

PAPER NO. CT 31

SECTION 3

**CERTIFIED
INFORMATION COMMUNICATION
TECHNOLOGISTS
(CICT)**

DATA BASE SYSTEMS

STUDY TEXT

KASNEB SYLLABUS

GENERAL OBJECTIVE

This paper is intended to equip the candidate with the knowledge, skills and attitude that will enable him/her to administer and manage databases

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

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

2. File organisation techniques

- Storage structures and blocking
- Unordered files
- Sequential files
- Indexing

3. Database models

- The role of data modeling
- The hierarchical model
- The relational model
- The object-oriented model
- The object-relational model

4. Database development life cycle

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

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 /DB2

6. Structured query language (SQL)

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

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

8. Database administration

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

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

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

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

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

13. Emerging issues and trends

CONTENT	PAGE
Chapter 1: Introduction to databases.....	5
Chapter 2: File organisation techniques.....	14
Chapter 3: Database models	20
Chapter 4: Database development life cycle.....	31
Chapter 5: Relational database model.....	41
Chapter 6: Structured query language (SQL).....	66
Chapter 7: Transaction management and concurrency control.....	76
Chapter 8: Database administration.....	93
Chapter 9: Database security and integrity.....	97
Chapter 10: Distributed database systems.....	103
Chapter 11: Data warehousing and data mining.....	113
Chapter 12: Integrating databases to other applications.....	121
Chapter 13: Emerging issues and trends.....	132

CHAPTER 1

INTRODUCTIONS TO DATABASES

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.

History of Database Systems

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 someway related Hierarchically begin with a strictly defined tree of data nodes Each node can contain some identifying data, plus a set of subnodes 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 decided 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 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

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 2

FILE ORGANISATION TECHNIQUES

DATA STRUCTURES AND BLOCKING

In computing (specifically data transmission and data storage), a **block**, sometimes called a **physical record**, is a sequence of bytes or bits, usually containing some whole number of records, having a maximum length, a *block size*. Data thus structured are said to be *blocked*. The process of putting data into blocks is called *blocking*, while *deblocking* is the process of extracting data from blocks. Blocked data is normally stored in a data buffer and read or written a whole block at a time. Blocking reduces the overhead and speeds up the handling of the data-stream. For some devices such as magnetic tape and CKD disk devices blocking reduces the amount of external storage required for the data. Blocking is almost universally employed when storing data to 9-track magnetic tape, to NAND flash memory, and to rotating media such as floppy disks, hard disks, and optical discs.

Most file systems are based on a block device, which is a level of abstraction for the hardware responsible for storing and retrieving specified blocks of data, though the block size in file systems may be a multiple of the physical block size. This leads to space inefficiency due to internal fragmentation, since file lengths are often not integer multiples of block size, and thus the last block of a file may remain partially empty. This will create slack space, which averages half a block per file. Some newer file systems attempt to solve this through techniques called block suballocation and tail merging.

Block storage is normally abstracted by a file system or database management system (DBMS) for use by applications and end users. The physical or logical volumes accessed via *block I/O* may be devices internal to a server, directly attached via SCSI or Fibre Channel, or distant devices accessed via a storage area network (SAN) using a protocol such as iSCSI, or AoE. DBMSes often use their own block I/O for improved performance and recoverability as compared to layering the DBMS on top of a file system.

Data processing from a computer science perspective:

- Storage of data
- Organization of data
- Access to data

This will be built on your knowledge of Data Structures

Data Structure VS. File Structure

Both involve:

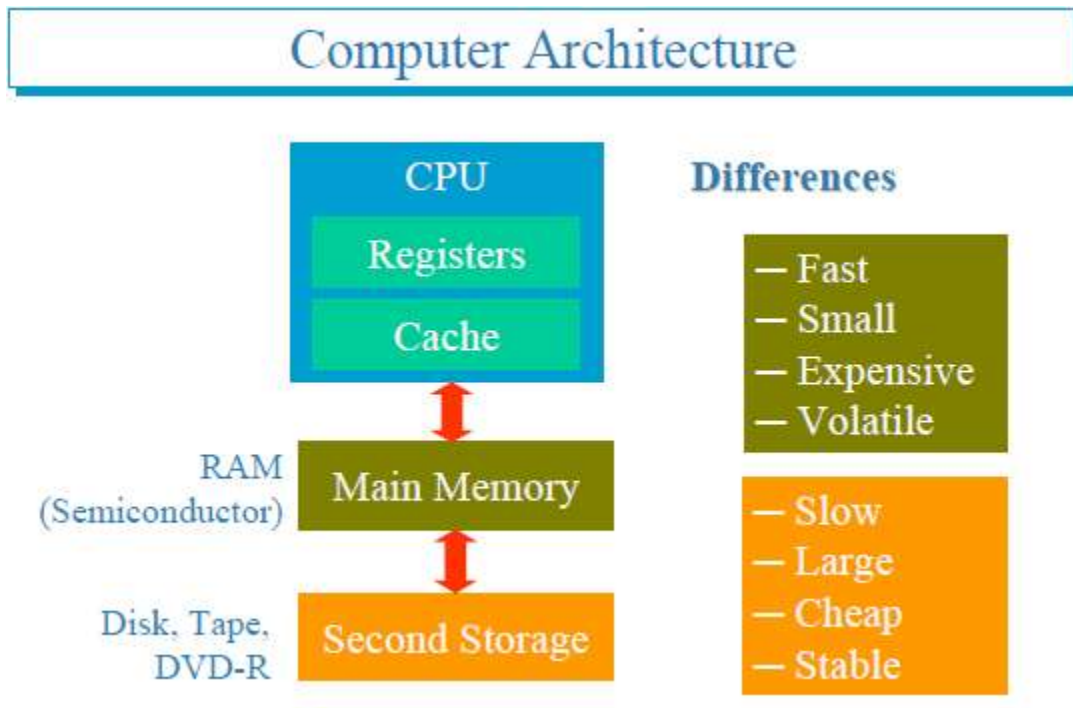
Representation of Data

+

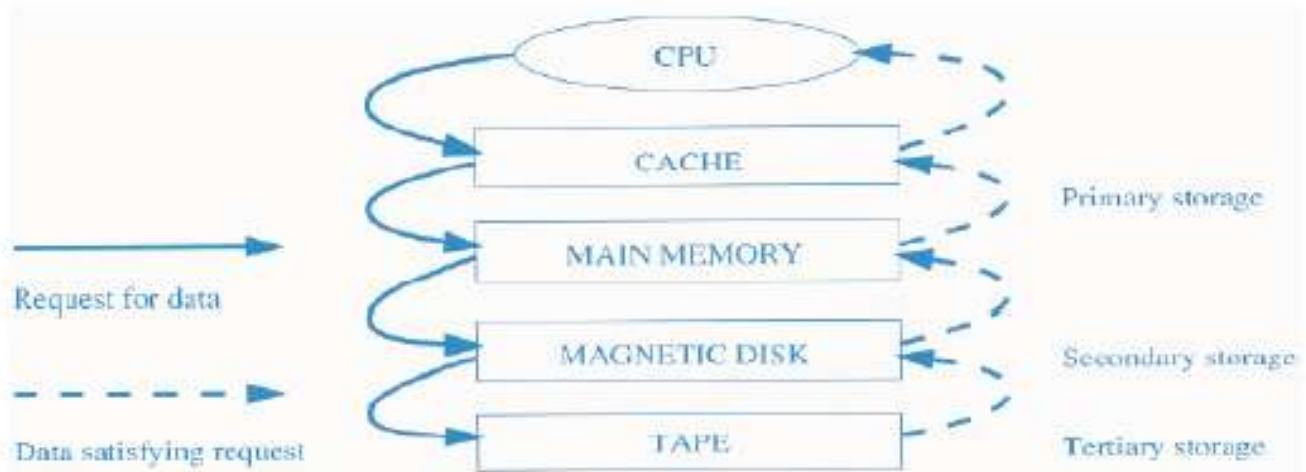
Operations for accessing data

Difference:

- Data Structures deal with data in main memory
- File Structures deal with data in secondary storage device (File).



Memory Hierarchy



Data must be maintained across program executions. This requires storage devices that retain information when the computer is restarted.

- We call such storage nonvolatile.
- Primary storage is usually volatile, whereas secondary and tertiary storage are nonvolatile.

File organization refers to the relationship of the key of the record to the physical location of that record in the computer file.

File organization may be either physical file or a logical file. A physical file is a physical unit, such as magnetic tape or a disk.

A logical file on the other hand is a complete set of records for a specific application or purpose.

A logical file may occupy a part of physical file or may extend over more than one physical file.

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 3

DATABASE MODELS

Introduction

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
- Multivalued model
- Semantic model
- XML database
- Named graph

THE ROLE OF DATA MODELING

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

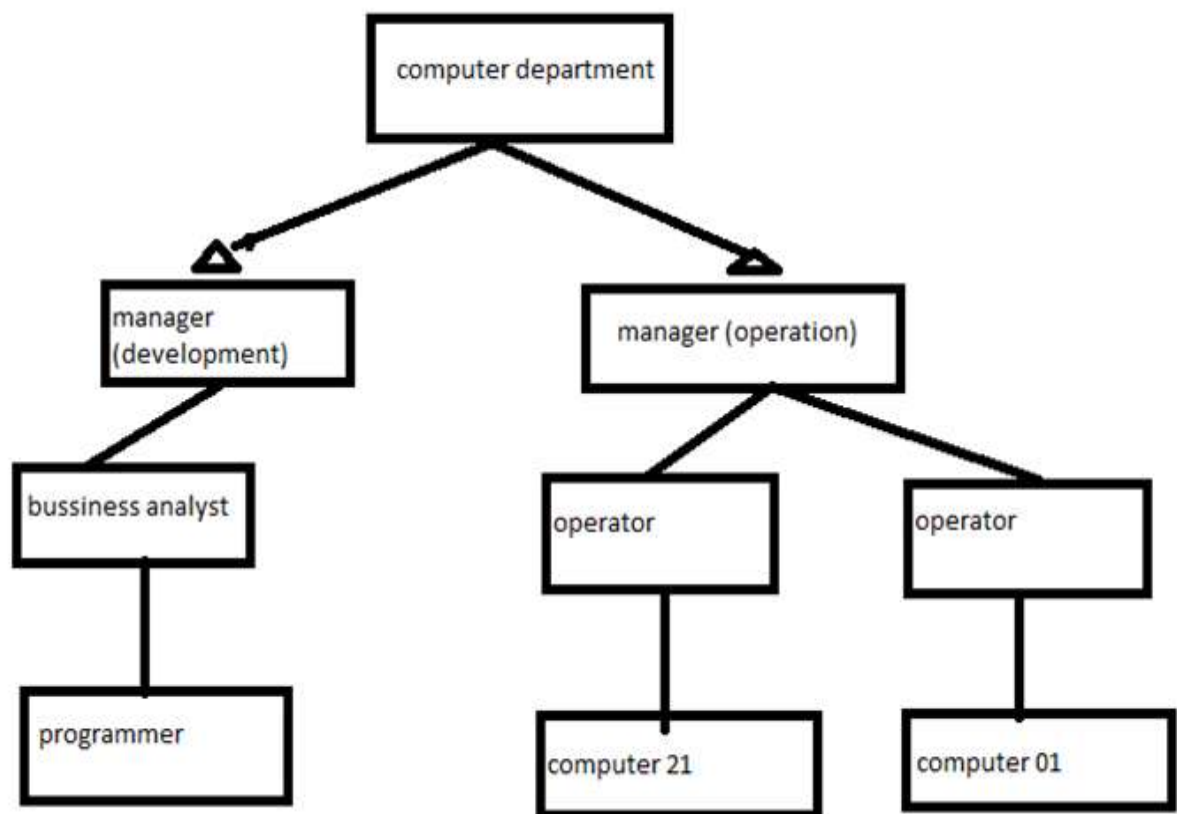
A flat file database is a database that stores data in a plain text file. Each line of the text file holds one record, with fields separated by delimiters, such as commas or tabs. While it uses a simple structure, a flat file database cannot contain multiple tables like a relational database can. Fortunately, 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

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 segments 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.



