BLOCK I:
RELATIONAL MODEL CONCEPTS, LANGUAGES, DESIGN THEORY AND METHODOLOGY

UNIT-1: RELATIONAL DATA MODEL AND RELATIONAL DATABASE CONSTRAINTS

Space for learners:

Unit Structure:

1.1 INTRODUCTION

In this unit, you will study about the relational database model: the various components, characteristics and limitations. This unit will also familiarize you with the key terms related to relational model such as domain, attribute, tuple, &the various types of keys such as primary, alternate, foreign, candidate, logical and super key with examples. Here, the emphasis will also be given on the various relational constraints, e.g., domain constraint, key constraint, entity integrity constraint and referential integrity constraint. And you will also learn about the Entity-Relationship diagram.

Space for learners:

1.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Understand in detail the relational model, its advantages, limitations and applications
- Explain the key terms of relational model
- Structure of the relational model
- Learn various characteristic of relations
- Understand the different keys in relational model
- Learn about the relational constraints
- Operations in Relational Model with Constraint Violations
- Analyze the E-R diagram

1.3RELATIONAL MODEL

Storing and managing information is one of the most important tasks for computers. The way in which information is organized can have a profound effect on how easy it is to access and manage. Perhaps the simplest but most versatile way to organize information is to store it in the forms of tables. Table is the backbone of the relational model.

The relational model is centered on this idea: the organization of data into collections of two-dimensional tables called “relations.” The data and relationships are represented by collection of inter-

related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records. The table name and column names are helpful to interpret the meaning of values in each row. In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that can appear in each column is represented by a domain of possible values.

Originally, the relational model of database was introduced in 1970, by English Computer Scientist **Edgar F. Codd**. The relational data model was developed for databases — that is, information stored over a long period of time in a computer system — and for database management systems, the software that allows people to store, access, and modify this information. Databases still provide us with important motivation for understanding the relational data model. They are found today not only in their original, large-scale applications such as train booking systems or hospital management systems, but in desktop computers handling individual activities such as maintaining expense records, homework grades, and many other uses.

Space for learners:

1.3.1 Advantages of Relational Model

The relational model is the most dominant database model. It has lots of advantages:

1. **Simple Model**

Compared to other types of models, a relational database model is much simpler. It is free from query processing and complex structuring. As a result, it does not require any complex queries. A simple SQL query is sufficient enough for handling.

2. **Data Accuracy**

In the relational database system, there can be multiple tables related to one another with the use of primary key and foreign key concepts. Hence, there is no repletion of data. There is no chance for duplication of data. Hence the accuracy of data in the relational database is more than any other database system.

3. **Easy Access to Data**

In the Relational Database System, there is no pattern or pathway for accessing the data, as to another type of databases can be accessed only by navigating through a tree or a hierarchical model. Anyone who accesses the data can query any table in the relational database. Using join queries and conditional statements one can combine all or any number of related tables in order to fetch the required data. Resulting data can be modified based on the values from any column, on any number of columns, which permits the user to effortlessly recover the relevant data as the result. It allows one to pick on the desired columns to be incorporated in the outcome so that only appropriate data can be displayed.

4. **Data Integrity**

Data integrity is a crucial characteristic of the Relational Database system. It ensures that all the data in the database confines within suitable arrangements and the data necessary for creating the relationships are present. This relational reliability amongst the tables in the database helps in avoiding the records from being imperfect, isolated or unrelated. Data integrity aids in making sure of the relational database's other significant characteristics like case of use, precision, and stability of the data.

5. **Flexibility**

A Relational Database system by itself possesses qualities for leveling up, expanding for bigger lengths, as it is endowed with a bendable structure to accommodate the constantly shifting requirements. This facilitates the increasing incoming amount of data, as well as the update and deletes wherever required. This model consents to the changes made to a database configuration as well, which can be applied without difficulty devoid of crashing the data or the other parts of the database.

A Data Analyst can insert, update or delete tables, columns or individual data in the given database system promptly and easily, in order to meet the business needs. There is supposedly no boundary on the number of rows, columns or tables a relational database can hold. In any practical application, development and transformation are restricted by the Relational Database Management System and the hardware contained by the servers. So, these changes can create

Space for learners:

an alteration in other peripheral functional devices connected to the particular relational database system.

6. **Normalization**

The normalization process provides a set of regulations, characteristics, and purposes for the database structure and evaluation of a relational database model. Normalization aims at illustrating multiple levels of breaking down the data. Any level of normalization is expected to be accomplished on the same level, that is, before moving ahead to the next levels. A relational database model is usually confirmed to be normalized, only when it satisfies the necessary conditions of the third normalization form. Normalization offers an impression of reassurance on the database plan, to be extra strong and reliable.

7. **High Security**

As the data is divided amongst the tables of the relational database system, it is possible to make a few tables to be tagged as confidential and others not. This segregation is easily implemented with a relational database management system, unlike other databases. When a data analyst tries to login with a username and password, the database can set boundaries for their level of access, by providing admission only to the tables that they are allowed to work on, depending on their access level.

8. **Feasible for Future Modifications**

As the relational database system holds records in separate tables based on their categories, it is straightforward to insert, delete or update records that are subjected to the latest requirements. This feature of the relational database model tolerates the newest requirements that are presented by the business. Any number of new or existing tables or columns of data can be inserted or modified depending on the conditions provided, by keeping up with the basic qualities of the relational database management system.

1.3.2 Limitations of Relational Model

The relational model suffers from certain limitations. The relational model has been developed to meet the requirements of business information processing. While applying the relational model to the

Space for learners:

application areas, such as Computer-Aided Design (CAD), simulation and image processing, many shortcomings have been noticed in this model also. The various shortcomings of this model may be discussed as follows:

1. **Cost**

The underlying cost involved in a relational database is quite expensive. For setting up a relational database, there must be separate software which needs to be purchased. And a professional technician should be hired to maintain the system. All these can be costly, especially for businesses with small budget.

2. **Performance**

Always the performance of the relational database depends on the number of tables. If there are a greater number of tables, the response given to the queries will be slower. Additionally, more data presence not only slows down the machine, it eventually makes it complex to find information. Thus, a relational database is known to be a slower database.

3. **Physical Storage**

A relational database also requires tremendous amount of physical memory since it is with rows and columns. Each of the operation depends on separate physical storage. Only through proper optimization, the targeted applications can be made to have maximum physical memory.

4. **Complexity**

Although a relational database is free from complex structuring, occasionally it may become complex too. When the amount of data in a relational database increases, it eventually makes the system more complicated.

5. **Information Loss**

Large organizations tend to use a greater number of database systems with more tables. This information can be used to be transferred from one system to another. This could pose a risk of data loss.

6. **Structure Limitations**

The fields that are present on a relational database has limitations. Limitations are in that sense that it cannot accommodate more

Space for learners:

information. Despite if more information is provided, it may lead to data loss. Therefore, it is necessary to describe the exact amount of data volume which the field will be given.

Space for learners:

1.4 COMPONENTS AND RELATIONAL TERMINOLOGIES

The main principle of the relational model is the information principle: all information is represented by data values in relations. The three components- structural, manipulative and integrity- make up this model. These components are defined as follows:

- The structural component is concerned with how data is represented.
- The manipulative component is concerned with how data is operated upon.
- The integrity component is concerned with determining which states are valid for a database.

The various relational terminologies are described as follows-

Domain: A domain is a set of values permitted for an attribute in a table. Domain is atomic. For example, ROLL_NO can only be a positive integer. A data type or format is also specified for each domain. It is possible for several attributes to have the same domain.

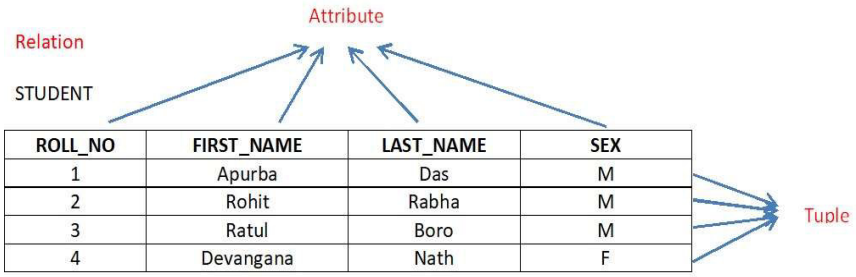


Fig 1.1: Relational Model Concepts

Attribute: Attributes are the characteristics of a relation. Each column in a table is the attribute. Attributes are the properties which define a relation. e.g.,ROLL_NO, FIRST_NAME etc of the relation Student. Each attribute in a relational model must have domain information. Domain information contains the following:

- ***Data Type***: Databases provide support for different types of data and their variants. For example, integer, float etc.
- ***Length***: Length means number of characters or digits that an attribute value has. For example, when we assign a PIN code, it has 6 digits.
- ***Date format***: A date contains day, month and year. These three must be given in combination. Such as DD/MM/YYYY or MM/DD/YYYY or YYYY/MM/DD etc.
- ***Range***: A range is specified by lower and upper bounds of data values that an attribute may have.
- ***Constraints***: These are particular type of conditions that put restrictions on values that are allowed.
- ***NULL support***: There is a support for NULL values in relational model. Some particular attribute may remain blank. For example, in a relation the column "PAN_Number" may be blank as a person may not have a PAN number.
- ***Default Value***: If nothing is entered, database assigns a default value. Relational model supports the facility that the default value may be set for every attribute.

Attributes can be of many types:

- ***Composite vs Simple attribute***: Composite attributes can be subdivided into smaller attributes. For example, the "Name" of a student can be divided into "First_Name", "Middle_Name" and "Last_Name". On the otherhand, simple attributes are the attributes that can't be subdivided into smaller attributes. For example, the "Roll_No" of a STUDENT.
- ***Single Valued vs Multivalued***: Single valued attributes are the attributes that have a single value for a particular entity. For example, the "Last_Name" of an EMPLOYEE. But the multivaled attributes can have more than one value for a particular entity. For example, "Mobile _Number" of an EMPLOYEE.

Space for learners:

- ***Derived vs Stored attributes***: Derived attributes are the attributes whose values can be derived from the value of some other attributes. For example, the age of a student can be derived from date of birth of the student. The "Date_of_Birth" is called the stored attribute from which you can derive some other attribute.
- ***NULL attribute***: A certain entity may not possess a value for an attribute. This will mean "not applicable" or that the value is unknown" or "non-existent". For example, there may be chance when a student has no phone no. In that case the "Phone_No" attribute is called NULL attribute.

Tuple – It is nothing but a single row of a table, which contains a single record.

Relations- are 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.

Relation Schema- A relational schema is the design for the table. It includes none of the actual data, but is like a blueprint or design for the table, so describes what columns are on the table and the data types. It may show basic table constraints (e.g., if a column can be null) but not how it relates to other tables.

A relation schema R, denoted by R (A1,A2, ..., An), is made up of a relation name R and a list of attributes, A1,A2, ...,An. Each attribute Ai is the name of a role played by some domain D in the relation schema R.D is called the domain of Ai and is denoted by dom(Ai). The relation schema R(A1,A2, ...,An), also denoted by r(R),is a set of n -tuplesr= {t1,t2, ...,tm}.

Degree- is the number of attributes n of its relation schema. A relation ofdegree four, which stores information about college students, would contain four attributes describing each student as follows:

STUDENT(Roll_No, First_Name, Last_name, Sex)

Cardinality: Total number of rows present in the Table.

Relation Instance–Relation instance is a finite set of tuples at a given time. Relation instances do not have duplicate tuples.

Space for learners:

Null Value: Afield with a NULL value is a field with no value. Primary key can't be a null value.

Table 1.1: STUDENT Relation

Roll	Name	Phone	Age
1	Nipun	1234567890	26
2	Nava	1234567891	28
3	Mohit	1234567892	19

Characteristics of Relations

- **Interpretation(Meaning)ofaRelation:**Therelationschema canbeinterpreted asa declaration or a type of assertion. Each tuple in the relation can then be interpreted as a fact or a particular instance of the assertion. For example (Table 1.1), the first tuple in above table asserts the fact that there is a STUDENT whose Roll Number is 1, Name is "Nipun", Phone is 1234567890 and Age is 26, and so on.
- **Ordering of Tuples in a Relation:** A relation is defined as a set of tuples. The tuples in a relation do not have any particular order. In other words, a relation is not sensitive to the ordering of tuples. However, in a file, there always is an order among the records. Tuple ordering is not part of a relation definition because a relation attempts to represent facts at a logical or abstract level. Many tuple orders can be specified on the same relation. For example, tuples in the STUDENT relation (Table 1.1) could be ordered by values of Name, Roll, Age etc. The definition of a relation does not specify any order: There is no preference for one ordering over another.
- **Ordering of attributes in a Relation:** The ordering of attributes is not important, because the attribute name appears with its value. There is no reason to prefer having one attribute value appear before another in a tuple. When a relation is implemented as a file, the attributes and the values within tuples are ordered.
- **Values in a tuple**: All values are considered atomic. A special null value is used to represent values that are

Space for learners:

unknown or inapplicable to certain tuples. In general, NULL values, means value unknown or value exists but is not available.

Relational Model Notations

We will use the following notation in our presentation:

- A relation schema R of degree n is denoted by R(A 1, A 2 ,..., A n).
- The uppercase letters Q, R,S denote relation names.
- The lowercase letter sq, r,s denote relation states.
- The letters t, u, v denote tuples.
- In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation—the current relation state—whereas STUDENT (Roll , Name , ...)refers only to the relation schema.
- An attribute A can be qualified with the name of the relation, R, to which it belongs by using the dot notation R.A - for example (Table 1.1), STUDENT. Name or STUDENT. Age. This is because the same name may be used for two attributes in different relations. However, all attribute names in a particular relation must be distinct.
- An n-tuple t in a relation r(R) is denoted by t = $<v_1, v_2, ..., v_n>$, where v_i is the value corresponding to attribute A_i.

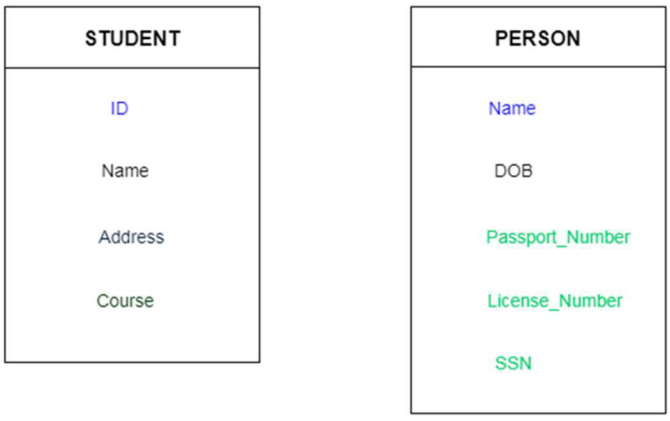


Fig 1.2: Illustration of Relational Schema

Space for learners:

1.5 KEYS IN RELATIONAL MODEL

Space for learners:

Keys are very important part of Relational database model. A Key can be a single attribute or a group of attributes, where the combination may act as a key. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

Why do we need a Key?

In real world applications, number of tables required for storing the data is huge, and the different tables are related to each other as well. Also, tables store a lot of data in them. A table generally extends to thousands of records stored in them, unsorted and unorganised.

Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data?

To avoid all this, Keys are defined to easily identify any row of data in a table.

For example: In STUDENT table (Fig 1.2), The attribute ID is used as a key because it is unique for each student. In PERSON table, passport_number, license_number, SSN are keys since they are unique for each person.

Types of Keys:

Different types of keys are shown in the following figure (Fig 1.3):

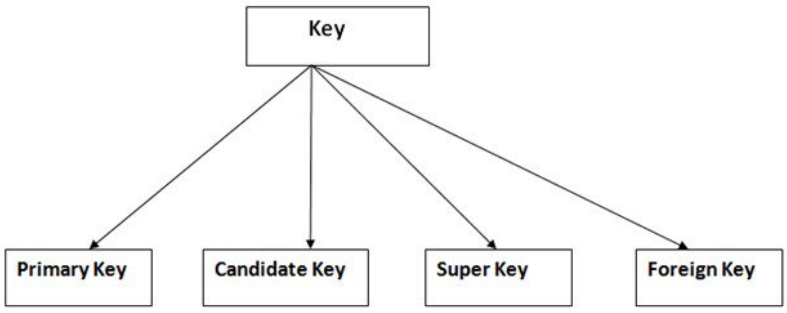


Fig 1.3 Different types of Keys

- **Primary key**
  - It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain

multiple keys as we saw in PERSON table (Fig 1.2). The key which is most suitable from those lists become a primary key.

- In the EMPLOYEE table (Fig. 1.4), ID can be primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary key since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.

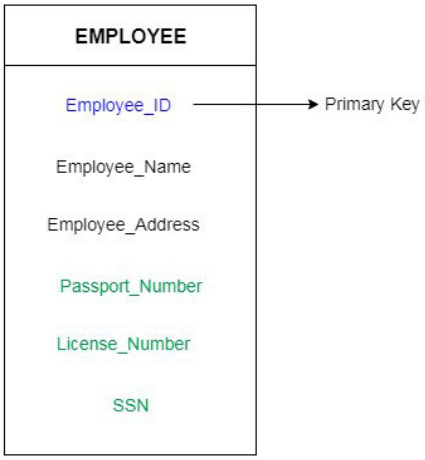


Fig 1.4: Illustration of Primary Key

- **Candidate key**

Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key.

For example: In the EMPLOYEE table(Fig 1.5), Employee_id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.

Space for learners:

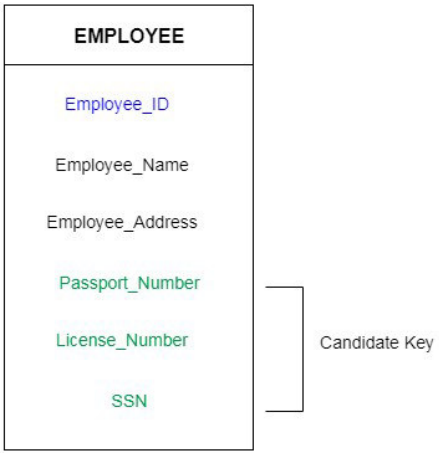


Fig 1.5: Illustration of Candidate Key

- **Super Key**

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.

For example: In the above EMPLOYEE table (Fig 1.5), for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key. The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

Let's take a simple **STUDENT** table (Table 1.2) with the attributes: **student_id, name**, **phone** and **age**.

Table 1.2STUDENT Relation

student_id	name	phone	age
1	Rohit	1234567890	17
2	Rohit	1234567891	18
3	Mohit	1234567892	19

In the table defined above super key would include student_id, (student_id, name), phone etc.

Confused? The first one is pretty simple as student_id is unique for every row of data; hence it can be used to identity each row uniquely.

Space for learners:

Next comes, (student_id, name), now name of two students can be same, but their student_id can't be same hence this combination can also be a key.

Space for learners:

- **Foreign key**
  - Foreign keys are the column of the table which is used to point to the primary key of another table. If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute.
  - In a company, every employee works in a specific department, and employee and department are two different entities. So, we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
  - We add the primary key of the DEPARTMENT table (Fig 1.6), Department_Id as a new attribute in the EMPLOYEE table.
  - Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.

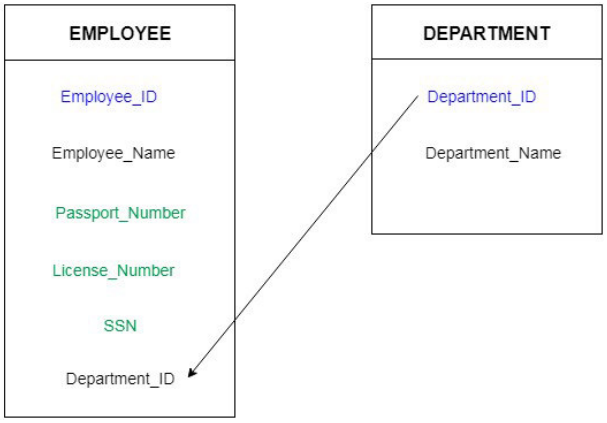


Fig 1.6 Illustration of Foreign Key

Moreover, the above main keys, we can have the following also:

Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independently or individually.

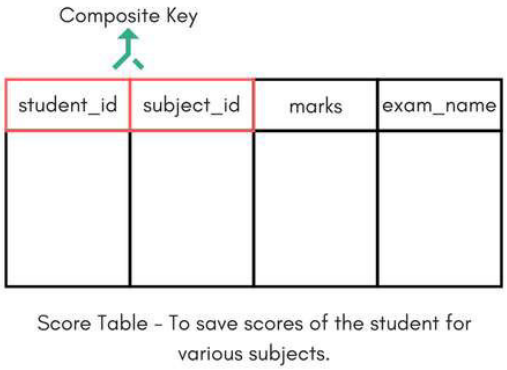


Fig 1.7 Illustration of Composite Key

In the above picture we have a Score table which stores the marks scored by a student in a particular subject. In this table, Fig 1.7, student_id and subject_id together will form the primary key and hence it is a composite key.

Secondary or Alternative Key

The candidate keys which are not selected as primary key are known as secondary keys or alternative keys.

Non-key Attributes

Non-key attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table.

Non-prime Attributes

Non-prime Attributes are attributes other than Primary Key attribute(s)**.**

1.6 RELATIONAL MODEL CONSTRAINTS

Constraintsenforcelimitstothedataorrestrictionsondatathatcanbeinserted/updated/deleted from a table. The whole purpose of constraints

Space for learners:

is to maintain the data integrity during an update/delete/insert into a table. Constraints on databases can generally be divided into three main categories:

1. Constraints those are inherent in the data model, we call these inherent model-based constraints or implicit constraints.

2. Constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL.

3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs. This is known as application-based or semantic constraints or business rules.

The schema-based constraints include: domain constraints, key constraints, entity integrity constraints, and referential integrity constraints.

Space for learners:

1.6.1 Domain Constraints

Eachtablehascertainsetofcolumnsandeachcolumnallowsthesametype ofdatabasedonitsdatatype. The column does not accept values of any other data type. Domain constraints can be defined as follows:

DomainConstraint=datatype+Constraints(NOTNULL/UNIQUE/ PRIMARYKEY /FOREIGNKEY / CHECK / DEFAULT).

