# DATA MODELS

# DBMS

Mrs. Soniya Sharma.

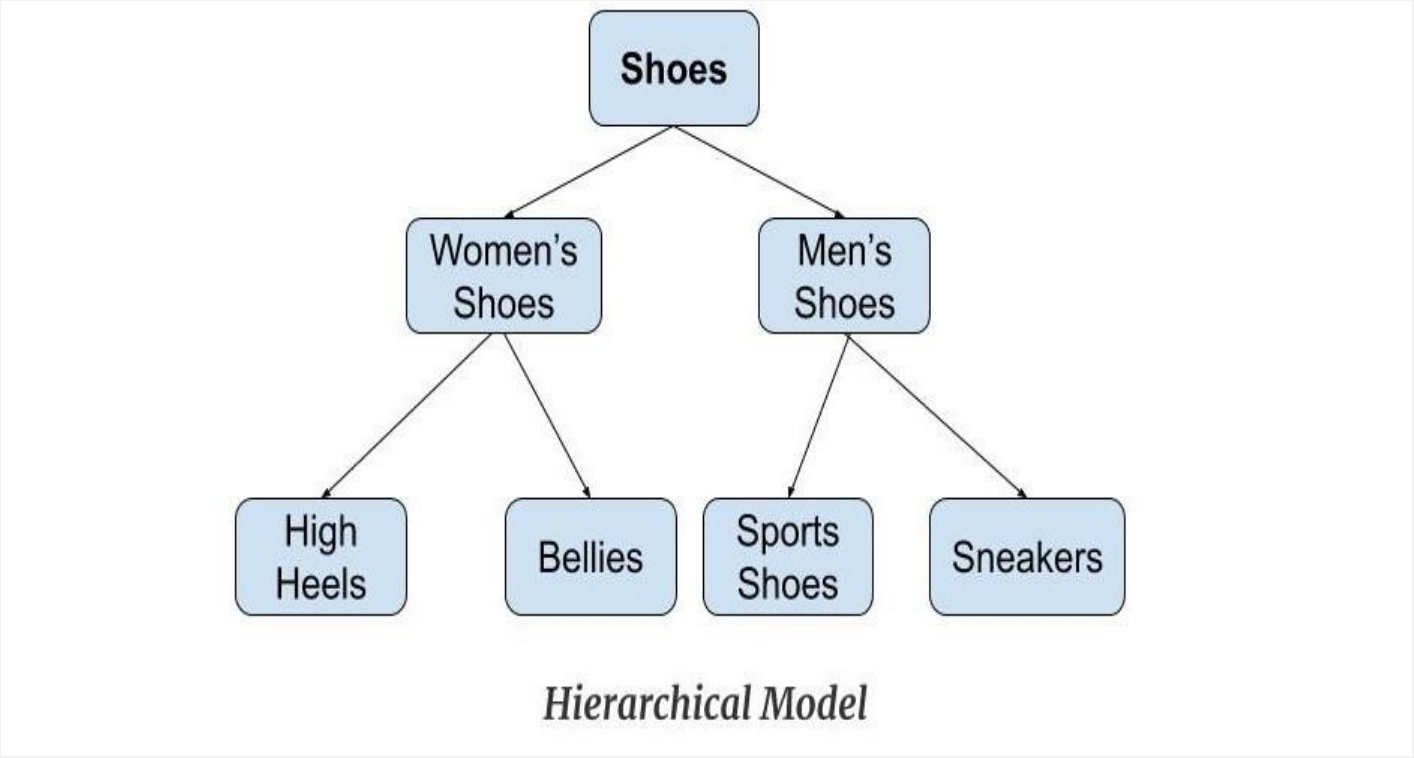
J. M. Patel College of Commerce

# DATA MODELS

- **Defi : graphical representation of data.**
- **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.
- **It is used for Understanding each aspect of business.**

# DATA MODELS

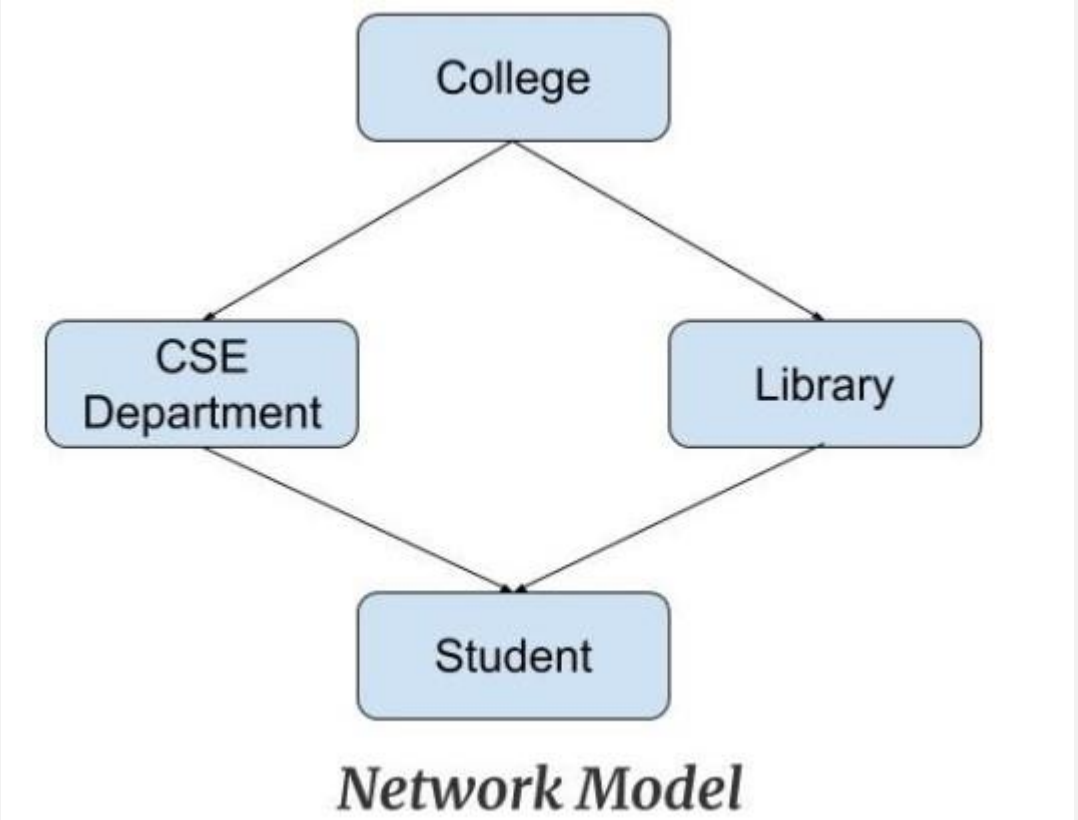
- **Hierarchical Model:**
- Hierarchical Model was the first DBMS model. This model organizes the data in the hierarchical tree structure. The hierarchy starts from the root which has root data and then it expands in the form of a tree adding child node to the parent node. This model easily represents some of the real-world relationships like food recipes, sitemap of a website etc. ***Example:*** We can represent the relationship between the shoes present on a shopping website in the following way:




