily complex operations and is not restricted to a predefined set of manipulators such as the relational SELECT, PROJECT and JOIN.

3. The code to implement services (functions) is stored in the data base and is activated by the object DBMS.

8.2 Methods of Implementing OODBMS.

There is a variety of methods for implementing an object-oriented DBMS, the most common are:

- *Extension of the relational model* to support object oriented features. According to Gardarin (1989), at least some features can be added to the relational model without losing its simplicity. These features include the support for user defined abstract data types, *ADT*, and the hierarchical objects.

  The ADT extension to the relational model requires that the First Normal Form constraint, which allows only single-valued attributes in each row, to be changed to allow the attributes to be complex data types , possibly structured as a class hierarchy. Such an extension requires the development of functions to manipulate complex objects and support an object hierarchy. Extensions to relational algebra and relational calculus are also required to cater for queries involving complex types. IBM's DB2 is moving in this direction.

- *Extending a programming language by adding persistence*, that is, allowing various objects to persist after a program execution has completed.

  In traditional programming languages the only persistent object is a file of records. It is possible to extend almost any programming language with object-oriented features, but some languages are more suited for such an extension than others i.e. the languages that already support abstract data types, encapsulation, inheritance and automatic memory management ("garbage collection") may be the prime candidates for object-oriented extension.

- *Starting over.* Another approach is to impose an object model on a data base foundation. For example, the Hewlett-Packard's *Iris* prototype implements an object model on a transaction and storage manager which provides for concurrency, recovery, buffer management, access path selection, and physical space management.

  The main component of Iris is an object manager which compiles and optimises queries and access to the object base, it runs on top of the transaction and storage manager. The object manager communicates with a relational DBMS and the Unix file system. It supports a variety of language interfaces, such as the prototype interface to Smalltalk, C++, Objective-C, Lisp and C programming environments. An SQL like language allows interactive access to objects and invocation of object sevices. A graphical editor/browser is also available to allow a user to scan through the hierarchy of object definitions, examine the object contents, modify an object's data values and to request object services. (Loomis, 1990).

8.3 Current Situation in OODBMS Technology.

During the past few years, there was a great rush to develop object-oriented data base systems. A number of prototypes of varying degrees of functionality were built in industrial and university research laboratories around the world. In the view of many authors (Hudson, 1989; Bary 1991; Ingari, 1991; Kim 1991), the object oriented technology is still very immature and suffers from a range of problems. Kim (1991) classifies these as follows:

- *Lack of standards.* The technology has matured enough for a standard to emerge. Work on object-orented technology standards is currently being conducted by the Object Management Group, *OMG* and the *ANSI SPARK OODB Task Group.* It is expected that the efforts of both groups should result, at least, in a standard core object-oriented data model within the next two years.

- *Lack of formalization.* An object-oriented data model, can be seen as a type of an extended relational model of data, and as such can be based on the same mathematical foundation (Beech 1988). The problem is that object-oriented extensions to a relational model require data to be denormalized, which causes the data base to be no longer fully relational. The debate on the object-orientation formalism is still open.

- *Implementation problems.*The first wave of object-oriented data base systems focused on one aspect of system performance, namely, management of memory resident objects (Kim, 1991). But an object-oriented data base management system is much more than just a persistent storage manager that supports retrieval of objects, one at a time. It is expected that the next wave of object-oriented data base systems will support all of the second-generation DBMS features (declarative query language, automatic query optimisation, access methods for efficient query processing, security and authorisation, recovery and semantic integrity specification and enforcement, etc.), and all additional data base features that are required for nonbusiness data processing (eg. versioning, parts assembly, long-duration transactions, etc.)

- *Complexity.* The richness of object-oriented structures and their flexibility can be a great advantage, but it can also be a problem. There are no clear guidelines for applications development, no real criteria to assess systems performance, and the OODBMS architecture is not clearly defined either. (Kim, 1991)

  According to Davis (1988, p. 16), "everyone will be in favor" of object-oriented technology. At present, an available option to prepare for object technology is to attend publicly offered seminars on object technology which are now being offered. Tutorial information is also available in the growing number of books on the subject. The most effective way in learning object oriented concepts is to experiment with object-oriented programming languages such as Smalltalk or C++.

Object oriented concepts are being used in applications such as CAD/CAM, CASE, and office automation. (Davis, 1988).

The major benefit expected from object technology is an improvement in systems development productivity, which should arise from the following:

- Models can better capture the characteristics of realworld systems which are based on objects that "know" and "can do".

- Adaptable and flexible software capable of handling different types of objects within the same data base.

- Reusable software modules which can be shared across various systems.

