DATABASE SYSTEMS

KASNEB CICT PAPER NO.7

7.0 LEARNING OUTCOMES
A candidate who passes this paper should be able to:
• Write structured query language (SQL) statements to manipulate data in Databases
• Develop a database application
• Handle transactions and concurrency controls
• Administer databases
• Integrate databases and other applications
• Manage database integrity issues

CONTENT

7.1 Introduction to databases
- Files, records, files and databases
- History of database systems
- Traditional file systems versus the database approach
- Characteristics, importance and limitations of database systems
- Database components and architecture

7.2 File organisation techniques
- Storage structures and blocking
- Unordered files
- Sequential files
- Indexing

7.3 Database models
- The role of data modeling
- The hierarchical model
- The relational model
- The object-oriented model
- The object-relational model

7.4 Database development life cycle
- Data and user requirements specification
- Stages of database development
- Conceptual, logical and physical database design
- Writing database requirements specifications

7.5 Relational database model
- Relational database concepts and properties
- E-R database design
- Database design anomalies
- Normalisation
- Relational algebra
- Creating database design
- Implementing database design in mysql/oracle /082

7.6 Structured query language (SQL)

- Data definition language
- Data manipulation language
- Structure of SOL statements
- Data control
- In-built functions
- Writing SQL statements
- Using SQL functions

7. 7 Transaction management and concurrency control

- Transaction management
- Properties of a transaction
- Serialisability and concurrency control
- Lock-based and timestamp-based protocols
- Types of failures
- Database recovery concepts and mechanisms

7.8 Database administration

- Database users
- Data administration
- Functions and roles of database administrators
- Monitoring database performance

7.9 Database security and Integrity

- Security and integrity concepts
- Social, ethical and legal database issues
- Threats to database security and integrity
- Managing threats
- Establishing data backup procedure

7.10 Distributed database systems

- Introduction of concepts
- Distribution methods - fragmentation and replication
- Concurrency control mechanisms in distributed systems
- Two-tier database architecture
- Three-tier database architecture

7.11 Data warehousing and data mining

- Overview of data warehousing
- Characteristics of a data warehouse
- Components of a data warehouse
- Types of data warehouses
- Elements of a data warehouse
- Over view of data mining
- Techniques of data mining

7.12 Integrating databases to other applications

- Importance of integrating databases to other applications
- Integrating databases to other applications (visual basic. net, C++, Java, C# among others)
- Developing web enabled database applications

7.13 Emerging trends in database systems

7.1 Introductions to data bases.

7.1.1 Data hierarchy.

Data Hierarchy refers to the systematic organization of data, often in a hierarchical form. Data organization involves fields, records, files and so on.

A data field holds a single fact or attribute of an entity. Consider a date field, e.g. "September 19, 2004". This can be treated as a single date field (e.g. birthdate), or 3 fields, namely, month, day of month and year.

A record is a collection of related fields. An Employee record may contain a name field(s), address fields, birthdate field and so on.

A file is a collection of related records. If there are 100 employees, then each employee would have a record (e.g. called Employee Personal Details record) and the collection of 100 such records would constitute a file (in this case, called Employee Personal Details file).

Files are integrated into a database. This is done using a Database Management System. If there are other facets of employee data that we wish to capture, then other files such as Employee Training History file and Employee Work History file could be created as well.

7.1.2History of Database Systems (see also Wikipedia notes)

Early Manual System

- Before-1950s
- Data was stored as paper records.
- Lot of man power involved.
- Lot of time was wasted. e.g. when searching
- Therefore inefficient.

Revolution began

- 1950s and early 1960s:
- Data processing using magnetic tapes for storage
- Tapes provide only sequential access
- Punched cards for input
- Late 1960s and 1970s:
- Hard disks allow direct access to data
- Data stored in files
- Known as File Processing System

File based systems

- Adequate for small applications
- **Drawbacks**
- Separation and isolation of data
- Each program maintains its own set of data.
- Users of one program may be unaware of potentially useful data held by other programs
- Duplication of data
- Same data is held by different locations.
- Wasted space and potentially different values and/or different formats for the same item.

 Data dependence
 File structure is defined in the program code.
 Incompatible file formats
 Programs are written in different languages, and so cannot easily access each other's files.
 Fixed Queries/Proliferation of application programs
 Programs are written to satisfy particular functions.
Any new requirement needs a new program.

Database Approach

 Arose because:
 Definition of data was embedded in application programs, rather than being stored separately and independently.
 No control over access and manipulation of data beyond that imposed by application programs.
 Result:
The database and Database Management System (**DBMS**).

Database Management Systems (DBMS)

Relational
Object-oriented Object-relational
XML
IMDB
Java
1960's Hierarchical Network
1970's
1990's
CMDB Mobile
Embedded
1995+

Hierarchical Model

Well suited for data which are in some way related Hierarchically begin with a strictly defined tree of data nodes Each node can contain some identifying data, plus a set of sub nodes of a specific child type

Network Model

 Supported more complex relations
 Physical file pointers were used to model the relations between files
 Relations had to be decide in advance
Most suitable for large databases with well-defined queries and well-defined applications.

Relational Model (1970's)

 E.F. Codd introduced the relational model in 1970
Provides a conceptually simple model for data as relations (Typically considered "tables") with all data visible.
 DB2 from IBM is the first DBMS product based on the relational model
 Other DBMS based on the relational model were developed in the late 1980s

 Today, DB2, Oracle, and SQL Server are the most prominent commercial DBMS products based on the relational model

Object Oriented Data Model (1990's)
 Goal of OODBMS is to store object-oriented programming objects in a database without having to transform them into relational format.
 Extend the entity-relationship data model by including encapsulation, methods and object identity
Object-relational models
 Extend the relational data model by including object orientation and constructs to deal with added data types.
 Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
 Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
Modern Database Management Systems
 DBMS are large complex pieces of software designed specifically for the efficient management of data.
 Examples:
 Oracle (Oracle Corporation)
 Ingres (Computer Associates)
 SQL Server (Microsoft Corporation)
 Access (Microsoft Corporation)
 IMS, DB2 (IBM)
And many more…s

7.1.3 Traditional file system versus the database approach

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.
Database Management System (DBMS):
Database Management System (DBMS) is a software package that allows data to be effectively stored, retrieved and manipulated and the data stored in a DBMS package can be accessed by multiple users and by multiple application programs like (SQL Server, Oracle, MS-Access) .

Comparison of Traditional File-Based Approach and Database Approach
At the beginning, you should understand the rationale of replacing the traditional file-based system with the database system.

File-based System
File-based systems were an early attempt to computerize the manual filing system. File-based system is a collection of application programs that perform services for the end-users. Each program defines and manages its data.
However, five types of problem occurred in using the file-based approach:

FOR FULLNOTES
CALL/TEXT:0713440925

1. Separation and isolation of data

When data is isolated in separate files, it is more difficult for us to access data that should be available. The application programmer is required to synchronize the processing of two or more files to ensure the correct data is extracted.

2. Duplication of data

When employing the decentralized file-based approach, the uncontrolled duplication of data is occurred. Uncontrolled duplication of data is undesirable because:

I. Duplication is wasteful
ii. Duplication can lead to loss of data integrity

3. Data dependence

Using file-based system, the physical structure and storage of the data files and records are defined in the application program code. This characteristic is known as **program-data dependence**. Making changes to an existing structure are rather difficult and will lead to a modification of program. Such maintenance activities are time-consuming and subject to error.