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.

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 4

DATABASE DEVELOPMENT LIFE CYCLES

Introduction: Database Development Life Cycle

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 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 Kerschberg, 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 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

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 5

RELATIONAL DATABASE MODEL

A relational database is based on the relational model developed by E.F. Codd. A relational database allows **the definition of data structures, storage and retrieval operations and integrity constraints**. In such a database the data and relations between them are organized into tables. A table is a collection of records and each record in a table contains the same fields. The contents of a table can be permanently saved for future use.

RELATIONAL DATABASE CONCEPTS AND PROPERTIES

Properties of the relational database model

Properties of Relational Tables:

1. Data is presented as a collection of relations.
2. Each relation is depicted as a table.
3. Columns are attributes that belong to the entity modeled by the table (ex. In a student table, you could have name, address, student ID, major, etc.).
4. Each row ("tuple") represents a single entity (ex. In a student table, John Smith, 14 Oak St, 9002342, Accounting, would represent one student entity).
5. Every table has a set of attributes that taken together as a "key" (technically, a "superkey") uniquely identifies each entity (Ex. In the student table, "student ID" would uniquely identify each student – no two students would have the same student ID).

Overview

Certain fields may be designated as keys, which means that searches for specific values of that field will use indexing to speed them up and more importantly, uniquely identify each entity. There are many types of keys, however, quite possibly the two most important are the primary key and the foreign key. The primary key is what uniquely identifies each entity. The foreign key is a primary key of one table that also sits in another table. Ultimately, the use of foreign keys is the heart of the relational database model. This linkage that the foreign key provides is what allows tables to pull data from each other and link data together. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Most often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID – primary key, product-code – foreign key) pairs and a "products" table might contain (product-code – primary key, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retrieval time, relational databases are classed as dynamic database management system. Also, another important role of the primary key is

called “determination”. What this means is that if you know the value of attribute X, you can determine the value of attribute Y. The relational database model is based on the Relational Algebra. Which means that operations in the relational database model are based on Select, Project, Join, Intersect, Union, Difference, and Product. Here is a brief description of each operation:

- Select: Shows values for all rows found a table that meet a given criteria.
- Project: Shows values for all selected attributes.
- Join: Will combine information from one or more tables.
- Intersect: Shows all rows that are found in both tables.
- Union: Combines all rows from multiple tables and removes the duplicate rows.
- Difference: Shows all rows from one table that are not contained in another table.
- Product: Combines all rows from two or more tables (does contain duplicates).

As you can see, this is a very powerful set of operations that can be used to manipulate data.

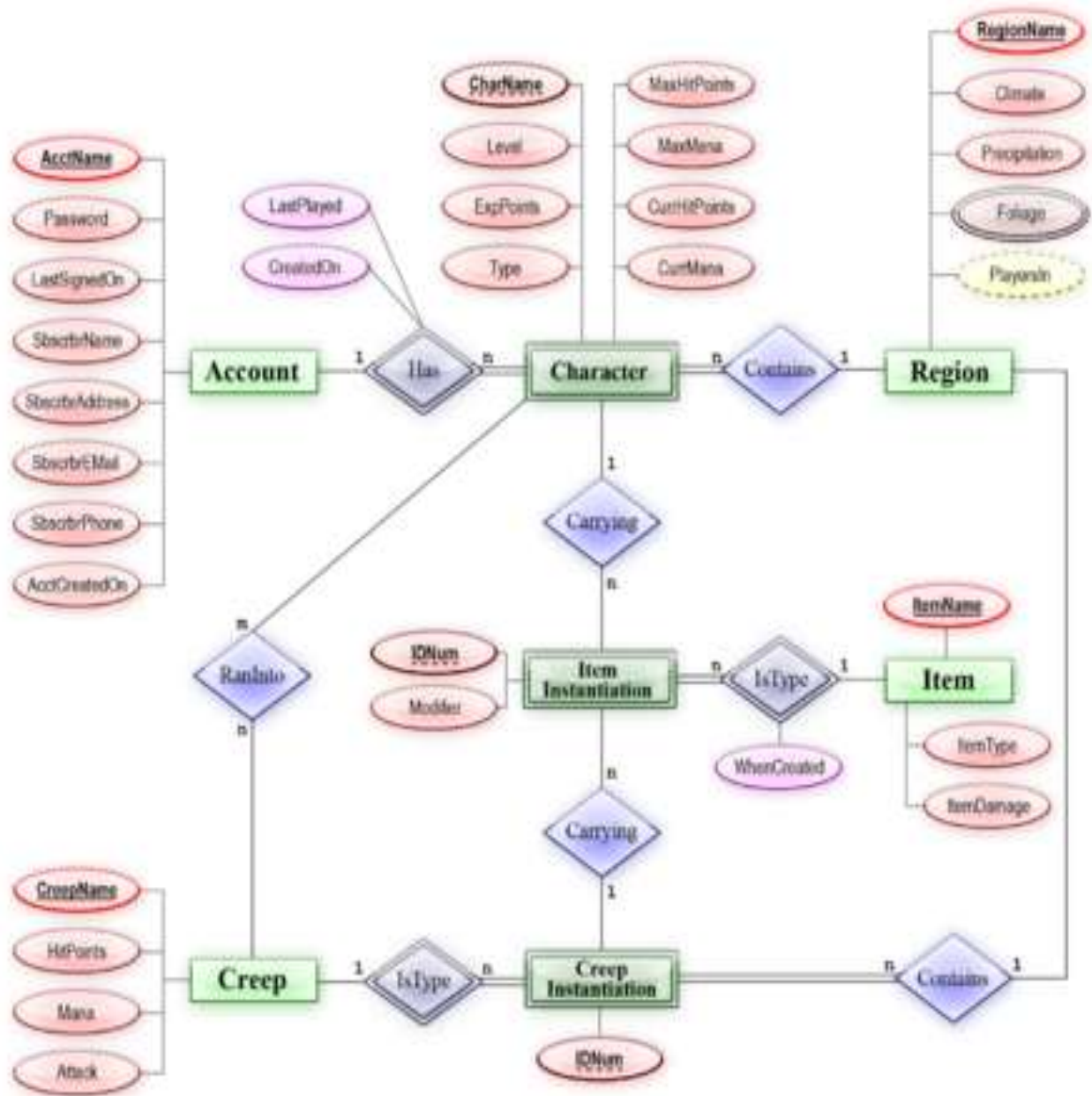
Rules

In the relational database model, there are five, very important rules. When followed, these rules help to ensure data integrity.