Let us consider the following STUDENT relation (Table 1.3):

Table 1.3: STUDENT Relation

ROLL	NAME	AGE
1	Rahul Sarmah	23
1	Smriti Gogoi	22
3	Shahidul Khan	24
4	Rupak Chetri	22

Here, value **A** is not allowed since only integer values can be taken by the age attribute.

1.6.2 Key Constraints

Anattributethatcanuniquelyidentifyatupleinarelationiscalledthekeyofthetable.Allthevalues in the primary key column must be unique.

Table 1.4: STUDENT Relation

ROLL	NAME	AGE
1	Rahul Sarmah	23
2	Smriti Gogoi	22
3	Shahidul Khan	24
4	Rupak Chetri	A

This relation/table (Table 1.4) does not satisfy the key constraint as here all the values of primary key are not unique.

1.6.3 Entity IntegrityConstraint

Entity integrity constraint specifies that in a relation no attribute value of primary key attribute must contain a null value. This is because the presence of null value in the primary key violates the uniqueness property.

Table 1.5: STUDENT Relation

ROLL	NAME	AGE
1	Rahul Sarmah	23
2	Smriti Gogoi	22
3	Shahidul Khan	24
	Rupak Chetri	22

This relation (Table 1.5) does not satisfy the entity integrity constraint as here the primary key contains a null value.

Space for learners:

1.6.4 Referential Integrity Constraint

Referential Integrity constraints work on the concept of Foreign Keys. A foreign key is an attribute of a relation which must be a primary key or the part of the primary key in another relation.

A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies. R1 is called the referencing relation and R2 is the referenced relation.

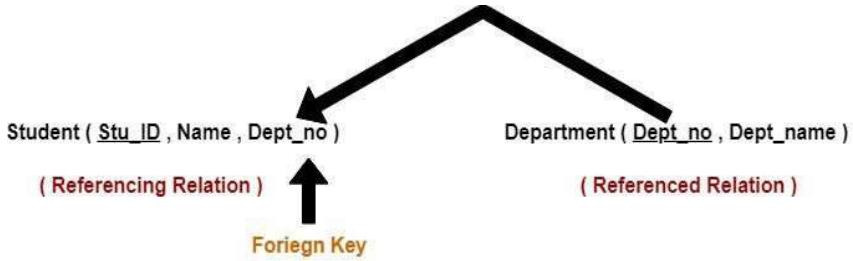


Fig 1.8: Illustration of the concept of Referential Integrity Constant

Table 1.6: STUDENT Relation

Roll	Name	Dept_No
1	Rahul Sarmah	D1
2	Smriti Gogoi	D1
3	Shahidul Khan	D2
4	Rupak Chetri	D5

Table 1.7: DEPARTMENT Relation

Dept_No	Dept_Name
D1	Mathematics
D2	Physics
D3	Chemistry
D5	Computer Science

Space for learners:

From the table 1.6 and 1.7 it is clear that,

- The relation 'Student' does not satisfy the referential integrity constraint.
- This is because in relation 'Department', no value of primary key specifies department no.14.
- Thus, referential integrity constraint is violated.

Space for learners:

1.6.5 Operations in Relational Model with Constraint Violations

Four basic operations performed on relational database model are insert, update, delete and select.

- **Insert Operation:** Insert operation is used to insert data into the relation. Insert can violate any of the four types of constraints mentioned above. Domain constraints can be violated when an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type. Key constraints can be violated when a key value in the new tuple "t" already exists in the relation "R". Entity integrity can be violated when any part of the primary key of the new tuple "t" is null. Referential integrity can be violated when the value of any foreign key in a tuple "t" refers to a tuple that does not exist in the referenced relation. If an insertion violates one or more constraints, the default option is to reject the insertion. If the insertion is not rejected then, the insertion violation can cause cascade in the relation. A foreign key with **cascade delete** means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a **cascade delete**.
- **Delete Operation:** This operation is used to delete tuples from the table (Table 1.6 & 1.7). The delete operation can violate only **referential integrity**. This occurs when the tuple being deleted is referenced by foreign keys from other tuples in the database. Here are some examples.

  Operation: Delete the Department tuple with Dept_No=1.
  Result: This deletion is acceptable and deletes exactly one

tuple.

Operation: Delete the Student tuple with Dept_No= 1. ***Result:*** This deletion is not acceptable, because there are tuples in Department those refer to this tuple.

DEPARTMENT Relation

Dno	Dname
1	Finance
2	Inventory
3	Admin
4	Marketing

EMPLOYEE Relation

Ssn	Name	Salary	Dno
121	Rahul Lahkar	22000	2
122	Ramesh Kalita	20000	4
123	Shamsaad Hasin	20000	1
124	Kunal Payeng	25000	3

Fig. 1.9: Relations (Department & Employee)

Several options are available if a deletion operation causes a violation. The first option, called restrict, is to reject the deletion. The second option, called cascade. A third option, called set null or set default, is to modify the referencing attribute values that cause the violation. The combinations of these three options are also possible.

- **Update Operation:** This operation is used to change/modify the values of the attributes in existing tuples. Consider two tables in figure-1.9, EMPLOYEE(Ssn, Name, Salary, Dno) and DEPARTMENT(Dno,Dname)

  Operation:Update the salary of the EMPLOYEE tuple with Ssn = '123' to 2800.***Result:*** Acceptable.

  Operation: Update the Dnoof the EMPLOYEE tuple with Ssn= '123' to7.***Result:*** Unacceptable, because it violates referentialintegrity.

  Operation: Update the Ssn of the EMPLOYEE tuple with

Space for learners:

Ssn = '123' to '321'.***Result:*** Unacceptable, because it violates primary key constraint

Updating an attribute that is neither part of a primary key nor of a foreign key usually causes no problems.

The Transaction Concept: A transaction is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema. A single transaction may involve any number of retrieval operations C and any number of update operations. For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.

CHECK YOUR PROGRESS-I

Multiple Choice Questions

1. What is the instance of a Database?
   a) The logical design of the database system
   b) The entire set of attributes of the database put together in a single relation
   c) The data or collection of information stored in a database at a particular moment of time.
   d) The initial values inserted into the database
2. An attribute is a _____ in a relation.
   a) Row
   b) Column
   c) Value
   d) Tuple
3. Constraints define a condition, which needs to be satisfied while storing data in a _____.
   a) Data
   b) Database
   c) Attribute
   d) Task

Space for learners:

4. An alternate key is a candidate key that is not the _____.
   a) Entity
   b) Attribute
   c) Secondary Key
   d) Primary Key
5. Advantages of relational model are
   a) Simplicity
   b) Data Integrity
   c) Flexibility
   d) All of the above

Fill in the Blanks

6. Primary key of a table never contains NULL and _____ values.
7. A _____ key allows us to identify uniquely an entity in the entity set.
8. What is the degree of a table with 1000 rows and 10 columns?
9. In a relational database a referential integrity constraint can be specified with the help of _____.
10. The format or data type must be specified for _____.

Space for learners:

1.7 SUMMING UP

- A Relation is a tabular structure defined by the heading and the data is entered in the body containing a set of rows.
- Domain is a set of possible values an attribute can acquire.
- A row of a relation in a relational data model that gives complete information of an entity is known as Tuple.
- Relational Schema can be defined as the description of the database that is specified during database design.
- An Attribute is the column header in a relation which is the properties of entity.
- The number of tuples in a relation is known as Cardinality.
- Foreign Key is an attribute of one relation R_2 whose values are required to match those of the primary key of some

relation R_1

- Relation that contains a foreign key in known as Referencing Relation.
- Super Key can be defined as the superset of primary key that can uniquely identify any data row in the table.
- Candidate Keys can be defined as the set of keys that is minimal and can uniquely identify any data row in the table.
- Constrains are used to enforce limits to the data or type of that data that can be inserted/updated/deleted from a table.
- E-R Diagram describes interrelated things of interest in a specific domain of knowledge.

Space for learners:

1.8 ANSWERS TO CHECK YOUR PROGRESS

1. (c)
2.(b)
3.(b)
4. (d)
5. (c)
6. Duplicate
7. Super
8. 10
9. Foreign key
10. Domain

1.9 POSSIBLE QUESTIONS

Short-Answer Questions

1. What are the different features of relational model?
2. What are the advantages of the relational model?
3. State the various features of relations.
4. Discuss the various components of domain information.
5. What is key? What is the importance of key in a relation?
6. With the help of example define a tuple in a relation.
7. What is constraint? Why they are important?
8. With the help of an example define a super key.
9. With the help of an example define foreign key.

10. With the help of an example define candidate key.

Long-Answer Questions

1. Discuss the relational model.
2. Explain the different types of constraints with example.
3. Explain the different types of keys with example.
4. What are domains? Explain its constraints.
5. Explain the different types of attributes with example.
6. Explain the different types of entity with example.
7. Discuss the advantages of relational model.
8. Discuss the tabular structure that is used to represent a relation in relational model.

Space for learners:

1.10 REFERENCES AND SUGGESTED READINGS

- Ramez, Elmasri. *Fundamentals of Database Systems*. Pearson Education India, 2020.
- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. McGraw-Hill, 1997.

UNIT-2: RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

Space for learners:

Unit Structure

2.1 INTRODUCTION

Relational algebra uses a procedural query language or formal query language. Relational algebra deals with the study of relational operations on single or multiple relations. After implementing a relation, it returns a new relation which can be again reuse in another relational operation. When we talk about of relational algebra it is having a fixed set of operation denoted by symbols.

Relational algebra is necessary as because-

i) it represents the extraction or retrieval of data easily and clearly, and

ii) from a relational algebra statement, a practical SQL notation can easily be derived.

2.2 UNIT OBJECTIVES

Space for learners:

After going through this unit, you will be able to:

- understand the fundamental concepts of relational algebra and relational calculus.
- know different relational algebra operations.
- know different types of relational calculus.

2.3 RELATIONAL ALGEBRA OPERATIONS

Relational algebra operations can be divided into two parts-

i) Relational-oriented operations

ii) Set-oriented operations

2.3.1 Relational-Oriented Operations

emp_no	emp_name	emp_age	designation	emp_address
1	Avik	35	Manager	Guwahati
2	Prakhya	50	Accountant	Jorhat
3	Rajib	45	Peon	Nalbari
4	Tridip	53	Peon	Golaghat

Table-1: EMPLOYEE Relation

2.3.1.1 SELECT (σ)

The SELECT operation retrieves specific rows from a relation or table. Sigma (**σ**) symbol is used to denote the SELECT operation.

The general syntax of select operation is **σ<selection_condition> (<realtion-name>)**

Where **σ** is used for select operation, **selection_condition** is nothing but a boolean expression which contains the specific attributes of relation and **relation-name** is the name of the table.

Considering table-1, to extract tuple having emp_name= “Avik”

from Employee table, the query can be written as follows:

$$\sigma_{emp\_name=\text{"Avik"}} \textbf{(Employee)}$$

We can also extract rows in which the age of the employees is more than 40 using the following query:

$$\sigma_{emp\_age>40} \textbf{(Employee)}$$

We can also use relational operators (=, ≠ ,<, >, ≤, ≥) as well as logical operators (∧, ∨, ¬).

Again, suppose we want to retrieve tuples which contain the emp_name "Prakhya" and emp_address "Guwahati"; to do so we can write the following query:

$$\sigma_{emp\_name=\text{"Prakhya"} \wedge emp\_address=\text{"Guwahati"}} \textbf{(Employee)}$$

2.3.1.2 PROJECT (π)

The project operation retrieves attributes or columns from a relation or table. Pi(**π**) symbol is used to denote the project operation. The general syntax of project operation is:

$$\pi_{<attribute\_list>} \textbf{(<relation-name>)}$$

Here, **π** is used for project operation, **attribute_list** is a list of specific attributes of relation and **relation-name** is nothing but a table name.

Now from table-1, Suppose we want to retrieve the emp_no and emp_name from Employee table, we can write the following query:

$$\pi_{emp\_no,emp\_name} \textbf{(Employee)}$$

Let us take another example. Suppose we want to retrieve the name and designation of those employees whose age are less than 50, then we can write the following query-

$$\pi_{emp\_name,designation} (\sigma_{age<50} \textbf{(Employee))}$$

2.3.1.3 JOIN

Join operations are used to combine two or more relations to form a single new relation. Join is a combination of Cartesian product and selection process. "⋈" symbol is used to denote the join.

i) Inner join- when we write an inner join in a query, inner join extract only those rows which are satisfy the

Space for learners:

matching criteria. There are three types of inner join- theta join, equi join and natural join.

- Theta join- The symbol "θ" is used to denote the join condition and "θ" can use any comparison operator(=, ≠ ,<, >, ≤, ≥). It is a general case of join. When we want to join two or more relation based on some condition, we use the theta join.

  Let us take an example-

  Example-1:

  Suppose we have two relations, called "P" and "Q".

P		
Emp_id	Emp_name	Age
10	Raj	40
20	Rajib	35
30	Aman	36

Q		
Emp_id	Dept_id	DOJ
10	111	10-10-2000
30	112	10-10-2017
40	113	01-02-2009

Then, we write the query using theta join

$$P \bowtie_{P.emp\_id<Q.emp\_id} Q$$

Table-2: The Resultant Table

P.Emp_id	Emp_name	Age	Q.Emp_id	Dept_id	DOJ
10	Raj	40	30	112	10-10-2017
20	Rajib	35	30	112	10-10-2017

If we use Cartesian product for the above query then the equivalent query is-

$$\sigma_{P.Emp\_id<Q.Eid}(PxQ)$$

- Equi join- Equi join is a special case of theta join. Equi join uses only equivalence(=) condition. If we write the

Space for learners:

following query we will get the same result as mentioned in table 1-

$$P \bowtie_{p.Emp\_id=Q.Emp\_id} Q$$

- Natural join- In natural join also equivalence(=) operator is used but the difference is that attributes appears only once in natural join. Using natural join we will get a new table that does not have any duplicate columns.
- Outer join- An outer join is a type of join that is used to show all tuples from one relation even when some of these are not found in second relation. There are three types of outer join-left outer join, right outer join and full outer join.

Consider the example 1 to describe the types of outer join-

Left outer join- $P \supset\!\bowtie Q$

Emp_id	Emp_name	Age	Dept_id	DOJ
10	Raj	40	111	10-10-2017
20	Rajib	35	NULL	NULL
30	Aman	36	112	01-02-2009

Right outer join- $P \bowtie\!\subset Q$

Emp_id	Emp_name	Age	Dept_id	DOJ
10	Raj	40	111	10-10-2017
20	Rajib	35	112	NULL
40	NULL	NULL	113	01-02-2009

- In left outer join, it contains all the tuples from left relation and only matching records from right relation.
- In right outer join, it contains all the tuples from right table and only matching tuples from left table.
- In full outer join, it contains all the tuples from both relation and the tuples of both relation which do not match the join condition, these attributes are made NULL.

Space for learners:

2.3.2 Set-Oriented Operations

2.3.2.1 SET-UNION

The symbol "∪" is used to denote the union operation. Suppose P and Q are two compatible relations. Now P∪Q denotes SET-UNION of P and Q, which is a relation that includes all rows that are either in P or in Q or in both P and Q. In set union, duplicate rows would be eliminated.

Example-2: P and Q are two compatible relations where P holding the details of department in which project P1 is assigned and Q holding the details of those departments in which project P2 is assigned.

P:

Dept_id	Dept_name
D1	Physics
D2	Chemistry
D3	Computer Science

Q:

Dept_id	Dept_name
D1	Physics
D3	Computer Science
D4	Electronics

The result of UNION operation is:

P∪Q:

Dept_id	Dept_name
D1	Physics
D2	Chemistry
D3	Computer Science
D4	Electronics

Space for learners:

2.3.2.2 SET-INTERSECTION

The symbol "∩" is used to denote SET INTERSECTION operation and the result contains all rows that are in both P and Q.

Considering the example-2 above, the result of P∩Q will be a relation which gives departments to those both P1 and P2 are assigned.

P∩Q:

Dept_id	Dept_name
D1	Physics
D3	Computer Science

2.3.2.3 SET-DIFFERENCE

P-Q is used to denote the set difference operation. It finds the rows those are in one table or relation but not in another table.

From example 2 , to select all those departments which are present in P but not in Q. The result is-

P-Q

Dept_id	Dept_name
D2	Chemistry

2.3.2.4 CARTESIAN PRODUCT (CROSS PRODUCT)

Cross product is denoted by P x Q and returns a table on rows whose schema contains all fields of P (in the same order, they appear in P) followed by all fields of Q (in the same order as they appear in Q).

Suppose, we have two relations P and Q-

Example 3:

P

Dept_id	Dept_name
D1	Physics
D2	Chemistry
D3	Computer science
D4	Electronics

Space for learners:

Q

Project No
P1
P2

The result of the operation PxQ is follows-

PxQ

Dept id	Dept name	Project No
D1	Physics	P1
D1	Physics	P2
D2	Chemistry	P1
D2	Chemistry	P2
D3	Computer science	P1
D3	Computer science	P2
D4	Electronics	P1
D4	Electronics	P2

2.4 RELATIONAL CALCULUS

Relational Calculus is a non procedural query language. In relational calculus, a query is formed as a formula consisting of a number of variables and an expression involving these variables. It uses mathematical predicate calculus. It tells what to do but never explain how to do. There is no such mechanism to evaluate the formula. DBMS decide how to transform such non procedural query language into equivalent and efficient procedural queries. There are two types of relational calculus. One is tuple relational calculus (TRC) and domain relational calculus (DRC).

2.4.1 Tuple Relational Calculus

It was proposed by E.F. Codd in the year 1972. A tuple calculus expression is essentially a non procedural definition of some relation

Space for learners:

in terms of some given set of relations. A query in tuple relational calculus is formed as: {t|cond(t)}, where t denotes a tuple variable and cond(t) denotes predicate or condition involving t. The result of the query is the set of all tuples "t" such that predicate p is true for t.

Example 4:

Consider the following relation-

DEPT (D_id, D_name D_location)

To find all department whose D_location are "Guwahati", we can write the following construct of the tuple relational calculus:

{t| DEPT(t) and t.D_location="Guwahati"}

The condition DEPT(t) specifies that the ranges relation of table variable t is department. Each department tuple "t" that specifies the condition t.D_location="Guwahati" will be retrieved. The above query retrieves all attributes values for each selected department tuple t. To retrieve only some of the attributes, we can write- {t.Dept_id, t.Dept_name|DEPT(t) and t.D_location="Delhi"}.

2.4.2 Domain Relational Calculus

In Domain Relational Calculus, variables uses the domain rather than relations. It uses the same operators as tuple calculus. It uses ∧ (and), ∨(or) and ¬(not) logical connectives. It also uses Existential (∃) and Universal Quantifiers (∀).

Syntax: {x1, x2, x3, ….. xn|p(x1,x2,…..xn)} where x1,x2,…xn are attributes and p is the formula which is formed by inner attributes.

For example: {<emp_id, emp_name, dept_name>|∃ EMP∧dept_name="sales"}

This query will retrieve the emp_id, emp_name and dept_name from the relation EMP where dept_name is sales.

2.5 EXAMPLES (RELATIONAL ALGEBRA)

Consider the following table structure:

EMPLOYEE (E_id, E_name, DOB, D_id)

DEPARTMENT (D_id, D_name, D_loc)

Space for learners:

PROJECT (P_id, P_name, D_id)

WORKS_ON (E_id, P_id, Hours)

Query1: Get names of the employee who worked in department "D1".

Answer: $\pi_{E\_name}(\sigma_{D\_id="D1"}$ **(EMPLOYEE⋈DEPARTMENT))**

Query2: Get all the information about employees whose date of birth are before 01-01-1990.

Answer: $\sigma_{DOB<"01\text{-}01\text{-}1990"}$**(EMPLOYEE)**

Query3: Print the D_id and D_name of those departments which are located in guwahati.

Answer: $\pi_{D\_id,D\text{-}name}(\sigma_{D\_loc="guwahati"}$**(DEPARTMENT))**

Query4: Get details of employees working on project P1.

Answer: EMPLOYEE⋈ $\pi_{E\_id}(\sigma_{P\_id="P1"}$**(WORKS_ON))**

Query 5: Get details of employees working on DBMS project.

Answer: EMPLOYEE⋈ π_{E_id}**(WORKS_ON)** ⋈ **($\pi_{P\_id}(\sigma_{P\_name="DBMS"}$(PROJECT)))**

CHECK YOUR PROGRESS

Fill-in the blanks:

1. _____ operation retrieves specific rows from a relation or table.
2. _____ operation retrieves attributes or columns from a relation or table.
3. _____ operations are used to combine two or more relations to form a single new relation.
4. P∪Q denotes _____ of P and Q.

Answer the following:

6. Write down the syntax for Select operation.
7. Write down the syntax for Project operation.

Space for learners:

2.6 SUMMING UP

- Relational algebra operations can be divided into two parts namely Relational-oriented operations and Set-oriented operations.
- The Select operation retrieves specific rows from a relation or table. Sigma (σ) symbol is used to denote the SELECT operation.
- The Project operation retrieves attributes or columns from a relation or table.
- Join operations are used to combine two or more relations to form a single new relation.
- If P and Q are two compatible relations, then P∪Q denotes SET-UNION of P and Q, which is a relation that includes all rows that are either in P or in Q or in both P and Q.
- SET INTERSECTION operation between P and Q will result a relation which contains all rows that are in both P and Q.
- Relational Calculus is a non-procedural query language.
- A tuple calculus expression is essentially a non-procedural definition of some relation in terms of some given set of relations. A query in tuple relational calculus is formed as: {t | cond(t)}, where t denotes a tuple variable and cond(t) denotes predicate or condition involving t.

2.7 ANSWERS TO CHECK YOUR PROGRESS

1. Select, 2. Project, 3. Join, 4. SET-UNION

5. $\sigma_{<selection\_condition>}(<realtion\text{-}name>)$

6. $\pi_{<attribute\_list>}$ (<relation-name>)

2.8 POSSIBLE QUESTIONS

Short Answer type questions:

1. What is Relational algebra?
2. What is the purpose of Relational algebra?
3. Discuss the selection operation.

Space for learners:

4. Discuss the project operation.
5. Discuss the theta join.
6. What do you mean by Equi join?
7. What do you mean Relational calculus?
8. What do you mean by Tuple relational calculus?
9. What do you mean by Domain relational calculus?
10. Discuss the Cartesian product in Relational algebra.

