Q:1
Design a ER Model for library Management System with required entities with their suitable attributes.

1. Entity:- i) REPORT
ii) STAFF
iii) READER
iv) BOOK
v) PUBLISHER

2. Entities with Attributes:-

i) REPORT :
Regno, User_id, Book_No, Issue/Return

ii) STAFF:-
Name, staff_id, age, gender

iii) READER :-
Rid, email, Rname, phone no, address

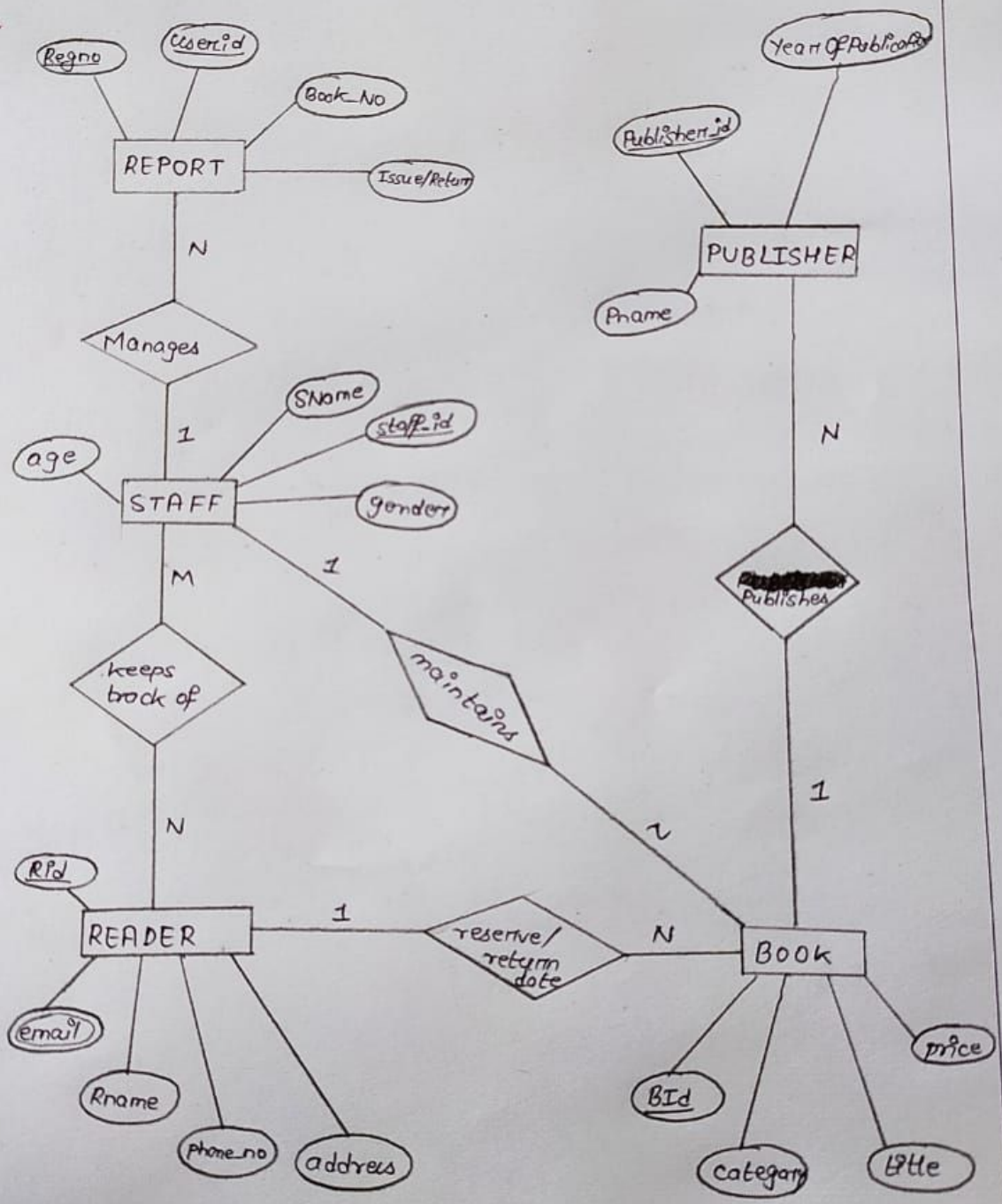
iv) Book :-
Title, BId, price, category

v) PUBLISHER :-
Publisher_id, Pname, Year of Publication

3. Relationship:-
i) Manages : REPORT & STAFF
ii) keeps_track_of : STAFF & READER
iii) reserve/return date : READER & Book
iv) Maintains : STAFF & Book
v) Publishes: Book & PUBLISHER

