

CONTENT

COURSE OBJECTIVES.....	5
EXPECTED OUTCOMES	5
COURSE PRESENTATION	5
REFERENCES AND RECOMMENDED TEXTBOOKS.....	5
COURSE ASSESSMENT.....	6
ATTENDANCE.....	7
OFFICE HOURS.....	7
1. Introduction to the Course	8
1.1 Chapter One Objectives	8
1.2 Definition of Database	8
1.3 Database Users	10
1.4 Types of Databases	11
1.5 Database System (DBS).....	13
1.6 Database Management System (DBMS).....	13
1.6.1 Advantages of using DBMS.....	14
1.6.2 Disadvantages of using DBMS.....	15
1.6.3 Components of DBMS.....	15
1.7 DBMS Essential Elements	16
1.7.1 Data Structure	16
1.7.2 Database Query Language	17
1.7.3 Transaction Mechanism	17
1.7.4 Database Modeling Language.....	17
1.7.5 Database System Architecture.....	22
2. Terminologies used in RDBMS.....	25
2.1 Objectives of Chapter.....	26
2.2 Value-Related Terms	26
2.2.1 Data	26
2.2.2 Information	26
2.2.3 A null.....	26

Database Systems (CE 279)

2.3	Structure-Related Terms	27
2.3.1	<i>Table</i>	27
2.3.2	<i>Field</i>	29
2.3.3	<i>A record</i>	29
2.3.4	<i>A view</i>	29
2.3.5	<i>Keys</i>	30
2.3.6	<i>Indexes</i>	31
2.4	Relationship-Related Terms	32
2.4.1	<i>Relationships</i>	32
2.4.2	<i>Types of Relationships</i>	32
2.5	Integrity-Related Terms	36
2.5.1	<i>Field specification</i>	36
2.5.2	<i>Data Integrity</i>	36
3.	Entity Relationship Model (Entity Diagrams).....	38
3.1	Chapter Objectives	39
3.2	Entity- Relationship Model.....	39
3.2.1	<i>Entity</i>	40
3.2.2	<i>Attribute</i>	40
3.2.3	<i>Relationships</i>	42
3.3	Cardinalities	44
4.	Normalization in Relational Databases	46
4.1	Chapter objectives.....	47
4.2	Normalization.....	47
4.3	Basic normalization Concepts.....	47
4.3.1	<i>Functional Dependencies</i>	48
4.3.2	<i>First Normal Form (or 1NF)</i>	49
4.3.3	<i>Second Normal Form</i>	49
4.3.4	<i>Third Normal Form</i>	49
4.4	A sample Database to illustrate the process of normalization	50
4.5	Conclusion on Database Normalization	51
4.6	Databases Transactions.....	52
4.6.1	<i>Transaction ACID Rules</i>	52

Database Systems (CE 279)

4.6.2	<i>Transactions Example & ACID Properties</i>	53
4.7	Concurrency Control in Relational Databases	54
4.7.1	<i>Reason for Concurrency Control</i>	55
4.7.2	<i>Concurrency control mechanisms</i>	55
5.	Database-Design Process	58
5.1	Chapter objectives	58
5.2	Facts-finding Techniques in Database Design	58
5.2.1	<i>When to use Fact-Finding Techniques</i>	58
5.2.2	<i>Fact-Finding Techniques</i>	59
5.3	Design Process	60
5.4	Steps involved in Database Design Processes	61
6.	Database language SQL	65
6.1	Chapter objectives	65
6.2	SQL	65
6.3	SQL Language Elements	65
6.4	SQL Queries	66
6.5	SQL Command Categories	67
6.5.1	<i>SQL Data Definition Language</i>	68
6.5.2	<i>SQL Data Manipulation Language</i>	69
6.5.3	<i>SQL Data Control</i>	73
6.6	SQL Joins: Data from multiple tables	74
6.6.1	<i>Inner joins</i>	74
6.6.2	<i>Outer Joins</i>	76
6.7	SQL Data types	77
6.8	Database Constraints and Triggers	79
6.8.1	<i>Database Constraints</i>	79
6.8.2	<i>Database Triggers</i>	84

COURSE OBJECTIVES

This course seeks to:

- Provide students with basic knowledge on Database Systems and the different Database Models used.
- To teach students Database Design processes and the practical use of Relational Database Management Systems with emphasis on MS Server.
- Introduce students to designing Relational Database and querying using the Structure Query Language.

EXPECTED OUTCOMES

Students completing this module are expected to:

- Know how to design simple databases
- Obtain problem solving approach to database design
- Employ basic fact-finding techniques for database design
- Know how to query databases using SQL commands
- Know how to create Relational Databases and do efficient querying in any Relational Database Management environment.

COURSE PRESENTATION

The course will be delivered through a series of lectures supported with handouts and tutorials, laboratory works and seminars. It is expected that some seminars will be student led.

REFERENCES AND RECOMMENDED TEXTBOOKS

- Coronel, C. and Morris, S., 2018. *Database systems: design, implementation, & management*. Cengage Learning.
- Taylor, J., 2018. *SQL For Dummies 2018: SQL QuickStart Guide the Simplified Beginner's Guide To SQL*.

Database Systems (CE 279)

- Gupta, S.B. and Mittal, A., 2017. *Introduction to Database Management System*. University Science Press.
- Elmasri, R., 2017. *Fundamentals of database systems*.
- Coronel, C. and Morris, S., 2016. *Database systems: design, implementation, & management*. Cengage Learning.
- Nield, T., 2016. *Getting Started with SQL: A Hands-on Approach for Beginners*. "O'Reilly Media, Inc."
- Gupta, S.K., 2016. *Advanced Oracle PL/SQL Developer's Guide*. Packt Publishing Ltd.
- Connolly, T. and Begg, C., 2015. *Database systems*. Pearson Education UK.
- www.cs.sfu.ca
- www.wikipedia.com
- www.wofford-ecs.org
- www.lkeydata.com/sql/sql.html

COURSE ASSESSMENT

Grading System	Factor	Weight	Location	Date	Time
	Assignment & quizzes	15 %	Assignment	Could be announced or NOT	
	Attendance	10 %	In class	Random	
	Semester Project	15 %		Date to be Announced	
	Final Exam	60 %	(TBA)	To Be Announced (TBA)	3 Hrs
80-100%		70-79.9%	60-69.9%	50-59.9%	0-49.9%
A		B	C	D	F

ATTENDANCE

UMaT rules and regulations say that, attendance is **MANDANTORY** for every student. A total of **FIVE (5)** attendances shall be taken at random to the 10%. The only acceptable excuse for absence is the one authorized by the Dean of Student on their prescribed form. However, a student can also ask permission from me to be absent from a particular class with a tangible reason. A student who misses all the five random attendances marked **WOULD** not be allowed to take the final exams.

OFFICE HOURS

I will be available in my office every Thursday (10.00-12.00hrs) to answering students' questions and provide guidance on any issues related to the course.

Please Note the Following:

- Students must endeavour to attend all lectures; lab works and do all their assignments and coursework.
- Students must be seated and fully prepared for lectures at least 5 minutes before scheduled time.
- Under no circumstance a student should be late more than 15 minutes after scheduled time
- **NO** student shall be admitted into the lecture room more than 15 minutes after the start of lectures unless pre-approved by me.
- All cell phones, iPod, MP3/MP4s, and PDAs e. t. c **MUST** remain switched off throughout the lecture period.
- There shall be no eating or gum chewing in class
- Plagiarism shall **NOT** be accepted in this course so be sure to do your referencing properly

Thank You

1. Introduction

1.1 Chapter Objectives

The objectives of this chapter are to:

- Understand the term database; types and uses of databases
- Explain the term Database System (DBS), Database Management System (DBMS) and functions of its various components.
- Know the different Database Architectures and various Database Users
- Know the different Database Models, with emphasises on the Relational Database Model and Relational Database Management System RDBMS

1.2 Definition of Database

The traditional approach of storing information in spreadsheets had several problems identified with it hence the significance of database systems. Some problems of the traditional approach include:

- Data redundancy- is the presence of duplicate data in multiple data files. In this situation confusion results because the data can have different meanings in different files.
- Program-data dependence- is the tight relationship between data stored in files and the specific programs required to update and maintain those files. This dependency is very inefficient, resulting in the need to make changes in many programs when a common piece of data (such as postal code) changes.
- Lack of flexibility -refers to the fact that it is very difficult to create new reports from the data when needed. Ad hoc reports are impossible; a new report could require several weeks of work by more than one programmer and the creation of intermediate files to combine data from disparate files.
- Poor security- is a problem resulting from the lack of control over the data because it is widespread and distributed into so many files.

Database Systems (CE 279)

Database addresses the issues of data redundancy, flexibility (availability) of data, security and program independence

Definition

Database—a collection of related data stored and organized in a manner so it can be retrieved as needed **OR** A database is a collection of interrelated data stored together with controlled redundancy to serve one or more applications in an optimal way.

It is also defined as a collection of logically related data stored together that is designed to meet information requirements of an organization.

Hierarchy from the smallest unit of information to the largest

Bit: Smallest unit of data; binary digit (0, 1)

Byte: Group of bits that represents a single character (8 bits= 1 byte)

Field: Group of related bytes - related words or complete number

Record: Group of related fields

File: Group of records of same type

Database: Group of related files

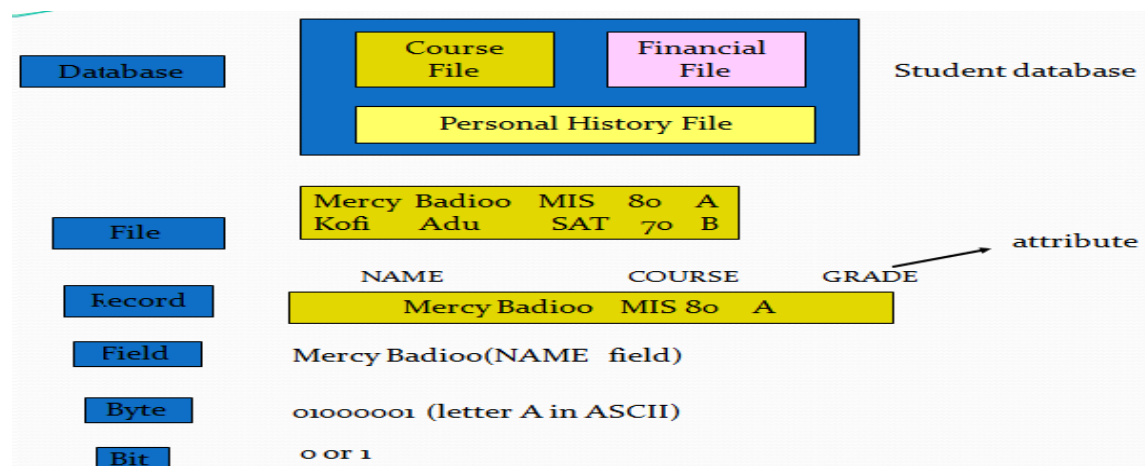


Figure 1.1: Hierarchy of database

Databases are very useful in today's world for the following reasons:

- Every organization has data that needs to be collected, managed, and analysed
- Data analysts, database designers, and database administrators (DBAs) translate the data in a database into useful information for both day-to-day operations and long-term planning of any organisation.
- Organization now can store large amount of data electronically in a format we human cannot read or understand

1.3 Database Users

There are four different types of database users, categorized by the way they expect to interact with the database system. Different types of user interfaces have been designed for the different types of users.

Naïve User

The end users or naive users use the database system through a menu-oriented application program, where the type and range of response is always displayed on the screen. The user need not be aware of the presence of the database system and is instructed through each step. A user of an ATM falls in this category. The typical user interface for naïve users is a forms interface, where the user can fill in appropriate fields of the form.

Application Programmer

These are the professional programmers or software developers who develop the application programs or user interfaces for the naive and online users. These programmers must have the knowledge of programming languages such as Assembly, C, C++, Java, or SQL, etc., since the application programs are written in these languages.

Specialized Users

These types of users interact with the system without writing programs instead; they form their requests in a database query language. They submit each query to a query processor; whose function is to break down statements into instructions that the storage manager

understands. Analysts who submit queries to explore data in the database fall in this category. They are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

Database Administrator

In any organization where many people use the same resources, there is a need for an administrator to oversee and manage these resources. In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA)/Data Analyst/Data Developer. The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.

1.4 Types of Databases

There are two types of databases found in database management:

- ***Operational Databases***
- ***Analytical Databases***

Operational databases -are the backbone of many companies, organizations, and institutions throughout the world today. This type of database is primarily used in ***on-line transaction processing*** (OLTP) scenarios, that is, in situations where there is a need to collect, modify, and maintain data on a daily basis. The type of data stored in an operational database is dynamic, meaning that it changes constantly and always reflects up-to-the-minute information. Examples: Organizations such as retail stores, manufacturing companies, hospitals and clinics, and publishing houses, use operational databases.

Analytical databases -are primarily used in on-line analytical processing (OLAP) scenarios, where there is a need to store and track historical and time-dependent data. An analytical database is a valuable asset when there is a need to track trends, view statistical data over a long period of time, and make tactical or strategic business projections. Examples: Chemical

Database Systems (CE 279)

labs, geological companies, and marketing-analysis firms are examples of organizations that use analytical databases. In general, we can assume that OLTP systems provide source data to data warehouses, whereas OLAP systems help to analyse it.

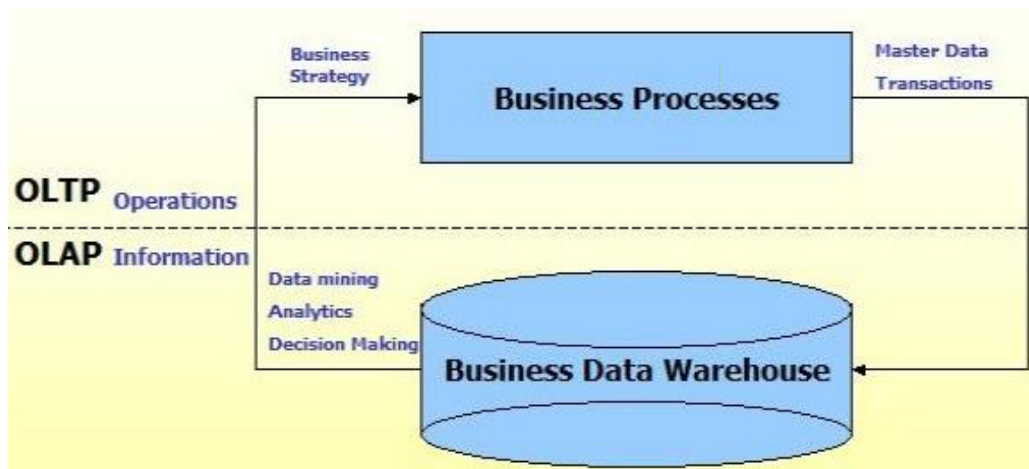


Figure 1.2: Distinction between OLTP and OLAP

Differences Between OLTP and OLAP Databases

	OLTP	OLAP
Purpose	Manage and control fundamental business operations	Provide a consolidated view of enterprise data for reporting, decision-making, and planning
Data source	The original source of data where business data is being recorded in real-time	Data is integrated from different OLTP systems
Query type	Simple, standardized queries based on online items. Typically returning a few records.	Large, complex queries used for business decisions. Aggregation of tables across multiple databases is often required.
Query form	Centered on INSERT, DELETE, and UPDATE commands	Transactions don't need to be recorded, so mostly SELECT command is used for reporting
View of data	A view of day-to-day business transactions is presented	A multi-dimensional view of enterprise data is presented
Data processing	Fast processing because queries are small and simple	Complex query processing may take hours. Full-load from OLTP in batches is also time-intensive

1.5 Database System (DBS)

A database is managed by a Database Management System (DBMS), typically referred to as Database System (DBS).

