**Unit I**

**Data:**

Data is a set of values of subjects with respect to qualitative or quantitative variables. Data and information or knowledge is often used interchangeably; however data becomes information when it is viewed in context.

Computer data is information processed or stored by a computer. This information may be in the form of text documents, images, audio clips, software programs, or other types of data.

**Database:**

A database is a collection of information that is organized so that it can be easily accessed, managed and updated. It is a collection of related data and data is a collection of facts and figures that can be processed to produce information.

Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it.

**Database Systems:**

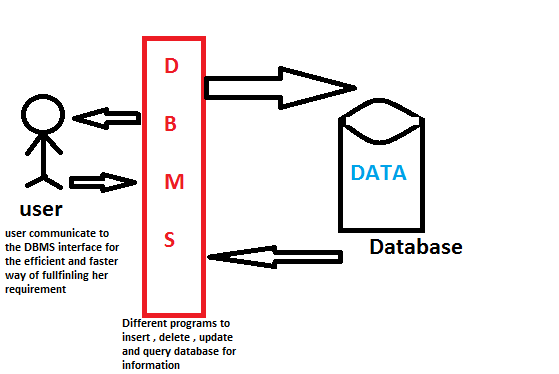
The collection of database and database software together is known as a database system.  Database system is a storage that store data and information of database.

**Database Management System:**

A database management system stores data in such a way that it becomes easier to retrieve, manipulate, and produce information. A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data.

A DBMS is software that allows creation, definition and manipulation of database, allowing users to store, process and analyze data easily.

DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.



DBMS also provides protection and security to the databases. It also maintains data consistency in case of multiple users.

**Database-System Applications:**

1. **Enterprise Information:**

* **Sales:** For customer, product, and purchase information.
* **Accounting:** For payments, receipts, account balances, assets and other accounting information.
* **Human resources:** For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.

1. **Manufacturing:** For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items. For each manufacturing unit or warehouse or distribution center a database is needed to keep the records of ins and outs. For example, distribution Centre should keep a track of the product units that supplied into the Centre as well as the products that got delivered out from the distribution Centre on each day; this is where DBMS comes into picture.
2. **Online retailers:** For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations. Online shopping websites such as Amazon, Flipkart stores the product information, addresses and preferences, credit details and provides the relevant list of products based on your query. All this involves a Database management system.
3. **Banking and Finance:**

* **Banking:** For customer information, accounts, loans, and banking transactions.
* **Credit card transactions:** For purchases on credit cards and generation of monthly statements.
* **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

1. **Universities:** For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).
2. **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
3. **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.

**Purpose of Database systems:**

Database systems arose in response to early methods of computerized management of commercial data. That is traditional file management system.

As an example, consider part of a university organization that, among other data, keeps information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files.

To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:

• Add new students, instructors, and courses

• Register students for courses and generate class rosters

• Assign grades to students, compute grade point averages (GPA), and generate transcripts

System programmers wrote these application programs to meet the needs of the university.

New application programs are added to the system as the need arises. For example, suppose that a university decides to create a new major (say, computer science). As a result, the university creates a new department and creates new permanent files or adds information to existing files to record information about all the instructors in the department, students in that major, course offerings, degree requirements, etc. The university may have to write new application programs to deal with rules specific to the new major. New application programs may also have to be written to handle new rules in the university. Thus, as time goes by, the system acquires more files and more application programs.

The following are the disadvantages of traditional system:

1. **Data redundancy and inconsistency:** Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files).

For example, if a student has a double major (say, music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree.

For example, a changed student address may be reflected in the Music department records but not elsewhere in the system.