*Hierarchical Model*

# DATA MODELS

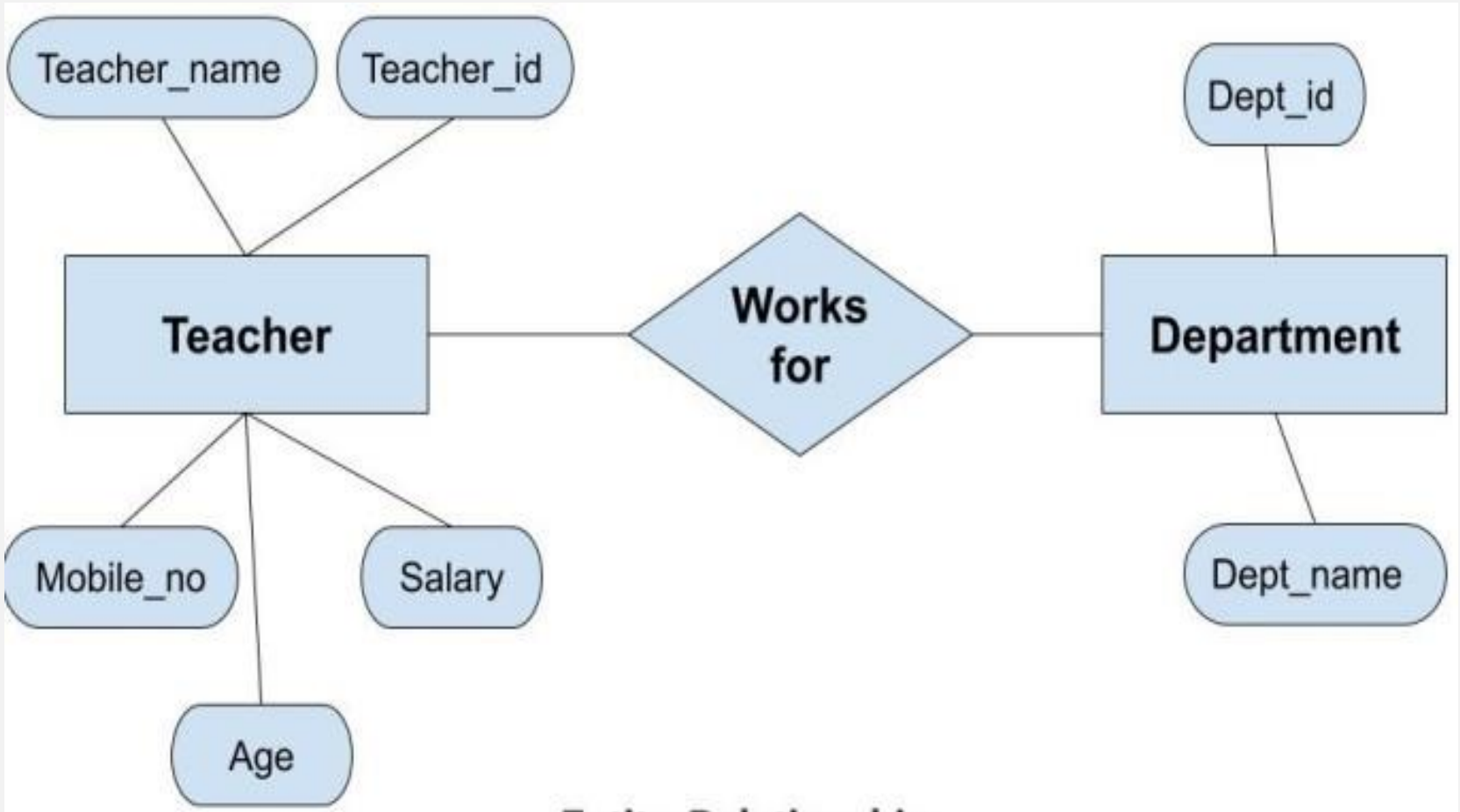
- **Network Model**
- This model is an extension of the hierarchical model. It was the most popular model before the relational model. This model is the same as the hierarchical model, the only difference is that a record can have more than one parent. It replaces the hierarchical tree with a graph. ***Example:*** In the example below we can see that node student has two parents i.e. CSE Department and Library. This was earlier not possible in the hierarchical model.