4. Cardinality Ratio :-
I) STAFF : REPORT ⟶ 1 : N
II) STAFF : READER ⟶ M : N
III) STAFF : Book ⟶ 1 : N
N) READER : Book ⟶ 1 : N
V) Book : PUBLISHER ⟶ M : N

ER Diagram :-


Regno
Userid
Book_No
REPORT
Issue/Return
N
Manages
1
SName
staff_id
age
STAFF
gender
M
keeps track of
N
RId
READER
1
reserve/ return date
N
Book
email
Rname
Phone_no
address
BId
Category
title
price
Year of Publication
Publisher_id
PUBLISHER
Pname
N
Publishes
1
1
maintains
N

Q:2
Write SQL query to CREATE TABLE named CUSTOMERS with ID as a PRIMARY KEY constraint along with other appropriate attributes.

The SQL CREATE TABLE statement is used to create a new table.

SYNTAX:

```
CREATE TABLE table_name(
        column1 datatype,
        column2 datatype,
        column3 datatype,
        .....
        columnN datatype,
        PRIMARY KEY ( one or more columns)
);
```

QUERY:

```
SQL> CREATE TABLE CUSTOMERS(
        ID INT NOTNULL,
        NAME VARCHAR(20) NOT NULL,
        AGE INT NOTNULL,
        ADDRESS CHAR (25),
        SALARY DECIMAL (18,2),
        PRIMARY KEY (ID)
);
```

Q:3
Write SQL Query to show the description of created table using DESC statement.

QUERY: SQL> DESC CUSTOMERS

OUTPUT:

Field	Type	Null	Key	Default	Extra
ID	int(11)	No	PRI		
NAME	varchar(20)	No			
AGE	int(11)	No			
ADDRESS	Char(25)	Yes		NULL	
SALARY	decimal(18,2)	Yes		NULL	

Q:4
Write SQL query to DROP a table from database.

SYNTAX: DROP TABLE table_name;

QUERY:
```
SQL> DROP TABLE CUSTOMERS;
```

Now, if you try to DESC command, then error occured

```
SQL> DESC CUSTOMERS;
ERROR 1146 (42S02): Table 'test.CUSTOMERS' doesn't exist
```

Q:5
Write SQL query to insert records in table CUSTOMERS using INSERT INTO.

SYNTAX: There are two basic syntaxes of INSERT INTO

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

If columns aren't specified explicitly, then value should be entered in order using below syntax:

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

QUERY:

```
INSERT INTO CUSTOMERS VALUES( 1, 'Reshav', 32, 'Amardaha', 2000.00);
INSERT INTO CUSTOMERS VALUES (2, 'Kiran', 25, 'Damak', 1500.00);
INSERT INTO CUSTOMERS VALUES (3, 'Karan', 23, 'Kathmandu', 2000.00);
INSERT INTO CUSTOMERS VALUES (4, 'Chitra', 25, 'Mugla', 6500.00);
INSERT INTO CUSTOMERS VALUES (5, 'Hari', 27, 'Biratnagar', 8500.00);
INSERT INTO CUSTOMERS VALUES (6, 'Komal', 22, 'Makwanpur', 4500.00);
INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
VALUES (7, 'Manoj', 24, 'Illam', 10000.00);
```

Q: 6

Write SQL Query to demonstrate the use of SELECT statement.

SYNTAX:

```
SELECT column1, column2, columnN FROM table_name;
```

Here column1, column2... are the fields of a table. If you want to fetch all the fields of a table, following syntax should be used.

```
SELECT * FROM table_name;
```

Query:

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS;

ID	NAME	SALARY
1	Reshav	2000.00
2	Kiran	1500.00
3	Karan	2000.00
4	Chitra	6500.00
5	Hari	8500.00
6	Komal	4500.00
7	Manoj	10000.00

SQL> SELECT * FROM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Reshav	32	Amardaha	2000.00
2	Kiran	25	Damak	1500.00
3	Karan	23	Kathmandu	2000.00
4	Chitra	25	Mugian	6500.00
5	Hari	27	Biratnagar	8500.00
6	Komal	22	Makwanpur	4500.00
7	Manoj	24	Illam	10000.00

Q:7
Write SQL Query to fetch ID, NAME, SALARY from the table where Salary is greater than 2000.