1. **Difficulty in accessing data:** Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of all students. The university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner
2. **Data isolation:** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
3. **Integrity problems:** The data values stored in the database must satisfy certain types of consistency constraints. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.
4. **Atomicity problems:** A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $500 from the account balance of department A to the account balance of department B. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of department A but was not credited to the balance of department B, resulting in an inconsistent database state. Clearly, it is essential to database consistency that either the credit or debit occur, or that neither occur. That is, the funds transfer must be atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.
5. **Concurrent-access anomalies:** For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. Consider department A, with an account balance of $10,000. If two department clerks debit the account balance (by say $500 and $100, respectively) of department A at almost exactly the same time, the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state. Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programs run concurrently, they may both read the value $10,000, and write back $9500 and $9900, respectively. Depending on which one writes the value last, the account balance of department A may contain $9500 or $9900, rather than the correct value of $9400. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because many different application programs that have not been coordinated previously may access data.

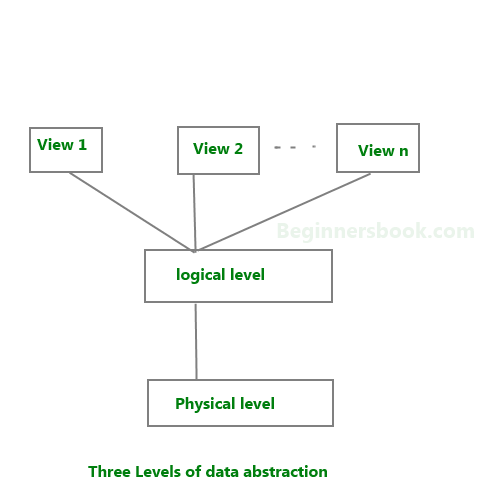
As another example, suppose a registration program maintains a count of students registered for a course, in order to enforce limits on the number of students registered. When a student registers, the program reads the current count for the courses, verifies that the count is not already at the limit, adds one to the count, and stores the count back in the database. Suppose two students may both read the value 39, and both would then write back 40, leading to an incorrect increase of only 1, even though two students successfully registered for the course and the count should be 41. Furthermore, suppose the course registration limit was 40; in the above case both students would be able to register, leading to a violation of the limit of 40 students.

1. **Security problems:** Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

**Views of Data:**

A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

1. **Physical level:** The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.
2. **Logical level:** The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as physical data independence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
3. **View level:** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.



For example, we may describe a record as follows:

type instructor = record

ID : char (5);

name : char (20);

dept name : char (20);

salary : numeric (8,2);

end;

This code defines a new record type called instructor with four fields. Each field has a name and a type associated with it. A university organization may have several such record types, including:

* Department, with fields dept name, building, and budget
* Course, with fields course id, title, dept name, and credits
* Student, with fields ID, name, dept name, and tot cred

At the **physical level,** an instructor, department, or student record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the **logical level,** each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the **view level**, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database.

For example, clerks in the university registrar office can see only that part of the database that has information about students; they cannot access information about salaries of instructors.

**Instances and Schemas:**

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the **database schema.**

Database systems have several schemas, partitioned according to the levels of abstraction.

The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level.

A database may also have several schemas at the view level, sometimes called subschemas, which describe different views of the database.

**Data Models:**

It is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.

* **Relational Model:** The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations. The relational model is an example of a record-based model.

Record-based models are so named because the database is structured in fixed- format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes.

The columns of the table correspond to the attributes of the record type. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

* **Entity-Relationship Model:** The entity-relationship (E-R) data model uses a collection of basic objects, called entities, and relationships among these objects.

An entity is a “thing” or “object” in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design.

* **Object-Based Data Model:** Object-oriented programming (especially in Java,

C++, or C#) has become the dominant software-development methodology.

This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

* **Semi-structured Data Model:** The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The Extensible Markup Language (XML) is widely used to represent semi-structured data.

**Relational Model:**

The relational model 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.