*Network Model*

# DATA MODELS

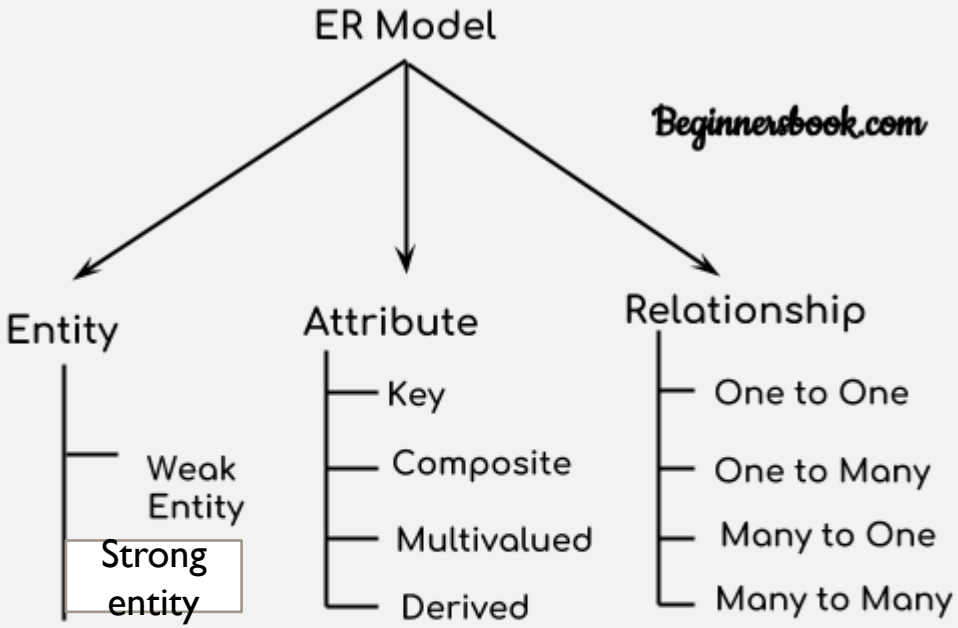
- **Entity-Relationship Model**
- Entity-Relationship Model or simply ER Model is a high-level data model diagram. In this model, we represent the real-world problem in the pictorial form to make it easy for the stakeholders to understand. It is also very easy for the developers to understand the system by just looking at the ER diagram. We use the ER diagram as a visual tool to represent an ER Model. ER diagram has the following three components:
- ***Entities:*** Entity is a real-world thing. It can be a person, place, or even a concept. *Example:* Teachers, Students, Course, Building, Department, etc are some of the entities of a School Management System.
- ***Attributes:*** An entity contains a real-world property called attribute. This is the characteristics of that attribute. *Example:* The entity teacher has the property like teacher id, salary, age, etc.
- ***Relationship:*** Relationship tells how two entities are related. *Example:* Teacher works for a department.



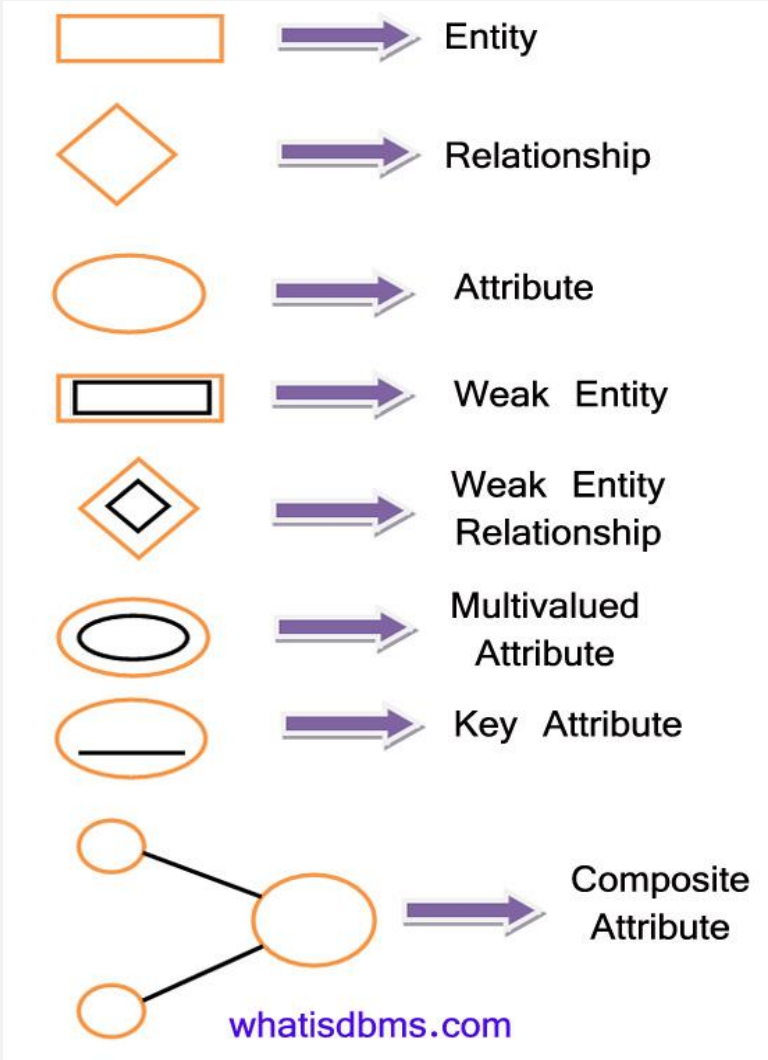

Entity-Relationship Model

# DATA MODELS

- An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. https://beginnersbook.com/2015/04/e-r-model-in-dbms/