4. Incompatible file formats

The structures of the file are dependent on the application programming language. However file structure provided in one programming language such as direct file, indexed-sequential file which is available in COBOL programming, may be different from the structure generated by other programming language such as C. The direct incompatibility makes them difficult to process jointly.

5. Fixed queries / proliferation of application programs

File-based systems are very dependent upon the application programmer. Any required queries or reports have to be written by the application programmer. Normally, a fixed format query or report can only be entertained and no facility for ad-hoc queries if offered.

Database Approach:

In order to overcome the limitations of the file-based approach, the concept of database and the Database Management System (DMS) was emerged in 60s.

Advantages

A number of advantages of applying database approach in application system are obtained including:

1. Control of data redundancy

The database approach attempts to eliminate the redundancy by integrating the file. Although the database approach does not eliminate redundancy entirely, it controls the amount of redundancy inherent in the database.

2. Data consistency

By eliminating or controlling redundancy, the database approach reduces the risk of inconsistencies occurring. It ensures all copies of the data are kept consistent.

3. More information from the same amount of data

With the integration of the operated data in the database approach, it may be possible to derive additional information for the same data.

4. Sharing of data

Database belongs to the entire organization and can be shared by all authorized users.

5. Improved data integrity

Database integrity provides the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate.

6. Improved security

Database approach provides a protection of the data from the unauthorized users. It may take the term of user names and passwords to identify user type and their access right in the operation including retrieval, insertion, updating and deletion.

7. Enforcement of standards

The integration of the database enforces the necessary standards including data formats, naming conventions, documentation standards, update procedures and access rules.

8. Economy of scale

Cost savings can be obtained by combining all organization's operational data into one database with applications to work on one source of data.

9. Balance of conflicting requirements

By having a structural design in the database, the conflicts between users or departments can be resolved.Decisions will be based on the base use of resources for the organization as a whole rather that for an individual entity.

10. Improved data accessibility and responsiveness

By having integration in the database approach, data accessing can be crossed departmental boundaries. This feature provides more functionality and better services to the users.

11. Increased productivity

The database approach provides all the low-level file-handling routines. The provision of these functions allows the programmer to concentrate more on the specific functionality required by the users. The fourth-generation environment provided by the database can simplify the database application development.

12. Improved maintenance

Database approach provides a data independence. As a change of data structure in the database will be affect the application program, it simplifies database application maintenance.

13. Increased concurrency

Database can manage concurrent data access effectively. It ensures no interference

FOR FULLNOTES
CALL/TEXT:0713440925

between users that would result any loss of information or loss of integrity.

14. Improved backing and recovery services

Modern database management system provides facilities to minimize the amount of processing that can be lost following a failure by using the transaction approach.

Disadvantages

In split of a large number of advantages can be found in the database approach, it is not without any challenge. The following disadvantages can be found including:

1. Complexity

Database management system is an extremely complex piece of software. All parties must be familiar with its functionality and take full advantage of it. Therefore, training for the administrators, designers and users is required.

2. Size

The database management system consumes a substantial amount of main memory as well as a large number amount of disk space in order to make it run efficiently.

3. Cost of DBMS

A multi-user database management system may be very expensive. Even after the installation, there is a high recurrent annual maintenance cost on the software.

4. Cost of conversion

When moving from a file-base system to a database system, the company is required to have additional expenses on hardware acquisition and training cost.

5. Performance

As the database approach is to cater for many applications rather than exclusively for a particular one, some applications may not run as fast as before.

6. Higher impact of a failure

The database approach increases the vulnerability of the system due to the centralization. As all users and applications reply on the database availability, the failure of any component can bring operations to a halt and affect the services to the customer seriously.

7.1.4 Characteristics ,importance and limitations of database systems(already covered above)

7.1.5 Components and architecture

Elements of Database System

- ✓ Database schema
- ✓ Schema objects
- ✓ Indexes
- ✓ Tables
- ✓ Fields and columns
- ✓ Records and rows

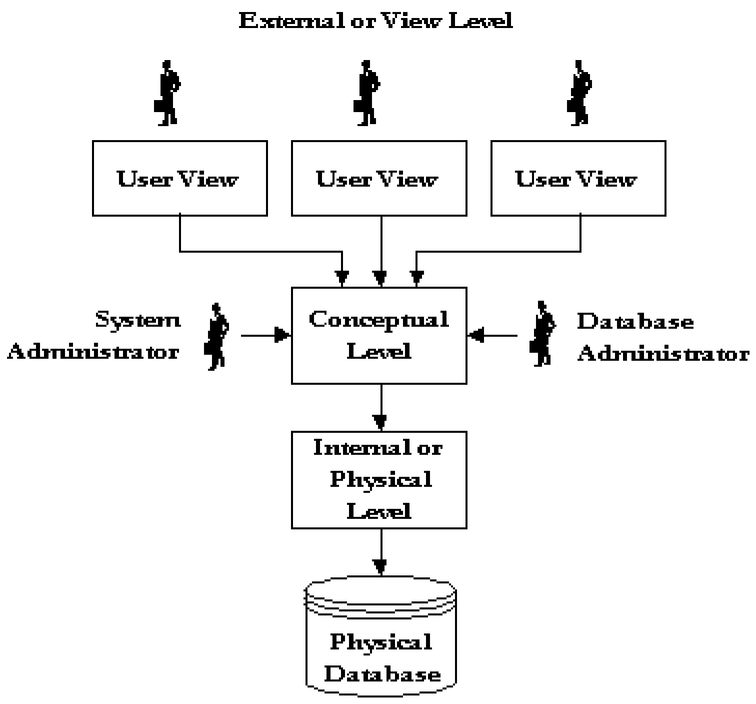
- ✓ Keys
- ✓ Relationships
- ✓ Data types

DBMS Architecture:

1. ***External view***: This is a highest level of abstraction **as seen by user**. This level of abstraction describes only the part of entire database. It is based on the conceptual model, is the end user view of data environment. Each external view described by means of a schema called an **external schema or subschema.**

2. ***Conceptual level***: At this level of database abstraction **all the database entities and the relationships among them** are included. One conceptual view represents the entire database. The conceptual schema defines this conceptual view.

3. ***Internal (physical) level***: This lowest level of abstraction. Its closest to physical storage device. It describes how data are actually stored on the storage medium. The internal schema, which contains the definition of the stored record, the method representing the data fields, expresses the internal view and the access aids used.



Data Independence:

1. The ability to modify a scheme definition in one level without affecting a scheme definition in a higher level is called **data independence**.
2. There are two kinds:
   - **Physical data independence**
     - The ability to modify the <u>physical scheme</u> without causing application programs to be rewritten
     - Modifications at this level are usually to improve performance

FOR FULLNOTES CALL/TEXT:0713440925

- **Logical data independence**
  - The ability to modify the <u>conceptual scheme</u> without causing application programs to be rewritten
  - Usually done when logical structure of database is altered

3. Logical data independence is harder to achieve as the application programs are usually heavily dependent on the logical structure of the data. An analogy is made to abstract data types in programming languages.

7.2 <u>file organization techniques (notes in pdf format in this file)</u>

7.2.1 Storage structures and blocking
7.2.2 Unordered files(pile/ heap/serial)
7.2.3 Sequential files(ordered)
7.2.4 Indexing

Basic file organization techniques

Given that a file consists, generally speaking, of a collection of records, a key element in file management is the way in which the records themselves are organized inside the file, since this heavily affects system performances as far as record finding and access. Note carefully that by ``organization'' we refer here to the *logical* arrangement of the records in the file (their ordering or, more generally, the presence of ``closeness'' relations between them based on their content), and not instead to the physical layout of the file as stored on a storage media, To prevent confusion, the latter is referred to by the expression ``**record blocking''**, and will be treated later on.

