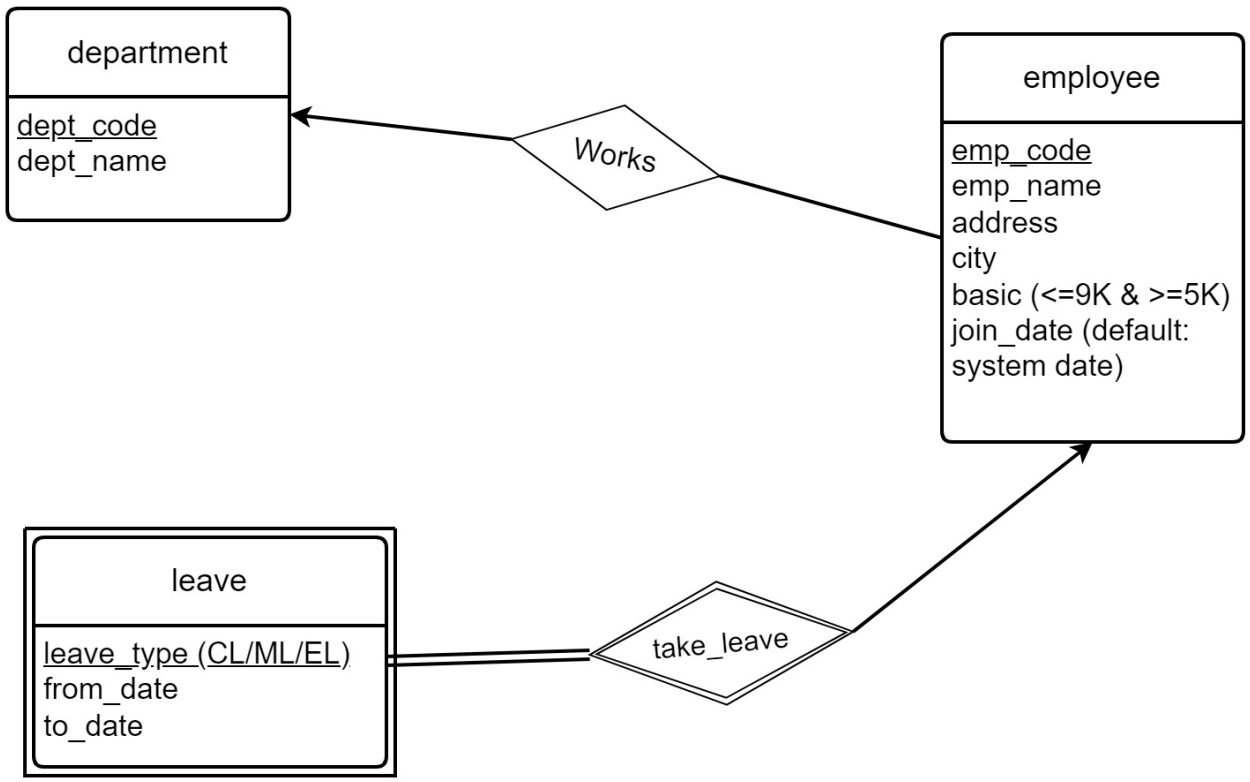
QUESTION NO 1.:

ER Diagram design:

Each **department** has a unique code. Suppose its *dept_code* and the primary key of department entity. Each department has a name say it is *dept_name*. Each **employee** has unique code named *emp_code* and it is primary key of employee entities. For each employee *name, address, basic, city join date* are also stored. As given in the question, each department may have number of employees. We can consider that, one employee may be associated to at most one department. So, we can think of a relationship between employee and department named *Works*. It is a many-to-one relationship from employee to department.

Leave information about employees are stored. Attributes for a leave are *leave_type, from_date, to_date*. It is clear that leave is a weak entity set. As leave is dependent on employee who takes the leave. So, there is a weak relationship *take_leave* between leave and employee. So, employee is an identifying strong entity set for leave and *take_leave* is identifying relationship. There is no meaning of leave without an employee. So, participation of leave in *take_leave* is total. The relationship 'take_leave' is many-to-one from leave to employee.