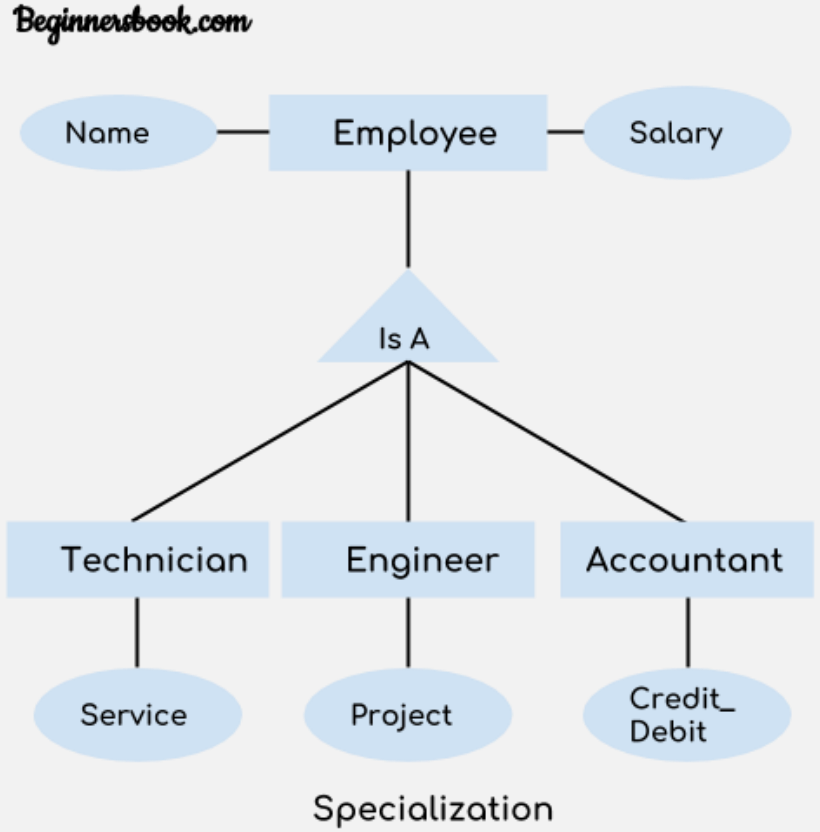



Components of ER Diagram



# DATA MODELS

- **Generalization** is a process in which the common attributes of more than one entities form a new entity. This newly formed entity is called generalized entity.
- **Specialization** is a process in which an entity is divided into sub-entities. You can think of it as a reverse process of generalization, in generalization two entities combine together to form a new higher level entity. Specialization is a top-down process.




Specialization

# DATA MODELS

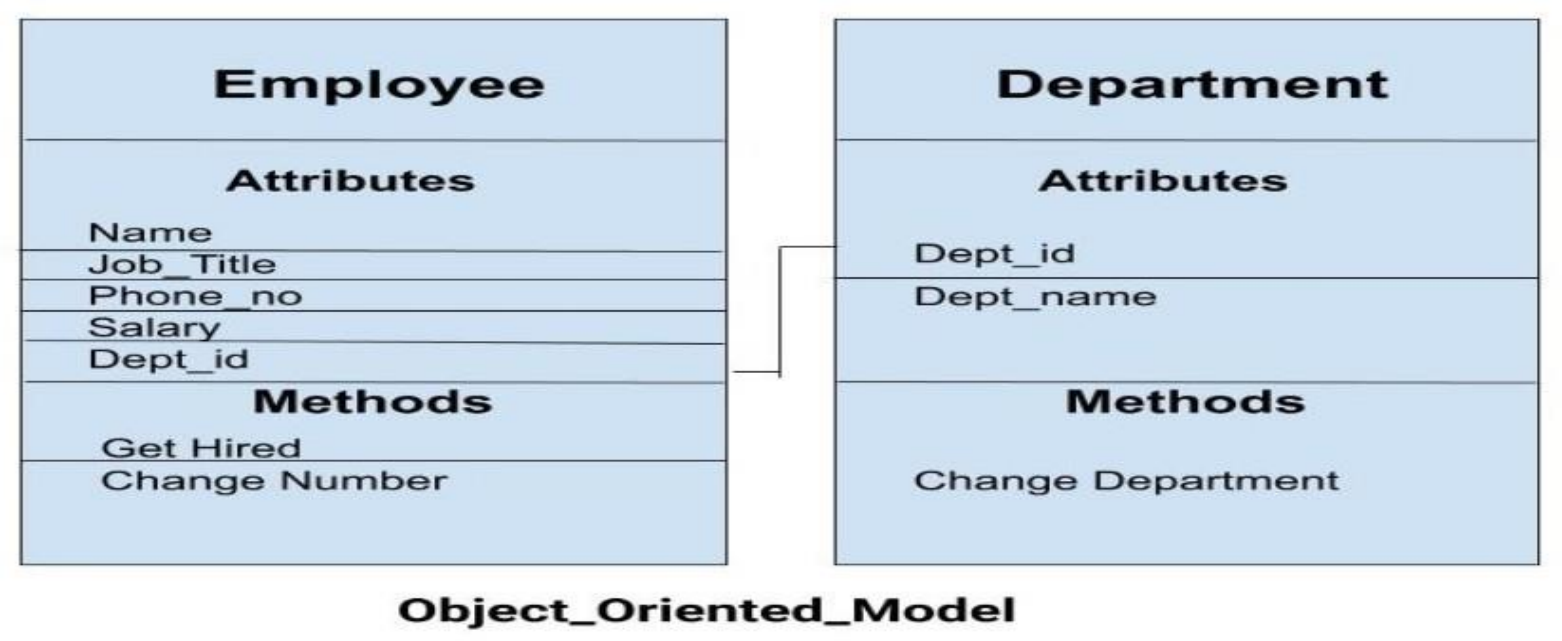
- **Relational Model**
- Relational Model is the most widely used model. In this model, the data is maintained in the form of a two-dimensional table. All the information is stored in the form of row and columns. The basic structure of a relational model is tables. So, the tables are also called *relations* in the relational model. ***Example:*** In this example, we have an Employee table.

| Emp_id | Emp_name | Job_name | Salary | Mobile_no | Dep_id | Project_id |
|---|---|---|---|---|---|---|
| AfterA001 | John | Engineer | 100000 | 9111037890 | 2 | 99 |
| AfterA002 | Adam | Analyst | 50000 | 9587569214 | 3 | 100 |
| AfterA003 | Kande | Manager | 890000 | 7895212355 | 2 | 65 |

**EMPLOYEE TABLE**

# DATA MODELS

- **Object-Oriented Data Model**
- The real-world problems are more closely represented through the object-oriented data model. In this model, both the data and relationship are present in a single structure known as an object. We can store audio, video, images, etc. in the database which was not possible in the relational model(although you can store audio and video in relational database, it is advised not to store in the relational database). In this model, two are more objects are connected through links. We use this link to relate one object to other objects. This can be understood by the example given below.