- Modes of interaction that are more natural for people (communication via messages). (Davis, 1988; Loomis, 1990)

9.0 Future Directions for Data Base Technology.

Further evolution of data base technology, just like at present and in the past, will be influenced by the following factors:

- ***Performance-oriented improvements to the basic technology.*** Performance is an area where improvements are always desired, particularly for frequently executed code. A DBMS is a collection of frequently executed programs which retrieve records, compare data values, lock records, update records, log changes, etc. DBMS performance may be enhanced as follows:

  - Further optimisation of programs using existing DBMSs.

  - The development of new, improved algorithms which include buffer management, recovery, concurrency control, query optimisation and compilation (Sellinger, 1987).

  - The coordination of data base algorithms with similar functions in the underlying operating system. For example, the coordination of the paging, done by the data base buffer manager and the operating system virtual memory manager, would avoid paging policy conflicts and reduce extra paging I/O required to resolve these conflicts (Atre, 1988).

  - Improved data modeling techniques could result in more efficient physical access paths to data, and compilation rather than interpretation of data base queries could also improve the system performance. Such concepts have already been introduced by IBM in DB2.

- ***New developments in hardware.*** Rapidly decreasing cost of hardware may have the following effects:

- A development of an in-memory data base. This could provide extremely fast service as no disk I/O would be required.

- Data base machines are already a realistic alternative to data base management. A ***data base machine*** is a dedicated hardware system which consists of multiple processors and its own operating system. The processors, depending on the architecture, include the data base processors (possibly in parallel), I/O processors, disk controller and other special purpose processors (Oppenheimer, 1989). The main purpose of data base machines is to offload the database management functions from the mainframe. This greatly improves data base performance and frees the mainframe resources for use in transaction processing, applications development, and all other processing that is best performed on the mainframe. An example of a commercially available data base machine is DBS/1012 manufactured by Teradata Corporation (Mayer-Lopez, 1990).

- As microprocessors become faster, as memory becomes cheaper, and as small disks grow in capacity, it will be possible to to enrich the functionality of PC based DBMSs so that they will rival those of mainframes. It is already possible to use a personal computer to access data stored on a mainframe or server workstation. Some vendors offer the capability of triggering a host DBMS update from a program running on a PC (Selinger, 1987).

• ***Customer demand for additional function.*** DBMS future functions may greatly depend upon user requests such as:

- The ability to store and manipulate efficiently nontraditional data such text, images, together with the traditional data records. OODBMS are being designed to handle rich, abstract user defined data types.

- Productivity and usability aids. As perceived by Schussel (1987, p. 15),

    * DBMS will evolve to be recognised as an essential but not sufficient component of productivity. Computer Aided Software Engineering (CASE) together with information and data base engineering are also required.

* Prototyping and data normalization will be universally adopted.

* Relational DBMS will continue to evolve, they will not be replaced by any other technology, at least for the next few years.

* Standard SQL will increase software portability.

* SQL will be discouraged as an end user language. Instead, SQL generators will be developed which will allow the use of natural language queries.

* Most products will continue to offer greater choices for internal schema and access methods.

* The trend toward interfacing with different products will continue.

* Active data dictionaries will be typical and expected.

* There will be continuing work to bring mainframe software products up to the syntactical sophistication of micro software.

▪ Distributed access to data. ***Distributed data base management*** is a type of multicomputer operation in which a large data base is partitioned into smaller physical fragments and distributed among multiple computers (Stevens, 1990, p. 12). Advantages of distribution may include improved reliability, better data availability, increased performance, reduced response time, and lower communication costs. The hardware and software components required for this new architecture are already available. Research is still under way on how workstation data base management systems should participate in a distributed data base management system, and how they can best cooperate to share data.

10.0 Summary.

A data base management system is a prerequisite technology for computer based management information systems, *MIS*, which help to accomplish managerial tasks by providing accurate and timely information.

Data base management systems evolved from file systems. Currently available DBMSs can be viewed as a three-level standard architecture which separates physical storage details from the logical view of data.

The most popular models, reviewed in this paper, include the hierarchical (IMS), network (DBTG) and relational (DB2) model.

The research on better ways of data management continues. The major goal is to accomplish maximum performance and functionality of DBMS, and maximum productivity in applications development.

The object-oriented paradigm is perceived to be as significant in the 1990s as the structured programming paradigm was in the 1970s. Other important areas of current research include distributed processing, data base machines and ***deductive data bases*** which use logic programming technology to extend the capabilities of relational data bases.

11.0 References.

1. Agha, G. (1990). Concurrent Object-Oriented Programming. Communications of the ACM, 33(9), 125-141.

2. Alfonseca, M. (1989). Frames, Semantic Networks, and Object-Oriented Programming in APL2. IBM Research and Development Journal, 33(5), 502-510.

3. Archer, J. B. (1989). IBM on the Repository, SAA, ADE and CASE. Data Base Newsletter, 17(2), 1 15-19.

4. Atkinson, C., Goldsack, S., Di Maio, A., Bayan, R. (1991, March/April). Object-Oriented Concurrency and Distribution in DRAGOON. Journal of Object-Oriented Programming, pp. 11-14 15-18.