Choosing a file organization is a design decision, hence it must be done having in mind the achievement of good performance with respect to the most likely usage of the file. The criteria usually considered are:

1. Fast access to single record or collection of related records.
2. Easy record adding/update/removal, without disrupting (1).
3. Storage efficiency.
4. Redundancies as a warranty against data corruption.

Needless to say, these requirements are in contrast with each other for all but the most trivial situations, and it's the designer job to find a good compromise among them, yielding and adequate solution to the problem at hand. For example, easiness of adding/etc. is not an issue when defining the data organization of a CD-ROM product, whereas fast access is, given the huge amount of data that this media can store. However, as it will become apparent shortly, fast access techniques are based on the use of additional information about the records, which in turn competes with the high volumes of data to be stored.

Logical data organization is indeed the subject of whole shelves of books, in the ``Database'' section of your library. Here we'll briefly address some of the simpler used techniques, mainly because of their relevance to data management from the lower-level (with respect to a database's) point of view of an OS. Five organization models will be considered:

- Pile/heap/serial(unordered records)
- Sequential(ordered records)
- Indexed-sequential.
- Indexed.
- Hashed.

Pile

It's the simplest possible organization: the data are collected in the file **in the order in which they arrive,** and it's not even required that the records have a common format across the file (different fields/sizes, same fields in different orders, etc. are possible). This implies that each record/field must be self-describing. Despite the obvious storage efficiency and the easy update, it's quite clear that this ``structure'' is not suited for easy data retrieval, since retrieving a datum basically requires detailed analysis of the file content. It makes sense only as temporary storage for data to be later structured in some way.

Sequential

This is the most common structure for large files that are typically processed in their entirety, and it's at the heart of the more complex schemes. In this scheme, <u>all the records have the same size and the same field format, with the fields having fixed size as well.</u> The records are sorted in the file according to the content of a field of a scalar type, called ``key''. The key must identify uniquely a records, hence different record have different keys. This organization is well suited for **batch processing** of the entire file, without adding or deleting items: this kind of operation can take advantage of the fixed size of records and file; moreover, this organization is easily stored both on disk and tape. The key ordering, along with the fixed record size, makes this organization amenable to dicotomic search◇ However, adding and deleting records to this kind of file is a tricky process: the logical sequence of records typically matches their physical layout on the media storage, so to ease file navigation, hence adding a record and maintaining the key order requires a reorganization of the whole file. The usual solution is to make use of a ``log file'' (also called ``transaction file''), structured as a pile, to perform this kind of modification, and periodically perform a batch update on the master file.

Indexed sequential

An index file can be used to effectively overcome the above mentioned problem, and to speed up the key search as well. The simplest indexing structure is the single-level one: a file whose records are pair's key-pointer, where the pointer is the position in the data file of the record with the given key. Only a subset of data records, evenly spaced along the data file, are indexed, so to mark intervals of data records.

A key search then proceeds as follows: the search key is compared with the index ones to find the highest index key preceding the search one, and a linear search is performed from the record the index key points onward, until the search key is matched or until the record pointed by the next index entry is reached. In spite of the double file access (index + data)

FOR FULLNOTES CALL/TEXT:0713440925

needed by this kind of search, the decrease in access time with respect to a sequential file is significant.

Consider, for example, the case of simple linear search on a file with 1,000 records. With the sequential organization, an average of 500 key comparisons is necessary (assuming uniformly distributed search key among the data ones). However, using and evenly spaced index with 100 entries, the number of comparisons is reduced to 50 in the index file plus 50 in the data file: a 5:1 reduction in the number of operations.

This scheme can obviously be hierarchically extended: an index is a sequential file in itself, amenable to be indexed in turn by a second-level index, and so on, thus exploiting more and more the hierarchical decomposition of the searches to decrease the access time. Obviously, if the layering of indexes is pushed too far, a point is reached when the advantages of indexing are hampered by the increased storage costs and by the index access times as well.

Indexed

Why using a single index for a certain key field of a data record? Indexes can be obviously built for each field that uniquely identifies a record (or set of records within the file), and whose type is amenable to ordering. Multiple indexes hence provide a high degree of flexibility for accessing the data via search on various attributes; this organization also allows the use of variable length records (containing different fields).

It should be noted that when multiple indexes are used the concept of sequentially of the records within the file is useless: each attribute (field) used to construct an index typically imposes an ordering of its own. For this very reason is typically not possible to use the ``sparse'' (or ``spaced'') type of indexing previously described. Two types of indexes are usually found in the applications: the exhaustive type, which contains an entry for each record in the main file, in the order given by the indexed key, and the partial type, which contain an entry for all those records that contain the chosen key field (for variable records only).

Hashed

As with sequential or indexed files, a key field is required for this organization, as well as fixed record length. However, no explicit ordering in the keys is used for the hash search, other than the one implicitly determined by a hash function.

7.3 database models

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model (or the SQL approximation of relational), which uses a table-based format.
Common logical data models for databases include:

- Hierarchical database model
- Network model

- Relational model
- Entity–relationship model
- Enhanced entity–relationship model
- Object model
- Document model
- Entity–attribute–value model
- Star schema

An object-relational database combines the two related structures.
Physical data models include:

- Inverted index
- Flat file

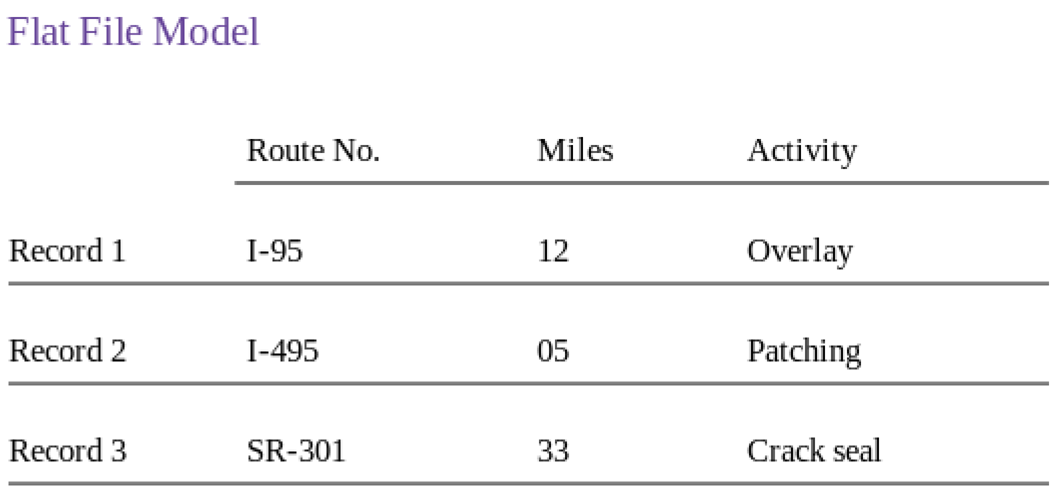
Other models include:

- Associative model
- Multidimensional model
- Multivalue model
- Semantic model
- XML database
- Named graph

7.3.1 The role of data modeling
7.3.2 Flat model

Simple database design consisting of one large table instead of several interconnected tables of a relational database. Called 'flat' because of its only two dimensional (data fields and records) structure, these databases cannot represent complex data relationships. Also called flat file database or flatform database

A flat file database describes any of various means to encode a database model (most commonly a table) as a single file. A flat file can be a plain text file or a binary file. There are usually no structural relationships between the records.