A database system is a term that is typically used to encapsulate the constructs of a data model, database Management system (DBMS) and database.

A database system like any information system consists of the following;

- Hardware- Can range from a PC to a network of computers
- Software- DBMS, operating system, network software (if necessary) and also the application programs.
- Data- Used by the organization and a description of this data called the schema.
- Procedures- Instructions and rules that should be applied to the design and use of the database and DBMS.
- People- Includes database designers, DBAs, application programmers, and end-users.

1.6 Database Management System (DBMS)

A database management system (DBMS) is a computerized system that enables users to create and maintain a database. The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating*, and *sharing* databases among various users and applications. Data in a database can be *added, deleted, changed, sorted or searched* all using a DBMS.

It allows organizations to place control of database development in the hands of database administrators (DBAs) and other specialists.

Functions performed by DBMS include:

- Creates and maintains databases
- Eliminates requirement for *data definition statements* in Application Programs
- Acts as interface between *application programs and physical data files*

- Separates logical and physical views of data (A logical view of data is the way data is perceived by end users or business specialists. A physical view of data is the way the data are actually organized and structured on physical storage media.)

A DBMS is a system software package that helps the use of integrated collection of data records and files known as databases. It allows different user application programs to easily access the same database. It helps to specify the logical organization for a database and access and use the information within a database. It provides facilities for controlling data access, enforcing data integrity, managing concurrency, and restoring the database from backups. A DBMS also provides the ability to logically present database information to users. When a DBMS is used, new categories of data can be added to the database without disruption to the existing system.

1.6.1 Advantages of using DBMS

Improved availability: One of the principal advantages of a DBMS is that the same information can be made available to different users.

Minimized redundancy: The data in a DBMS is more concise because information in it appears just once. Minimizing redundancy can therefore significantly reduce the cost of storing information on storage devices.

Accuracy: consistent and up-to-date data is a sign of data integrity. DBMSs foster data integrity because updates and changes to the data only have to be made in one place. The chances of making a mistake are higher if you are required to change the same data in several different places than if you only have to make the change in one place.

Program and file consistency: Using a database management system, file formats and system programs are standardized. This makes the data files easier to maintain because the same rules and guidelines apply across all types of data. The level of consistency across files and programs also makes it easier to manage data when multiple programmers are involved.

User-friendly: Data is easier to access and manipulate with a DBMS than without it. In most cases, DBMSs also reduce the reliance of individual users on computer specialists to meet their data needs.

Improved security: DBMSs allow multiple users to access the same data resources. This capability is generally viewed as a benefit, but there are potential risks for the organization. Through the use of passwords, database management systems can be used to restrict data access to only those who should see it.

1.6.2 Disadvantages of using DBMS

- Complexity
- Cost of DBMS
- Cost of conversion
- Performance
- Higher impact of a fail

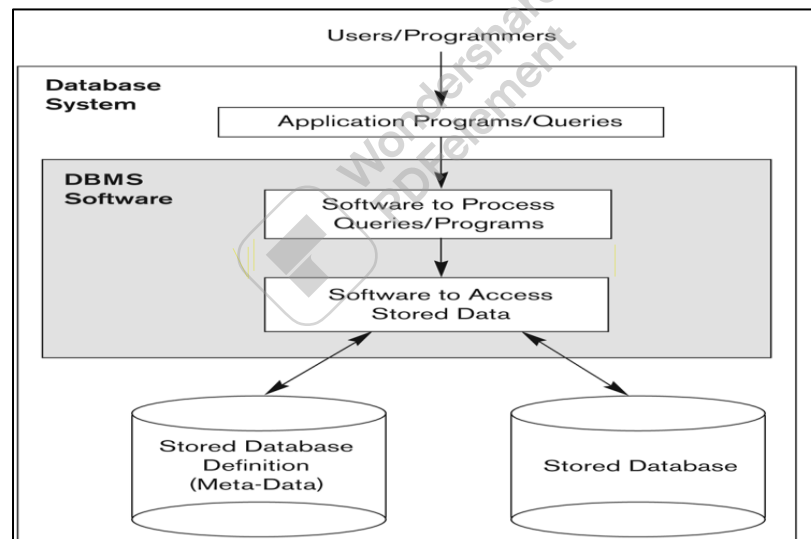


Figure 1.3: A Simplified Database System Environment

1.6.3 Components of DBMS

DBMS Engine accepts logical request from the various DBMS subsystems, converts them into physical equivalents, and actually accesses the database and **data dictionary** as they exist on a storage device.

Database Systems (CE 279)

Data Definition Subsystem helps user to create and maintain the data dictionary and define the structure of the files in a database.

Data Manipulation Subsystem helps user to add, change, and delete information in a database and query it for valuable information. Software tools within the data manipulation subsystem are most often the primary interface between the user and the information contained in a database. It allows user to specify its logical information requirements.

Application Generation Subsystem contains facilities to help users to develop transaction-intensive applications. It usually requires that user perform a detailed series of tasks to process a transaction. It facilitates easy-to-use data entry screens, programming languages, and interfaces.

Data Administration Subsystem helps users to manage the overall database environment by providing facilities for backup and recovery, security management, query optimization, concurrency control, and change management.

Examples of DBMS include: Oracle, DB2 (IBM), MS SQL Server, MS Access, Ingres, MySQL etc.

1.7 DBMS Essential Elements

There are four essential elements that are found with just about every example of DBMS currently on the market. These are as follows:

- Data Structure,
- Database Query Language
- Transaction Mechanisms and
- Modelling Language

1.7.1 Data Structure

Data structures are administered by the DBMS. Examples of data that are organized by this function are individual profiles or records, files, fields and their definitions, and objects such as visual media. Data structures are what allow DBMS to interact with the data without causing damage to the integrity of the data itself.

1.7.2 Database Query Language

This element is involved in maintaining the security of the database, by monitoring the use of login data, the assignment of access rights and privileges, and the definition of the criteria that must be employed to add data to the system. The data query language works with the data structures to make sure it is harder to input irrelevant data into any of the databases in use on the system.

1.7.3 Transaction Mechanism

This element helps to allow multiple and concurrent access to the database by multiple users, prevents the manipulation of one record by two users at the same time, and preventing the creation of duplicate records.

1.7.4 Database Modelling Language

Modelling language serves to define the language of each database that is hosted via the DBMS. Essentially, the modelling language ensures the ability of the databases to communicate with the DBMS and thus operate on the system.

There are several approaches currently in use,

- Hierarchical model,
- Network model,
- Relational model, and
- Object model

The Hierarchical Database Model

The hierarchical database model stores data logically in a vertical hierarchy resembling a tree-like structure. An upper record is connected logically to a lower record in a parent-child relationship. A parent segment can have more than one child, but a child can only have one parent. This model is good for treating one-to-many relationships. They can store large numbers of segments and process efficiently, but they can only deliver information if a request follows the linkages of the hierarchy. Their disadvantages include their low user-friendliness, inflexibility, and complexity of programming. However, they are good for high volume rapid response systems, such as airline reservation systems

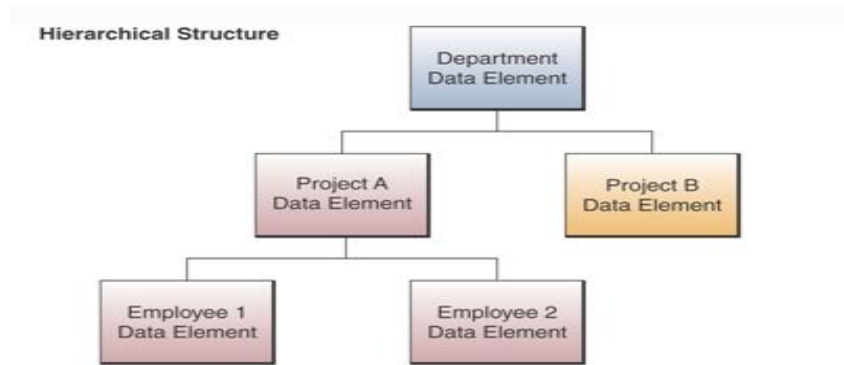


Figure 1.4: A typical example of hierarchical model

The Network Database Model

The network model stores data logically in a structure that permits many-to-many relationships. Through extensive use of pointers, child segment can have more than one parent. Network DBMS reduce redundancy and they process information efficiently. However, they are inflexible and very complex to maintain and program.

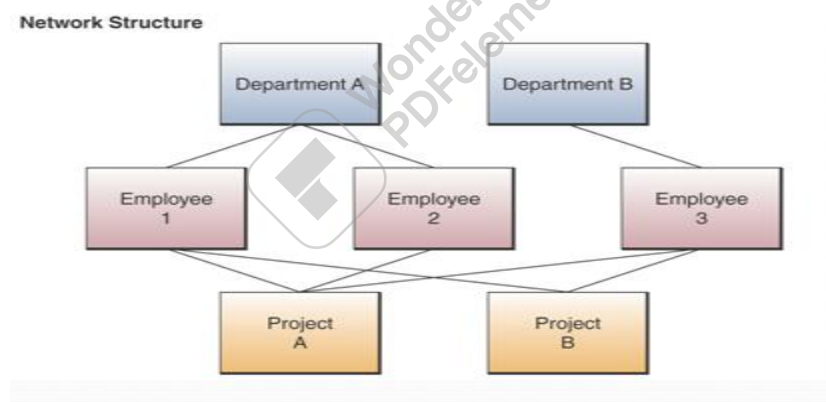


Figure 1.5: A typical example of network model

The Relational Database Model

A relational database stores data in relations, which the user perceives as tables. Each relation is composed of tuples, or records, and attributes, or fields. Each record in the table is identified by a field that contains a unique value. This is unlike the hierarchical and network database models, in which knowing the layout of the structures is crucial to retrieving data. Relational database model represents data as a two-dimensional table called a relation.

Model Structure of Relational Model

Relation: A relation comprises a set of tuples. Figure 1.6 is a tabular representation of the *Student* relation containing five tuples.

Tuple: A tuple is a sequence of attributes i.e. a row in the relation table. There are five tuples shown in the *Student* relation in Figure 1.6 - the one highlighted concerns *Student* identified by the *MatricNo* 's07'.

Attribute: An attribute is a named column in the relation table. The *Student* relation in Figure 6 contains four attributes - the *MatricNo* attribute is highlighted, other attributes are *Name*, *Registered* and *Counsellor*.

Domain: The domain construct is important as it identifies the *type* of an attribute. More formally the domain is a named set of values which have a common meaning - the domain of an attribute defines the set of values from which an attribute can draw.

For example, Figure 1.5 highlights the domain *Years* for the attribute *Registered*. This determines that the *Registered* attribute will draw a value from the valid set of integers which represent a year.

Properties of Relational Tables

- Values are atomic
- Each row is unique
- Column values are of the same kind
- The sequence of columns and rows are insignificant
- Each column has a unique name

Advantages of a Relational Database

The relational database provides a number of advantages over previous models, such as the following:

Built-in multilevel integrity: Data integrity is built into the model at the field level to ensure the accuracy of the data; at the table level to ensure that records are not duplicated and to

Database Systems (CE 279)

detect missing primary key values; at the relationship level to ensure that the relationship between a pair of tables is valid; and at the business level to ensure that the data is accurate.

Logical and physical data independence from database applications: Neither changes a user makes to the logical design of the database, nor changes a database software vendor makes to the physical implementation of the database, will adversely affect the applications built upon it.

Guaranteed data consistency and accuracy: Data is consistent and accurate due to the various levels of integrity you can impose within the database.

Easy data retrieval: At the user's command, data can be retrieved either from a particular table or from any number of related tables within the database. This enables a user to view information in an almost unlimited number of ways.

Attribute MatricNo: MatricNos	Name: PersonNames	Domain Registered: Years	Counsellor: StaffNos
S01	Bloggs	1993	4523
S02	Smith	1998	3412
S05	Jones	NULL	4523
S07	Stewart	1996	4538
S09	MacDonald	1995	4523

Figure 1.6: A typical example of Relational database model

Relational Database Management Systems (RDBMS)

A relational database management system (RDBMS) is a software program used to create, maintain, modify, and manipulate a relational database. RDBMS data is structured in database tables, fields and records. RDBMS store the data into collection of tables, which might be related by common fields (database table columns). The most popular RDBMS are Access, MS SQL Server, Oracle and MySQL.

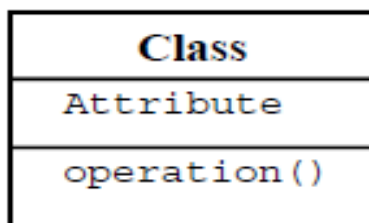
Object-Oriented Database

An object-oriented database is a database that subscribes to a model with information represented by objects. Object-oriented databases (OODBs) try to maintain a direct correspondence between real-world and database objects so that objects do not lose their integrity and identity and can easily be identified and operated upon. An object database (also object-oriented database management system) is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object-oriented databases is sometimes used together with relational database; this approach produces Object-relational databases.

Benefits of Object-Oriented Modelling

- Ability to tackle challenging problems
- Improved communication between users, analysts, designers, and programmers
- Increased consistency in analysis, design, and programming
- Explicit representation of commonality among system components
- System robustness
- Reusability of analysis, design, and programming results

In object-oriented database every instance of an object consists of attributes and operations (methods). Class diagrams are the most useful components in the UML for modelling databases. Class diagram is made up of the class, attributes and operations.

Example of a Class

A class icon is a rectangle divided into three compartments.

The topmost compartment contains the name of the class.

The middle compartment contains a list of attributes (member variables), and the bottom compartment contains a list of operations (member functions)

Class: An entity that has a well-defined role in the application domain, as well as state, behaviour, and identity

- Tangible: person, place or thing
- Concept or Event: department, performance, marriage, registration
- Artefact of the Design Process: user interface, controller, scheduler

An object can then be defined as a particular instance of a class.

1.7.5 Database System Architecture

Database Management Systems do not all conform to the same architecture. The three-level architecture forms the basis of modern database architectures. This is in agreement with the ANSI/SPARC study group on Database Management Systems. ANSI/SPARC is the American National Standards Institute/Standard Planning and Requirement Committee).

The architecture for DBMSs is divided into three general levels:

- External
- Conceptual
- Internal

The three-level architecture has the aim of enabling users to access the same data but with a personalized view of it.