5. Atkinson, M. P., Buneman, O. P. (1987). Types and Persitence in Data Base Programming Languages. ACM Computing Surveys, 19(2), 105-190.

6. Atre, S. (1988). Data Base: Structured Techniques for Design, Performance and Management. New York: John Wiley and Sons.

7. Ayers, T. R., Bary, D. K., Dolejsi, J. D., Galarneau J. R., Zoeller, R. V. (1991, July/August). Development of ITASCA. Journal of Object-Oriented Programming, pp. 46-49.

8. Barry, D. K. (1991, July/August). Perspectives on Changes for OODBMSs. Journal of Object-Oriented Programming,
pp. 19-20.

9. Beech, D. (1987, March). A foundation for Evolution from Relational to Object Database. Proceedings of the International Conference on Extending Data Base Technology. Venice.

10. Blasgen, M. W., Eswaran, K. P. (1977). Storage and Access in Relational Databases. IBM Systems Journal, 16(4), 363-378.

11. Bobrow, D. G. (1989, May). The Object of Desire. Datamation,
pp. 37-41.

12. Butterworth, P. (1991, July/August). ODBMS as Database Managers. Journal of Object-Oriented Programming,
pp. 55-57.

13. Codd, E. F. (1982). Relational Database: a Practical Foundation for Productivity. Communication of the ACM, 25(2), 109-117.

14. Codd, E. F. (1987). Data Base Management Systems: DB2 and IMS. Data Base Newsletter, 15(1), 1 9-10 13-15.

15. Codd, E. F. (1987). Questions and Answers Concerning Relational Languages. Data Base Newsletter, 15(4), 1 5-7.

16. Date, C. J. (1983). An Introduction to Data Base Systems. Massachusetts: Addison-Wesley Publishing Company.

17. Digitalk, (1991), Smalltalk/VPM Tutorial and Programming Handbook. Los Angeles: Digitalk Inc.

18. Freeman, P. (1975). Software Systems Principle a Survey. Chicago: Science Research Associates, Chicago.

19. Davis, J. M. (1988, October). Object Oriented Systems. Technical Support, pp. 13-14 16.

20. Fergusson R. K.(1983). Virtual Access Method. The complete Source Book for VSAM File Structures. Washington: Software Information Services.

21. Fry, J. P., Sibley, E. H. (1976). Evolution of Data-Base Management Systems. ACM Computing Surveys, 8(1), 7-42.

22. Gardarin, G., Valduriez, P. (1989). Relational Databases and Knowledge Bases. New York: Addison-Wesley Publishing Company.

23. Henderson-Sellers, B., Edwards, J. M. (1990). The Object-Oriented Systems Design. Communications of the ACM, 33(9), 143-159.

24. Gibbs, S., Tsichritzis, D., Casais, E. (1990). Class Management for Software Communities. Communications of the ACM, 33(9), 91-103.

25. Howard, B. (1990, October). Examining Migration Pattern to the New Silver Bullet, OOP. Computing, pp. 41-43.

26. IBM, (1983a), IBM DATABASE 2 Concepts and Facilities Guide (GG24-1582-00). Santa Teresa: International Systems Center.

27. IBM, (1983b), IBM DATABASE 2 SQL Usage Guide (GG24-1583-00). Santa Teresa: International Systems Center.

28. IBM, (1986), IMS DATA BASE Administration Guide (SH20-9025-10). Santa Teresa: International Systems Center.

29. IBM, (1987), DB2 Utilities (GG24-3130-00). Santa Teresa: International Systems Center.

30. IBM, (1988), DB2 Diagnosis Guide and Reference (LY27-9536-1). San Jose: IBM Corporation.

31. Ingari, F. (1991, July/August). The Object Database Market: Stranger than it Needs to be ?. Journal of Object-Oriented Programming, pp. 16-18.

32. Jordan, D. (1990). Implementation Benefits of C++ Language Mechanisms. Communications of the ACM, 33(9), 61-67.

33. Kadar, J. (1989). Trends in New Hardware and their Impact on Software. Technical Support, 3(5), 10-15.

34. Kadar, J. (1990).'Con Edison Data Base Administrators Juggle IMS and DB2. Technical Support, 4(11), 21-25.

35. Kap, D., Leben, J. F. (1986). IMS Programming Techniques. A guide to using DL/I. New York: Van Nostrad Reinhold Publishing Company.

36. Kent, W. (1991, June). A Rigorous Model of Object Reference, Identity, and Existence. Journal of Object-Oriented Programming, pp. 28-34 36.

37. Kim, W. (1990, February). Defining Object Databases Anew. Datamation, pp. 33-34 36.

38. Kim, W. (1991 July/August). Object-Oriented Database systems: Strengths and Weaknesses. Journal of Object-Oriented Programming, pp. 21-24 26-29.

