Subject: Database Management Systems(DBMS)

By: Mr.Loukik S.Salvi

Topics to be covered:

- Databases
- DBMS
- DBMS-Defination
- DBMS-overview

Databases

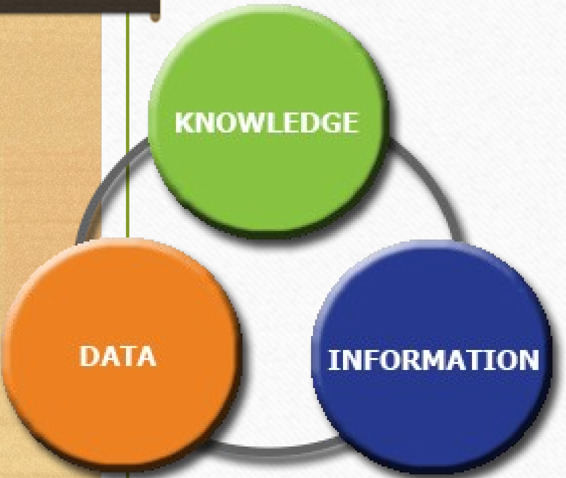
- The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently.
- It is also used to organize the data in the form of a table, schema, views, and reports, etc.
- **For example:** **Aadhar-card Database**
- Using the database, we can easily retrieve, insert, and delete the information.

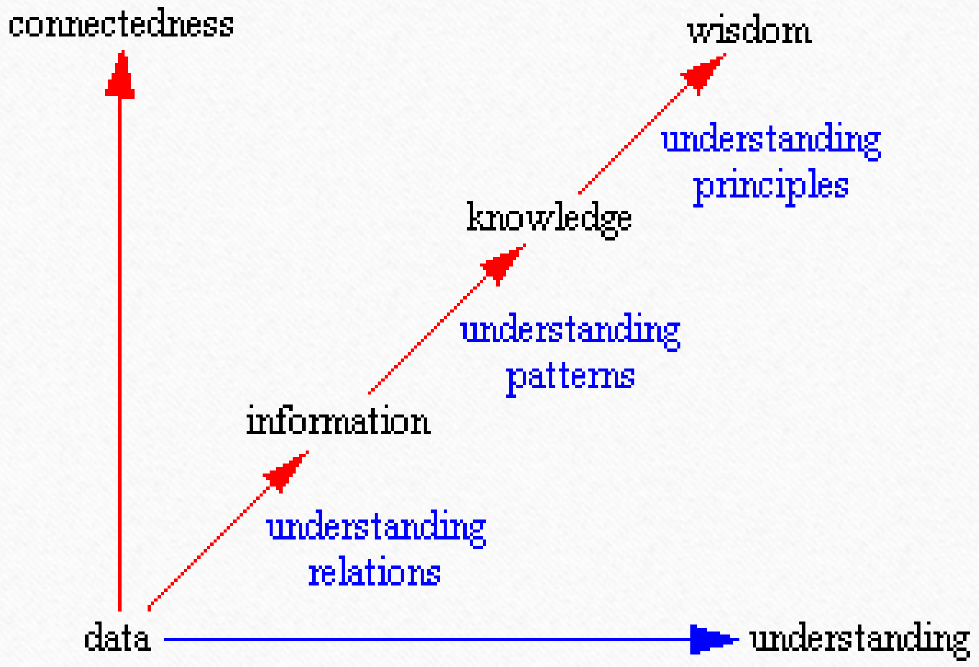
Data-Information-Knowledge Cycle

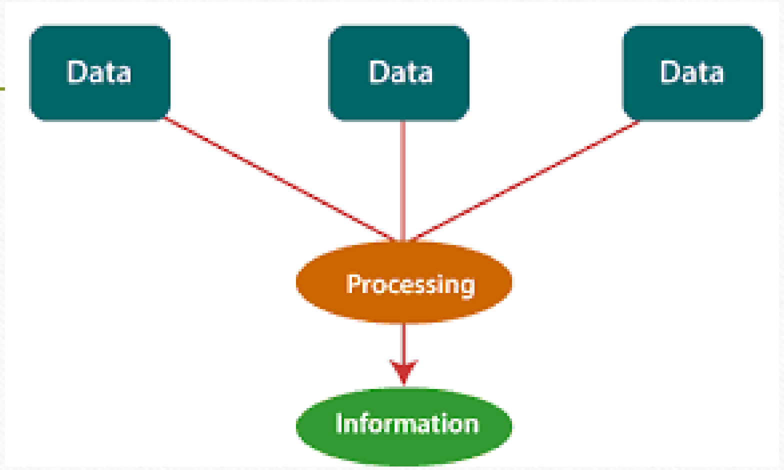
- *Data: in context* are individual facts that have meaning and can be readily understood. They are the raw facts wrapped with meaning. *Datum in context* is a single fact wrapped with meaning.

- *Information* is a set of data in context with relevance to one or more people at a point in time or for a period of time. Information is more than data in context—it must have relevance and a time frame.

- *Knowledge* is cognizance, cognition, the fact or condition of knowing something with familiarity gained through experience or association.Knowledge is information that has been retained with an understanding about the significance of that information. Knowledge includes something gained by experience, study, familiarity, association, awareness, and/or comprehension.





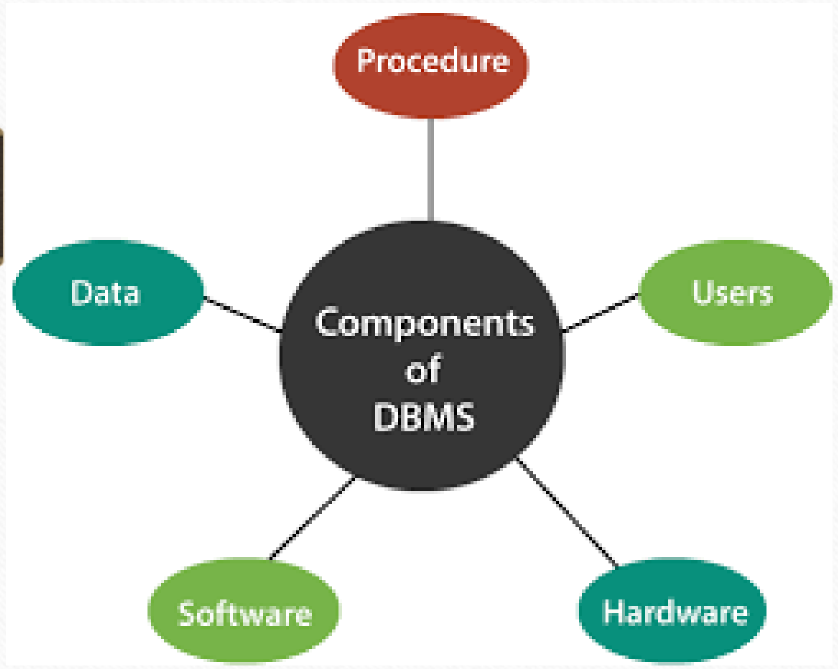


Abstract examples:
1. Selecting a car

Database management system

- Database management system is a software which is used to manage the database. For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMS allows users the following tasks:

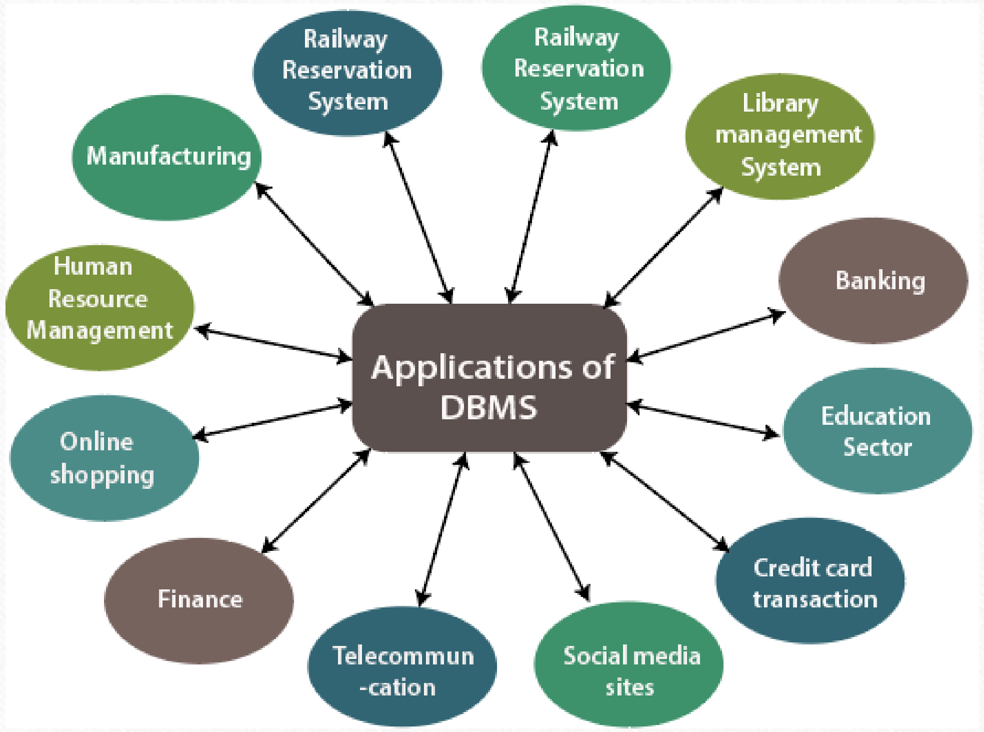


- **Data Definition:** It is used for creation, modification, and removal of definition that defines the organization of data in the database.
- **Data Updation:** It is used for the insertion, modification, and deletion of the actual data in the database.
- **Data Retrieval:** It is used to retrieve the data from the database which can be used by applications for various purposes.
- **User Administration:** It is used for registering and monitoring users, maintain data integrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristics of DBMS

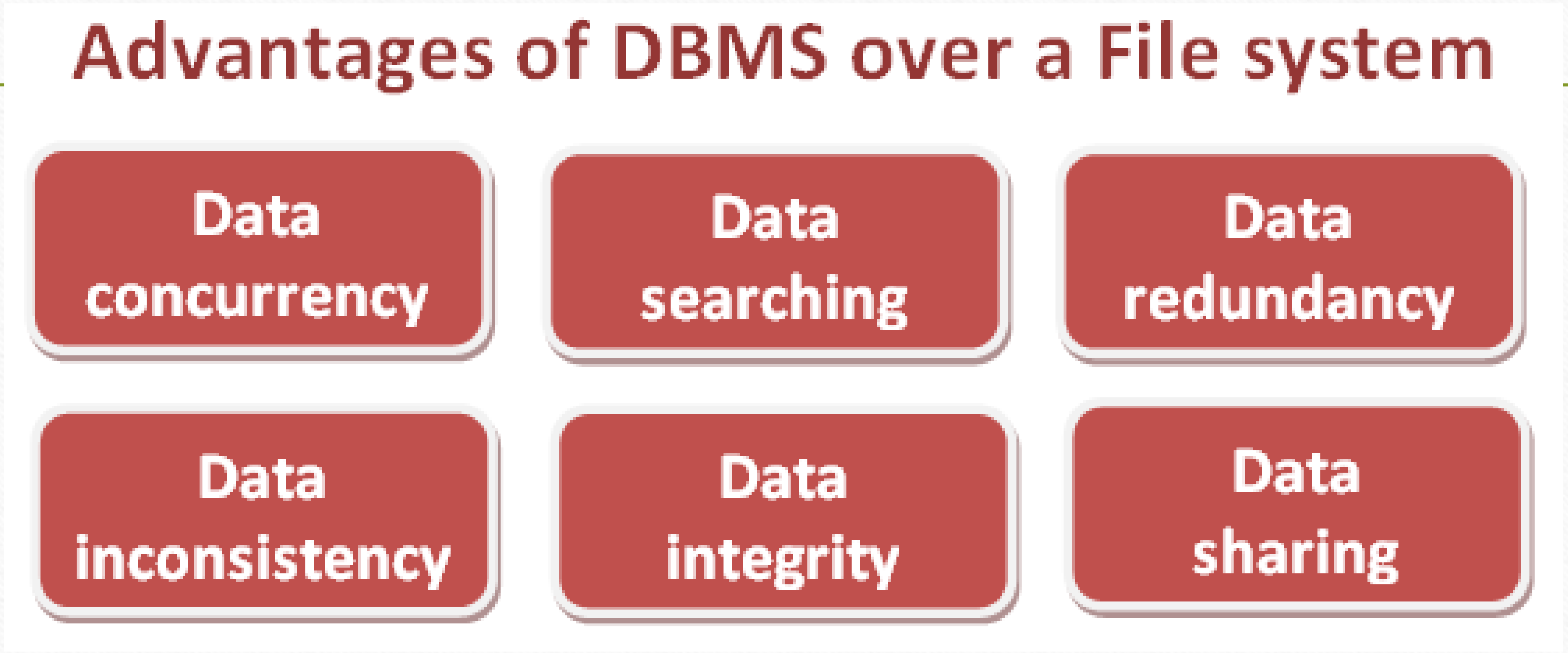
- It uses a **digital repository** established on a server to store and manage the information.
- It can provide a clear and logical view of the process that **manipulates data.**
- DBMS contains **automatic backup and recovery procedures**.
- It contains **ACID properties** which maintain data in a healthy state in case of failure.
- It can reduce the **complex relationship** between data.
- It is used to provide **security of data.**
- It can view the database from different **viewpoints** according to the requirements of the user.

