E-DATABASE DESIGN

RAMLAH BINTI MD ZAIN
MAZLINA BINTI MUSTAPHA

E - DATABASE DESIGN

Ramlah Binti Md Zain
Mazlina Binti Mustapha

Department of Information and CommunicationTechnology
Sultan Mizan Zainal Abidin Polytechnic

EDITION 2021

First Publishing 2021

Published by:
Politeknik Sultan Mizan Zainal Abidin.
KM 08, Jalan Paka,
23000 Dungun, Terengganu Darul Iman.
Tel : 098400800
Fax : 09-8458781
www.psmza.edu.my

E-DATABASE DESIGN
EDITION 2021

e ISBN 978-967-2099-74-1

Ramlah Binti Md Zain
Mazlina Binti Mustapha

PREFACE

In the name of Allah, The Most Gracious and Merciful. All praise to Allah S.W.T for His great loving kindness and blessing, this book is successfully published.

E - Database Design book is designed specifically for a first course in databases at the junior or senior undergraduate, or first year graduate level. The purpose in this text is to present the fundamental concepts of database design.These concepts include aspects of database design like fundamental of database, DBMS, relational data model, Entity Relationship Model, normalization, structured query language (SQL) and database transaction management.

The authors would like to express deepest appreciation to all those who have provided the possibility in publishing this book especially family, friends and colleagues
The book covers all the essential aspects of database design based on those used in existing commercial or experimental database design. Hopefully students and lecturers can use it for a learning process.

Thank you.

ABSTRACT

This digital writing reviewing basic concepts of databases and database design, then turns to creating, populating, and retrieving data using SQL. Topics such as Database Management System, the relational data model, Entity Relationship Diagram, normalization, data entities, and database transaction management are covered clearly and concisely. This book provides the conceptual and practical information necessary to develop a database design and management scheme that ensures data accuracy and user satisfaction while optimizing performance.

Table of Contents

FUNDAMENTALS OF DATABASE MANAGEMENT SYSTEM PAGE 1 |
RELATIONAL DATA MODEL PAGE 22 |
ENTITY-RELATIONSHIP (ER) MODEL & NORMALIZATION PAGE 34 |
STRUCTURED QUERY LANGUAGE (SQL) PAGE 60 |
DATABASE TRANSACTION MANAGEMENT PAGE 86 |

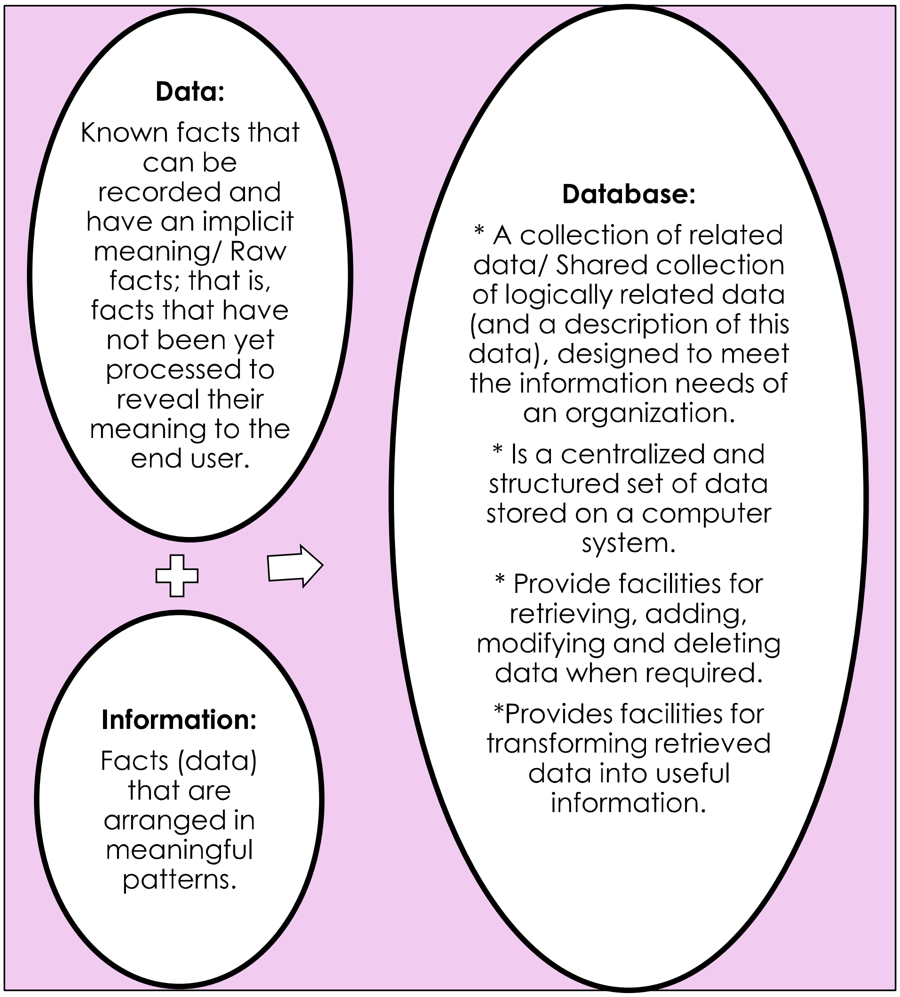
CHAPTER 1

FUNDAMENTALS OF DATABASE MANAGEMENT SYSTEM

In this chapter, you will:

- **Understand Database**
- **Understand DBMS**
- **Understand Data Model**

FUNDAMENTALS OF DATABASE MANAGEMENT SYSTEM



USES OF DATABASES IN THE BUSINESS WORLD

Businesses may use databases to manage customers, inventory and personnel. Databases are powerful organizational tools that help businesses quickly record, view and respond to important information. When used effectively, they can improve the efficiency and profitability of a business

Customer relationship management (CRM) software allows businesses to document every interaction with a current or potential customer, leading to more efficient marketing and sales departments. Some modern CRM databases even integrate information from traditional contact methods such as phone calls and printed mail with data obtained from a company's social media efforts.

Businesses can use databases to keep track of inventory so they know how much merchandise is in a warehouse and how much is available for customers to purchase from a store's shelves. Companies also manage their employees using databases, effectively tracking large amounts of salary, payroll and tax data.

IMPORTANCE OF DATABASES TO EVERYDAY LIFE

Data Integration
- Achieved by combining master files into larger pools of data accessible by many programs.

Data Sharing
- It's easier to share data that's integrated—Example : the FBI is planning an 8 year, $400 million database project to make data more available to agency users.

Reporting Flexibility
- Reports can be revised easily and generated as needed.
- The database can easily be browsed to research problems or obtain detailed information.

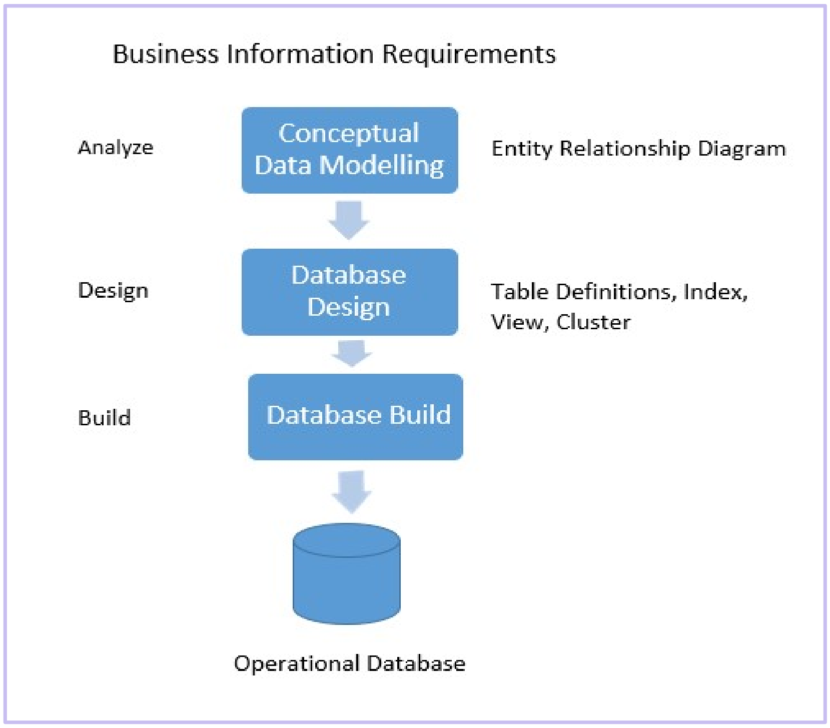
Minimal Data Redundancy and Inconsistencies
- Because data items are usually stored only once.

Data Independence
- Relationships can be explicitly defined and used in the preparation of management reports.
- EXAMPLE: Relationship between selling

Central Management of Data
- Data management is more efficient because the database administrator is responsible for coordinating, controlling, and managing data

Cross-Functional Analysis
- Data items are independent of the programs that use them..
- Consequently, a data item can be changed without changing the program and vice versa.
- Makes programming easier and simplifies data management.

MAJOR STEPS IN THE DATABASE DEVELOPMENT PROCESS



SHARING CONCEPT OF DATA IN DATABASE

Definition

* The ability to share the same data resource with multiple applications or users.

* It implies that the data are stored in one or more servers in the network and that there is some software locking mechanism that prevents the same set of data from being changed by two people at the same time.

* Data sharing is a primary feature of a database management system (DBMS)

Characteristic

*The most significant difference between a file based systems and database systems is data sharing.

*Data sharing also requires a major change in the way of data are handled and managed within the organization.

Types

Three types of data sharing :

*Sharing Data between functional units.

*Sharing data between management units.

*Sharing data between geographically dispersed location.

Sharing Data Between Functional Units

- The term data sharing suggests that people in different functional areas are use a common pool of data. Each of these are own applications without data sharing the marketing group may have their data files. The purchasing group like accounts group their own data files and marketing group have their own data files and each group benefits from its own data.

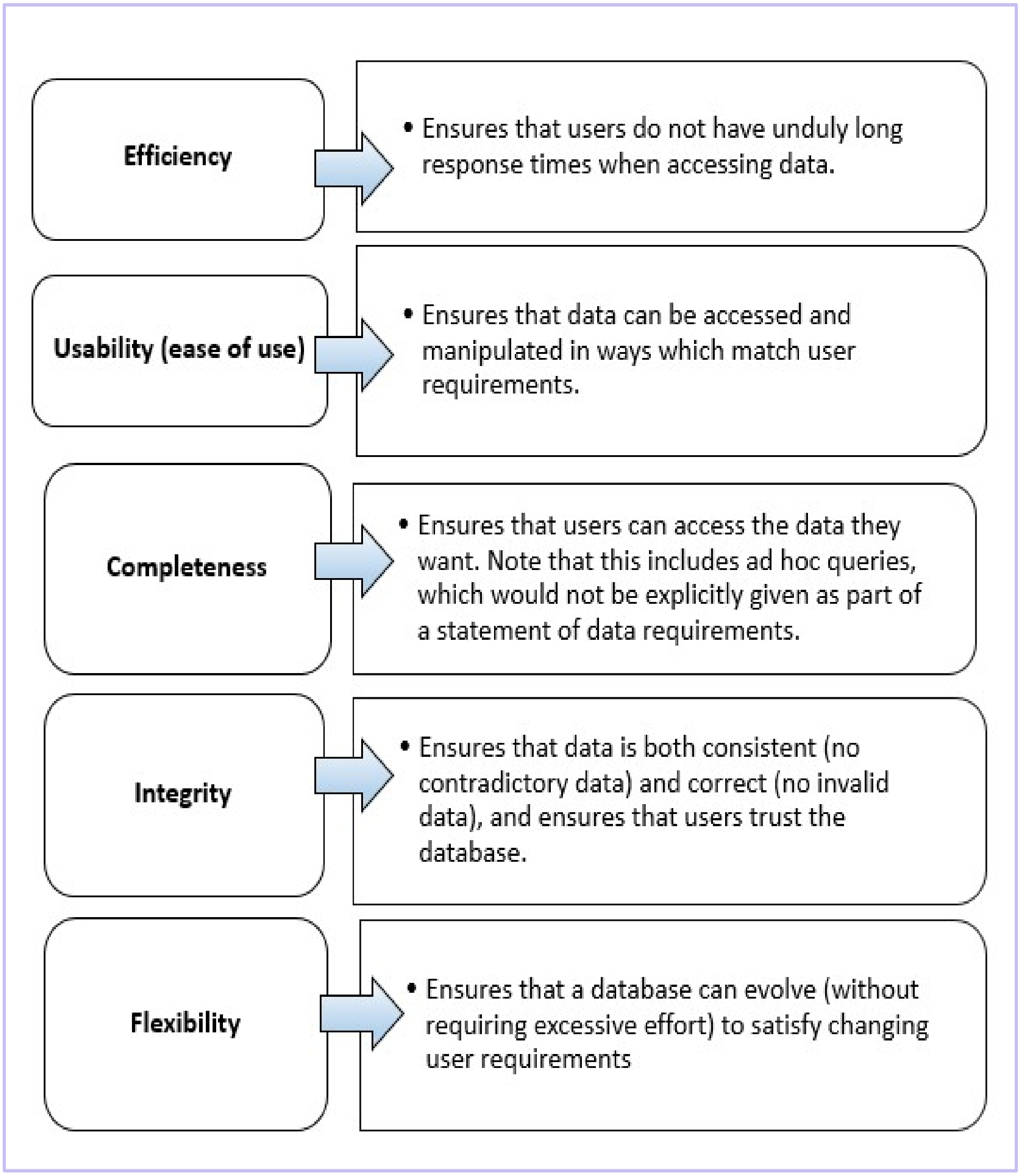
Sharing Data Between Management Units

- Different levels of users also need to share data . The three different levels of users are 1. Operation level, 2. Middle Management Level, 3. Execute level.
- These three levels are corresponded to the three different types of system these are *Electronic data processing, Management information system, and Decision support system.*

Sharing data between geographically dispersed location

- A company with several locations has important data distributed over a valid geographically area sharing. These data is a significant problems. A centralized database is physically contained to a single location controlled by a single computer that is Personal computer most function for which databases are created and accomplished more easily . If the database is centralized and it is easily to update and back up , recovery and control access to a database . If we know database exactly where it is and what's software control it and identify the remote place where it is located.

PROPERTIES OF DATABASES



UNDERSTAND DBMS

Definition

- A database management system (DBMS) is the software system that allows users to define, create and maintain a database and provides controlled access to the data.
- A Database Management System (DBMS) is basically a collection of programs that enables users to store, modify, and extract information from a database as per the requirements. DBMS is an intermediate layer between programs and the data. Programs access the DBMS, which then accesses the data.
- There are different types of DBMS ranging from small systems that run on personal computers to huge systems that run on mainframes.

Examples of database application

-> Computerized library systems

-> Automated teller machines

-> Flight reservation systems

-> Computerized parts inventory systems

Various Common of DBMS

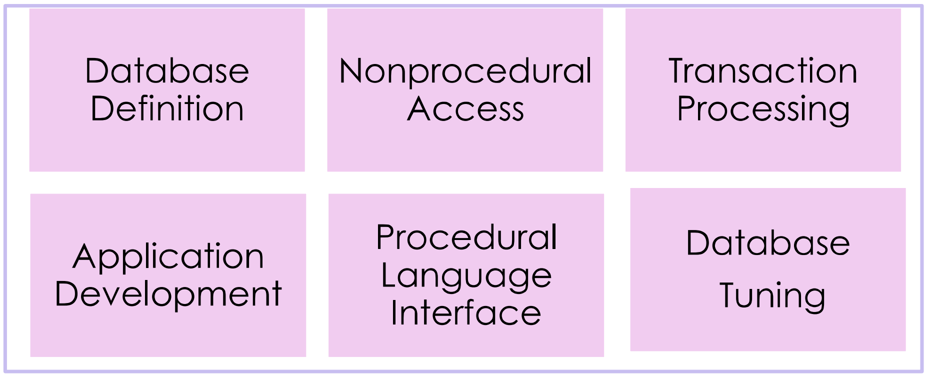
Paradox, Lotus, FileMaker, Microsoft Access, Dbase, FoxPro, IMS and Oracle, MySQL, Microsoft SQL Server, PostgreSQL and DB2

Functions of DBMS

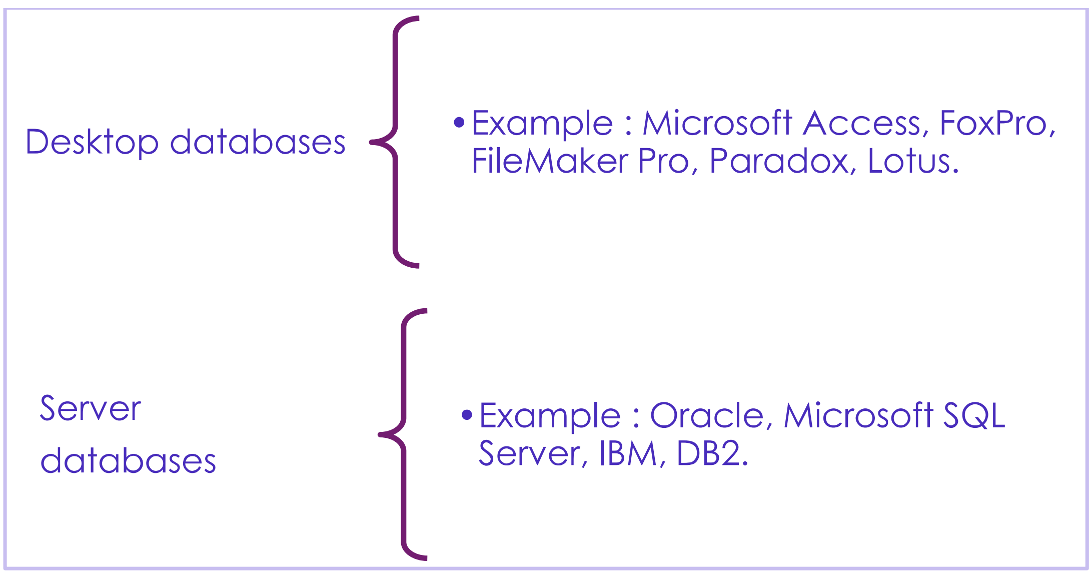
-> create update, and extract information from their databases.

-> Compared to a manual filing system, the biggest advantages to a computerized database system are speed, accuracy, and' accessibility.

FEATURES OF DMBS



CATEGORIES DBMS



THE TRADITIONAL APPROACH TO INFORMATION PROCESSING

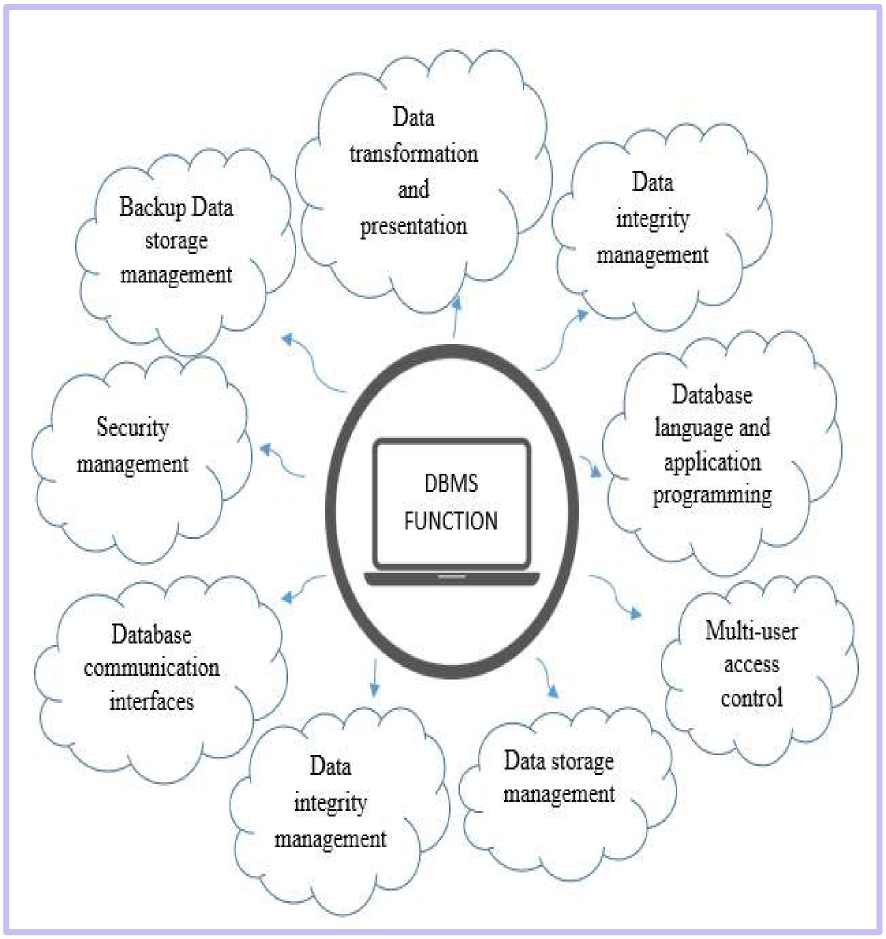
In the early days of computing, data management and storage was a very new concept for organizations. The traditional approach to data handling offered a lot of the convenience of the manual approach to business processes (e.g. hand written invoices & account statements, etc.) as well as the benefits of storing data electronically.

The traditional approach usually consisted of custom built data processes and computer information systems tailored for a specific business function. An accounting department would have their own information system tailored to their needs, where the sales department would have an entirely separate system for their needs.

