GCA S3 02

Database Management Systems

SEMESTER - III

BACHELOR OF COMPUTER APPLICATION

BLOCK 2



KRISHNA KANTA HANDIQUI STATE OPEN UNIVERSITY

Subject Experts

Prof. Anjana Kakati Mahanta,	Gauhati University
Prof.(Retd.) Pranhari Talukdar,	Gauhati University
Dr. Jyotiprakash Goswami,	Assam Engineering College

Course Coordinator

Dr. Sanjib Kr. Kalita,	KKHSOU
Dr. Tapashi Kashyap Das,	KKHSOU
Sruti Sruba Bharali,	KKHSOU

SLM Preparation Team

UNITS	CONTRIBUTORS
7	Arabinda Saikia, KKHSOU
8, 9 & 10	Sruti Sruba Bharali, KKHSOU
11 & 12	Chandan Jyoti Kumar, Cotton College State University

Editorial Team

Content	:	Dr. Tapashi Kashyap Das, KKHSOU
Language	:	Prof. Robin Goswami
Structure, Format & Graphics	:	Dr. Tapashi Kashyap Das, KKHSOU

Jun 2018

This Self Learning Material (SLM) of the Krishna Kanta Handiqui State Open University is made available under a Creative Commons Attribution-Non Commercial-Share Alike 4.0 License (international): http://creativecommons.org/licenses/by-nc-sa/4.0/

Printed and published by Registrar on behalf of the Krishna Kanta Handiqui State Open University

Headquarters : Patgaon, Rani Gate, Guwahati - 781017

Housefed Complex, Dispur, Guwahati-781006; Web: www.kkhsou.in

The University acknowledges with thanks the financial support provided by the **Distance Education Bureau**, **UGC** for the preparation of this study material.

BLOCK INTRODUCTION

This is the *Second Block* of the course '***Database Management Systems***'. This block describes a most important topic *Normaization*. Besides introducing Structured Query Language(SQL) Database Recovery and Security is also discussed in this block.

Unit - 7 discusses Normalization. The basic objectives of normalization is to reduce redundancy which means that information is to be stored only once. With this unit you will be able to normalize tables or relations upto different normal forms like 1NF, 2NF, 3NF etc.

Unit - 8 introduces Structured Query Language(SQL). SQL characteristics, SQL data types and different SQL commands are discussed in this unit.

Unit -9 discusses how to enter information through SQL. Inserting new data, updating data through SQL is discussed in this unit.

Unit -10 extracting information through SQL is discussed in this unit. Various important statements and clause like SELECT, WHERE, ORDER BY, BETWEEN etc. are discussed in this unit.

Unit -11 is on Database Recovery. In this unit, various important issues like database failure and its types, backup of database is covered.

Unit -12 is on Database Security. Discretionary Access Control, Mandatory Access control, Statistical Databases, Data Encryption are discussed in this unit.

Units of Block 2 are as follows:

Unit –7 : Normalization

Unit –8 : Introduction to SQL

Unit –9 : Entering Information through SQL

Unit –10 : Extracting Information through SQL

Unit –11 : Database Recovery

Unit –12 : Database Security

BACHELOR OF COMPUTER APPLICATION

Database Management Systems

DETAILED SYLLABUS

BLOCK-2

Semester 3 **Page No.**

UNIT – 7: Normalization 5-23

Normalization, First Normal form (1NF), Second Normal form (2NF), Third Normal form (3NF), Boyce-Codd Normal form (BCNF)

UNIT – 8: Introduction to SQL 24-40

Structured Query Language (SQL), Characteristics of SQL, Advantages of SQL, SQL Data types, Types of SQL commands, DDL, DML, Creating a Database, Creating, Altering, and Deleting Tables,

UNIT -9: Entering Information through SQL 41-62

Inserting new data, Updating data (the WHERE Clause, The Logical AND and OR operator), Deleting Data.

UNIT -10: Extracting Information through SQL 63-96

The SELECT statement, Filtering results with the WHERE Clause; Logical Operators and Operator Precedence (Using logical operator NOT, BETWEEN, LIKE, IN); Ordering Results with ORDER BY(Joining columns-concatenation); Selecting data from more than one table, Join, Types of Join, SQL commands: Select… From…Where… Group by ….. Having… Order by…, Tables, Views and Indexes, Queries, Sub Queries, Insert, Update and Delete operations, Constraints considers (NOT NULL, UNIQUE, Check Primary key, Foreign key)

UNIT – 11: Database Recovery 97-105

Concept of database recovery, Backup of database, Types of database failure, Types of database recovery.

UNIT -12: Database Security 106-113

Goals of database security, Discretionary Access Control, Mandatory Access control, Statistical Databases, Data Encryption.

UNIT 7 : NORMALIZATION

UNIT STRUCTURE

7.1 Learning Objectives
7.2 Introduction
7.3 Normalization and Its Objectives
7.4 Normal Forms
    7.4.1 First Normal Form (1NF)
    7.4.2 Second Normal Form (2NF)
    7.4.3 Third Normal Form (3NF)
7.5 Boyce-Codd Normal Form (BCNF)
7.6 Fourth Normal Form (4NF)
7.7 Fifth Normal Form (5NF)
7.8 Let Us Sum Up
7.9 Further Reading
7.10 Answers to Check Your Progress
7.11 Model Questions

7.1 LEARNING OBJECTIVES

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

- understand what normalization is.
- list out the objectives of Normalizations
- describe 1st, 2nd and 3rd Normal forms
- illustrate fourth and fifth Normal forms

7.2 INTRODUCTION

In the previous unit, we have learnt two important concepts in database design i.e., Functional Dependency and Decomposition which usually help in minimizing the redundancy in database design. In this unit we will discuss Normalization which is one of the most important concept in database.

Normalization is the process of efficiently organizing data in a database. Normalization is the name given to the process of simplifying the relationship among data elements in a record. In this unit, we will discuss

different normal forms. The learners will be able to normalize database tables into different normal forms after completing this unit.

7.3 NORMALIZATION AND ITS OBJECTIVES

Normalization is a process of decomposing a set of relations(table) with anomalies to produce smaller and well structured relations that contain minimum or no redundancy. The basic objectives of normalization are to reduce redundancy, which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored. Normalization provides the designer with a systematic and scientific process of grouping of attributes in a relation.

The normalization process can be defined as a procedure of analyzing and successive reduction of the given relational schemas based on their FDs and primary keys to achieve the desirable properties of -

(i) minimizing redundancy,

(ii) minimizing insertion, deletion and update anomalies.

The process of normalization was first proposed by ***E.F. Codd***. Normalization is a bottom up design technique for relational database.

The objectives of the normalization process are :

- To create a formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- To free relations from undesirable insertion, update and deletion anomalies.
- To reduce the need for restructuring the relations as new data types are introduced.
- To carry out series of tests on individual relation schema so that the relational database can be normalized to some degree. When a test

fails, the relation violating that test must be decomposed into relations that individually meet the normalization tests.

7.4 NORMAL FORMS

Whenever the simple rules of Functional Dependencies are applied to a relation, it transforms the relation into a state which called *normal form.* The normal forms are used to ensure that various types of anomalies and inconsistencies are not introduced into the database. Various types of normal forms are used in relational database. They are :

- First normal form (1 NF)
- Second normal form (2 NF)
- Third normal form (3 NF)
- Boyce/Codd normal form (BCNF)
- Fourth normal form (4 NF)
- Fifth normal form (5NF)

7.4.1 First Normal Form (1NF)

A relation is said to be in First normal form if the values in the domain of each attribute of the relation are atomic. The first normal form prohibits multivalued attributes, composite attributes and their combinations. It means that, 1NF disallows having a set of values, a tuple of values, or combination of both as an attribute value for a single tuple.

Let us consider the relation TRAVEL_INFO as shown in Table 7.1.

Table 7.1: Un-normalized relation TRAVEL_INFO

PERSON	VISITED_CITY	
HEMANGA	CITY	JOURNEY_DATE
	DELHI	04-07-02
	MUMBAI	15-09-03
	AGRA	02-03-04
BIKASH	CITY	JOURNEY_DATE
	CHENNAI	05-01-02
	KOLKATA	13-02-03
	PUNE	21-08-04

Here, in the relation the domain VISITED_CITY is not very simple. and in un-normalized form. Now, let us combine the respective rows in VISITED_CITY with the value of the attribue PERSON and the resultant relation will be as follows:

Table 7.2 Modified relation TRAVEL_INFO

PERSON	CITY	JOURNEY_DATE
HEMANGA	DELHI	04-07-02
HEMANGA	MUMBAI	15-09-03
HEMANGA	AGRA	02-03-04
BIKASH	CHENNAI	05-01-02
BIKASH	KOLKATA	13-02-03
BIKASH	PUNE	21-08-04

Let us consider another relation PATIENT_DOCTOR, which keeps the records of appointment details of patient to doctors. This relation is in first normal form (1NF). The relational table can be depicted as:

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE_TIME, PHONE, DURATION)

Table 7.3 Relation PATIENT_DOCTOR

P__NAME	DOB	D__NAME	DATE__TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	06-02-05 16:00	15

From the relational table we have observed that a doctor cannot have two simultaneous appointments so D_NAME and DATE_TIME is a compolsite key. Similarly, a patient cannot have same time from two different doctors. Therefore, P_NAME and DATE_TIME attributes are also a candidate key.

Problems in 1 NF :

1NF contains redundant information. In our example, PATIENT_DOCTOR relation has the following errors :

a) There exists redundant information in patients date of birth and phone number.

b) A doctor, who does not currently have an appointment with a patient, cannot be represented.

c) A patient, who does not currently have an appointment with a doctor, cannot be represented.

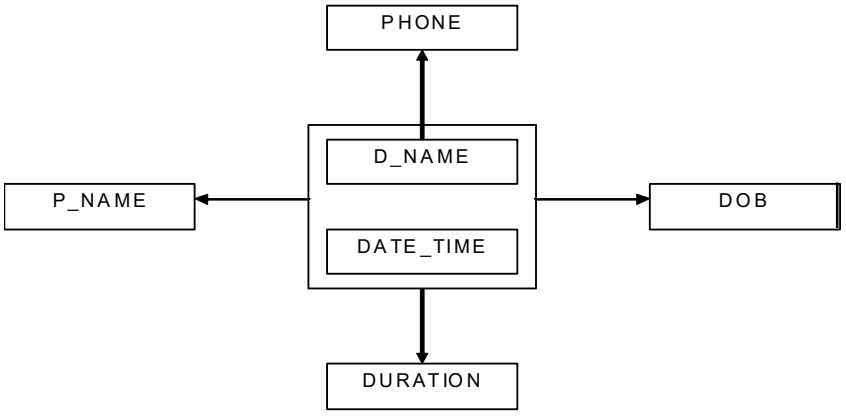


Fig. 7.1: FDD of relation PATIENT_DOCTOR

Functional dependancy diagram(FDD) of the relation PATIENT_DOCTOR is shown. Here, P_NAME, DURATION, DOB are dependent on the key D_NAME & DATE_TIME and doctor's contact number i.e., PHONE is only dependent on the D_NAME.

7.4.2 Second Normal Form (2NF)

A relation or table is said to be in Second normal form (2NF) if and only if -

a) it is in 1NF,

b) each non-primary key attribute is fully functionally dependent on primary key of that relation.

2NF is an intermediate step towards higher normal forms. 2NF is based on the concept of *full functional dependency*. It eliminates the problems of 1NF.

So, we come to know that no attributes of the relation (or table) should be functionally dependent on only one part of a concatenated primary key. In our example, we have seen from the functional dependancy diagram that PHONE is partiallly dependent only on D_NAME for which the relation is not in 2NF.

Therefore, to bring the relation into 2NF, the information about doctor and their contact numbers have to be separated from the information about patient and their appointments with doctors. Thus, the relation is decomposed into two tables, namely PATIENT_DOCTOR and DOCTOR as shown below. The relational table can be depicted as :

PATIENT_DOCTOR (P_NAME, DOB, D_NAME, DATE_TIME, DURATION)

DOCTOR (D_NAME, PHONE)

Table 7.4: Relation PATIENT_DOCTOR

P__NAME	DOB	D__NAME	PHONE	DATE__TIME	DURATION
MUKUL	10-02-1975	Dr Deepak	0361-223470	10-01-05 10:00	15 (minutes)
RUHIT	27-01-1971	Dr Kalyan	0361-270512	10-01-05 11:00	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 10:30	10
JAMES	30-03-1977	Dr Dhiraj	0361-270615	10-01-05 09:00	10
RUHIT	27-01-1971	Dr Kalyan	0361-223470	10-01-05 10:15	15
MUKUL	10-02-1975	Dr Dhiraj	0361-270615	10-01-05 10:50	20
RANA	02-11-1980	Dr Kalyan	0361-270512	10-01-05 11:10	20
MUKUL	10-02-1975	Dr Deepak	0361-223470	06-02-05 16:00	15

Table 7.5 Relation DOCTOR

D__NAME	PHONE
Dr Deepak	0361-223470
Dr Kalyan	0361-270512
Dr Dhiraj	0361-270615
Dr Dhiraj	0361-270615
Dr Kalyan	0361-223470
Dr Dhiraj	0361-270615
Dr Kalyan	0361-270512
Dr Deepak	0361-223470

The functional dependency diagram of the above two relations are

shown below :

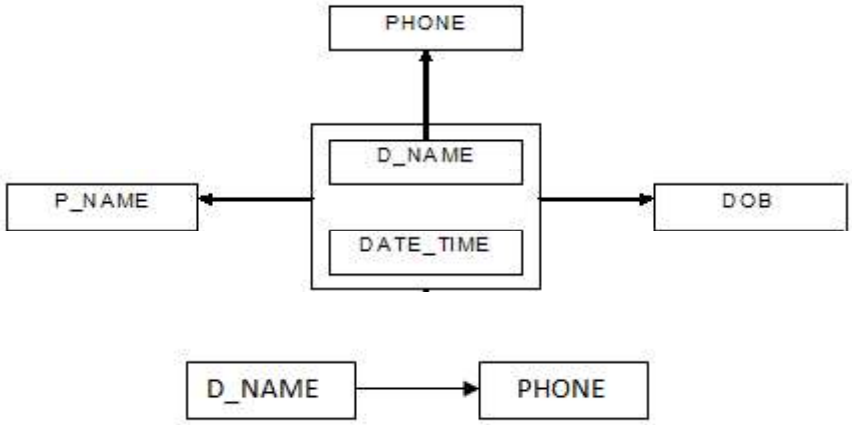


Fig. 7.2: FFD of relation PATIENT_DOCTOR

Problems in 2 NF :

a) In the relation (or table) PATIENT_DOCTOR, deleting a record from it may lose the patient's details.

b) Any changes made on the PATIENT_DOCTOR table may involve in changing multiple records because the information in the table is still redundant.

7.4.3 Third Normal Form (3NF)

A relation or table is said to be in Third normal form (3NF) if the relation is in 2NF and the non-prime attributes are -

a) mutually independent,

b) functionally dependent on the primary key.

It means that no attributes of the relation should be transitively functionally dependent on the primary key. Thus, in 3NF, no non-prime attribue is functionally dependent on another non-prime attribute. This means that *a relation in 3NF consists of the primary key and a set of independent nonprime attributes*. 3NF is based on the problem of transitive dependency. The 3NF eliminates the problem of 2NF.

In our example, in the relation PATIENT_DOCTOR (Table 7.4) , there is no dependency between the attributes P_NAME and DURATION. Again, P_NAME and DOB are mutually dependent. So, the relation is not in 3NF.