→ The SQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

→ WHERE clause can be used with SELECT, UPDATE & DELETE statement.

SYNTAX:
```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

Query:
```
SQL> SELECT ID, NAME, SALARY
     FROM CUSTOMERS
     WHERE SALARY >2000;
```

OUTPUT:

ID	NAME	SALARY
4	Chitra	6500.00
5	Hari	8500.00
6	Komal	4500.00
7	Manoj	10000.00

Q:8
Write SQL query to Modify existing record in a table using UPDATE query.

→ Use WHERE clause with UPDATE query to update selected rows, otherwise all rows will be effected.

SYNTAX:
```
UPDATE table_name
SET column1 = value1, column2 = value2,....
WHERE [condition];
```

Query:

```
SQL> UPDATE CUSTOMERS
     SET ADDRESS = 'Pokhara'
     WHERE ID = 6;
```

To view the effects:

```
SQL> SELECT ID, ADDRESS FROM CUSTOMERS;
```

ID	ADDRESS
1	Amar daha
2	Damak
3	Kathmandu
4	Muglan
5	Biratnagar
6	Pokhara
7	Illam

Increase salary by 500.

```
SQL> UPDATE CUSTOMERS
     SET SALARY = SALARY + 500;
SQL> SELECT ID, Name, Salary .FROM CUSTOMERS;
```

ID	NAME	SALARY
1	Reshav	2500.00
2	Kiran	2000.00
3	Karan	2500.00
4	Chitra	7000.00
5	Hari	9000.00
6	Komal	5000.00
7	Manoj	10500.00

Q:9

Write SQL query to DELETE existing Records from a table.

→ WHERE clause can be used with DELETE query to delete the selected rows.

SYNTAX:

```
DELETE FROM table_name
WHERE [condition];
```

QUERY:

```
SQL> DELETE FROM CUSTOMERS
     WHERE ID = 6;
```

```
SQL> SELECT DELETE FROM CUSTOMERS
     WHERE SALARY < 2500;
```

```
SQL > SELECT, ID, NAME, SALARY FROM CUSTOMERS;
```

ID	NAME	SALARY
4	chitra	6500.00
5	Hari	8500.00
7	Manoj	1000.00

Q:10

Write SQL query to use wildcard operators with the LIKE.

→ There are two wildcard operators

- percent sign (%)
- underscore (_)

SYNTAX:

```
SELECT * FROM table_name
where column LIKE '%xxx%';

SELECT * FROM table_name
where column LIKE '_xxx_';
```

Query to display all the records from table where SALARY starts with 200.

```
SQL> SELECT * FROM CUSTOMERS
     WHERE SALARY LIKE '200%';
```

ID	NAME	AGE	ADDRESS	SALARY
1	Peshav	32	Amardaha	2000.00
3	Karan	23	Kathmandu	2000.00

Q:11
Write SQL query to sort the data using ORDER BY clause.

→ The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Databases sort query in ascending order by default.

SYNTAX:

```
SELECT [column-list]
FROM table_name
[WHERE condition]
ORDER BY [column1, column2, ..][ASC | DESC];
```

Query:

```
SQL> SELECT Name, SALARY FROM CUSTOMERS
     ORDER BY SALARY DESC;
```

NAME	SALARY
Manoj	10000.00
Hari	8500.00
Chitra	6500.00
Komal	4500.00
Peshav	2000.00
Karan	2000.00
Kiran	1500.00

Q:12

Write SQL query to arrange identical data into groups using GROUP BY clause.

→ GROUP BY is used in collaboration with the SELECT statement to arrange identical data into groups.

SYNTAX:

```
SELECT column1, column2
FROM table_name
WHERE [conditions]
GROUP BY column1, column2
```

Making identical records:

```
SQL> UPDATE CUSTOMERS
     SET NAME = 'Kaushik'
     WHERE ID = 4
```

~~Now, table is~~:

Name	~~Salary~~
~~Reshav~~	
~~Kiran~~	
~~Kaushik~~	

```
SQL> UPDATE CUSTOMERS
     SET NAME = 'Kaushik'
     WHERE ID = 3
```

To find total amount of salary of each customer.