Applications of dbms---in a nutshell



Advantages of DBMS

- **Controls database redundancy**: It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance**: It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time**: It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.
- **Multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces



Advantages of DBMS over a File system

- Data concurrency
- Data searching
- Data redundancy
- Data inconsistency
- Data integrity
- Data sharing

Disadvantage of DBMS

- **1.Increased costs:**Database systems require sophisticated hardware and software and highly skilled personnel.The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.
- **2. Management comple**xity:Database systems interface with many different technologies and have a significant impact on a company's resources and culture.The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives. Given the fact that database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.
- **3. Maintaining currency:**To maximize the efficiency of the database system, you must keep your system current.Therefore, you must perform frequent updates and apply the latest patches and security measures to all components.Because database technology advances rapidly, personnel training costs tend to be significant. Vendor dependence.Given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors.
- **4. Frequent upgrade/replacement cycles**:DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software.Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and administrators to properly use and manage the new features.

ACID properties

- A transaction is a collection of instructions.
- To maintain the integrity of a database, all transactions must obey ACID properties.
- **ACID** is an acronym for *atomicity*, *consistency*, *isolation*, and *durability*.

1. Atomicity

- A transaction is an atomic unit; hence, all the instructions within a transaction will successfully execute, or none of them will execute.
- Consider a transaction transfers 20 dollars from Alice's bank account to Bob's bank account. If any of the instructions fail, the entire transaction should abort and rollback.

2. Consistency

- A database is initially in a consistent state, and it should remain consistent after every transaction. Suppose that the transaction in the previous example fails after Write(A_b) and the transaction is not rolled back; then, the database will be inconsistent as the sum of Alice and Bob’s money, after the transaction, will not be equal to the amount of money they had before the transaction.

3. Isolation

- If the multiple transactions are running concurrently, they should not be affected by each other; i.e., the result should be the same as the result obtained if the transactions were running sequentially.

4. Durability

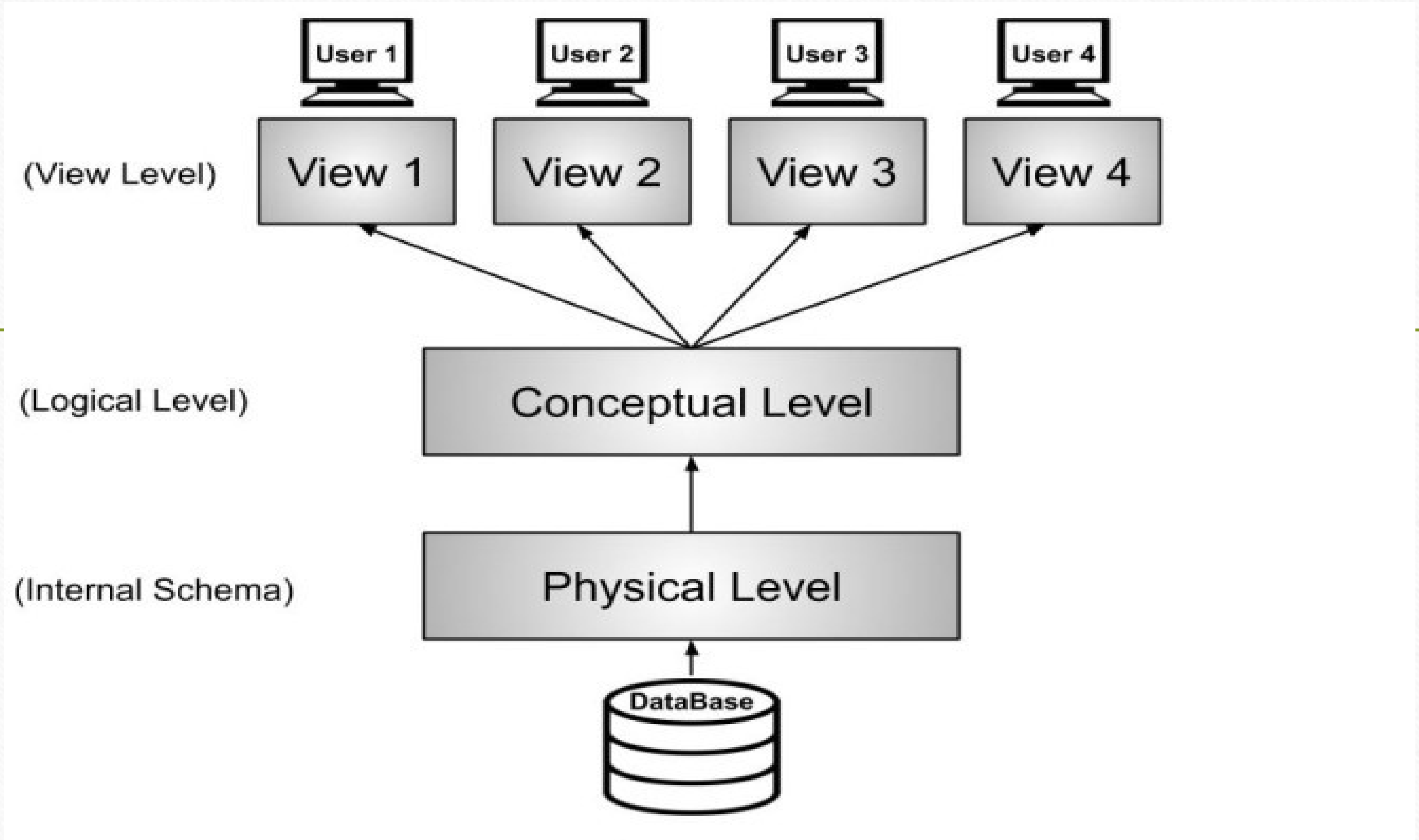
- Changes that have been committed to the database should remain even in the case of software and hardware failure. For instance, if Bob’s account contains $120, this information should not disappear upon hardware or software failure.

Levels of abstractions in DBMS

Data Abstraction

- Data Abstraction refers to the process of hiding irrelevant details from the user. So, what is the meaning of irrelevant details
- ***Example:*** If we want to access any mail from our Gmail then we don't know where that data is physically stored i.e is the data present in India or USA or what data model has been used to store that data? We are not concerned about these things. We are only concerned with our email. So, information like these i.e. location of data and data models are irrelevant to us and in data abstraction, we do this only. Apart from the location of data and data models, there are other factors that we don't care of. We hide the unnecessary data from the user and this process of hiding unwanted data is called Data Abstraction.

- There are mainly three levels of data abstraction and we divide it into three levels in order to achieve *Data Independence*. Data Independence means users and data should not directly interact with each other. The user should be at a different level and the data should be present at some other level. By doing so, Data Independence can be achieved.
- Three levels of data abstraction:
  - View Level
  - Conceptual Level
  - Physical Level



Levels of Data Abstraction

View Level or External Schema

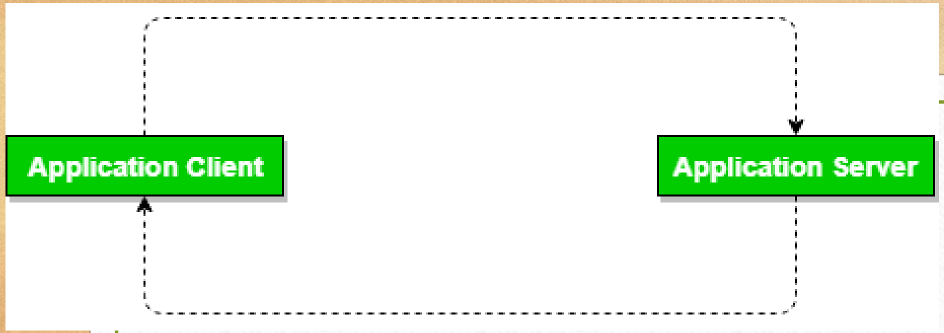
- This level tells the application about how the data should be shown to the user.
- ***Example:***
  - If we have a login-id and password in a university system, then as a student, we can view our marks, attendance, fee structure, etc. But the faculty of the university will have a different view.
  - He will have options like salary, edit marks of a student, enter attendance of the students, etc. So, both the student and the faculty have a different view.
  - By doing so, the security of the system also increases. In this example, the student can't edit his marks but the faculty who is authorized to edit the marks can edit the student's marks. Similarly, the dean of the college or university will have some more authorization and accordingly, he will has his view. So, different users will have a different view according to the authorization they have.

Conceptual Level or Logical Level

- This level tells how the data is actually stored and structured. We have different data models by which we can store the data(You can read more about the different types of data model from here).
- ***Example***:
  - Let us take an example where we use the relational model for storing the data. We have to store the data of a student, the columns in the student table will be student_name, age, mail_id, roll_no etc.
  - We have to define all these at this level while we are creating the database. Though the data is stored in the database but the structure of the tables like the student table, teacher table, books table, etc are defined here in the conceptual level or logical level.
  - Also, how the tables are related to each other are defined here. Overall, we can say that we are creating a blueprint of the data at the conceptual level.

Physical Level or Internal Schema

- As the name suggests, the Physical level tells us that where the data is actually stored i.e. it tells the actual location of the data that is being stored by the user.
- The Database Administrators(DBA) decide that which data should be kept at which particular disk drive, how the data has to be fragmented, where it has to be stored etc. They decide if the data has to be centralized or distributed.
- Though we see the data in the form of tables at view level the data here is actually stored in the form of files only. It totally depends on the DBA, how he/she manages the database at the physical level.



DBMS Architecture 2-Level, 3-Level

Two tier architecture:

- Two tier architecture is similar to a basic **client-server** model. The application at the client end directly communicates with the database at the server side. API's like ODBC,JDBC are used for this interaction.
- The server side is responsible for providing query processing and transaction management functionalities. On the client side, the user interfaces and application programs are run.
- The application on the client side establishes a connection with the server side in order to communicate with the DBMS.
- An advantage of this type is that maintenance and understanding is easier, compatible with existing systems. However this model gives poor performance when there are a large number of users.

