Module I

Chapter 1

Introduction to Databases and Transactions

Syllabus

What is database system, purpose of database system, view of data, relational databases, database architecture, transaction management

Module I

Chapter 2

Data Models

Syllabus

The importance of data models, Basic building blocks, Business rules, The evolution of data models, Degrees of data abstraction

Module I

Chapter 3

Database Design, R-Diagram

Syllabus

Database design and ER Model: overview, ER-Model, Constraints, ER-Diagrams, ERD Issues, Codd's rules, Relational Schemas

Module I

CHAPTER 4

Relational Database Model

Syllabus

Logical view of data, keys, integrity rules

Module II

Chapter 5

Database Design Theory and Normalization

Syllabus

Basics of functional dependencies and normalization for relational databases. Relational database design and further dependencies.

Module II

CHAPTER 6

SQL, Indexing

Syllabus

Introduction to SQL, Complex queries, triggers, views, joining database tables and schema modification. Query Processing and optimization. File structure, hashing and indexing

Module II

Chapter 7

Transaction Management and Concurrency Control and Recovery

Syllabus

Introduction to transaction processing concepts and theory. Concurrency control technique. Database recovery technique

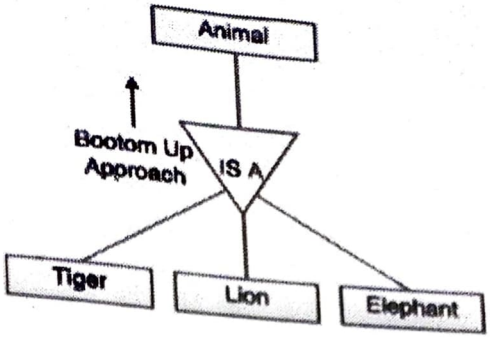


Fig. P15 : Generalization

Practical 2 : Perform the following :

- Viewing all databases
- Creating a Database
- Viewing all Tables in a Database
- Creating Tables (With and Without Constraints)
- Inserting/Updating/Deleting Records in a Table

Soln. :

Viewing all databases : The SQL **SHOW DATABASES** statement is used to list down all the available databases in MySQL database.

```
show databases;
```

Creating a Database

Syntax : CREATE DATABASE *databasename*;
Example : CREATE DATABASE fyit;

Viewing all Tables in a Database

SQL command to list all tables in MySQL

- To list all tables in MySQL, first, you connect to the MySQL database server using the following command:

  mysql -u username -p

- MySQL then prompts for the password; just enter the correct one for the user and press enter.
- After that, select a database to work with:

  use database_name;

- And finally, issue the SHOW TABLES command to display all tables in the current database:

  show tables;

SQL command to list all tables in SQL Server

*SELECT * FROM information_schema.tables;*

Creating Tables (With & Without constraints)

Creating Table (without constraints)

Student Table

```
CREATE Table student (roll_no integer(3), stud_name varchar(20), bdate date , marks integer(3));
```

2. Employee Table

```
CREATE Table employee
(emp_id integer(3), emp_name varchar(20), Salary integer(7), department varchar(20));
```

☞ **Creating Table (with constraints)**

1. Primary Key Constraint

```
Create table emp(eno number(5) constraint cns1 primary key)
```

2. Foreign Key Constraint

```
Create table student(stud_id number(5), name varchar2(10), course_code references
course_details(course_code));
```

3. UNIQUE Key Constraint

```
Create table emp(eno number(5) constraint cns2 Unique, ename varchar2(10));
```

4. NULL

The NULL keyword is used to specify that a column can store the NULL value for its data type. This implies that the column need not receive any value during insert or update operations. The NULL constraint is logically equivalent to omitting the NOT NULL constraint from the column definition.

The following example creates the newitems table. In newitems, the column description does not have a default value, but it allows NULL values.

Example

```
CREATE TABLE newitems (
 newitem_num number(3),
 promotype number(5),
 descrip varchar2(20) NULL);
```

5. NOT NULL

```
Create table emp(eno number(5) constraint cns3 not null, ename varchar2(10));
```

6. CHECK

```
Create table stud(rno number(5), ename varchar2(10), age number(5) constraint cns4 check age between 15 and
20);
```

☞ **Inserting/Updating/Deleting Records in a table.**

(i) Inserting

1. Inserting values in all columns

Example

```
Insert into emp values(106,'Rajesh','Clerk',12000);
```

2. Inserting values in specific columns

```
insert into emp(Eno,Ename,Job) values(107,'Ankur','Salesman');
```

3. Inserting records from existing table into new table

```
Insert into emp1
Select eno,ename,job,sal from emp
Where sal > 15000;
```

Output

Emp1

Eno	Ename	Job	Sal
102	Dinesh	Manager	18000
104	Bharati	Manager	17000

(ii) Updating

Example

```
Update emp set sal = 15000;
```

This query will make salary of all the employees to 15000.