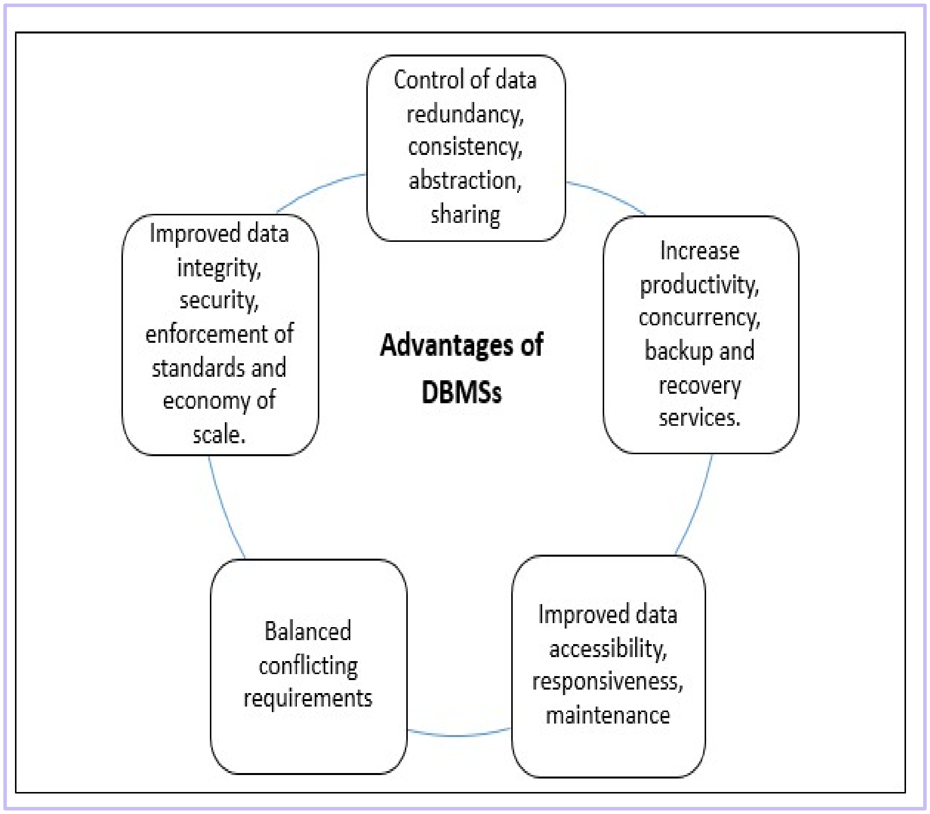
Separate information systems for each business function also led to conflicts of interest within the company. Departments felt a great deal of ownership for the data that they collected, processed, and managed which caused many issues among company-wide collaboration and data sharing. This separation of data also led to unnecessary redundancy and a high rate of unreliable and inconsistent data.

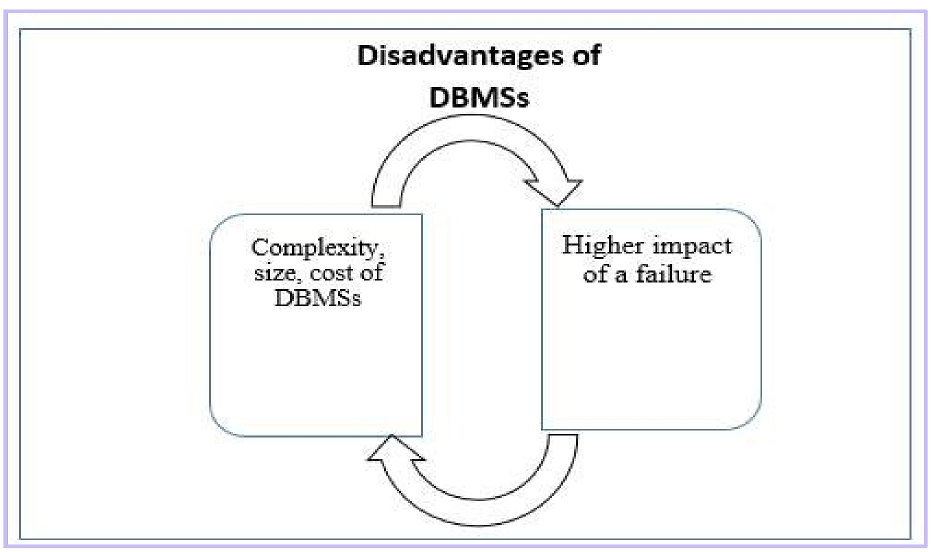
Initially, these separate systems were very simple to set up as they mostly mirrored the business process that departments had been doing for years but allowed them to do things faster with less work. However, once the systems were in use for so long, they became very difficult for individual departments to manage and rely on their data because there was no reliable system in place to enforce data standards or management.

DBMS FUNCTIONS

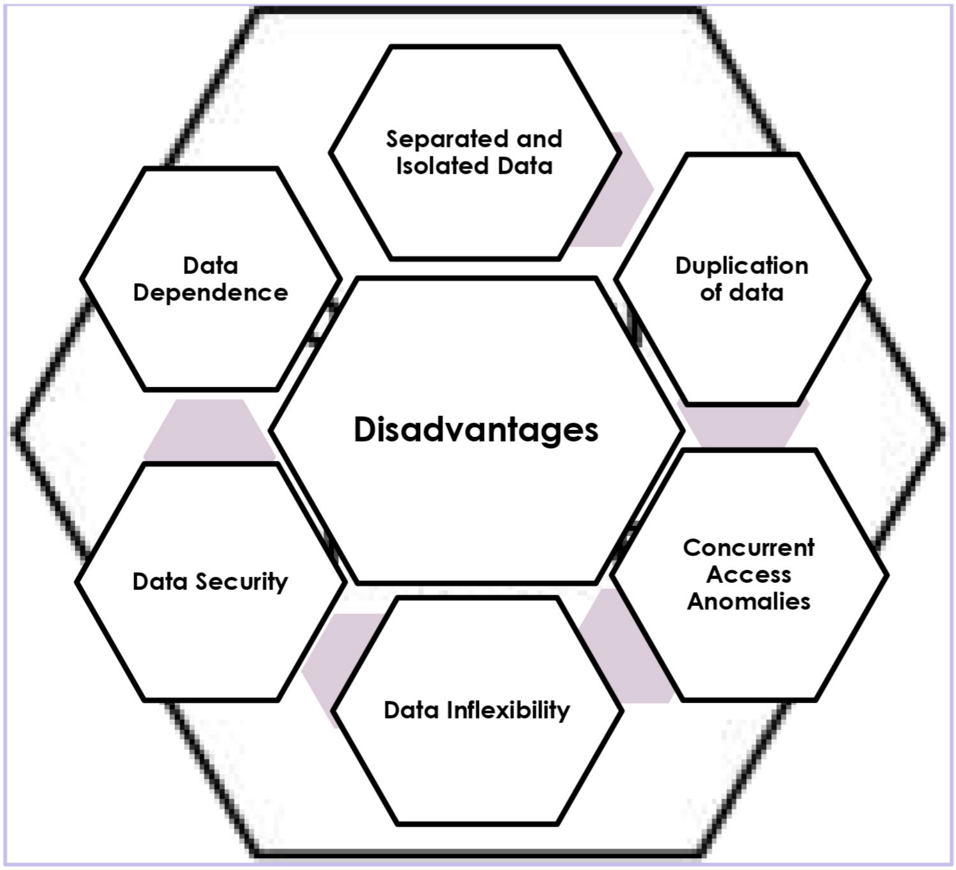


ADVANTAGES AND DISADVANTAGES OF DBMS'S

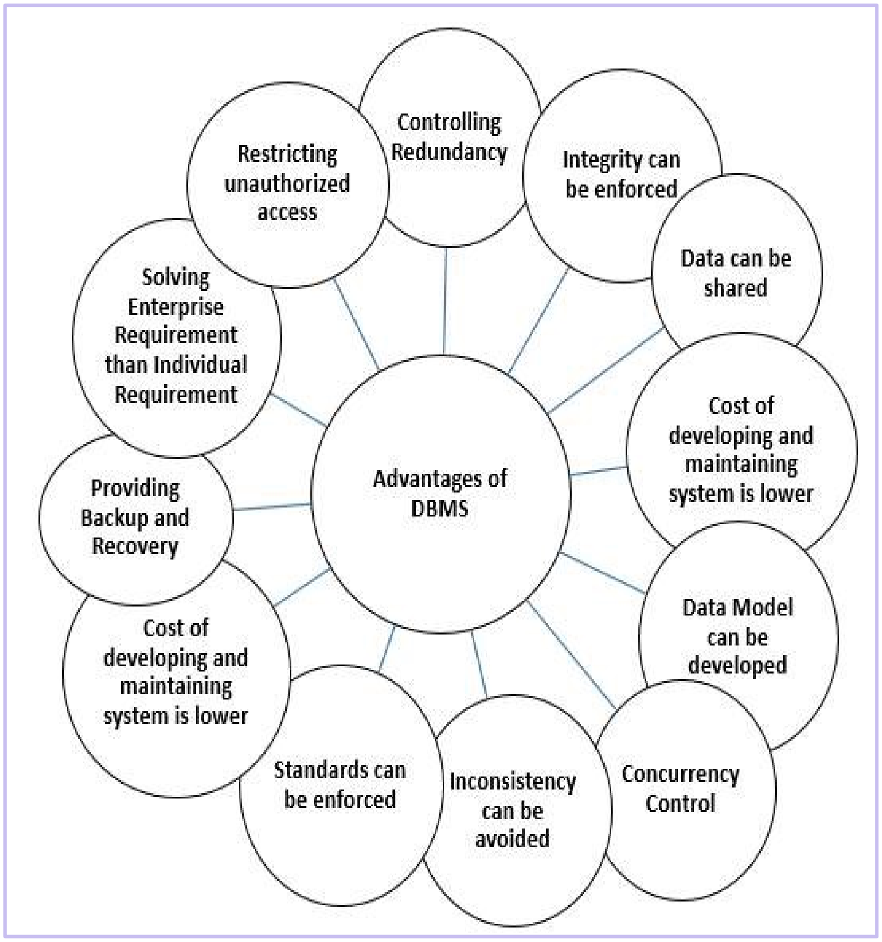




DISADVANTAGES OF TRADITIONAL APPROACH TO INFORMATION PROCESSING



IMPORTANCE OF HAVING DBMS



DATABASE ARCHITECTURE

A Distributed Database

- is a database that is under the control of a central database management system (DBMS) in which storage devices are not all attached to a common CPU. It may be stored in multiple computers located in the same physical location, or may be dispersed over a network of interconnected computers.
- Collections of data (e.g. in a database) can be distributed across multiple physical locations. A distributed database can reside on network servers on the Internet, on corporate intranets or extranets, or on other company networks. The replication and distribution of databases improves database performance at end-user worksites.

Centralized Database

- has all its data on one place. As it is totally different from distributed database which has data on different places. In centralized database as all the data reside on one place so problem of bottle-neck can occur, and data availability is not efficient as in distributed database.

UNDERSTAND DATA MODEL

Data models define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS.

Data models define how data is connected to each other and how they are processed and stored inside the system

The very first data model could be flat data-models, where all the data used are to be kept in the same place.

Earlier data models were not so scientific, hence they were prone to introduce lots of duplication and update anomalies

LOGICAL DATA MODEL

A **logical data model** or **logical schema** is a data model of a specific problem domain expressed independently of a particular database management product or storage technology (physical data model) but in terms of data structures such as relational tables and columns, object-oriented classes, or XML tags. This is as opposed to a conceptual data model, which describes the semantics of an organization without reference to technology

TYPES OF LOGICAL DATA MODEL

Object Based Logical Model	Record Based Logical Model
Entity Relationship Data Model	Hierarchical data model
	Network data model
	Relational data model

THREE LEVEL ARCHITECTURE OF DBMS

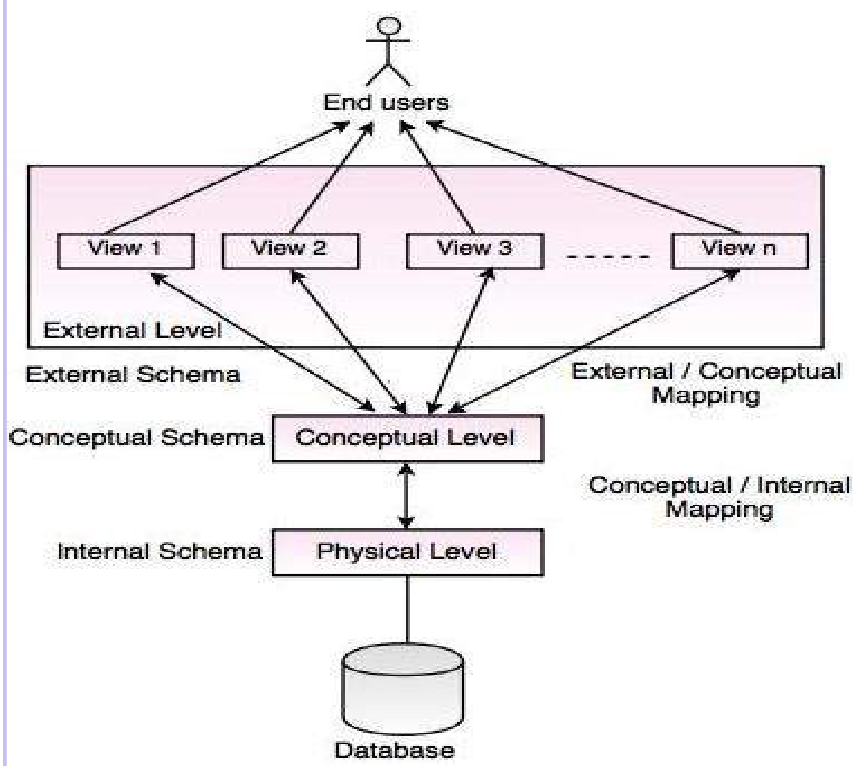


Fig. Three Level Architechture of DBMS

Three Level Architecture of DBMS

- It shows the architecture of DBMS.
- Mapping is the process of transforming request response between various database levels of architecture.
- The goal of the three-schema architecture is to separate the user applications and the physical database.
- Mapping is not good for small database, because it takes more time.
- In External / Conceptual mapping, DBMS transforms a request on an external schema against the conceptual schema.
- In Conceptual / Internal mapping, it is necessary to transform the request from the conceptual to internal levels.

Conceptual Level

Conceptual level describes the structure of the whole database for a group of users.

It is also called as the data model.

Conceptual schema is a representation of the entire content of the database.

These schema contains all the information to build relevant external records.

These schema contains all the information to build relevant external records.

It hides the internal details of physical

Physical Level	Physical level describes the physical storage structure of data in database. It is also known as Internal Level.
	This level is very close to physical storage of data. At lowest level, it is stored in the form of bits with the physical addresses on the secondary storage device.
	At highest level, it can be viewed in the form of files. The internal schema defines the various stored data types. It uses a physical data model.

External Level	External level is related to the data which is viewed by individual end users. This level includes a no. of user views or external schemas.
	This level is closest to the user. External view describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group

External Level

- External level is related to the data which is viewed by individual end users.
- This level is closest to the user.
- This level includes a number of user views or external schemas.
- External view describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group

Conceptual Level

- Conceptual level describes the structure of the whole database for a group of users.
- It is also called as the data model.
- Conceptual schema is a representation of the entire content of the database.
- These schema contains all the information to build relevant external records.
- These schema contains all the information to build relevant external records.
- It hides the internal details of physical storage.

Physical Level

- Physical level describes the physical storage structure of data in database.
- It is also known as Internal Level.
- This level is very close to physical storage of data.
- At lowest level, it is stored in the form of bits with the physical addresses on the secondary storage device.
- At highest level, it can be viewed in the form of files.
- The internal schema defines the various stored data types. It uses a physical data model.

QUESTIONS

Chapter 1 Exercise: Fundamentals of Database Management System

1. Discuss the information needs of a: (a) bank, (b) shopping, (c) restaurant, (d) student registration, (e) and (f)
2. List and discuss the characteristic of good database design.
3. Differentiate database and database management system

CHAPTER 2

RELATIONAL DATA MODEL

In this chapter, you will:

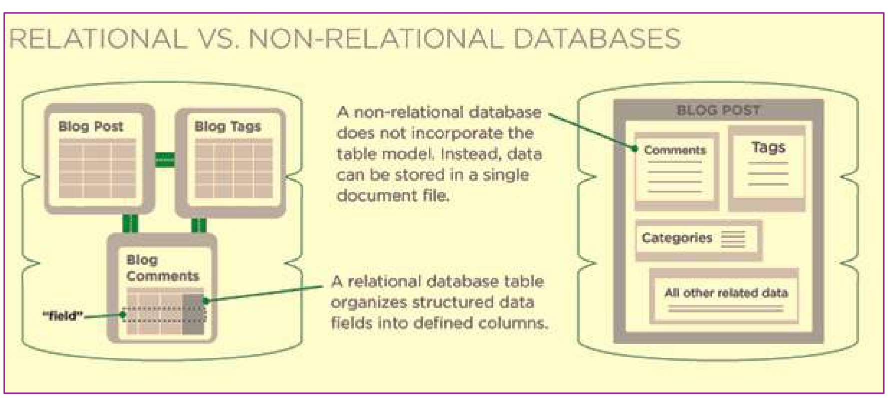
- Understand Relational Databases
- Understand Operators of Relational Algebra

RELATIONAL DATA MODEL

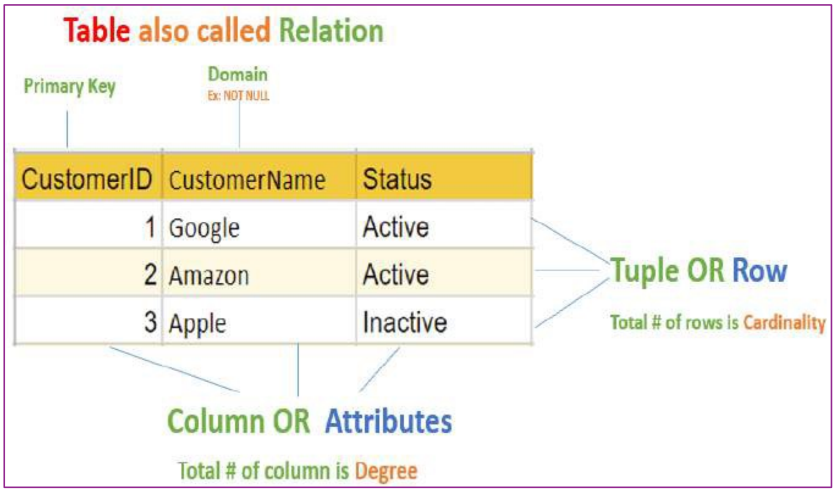
A **relational database** is a digital database based on the relational model of data, as proposed by E. F. Codd in 1970. A software system used to maintain relational databases is a relational database management system (RDBMS). Virtually all relational database systems use SQL (Structured Query Language) for querying and maintaining the database

- Some popular **RDBMS packages** are Oracle RDBMS, IBM DB2, Microsoft SQL Server, SAP Sybase ASE, Teradata, ADABAS, MySQL, FileMaker, Microsoft Access, Informix, SQLite, PostgreSQL, Amazon RDS, MongoDB, Redis etc.

Relational versus Non - Relational Databases



Relational Data Structure. (Components of database tables)



COMPONENTS OF DATABASE TABLES

- Relation –> a table with columns and rows.
- Attribute(field) –> a named column of a relation
- Domain –> the set of allowable values for one or more attributes
- Tuple(record) –> a record of a relation
- Cardinality --> a number of tuple in relation
- Degree –> a number of attribute in a relation
- Primary Key --> the column or columns that contain values that uniquely identify each row in a table.

PROPERTIES OF RELATIONAL TABLE

- The table has a name that is distinct from all other tables in the database.
- Each cell of the table contains exactly one value (atomic)
- Each column has a distinct name (technically called the attribute name)
- The order of columns and rows is immaterial
- Each row (record) is distinct, there are no duplicate records

RELATIONAL MODEL SCHEMA AND EVENTS

✓ **Relation schema is a name relation defined by a set of attribute and domain name pairs.**

- **Let A1, A2….. An be attributes with domains D1, D2….Dn. then, the set {A1:D1, A2:D2…..An:Dn} is a relation schema.**

Or more correctly:

{(branchNo : B005, street:22 Deer Rd, city:London, postcode: SW1 4EH)}

- ✓ **Relational database schema is a set of relation schemas, each with a distinct name.**
    - **If R1, R2…. Rn are a set of relation schemas, then we can write the relational database schema, or simply relational schemas, R as:**

      R = {R1, R2…..Rn}Example:

      Branch (branchNo, street, city, state, zip code, mgrStaffNo)

RELATIONAL INTEGRITY

Definition - The attributes which has a relation with the domain. The relational integrity has a constraint which is called domain constraint

Null	A special column value, distinct from 0, blank, or any other value that indicates that the value for the column is missing or otherwise unknown.
Entity Integrity	Each instance of an entity (type) must have a unique primary key value that is not null. Null means empty, not blank or zero.
Referential Integrity	This refers to rules about the relationship between entities. A referenced item in one table (entity) must exist in another (related) table. for example, if there is a reference to a product code in one table, then information about that product (e.g, product name, unit price) must exists in another table.

RELATIONAL MODEL RELATIONSHIPS

- One to many relationship
- Many to many relationship
- Self referencing relationship

RELATIONAL ALGEBRA

The data in relational tables are of limited value unless the data can be manipulated to generate useful information.

Relational algebra defines the theoretical way of manipulating table contents using the relational operators: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE and PRODUCT.