To bring the relation PATIENT_DOCTOR into 3NF, it has to be decomposed and has to remove the attributes that are not directly dependent on the primary key. Now, using the *transitive dependency,* DOB can be linked to the primary key, through its dependency on the P_NAME. The functional dependency diagram is as follows. Now, the new relations will be:

PATIENT_DOCTOR(P_NAME,D_NAME, DATE_TIME,DURATION)

DOCTOR (D_NAME, PHONE)

PATIENT (P_NAME, DOB)

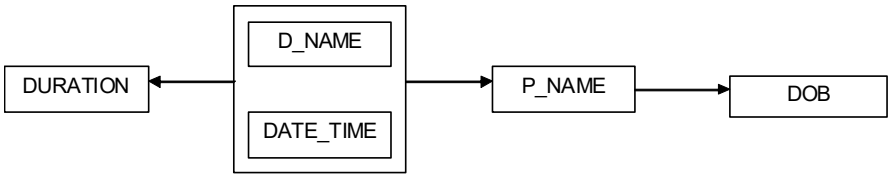


Fig. 7.3: FDD for relation PATIENT_DOCTOR

Now, let us consider another relation STUDENT as shown in Table 7.6

Table 7.6: STUDENT relation

STUDENT (Roll_No, Name, Dept, Year, Hostel_Name)

Roll_No	Name	Dept	Year	Hostel_Name
1784	Ruhit	Physics	1	Ganga
1648	Keshab	Chemistry	1	Ganga
1768	Gopal	Maths	2	Kaveri
1848	Raja	Botany	2	Kaveri
1682	Mrinal	Geology	3	Krishna
1485	Sumit	Zology	4	Godavari

Here, the dependency **Roll_No** $\rightarrow$ **Hostal_Name** is transitive through the following two dependencies :

Roll_No $\rightarrow$ **Year**

Year $\rightarrow$ **Hostal_Name**

Thus, the STUDENT relation is not in 3NF. To bring the relation into 3NF we can decompose the relation into two relation STUD1 and STUD2, as shown below:

STUD1 (Roll_No, Name, Dept)

STUD2 (Year, Hostel_Name)

Table 7.7 Relation STUD1

Roll_No	Name	Dept	Year
1784	Ruhit	Physics	1
1648	Keshab	Chemistry	1
1768	Gopal	Maths	2
1848	Raja	Botany	2
1682	Mrinal	Geology	3
1485	Sumit	Zology	4

Table 7.8 Relation STUD2

Year	Hostel_Name
1	Ganga
1	Ganga
2	Kaveri
2	Kaveri
3	Krishna
4	Godavari

In the above examples, the conversion into 3NF is not hard, but whenever a relation has more than one combination of attributes that

may be considered as primary key then the conversion takes a number of steps. Let us consider the following relation UTILIZE as shown in Table 7.9.

Table 7.9: Relation UTILIZE

Project	Proj_Manager	Machine	Qty_Used
P1	Thomas	Excavator	5
P3	John	Shovel	2
P2	Abhishek	Drilling	5
P4	Avinash	Dumper	10
P3	John	Welding	3
P1	Thomas	Drilling	4

The relation UTILIZE stores the machines information used by both projects and project managers. Each project has one project manager and each project manager manages one project. Now, it is obvious from the table that we can consider any one of the combination of attributes as primary key, namely, {Project, Machine} or {Proj_Manager, Machine}. The FDD for relation UTILIZE is shown Figure 7.4.

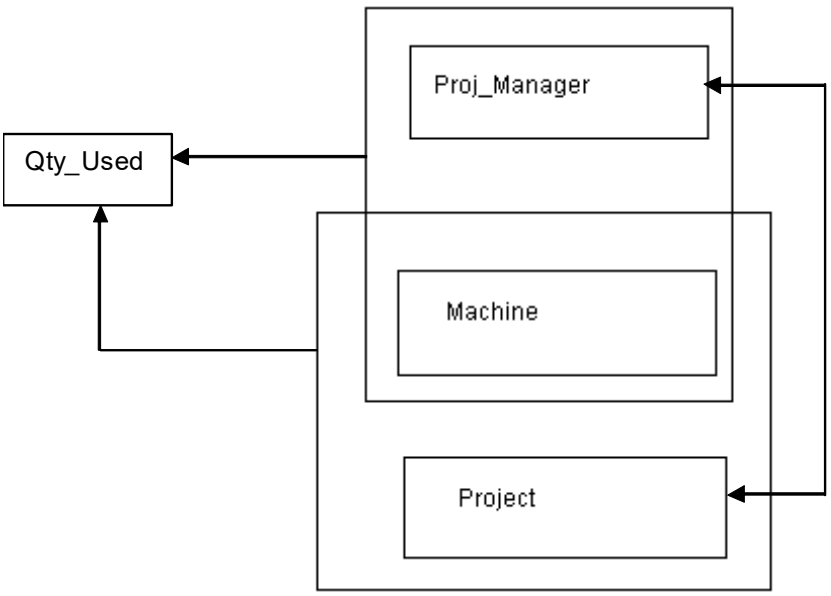


Fig.7.4: FDD of UTILIZE

In the relation UTILIZE, there is only one non-prime attribute called, **QTY_Used**, which is fully functionally dependent on each of the two relations. Thus, the relation UTILIZE is in 2NF. Moreover, there is only one non-prime attribute Qty_Used. There can be no dependencies between the non-prime attributes. Thus the relation is also in 3NF.

Problems in 3NF :

If we consider the above relation i.e., UTILIZE, which is in 3NF, it has the following undesirable properties :

a) The project manager of each project is stored more than once.

b) A project manager cannot be stored until the project has ordered some machines.

c) A project cannot be entered unless that project's manager is known.

d) If a project's manager changes, some rows must also be changed.

7.5 BOYCE-CODD NORMAL FORM (BCNF)

The redundancy and the problems of 3NF can be eliminated by the use of the **Boyce-Codd Normal Form** (**BCNF**) which was proposed by **R.F. Boyce**.

We know that if Y is funcionally dependent on X, that is, X funcionally determines Y, then the following relation holds good:

$$X \rightarrow Y$$

Here, we call X the determinant. From this, we can define BCNF as follows: A table is in Boyce-Codd normal form (BCNF) if the only determinants in the table are the candidate keys.

A relation (or table) R is said to be in BCNF if for every nontrivial FD : $X \rightarrow Y$ between attributes X and Y holds in R. It means :

- X is super key of R,
- $X \rightarrow Y$ is a trivial FD, that is $Y \subset X$.

From the above conditions we have come to know that a relation must only

have candidate keys as determinants. Any relation in BCNF is also in 3NF and consequently in 2NF. However, a relation in 3NF is not necessarily in BCNF.

The difference between 3NF and BCNF is that if the functional dependency $A \rightarrow B$ satisfy that B is a *primary key* attribute and A is not a candidate key, then 3NF will allow this dependency in a relation. Otherwise, if the functional dependency $A \rightarrow B$ satisfy that A must be a candidate key, then this dependency will belong to BCNF.

7.6 FOURTH NORMAL FORM (4NF)

A table is in the Fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.

The fourth normal form is related to the concept of a ***multi-valued dependancy*** (MVD). In simple terms, if there are two columns - A and B - and if for a given A, there can be multiple values of B, then we say that an MVD exists between A and B.

The fourth normal form is theoretical in nature. In practice, normalization up to the third normal form are generally adequate. In certain situations, the designers may also have to look at the BCNF.

Let us consider the following STUDENT table (or relation) :

Table 7.10 STUDENT relation

Sudent	Subject	Language
Geeta	Mythology	English
Geeta	Psycology	English
Geeta	Mythology	Hindi
Geeta	Psycology	Hindi
Sikhar	Gardening	English

We can see that there are two independent MVD facts in this relationship :

a) A student can study many subjects (i.e., Student $\rightarrow \rightarrow$ *Subject)*

b) A student can learn many languages (i.e., Student $\rightarrow \rightarrow$ *Language)*

The primary key for the STUDENT table is currently a composite key made up of all the three columns in the table - *Student*, *Subject* and *Language*. ,i.e., here the primary key is a combination of the three fields *Student, Subject* and *Language.*

The process of bringing STUDENT table into 4NF is by spliting the independent multi-valued components of the primary key into two tables. Now, let us split these two independent multi-valued dependencies into two separate tables namely *Student_Subject* and *Student_Language*. The resulting tables are shown below :

Table 7.11: Relation Student_Subject

Student	Subject
Geeta	Mythology
Geeta	Psycology
Sikhar	Gardening

Table 7.16 Relation Student_Language

Student	Language
Geeta	English
Geeta	Hindi
Sikhar	English

We have seen that, this decomposition reduces redundancy with respect to both the independent multivalued dependency relationships, that is *Subject* and *Language*.

7.7 FIFTH NORMAL FORM (5NF)

A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.

There are some relations which cannot be decomposed into two or higher normal form relations by means of projection methods discussed in 1NF, 2NF, 3NF and BCNF. Such relations are decomposed into three or more relations, which can be reconstructed by means of a three-way or more join operation. This is called fifth normal form (5NF). The 5NF eliminates the problems of 4NF. 5NF allows for relations with join dependencies. Any relation that is in 5NF, is also in other normal forms namely 2NF, 3NF and 4NF. 5NF is mainly used from theoretical point of view and not for practical database design.



CHECK YOUR PROGRESS

1. Select the correct answer from the following :

a) Normalization is a process of
   (i) decomposing of a set of relation.
   (ii) successive reduction of relation schema.
   (iii) deciding which attributes in a relation are to be grouped together.
   (iv) all of these.

b) A normal form is
   (i) a state of a relation that results from applying simple rules regarding FD.
   (ii) the highest normal form condition that it meets.
   (iii) an indication of the degree to which it has been normalized.
   (iv) all of these.

c) In 1NF,
   (i) all domains are simple.
   (ii) in a simple domain, all elements are atomic
   (iii) both (i) & (ii)
   (iv) none of these

d) 2NF is always in
   (i) 1NF (ii) BCNF

(iii) MVD (iv) none of these

e) A relation R is said to be in 2NF
   (i) if it is in 1NF.
   (ii) every non-prime key attributes of R is fully functionally dependent on each relation key of R.
   (iii) if it is in BCNF.
   (iv) both (i) & (ii).

f) A relation R is said to be in 3NF if the
   (i) relation R is in 2NF
   (ii) non prime attributes are mutually independent.
   (iii) functionally dependent on the primary key.
   (iv) all of these.

g) 4NF is concerned with dependencies between the elements of compound keys composed of
   (i) one attributes
   (ii) two attributes
   (iii) three or more attributes
   (iv) none of these

2. Fill in the blanks of the following :

a) _____ first developed the process of normalization.

b) A relation is said to be in 1NF if the values in the domain of each attribute of the relation are _____.

c) 2NF can be violated only when a key is a _____ key or one that consists of more than one _____.

d) In 3NF, no non-prime attribute is functionally dependent on _____.

e) Relation R is said to be in BCNF if for every non-trivial FD : _____ between attributes Xand Y holds in R.

f) A relation is in BCNF if and only if every determinant is a _____.

g) Any relation in BCNF is also in _____ and consequently in _____.

h) 4NF is violated when a relation has undesirable _____.



7.8 LET US SUM UP

1. Normalization is a design technique that is widely used in designing relational model . There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table).

2. Functional dependencies play an important role in differencing good database design from not so good database design.

3. A relation is said to be in First Normal Form (1NF) if only every entry of the relation (the insertion of a tuple and a column) has at most a single value. In other words, “a relation is in First Normal Form if and only if all underlying domains contain atomic values or single value only”.

4. A relation R is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.

5. A relation or table is said to be in third normal form (3NF) if the relation is in 2NF and the non-prime attributes are mutually independent and functionally dependent on the primary key.

6. A relation R is in 4NF if and only if the following conditions are satisfied simultaneously: (i) R is already in 3NF or BCNF. (ii) If it contains no multi-valued dependencies.

7. A relation (or table) R is said to be in BCNF if for every nontrivial FD : X $\rightarrow$ Y between attributes X and Y holds in R.

8. A table is in the fourth normal form (4NF) if it is in BCNF and does not have any independent multi-valued parts of the primary key.

9. A relation (or table) is said to be in the 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate key.



7.9 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



7.10 ANSWERS TO CHECK YOUR PROGRESS

1. a. (iv), b. (iv), c. (ii), d. (i), e. (iv), f. (iv), g. (iii)
2. a) E.F.Codd b) atomic c) composite, attribute
   d) another non prime attributes e) X $\rightarrow$ Y
   f) candidate key g) 3NF, 2NF h) multi-valued dependencies.



7.11 MODEL QUESTIONS

1. What is meant by normalization? Why do we normalize the database? Describe the data normalization process.
2. What are the different normal forms ?
3. Define 1NF, 2NF and
5. How do we convert a database in 1NF to 2NF?
6. How we convert a database in 2NF to 3NF?
7. Explain 3NFwith example.
8. Define BCNF. How does it differ from 3NF ? Why is BCNF stronger than 3NF? Illustrate with an example.

9. Consider a relation-
   Student(Student_Id, Student_Name, Student_Address, Date_of_Birth, Course_Id, Course_Name, Duration, No_of_Papers)
   The functional dependencies are-
   Student_Id-> Student_Name
   Student_Id->Student_Address
   Student_Id->Course_Id
   Student_Id->Date _of_Admission
   Student_Id->Date_of_Birth
   Course_Id->Course_Name
   Course_Id->Duration
   Course_Id->No_of_Papers
   Decompose the above relation into equivalent Third Normal Form (3NF) relations.

10. Following relation for published books is given :
    BOOK(BOO-TITLE, AUTH-NAME, BOOK-TYPE, LIST-PRICE, AUTH-AFFL, PUBLISHER)
    here AUTH-AFFL refers to the affiliation of author. Suppose that the following FDs exists
    BOOK-TITLE $\rightarrow$ PUBLISHER, BOOK-TYPE
    BOOK-TYPE $\rightarrow$ LIST-PRICE
    AUTH-NAME $\rightarrow$ AUTH-AFFL
    a) What normal form is the relation in ? Explain your answer.
    b) Apply normalization until the relations cannot be decomposed ant further. State the reason behind each decomposition.

11. Explain the following terms with an example each :
    (a) Normalization (b) 1NF (c) BCNF (d) 3NF
    (e) 4NF (f) 5NF (g) MVD
    (h) Candidate key (i) Primary Key (j) Super Key

*****

UNIT 8: INTRODUCTION TO SQL

UNIT STRUCTURE

8.1 Learning Objectives

8.2 Introduction

8.3 Structured Query Language (SQL)

- 8.3.1 Characteristics of SQL
- 8.3.2 Advantages of SQL
- 8.3.3 Disadvantages of SQL

8.4 SQL Data Types

- 8.4.1 Numeric Data Type
- 8.4.2 Character Data Type
- 8.4.3 Bit String Data Type
- 8.4.4 Boolean Data Type
- 8.4.5 Date and Time Data Type

8.5 Types of SQL Commands

- 8.5.1 Data Definition Language (DDL)
- 8.5.2 Data Manipulation Language (DML)

8.6 Let Us Sum Up

8.7 Further Reading

8.8 Answers to Check Your Progress

8.9 Model Questions

8.1 LEARNING OBJECTIVES

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

- define SQL
- describe the different data types used in SQL
- describe DDL and DML commands

8.2 INTRODUCTION

In the previous unit you have been introduced to the concepts of relational databases and different forms of normalization. The first, second and third stages of normalization have been covered in the previous unit. In addition to these three stages, BCNF has also been covered in the earlier unit.

In the present unit you will be introduced to the concepts of Structured Query Language (SQL). The different data types in SQL are discussed in detail in this unit. Data Definition Language (DDL) and Data Manipulation Language (DML) SQL commands are also introduced in this unit along with a brief introduction to the Create and Delete SQL tables.

Besides, you will also be made familiar with the basics of SQL tables here while in the next unit we will discuss Insertion and Updating of data into the SQL tables.

8.3 STRUCTURED QUERY LANGUAGE (SQL)

Structured Query Language (SQL) is the standard language for relational databases. SQL provides a common standard and defines the methods to create and maintain databases across different platforms. It was initially developed as "SEQUEL (Structured English Query Language)" at IBM and was designed to manipulate and retrieve the data stored in the database management system at IBM. SQL became a standard later with the joint effort of American National Standards Institute (ANSI) and International Organization for Standardization (ISO).

SQL is based on relational algebra and tuple relational calculus. It contains both data definition and data manipulation commands needed to create and provide maintenance of the databases. In SQL the relational database terms relation, tuple and attribute are described as table, row and column. These terms may be interchangeably used. A relation of a relational database is implemented as a table in SQL; the tuple is implemented as a row and the attributes are implemented as columns in SQL. You need to be familiar with these terms, since they will be used extensively in this unit and in the next few units.

Table 8.1: Terms in relational algebra and SQL

Relational Algebra	SQL
Relation	Table
Tuple	Row
Attribute	Column

Why do we need SQL?

We may have come across the many different databases that are in use today, namely - Microsoft SQL Server, Oracle, MySQL etc. among many. The basic commands used in all of these databases uses the same syntax of structured query language. This helps in maintaining uniformity among the databases. Let us consider the scenario when all these database companies did not have a common standard and used different query languages. In this case, we would have to make different database management software based on the different query language adopted by the database platform. It would also be very difficult for the users to remember the different set of commands needed to perform database operations in various database platforms. SQL, being a common query language standard, saves us all this trouble. The database software now only needs to be based on SQL.

8.3.1 Characteristics of SQL

Some of the characteristics of SQL are listed below.

- SQL is the standard database query language for creating and maintaining database
- SQL is based on relational algebra
- Database system softwares based on SQL are portable across different databases platforms
- SQL commands are independent of the way the data is stored internally
- SQL is simple and easy to learn
- SQL allows the users to create, update, delete and retrieve data objects from the database

8.3.2 Advantages of SQL

Some of the advantages of SQL are listed below.