Flat File Model

	Route No.	Miles	Activity
Record 1	I-95	12	Overlay
Record 2	I-495	05	Patching
Record 3	SR-301	33	Crack seal

FOR FULLNOTES
CALL/TEXT:0713440925

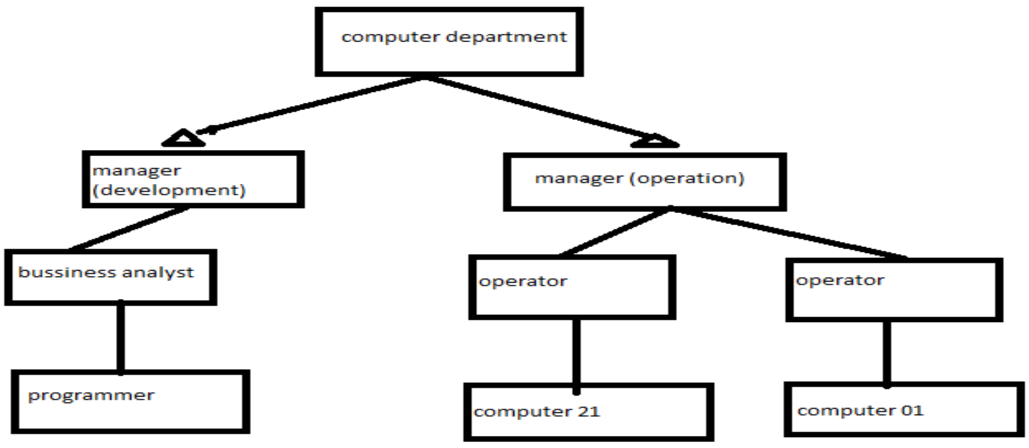
A flat file database is a database that stores data in a plain text file. Each line of the text file holds one record, with fields separated by delimiters, such as commas or tabs. While it uses a simple structure, a flat file database cannot contain multiple tables like a relational database can. Fortunately, most database programs such as Microsoft Access and FileMaker Pro can import flat file databases and use them in a larger relational database.

Flat file is also a type of computer file system that stores all data in **a single directory**. There are no folders or paths used to organize the data. While this is a simple way to store files, a flat file system becomes increasingly inefficient as more data is added. The original Macintosh computer used this kind of file system, creatively called the Macintosh File System (MFS). However, it was soon replaced by the more efficient Hierarchical File System (HFS) that was based on a directory structure

7.3.3 The hierarchical model

This model allows the data to be structured in a parent /child relationship (each parent may have many children, but each child would be restricted to having only one parent). Under this model, it’s difficult to express relationships when children need to relate to more than one parent. When the data relationships are hierarchical, the database is easy to implement, modify and search.

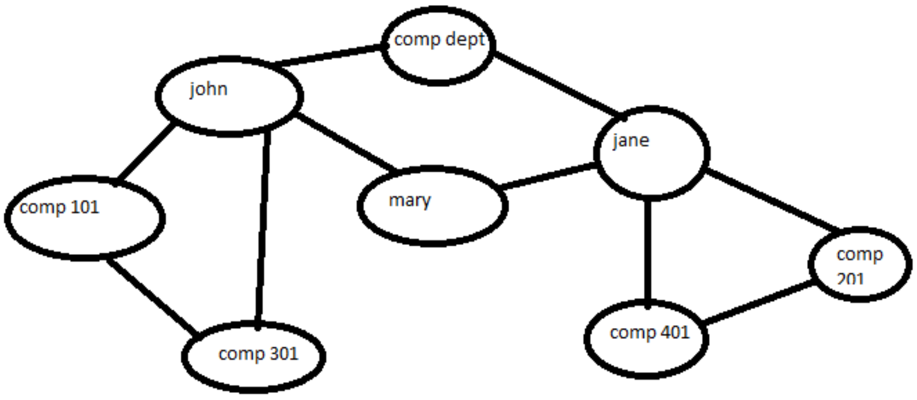
A hierarchical structure has only one root. Each parent can have numerous children but a child can have only one parent. Subordinate segment are retrieved through the parent segment. Reverse pointers are not allowed. Pointers can be set only for nodes on a lower level; they cannot be set to a node a predetermined access path.



7.3.4 Network models

The model allows children to relate to more than one parent. A disadvantage to the network model is that such structure can be extremely complex and difficult to comprehend, modify or reconstruct in case of failure. The network structure is effective in stable environments where the complex interdependence of the requirement have been clearly defined.

The network structure is more flexible, yet more complex, than the hierarchical structure. Data records are related through **logical entities called sets**. Within a network, any data element can be connected to any item. Because networks allow reverse pointers, an item can be an owner and a member of the same data. Members are grouped together to form records, and records are linked together to form a set. A set can have only one owner record but several member records.



7.3.5 Relational model

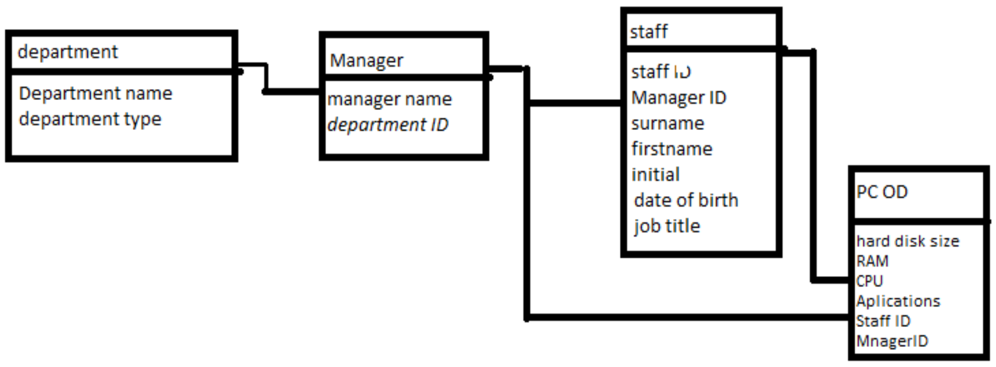
The model is independent from the physical implementation of the data structure. The relational database organization has many advantages over the hierarchical and network database models. They are

- ✓ Easier for users to understand and implement in a physical database system
- ✓ Easier to convert from other database structure.
- ✓ Projection and joint operations (referencing groups of related data elements not stored together) are easier to implement and creation of new relations for applications is easier to do.
- ✓ Access control over sensitive data is easy to implement
- ✓ Faster in data search
- ✓ Easier to modify than hierarchical or network structure

Relational database technology separates data from the application and uses a simplified data model. Based on set theory and relational calculations, relational database models information in a table structure with columns and rows. Columns, called domains or attributes, correspond to fields. **Row or tuples are equal to records in a conventional file**

FOR FULLNOTES
CALL/TEXT:0713440925

structure. Relational database use normalization rules to minimize the amount of information needed in tables to satisfy users’ structure and unstructured queries to the database.



7.3.6 The object-oriented model

An alternative strategy for converting isa-hierarchies to relations is to enumerate all the possible sub trees of the hierarchy. For each, create one relation that represents entities that have components in exactly those sub trees; the schema for this relation has all the attributes of any entity set in the sub tree. We refer to this approach as "object-oriented," since it is motivated by the assumption that entities are "objects" that belong to one and only one class.

Example:

There are four possible subtrees in movie library DBMS including the root:

1. *Movies* alone.
2. Movies and *Cartoons* only.
3. *Movies* and *Murder-Mysteries* only.
4. All three entity sets.