Three Tier architecture:

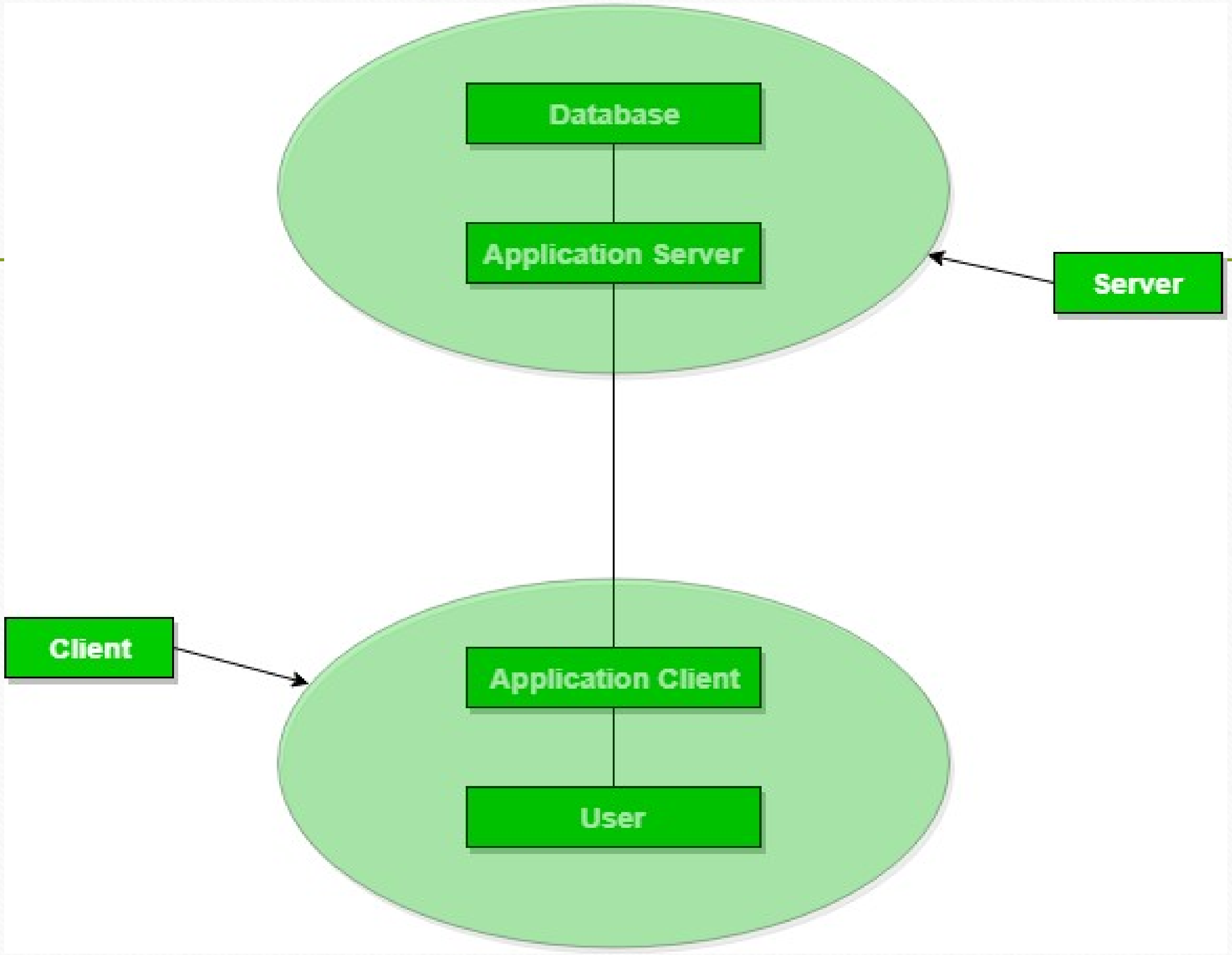
- **In** this type, there is another layer between the client and the server. The client does not directly communicate with the server.
- Instead, it interacts with an application server which further communicates with the database system and then the query processing and transaction management takes place.
- This intermediate layer acts as a medium for exchange of partially processed data between server and client. This type of architecture is used in case of large web applications.

Advantages:

- **Enhanced scalability** due to distributed deployment of application servers. Now,individual connections need not be made between client and server.
- **Data Integrity** is maintained. Since there is a middle layer between client and server, data corruption can be avoided/removed.
- **Security** is improved. This type of model prevents direct interaction of the client with the server thereby reducing access to unauthorized data.

Disadvantages:

- Increased complexity of implementation and communication. It becomes difficult for this sort of interaction to take place due to presence of middle layers.

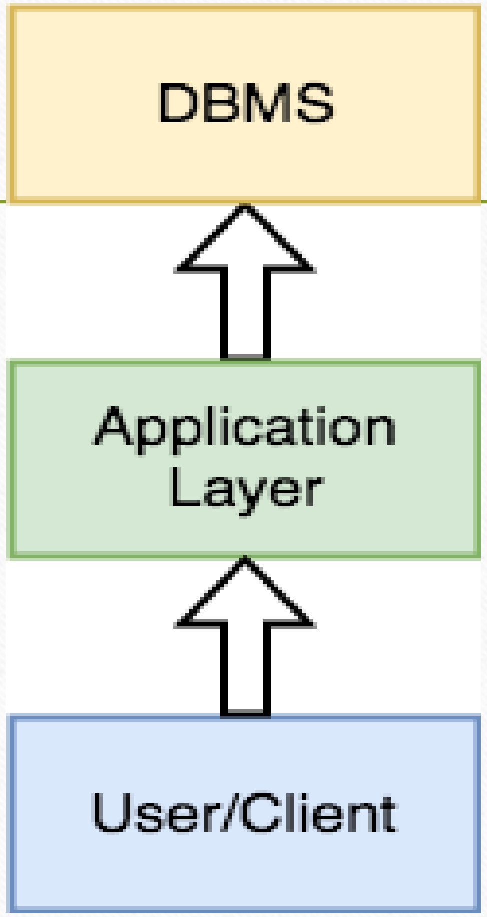

Database
Application Server
Server
Client
Application Client
User

DBMS Architecture

- A Database Management system is not always directly available for users and applications to access and store data in it. A Database Management system can be **centralised**(all the data stored at one location), **decentralised**(multiple copies of database at different locations) or **hierarchical**, depending upon its architecture.
- **1-tier DBMS** architecture also exist, this is when the database is directly available to the user for using it to store data. Generally such a setup is used for local application development, where programmers communicate directly with the database for quick response.
  - Database Architecture is logically of two types:
    - 2-tier DBMS architecture
    - 3-tier DBMS architecture

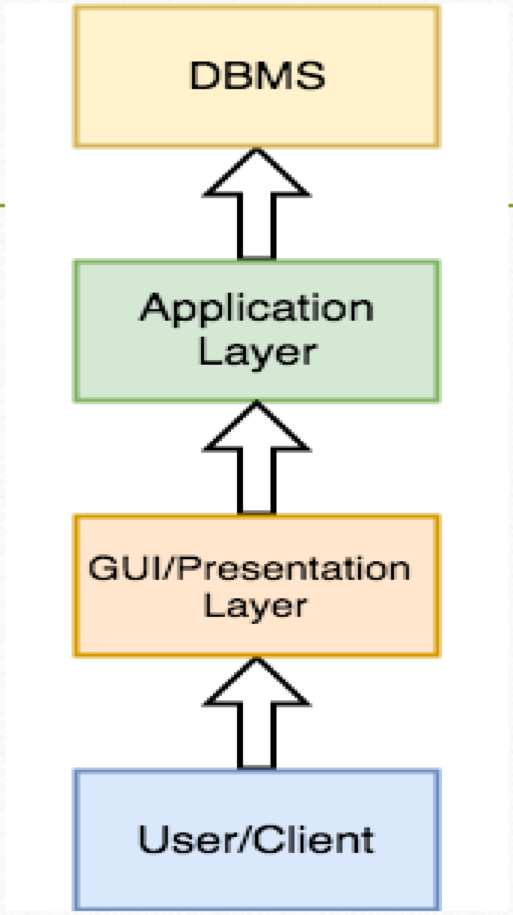
2-tier DBMS Architecture

- 2-tier DBMS architecture includes an **Application layer** between the user and the DBMS, which is responsible to communicate the user's request to the database management system and then send the response from the DBMS to the user.
- An application interface known as **ODBC**(Open Database Connectivity) provides an API that allow client side program to call the DBMS. Most DBMS vendors provide ODBC drivers for their DBMS.
- Such an architecture provides the DBMS extra security as it is not exposed to the End User directly. Also, security can be improved by adding security and authentication checks in the Application layer too.


DBMS
Application Layer
User/Client

3-tier DBMS Architecture

- 3-tier DBMS architecture is the most commonly used architecture for web applications.
- It is an extension of the 2-tier architecture. In the 2-tier architecture, we have an application layer which can be accessed programatically to perform various operations on the DBMS. The application generally understands the Database Access Language and processes end users requests to the DBMS.
- In 3-tier architecture, an additional Presentation or GUI Layer is added, which provides a graphical user interface for the End user to interact with the DBMS.
- For the end user, the GUI layer is the Database System, and the end user has no idea about the application layer and the DBMS system.


DBMS
Application Layer
GUI/Presentation Layer
User/Client

Centralized Database Design:

- For a small organization and limited scope of operations, the database may be small in terms of data. Therefore, the database design may be relatively simple and can be easily done by one group of designer or even a single person. This is called centralized design of database.
- The designer can study the system processes, identify the constraints, and create conceptual schema, whereas the users can verify it to ensure that the database meets their needs and processing requirements.

Decentralized Database Design:

- When the database study reveals that the resulting database is for the ‘whole organizations and that it has large number of entities and complex relations on which very complex operations are performed, then the database design may be undertaken by division of work. Here it may be suitable to study and design conceptual schema for each department or function for which the database is to be designed.
- The database design project may be thought of as one large project subdivided into smaller modules, and each module is designed by a group of people.
- Each module is in itself a system and must meet the system requirements as a whole. These modules when integrated to form a single database must meet the processing requirements. This is called decentralized design approach. This approach is also suitable when the database design is spread across several operational sites, and each element is a subset of the entire data set.

Characteristics of Database Approach

1. Manages Information

- A database always takes care of its information because information is always helpful for whatever work we do. It manages all the information that is required to us. Managing information by using a database, we become more deliberated user of our data.

2. Easy Operation Implementation

- All the operations like insert, delete, update, search etc. are carried out in a flexible and easy way. Database makes it very simple to implement these operations. A user with little knowledge can perform these operations. This characteristic of database makes it more powerful.

3. Multiple Views of Database

- Basically, a view is a **subset of the database**. A view is defined and devoted for a particular user of the system. Different users of the system may have different views of the same system.

4. Data For Specific Purpose

- A database is designed for data of specific purpose. **For example**, a database of student management system is designed to maintain the record of student's marks, fees and attendance etc. This data has a specific purpose of maintaining student record.

5. It has Users of Specific Interest

- A database always has some indented group of users and applications in which these user groups are interested.
- **For example**, in a library system, there are three users, official administration of the college, the librarian, and the students.

6. Represent Some Aspects of Real World Applications

- A database represents some features of real world applications. Any change in the real world is reflected in the database. If we have some changes in our real applications like railway reservation system then it will be reflected in database too.

7. Self Describing nature

- A database is of self describing nature; it always describes and narrates itself. It contains the description of the whole data structure, the constraints and the variables.
- It makes it different from traditional file management system in which definition was not the part of application program. These definitions are used by the users and DBMS software when needed.

8. Logical Relationship Between Records and Data

- A database gives a logical relationship between its records and data. So a user can access various records depending upon the logical conditions by a single query from the database.