- SQL provides a well-defined standard for database query languages
- SQL is easy to learn and understand
- It is portable across various database platforms
- Different views can be given to different users
- SQL queries can be used to retrieve large amount of data efficiently
- Complex queries can be efficiently written using SQL
- SQL is used by all major database vendors like IBM, Oracle and Microsoft
- Many open source databases like MySQL use SQL

8.3.3 Disadvantages of SQL

Some of the disadvantages of SQL are listed below.

- Interfacing an SQL database is complex
- Although SQL databases conform to ANSI & ISO standards, some databases may go for proprietary extensions



CHECK YOUR PROGRESS

Q1. Mention two advantages of SQL.

Q2. Give two characteristics of SQL.

Q3. Mention two disadvantages of SQL.

8.4 SQL DATA TYPES

Data types describe the type of data an attribute can have. We are familiar with the different data types in different programming languages like C, C++ etc. Similarly, there are different data types in SQL also. The basic data types in SQL fall into *numeric*, *character string*, *bit string*, *Boolean*, *date and time* category. Let us discuss each of these categories of data types in detail now.

8.4.1 Numeric Data Type

Numeric data type includes integer numbers of different sizes and also floating point numbers of different precision. Formatted numbers can also be declared in SQL. Some of the numeric data types are listed below.

- **Integer (INT) Data Type:** Integer data type is used to describe integer type data in SQL. There is a separate SMALLINT data type to contain integers of small size. The range of values for Integer (INT) data type is from - 2,147,483,648 to + 2,147,483,647. Examples of integers are 45, 34,567, and -672345 among many.
- **Small Integer (SMALLINT) Data Type:** Small integer data type is used to describe integer values of small sizes. The range of values for SMALLINT data type is -32,768 to + 32,767. Examples of small integers are 45, 31,567, and 28973 among many.
- **Formatted Data Types:** Formatted numbers can be used for both numeric (NUMERIC(i,j))and decimal values (DECIMAL(i,j)). For both the formatted data types DECIMAL(i,j) and NUMERIC(i,j), the i stands for precision, that is the number of total digits in the number and j stands for scale, that is the number of digits before the decimal point. For example, DECIMAL(7,3) is a number that has 4 digits before the decimal and 3 digits after the decimal point.

8.4.2 Character-string Data Type

Character-string data types are used to represent characters of fixed and variable lengths. Some of the character-string data types are listed below.

- **Character (CHAR(n)) data type:** Character (CHAR) data type is used to represent character data in SQL whose size is fixed, meaning that we know the exact number of characters to be put in that column. The length of the string must be specified when we declare a character type data. For example, CHAR (6) means that a character of length 6 is expected. The default value for the length (n) is 1. Examples of characters are 'database', 'DATABASE', 'Query Language', and '45', among many.

  If we assign a character value that contains less number of characters than what was defined in the character column, than the rest of the column is filled with spaces. For example, if we define a character of size 10, and then put a value of 'SQL' in that column, then the first three positions contains 'SQL' but the rest of the column length will be filled with spaces.

- **Variable Character (VARCHAR(n)) data type:** Variable character (VARCHAR) data type is used to represent character data in SQL whose length may vary from 1 up to the maximum length that is specified in the declaration. The length of the string must be specified when we declare a variable character type data and the length starts from 1. For example, VARCHAR (6) means that a character of length six or less than six is expected. Examples of variable characters are 'data', 'SQL', 'Query Language', and '45', among many.
  A CHAR field is a fixed length, and VARCHAR is a variable length field. This means that the storage requirements are different - a CHAR always takes the same amount of space regardless of what you store, whereas the storage requirements for a VARCHAR vary depending on the specific string stored. The character (CHAR) data

type fields have a fixed length and take the same amount of space in computer memory irrespective of the string we store in that field. Whereas for variable character (VARCHAR) data type, the amount of space in computer memory depends on the length of the string we store in that field. For example, let us make two declarations for a field name once using character and once using variable character. In both the declarations, let the maximum size be declared as 30 to store a maximum of 30 characters.

name CHAR(30)
name VARCHAR (30)

Now, if we try to store the string "INDIA" in the field declared using CHAR, even though the string requires space for only 5 characters still it will be stored using space for 30 characters. But in case of VARCHAR, the string "INDIA" will be stored in memory using space required for only 5 characters.

8.4.3 Bit-string Data Type

Bit-string data type is used to represent strings of values 0's and 1's. Some of the bit string data types are listed below:

- **Bit Strings (BIT (n))**: Bit string data type is used to represent strings containing bit values in SQL whose size is fixed meaning that we know the exact number of bits to be put in that column. The length of the string must be specified when we declare a bit string type data. For example, BIT (4) means that a bit string of length 4 is expected. The default value for the length (n) is 1. Examples of bit strings are '0100', '111000' and '1010100' among many.

- **Varying Bit Strings (BIT VARYING(n))**: Varying bit string data type is used to represent bit strings in SQL whose length may vary from 1 up to the maximum length that is specified in the declaration. The length of the string must be specified when we declare a variable bit string type data and the length starts from 1. For example, BIT

VARYING (4) means that a bit string of length four or less than four is expected. Some examples for variable bit strings among many are '010', '1111' and '010101'.

8.4.4 Boolean Data Type

Boolean data type stores two values "TRUE" and "FALSE". The Boolean data type does not require the use of any parameters. The two values are case insensitive. Some examples among many for Boolean data type values are 'TRUE' and 'FALSE'.

If we want to compare any field with Boolean data type then we will have to use the two keywords "True" and "False". If we compare any Boolean data type field with any other value then we will get an error.

8.4.5 Date and Time Data Type

Date and time data type is used to represent and store date and time information in SQL. Some of the date and time data types are listed below.

- **Date (DATE):** The date (DATE) data type stores the date information like years, months and days in SQL. This data type does not require any parameters in declaration. The date values are stored in the "YYYY-MM-DD" format. The range for years (YYYY) is from 0001 to 9999, for months (MM) is from 1 to 12 and for days (DD) from 1 to 31. Although for some databases like Informatica, the years (YYYY) range is from 1582 to 9999. Some examples among many for date data type values are '1990-05-12', '2017-12-25' and '8999-01-01'.

- **TIME (TIME):** The time (TIME) data type stores the time information like hours, minutes and seconds in SQL. This data type does

not require any parameters in declaration. The time values are stored in the "HH:MM:SS" format. The range for hours (HH) is from 0 to 23, for minutes (MM) is from 0 to 59 and for seconds (SS) is from 0 to 59. Some examples among many for time data type values are '05:35:47', '23:32:25' and '17:01:59'.

- **TIMESTAMP (TIMESTAMP):** The timestamp (TIMESTAMP) data type stores timestamp values that include both the date and time information like years, months, days, hours, minutes and seconds in SQL. This data type like date and time data types does not require any parameters in declaration. The timestamp values are stored in the "YYYY-MM-DD HH:MM:SS" format. The date and time portions of the timestamp data type is separated by a space. The ranges for timestamp are the same as applicable to date and time data type. The range for years (YYYY) is from 0001 to 9999, for months (MM) is from 1 to 12, for days (DD) from 1 to 31, for hours (HH) is from 0 to 23, for minutes (MM) is from 0 to 59 and for seconds (SS) is from 0 to 59. Some examples among many for timestamp data type values are '1990-05-12 05:35:47', '2017-12-25 23:32:25' and '8999-01-01 17:01:59'.

Thus the common and frequently used data types have been described in this section. There are many more data types in SQL and all of these data types have not been described here. We will discuss these other data types as and when we will encounter them in the two new units.

Note: Different data types might have different names in different databases. It is always wise to read the documentation of the database you are using. It may happen that even though the names of the data types are same, the precision values may be different for the same data types in different databases.

Table 8.2: Data Type Summary

Data Type	**Keyword**	**Range**	**Valid Examples**	**Invalid Examples**
Integer	INT	– 2,147,483,648 to + 2,147,483,647	267, 9876, -128934	‘Assam’, TRUE, ’10:20:25’
Small integer	SMALLINT	-32,768 to + 32,767	345, -19876, 30345	-128934, ‘Assam’, TRUE
Decimal	DECIMAL(i.j)	- 10^38 + 1 to 10^38 - 1	1345.67, -690.25	False, ‘India’
Numeric	NUMERIC(i,j)	- 10^38 + 1 to 10^38 - 1	-1345.67, 690.25	true ’16:45:37’
Character	CHAR(n)	Length of n is from 1 to 8000	‘Assam’, ‘GUWAHATI’	674, ’16:45:37’
Variable length character	VARCHAR(n)	Length of n is from 1 to 8000	‘KKHSOU’, ‘database’	-90223, FALSE
Bit string	BIT(n)	0 to 1	‘0101’, ’10001’	‘Assam’, 100
Variable Bit string	BIT VARYING (n)	0 to 1	‘11110’, ‘1010’	‘Assam’, 4050
Boolean	BOOLEAN	True and False	false, FALSE, True	1001, 1234, ‘Assam’

Date	DATE	0001 to 9999 for years, 1 to 12 for months, 1 to 31 for days	'1870-08-23', '2015-11-11', '7824-12-30'	True, 365, 'India', '2017-13-32'
Time	TIME	0 to 23 for hours, 0 to 59 for minutes, 0 to 59 for seconds	'07:13:57', '21:32:45', '13:01:05'	False, 1234, 'Assam', '13:11:65'
Timestamp	TIMESTAMP	0001 to 9999 for years, 1 to 12 for months, 1 to 31 for days, 0 to 23 for hours, 0 to 59 for minutes, 0 to 59 for seconds	'1870-08-23 07:13:57', '2015-11-11 21:32:45', '7824-12-30 13:01:05'	False, 12, 'India', '2017-13-32 13:11:65'

8.5 Types of SQL Commands

We now have some basic ideas about the various data types available in SQL. Now, let us look at the commands that we use in SQL. SQL uses commands to implement the relations, fields and tuple of relational algebra into tables, rows and columns fields in database. These commands help in building and maintaining the databases. The most commonly used SQL commands broadly falls into two language categories: *Data Definition Language (DDL)* and *Data Manipulation Language (DML)*. Besides these two categories, some SQL commands fall under Transaction Control Statements (TCS) which manage the changes made by the earlier two categories of commands.

8.5.1 Data Definition Language (DDL)

Data Definition Language (DDL) commands are used to create and restructure database objects like databases, tables, views and index. Some of the commands in data definition language are: CREATE, ALTER, DROP, TRUNCATE, and RENAME.

The functionalities of these commands are described in Table 8.2.

Table 8.2: Data Definition Language Commands

Command	Function
CREATE	Used to create database, tables, views and index
ALTER	Used to alter the structure of an already existing database table
DROP	Used to delete database, tables, views and index
TRUNCATE	Used to delete all details from an existing database table but not the table itself
RENAME	Used to rename a database table

8.5.2 Data Manipulation Language (DML)

Data Manipulation Language (DML) commands are used to manipulate data in database objects. Some of the commands in data manipulation language are: INSERT, UPDATE, and DELETE. The functionalities of these commands are described in table 8.3.

Table 8.3: Data Manipulation Language Commands

Command	Function
INSERT	Used to insert data in a table
UPDATE	Used to update the old data with new data in a table
DELETE	Used to delete data in a table
SELECT	Used to retrieve selected portions of data from a table according to preferences

8.5.3 Transaction Control Statements (TCS)

Transaction Control Statements (TCS) commands are used to manage the changes that are made in database after the use of data definition language and data manipulation language commands. Some of the commands in Transaction Control Statements (TCS) are COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION. The functionalities of these commands are described in table 8.4.

Table 8.4: Transaction Control Statements (TCS) Commands

Command	Function
COMMIT	Used to save the changes in the database permanently
ROLLBACK	Used to restore the database to the point of the last COMMIT
SAVEPOINT	Used to mark a save point to which we can rollback later if required
SET TRANSACTION	Used to configure and start a transaction

Most of these commands are discussed in detail in the next few units.



CHECK YOUR PROGRESS

Q4. How is CHAR data type different from VARCHAR data type?

Q5. Give the range for small integer data type.

Q6. How can we save our transactions permanently in the database?

Q7. Give the difference between TRUNCATE and DROP command.



8.6 LET US SUM UP

- Structured Query Language (SQL) is the standard language for relational databases. SQL provides a common standard and defines methods to create and maintain databases across different platforms.
- SQL is based on relational algebra and tuple relational calculus. It contains both data definition and data manipulation commands needed to create and provide maintenance of the databases.
- The basic data types in SQL fall into numeric, character string, bit string, Boolean, date and time category.
- Integer data type is used to describe integer type data in SQL. Small integer data type is used to describe integer values of small sizes.
- Formatted numbers can be used for both numeric (NUMERIC(i,j))and decimal values (DECIMAL(i,j)). For both the formatted data types DECIMAL(i,j) and NUMERIC(i,j), the i stands for precision, that is the number of total digits in the number and j stands for scale, that is the number of digits before the decimal point.
- Character (CHAR) data type is used to represent character data in SQL whose size is fixed meaning that we know the exact number of characters to be put in that column. Variable character (VARCHAR) data type is used to represent character data in SQL whose length may vary from 1 up to the maximum length that is specified in the declaration.
- Bit-string data type is used to represent strings of values 0's and 1's.
- Boolean data type stores two values "TRUE" and "FALSE". The Boolean data type does not require the use of any parameters. The two values are case insensitive.
- The date (DATE) data type stores the date information like years, months and days in SQL. This data type does not require any parameters in declaration. The date values are stored in the

"YYYY-MM-DD" format.

- The time (TIME) data type stores the time information like hours, minutes and seconds in SQL. The time values are stored in the "HH:MM:SS" format.
- The timestamp (TIMESTAMP) data type stores timestamp values that include both the date and time information like years, months, days, hours, minutes and seconds in SQL. The timestamp values are stored in the "YYYY-MM-DD HH:MM:SS" format.
- Most commonly used SQL commands broadly falls into two language categories: Data Definition Language (DDL) and Data Manipulation Language (DML).
- Data Definition Language (DDL) commands are used to create and restructure database objects like databases, tables, views and index. Some of the commands in data definition language are: CREATE, ALTER, DROP, TRUNCATE, and RENAME.
- Data Manipulation Language (DML) commands are used to manipulate data in database objects. Some of the commands in data manipulation language are: INSERT, UPDATE, and DELETE.
- Transaction Control Statements (TCS) commands are used to manage the changes that are made in database after use of data definition language and data manipuZ



8.7 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



8.8 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: Two advantages of SQL are:

i. SQL provides a well-defined standard for database query languages

ii. Easy to learn and understand

Answer to Q2: Two characteristics of SQL are:

i. SQL is the standard database query language for creating and maintaining database

ii. Database system software's based on SQL are portable across different databases platforms

Answer to Q3: Two disadvantages of SQL are:

i. Interfacing an SQL database is complex

ii. Although SQL databases conform to ANSI & ISO standards, some databases may go for proprietary extensions

Answer to Q4: A CHAR data type field is fixed length whereas for VARCHAR data type it is variable length field. This means that the storage requirements are different - a CHAR always takes the same amount of space regardless of what we store, whereas the storage requirements for a VARCHAR varies depending on the specific string stored.

Answer to Q5: The range for small integer data type is from -32,768 to +32,767

Answer to Q6: We can save our transactions permanently in the database by using the COMMIT command.

Answer to Q7: DELETE command is used to delete the whole database table whereas TRUNCATE command is used to only empty the contents of the table.



8.9 MODEL QUESTIONS

Q1. Describe the different data types in SQL.

Q2. Explain the need for SQL.

Q3. Describe the different DML commands.

Q4. Describe the different DML commands.

Q5. Name the different commands used in transaction control state ments.

Q6. Differentiate among DATE, TIME and TIMESTAMP commands using examples.

Q7. Describe the different types of Bit-String data type.

Q8. What are formatted data types? Describe them in brief.

Q9. Describe in brief with examples where we can use Boolean data type.

Q10. Why do we need SQL?

*****

UNIT 9: SQL Commands 1

UNIT STRUCTURE

9.1 Learning Objectives

9.2 Introduction

9.3 Basic SQL Commands

- 9.3.1 Creating and Deleting Database
- 9.3.2 Creating Table
- 9.3.3 Altering Tables
- 9.3.4 Deleting Tables
- 9.3.5 Inserting Data
- 9.3.6 Updating Data
- 9.3.7 Deleting Data

9.4 Let Us Sum Up

9.5 Further Reading

9.6 Answers to Check Your Progress

9.7 Model Questions

9.1 LEARNING OBJECTIVES

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

- Create a new database, delete an existing database
- create, alter and delete tables in a database.
- insert data into tables
- update data in tables
- delete data from a table

9.2 INTRODUCTION

In the previous unit you have been introduced to the data types in SQL. The Data Definition Language and Data Manipulation Language commands have

been covered in the previous unit.