1. The order of tuples and attributes is not important. (Ex. Attribute order not important...if you have name before address, is the same as address before name).
2. Every tuple is unique. This means that for every record in a table there is something that uniquely identifies it from any other tuple.
3. Cells contain single values. This means that each cell in a table can contain only one value.
4. All values within an attribute are from the same domain. This means that however the attribute is defined, the values for each tuple fall into that definition. For example, if the attribute is labeled as Date, you would not enter a dollar amount, shirt size, or model number in that column, only dates.
5. Table names in the database must be unique and attribute names in tables must be unique. No two tables can have the same name in a database. Attributes (columns) cannot have the same name in a table. You can have two different tables that have similar attribute names.

Databases are very commonly used in everyday life. The relational model of databases provides a very simple way of looking at data structured into tables, and there are straightforward techniques, such as ER modeling to represent a world view from which to build a relational database.

E-R data base design



An entity-relationship diagram using Chen's notation

In software engineering, an **entity-relationship model (ER model)** is a data model for describing the data or information aspects of a business domain or its process requirements, in an

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 6

STRUCTURED QUERY LANGUAGE (SQL)

Database languages are special-purpose languages, which do one or more of the following:

- Data definition language – defines data types and the relationships among them
- Data manipulation language – performs tasks such as inserting, updating, or deleting data occurrences
- Query language – allows searching for information and computing derived information

Database languages are specific to a particular data model. Notable examples include:

- SQL combines the roles of data definition, data manipulation, and query in a single language. It was one of the first commercial languages for the relational model, although it departs in some respects from the relational model as described by Codd (for example, the rows and columns of a table can be ordered). SQL became a standard of the American National Standards Institute (ANSI) in 1986 and of the International Organization for Standardization (ISO) in 1987. The standards have been regularly enhanced since and is supported (with varying degrees of conformance) by all mainstream commercial relational DBMSs.
- OQL is an object model language standard (from the Object Data Management Group). It has influenced the design of some of the newer query languages like JDOQL and EJB QL.
- XQuery is a standard XML query language implemented by XML database systems such as MarkLogic and eXist, by relational databases with XML capability such as Oracle and DB2, and also by in-memory XML processors such as Saxon.
- SQL/XML combines XQuery with SQL.

A database language may also incorporate features like:

- DBMS-specific Configuration and storage engine management
- Computations to modify query results, like counting, summing, averaging, sorting, grouping, and cross-referencing
- Constraint enforcement (e.g. in an automotive database, only allowing one engine type per car)
- Application programming interface version of the query language, for programmer convenience

DATA MANIPULATION LANGUAGE

A **data manipulation language (DML)** is a family of syntax elements similar to a computer programming language used for inserting, deleting and updating data in a database. Performing read-only queries of data is sometimes also considered a component of DML.

A popular data manipulation language is that of Structured Query Language (SQL), which is used to retrieve and manipulate data in a relational database. Other forms of DML are those used by IMS/DLI, CODASYL databases, such as IDMS and others.

Data manipulation language comprises the SQL data change statements, which modify stored data but not the schema or database objects. Manipulation of persistent database objects, e.g., tables or stored procedures, via the SQL schema statements, rather than the data stored within them, is considered to be part of a separate data definition language. In SQL these two categories are similar in their detailed syntax, data types, expressions etc., but distinct in their overall function.

Data manipulation languages have their functional capability organized by the initial word in a statement, which is almost always a verb. In the case of SQL, these verbs are:

- SELECT ... FROM ... WHERE ...
- INSERT INTO ... VALUES ...
- UPDATE ... SET ... WHERE ...
- DELETE FROM ... WHERE ...

The purely read-only SELECT query statement is classed with the 'SQL-data' statements and so is considered by the standard to be outside of DML. The SELECT ... INTO form is considered to be DML because it manipulates (i.e. modifies) data. In common practice though, this distinction is not made and SELECT is widely considered to be part of DML.

Most SQL database implementations extend their SQL capabilities by providing imperative, i.e. procedural languages. Examples of these are Oracle's PL/SQL and DB2's SQL PL.

Data manipulation languages tend to have many different flavors and capabilities between database vendors. There have been a number of standards established for SQL by ANSI, but vendors still provide their own extensions to the standard while not implementing the entire standard.

Data manipulation languages are divided into two types, procedural programming and declarative programming.

Data manipulation languages were initially only used within computer programs, but with the advent of SQL have come to be used interactively by database administrators.

STRUCTURE OF SQL STATEMENT

Basic Structure

1. Basic structure of an SQL expression consists of **select**, **from** and **where** clauses.
 - o **Select** clause lists attribute to be copied - corresponds to relational algebra **project**.
 - o **From** clause corresponds to Cartesian product - lists relations to be used.
 - o **Where** clause corresponds to selection predicate in relational algebra.
2. Typical query has the form

select A_1, A_2, \dots, A_n

from r_1, r_2, \dots, r_m

where P

Where each A_i represents an attribute, each r_i a relation and P is a predicate.

3. This is equivalent to the relational algebra expression

$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_P(r_1 \times r_2 \times \dots \times r_m))$$

- o If the where clause is omitted, the predicate P is true.
- o The list of attributes can be replaced with a * to select all.
- o SQL forms the Cartesian product of the relations named, performs a selection using the predicate, then projects the result onto the attributes named.
- o The result of an SQL query is a relation.
- o SQL may internally convert into more efficient expressions.