Some popular Relational Database management systems are:

* DB2 and Informix Dynamic Server - IBM
* Oracle and RDB – Oracle
* SQL Server and Access – Microsoft

**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

**Advantages of using Relational model**

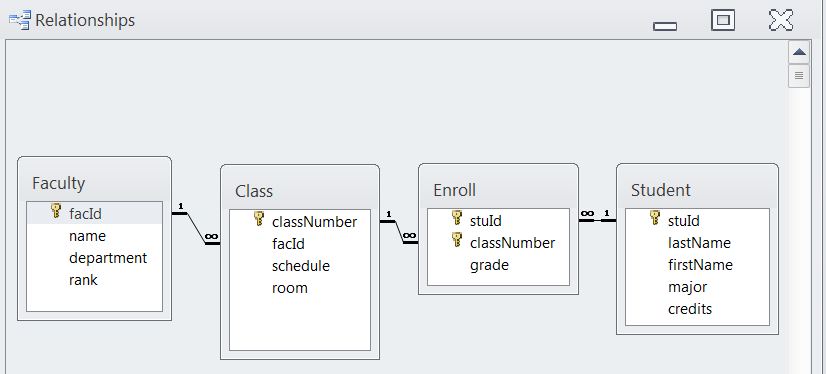
* **Simplicity**: A relational data model is simpler than the hierarchical and network model.
* **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
* **Easy to use**: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
* **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
* **Data independence**: The structure of a database can be changed without having to change any application.
* **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

**Disadvantages of using Relational model**

* Few relational databases have limits on field lengths, which can't be exceeded.
* Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
* Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

**STUDENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Roll\_No | Name | Address | Phone | Age |
| 1 | Ram | Hyderabad | 9456789 | 18 |
| 2 | Ramesh | Medak | 9657843 | 18 |
| 3 | Raju | Adilabad | 9876543 | 20 |
| 4 | Ram | Secunderabad | 9823456 | 20 |



**Entity Relational Model:**

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).

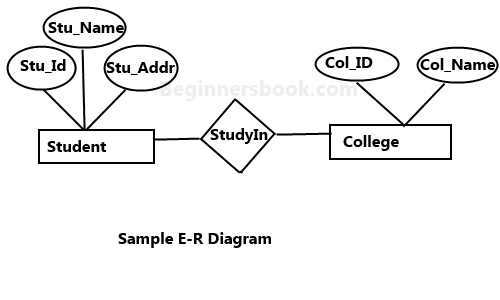
An ER model is a design or blueprint of a database that can later be implemented as a database.

The main components of E-R model are: entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes.

In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.

In the following diagram we have two entities Student and College and their relationship. Student entity has attributes such as Stu\_Id, Stu\_Name & Stu\_Addr and College entity has attributes such as Col\_ID & Col\_Name.

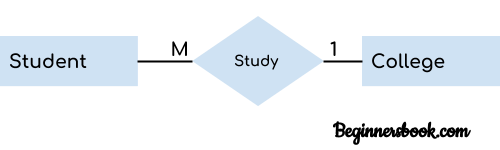


## Components of a ER Diagram:

As shown in the above diagram, an ER diagram has three main components:  
1. Entity  
2. Attribute  
3. Relationship

### 1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.  
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college.



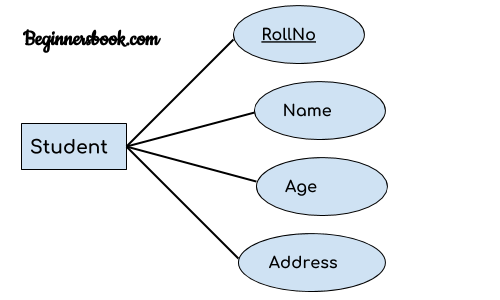
**Weak Entity:**  
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.  