In the present unit you will be introduced to the basic SQL commands. In this unit, we will learn how to create and delete a database. We will also discuss how to create, alter and delete a table in SQL. This unit also covers how we can insert, update and delete data from a table.

In this unit you will be made familiar with the basic SQL commands and then in the next unit we will discuss additional SQL commands.

9.3 BASIC SQL COMMANDS

We have been introduced to SQL in the previous unit. We already have an idea of data definition and data manipulation language commands from the previous unit. Most of these commands will be covered in detail in this unit and in the next unit. In this section we will go through each of the basic SQL commands in detail.

9.3.1 Creating and Deleting a Database

Creating a database

Creating a database is generally the first step towards designing a database system. We can create a database using the create command. The syntax for creating a database is listed below.

```
CREATE DATABASE database_name;
```

In the above syntax, create database followed by the identifier name is the command that tells the database system that we want to create a database with that identifier name. Every command is followed by a semicolon to indicate the end of the command. Now, let us try to create a KKHSOU university database. We will write the following command to create this database.

```
CREATE DATABASE KKHSOU;
```

This command will create a database named "KKHSOU". The following statement is another example where we will create another database for keeping company details in a database named "Company".

```
CREATE DATABASE company;
```

Listing all databases present in the database system

If we want to see all the databases present in our database system then we can use the show command. This command can also be used to verify if we have created a database successfully.

```
SHOW DATABASES;
```

This command will list all the databases present in our database system. The output of the show database command will be the names of the two databases that we have created earlier.

```
KKHSOU
COMPANY
```

Using a database

Now, if we want to create tables in the "KKHSOU" database then we will have to enter the specified database. To do so, we will use the use command. The syntax for using a particular database is given below.

```
USE database_name;
```

For example, if we want to use the KKHSOU database and create tables there then we will use the following statement.

```
USE KKHSOU;
```

Deleting a database

If we want to delete a particular database then we use the drop command. The syntax for deleting a database is given below.

```
DROP DATABASE database_name;
```

For example, if we want to drop the "company" database that we created earlier then we can write the following statement.

```
DROP DATABASE company;
```

This statement will delete the database named "company" from the database system.

9.3.2 Creating a Table

Creating a table means naming a table, defining its column names and assigning data types to the columns. A table is created using the create table command. The syntax of create table statement is given below.

```
CREATE TABLE table_name
(
      column_name1 datatype,
      column_name2 datatype,
      column_name3 datatype,
      ... ...
      column_nameN datatype,
      primary key (one or more columns)
);
```

In the above syntax, create table is the command that tells the database system that we want to create a table. This is followed by the unique identifier name for the table. Then the list of columns along with their data type and sizes are mentioned within brackets followed by a semicolon. Let us look at an example to understand the creation of tables.

```
CREATE TABLE students
(
      rollno          int             not null,
      name            varchar(30)     not null,
      course          varchar(10)     not null,
      date_of_birth   date            not null,
      primary key     (rollno)
);
```

In this example, we have created a table named **"students"** which will keep student information details for the university. The table has four columns where the first column represents roll number, so it is defined as of type integer data type. The second and third fields represent the name and course taken by the student and so are of variable character type. The fourth field represents the date of birth of the student and so is declared as date data type. Each student can be represented uniquely by the roll number of the student, so the roll number is defined as the primary key in this student table. The "NOT NULL" is defined for all the columns which means that the database system will not accept NULL values for any field. The data type and the size of the fields for the table must be carefully defined. Declaring a field with large size which in reality is not needed leads to wastage of memory space. Let us create another table "course", which keep the information of various courses offered by the university in the same database. In this "course" table, we have defined three columns - course_id, course_name and semesters.

```
CREATE TABLE course
(
      course_id       varchar(5),
      course_name     varchar(30),
      semesters       int
);
```

Viewing structure of a database

We can view the structure of the table we have created using the desc command. This command can also be used to verify if the database has been successfully created according to our specifications. This command gives us the field information of any existing table in six columns. The information provided includes the field name, the data type of the field, whether the field is defined as NULL, whether the field is primary key, its default values and extra information. The syntax for using this command is given below.

```
DESC table_name;
```

This statement will show us the structure of the database table that is mentioned. For example, if we want to view the structure of the "student" table we created then we will write the following statement.

```
DESC students;
```

This statement will give us an output as below.

```
mysql> desc students;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| rollno       | int(11)     | NO   | PRI | NULL    |       |
| name         | varchar(30) | NO   |     | NULL    |       |
| course       | varchar(10) | NO   |     | NULL    |       |
| date_of_birth | date       | NO   |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

9.3.3 Altering a Table

Altering a table means modifying the structure of an existing table. The ***alter table*** command is used to alter a table. Altering the table may include renaming a table or a column, adding columns, deleting columns, changing the size of columns and adding primary keys to an already existing table. Different statements are used to make all of these changes in a table. Let us look at the alter table statements in detail.

Renaming a table

```
ALTER TABLE table_name

RENAME TO new_table_name ;
```

This is the alter table statement for renaming a table. We specify the name of the table to be renamed after the **"alter table"** keywords. This is followed by the keywords **"rename to"** and the new table name.

If we use the show tables command then we will get a list of the existing tables. We can see the names of the two tables we created "students" and "course" in this list. Now, let us change the name of table "course" that we have earlier created to "course_details". To make this change we can use this statement below.

```
ALTER TABLE course

RENAME TO course_details ;
```

The above statement will change the name of the table "course" from "course" to "course_details". We can easily verify whether this change has been successful by using the show tables command. The output of the show tables command will now show a table with name "course_details" instead of "course".

Renaming a column in table

```
ALTER TABLE table_name

CHANGE COLUMN old_column_name new_column_name

      column definition ;
```

This is the alter table statement for renaming a column. We first specify the name of the table in which the column is to be renamed after the "alter table" keywords. This is followed by the keywords "change col-

umn" after which we specify the old column name and the new column name. We also have to give a column definition specifying its data type and whether the column is NULL or NOT NULL. Now, let us change the name of the column "semesters" in the table "course_details" to "sem". To make this change we can use this statement below.

```
ALTER TABLE course_details

CHANGE COLUMN semesters sem

      int NULL ;
```

The above statement will change the name of the column in the table "course" from "semesters" to "sem". We can easily verify whether this change has been successful by using the ***desc*** command.

```
DESC course_details;
```

The output of the desc command will now show a table with a column name "sem" instead of "semesters".

```
mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| semesters   | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE course_details
    -> CHANGE COLUMN semesters sem
    ->  int NULL ;
Query OK, 0 rows affected (0.27 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| sem         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

Adding column to a table

```
ALTER TABLE table_name

ADD column_name column definition;
```

This is the alter table statement for adding a column in the table. We first specify the name of the table in which we want to add a new column after the **"alter table"** keywords. This is followed by the keyword **"add"** after which we specify the name of the column that we want to add. We also have to give a column definition specifying its data type, size and whether the column is NULL or NOT NULL. Now, let us add a column "gender" to the "students" table which will keep the gender information of the student. To make this change we can use the statement below.

```
ALTER TABLE students

ADD gender varchar(15) NOT NULL;
```

The above statement will add a new column "gender" in the table "students". We can easily verify whether this change has been successful by using the desc command.

```
DESC students;
```

The output of the desc command will now show that a new field "gender" has been added to table "students".

```
mysql> desc students;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| rollno       | int(11)     | NO   | PRI | NULL    |       |
| name         | varchar(30) | NO   |     | NULL    |       |
| course       | varchar(10) | NO   |     | NULL    |       |
| date_of_birth | date       | NO   |     | NULL    |       |
| gender       | varchar(15) | NO   |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Modifying a column in a table

```
ALTER TABLE table_name
MODIFY column_name column definition;
```

This is the alter table statement for modifying a column in the table. We first specify the name of the table in which we want to modify the column after the "alter table" keywords. This is followed by the keyword "modify" after which we specify the name of the column that we want to modify. We also have to give a column definition specifying its new modified data type, size and whether the column is NULL or NOT NULL. Now, let us modify the column "course" in the "students" table. Let it now store course names with 20 characters instead of 10 characters like in the earlier definition. Also, let us add another modification to this column to accept NULL values. To make this changes we can use the statement below.

```
ALTER TABLE students
MODIFY course varchar(20) NULL;
```

The above statement will modify the column "course" in the table "students" to store new up to 20 characters and to accept NULL values in this field. We can easily verify whether this change has been successful by using the desc command.

```
DESC students;
```

The output of the desc command shows that the field "course" has been modified in table "students". The figure below shows the modification of the column "course" in the table "students" after the execution of alter table statement.

```
mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(10) | NO   |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
| gender        | varchar(15) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql> ALTER TABLE students
    -> MODIFY course varchar(20) NULL;
Query OK, 0 rows affected (0.25 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(20) | YES  |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
| gender        | varchar(15) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Deleting column from a table

```
ALTER TABLE table_name

DROP COLUMN column_name ;
```

This is the alter table statement for deleting a column from a table. Just like in the earlier definitions, here also, we first specify the name of the table from which we want to delete a column after the **"alter table"** keywords. This is followed by the keywords **"drop column"** after which we specify the name of the column that we want to delete. Now, let us delete the column "gender" from the "students" table. To make this change we can use the statement below.

```
ALTER TABLE students

DROP COLUMN gender;
```

The above statement will delete the column "gender" from the table "students". We can easily verify whether this change has been successful by using the ***desc*** command.

```
DESC students;
```

The output of the **desc** command will now show that field "gender" has been deleted from the table "students".

```
mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(20) | YES  |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
| gender        | varchar(15) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> ALTER TABLE students
    -> DROP COLUMN gender ;
Query OK, 0 rows affected (0.28 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc students;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| rollno        | int(11)     | NO   | PRI | NULL    |       |
| name          | varchar(30) | NO   |     | NULL    |       |
| course        | varchar(20) | YES  |     | NULL    |       |
| date_of_birth | date        | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.08 sec)
```

Adding a primary key in a table

```
ALTER TABLE table_name

ADD PRIMARY KEY (column_name);
```

This is the alter table statement for adding primary keys to the table. Like in the previous definitions, here also, we first specify the name of the table in which we want to add primary keys after the "alter table" keywords. This is followed by the keywords "add primary key" after which we specify the column name/s that we want to add as primary key within brackets. Now, let us add the column "course_id" as pri-

mary key in the "course_details" table. To make this change we can use this statement below.

```
ALTER TABLE course_details

ADD PRIMARY KEY (course_id);
```

The above statement will add the column "course_id" as primary key in the table "course_details". We can easily verify whether this change has been successful by using the ***desc*** command.

```
DESC course_details;
```

The output of the ***desc*** command will now show that the column "course_id" has been added as primary key to the table "course_details".

```
mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | YES  |     | NULL    |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| sem         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> ALTER TABLE course_details
    -> ADD PRIMARY KEY (course_id);
Query OK, 0 rows affected (0.20 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc course_details;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| course_id   | varchar(5)  | NO   | PRI |         |       |
| course_name | varchar(30) | YES  |     | NULL    |       |
| sem         | int(11)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

9.3.4 Deleting a Table

Deleting a table means removing the table from the database system. A table is deleted by using the drop table command. The syntax

of ***drop table*** statement is given below.

```
DROP TABLE table_name ;
```

For example, if we want to delete the table "course_details" then we can use the following statement.

```
DROP TABLE course_details ;
```

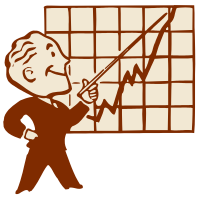
The above statement will delete the table "course_details". We can easily verify whether the table deletion has been successful by using the show tables command.

```
SHOW TABLES;
```

The output of the show tables command will now display the list of existing tables in this database. Since "course_details" table has just been deleted so it would not be displayed in this list.

```
mysql> show tables;
+------------------+
| Tables_in_kkhsou |
+------------------+
| students         |
+------------------+
1 row in set (0.00 sec)
```

Till now we have discussed how to create, delete and alter tables. We have learnt how to add columns, delete columns, add primary columns, rename tables and columns till now. In the next few pages, we will learn how to insert, update and delete data from a table.



CHECK YOUR PROGRESS

Q1. Give the command to view the structure of a table.

Q2. What is the syntax for renaming a database table?

Q3. Write the statement to add a column "age" to the "students" table.

Q4. Give the command to see the list of tables in a database.

9.3.5 Inserting Data in Table

The INSERT INTO command is used to insert data into a table. The syntax for this command is given below.

```
INSERT INTO table_name (column1, column2, ..., columnN)
VALUES (value1, value2, ..., valueN);
```

In the above syntax, we specify the name of the table where we want to insert values after the **"insert into"** keywords. This is followed by the list of column names separated by commas before the keyword **"values"**. A list of values separated by commas and within single quotes is written after the keyword "values" corresponding to the order of the list of columns. Let us insert two tuples into the "students" table that we had created earlier.

```
INSERT INTO students(rollno,name,course,date_of_birth)
VALUES('1','ria','mca','1986-07-25');
```

```
INSERT INTO students(rollno,name,course,date_of_birth)
VALUES('2','bina','mscit','1990-03-11');
```

We can verify that whether the data has been successfully inserted by using the select command. We will learn more about **"select"** command in the later unit. The records inserted into the table "students" will be displayed when we use the following statement.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table.

```
mysql> select * from students;
+--------+------+--------+---------------+
| rollno | name | course | date_of_birth |
+--------+------+--------+---------------+
|      1 | ria  | mca    | 1986-07-20    |
|      2 | bina | mscit  | 1990-03-11    |
+--------+------+--------+---------------+
2 rows in set (0.00 sec)
```

There is another way to insert values in a table without mentioning the column names. The syntax for this is given below.

```
INSERT INTO table_name VALUES (value1, value2, ..., valueN);
```

This is an alternative way to insert values into the table in which we do not have to mention the column names. Only the values are provided in this syntax but in this way we have to make sure that the values are provided in the order the columns are arranged. Let us insert two tuples into the "students" table that is using this method.

```
INSERT INTO students VALUES('3','ram','pgdca','1985-12-31');
```

```
INSERT INTO students VALUES('4','abdul','mca','1972-08-01');
```

We can verify that whether the data has been successfully inserted by using the select command. The records inserted into the table "students" using the new method will be displayed when we use the following statement.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table.

```
mysql> select * from students;
+--------+-------+--------+---------------+
| rollno | name  | course | date_of_birth |
+--------+-------+--------+---------------+
|      1 | ria   | mca    | 1986-07-20    |
|      2 | bina  | mscit  | 1990-03-11    |
|      3 | ram   | pgdca  | 1985-12-31    |
|      4 | abdul | mca    | 1972-08-01    |
+--------+-------+--------+---------------+
4 rows in set (0.00 sec)
```

9.3.6 Updating Data in Table

The UPDATE command is used to modify the existing records in a table. The syntax for this command is given below.

```
UPDATE table_name
SET column1 = value1, column2 = value2,...,columnN = valueN
WHERE condition;
```

In the above syntax, we specify the name of the table where we want to update records after the "update" keyword. This is followed by the list of column names and their new assigned values after the keyword "set". The "where" keyword is used which provides us the condition to select only those records in which we want to make the changes. We will learn about where clause in more details in the next unit. Let us update a record in the "students" table.

```
UPDATE students
SET name = 'gazal'
WHERE rollno = '2';
```

In the above statement, we have modified the name column of the record to "gazal" where the "rollno" is "2". We can verify whether the data has been successfully modified by using the select command.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table.