SQL Queries	
SQL (pronounced "ess-que-el") is the acronym for Structured Query Language. SQL is the standard language for communicating with relational database management systems. SQL statements are used to retrieve data from the database as well as perform tasks such as adding updating and deleting the data. Some common relational database management systems that use SQL are: Oracle, Sybase, MS SQL Server, MS Access, MySQL, etc.	
	Basic Query Structure
	SELECT field1 [,"field2", etc.] FROM table [WHERE "condition"] [GROUP BY "field"] [ORDER BY "field"]

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 7

TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL

TRANSACTIONAL MANAGEMENT

A **transaction** is one or more SQL statements that make up a unit of work performed against the database, and either all the statements in a transaction are committed as a unit or all the statements are rolled back as a unit. This unit of work typically satisfies a user request and ensures data integrity. For example, when you use a computer to transfer money from one bank account to another, the request involves a transaction: updating values stored in the database for both accounts. For a transaction to be completed and database changes to be made permanent, a transaction must be completed in its entirety.

What is the correct transaction commit mode to use in your application? What is the right transaction model for your database application: local or distributed? Use the guidelines in this section to help you manage transactions more efficiently.

You should also read the chapter for the standards-based API that you work with; these chapters provide specific examples for each API:

Managing Commits in Transactions

Committing (and rolling back) transactions is slow because of the disk I/O and potentially the number of network round trips required. What does a commit actually involve? The database must write to disk every modification made by a transaction to the database. This is usually a sequential write to a journal file (or log); nevertheless, it involves expensive disk I/O.

In most standards-based APIs, the default transaction commit mode is autocommit. In auto-commit mode, a commit is performed for every SQL statement that requires a request to the database, such as Insert, Update, Delete, and Select statements. When auto-commit mode is used, the application does not control when database work is committed. In fact, commits commonly occur when there's actually no real work to commit.

Some database systems, such as DB2, do not support auto-commit mode. For these databases, the database driver, by default, sends a commit request to the database after every successful operation (SQL statement). This request equates to a network round trip between the driver and the database. The round trip to the database occurs even though the application did not request the commit and even if the operation made no changes to the database. For example, the driver makes a network round trip even when a Select statement is executed.

Because of the significant amount of disk I/O required to commit every operation on the database server and because of the extra network round trips that occur between the driver and

the database, in most cases you will want to turn off auto-commit mode in your application. By doing this, your application can control when the database work is committed, which provides dramatically better performance.

Performance Tip

Although turning off auto-commit mode can help application performance, do not take this tip too far. Leaving transactions active can reduce throughput by holding locks on rows for longer than necessary, preventing other users from accessing the rows. Typically, committing transactions in intervals provides the best performance as well as acceptable concurrency

Consider the following real-world example. ASoft Corporation coded a standards-based database application and experienced poor performance in testing. Its performance analysis showed that the problem resided in the bulk five million Insert statements sent to the database. With auto-commit mode on, this meant an additional five million Commit statements were being issued across the network and that every inserted row was written to disk immediately following the execution of the Insert. When auto-commit mode was turned off in the application, the number of statements issued by the driver and executed on the database server was reduced from ten million (five million Inserts + five million Commits) to five million and one (five million Inserts + one Commit). As a consequence, application processing was reduced from eight hours to ten minutes. Why such a dramatic difference in time? There was significantly less disk I/O required by the database server, and there were 50% fewer network round trips.

If you have turned off auto-commit mode and are using manual commits, when does it make sense to commit work? It depends on the following factors:

- The type of transactions your application performs. For example, does your application perform transactions that modify or read data? If your application modifies data, does it update large amounts of data?
- How often your application performs transactions.

For most applications, it's best to commit a transaction after every logical unit of work. For example, consider a banking application that allows users to transfer money from one account to another. To protect the data integrity of that work, it makes sense to commit the transaction after both accounts are updated with the new amounts.

However, what if an application allows users to generate reports of account balances for each day over a period of months? The unit of work is a series of Select statements, one executed after the other to return a column of balances. In most cases, for every Select statement executed against the database, a lock is placed on rows to prevent another user from updating that data. By holding locks on rows for longer than necessary, active transactions can prevent other users from updating data, which ultimately can reduce throughput and cause concurrency issues. In this case, you may want to commit the Select statements in intervals (after every five Select statements, for example) so that locks are released in a timely manner.

In addition, be aware that leaving transactions active consumes database memory. Remember that the database must write every modification made by a transaction to a log that is stored in database memory. Committing a transaction flushes the contents of the log and releases database memory. If your application uses transactions that update large amounts of data (1,000 rows, for example) without committing modifications, the application can consume a substantial amount of database memory. In this case, you may want to commit after every statement that updates a large amount of data.

How often your application performs transactions also determines when you should commit them. For example, if your application performs only three transactions over the course of a day, commit after every transaction. In contrast, if your application constantly performs transactions that are composed of Select statements, you may want to commit after every five Select statements.

Isolation Levels.

We will not go into the details of isolation levels in this book, but architects should know the default transaction isolation level of the database system they are using. A **transaction isolation level** represents a particular locking strategy used in the database system to improve data integrity.

Most database systems support several isolation levels, and the standards-based APIs provide ways for you to set isolation levels. However, if the database driver you are using does not support the isolation level you set in your application, the setting has no effect. Make sure you choose a driver that gives you the level of data integrity that you need.

Local Transactions Versus Distributed Transactions A **local transaction** is a transaction that accesses and updates data on only one database. Local transactions are significantly faster than distributed transactions because local transactions do not require communication between multiple databases, which means less logging and fewer network round trips are required to perform local transactions.

Use local transactions when your application does *not* have to access or update data on multiple networked databases.

A **distributed transaction** is a transaction that accesses and updates data on multiple networked databases or systems and must be coordinated among those databases or systems. These databases may be of several types located on a single server, such as Oracle, Microsoft SQL Server, and Sybase; or they may include several instances of a single type of database residing on numerous servers.