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	15000
102	Dinesh	Manager	15000
103	Dinesh	Clerk	15000
104	Bharati	Manager	15000
105	Prajakta	Salesman	15000

Example

```
Update emp set sal = 20000 where job = 'Manager';
```

This query will change salary of mangers to 20000.

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	15000
102	Dinesh	Manager	20000
103	Dinesh	Clerk	15000
104	Bharati	Manager	20000
105	Prajakta	Salesman	15000

(iii) Deleting

Example

```
Delete from emp;
```

This query will delete all the records from Table L.4.

```
Delete from emp where Eno = 103;
```

▶ **Practical 3 : Perform the following :**

- **Altering a Table**
- **Dropping/Truncating/Renaming Tables**
- **Backing up / Restoring a Database**

☑ **Soln. :**

(I) Altering a table

(i) Adding New Column in a Table

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example : Adding column grade in the table student.

```
ALTER TABLE student
ADD Grade varchar(2);
```

(ii) Dropping Column from Table

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example : Deleting column grade from the table student.

```
ALTER TABLE student
DROP column Grade;
```

(iii) Modifying Column of a Table

```
ALTER TABLE table_name
MODIFY COLUMN column_name data_type;
```

Example : Changing the data type and size of column roll_number of student table.

```
ALTER TABLE student
modify column roll_no varchar(4);
```

(II) Dropping/Truncating/Renaming Tables

(i) Drop Table

Example : Deleting the newstudent1 table from the database.

```
Drop table newstudent1;
```

(ii) runcate Table

Example : Deleting all the records from newstudent1

```
TRUNCATE TABLE newstudent1;
```

(iii) Rename

Example : Deleting the newstudent1 table from the database.

```
alter table newstudent1
rename to newstudent2;
```

(3) Backing up/Restoring a Database

The **BACKUP DATABASE** statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax : BACKUP DATABASE *databasename*
TO DISK = '*filepath*';

Example : BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak';

▶ **Practical 4 : Perform the following :**

- **Simple Queries**
- **Simple Queries with Aggregate functions**

☑ **Soln. :**

Employee table :

Id	Name	Salary
1	A	800
2	B	400
3	C	600
4	D	700
5	E	600
6	F	Null

(1) Min()

Query

```
Select min(salary) from Employee;
```

Output

```
400
```

(2) Max()

Query

```
Select max(salary) from Employee;
```

Output

```
800
```

(3) Sum()

Query

```
Select sum(salary) from Employee;
```

Output

```
3100
```

(4) Avg()

Query

```
Select avg(salary) from Employee;
```

Output

```
620
```

(5) Count()

Query

```
Select count(Name) from Employee;
```

Output

```
6
```

▶ **Practical 5 : Queries involving**

- **Date Functions**
- **String Functions**
- **Math Functions**

☑ **Soln. :**

(A) Date functions

1. **ADD_MONTHS(d,n) : Return date after adding the number of months specified in the function.**

```
SQL>Select add_months(sysdate,4) from dual;
```

2. **LAST_DAY : Returns the last date of the month specified with the function.**

```
SQL> Select sysdate, last_day(sysdate) from dual;
```

3. **MONTHS_BETWEEN (d1,d2) : Returns number of months between d1 and d2.**

```
SQL> Select months_between('02-feb-06','02-jan-07') from dual;
```

4. **NEXT_DAY (date,weekday) : Returns the date of the first weekday coming after the date.**

```
SQL> Select next_day('06-jul-02','monday') from dual;
```

Practical 6 : Join Queries

- **Inner Join**
- **Outer Join**

Soln. :

1. Inner Join :

Consider following two tables.

```
mysql> select * from product_details;
+------------+--------------+----------+-------+
| product_id | product_name | quantity | price |
+------------+--------------+----------+-------+
|       1001 | pendrive     |      100 |   900 |
|       1002 | harddisk     |      200 |  4000 |
|       1003 | headphone    |     1000 | 15000 |
|       1004 | DVD          |       20 |  1000 |
|       1005 | speaker      |      600 |  2400 |
+------------+--------------+----------+-------+
5 rows in set (0.00 sec)

mysql> select * from sale_details;
+---------+------------+----------+-------+---------------+
| sale_no | product_id | quantity | price | customer_name |
+---------+------------+----------+-------+---------------+
|    2001 |       1001 |       50 |   900 | savni         |
|    2002 |       1004 |       10 |  1000 | savni         |
|    2003 |       1003 |      120 | 15000 | savni         |
|    2004 |       1005 |      420 |  2400 | harsh         |
|    2005 |       1002 |       40 |  4000 | Akash         |
+---------+------------+----------+-------+---------------+
5 rows in set (0.02 sec)

mysql>
```

Inner Join (Equi Join)

The INNER JOIN is used to display records that have matching values in both tables.