Long answer type questions:

11. Explain selection and projection with examples.
12. Explain different types of Outer join with examples in Relational algebra.
13. Explain different types of set-oriented operations with examples in relational algebra.
14. Explain the different types of relational calculus with examples.

Space for learners:

2.9 REFERENCES AND SUGGESTED READINGS

- Ramez, Elmasri. *Fundamentals of Database Systems*. Pearson Education India, 2020.
- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. McGraw-Hill, 1997.

UNIT 3: STRUCTURED QUERY LANGUAGE - 1

Space for learners:

Unit Structure:

3.1 INTRODUCTION

SQL stands for Structured Query Language. It is used for storing, manipulating and retrieving data stored in a relational database. With the help of SQL, within a few microseconds, the records in a table(s)/relation(s) can be searched, retrieved and manipulated. Various RDBMS are available to work with SQL. Some of the popular RDBMS are: MySQL, PostgreSQL (both are free and open source), Oracle, SQL Server etc. MySQL is the most popular open-source Database Management System and now it is distributed and supported by Oracle Corporation. In this unit, we will discuss the SQL commands using MySQL.

3.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- define SQL
- understand the installation and working process of MySQL in Windows Environment
- use various DDL commands
- use various DML commands

3.3 MY SQL INSTALLATION IN WINDOWS

Space for learners:

The latest version of MySQL can be download from the link: https://dev.mysql.com/downloads/installer/ as shown below (Fig. 3.1):

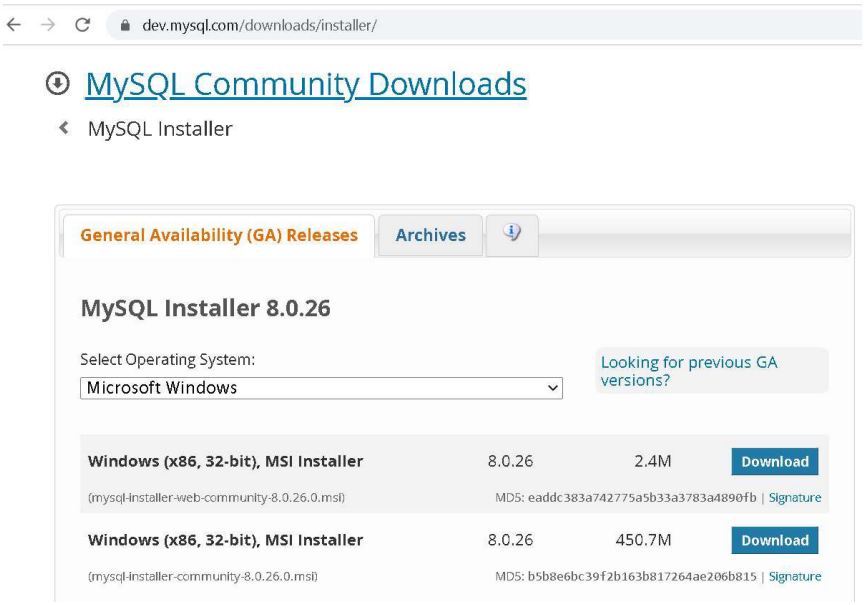


Fig. 3.1

After downloading the Installer, simply double click over the file and the following window will appear (Fig. 3.2).

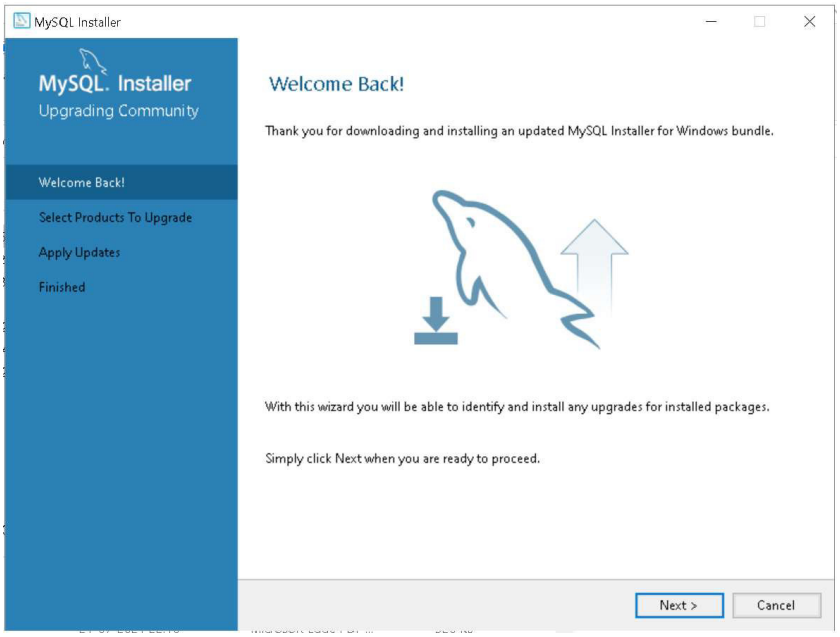


Fig. 3.2

Click the **Next** button. The following window will appear (Fig. 3.3):

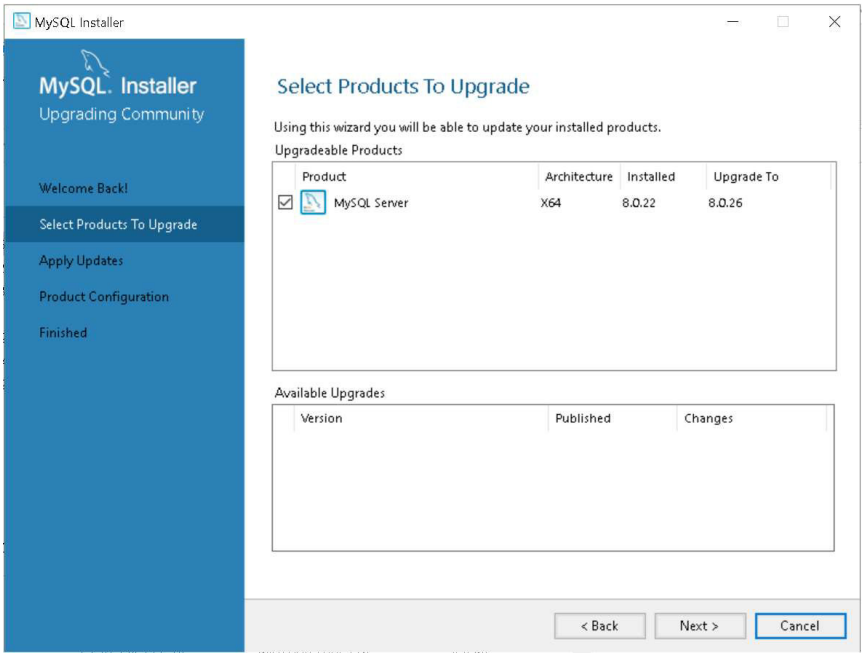


Fig. 3.3

Click the **Next** button. You may see **Upgrade Now** option in a window. If shown, click on it. The following window will appear(Fig. 3.4):

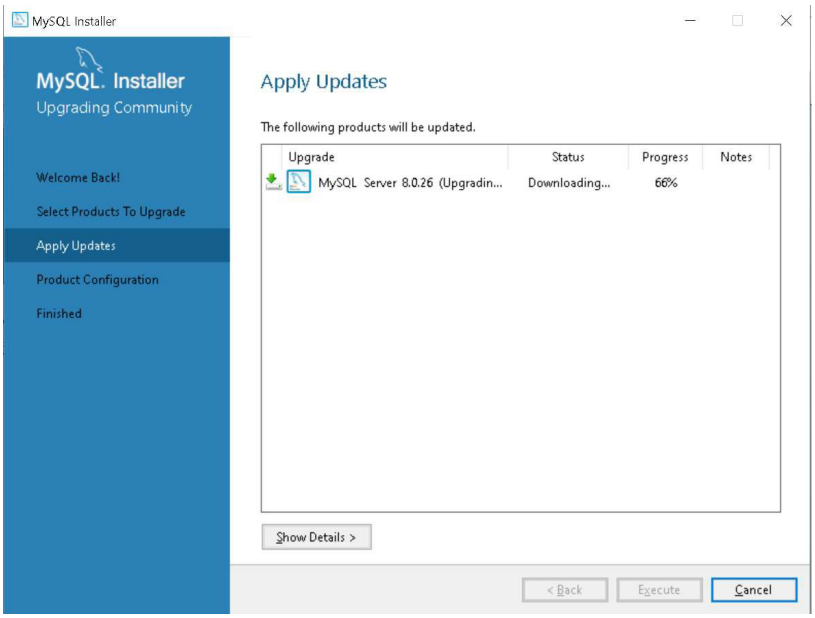


Fig. 3.4

After upgradation completed, the following window (Fig. 3.5) will appear:

Space for learners:

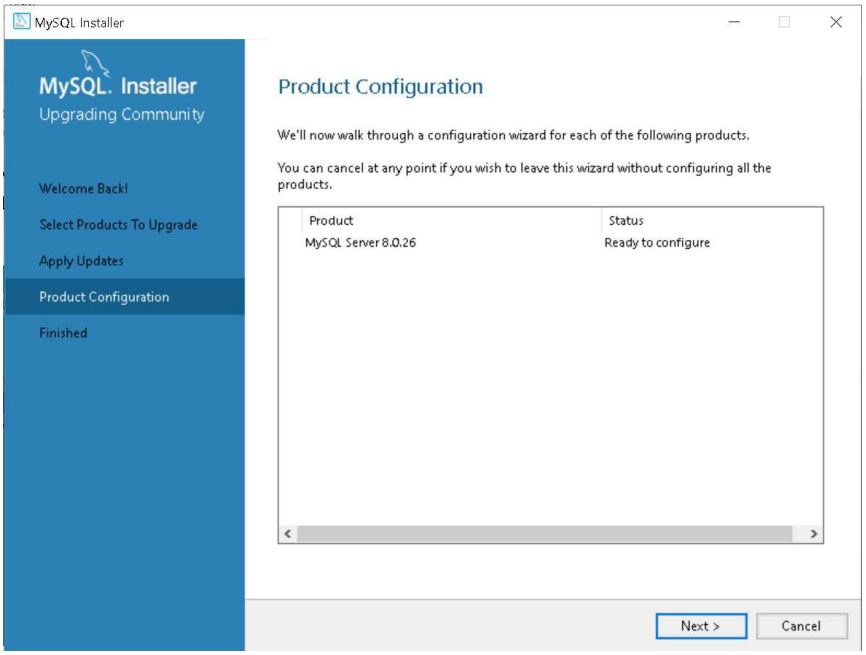


Fig. 3.5

Now, you can skip the process of product configuration process. So, you can click the **Cancel** button.

MySQL is now installed in your computer.

STOP TO CONSIDER

For MySQL 8.0 on Windows, the default installation directory is **C:\Program Files\MySQL\MySQL Server 8.0** for installations performed with MySQL Installer.

You can now check whether MySQL server is working or not. For that, open the DOS prompt as shown (Fig. 3.6) below:

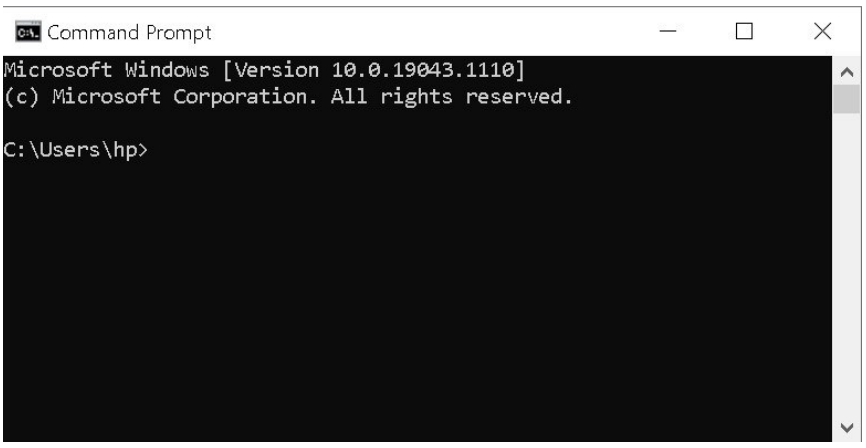


Fig. 3.6

Space for learners:

Type **cd** to go to the drive **C** as shown (Fig. 3.7) below:

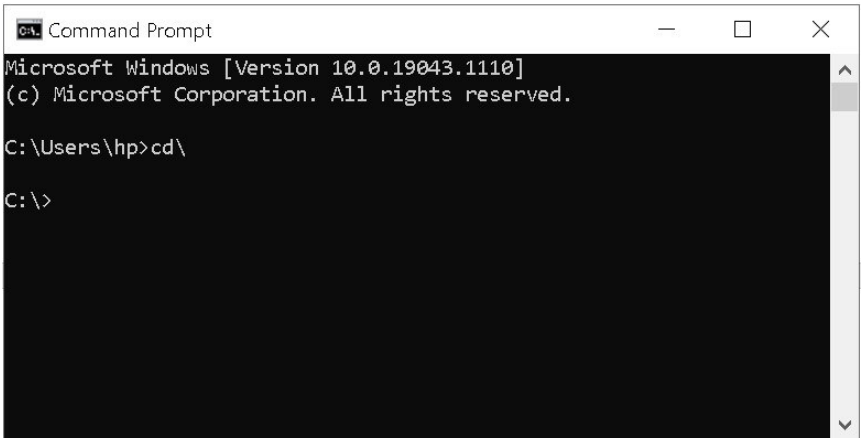


Fig. 3.7

Then go to the default installation directory as shown (Fig. 3.8) below:

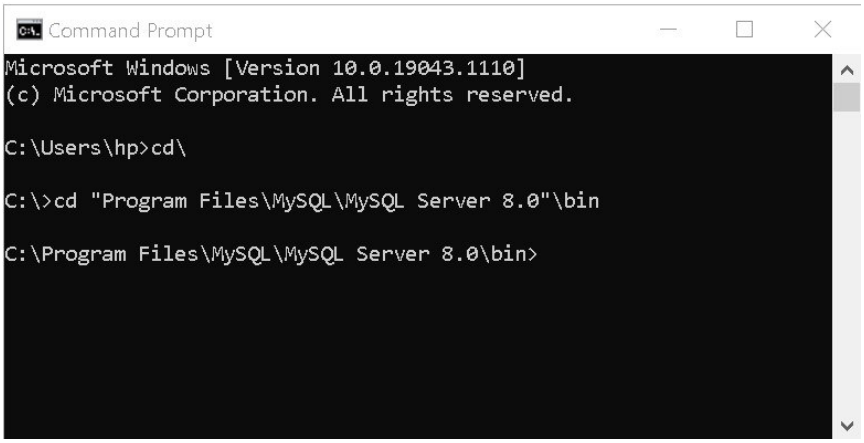


Fig. 3.8

Then type the following command

mysql -uroot -proot

Here,
-usignifies that the word follows is the User Name which is **root**,
-psignifies that the word follows is the Password which is **root**.

The mysql prompt will be displayed as shown (Fig. 3.9) below:

Space for learners:

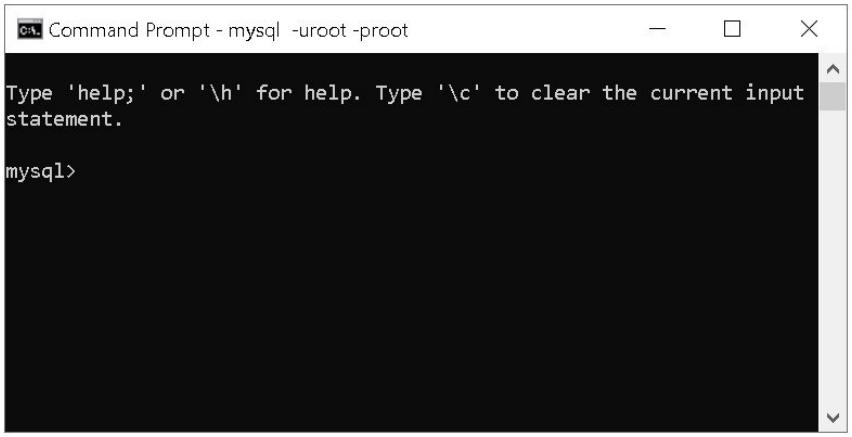


Fig. 3.9

To exit from mysql server, type **\q**

3.4 TYPES OF SQL COMMANDS

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL. In this unit, we will discuss about the DDL and DML commands.

3.4.1 Data Definition Language (DDL)

Data Definition Language (DDL) commands are used to define the structure of the database or the database schema. The changes made in the database using DDL commands are saved permanently. Following SQL commands falls under the DDL:

(a)CREATE: It is used to create a new database, table, views and index.Suppose, we want to create a database named "GUIDOL". The SQL statement for creating a database is as follows:

CREATE DATABASE database_name;

Let us execute the above statement using MySQL (Fig. 3.10):

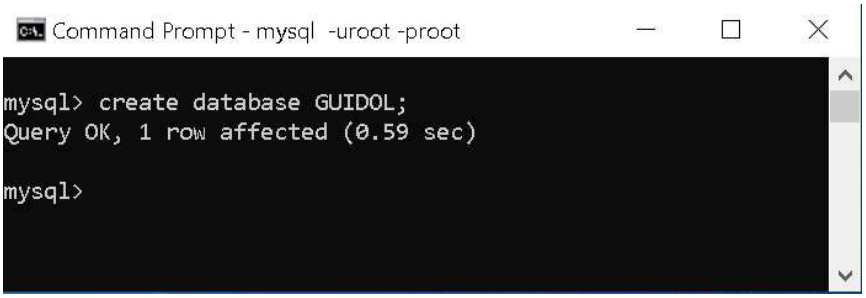


Fig. 3.10

Space for learners:

Let's now create a table under the database GUIDOL. For this, first we have to use this database using the following SQL statement (Fig. 3.11) as:

USE database_name;

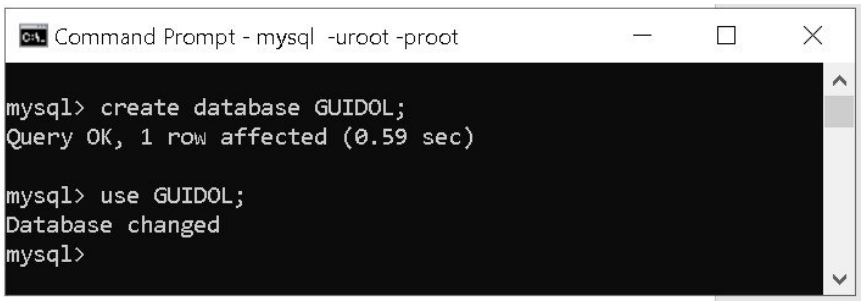


Fig. 3.11

Now, let us create a table named **Student**under the database "**GUIDOL**". Suppose the attributes or field names of the table are rollno, name, date_of_birth, vill_town, city, mobile. The SQL statement will be:

create table student(rollno varchar(10) primary key, name varchar(30), date_of_birth date,vill_town varchar(20), city varchar(20), mobile varchar(10));

If you execute the SQL statement in mysql, it will be shown (Fig. 3.12) below:

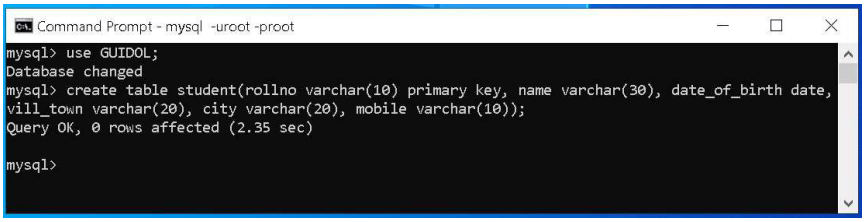


Fig. 3.12

(b) ALTER: It is used to change the structure of an already existing table of a database.

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table.

(i) To add a column in a table, use the following syntax:

ALTER TABLE *table_name*ADD *column_namedatatype*;

Suppose, We want to add a column name email with data types varchar(20) to the existing table student. The SQL statement will be (Fig. 3.13):

Space for learners:

Space for learners:

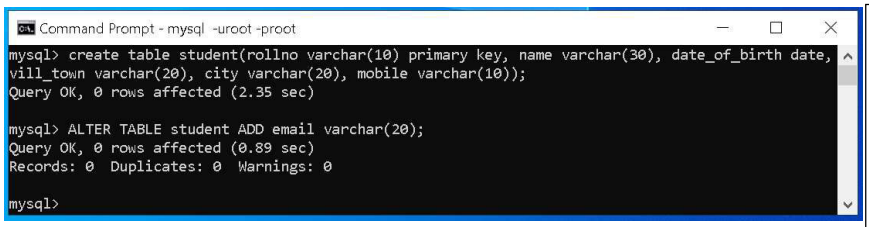


Fig. 3.13

To check the structure of the table, use the following SQLstatement (Fig. 3.14):

DESC table_name;

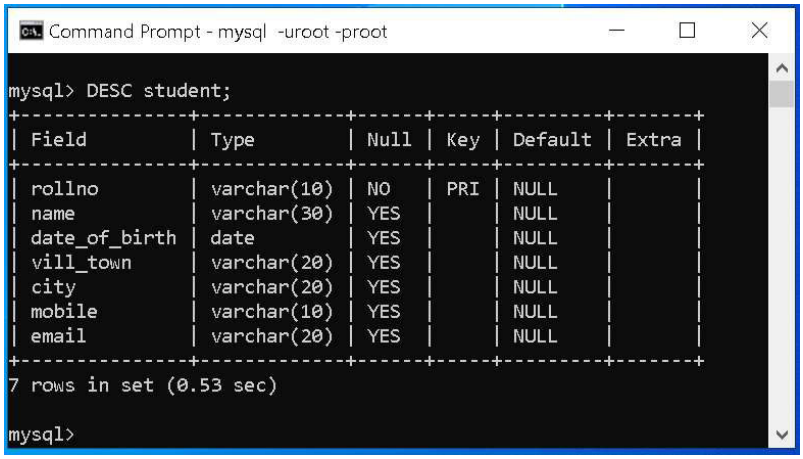


Fig. 3.14

(ii) To delete a column in a table, use the following syntax:

ALTER TABLE *table_name* DROP COLUMN *column_name*;

Let us remove the column *email* that is just created (Fig. 3.15).