OOM must construct relations for all four "classes." Since only Murder-Mysteries contribute an attribute that is unique to its entities, there is actually some repetition, and these four relations are:

Movies (title, year, length, film~~~e)
Movies C (title, year, length, film~~~e)
Movies MM (title, year, length, film Type, weapon)
Movies CMM (title, year, length, film Type, weapon)

Had *Cartoons* had attributes unique to that entity set, then all four relations would have different sets of attributes. As that is not the case here, we could combine Movies with Movies C (i.e., create one relation for non-murder mysteries) and combine Movies MM with Movies CMM (i.e., create one relation for all murder mysteries), although doing so loses some information i.e. which movies are cartoons.

We also need to consider how to handle the relationship *Voices* from *Cartoons* to *Stars.* If *Voices* were many-one from *Cartoons,* then we could add a voice attribute to Movies C and Movies CMM, which would represent the *Voices* relationship and would have the side-effect of making all four relations different.
However, *Voices* is many-many, so we need to create a separate relation for this relationship. As always, its schema has the key attributes from the entity sets connected; in this case Voices (title, year, star name) would be an appropriate schema.
One might consider whether it was necessary to create two such relations, one connecting cartoons that are not murder mysteries to their voices, and the other for cartoons that *are* murder mysteries. However, there does not appear to be any benefit to doing so in this case.
Object-oriented Data Model(s): several models have been proposed for implementing in a database system. One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE). Additionally, systems like $O_{2,}$ ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).

Additional notes

Definition - What does Object-Oriented Modeling (OOM) mean?

Object-oriented modeling (OOM) is the construction of objects using a collection of objects that contain stored values of the instance variables found within an object. Unlike models that are record-oriented, object-oriented values are solely objects.

The object-oriented modeling approach creates the union of the application and database development and transforms it into a unified data model and language environment. Object-oriented modeling allows for object identification and communication while supporting data abstraction, inheritance and encapsulation.

Object-Oriented Modeling (OOM)

Object-oriented modeling is the process of preparing and designing what the model’s code will actually look like. During the construction or programming phase, the modeling techniques are implemented by using a language that supports the object-oriented programming model.

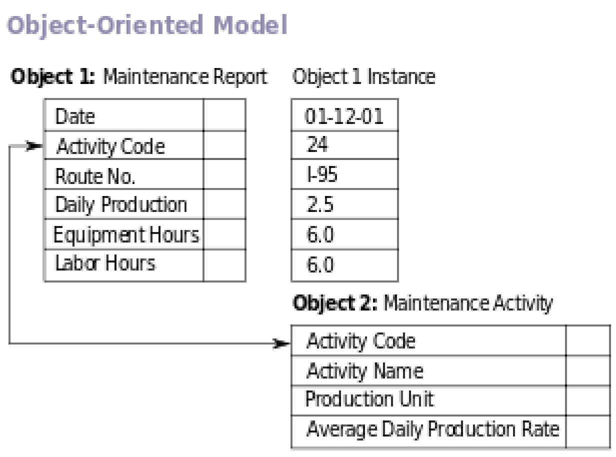
OOM consists of progressively developing object representation through three phases: analysis, design, and implementation. During the initial stages of development, the model developed is abstract because the external details of the system are the central focus. The model becomes more and more detailed as it evolves, while the central focus shifts toward understanding how the system will be constructed and how it should function.

7.3.7 The object-relational models

An **object-relational database** (**ORD**), or **object-relational database management system** (**ORDBMS**), is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly

FOR FULLNOTES
CALL/TEXT:0713440925

supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data-types and methods.



Example of an object-oriented database model.

An object-relational database can be said to provide a middle ground between relational databases and *object-oriented databases* (OODBMS). In object-relational databases, the approach is essentially that of relational databases: the data resides in the database and is manipulated collectively with queries in a query language; at the other extreme are OODBMSes in which the database is essentially a persistent object store for software written in an object-oriented programming language, with a programming API for storing and retrieving objects, and little or no specific support for querying.

Additional notes

What is the Object-Relational Model?

The object-relational model is designed to provide a relational database management that allows developers to integrate databases with their data types and methods. It is essentially a relational model that allows users to integrate object-oriented features into it.

This design is most recently shown in the Nordic Object/Relational Model. The primary function of this new object-relational model is to more power, greater flexibility, better performance, and greater data integrity then those that came before it.

Some of the benefits that are offered by the Object-Relational Model include:

- **Extensibility** – Users are able to extend the capability of the database server; this can be done by defining new data types, as well as user-defined patterns. This allows the user to store and manage data.
  .

- **Complex types** – It allows users to define new data types that combine one or more of the currently existing data types. Complex types aid in better flexibility in organizing the data on a structure made up of columns and tables.
  .
- **Inheritance** – Users are able to define objects or types and tables that procure the properties of other objects, as well as add new properties that are specific to the object that has been defined.
  .
- A field may also contain an object with attributes and operations.
  ..
- Complex objects can be stored in relational tables.

The object-relational database management systems which are also known as ORDBMS, these systems provide an addition of new and extensive object storage capabilities to the relational models at the center of the more modern information systems of today.

These services assimilate the management of conventional fielded data, more complex objects such as a time-series or more detailed geospatial data and varied dualistic media such as audio, video, images, and applets.

This can be done due to the model working to summarize methods with data structures, the ORDBMS server can implement complex analytical data and data management operations to explore and change multimedia and other more complex objects.

What are some of the functions and advantages to the Object-Relational Model?

It can be said that the object relational model is an evolutionary technology, this approach has taken on the robust transaction and performance management aspects of its predecessors and the flexibility of the object-oriented model.

Database developers can now work with somewhat familiar tabular structures and data definition but with more power and capabilities. This also allows them to perform such task all the while assimilating new object management possibilities. Also the query and procedural languages and the call interfaces in the object relational database management systems are familiar.

The main function of the object relational model is to combine the convenience of the relational model with the object model. The benefits of this combination range from scalability to support for rich data types.

However, the relational model has to be drastically modified in order to support the classic features of the object oriented programming. This creates some specific characteristics for the object-relational model.

Some of these characteristics include:

- Base Data type extension
- Support complex objects

FOR FULLNOTES CALL/TEXT:0713440925

- Inheritance (which we discussed in more detail above.)
- And finally Rule systems

Object-relational models allow users to define data types, function, and also operators. As a direct result of this the functionality and performance of this model are optimized. The massive scalability of the object-relational model is its most notable advantage, and it can be seen at work in many of today's vendor programs.

The History of the Object-Relational Model

As said before the Object-Relational model is a combination of the Relational Model and the Object Oriented Model. The Relational Model made its way into the world of data in the 1970s; it was a hit but managed to leave developers wanting more flexibility and capability.

The Object Oriented model seemed to move into the spot light in the 1990s, the idea of being able to store object oriented data was a hit, but what happened to the relational data?

Later in the 1990s the Object-Relational model was developed, combining the advantages of its most successful predecessors such as; user defined data types, user defined functions, and inheritance and sub-classes.

This model grew from the research conducted in the 1990s. The researches many goal was to extend the capabilities of the relational model by including objects oriented concepts. It was a success.

What about Object Relational Mapping?

Object-Relational mapping is a programming method used to convert data between incompatible data type systems in relational databases and object oriented languages. Here are some basics when it comes to mapping. Java classes can be mapped to relational database management systems tables.

The easiest way to begin mapping between an enduring class and a table is one-on-one. In a case such as this, all of the attributes in the enduring class are represented by all of the columns of the table. Each case in point of a business class is then in turn stored in a row of that table.