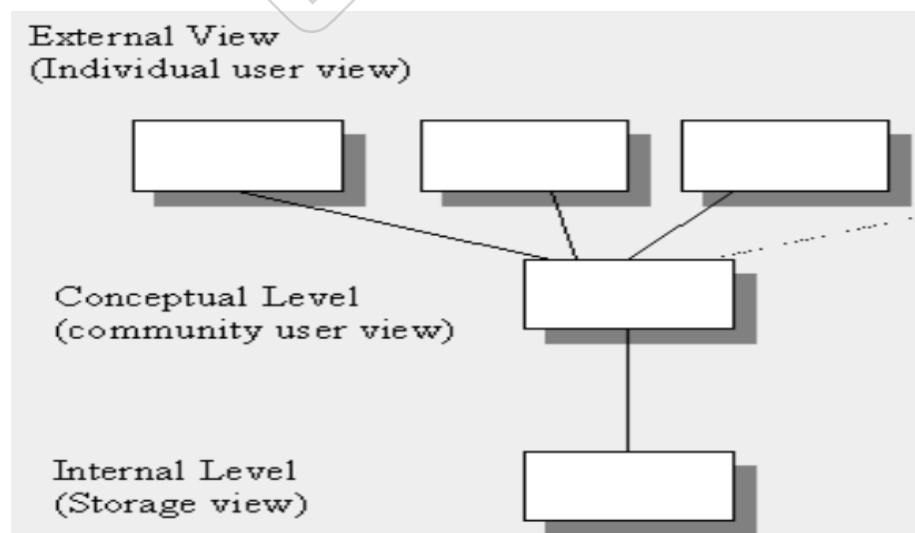


Figure 1.7: Three level database architecture

External View

A user is anyone who needs to access some portion of the data. They may range from application programmers to casual users with ad-hoc queries. Each user has a language at his/her disposal. The application programmer may use a high-level language (e.g. COBOL) while the casual user will probably use a query language.

Each user sees the data in terms of an external view: Defined by an external schema, consisting basically of descriptions of each of the various types of external record in that external view, and also a definition of the mapping between the external schema and the underlying conceptual schema.

Conceptual View

The conceptual level describes the overall logical structure of whole database for a community of users. This level is relational because data visible at this level will be relational tables and operators will be relational operators. The conceptual level does not specify how the data is physically stored. Conceptual view is:

- An abstract representation of the entire information content of the database.
- A general view of the data as it actually is, that is, it is a model of the real world.
- Consists of multiple occurrences of multiple types of conceptual record, defined in the conceptual schema.
- Access strategy is ignored

Internal View

The internal view is a low-level representation of the entire database consisting of multiple occurrences of multiple types of internal (stored) records. The internal view described by the internal schema:

- defines the various types of stored record
- what indices exist
- how stored fields are represented
- what physical sequence the stored records are in

In effect the internal schema is the storage structure definition.

Database Systems (CE 279)

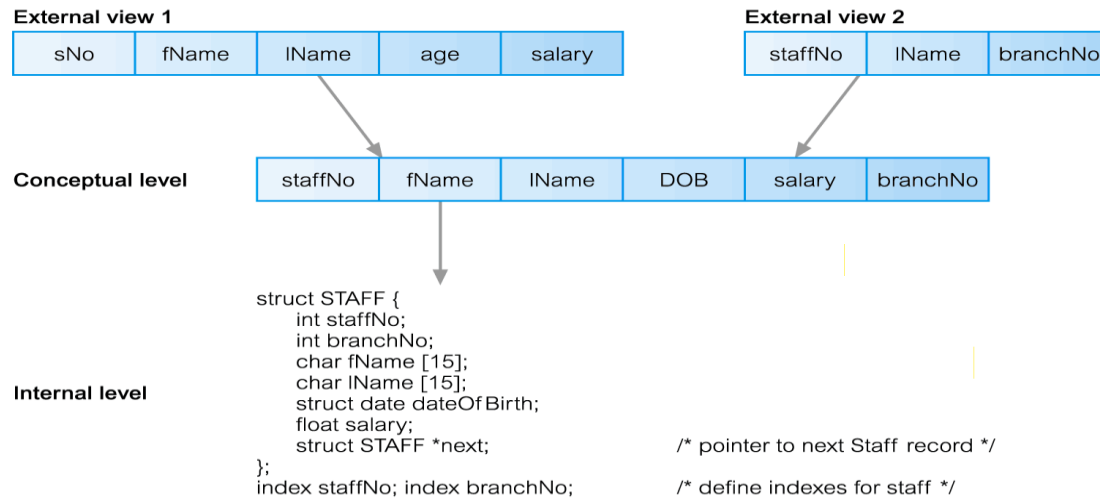


Figure 1.8: Differences between Three Levels of Database System Architecture

In summary, external view is the closest to the user, the conceptual view describes how the database is logically organized, internal view describes on the detailed physical structures of the database. The distancing of the internal level from the external level means that users do not need to know how the data is physically stored in the database. This level separation also allows the Database Administrator (DBA) to change the database storage structures without affecting the users' views.

Chapter Questions

1. Discuss in your own words some problems associated with the traditional approach of storing data.
2. Explain how database addresses each of the challenges mentioned above in (1).
3. List the types of database users and discuss how each type interact with the database system.
4. Define database and discuss the various types
5. Explain the following terms:
 - a. OLTP
 - b. OLAP
 - c. DBMS
 - d. Relation
 - e. Tuple
 - f. Attribute
 - g. Domain
 - h. RDBMS
 - i. DDL
 - j. DML
6. Differentiate between OLTP and OLAP
7. Discuss the essential elements of DBMS
8. List the properties of relational tables
9. Discuss the advantages and disadvantages of a relational database
10. Differentiate between the general levels of the architecture of a DBMS

2. Terminologies used in RDBMS

2.1 Objectives of Chapter

This chapter covers common terms used to define the ideas and concepts of designing a database, and each term is defined and discussed in some detail.

There are four categories of terms used in Relational Databases and RDBMS:

- Value-related,
- Structure-related,
- Relationship-related, and
- Integrity-related.

2.2 Value-Related Terms

2.2.1 Data

Data are the values stored in the database. Data is static in the sense that it remains in the same state until you modify it by some manual or automated process.

Example of data: Ama Bruce 5233 05/05/2005 5000.00

On the surface, this data is meaningless. For example, there is no way for you to determine what "5233" represents. Is it a zip code? Is it a postal code? Even if you know it represents a customer identification number, is it one that is associated with Ama Bruce?

2.2.2 Information

It is data that you process in a manner that makes it meaningful and useful to you when you work with it or view it. It is dynamic in the sense that it constantly changes relative to the data stored in the database. Data must be processed in some manner so that it is turned into meaningful information.

2.2.3 A null

A null represents a missing or unknown value. You must understand from the outset that a null does not represent a zero or a text string of one or more blank spaces. This is because:

- zero can have a very wide variety of meanings. It can represent the state of an account balance, the current number of available first-class ticket upgrades, or the current stock level of a product.
- Although a text string of one or more blank spaces is guaranteed to be meaningless to most of us, it is meaningful to a query language like SQL. A blank space is a valid character. A string composed of three blank spaces (' ') is just as legitimate as a character string composed of three letters ('abc'). In Table 2.1, a blank represents the facts that Leverling Janet has no dependants and Fuller Andrew has just two dependants. Null value is not the same as zero as Null sometimes act as a domain constraint.

Table 2.1: An example of a table containing null values

Last Name	First Name	Home Phone	Dependent1	Dependent2	Dependent3	Dependent4
Davolio	Nancy	(206) 555-9857	Ann	Bob	Jill	
Fuller	Andrew	(206) 555-9482	Barney	Anne		
Leverling	Janet	(206) 555-3412				
Peacock	Margaret	(206) 555-8122	Tom	Jane		
Callahan	Laura	(206) 555-1189	Ron	Sue	Sally	

2.3 Structure-Related Terms

2.3.1 Table

A database consists of one or more tables. A table is a collection of data, arranged in rows (records) and columns (fields). Table 2.2 shows a typical table structure.

Table 2.2: A typical table structure

Client ID	Client First Name	Client Last Name	Client City	<< other fields >>
9001	Stewart	Jameson	Seattle
9002	Shannon	McLain	Poulsbo
9003	Estela	Pundt	Tacoma
9004	Timothy	Ennis	Seattle
9005	Marvin	Russo	Bellingham
9006	Kendra	Bonnicksen	Tacoma

} Records

{ Fields

Database Systems (CE 279)

Tables are the key structures in the database and each table always represents a single, specific subject. The logical order of records and fields within a table is not important, and every table contains at least one field—known as a primary key—that uniquely identifies each of its records. (In Table 2.2, for example, CLIENT ID is the primary key of the CLIENTS table.)

The subject that a given table represents can either be an *object* or *event*. When the subject is an object, it means that the table represents something that is tangible, such as a person, place, or thing. When the subject of a table is an event, it means that the table represents something that occurs at a given point in time having characteristics you wish to record. These characteristics can be stored as data and then processed as information in exactly the same manner as a table that represents some specific object. Table 2.3 shows an example of a table representing an event that we all have experienced at one time or another—a doctor's appointment.

Table 2.3: A table representing an event

Patient ID	Visit Date	Visit Time	Physician	Blood Pressure	<< other fields >>
92001	05/01/96	10:30	Hernandez	120/80
97002	05/01/96	13:00	Piercy	112/74
99014	05/02/96	09:30	Rolson	120/80
96105	05/02/96	11:00	Hernandez	160/90
96203	05/02/96	14:00	Hernandez	110/75
98003	05/03/96	09:30	Rolson	120/80

A table that stores data used to supply information is called a data table, and it is the most common type of table in a relational database. Data in this type of table is dynamic because it can be manipulated and processed into information in some form or fashion.

A **validation table** (also known as a lookup table) -on the other hand, stores data that you specifically use to implement data integrity. A validation table usually represents subjects, such as city names, skill categories and project identification numbers. Data in this type of table is **static**.

Table 2.3: An example of a validation table

Department Code	Department Name
0001	Computer Science and Engineering
0002	Mining
0003	Electrical Engineering
0004	Mechanical Engineering

2.3.2 Field

It is the smallest structure in the database, and it represents a characteristic of the subject of the table to which it belongs. Fields are the structures that actually store data. The data in these fields can then be retrieved and presented as information in almost any configuration.

Every field in a properly designed database contains one and only one value, and its name will identify the type of value it holds. There three other types of fields in an improperly or poorly designed database.

- A multipart field (also known as a composite field), which contains two or more distinct items within its value.
- A multi-valued field, which contains multiple instances of the same type of value.
- A calculated field, which contains a concatenated text value or the result of a mathematical expression.

2.3.3 A record

It represents a unique instance of the subject of a table. It is composed of the entire set of fields in a table, regardless of whether or not the fields contain values. Because of the manner in which a table is defined, each record is identified throughout the database by a unique value in the primary key field of that record.

2.3.4 A view

It is a "virtual" table composed of fields from one or more tables in the database; the tables that comprise the view are known as **base tables**. The relational model refers to a view as "virtual" because it draws data from base tables rather than storing data on its own. In fact, the only information about a view that is stored in the database is its structure.

There are three major reasons why views are important:

- They enable working with data from multiple tables simultaneously.
- They help to prevent certain users from manipulating specific fields within a table or group of tables. This capability can be very advantageous in terms of security.
- They can be used to implement data integrity. A view you use for this purpose is known as a *validation view*.

2.3.5 Keys

Keys are special fields that play very specific roles within a table, and the type of key determines its purpose within the table. There are several types of keys a table may contain, but the two most significant ones are:

- The primary key and
- The foreign key.

A **primary key** - is a field or group of fields that uniquely identifies each record within a table; if a primary key is composed of two or more fields, it is known as a composite primary key. The primary key enforces table-level integrity and helps establish relationships with other tables in the database.

The AGENT ID field in Table 2.4 is a good example of a primary key. It uniquely identifies each agent within the AGENTS table and helps to guarantee table-level integrity by ensuring there are no duplicate records. It can also be used to establish relationships between the AGENTS table and other tables in the database, such as the ENTERTAINERS table shown in the example.

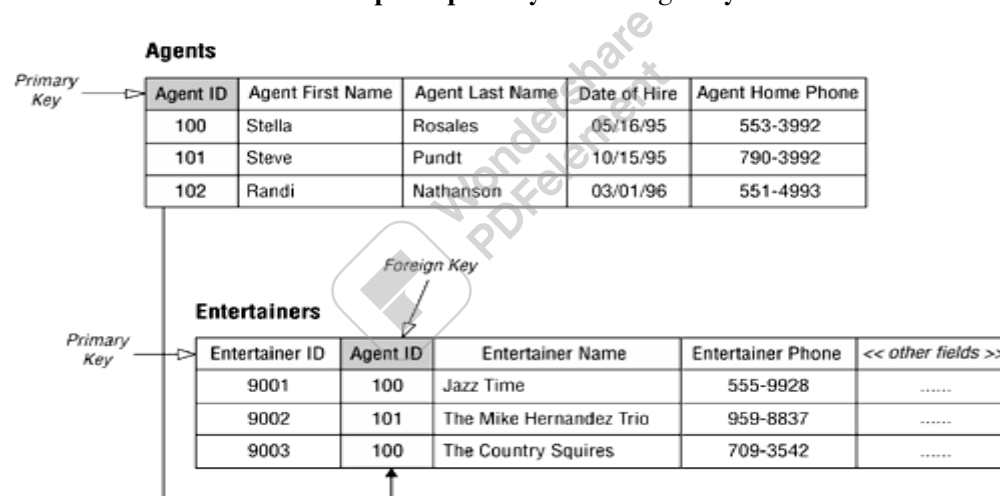
A **foreign key** is a column (the child column) in a table which has a corresponding relationship and a dependency on another column (the parent column) that is usually in a different table. Parent columns can have multiple child columns, but a child column can only have one parent column. The child column is the column with the foreign key; the parent column does not have the foreign key "set" on it, but most databases require the parent column to be indexed.

Foreign keys are made to link data across multiple tables. A child column cannot have a

Table 2.4 also shows a good example of a foreign key. Note that AGENT ID is the primary key of the AGENTS table and a foreign key in the ENTERTAINERS table. AGENT ID assumes this role because the ENTERTAINERS table already has a primary key—ENTERTAINER ID. As such, AGENT ID establishes the relationship between both of the tables.

Besides helping to establish relationships between pairs of tables, foreign keys also help implement and ensure relationship-level integrity. This means that the records in both tables will always be properly related because the values of a foreign key must match existing values of the primary key to which it refers.

Table 2.4: An example of primary and foreign key fields



2.3.6 Indexes

A **database index** is a data structure that improves the speed of data retrieval operations on a database table at the cost of slower writes and increased storage space. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records. Index is a physical structure in RDBMS that is provided to improve data processing.

2.4 Relationship-Related Terms

2.4.1 Relationships

Database relationships are the backbone of all relational databases. A relationship is established between two database tables when one table uses a foreign key that references the primary key of another table. This is the basic concept behind the term relational database.

A relationship is an important component of a relational database because:

- It enables you to create multi-table views.
- It is crucial to data integrity because it helps reduce redundant data and eliminate duplicate data.