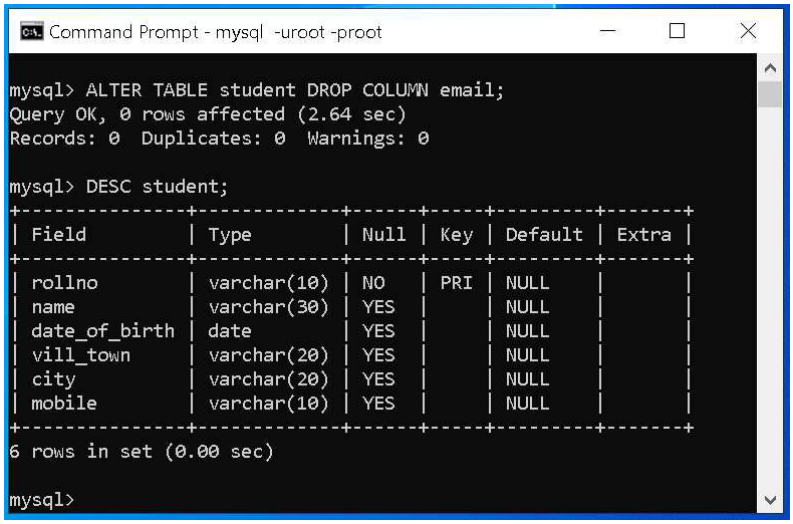


Fig. 3.15

(iii) To change the data type of a column in a table, use the following syntax:

ALTER TABLE *table_name* MODIFY COLUMN *column_name datatype*;

Let us change the data type of the column city to varchar(15) (Fig. 3.16).

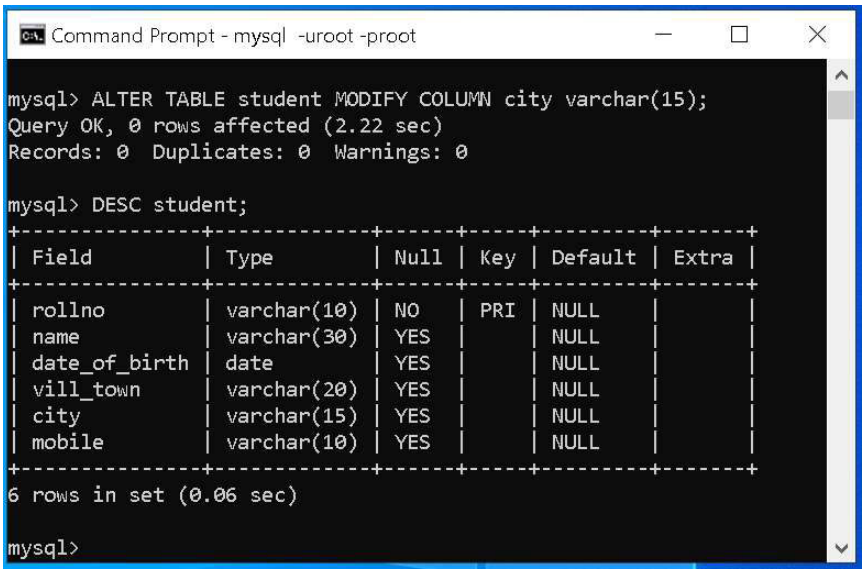


Fig. 3.16

(c) DROP: It is used to delete a database, table, views or index.

The following SQL statement drops the existing database "GUIDOL":

DROP DATABASE GUIDOL;

The DROP TABLE statement is used to drop an existing table in a database.For example,

DROP TABLE student;

Note: Do not try to execute these statements unnecessarily as it will delete the existing database and existing table and all information will be lost.

(d) TRUNCATE: It is used to completely remove all data from a table, including their structure and space allocates on the server.

For Example, TRUNCATE TABLE student;

(e) RENAME: It is used to rename a database table.

The syntax is:

Space for learners:

RENAME TABLE existing_table_name TO new_table_name;

For example, the table student will be renamed as student_master (Fig. 3.17).

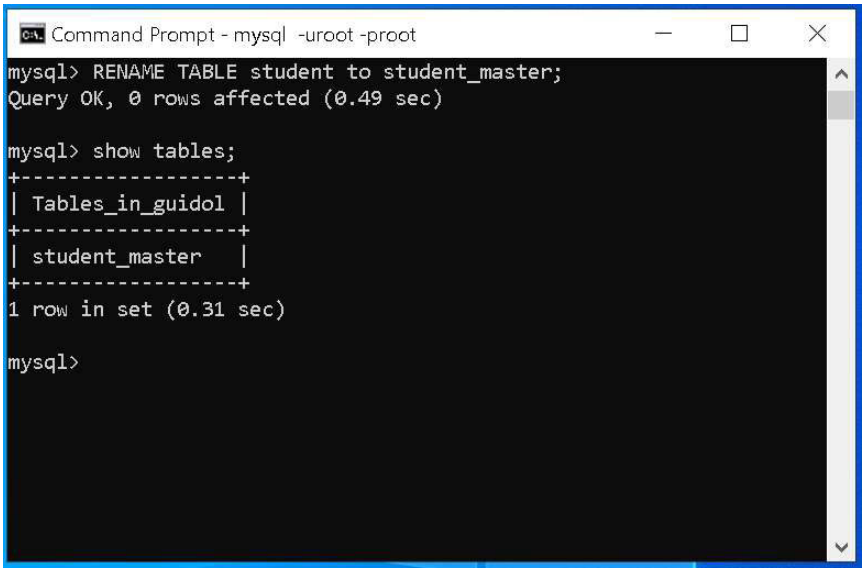


Fig. 3.17

Here, SHOW TABLES statement shows the available tables under the current database.

CHECK YOUR PROGRESS-I

1. Fill in the blanks:
   (a) _____ command is used to create a database.
   (b) _____ command is used to create a table.
   (c) To change the structure of a table, _____ command is used.
   (d) _____ command completely remove all data from a table, including their structure and space allocates on the server.
   (e) To change the name of a table _____ command is used.

3.4.2 Data Manipulation Language (DML)

DML stands for Data Manipulation Language. It includes various types of data manipulation SQL statements. DML statements are used to store, modify, retrieve, delete and update data in a database. Followings are some DML statements:

Space for learners:

(a)**INSERT:** It is used to insert data into a table. The syntax is:

INSERT INTO table_name (column 1, column 2,.....column n) VALUES (value 1, value 2, ……value n);

Or

INSERT INTO table_name VALUES (value 1, value 2, ……value n);

For example, to insert data into the table student_master, the SQL statement will be (Fig. 3.18):

INSERT INTO student_master (rollno, name, date_of_birth, vill_town, city, mobile) VALUES ('2100000001', 'Rajib Sarma', '1990-12-01', 'Ganeshguri', 'Guwahati', '98540XXXXX');

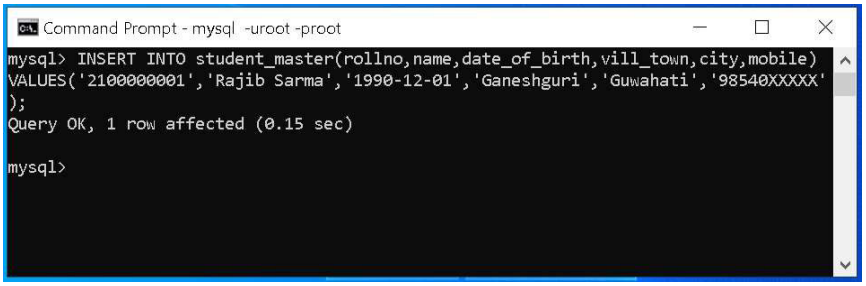


Fig. 3.18

MySQL Insert Multiple Rows:

To insert multiple rows into a table, you use the following form of the INSERT statement:

INSERT INTO table_name (column_list) VALUES (value_list_1), (value_list_2), ...(value_list_n);

Example (Fig. 3.19)**:**

INSERT INTO student_master (rollno, name, date_of_birth, vill_town, city, mobile) VALUES ('2100000002', 'Jyoti Saikia', '1990-01-02', 'Haripur', 'Pathsala', '98541XXXXX'), ('2100000003', 'Sanjib Kalita', '1988-06-07', 'Belsor', 'Nalbari', '98640XXXXX');

Space for learners:

Fig. 3.19

Suppose, we want to add another two records into the table student_master. The SQL statement will be (Fig. 3.20):

INSERT INTO student_master (rollno, name, date_of_birth, vill_town, city, mobile) VALUES ('2100000004', 'Jiten Kalita', '1990-07-02', 'Chanmari', 'Guwahati', '98541XXXXX'), ('2100000005', 'Jilmil Choudhury', '1987-09-07', 'Belsor', 'Nalbari', '98640XXXXX');

Fig. 3.20

(b) SELECT: It is used to display records from a table.

The syntax is:

SELECT column1, column2, ...FROM table_name;

If you want to select all the fields available in the table, the syntax will be:

SELECT * FROM table_name;

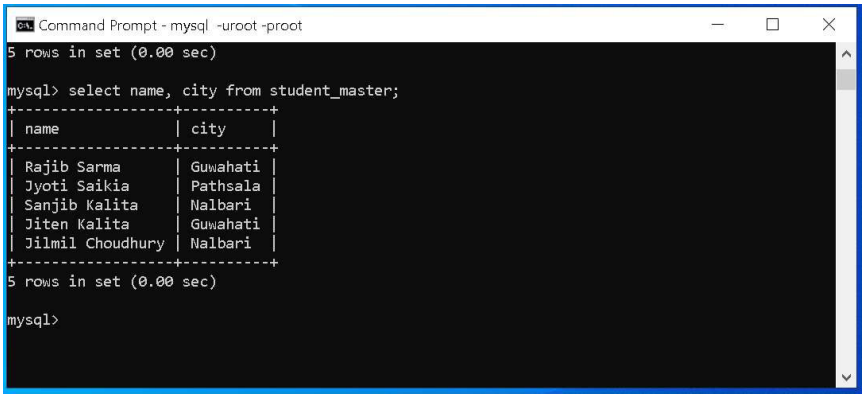
Space for learners:

Example: Display all the records from the table student_master;

SELECT * FROM student_master;

Example: Display the name and city of the students.

SELECT name, city FROM student_master;



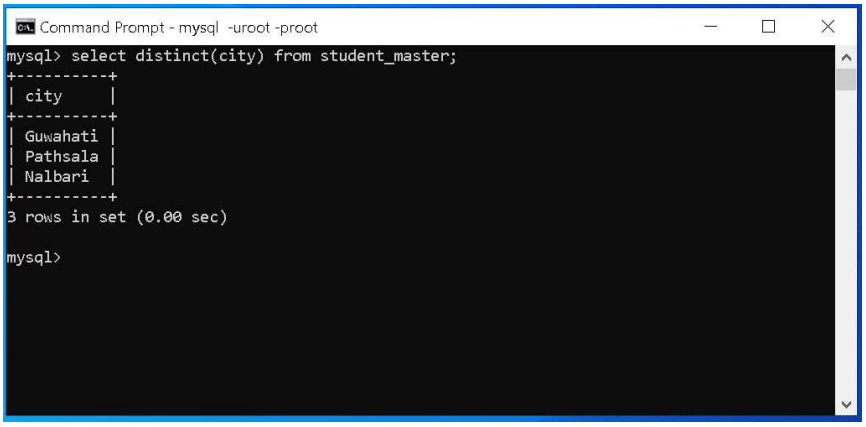
SELECT DISTINCT: It is used to display only the distinct values.

For example, the following SQL statement will display only the distinct values from the column city.

SELECT DISTINCT(city) FROM student_master;

Space for learners:

Space for learners:

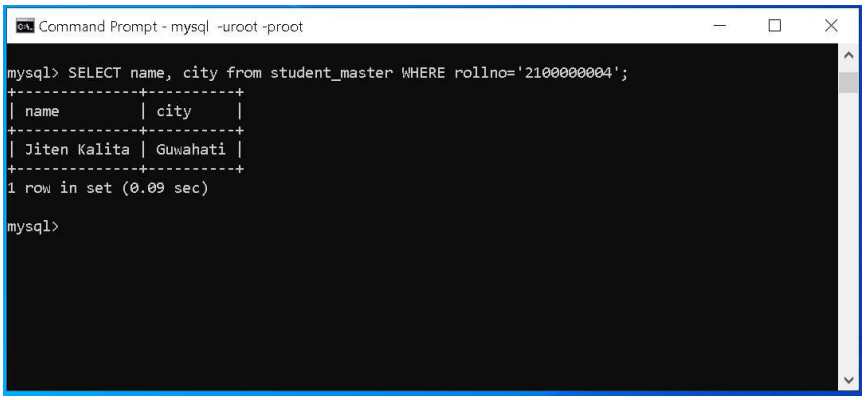


WHERE clause:

In MySQL, WHERE is a keyword used for the criteria or conditions to be applied for filtering the rows from a table or database. The WHERE clause can be used with INSERT, UPDATE, SELECT and DELETE statements to filter records and perform various operations on the data.

Example: Display the name of the student along with the city name whose roll number is 2100000004.

SELECT name, city from student_master WHERE rollno=’2100000004’;



LIKE Operator:

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. Here, two wildcards are used in conjunction with the LIKE operator.

- The percent sign (%) represents zero, one, or multiple characters.
- The underscore sign (_) represents one, single character.

Example:

(i) Display the student information whose name starts with the letter J.

SELECT * FROM student_master where name LIKE 'J%';

(ii) Display the student information whose name ends with the letter a.

SELECT * FROM student_master where name LIKE'%a';

(iii) Display the student information whose name contains'it'

SELECT * FROM student_master where name LIKE '%it%';

(iv) Display the student information whose name contains the letter 'y' in the second position.

SELECT * FROM student_master where name LIKE '_y%';

(v) Display the student information whose name starts with the letter 'S' and ends with the letter 'a'

SELECT * FROM student_master where name LIKE 'S%a';

AND, OR and NOT Operations**:**

AND, OR and NOT operators can be combined with the WHERE clause.

Example:

(i) Display the student information whose name starts with the letter J and address is Guwahati

SELECT * FROM student_master where name LIKE 'J%' AND address='Guwahati';

(ii) Display the student information whose name starts with the letter J and address is either Guwahati or Nalbari.

Space for learners:

SELECT * FROM student_master where name LIKE ‘J%’ AND (address=‘Guwahati’ OR address=‘Nalbari’);

(iii) Display the student information who resides except Guwahati;

SELECT * FROM student_master where NOT address = ‘Guwahati’;

ORDER BY Keyword**:**

It is used to sort the records in ascending or descending order. By default, it sorts the records in ascending order. To sort the records in descending order, the DESC keyword is used.

Example:

(i) SELECT * FROM student_master ORDER BY name;
(ii) SELECT * FROM student_master ORDER BY name DESC;

BETWEEN Operator:

It selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive i.e. begin and end values are included.

(c) UPDATE:

The update command is used to update existing data in a table.

Example: Change the name of the student to Rajib Saikia whose Roll Number is 21000000001.

UPDATE student_master SET name=‘Rajib Saikia’ where rollno = ‘21000000001’;

(d) DELETE

It is used to delete records from a table according to a given condition.

Space for learners:

Example: Delete the student record whose roll number is 21000000002.

DELETE from student_master WHERE rollno='21000000001';

JOIN Clause:

The Join command is used to combine rows from multiple tables in a database. Join operation between multiple tables is done with a common field with same attribute in the tables.

Let us consider the following two tables:

hostel(hostel_id, hostel_name, type, seat_capacity)

student_master(rollno, name, address, date_of_birth, sex, mobile, hostel_id)

Here, in the hostel table hostel_id is the primary key and in the student_master table rollno is the primary key. hostel_id field is common in both the tables. So, to join the two tables we will use the hostel_id field.

Example: Write SQL statement to display the name and seat capacity of the hostel for the student whose roll number is '21000000001'.

SELECT hostel.hostel_name, hostel.seat_capacity FROM hostel, student_master WHERE hostel.hostel_id=student_master.hostel_id and student_master.rollno='21000000001';

CHECK YOUR PROGRESS-II

2. Fill in the blanks:

   (a) To insert data into a table, _____ command is used.
   (b) _____ command is used to display data from a table.
   (c) _____ is a keyword used for the criteria or conditions to be applied for filtering the rows from a table or database.

Space for learners:

(d) The _____ operator is used in a WHERE clause to search for a specified pattern in a column.

(e) The _____ command is used to update existing data in a table.

Space for learners:

3.5 SUMMING UP

- SQL stands for Structured Query Language
- DDL stands for Data Definition Language
- Create, Drop, Alter, Truncate and Rename are DDL commands.
- DML stands for Data Manipulation Language
- Insert, Select, Update are DML commands.
- To create a new database, CREATE DATABASE command is used
- Before creating tables in a database, we have to use the database.
- CREATE TABLE command is used to create a new table.
- INSERT INTO command is used to insert records into a table.
- SELECT command is used to display records from a table.
- ORDER BY keyword is used to sort the records in ascending or descending order.
- The update command is used to update existing data in a table.
- The JOIN command is used to combine rows from multiple tables in a database.

3.6 ANSWERS TO CHECK YOUR PROGRESS

1. (a) CREATE DATABSE
   (b) CREATE TABLE
   (c) ALTER TABLE
   (d) TRUNCATE TABLE
   (e) RENAME TABLE

2. (a) INERT INTO
   (b) SELECT FROM
   (c) WHERE
   (d) LIKE

(e) UPDATE

Space for learners:

3.7 POSSIBLE QUESTIONS

Short answer type questions:

1. Define SQL. What are the uses of SQL?
2. Differentiate between DDL and DML statements.
3. In SQL, how a new table can be created? Explain with an example.
4. How multiple records can be inserted into a table? Explain with an example.
5. How will you display records from a table with certain conditions? Explain with examples.
6. Explain the uses of ALTER TABLE command with an example.
7. How records can be displayed in sorted order? Explain with an example.

Long answer type questions:

1. What are Data Definition Language statements? Explain with examples.
2. What are Data Manipulation Language statements? Explain with examples.
3. How will you insert a new field in an existing table? After insertion, how the data in the newly created field will be updated? Explain with an example.
4. What is LIKE operator? Explain its uses with examples.

3.8 REFERENCES AND SUGGESTED READINGS

1. https://www.w3schools.com/sql/default.asp

UNIT 4: STRUCTURED QUERY LANGUAGE - II

Unit Structure:

4.1 INTRODUCTION

The full form of SQL is Structured Query Language. It is used for storing, manipulating and retrieving data stored in a relational database. With the help of SQL, within a few microseconds, the records in a table(s) can be searched, retrieved and manipulated. Various types of RDBMS tools are available to work with SQL. Some of them are: MySQL, PostgreSQL (both are free and open source), Oracle, SQL Server etc. MySQL is the most popular open source database management system and now it is distributed, and supported by Oracle Corporation. In this unit, we will discuss the SQL commands using MySQL.

Space for learners:

4.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- define SQL
- understand the installation and working process of MySQL in Windows Environment
- describe the use of various DDL commands
- describe the use of various DML commands

Space for learners:

4.3 DATA CONTROL LANGUAGES

There are two types of data control languages: (a) grant and (b) revoke. Let us discuss with examples.

4.3.1 GRANT

It is employed to grant a privilege to a user. GRANT command allows specified users to perform specified tasks.

Syntax:

GRANT privilege_name on object name to user; Here,

- privilege names are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL
- object name is table name
- user is the name of the user to whom we grant privileges

4.3.2 REVOKE

It is employed to remove a privilege from a user. REVOKE helps the owner to cancel previously granted permissions.

Syntax:

REVOKE privilege_name on object name from user;

Here,

- privilege names are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL
- object name is table name
- user is the name of the user whose privileges are removing

Examples:

GRANT SELECT, UPDATE ON employees TO Bhanu

Explanation − Firstly, to give the permissions to user, we have to use GRANT command. The privileges are SELECT because to view the records and UPDATE to modify the records. The object name is table name which is Employee. The user name is bhanu.

REVOKE SELECT, UPDATE ON employees TO Bhanu

Explanation − Firstly, to revoke the permissions to user, we have to use REVOKE command. The privileges Need to revoke are SELECT because to view the records and UPDATE to modify the records. The object name is table name which is Employee. The user name is Bhanu.

STOP TO CONSIDER

For MySQL 8.0 on Windows, the default installation directory is **C:\Program Files\MySQL\MySQL Server 8.0** for installations performed with MySQL Installer.

4.4 AGGREGATE FUNCTIONS

SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value. It is also used to summarize the data. There are five types of aggregate functions: (a) avg (b) count (c) sum (d) max (e) min

4.4.1 AVG

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax:

AVG()

or

AVG([ALL|DISTINCT] expression)

Example:

SELECT AVG(COST)

FROM PRODUCT_MAST;

Space for learners:

Output:

67.00

4.4.2 COUNT

COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.

COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Examples:

SELECT COUNT(*)FROM PRODUCT_MAST;

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Output:

10

Example:

Space for learners:

SELECT COUNT(*) FROM PRODUCT_MAST WHERE RATE>=20;

Output:

7

Example: COUNT() with DISTINCT

SELECT COUNT(DISTINCT COMPANY) FROM PRODUCT_MAST;

Output:

3

Example:

SELECT COMPANY, COUNT(*) FROM PRODUCT_MAST GROUP BY COMPANY;

Output:

Com1 5

Com2 3

Com3 2

Example: COUNT() with HAVING

SELECT COMPANY, COUNT(*)FROM PRODUCT_MASTGROUP BY COMPANY

HAVING COUNT(*)>2;

Output:

Com1 5

Com2 3

4.4.3 MIN

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

Syntax

MIN()

or

MIN([ALL|DISTINCT] expression)

Space for learners:

Example:

SELECT MIN(RATE) FROM PRODUCT_MAST;

Output:

10

4.4.4 MAX

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

MAX()

or

MAX([ALL|DISTINCT] expression)

Example:

SELECT MAX(RATE)

FROM PRODUCT_MAST;

Output:

30

4.4.5 SUM

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

SUM()

or

SUM([ALL|DISTINCT] expression)

Example: SUM()

SELECT SUM(COST) FROM PRODUCT_MAST;

Output:

670

Space for learners:

CHECK YOUR PROGRESS

1. Fill in the blanks:

(a) _____ command is used to create a database.

(b) _____ command is used to create a table.