Object_Oriented_Model

# HOMEWORK

- Make ER diagram for following:
- 1. Library Management System (staff, students, teachers, book)
- 2. Hospital Management System (doctor, patient, staff, bill, room, ambulance)
- 3. Online Shopping (customer, product, bill)
- 4. College management System (department, student, teacher , staff, library, examination, admission)
- 5. Hotel management System (customer, rooms, staff, bill)
- 6. Online Examination System (student, admin, subject, question, answer, result)

# AGGREGATE FUNCTION (SUMMARY FUNCTION)

- Count() - function returns the number of rows that matches a specified criterion.

select count(*) from syit_class where per>60

select count(*) from syit_class where bloodgr like "b +“

select count(*) from syit_class (it will give all rows present in table)

- Sum() – function returns the addition value of numeric column. Ignores NULL value automatically.

select sum(per) from syit_class

- Avg() - function returns the average value of a numeric column. Ignores NULL value automatically.

select avg(per) from syit_class

# AGGREGATE FUNCTION (SUMMARY FUNCTION)

- Min() - function returns the smallest value of the selected column.

select rollno,name,min(per) from syit_class

- Max() - function returns the largest value of the selected column.

select rollno,name,max(per) from syit_class

- Extra notes
- Like - The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

# NULL VALUE

- A field with a NULL value is a field with no value.
- A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!
- Null value is missing or unknown value.
- Test for NULL Values – is null and is not null
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.
- Is null - The IS NULL operator is used to test for empty values (NULL values). It will look for NULL values.

Eg. Select * from syit_class where per is null

- Is not null - The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

Eg. Select * from syit_class where per is not null

# E. F. CODD'S RULES

- **Rule 1: Information rule**
  - All information(including metadata) is to be represented as stored data in cells of tables i.e. all information must be stored in **rows and columns format.**
- **Rule 2: Guaranteed Access**
  - Each unique piece of data(atomic value) should be **accessible** by : **Table Name + Primary Key(Row) + Attribute(column)**.
- **Rule 3: Systematic treatment of NULL**
  - Null has several meanings, it can mean missing data, not applicable or no value. It should be handled consistently. Also, Primary key must not be null, ever.
- **Rule 4: Active Online Catalog**
  - Database dictionary(catalog) is the structure description of the complete **Database** and it must be stored online. The Catalog must be governed by same rules as rest of the database. The same query language should be used on catalog as used to query database.
- **Rule 5: Powerful and Well-Structured Language**
  - One well structured language must be there to provide all manners of access to the data stored in the database. Example: **SQL**, etc. If the database allows access to the data without the use of this language, then that is a violation.
- Rule 6:View Updation Rule
  - All the view that are theoretically updatable should be updatable by the system as well.