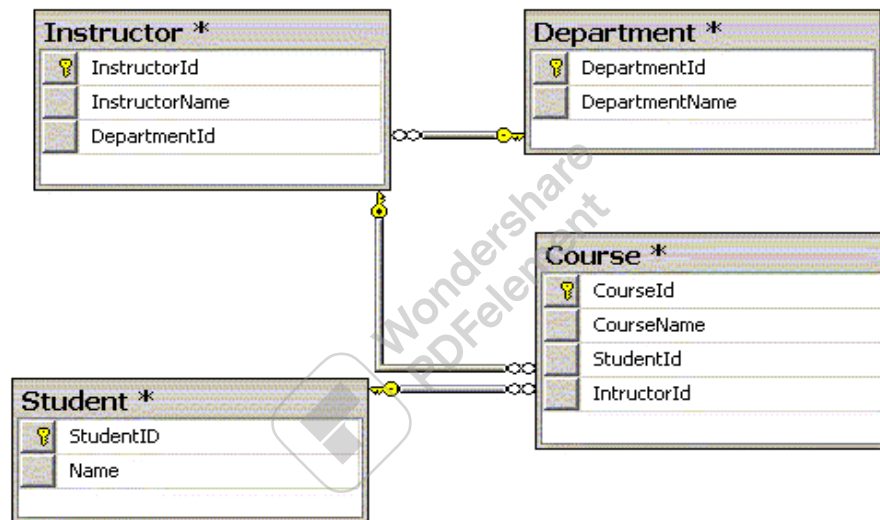


Figure 2.1: Example of relationship between tables

2.4.2 Types of Relationships

There are three specific types of relationship that can exist between a pair of tables namely:

One-to-one, One-to-many and Many-to-many.

One-to-One Relationships

A pair of tables bears a one-to-one relationship when a single record in the first table is related to only one record in the second table, and a single record in the second table is related to only one record in the first table. In this type of relationship, one table serves as a "parent" table and the other serves as a "child" table.

Database Systems (CE 279)

Figure 2.2 shows an example of a typical one-to-one relationship. In this case, EMPLOYEES is the parent table and COMPENSATION is the child table. The relationship between these tables is such that a single record in the EMPLOYEES table can be related to only one record in the COMPENSATION table, and a single record in the COMPENSATION table can be related to only one record in the EMPLOYEES table.

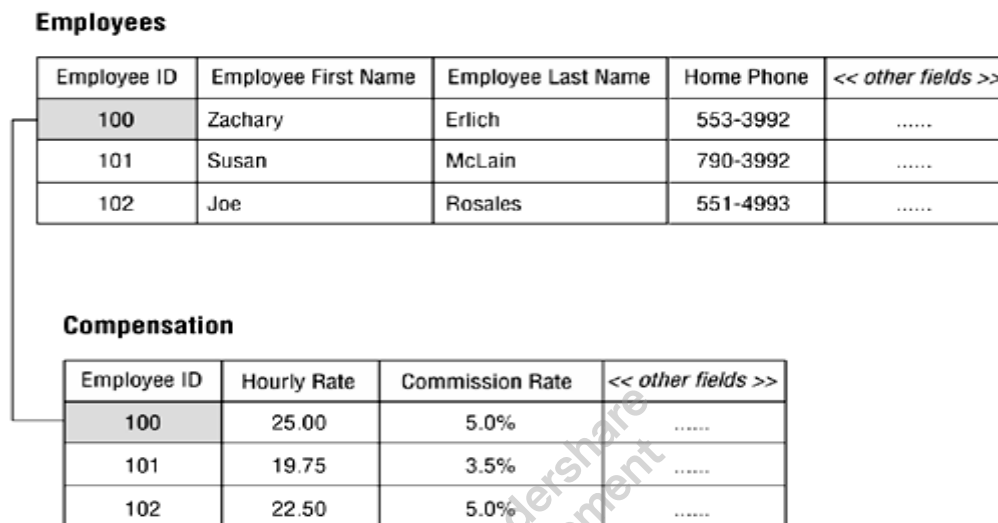


Figure 2.2 an example of a one-to-one relationship

One-to-Many Relationships

A one-to-many relationship exists between a pair of tables when a single record in the first table can be related to many records in the second table, but a single record in the second table can be related to only one record in the first table. In this case, the table on the "one" side of the relationship is the parent table, and the table on the "many" side is the child table.)

The example in Figure 2.3 illustrates a typical one-to-many relationship. A single record in the AGENTS table can be related to one or more records in the ENTERTAINERS table, but a single record in the ENTERTAINERS table is related to only one record in the AGENTS table. As you probably have already guessed, AGENT ID is a foreign key in the ENTERTAINERS table.

Database Systems (CE 279)

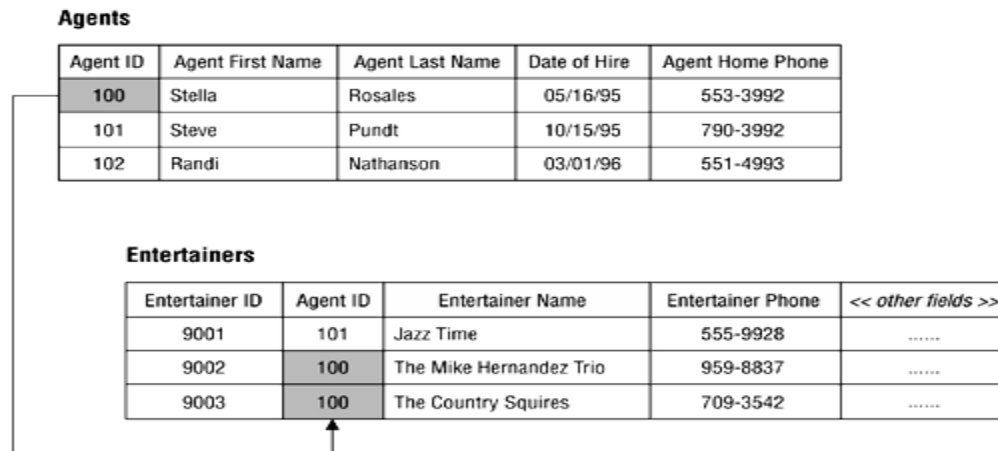


Figure 2.3: Example of a one-to-many relationship

Many-to-Many Relationships

A pair of tables bears a many-to-many relationship when a single record in the first table can be related to many records in the second table and a single record in the second table can be related to many records in the first table. To establish this relationship, a *linking table* is used. A linking table makes it easy to associate records from one table with those of the other. Linking table is defined by taking copies of the primary key of each table in the relationship and using them to form the structure of the new table.

A many-to-many relationship that is not properly established is "unresolved." Figure 2.4 shows an example of an unresolved many-to-many relationship. In this instance, a single record in the STUDENTS table can be related to many records in the CLASSES table and a single record in the CLASSES table can be related to many records in the STUDENTS table.

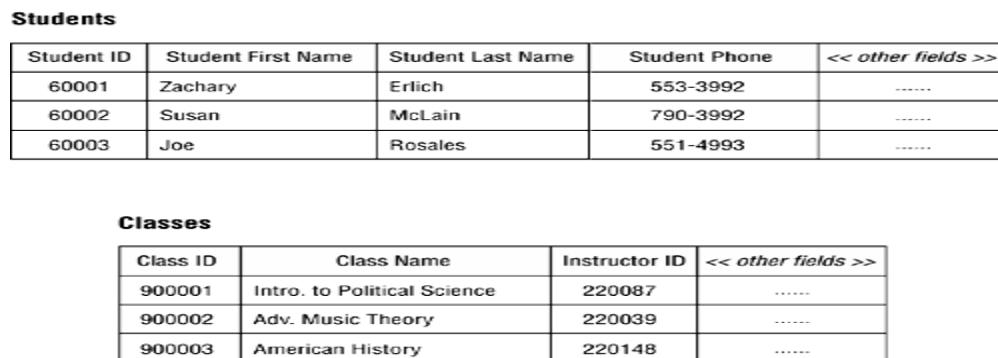


Figure 2.4 an example of an unresolved many-to-many relationship

Database Systems (CE 279)

This relationship is unresolved due to the inherent peculiarity of the many-to-many relationship. In tables shown in Figure 2.4, how do you associate a single student with several classes or a specific class with several students? By creating and using a linking table, which will resolve the many-to-many relationship in the most appropriate and effective manner. Figure 2.5 shows this solution in practice.

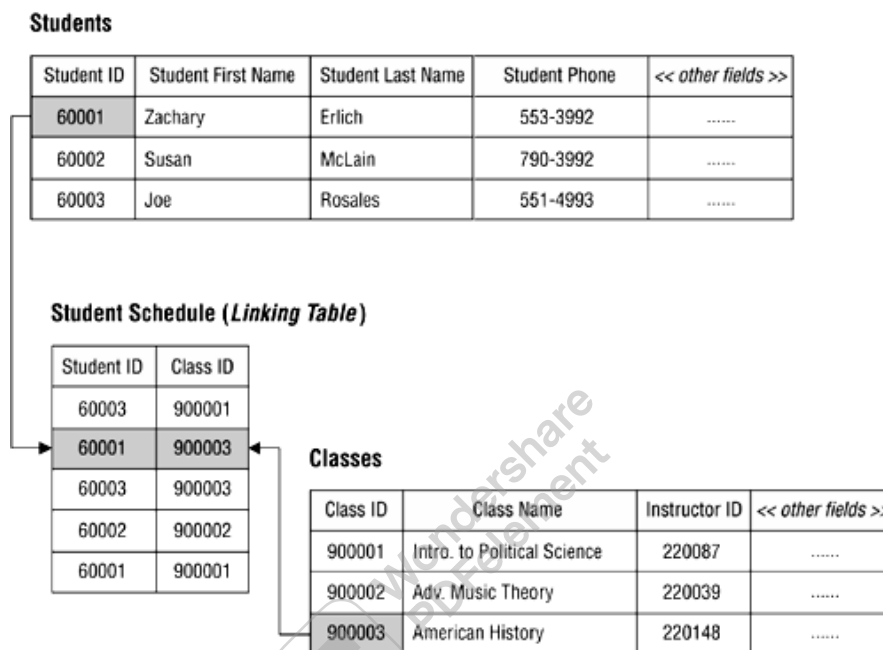


Figure 2.5: Resolving the many-to-many relationship with a linking table

Types of Participation

A table's participation within a relationship can be either mandatory or optional. Assuming there is a relationship between two tables called TABLE_A and TABLE_B. TABLE_A's participation is mandatory if you must enter at least one record into TABLE_A before you can enter records into TABLE_B. TABLE_A's participation is optional if you are not required to enter any records into TABLE_A before you can enter records into TABLE_B.

Degree of Participation

The *degree of participation* determines the minimum number of records that a given table must have associated with a single record in the related table and the maximum number of

records that a given table is allowed to have associated with a single record in the related table.

Consider, once again, a relationship between two tables called TABLE_A and TABLE_B. You establish the degree of participation for TABLE_B by indicating a minimum and maximum number of records in TABLE_B that can be related to a single record in TABLE_A. If a single record in TABLE_A can be related to no fewer than 1 but no more than 10 records in TABLE_B, then the degree of participation for TABLE_B is 1,10. (The degree of participation is notated with the minimum number on the left and the maximum number on the right, separated by a comma.) The degree of participation for TABLE_A can be established in a same manner.

2.5 Integrity-Related Terms

2.5.1 Field specification

Traditionally known as a *domain*, represents all the elements of a field. Each field specification incorporates three types of elements: general, physical, and logical.

General elements constitute the most fundamental information about the field and include items such as Field Name, Description, and Parent Table.

Physical elements determine how a field is built and how it is represented to the person using it. This category includes items such as Data Type, Length, and Display Format.

Logical elements describe the values stored in a field and include items such as Required Value, Range of Values, and Default Value.

2.5.2 Data Integrity

Data integrity refers to the validity, consistency, and accuracy of the data in a database.

Data integrity is one of the most important aspects of the database-design process, and you cannot underestimate, overlook, or even partially neglect it. There are four types of data integrity that is implemented during the database-design process. Three types of data integrity are based on various aspects of the database structure and are labelled according to the area (level) in which they operate. The fourth type of data integrity is based on the way an organization perceives and uses its data. The following is a brief description of each:

Database Systems (CE 279)

- **Table-level integrity** (traditionally known as *entity integrity*) ensures that there are no duplicate records within the table and that the field that identifies each record within the table is unique and never null.
- **Field-level integrity** (traditionally known as *domain integrity*) ensures that the structure of every field is sound; that the values in each field are valid, consistent, and accurate; and that fields of the same type (such as CITY fields) are consistently defined throughout the database.
- **Relationship-level integrity** (traditionally known as *referential integrity*) ensures that the relationship between a pair of tables is sound and that the records in the tables are synchronized whenever data is entered into, updated in, or deleted from either table.
- **Business rules** impose restrictions or limitations on certain aspects of a database based on the ways an organization perceives and uses its data. These restrictions can affect aspects of database design, such as the range and types of values stored in a field, the type of participation and the degree of participation of each table within a relationship, and the type of synchronization used for relationship-level integrity in certain relationships.

Chapter Questions

1. Distinguish between null and zero
2. Differentiate between data and information
3. Discuss the use of validation table and how it ensures data integrity
4. Explain the following terms:
 - i. Table
 - ii. Field
 - iii. Record
 - iv. View
 - v. Primary key
 - vi. Foreign key
 - vii. Database index
 - viii. Mandatory participation
 - ix. Optional participation
 - x. Field specification
5. Discuss in brief why views are important
6. Explain the various types of relationships permissible among tables in a database
7. Discuss the use of linking tables
8. Describe two scenarios that necessitates the use of linking tables
9. Explain data integrity
10. Mention and describe the four types of data integrity in databases

3. Entity Relationship Model (Entity Diagrams)

3.1 Chapter Objectives

Objectives of this chapter are to help students to:

- Identify what entities, attributes and relationship are.
- Know and have good knowledge on use ERD elements
- Design an Entity Relational Diagram to represent a real-world scenario.
- Identify primary, foreign and composite keys.

3.2 Entity- Relationship Model

In 1976, Chen developed the Entity-Relationship (ER) model, a high-level data model that is useful in developing a conceptual design for a database. Creation of an ER diagram helps designers to understand and to specify the desired components of the database and the relationships among those components.

To make the description of the model more complete, consider the example of a physics department at a college that maintains a database of experimental results. Throughout a laboratory, students collaborate and share their results and access data sets from other semesters on a computer system. For example, in the laboratory session on "Freely Falling Objects with Significant Drag," students determine the drag coefficient by dropping dust balls from different heights and measuring the times they take to fall. Each team enters its results into the distributed database, and the class analyzes the data. After a team enters data into the web-accessed database, all students can obtain the measurements simultaneously. To simplify the analysis, we assume that the database only stores results related to this experiment over a period of several years.

A *database* can be modelled as:

- A collection of entities
- Relationship among entities
- And their attributes

3.2.1 Entity

An entity is a real-world item or concept that exists on its own. In our example, a student, team, lab section, or experiment is an entity. The set of all possible values for an entity, such as all possible students, is the entity type. In an ER model, we diagram an entity type as a rectangle containing the type name, such as *student* (see Figure 3.1).