(c) To change the structure of a table, _____ command is used.

(d) _____ command completely remove all data from a table, including their structure and space allocates on the server.

(e) To change the name of a table _____ command is used.

Space for learners:

4.5 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.

Syntax

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
ORDER BY *column_name(s);*

4.6 HAVING CLAUSE

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

Syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*

GROUP BY *column_name(s)*
HAVING *condition*
ORDER BY *column_name(s);*

Space for learners:

4.7 SUMMING UP

- Various types of RDBMS tools are available to work with SQL. Some of them are: MySQL, PostgreSQL (both are free and open source), Oracle, SQL Server etc.
- MySQL is the most popular open source database management system and now it is distributed, and supported by Oracle Corporation.
- GRANT command grants permissions to users on database objects. It can also be used to assign access rights to users. For every user, the permissions need to be specified.
- It is used to remove the privileges on user accounts for access to a database object. It revokes permission granted to a user on a database object and also revokes the access rights assigned to users.

4.8 ANSWERS TO CHECK YOUR PROGRESS

1.

(a) CREATE DATABSE
(b) CREATE TABLE
(c) ALTER TABLE
(d) TRUNCATE TABLE
(e) RENAME TABLE

4.9 POSSIBLE QUESTIONS

Short Answer Type Questions:

1. Define SQL. What are the uses of SQL?
2. Differentiate between DDL and DML statements.
3. In SQL, how a new table can be created? Explain with an example.

4. How multiple records can be inserted into a table? Explain with an example.
5. How will you display records from a table with certain conditions? Explain with examples.
6. Explain the uses of ALTER TABLE command with an example.
7. How records can be displayed in sorted order? Explain with an example.

Long Answer Type Questions:

1. What are Data Definition Language statements? Explain with examples.
2. What are Data Manipulation Language statements? Explain with examples.
3. How will you insert a new field in an existing table? After insertion, how the data in the newly created field will be updated? Explain with an example.
4. What is LIKE operator? Explain its uses with examples.

Space for learners:

4.10 REFERENCES AND SUGGESTED READINGS

1. https://www.w3schools.com/sql/default.asp

UNIT-5: SEMANTIC MODELING

Unit Structure:

5.1 INTRODUCTION

In DBMS, Entity Relationship (ER) model is one of the important topics. In 1970 relational databases were introduced. Whenever we want to develop a software, DBMS plays an important role. Without a database we cannot build proper software. In Relational Database Management System, first of all we have to develop a design using an ER model. And then we convert this developed model into relations/tables so that we design a database with required and necessary properties. In an ER diagram, there are different components which help us to know the relationship among different entity sets.

Space for learners:

5.2 UNIT OBJECTIVES

After going through this unit, you will be able to:

- understand the fundamental concepts of ER model
- know different components are used in ER model.
- know how to design database using ER model.
- understand the fundamental concepts of generalization, specialization and aggregation.

5.3 E-R MODEL

The E-R model was developed by P.P. Chen in around 1976. He introduced E-R model as well as corresponding diagramming techniques. A model that contains entity and relationship sets to represent system data is called E-R (Entity-Relationship) model. E-R model is an important part of DBMS. E-R model is used to model the logical structure of a database.

5.3.1 Components of E-R Model

There are different components of E-R model. The major components of E-R model as follows-

i) Entity

ii) Attributes

iii) Relationships

5.3.1.1 Entity

An entity is any kind of objects having physical existence or conceptual existence. For example, an entity may be person, place, event, or concept etc.

Each entity is distinct from the other entities.

Persons: Students, Department, Customer, Supplier.

Places: Building, Department office etc.

Objects: Car, Machine etc.

Space for learners:

Events: Order, Purchase, Registration etc.

Entity Type: An entity is any object of entity type. For example: S1 is an entity having entity type student.

Entity Set: The collection of entities which are sharing common characteristic is called entity set. For example, the set of all students of a university can be called as the entity set student.

5.3.1.2 Attributes

Attributes are the properties or characteristics of an entity. Suppose, we have an entity called **student**, then **roll-number, regn-no, name**, **address**, **date-of-birth**, **phone-number, age** etc. are the attributes. There are different types of attributes of an entity:

i) Single Valued Attribute: The attributes those have a single value for a particular entity. For example, student registration number is a single valued attribute.

ii) Multi Valued Attribute: Attributes having more than one value are called multi valued attributes. For example, phone number, email address of a student entity is multi valued attributes.

iii) Composite Attribute: Composite attributes are the attributes which can be sub divided into different parts. For example, Student_id of a student entity is a simple attribute and address, name of a student can be composite attributes. We can divide address of a student into house_no, bylane_no, post_office, pin_no etc. and name can be divided into first_name, middle_name and last_name.

iv) Derived Attribute: This is the kind of attribute whose values are derived from other attributes. For example, the age attribute of the student entity can be calculated from the date-of-birth attribute and thus age attribute is called as derived attribute.

5.3.1.3 Relationships

In E_R model technique, Relationships among two entities types can be classified into three categories. These are: One-to-One,

Space for learners:

One-to-Many and Many-to-Many. For our discussion, let's consider few entity types – Faculty, Department, Student and Course. But first discuss few terminologies.

Degree:

The Degree of a Relationship is nothing but the number of entity types which are participating in the relationship. Since we will be discussing the Relationship Concept in terms of two entities and therefore the degree will be 2.

Participation Constraint:

This specifies that how strong is the presence of an entity type when it is related to the other entity type in a Relationship. It is also termed as the minimum Cardinality Constraint. In simple term it specifies the no. of entities of an entity type, participating in a Relationship. There are two types of Participation Constraints – Total and Partial Participations.

i) One-to-One (1:1) Relationship:

In this kind of relationship, one entity of an entity type can only be associated with only one entity of the other entity type. Now, let's find the Relationship between Faculty and Department entities in terms of "Head of". First try to associate Faculty entities with the Department entities in terms of "Head of" a Department. It is obvious that only 1 (one) Faculty can be a "Head of" only 1 (one) Department. Also in a Department, there will be only 1 (one) Head. Fig.-1 depicts this One-to-One association/mapping.

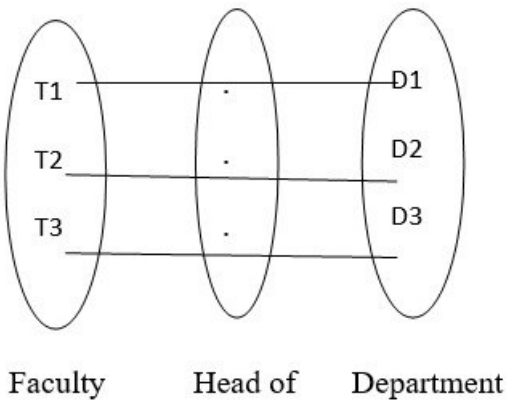


Fig.-1: One-to-One Association

Space for learners:

Thus, we can find a 1:1 Relationship, termed as "Head of", between the Department and the Faculty entity types and is presented in Fig.-2.

Fig.-2: One-to-One (1:1) Relationship

i) One-to-Many (1:M) Relationship:

In this kind of relationship between two entity types, one entity of an entity type may be associated with more than one entity of the other entity type. Now, lets' try to find the Relationship, "Enroll in", between Course and Student. For this, let's assume that a Student can only "Enroll in" a Course and a Course can have many Students. This one-to-many association/mapping between Course and Student is illustrated in fig.-3. Based on this association we can find the required Relationship, which is 1:M, shown in fig.-4.

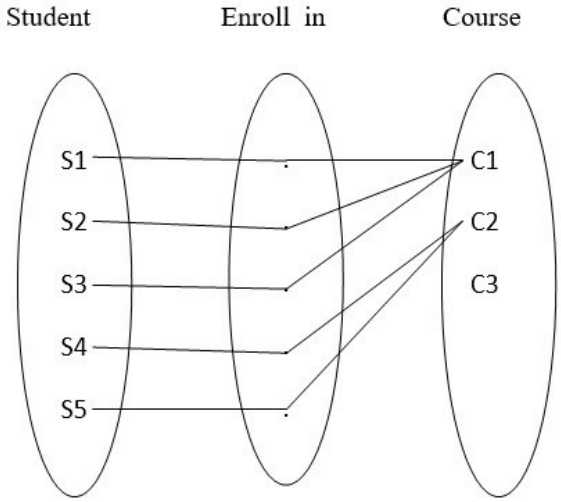


Fig.-3: One-to-Many Association

Fig.-4: One-to-Many (1:M) Relationship

Space for learners:

i) Many-to-Many (M:N) Relationship:

Entities that have many relationships among each other is called a many-to-many relationship. For example, we have two entities namely customer and items and a relationship "buy". One customer can buy at least one item or many items and one item may be bought by one customer or many customers. Suppose we have five customers C1, C2, C3, C4 and C5 and two items I1 and I2. Item I1 is bought by C1, C2 and C3 and Item I2 is bought by C2, C4, and C5. This many-to-many association/mapping between Item and Customer is illustrated in fig.-5. Based on this association we can find the required Relationship, which is M:N, shown in fig.-6.

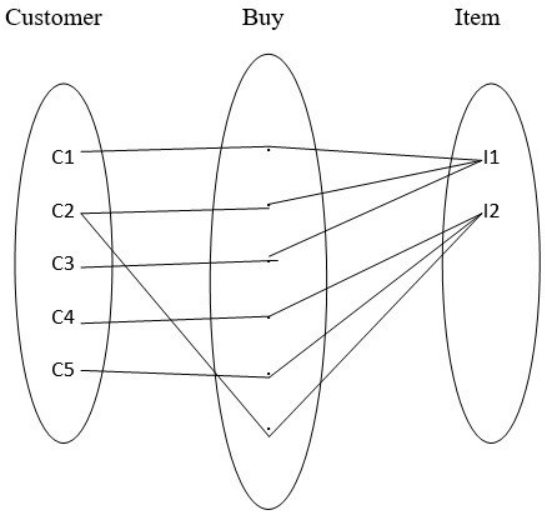


Fig.-5: Many-to-Many Association

Fig.-6: Many-to-Many (M:N) Relationship

5.3.1.4 Key Attributes

The attribute which uniquely identifies an entity in the entity set is called key attribute. For example, student_id can be the key attribute for the entity set student.

Space for learners:

5.4 E-R DIAGRAM

E-R diagram stands for Entity Relationship diagram. It is a graphical representation of the logical structure of the database. The main constituents of an E-R diagram are entities, attributes and relationships.

5.4.1 Symbols used in E-R Diagram

The following symbols are used in E-R diagram (Fig.-7).

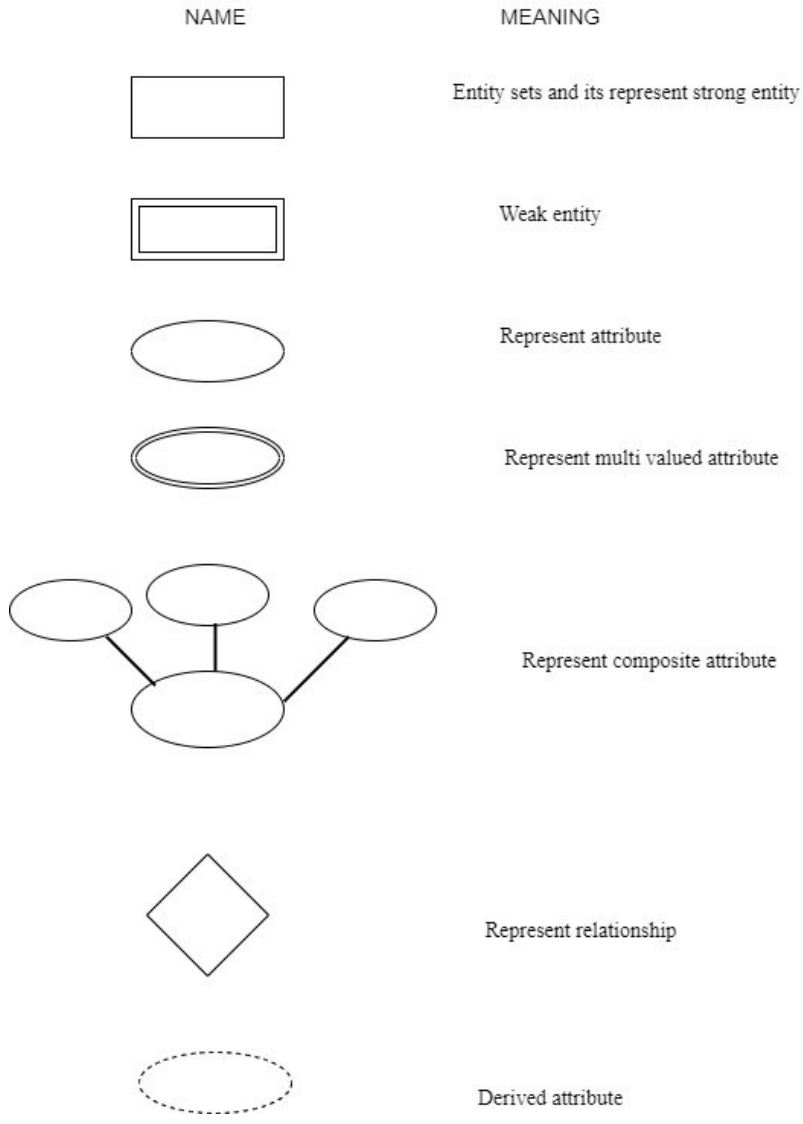


Fig.-7: Symbols used in E-R Diagram

Space for learners:

5.4.2 Example of E-R Diagram

Now let's try to draw an E-R diagram by considering an example of a simple company XYZ. Suppose, the entity types associated are namely – Employee, Department and Projects. Each department has employees and also different projects (may be completed or in hand). Thus, each of the employee is associated with one or more projects. Few of the employees are also assigned with the responsibility of managing their respective departments. The company has to keep information of dependents of each of the employees. Keeping these facts into consideration, the E-R diagram is depicted in fig. 8.

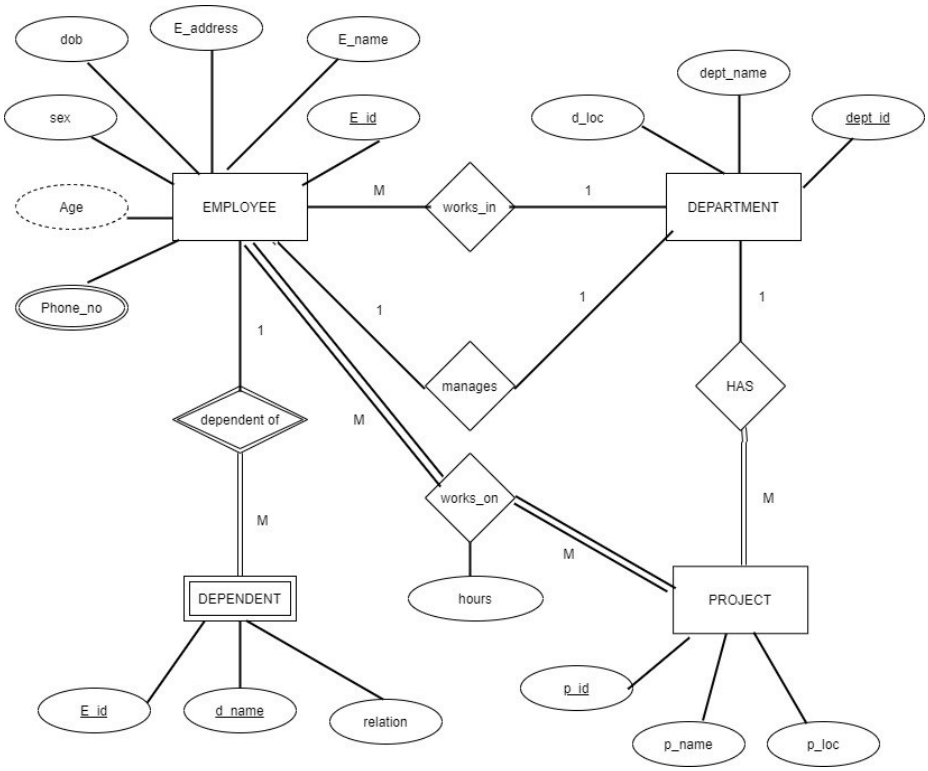


Fig.-8: E_R diagram of the XYZ Company

5.4.3 Transformation of E-R Model to Relational Schema

The following steps are to be followed to transform an E-R model to Relational Schema.

Space for learners:

Step1: Convert all strong entity sets into tables/relations. Simple attributes are mapped. Composite and multi valued attributes are excluded from tables/relations. In our fig.-8, we have three strong entities, so we have to create the following three tables (fig.-9).

EMPLOYEE

E_id	E_name	E_address	dob	sex

DEPARTMENT

dept_id	dept_name	d_loc

PROJECT

p_id	p_name	p_loc

Fig.-9: Relations from Strong Entities

Step2: Convert all weak entities into tables or relations. Primary key of the strong entity is added into the weak entity as a foreign key. In our example we have only one weak entity. So, we have only one table.

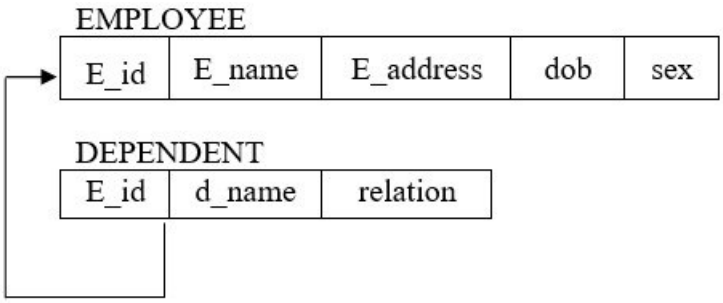


Fig.-10: Mapping of Weak Entity

Step3: Mapping of 1:1 relationship types.

Method 1: Foreign key approach

Let A and B be two entity sets.

i) Identify the entity set with total participation. (say B)

ii) Add primary key of A into B as foreign key.

Method 2: Merged relation approach.

If both entity sets are having total participate then they can be merged into a single relation.

Space for learners:

Method 3: Cross reference approach.

Create a third relation comprising primary key of both entity sets.

In our example there is a 1:1 relationship between EMPLOYEE and DEPARTMENT with mange relationship.

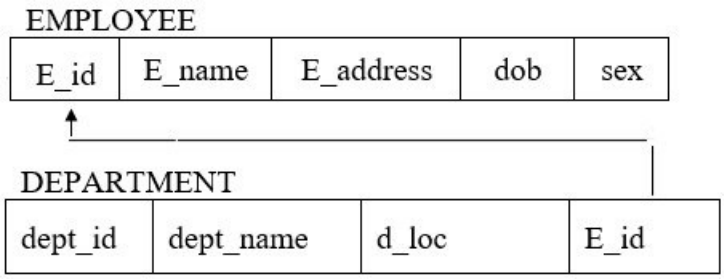


Fig.-10: Mapping of 1:1 with mange relationship

Step4: Mapping of 1: M relationship types.

Let A and B be the entity sets (with 1: M) where B is having total participation in relationships. Add primary key of A in B as foreign key.

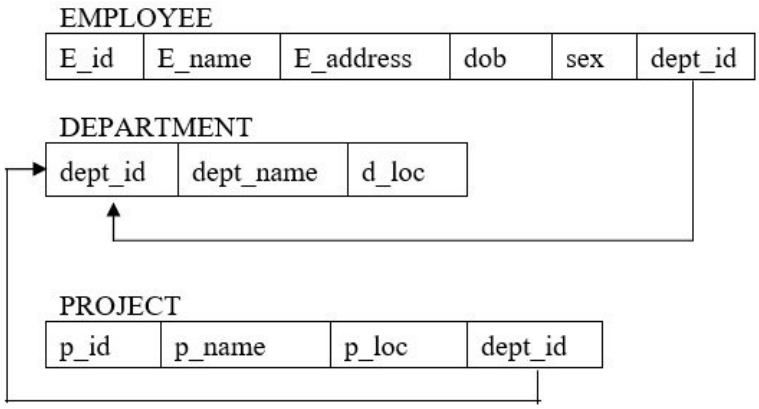


Fig.-11: Mapping 1: M relationships

Step5: Mapping M: N relationship types.

Create a third relation containing the primary keys of both the entity sets and attributes which are in the relation (if any).

Space for learners:

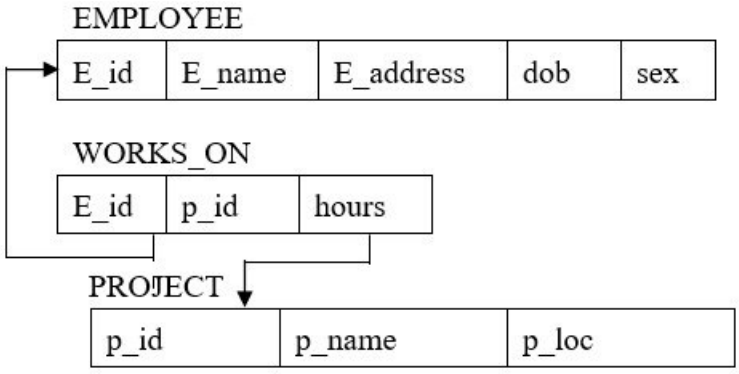


Fig.-12: Mapping of M : N relationship having attribute

Step 6: Mapping multi valued attribute,

For each multi valued attribute, create a separate relation. Add primary key of the entity set in new relation as a foreign key. The foreign key attribute and multi valued attribute will become composite key.

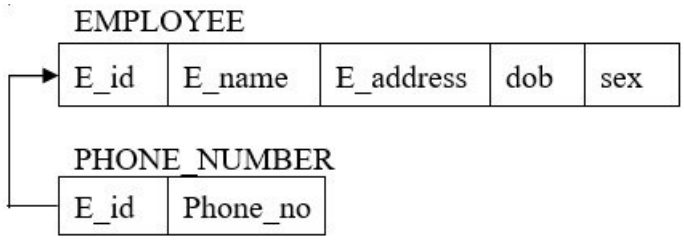


Fig.-13: Mapping multi valued attribute

5.5 GENERALIZATION

Generalizations is the process of retrieving similar properties from a set of lower-level entity sets and create a generalized entity from it. It is a bottom-up approach. Two or more entities which have some common attributes can be generalized to a higher-level entity. For example, Customer and Employee can be generalized to a higher-level entity called Person as shown in fig.-14.