### 2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

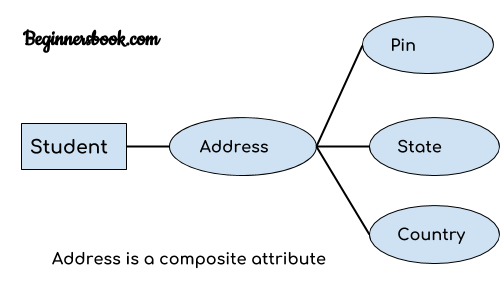
1. Key attribute  
2. Composite attribute  
3. Multivalued attribute  
4. Derived attribute

#### 1. Key attribute:

A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the**text of key attribute is underlined.**

#### 2. Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute.

For example, in student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.  


#### 3. Multivalued attribute:

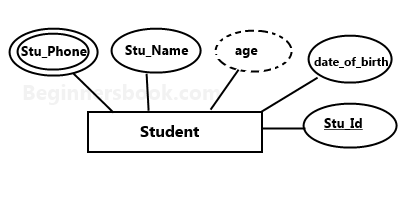
An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram.

For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

#### 4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram.

For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

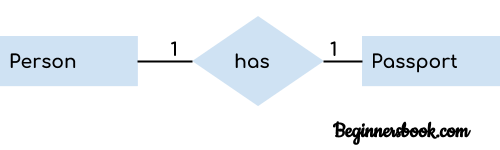
**E-R diagram with multivalued and derived attributes**:  


### 3. Relationship

A relationship is represented by diamond shape in ER diagram; it shows the relationship among entities. There are four types of relationships:  
1. One to One  
2. One to Many  
3. Many to One  
4. Many to Many

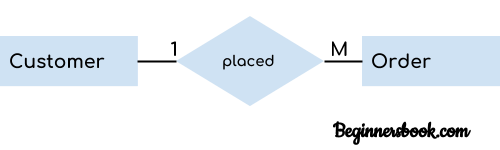
#### 1. One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship.

For example, a person has only one passport and a passport is given to one person.  


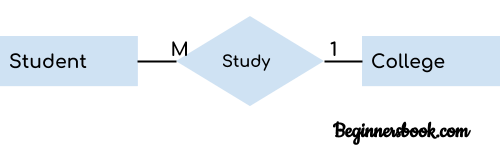
#### 2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships.

For example – a customer can place many orders but a order cannot be placed by many customers.  


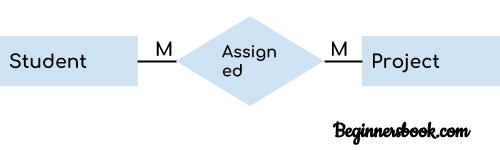
#### 3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship.

For example – many students can study in a single college but a student cannot study in many colleges at the same time.  


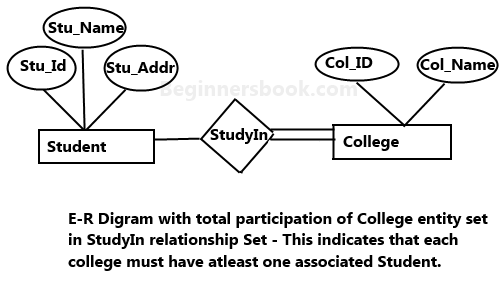
#### 4. Many to Many Relationship

When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationships.

For example, a can be assigned to many projects and a project can be assigned to many students.  


## Total Participation of an Entity set

A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set.

For example: In the below diagram each college must have at-least one associated Student.  


**Normalization:**

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

|  |  |  |  |
| --- | --- | --- | --- |
| EmpId | Empname | Empaddress | Empdept |
| 101 | Rick | Delhi | D001 |
| 101 | Rick | Delhi | D002 |
| 123 | Sachin | Mumbai | D003 |
| 166 | Rahul | Chennai | D004 |
| 166 | Rahul | Chennai | D005 |

**Update anomaly**: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent.

If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

**Insert anomaly**: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp\_dept field doesn’t allow nulls.

**Delete anomaly**: Suppose, if at a point of time the company closes the department D003 then deleting the rows that are having emp\_dept as D003 would also delete the information of employee Maggie since she is assigned only to this department.