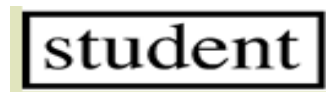


Figure 3.1 ER diagram notation for entity student

An entity instance is a specific value of an entity. For example, John and Laura are students; they are instances of the entity Student while an *entity set* is a set of entities of the same type that share the same properties. Example: set of all persons, companies, trees, holidays

3.2.2 Attribute

Each entity has attributes, or properties that describe the entity. For example, student has properties of his own Student Identification number, name, and grade. A value of an attribute, such as 93 for the grade, is a value of the attribute. Most of the data in a database consists of values of attributes. The set of all possible values of an attribute, such as integers from 0 to 100 for a grade, is the attribute domain. In an ER model, an attribute name appears in an oval that has a line to the corresponding entity box, such as in Figure 3.2.

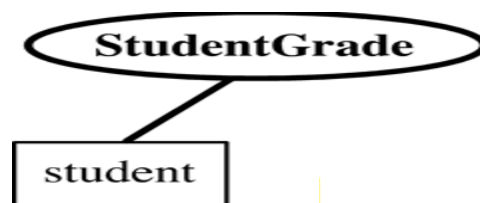


Figure 3.2: ERD notation for an attribute domain (*Student Grade*) of an entity type (*student*)

An attribute can be simple or composite. A simple attribute, such as grade, is one component that is atomic. If we consider the name in two parts, last name and first name, then the name attribute is a composite. A composite attribute, such as "Emmanuel Kwakye", has multiple components, such as "Emmanuel" and "Kwakye"; and each component is atomic or

composite. We illustrate this composite nature in the ER model by branching off the component attributes, such as in Figure 3.3.

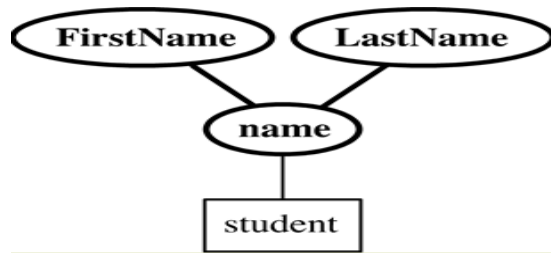


Figure 3.3: ER diagram notation for composite attribute domain, *name*

Another way to classify attributes is either as single-valued or multi-valued. For an entity an attribute, such as *StudentGrade*, usually holds exactly one value, such as 93, and thus is a single-valued attribute. However, two lab assistants might assist in a laboratory section. Consequently, the *LabAssistant* attribute for the entity *LabSection* is multi-valued. A multi-valued attribute has more than one value for a particular entity. We illustrate this situation with a double oval around the lab assistant type, *LabAssistant*



Figure 3.4: ER diagram notation for multi-valued attribute domain, *LabAssistant*

A **derived attribute** can be obtained from other attributes or related entities. For example, the radius of a sphere can be determined from the circumference. We represent the derived attribute with a dotted oval and line, such as in Figure 3.5.

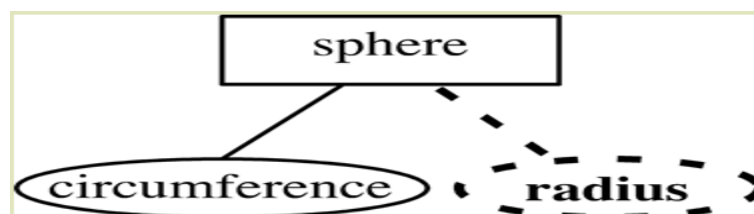


Figure 3.5: ER diagram notation for derived attribute, *radius*

3.2.3 Relationships

A relationship is a crucial part of the design of a database. It is used to establish a connection between a pair of logically related entities. A relationship type is a set of associations among entity types. For example, the *student* entity type is related to the *team* entity type because each student is a member of a team. In this case, a relationship or relationship instance is an ordered pair of a specific student and the student's particular physics team, such as (Emmanuel Kwakye, Phys201F2005A04), where Phys201F2005A04 is Emmanuel's team number.

A diamond is used to illustrate the relationship type in an ER diagram, such as in Figure 3.6. We arrange the diagram so that the relationship reads from left to right, "a student is a member of a team." Alternatively, we can arrange the components from top to bottom.



Figure 3.6 ER diagram notation for relationship type, *MemberOf*

The degree of a relationship type is the number of entity types that participate. Thus, the *LabSecMemberOf* relationship type of Figure 3.6 has degree 2, which we call a *binary relationship type*. To clarify the role that an entity plays in each relationship instance, we can label a connecting edge with a **role name** that indicates the purpose of an entity in a relationship. For example, we can have two *binary relationship* types associating the *student* and *team* types, *TeamMemberOf* and *LeaderOf*. In the former case, a student entity is a member of a team entity; in the latter case, a student can be a leader of a team. We illustrate the situation in Figure 3.7.

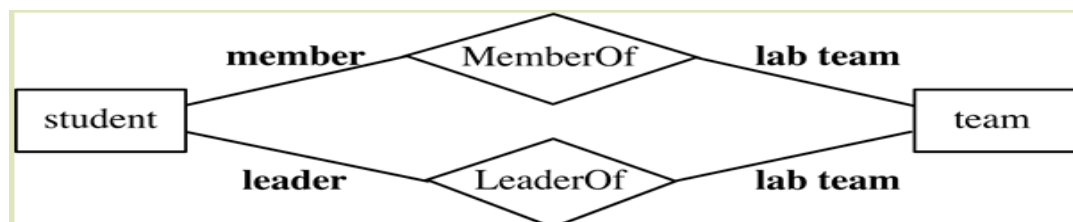


Figure 3.7. ER diagram notation with roles *member*, *leader*, and *lab team*

As Figure 3.8 illustrates, a relationship type can also have attributes. The relationship type *order* connects entities *chemical* and *supplier*. The relationship is many-to-many because each chemical can be from several suppliers and each supplier has a number of chemicals. An order has a purchase date, amount, and total cost as well as the chemical and supplier information. Thus, *order* has attributes *PurchaseDate*, *amount*, and *TotalCost* that we cannot appropriately associate with *chemical* or *supplier*.

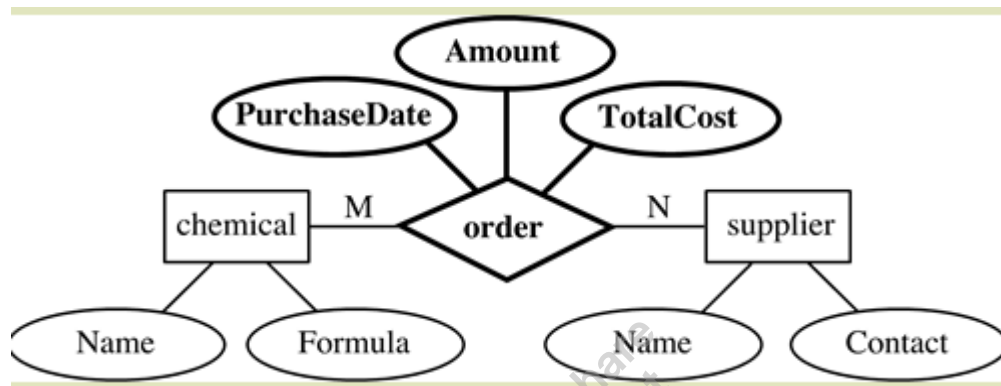


Figure 3.9. Relationship type with attributes

Types of relationship

There are three types of relationships that can exist between entities:

1. One-to-one (1:1)
2. One-to-many(1:m) or many-to-one(m:1)
3. Many-to-many (m:m)

One-to-One (1:1) Relationship

Two entities have a one-to-one relationship if for every instance of the first entity; there is only one instance of the second entity. Consider the example of a university. For one department (say, the department of Social Sciences), there can be only one department head. One faculty member cannot head more than one department. This is an example of a one-to-one relationship.



One-to-Many (1:m) Relationship

Two entities are related in a one-to-many relationship if for every instance of the first entity, there can be zero, one or several instances of the second entity, and for every instance of the second entity, there is exactly one instance of the first entity. For example, a student can major in only one discipline, but many students can register for a discipline. This is an example of a many-to-one relationship.



A many-to-one relationship is the same as a one-to-many relationship. A one-to-many or many-to-one relationship is also referred to as a parent-child or a master-detail relationship. Two entities are related in a many-to-one relationship if for every instance of the first entity, there is exactly one instance of the second entity. For every instance of the second entity, there can be zero, one or several instances of the first entity. For example, an employee can belong to a single department, but a department can have several employees.

Many-to-Many (m: m) Relationship

Two entities are related in a many-to-many relationship when for every instance of the first entity, there can be multiple instances of the second entity, and for every instance of the second entity there can be multiple instances of the first entity. For example, a product can be sold to several customers and a customer can buy several products.

3.3 Cardinalities

Cardinalities constrain participation in relationships is the maximum and minimum number of relationship instances in which an entity instance can participate, or the number of

Database Systems (CE 279)

instances of one entity that can be associated with each instance of another entity. An E-R diagram may also indicate the cardinality of a relationship.

Example:



Cardinality is any pair of non-negative integers (a, b) such that $a \leq b$.

If $a=0$ then entity participation in a relationship is optional

If $a=1$ then entity participation in a relationship is mandatory.

If $b=1$ each instance of the entity is associated at most with a single instance of the relationship

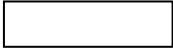
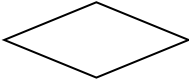



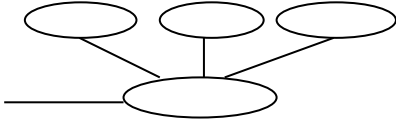
If $b="N"$ then each instance of the entity is associated with an arbitrary number of instances of the relationship.

Summary of ER Notation

<u>Notation</u>	<u>Meaning</u>
	Entity type
	Attribute
	Key attribute
	Derived attribute
	Multivalued attribute
	Composite attribute
	Relationship type
	Total participation
	Many-to-one relationship

Chapter Questions

1. Explain the ER Model and how it models a database
2. List the various elements of Chen's ER model and their respective diagram notation
3. Explain the following terms:
 - i. Entity
 - ii. Entity instance
 - iii. Entity set
 - iv. Attribute
 - v. Attribute domain
 - vi. Derived attribute
 - vii. Multi-valued attribute
 - viii. Relationship instance
 - ix. Cardinality
4. Distinguish between simple attribute and composite attribute
5. Write down the meaning of the following notations for ER diagrams

4. Normalization in Relational Databases

4.1 Chapter objectives

The objectives of this chapter are to:

- Introduce students to normalization in Database
- Normalization processes in Database design
- Understand concurrency control in relational databases

4.2 Normalization

Normalization - is defined as the process of decomposing relations with anomalies to produce smaller, well-organized relations. Thus, in normalization process, a relation with redundancy can be refined by decomposing it or replacing it with smaller relations that contain the same information, but without redundancy.

There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Normalization theory is based on the concepts of normal forms. A relational table is said to be a normal form if it satisfied a certain set of constraints. There are currently five normal forms that have been defined and these are:

- First Normal Form(1NF)
- Second Normal Form(2NF)
- Third Normal Form(3NF)
- Boyce-Codd Normal Form(4NF)
- Domain-key Normal Form (5NF)

4.3 Basic normalization Concepts

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a relational database should be in the third normal form (3NF). Mutual independence in normalization means that no non-key column is dependent upon any combination of the other columns.

The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF.

4.3.1 Functional Dependencies

The concept of functional dependencies is the basis for the first three normal forms. A column, Y, of the relational table R is said to be functionally dependent upon column X of R if and only if each value of X in R is associated with precisely one value of Y at any given time. X and Y may be composite. Saying that column Y is functionally dependent upon X is the same as saying the values of column X identify the values of column Y. If column X is a primary key, then all columns in the relational table R must be functionally dependent upon X.

A short-hand notation for describing a functional dependency is:

$R.x \rightarrow R.y$, which can be read as in the relational table named R, column X functionally determines (identifies) column Y.

Full functional dependence applies to tables with composite keys. Column Y in relational table R is fully functional on X of R if it is functionally dependent on X and not functionally dependent upon any subset of X. Full functional dependence means that when a primary key is composite, made of two or more columns, then the other columns must be identified by the entire key and not just some of the columns that make up the key.

Example of FD constraints:

- ♦ Social Security Number determines employee name

$SSN \rightarrow ENAME$

- ♦ Project Number determines project name and location

$PNUMBER \rightarrow \{PNAME, PLOCATION\}$

- ♦ Employee SSN and project number determines the hours per week that the employee works on the project

$\{SSN, PNUMBER\} \rightarrow HOURS$

4.3.2 First Normal Form (or 1NF)

First Normal Form (1NF) sets the very basic rules for an organized database:

- Eliminate duplicative columns from the same table.
- Create separate tables for each group of related data and identify each row with a unique column (the primary key).

In other words, a relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only

4.3.3 Second Normal Form

Second normal form (2NF) addresses the concept of removing duplicative data:

- Meet all the requirements of the first normal form.
- Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
- Create relationships between these new tables and their predecessors through the use of foreign keys.

In other words, a relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key

4.3.4 Third Normal Form

A relation R is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key. An attribute C is transitively dependent on attribute A if there exist an attribute B such that: $A \rightarrow B$ and $B \rightarrow C$. Note that 3NF is concerned with transitive dependencies which do not involve candidate keys. A 3NF relation with more than one candidate key will clearly have transitive dependencies of the form: $\text{primary_key} \rightarrow \text{other_candidate_key} \rightarrow \text{any_non-key_column}$.

There are two basic requirements for a database to be in third normal form:

- Already meet the requirements of both 1NF and 2NF
- Remove columns that are not fully dependent upon the primary key.

Database Systems (CE 279)

4.4 A sample Database to illustrate the process of normalization

These steps demonstrate the process of normalizing a fictitious contact table.

Unnormalized table:

Contacts						
Name	Company	Address	Phone1	Phone2	Phone3	ZipCode
Joe	ABC	123	5532	2234	3211	12345
Jane	XYZ	456	3421			14454
Chris	PDQ	789	2341	6655		14423

1NF: there should be no repeating groups

Hence the above contact table will look like this after removing repeating groups

Table in 1NF

Contacts					
Id	Name	Company	Address	Phone	ZipCode
1	Joe	ABC	123	5532	12345
1	Joe	ABC	123	2234	12345
1	Joe	ABC	123	3211	12345
2	Jane	XYZ	456	3421	14454
3	Chris	PDQ	789	2341	14423
3	Chris	PDQ	789	6655	14423

2NF: Eliminate Redundant Data

The following two tables demonstrate second normal form:

People				
Id	Name	Company	Address	Zip
1	Joe	ABC	123	12345
2	Jane	XYZ	456	14454
3	Chris	PDQ	789	14423

PhoneNumbers		
PhoneID	Id	Phone
1	1	5532
2	1	2234
3	1	3211
4	2	3421
5	3	2341
6	3	6655

3NF: No dependencies on non-key attributes

PhoneNumbers		
PhoneID	Id	Phone
1	1	5532
2	1	2234
3	1	3211
4	2	3421
5	3	2341
6	3	6655

People		
Id	Name	AddressID
1	Joe	1
2	Jane	2
3	Chris	3

Address			
AddressID	Company	Address	Zip
1	ABC	123	12345
2	XYZ	456	14454
3	PDQ	789	14423

Summary

The process for transforming a 1NF table to 2NF is:

- Identify any determinants other than the composite key, and the columns they determine.
- Create and name a new table for each determinant and the unique columns it determines.
- Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
- Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
- The original table may be renamed to maintain semantic meaning.