OPERATOR	SYMBOL
Selection	**б**
Projection	**π**
Renaming	ρ
Union	∪
Intersection	∩
Difference	--
Cartesian Product	**X**
Join	⋈
Logical AND	**^**
Logical OR	**v**
Logical NOT	**~**

Example : The table Employee

nostaff	name	salary
123	Aisyah	5000
289	Zahra	6000
666	Azib	7000

Projection

PROJECT $_{salary}$ (Employee)

Π $_{salary}$ (Employee)

Result:

salary
5000
6000
7000

Selection

SELECT $_{salary\ <7000}$ (Employee

σ $_{salary\ <7000}$ (Employee)

Result :

nostaff	name	salary
123	Aisyah	5000
289	Zahra	6000

Projection & Selection

PROJECT$_{name, salary}$ (**SELECT**$_{salary < 7000}$ (EMPLOYEE))

$\Pi_{name, salary}$ ($\sigma_{salary < 7000}$ (EMPLOYEE))

or, step by step, using an intermediate result

Temp <- **SELECT**$_{salary <70000}$(EMPLOYEE)
Result <- **PROJECT**$_{name, salary}$(Temp)

Or Temp <- $\sigma_{salary <70000}$(EMPLOYEE)

Result <- $\Pi_{name, salary}$(Temp)

Result :

name	salary
Aisyah	5000
Zahra	6000

Cartesian Product

EMPLOYEE

enr	ename	dept
1	Ahmad	A
2	Sarah	C
3	Sabri	A

DEPARTMENT

dnr	dname
A	Marketing
B	Sales
C	Legal

Result : EMPLOYEE X DEPARTMENT

enr	ename	dept	dnr	dname
1	Ahmad	A	A	Marketing
1	Ahmad	A	B	Sales
1	Ahmad	A	C	Legal
2	Sarah	C	A	Marketing
2	Sarah	C	B	Sales
2	Sarah	C	C	Legal
3	Sabri	A	A	Marketing
3	Sabri	A	B	Sales
3	Sabri	A	C	Legal

Natural Join

$\textbf{SELECT}_{dept = dnr}$ (EMPLOYEE **X** DEPARTMENT) or

EMPLOYEE $\textbf{JOIN}_{dept = dnr}$ DEPARTMENT

EMPLOYEE $\bowtie_{dept = dnr}$ DEPARTMENT

Result :

enr	ename	dept	dnr	dname
1	Ahmad	A	A	Marketing
2	Sarah	C	C	Legal
3	Sabri	A	A	Marketing

UNION, INTERSECTION AND DIFFERENCE

- All of these operations take two input relations, which must be union-compatible:
  - Same number of fields.
  - `Corresponding' fields have the same type.

Example :

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Union

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$$S1 \cup S2$$

Intersection

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$$S1 \cap S2$$

Difference

sid	sname	rating	age
22	dustin	7	45.0

$$S1 - S2$$

QUESTIONS

Chapter 2 Exercise: Relational Data Model

1. Define the term relational data model. List the characteristic of this model
2. Explain the relational data structure:
    a. Relation
    b. Attribute (field)
    c. Domain
    d. Tuple (record)
    e. Degree
    f. Cardinality
    g. Relational database
3. Explain the terms (a) primary key, (b) foreign key and (c) composite key.
4. List and discuss the major components of a relational database environment.
5. Based on table 2.1, extract and combine the data from Professor and Student table.

Table 2.1: Professor and student tables

Professor

FN	LN
John	Smith
Ricardo	Brown
Susan	Yao
Francis	Johnson
Ramesh	Shah

Student

FN	LN
Susan	Yao
Ramesh	Shah
Barbara	Jones
Amy	Ford
Jimmy	Wang

a. Professor Union Student (Professor ∪ Student)
b. Professor Intersection Student (Professor ∩ Student)
c. Professor difference Student (Professor - Student)
d. Student difference Professor (Student - Professor)

CHAPTER 3

ENTITY E-R MODEL & NORMALIZATION

In this chapter, you will:

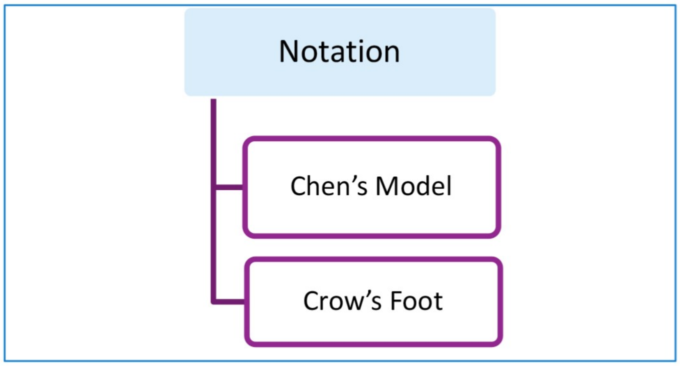
- **Apply E-R Diagram(ERD) in database development**
- **Apply the Normalization**

ENTITY RELATIONSHIP DIAGRAM

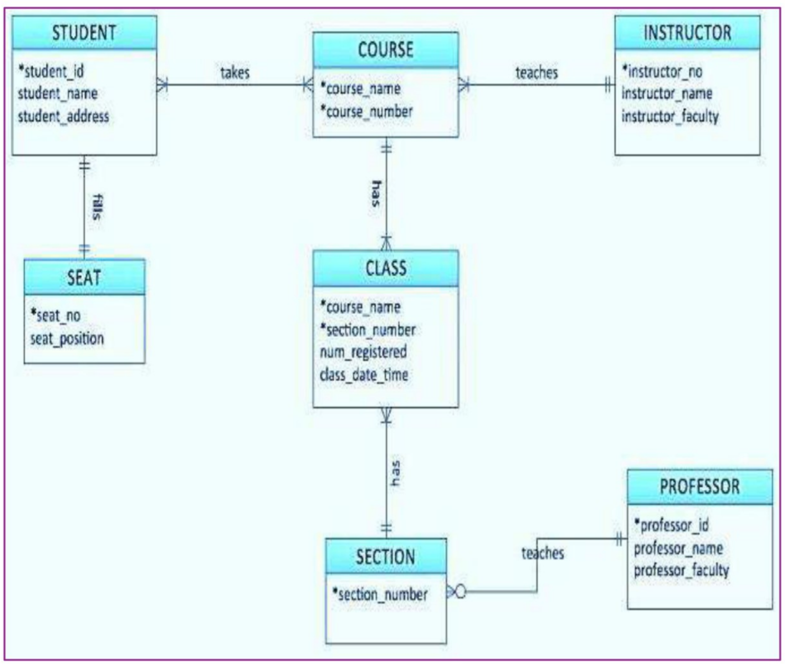
- Entity Relationship modelling was develop for database design by Peter Chen in 1976
- is a graphical representation of the database system
- provides a high-level conceptual data model
- supports the user's perception of the data
- is composed of entities, attributes, and relationships

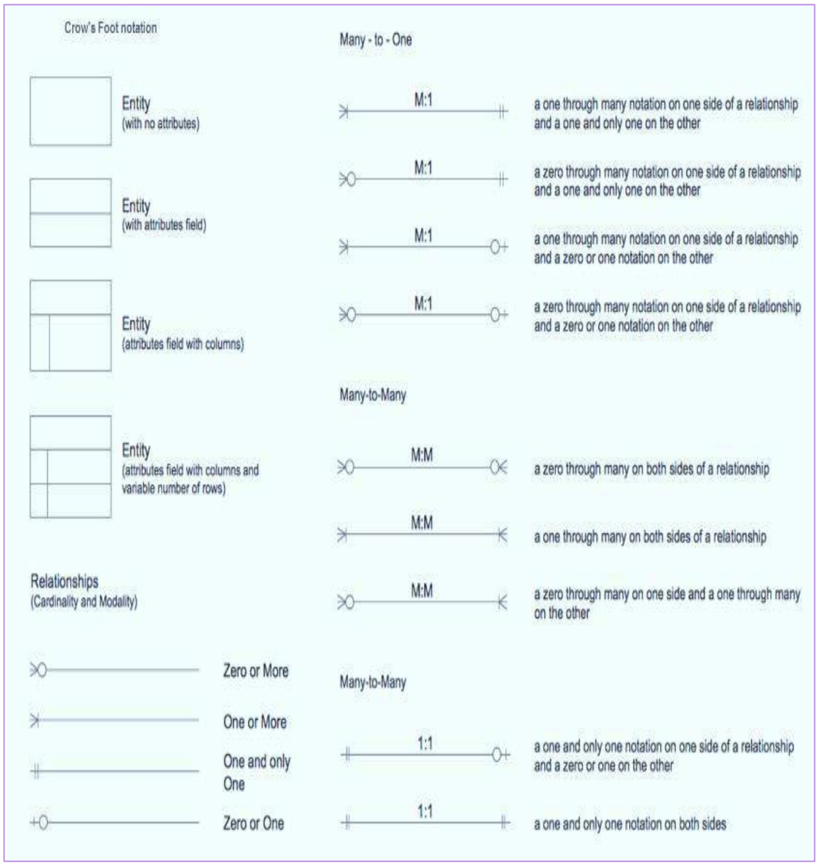
3 BASIC COMPONENT

- **ENTITIES**
- **ATTRIBUTES**
- **RELATIONSHIPS**

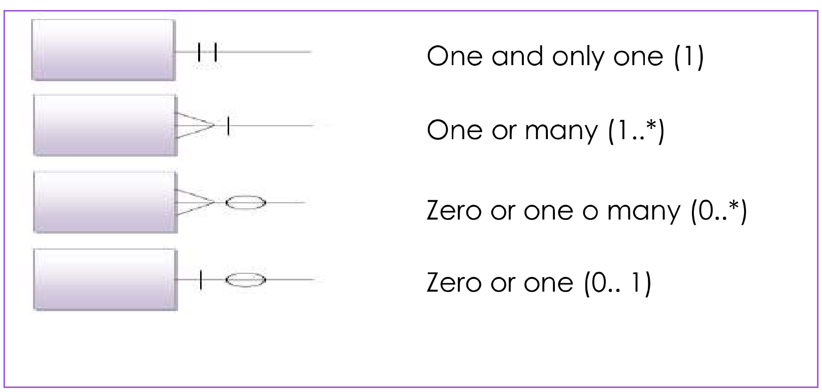


Example : Crow's Foot Model



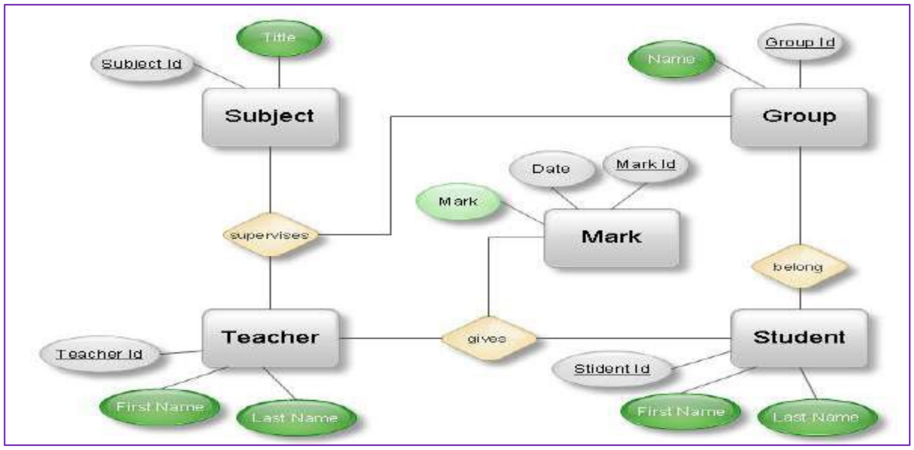


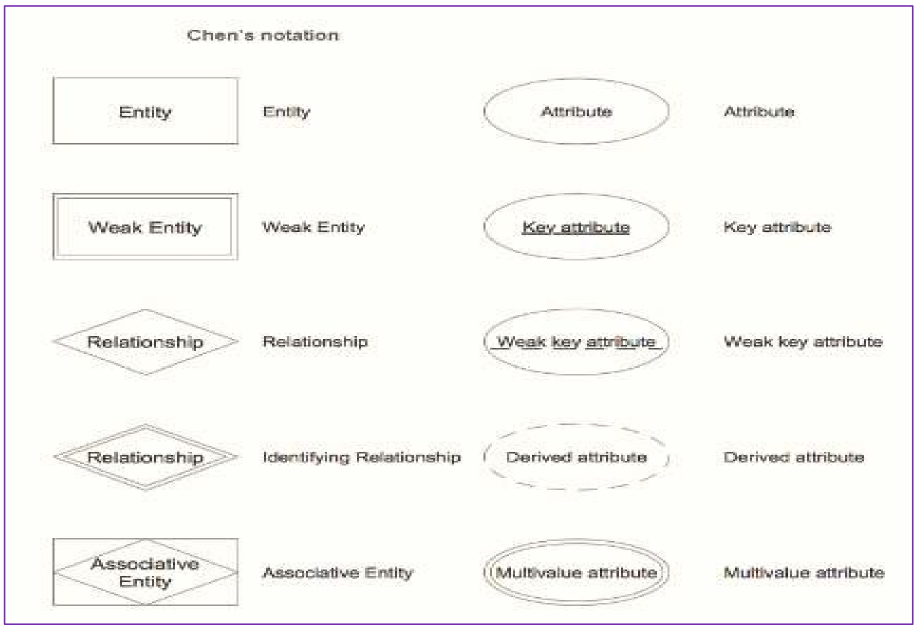
Crow's Foot notation
Entity (with no attributes)
Entity (with attributes field)
Entity (attributes field with columns)
Entity (attributes field with columns and variable number of rows)
Relationships (Cardinality and Modality)
Zero or More
One or More
One and only One
Zero or One
Many - to - One
M:1
a one through many notation on one side of a relationship and a one and only one on the other
M:1
a zero through many notation on one side of a relationship and a one and only one on the other
M:1
a one through many notation on one side of a relationship and a zero or one notation on the other
M:1
a zero through many notation on one side of a relationship and a zero or one notation on the other
Many-to-Many
M:M
a zero through many on both sides of a relationship
M:M
a one through many on both sides of a relationship
M:M
a zero through many on one side and a one through many on the other
Many-to-Many
1:1
a one and only one notation on one side of a relationship and a zero or one on the other
1:1
a one and only one notation on both sides

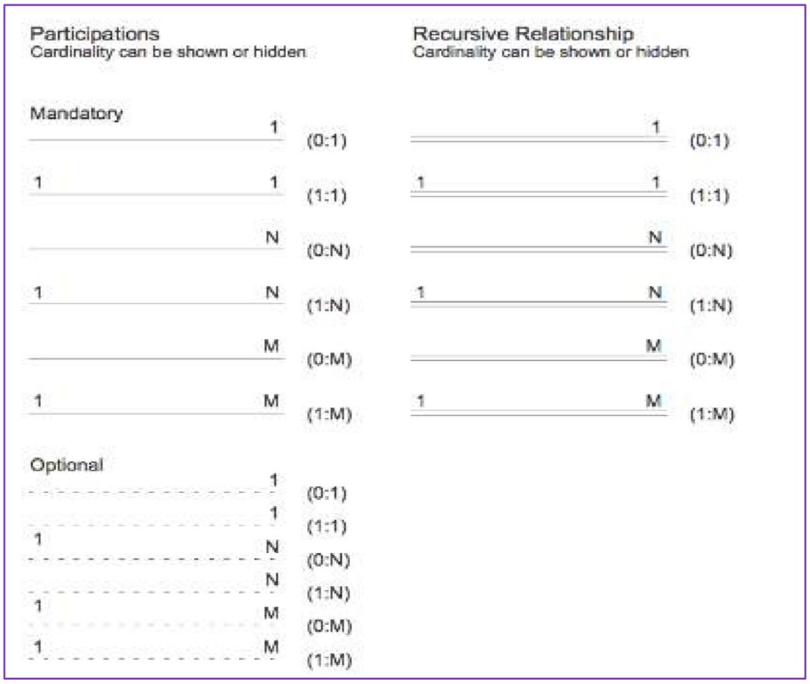


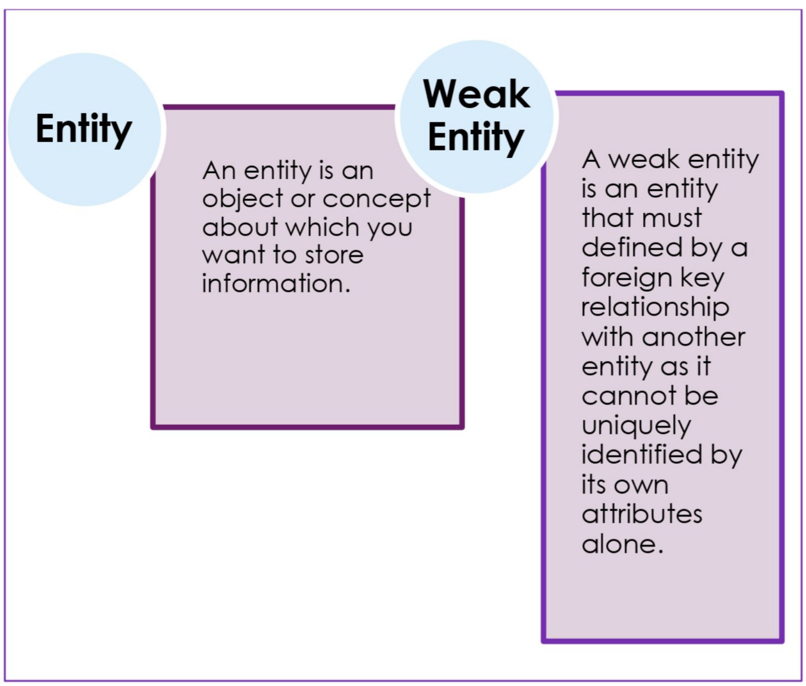
One and only one (1)
One or many (1..*)
Zero or one o many (0..*)
Zero or one (0.. 1)

Example : Chen's Model





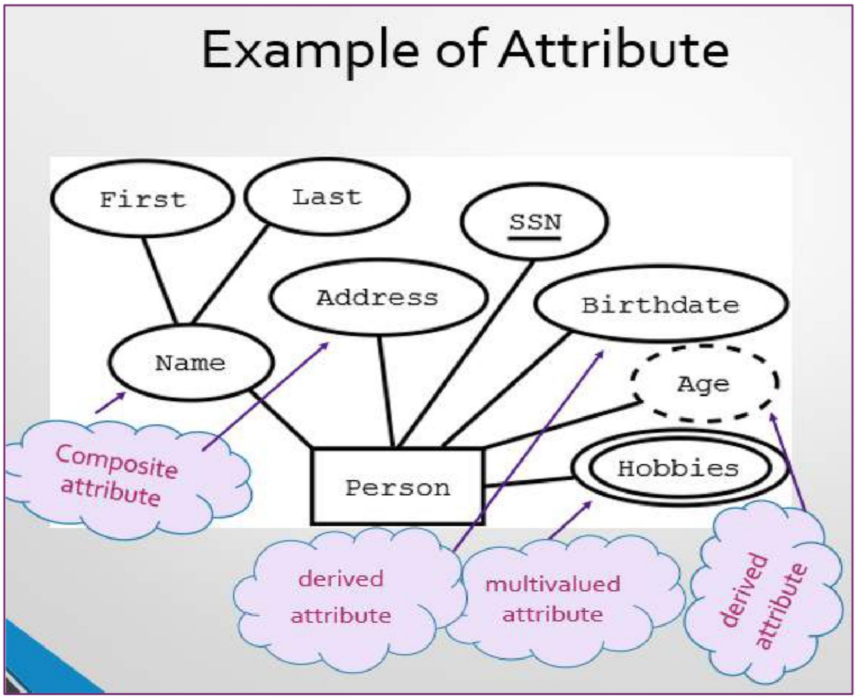

Participations
Cardinality can be shown or hidden
Mandatory
1 (0:1)
1 1 (1:1)
N (0:N)
1 N (1:N)
M (0:M)
1 M (1:M)
Optional
1 (0:1)
1 (1:1)
1 N (0:N)
N (1:N)
1 M (0:M)
1 M (1:M)
Recursive Relationship
Cardinality can be shown or hidden
1 (0:1)
1 1 (1:1)
N (0:N)
1 N (1:N)
M (0:M)
1 M (1:M)


Entity
An entity is an object or concept about which you want to store information.
Weak Entity
A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.

ENTITY IDENTITY	EXAMPLE
PERSON	STAFF, STUDENT, LECTURER,EMPLOYEE
PLACE	DISTRICT, TOWN, STATE
OBJECT	BUILDING, TOOL, PRODUCT
EVENT	SALE, REGISTRATION, APPLICATION
CONCEPT	COURSE, ACCOUNT

Guidelines for naming and defining entity types

- An attribute name is a noun
- An attribute name should be unique
- To make an attribute name unique and clear, each attribute name should follow a standard format
- Similar attributes of different entity types should use similar but distinguishing names.



IDENTIFIER

Characteristic of Identifier

- **Will not change in value**
- **Will not be null**

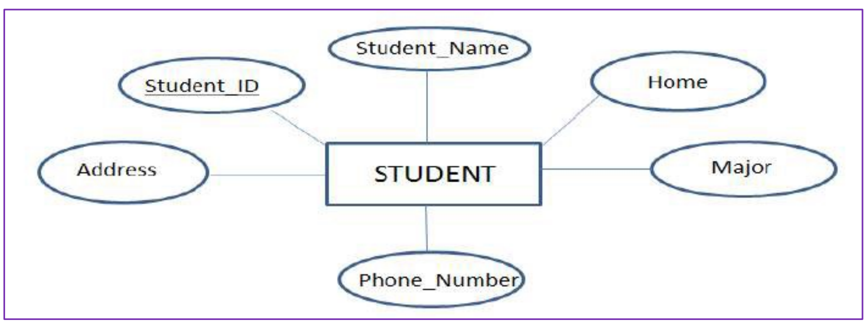
Candidate key

- An identifier that could be a key that satifies the requirements for being a key.
- Some entities may have more than one candidate key
  - Ex: A candidate key for EMPLOYEE is Employee_ID, a second is the combination of Employee_Name and Address.
  - If there is more than one candidate key, need to make a choice.