The ER Diagram is given below,

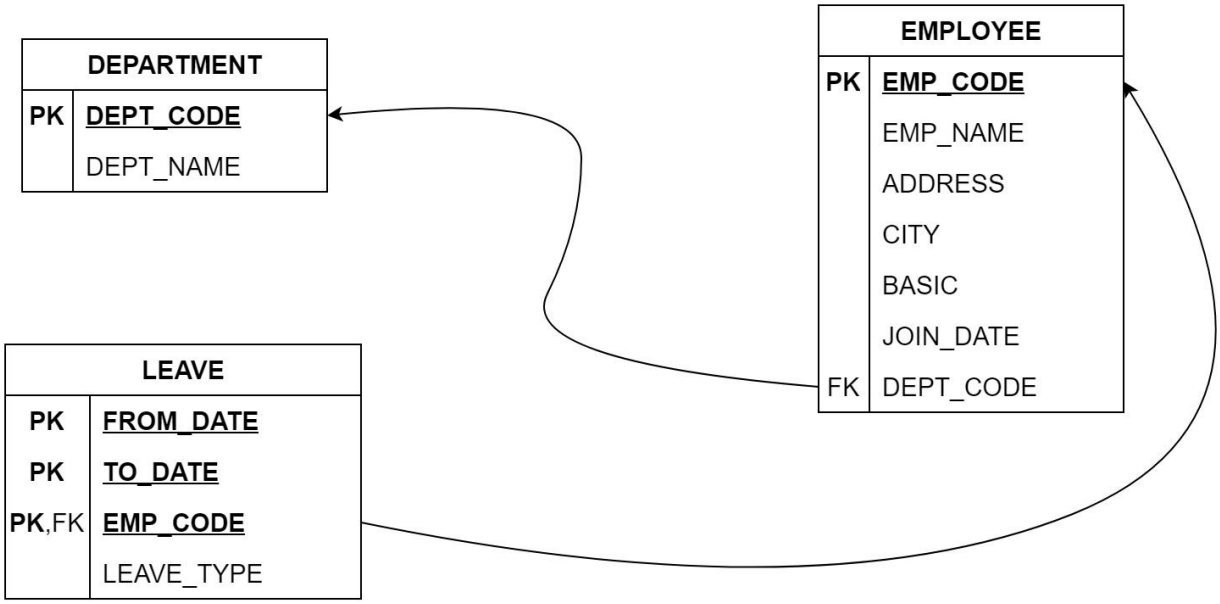


Design and implement Tables(relations):

According to the ER diagram depicted above, three tables are created – Employee, Department and Leave. Department has primary key

dept_code and another attribute dept_name. Each employee has primary key emp_code and other attributes as emp_name, basic etc. As each employee can have at most one department as per our design, there will be a foreign key dept_code referring to the department table. As we know leave is a weak entity set. So, leave table will contain emp_code from the employee table. Emp_code in leave table will be foreign key that refers the employee table. And, emp_code,from_date,to_date together will be primary key. Also, after the deletion of an employee, all leave info related to that employee must be deleted(cascaded).

Diagram for table design:



SQL for table design:

```
CREATE TABLE DEPARTMENT (
      DEPT_CODE CHAR(3) PRIMARY KEY,
      DEPT_NAME CHAR(10) NOT NULL
);


CREATE TABLE EMPLOYEE (
      EMP_CODE CHAR(3) PRIMARY KEY,
      EMP_NAME CHAR(20) NOT NULL,
      ADDRESS CHAR(40),
      CITY CHAR(10),
      BASIC NUMBER(4,0) CHECK (BASIC>=5000 AND BASIC<=9000),
      JOIN_DATE DATE DEFAULT CURRENT_DATE,
      DEPT_CODE CHAR(3) FOREIGN KEY REFERENCES DEPARTMENT(DEPT_CODE)
```

```
);

CREATE TABLE LEAVE (
      LEAVE_TYPE CHAR(2) CHECK (leave_type IN ('cl', 'ml', 'el')),
      FROM_DATE DATE,
      TO_DATE DATE,
      EMP_CODE CHAR(3) FOREIGN KEY REFERENCES EMPLOYEE(EMP_CODE) ON DELETE
CASCADE,
      PRIMARY KEY(FROM_DATE, TO_DATE, EMP_CODE)
);
```

QUESTION NO 3.:

A)) Create a table having empcode , Name, deptname, & basic From the existing tables along with the records of the employee who are in a particular department (say, d1) and with a basic Rs. 7000/-

```
CREATE TABLE EMP_DEPT_BASIC AS
SELECT e.EMP_CODE, e.EMP_NAME, d.DEPT_NAME, e.BASIC
FROM EMPLOYEE AS e
INNER JOIN DEPARTMENT AS d ON e.DEPT_CODE = d.DEPT_CODE
WHERE d.DEPT_CODE = 'd1' AND e.BASIC = 7000;
```

B)) From the existing table, add the employees with the basic salary greater than or equal to 7000/-

```
INSERT INTO EMP_DEPT_BASIC (EMP_CODE, EMP_NAME, DEPT_NAME, BASIC)
SELECT e.EMP_CODE, e.EMP_NAME, d.DEPT_NAME, e.BASIC
FROM EMPLOYEE AS e
INNER JOIN DEPARTMENT AS d ON e.DEPT_CODE = d.DEPT_CODE
WHERE e.BASIC >= 7000;
```

C)) Alter the table to add a net pay column.

```
ALTER TABLE EMP_DEPT_BASIC
ADD NET_PAY NUMBER(8,2);
```

D)) Replace net pay with 1.5* Basic.

```
UPDATE EMP_DEPT_BASIC
SET NET_PAY = 1.5 * BASIC;
```

E)) Try to remove the net net pay column.