```
mysql> select * from students;
+--------+-------+--------+---------------+
| rollno | name  | course | date_of_birth |
+--------+-------+--------+---------------+
|      1 | ria   | mca    | 1986-07-20    |
|      2 | gazal | mscit  | 1990-03-11    |
|      3 | ram   | pgdca  | 1985-12-31    |
|      4 | abdul | mca    | 1972-08-01    |
+--------+-------+--------+---------------+
4 rows in set (0.00 sec)
```

9.3.7 Deleting Data from Table

The DELETE command is used to delete any record from the table. The syntax for this command is given below.

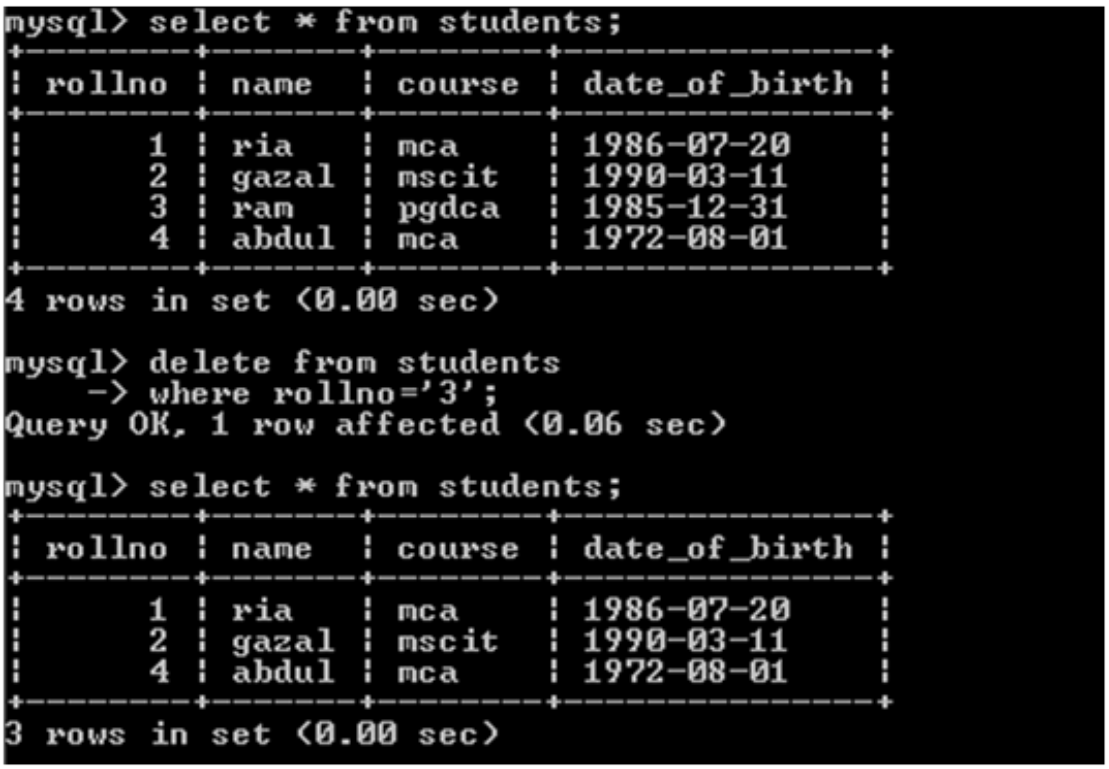
In the above syntax, we specify the name of the table from where we want to delete records after the "delete from" keywords. The "where" keyword is used which provides us the condition to select only those records which we want delete. Let us delete a record from the "students" table.

```
DELETE FROM students
WHERE rollno='3';
```

In the above statement, we have chosen to delete the record where the "rollno" is "3". We can verify whether the data has been successfully deleted by using the select command.

```
SELECT * from students;
```

The output of the above statement will display the contents of the "students" table. In the figure below, we can see that the record which had the value of field "rollno" as "3" has been deleted from the table.





CHECK YOUR PROGRESS

Q5. Let us consider a table employee(id, name, department, salary) where field "id" is the primary key. Write the statements in SQL to perform the following functions.

i. Create this table

ii. Insert two records in the table.

iii. Update the first record in the table so that the salary is now "35000".

iv. Delete the records from the table where salary is less than 30000.



9.4 LET US SUM UP

- We can create a database using the create command. The syntax for creating a database is:
  CREATE DATABASE *database_name;*

- If we want to delete a particular database then we use the drop command. The syntax for deleting a database is **DROP DATABASE** *database_name;*

- Creating a table means naming a table, defining its column names and assigning data types to the columns. A table is created using the ***create table*** command.

- Altering a table means modifying the structure of an existing table. The ***alter table*** command is used to alter a table.

- Altering the table may include renaming a table or a column, adding columns, deleting columns, changing the size of columns and adding primary keys to an already existing table

- Deleting a table means removing the table from the database system. A table is deleted by using the ***drop table*** command. The syntax of drop table statement is **DROP TABLE** *table_name;*

- The INSERT INTO command is used to insert data into a table. The syntax for this command is

  INSERT INTO table_name (column1, column2, ..., columnN) **VALUES** (value1, value2, ..., valueN);

- The UPDATE command is used to modify the existing records in a table. The syntax for this command is

  UPDATE *table_name*
  SET *column1 = value1, column2 = value2,...,*columnN = *valueN*
  WHERE *condition;*

- The DELETE command is used to delete any record from the table. The syntax for this command is

  DELETE FROM *table_name*
  WHERE *condition;*



9.5 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



9.6 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: We can view the structure of the table by using the ***desc*** command. The syntax for using this command is:

```
DESC table_name;
```

Answer to Q2: The syntax for renaming a database is:

```
ALTER TABLE table_name RENAME TO new_table_name;
```

We specify the name of the table to be renamed after the "alter table" keyword. This is followed by the keywords "rename to" and the new table name.

Answer to Q3: The following statement can be used to add a column age to the "students" table.

```
ALTER TABLE students ADD age int NOT NULL;
```

Answer to Q4: The command "SHOW TABLES" displays the list of tables present in a database.

Answer to Q5: The statements to perform the functions are:

i.
```
CREATE TABLE employee
(
id   int,
name  varchar(30),
dept  varchar(20),
salary int,
PRIMARY KEY(id)
);
```

ii.
```
INSERT INTO employee VALUES('01','Pratibha
Bora','Accounts','25000');
INSERT INTO employee VALUES('02','Jaynta
Medhi','Marketing','20000');
```

iii.
```
UPDATE employee SET salary='35000' WHERE
id='01';
```

iv.
```
DELETE FROM employee WHERE salary<'30000';
```



9.7 MODEL QUESTIONS

Q1. Describe the different ways in which a table can be altered.

Q2. Write SQL commands for the following:

i. Create a database "hostel"

ii. Create two tables "boys" and "girls" in hostel database. Include room_number, broader_name, course, semester as attributes in both the tables

iii. Insert three records in both the tables

iv. Update the field "semester" to a value "4" in table "boys" where "course" is "mca"

v. Update the field "course" to a value "mscit" in table "girls" where is "room_number" is "1"

vi. Delete the record in "boys" table where "room_number" is less than 3.

Q3. Give the command to show all the databases in the database system.

Q4. Give the syntax for deleting a database.

Q5. Describe how "create table" is used to for creating tables in a database using example.

Q6. How can a primary key be added after the table has already been created?

*****

UNIT 10: ADVANCED SQL COMMANDS

UNIT STRUCTURE

10.1 Learning Objectives
10.2 Introduction
10.3 Advanced SQL Commands
    10.3.1 Select
    10.3.2 Where
    10.3.3 Order By
    10.3.4 Group By
    10.3.5 Having
    10.3.6 Logical Operators (AND, OR, NOT)
10.4 Join
    10.4.1 Inner Join
    10.4.2 Left (Outer) Join
    10.4.3 Right (Outer) Join
    10.4.4 Full (Outer) Join
10.5 SQL Sub-queries
10.6 SQL Constraints
    10.6.1 Not Null Constraint
    10.6.2 Unique Constraint
    10.6.3 Primary key Constraint
    10.6.4 Foreign key Constraint
10.7 Let Us Sum Up
10.8 Further Reading
10.9 Answers to Check Your Progress
10.10 Model Questions

10.1 LEARNING OBJECTIVES

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

- describe Select, Where, Having, Group By clauses
- describe different types of join
- learn Views and Indexes
- describe Queries and Sub-queries
- describe different constraint considerations

10.2 INTRODUCTION

In the previous unit you have been introduced to the concepts of Structured Query Language (SQL). Data Definition Language (DDL) and Data Manipulation Language (DML) SQL commands were introduced in the previous unit. In addition to this, topics like how to create and delete SQL databases and tables and also how to insert and update data in tables were also covered in the earlier unit.

In the present unit you will be introduced to the advanced SQL commands. Clauses like Select, Where, Group By, Having are being discussed in this unit. The different types of Join are also discussed in detail. Besides, Queries and Sub-queries are also introduced and a A brief introduction to views and indexes are also covered in this unit.

10.3 ADVANCED SQL COMMANDS

We have been introduced to basic SQL commands in the previous unit. We already have an idea of inserting and updating data in tables from the previous unit. We will now learn to access these data from tables in different ways. In this section we will go through many advanced SQL commands in detail.

10.3.1 Select

The SELECT statement is used to retrieve data from a database table which is displayed in the form of a database table. We can retrieve data from the database using the select command. The basic syntax for select statement is listed below.

```
SELECT column1, column2, ..., columnN from tablename;
```

In the above syntax, we mention the names of those columns from which we want to retrieve information after the select keyword. The names of these columns are followed by the keyword from followed by the name of the database table. Let the table contain the following

rows as displayed in the figure 1 below.



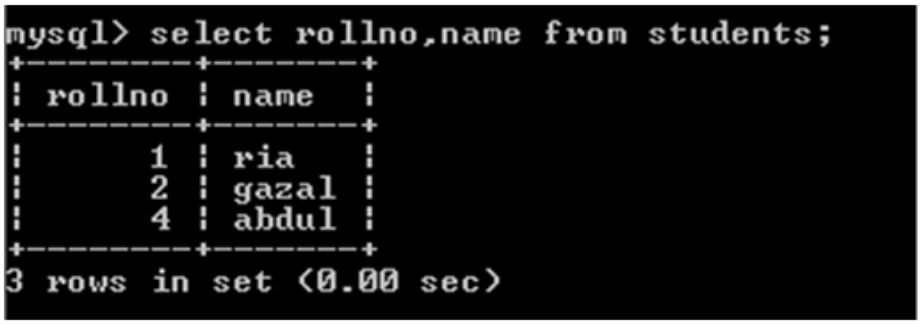
rollno	name	course	date_of_birth
1	ria	mca	1986-07-20
2	gazal	mscit	1990-03-11
4	abdul	mca	1972-08-01

Figure 1: Example showing all records of table "students"

Now, let us try to retrieve only rollno and name information from the "students" table from the database system. We will write the following command to retrieve this information.

```
SELECT rollno, name from tablename;
```

The information that will be displayed by the above statement is shown in figure 2 below.



```
mysql> select rollno,name from students;
+--------+-------+
| rollno | name  |
+--------+-------+
|      1 | ria   |
|      2 | gazal |
|      4 | abdul |
+--------+-------+
3 rows in set (0.00 sec)
```

Figure 2: Example showing rollno and name field information only from table "students"

If we want to retrieve information from all the fields of the database table, then there is another easier way to do it using Select statement. The syntax for retrieving all fields of the database table is given below.

```
SELECT * from tablename;
```

If we want to retrieve information from all the fields of "students" database table, then we use the following statement.

```
SELECT * from students;
```

This statement will display all the records in the students table as displayed in figure 1.

10.3.2 Where

The WHERE clause is used when we want to specify a condition while retrieving data from any database table. It returns as output only those rows which fulfill the specified condition. The syntax of where clause is given below.

```
SELECT column1, column2, ..., columnN
from tablename
WHERE [condition];
```

In the above syntax, we mention the names of those columns from which we want to retrieve information after the select keyword. The names of these columns are followed by the keyword from followed by the name of the database table. Now we specify the condition after the where clause. The condition can be specified by using logical and comparison operators like =, <, >, LIKE, NOT etc. Now, let us try to retrieve only those rollno, name and course field information from the "students" table where the students have opted for MCA course. We will write the following command to retrieve this information.

```
SELECT name, rollno, course
from students
WHERE course='mca';
```

In the above statement, we select to display the name, rollno and course fields of only those students who have chosen their course as MCA as shown in figure 3. It should be noted here that the string should be mentioned inside single quotes.

```
mysql> select rollno, name, course
    -> from students
    -> where course='mca';
+--------+-------+--------+
| rollno | name  | course |
+--------+-------+--------+
|      1 | ria   | mca    |
|      4 | abdul | mca    |
+--------+-------+--------+
2 rows in set (0.03 sec)
```

Fig 3: Example showing use of where clause in table "students"

Now, let us look at another example to show the use of where clause. For this new example, let us consider another table "employee" which has the following records.

```
mysql> select * from employee;
+----+---------------+-----------+--------+
| id | name          | dept      | salary |
+----+---------------+-----------+--------+
|  1 | Pratibha Bora | Accounts  |  35000 |
|  2 | Priya         | Marketing |  40000 |
|  3 | Harry         | Finance   |  25000 |
|  4 | Sunil         | Accounts  |  20000 |
|  5 | Ravi          | Finance   |  35000 |
+----+---------------+-----------+--------+
5 rows in set (0.03 sec)
```

Figure 4: Example showing all records of table "employee"

Suppose, we want to know the name of those employees whose salary is greater than 30,000 from the "employees" table. We can use the Where clause to write this query as given in the following statement.

```
SELECT name

from employee

WHERE salary > 30000;
```

The output of the above statement is displayed in the figure 5.

```
mysql> select name
    -> from employee
    -> where salary > 30000;
+---------------+
| name          |
+---------------+
| Pratibha Bora |
| Priya         |
| Ravi          |
+---------------+
3 rows in set (0.00 sec)
```

Figure 5: Example showing use of where clause in table "employee"

Now, let us look at another example to show the use of Where clause along with Like clause. Suppose, we want to know the names and salary of those employees whose names start from the letter "P" in the "employees" table. We can use the Where clause along with Like clause to write this query.

LIKE Clause

The LIKE clause is used along with WHERE clause to specify a particular pattern. The pattern is expressed with the help of wild cards operators. The pattern may then be used to represent any individual or group of words, numbers or dates. Two commonly used wild car operators are listed below.

" Percent sign (%)

" Underscore (_)

The percent sign represents zero, one or multiple characters. The underscore represents a single character. Both of the wild card operators can be used in combinations. The syntax for using LIKE clause along with WHERE clause id listed below.

```
SELECT column1, column2, ..., columnN
from tablename
where [condition] LIKE [pattern];
```

The above syntax is similar to the where clause syntax except the LIKE part. Here, the desired pattern is specified after the LIKE clause. We can use the where clause along with like clause to write the query to get details of only those employees whose name starts with 'P'. The SQL statement for this query is listed below.

```
SELECT name, salary
from employee
WHERE name LIKE 'P%';
```

In the above statement we select the name and salary fields for display from the table 'employee'. We specify the pattern after the where

and like clause. To specify the pattern that we want all names that start with "P" we denote the pattern using "P%", where P is the initial letter and % means that P can be followed by zero, one or multiple characters . The output of the above statement is displayed in figure 6.

```
mysql> select name, salary
    -> from employee
    -> where name LIKE 'P%';
+---------------+--------+
| name          | salary |
+---------------+--------+
| Pratibha Bora |  35000 |
| Priya         |  40000 |
+---------------+--------+
2 rows in set (0.00 sec)
```

Figure 6: Example showing use of like clause

10.3.3 Order By

The ORDER BY clause is used to sort the output of the query in ascending or descending order according to the columns specified. When we simply use select statement then the output records are not ordered in any specific order. But when we use ORDER BY clause the output is sorted in ascending order by default. The keyword DESC is used to specify if we want the output to be sorted in descending order. The syntax of the ORDER BY clause is listed below.

```
SELECT column1, column2, ..., columnN

from table_name

ORDER BY column1, column2, ..., columnM ASC|DESC;
```

If only one column is specified after the ORDER BY keywords then the resulting table must be displayed in order according to that particular column. However, if more than one columns are specified, then the records are order according to the column whose name appears first, followed by the second and later column names. Let us look at

the example to understand this clause better.

```
SELECT *

from employee

ORDER BY dept, name ASC;
```

In the above SQL statement, we display all the records of the table 'employee' in ascending order first by column name 'dept' and then by column name 'name'. The columns are first ordered according to the first column name that is mentioned and then these are again ordered by the second column name. Hence in this example, the records are ordered in ascending order by 'dept' field values and then if there are same 'dept' values then the records are ordered according to 'name' field. The output of the SQL statement is shown in the figure below.

```
+----+---------------+------------+--------+
| id | name          | dept       | salary |
+----+---------------+------------+--------+
|  1 | Pratibha Bora | Accounts   |  35000 |
|  4 | Sunil         | Accounts   |  20000 |
|  3 | Harry         | Finance    |  25000 |
|  5 | Ravi          | Finance    |  35000 |
|  2 | Priya         | Marketing  |  40000 |
+----+---------------+------------+--------+
5 rows in set (0.03 sec)
```

Figure 7: Example showing use of ORDER BY clause

10.3.4 Group By

The GROUP BY clause is used to group the output records in accordance with one or more columns. Group By clause is used along with the select clause in SQL. The Group by clause is generally used with aggregate functions like COUNT, SUM, Max, MIN etc. The syntax for group by clause is listed below.