9. Shelter Between Program and Data

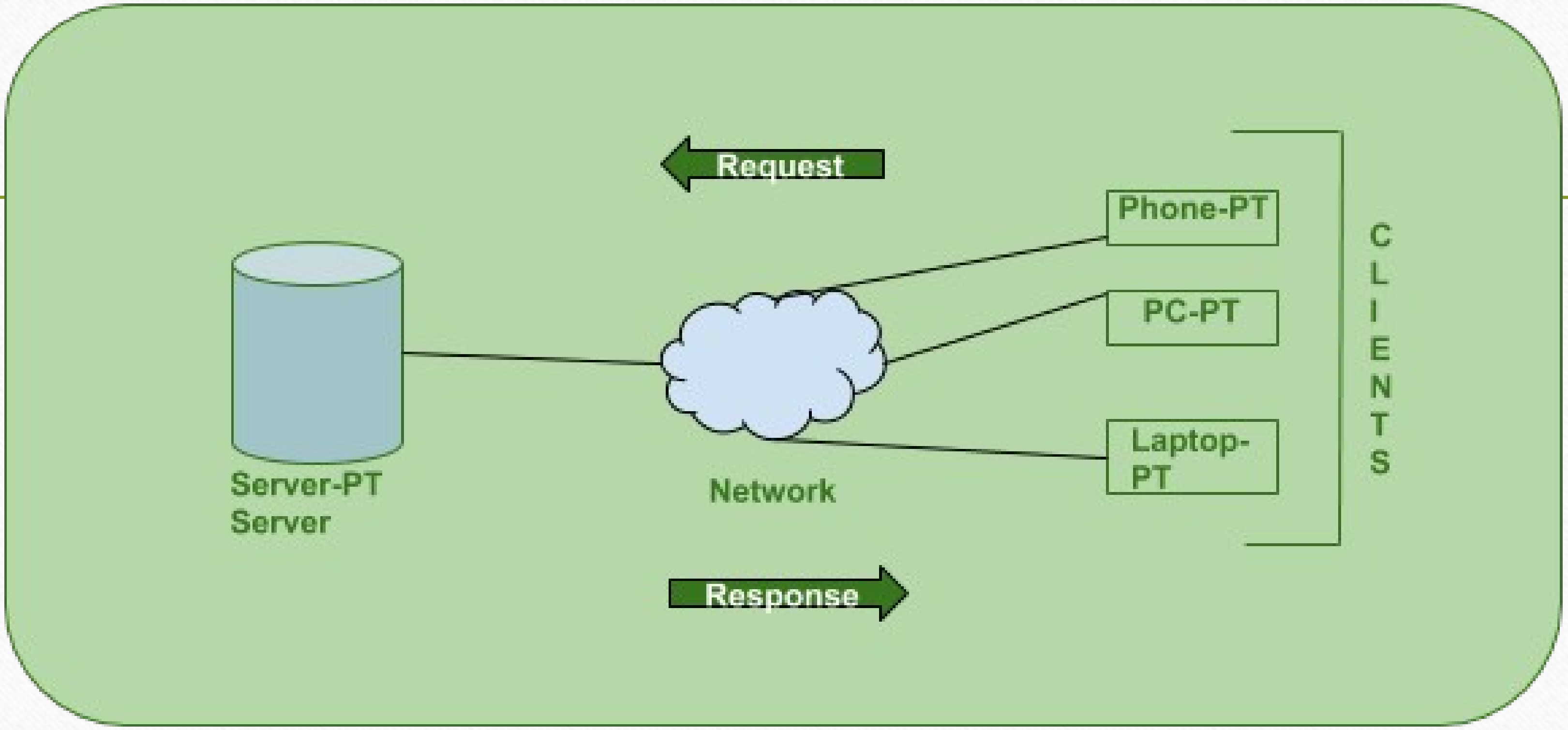
- In traditional file management system, if any user makes changes in the structure of a file then all the programs accessed by that file needed to be changed. The structure of data files is defined by the application programs.

Client-Server Model

- The Client-server model is a distributed application structure that partitions task or workload between the providers of a resource or service, called servers, and service requesters called clients.
- In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and deliver the data packets requested back to the client.
- Clients do not share any of their resources. Examples of Client-Server Model are Email, World Wide Web, etc.

How the Client-Server Model works ?

- **Client:** When we mention the word **Client**, it mean to talk of a person or an organization using a particular service. Similarly in the digital world a **Client** is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).
- **Servers:** Similarly, when we mention the word **Servers**, It mean a person or medium that serves something. Similarly in this digital world a **Server** is a remote computer which provides information (data) or access to particular services.
- **So, its basically the Client requesting something and the Server serving it as long as its present in the database**


Request
Phone-PT
PC-PT
Laptop-PT
CLIENTS
Server-PT Server
Network
Response

Few steps to follow to interacts with the servers a client.

- User enters the **URL**(Uniform Resource Locator) of the website or file. The Browser then requests the **DNS**(DOMAIN NAME SYSTEM) Server.
- **DNS Server** lookup for the address of the **WEB Server**.
- **DNS Server** responds with the **IP address** of the **WEB Server**.
- Browser sends over an **HTTP/HTTPS** request to **WEB Server's IP** (provided by **DNS server**).
- Server sends over the necessary files of the website.
- Browser then renders the files and the website is displayed. This rendering is done with the help of **DOM** (Document Object Model) interpreter, **CSS** interpreter and **JS Engine** collectively known as the **JIT** or (Just in Time) Compilers.

Advantages of Client-Server model:

- Centralized system with all data in a single place.
- Cost efficient requires less maintenance cost and Data recovery is possible.
- The capacity of the Client and Servers can be changed separately.

Disadvantages of Client-Server model:

- Clients are prone to viruses, Trojans and worms if present in the Server or uploaded into the Server.
- Server are prone to Denial of Service (DOS) attacks.
- Data packets may be spoofed or modified during transmission.
- Phishing or capturing login credentials or other useful information of the user are common and MITM(Man in the Middle) attacks are common.

Object based model

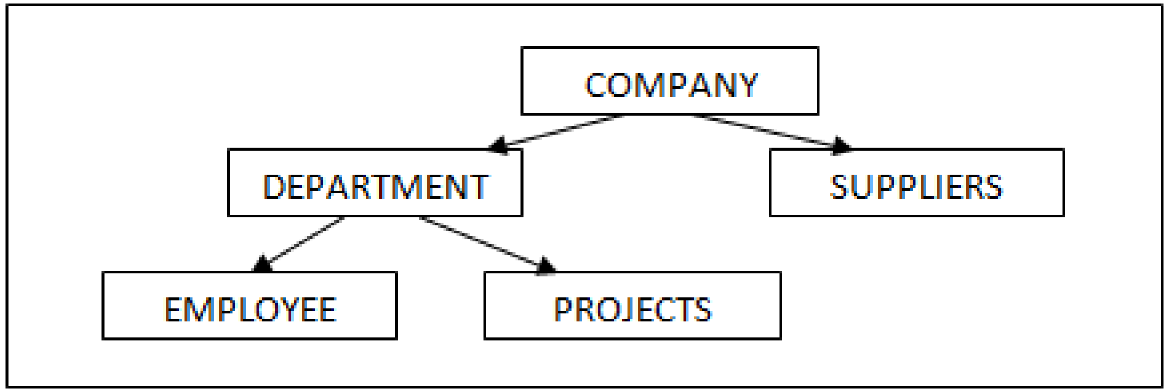
- In object based data models, the focus is on how data is represented. The data is divided into multiple entities each of which have some defining characteristics. Moreover, these data entities are connected with each other through some relationships.
- So, in object based data models the entities are based on real world models, and how the data is in real life. There is not as much concern over what the data is as compared to how it is visualised and connected.
- Some examples of object based data models are
- Entity Relationship Data Model
- Object Oriented Data Model
- Semantic Data Model
- Functional Data Model

Record based model

- The actual relationship between any two entities can be observed in record based data models.
- There are 3 types of record based data models defined so far- Hierarchical, Network and Relational data models. Most widely used record based data model is relational data model. Other two are not widely used.

Hierarchical Data Models

- Imagine we have to create a database for a company. What are the entities involved in it? Company, its department, its supplier, its employees, different projects of the company etc are the different entities we need to take care of.
- If we observe each of the entity they have parent –child relationship. We can design them like we do ancestral hierarchy. In our case, Company is the parent and rests of them are its children. Department has employees and project as its children and so on. This type of data modelling is called hierarchical data model.
- In this data model, the entities are represented in a hierarchical fashion. Here we identify a parent entity, and its child entity. Again we drill down to identify next level of child entity and so on. This model can be imagined as folders inside a folder!


COMPANY
DEPARTMENT
SUPPLIERS
EMPLOYEE
PROJECTS

- **Advantages**
- It helps to address the issues of flat file data storage. In flat files, data will be scattered and there will not be any proper structuring of the data. This model groups the related data into tables and defines the relationship between the tables, which is not addressed in flat files.

Disadvantages

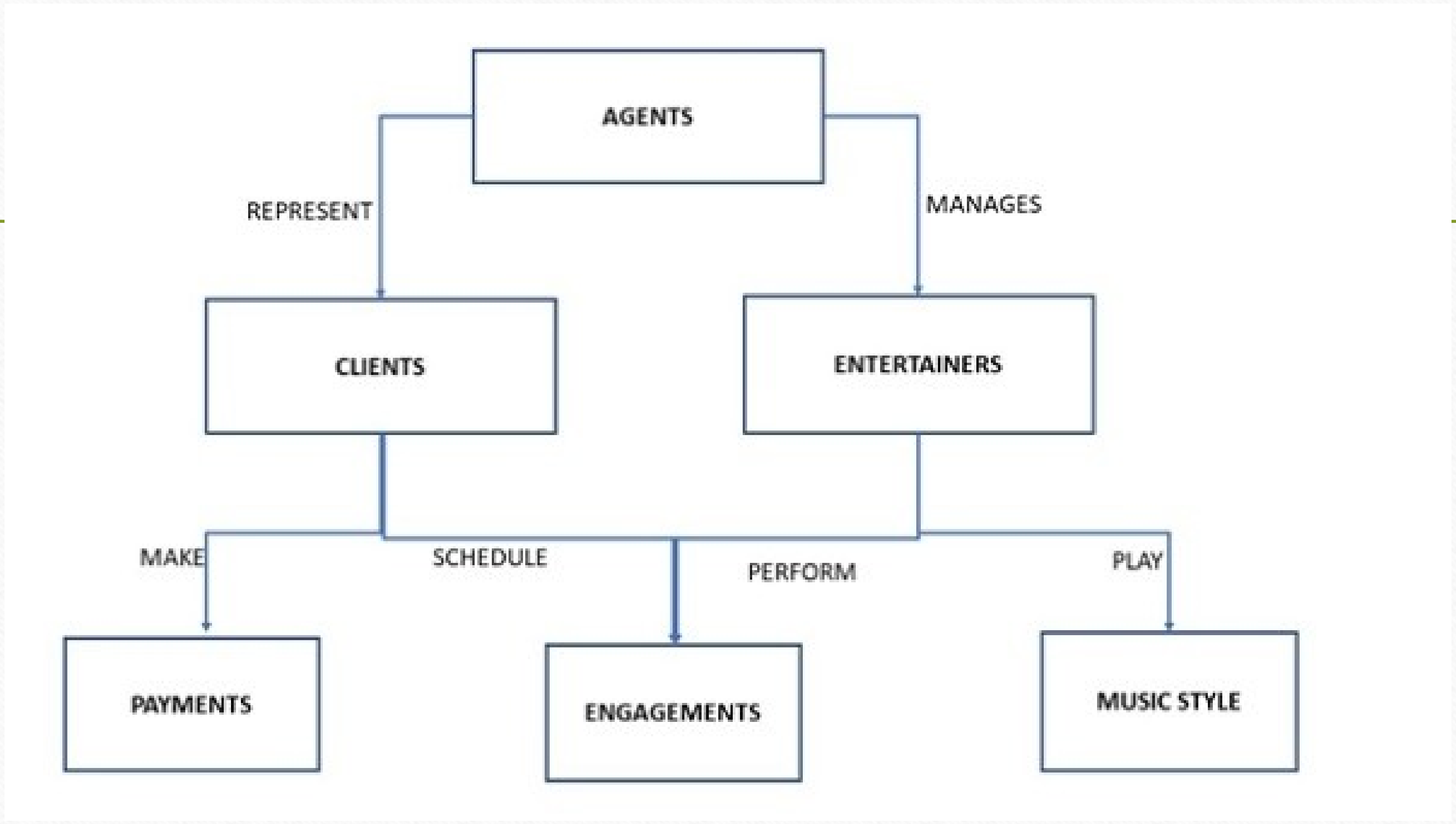
- Redundancy: – When data is stored in a flat file, there might be repetition of same data multiple times and any changes required for the data will need to change in all the places in the flat file. Missing to update at any one place will cause incorrect data. This kind redundancy is solved by hierarchical model to some extent. Since records are grouped under related table, it solves the flat file redundancy issue. But look at the many to many relationship examples given above. In such case, we have to store same project information for more than one department. This is duplication of data and hence a redundancy. So, this model does not reduce the redundancy issue to a significant level.
- As we have seen above, it fails to handle many to many relationships efficiently. It results in redundancy and confusion. It can handle only parent-child kind of relationship.
- If we need to fetch any data in this model, we have to start from the root of the model and traverse through its child till we get the result. In order to perform the traversing, either we should know well in advance the layout of model or we should be very good programmer. Hence fetching through this model becomes bit difficult.
- Imagine company has got some new project details, but it did not assign it to any department yet. In this case, we cannot store project information in the PROJECT table, till company assigns it to some department. That means, in order to enter any child information, its parent information should be already known / entered.