Though this particular type of mapping is pretty straightforward, it can conflict with the existing object and entity-relation (E-R) models. This is partially due to the fact that the goal of the object modeling is to model an organizational process using real world objects, (we will discuss more about object modeling in the following article), whereas the goal of entity-relational modeling is to normalize and retrieve data in a quick manner.

Due to this two types of class to table modeling methods have been adopted by most users. This was to help overcome the issues caused by differences between the relational and object models. The two methods are known as SUBSET mapping and SUPERSET mapping.

Let's talk briefly about these two methods.

With SUBSET Mapping the attributes of a persistent class, or described above as an enduring class, represent either **a section of the columns in a table or all of the columns in the table.**

SUBSET Mapping is used mostly when all of the attributes of a class are mapped to the same table. This method is useful also when a class is not concerned with a portion of the columns of its table in the database due to the fact that they are not a part of the business model.

SUBSET Mapping is used to create projection classes as well for tables with a sizable number of columns. A projection class contains enough information to enable the user to choose a row for complete retrieval from a database.

This essentially reduces the amount of information passed throughout the network. This type of mapping can also be used to help may a class inheritance tree to a table of using filters.

Now let's consider SUPERSET Mapping. With a persistent class the superset mapping method **holds attributes taken from columns of more than one table**. This particular method of mapping is also known as **table spanning**.

Mapping using the SUPERSET method is meant to create view classes that cover the underlying data model, or to map a class inheritance tree to a database by using a Vertical mapping tactic.

The final word

There are millions of other aspects and advantages to this model. The Object-Relational model does what no other single model before it could do. By combining the strongest points of those that did come before it, this model has surpasses expectations, and taken on a definitive role in database technology. Despite what models follow it, this model is here to stay.

7.4 data base development application life cycles

Introduction: Database Development Life Cycle
Abstract

A software development life cycle model (SDLC) consists of a set of processes (planning, requirements, design, development, testing, installation and maintenance) defined to accomplish the task of developing a software application that is functionally correct and satisfies the user's needs. These set of processes, when arranged in different orders, characterize different types of life cycles. When developing a database, the order of these tasks is very important to efficiently and correctly transform the user's requirements into an operational database. These SDLCs are generally defined very broadly and are not specific for a particular type of application. In this paper the authors emphasize that there should be a SDLC that is specific to database applications. Database applications do not have the same characteristics as other software applications and thus a specific database development life

FOR FULLNOTES CALL/TEXT:0713440925

cycle (DBDLC) is needed. A DBDLC should accommodate properties like scope restriction, progressive enhancement, incremental planning and pre-defined structure.

Keywords: Software Development, Database, DBMS, lifecycle model, traditional lifecycles

Introduction

Database management systems are generally categorized as transaction processing systems, decision support systems and/or knowledge-based systems. During their development each of these types of DBMS introduces different problems and challenges. Traditionally, SDLC models designed for developing DBMS followed the design-first-implement-later approach because of the DBMS were mainly of the transaction processing type [Wetzel and Kerschbergl, 1989]. The authors believe, as we will explain later, that the design-first-implement-later approach does not work for the databases underlying data mining or knowledge-base systems or for that matter for any system where the requirements change very frequently.

Some of the traditional SDLCs models used for software development are: waterfall, prototypes, spiral and rapid application development (RAD). These life cycles models are defined broadly in terms of what each individual phase accomplish, the input and output documents it produces or requires, and the processes that are necessary in completing each phase. In general, the output deliverables from the previous phase serve as an input to the next phase. However, in these models it can be observed also that usually there is no interaction between two consecutive phases; therefore, no feedback between these phases exists. When creating a database system the feedback between some of the life cycle phases is very critical and necessary to produce a functionally complete database management system [Mata-Toledo, Adams and Norton, 2007].

When choosing or defining a lifecycle model for database systems we need to take into account properties such as: **scope restriction, progressive enhancement, incremental planning and pre-defined structure** [Wetzel and Kerschberg, 1989]. In addition, it is essential that the requirements and goals should be documented using a requirements traceability matrix (RTM) that will help in limiting the project to its envisioned scope. The database development life cycle should allow the incorporation of new user’s requirements at a later phase due to the interactive nature that should exist between the user and the developers. This would make the enhancement of a product easier and would not increase the cost significantly. For this reason incremental planning is important for database system development. Apart from the initial planning phase, individual planning is required for the design and the requirements revision phases as they highly influence the overall implementation and the evaluation of the entire system. A life cycle model lacking any of aforementioned properties (scope restriction, progressive enhancement, incremental planning and pre-defined structure) would increase the cost, time and effort to develop a DBMS.

Traditional Lifecycle Models

This section discusses the traditional lifecycle models and shows that, at least one of the properties required for database system development (scope restriction, progressive enhancement, incremental planning and pre-defined structure), is missing from each of these lifecycles. For this reason, these life cycle models are not completely suitable for developing database systems. In the remaining of this section we briefly describe some of the most popular software models and point out their deficiencies for developing DBMSs.

Waterfall model: This is the most common of all software models [Pressman, 2007]. The phases in the waterfall cycle are: **project planning, requirements definition, design, development, testing, and installation and acceptance** (See Figure 1). Each of these phases receives an input and produces an output (that serves as the input for next phase) in the form of deliverables.
The waterfall model accommodates the scope restriction and the pre-defined structure properties of the lifecycle. The requirements definition phase deals with scope restriction based on the discussions with the end user. The pre-defined structure establishes a set of standard guidelines to carry out the activities required of each phase as well as the documentation that needs to be produced. Therefore, the waterfall model, by taking into account the pre-defined structure property, helps the designers, developers, and other project participants to work in a familiar environment with fewer miscommunications while allowing completion of the project in a timely manner [Shell Method™ Process Repository, 2005].
On the other hand, the waterfall model lacks the progressive enhancement and incremental planning property. In this model, the requirements are finalized early in the cycle. In consequence, it is difficult to introduce new requirements or features at later phases of the development process [Shell Method™ Process Repository, 2005]. This waterfall model, which was derived from the "hardware world", views the software development from a manufacturing perception where items are produced once and reproduced many times [Pfleeger and Atlee, 2010]. A software development process does not work this way because the software evolves as the details of the problem are understood and discussed with the end user.
The waterfall model has a documentation driven approach which, from the user's point of view, is considered one of its main weaknesses. The system specifications, which are finalized early in the lifecycle, may be written in a non-familiar style or in a formal language that may be difficult for the end user to understand [Schach, 2008]. Generally, the end user agrees to these specifications without having a clear understanding of what the final product will be like. This leads to misunderstood or missing requirements in the **software requirements specifications** (SRS). For this reason, in general, the user has to wait until the installation phase is complete to see the overall functionality of the system. It should be obvious then that the lack of incremental planning in this model makes it difficult to use when developing a database system particularly when the latter supports, for instance, a data mining or data warehouse operations where the "impromptu" demands imposed on the system vary frequently or cannot be easily anticipated.
Draw the figure
Figure.1. Waterfall model [Pressman, 2007]

Prototype model: In this life cycle model, the developers create a prototype of the application based on a limited version of the user requirements [Pfleeger and Atlee, 2010]. The prototype consists mainly of a "hallow graphics" which shows some basic and simple functionality. However, this may create a problem because the user may view the prototype as it were the final product overlooking some of the requirements specified in the SRS which may not be met fully by this "final product" [Pfleeger and Atlee, 2010].
The prototype model limits the pre-defined structure property of a lifecycle. When a prototype is designed, the developer uses minimal code to show some requirements. During this process no integration with other tools is shown. This leads to uncertainty about the

FOR FULLNOTES
CALL/TEXT:0713440925

final product. The prototype may have to be re-designed in order to provide a finalized product and thus it may not look the same as the one shown to the user initially

Draw the figure

This lifecycle model does support the progressive enhancement property. However, since the user is only shown a prototype there may be features that the user would like to incorporate but which may too costly or time consuming to incorporate later in the project. [Shell Method™ Process Repository, 2005].