```
SELECT column(s)
from table_name
WHERE [condition]
GROUP BY column(s)
ORDER BY column(s);
```

In the above syntax, we choose one or more columns from a table to be displayed in the result set which satisfies the where condition. This is followed by Group By keyword where we specify one or more columns which are to be grouped and then display the result according to some order if the order by clause is used. The Group By clause is generally specified after the where clause and before the order by clause in SQL if the Where and Order By clause are used. Let us look at an example to understand the group by clause.

```
SELECT dept, count(dept)
from employee
GROUP BY dept
ORDER BY count(dept);
```

In this example, we try to count the number of employees in each department. For this, we use the aggregate function COUNT to count the number of times a dept has been repeated in the group. The GROUP BY clause is used to group the records according to their 'dept' values. The ORDER BY clause arranges the output in ascending order of the count of 'dept' values. The output of the above statement is displayed in figure 8.

```
mysql> select dept,count(dept)
    -> from employee
    -> group by dept
    -> order by count(dept);
+-----------+-------------+
| dept      | count(dept) |
+-----------+-------------+
| Marketing |           1 |
| Finance   |           2 |
| Accounts  |           2 |
+-----------+-------------+
3 rows in set (0.00 sec)
```

Figure 8: Example showing use of GROUP BY clause

10.3.5 Having

The HAVING clause is used to set conditions on the SQL query. It is similar to the Where clause which also sets condition. The HAVING clause is used to set conditions on the groups of rows that is returned by GROUP BY clause. Like the WHERE clause the HAVING clause also returns only those groups of records which satisfy the condition. The syntax of HAVING clause is listed below.

```
SELECT column(s)
from table_name
WHERE [condition]
GROUP BY column(s)
HAVING [condition]
ORDER BY column(s);
```

In the above syntax, we can see that the HAVING clause comes after the GROUP BY clause and before the ORDER BY clause in SQL. This order for clauses needs to be followed. Let us look at an example to understand the concept of HAVING clause.

```
SELECT dept, count(dept)
from employee
GROUP BY dept
HAVING count(dept) > 1
ORDER BY dept;
```

In this example, we want to display the departments where the number of employees is greater than 1. The GROUP BY clause is used to group the records according to their 'dept' values. We then use the aggregate function COUNT to count the number of times a dept has been repeated in each group. Now we use the HAVING clause to set the condition to display only those departments whose count is greater than 1. The HAVING clause will now filter all the groups and pass only

those groups that have count more than 1. The ORDER BY clause then arranges the output in ascending order of the names of 'dept' values. The output of the above statement is displayed in figure 9.

```
mysql> select dept, count(dept)
    -> from employee
    -> group by dept
    -> having count(dept) > 1
    -> order by dept;
+----------+--------------+
| dept     | count(dept)  |
+----------+--------------+
| Accounts |            2 |
| Finance  |            2 |
+----------+--------------+
2 rows in set (0.00 sec)
```

Figure 9: Example showing use of HAVING clause

10.3.6 Logical Operators

Logical operators like AND, OR and NOT can be used in SQL. The logical operators are used along with the WHERE clause. The logical operators AND and OR are used to filter records with more than one condition. The AND operator is used when we want all the specified conditions to be fulfilled. The OR operator is used when we want only any one of specified conditions to be fulfilled. The NOT operator is used to display the records which do not fulfill the conditions. The syntax of AND, OR and NOT logical operators is listed below.

```
SELECT column(s)
from table_name
WHERE [condition1] AND [condition2] ... AND [conditionN];
```

```
SELECT column(s)
from table_name
WHERE [condition1] OR [condition2] ... OR [conditionN];
```

```
SELECT column(s)
from table_name
WHERE NOT [condition];
```

The AND, OR and NOT logical operators can be used in combination with each other. Let us look at some of the examples to understand this concept better.

```
SELECT name, dept
from employee
WHERE dept='Accounts' OR dept='Finance';
```

In the above example, we select to display records with only name and department fields from the table 'employee' where the department values may be 'Accounts' or 'Finance'. The output of this query is displayed in figure 10.



Figure 10: Example showing use of OR logical operator

Now, let us try to write a query that will display all fields of the table 'employee' whose departments is other than 'Accounts'. This query can be written as in the SQL statement listed below. The output of this query is displayed in figure 11. In figure 11, it can be seen that only records where department values are not "Accounts" are displayed.

```
SELECT name, dept
from employee
WHERE NOT dept ='Accounts';
```

```
mysql> SELECT name, dept
    -> from employee
    -> WHERE NOT dept ='Accounts';
+-------+-----------+
| name  | dept      |
+-------+-----------+
| Priya | Marketing |
| Harry | Finance   |
| Ravi  | Finance   |
+-------+-----------+
3 rows in set (0.00 sec)
```

Figure 11: Example showing use of NOT logical operator



CHECK YOUR PROGRESS

Q1. Let us consider a table **employee(id, name, department, salary) where** field "id" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.

i. Display all the records with only name and salary fields.
ii. Display the records where salary is greater than 30,000.
iii. Display the name in the table where the name starts from "S".
iv. Display the records in descending order of their departments.
v. Display the records where salary is less than 20000 and department is "Finance".

10.4 JOIN

In SQL, JOIN is used to combine records from two or more tables provided there is a common field between these tables.

The different types of Join are

- Inner Join
- Left (Outer) Join
- Right (Outer) Join
- Full (Outer) Join

Let us look at each of these different types of join in detail.

10.4.1 Inner Join

INNER JOIN is used to retrieve only records that have matching values in all the tables considered. INNER JOIN is also referred to as EQUIJOIN. There must be a common field between all the tables that are considered of inner join. The INNER JOIN will return only those records which have matching values for the common field. The concept of Inner Join can be understood clearly from figure 12 where the shaded part is the selected region from both the tables.

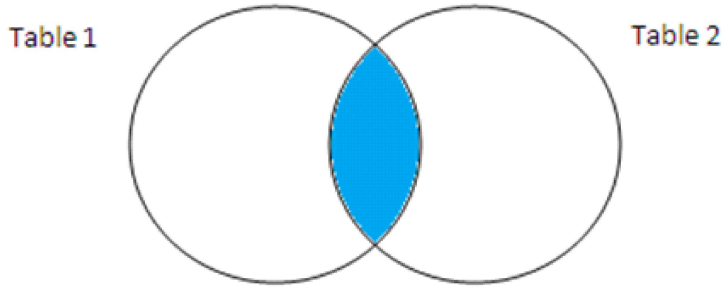


Figure 12: Inner Join

It can be clearly understood from Figure 12 that only the common set values from both the tables are considered in INNER JOIN. The query for INNER JOIN compares each row of the first table with each row of the second table for the common field. The records from both the tables which match the values of the common field are then joined together and displayed as result. The syntax for INNER JOIN is listed below.

```
SELECT table1.column, table2.column

from table1

INNER JOIN table2

ON table1.common_field = table2.common_field;
```

In the above syntax for INNER JOIN, we first select the columns whose values we want to display from both the tables. We then specify the name of the first table after the keyword "'from". The name of the second table is specified after the keywords "inner join". The common field name for both the tables is mentioned after the keyword "on". An "equal to" sign is used between the common field names. Let us look at an example to understand this concept better.

Let us look at two tables "customer" and "odr" with the following structure and record set values. We will use these two tables to explain the different types of join. The structure and record details for table 'customer' are displayed in figure 13. The structure and record details for table 'odr' are displayed in figure 14.

```
mysql> desc customer;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| CustomerName | varchar(30) | YES  |     | NULL    |       |
| CustomerID   | varchar(5)  | YES  |     | NULL    |       |
| City         | varchar(15) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> select * from customer;
+--------------+------------+----------+
| CustomerName | CustomerID | City     |
+--------------+------------+----------+
| bina         | cus01      | ghy      |
| gita         | cus02      | chennai  |
| gita         | cus02      | ghy      |
| mary         | cus03      | shillong |
+--------------+------------+----------+
4 rows in set (0.00 sec)
```

Figure 13: Structure and record values for 'customer' table

```
mysql> desc odr;
+------------+------------+------+-----+---------+-------+
| Field      | Type       | Null | Key | Default | Extra |
+------------+------------+------+-----+---------+-------+
| orderid    | varchar(3) | YES  |     | NULL    |       |
| customerid | varchar(5) | YES  |     | NULL    |       |
| orderdate  | date       | YES  |     | NULL    |       |
+------------+------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> select * from odr;
+---------+------------+------------+
| orderid | customerid | orderdate  |
+---------+------------+------------+
| 001     | cus01      | 1989-07-02 |
| 002     | cus02      | 1995-05-22 |
+---------+------------+------------+
2 rows in set (0.00 sec)
```

Figure 14: Structure and record vlues for 'odr' table

Let us now perform INNER JOIN on these two tables. Let us try to write a SQL statement where we select to display customer names from the "customer" table and order id from the "odr" table after performing inner join on both the tables using the common field "customerid". The SQL query for INNER JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 15. In figure 15, it is seen that only the records that have matching values for the common field "customer id" are displayed.

```
SELECT customer.customername, odr.orderid

from customer

INNER JOIN odr

ON customer.customerid = odr.customerid;
```

```
mysql> select customer.customername,
    -> odr.orderid
    -> from customer
    -> inner join odr
    -> on customer.customerid=odr.customerid;
+--------------+---------+
| customername | orderid |
+--------------+---------+
| bina         | 001     |
| gita         | 002     |
| gita         | 002     |
+--------------+---------+
3 rows in set (0.03 sec)
```

Figure 15: Example showing use of INNER JOIN

10.4.2 Left (Outer) Join

LEFT (OUTER) JOIN or LEFT JOIN is used to retrieve all the records from the left table (or table 1) and only matching records from the right table (table 2). If there are no matching records from the right table then a value of NULL is displayed. Like INNER JOIN there must be a common field between the tables that are considered for LEFT JOIN. The concept of LEFT JOIN can be understood clearly from figure 16 where the shaded part is the region that is selected when we use LEFT JOIN.

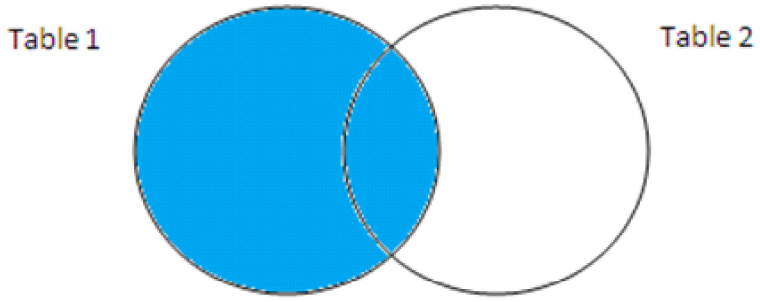


Figure 16: Left (Outer) Join

It can be clearly understood from Figure 16 that all the records from the left table are displayed along with only the matching values from right table in LEFT JOIN. The query for LEFT JOIN compares each row of the first or left table with each row of the second or right table for the common field and displays only those rows of the second table where matching values are found. The records from both the tables are then joined together using the common field and displayed as result. The syntax for LEFT JOIN is listed below.

```
SELECT table1.column, table2.column

from table1

LEFT JOIN table2

ON table1.common_field = table2.common_field;
```

In the above syntax for LEFT JOIN, we first select the columns whose values we want to display from both the tables. We specify the name of the first table after the keyword "'from" and the name of the second table is specified after the keywords "left join". The common field name for both the tables is specified after the keyword "on". An "equal to" sign is used between the common field names. Let us look at an example to understand this concept better.

Let us now perform LEFT JOIN on the tables 'customer' and 'odr' that we have also used in inner join. Let us try to write a SQL statement

where we select to display customer names from the "customer" table and order id from the 'odr' table after performing left join on both the tables using the common field "customerid". The SQL query for LEFT JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 17. In figure 17, it is seen that all the records from table 1 or left table are displayed and only those records of table 2 or right table that have matching values for the common field "customer id" are displayed. For the record with customer id "cus03" in table 'customer' since there is no matching customer id in table 'odr' a NULL value is displayed for "order id" that record.

```
SELECT customer.customerid, customer.customername,
odr.orderid
from customer
LEFT JOIN odr
ON customer.customerid = odr.customerid;
```

```
mysql> select customer.customerid, customer.customername,
    -> odr.orderid
    -> from customer
    -> left join odr
    -> on customer.customerid=odr.customerid;
+------------+--------------+---------+
| customerid | customername | orderid |
+------------+--------------+---------+
| cus01      | bina         | 001     |
| cus02      | gita         | 002     |
| cus02      | gita         | 002     |
| cus03      | mary         | NULL    |
+------------+--------------+---------+
4 rows in set (0.03 sec)
```

Figure 17: Example showing use of LEFT JOIN

10.4.3 Right (Outer) Join

RIGHT (OUTER) JOIN or RIGHT JOIN is used to retrieve all the records from the right table (or table 2) and only matching records from the left table (table 1). If there are no matching records from the left table then

a value of NULL is displayed. Like INNER JOIN and LEFT JOIN there must be a common field between the tables that are considered for RIGHT JOIN. The concept of RIGHT JOIN can be understood clearly from figure 18 where the shaded part is the region that is selected when we use RIGHT JOIN.

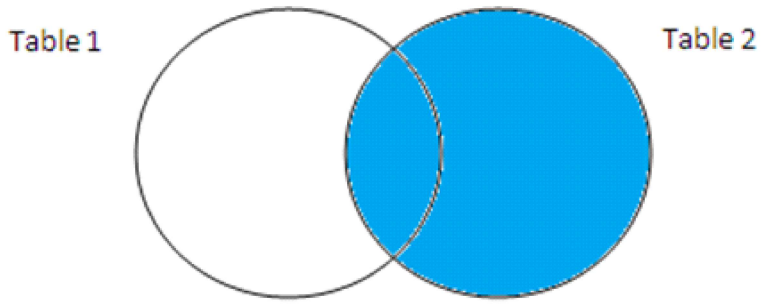


Figure 18: Right (Outer) Join

It can be clearly understood from Figure 18 that all the records from the right table are displayed along with only the matching values from left table in RIGHT JOIN. The query for RIGHT JOIN compares each row of the second or right table with each row of the first or left table for the common field and displays only those rows of the first table where matching values are found for the common field. The records from both the tables are then joined together using the common field and displayed as result. The syntax for RIGHT JOIN is listed below.

```
SELECT table1.column, table2.column
from table1
RIGHT JOIN table2
ON table1.common_field = table2.common_field;
```

In the above syntax for RIGHT JOIN, we first select the columns whose values we want to display from both the tables. We specify the name of the first table after the keyword "'from" and the name of the second table is specified after the keywords "right join". The common field name for both the tables is specified after the keyword "on". An "equal to" sign is used between the common field names. Let us look at an example to understand the concept of right join.

Let us now perform RIGHT JOIN on the tables 'customer' and 'odr' that we have also used earlier. Let us try to write a SQL statement where we select to display customer id and city names from the "customer" table and order id from the 'odr' table after performing right join on both the tables using the common field "customerid". The SQL query for RIGHT JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 19.

In figure 19, it is seen that all the records from table 2 or right table are displayed and only those records of table 1 or left table that have matching values for the common field "customer id" are displayed. So the record with customer id "cus03" is not displayed in the output records since there is no matching value of this customer id in table 'odr'.

```
SELECT customer.customerid, customer.city,
odr.orderid
from customer
RIGHT JOIN odr
ON customer.customerid = odr.customerid;
```

```
mysql> select customer.customerid, customer.city,
    -> odr.orderid
    -> from customer
    -> right join odr
    -> on customer.customerid=odr.customerid;
+------------+---------+---------+
| customerid | city    | orderid |
+------------+---------+---------+
| cus01      | ghy     | 001     |
| cus02      | chennai | 002     |
| cus02      | ghy     | 002     |
+------------+---------+---------+
3 rows in set (0.02 sec)
```

Figure 19: Example showing use of RIGHT JOIN

10.4.4 Full (Outer) Join

FULL (OUTER) JOIN or FULL JOIN is used to retrieve all the records from both the tables when there is a match. If some values are miss-

ing from matched records then NULL values are displayed. Like INNER JOIN, LEFT JOIN and RIGHT JOIN there must be a common field between the tables that are considered for FULL JOIN. The concept of FULL JOIN can be understood clearly from figure 20 where the shaded part is the region that is selected when we use FULL JOIN.

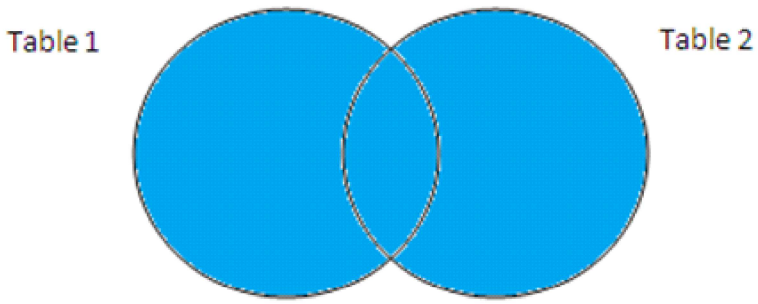


Figure 20: Full (Outer) Join

It can be clearly understood from Figure 20 that all the matched records from both the right and left table are displayed in FULL JOIN. The records from both the tables are then joined together using the common field and displayed as result. The syntax for FULL JOIN is listed below.