Network Based Model

- The network model is the extension of the hierarchical structure because it allows many-to-many relationships to be managed in a tree-like structure that allows multiple parents.
- There are two fundamental concepts of a network model −
  - Records contain fields which need hierarchical organization.
  - Sets are used to define one-to-many relationships between records that contain one owner, many members.
- A record may act as an owner in any number of sets, and a member in any number of sets.

- A set is designed with the help of circular linked lists where one record type, the owner of the set also called as a parent, appears once in each circle, and a second record type, also known as the subordinate or child, may appear multiple times in each circle.
- A hierarchy is established between any two record types where one type (A) is the owner of another type (B). At the same time, another set can be developed where the latter set (B) is the owner of the former set (A). In this model, ownership is defined by the direction, thus all the sets comprise a general directed graph. Access to records is developed by the indexing structure of circular linked lists.

- The network model has the following major features −
- It can represent redundancy in data more efficiently than that in the hierarchical model.
- There can be more than one path from a previous node to successor node/s.
- The operations of the network model are maintained by indexing structure of linked list (circular) where a program maintains a current position and navigates from one record to another by following the relationships in which the record participates.
- Records can also be located by supplying key values.


AGENTS
REPRESENT
MANAGES
CLIENTS
ENTERTAINERS
MAKE
SCHEDULE
PERFORM
PLAY
PAYMENTS
ENGAGEMENTS
MUSIC STYLE

- **Advantages**
- fast data access.
- It also allows users to create queries that are more complex than those they created using a hierarchical database. So, a variety of queries can be run over this model.
- **Disadvantages**
- A user must be very familiar with the structure of the database to work through the set structures.
- Updating inside this database is a tedious task. One cannot change a set structure without affecting the application programs that use this structure to navigate through the data. If you change a set structure, you must also modify all references made from within the application program to that structure.

What is Relational Model?

- **RELATIONAL MODEL (RM)** represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.
- The table name and column names are helpful to interpret the meaning of values in each row. The data are represented as a set of relations. In the relational model, data are stored as tables. However, the physical storage of the data is independent of the way the data are logically organized.

Relational Model Concepts

1.Attribute: Each column in a Table. Attributes are the properties which define a relation. e.g., Student_Rollno, NAME,etc.
2.Tables – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes.
3.Tuple – It is nothing but a single row of a table, which contains a single record.
4.Relation Schema: A relation schema represents the name of the relation with its attributes.
5.Degree: The total number of attributes which in the relation is called the degree of the relation.

6.Cardinality: Total number of rows present in the Table.

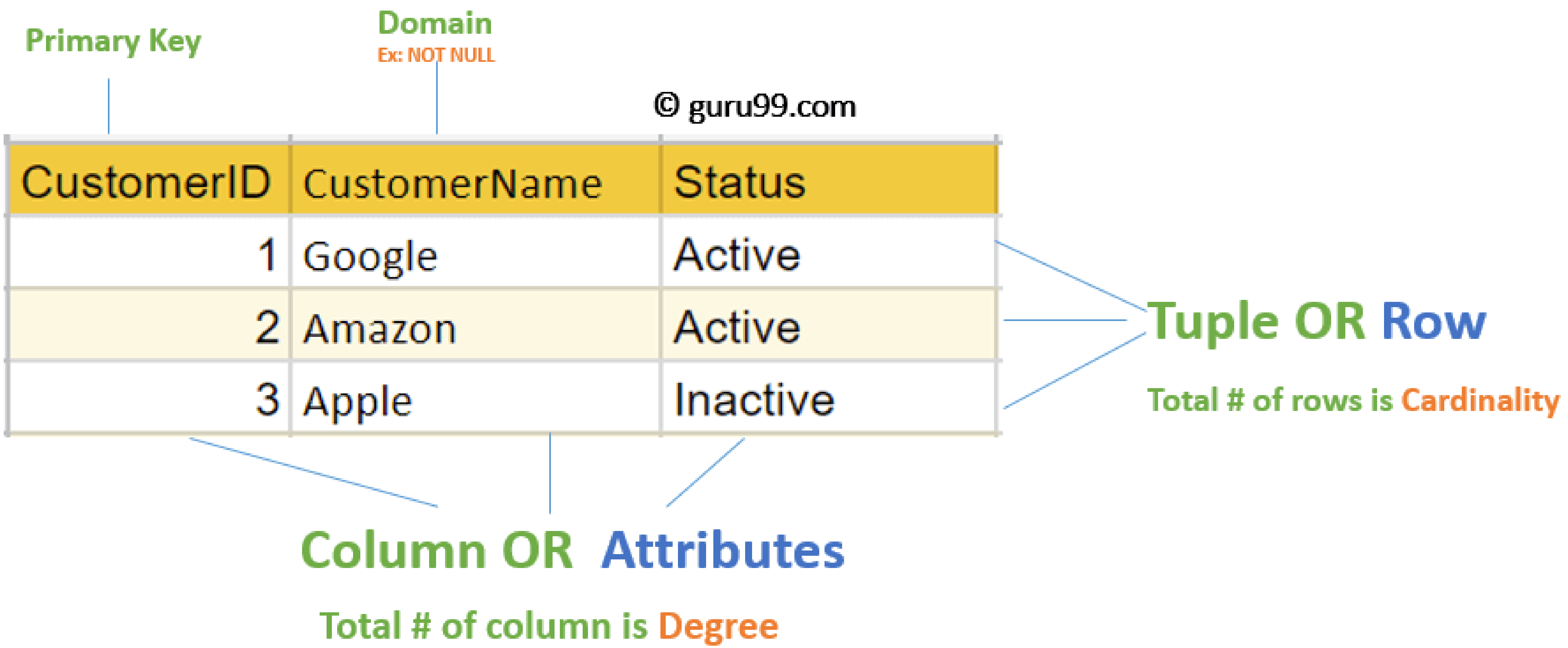
7.Column: The column represents the set of values for a specific attribute.

8. Relation instance – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.

9. Relation key - Every row has one, two or multiple attributes, which is called relation key.

10. Attribute domain – Every attribute has some pre-defined value and scope which is known as attribute domain

Table also called Relation

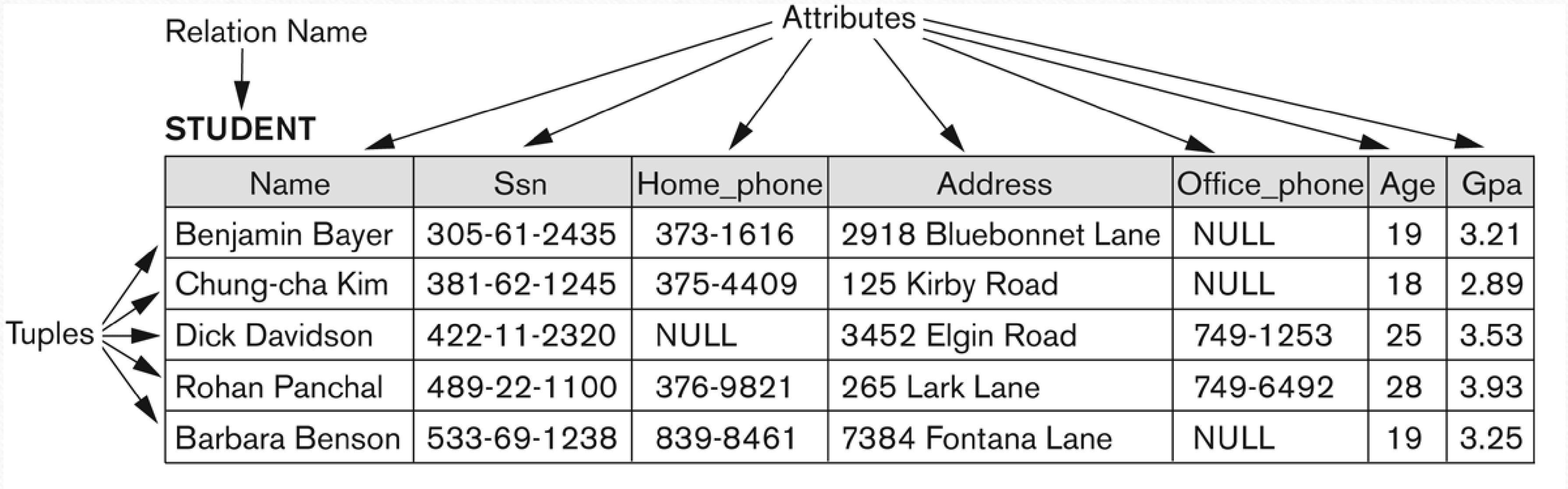


CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Informal Definitions

- Informally, a **relation** looks like a **table** of values.
- A relation typically contains a **set of rows**.
- The data elements in each **row** represent certain facts that correspond to a real-world **entity** or **relationship**
  - In the formal model, rows are called **tuples**
- Each **column** has a column header that gives an indication of the meaning of the data items in that column
  - In the formal model, the column header is called an **attribute name** (or just **attribute**)

Example of a Relation



STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

Informal Definitions

- Key of a Relation:
  - Each row has a value of a data item (or set of items) that uniquely identifies that row in the table
    - Called the *key*
  - In the STUDENT table, SSN is the key
  - Sometimes row-ids or sequential numbers are assigned as keys to identify the rows in a table

Called *artificial key* or *surrogate key*

Formal Definitions - Schema

- The **Schema** (or description) of a Relation:
  - Denoted by R(A1, A2, ...An)
  - R is the **name** of the relation
  - The **attributes** of the relation are A1, A2, ..., An

Example:

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

  - CUSTOMER is the relation name
  - Defined over the four attributes: Cust-id, Cust-name, Address, Phone#
- Each attribute has a **domain** or a set of valid values.
  - For example, the domain of Cust-id is 6 digit numbers.

Formal Definitions - Tuple

- A **tuple** is an ordered set of values (enclosed in angled brackets '< … >')
- Each value is derived from an appropriate *domain*.
- A row in the CUSTOMER relation is a 4-tuple and would consist of four values, for example:
  - <632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">
  - This is called a 4-tuple as it has 4 values
  - A tuple (row) in the CUSTOMER relation.
- A relation is a **set** of such tuples (rows)

Formal Definitions - Domain

- A **domain** has a logical definition:
  - Example: “USA_phone_numbers” are the set of 10 digit phone numbers valid in the U.S.
- A domain also has a data-type or a format defined for it.
  - The USA_phone_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
  - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.
- The attribute name designates the role played by a domain in a relation:
  - Used to interpret the meaning of the data elements corresponding to that attribute
  - Example: The domain Date may be used to define two attributes named “Invoice-date” and “Payment-date” with different meanings

Formal Definitions - State

- The **relation state** is a subset of the Cartesian product of the domains of its attributes
  - each domain contains the set of all possible values the attribute can take.
- Example: attribute Cust-name is defined over the domain of character strings of maximum length 25
  - dom(Cust-name) is varchar(25)
- The role these strings play in the CUSTOMER relation is that of the *name of a customer*.