Syntax :

Select column_name_list from table_1 INNER JOIN table_2where table_1.column_name = table_2.column_name

For example:

select product_details.product_id, product_name, customer_name, sale_details.quantity, product_details.price from product_details INNER JOIN sale_details on product_details.product_id=sale_details.product_id ;

```
mysql> select product_details.product_id, product_name, customer_name , sale_de
tails.quantity, product_details.price from product_details INNER JOIN sale_detai
ls on product_details.product_id=sale_details.product_id ;
+------------+--------------+---------------+----------+-------+
| product_id | product_name | customer_name | quantity | price |
+------------+--------------+---------------+----------+-------+
|       1001 | pendrive     | savni         |       50 |   900 |
|       1004 | DVD          | savni         |       10 |  1000 |
|       1003 | headphone    | savni         |      120 | 15000 |
|       1005 | speaker      | harsh         |      420 |  2400 |
|       1002 | harddisk     | Akash         |       40 |  4000 |
+------------+--------------+---------------+----------+-------+
5 rows in set (0.02 sec)

mysql>
```

2. Outer Join :

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

(i) Left Outer Join
(ii) Right Outer Join

(i) Left Outer Join

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. Null values are shown at the place of right table values.

Syntax

```
SELECT column-name-list
from table-name1
LEFT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Example

```
SELECT product_details.product_id, product_details.product_name, sale_details.customer_name
FROM product_details
LEFT OUTER JOIN
sale_details ON sale_details.product_id =product_details.product_id;
```

(ii) Right Outer Join

Returns all rows from the right table even if there are no matches in the left table. Null values are shown at the place of left table values.

Syntax

```
select column-name-list
from table-name1
RIGHT OUTER JOIN
table-name2
on table-name1.column-name = table-name2.column-name;
```

Example

```
SELECT product_details.product_id, product_details.product_name, sale_details.customer_name
FROM product_details
RIGHT OUTER JOIN sale_details
ON sale_details.product_id =product_details.product_id;
```

Practical 7 : Subqueries

- **With IN clause**
- **With EXISTS clause**

Soln. : Consider below tables :

```
mysql> select * from employee;
+--------+----------+-----------+--------+--------+------+
| EMP_ID | EMP_NAME | PHONE_NO  | SALARY | CITY   | AGE  |
+--------+----------+-----------+--------+--------+------+
|      1 | sameer   |  80800443 |  20000 | mumbai |   30 |
|      2 | ajay     |  80800443 |  30000 | nagpur |   26 |
|      3 | SAM      | 930303030 |  23000 | VASAI  |   27 |
|      4 | KIM      | 930303030 |  25000 | VIRAR  |   27 |
|      5 | JACK     | 930303030 |  25000 | VIRAR  |   36 |
+--------+----------+-----------+--------+--------+------+
```

With IN clause :

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
mysql> select * from employee where CITY in('virar');
+--------+----------+-----------+--------+--------+------+
| EMP_ID | EMP_NAME | PHONE_NO  | SALARY | CITY   | AGE  |
+--------+----------+-----------+--------+--------+------+
|      4 | KIM      | 930303030 |  25000 | VIRAR  |   27 |
|      5 | JACK     | 930303030 |  25000 | VIRAR  |   36 |
+--------+----------+-----------+--------+--------+------+
2 rows in set (0.00 sec)

mysql>
```