```
SELECT table1.column, table2.column

from table1

FULL JOIN table2

ON table1.common_field = table2.common_field;
```

In the above syntax for FULL JOIN, we first select the columns whose values we want to display from both the tables. We specify the name of the first table after the keyword "'from" and the name of the second table is specified after the keywords "full join". The common field name for both the tables is specified after the keyword "on". An "equal to" sign is used between the common field names. Full join is not supported in some database software's like MySQL. So, instead full join is realized by combining left join and right join using UNION. The syntax to achieve FULL JOIN by combining LEFT JOIN and RIGHT JOIN

is listed below. This syntax can be said to be divided into three parts. The first part of the syntax is the syntax for LEFT JOIN, the second part is the keyword "UNION" and the third part is the syntax for RIGHT JOIN. Here, FULL JOIN is achieved by performing union on the individual LEFT JOIN and RIGHT JOIN of the tables.

```
SELECT table1.column, table2.column
from table1
LEFT JOIN table2
ON table1.common_field = table2.common_field
UNION
SELECT table1.column, table2.column
from table1
RIGHT JOIN table2
ON table1.common_field = table2.common_field;
```

Let us look at an example to understand the concept of FULL JOIN. Now let us perform FULL JOIN on the tables 'customer' and 'odr' that we have also used earlier. Let us try to write a SQL statement where we select to display customer id from the "customer" table and order id from the 'odr' table after performing FULL JOIN on both the tables using the common field "customerid". This FULL JOIN is achieved by performing union of LEFT JOIN and RIGHT JOIN of both the tables. The SQL query for FULL JOIN on these two tables is listed below. The output of this SQL query is displayed in figure 21. In figure 21, it is seen that all the records are displayed.

```
SELECT customer.customerid, customer.city,

odr.orderid

from customer

LEFT JOIN odr

ON customer.customerid = odr.customerid

UNION

SELECT customer.customerid, customer.city,

odr.orderid

from customer

RIGHT JOIN odr

ON customer.customerid = odr.customerid;
```

```
mysql> SELECT customer.customerid, customer.city,
    -> odr.orderid
    -> from customer
    -> LEFT JOIN odr
    -> ON customer.customerid = odr.customerid
    -> UNION
    -> SELECT customer.customerid, customer.city,
    -> odr.orderid
    -> from customer
    -> RIGHT JOIN odr
    -> ON customer.customerid = odr.customerid;
+------------+----------+---------+
| customerid | city     | orderid |
+------------+----------+---------+
| cus01      | ghy      | 001     |
| cus02      | chennai  | 002     |
| cus02      | ghy      | 002     |
| cus03      | shillong | NULL    |
+------------+----------+---------+
4 rows in set (0.32 sec)
```

Figure 21: Example showing use of FULL JOIN

10.5 SQL SUB-QUERIES

A subquery is a query written within another query. Subqueries are also called as nested query or inner queries. Subqueries return data to the main query. Subqueries are often used with the SELECT clause and embedded

within the WHERE clause in SQL. Subqueries do not have a defined syntax but it can be defined as in the general syntax listed below.

```
SELECT table1.column, table2.column
from table1
WHERE condition1 IN
(SELECT table1.column, table2.column
from table2
WHERE condition2);
```

The subqueries are written inside brackets. Let us now look at an example to understand how subqueries work.

```
SELECT *
from customer
WHERE customerid IN
(SELECT customerid
from customer
WHERE city='ghy');
```

In the above subquery example, the subquery is used to get the customerid of all customers whose city is "ghy". Then the main query is used to display all the information of these selected customers. The output of the above SQL statement is displayed in figure 22.

```
mysql> SELECT *
    -> from customer
    -> WHERE customerid IN
    -> (SELECT customerid
    -> from customer
    -> WHERE city='ghy');
+--------------+------------+---------+
| CustomerName | CustomerID | City    |
+--------------+------------+---------+
| bina         | cus01      | ghy     |
| gita         | cus02      | chennai |
| gita         | cus02      | ghy     |
+--------------+------------+---------+
3 rows in set (0.01 sec)
```

Figure 21: Example showing use of subquery

10.6 SQL Constraints

Constraints can be said to be rules and conditions that are applied on the data of the database. The constraints can either be enforced when we create the database or even later on as an update to the database. Constraints are used to limit the type of data that we enter in the database. Constraints can be applied to either a specific column of the table or to the database table as a whole.

Different types of constraints can be applied to the database. Some of them are:

- Not Null Constraint
- Unique Constraint
- Primary Key Constraint
- Foreign Key Constraint

Let us look at some of the constraints in detail.

10.6.1 Not Null Constraint

In SQL database tables, by default NULL values are accepted. The NOT NULL constraint is used to limit the database from accepting NULL values in the database table or column. When a NOT NULL constraint is applied on a table or a column it ensures that the table or column cannot have not null values.

NOT NULL constraints are generally applied at the creation of the table. Let us look at an example to understand how this constraint can be applied.

```
CREATE TABLE book
(
      book_id          int              not null,
      book_name        varchar(30)      not null,
      price            int,
      publisher        varchar(10),
      primary key      (book_id)
);
```

In the above example, we have applied NOT NULL constraints on the columns "book_id" and "book_name" in the database table "book". The other columns in this table "book", "price" and "publisher" can accept NULL values since by default NULL values are accepted and no NOT NULL constraint has been applied on these two columns.
NOT NULL values can also be applied after the table has been applied using an UPDATE command. Let us make all the columns of the table "book" not accept NULL values. For this, we will have to update the remaining columns of the table "price" and "publisher" to not accept NULL values. The SQL statement to update the table with NOT NULL constraints for both the columns "price" and "publisher" is listed below.

```
ALTER TABLE book MODIFY
price      int              NOT NULL;


ALTER TABLE book MODIFY
publisher  varchar(10)      NOT NULL ;
```

In the first SQL statement we use ALTER command to modify the column "price" in the table "book" to not accept NOT NULL values. In the second statement, we modify the column "publisher" in the table "book" to not accept NULL values.

10.6.2 Unique Constraint

The UNIQUE constraint is used to limit the columns of database to have identical values. The UNIQUE constraint restricts the database to have unique values for those columns for which this constraint has been applied. Like the NOT NULL constraint the UNIQUE constraint can also be either applied at the creation of the database or after creation. Let us look at an example where the UNIQUE constraint is declared at the creation of the database table.

```
CREATE TABLE book
(
      book_id          int              not null,
      book_name        varchar(30)      not null  unique
);
```

In the above SQL statement, a table "book" has been created where both the columns "book_id" and "book_name" have NOT NULL constraints and the column "book_name" has UNIQUE constraint. This means that no two values for the column "book_name" should have identical values. All the values for "book_name" should be different. The UNIQUE constraint can also be applied afterwards using ALTER TABLE command, if the table has already been created. Let us look at an example where the UNIQUE constraint is declared after the table is already created. Let us make even the "book_id" column of the table "book" have the UNIQUE constraint. For this, we will have to update the table "book". The SQL statement to update the table with UNIQUE constraints for "book_id" column is listed below.

```
ALTER TABLE book MODIFY
book_id    int   NOT NULL UNIQUE;
```

In this SQL statement we use ALTER command to modify the column "book_id" in the table "book" to accept only UNIQUE vales.

10.6.3 Primary Key Constraint

A primary key is a column or a set of columns used to uniquely identify the records in a database table. A primary key cannot have NULL values and all the values for primary key column must have unique values. Since the primary key uniquely define each record in the database table, so for each record to be unique, the values for those col-

umns declared as primary key must not have identical values. Let us look at an example where the PRIMARY KEY constraint is declared at the creation of the database table.

```
CREATE TABLE book
(
      book_id          int              not null,
      book_name        varchar(30)      not null,
      primary key (book_id)
);
```

In the above SQL statement, a table "book" has been created where both the columns "book_id" and "book_name" have NOT NULL constraints and the column "book_id" has PRIMARY KEY constraint. This means that no two values for the column "book_id" should have identical values. All the values for "book_id" should be different.

The PRIMARY KEY constraint can also be applied afterwards using ALTER TABLE command, if the table has already been created. Let us look at an example where the PRIMARY KEY constraint is declared after the table is already created. Suppose we want to add PRIMARY KEY constraint to "book_id" column of the table "book" after the table "book" has already been created. For this, we will have to update the table "book". The SQL statement to update the table with PRIMARY KEY constraints for "book_id" column is listed below.

```
ALTER TABLE book ADD PRIMARY KEY (book_id);
```

In this SQL statement we use ALTER command to add primary key constraint to the column "book_id" in the table "book".

10.6.4 Foreign Key Constraint

A Foreign Key is used as a link between two tables in a database. A foreign key is a column or a collection of columns in one table and a

primary key in the other table. The foreign key is a primary key that uniquely defines the records in one table and all the values of the foreign key in the second table must correspond with those values in the first table. Let us look at an example where a column is declared as PRIMARY KEY in one table and as FOREIGN KEY in another table at the time of creation of both the database tables.

```
CREATE TABLE book
(
      book_id          int              not null,
      book_name        varchar(30)      not null,
      primary key (book_id)
);
```

In the above SQL statement, the column "book_id" is declared as primary key in the first database table "book". In the SQL statement below, the same column will be referenced as foreign key in a second database table "customer".

```
CREATE TABLE customer
(
      customer_id      int              not null,
      customer_name    varchar(30)      not null,
      book_id          int   references book(book_id),
      primary key (customer_id)
);
```

The FOREIGN KEY constraint can also be applied afterwards using ALTER TABLE command, if the second table has already been created. Let us look at an example where the FOREIGN KEY constraint is declared after the table is already created. Suppose we want to add FOREIGN KEY constraint to "book_id" column of the table "customer" after the table "customer" has already been created. For this, we will

have to alter the table "customer". The SQL statement to alter the table with FOREIGN KEY constraints for "book_id" column is listed below.

```
ALTER TABLE customer ADD FOREIGN KEY (book_id)
REFERENCES book(book_id);
```

In this SQL statement we use ALTER command to add foreign key constraint to the column "book_id" in the table "customer", which references it from the column "book_id" in the table "book".

Apart from these four constraints there are many other constraints like Check constraint, Default constraint etc.



CHECK YOUR PROGRESS

Q2. Give the syntax of INNER JOIN.

Q3. Define subqueries.

Q4. Define constraints.

Q5. What is the purpose of using Not Null constraint?



10.7 LET US SUM UP

- The SELECT statement is used to retrieve data from a database table which is displayed in the form of a database table.
- The WHERE clause is used when we want to specify a condition while retrieving data from any database table. It returns as output only those rows which fulfill the specified condition.
- The LIKE clause is used along with WHERE clause to specify a particular pattern. The pattern is expressed with the help of wild cards operators.

- The ORDER BY clause is used to sort the output of the query in ascending or descending order according to the columns specified.
- The GROUP BY clause is used to group the output records in accordance to one or more columns. The Group by clause is generally used with aggregate functions like COUNT, SUM, Max, MIN etc.
- The HAVING clause is used to set conditions on the SQL query. The HAVING clause is used to set conditions on the groups of rows that is returned by GROUP BY clause.
- The logical operators AND and OR are used to filter records with more than one condition. The NOT operator is used to display the records which do not fulfill the conditions.
- INNER JOIN is used to retrieve only records that have matching values in all the tables considered.
- LEFT (OUTER) JOIN or LEFT JOIN is used to retrieve all the records from the left table (or table 1) and only matching records from the right table (table 2). If there are no matching records from the right table then a value of NULL is displayed.
- RIGHT (OUTER) JOIN or RIGHT JOIN is used to retrieve all the records from the right table (or table 2) and only matching records from the left table (table 1). If there are no matching records from the left table then a value of NULL is displayed.
- FULL (OUTER) JOIN or FULL JOIN is used to retrieve all the records from both the tables when there is a match. If some values are missing from matched records then NULL values are displayed.
- A subquery is a query written within another query. Subqueries are also called as nested query or inner queries. Subqueries return data to the main query.
- In SQL database tables, by default NULL values are accepted. The NOT NULL constraint is used to limit the database from accepting NULL values in the database table or column.
- The UNIQUE constraint is used to limit the columns of database to have identical values. The UNIQUE constraint restricts the da-

tabase to have unique values for those columns for which this constraint has been applied.

- A primary key is a column or a set of columns used to uniquely identify the records in a database table.
- A foreign key is used as a link between two tables in a database. A foreign key is a column or a collection of columns in one table and a primary key in the other table.



10.8 FURTHER READING

1. Elsmari, R., & Navathe, S. (2000). Fundamentals of database systems. Addison-Wesley, 106, 107.
2. Singh, S. K. (2011). Database systems: Concepts, design and applications. Pearson Education India.
3. Date, C. J. (2006). An introduction to database systems. Pearson Education India.



10.9 ANSWERS TO CHECK YOUR PROGRESS

Answer to Q1: The statements to perform the functions are:

i. `select name, salary from employee;`

ii. `select * from employee where salary > 30000;`

iii. `select name from employee where name LIKE 'S%';`

iv. `select * from employee order by department desc;`

v. `select * from employee where salary < 20000 AND department='Finance' ;`

Answer to Q2: The syntax of INNER JOIN is:

```
SELECT table1.column, table2.column
from table1
```

```
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

Answer to Q3: A subquery is a query written within another query. Subqueries are also called as nested query or inner queries.

Answer to Q4: Constraints are rules and conditions that are applied on the data of the database. The constraints can either be enforced when we create the database or even later on as an update to the database. Constraints are used to limit the type of data that we enter in the database.

Answer to Q5: The NOT NULL constraint is used to limit the database from accepting NULL values in the database table or column. When a NOT NULL constraint is applied on a table or a column it ensures that the table or column cannot have not null values.



10.10 MODEL QUESTIONS

Q1. Consider a table home_loan (loan_no, account_no, name, total_amount, amount_due, interest, start_date, end_date) where field "loan_no" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.

i. Display the names of customers where the amount of loan is more than 10lakhs.

ii. Display the names and account no of customers who have less than 5lakhs payment due.

iii. Display the number of loans given from 1/1/2011 to 31/12/2016.

iv. Display the account_no of customers whose interest is less than 5%.

v. Perform any function using group by clause.

Q2. Consider a table bank_customer (account_no, name, amount,

account_type, city) where field "account_no" is the primary key. Insert five records in this database table. Write the statements in SQL to perform the following functions.

i. Display the names of customers where name starts with "As".
ii. Display the names of customers whose account has more than 50000 but less than 200000.
iii. Display the names and account_no of customers with city as "Guwahati".
iv. Display the account_no of customers whose account_type is not "savings"
v. Perform any function using Having By clause.
vi. Display the number of account holders with city as "Guwahati".

Q3. Perform Inner Join on the tables "bank_customer" and "home_loan".

Q4. Perform Left Join on the tables "bank_customer" and "home_loan".

Q5. Describe NOT NULL constraints. Why do we need NOT NULL constraints?

Q6. Explain the concept of Right Join.

Q7. Describe subqueries using examples.

Q8. Describe the use of different logical operators in database.

Q9. Describe Primary key and Foreign key constraints.

Q10. Perform FULL OUTER JOIN on the tables ""bank_customer" and "home_loan".

*****

UNIT 11: DATABASE RECOVERY

UNIT STRUCTURE

11.1 Learning Objectives

11.2 Introduction

11.3 Database Recovery

11.4 Database Failure

11.5 Backup of Database for Catastrophic Failure

11.6 Recovery for Non-Catastrophic Failures

11.7 Let Us Sum Up

11.8 Further Reading

11.9 Answers to Check Your Progress

11.10 Model Questions

11.1 LEARNING OBJECTIVES

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

- identify the types of database failure
- explain the types of database failure
- learn about database recovery
- learn to back up as database

11.2 INTRODUCTION

We have already been acquainted with various important concepts related with database and DBMS. We have also learnt to insert, delete, update and other associated operations in a database through SQL commands. In this unit we will learn how a database can be recovered.

A database may become inconsistent because of a transaction failure, database system failure or media crash. Now in this case bringing the database into the last consistent state, which existed prior to the failure and to preserve transaction properties; recovery mechanism is a must.

In this unit we will learn about the types of failures, concept of database recovery and its need. At the end of this unit, we discuss some of the techniques that can be used for database recovery from these failures.