The process of transforming a table into 3NF is:

- Identify any determinants, other the primary key, and the columns they determine.
- Create and name a new table for each determinant and the unique columns it determines.
- Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
- Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
- The original table may be renamed to maintain semantic meaning.

4.5 Conclusion on Database Normalization

It is generally a good idea to make sure that your data at least conforms to Second Normal Form. Our goal is to avoid data redundancy to prevent corruption and make the best possible use of storage. Make sure that the same value is not stored in more than one place. With data in multiple locations we have to perform multiple updates when data needs to be changed, and corruption can begin to creep into the database.

Third Normal Form removed even more data redundancy, but at the cost of simplicity and performance.

4.6 Databases Transactions

A database transaction comprises a unit of work performed within a database management system (or similar system) against a database and treated in a coherent and reliable way independent of other transactions.

Transactions in a database environment have two main purposes:

- To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
- To provide isolation between programs accessing a database concurrently. Without isolation the programs' outcomes are possibly erroneous.

A database transaction, by definition, must be atomic, consistent, isolated and durable. Database practitioners often refer to these properties of database transactions using the acronym **ACID**.

4.6.1 Transaction ACID Rules

Every database transaction must obey the following rules:

Atomicity - states that database modifications must follow an “all or nothing” rule. Each transaction is said to be “atomic.” If one part of the transaction fails, the entire transaction fails.

Consistency - states that only valid data will be written to the database. If, for some reason, a transaction is executed that violates the database's consistency rules, the entire transaction will be rolled back, and the database will be restored to a state consistent with those rules.

On the other hand, if a transaction successfully executes, it will take the database from one state that is consistent with the rules to another state that is also consistent with the rules.

Database Systems (CE 279)

Isolation - Transactions cannot interfere with each other. Moreover, an incomplete transaction is not visible to another transaction. Providing isolation is the main goal of concurrency control.

Durability - ensures that any transaction committed to the database will not be lost. Durability is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures.

4.6.2 Transactions Example & ACID Properties

Consider a money “transfer” transaction in a bank that transfers money from one account. We want to transfer \$100 from account number 1001 to account number 1005. If account 1001 does not have at least \$100, the transaction does nothing to the database
Accounts Table Before & After Transfer, (ActNum is the primary key)

Before transaction

AccNum	Firstname	Surname	Balance (\$)
1001	Peace	Dadzie	700.00
1002	Grace	Pomaa	1000.00
1003	George	Antwi	400.00
1004	Papa	Kojo	800.00
1005	Doris	Sekyi	500.00

After transaction

AccNum	Firstname	Surname	Balance (\$)
1001	Peace	Dadzie	700.00
1002	Grace	Pomaa	1000.00
1003	George	Antwi	400.00
1004	Papa	Kojo	800.00
1005	Doris	Sekyi	600.00

First, the transaction needs to see if there is at least \$100 in the account.

START TRANSACTION;

SELECT AccNum, Balance

FROM Accounts

WHERE AccNum = 1001;

If the result is less than 100, then the transaction should be aborted by executing

ROLLBACK;

Otherwise, the transfer can proceed by executing the statements:

UPDATE Accounts

SET Balance = Balance - 100

WHERE AccNum = 1001;

Database Systems (CE 279)

UPDATE Accounts

SET Balance = Balance + 100

WHERE AccNum = 1005;

COMMIT;

Transaction Properties

The transfer transaction has three parts. It first checks the account balance. Then if the balance is less than \$100, the transaction is aborted. Otherwise, the money is transferred, and the transaction committed. ACID Guarantees:

Atomicity: The amount withdrawn from one account and its deposit to the other either succeeds or fails but there is no partial execution.

Consistency: No constraints will be violated (only constraint is the primary key column AccNum which is not impacted by the transaction).

Isolation: The transfer transaction will correctly move \$100 from one account to another even in the presence of other simultaneously executing transactions that may be interested in modifying the same accounts.

Durability: Once the transaction has successfully executed, its effects will be reflected in the database, i.e., they will become permanent.

4.7 Concurrency Control in Relational Databases

Concurrency control in Database management systems ensures that database transactions are performed concurrently without violating the data integrity of the respective databases.

Concurrency control deals with the issues involved with allowing multiple people simultaneous access to shared entities (objects, data records, or some other representation).

Concurrency control is the activity of coordinating the actions of transactions that operate, simultaneously or in parallel to access shared data.

Thus, concurrency control is an essential element for correctness in any system where two database transactions or more can access the same data concurrently, e.g., virtually in any general-purpose database system.

4.7.1 Reason for Concurrency Control

If concurrent transactions are allowed in an uncontrolled manner, some unexpected result may occur. Here are some typical examples:

- **Lost update problem:** A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.
- **The dirty read problem:** Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.
- **The incorrect summary problem:** While one transaction takes a summary over values of a repeated data-item, a second transaction updated some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether a certain update result has been included in the summary or not.

4.7.2 Concurrency control mechanisms

The main categories of concurrency control mechanisms are:

Optimistic - With multi-user systems it is quite common to be in a situation where collisions are infrequent. With optimistic concurrency control, you accept the fact that collisions occur infrequently, and instead of trying to prevent them you simply choose to detect them and then resolve the collision when it does occur.

Database Systems (CE 279)

Pessimistic - is an approach where an entity is locked in the database for the entire time that it is in application memory. For example, a lock either limits or prevents other users from working with the entity in the database. A write lock indicates that the holder of the lock intends to update the entity and disallows anyone from reading, updating, or deleting the entity.

The advantages of pessimistic locking are that it is easy to implement and guarantees that your changes to the database are made consistently and safely. The primary disadvantage is that this approach isn't scalable. When a system has many users, or when the transactions involve a greater number of entities, or when transactions are long lived, then the chance of having to wait for a lock to be released increases. Therefore, this limits the practical number of simultaneous users that your system can support.

Overly optimistic - With the strategy you neither try to avoid nor detect collisions, assuming that they will never occur. This strategy is appropriate for single user systems, systems where the system of record is guaranteed to be accessed by only one user or system process at a time, or read-only tables. These situations do occur. It is important to recognize that this strategy is completely inappropriate for multi-user systems.

Chapter Questions

1. Explain the concept of Normalization and discuss the goal it seeks to achieve
2. Discuss the principle of functional dependencies and full functional dependence
3. List the process of transforming a table into the following:
 - i. First Normal Form
 - ii. Second Normal Form
 - iii. Third Normal Form
4. Define the following:
 - i. Database transaction
 - ii. Concurrency control
5. What does the acronym ACID stand for?
6. Discuss the various transaction ACID rules
7. Discuss the reasons for ensuring concurrency control in database transactions
8. Explain the following concurrency control mechanisms and state at least one scenario each in which a mechanism is applicable:
 - i. Optimistic
 - ii. Pessimistic
 - iii. Overly optimistic

5. Database-Design Process

5.3 Chapter objectives

The objectives of this chapter are to:

- Introduce students to various fact-finding techniques used in Database Design
- Introduce students to the steps taken in Database Design
- Explain these processes in Database design in a case study business or organization

5.4 Facts-finding Techniques in Database Design

To study any system, the analyst needs to do collect facts and all relevant information. The facts when expressed in quantitative form are termed as data. The success of any project is depended upon the accuracy of available data. Accurate information can be collected with help of certain methods/ techniques. These specific methods for finding information of the system are termed as fact finding techniques. Formal process of using techniques such as interviews and questionnaires to collect facts about systems, requirements, and preferences. This is the first step we must do before designing a Database, Database Application or any Software Application. It is also called information gathering or data collection.

5.4.1 When to use Fact-Finding Techniques

There are many occasions for fact-finding during the database system development lifecycle. However, fact-finding is particularly crucial to the early stages of the lifecycle including the database planning, system definition, and requirements collection and analysis stages. It is during these early stages that the database developer captures the essential facts necessary to build the required database. Fact-finding is also used during database design and the later stages of the lifecycle, but to a lesser extent. For example, during physical database design, fact-finding becomes technical as the database developer attempts to learn more about the DBMS selected for the database system. Also, during the final stage, operational maintenance, fact-finding is used to determine whether a system requires tuning to improve performance or further development to include new requirements.

5.4.2 Fact-Finding Techniques

A database developer normally uses several fact-finding techniques during a single database project. There are five commonly used fact-finding techniques:

- Examining documentation
- Interviewing
- Observing the enterprise in operation
- Research
- Questionnaires.

In the following sections we describe these fact-finding techniques and identify the advantages and disadvantages of each.

Examining Documentation

Examining documentation can be useful when we are trying to gain some insight as to how the need for a database arose. We may also find that documentation can help to provide information on the part of the enterprise associated with the problem. If the problem relates to the current system, there should be documentation associated with that system. By examining documents, forms, reports, and files associated with the current system, we can quickly gain some understanding of the system.

Interviewing

Interviewing is the most commonly used, and normally most useful, fact-finding technique. We can interview to collect information from individuals face-to-face. There can be several objectives to using interviewing, such as finding out facts, verifying facts, clarifying facts, generating enthusiasm, getting the end-user involved, identifying requirements, and gathering ideas and opinions. However, using the interviewing technique requires good communication skills for dealing effectively with people who have different values, priorities, opinions, motivations, and personalities. As with other fact-finding techniques, interviewing is not always the best method for all situations.

Observing the Enterprise in Operation

Observation is one of the most effective fact-finding techniques for understanding a system. With this technique, it is possible to either participate in, or watch, a person performs activities to learn about the system. This technique is particularly useful when the validity of data collected through other methods is in question or when the complexity of certain aspects of the system prevents a clear explanation by the end-users. As with the other fact-finding techniques, successful observation requires preparation.

Research

A useful fact-finding technique is to research the application and problem. Computer trade journals, reference books, and the Internet (including user groups and bulletin boards) are good sources of information. They can provide information on how others have solved similar problems, plus whether or not software packages exist to solve or even partially solve the problem.

Questionnaires

Another fact-finding technique is to conduct surveys through questionnaires. Questionnaires are special-purpose documents that allow facts to be gathered from a large number of people while maintaining some control over their responses. When dealing with a large audience, no other fact-finding technique can tabulate the same facts as efficiently.

5.5 Design Process

It is a bad idea to attempt to design a database without undertaking a complete database-design process. Many database problems are caused by poor database design, and partially following the design process is just about as bad as not using it at all. **An incomplete design is a poor design.**

An important point to keep in mind is that the level of structural integrity and data integrity in your database is directly proportional to how thoroughly you follow the design process. The less time you spend on the design process, the greater the risk you run of encountering problems with the database. Although thoroughly following the database-design process may

Database Systems (CE 279)

not eliminate all of the problems you may encounter when designing a database, it will greatly help to minimize them.

Understanding how to design a relational database isn't quite as hard as understanding the universe; in fact, it's much easier. It is important for you, however, to have an overall idea of the way the database-design process works, and a general idea of the steps involved within the process.

Database designs also include ER (Entity-relationship model) diagrams. An ER diagram is a diagram that helps to design databases in an efficient way. Within the relational model the final step can generally be broken down into two further steps that of determining the grouping of information within the system, generally determining what are the basic objects about which information is being stored, and then determining the relationships between these groups of information, or objects.

5.6 Steps involved in Database Design Processes

1. Define a mission statement and mission objectives for the database.

- The mission statement defines the purpose of the database, and
- The mission objectives define the tasks that are to be performed by users against the data in the database.

2. Analyze the current database.

- You identify the organization's data requirements by reviewing the way your organization currently collects and presents its data and
- By conducting interviews with users and management to determine how they use the database on a daily basis.

3. Create the data structures.

- You establish tables by identifying the subjects that the database will track.
- You associate each table with fields that represent distinct characteristics of the table's subject, and you designate a particular field (or group of fields) as the primary key.

Database Systems (CE 279)

- You then establish **field specifications** for every field in the table.
- 4. **Determine and establish table relationships.**
 - You identify relationships that exist between the tables in the database and establish the logical connection for each relationship using primary keys and foreign keys or by using linking tables.
 - You set the appropriate characteristics for each relationship.
- 5. **Determine and define business rules.**
 - You conduct interviews with users and management to identify constraints that must be imposed upon the data in the database. The manner in which your organization views and uses its data typically determines the types of constraints you must impose on the database.
 - **You then declare these constraints as business rules, and they will serve to establish various levels of data integrity.**
- 6. **Determine and establish views.**
 - You interview users and management to identify the various ways they work with the data in the database. When your interviews are complete, you establish views as appropriate.
 - You define each view using the appropriate tables and fields, and you establish criteria for those views that must display a limited or finite set of records.
- 7. **Review data integrity.**

This phase involves four steps:

- First, you review each table to ensure that it meets proper design criteria.
- Second, you review and check all field specifications.
- Third, you test the validity of each relationship.
- Fourth, you review and confirm the business rules.

Database Systems (CE 279)

In simple terms follow these processes to easily design a database

1. **Determine the purpose of your database** - This helps prepare you for the remaining steps.
2. **Find and organize the information required** - Gather all of the types of information you might want to record in the database, such as product name and order number.
3. **Divide the information into tables** - Divide your information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.
4. **Turn information items into columns** - Decide what information you want to store in each table. Each item becomes a field and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.
5. **Specify primary keys** - Choose each table's primary key. The primary key is a column that is used to uniquely identify each row. An example might be Product ID or Order ID.
6. **Set up the table relationships** - Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. **Refine your design** - Analyze your design for errors. Create the tables and add a few records of sample data. See if you can get the results you want from your tables. Make adjustments to the design, as needed.
8. **Apply the normalization rules** - Apply the data normalization rules to see if your tables are structured correctly. Make adjustments to the tables

Chapter Questions

1. Explain why fact-finding precedes database design
2. Discuss the various fact-finding techniques used in database design
3. List at least three advantages of the various fact-finding techniques discussed in (2) above
4. List at least three disadvantages of the various fact-finding techniques discussed in (2) above
5. Distinguish between the following terms:
 - i. Open-ended questions and Close-ended questions
 - ii. Structured and Unstructured interviews
 - iii. Free-format questions and Fixed-format questions
6. Discuss the steps involved in database design process



6. Database language SQL

6.1 Chapter objectives

The objectives of this chapter are to introduce students to:

- SQL commands used in database in their various categories
- Create database objects (tables, views, indexes, sequence and triggers) using SQL commands
- Constraints and triggers are in SQL and how to create them

6.2 SQL

SQL (Structured Query Language) is a computer language aimed to store, manipulate, and query data stored in relational databases. Standards for SQL exist. However, the SQL that can be used on each one of the major RDBMS today is in different flavours. This is due to two reasons: 1) the SQL command standard is fairly complex, and it is not practical to implement the entire standard, and 2) each database vendor needs a way to differentiate its product from others. SQL is very popular and offers great flexibility to users by supporting distributed databases, i.e. databases that can be run on several computer networks at a time. SQL-based applications have become increasingly affordable for the regular user. This is due to the introduction of various open-source SQL database solutions such as MySQL, PostgreSQL, SQLite, Firebird, and many more.