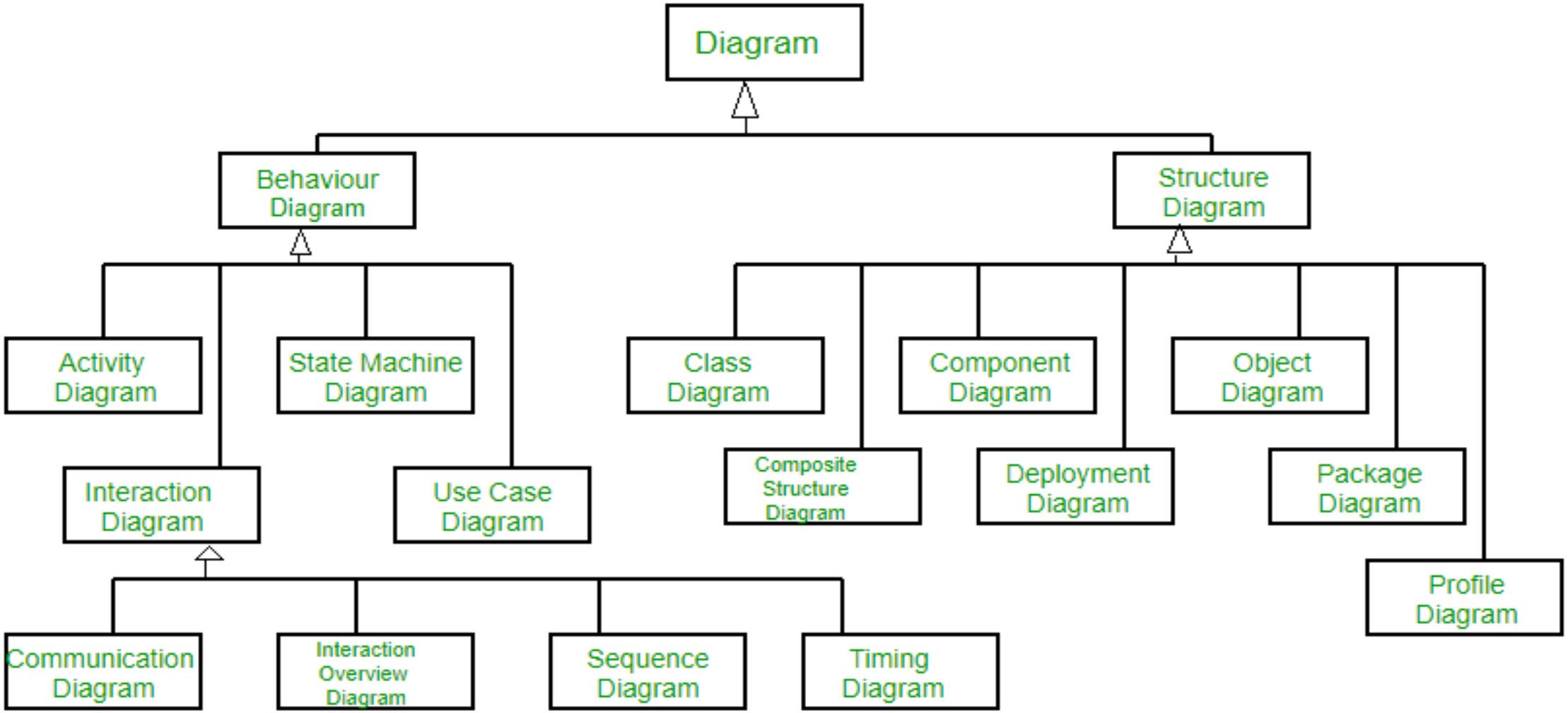
# E. F. CODD'S RULES

- **Rule 7: Relational Level Operation**
  - There must be Insert, Delete, Update operations at each level of relations. Set operation like Union, Intersection and minus should also be supported.
- **Rule 8: Physical Data Independence**
  - The physical storage of data should not matter to the system. If say, some file supporting table is renamed or moved from one disk to another, it should not effect the application.
- **Rule 9: Logical Data Independence**
  - If there is change in the logical structure(table structures) of the database the user view of data should not change. Say, if a table is split into two tables, a new view should give result as the join of the two tables. This rule is most difficult to satisfy.
- **Rule 10: Integrity Independence**
  - The database should be able to enforce its own integrity rather than using other programs. Key and Check constraints, trigger etc, should be stored in Data Dictionary. This also make **RDBMS** independent of front-end.
- **Rule 11: Distribution Independence**
  - A database should work properly regardless of its distribution across a network. Even if a database is geographically distributed, with data stored in pieces, the end user should get an impression that it is stored at the same place. This lays the foundation of **distributed database**.
- **Rule 12: Nonsubversion Rule**
  - If low level access is allowed to a system it should not be able to subvert or bypass integrity rules to change the data. This can be achieved by some sort of looking or encryption.