With EXISTS clause : EXISTS is a Boolean operator which checks the subquery result and returns an either TRUE or FALSE value.

```
SELECT col_names FROM tab_name WHERE [NOT] EXISTS ( SELECT col_names
FROM tab_name WHERE condition );
```

```
mysql> SELECT * FROM customer;
+---------+---------+------------+------+
| cust_id | name    | occupation | age  |
+---------+---------+------------+------+
|     101 | Peter   | Engineer   |   32 |
|     102 | Joseph  | Developer  |   30 |
|     103 | John    | Leader     |   28 |
|     104 | Stephen | Scientist  |   45 |
|     105 | Suzi    | Carpenter  |   26 |
|     106 | Bob     | Actor      |   25 |
|     107 | NULL    | NULL       | NULL |
+---------+---------+------------+------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM Orders;
+----------+---------+-----------+------------+
| order_id | cust_id | prod_name | order_date |
+----------+---------+-----------+------------+
|        1 |     101 | Laptop    | 2020-01-10 |
|        2 |     103 | Desktop   | 2020-02-12 |
|        3 |     106 | Iphone    | 2020-02-15 |
|        4 |     104 | Mobile    | 2020-03-05 |
|        5 |     102 | TV        | 2020-03-20 |
+----------+---------+-----------+------------+
```

```
mysql> SELECT name, occupation FROM customer
    -> WHERE EXISTS (SELECT * FROM Orders
    -> WHERE customer.cust_id = Orders.cust_id);
+---------+------------+
| name    | occupation |
+---------+------------+
| Peter   | Engineer   |
| Joseph  | Developer  |
| John    | Leader     |
| Stephen | Scientist  |
| Bob     | Actor      |
+---------+------------+
```

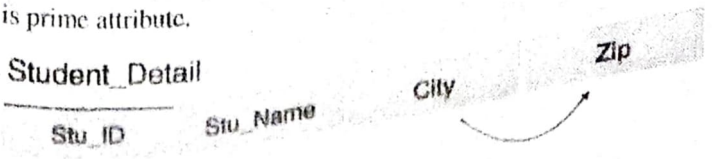
▶ **Practical 8 : Converting ER Model to Relational Model and apply Normalization on database. (Represent entities and relationships in Tabular form, Represent attributes as columns, identifying keys and normalization up to 3rd Normal Form).**

☑ **Soln.** : After designing the ER diagram of system, we need to convert it to Relational models which can directly be implemented by any RDBMS like Oracle, MySQL.

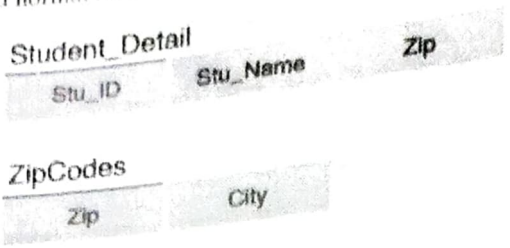
Mapping Entity

An entity is a real-world object with some attributes.

- X is a superkey or,A is prime attribute.



- We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists transitive dependency.
- To bring this relation into third normal form, we break the relation into two relations as follows –



Boyce-Codd Normal Form

- Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that –
- For any non-trivial functional dependency, X → A, X must be a super-key.
- In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

  Stu_ID → Stu_Name, Zip

  and

  Zip → City

Which confirms that both the relations are in BCNF.

Practical 9 : Views

- **Creating Views (with and without check option)**
- **Dropping views**
- **Selecting from a view**

(a) Create views

Consider below existing Table

Student

roll_no	stud_name	bdate	Marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

With Check :

```
CREATE VIEW stud AS
SELECT roll_no, stud_name
FROM  Student
WHERE stud_name IS NOT NULL
WITH CHECK OPTION;
```

- In this view, if we now try to insert a new row with a null value in the stud_name column then it will give an error because the view is created with the condition for the NAME column as NOT NULL.
- For example, though the View is updatable then also the below query for this View is not valid:

```
INSERT INTO stud(roll_no)
VALUES(6);
```

➢ **NOTE :** The default value of **stud_name** column is null.

Without Check:

1. **Creating view having all records and fields from existing table**

Example: Creating a view of base table student with same structure and all the records.

```
Create or replace view stud_view1
as select * from student;
```

Output

stud_view1

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

2. **Creating view having specific fields but all the records from existing table**

Example

```
Create or replace view stud_view2
as select roll_no, name from student;
```

Output

The newly created view will be

stud_view2

roll_no	stud_name
101	Kunal
102	Jay
103	Radhika
104	Sagar
105	Supriya

3. Creating new view having specific records but all the fields from existing table

```
Create or replace view sud_view3
as select * from student
where marks > 80;
```

Output

stud_view3

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

(b) Dropping views :

For dropping the view we can use below command

```
DROP view stud_view2;
```

(c) Selecting from a view :

```
Select * from stud_view3;
```

Output

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

▶ Practical 10 : DCL statements

- **Granting and revoking permissions**
- **Saving (Commit) and Undoing (rollback)**

1. Granting and revoking permissions

(a) Create user

```
mysql> CREATE USER bhakti@localhost IDENTIFIED BY '123';
Query OK, 0 rows affected (0.02 sec)

mysql>
```

(b) Grant privileges to users

```
mysql> grant select on students to bhakti@localhost;
Query OK, 0 rows affected (0.02 sec)
```