The main reason to use distributed transactions is when you need to make sure that databases stay consistent with one another. For example, suppose a catalog company has a central database that stores inventory for all its distribution centers. In addition, the company has a database for its east coast distribution center and one for the west coast. When a catalog order is placed, an application updates the central database and updates either the east or west coast database. The application performs both operations in one distributed transaction to ensure that the information

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 8

DATABASE ADMINISTRATION

Database users

Types of Database Users:

Users are differentiated by the way they expect to interact with the system:

1. · **Application programmers** - interact with system through DML calls.
2. · **Sophisticated users** - form requests in a database query language.
3. · **Specialized users** - write specialized database applications that do not fit into the traditional data processing framework.
4. · **Naïve users** - invoke one of the permanent application programs that have been written previously

DATABASE ADMINISTRATION

Database administration is the function of managing and maintaining database management systems (DBMS) software. Mainstream DBMS software such as Oracle, IBM DB2 and Microsoft SQL Server need ongoing management. As such, corporations that use DBMS software often hire specialized IT (Information Technology) personnel called Database Administrators or DBAs.

FUNCTIONS AND ROLES OF DATABASE ADMINISTRATOR

Database Administrator Roles and Responsibilities: A Database Administrator, Database Analyst or Database Developer is the person responsible for managing the information within an organization. As most companies continue to experience inevitable growth of their databases, these positions are probably the most solid within the IT industry. In most cases, it is not an area that is targeted for layoffs or downsizing. On the downside, however, most database departments are often understaffed, requiring administrators to perform a multitude of tasks.

Depending on the company and the department, this role can either be highly specialized or incredibly diverse. The primary role of the Database Administrator is to administer, develop, maintain and implement the policies and procedures necessary to ensure the security and integrity of the corporate database. Sub roles within the Database Administrator classification may include security, architecture, and warehousing and/or business analysis.

DBA Responsibilities

- Installation, configuration and upgrading of Database server software and related products.
- Evaluate Database features and Database related products.
- Establish and maintain sound backup and recovery policies and procedures.
- Take care of the Database design and implementation.
- Implement and maintain database security (create and maintain users and roles, assign privileges).
- Database tuning and performance monitoring.
- Application tuning and performance monitoring.
- Setup and maintain documentation and standards.
- Plan growth and changes (capacity planning).
- Work as part of a team and provide 24x7 support when required.
- Do general technical troubleshooting and give cons.
- Database recovery.

Types of database administration

There are three types of DBAs:

1. Systems DBAs (also referred to as Physical DBAs, Operations DBAs or Production Support DBAs): focus on the physical aspects of database administration such as DBMS installation, configuration, patching, upgrades, backups, restores, refreshes, performance optimization, maintenance and disaster recovery.
2. Development DBAs: focus on the logical and development aspects of database administration such as data model design and maintenance, DDL (data definition language) generation, SQL writing and tuning, coding stored procedures, collaborating with developers to help choose the most appropriate DBMS feature/functionality and other pre-production activities.
3. Application DBAs: usually found in organizations that have purchased 3rd party application software such as ERP (enterprise resource planning) and CRM (customer relationship management) systems. Examples of such application software includes Oracle Applications, Siebel and PeopleSoft (both now part of Oracle Corp.) and SAP. Application DBAs straddle the fence between the DBMS and the application software and are responsible for ensuring that the application is fully optimized for the database and vice versa. They usually manage all the application components that interact with the database and carry out activities such as application installation and patching, application upgrades, database cloning, building and running data cleanup routines, data load process management, etc.

While individuals usually specialize in one type of database administration, in smaller organizations, it is not uncommon to find a single individual or group performing more than one type of database administration.

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 9

DATABASE SECURITY AND INTEGRITY

Database security deals with all various aspects of protecting the database content, its owners, and its users. It ranges from protection from intentional unauthorized database uses to unintentional database accesses by unauthorized entities (e.g., a person or a computer program).

Database access control deals with controlling who (a person or a certain computer program) is allowed to access what information in the database. The information may comprise specific database objects (e.g., record types, specific records, data structures), certain computations over certain objects (e.g., query types, or specific queries), or utilizing specific access paths to the former (e.g., using specific indexes or other data structures to access information). Database access controls are set by special authorized (by the database owner) personnel that use dedicated protected security DBMS interfaces.

This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called "subschemas". For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data. If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases.

Data security in general deals with protecting specific chunks of data, both physically (i.e., from corruption, or destruction, or removal; e.g., see physical security), or the interpretation of them, or parts of them to meaningful information (e.g., by looking at the strings of bits that they comprise, concluding specific valid credit-card numbers; e.g., see data encryption).

Change and access logging records that accessed which attributes, what was changed, and when it was changed. Logging services allow for a forensic database audit later by keeping a record of access occurrences and changes. Sometimes application-level code is used to record changes rather than leaving this to the database. Monitoring can be set up to attempt to detect security breaches.

SECURITY AND INTEGRITY CONCEPT

Database security

Database security concerns the use of a broad range of information security controls to protect databases (potentially including the data, the database applications or stored functions, the database systems, the database servers and the associated network links) against compromises of their confidentiality, integrity and availability. It involves various types or categories of controls, such as technical, procedural/administrative and physical. *Database security* is a specialist topic within the broader realms of computer security, information security and risk management.

Security risks to database systems include, for example:

- Unauthorized or unintended activity or misuse by authorized database users, database administrators, or network/systems managers, or by unauthorized users or hackers (e.g. inappropriate access to sensitive data, metadata or functions within databases, or inappropriate changes to the database programs, structures or security configurations);
- Malware infections causing incidents such as unauthorized access, leakage or disclosure of personal or proprietary data, deletion of or damage to the data or programs, interruption or denial of authorized access to the database, attacks on other systems and the unanticipated failure of database services;
- Overloads, performance constraints and capacity issues resulting in the inability of authorized users to use databases as intended;
- Physical damage to database servers caused by computer room fires or floods, overheating, lightning, accidental liquid spills, static discharge, electronic breakdowns/equipment failures and obsolescence;
- Design flaws and programming bugs in databases and the associated programs and systems, creating various security vulnerabilities (e.g. unauthorized privilege escalation), data loss/corruption, performance degradation etc.;
- Data corruption and/or loss caused by the entry of invalid data or commands, mistakes in database or system administration processes, sabotage/criminal damage etc.