6.3 SQL Language Elements

The SQL language is based on several elements. For the convenience of SQL developers all necessary language commands in the corresponding database management systems are usually executed through a specific SQL command-line interface (CLI).

- **Clauses** - the clauses are components of the statements and the queries
- **Expressions** - the expressions can produce scalar values or tables, which consist of columns and rows of data

Database Systems (CE 279)

- **Predicates** - they specify conditions, which are used to limit the effects of the statements and the queries, or to change the program flow
- **Queries** - a query will retrieve data, based on a given criterion
- **Statements** - with the statements one can control transactions, program flow, connections, sessions, or diagnostics. In database systems the SQL statements are used for sending queries from a client program to a server where the databases are stored. In response, the server processes the SQL statements and returns replies to the client program.

6.4 SQL Queries

The most common operation in SQL is the query, which is performed with the declarative **SELECT** statement. **SELECT** retrieves data from one or more tables, or expressions. Standard **SELECT** statements have no persistent effects on the database. Some non-standard implementations of **SELECT** can have persistent effects, such as the **SELECT INTO** syntax that exists in some databases.

Queries allow the user to describe desired data, leaving the database management system (DBMS) responsible for planning, optimizing, and performing the physical operations necessary to produce that result as it chooses.

A query includes a list of columns to be included in the final result immediately following the **SELECT** keyword. An asterisk ("*****") can also be used to specify that the query should return all columns of the queried tables. **SELECT** is the most complex statement in SQL, with optional keywords and clauses that include:

- The **FROM** clause which indicates the table(s) from which data is to be retrieved. The **FROM** clause can include optional **JOIN** sub-clauses to specify the rules for joining tables.
- The **WHERE** clause includes a comparison predicate, which restricts the rows returned by the query. The **WHERE** clause eliminates all rows from the result set for which the comparison predicate does not evaluate to True.

- The **GROUP BY** clause is used to project rows having common values into a smaller set of rows. GROUP BY is often used in conjunction with SQL aggregation functions or to eliminate duplicate rows from a result set. The WHERE clause is applied before the GROUP BY clause.
- The **HAVING** clause includes a predicate used to filter rows resulting from the GROUP BY clause. Because it acts on the results of the GROUP BY clause, aggregation functions can be used in the HAVING clause predicate.
- The **ORDER BY** clause identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending). Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

An SQL query example

SELECT (field(s) or attribute(s))
FROM (table(s))
WHERE (condition statement)
ORDER BY (fields, ASCENDING, DESCENDING)

6.5 SQL Command Categories

The SQL Language as defined by ANSI is subdivided into several different sections. SQL commands fall into different categories depending on what function they perform.

- **Data Definition Language or DDL** – This consists of those commands in SQL that directly create database objects such as tables, indexes and views.
- **Data Manipulation Language or DML** – Consists of those commands which operate on the data in the database. This includes statements which add data to the tables as well as those statements which are used to query the database.
- **Data Control Language or DCL** – This is generally used to refer to those SQL commands which are used for data security. These are commands that are used to

determine whether a user is allowed to carry out a particular operation or not on data in a database.

- **Data Query Language or DQL**- This is basically made up of one command (SELECT). It is used together with several other SQL commands to retrieve data.
- **Transaction Control Language or TCL** - a subset of SQL, used to control transactional processing in a database. A transaction is logical unit of work that comprises one or more SQL statements, usually a group of DML statements.

6.5.1 SQL Data Definition Language

The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands are primarily used by database administrators during the setup and removal phases of a database project.

❖ **CREATE** - command used to establish database or database objects.

Examples of an SQL CREATE commands:

- CREATE DATABASE employees ; Creates an empty database named "employees"
- CREATE TABLE personal_info (first_name char(20) not null, last_name char(20)), Establishes a table titled "personal_info" in the current database.

❖ **USE**- it allows to specify the database you wish to work with within your DBMS. For example, if you want to work within the employees you issue the following SQL command:

USE employees;

❖ **ALTER**

The ALTER command allows you to make changes to the structure of a table without deleting and recreating it. Take a look at the following command:

ALTER TABLE personal_info

ADD salary money null

This example adds a new attribute to the personal_info table -- an employee's salary. The "money" argument specifies that an employee's salary will be stored using a dollars and cents format.

- ❖ **DROP**- it allows us to remove entire database objects from our DBMS. For example, if we want to permanently remove the personal_info table that we created, we'd use the following command:

```
DROP TABLE personal_info;
```

DROP command removes entire data structures from your database and must be used with care. If you want to remove individual records, use the DELETE command of the Data Manipulation Language.

- ❖ **TRUNCATE**

With the TRUNCATE statement, you can delete all the content in the table, but keep the actual table intact and ready for further use

An example of an SQL TRUNCATE

```
TRUNCATE TABLE phonebook;
```

6.5.2 SQL Data Manipulation Language

The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database.

- ❖ **INSERT**

The INSERT command in SQL is used to add records to an existing table. Returning to the personal_info example from the previous section, imagine that our HR department needs to add a new employee to their database. They could use a command similar to the one shown below:

```
INSERT INTO personal_info  
values ('bart','simpson', 12345, 45000)
```

To insert multiple Rows into the table

```
INSERT INTO  
table_name (column1,  
VALUES (value1,value2,...), (value1,value2,...)
```

Example:

```
INSERT INTO Employee (Firstname , Lastname ,Title)  
VALUES (Milcah ' , 'Agangiba', 'Director'),('Dora', 'Asare ' , 'Engineer')
```

To view what values were modified

```
INSERT INTO Employee  
last_name,first_name,salary  
OUTPUT  
inserted.first_name,inserted.last_name ,  
inserted.salary  
VALUES ('Mart','Martha',3100), ('Faa ' , 'Grace', 60)
```

❖ SELECT

The SELECT allows database users to retrieve the specific information they desire from an operational database. The command shown below retrieves all of the information contained within the personal_info table. Note that the asterisk is used as a wildcard in SQL. This literally means "Select everything from the personal_info table."

```
SELECT * FROM personal_info
```

Alternatively, users may want to limit the attributes that are retrieved from the database. The following SQL command would retrieve only last names of all employees in the company:

```
SELECT last_name  
FROM personal_info;
```

Finally, the **WHERE** clause can be used to limit the records that are retrieved to those that meet specified criteria. For example, the following command retrieves all data contained within `personal_info` for records that have a salary value greater than \$50,000:

```
SELECT *  
FROM personal_info  
WHERE salary > $50000;
```

- ❖ **UPDATE-** command used to modify information contained within a table, either in bulk or individually. The following SQL command could be used to increase all employees' salaries by 30%:

```
UPDATE personal_info  
SET salary = salary * 1.03;
```

- ❖ **DELETE-** the syntax of this command is similar to that of the other DML commands. The **DELETE** command with a **WHERE** clause can be used to remove a record from a table:

```
DELETE FROM personal_info  
WHERE employee_id = 12345;
```

Basic Mathematical Functions

To count the number of records in a table within the database the *COUNT* function is used.

Count function ignores null values. The following command

```
SELECT COUNT (*)  
FROM  
Table_name
```

To find the maximum value in the field the **MAX** function is used. Syntax:

```
SELECT MAX ()  
column_name  
FROM  
Table_name
```

Database Systems (CE 279)

To find the minimum value in the field, the MIN function is used. Syntax:

```
SELECT MIN ()
column_name
FROM
Table_name
```

To find the average value in the field the AVG function is used. Syntax:

```
SELECT AVG ()
column_name
FROM
Table_name
```

To find the total value in the field the SUM function is used. Syntax:

```
SELECT SUM ()
column_name
FROM
Table_name
```

DATE FUNCTIONS

Name	Description
<u>ADDDATE ()</u>	Add time values (intervals) to a date value
<u>ADDTIME ()</u>	Add time
<u>CONVERT_TZ ()</u>	Convert from one time zone to another
<u>CURDATE ()</u>	Return the current date
<u>CURRENT_DATE ()</u> , <u>CURRENT_DATE</u>	Synonyms for CURDATE()
<u>CURRENT_TIME ()</u> , <u>CURRENT_TIME</u>	Synonyms for CURTIME()
<u>CURRENT_TIMESTAMP ()</u> , <u>CURRENT_TIMESTAMP</u>	Synonyms for NOW()
<u>CURTIME ()</u>	Return the current time
<u>DATE ()</u>	Extract the date part of a date or datetime expression
<u>DATE_ADD ()</u>	Add time values (intervals) to a date value
<u>DATE_FORMAT ()</u>	Format date as specified
<u>DATE_SUB ()</u>	Subtract a time value (interval) from a date
<u>DATEDIFF ()</u>	Subtract two dates
<u>DAY ()</u>	Synonym for DAYOFMONTH()
<u>DAYNAME ()</u>	Return the name of the weekday
<u>DAYOFMONTH ()</u>	Return the day of the month (0-31)
<u>DAYOFWEEK ()</u>	Return the weekday index of the argument
<u>DAYOFYEAR ()</u>	Return the day of the year (1-366)

Examples: displays current date and time

```
SELECT GETDATE()
```

```
SELECT DAY (GETDATE()) displays day
```

```
SELECT MONTH (GETDATE()) displays  
month
```

```
SELECT YEAR (GETDATE()) displays year
```

```
SELECT DATENAME (WEEKDAY, GETDATE()) displays the day of the week
```

Examples: displays current date and time

```
SELECT DATEADD (DAY, 15,GETDATE())
```

Difference between 2 dates

```
SELECT DATEDIFF(DAY, '2019/10/25', '2019/09/15') AS REMAINING
```

```
SELECT DATEDIFF(YEAR, '2019/10/25', '2019/09/15') AS REMAINING
```

```
SELECT DATEDIFF(MONTH, '2019/10/25', '2019/09/15') AS REMAINING
```

6.5.3 SQL Data Control

SQL allows the user to define the access each of the table users can have to the actual table.

GRANT - with the GRANT statement, you can authorize users to modify the selected table

An example of an SQL GRANT

```
GRANT ALL PRIVILEGES ON database_name TO database_user;
```

REVOKE - with the REVOKE statement you can remove all privileges, previously granted to a user.

An example of an SQL REVOKE

```
REVOKE ALL PRIVILEGES ON database_name TO database_user;
```

6.6 SQL Joins: Data from multiple tables

The JOIN statements in SQL allows to combine data in multiple tables to efficiently process large quantities of data. A join is used to combine columns from two or more tables into a single result set. To join data from two tables you write the names of two tables in the FROM clause along with JOIN keyword and an ON phrase that specifies the join condition.

They are different types of SQL joins used in and these are:

- Inner joins and
- Outer joins

6.6.1 Inner joins

Here are two tables – Employee table and Department table. This is how the database tables and data look like. These tables will be used in samples below.

Employee Table		Department Table	
LastName	DepartmentID	DepartmentID	DepartmentName
Rafferty	31	31	Sales
Jones	33	33	Engineering
Steinberg	33	34	Clerical
Robinson	34	35	Marketing
Smith	34		
John	NULL		

Note: The "Marketing" Department currently has no listed employees. Also, employee "John" has not been assigned to any Department yet.

An inner join is the most common join operation used in applications and can be regarded as the default join-type. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. The result of the join can be defined as the outcome of first taking the

Cartesian product (or Cross join) of all records in the tables (combining every record in table A with every record in table B)—then return all records which satisfy the join predicate. Actual SQL implementations normally use other approaches like a hash join or a sort-merge join where possible, since computing the Cartesian product is very inefficient.

SQL specifies two different syntactical ways to express joins: "explicit join notation" and "implicit join notation".

The "explicit join notation" uses the JOIN keyword to specify the table to join, and the ON keyword to specify the predicates for the join, as in the following example:

```
SELECT *  
FROM employee INNER JOIN department  
ON employee.DepartmentID = department.DepartmentID;
```

The "implicit join notation" simply lists the tables for joining (in the FROM clause of the SELECT statement), using commas to separate them. Thus, it specifies a cross join, and

the WHERE clause may apply additional filter-predicates (which function comparably to the join-predicates in the explicit notation).

The following example shows a query which is equivalent to the one from the previous examples, but this time written using the implicit join notation:

```
SELECT *  
FROM employee, department  
WHERE employee.DepartmentID = department.DepartmentID;
```

The queries given in the examples above will join the Employee and Department tables using the DepartmentID column of both tables. Where the DepartmentID of these tables match (i.e. the join-predicate is satisfied), the query will combine the *LastName*, *DepartmentID* and *DepartmentName* columns from the two tables into a result row. Where the DepartmentID does not match, no result row is generated.

Thus, the result of the execution of either of the two queries above will be:

Database Systems (CE 279)

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31

Note: Take special care when joining tables on columns that can contain NULL values, since NULL will never match any other value (not even NULL itself), unless the join condition explicitly uses the IS NULL or IS NOT NULL predicates.

Notice that the employee "John" and the department "Marketing" do not appear in the query execution results. Neither of these has any matching records in the respective other table: "John" has no associated department, and no employee has the department ID 35. Thus, no information on John or on Marketing appears in the joined table. There exist different types of inner joins namely: Equi-Join, Natural Join and Cross Join

6.6.2 Outer Joins

An **outer join** does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table(s) one retains the rows from (left, right, or both).

(In this case *left* and *right* refer to the two sides of the JOIN keyword.)

No implicit join-notation for outer joins exists in standard SQL.

There are several types of outer joins. These include left outer join, right outer join and full outer joins.

Database Systems (CE 279)

6.7 SQL Data types

Field Type	Description	Storage
Bigint	New to SQL Server 2000. Can hold numbers ranging from 2^{63} to 2^{63} .	8 bytes.
Binary	Holds from 1 to 8,000 bytes of fixed-length binary data.	Whatever is in the column, plus 4 additional bytes
Bit	Can hold a value of either 1 or 0. Nulls not allowed.	8-bit fields take up 1 byte of data.
Char	Holds from 1 to 8,000 bytes of fixed-length non-Unicode characters.	The number of bytes corresponds to the length of the field (regardless of what is stored in it).
DateTime	Holds valid dates from January 1, 1753 to December 31, 9999.	4 bytes.
Decimal	Used for numbers with fixed precision and scale. When maximum precision is used, values can range from 10^{38} 1 to 10^{38} 1. Scale must be less than or equal to the precision.	Depends on the precision.
Float	Can hold positive and negative numbers from $1.79E + 308$ to $1.79E + 308$. It offers binary precision up to 15 digits.	8 bytes.
Image	Consists of linked data pages of binary data. It can contain up to 2,147,483,647 bytes of binary data.	Depends on what is stored in it.
Int	Can store whole numbers from, 2,147,483,648 to 2,147,483,647.	4 bytes.
Money	Can store decimal data ranging from 2^{63} to 2^{63} , scaled to four digits of precision. It offers accuracy to 1/10,000 of a monetary unit.	8 bytes.
NChar	Can contain from 1 to 4,000 Unicode characters.	Twice the amount of bytes of Char. Corresponds to the length of the field (regardless of what is stored in it).
NText	Can hold data up to 1,073,741,823 Unicode characters.	Each character takes 2 bytes of storage.
Numeric	Used for numbers with fixed precision and scale. When maximum precision Values can range from 10^{38} to 1 to $10^{38} + 1$.	Depends on the precision.