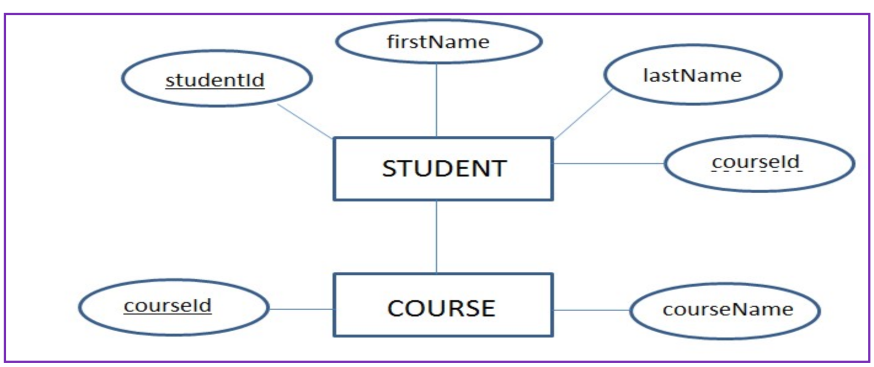
Identifier (Primary Key)

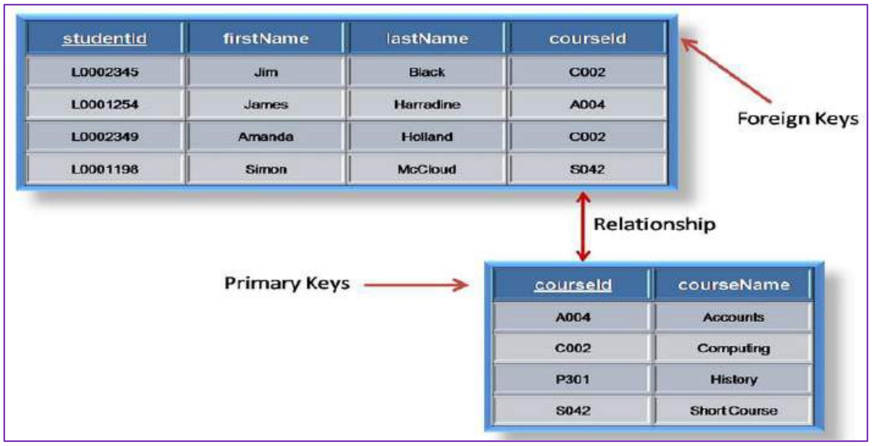
- A candidate key that has been selected as the unique identifying characteristic for an entity type

Identifier (Key)

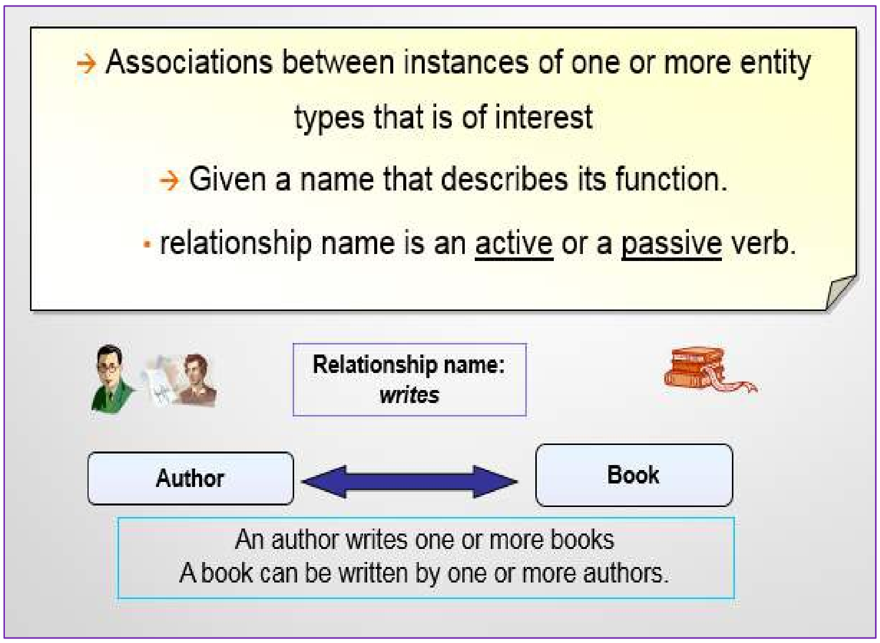


Referential (Key)



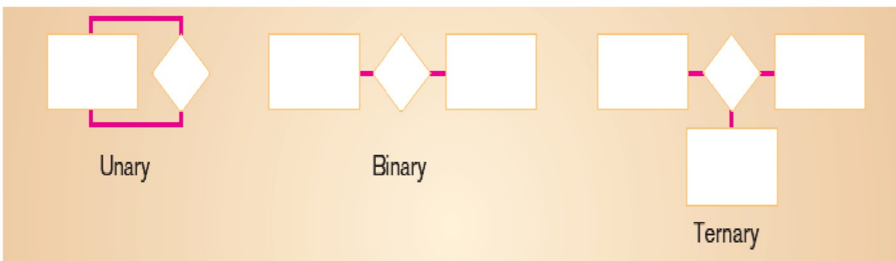


RELATIONSHIPS



Degree of Relationships

- Degree: number of entity types that participate in a relationship
- Three cases :
    - Unary: between two instances of one entity type
    - Binary: between the instances of two entity types
    - Ternary: among the instances of three entity types
    - Higher Degree



Cardinality and Connectivity

Connectivity

- one-to-many
- one-to-many
- many-to-many

Cardinality

- minimum and maximum number of instances of Entity B that can (or must be) associated with each instance of entity A.
- Cardinality specifies how many instances of an entity relate to one instance of another entity.

This is described by the *cardinality* of the relationship, for which there are four possible categories.

- One to one (1:1) relationship
- One to many (1:m) relationship
- Many to one (m:1) relationship
- Many to many (m:n) relationship

One to One Relationship (1:1)

- A single entity instance in one entity class is related to a single entity instance in another entity class.
- Example : Each student fills one seat and one seat is assigned to only one student.

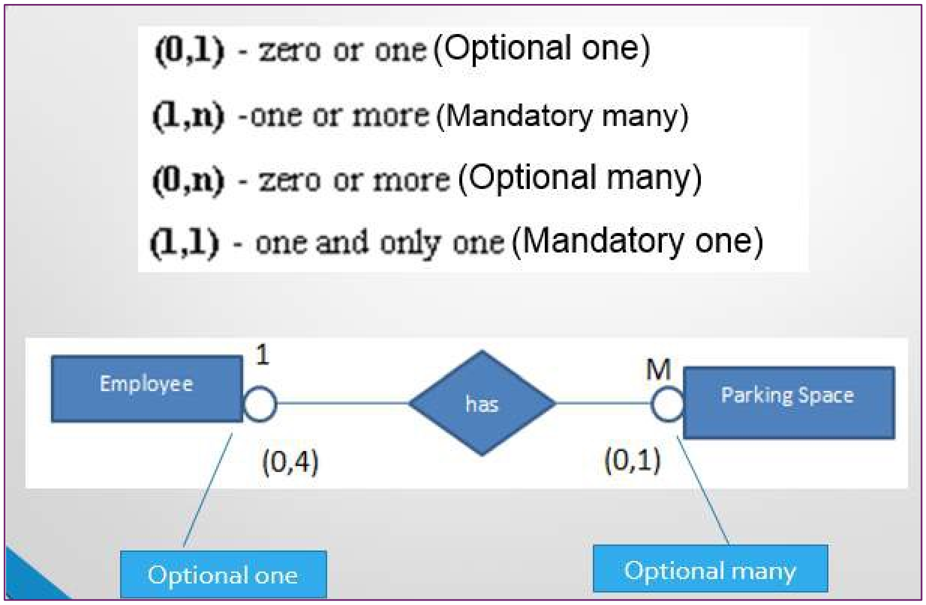
One to Many Relationship (1:M)

- A single entity instance in one entity class (parent) is related to multiple entity instances in another entity class (child)
- Example : One instructor can teach many courses, but one course can only be taught by one instructor

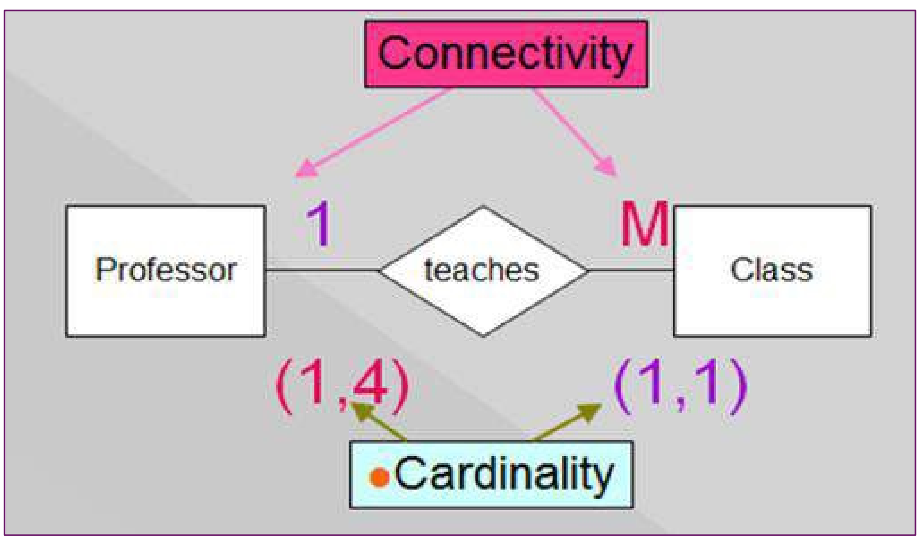
Many to Many Relationship (M:N)

- Each entity instance in one entity class is related to multiple entity instances in another entity class; and vice versa.
- Example : Each student can take many classes, and each class can be taken by many students.

Cardinality Optional



Cardinality Mandatory



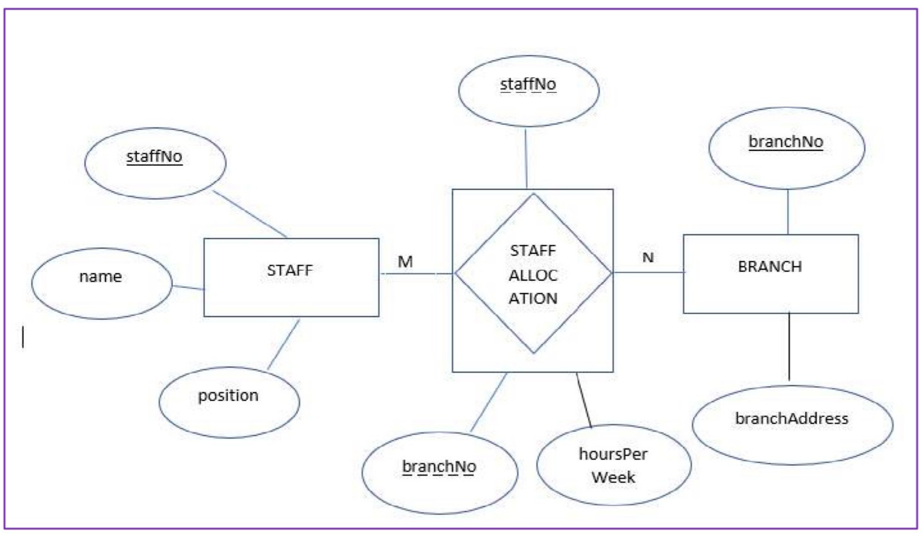
Associate Entities

Also known as Composite Entities or Bridge Entities
It's an entity – it has attributes
AND it's a relationship – it links entities together

When should a relationship with attributes instead be an associative entity?

- The relationship should be many-to-many.
- Composed of the primary keys of each of the entities to be connected
- May also contain additional attributes that play no role in the connective process

Examples of associate entity



NORMALIZATION

Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy

The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database using the defined relationships. Database Normalization Steps From 1NF to 3NF.

Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them

We have to normalize the database in order to make it easier to maintain, develop, or to resolve the error. It will be several steps to do, but usually it just only need till the third step.

The goal of a relational database design is to generate a set of relation scheme that allow us to store information easily.

Benefits of Normalization

- Normalization produces smaller tables with smaller rows
- Searching, sorting, and creating indexes is faster, since tables are narrower, and more rows fit on a data page.
- Normalization is conceptually cleaner and easier to maintain and change as your needs change.
- Data modification anomalies are reduced.
- More tables allow better use of segments to control physical placement of data.
- The cost of finding rows already in the data cache is extremely low.

Functional Dependency

Definition

- **A functional dependency occurs when one attribute in a relation uniquely determines another attribute. This can be written A -> B which would be the same as stating "B is functionally dependent upon A."**

Example

- **In a table listing employee characteristics including Social Security Number (SSN) and name, it can be said that name is functionally dependent upon SSN (or SSN -> name) because an employee's name can be uniquely determined from their SSN. However, the reverse statement (name -> SSN) is not true because more than one employee can have the same name but different SSNs.**

Transitive Dependencies

Definition

- **Transitive dependencies occur when there is an indirect relationship that causes a functional dependency**

Example

- **For example, ”A -> C” is a transitive dependency when it is true only because both “A -> B” and “B -> C” are true**

Example Of A transitive dependency occurs in the following relation:

Book	Genre	Author	Author Nationality
Twenty Thousand Leagues Under the Sea	Science Fiction	Jules Verne	French
Journey to the Center of the Earth	Science Fiction	Jules Verne	French
Leaves of Grass	Poetry	Walt Whitman	American
Anna Karenina	Literary Fiction	Leo Tolstoy	Russian
A Confession	Religious Autobiography	Leo Tolstoy	Russian

The functional dependency {Book} → {Author Nationality} applies; that is, if we know the book, we know the author's nationality. Furthermore:

- {Book} → {Author}
- {Author} does not → {Book}
- {Author} → {Author Nationality}

Therefore {Book} → {Author Nationality} is a transitive dependency.

Transitive dependency occurred because a non-key attribute (Author) was determining another non-key attribute (Author Nationality).

First Normal Form (1NF)

A table meets 1st Normal form if it doesn't have multivalued attribute, composite attribute or its combination in the same data domain.

Each attribute in that table should have an atomic value (can be divided).

There are no duplicated rows in the table.

Each cell is single-valued (i.e., there are no repeating groups or arrays).

Entries in a column (attribute, field) are of the same kind.

Steps to transform unnormalized to 1NF

Choose one attribute or a group of attribute to be the key in the table

Identify redundant groups in the unnormalized table

Delete the redundant groups

The example below doesn't meet the 1NF

Employee (emp_num, emp_lname, dept__no)

Employee

emp_num	emp_lname	dept_no
10052	Jones	A10 C66
10101	Sims	D60

Repeating

Normalization creates two tables and moves *dept_no* to the second table

Employee (emp_num, emp_lname)

Employee

emp_num	emp_lname
10052	Jones
10101	Sims

Emp_dept (emp_num, dept_no)

Emp_dept

emp_num	dept_no
10052	A10
10052	C66
10101	D60

Second Normal Form (2NF)

A table is in 2NF if it is in 1NF and if all non-key attributes are dependent on all of the key.

Since a partial dependency occurs when a non-key attribute is dependent on only a part of the (composite) key, the definition of 2NF is sometimes phrased as, "A table is in 2NF if it is in 1NF and if it has no partial dependencies."

A table meets 2NF when the 1NF requirement is met, and all attributes except the primary key have functional dependency entirely to the primary key

A table doesn't meet 2NF, if there is an attribute that it's functional dependency just partial. Partially dependent on primary key

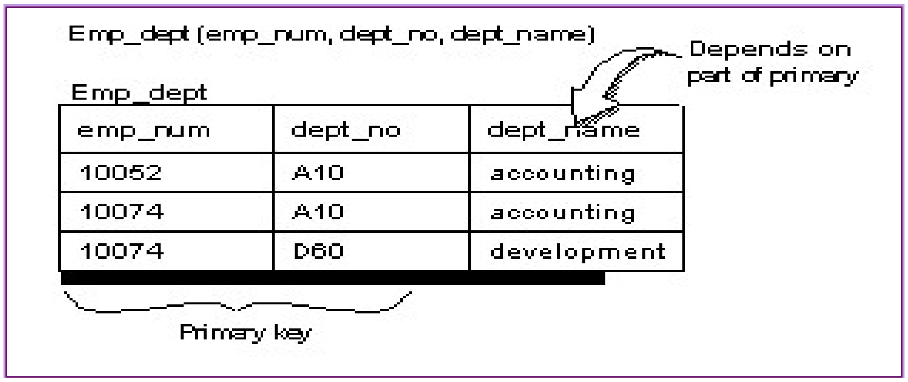
If there is an attribute that doesn't have a dependency to the primary key, then the attribute should be moved or deleted

Steps to transform 1NF to 2NF

Identify primary key to the 1NF relationship (based on the example above, the primary key is lesson_id)

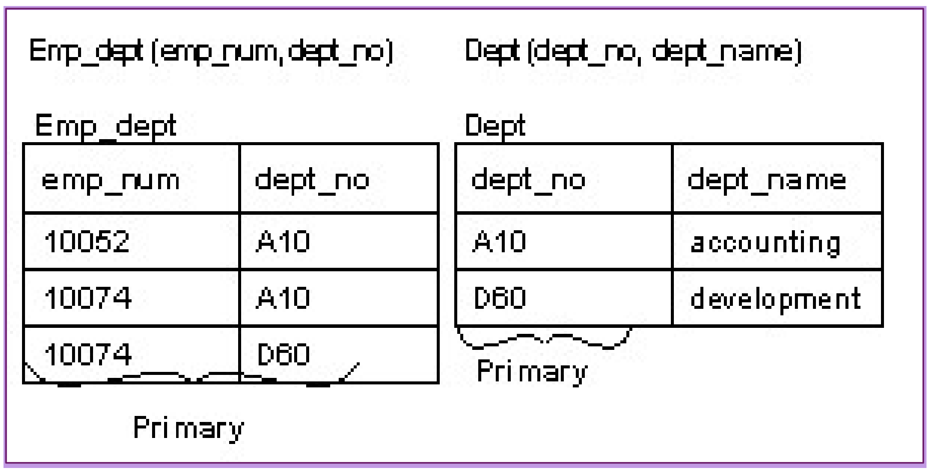
Identify functional dependencies in the relationship (the FD is lesson_id -> lesson_name)

If there is partial dependencies to the primary key, delete and place it into new relationship with the copy of its determinan (lesson_name is deleted from the table student and move to the new table)



emp_num	dept_no	dept_name
10052	A10	accounting
10074	A10	accounting
10074	D60	development

To normalize this table, move *dept_name* to a second table



emp_num	dept_no
10052	A10
10074	A10
10074	D60

dept_no	dept_name
A10	accounting
D60	development

3rd Normal Form (3NF)

A table is in 3NF if it is in 2NF and if it has no transitive dependencies.

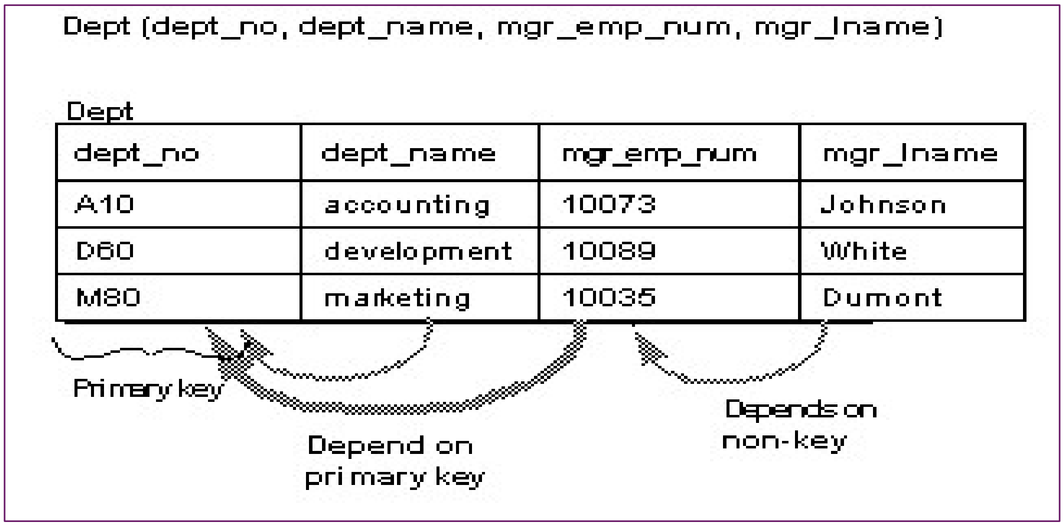
When it has met the 2NF, and there is no non primary key attribute that dependent to the other non primary key, the table is met 3NF.