Formal Definitions - Summary

- Formally,
  - Given R(A1, A2, ..., An)
  - r(R) ㅋ dom (A1) X dom (A2) X ....X dom(An)
- R(A1, A2, …, An) is the **schema** of the relation
- R is the **name** of the relation
- A1, A2, …, An are the **attributes** of the relation

Formal Definitions - Example

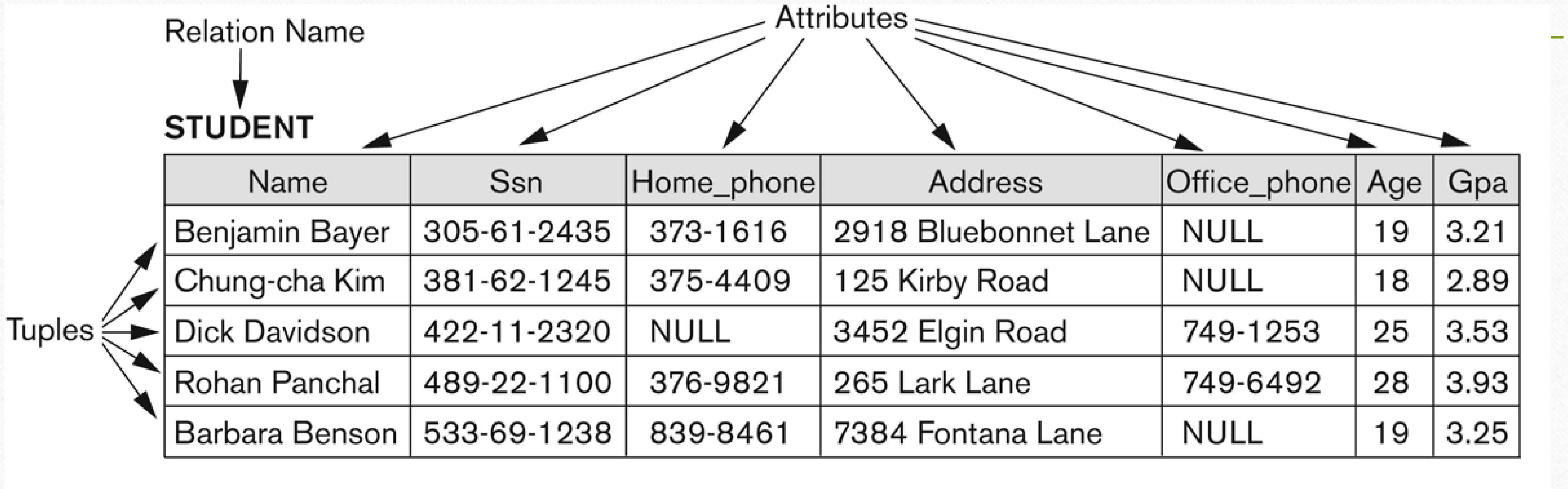
- Let R(A1, A2) be a relation schema:
  - Let dom(A1) = {0,1}
  - Let dom(A2) = {a,b,c}
- Then: dom(A1) X dom(A2) is all possible combinations:

  {<0,a> , <0,b> , <0,c>, <1,a>, <1,b>, <1,c> }
- The relation state r(R) $\subset$ dom(A1) X dom(A2)
- For example: r(R) could be {<0,a> , <0,b> , <1,c> }
  - this is one possible state (or "population" or "extension") r of the relation R, defined over A1 and A2.
  - It has three 2-tuples: <0,a> , <0,b> , <1,c>

Definition Summary

Informal Terms	Formal Terms
Table	Relation
Column Header	Attribute
All possible Column Values	Domain
Row	Tuple
Table Definition	Schema of a Relation
Populated Table	State of the Relation

Example – A relation STUDENT


Relation Name
STUDENT
Attributes
Tuples

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Figure 5.1
The attributes and tuples of a relation STUDENT.

Characteristics Of Relations

Ordering of tuples in a relation r(R):

- The tuples are *not considered to be ordered*, even though they appear to be in the tabular form.
- Ordering of attributes in a relation schema R (and of values within each tuple):
- We will consider the attributes in R(A1, A2, ..., An) and the values in t=<v1, v2, ..., vn> to be ordered .(However, a more general alternative definition of relation does not require this ordering).

Same state as previous Figure (but with different order of tuples)

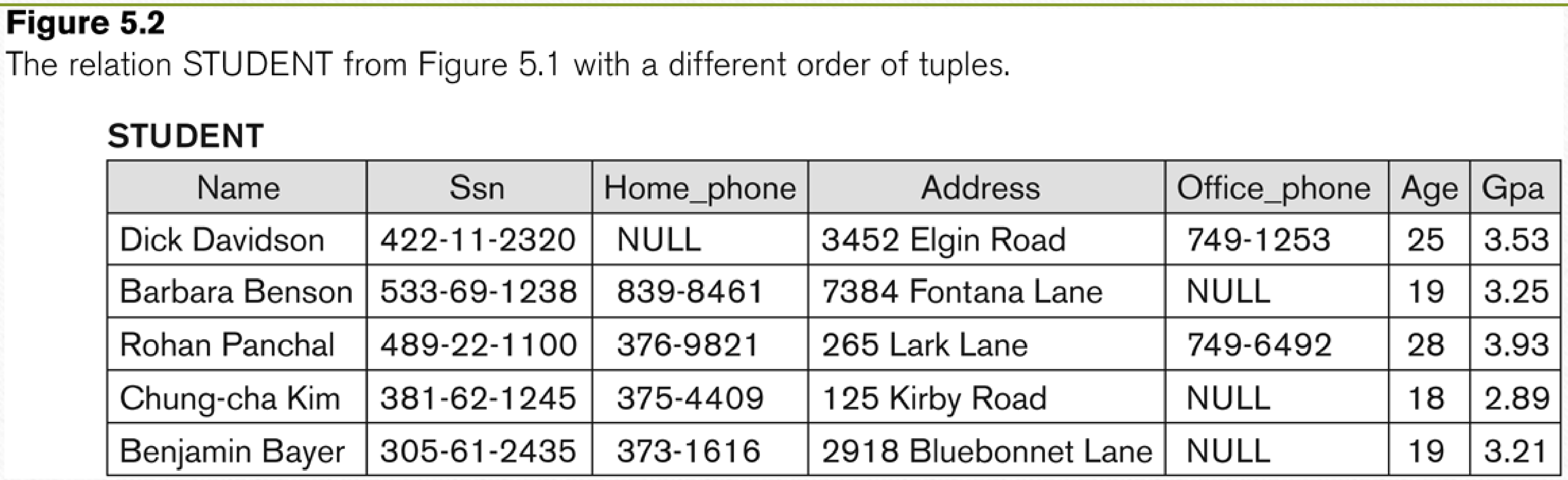


Figure 5.2
The relation STUDENT from Figure 5.1 with a different order of tuples.

STUDENT

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21

Characteristics Of Relations

- Values in a tuple:
  - All values are considered atomic (indivisible).
  - Each value in a tuple must be from the domain of the attribute for that column
    - If tuple t = <v1, v2, …, vn> is a tuple (row) in the relation state r of R(A1, A2, …, An)
    - Then each *vi* must be a value from *dom(Ai)*
  - A special **null** value is used to represent values that are unknown or inapplicable to certain tuples.

Characteristics Of Relations

Notation:

- We refer to **component values** of a tuple t by:
- t[Ai] or t.Ai
- This is the value vi of attribute Ai for tuple t
- Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing the values of attributes Au, Av, ..., Aw, respectively in t

Relational Integrity Constraints

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of constraints in the relational model:
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints
- Another implicit constraint is the **domain** constraint
  - Every value in a tuple must be from the *domain of its attribute* (or it could be **null**, if allowed for that attribute)

Key Constraints

- **Superkey** of R:
  - Is a set of attributes SK of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for SK
    - That is, for any distinct tuples t1 and t2 in r(R), t1[SK] 늫 t2[SK]
    - This condition must hold in *any valid state* r(R)
- **Key** of R:
  - A "minimal" superkey
  - That is, a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (does not possess the superkey uniqueness property)

Key Constraints (continued)

Example: Consider the CAR relation schema:

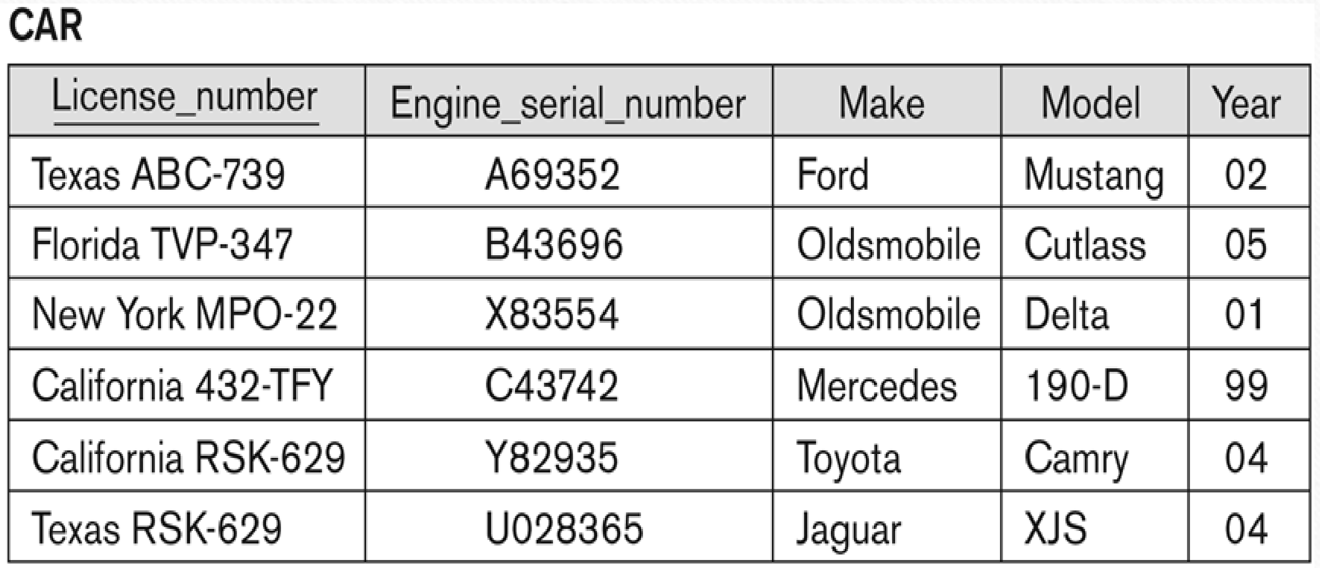
- CAR(State, Reg#, SerialNo, Make, Model, Year)
- CAR has two keys:
  - Key1 = {State, Reg#}
  - Key2 = {SerialNo}
- Both are also superkeys of CAR
- {SerialNo, Make} is a superkey but *not* a key.

- In general:
  - Any *key* is a *superkey* (but not vice versa)
  - Any set of attributes that *includes a key* is a *superkey*
  - A *minimal* superkey is also a key

Key Constraints (continued)

- If a relation has several **candidate keys**, one is chosen arbitrarily to be the **primary key**.
  - The primary key attributes are underlined.
- Example: Consider the CAR relation schema:
  - CAR(State, Reg#, SerialNo, Make, Model, Year)
  - We chose SerialNo as the primary key
- The primary key value is used to *uniquely identify* each tuple in a relation
  - Provides the tuple identity
- Also used to *reference* the tuple from another tuple
  - General rule: Choose as primary key the smallest of the candidate keys (in terms of size)
  - Not always applicable – choice is sometimes subjective