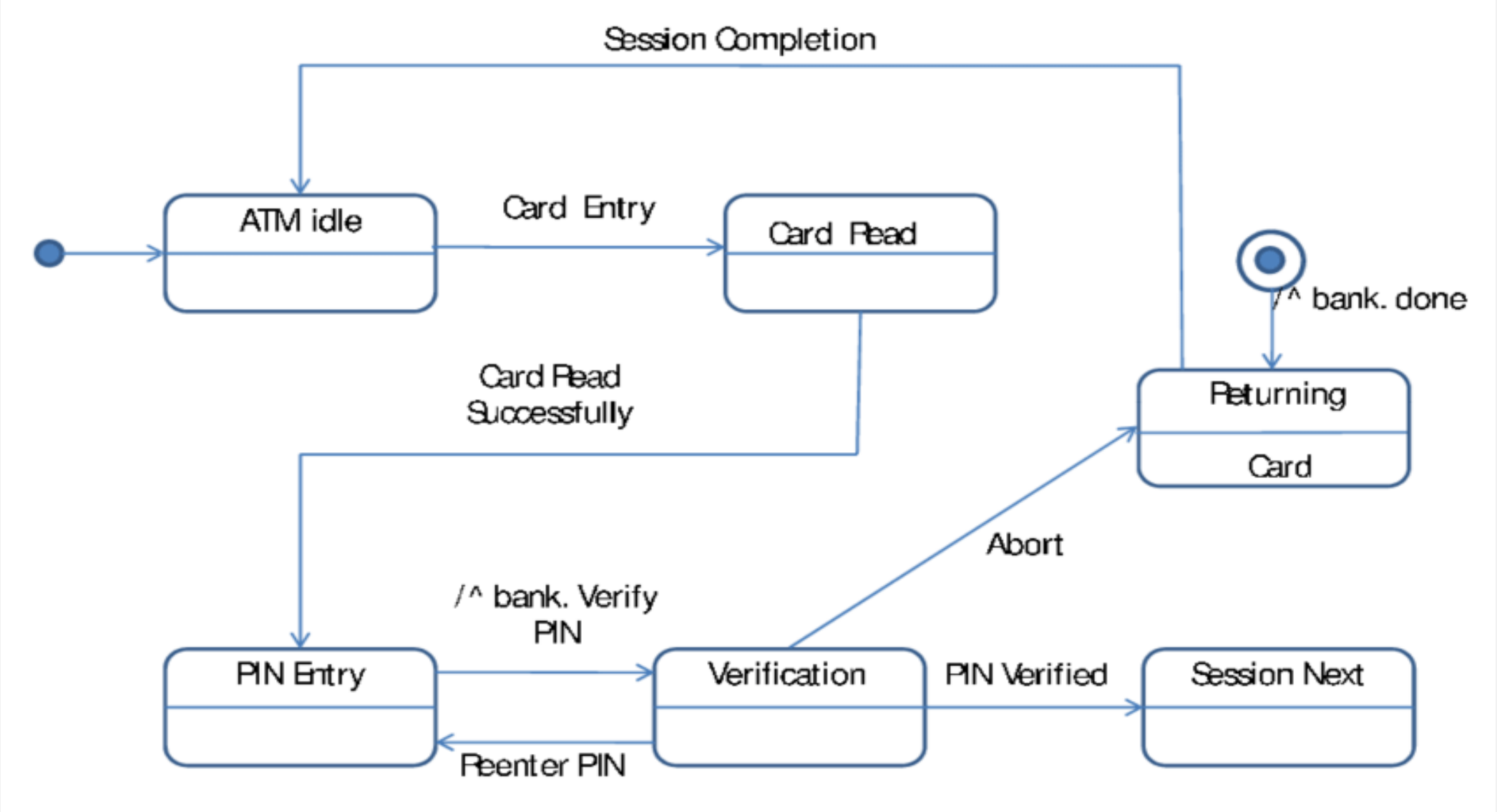
# INTRODUCTION TO UML

- **UML - Unified Modeling Language (UML)** is a general purpose modelling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.
- UML is **not a programming language**, it is rather a visual language. We use UML diagrams to portray the **behavior and structure** of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis.
- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So UML becomes essential to communicate with non programmers essential requirements, functionalities and processes of the system.
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system.

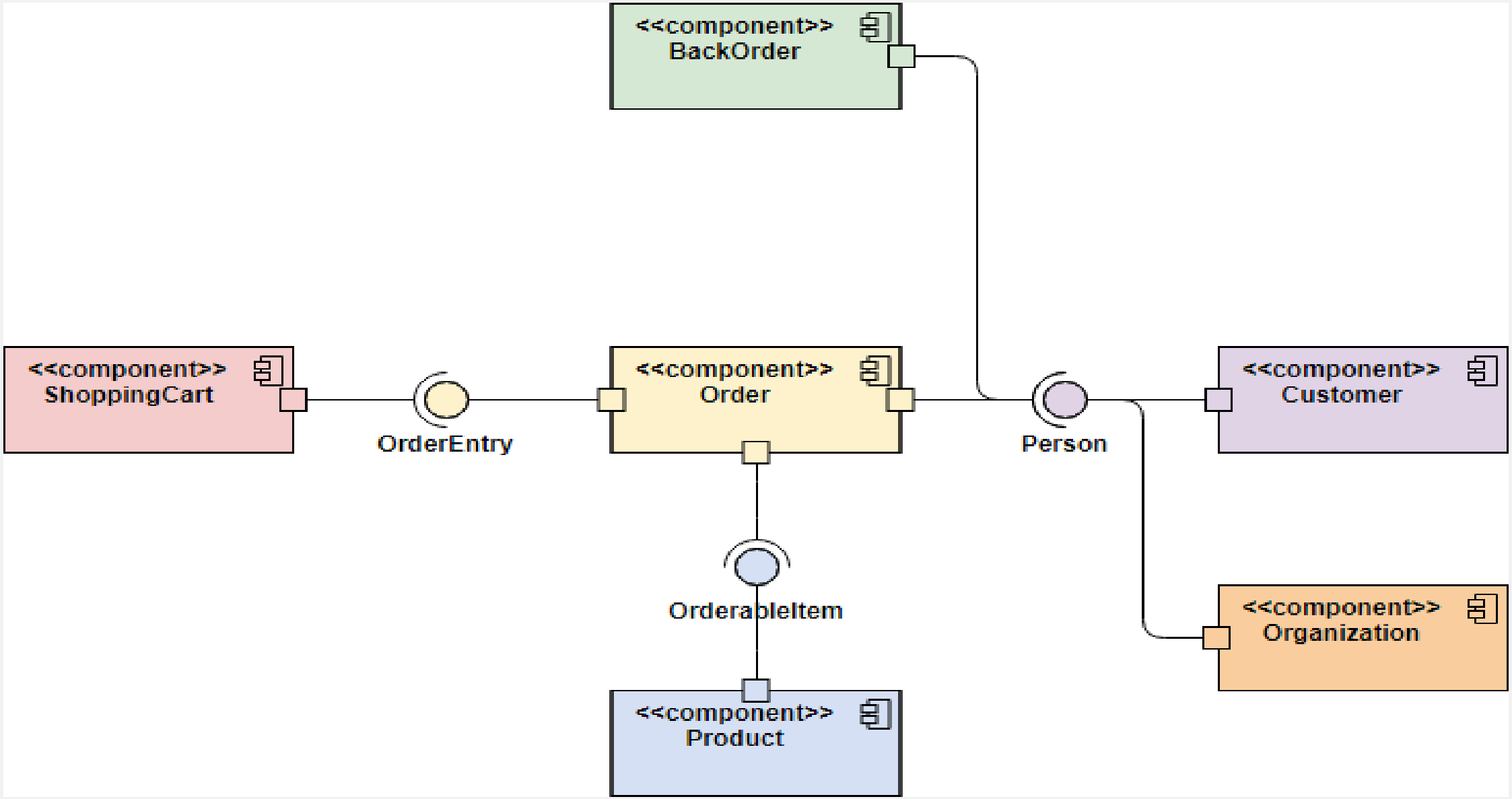
# INTRODUCTION TO UML



# INTRODUCTION TO UML – EXAMPLE OF STATE DIAGRAM



# INTRODUCTION TO UML – EXAMPLE OF COMPONENT DIAGRAM



# ORDER BY CLAUSE

- **To arrange data in ascending or descending order we can make use of  order by clause.**
- **We can arrange data in ascending order by using asc as parameter. By default asc is selected.**
- **We can arrange data in descending order by using desc as parameter. LiveClass@0480982**
- **Example**
- select * from student order by percent desc
- Above query will arrange data of students as per their percentage in descending order.