Steps to transform 2NF to 3NF

Identify primary key in the 2NF relationship

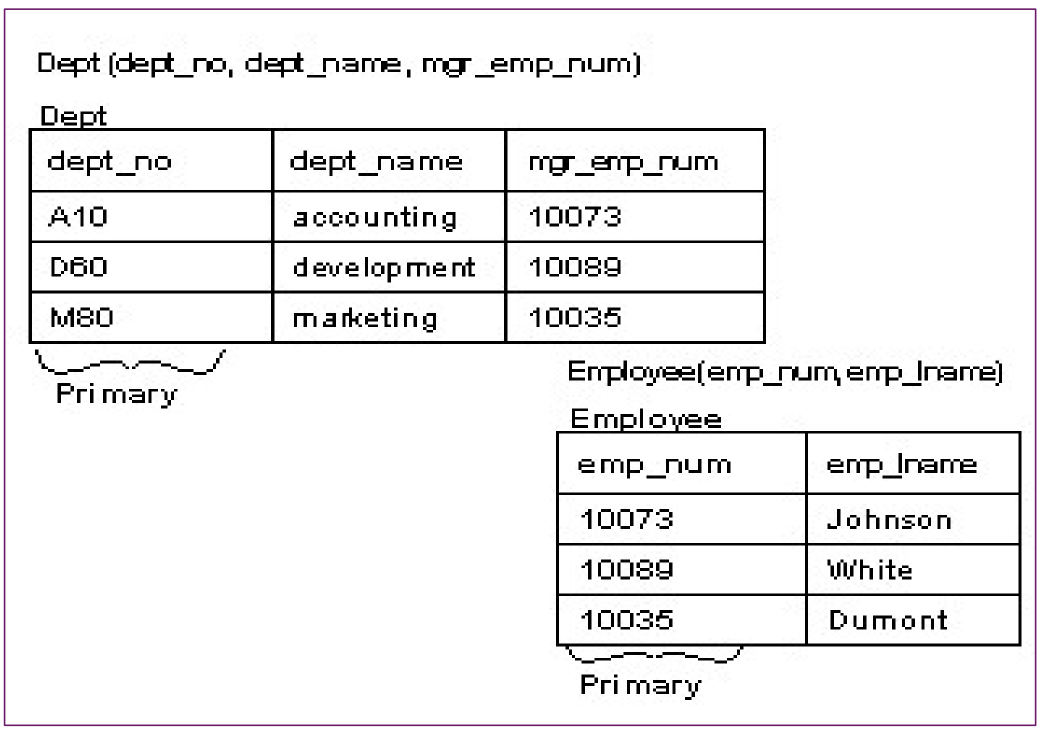
Identify functional dependencies in the relationship

If there is a transitive dependency to the primary key, delete and place it into new relationship with the copy of its determinan.



dept_no	dept_name	mgr_emp_num	mgr_lname
A10	accounting	10073	Johnson
D60	development	10089	White
M80	marketing	10035	Dumont

The solution is to split the *Dept* table into two tables. In this case, the *Employees* table, already stores this information, so removing the *mgr_lname* field from *Dept* brings the table into Third Normal Form.



dept_no	dept_name	mgr_emp_num
A10	accounting	10073
D60	development	10089
M80	marketing	10035

emp_num	emp_lname
10073	Johnson
10089	White
10035	Dumont

QUESTIONS

Chapter 3 Exercise: Entity Relationship Model and Normalization

1. What is a well-structured relation? Why must a database have well-structured relations?
2. What is ERD?

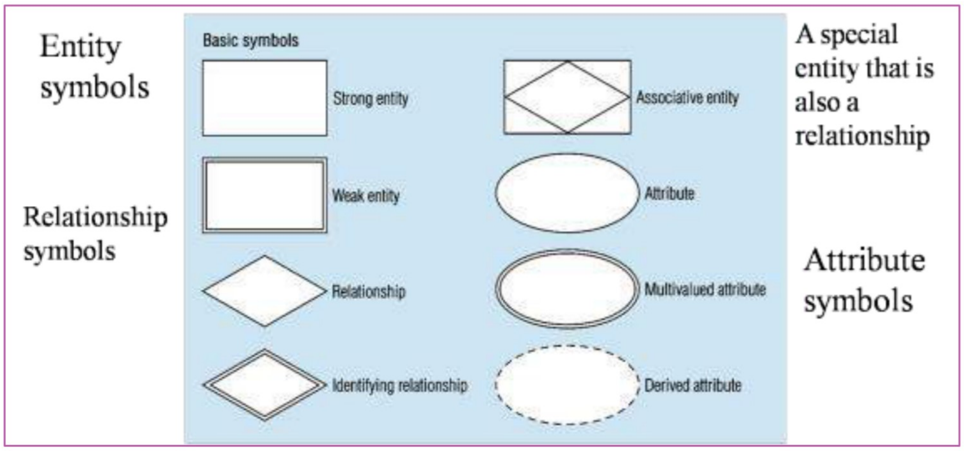


Figure 3.1: ERD symbol

3. Based on Figure 3.2, explain the ERD symbol below and give the example for each symbol.
   a. Entity
   b. Relationship
   c. Attributte

QUESTIONS

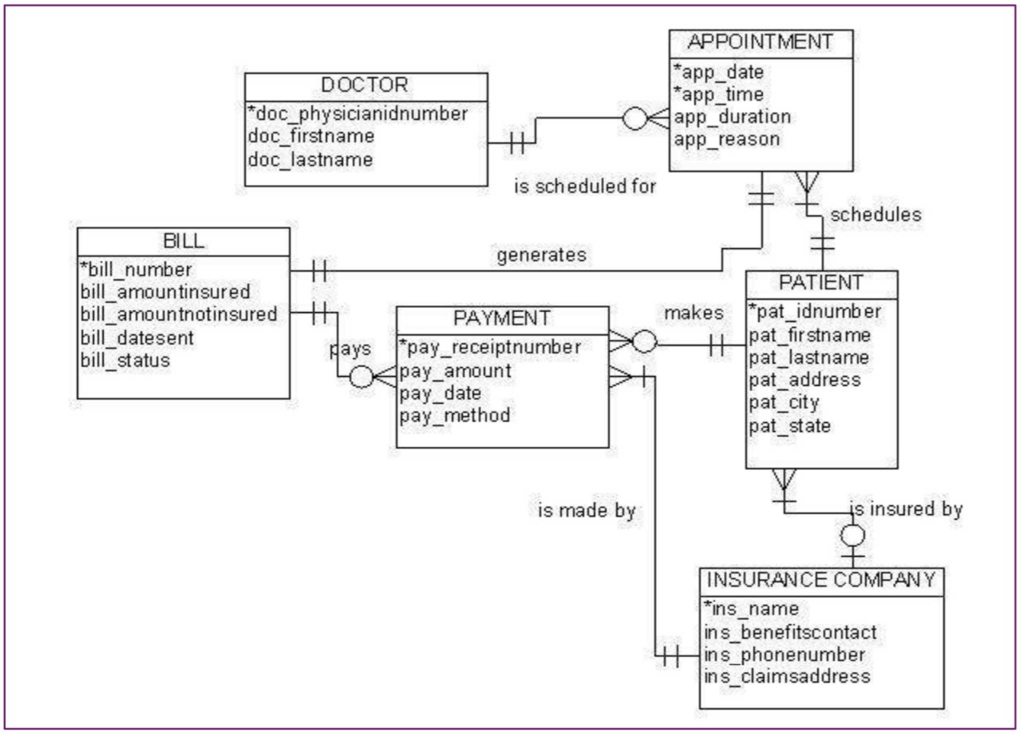


Figure 3.2: Insurance ERD

4. Based on Figure 3.2, convert this ERD using Chen Model;

   a. Entity
   b. Attribute
   c. Relationship
   d. Cardinality
   e. Keys

QUESTIONS

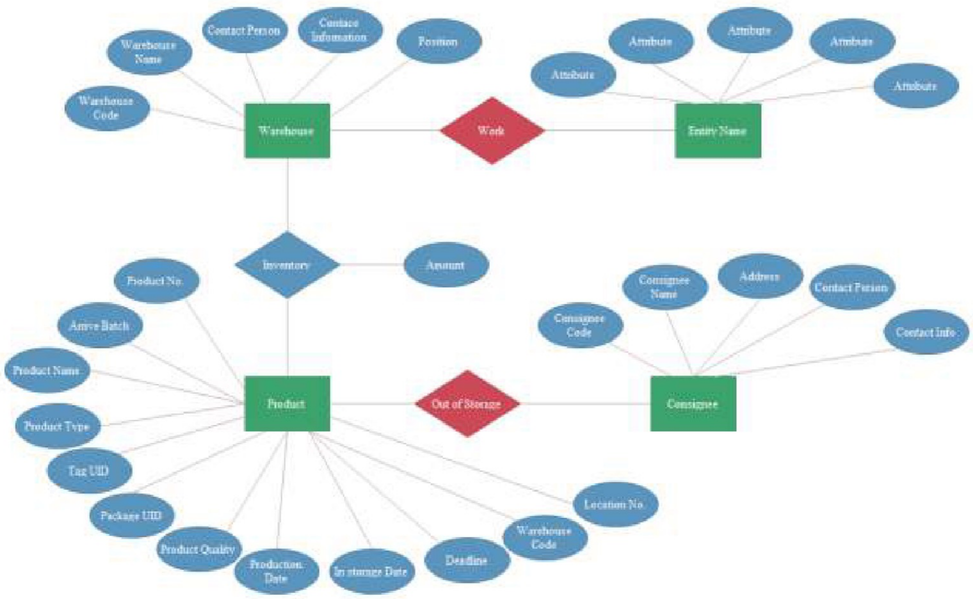


Figure 3.3: Warehouse ERD

5. Based on Figure 3.3, convert this ERD using using Crow Foot Model
   a. Entity
   b. Attribute
   c. Relationship
   d. Cardinality
   e. Keys

6. What is normalization
7. Why normalization need?
   a. Explain the process of normalization.
   b. Explain and give example of update anomalies. Types of update anomalies include:
      i. Insertion
      ii. Deletion
      iii. Modification

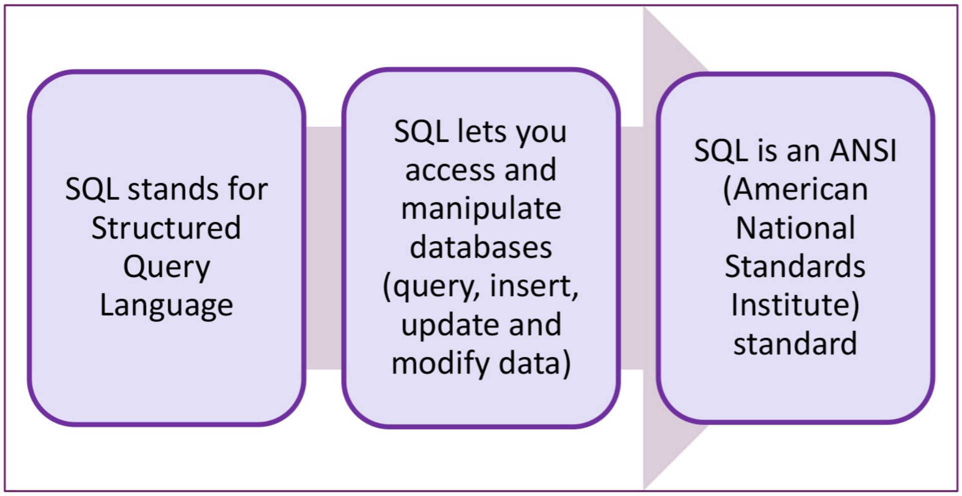
CHAPTER 4

STRUCTURED QUERY LANGUAGE

In this chapter, you will:

- **Apply SQL commands to a database**

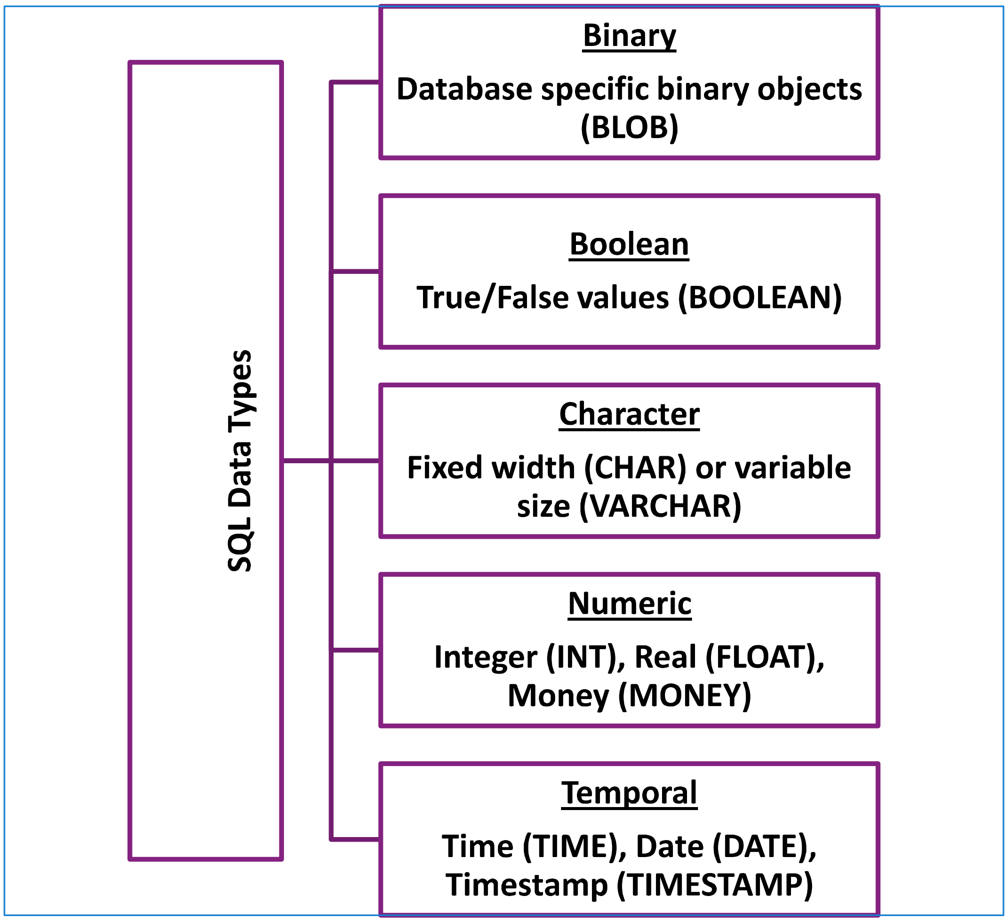
STRUCTURED QUERY LANGUAGE

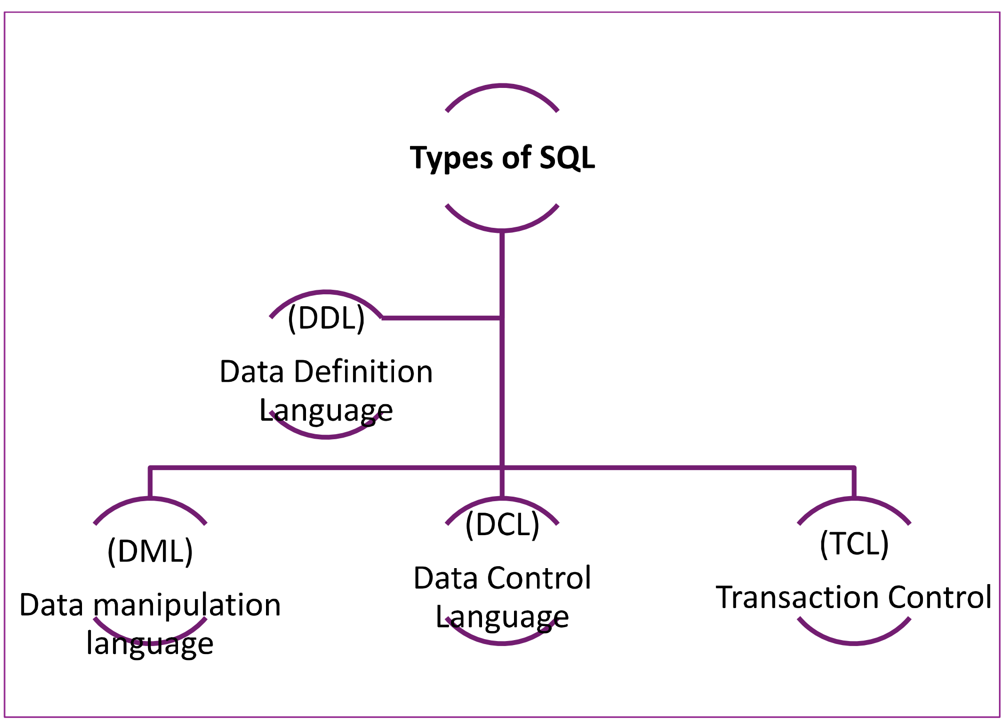


SQL DATA TYPES

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.


SQL Data Types
Binary
Database specific binary objects (BLOB)
Boolean
True/False values (BOOLEAN)
Character
Fixed width (CHAR) or variable size (VARCHAR)
Numeric
Integer (INT), Real (FLOAT), Money (MONEY)
Temporal
Time (TIME), Date (DATE), Timestamp (TIMESTAMP)


Types of SQL
(DDL)
Data Definition Language
(DML)
Data manipulation language
(DCL)
Data Control Language
(TCL)
Transaction Control

DDL

- Defining the database structure and controlling access the data.
- used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.
- Example: CREATE, ALTER, DROP, USE.

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.
- Example : SELECT, UPDATE, DELETE, INSERT INTO

DCL

- Is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,
- 1. **System:** This includes permissions for creating session, table, etc and all types of other system privileges.
- **2. Object:** This includes permissions for any command or query to perform any operation on the database tables.
- Example: GRANT, REVOKE.

TCL

- TCL stands for Transaction Control Language
- This command is used to manage the changes made by DML statements.
- TCL allows the statements to be grouped together into logical transactions
- Example : COMMIT, SAVEPOINT, ROLLBACK, SET TRANSACTION

BASIC DDL COMMAND

CREATE

- Installing a database management system (DBMS) on a computer allows to create and manage many independent databases.
- For example, to maintain a database of customer contacts for a sales department and a personnel database for HR department. The CREATE command can be used to establish each of these databases on the platform. For example, the command:
  CREATE DATABASE employees

USE

- The **USE** command allows to specify the work with within the DBMS. For example , to issue some commands that will affect the employees database, preface them with the following SQL command:
  USE employees
- It's important to always be conscious of the database that working in before issuing SQL commands that manipulate data.

ALTER

- Once created a table within a database, you may wish to modify the definition of it. The ALTER command allows 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. Finally, the "null" keyword tells the database that it's OK for this field to contain no value for any given employee.

DROP

- The final command of the Data Definition Language, DROP, 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

- Similarly, the command below would be used to remove the entire employees database:
  DROP DATABASE employees

- Use this command with care! Remember that the DROP command removes entire data structures from your database. If you want to remove individual records, use the DELETE command of the Data Manipulation Language.

SQL CONSTRAINTS

- SQL constraints are used to specify rules for the data in a table.
- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Syntax ;

```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   column3 datatype constraint,
   ....
);
```

Constraints are commonly used in SQL

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition

BASIC DML COMMAND

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, let's 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)**
- Note that there are four values specified for the record. These correspond to the table attributes in the order they were defined: first_name, last_name, employee_id, and salary.

SELECT

- The INSERT command in SQL is used to add records to an existing table. Returning to the personal_info example from the previous section, let's 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)**
- Note that there are four values specified for the record. These correspond to the table attributes in the order they were defined: first_name, last_name, employee_id, and salary.

DELETE

- The syntax of this command is similar to that of the other DML commands. Unfortunately, our latest corporate earnings report didn't quite meet expectations and poor Bart has been laid off. The DELETE command with a WHERE clause can be used to remove his record from the personal_info table:

  DELETE FROM personal_info
  WHERE employee_id = 12345

UPDATE

- The UPDATE command can be used to modify information contained within a table, either in bulk or individually. Each year, our company gives all employees a 3% cost-of-living increase in their salary. The following SQL command could be used to quickly apply this to all of the employees stored in the database:
  UPDATE personal_info
  SET salary = salary * 1.03

- On the other hand, our new employee Bart Simpson has demonstrated performance above and beyond the call of duty. Management wishes to recognize his stellar accomplishments with a $5,000 raise. The WHERE clause could be used to single out Bart for this raise:
  UPDATE personal_info
  SET salary = salary + $5000
  WHERE employee_id = 12345

SQL DATA DEFINITION COMMANDS

CREATE SCHEMA AUTHORIZATION	Create a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have duplicate value
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Define a primary key for a table
FOREIGN KEY	Define a foreign key for a table
DEFAULT	Defines a default value for a column (when no values is given)
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows / columns from one or more tables
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on query in user's database schema
DROP TABLE	Permanently deletes a table (thus its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

CREATE SCHEMA AUTHORIZATION	Create a database schema
INSERT	Inserts row(s) into table
SELECT	Select attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on one or more attributes
GROUP BY	Groups the selected rows based on the a conditional expression
HAVING	Restricts the selection grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies the attribute's values in one or more attributes
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restore data to their original values
COMPARISON OPERATORS	Used in conditional expressions
LOGICAL OPERATOR	Used in conditional expressions
AND / OR / NOT	Used in conditional expressions
SPECIAL OPERATORS	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range

CREATE SCHEMA AUTHORIZATION	Create a database schema
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXIST	Checks whether a sub query returns any rows
DISTINCT	Limits value to unique values
AGGREGATE FUNCTIONS	Used with SELECT to return mathematical summaries on column.
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column.

SQL QUERIES

With SQL, we can query a database and have a result set returned.

All queries are based on the SELECT command.

Syntax:

SELECT column_name(s)

FROM table_name;

** SELECT, FROM can be written in lower case.*

Example:

workerno	workername	position	address	entrydate	tel_no	salary
A01	JOHN	MANAGER	CHERAS	1995-01-01	0199292123	7000
A02	ANI	ASSISTANT	BANGI	1997-05-30	0132254040	2000
A03	DAVID	VICE MANAGER	BANGI	1995-05-01	0182852525	4000
A04	MARYAM	CLERK	AMPANG	1996-07-22	null	1000
A05	SALMAH	ACCOUNTANT	BANGI	1996-07-12	0174285445	2500
A06	JENNY	SYSTEM ANALYST	KAJANG	1996-07-30	0137878220	2500

Example:

<u>SELECT</u>

✓ Select certain columns:

SELECT workerno, workername FROM worker;

✓ Result:

workerno	workername
A01	JOHN
A02	ANI
A03	DAVID
A04	MARYAM
A05	SALMAH
A06	JENNY

✓ Select all columns:

SELECT * FROM worker;

✓ Result: *will display the entire table.*

SELECT DISTINCT STATEMENT

✓ The DISTINCT keyword is used to return only distinct (different) values.