CAR table with two candidate keys – LicenseNumber chosen as Primary Key



CAR

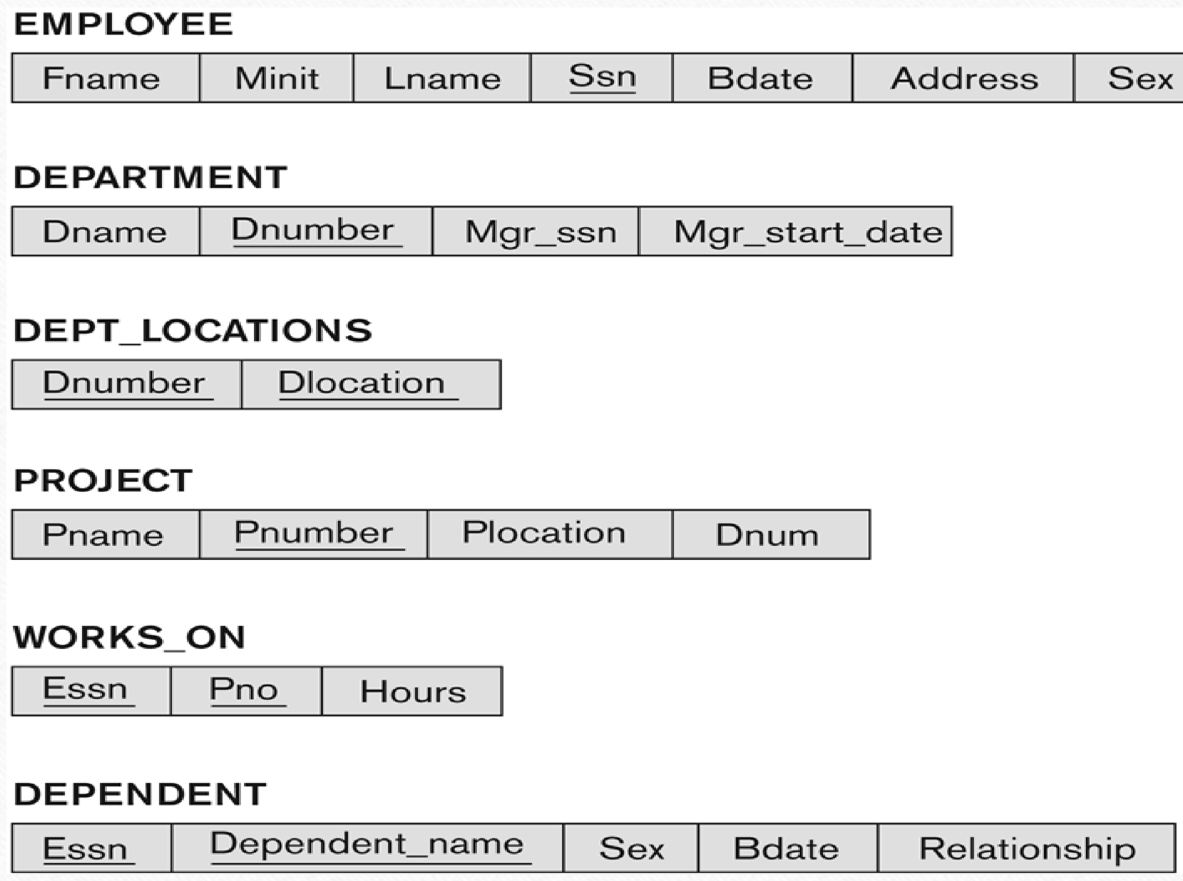
License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Figure 5.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.

Relational Database Schema

- **Relational Database Schema:**
  - A set S of relation schemas that belong to the same database.
  - S is the name of the whole **database schema**
  - S = {R1, R2, ..., Rn}
  - R1, R2, …, Rn are the names of the individual **relation schemas** within the database S
- Following slide shows a COMPANY database schema with 6 relation schemas

COMPANY Database Schema



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 5.5
Schema diagram for the COMPANY relational database schema.

Entity Integrity

- **Entity Integrity:**
  - The *primary key attributes* PK of each relation schema R in S cannot have null values in any tuple of r(R).
    - This is because primary key values are used to *identify* the individual tuples.
    - t[PK] ㄴㅎ null for any tuple t in r(R)
    - If PK has several attributes, null is not allowed in any of these attributes
  - Note: Other attributes of R may be constrained to disallow null values, even though they are not members of the primary key.

Referential Integrity

- A constraint involving **two** relations
  - The previous constraints involve a single relation.
- Used to specify a **relationship** among tuples in two relations:
  - The **referencing relation** and the **referenced relation**.

Referential Integrity

- Tuples in the **referencing relation** R1 have attributes FK (called **foreign key** attributes) that reference the primary key attributes PK of the **referenced relation** R2.
  - A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK].
- A referential integrity constraint can be displayed in a relational database schema as a directed arc from R1.FK to R2.

Referential Integrity (or foreign key) Constraint

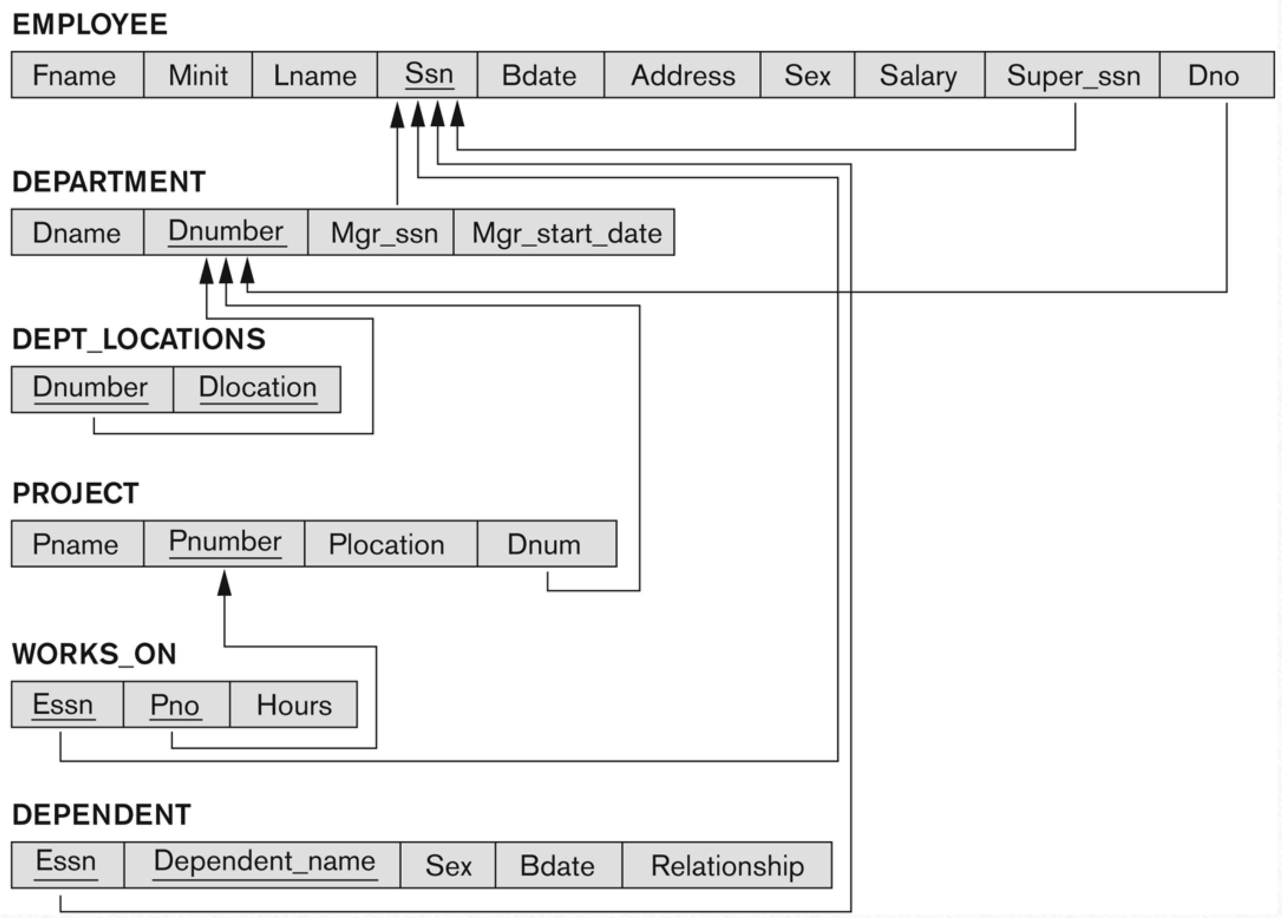
- Statement of the constraint
  - The value in the foreign key column (or columns) FK of the the **referencing relation** R1 can be **either**:
    - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, or
    - (2) a **null**.
- In case (2), the FK in R1 should **not** be a part of its own primary key.

Displaying a relational database schema and its constraints

- Each relation schema can be displayed as a row of attribute names
- The name of the relation is written above the attribute names
- The primary key attribute (or attributes) will be underlined
- A foreign key (referential integrity) constraints is displayed as a directed arc (arrow) from the foreign key attributes to the referenced table
    - Can also point the the primary key of the referenced relation for clarity

Referential Integrity Constraints for COMPANY database

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Populated database state

- Each *relation* will have many tuples in its current relation state
- The *relational database state* is a union of all the individual relation states
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple
- Next slide shows an example state for the COMPANY database

Populated database state for COMPANY

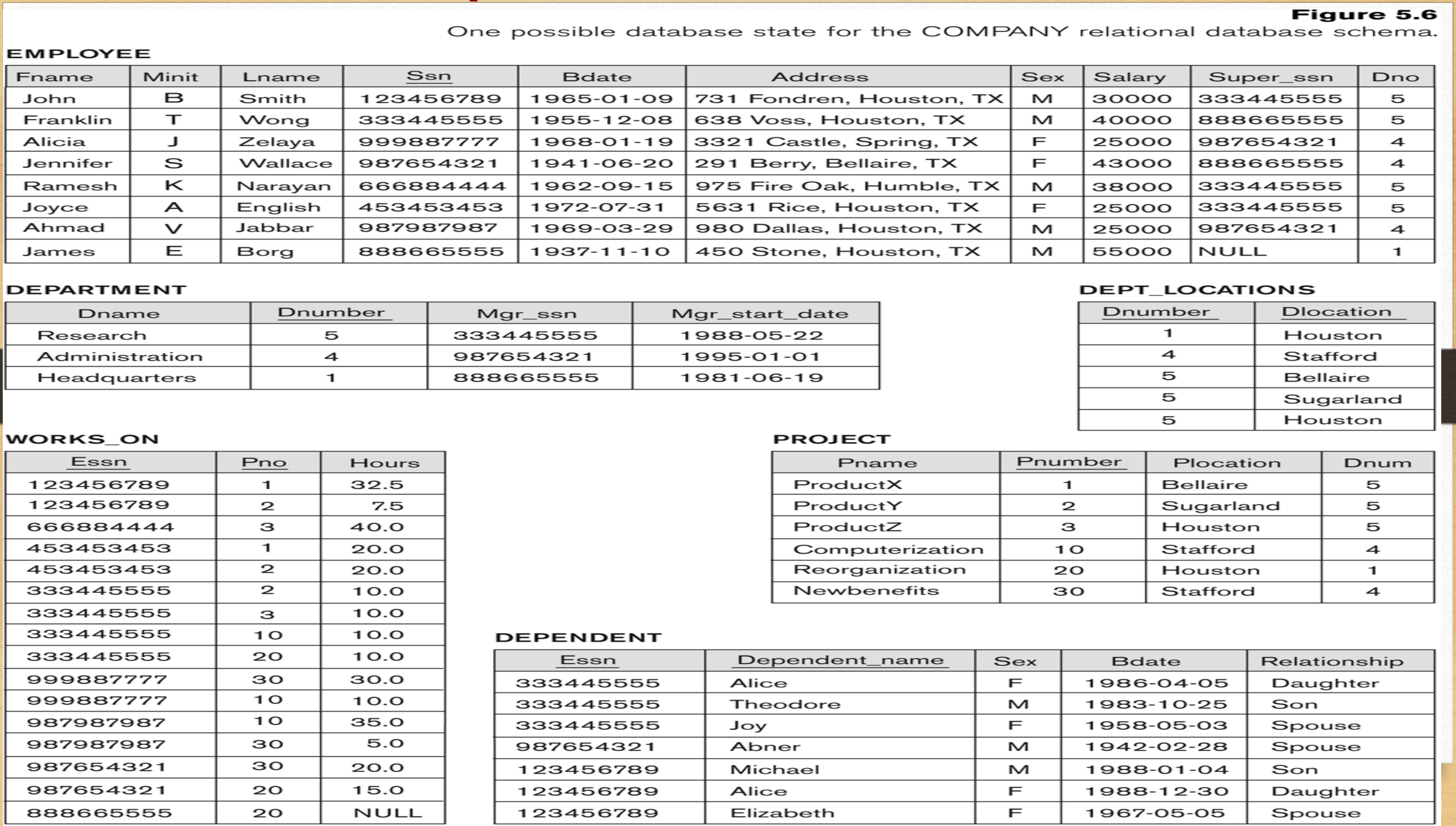


Figure 5.6
One possible database state for the COMPANY relational database schema.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Update Operations on Relations

- INSERT a tuple.
- DELETE a tuple.
- MODIFY a tuple.
- Integrity constraints should not be violated by the update operations.
- Several update operations may have to be grouped together.
- Updates may **propagate** to cause other updates automatically. This may be necessary to maintain integrity constraints.