Database Systems (CE 279)

NVarChar NVarChar(MAX)	Can contain from 1 to 4,000 Unicode characters.	2 bytes per character stored.
Real	A smaller version of float. Contains a single-precision floating-point number from $3.40E + 38$ to $3.40E - 38$.	4 bytes.
SmallDateTime	Consists of two 2-byte integers. Can store dates only between 1/1/1900 and 6/6/2079.	4 bytes.
SmallInt	A smaller version of int. Can store values between 32,768 and 32,767.	2 bytes.
SmallMoney	A smaller version of money. Can store decimal data scaled to four digits of precision. Can store values from 214,748.3648 to +214,748.3647.	4 bytes.
SQL_Variant	New to SQL 2000. Can store int, binary, and char values. Is a very inefficient data type.	Varies.
Text	Stores up to 2,147,483,647 characters of non-Unicode data.	1 byte for each character of storage.
TimeStamp	Generates a unique binary value that SQL Server automatically creates when a row is inserted and that SQL Server updates every time that the row is edited.	8 bytes.
TinyInt	Stores whole numbers from 0 to 255.	1 byte.
UniqueIdentifier	A globally unique identifier (GUID) that is automatically generated when the NEWID() function is used.	16 bytes.
VarBinary VarBinary(MAX)	Can hold variable-length binary data from 1 to 8000 bytes.	Varies from 1 to 8000 bytes.
VarChar VarChar(MAX)	A variable-length string that can hold 1 to 8,000 non-Unicode characters.	1 byte per character stored.

6.8 Database Constraints and Triggers

6.8.1 Database Constraints

A constraint in a database is a restriction placed at column or table level, a constraint ensures that your data meets certain data integrity rules.

There are 3 types of constraints in database at a very high level.

1. Entity constraints

Entity constraints are all about individual rows. This kind of constraints doesn't really care about a column value; it's interested in a particular row, and would best be exemplified by a constraint that requires every row to have a unique value for a column or combination of Columns.

Entity constraints are:

PRIMARY KEY and UNIQUE

2. Domain constraints

Domain constraints are column level constraints. Either these constraints are based on one or more columns. This is basically to ensure that a particular column or set of columns meets particular criteria. For example, if we want to confine the Salary column only to values that are greater than zero, that would be a domain constraint. While any Employee that had a Salary that didn't meet the constraint would be rejected, we're actually enforcing integrity to make sure that entire column meets the constraint.

Domain constraints are:

CHECK and DEFAULTS

3. Referential integrity constraints

Referential integrity constraints are created when a value in one column must match the value in another column — in either the same table or a different table.

Referential Constrains are:

3.1) FOREIGN KEY

Constraint details:

1) PRIMARY KEY

A primary key constraint ensures uniqueness within the columns declared as being part of that primary key, and that unique value serves as an identifier for each row in that table. There are two ways to create a primary key, either in your CREATE TABLE command or with an ALTER TABLE command.

The below table is to hold Employee information.

CREATE TABLE Employee

```
(  
    Employee_id    int    IDENTITY NOT NULL,  
    EmployeeName   varchar(30) NOT NULL,  
    Address        varchar(30) NOT NULL,  
    City           varchar(20) NOT NULL,  
    Contact        varchar(25) NOT NULL,  
    Phone          char(15)  NOT NULL  
);
```

Here we want to define primary key on Employee_id.

Now if you want to define primary key on Employee_id column then

Also, you can use ALTER statement to define the constraint.

ALTER TABLE Employee

ADD CONSTRAINT PK_Employee_id

PRIMARY KEY (Employee_id);

2) UNIQUE KEY

UNIQUE Key constraints are essentially close to primary keys in that they require a unique value throughout the named column (or combination of columns) in the table. UNIQUE Key constraints referred to as alternate keys. The major differences are that they are not considered to be the unique identifier of a record in that table (even though you could effectively use it that way) and that you can have more than one UNIQUE constraint (remember that you can only have one primary key per table). Once you establish a UNIQUE Key constraint, every value in the named columns must be unique. If you go and update or insert a row with a value that already exists in a column with a unique constraint, SQL Server will raise an error and reject the record.

UNIQUE Key constraint does allow NULL value.

CREATE TABLE Employee

```
(  
    Employee_id    int    IDENTITY NOT NULL,  
    EmployeeName   varchar(30) NOT NULL,  
    Address        varchar(30) NOT NULL,  
    City           varchar(20) NOT NULL,  
    State          char(2)   NOT NULL,  
    Contact        varchar(25) NOT NULL,  
    Phone          char(15)  NOT NULL UNIQUE  
    PRIMARY KEY (Employee_id )  
);
```

Also, you can use ALTER statement to define the constraint.

ALTER TABLE Employees

ADD CONSTRAINT AK_EmployeePhone

UNIQUE (Phone)

3) CHECK

Check Constraint allows you to limit the types of data that users may insert in a database. They go beyond data types and allow you to define the specific values that may be included in a column.

You can create a CHECK constraint as part of the table definition when you create a table. If a table already exists, you can add a CHECK constraint. Tables and columns can contain multiple CHECK constraints.

If a CHECK constraint already exists, you can modify or delete it. The CHECK constraint is used to limit the value range that can be placed in a column.

```
CREATE TABLE Employee
```

```
(  
    Employee_id    int    IDENTITY NOT NULL,  
    EmployeeName   varchar(30)    NOT NULL,  
    Address        varchar(30)    NOT NULL,  
    City           varchar(20)    NOT NULL,  
    State          char(2)        NOT NULL,  
    Contact        varchar(25)    NOT NULL,  
    Phone          char(15)       NOT NULL UNIQUE  
    CHECK (Employee_id>0 )  
);
```

If you don't want to have this Check Field as part of table script then you can use ALTER statement to define the constraint.

```
ALTER TABLE Employees
```

```
ADD CONSTRAINT CK_EmployeePhone
```

```
CHECK (Employee_id>0 )
```

4) DEFAULT

Default Constraint allows you to specify a value that the database will use to populate fields that are left blank in the input source. They're a replacement for the use of NULL values that provide a great way to predefine common data elements. You can create a DEFAULT definition as part of the table definition when you create a table. If a table already exists, you can add DEFAULT definition to it. Each column in a table can contain one DEFAULT definition.

If a DEFAULT definition already exists, you can modify or delete it. For example, you can modify the value that is inserted in a column when no value is entered.

```
CREATE TABLE Employee
```

```
(  
    Employee_id    int    IDENTITY NOT NULL,  
    EmployeeName   varchar(30) NOT NULL,  
    Address        varchar(30) NOT NULL,  
    City           varchar(20) DEFAULT 'ACCRA',  
    State          char(2)   NOT NULL,  
    Contact        varchar(25) NOT NULL,  
    Phone          char(15)  NOT NULL UNIQUE  
);
```

If you don't want to have this Check Field as part of table script then you can use ALTER statement to define the constraint.

```
ALTER TABLE Employees
```

```
ALTER Column City SET DEFAULT 'ACCRA';
```

5) FOREIGN KEY

Foreign key -Constraint is field in a relational database table that match the primary key column of another table. Foreign keys can be used to cross-reference tables.

Let's create a second table called Department

```
CREATE TABLE Department
(
    Department_id    int    IDENTITY NOT NULL,
    DepartmentName   varchar(30)  NOT NULL,
    Address          varchar(30)  NOT NULL,
    Contact          varchar(25)  NOT NULL,
    PRIMARY KEY (Department_id)
);
```

Now in order to link the two tables Employee and Department you need a foreign key

Logically we can have several employees in one department hence a relationship of one-to-many (Department-Employee)

```
ALTER TABLE Employee
ADD (CONSTRAINT fk_employee) FOREIGN KEY (Department_id) REFERENCES
Department (Department_id);
```

6.8.2 Database Triggers

A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for keeping the integrity of the information on the database. For example, when a new record (representing a new worker) is added to the employees table, new records should be created also in the tables of the taxes, vacations and salaries.

The need for trigger and the usage

Triggers are commonly used to:

Database Systems (CE 279)

- prevent changes (e.g. prevent an invoice from being changed after it's been mailed out)
- log changes (e.g. keep a copy of the old data)
- audit changes (e.g. keep a log of the users and roles involved in changes)
- enhance changes (e.g. ensure that every change to a record is time-stamped by the server's clock, not the client's)
- enforce business rules (e.g. require that every invoice have at least one-line item)
- execute business rules (e.g. notify a manager every time an employee's bank account number changes)
- replicate data (e.g. store a record of every change, to be shipped to another database later)
- enhance performance (e.g. update the account balance after every detail transaction, for faster queries)

The examples above are called Data Manipulation Language (DML) triggers because the triggers are defined as part of the Data Manipulation Language and are executed at the time the data are manipulated. Some systems also support non-data triggers, which fire in response to Data Definition Language (DDL) events such as creating tables, or runtime or and events such as logon, commit, and rollback. Such DDL triggers can be used for auditing purposes.

The following are major features of database triggers and their effects:

- triggers do not accept parameters or arguments (but may store affected data in temporary tables)
- triggers cannot perform commit or rollback operations because they are part of the triggering SQL statement (only through autonomous transactions)
- triggers can cancel a requested operation
- triggers can cause mutating table errors

Creating a Database Trigger

CREATE TRIGGER command is used to create a database trigger. The following details are to be given at the time of creating a trigger.

- _ Name of the trigger
- _ Table to be associated with
- _ When trigger is to be fired - before or after
- _ Command that invokes the trigger - UPDATE, DELETE, or INSERT
- _ Whether row-level trigger or not
- _ Condition to filter rows.
- _ PL/SQL block that is to be executed when trigger is fired.

The following is the syntax of CREATE TRIGGER command.

```
CREATE [OR REPLACE] TRIGGER triggername
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE [OF columns]}
[OR {DELETE | INSERT | UPDATE [OF columns]}].
ON table
[FOR EACH ROW [WHEN condition]]
[REFERENCING [OLD AS old] [NEW AS new]]
PL/SQL block
```

SQL Triggers

The SQL CREATE TRIGGER statement provides a way for the database management system to actively control, monitor, and manage a group of tables whenever an insert, update, or delete operation is performed. The statements specified in the SQL trigger are executed each time an SQL insert, update, or delete operation is performed. An SQL trigger may call stored procedures or user-defined functions to perform additional processing when the trigger is executed.

Unlike stored procedures, an SQL trigger cannot be directly called from an application. Instead, an SQL trigger is invoked by the database management system on the execution of a triggering insert, update, or delete operation. The definition of the SQL trigger is stored in the database management system and is invoked by the database management system, when the SQL table, that the trigger is defined on, is modified.

An SQL trigger can be created by specifying the CREATE TRIGGER SQL statement. The statements in the routine-body of the SQL trigger are transformed by SQL into a program (*PGM) object.

Types of Triggers

1. DML Triggers

- AFTER Triggers
- INSTEAD OF Triggers

2. DDL Triggers

3. CLR Triggers

DML Triggers - These triggers are fired when a Data Manipulation Language (DML) event takes place. These are attached to a Table or View and are fired only when an

INSERT, UPDATE and/or DELETE event occurs. The trigger and the statement that fires it are treated as a single transaction. Using this we can cascade changes in related tables, can do check operations for satisfying some rules and can get noticed through firing Mails. We can even execute multiple triggering actions by creating multiple Triggers of same action type on a table. We have to specify the modification action(s) at the Table level that fires the trigger when it is created.

AFTER Triggers

As the name specifies, AFTER triggers are executed after the action of the INSERT, UPDATE, or DELETE statement is performed. AFTER triggers can be specified on tables only, here is a sample trigger creation statement on the Users table.

Create table Person (age int);

```
CREATE TRIGGER PersonCheckAge
AFTER INSERT OR UPDATE OF age ON Person
FOR EACH ROW
BEGIN
    IF (:new.age < 0) THEN
        RAISE_APPLICATION_ERROR (-20000, 'no negative age allowed');
    END IF;
END;
```

If we attempted to execute the insertion:

Insert into Person values (-3);

We would get the error message:

ERROR at line 1:

ORA-20000: no negative age allowed

ORA-06512: at "MYNAME.PERSONCHECKAGE", line 3

ORA-04088: error during execution of trigger 'MYNAME.PERSONCHECKAGE' and nothing would be inserted.

INSTEAD OF Triggers

INSTEAD OF triggers are executed in place of the usual triggering action. INSTEAD OF triggers can also be defined on views with one or more base tables, where they can extend the types of updates a view can support.

DDL Triggers - This type of triggers, like regular triggers, fire stored procedures in response to an event. They fire in response to a variety of Data Definition Language (DDL) events. These events are specified by the T-SQL statements that are start with the keywords CREATE, ALTER, and DROP. Certain stored procedures that perform DDL-like operations can also fire this. These are used for administrative tasks like auditing and regulating database operations.

DDL triggers can fire in response to a Transact-SQL event processed in the current database, or on the current server. The scope of the trigger depends on the event. For example, a DDL trigger created to fire in response to a CREATE_TABLE event can do so whenever a CREATE_TABLE event occurs in the database, or on the server instance. A DDL trigger created to fire in response to a CREATE_LOGIN event can do so only when a CREATE_LOGIN event occurs in the server.

In the following example, DDL trigger safety will fire whenever a DROP_TABLE or ALTER_TABLE event occurs in the database.

```
CREATE TRIGGER safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
PRINT 'You must disable Trigger "safety" to drop or alter tables!'
ROLLBACK;
```

CLR Triggers - A CLR triggers can be any of the above, e.g. can be a DDL or DML one or can also be an AFTER or INSTEAD OF trigger. Here we need to execute one or more methods written in managed codes that are members of an assembly created in the .Net framework. Again, that assembly must be deployed in SQL Server 2005 using CREATE assembly statement.

Chapter Questions

1. Discuss the various elements of SQL language and give two examples each
2. Discuss the uses of the following SQL clauses:
 - a. FROM
 - b. WHERE
 - c. GROUP BY
 - d. HAVING
 - e. ORDER BY
3. Discuss the various SQL Command categories and their general functions
4. State the use of the following SQL commands and give two (2) examples each:
 - a. CREATE
 - b. ALTER
 - c. DROP
 - d. TRUNCATE
 - e. RENAME
 - f. INSERT
 - g. SELECT
 - h. UPDATE
 - i. DELELTE
 - j. GRANT
 - k. REVOKE
 - l. JOIN
5. Differentiate between Inner and Outer joints
6. Explain the following and state their uses:
 - a. Database Constraints
 - b. Database Triggers
7. Distinguish between the following in terms of their uses:
 - a. Int, SmallInt, TinyInt and BigInt
 - b. Float and Decimal
 - c. Char, VarChar and NVarChar
8. Define the following and give two (2) examples each:
 - a. Entity constraints
 - b. Domain constraints
 - c. Referential integrity constraints
9. Discuss the two types of SQL Triggers and how to use them