Many layers and types of information security control are appropriate to databases, including:

- Access control
- Auditing
- Authentication
- Encryption
- Integrity controls
- Backups
- Application security
- Database Security applying Statistical Method

Traditionally databases have been largely secured against hackers through network security measures such as firewalls, and network-based intrusion detection systems. While network security controls remain valuable in this regard, securing the database systems themselves, and

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 10

DISTRIBUTED DATABASE SYSTEMS

CONCEPTS OF DISTRIBUTED DATABASES

A **distributed database** is a database in which storage devices are not all attached to a common processing unit such as the CPU, controlled by a distributed database management system (together sometimes called a distributed database system). It may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely-coupled sites that share no physical components.

System administrators can distribute collections of data (e.g. in a database) across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. Because they store data across multiple computers, distributed databases can improve performance at end-user worksites by allowing transactions to be processed on many machines, instead of being limited to one.

DISTRIBUTION METHODS –FRAGMENTATION AND REPLICATION

There are two principal approaches to store a relation r in a distributed database system:

- A) Replication
- B) Fragmentation/Partitioning

A) Replication: In replication, the system maintains several identical replicas of the same relation r in different sites.

- Data is more available in this scheme.
- Parallelism is increased when read request is served.
- Increases overhead on update operations as each site containing the replica needed to be updated in order to maintain consistency.
- Multi-datacenter replication provides geographical diversity, like in Clusterpoint or Riak.

B) Fragmentation: The relation r is fragmented into several relations $r_1, r_2, r_3, \dots, r_n$ in such a way that the actual relation could be reconstructed from the fragments and then the fragments are scattered to different locations. There are basically two schemes of fragmentation:

- Horizontal fragmentation - splits the relation by assigning each tuple of r to one or more fragments.
- Vertical fragmentation - splits the relation by decomposing the schema R of relation

CONCURRENCY CONTROL MECHANISMS IN DISTRIBUTED SYSTEMS

In information technology and computer science, especially in the fields of computer programming, operating systems, multiprocessors, and databases, **concurrency control** ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

Computer systems, both software and hardware, consist of modules, or components. Each component is designed to operate correctly, i.e., to obey or to meet certain consistency rules. When components that operate concurrently interact by messaging or by sharing accessed data (in memory or storage), a certain component's consistency may be violated by another component. The general area of concurrency control provides rules, methods, design methodologies, and theories to maintain the consistency of components operating concurrently while interacting, and thus the consistency and correctness of the whole system. Introducing concurrency control into a system means applying operation constraints which typically result in some performance reduction. Operation consistency and correctness should be achieved with as good as possible efficiency, without reducing performance below reasonable levels. Concurrency control can require significant additional complexity and overhead in a concurrent algorithm compared to the simpler sequential algorithm.

For example, a failure in concurrency control can result in data corruption from torn read or write operations.

Concurrency control in databases

Comments:

1. This section is applicable to all transactional systems, i.e., to all systems that use *database transactions (atomic transactions)*; e.g., transactional objects in Systems management and in networks of smartphones which typically implement private, dedicated database systems), not only general-purpose database management systems (DBMSs).
2. DBMSs need to deal also with concurrency control issues not typical just to database transactions but rather to operating systems in general. These issues (e.g., see *Concurrency control in operating systems* below) are out of the scope of this section.

Concurrency control in Database management systems (DBMS; e.g., Bernstein et al. 1987, Weikum and Vossen 2001), other transactional objects, and related distributed applications (e.g., Grid computing and Cloud computing) ensures that *database transactions* are performed concurrently without violating the data integrity of the respective databases. Thus concurrency control is an essential element for correctness in any system where two database transactions or

more, executed with time overlap, can access the same data, e.g., virtually in any general-purpose database system. Consequently, a vast body of related research has been accumulated since database systems emerged in the early 1970s. A well established concurrency control theory for database systems is outlined in the references mentioned above: serializability theory, which allows to effectively design and analyze concurrency control methods and mechanisms. An alternative theory for concurrency control of atomic transactions over abstract data types is presented in (Lynch et al. 1993), and not utilized below. This theory is more refined, complex, with a wider scope, and has been less utilized in the Database literature than the classical theory above. Each theory has its pros and cons, emphasis and insight. To some extent they are complementary, and their merging may be useful.

To ensure correctness, a DBMS usually guarantees that only *serializable* transaction schedules are generated, unless *serializability* is intentionally relaxed to increase performance, but only in cases where application correctness is not harmed. For maintaining correctness in cases of failed (aborted) transactions (which can always happen for many reasons) schedules also need to have the *recoverability* (from abort) property. A DBMS also guarantees that no effect of *committed* transactions is lost, and no effect of *aborted* (rolled back) transactions remains in the related database. Overall transaction characterization is usually summarized by the ACID rules below. As databases have become distributed, or needed to cooperate in distributed environments (e.g., Federated databases in the early 1990, and Cloud computing currently), the effective distribution of concurrency control mechanisms has received special attention.

Database transaction and the ACID rules

The concept of a *database transaction* (or *atomic transaction*) has evolved in order to enable both a well understood database system behavior in a faulty environment where crashes can happen any time, and *recovery* from a crash to a well understood database state. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems. Each transaction has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands). Every database transaction obeys the following rules (by support in the database system; i.e., a database system is designed to guarantee them for the transactions it runs):

- **Atomicity** - Either the effects of all or none of its operations remain ("all or nothing" semantics) when a transaction is completed (*committed* or *aborted* respectively). In other words, to the outside world a committed transaction appears (by its effects on the database) to be indivisible (atomic), and an aborted transaction does not affect the database at all, as if never happened.
- **Consistency** - Every transaction must leave the database in a consistent (correct) state, i.e., maintain the predetermined integrity rules of the database (constraints upon and among the database's objects). A transaction must transform a database from one consistent state to another consistent state (however, it is the responsibility of the transaction's programmer to make sure that the transaction itself is correct, i.e., performs correctly what it intends to perform (from the application's point of view) while the

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates

CHAPTER 12

INTEGRATING DATABASES TO OTHER APPLICATIONS

A **web application** or **web app** is any application software that runs in a web browser or is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a common web browser to render the application.

Web applications are popular due to the ubiquity of web browsers, and the convenience of using a web browser as a client, sometimes called a thin client. The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity, as is the inherent support for cross-platform compatibility. Common web applications include webmail, online retail sales, online auctions, wikis and many other functions.