✓ Consider this table: worker

workerno	workername	position	address	entrydate	tel_no	salary
A01	JOHN	MANAGER	CHERAS	1995-01-01	0199292123	7000
A02	ANI	ASSISTANT	BANGI	1997-05-30	0132254040	2000
A03	DAVID	VICE MANAGER	BANGI	1995-05-01	0182852525	4000
A04	MARYAM	CLERK	AMPANG	1996-07-22	null	1000
A05	SALMAH	ACCOUNTANT	BANGI	1996-07-12	0174285445	2500
A06	JENNY	SYSTEM ANALYST	KAJANG	1996-07-30	0137878220	2500

✓ If we use:

SELECT address FROM worker;

✓ Result:

address
CHERAS
BANGI
BANGI
AMPANG
BANGI
KAJANG

✓ If we use:

SELECT DISTINCT address FROM worker;

✓ Result:

address
CHERAS
BANGI
AMPANG
KAJANG

Calculated Field

✓ Example:

SELECT workerno, workername, salary /2

FROM worker;

- ✓ Result:

workerno	workername	Salary/2
A01	JOHN	5000.0000
A02	ANI	1000.0000
A03	DAVID	4000.0000
A04	MARYAM	3500.0000
A05	SALMAH	1750.0000
A06	JENNY	1750.000

Rename Column

- ✓ To rename a column, use AS statement.
- ✓ Example:

```
SELECT workerno AS Number,
workername AS Name
FROM worker;
```

- ✓ Result:

Number	Name
A01	JOHN
A02	ANI
A03	DAVID
A04	MARYAM
A05	SALMAH
A06	JENNY

SQL Where Clause

WHERE clause is to specify a selection criterion.

Syntax:

```
SELECT  column_name(s)
FROM    table_name
WHERE   conditions;
```

With WHERE clause, the following operators can be used:

in some versions of SQL,

<> operator may be written as !=

Operator	Description
=	Equal
< >	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
WILDCARDS or LIKE	Search for a pattern
IN	If you know that exact value want to return for at least one of the columns

Simple Queries

- ✓ List all the workers you earn more than 4000.

```
SELECT workername, salary
FROM worker
WHERE salary >4000;
```

- ✓ Result:

workername	salary
JOHN	10000
DAVID	8000
MARYAM	7000

- ✓ List all worker who live in Bangi or Kajang.

```
SELECT workername, address
FROM worker
WHERE address = 'Bangi'
OR address = 'Kajang‘;
```

- ✓ Result:

workername	address
ANI	BANGI
DAVID	BANGI
SALMAH	BANGI
JENNY	KAJANG

✓ List all the worker who earn between 3000 to 9000.

```
SELECT workername, salary
FROM worker
WHERE  salary BETWEEN 3000 AND 9000;
```

✓ Result:

workername	salary
DAVID	8000
MARYAM	7000
SALMAH	3500
JENNY	3500

BETWEEN…AND operator selects a range of data between two values

can be numbers, texts or dates.

✓ List the Director and Vice Director.

```
SELECT workername, position
        FROM worker
        WHERE position
        IN ('DIRECTOR', 'VICE DIRECTOR');
```

✓ Result:

workername	position
JOHN	DIRECTOR
DAVID	VICE DIRECTOR

IN can be used if you know the exact value that you seek for at least one of the columns.

- ✓ List the worker who is not living in Bangi.

```
SELECT    workername, address
FROM      worker
WHERE     address NOT IN ('BANGI');
Or
SELECT    workername, address
FROM      worker
WHERE     address <> 'BANGI‘;
```

- ✓ Result:

workername	address
JOHN	CHERAS
MARYAM	AMPANG
JENNY	KAJANG

- ✓ Find worker who doesn’t have phone number.
- ✓ Consider this table : worker

workerno	workername	position	address	entrydate	tel_no	salary
A01	JOHN	DIRECTOR	CHERAS	1995-01-01	0199292123	10000
A02	ANI	SECRETARY	BANGI	1997-05-30	0132254040	2000
A03	DAVID	VICE DIRECTOR	BANGI	1995-05-01	0182852525	8000
A04	MARYAM	MANAGER	AMPANG	1996-07-22	NULL	7000
A05	SALMAH	SYSTEM ANALYST	BANGI	1996-07-12	NULL	3500
A06	JENNY	ACCOUNTANT	KAJANG	1996-07-30	0137878220	3500

```
SELECT workername, tel_no
FROM worker
WHERE tel_no IS NULL;
```

✓ Result:

workername	tel_no
MARYAM	NULL
SALMAH	NULL

Using SQL % Wildcards

In SQL, wildcard characters are used with the SQL LIKE operator.

SQL wildcards are used to specify a search for a pattern in a column.

A "**%**" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Using **LIKE**

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT *
FROM Persons
WHERE FirstName
LIKE  'O%' ;
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT *
FROM Staff
WHERE FirstName
LIKE  '%a' ;
```

The following SQL statement will return persons with first names that contain the pattern 'la':

```
SELECT *
FROM Staff
WHERE FirstName
LIKE   '%la%' ;
```

✓ List all the building in Taman Kota.

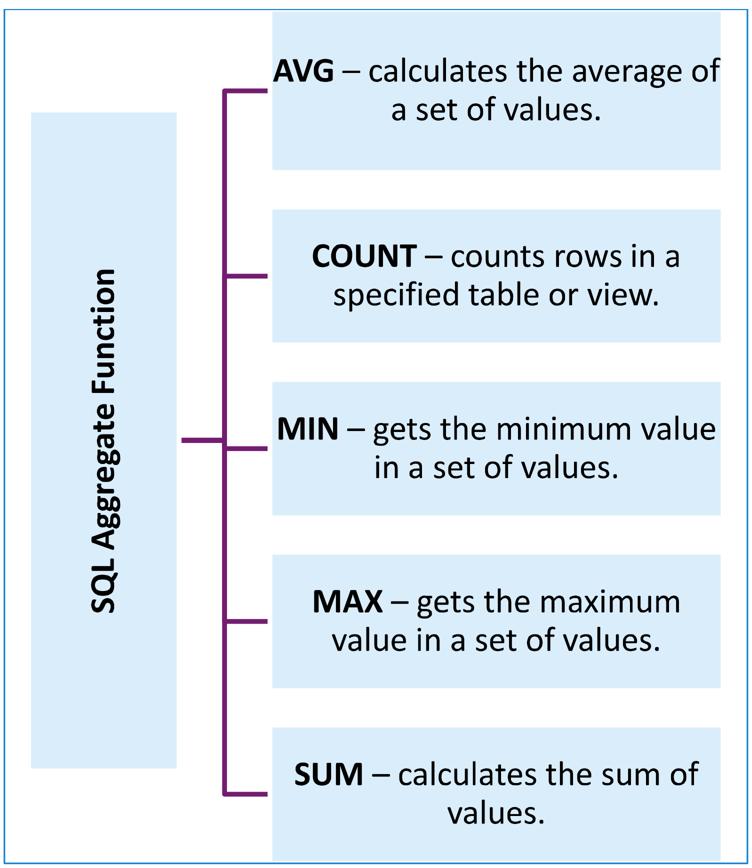
```
SELECT buildno, address
FROM building
WHERE address LIKE '%TAMAN KOTA%‘;
```

✓ Result:

buildno	address
B03	6, TAMAN KOTA
B04	2, TAMAN KOTA

SQL AGGREGATE FUNCTIONS

An aggregate function allows you to perform a calculation on a set of values to return a single scalar value. We often use aggregate functions with the GROUP BY and HAVING clauses of the SELECT statement.



SQL AGGREGATE FUNTION EXAMPLES

✓ **AVG** **SELECT AVG (unitsinstock)** **FROM products;**	✓ **COUNT** **SELECT COUNT(*)** **FROM products;**
✓ **MIN** **SELECT MIN (unitsinstock)** **FROM products;**	✓ **MAX** **SELECT MAX (unitsinstock)** **FROM products;**
✓ **SUM** **SELECT categoryid, SUM (unitsinstock)** **FROM products** **GROUP BY categoryid;**	

QUESTIONS

Chapter 4 Exercise: Structured Query Language

1. Explain the terms below;
   a. Data definition language (DDL)
   b. Data manipulation language (DML)
   c. Transaction control language (TCL)

2. Based on the figure 4.1, write SQL statement for the following:

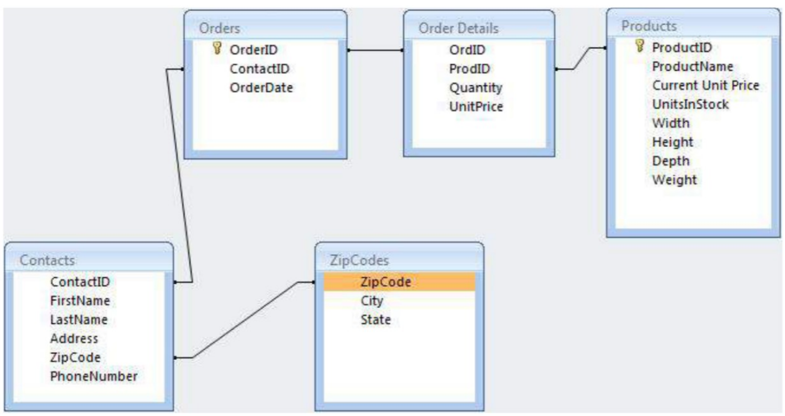


Figure 4.1 Store database

   a. Create a store database
   b. Create the table below with primary key
   c. Update table contacts and add new column state
   d. Update table order details and add a new column description
   e. Update table product and drop a depth column

3. Based on Figure 4.2, write SQL statement for the following:
   a. Find the total of cost, sales and profit
   b. Find the minimum and maximum for cost
   c. Count the number of product
   d. Count the number of product for stationary

ProductID	name	type	cost	sales	profit
S0001	eraser	stationary	0.20	0.50	0.30
S0002	pen	stationary	0.50	1.00	0.50
B0001	File	book	1.00	2.50	1.50
S0003	glue	stationary	0.70	1.50	0.80
S0004	Stapler	stationary	2.00	3.50	1.50
B0002	Learn ABC	book	2.50	4.00	1.50
B0003	Magazine	book	5.00	7.00	2.00

Figure 4.2 Product table

4. Based on Figure 4.3, write SQL statement for the following:

FName	Lname	City	Age	Salary (RM)
Hamizan	Kamal	Dungun	40	5000
Sarah	Firdaus	Kemaman	45	5500
Zainuddin	Abdullah	Dungun	29	3000
Hadi	Kamarul	Kemaman	27	2700
Haziq		Marang	43	4000

Figure 4.3 Employee table

a. Find the total average for age

b. Find the minimum and maximum for salary

c. Count the number of employee

d. Find the Fname that begin with H letter

e. Find the Lname that contains the pattern “a” in employee table.

CHAPTER 5

DATABASE TRANSACTION MANAGEMENT

In this chapter, you will:

- **Demonstrate database transaction management**

DATABASE TRANSACTION MANAGEMENT

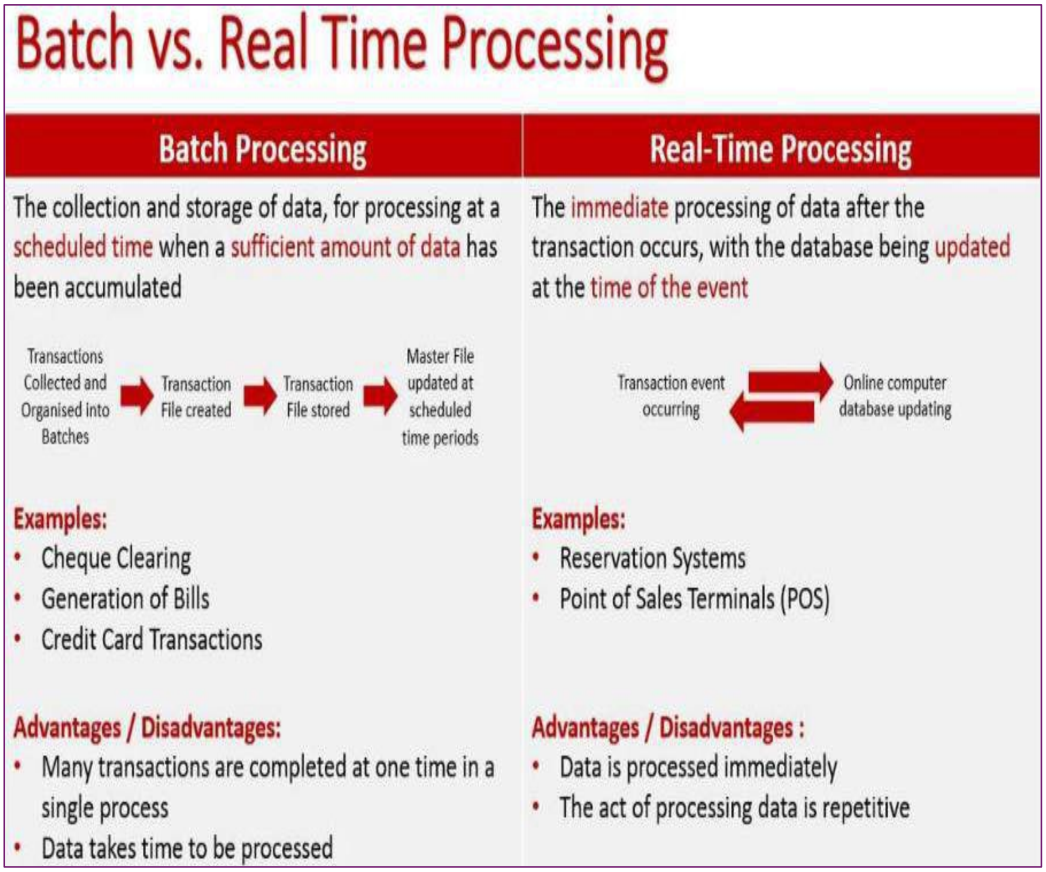
A **transaction** symbolizes a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in a database

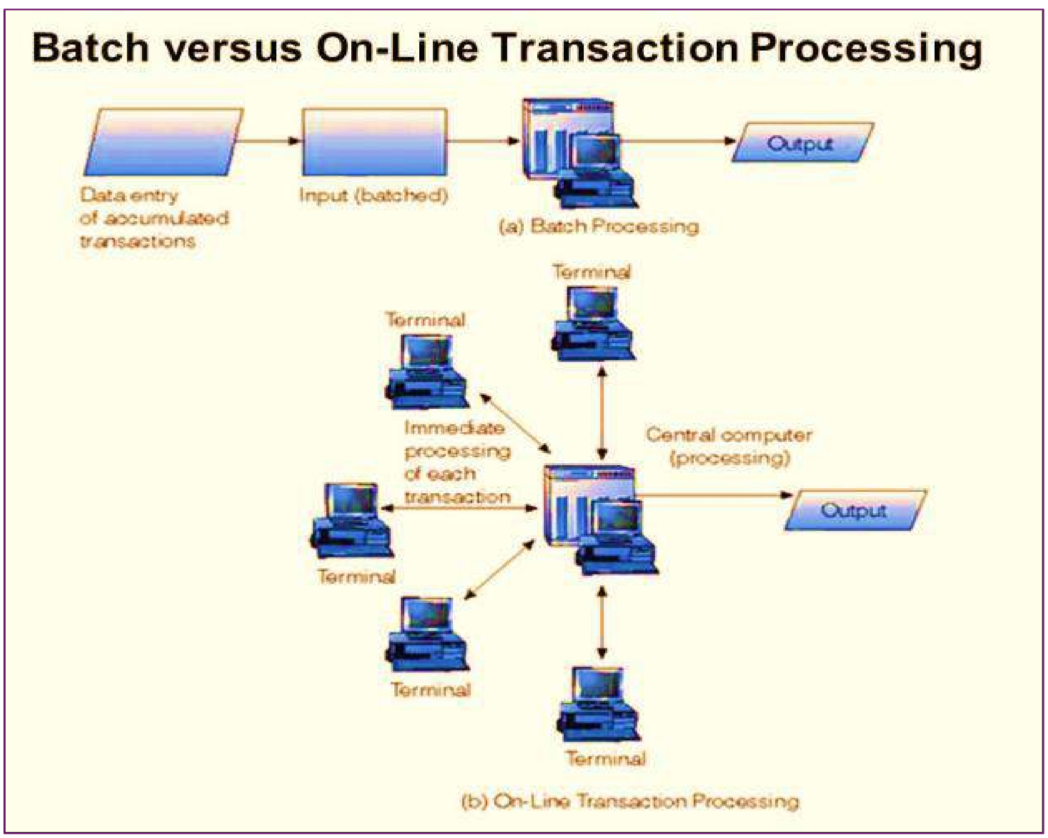
To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.

To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

DATABASE TRANSACTION MANAGEMENT

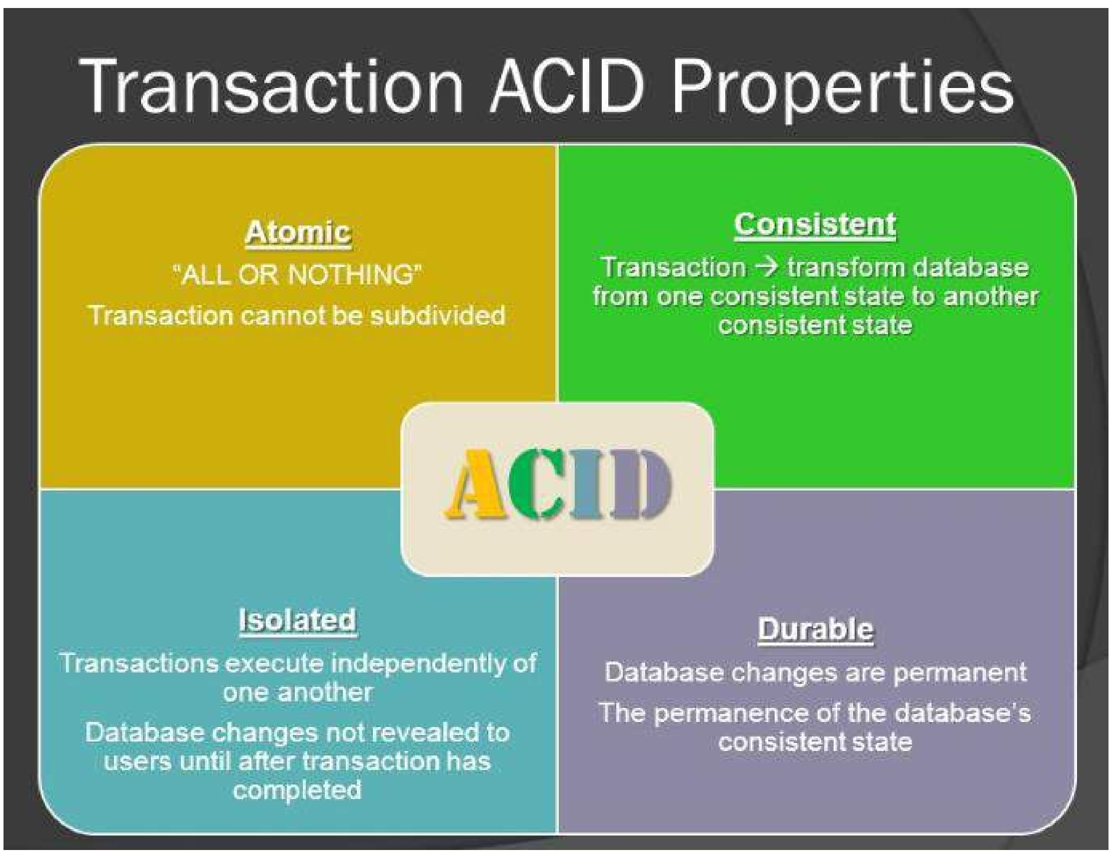
Batch Transaction	• Transactions are accumulated over a period of time and processed as a single unit, or batch. For example, a store may update its sales records every day after the store closes
On-line transaction (OLTP)	• OLTP database systems are commonly used for order entry, financial transactions, customer relationship management and retail sales via the Internet. Almost any business that has a large number of users who conduct short online transactions needs an OLTP system. Database queries with online transaction processing systems are simple, typically with sub-second response times and in most cases return relatively few records. OLTP databases need to operate in as close to real time as possible.
On-line transaction (OLTP)	• OLTP database systems are commonly used for order entry, financial transactions, customer relationship management and retail sales via the Internet. Almost any business that has a large number of users who conduct short online transactions needs an OLTP system. Database queries with online transaction processing systems are simple, typically with sub-second response times and in most cases return relatively few records. OLTP databases need to operate in as close to real time as possible.


Batch vs. Real Time Processing
Batch Processing
Real-Time Processing
The collection and storage of data, for processing at a scheduled time when a sufficient amount of data has been accumulated
The immediate processing of data after the transaction occurs, with the database being updated at the time of the event
Transactions Collected and Organised into Batches
Transaction File created
Transaction File stored
Master File updated at scheduled time periods
Transaction event occurring
Online computer database updating
Examples:
Cheque Clearing
Generation of Bills
Credit Card Transactions
Examples:
Reservation Systems
Point of Sales Terminals (POS)
Advantages / Disadvantages:
Many transactions are completed at one time in a single process
Data takes time to be processed
Advantages / Disadvantages :
Data is processed immediately
The act of processing data is repetitive

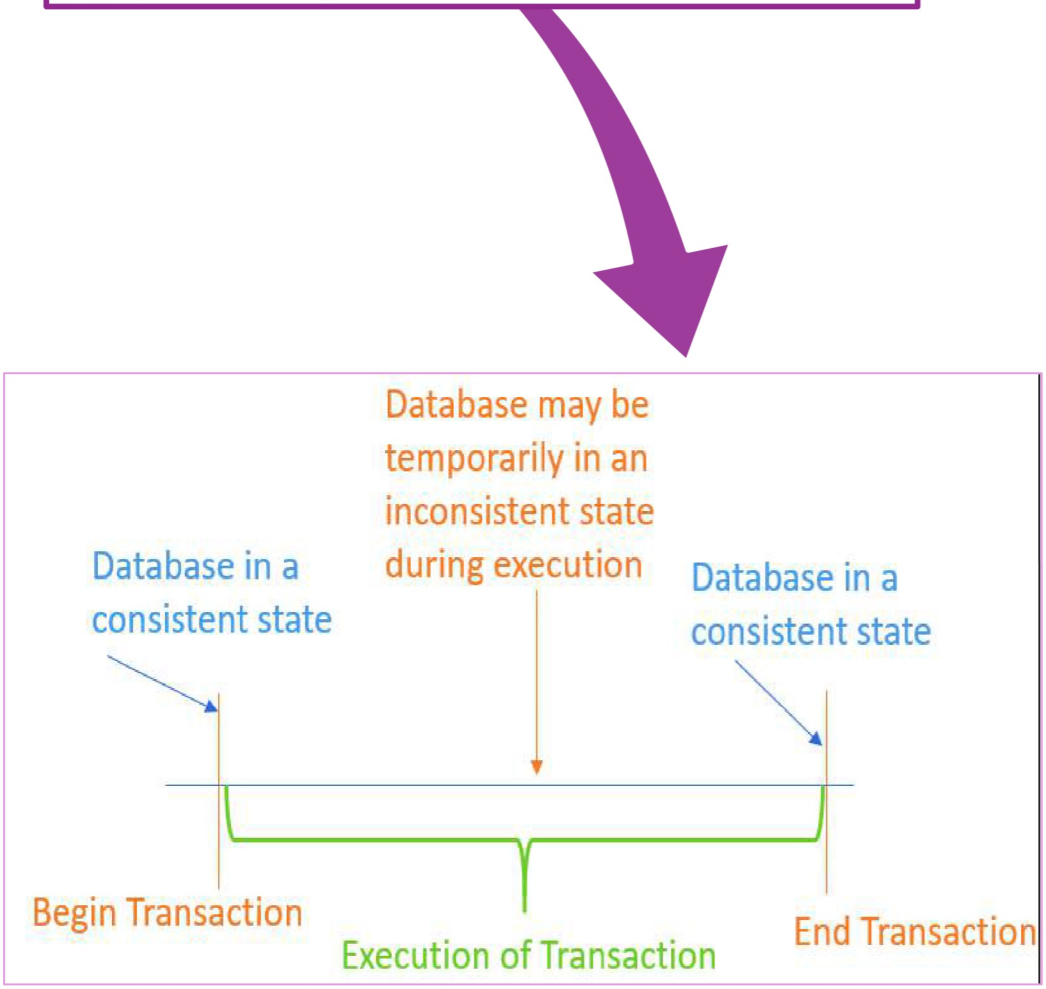

Batch versus On-Line Transaction Processing
Data entry of accumulated transactions
Input (batched)
Output
(a) Batch Processing
Terminal
Terminal
Immediate processing of each transaction
Central computer (processing)
Output
Terminal
Terminal
Terminal
(b) On-Line Transaction Processing

A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation and **D**urability commonly known as **ACID** properties

in order to ensure accuracy, completeness, and data integrity.



All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.



Transactions and SQL

- A transaction ends with a COMMIT, ROLLBACK, or disconnection (intentional or unintentional) from the database.
- A transaction begins with the first executable SQL statement after a COMMIT, ROLLBACK, or connection to the database
- Oracle issues an implicit COMMIT before and after any data definition language (DDL) statement.

Transaction Begins