Update Operations on Relations

- In case of integrity violation, several actions can be taken:
  - Cancel the operation that causes the violation (RESTRICT or REJECT option)
  - Perform the operation but inform the user of the violation
  - Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
  - Execute a user-specified error-correction routine

Possible violations for each operation

- INSERT may violate any of the constraints:
  - **Domain constraint:**
    - if one of the attribute values provided for the new tuple is not of the specified attribute domain
  - **Key constraint:**
    - if the value of a key attribute in the new tuple already exists in another tuple in the relation
- **Referential integrity:**
  - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
- **Entity integrity:**
  - if the primary key value is null in the new tuple

Possible violations for each operation

- DELETE may violate only referential integrity:
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
    - Can be remedied by several actions: RESTRICT, CASCADE, SET NULL (see Chapter 8 for more details)
      - RESTRICT option: reject the deletion
      - CASCADE option: propagate the new primary key value into the foreign keys of the referencing tuples
      - SET NULL option: set the foreign keys of the referencing tuples to NULL
  - One of the above options must be specified during database design for each foreign key constraint

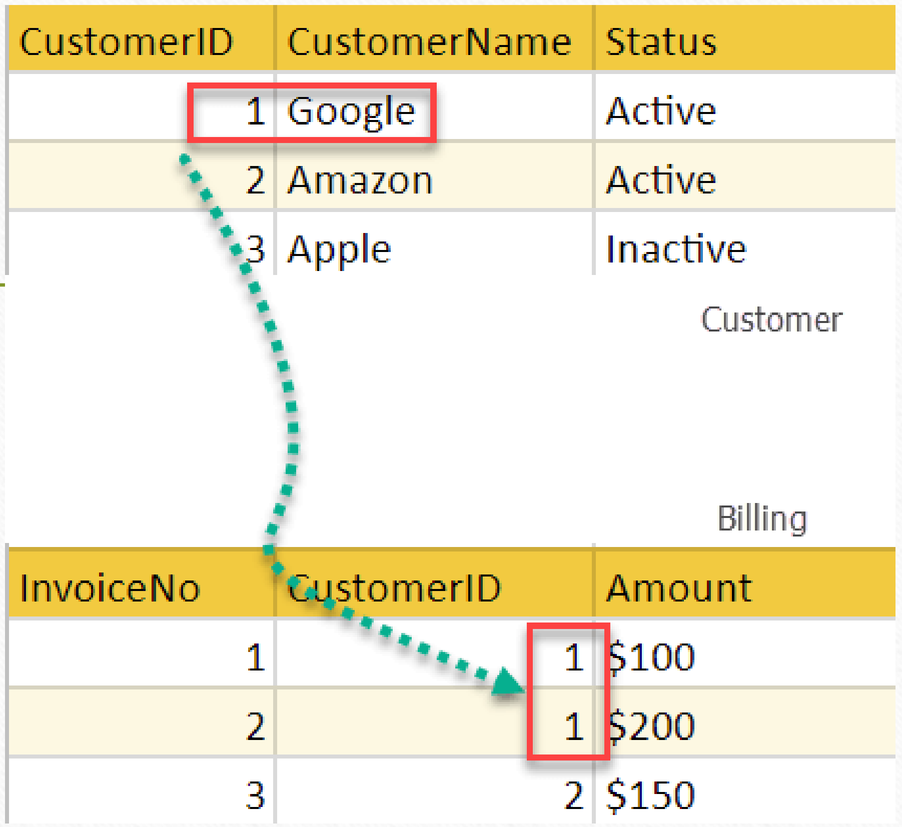
Possible violations for each operation

- UPDATE may violate domain constraint and NOT NULL constraint on an attribute being modified
- Any of the other constraints may also be violated, depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain constraints

Relational Integrity constraints

- Relational Integrity constraints is referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.
- There are many types of integrity constraints. Constraints on the Relational database management system is mostly divided into three main categories are:
- Domain constraints
- Key constraints
- Referential integrity constraints

- In the above example, we have 2 relations, Customer and Billing.
- Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300



Customer

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Billing

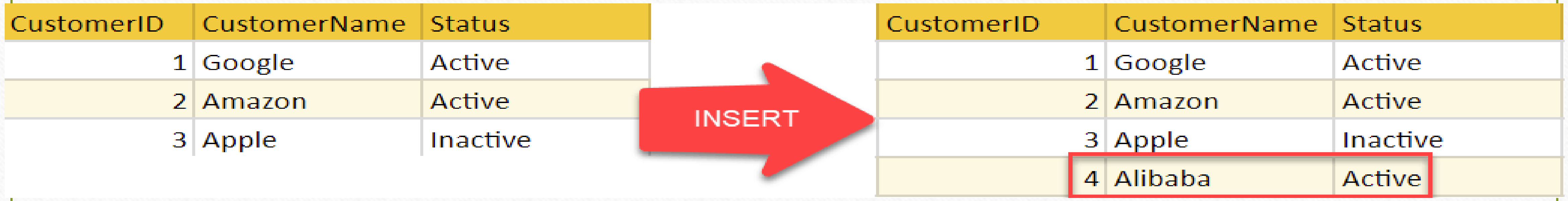
InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

Operations in Relational Model

- Four basic update operations performed on relational database model are
- Insert, update, delete and select.
- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.
- Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

Insert Operation

- The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.

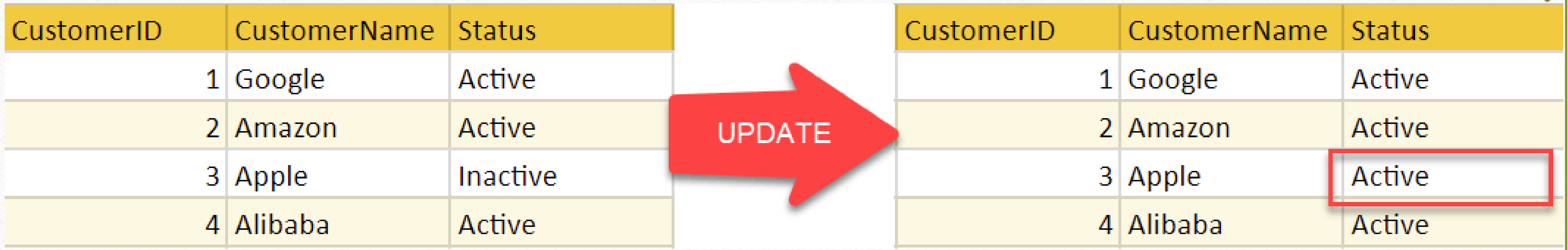


CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Update Operation

- You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

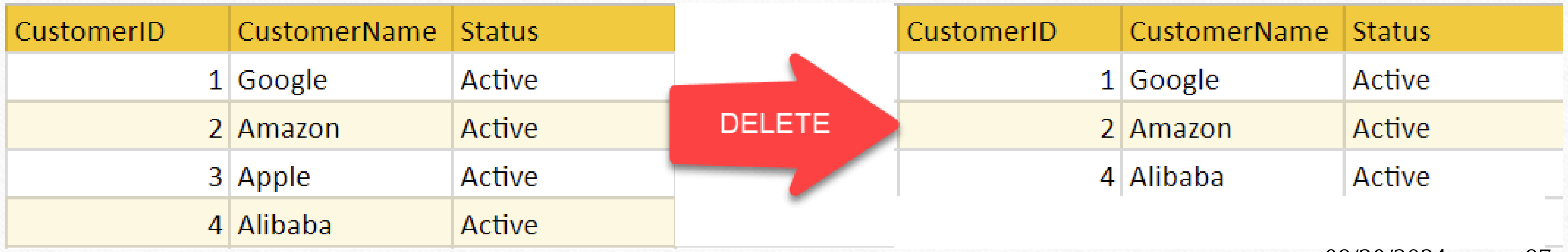


CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

Delete Operation

- To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.
- In the below-given example, CustomerName= "Apple" is deleted from the table.
- The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Active
4	Alibaba	Active

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
4	Alibaba	Active

- SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.
- Constraints can be divided into the following two types,
- **Column level constraints:** Limits only column data.
- **Table level constraints:** Limits whole table data.

- Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.
- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL Constraint

- **NOT NULL** constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.
- One important point to note about this constraint is that it cannot be defined at table lev

UNIQUE Constraint

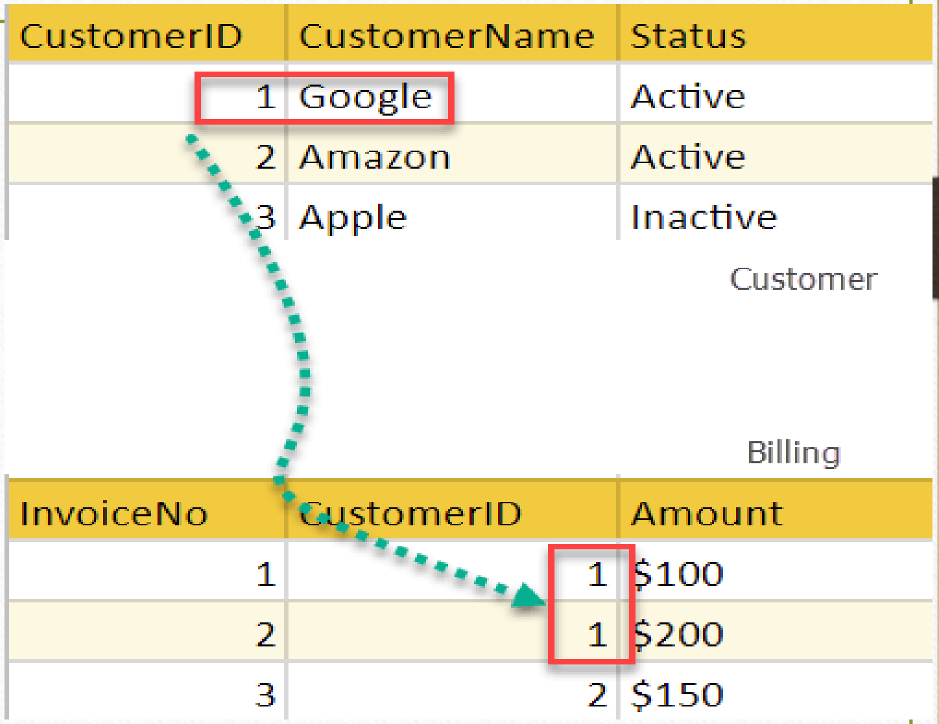
- **UNIQUE** constraint ensures that a field or column will only have unique values. A **UNIQUE** constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Primary Key Constraint

- Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Foreign Key Constraint

- FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:



Customer

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Billing

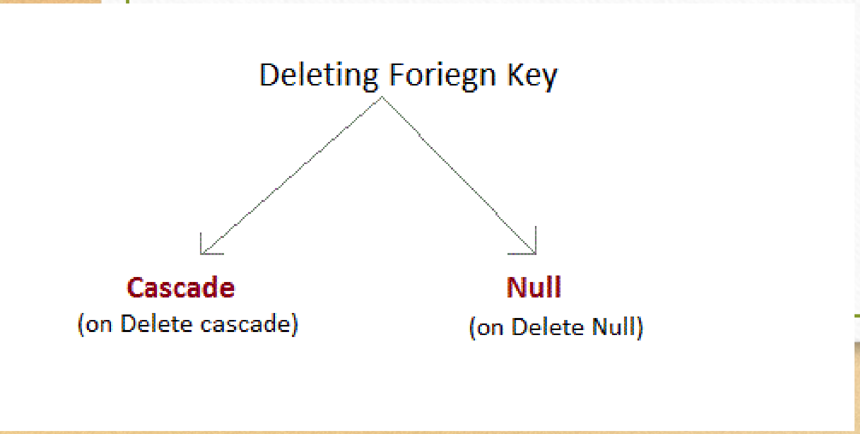
InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

CHECK Constraint

- **CHECK** constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Behaviour of Foriegn Key Column on Delete

- There are two ways to maintain the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child



- **On Delete Cascade :** This will remove the record from child table, if that value of foriegn key is deleted from the main table.
- **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.
- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.