In the following example, similar attributes like Name, Address become part of Person entity and Emp_code and Emp_salary attributes become part of specialized Employee entity.

Space for learners:

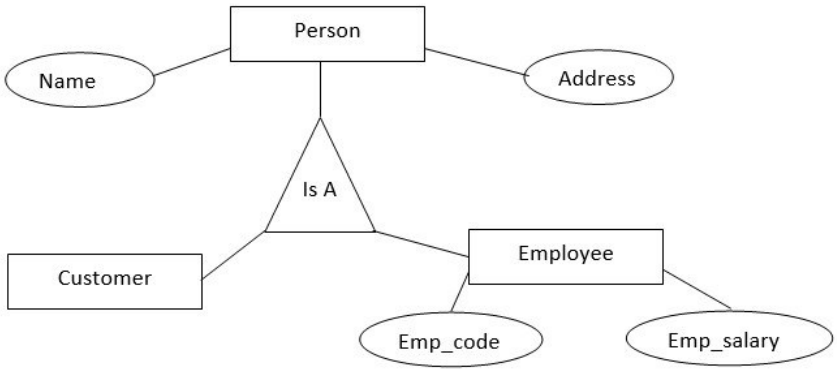


Fig.-14: Generalization

In the above figure, Person is the higher-level entity set and Customer and Employee are lower-level entity sets. The higher-level entity set is called super class and lower-level entity set is called sub class.

5.6 SPECIALIZATION

It is top-down approach. It is the result of taking subsets of a higher-level entity set to form lower-level entity sets. For example, Employee entity can be specialized into set of sub classes namely Salaried_employee and Hourly_Employee. In the following figure, fig.-15, EName (employee's name), address etc. are common for both Salaried_Employee and Hourly_Employee. They become part of higher entity Employee and attributes like mode of payment is called specialized attribute.

Space for learners:

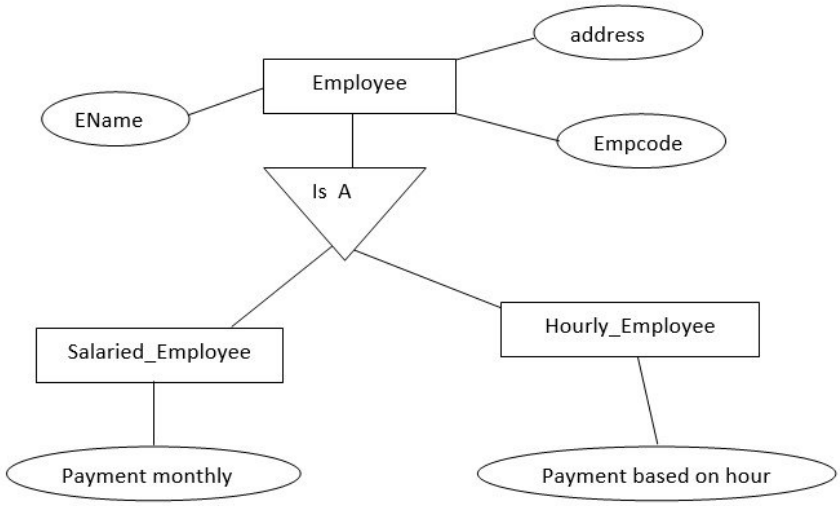


Fig.-15: Specialization

5.7 AGGREGATION

One limitation of the E_R diagram is that it is not express relationships within relationships. In those cases, a relationship with its entities is aggregated into a higher-level entity. Aggregation is the process of compiling information on an object, thereby abstracting a higher-level entity sets.

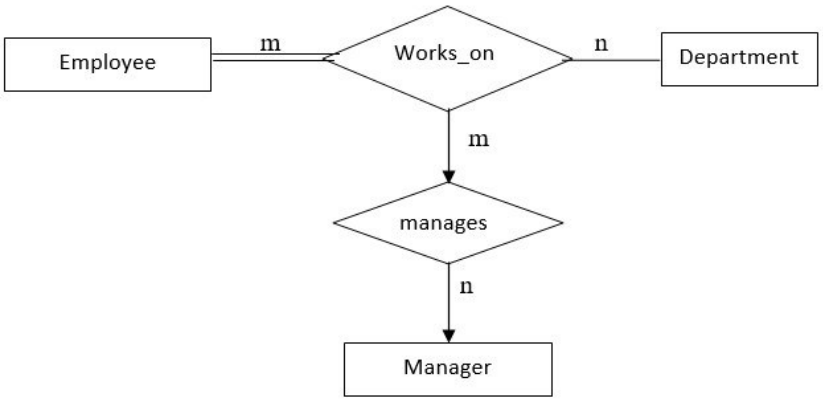


Fig.-16: Aggregation

For example, employee working on a department may need a manager. So, manages relationships is needed between relationship Works_on and entity Manager. Using aggregation, Works_on relationship with its entities. Employee and

Space for learners:

Department is aggregated into single entity and relationship manages is created between aggregated entity and Manager.

To represent aggregation via schema we need primary key of the aggregated relationship; primary key of the associated entity set and descriptive attribute, if exists.

CHECK YOUR PROGRESS

1. What is E-R Model?
2. What is the use of E-R Model?
3. Write down the components of E-R Model.
4. What do you understand by an Entity?
5. What is Degree of a Relationship?

State TRUE or FALSE:

6. A derived attribute is the kind of attribute whose values are derived from other attributes.
7. There are two types of Participation Constraints – Total and Partial Participations.
8. The attribute which does not uniquely identify an entity in the entity set is called key attribute.
9. It is a graphical representation of the physical structure of the database.
10. In E-R Diagram, the Diamond symbol represents a Relationship.

Space for learners:

5.8 SUMMING UP

- In Relational Database Management System E-R Model helps in developing a design.
- P.P. Chen, in around 1976, introduced E-R model as well as corresponding diagramming techniques.
- The major components of E-R model are – Entity, Attributes and Relationships.

- The Degree of a Relationship is nothing but the number of entity types which are participating in the relationship.
- Participation Constraints specifies that how strong is the presence of an entity type when it is related to the other entity type in a Relationship.
- In E_R model technique, Relationships among two entities types can be classified into three categories: One-to-One, One-to-Many and Many-to-Many.
- The steps to be followed to transform an E-R model to Relational Schema are:
    - Convert all strong entity sets into tables/relations,
    - Convert all weak entities into tables or relations,
    - Mapping of 1:1 relationship types,
    - Mapping of 1: M relationship types,
    - Mapping M: N relationship types,
    - Mapping multi valued attribute.
- Generalization is the process of retrieving similar properties from a set of lower-level entity sets and create a generalized entity from it. It is a bottom-up approach.
- Specialization is top-down approach which is the result of taking subsets of a higher-level entity set to form lower-level entity sets.
- Aggregation is the process of compiling information on an object, thereby abstracting a higher-level entity sets.

Space for learners:

5.9 ANSWERS TO CHECK YOUR PROGRESS

1. A model that contains entity and relationship sets to represent system data is called E-R (Entity-Relationship) model.

2. E-R model is used to model the logical structure of a database.

3. There are different components of E-R model. The major components of E-R model as follows-

i) Entity

ii) Attributes

iii) Relationships

4. An entity is any kind of objects having physical existence or conceptual existence.

5. The Degree of a Relationship is nothing but the number of entity types which are participating in the relationship.

6. T

7. T

8. F

9. F

10. T

Space for learners:

5.10 POSSIBLE QUESTIONS

Short Answer type Questions:

1. What is entity?
2. What entity type and entity set?
3. What do you mean by stored and derived attribute? Give examples
4. What do you mean by simple and composite attribute? Give examples
5. What do you mean by single valued and multi valued attribute? Give examples
6. Why we need relationships between two entities?
7. Write the basic difference between strong entity and weak entity.
8. What do you mean by 1: 1 relationship?
9. What do you mean 1: M relationship?
10. What do you mean by M: M relationship?
11. What do you mean by key attributes?
12. Why we need convert E_R model into tables in RDBMS.

13. What do you mean by aggregation?

Long Answer type Questions:

1. Explain the different symbols used in E_R diagram with proper meaning.
2. Explain the E_R diagram with a suitable example.
3. Briefly explain the generalization and specialization with suitable examples.
4. Briefly explain the rules for converting E_R model into tables or relations with suitable examples.

Space for learners:

5.11 REFERENCES AND SUGGESTED READINGS

- Ramez, Elmasri. *Fundamentals of Database Systems*. Pearson Education India, 2020.
- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. McGraw-Hill, 1997.

UNIT6: NORMALIZATION AND FUNCTIONAL DEPENDENCIES

Unit Structure:

Space for learners:

Space for learners:

6.1 INTRODUCTION

A relational database schema comprises of a number relational schemas, where each relational schema is designed by grouping the related attributes. While there are numerous groupings possible for the same set of attributes, not all the groupings lead to a "good" design. A good design can be easily understood by the users, follows a logical organization of the attributes, and minimize redundancy. In this model, we are going to discuss some informal guidelines to measure the "goodness" of a relation. Another important concept- functional dependency, which refers to the

constraints that exist among the attributes of a relation, is also introduced in this module. Functional dependency is an important tool to measure how appropriate is the grouping of the attributes in a relation. This module also discusses normal forms and the process of normalization. A relational schema is said to be in a normal form if it meets certain desirable properties. The process of converting a relation into a normal form is called normalization. Functional dependency and constraints on key attributes can be used to analyze which normal form relation is and also help in further normalizing the relation if possible. Some other advanced concepts like - multivalued dependency, join dependency and lossless join property are also presented in this module.

Space for learners:

6.2UNIT OBJECTIVES

After completion of this module, you will be able to -

- *list* the informal measures to assess the quality of relational schema design.
- *describe* the various functional dependencies and normal forms.
- *understand* the concept of null values, redundant information, and spurious tuples and how to eliminate these by performing normalization.
- *apply* the concept of database normalization (1 NF, 2NF, 3 NF, etc.) to create an efficient relational schema design to organize the data logically and meaningfully and eliminate redundancy.
- *analyze* whether a given relational schema design follows the basic guidelines of design or not.
- *evaluate* in which normal form a given relational schema is, and if possible, convert it to a higher normal form.
- *create* good relational schema designs by applying the algorithms for losses join properties.

6.3 INFORMAL DESIGN OUTLINES FOR RELATIONAL DATABASES

Space for learners:

A relational schema can be defined as a set of relational tables and associated items related to each other. While it is possible to design multiple relational schemas for the same problem, the challenging task is to choose the good one. The design guidelines help us to assess the quality of the relational schemas and thus enable us to achieve good quality relational schema designs. The following are the four informal design guidelines-

- The semantics of the Relation
- Minimizing redundancy
- Reduction of the null values in tuples.
- Discarding the possibility of generating spurious tuples.

FACULTY

F_NAME	F_ID	F_GENDER	F_DOB	D_NO
	p.k.			

DEPARTMENT

D_NO	D_NAME	D_EMAIL
p.k.		

COURSE

C_CODE	C_NAME	C_CREDIT	F_ID	D_NO
p.k.			f.k.	f.k.

Fig-6.1: Relational Schema design with clear semantics

6.3.1 Semantics of a Relation

When we arrange attributes to construct a relation schema, we presume that each attribute has a specific meaning. This meaning, or semantics, describes how to interpret the attribute values recorded in a tuple of the relation, or how the attribute values in a tuple

relate to one another. For example, in figure 1.1, the relations FACULTY, COURSE, and DEPARTMENT have distinct semantics. The attributes in the relations are also self-explanatory. The relation DEPARTMENT represents details of a department- *department number* (D_NO), *name of the department* (D_NAME), and the *department email id* (D_EMAIL). D_NO is the primary key of the relation, inferring that each department has a unique department number. The relation FACULTY, on the other hand, outlines the details of a faculty, like-*name* (F_NAME), *id* (F_ID), *gender*(F_GENDER), and *date of birth*(F_DOB). F_ID is the primary key of the relation. The attribute, D_NO in FACULTY is the foreign key from the relation DEPARTMENT, indicating the implied relationship between the two relations. Similarly, the relation COURSE also has a distinct meaning. It depicts course details like - *name of the course* (C_NAME), *course code* (C_CODE), and *credit* (CREDIT). The attributes, F_ID and D_NO, in COURSE are the foreign keys from FACULTY and DEPARTMENT respectively and the attribute C_CODE is the primary key.

Guideline 1: Create a relationship schema that is self-explanatory and thus simple to understand. If a relation schema relates to a single entity type or relationship type, the meaning is usually obvious. However, in a single relation, if attributes from different entity types and relationship types are combined, the relation becomes semantically unclear.

6.3.2 Minimization of Redundancy

Redundancy is the repetition of the same fact again and again across multiples places in the same database. Apart from wastage of storage space, redundancy also results in various other side effects. In designing a relational schema, therefore, one of the most important goals is to minimize redundancy across. Proper grouping of the attributes in a relation schema helps significantly in minimizing redundancy. This can be illustrated with the example in figure 6.2. The relations FACULTY_DEPT and COURSE_DEPT are being designed to represent the faculties and the courses. The rela-

Space for learners:

tions cover all the aspects the such as - which faculty works for which department and which course is offered by which department. However, if compared with the design in figure 6.1, the design in figure 6.2 consumes more storage space. In figure 6.1, the department number and department email id have been mentioned only once for a particular department in the DEPARTMENT relation. However, in figure 6.2, in the FACULTY_DEPT relation, these two details are repeated for every employee that belongs to a particular department. The same is also the case with the COURSE_DEPT relation.

Apart from wastage of storage space, redundancy leads to another serious issue of update anomalies. Insertion, deletion, and modification anomalies are the three categories of update anomalies. A brief discussion of each is presented in this section.

Space for learners:

6.3.2.1 INSERT Anomalies

FACULTY_DEPT

F_NAME	F_ID	F_GENDER	F_DOB	D_NO	D_NAME	D_EMAIL
	p.k.					

COURSE_DEPT

C_NAME	C_CODE	CREDIT	F_ID	D_NO	D_NAME	D_EMAIL
	p.k.		f.k.			

Fig-6.2: Relational Schema design with redundancy

Consider table 6.1, which is the populated table for the relation FACULTY_DEPT. Every time we enter a faculty detail, we must also enter the corresponding department details. While entering these details, one must be careful about entering all the fields correctly. For example, two faculty members working for department number 1, must have the same values for the attributes D_NAME and D_EMAIL. However, as we can observe from table 6.1, the faculty members with id 123 and 124 work for the same department but D_NAME and D_EMAIL values are different. This results in the inconsistency of the database.

Table 6.1: Populated FACULTY_DEPT table

F_ID	F_NAME	F_GENDER	F_DOB	D_NO	D_NAME	D_EMAIL
123	Ravi Singh	Male	06-02-1972	1	Geography	geo@gmail.com
124	Rahul Bose	Male	05-04-1980	1	English	eng@gmail.com
125	P. Joseph	Male	12-07-1979	2	Social Science	Sssc@gmail.com
126	Sima Mishra	Female	08-11-1985	3	Mathematics	maths@gmail.com

Another difficulty is that there is no option to enter the details of a department which has not appointed any faculty yet. The department details can be entered into FACULTY_DEPT relation, only where there is at least one faculty who is working in that department. These issues will not occur in the design of figure 6.1 as the department details are not clubbed with the faculty details and thus there is no redundancy.

6.3.2.2 DELETION Anomalies

This can be inferred from the second issue in insertion anomaly. From table 6.1, if we delete the faculty information with id 125, then we will lose all the details of department 2. This is because the faculty, with id 125, is the only faculty working in department 2. The same will be the problem if we delete the tuple with id 126. This problem does not occur in the database of figure 6.1, as deleting a tuple from the FACULTY relation will not cause any deletion of tuples from the DEPT table. Thus all the tuples in DEPT will still be intact.

6.3.2.3 MODIFICATION Anomalies

This again can be inferred from the first issue of insertion anomaly. If there are some changes made in one department details- such as the department name, the same has to be updated in all the tuple of the FACULTY_DEPT relation with that department number. For example, if we wish to change the department email id of department number 1, then we need to update the same in all the faculty tuples that are working in department 1. Even if we forget to update it in one tuple, the database will be inconsistent.

Space for learners:

Guideline 2: Design a schema with minimum redundancy, so that there are no update anomalies. In case of any unavoidable redundancy, the program must be designed to tackle all the related insertion, deletion, and modification anomalies.

Space for learners:

6.3.3 Reducing the NULL values in tuples

A NULL value for an attribute in a tuple can have multiple interpretations, such as-

- The attribute doesn't apply to this tuple.
- The value of the attribute is unknown for this tuple
- The value of the attribute is known but has not been recorded yet.

The NULL values not only result in wastage of space but also creates problems in many operations such as JOIN operations, aggregate operations such as COUNT or SUM, etc.

Guideline 3: While designing a relational schema we should group the attributes in such a way that produces as few NULL values as possible.

6.3.4 Spurious Tuples

Many a time, a relational schema has to be decomposed into smaller relations. Inappropriate decomposition of the relation may result in some information that originally did not exist in the original relation. For example, let a relation R be decomposed into two smaller relations R1 and R2. If the natural join of R1 and R2 produces any extra tuple that does not exist in the original relation R, then that tuple is called the spurious tuple. Let's consider the relation R in table 6.2(a). Decomposition of R into relations R1(A, B) and R2(B, C) will result in the following two relations as shown in tables 6.2(b) and 6.2(c) respectively. Now the natural join over R1 and R2 will result in table 6.2(d). As we may observe, table 6.2(d) has two extra tuples that are originally not present in R. These are called spurious tuples. Spurious tuples represent wrong or invalid information and thus leads to the inconsistency of the database.

Table 6.2 (a): Spurious tuples: Relational schema R

A	B	C
a1	b1	c1
a2	b1	c2

Table 6.2 (b): Spurious tuples: Relational schema R1

A	B
a1	b1
a2	b1

Table 6.2 (c): Spurious tuples: Relational schema R2

B	C
b1	c1
b1	c2

Table 6.2 (d): Spurious tuples: R1*R2

A	B	C
a1	b1	c1
a1	b1	c2
a2	b1	c1
a1	b1	c2

Guideline 4: Decompose a relation into multiple relations in such a way that the natural join of the smaller relations does not produce any spurious tuple. This can be done by having relations, where the common attributes are either the primary key or foreign key of the

Space for learners:

relations. In case of an unavoidable situation where two relations have common non-key attributes, extra care should be taken not to join such relations.

CHECKYOURPROGRESS-I

1. Redundancy in a database lead to _____, _____ and _____ anomalies.
2. If the natural join of two relations results in extra tuples that are not in the original relation, then those tuples are called as _____ tuples.
3. State true or false
   a. NULL vales in a relation always have specific meaning.
   b. Spurious tuples represent wrong or invalid information.
   c. A good relation always has complex semantics.

Space for learners:

6.4 FUNCTIONAL DEPENDENCIES

In Database Management System (DBMS), functional dependency (FD) refers to the relationship between two attributes in a table or relation. For any relation, if the value of the set of attributes Y is determined by the value of the set of attributes X, then Y is said to be functionally dependent on X. This is symbolically represented by $X \rightarrow Y$. This notation can be read as "*X functionally determines Y*" or "*Y is functionally determined by X*". If this functional dependency holds, then for every valid instance of X there will be a unique value of Y in the table. Usually, the functional dependency exists between a prime key attribute and a non-key attribute(s). Thus, in a relation R, if two tuples, say t1 and t2 have the same values of X, then they must have the same values of Y as well. A functional dependency is the property of the attributes in a relation. It must hold for every tuple in a relation. The concept of functional dependency was introduced by E.C. Codd. It helps in avoiding bad design and in avoiding data redundancy. To better understand functional de-

pendency, let us consider the relations in figure 6.1. Tables 6.3, 6.4, and 6.5 are the populated tables for the relations FACULTY, DEPARTMENT, and COURSE respectively.

Table 6.3: Populated FACULTY relation

F_ID	F_NAME	F_GENDER	F_DOB	D_NO
123	Ravi Singh	Male	06-02-1972	1
124	Rahul Bose	Male	05-04-1980	1
125	P. Joseph	Male	12-07-1979	2
126	Sima Mishra	Female	08-11-1985	3

Table 6.4: Populated DEPARTMENT relation

D_NO	D_NAME	D_EMAIL
1	Geography	geo@gmail.com
2	Social Science	Sssc@gmail.com
3	Mathematics	maths@gmail.com

Table 6.5: Populated COURSE relation

C_NAME	C_CODE	CREDIT	F_ID	D_NO
Physical geography	230	3	123	1
Anthropology	231	3	125	2
Graph Theory	232	3	126	3
Real Analysis	233	4	126	3

We can see from table 6.4, D_NAME is uniquely determined by D_NO. Thus in this relation the functional dependency, D_NO→ D_NAME holds. Similarly, a few other functional dependencies that hold in the relations DEPARTMENT, FACULTY, and COURSE are-

Space for learners:

- F_ID → F_GENDER
- F_ID → F_GENDER
- D_NO → D_EMAIL
- C_NAME → C_CREDIT
- C_CODE → F_ID
- C_NAME → F_NAME

Space for learners:

6.4.1 Types of Functional Dependencies

Functional dependencies can be classified into the following forms-

- Trivial functional dependencies
- Non-trivial functional dependencies
- Multivalued functional dependencies
- Transitive functional dependencies
- Full Functional dependencies

6.4.1.1 Trivial functional dependencies

A functional dependency X→Y is said to be trivial if Y is a subset of X. For example, the functional dependency {F_ID, F_NAME}→ F_NAME is a trivial functional dependency since F_NAME is a subset of {F_ID, F_NAME}. Similarly, {D_NUMBER, D_NAME}→ DNAME is also another example of trivial functional dependency.

6.4.1.2 Non-trivial functional dependencies

Unlike in trivial function dependency, in non-trivial functional dependency, the set of attributes on the right-hand side is not a subset of the attributes on the left-hand side. In other words, if X→Y, and Y is not a subset of X, then the functional dependency is said to be non-trivial. For example, the functional dependency, F_ID→ {F_NAME} in table 6.3 is an example of non-trivial functional dependencies. Another example of non-trivial functional dependency from table 6.5 is
COURSE_ID →C_NAME.