```
UPDATE savings_accounts
     SET balance = balance - 500
     WHERE account = 3209;
```

Decrement Savings Account

```
UPDATE checking_accounts
     SET balance = balance + 500
     WHERE account = 3208;
```

Increment Checking Account

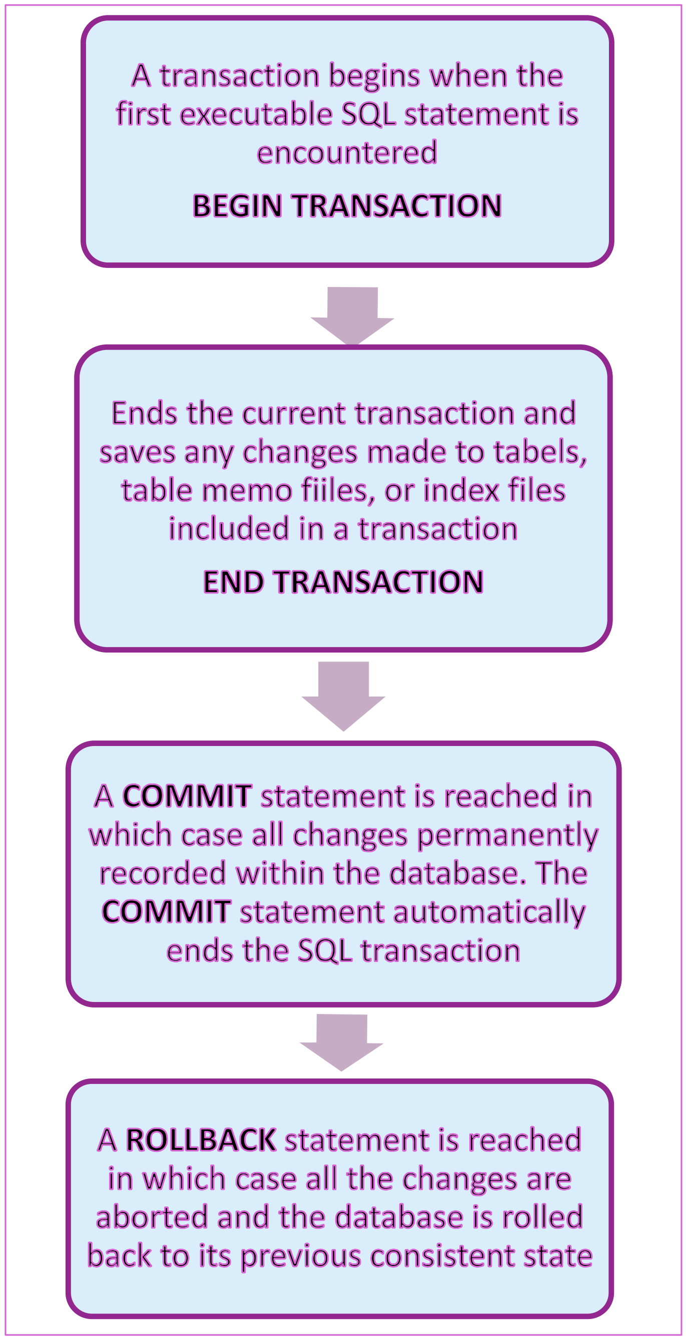
```
INSERT INTO journal VALUES
     (journal_seq.NEXTVAL, '1B'
     3209, 3208, 500);
```

Record in Transaction Journal

```
COMMIT WORK;
```

End Transaction

Transaction Ends


A transaction begins when the first executable SQL statement is encountered
BEGIN TRANSACTION
Ends the current transaction and saves any changes made to tabels, table memo fiiles, or index files included in a transaction
END TRANSACTION
A COMMIT statement is reached in which case all changes permanently recorded within the database. The COMMIT statement automatically ends the SQL transaction
A ROLLBACK statement is reached in which case all the changes are aborted and the database is rolled back to its previous consistent state

EXAMPLE

```
BEGIN TRANSACTION

UPDATE customers
SET ContactName='Jenn'
WHERE CustomerId = 'XYZ';

COMMIT TRANSACTION
```

These statement will writes directly to disk.

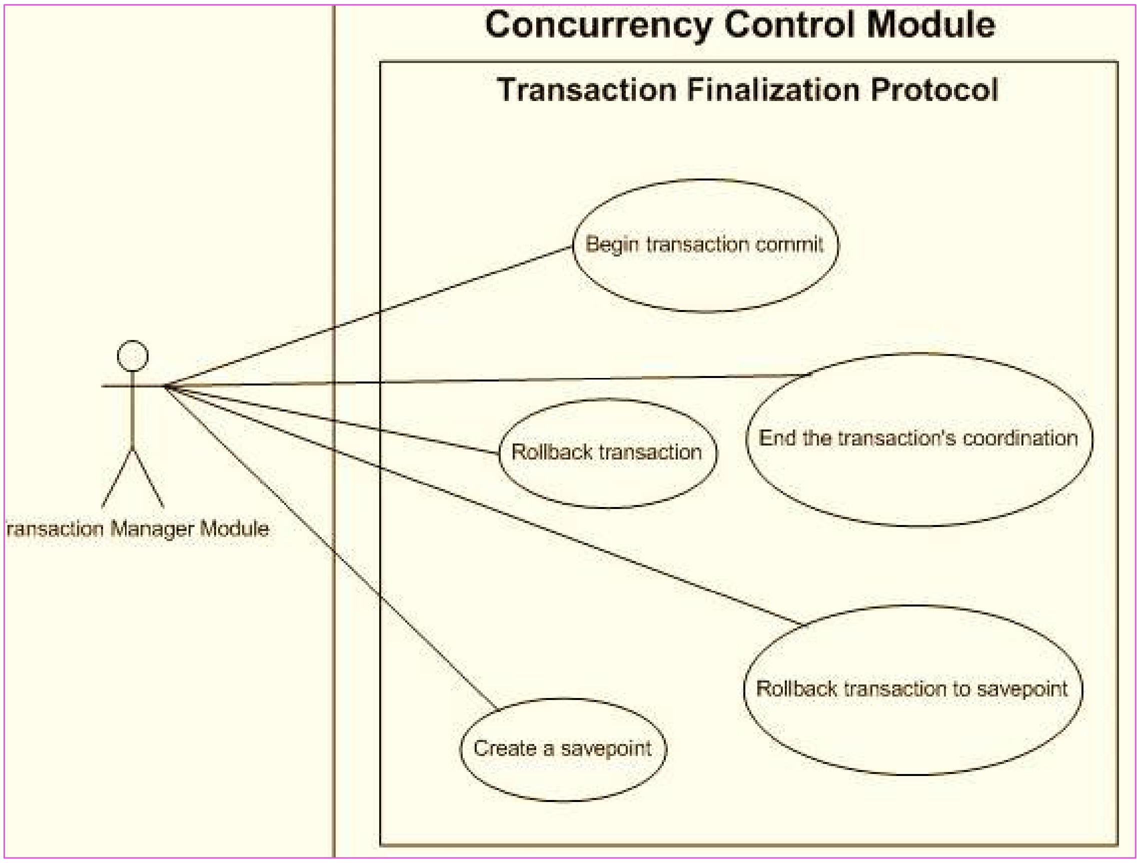
EXAMPLE

```
BEGIN TRANSACTION

UPDATE customers
SET ContactName='David'
WHERE CustomerId = 'XYZ';