# CONSTRAINTS

- 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 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 - **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.
- **Eg.** `CREATE TABLE Student(s_id int NOT NULL, Name varchar(60) not null, Age int)`
- `S_id and name will not accept null values.`

# CONSTRAINTS

- **Unique** -A **UNIQUE** constraint field will not have duplicate data.
- Eg. `CREATE TABLE Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);`
- S_id will be unique and not null.
- **Primary key** - A Primary Key must contain unique value and it must not contain null value.
- Eg. `CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);`
- S_id will not accept null and duplicate values.
- 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.
- Foreign key values matches with the primary key.
- Foreign key can be null and it can be duplicated.
- Eg. `CREATE table Order_Detail( order_id int PRIMARY KEY, order_name varchar(60) NOT NULL, c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id) );`

# CONSTRAINTS

- **Check Constraint**
- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.
- Eg. Create table student( roll int. name char(20), gender char(2), check (gender in ('M','F')))
- This query will allow to insert values 'M' and 'F' in gender column.

- **DEFAULT Constraint**
- The DEFAULT constraint is used to provide a default value for a column.
- The default value will be added to all new records IF no other value is specified.
- Eg. Create table student(roll int, name char(20), grade char(2) default 'P')
- In above query the default grade will be P.

# JOINS

- **Subquery:** A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- Example :

select cid,cname from customer where cid not in (select cid from orders)

- **Equi join :** SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables. An equal sign (=) is used as comparison operator in the where clause to refer equality.
- Example

select distinct cname from customer , orders where customer.cid = orders.cid

- **Non equi join :** The SQL NON EQUI JOIN uses comparison operator instead of the equal sign like **>, <, >=, <=** along with conditions.

Example : Select * From orders o, customer c Where price **>**300 and c.cid=o.cid

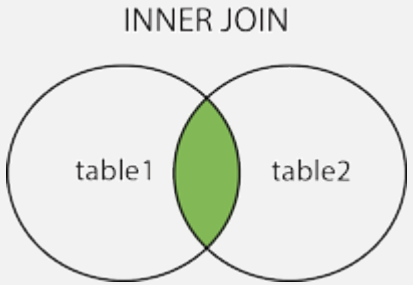
# JOINS

- **Inner join – display the records from tables that satisfies the conditions.**
- **Left outer join – displays all the record from left table and only the condition satisfying records from right table.**
- **Right outer join - displays all the record from right table and only the condition satisfying records from left table.**
- **Full outer join – display all the record from both table even if condition is not satisfied in following sequence :**
  - Condition satisfying records from both table.
  - Condition not satisfying records from left table.
  - Condition not satisfying records from right table.
- **Self join**

**[Study practical no. 4 ]**

# JOINS

- **Inner Join :** The INNER JOIN keyword selects records that have matching values in both tables.




- **Example**

select distinct cname

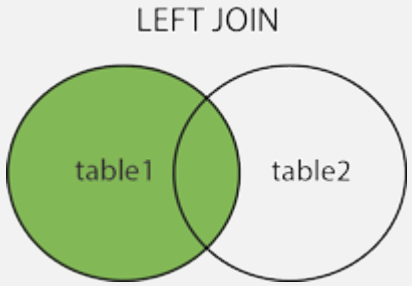
from customer **inner join** orders

**on** customer.cid = orders.cid

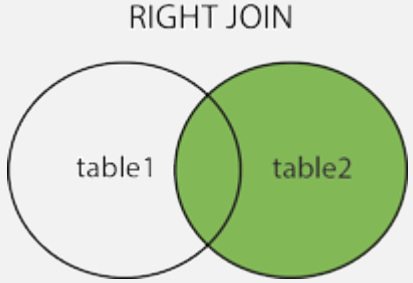
-

# JOINS

- **Left outer join** - In some databases LEFT JOIN is called LEFT OUTER JOIN. The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.



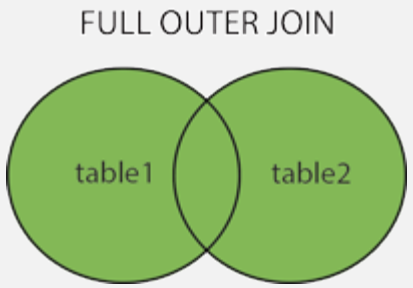

- Example - select * from customer c right outer join orders o ON prod_name="Maggi" and c.cid = o.cid
- **Right outer join** -In some databases RIGHT JOIN is called RIGHT OUTER JOIN. The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.




- Example - select * from customer c right outer join orders o ON prod_name="Maggi" and c.cid = o.cid

# JOINS

- **Full Outer join -**
- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
- **Note:** FULL OUTER JOIN can potentially return very large result-sets!
- **Tip:** FULL OUTER JOIN and FULL JOIN are the same.
-




- Exanple :

```
select * from customer c full outer join orders o
on prod_name="Maggi" and c.cid = o.cid
```

# SELF JOIN

- A self JOIN is a regular join, but the table is joined with itself.

- Example : Display the customer name having same age.

```
select c1.cname,c1.age
from customer c1,customer c2
where c1.age=c2.age and c1.cid<>c2.cid
```

# GROUP BY CLAUSE

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.
- Example :

```
select address, count(cid)
 from customer
group by address
```

- Having clause : The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
- Example :

```
select address, count(cid) from customer group by address having count(cid)>2
```

# VIEWS

- A view is a virtual table based on the result set of an SQL statement.
- **CREATE VIEW Indians AS**
  SELECT CustomerName, phone
  FROM Customers
  WHERE Country = “India";
- A view can contain all rows/columns of a table or some selected rows/columns from a table.
- A view can be created from one or many tables which depends on the written SQL query to create a view.
- It simplifies multi table queries.
- Views summarize data from various tables which can be used to generate reports.
- View can be updatable.
- If we update view the changes are updated in underlying tables.

# VIEWS

- Advantages of Views :
- **Security :** Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data.
- **Query simplicity :** A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- **Structural simplicity**
- **Insulation from change:** A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.
- **Data integrity :** If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets specified integrity constraints.