In the prototype model, the requirements are finalized early in lifecycle as shown in Figure 2. The iterations are focused on design, prototyping, customer evaluation and review phases. This model lacks the incremental planning property as there is no planning after the initial planning phase.

Spiral model: This model is a combination of the prototyping and waterfall model [Pfleeger and Atlee, 2010]. Starting with the requirements and a development plan, the system prototypes and the risks involved in their developments are analyzed through an iterative process. During each iteration alternative prototypes are considered based upon the documented constraints and risks of the previous iteration [Pfleeger and Atlee, 2010]. With each subsequent prototype the risks or constraints are minimized or eliminated. After an operational prototype has been finalized (with minimal or no risks), the detailed design document is created (See Figure 3).

The spiral model supports the scope restriction property of a lifecycle. The requirements are designed in a hierarchical pattern; any additional requirements are built on the first set of requirements implemented [Shell Method™ Process Repository, 2005]. In this model, the problem to be solved is well defined from the start. In consequence, the scope of the project is also restricted. To control risk, the spiral model combines the development activities with a risk management process [Pfleeger and Atlee, 2010]. This latter process requires expertise in the area of risk evaluation which makes the activities that need to be carried out very complex and difficult. The risk evaluation process imposes the consideration of constraints such as cost, time and effort for the entire project. The pre-defined structure property for this lifecycle model, in terms of the number of activities, is so complex that it raises the problem of controllability and efficiency during development of the system

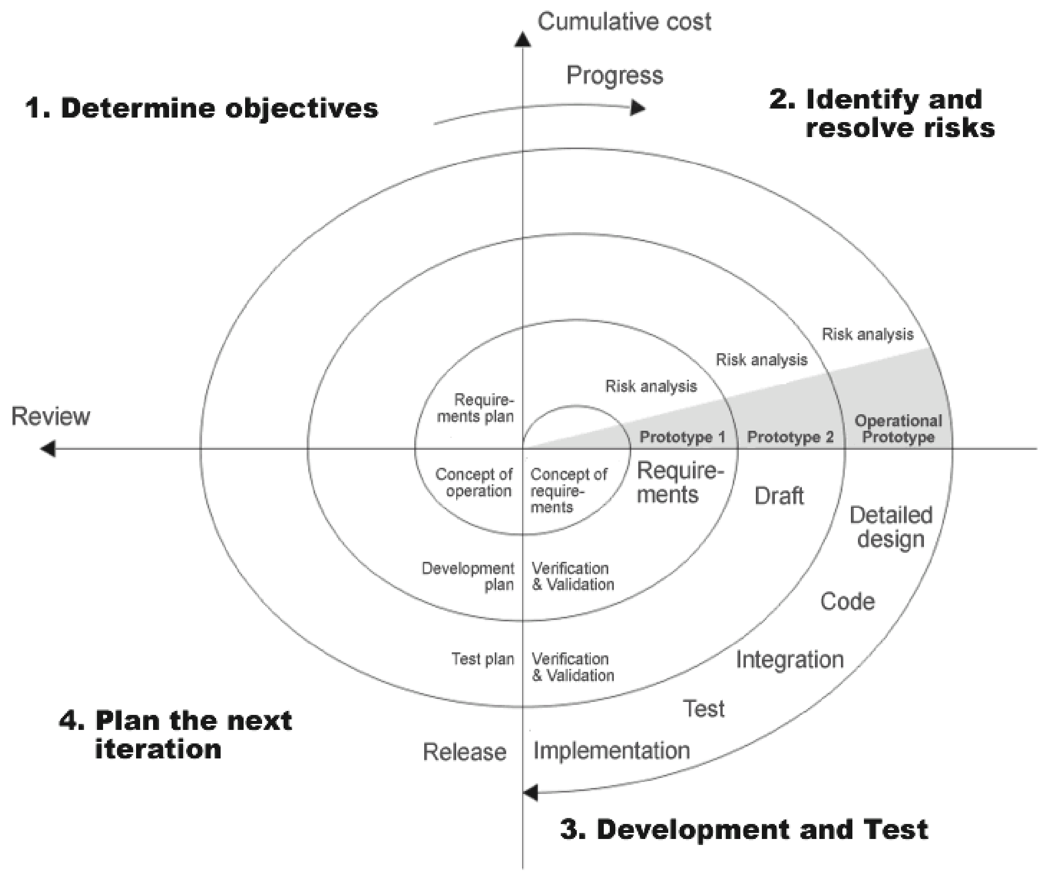


Figure.3. Spiral model [Schach, 2008]

The **progressive enhancement** property is not accommodated in this lifecycle model because, even though, the system is evolving with each phase, no new features can be added to the SRS due to the fact that the requirements have been finalized in an earlier phase.

Figure 3 shows the activities and phases of the spiral model and its iterative nature. However, notice that the **incremental planning** property is still missing from this lifecycle. The initial iterations are focused on alternatives and risks involved in the prototype selected. However, none of these iterations focus on updating the SRS by discussing it with the end user. As a result of this the requirements may not be updated; this may lead to having missing or misunderstood requirements. Due to its iterative nature this model may work well for developing requirements that are well understood from the beginning of the project. However, it is not a good model for developing database systems where new requirements may arise during the later phases of the project. The spiral model also assumes that software is developed in discrete phases; for this reason it does not satisfy the property of incremental planning [Schach, 2008].

FOR FULLNOTES
CALL/TEXT:0713440925

Rapid application development model (RAD): The basic approach of this model is to let the user try the application before it is finally delivered. The users provide feedback based upon their hands-on experience with the system.
The foremost problem with this model is that it is very easy to get caught in an unending and uncontrollable cycle of enhancements. This will lead to violations of the progressive enhancement and scope restriction property.
As the name of this model implies a prototype is created and installed as soon as possible at the user's site for their review. This model lacks the predefined structure because, in general, the rapid prototype phase is completed without strictly adhering to the guideline documents and the processes already defined to complete this phase [Schach, 2008].
As Figure 4 shows the incremental planning property of a lifecycle is missing in this model too. After the prototype is completed and evaluated by the end user the requirements may or may not change. If there are no changes in the requirements, then development of the system will continue as initially envisioned. However, if significant requirement changes are necessary, then it is imperative that a timeline for the remaining of the project be established but this is not generally done [Schach, 2008].
Figure.4. Rapid prototyping model [Schach, 2008]

Database Development Lifecycle
As we have shown in the previous paragraphs, each of the traditional lifecycle models is missing at least one of the four properties required for database system development. In this section the authors propose a new lifecycle model that is adapted from the traditional lifecycles and which is enhanced for database system development (See Figure 7). This new model satisfies properties such as **scope restriction, progressive enhancement, incremental planning and pre-defined structure.**
In most traditional life cycles, the first phase is the project planning phase. Although it is a good idea to plan the project from its inception it is also true that, unless the problem, its requirements, and its constraints are well understood it is very difficult to lay out a realistic timeline for the entire project. For this reason, we propose that this initial phase be limited to planning, not about the entire project, but about the collection of requirements definition and information about the organization. <u>In other words, we need a plan on how we are going to proceed to identify the problem as a whole, its scope, constraints, and overall functionality</u>. The resulting document is generally the project plan document.
The next phase of this model, <u>the requirement definition and organizational data collection phase,</u> should have as its ultimate goal to provide a complete set of requirements, from the user point of view, for the database system under consideration. This phase, by its very nature, requires a high degree of interaction with people at all levels of the organization, from top management to the entry level clerical workers. Essential activities of this phase are:
- Direct examination of the organizational documents as well as their dataflow through the organization and the overall operation of the latter.
-Additional information can be collected by means of interviews, questionnaires, and insitu inspection of personnel activities at all organizational levels. This phase should also produce a preliminary document of the present needs and future expansion as currently perceived by all users. Figure 5 shows the deliverables for this phase, namely, the software requirement specification (SRS) and the requirements traceability matrix (RTM). These deliverables serve as the input to the next phase<u>, the requirement analysis phase.</u>

Figure.5. Requirements definition phase

After the previous phase has been completed it is necessary to analyze the data to consider issues of extreme importance such as feasibility, cost, scope and boundaries, performance issues, security issues, portability requirements, maintenance and the design model of the expected system. This analysis of the requirements and organizational data helps to identify potential problems and constraints that could arise during development phases.