ROLLBACK TRANSACTION
```

The ROLLBACK TRANSACTION statement “undoes” all the work since the matching BEGIN TRANSACTION

DATABASE TRANSACTION MANAGEMENT

- ✓ To allow many transactions to access the same data at the same time.
- ✓ Concurrency control mechanism is needed to ensure that concurrent transactions do not interfere with each other's operation.
- ✓ To ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- ✓ Example: Several reservation clerks try to assign a hotel room; the DBMS should ensure that only clerk could access each hotel room at a time for assignment to a customer.
- ✓ Process of managing simultaneous operations on the database without having them interfere with one another.



DATABASE TRANSACTION MANAGEMENT

Lost Update

- Successfully completed update overridden by another user
- Example: T1 withdrawing RM10 from an account with balX, initially RM100. T2 depositing RM100 into same account. Serially, final balance would be RM190.

Uncommitted Data

- Occurs when one transaction can see intermediate results of another transaction before it has committed
- Example: T1 updates balX to RM200 but it aborts, so balX should be back at original value of RM100. T3 has read new value of balX (RM200) and uses value as basis of RM10 reduction, giving a new balance of RM190, instead of RM90.

Inconsistent Retrieval

- Occurs when a transaction calculates an aggregate or summary function (e.g SUM) over a set of data, which the other transactions are updating
- The inconsistency happens because the transaction may read some data before they are changed and read other data after they are changed

The Scheduler

- Establishes the order in which the operations within concurrent transaction are executed.
- Interleaves the execution of database operations to ensure serializability
- To determine the appropriate order,the scheduler bases its actions on concurrency control algoritms such as locking or time stamping methods.

Transaction management problem 1: Lost updates

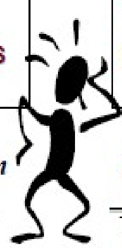
Initial balance $1000.00

Transaction 1:

Withdraw $400.00 from account 6676

Transaction 2:

Deposit $500.00 into account 6676



Correct execution of transaction

Time	TID	Step	Value stored
1	T1	Read Balance	1000.00
2	T1	Balance-400	
3	T1	Write Balance	600.00
4	T2	Read Balance	600.00
5	T2	Balance+500	
6	T2	Write Balance	1100.00

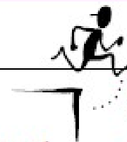
Lost update

Time	TID	Step	Value stored
1	T1	Read Balance	1000.00
2	T2	Read Balance	1000.00
3	T1	Balance-400	600.00
4	T2	Balance+500	1500.00
5	*T1*	*Write Balance*	*600.00*
6	T2	Write Balance	1500.00

Lost update

Transaction management problem 2: uncommitted data

Initial balance $1000.00



Transaction 1:

Start to withdraw $400.00 from account 6676; but decide against it and cancel transaction.

Transaction 2:

Deposit $500.00 in account 6676

Correct execution of transaction

Time	Tid	Step	Value stored
1	T1	Read Balance	1000.00
2	T1	Balance+400	
3	T1	Write Balance	600.00
4	T1	**ROLLBACK**	1000.00
4	T2	Read Balance	1000.00
5	T2	Balance+500	
6	T2	Write Balance	1500.00

Uncommitted data

Time	Tid	Stap	Value stored
1	T1	Read Balance	1000.00
2	T1	Balance-400	
3	T1	Write Balance	600.00
4	***T2***	***Read Balance***	***600.00***
5	T2	Balance+500	
6	T1	***ROLLBACK***	1000.00
7	T2	Write Balance	1100.00

Read uncommitted data

Transaction management problem 3: Inconsistent retrievals

T1: Select SUM(Quantity-on-Hand)
From Inventory;
COMMIT;

T2: Update Inventory
Set Quantity-on-Hand =
Quantity-on-Hand + 800
Where Product = "Towels";
Update Inventory
Set Quantity-on-Hand =
Quantity-on-Hand - 1000
Where Product = "Glass-bowls";
COMMIT;

Inconsistent retrievals:

Time	TID	Action	Value	Total
1	T1	Read Cutlery	1000	1000
2	T2	Read Towels	**1500**	
3	T1	Read Towels	1500	*2500*
4	T2	Towels = **1500** + 800	2300	
5	T1	Read glass bowls	*1001*	*3501*
6	T2	Read glass bowls	**1001**	
7	T2	Glass bowls = **1001** - 1000	1	
8	T2	***COMMIT***		
9	T1	Read duvets	*200*	***3701***
10	T1	***COMMIT**		

The solution:
Use a transaction scheduler

Determine order of concurrent execution

T1	T2	Conflict
Read	Read	no
Read	Write	yes
Write	Read	yes
Write	Write	yes

Scheduler ensures **serializability***:
The result of concurrent execution is equivalent to a serial execution

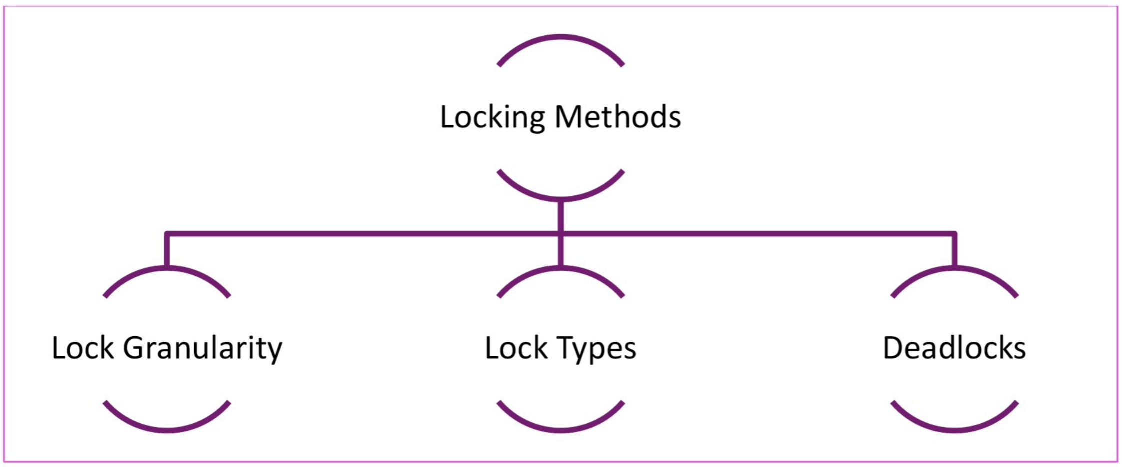
That is,
it appears as if the transactions are serially executed

*different types of serializability- next time

DATABASE TRANSACTION MANAGEMENT

LOCKING METHODS

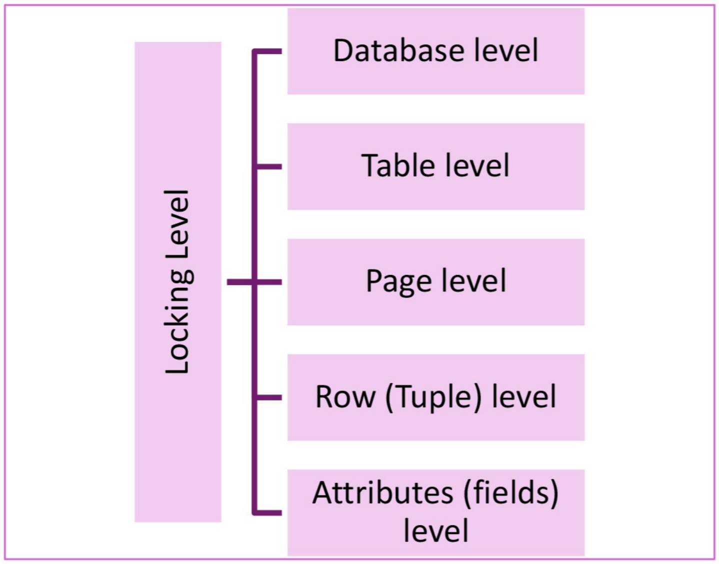
- A lock is a variable, associated with the data item, which controls the access of that data item. Locking is the most widely used form of the concurrency control.



1. Lock Granularity:

- A database is basically represented as a collection of named data items
- The size of the data item chosen as the unit of protection by a concurrency control program is called GRANULARITY

Lock Granularity



Database level locking

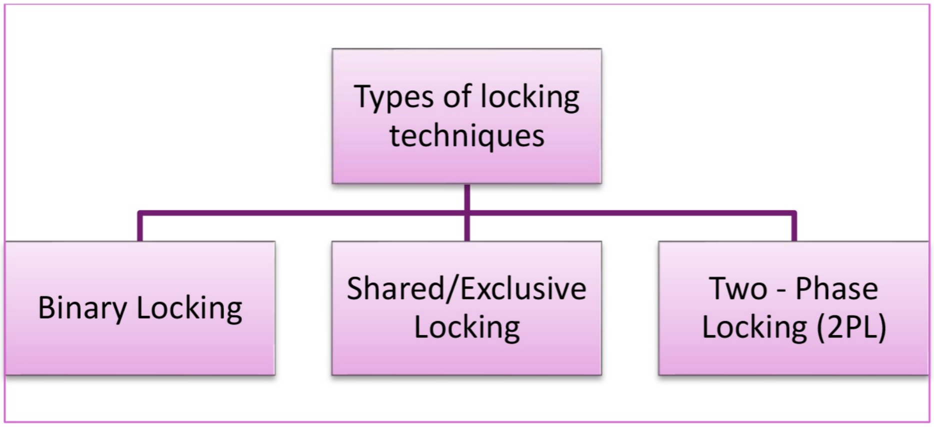
- At database level locking, the entire database is locked. Thus, it prevents the use of any tables in the database by transaction T2 while transaction T1 is being executed.
- Database level of locking is suitable for batch processes. Being very slow, it is unsuitable for on-line multi-user DBMSs.

Table level locking

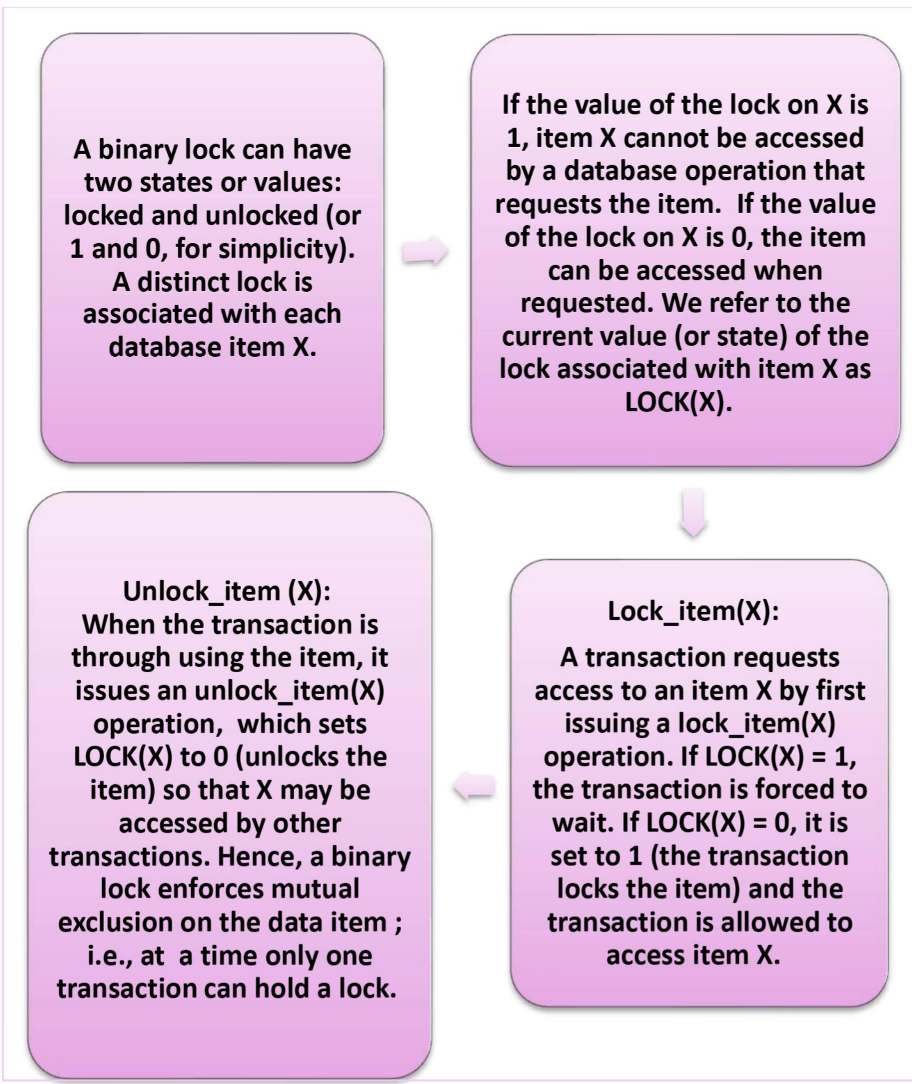
- At table level locking, the entire table is locked. Thus, it prevents the access to any row (tuple) by transaction T2 while transaction T1 is using the table. if a transaction requires access to several tables, each table may be locked.
- However, two transactions can access the same database as long as they access different tables. Table level locking is less restrictive than database level. Table level locks are not suitable for multi-user DBMS

Page level locking	• At page level locking, the entire disk-page (or disk-block) is locked. A page has a fixed size such as 4 K, 8 K, 16 K, 32 K and so on. A table can span several pages, and a page can contain several rows (tuples) of one or more tables. Page level of locking is most suitable for multi-user DBMSs.
Row (Tuple) level Locking	• At row level locking, particular row (or tuple) is locked. A lock exists for each row in each table of the database. The DBMS allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page • The row level lock is much less restrictive than database level, table level, or page level locks. The row level locking improves the availability of data. However, the management of row level locking requires high overhead cost.
Attributes (fields) level Locking	• At attribute level locking, particular attribute (or field) is locked. Attribute level locking allows concurrent transactions to access the same row, as long as they require the use of different attributes within the row. The attribute level lock yields the most flexible multi-user data access. It requires a high level of computer overhead.

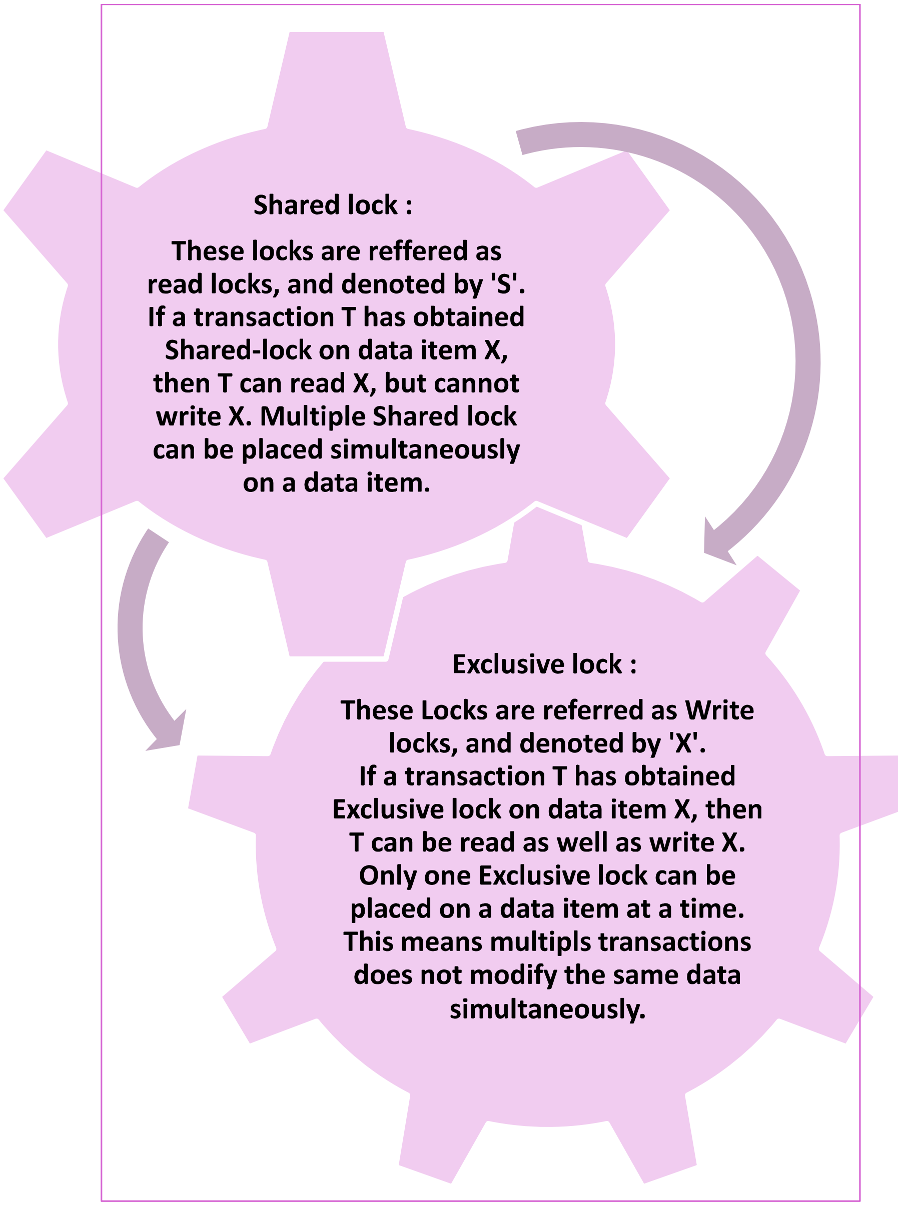
2. Lock Types:



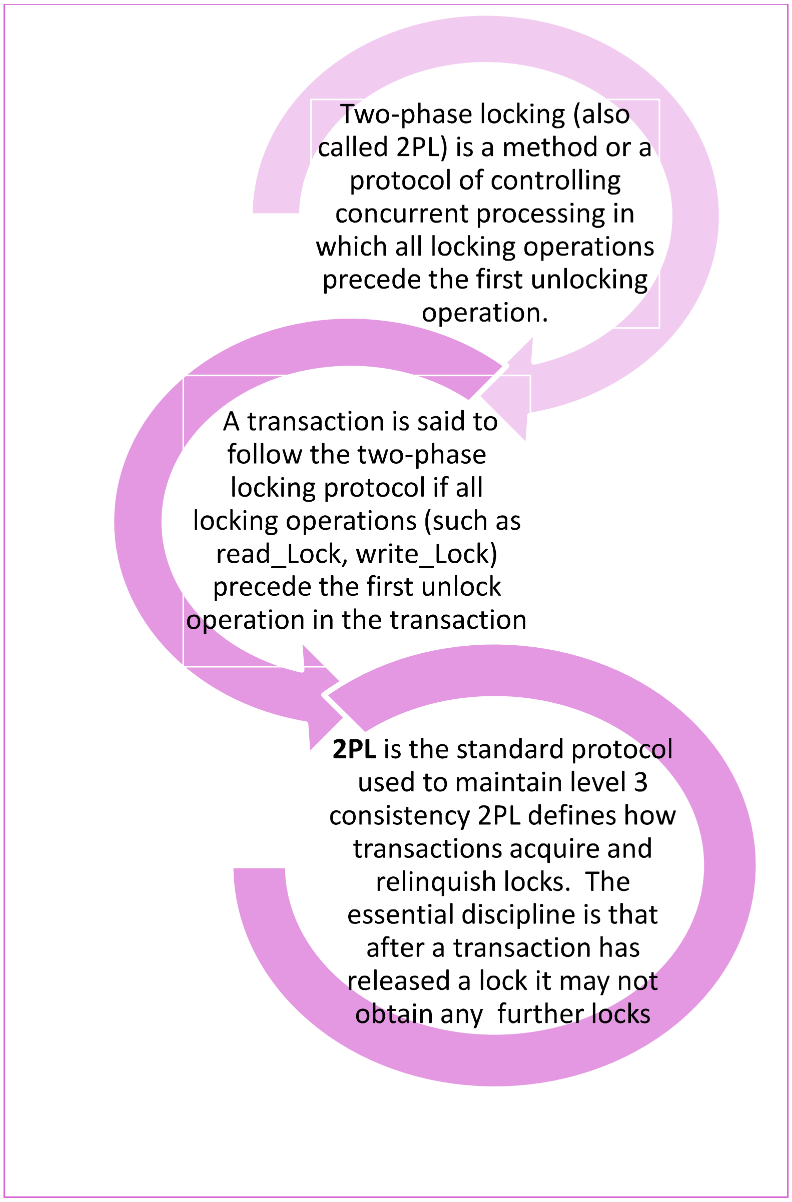
a. Binary Locking

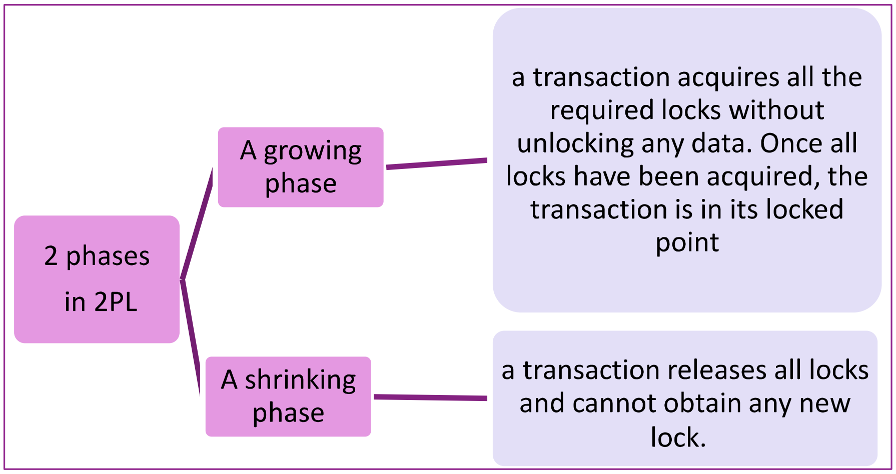


b. Shared / Exclusive Locking



c. Two - Phase Locking (2PL)

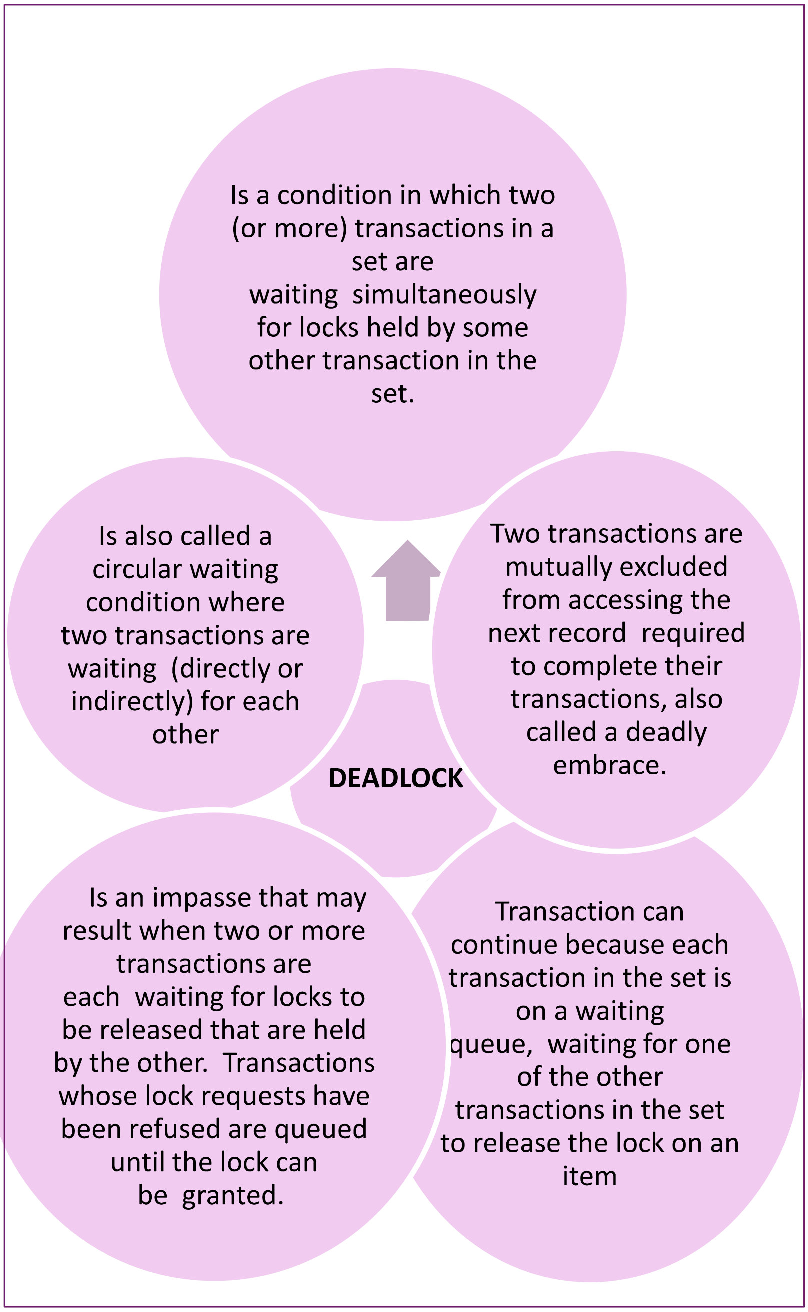




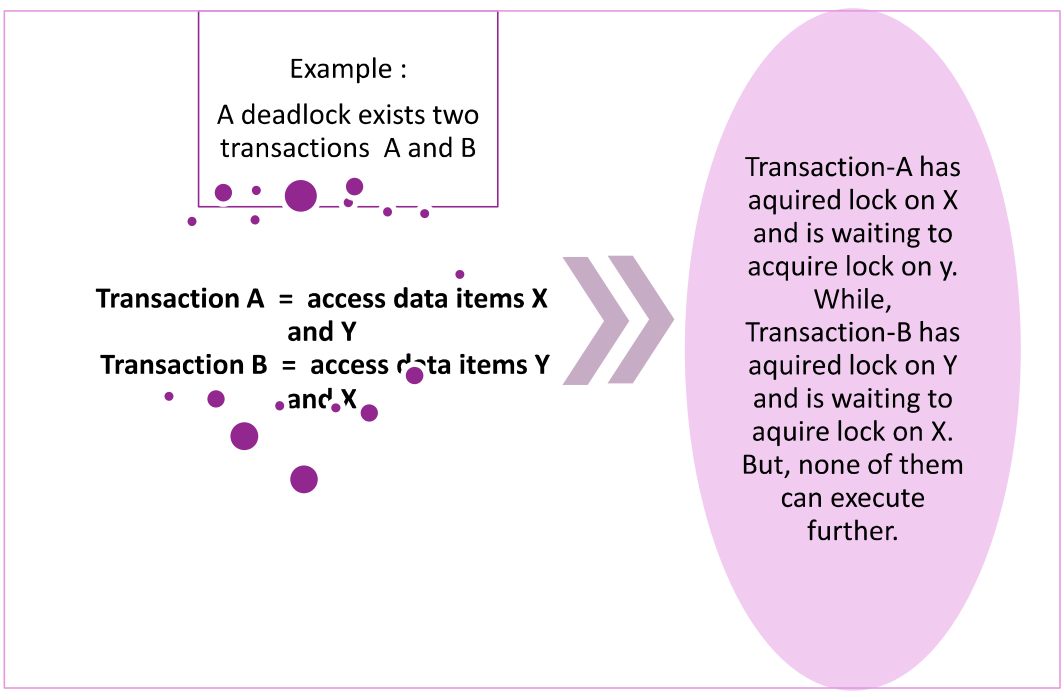
A transaction shows Two-Phase Locking technique.

Time	Transaction	Remarks
t0	Lock - X (A)	acquire Exclusive lock on A.
t1	Read A	read original value of A
t2	A = A - 100	subtract 100 from A
t3	Write A	write new value of A
t4	Lock - X (B)	acquire Exclusive lock on B.
t5	Read B	read original value of B
t6	B = B + 100	add 100 to B
t7	Write B	write new value of B
t8	Unlock (A)	release lock on A
t9	Unock (B)	release lock on B

3. Deadlocks:


Is a condition in which two (or more) transactions in a set are waiting simultaneously for locks held by some other transaction in the set.
Is also called a circular waiting condition where two transactions are waiting (directly or indirectly) for each other
Two transactions are mutually excluded from accessing the next record required to complete their transactions, also called a deadly embrace.
DEADLOCK
Is an impasse that may result when two or more transactions are each waiting for locks to be released that are held by the other. Transactions whose lock requests have been refused are queued until the lock can be granted.
Transaction can continue because each transaction in the set is on a waiting queue, waiting for one of the other transactions in the set to release the lock on an item

Transaction-A	Time	Transaction-B
---	t0	---
Lock (X) (acquired lock on X)	t1	---
---	t2	Lock (Y) (acquired lock on Y)
Lock (Y) (request lock on Y)	t3	---
Wait	t4	Lock (X) (request lock on X)
Wait	t5	Wait
Wait	t6	Wait
Wait	t7	Wait



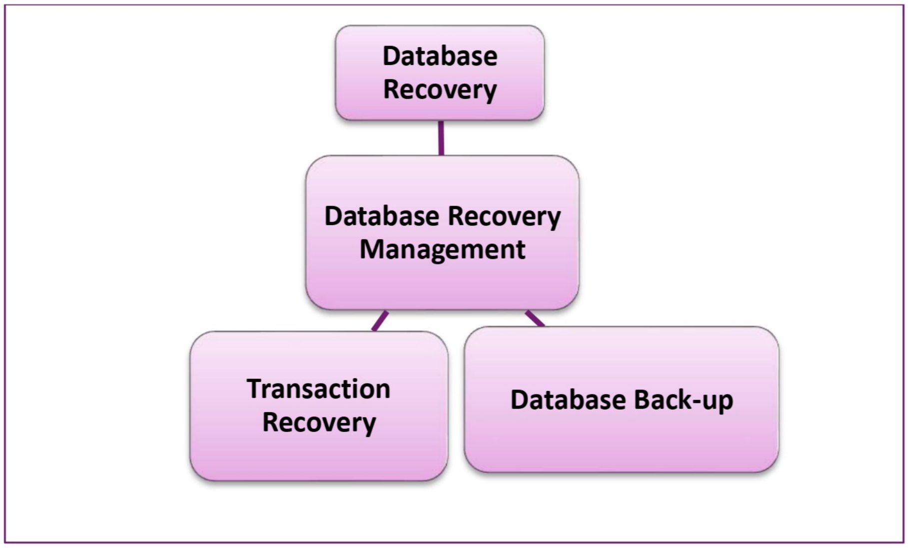
Deadlock detection

- This technique allows deadlock to occur, but then, it detects it and solves it
- Here, a database is periodically checked for deadlocks
- If a deadlock is detected, one of the transactions, involved in deadlock cycle, is aborted. other transaction continue their execution
- An aborted transaction is rolled back and restarted.

Deadlock Prevention

- Deadlock prevention technique avoids the conditions that lead to deadlocking. It requires that every transaction lock all data items it needs in advance
- If any of the items cannot be obtained, none of the items are locked. In other words, a transaction requesting a new lock is aborted if there is the possibility that a deadlock can occur.
- Thus, a timeout may be used to abort transactions that have been idle for too long.
- If the transaction is aborted, all the changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution

DATABASE TRANSACTION MANAGEMENT



Database Recovery

-> Restore a database from a given state to a previous consistent state

-> Atomic Transaction Property (All or None)

-> Backup Levels:

* Full Backup
* Differential Backup
* Transaction Log Backup

-> Database / System Failures:

* Software (O.S., DBMS, Application Programs, Viruses)
* Hardware (Memory Chips, Disk Crashes, Bad Sectors)
* Programming Exemption (Application Program rollbacks)
* Transaction (Aborting transactions due to deadlock detection)
* External (Fire, Flood, etc)

Transaction Recovery

-> Recover Database by using data in the Transaction Log

-> Write-Ahead-Log – Transaction logs need to be written before any database data is updated

-> Redundant Transaction Logs – Several copies of log on different devices

-> Database Buffers – Buffers are used to increase processing time on updates instead of accessing data on disk

-> Database Checkpoints – Process of writing all updated buffers to disk → While this is taking place, all other requests are not executes

* Scheduled several times per hour
* Checkpoints are registered in the transaction log

Database Backup

-> Database backup is a way to protect and restore a database. It is performed through database replication and can be done for a database or a database server.

-> Typically, database backup is performed by the RDBMS or similar database management software.

-> Database administrators can use the database backup copy to restore the database to its operational state along with its data and logs. The database backup can be stored locally or on a backup server.

-> Database backup is also created/performed to ensure a company's compliance with business and government regulations and to maintain and ensure access to critical/essential business data in case of a disaster or technical outage

QUESTIONS

Chapter 5 Exercise: Database Transaction Management

1. What is transaction?
2. Explain the properties of transaction:

Figure 5.1 Properties of transaction

3. What is concurrency control?
4. Explain concurrency control algorithm.
5. Why database security is so important? Discuss the impact of a database failure in (a) an airline, (b) a bank and (c) a politeknik
6. Discuss some of the main technique used to recover from a database failure.

E-DATABASE DESIGN

Reviewing basic concepts of databases and database design, then turns to creating, populating, and retrieving data using SQL. Topics such as Database Management System, the relational data model, Entity Relationship Diagram, normalization, data entities, and database transaction management are covered clearly and concisely. This book provides the conceptual and practical information necessary to develop a database design and management scheme that ensures data accuracy and user satisfaction while optimizing performance.

HIGHLIGHTS

Database
Database Management System (DBMS)
Data Model
Relational Data Model
Entity Relationship Model
Normalization
Structured Query Language
Database Transaction Management

e ISBN 978-967-2099-74-1