```
SQL> SELECT NAME, SUM(SALARY) FROM CUSTOMERS
     GROUP BY NAME;
```

NAME	SUM(SALARY)
Reshav	2500.00
Kiran	2000.00
Kaushik	9500.00
~~Khushita~~	
Hari	9000.00
Komal	5000.00
Manoj	10500.00

[Refer table from Q:8]

Q:13
write SQL query to eliminate all the duplicate records using DISTINCT keyword.

→ used in a situation when you don't want to have multiple duplicate records in a table.

SYNTAX:
```
SELECT DISTINCT [column-lists]
FROM table.name
WHERE [condition];
```

Query: SQL> SELECT DISTINCT SALARY FROM CUSTOMERS
ORDER BY SALARY;

OUTPUT:

SALARY
1500.00
2000.00
4500.00
6500.00
8500.00
10000.00

Q:14
Write SQL query to validate the value being entered into a record using CHECK constraint.
Create a new table CUSTOMERS and add a check with AGE column, so that table can't have CUSTOMERS below 18 years.

```
SQL>CREATE TABLE CUSTOMERS(
        ID INT          NOT NULL,
        NAME VARCHAR(20) NOT NULL,
        AGE INT         NOT NULL  CHECK(AGE >=18),
        ADDRESS CHAR(25),
        SALARY DECIMAL(18,2),
        PRIMARY KEY(ID)
);
```

Q:15

Write SQL query to filter which group result to appear in the final result using HAVING clause.

⇒ HAVING clause places conditions on groups created by GROUP BY clause.

SYNTAX:

```
SELECT [column-list]
FROM table_name
WHERE [conditions]
GROUP BY [column-list]
HAVING [conditions]
ORDER BY [column-list]
```

To display record for which similar age count would be more than or equal to 2.

```
SQL> SELECT * FROM CUSTOMERS
     GROUP BY age
     HAVING COUNT(age) >= 2;
```

ID	NAME	AGE	ADDRESS	SALARY
2	Kiran	25	Damak	1500.00

[Table from Q:5 is used]

Q:16

Write SQL Query to CREATE TABLE named ORDERS with necessary attributes and referencing to table CUSTOMERS(ID) with CUSTOMER_ID

- Foreign key is used to link two tables together. This is sometimes called a referencing key
- It is a column or a combination of columns whose values match a Primary key in a different table.

```
SQL>
  CREATE TABLE CUSTOMERS(
          ID INT  NOT NULL,
          NAME VARCHAR(20) NOT NULL,
          AGE  INT     NOT NULL,
          ADDRESS  CHAR(25),
          SALARY   DECIMAL(18,2).
          PRIMARY KEY(ID)
      );
```

```
SQL> CREATE TABLE ORDERS(
          ID   INT   NOT NULL,
          DATE  DATETIME,
          CUSTOMER_ID  INT  references CUSTOMERS(ID),
          AMOUNT  double,
          PRIMARY KEY(ID)
      );
```

If table orders has already been created, foreign key can be implemented as follows

```
SQL> ALTER TABLE ORDERS
     ADD FOREIGN KEY (CUSTOMER_ID)
     REFERENCES CUSTOMERS(ID);
```

Consider ORDERS table containing following records

OID	DATE	CUSTOMER_ID	AMOUNT
102	2079-10-08 17:12:00	3	3000
100	2079-10-08 05:36:47	3	1500
101	2079-11-20 21:12:41	2	1560
103	2079-05-20 20:25:07	4	2060

Q:17

Write SQL query to perform combined operations on two tables CUSTOMERS, ORDERS using JOIN clause

Consider tables CUSTOMERS and ORDERS ~~As~~ from previous queries (Q:5 and Q:16)

To find name of those customer who has made an order

```
SQL> SELECT ID, NAME, AGE, AMOUNT
     FROM CUSTOMERS, ORDERS
     WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Output:

ID	NAME	AGE	AMOUNT
8	Karan	23	3000
3	Karan	23	1500
2	Kiran	25	1560
4	Chitra	25	2060