39. King, R. (1989). Cactis: a Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System. ACM Transactions on Database Systems, 14(3), 291-321.

40. Korson, T., McGregor, J. D. (1990). September 1990 Communications of the ACM. Communications of the ACM, 33(9), 38-60.

41. Krasner, G. (1983). SMALLTALK-80 Bits of History, Words of Advice. Massachusetts: Addison-Wesley Publishing Company.

42. Lai, K. W. L., Guzenda, L. (1991 July/August). How to Benchmark an OODBMS. Journal of Object-Oriented Programming, pp. 12-15.

43. Lees, R. (1991, January). DB2 is Fast IMS is Slow. Enterprise Systems Journal, pp. 23 26 28.

44. Loomis, M. E. S. (1990). Object Database. Data Base Newsletter, 18(2), 1 10-15.

45. Mayer-Lopez, I. (1990). The Data Base Machine: What, How and Why?. Technical Support, 4(11), 18-20.

46. Meyer, B. (1990). Lessons from the Design of the EIFFEL Libraries. Communications of the ACM, 33(9), 69-88.

47. Odell, J. (1990). Object Orientation and its Software Implementation, Data Base Newsletter, 18(2), 1 5-9.

48. Oppenheimer, J. H. (1989). Database Computers; The Hardware Alternatives to RDBMS Software. Technical Support, 3(5), 52-55.

49. Popple, J. (1991). Legal Expert Systems: The Inadequacy of a Rule-Based Approach. The Australian Computer Journal, 23(1), 11-21.

50. Ricardo, C. M. (1990). Database Systems: Principles, Design, and Implementation. New York: Macmillian Publishing Company.

51. Roddick, J. F. (1991). Dynamically Changing Schemas within Database Models. The Australian Computer Journal, 23(3), 105-109.

52. Rolland, F. D. (1989). Relational Database Management with ORACLE. New York: Addison-Wesley Publishing Company.

53. Romero, R. (1988). Intelligent Data Bases: Bringing Expert Systems Capabilities to Database Systems. Technical Support, 2(9), 48-51.

54. Romero, R. (1988). Intelligent Database Communications with Baldur's IQ-200. Technical Support, 2(12), 39-40.

55. Ross, N.(1991). Birth of a Relational Data Base Migration Technology: the Challenge of CODASYL to DB2 Data Base Conversion. Technical Support. 5(11), 69-70.

56. Ross, R. G. (ed.). (1988). An Interview with James Martin Part I. Data Base Newsletter, 16(3), 1 14-19.

57. Ross, R. G. (ed.). (1988). An Interview with James Martin Part II. Data Base Newsletter, 16(4), 1 10-17.

58. Ross, R. G. (ed.). (1991). An Interview with Michael Stonbraker Part I. Data Base Newsletter, 19(1), 1 12-16.

59. Ross, R. G. (ed.). (1991). An Interview with Michael Stonbraker Part II. Data Base Newsletter, 19(3), 1 13-19.

60. Ross, R. G. (ed.). (1991). Third-Generation Database Systems Manifesto. Database Newsletter, 19(1), 1 3-11.

61. Rotzel, K., Loomis, M. E. S. (1991, March/April). Benchmarking an ODBMS. Journal of Object-Oriented Programming, pp. 66-70 72.

62. Schussel, G. (1987). Future Forecasts: DBMS Software, Data Base Newsletter, 15(4), 15.

63. Selinger, P. G. (1987). Database Technology. IBM System Journal, 26(1), 96-106.

64. Senko, M. E. (1977). Data Structures and Data Accessing in Data Base Systems Past, Present, Future. IBM Systems Journal, 16(3), 208-257.

65. Snell, B. (1984). The Relational Database Model and its Implementation in Database 2. Mountain View, Ca: National Advanced Systems Corporation.

66. Stevens, O. (1990). Distributed Database and Cooperative Processing a Continuing Evolution. Technical Support, 4(2), 16-21.

67. Szakach, G. (1991). Adabas/Natural Performance Monitoring. Technical Support, 5(8), 29-32.

68. Taylor, R. W., Frank, R. L. (1976). CODASYL Data Base Mangement Systems. ACM Computing Surveys, 8(1), 67-104.

69. Vaghani, J., Kotagiri, R., Kemp, D., Somogui, P. (1991).. An Introduction to the ADITI Deductive Database System. The Australian Computer Journal, 23(3), 37-52.

70. Van Fleet, D. (1989). IBM on Database, SAA, Distributed Database, and Related Trends. Data Base Newsletter, Special Print, pp. 1-8.

71. Wirfs-Brock, R., Johnson, R. E. (1990). Surveying Current Research in Object-Oriented Design. Communications of the ACM, 33(9), 105-123.