6.4.1.3 Multivalued functional attributes

If there exists a functional dependency of the form X→ {Y, Z} such that there is no dependency between Y and Z, then the FD is said to be a multivalued functional dependency. In other words, multi values dependency occurs when two or more attributes in a table are functionally independent of each other but are functionally determined by a specific attribute. Multivalued dependency is represented by the symbol "→→". For multivalued dependency, we must have at least three attributes in the relation. A more detailed discussion of the multivalued attribute is presented in section 6.5.

6.4.1.4 Transitive functional dependencies

In a relation, if the functional dependencies X→Y and Y→ Z exist, then the functional dependency X→Z also exists. This is called transitive dependency. For transitive dependency to exists, there must be at least three attributes in the relation. For example, consider table 6.6. In this table, S_ID determines S_Name and S_Name determines S_Age. Due to transitive dependency, we can also state that S_ID determines S_Age. Thus we can summarise as-

- S_ID→S_Name
- S_NAME→S_Age
- S_ID→S_Age [due to transitivity]

Table 6.6: Example of transitive dependency

S_ID	S_Name	S_Age
1	Ravi	20
2	Rohan	19
3	Sukanya	21
4	Puja	20

.

6.4.1.5 Full Functional dependencies

A functional dependency X→Y is said to be a full functional dependency if removal of any attribute from X means the functional

Space for learners:

dependency doesn't exist any longer. For example, consider table 6.7, showing the number of hours (per week) assigned to the employees for different projects. In this table {E_ID, PROJECT_ID}→ HOURS. If we remove any attribute from the left-hand side then the dependency no longer holds as neither E_ID→HOURS nor PROJECT_ID→HOURS. Thus, it is an example of full functional dependency.

Table 6.7: Example of full functional dependency

E_ID	PROJECT_ID	HOURS
1	3	16
1	2	20
2	1	12
3	3	10

6.4.2 Inference Rules for Functional Dependencies

While designing a relational schema R, the designer also specifies a set of functional dependencies. Let's consider that this set of functional dependencies is denoted by F. Usually, the schema designers list only the functional dependencies that are semantically obvious. Apart from the functional dependencies in F, it is possible to infer several other functional dependencies that hold in any legal relation instances in R. For example, one of the functional dependencies that hold for the FACULTY relation in figure 6.1 is F_ID→ {F_NAME, F_GENDER, F_DOB, D_NO}. We can easily infer a number of other functional dependencies from the given FD. Some of these are-

- F_ID→ {F_NAME}
- F_ID→{F_GENDER, F_DOB}
- F_NAME→{F_DOB}

It is not practically possible to mention all the functional dependencies that hold in a relation schema. However, we can systematically

Space for learners:

infer the other functional dependencies with the help of the inference rules. This set of inference rules were first introduced by William W. Armstrong in 1974. These rules are thus also called Armstrong's axioms. These axioms define a set of rules which, if applied repeatedly, generate all the other functional dependencies that can be inferred from a set of functional dependencies originally specified by the designer.

Armstrong's Axioms:

- **IR1: Axiom of reflexivity:**

  If Y is a subset of X, i.e, $Y \subseteq X$, the $X \rightarrow Y$.

- **IR2: Axiom of augmentation:**

  If in a relation the functional dependency $X \rightarrow Y$ holds, then the functional dependency $XZ \rightarrow YZ$ also holds in that relation.

- **IR3: Axiom of transitivity:**

  In a relation if the two functional dependencies $X \rightarrow Y$ and $Y \rightarrow Z$ hold, the functional dependency $X \rightarrow Z$ also holds.

Armstrong showed that the inference rules from IR1 to IR3 are sound and complete. Soundness means - if we consider any relational schema R with a set of functional dependencies as F, then for any legal relational instance r of R, which satisfies the functional dependencies in F also satisfies the functional dependencies inferred using IR1 to IR3. On the other hand, the rules are complete in the sense that, application of the rules IR1 to IR3 over F until no additional functional dependencies are generated will result in all the possible dependencies that can be inferred from F.

Some other important secondary rules that can be derived from the above inference rules are-

- **IR4:Decomposition or Projective rule**

  If $X \rightarrow YZ$ holds in a relation, then the functional dependencies $X \rightarrow Y$ and $X \rightarrow Z$ also hold.

Proof:

Step 1: $X \rightarrow YZ$ [Given]

Step 2: $YZ \rightarrow Y$ [Applying IR1, as $Y \subseteq YZ$]

Step 3: $X \rightarrow Y$ [Applying IR3 on step 1 and step 2]

Space for learners:

Similarly, we can prove that X→ Z.

- **IR5: Union or additive rule:**

If the functional dependencies X→Y and X→Z hold in a relation, then the functional dependency X→YZ also holds in the relation.

Proof:

Step 1: X→Y	[Given]
Step2: X→Z	[Given]
Step3: XY →YZ	[Applying IR2 on step2]
Step4: XX→XY	[Applying IR2 on step1]
Step5: X→XY	[As XX=X]
Step6: X→YZ	[Applying IR3 over steps 5 and 3]

- **IR6: Pseudo Transitivity**

If the functional dependencies X→Y and WY→Z hold in a relation, then the relation WX→Z also holds in the same relation.

Proof:

Step 1: X→Y	[Given]
Step 2: WX→WY	[Applying IR2 on step 1]
Step 3: WY→ Z	[Given]
Step 4: WX→Z	[Applying IR3 on step2 and step 3]

CHECK YOUR PROGRESS-II

4. The functional dependency {ISBN, Book_Name} → Book_Name is an example of _____ functional dependency.
5. If in a relation named COMPANY, C_name → C_location and C_location → C_pincode, then we can say infer that C_name → C_pincode due to _____.
6. If A→B and A→ C then, due to additive rule we can infer that _____ .
7. State true or false
   a. Functional dependency represents the relation between two tables.
   b. If A and B are two sets of attributes and A is a subset of B, then we can say that B → A.
   c. The three axioms- reflexivity, augmentation and transitivity represent the complete and sound sound set of inference rules.

Space for learners:

6.4.3 CLOSURE of Functional Dependencies

Space for learners:

For any relational schema R, if the set of functional dependencies is specified as F, then the set of all the functional dependencies that can be inferred from F, is called the closure of F. The closure of F is denoted as F^+. The closure of a set of functional dependencies F, F^+, can be derived by repeatedly applying the inference rules over F unless a point is reached where no additional functional dependencies are generated.

To find closure of a functional dependency F systematically, first, we need to identify each set of attributes X that occurs as the left-hand side of any functional dependence in F. The next step is to identify the set of all attributes that are dependent on X. As a result, for each such set of attributes X, we find the set of attributes that are functionally determined by X based on F; this is referred to as the closure of X under F and is denoted by X^+.

Example: Let's consider a relation STUDENT (ID, NAME, CGPA, LOCATION) with the set of functional dependencies specified is F={ID → NAME, NAME→CGPA, ID →LOCATION}

For finding the closure of F, we need to find the closure of the attribute present in the left-hand side of the functional dependencies. Thus, we need to find ID^+ and $NAME^+$

Step 1: ID → ID [Due to IR1]

Step 2: ID →NAME [Given]

Step 3: ID →{ID, NAME} [Applying IR5 on step1 and 2]

Step 4: ID →LOCATION [Given]

Step 5: ID →{ID, NAME, LOCATION} [Applying IR5 on step3 and 4]

Step 6:NAME→CGPA [Given]

Step7: ID → CGPA [Applying IR3 on step 2 and step 6]

Step 8: ID →{ID, NAME, LOCATION, CGPA} [Applying IR5 over step 5 and 7]

Thus

ID^+={ID, NAME, LOCATION, CGPA}

Similarly, we can find that

$NAME^+$={NAME, CGPA}

Thus, the closure set with respect to F is:

ID^+={ID, NAME, LOCATION, CGPA}

$NAME^+$={NAME, CGPA}

Space for learners:

6.4.4 Equivalent Sets of Functional Dependencies

Let F and G be two sets of functional dependencies for a relational schema R. G is said to be **covered** by F if all the dependencies in G can be inferred from F. In other words, F **covers** G if G^+ is a subset of F^+ i.e. $G^+ \subseteq F^+$. On the other hand, F and G are said to be **equivalent** if the following contains are satisfied:

- All the functional dependencies in F can be derived from functional dependencies in G.
- All the functional dependencies in G can be derived from functional dependencies in F.

In other words, if the closure of F is equal to the closure of G, i.e., if $F^+ = G^+$, then F and G are said to be equivalent. Alternatively, we can also say that F and G are equivalent if F covers G and G covers F.

Example: A relation R (P, Q, R, S, T) is has two set of FDs F and G specified as follows-

F={P → Q, PQ → R, S → PR, S → T}

G= {P → QR, S → PT}

Determine whether F covers G:

Step-1:

- $(P)^+$ = { P, Q, R } // closure of left side of P → QR using set G
- $(S)^+$ = { P, Q, R, S, T } // closure of left side of S → PT using set G

Step-2:

- $(P)^+$ = { P, Q, R } // closure of left side of P → QR using set F

- $(S)^+$ = { P, Q, R, S, T } // closure of left side of S → PT using set F

From Step-1 and Step-2, we can conclude that F covers G i.e. F ⊇ G, as the FDs in F can determine all the attributes that are determined by the FDs in G.

Determining whether G covers F

Step-1:

- $(P)^+$ = { P, Q, R } // closure of left side of P → Q using set F
- $(PQ)^+$ = { P, Q, R } // closure of left side of PQ → R using set F
- $(S)^+$ = { P, Q, R, S, T } // closure of left side of S → PR and S → T using set F

Step-2:

- $(P)^+$ = { P, Q, R } // closure of left side of P → Q using set G
- $(PQ)^+$ = { P, Q, R } // closure of left side of PQ → R using set G
- $(S)^+$ = { P, Q, R, S, T } // closure of left side of S → PR and S → T using set G

From Step-1 and Step-2, we can conclude that G covers F i.e. G ⊇ F, as the FDs in G can determine all the attributes that are determined by the FDs in F.Thus we can conclude that F=G.

6.4.5 Minimal Cover of Functional Dependencies

A set of FDs, F_{min} issaid to be the minimal cover of another set of functional dependency F if-

- F_{min} is the minimal set of functional dependencies and
- F_{min} is equivalent to F.

A set of functional dependency F_{min} is said to be minimal if the following conditions are satisfied-

- Each functional dependency in the set has only one attribute to the Right Hand Side (RHS)

Space for learners:

- Any dependency in F_{min},say X→Z, can not be replaced by some other functional dependency Y→Z, where Y ⊂ X.
- If we remove any dependency from F_{min}, the resultant set will no longer be equivalent to F_{min}.

Algorithm 6.1:Finding minimal cover of a functional dependency
Step1: Identify the functional dependencies that have more than one attribute to the RHS. Transform them into a series of functional dependencies having only one attribute to the RHS.
Step2: Remove the redundant attributes on the left-hand side.
Step3: Eliminate the redundant functional dependencies.

Example:
Let's consider the functional dependency F={P → R, PQ → R, R → SU, RS → U, TR → PQ, TU → R}
Step 1: F1={P→R, PQ→R, R→S, R→U, RS→U, TR→P, TR→ Q, TU→R}
Step 2: To find the redundant attributes, we need to first find the closure of each attribute.

i. P^+-=PRSU
ii. Q^+ =Q
iii. R^+=RSU
iv. S^+=S
v. T^+=T

We can see from (i) that P^+ includes R. Thus Q is extraneous in PQ→R and thus Q can be removed. So, this dependency can be rewritten as P→R.
From (iii), we can see that R^+ includes U, thus in RS→U, S is extraneous. So, we can rewrite it as R→U.
Thus the new reduced set of functional dependencies can be written as,

F2={P→R, R→S,R→U, TR→P, TR→Q, TU→R}

Step 3: The last step is to eliminate the redundant dependencies
Here, TU→R is redundant as R can be determined using P due to the functional dependency P→R. Thus the final set of minimal cover for F is

F_{min}={P→R, R→S,R→U, TR→P, TR→Q}

Space for learners:

CHECKYOURPROGRESS-III

8. If a and B are two sets of functional dependencies such that A covers B and B covers A, then A and B are said to be _____.
9. If G is a set of functional dependencies, then its closure is denoted by _____ .
10. In a relation R, the functional dependencies A→BC and B→D hold. The closure of the attribute A in that relation is _____.
11. State true or false
    a. Two sets of functional dependencies F and G are said to be equivalent if $F^+=G^+$.
    b. F is the minimal cover of G. If we remove any functional

Space for learners:

6.5 NORMALIZATION AND NORMAL FORMS

In DBMS normalization is used to minimize redundancy. As we have already discussed in section 6.2.2, redundancy in relations may lead to insert, delete, and modification anomalies. Normalization helps in breaking down big relations into smaller relations and ensures that data is stored logically with minimal redundancy.

Normal forms a relation reflects its degree of normalization. It refers to the highest normal form condition that the relation satisfies. The normal forms that will be discussed in this unit are- first normal form (1NF), second normal form (2NF), third normal form (3NF), Boyce- Codd normal form (BCNF), and fourth normal form (4NF). In practice, the database designer has to normalize the relations to the highest normal form possible (usually up to 3NF, BCNF, or 4NF).

6.5.1 Definition of Keys

- **Super Key**: For a relational schema R={A, B, C,….., J}, a super key, S, is a set of attributes such that S ⊆ R and for each legal tuple in R, S has a unique value.
- **Key**: A key, K is a minimal super key. This implies that if we remove any attribute from K, then K will no longer hold the super key property.
- **Candidate Key:** A relational schema sometimes may have more than one key. In that case, each key is referred to as a candidate key. The designer may assign any of the candidate keys as the **primary key**. The other candidate keys are then referred to as **secondary keys**.
- Prime and non-prime attributes: If an attribute is a member of a candidate key, it is referred to as a prime attribute, otherwise as a non-prime attribute.

6.5.2 First Normal Form

The first normal states that each attribute in a relation must have atomic values. For a relation to be in1NF, each tuple in that relation must have single values for each attribute. Thus, 1NF disallows multivalued and composite attributes.

Let's consider the relation shown in table 6.8(a). It is not in 1NF as E_PHONE_NO is a multivalued attribute. We can convert it to 1NF by distributing the multiple values of phone number across the rows and making E_ID and E_PHONE_NO a combined primary key as shown in table 6.8 (b). By definition, each relation in a relational model by default is in 1NF.

Table 6.8(a): Example of a relation that is not in 1NF

E_ID	E_NAME	E_PHONE_NO
1	Ravi Sharma	912346795
2	Erica Swift	8123456745
3	Rahul Nath	{6712387453, 6532478456}
4	Prabin Kumar	77345546734

Space for learners:

Space for learners:

Table 6.8 (b): Normalized version of the relation in table 6.8(a)

E_ID	E_PHONE_NO	E_NAME
1	912346795	Ravi Sharma
2	8123456745	Erica Swift
3	6712387453	Rahul Nath
3	6532478456	Rahul Nath
4	77345546734	Prabin Kumar

6.5.3 Second Normal Form

The second normal form is based on full functional dependency. A relation is in second normal form it is already in 1NF and all the non-prime attributes in the relation are fully functionally dependent on the prime key. Let's consider the following relation in table 6.9(a)-

Table 6.9(a): Example of a relation that is not in 2NF

S_ID	COURSE_ID	S_NAME	COURSE_NAME	GRADE
1	1	Ravi	Java	A+
1	2	Ravi	Python	B
2	1	Rahul	Java	A
2	2	Rahul	Python	A+

The above table stores the grade scored by the students in different subjects. The primary key of the table is {S_ID, COURSE_ID}. The following are some of the functional dependencies that hold in the above relation-

i. {S_ID,COURSE_ID}→GRADE
ii. {S_ID}→S_NAME
iii. {COURSE_ID}→COURSE_NAME

The FD (i) is full functional dependency as GRADE is dependent on the S_ID and the COURSE_ID. Neither S_ID nor COURSE_ID alone can determine the GRADE. However, as we can see that the FDs (ii) and (iii) are partial dependencies as S_NAME can be determined by S_ID alone. Similarly, COURSE_NAME can be uniquely identified by COURSE_ID only. Due to these partial dependencies, the relation is not in 2NF.

A relation that is not in 2NF, can be converted to 2NF by breaking it into multiple relations where the nonprime attributes are fully dependent on the primary key. For example, the relation in table 6.9 (a) can be normalized to 2NF by breaking it down into Grade, Student and Course tables as shown in tables 6.9(b)-6.9(d).

Space for learners:

Table 6.9 (b): Grade table which is in 2NF

S_ID	**COURSE_ID**	**GRADE**
1	1	A+
1	2	B
2	1	A
2	2	A+

Table 6.9(c): Student table(2NF)

S_ID	**S_NAME**
1	Ravi
2	Rahul

Table 6.9(d): Course table (2NF)

COURSE_ID	**COURSE_NAME**
1	Java
2	Python

6.5.4 Third Normal Form

Space for learners:

The third normal form is based on transitive dependency. To be in 3NF, a relation must also be in 2NF, and no non-prime attribute should be transitively dependent on the primary key. An attribute, Z, in a relation is transitively dependent on the primary key X, if the functional dependencies X→Y and Y→Z hold, where Y is neither a candidate key nor a subset of any key in that relation.
Consider the relation in table 6.10(a) that stores the information of the students and the corresponding programs they are enrolled in. The primary key of the table is S_ID.

Table 6.10(a): Example of a relation violating 3NF

S_ID	**S_NAME**	**PROGRAM_NAME**	**PROGRAM_DURATION**
1	Ravi	B. Tech	4
2	Rahul	BCA	3
3	Arati	BCOM	3
4	Arif	MCA	2

Some of the functional dependencies that hold in the relation are-

i. S_ID→S_NAME
ii. S_ID→PROGRAM_NAME
iii. PROGRAM_NAME→PROGRAM_DURATION
iv. S_ID→PROGRAM_DURATION

We can see that the FD (iv) is a transitive dependency that can be inferred from FDs (i) and (ii) using the IR3. However, PROGRAM_NAME is not a candidate key in this relation neither it is a subset of any key. Thus, it can be concluded that the relation is not in 3NF as PROGRAM_NAME is transitively dependent on the primary key S_ID via the non-prime attribute PROGRAM_NAME. We can covert it to 3NF by breaking it into two relations as shown in table 6.10(b) and 6.10(c).

Table 6.10(b): Student relation which is in 3NF

S_ID	S_NAME	PROGRAM_NAME
1	Ravi	B. Tech
2	Rahul	BCA
3	Arati	BCOM
4	Arif	MCA

Table 6.10(c): Program relation which is in 3NF

PROGRAM_NAME	PROGRAM_DURATION
B. Tech	4
BCA	3
BCOM	3
MCA	2

6.5.5 Boyce Codd Normal Form (BCNF)

The Boyce codd normal form is a stricter version of 3NF. Originally, it was proposed to simplify the definition of 3NF, however ended up putting more constraints on the relation. Every relation to be in BCNF must be in 3NF.

The general definition of 3NF states that- whenever a non-trivial functional dependency X→Y holds then it must satisfy either of the following conditions-

i. X is a super key in the relation.
ii. Y is a prime attribute.

In BCNF the condition (ii) is eliminated. This implies that a relation is in BCNF if, for each functional dependency X→Y, X is a super key of R.

Example: To better understand the concept let's consider the relation in table 6.11(a).

Space for learners:

Table 6.11(a): Example of a relation which is not in BCNF

S_ID	COURSE	INSTRUCTOR
101	Java	Rupam
101	DBMS	Priya
103	DBMS	Trisha
104	Python	Ashmita
105	Java	Kalyan

The constraints in the above table are-

- A student can enrol in multiple courses
- For each course, an instructor is assigned to the student.
- A course can be thought by multiple instructors.

The primary key in this relation is {S_ID, COURSE}, as it uniquely determines the INSTRUCTOR. Another point to be noted is that the course is dependent on the instructor as one instructor can teach only one subject. Thus the functional dependencies are-

i. {S_ID, COURSE} →INSTRUCTOR

ii. INSTRUCTOR → COURSE

The table is in 3NF as both the FDs either the right-hand side is a key or the left-hand side is a prime attribute. However, it is not in BCNF as in the FD (ii) INSTRUCTOR is not a prime attribute. To covert the relation to BCNF, we may decompose the relation into two other relations as shown in table 6.11(b) and 6.11(c).

Table 6.11(b): Student_instructor table

S_ID	INSTRUCTOR
101	Rupam
101	Priya
103	Trisha
104	Ashmita
105	Kalyan

Space for learners:

Table 6.11(c): Instructor_course table

INSTRUCTOR	**COURSE**
Rupam	Java
Priya	DBMS
Trisha	DBMS
Ashmita	Python
Kalyan	Java

Consider another example. The relation in table 6.11(d) stores employee details- id, name, pan number, and age. The candidate keys for this relation are- E_ID and PAN_NO. From these E_ID has been chosen as the primary key. Following are some of the dependencies that exist in the relation-

iii. E_ID→ E_NAME
iv. E_ID→PAN_NO
v. PAN_NO→ AGE
vi. PAN_NO→E_ID

As we can see that in all the FDs the left-hand side is a candidate key, so the relation is in BCNF.

Table 6.11(d): Example of a relation which is in BCNF

E_ID	**E_NAME**	**PAN_NO**	**AGE**
1	Xavier Mavely	ABCFX985B	48
2	Manish Pandey	PQRET654X	34
3	Rakesh Sharma	GFRTE4563B	54
4	Shilpi Molohtra	FDREU004T	43

Space for learners:

CHECKYOURPROGRESS-IV

12. A legal relational schema by default is in _____ normal form.
13. If a relation is in 2NF, then non no-prime attribute can be _____ dependent on the key.
14. For a relation R to be in BCNF, if the functional dependency A → B holds in R then A must be a _____.
15. In a relation, which is in 3NF, no non-prime attribute is _____ dependent on the primary key.
16. State true or false
    a. BCNF is stricter than 3NF.
    b. Normalization is a tool to minimize NULL values.

Space for learners:

6.6 MULTIVALUED DEPENDENCY AND FOURTH NORMAL FORM

In section 6.3.1.3, we introduced the concept of multivalued dependency. In this section, we will present an elaborate discussion on multivalued attributes and the fourth normal form (4NF).

6.6.1 Formal Definition of Multivalued Dependency

Lets X and Y be two attributes in a legal relation and let t1 and t2 be any two legal tuples in that relation such that-

t1(X)=t2(X) .