Once the aforementioned requirements and issues have been thoroughly analyzed it is necessary to envision a timeline for future work. During this timeline planning phase it is necessary to update the project plan document initially created and thus addressing the issue of incremental planning. As was indicated early incremental planning is missing in some of the traditional lifecycle models. It is the opinion of the authors that incremental planning is an essential property which needs to be satisfied throughout the entire lifecycle as indicated in Figure 5 While the database is being designed, the application design phase is carried out in parallel. The application design documents should be discussed with the user and changes should be made to the RTM if needed. The design phase is followed by the database implementation and loading phase. The database is implemented using the physical design documents developed earlier during the design phase. The database implementation and loading phase includes steps such as the follows: creating database tables, populating the tables, building constraints and querying the data.

FOR FULLNOTES
CALL/TEXT:0713440925

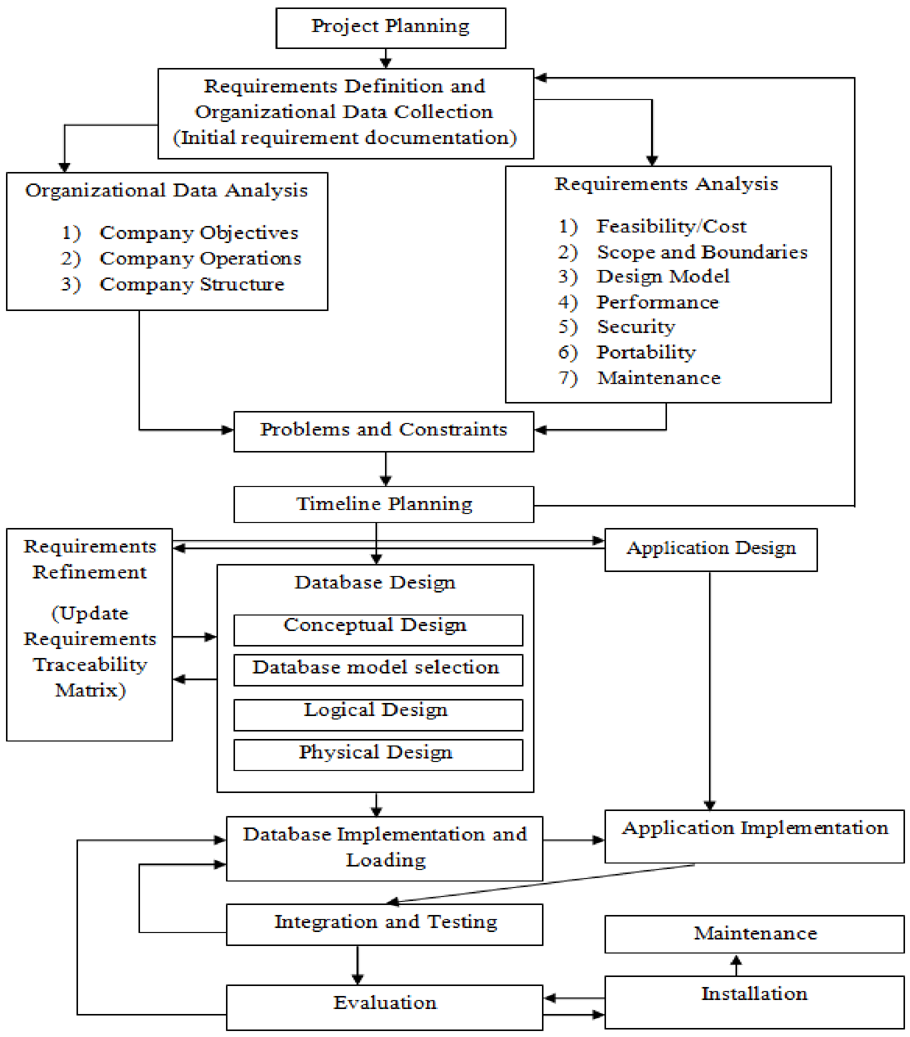


Figure.7.Database development life cycle

FOR FULLNOTES
CALL/TEXT:0713440925

Next follows the application implementation phase. The application design documents from the application design phase serve as an input to this phase. The database is then integrated with the application(s) in the next phase i.e. the integration and testing phase. The integrated system is tested in this phase.
Finally we have the installation/evaluation phase. Here the user tries out the product and appraises its functionality and performance. After the system has been accepted by the user and it is operational, the maintenance phase begins. This maintenance phase will continue until the product has reached the end of its useful life. That is, until it no longer meets the new requirements of the user. At this point the whole process of developing a new system starts anew.

Conclusion

A complete and correct database system is difficult to create if the SDLC does not take into account the intrinsic characteristics of the system to be developed and the SDLC itself does not accommodate properties like scope restriction, progressive enhancement, incremental planning and pre-defined structure. As indicated before, traditional SDLCs lack at least one of the aforementioned properties making them not all suitable for the development of DBMSs, particularly, when the demands on the DBMS are unpredictable. One of main characteristics of this new proposed model is that it makes emphasis on activities that go back and forth between phases allowing either the incorporation of new requirements, if needed, or the correction of incomplete or misunderstood requirements. The idea is to allow for a system that is more flexible of the realities of developing a DBMS.

7.4.1 Data and user requirement specification

The software development is the group of actions needed to transform the user's need into an effectual software solution. Software development procedure consist the activities needed for building the software systems and integrating the techniques and practices to be accepted. It also includes the planning of project, tracking development and managing the complications of building software.

This different database related activities can be grouped into below phases (more commonly known as DDLC – Database Development Life Cycle):

Requirements Analysis, Database Design, Evaluation and Selection Logical Database Design, Physical Database Design, Implementation, Data Loading, Testing and Performance Tuning, Operation Maintenance

7.4.2 Stages of database development

Requirements Analysis

The most important step in implementing a database system is to find out what is needed. What type of a database is required for the business organization, daily volume of the data, how much data needs to be stored in the master files etc. In order to collect all this required

THIS IS A FREE SAMPLE OF THE ACTUAL NOTES

TO GET FULL/COMPLETE NOTES:

Call/Text/Whatsapp:0713440925

You can also write to us at: topexamskenya@gmail.com
or info@mykasnebnotes.com

To download more resources visit: http://www.mykasnebnotes.com

For updates and insights like us on Facebook

STUDY NOTES|REVISION KITS|PILOT PAPERS|COURSE OUTLINE|STUDY TIPS|

CPA CIFA CS ATD CIPS CCP DICT DCM