11.3 DATABASE RECOVERY

Database recovery is the process of restoring the database to a consistent state that existed just before the time of failure. In this process, the information about the changes which are applied to data items carried out by various transactions are usually kept by the system. Typically this information is kept by the *system log.* System log files may contain information about device changes, device drivers, system changes, operations and more.

11.4 DATABASE FAILURES

A computer system, like any other device, is subject to failure from variety of causes. There are various types of failure that may occur in a system. Different failures are handled with different manner. In some failure loss of information does not occur. The failures that are more difficult to deal with are those that result in loss of information. Some failures affect the main memory only, while others involve secondary storage.

Failures are generally classified as catastrophic and non-catastrophic failure. System crashes, media failures etc. fall under catastrophic failure and transaction failure fall under non-catastrophic failure.

CATASTROPHIC FAILURE

In case of catastrophic failure like disk crash, there is an extensive damage to a wide portion of the database. In this condition, the recovery method restores a past copy of database that was backed up earlier to the archival storage and also reconstructs a more current state by reapplying and redoing the operations of committed transactions from the backed up log, to the time of failure. Some examples under this category: *disk head crash*, *power disruption*, *media failure* etc.

NON-CATASTROPHIC FAILURE:

In case of non-catastrophic failure, the database on the disk is not physically damaged. Therefore, the main recovery strategy is to identify the changes that may cause any inconsistency in the database. For example, a transaction that has updated some database items on disk but has not committed needs to have its changes reversed by undoing its write operation. It may also be necessary to redo some operations in order to restore a consistent state of the database. Here, a complete archival copy of database is not necessarily needed by the recovery protocol. Rather, to determine the appropriate actions for recovery, the entries kept in online system log on disk are analyzed. Some example: Transaction error due to incorrect input, deadlock, incorrect synchronization etc.

Let us discuss briefly discuss different types of failures:

1. System crashes: System crashes are due to hardware malfunction or software errors or the operating system failure, resulting in the loss of the content of main memory. The following failures can be included in system crash.

- **Hardware failures:** Hardware failures may include disk full errors, memory errors, disk crashes, bad disk sectors and so on.

- **Software failures:** Software failure generally include operating system failure, DBMS failure, application program failure and so on i.e., failures related to software.

- **Network failures:** Network failure usually occur while using a client-server system or a distributed system where multiple servers are connected by common networks.

2. Transaction failure: A a transaction may fail due to *logical error* and s*ystem error.*

- **Local errors:** Sometime while executing a transaction, certain condi tions may occur that necessitate cancellation of the transaction. Some examples of logical errors are wrong input, overflow, indufficient data for the transaction etc. For instance, some exceptional situa tion may arise such as insuffiecient account balance in a banking database may cause a transaction like fund withdrawal to be can celed.

- **System errors:** Normal execution of a transaction may be inturrupted as the system enter an undesirable state such as deadlock. The transaction, however, can be reexecuted. Such type of errors fall under system error.

3. Media failures: Media failures are most dangerous. It occurs due to head crashes or unreadable media which results loss of parts of second-ary storage. They are the most danderous failure.

4. Natural disaster failures: We may sometime loss our databases due to natural disaster such as fires, power failures, floods or earthquake.

5. Carelessness: Sometime we may loss our important databases due to our carelessness or carelessness of users or operators.

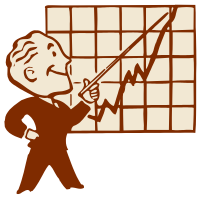
6. Sabotage: These failures are occur due to intentional corruption or de-struction of data, hardware, or software facilities.

11.5 BACKUP OF DATABASE FOR CATASTROPHIC FAILURE

In catastrophic failure, the main technique used is a database back up. In this process, the whole database and the log is periodically copied to

a storage medium like magnetic tapes or other large capacity offline storage devices. In case of catastrophic system failure, the latest back up copy can be reloaded from the tape to the disk, and the system can be restarted. Data from critical applications such as banking, insurance, stock market, and other databases are periodically backed up and moved to a physically safe separate locations.

To avoid loss of effects of transactions that have been executed since the last back up, it is mandatory to back up the system log at more frequent intervals than full database back up by periodically copying it to magnetic tape. The system log can be backed up more frequently as it is usually smaller than the database. Therefore, users do not have to lose all the transactions being performed.



CHECK YOUR PROGRESS

1. Discuss the different types of transaction failures. What is meant by catastrophic failure?
2. How catastrophic failure and non-catastrophic failure recovery techniques are different?
3. Discuss how disaster recovery from catastrophic failures is handled.

11.6 RECOVERY FOR NON-CATASTROPHIC FAILURES

Recovery mechanism from Non-catastrophic transaction failures are of two types: *deferred update* and *immediate update*.

11.6.1 Deferred Update

In deferred update strategy, database is modified only after reaching the commit point of the transactions. The idea behind

deferred update is to postpone any actual updates to the database on disk until the transaction reach its commit point by executing successfully. During the execution of the transaction, the updates are recorded only in the log and in the cache buffers. When the transaction reaches its commit point and the log is forced written to the disk, the updates are recorded in the database. If before reaching its commit point the transaction fails, there is no need to undo any operations as the transaction has not affected the database on disk in any way. We can summerized the deferred update recovery in the following steps:

- A transaction cannot change the database on disk until the commit point is achieved.
- A transaction cannot reach the commit point until all its update operations are written onto log and the log is force written to disk.

11.6.2 Immediate Update

In this technique when a transaction issues an update command, immediately the database on disk can be updated. Here, it is not required to wait for the transaction to reach the commit point. Provisions must be made for undoing the effect of updated operations that have been applied by the failed transactions to the database. This is performed by rolling back the transaction and undoing the effect of the transaction's write item operations. Therefore, the UNDO-type log entries that include the old value of the items must be stored in the log. Two main types of immediate update algorithms are:

• UNDO/NO-REDO recovery algorithm
• UNDO/REDO recovery algorithm

Undo/No-Redo Recovery Algorithm

The recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is no need to REDO any operations of committed transactions

ever. In this method, all the updates by the transaction must be recorded on disk before reaching the commit point, so that REDO is never needed. It must utilize the force strategy for deciding when update main memory buffers are written back to disk.

Undo/Redo Recovery Algorithm

In this technique the transaction is allowed to commit before all its changes are written to the database.



CHECK YOUR PROGRESS

4. What is the difference between the UNDO/REDO and the UNDO/NO-REDO algorithms for recovery with immediate update?



11.7 LET US SUM UP

In this chapter we have discussed about various types of database failures: catastrophic failures and non-catastrophic failure. The database may lose transaction consistency because of the occurrence of these failures. Because of this various recovery techniques need to be implemented. Here we have discussed various database recovery techniques for this very purpose.



11.8 FURTHER READING

1. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems. Pearson.
2. Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. McGraw Hill.
3. Singh, S.K.., Database Systems: Concepts, Design and Applications



11.9 ANSWERS TO CHECK YOUR PROGRESS

1. Catastrophic and Non-Catastrophic Failure. If there is extensive damage to a wide portion of the database then it is called Catastrophic Failure.
2. In catastrophic failure, there is extensive damage to a wide portion of the database; while if a non-catastrophic failure has occurred then the database on disk is not physically damaged.
3. In case of a catastrophic failure like such as disk crashes, the main technique used is a database backup, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices.
4. In Undo/Redo technique the transaction is allowed to commit before all its changes are written to the database; while in Undo/No-Redo method, all updates by a transaction must be recorded on disk before the transaction commits, so that REDO is never needed.



11.10 MODEL QUESTIONS

1. Discuss the actions taken by the read_item and write_item operations on a database.
2. What is the system log used for? What are the typical kinds of entries in a system log? What are checkpoints, and why are they important? What are transaction commit points, and why are they important?
3. How are buffering and caching techniques used by the recovery subsystem?
4. What are the before image (BFIM) and after image (AFIM) of a data item? What is the difference between in-place updating and shadowing, with respect to their handling of BFIM and AFIM?
5. What are UNDO-type and REDO-type log entries?
6. Describe the write-ahead logging protocol.

7. Identify three typical lists of transactions that are maintained by the recovery subsystem.
8. What is meant by transaction rollback? What is meant by cascading rollback? Why do practical recovery methods use protocols that do not permit cascading rollback? Which recovery techniques do not require any rollback?
9. Discuss the UNDO and REDO operations and the recovery techniques that use each.
10. How can recovery handle transaction operations that do not affect the database, such as the printing of reports by a transaction?
11. Discuss the immediate update recovery technique in both single-user and multiuser environments. What are the advantages and disadvantages of immediate update?
12. Describe the shadow paging recovery technique. Under what circumstances does it not require a log?

*****

UNIT 12: DATABASE SECURITY

UNIT STRUCTURE

12.1 Learning Objectives
12.2 Introduction
12.3 Database Security
12.4 Access Control
12.4.1 Discretionary Access Control
12.4.2 Mandatory Access Control
12.5 Cryptanalysis
12.6 Data Encryption
12.7 Let Us Sum Up
12.8 Further Reading
12.9 Answers to Check Your Progress
12.10 Model Questions

12.1 LEARNING OBJECTIVES

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

- learn about database security
- describe the need of database security
- define access control
- explain various data encryption methods

12.1 INTRODUCTION

In the previous unit we have discussed various types of database failures and recovery mechanisms. In this unit, we aregoing to discuss the techniques for making a database secure from variety of threats. We will also discuss the process of granting access privileges to authorized users at the end of this unit.

12.3 DATABASE SECURITY

The basic security goals for any database are:

Confidentiality: Database confidentiality refers to the protection of data from unauthorized disclosure.

Integrity: Database integrity refers to the requirement that information be protected from improper modification.

Availability: Resources should be available to human user or a program having legitimate right, that is what database availability means.
There exist some threats that may lead to degrade or loss in achieving these security goals. Therefore, to protect the database, different control measures are used, following are some of them:

- Access control
- Inference control
- Flow control
- Encryption

12.4 ACCESS CONTROL

The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function, called access control, is handled by creating user accounts and passwords to control the login process by the DBMS.

12.4.1 Discretionary Access Control

The typical method of enforcing discretionary access control in a database system is based on the granting and revoking of privileges. This is an all-or-nothing method: A user either has or does not have a certain privilege.

Types of Discretionary Privileges:
The account level: At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.

The relation level: At this level, the DBA has the privilege to control the access of each individual relation or view in the database.

Revoking of Privileges:

In some cases it is desirable to grant a privilege to a user temporarily. Hence, a mechanism for revoking privileges is needed. For the purpose of cancelling privileges given, one have to use REVOKE command in SQL.

12.4.2 Mandatory Access Control

In many applications, an additional security policy is needed that classifies data and users based on security classes. This approach, known as mandatory access control (MAC), would typically be combined with the discretionary access control mechanisms. Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest.

Two main restrictions are imposed on data access depending upon the subject or object classifications:

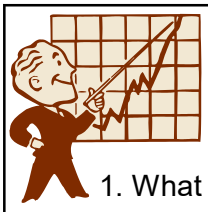
1. A subject S is not allowed read access to an object O unless class(S) e” class(O). This is known as the simple security property.
2. A subject S is not allowed to write an object O unless class(S) d” class(O). This is known as the star property (or *-property).

12.5 STATISTICAL DATABASES

Statistical databases are used mainly to produce statistics about various populations. The database may contain confidential data about individuals, which should be protected from user access. However, users are permitted to retrieve statistical information about the populations, such as averages, sums, counts, maximums, minimums, and standard deviations.

A population is a set of tuples of a relation (table) that satisfy some selection condition. Statistical queries involve applying statistical functions to a population of tuples. Security techniques used in statistical database must have to protect the confidentiality of individual information.

This can be achieved by prohibiting queries that retrieve attribute values and by allowing queries that involve only statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION. Such queries are sometimes called statistical queries.

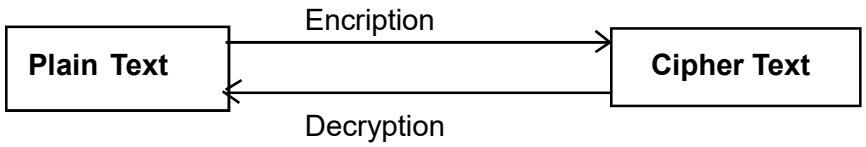


CHECK YOUR PROGRESS

1. What are the basic security goals for any database?
2. How to protect databases from threats?
3. What is the use of Statistical Database?
4. What is a population?

12.6 DATA ENCRYPTION

Encryption is the conversion of data into a form, called a cipher text, which cannot be easily understood by unauthorized persons. It enhances security and privacy when access controls are bypassed, because in cases of data loss or theft, encrypted data cannot be easily understood by unauthorized persons.



A message is plaintext. The process of disguising a message in such a way as to hide its substance is encryption. An encrypted message is ciphertext. The process of turning ciphertext back into plaintext is decryption. Mathematically,

$$E(M) = C; D(C) = M$$

where M is the message , C is the ciphertext , E is the encryption process and D is the decryption process.

A cryptographic algorithm, also called a cipher, is the mathematical function used for encryption and decryption. Keys are used for encryption and decryption process.

There are two general types of key-based algorithms: *symmetric* and *public-key*.

Symmetric key algorithms use the same key for both the encryption and the decryption process. The key is kept secret.

$$E_k(M) = C;\ D_k(C) = M$$

In public key cryptography different keys are used for encryption and decryption process. A pair of keys is generated; out of which one key is made available to all, while the other is kept secret.



CHECK YOUR PROGRESS

5. What is data Encryption?
6. Explain public key cryptography.

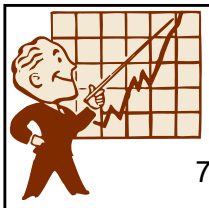
12.5. CRYPTANALYSIS

Cryptanalysis is the science of extracting the plaintext of a cipher message without knowing the key. It shows the vulnerability of a cryptosystem. An attempted cryptanalysis is called an attack.

There are four general types of cryptanalytic attacks, assuming that the cryptanalyst has complete knowledge of the encryption algorithm used:

- Ciphertext-only attack.
- Known-plaintext attack
- Chosen-plaintext attack
- Adaptive-chosen-plaintext attack

Good cryptosystems are designed to be infeasible to break with the computing power that is expected to evolve many years in the future.



CHECK YOUR PROGRESS

7. What is Cryptanalysis? How many different types of Cryptanalysis attack are possible?



12.7 LET US SUM UP

In this chapter we have discussed about achieving security goals of a database. Confidentiality, Integrity and availability of a database need to be maintained. To protect a database from threat different control measures are taken such as access control, inference control, flow control and encryption. All the access control mechanisms are discussed in detail. Then we have discussed about the encryption technique, where one can encode the massage using different techniques. The idea of secret key and public key are discussed in detail. Finally we have discussed about vulnerability of a cryptosystem with various possible types of attack.



12.8 FURTHER READING

1. Elmasri R. and Navathe B, “Fundamentals of Database Systems”.
2. Ramakrishnan R., “Database Management Systems”.
3. Schneier B., “Applied Cryptography”.



12.9 ANSWERS TO CHECK YOUR PROGRESS

1. The basic security goals for any database are:
   - Confidentiality
   - Integrity
   - Availability
2. For protecting databases four different control measures are taken:
   - Access Control
   - Inference Control
   - Flow Control
   - Encryption.
3. Statistical databases are used mainly to produce statistics about various populations. Statistical information about the populations, such as averages, sums, counts, maximums, minimums, and standard deviations are furnished keeping personal information hidden.
4. A population is a set of tuples of a relation (table) that satisfy some selection condition.

5. Encryption is the conversion of data into a form, called a cipher text, which cannot be easily understood by unauthorized persons. It enhances security and privacy when access controls are bypassed, because in cases of data loss or theft, encrypted data cannot be easily understood by unauthorized persons.

6. In public key cryptography different keys are used for encryption and decryption process. A pair of keys is generated; out of which one key is made available to all, while the other is kept secret.

7. Cryptanalysis is the science of extracting the plaintext of a cipher message without knowing the key.
   Four different attacks are possible:.
   - Ciphertext-only attack.
   - Known-plaintext attack
   - Chosen-plaintext attack
   - Adaptive-chosen-plaintext attack



12.10 MODEL QUESTIONS

Q1.Discuss what is meant by each of the following terms: database authorization, access control and data encryption.

Q2. What is a statistical database? Discuss the problem of statistical database security.

Q3.How is privacy related to statistical database security? What measures can be taken to ensure some degree of privacy in statistical databases?

Q4.What is flow control as a security measure? What types of flow control exist ?

Q5.How can privacy of data be preserved in a database?

Q6.What are some of the current outstanding challenges for database security?

Q7.Explain RSA algorithm.

Q8. What is a symmetric key algorithm for key-based security?

Q9. What is the public key infrastructure scheme? How does it provide security?

*****