The multivalued dependency X→→ Y holds in the relation, if there exists another two tuples t3 and t4 with the following conditions-

t1(X)=t2(X)=t3(X)=t4(X)
t1(Y)=t3(Y)
t2(Y)=t4(Y)

Example: Consider the relation in table 6.12. We may observe that the students Ravi and Rahul have interests in multiple indoor and outdoor games. In the first four tuples, S_Name is the same, i.e,

Ravi. As the tuple (Ravi, Badminton, Cricket) and (Ravi, Table Tennis, Football) exist in the relation, another two tuples (Ravi, Badminton, Football) and (Ravi, Table Tennis, Cricket) also exist in the same relation. Sam can be observed from the last four tuples as well. Thus, we can say that-

S_Name→→ IndoorGame

S_Name→→ OutdoorGame

Table 6.12: Example of multivalued dependency

S_Name	**IndoorGame**	**OutdoorGame**
Ravi	Badminton	Cricket
Ravi	Table Tennis	Cricket
Ravi	Badminton	Football
Ravi	Table Tennis	Football
Rahul	Chess	Cricket
Rahul	Volleyball	Football
Rahul	Chess	Football
Rahul	Volleyball	Cricket

6.6.2 Fourth Normal Form

The definition of the fourth normal form is based on multivalued dependency. For a relation to be in 4NF, it must be in BCNF and should not have any multivalued dependency. The relation in table 6.12 is in BCNF but not in 4NF as it contains multivalued dependencies. To transform the said relation into 4NF, we may decompose it into tables 6.13(a) and 6.13(b)

Space for learners:

Space for learners:

Table 6.13(a): Student_Indoor Games relation

S_Name	**IndoorGame**
Ravi	Badminton
Ravi	Table Tennis
Rahul	Chess
Rahul	Volleyball

Table 6.13(b): Student_Outdoor Games relation

S_Name	**OutdoorGame**
Ravi	Cricket
Rahul	Cricket
Ravi	Football
Rahul	Football

Both the relations in table 6.13(a) and 6.13(b) are in 4 NF as there is no multivalued dependency.

6.7 RELATIONAL DECOMPOSITION AND ITS PROPERTIES

In the earlier sections, we have discussed the normal forms- 1NF, 2NF, 3NF, BCNF, and 4NF. In all the cases we have seen a relation can be upgraded to a higher normal form by decomposing it into multiple relations. Decomposition helps in removing redundancies and inconsistencies.

While decomposing a relational schema into multiple relational schemas one must make sure that the decomposition preserves all the original attributes. If a relational schema, R, is decomposed into multiple relations D={R1, R2, R3,……, Rn}, then each attribute in R must appear in at least one relation Ri in D. This property is called the ***attribute preserving*** property of decomposition. Another additional aim of decomposition is that each relation Ri in D must

be at least in either 3NF or BCNF. Unfortunately, these two properties alone don't guarantee a good database design. In the following sections, we discuss some additional criteria that must hold in a decomposition.

Space for learners:

6.7.1 Dependency Preservation Property of a Decomposition

Let's consider the decomposition of the relational schema R into D as discussed above. The dependency preserving property of a decomposition states that- every functional dependency X→Y specified in R, must appear either directly in one of the relations Ri ∈ D or can be inferred from other dependencies specified in some relation Ri ∈ D. Let F be the set of FDs specified in R. Let F1, F2… Fn is the set of functional dependencies specified in R1, R2…. Ri respectively. The decomposition D is said to be dependency preserving if-

$$(F1 \cup F2 \ldots \cup Fn)^{+}=F^{+}$$

Example: Let's consider a relation R (W, X, Y, Z). The specified set of functional dependencies for this relation is F= {WX → Y, Y → Z, Z → W}. R is decomposed into two relations - R1(W, X, Y) and R2(Y, Z). Let F1 and F2 be the set of functional dependencies for R1 and R2 respectively.First, we will find the closure of F1. To do so, we will consider the combinations- W, X, Y, WX, XY, and XY.

W^{+} = { W } // Trivial

X^{+} = { X } // Trivial

Y^{+}= {Y, W, Z} = {Y, W} [As Z is not in R1, it has been removed from the closure]

Y→ W [Removing Y from right side as it is trivial attribute]

WX^{+} = {W, X, Y, Z} [As Z is not in R1, it has been removed from the closure]

= {W, X, Y}

$WX \rightarrow Y$ [Removing WX from right-side as these are trivial attributes]

$XY^+ = \{X, Y, Z, W\}$

$= \{W, X, Y\}$

$XY \rightarrow W$ [Removing XY from right side as these are trivial attributes]

$WY^+ = \{W, Y, Z\}$

$WY \rightarrow Z$ [Removing WY from right side as these are trivial attributes]

Thus, F1 $=\{Y \rightarrow W, WX \rightarrow Y, XY \rightarrow W\}$

Similarly, F2= $\{ Y \rightarrow Z \}$

In the original relation R,

$F=\{ WX \rightarrow Y, Y \rightarrow Z, Z \rightarrow W\}$

$WX \rightarrow Y$ is present in F1.

$Y \rightarrow Z$ is present in F2.

But, $Z \rightarrow W$ is not preserved.

F1 U F2 is a subset of F. So, the given decomposition is not dependency preserving.

6.7.2 Lossless (Non-Additive) Join Property of a Decomposition

Another important property that a decomposition should satisfy is the lossless or non-additive join property. This ensures that if we reconstruct the relation R by performing natural join (*) on R1, R2 … and Rn, then it should not produce any spurious tuples. To put it in another way, the decomposition is -

- lossy if R1*R2*…..*Rn $\supset$ R
- Lossless if R1*R2*…..*Rn = R

The problem of spurious tuples has already been discussed in section 6.2.4.

To illustrate this concept lets consider the following relational schema in table 6.14(a).

Space for learners:

Table 6.14(a): Employee_Department relation

E_ID	E_NAME	E_AGE	DEPT_ID	DEPT_NAME
1	Xavier	42	1	Sales
2	Pallavi	34	1	Sales
3	Arun	42	2	Marketing
4	Susane	28	3	HR

If we decompose this relation into two smaller schemas R1(E_ID, E_NAME, E_AGE, DEPT_ID) and R2(DEPT_ID, D_NAME), then following tables 6.14(b) and 6.14(c) will be the result of the decomposition.

Table 6.14(b): Decomposition of Employee_Department relation to relation R1

E_ID	E_NAME	E_AGE	DEPT_ID
1	Xavier	42	1
2	Pallavi	34	1
3	Arun	45	2
4	Susane	28	3

Table 6.14(c): Decomposition of Employee_Department relation to relation R2

DEPT_ID	DEPT_NAME
1	Sales
2	Marketing
3	HR

If we perform natural join over R1 and R2 then we will get back exactly the tuples present in the relation R. Neither will any extra tuple be generated nor will there be any missing tuple. Thus this decomposition is lossless.

Space for learners:

However, if the same relation R is decomposed into two other relations- R3(E_ID, E_NAME, E_AGE) and R4(DEPT_ID, D_NAME, E_AGE), then this would result in the following tables 6.14(d) and 6.14(e).

Table 6.14(d): Decomposition of Employee_Department relation to relation R3

E_ID	E_NAME	E_AGE
1	Xavier	42
2	Pallavi	34
3	Arun	42
4	Susane	28

Table 6.14(e): Decomposition of Employee_Department relation to relation R4

DEPT_ID	E_AGE	DEPT_NAME
1	42	Sales
1	34	Sales
2	42	Marketing
3	28	HR

The natural join of R3 and R4 (R3*R4) would result in table 6.14(f).

Table 6.14(f): Natural join of R3 and R4

E_ID	E_NAME	E_AGE	DEPT_ID	DEPT_NAME
1	Xavier	42	1	Sales
1	Xavier	42	2	Marketing
2	Pallavi	34	1	Sales
3	Arun	42	2	Marketing
4	Susane	28	3	HR

Space for learners:

As we can see that the natural join of R3 and R4 results in extra information that does not exist in the original relation R. Thus, the decomposition of R into R3 and R4 is a lossy join.

CHECKYOURPROGRESS-V

17. For multivalued dependency to occur in a relation, it must have at least ____ attributes.
18. A decomposition is loss-less if natural join of the relations in the decomposition does not produce any _____ tuple.
19. 4NF is based on _____ dependency.
20. _____ states that each attribute of the original relation must appear in at least one of the relation in its's decomposition.

6.8 ALGORITHMS FOR RELATIONAL DATABASE SCHEMA

In this section, we present some algorithms related to the decomposition of a relational schema.

6.8.1 Relational Synthesis

Algorithm 6.2: Relational Synthesis into 3NF with Dependency Preservation

Input: A universal relation R with a set of a functional dependency F

Step 1: Find the minimal cover G of F.

Step 2: For each left-hand side of X of a functional dependency in G, construct a relational schema with attributes

$\{X \cup A1 \cup A2 \ldots. \cup Ak\}$ with X as key, where $X \rightarrow A1$, $X \rightarrow A2 \ldots$

$X \rightarrow Ak$.

Space for learners:

Step3: Place the remaining attributes (which could not be placed in any relation in step2) in single relation.

Claim: All the relational schemas created by algorithm 6.2 are in 3NF.

6.8.2 Testing Lossless Join Property

Algorithm 6.3: Testing for lossless or non-additive join property
Input:A universal relation R, R's decomposition D = {R1, R2, . . ., Rm}, and a set of functional dependencies F.
Step1: Create an initial matrix *S* with dimension 'm' rows and 'n' columns, where 'm' is the number of relations in D and 'n' is the number of attributes in R.
Step2: Set each *S(i,j)* in the matrix to b_{ij}, where b_{ij} is a distinct symbol associated with S(i,j).
Step 3: For each row *i* in *S*:

For each column *j* in *S*:

If Ri contains attribute a_j then

set *S(i,j)*=a_j

Step 4: Repeat the following loop until S remains unchanged after a complete loop execution: For each $A \rightarrow B$ in F:

For each row *i* in S, having the same symbol in the column corresponding to attribute for A:

Set the symbols in each column corresponding the attribute B to be the same. If there exists an 'a' symbol for any of these columns, set all other columns to symbols 'a', else chose any of the 'b' symbols that appear for any of these columns and update the symbols in the rest of the columns in all such rows to 'b'.

Step5: The decomposition has lossless join property only if there exists a row in S that contains only 'a' symbol. Otherwise, the decomposition is lossy.

Space for learners:

Example: Lets consider a relation R = {E_ID, E_NAME, P_NO, P_NAME, P_LOC, HOURS} with the following set of functional dependency-

F = {E_ID → E_NAME, P_NO → {P_NAME, P_LOC}, {E_ID, P_NO} → HOURS}

Let D = {R1, R2, R3} be the decomposition the relation, where,

R1 = {E_ID, E_NAME}

R2 = {P_NO, P_NAME, P_LOC}

R3 = {E_ID, P_NO, HOURS}

Application of steps 1,2 and 3 results in the matrix in table 6.15(a).

Table 6.15(a): Example for testing loss-less join property

	E_ID	E_NAME	P_NO	P_NAME	P_LOC	HOURS
R1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

Now, for the functional dependency E_ID → E_NAME, the E_ID attribute in R1 and R3 have the same symbol. So, the symbol for the E_NAME attribute in R3 will be updated to a_2 as R1 has a_2 for E_NAME. This results in the matrix in table 6.15(b).

Table 6.15(b): Example for testing loss-less join property

	E_ID	E_NAME	P_NO	P_NAME	P_LOC	HOURS
R1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	a_2	a_3	b_{34}	b_{35}	a_6

Similarly, for the functional dependency P_NO → {P_NAME, P_LOC}, in the values for attributes, P_NAME and P_LOC are updated to a_4 and a_5 respectively as R2 and R3 have the same symbol for P_NO. Thus the updated matrix will be as shown in table 6.15(c).

Space for learners:

Table 6.15(c): Example for testing loss-less join property

	E_ID	E_NAME	P_NO	P_NAME	P_LOC	HOURS
R1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R3	a_1	a_2	a_3	a_4	a_5	a_6

We can see observe from the above matrix that the row R3 has all 'a' symbols. Thus, the decomposition D has lossless join property.

6.8.3 Testing Lossless Join Propertyin Binary Decomposition (Property LJ1)

Breaking down a relation into two relations is called binary decomposition. The following property helps to test for lossless join property in binary decomposition-

- ***Property LJ1:*** The binary decomposition D = {R1, R2} of a relation R has the lossless join property with respect to a set of functional dependencies F on R *if and only if* either
  - $((R1 \cap R2) \rightarrow (R1 - R2))$ is in F^+, or
  - $((R1 \cap R2) \rightarrow (R2 - R1))$ is in F^+.

6.8.4 Successive Lossless Join Decomposition (PROPERTY LJ2)

Let a decomposition D={R1, R2…. Rm} of a relation R, concerning a set of functional dependency F, has lossless join property. Now, let's divide a relation Ri in D to smaller relations Q={Q1, Q2…Qp} in such a way that Q also has lossless join property for F. If we now replace Ri by Q in D, then ***property LJ2*** states that a set of decomposition D1={R1, R2,…Ri-1,Q1,Q2,…Qp, …, Rm}) will also have lossless join property.

Space for learners:

6.8.5 Non-additive Join Decomposition into BCNF Schemas

Algorithm 6.4: Relational decomposition into BCNF relations with lossless join property

Input: A universal relation R with F as set of specified functional dependencies .

Step1: Set D={R}

Step2: For each relation Q in D, which is not in BCNF:

Identify the functional dependency A→B, that violates the BCNF.

Replace Q in D by two relations (Q − B) and (A ∪ B)

Step3: Stop

Explanation: Since (Q − B) ∩ (A ∪ B) → (A ∪ B) − (Q − B) is equivalent to A → B ∈ F^+. By virtue of property LJ1, the decomposition is lossless.

Example:R = {X, Y, Z} F = {XY → Z, Z → Y}
Let D= {{X, Y, Z}};
{X, Y, Z} in D is not in BCNF due to the functional dependency Z → Y.
Thus, decompose {X, Y, Z} to ({X, Y, Z} − Y) and (X ∪ Y), i.e. to {X, Z} and {X,Y}.
Replace {X, Y, Z} in D by {X, Z} and {X,Y}.
D={{X, Z},{X,Y}}
{X, Z} and {X,Y} both are now in BCNF.

6.8.6 Relational synthesis algorithm into 3NF with dpendency preservation and lossless join property

Space for learners:

Algorithm 6.5: Relational synthesis algorithm into 3NF with dependency preservation and lossless join property

Input: A universal relation R and a set of FDs F

Step 1: Compute a minimal cover G for F

Step 2: Construct a relational schema in D with attributes {X ∪ A1 ∪ A2…. ∪ Ak} with X as key, for each left-hand side of X of a functional dependency in G. The functional dependencies: X→A1, X→A2… X→Ak, should be the only dependencies in G with X on the left-hand side.

Step 3: If there is no relation in D that contains a key of R, then create one with the attributes of a key in R.

Example: R = {A, B, C, D, E, H} is a relation with functional dependencies F = {AE → BC, B → AD, CD → E, E → CD, A → E}.

Step 1: The minimal cover of F is G = {A → B, A → E, B → A, CD → E, E → CD} [Derived using the algorithm 6.1]

Step 2: Based on the functional dependencies in G the R will be decomposed into D={R1, R2, R3, R4, R5}, with the set of functional dependencies F1, F2, F3, F4 and F5 respectively, where

- R1 = {A, B, E} with F1 = {A → B, A → E}
- R2 = {B, A} with F2 = {B → A}
- R3 = {C, D, E} with F3 = {CD → E}
- R4 = {E, C, D} with F4 = {E → CD}

Combine R3 and R4 into one relation schema R5 = {C, D, E} and F5 = {CD → E, E → CD}. So, D={R1, R2, R5}

Step 3: In R, AH, and BH are candidate keys. As we can see that neither of these appears as key in any of the relations in D. So, we create another relational schema R6 = {A, H} and F = {}

Now, all the relations in D = {R1, R2, R5, R6} are in 3NF.

Space for learners:

6.8.7 Finding a key *K* for relation schema *R* based on a set *F* of functional dependencies

Algorithm 6.6: Finding a Key for a relation schema based on a set of functional dependencies.

Input: A relational schema R(A1, A2,…, Am)
Step 1: Set the key K={A1, A2,…, Am}
Step 2:For each attribute Ai in K:

Determine $(K-Ai)^+$ with respect to F. If $(K-Ai)^+$ contains all the attributes in R, then set K=K-{Ai}

Example: Lets consider the relation R={A, B, C, D} with F={A → BCD, C → A}

- A^+=ABCD
- B^+=B
- C^+=ABCD
- D^+=D

So, the candidate keys are A and C as the closure of A and C contains all the attributes of R.

6.8.8 Relational decomposition into 4NF relations with lossless join property

Whenever a relational schema R is decomposed into D={R_1, R_2} based on multivalued dependency A→→B that holds in R, then property LJ1' presents the necessary and sufficient condition to check whether the decomposition is lossless or not.

- **PROPERTY LJ1'**
  - The decomposition D={R_1, R_2} of R, is a lossless (non-additive) join decomposition with respect to a set F of functional *and* multivalued dependencies if and only if
    - $(R_1 \cap R_2) \rightarrow\rightarrow (R_1 - R_2)$
  - or by symmetry, if and only if
    - $(R_1 \cap R_2) \rightarrow\rightarrow (R_2 - R_1)$).

Algorithm 6.7: Decomposition of a relation into 4NF relations with lossless join property
Input:A universal relation R and a set of functional and multi-valued dependencies F.
Step 1: Set D := { R };
Step 2:While there exists a relation *Ri* in *D* that is not in 4NF do {

Space for learners:

choose a relation schema *Ri* in *D* that is not in 4NF;
find a nontrivial MVD A →→ B in Ri that violates 4NF;
replace *Ri* in *D* by two relation schemas (*Ri* - *B*) and (*A* υ *B*);
};

6.9 SUMMING UP

- A relational schema can be defined as a set of relational tables and associated items related to each other.
- Following are the four informal design guidelines-
  - The semantics of the Relation
  - Minimizing redundancy
  - Reduction of the null values in tuples.
  - Discarding the possibility of generating spurious tuples.
- Redundancy is the repetition of the same fact again and again across multiples places in the same database.
- Apart from wastage of storage space, redundancy leads to another serious issue of update anomalies. Insertion, deletion, and modification anomalies are the three categories of update anomalies.
- Spurious tuples represent wrong or invalid information and thus leads to the inconsistency of the database.
- In Database Management System (DBMS), functional dependency (FD) refers to the relationship between two attributes in a table or relation.
- A functional dependency X→Y is said to be trivial if Y is a subset of X.
- If X→Y, and Y is not a subset of X, then the functional dependency is said to be non-trivial.
- If there exists a functional dependency of the form X→ {Y, Z} such that there is no dependency between Y and Z, then the FD is said to be a multivalued functional dependency.
- In a relation, if the functional dependencies X→Y and Y→ Z exist, then the functional dependency X→Z also exists.

Space for learners:

- A functional dependency X→Y is said to be a full functional dependency if removal of any attribute from X means the functional dependency doesn't exist any longer.
- The set of inference rules were first introduced by William W. Armstrong in 1974. These rules are thus also called Armstrong's axioms. These axioms define a set of rules which, if applied repeatedly, generate all the other functional dependencies that can be inferred from a set of functional dependencies originally specified by the designer.
- For any relational schema R, if the set of functional dependencies is specified as F, then the set of all the functional dependencies that can be inferred from F, is called the closure of F. The closure of F is denoted as F^+.
- Let F and G be two sets of functional dependencies for a relational schema R. G is said to be covered by F if all the dependencies in G can be inferred from F.
- Normalization helps in breaking down big relations into smaller relations and ensures that data is stored logically with minimal redundancy.
- The first normal states that each attribute in a relation must have atomic values.
- A relation is in second normal form it is already in 1NF and all the non-prime attributes in the relation are fully functionally dependent on the prime key.
- To be in 3NF, a relation must also be in 2NF, and no non-prime attribute should be transitively dependent on the primary key.
- For a relation to be in 4NF, it must be in BCNF and should not have any multivalued dependency.

Space for learners:

6.10 ANSWERS TO CHECK YOUR PROGRESS

1. Insertion, deletion and modification
2. Spurious
3. 3.a.False

3.b.True

4. Trivial
5. Transitivity
6. A→BC
7.
   7.a. False
   7.b.True
   7.c.True
8. Equivalent
9. G^+
10. {A, B, C, D}
11.
   11.a.True
   11.b.False
12. 1NF
13. Partially
14. Superkey
15. Transitively
16.
   16.a. True
   16.b.False
17. 3
18. Spurious
19. Multivalued
20. Attribute preservation property

6.11POSSIBLE QUESTIONS

1. Define closure of a set of functional dependencies.
2. Write down the properties LJ1 and LJ2.
3. State when two sets of functional dependencies are considered to be equivalent.
4. What are spurious tuples? Why are they considered as bad?
5. What are the problems with null values in a relation?
6. State the condition a relation must satisfy to be in 2NF.
7. List the conditions a binary decomposition must satisfy to be loss-less.

Space for learners:

8. Write the condition for a relation to be in BCNF.
9. Discuss the four informal guidelines for designing a good relation.
10. Discuss the problem of update anomalies in a relation with appropriate examples.
11. What is normalization? Why is it important? Discuss, with an example, how a relation which is not in 1NF can be converted to 1NF.
12. Define functional dependency. Briefly discuss the types of functional dependencies. Give one example each.
13. Write down the Armstrong's axioms for functional dependency. Why are these rules important?
14. Discuss the 1st, 2nd and 3rd normal forms with suitable examples.
15. What is multivalued functional dependency? Discuss the fourth normal forms with an example.
16. Discuss the attribute preservation and dependency preservation properties of a relational decomposition. Write the algorithm to decompose a relation to smaller relations where each smaller relation is in 3NF and dependency is also preserved.
17. What is loss-less join property of a relation? Write an algorithm to test the loss-less join property of a decomposition.

Space for learners:

6.12REFERENCES AND SUGGESTED READINGS

- Ramez, Elmasri. *Fundamentals of Database Systems*. Pearson Education India, 2020.
- Silberschatz, Abraham, Henry F. Korth, and Shashank Sudarshan. *Database system concepts*. McGraw-Hill, 1997.