History

In earlier computing models, e.g. in client-server, the load for the application was shared between code on the server and code installed on each client locally. In other words, an application had its own client program which served as its user interface and had to be separately installed on each user's personal computer. An upgrade to the server-side code of the application would typically also require an upgrade to the client-side code installed on each user workstation, adding to the support cost and decreasing productivity.

In contrast, web applications use web documents written in a standard format such as HTML and JavaScript, which are supported by a variety of web browsers. Web applications can be considered as a specific variant of client-server software where the client software is downloaded to the client machine when visiting the relevant web page, using standard procedures such as HTTP. Client web software updates may happen each time the web page is visited. During the session, the web browser interprets and displays the pages, and acts as the *universal* client for any web application.

In the early days of the Web each individual web page was delivered to the client as a static document, but the sequence of pages could provide an interactive experience, as user input is returned through web form elements embedded in the page markup.

In 1995 Netscape introduced a client-side scripting language called JavaScript allowing programmers to add some dynamic elements to the user interface that ran on the client side. So instead of sending data to the server in order to generate an entire web page, the embedded scripts of the downloaded page can perform various tasks such as input validation or showing/hiding parts of the page.

In 1996, Macromedia introduced Flash, a vector animation player that could be added to browsers as a plug-in to embed animations on the web pages. It allowed the use of a scripting language to program interactions on the client side with no need to communicate with the server.

In 1999, the "web application" concept was introduced in the Java language in the Servlet Specification version 2.2. [2.1?]. At that time both JavaScript and XML had already been developed, but Ajax had still not yet been coined and the XMLHttpRequest object had only been recently introduced on Internet Explorer 5 as an ActiveX object.

In 2005, the term Ajax was coined, and applications like Gmail started to make their client sides more and more interactive. A web page script is able to contact the server for storing/retrieving data without downloading an entire web page.

In 2011, HTML5 was finalized, which provides graphic and multimedia capabilities without the need of client side plugins. HTML5 also enriched the semantic content of documents. The APIs and document object model (DOM) are no longer afterthoughts, but are fundamental parts of the HTML5 specification. WebGL API paved the way for advanced 3D graphics based on HTML5 canvas and JavaScript language. These have significant importance in creating truly platform and browser independent rich web applications.

Interface

Through Java, JavaScript, DHTML, Flash, Silverlight and other technologies, application-specific methods such as drawing on the screen, playing audio, and access to the keyboard and mouse are all possible. Many services have worked to combine all of these into a more familiar interface that adopts the appearance of an operating system. General purpose techniques such as drag and drop are also supported by these technologies. Web developers often use client-side scripting to add functionality, especially to create an interactive experience that does not require page reloading. Recently, technologies have been developed to coordinate client-side scripting with server-side technologies such as PHP. Ajax, a web development technique using a combination of various technologies, is an example of technology which creates a more interactive experience.

Structure

Applications are usually broken into logical chunks called "tiers", where every tier is assigned a role. Traditional applications consist only of 1 tier, which resides on the client machine, but web applications lend themselves to an n-tiered approach by nature. Though many variations are possible, the most common structure is the three-tiered application. In its most common form, the three tiers are called *presentation*, *application* and *storage*, in this order. A web browser is the first tier (presentation), an engine using some dynamic Web content technology (such as ASP, ASP.NET, CGI, ColdFusion, JSP/Java, PHP, Perl, Python, Ruby on Rails or Struts2) is the middle tier (application logic), and a database is the third tier (storage). The web browser sends requests to the middle tier, which services them by making queries and updates against the database and generates a user interface.

For more complex applications, a 3-tier solution may fall short, and it may be beneficial to use an n-tiered approach, where the greatest benefit is breaking the business logic, which resides on the application tier, into a more fine-grained model. Another benefit may be adding an integration tier that separates the data tier from the rest of tiers by providing an easy-to-use interface to access the data. For example, the client data would be accessed by calling a "list_clients()" function instead of making an SQL query directly against the client table on the database. This allows the underlying database to be replaced without making any change to the other tiers.

There are some who view a web application as a two-tier architecture. This can be a "smart" client that performs all the work and queries a "dumb" server, or a "dumb" client that relies on a "smart" server. The client would handle the presentation tier, the server would have the database (storage tier), and the business logic (application tier) would be on one of them or on both. While this increases the scalability of the applications and separates the display and the database, it still doesn't allow for true specialization of layers, so most applications will outgrow this model.

IMPORTANCE OF INTEGRATING INTERNET TECHNOLOGIES TO AN ORGANIZATIONS DATABASE

Business use

An emerging strategy for application software companies is to provide web access to software previously distributed as local applications. Depending on the type of application, it may require the development of an entirely different browser-based interface, or merely adapting an existing application to use different presentation technology. These programs allow the user to pay a monthly or yearly fee for use of a software application without having to install it on a local hard drive. A company which follows this strategy is known as an application service provider (ASP), and ASPs are currently receiving much attention in the software industry.

Security breaches on these kinds of applications are a major concern because it can involve both enterprise information and private customer data. Protecting these assets is an important part of any web application and there are some key operational areas that must be included in the development process. This includes processes for authentication, authorization, asset handling, input, and logging and auditing. Building security into the applications from the beginning can be more effective and less disruptive in the long run.

In cloud computing model web applications are software as a service (SaaS). There are business applications provided as SaaS for enterprises for fixed or usage dependent fee. Other web applications are offered free of charge, often generating income from advertisements shown in web application interface.

Many businesses are enabled by open source web applications such as e-commerce software that facilitates easily creating an online retail store. Most businesses today do not need to buy data center hardware such as servers because they are affordably rented on a short term basis from a

THIS IS A SAMPLE OF THE REAL NOTES
and it's not for sale

This sample contains only a few pages extracted from the real notes to show you how the real notes look like before you decide to purchase them.

For the FULL study notes or revision kits call/text/whatsapp 0707737890

Or email someakenya@gmail.com

You may also like to visit our site www.revisednotes.blogspot.com

Or like our facebook page [www.fb.com/studycpa](https://www.facebook.com/studycpa) for news and updates