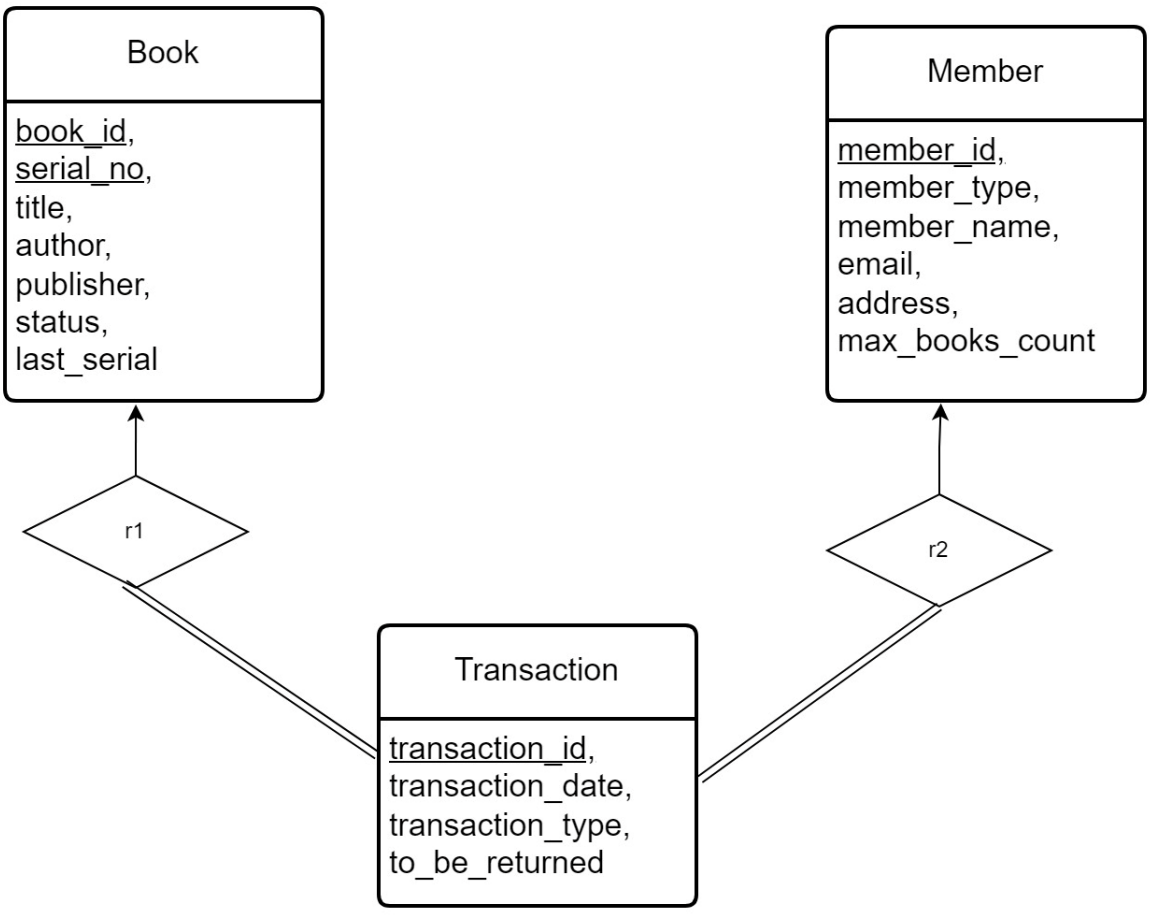
```
ALTER TABLE EMP_DEPT_BASIC

DROP COLUMN NET_PAY;
```

QUESTION NO 5.:

ER Diagram design:

Each Book has book-id, serial-no both taken as unique primary key. Also other info related to book as title, author, publisher, status, last_serial etc are there. Each Member is uniquely identified by primary key member-id. Each member also have attributes such as name, member-type, email, address, max-no-books etc. Each Transaction has unique transaction-id which is the primary key. Each transaction entity also holds transaction_date(dt_issue or dt_return), transaction_type(issue or return), to_be_returned(to be used for during return). There is a many(total) to one relationship from Transaction to Book. Similarly from Transaction to member.

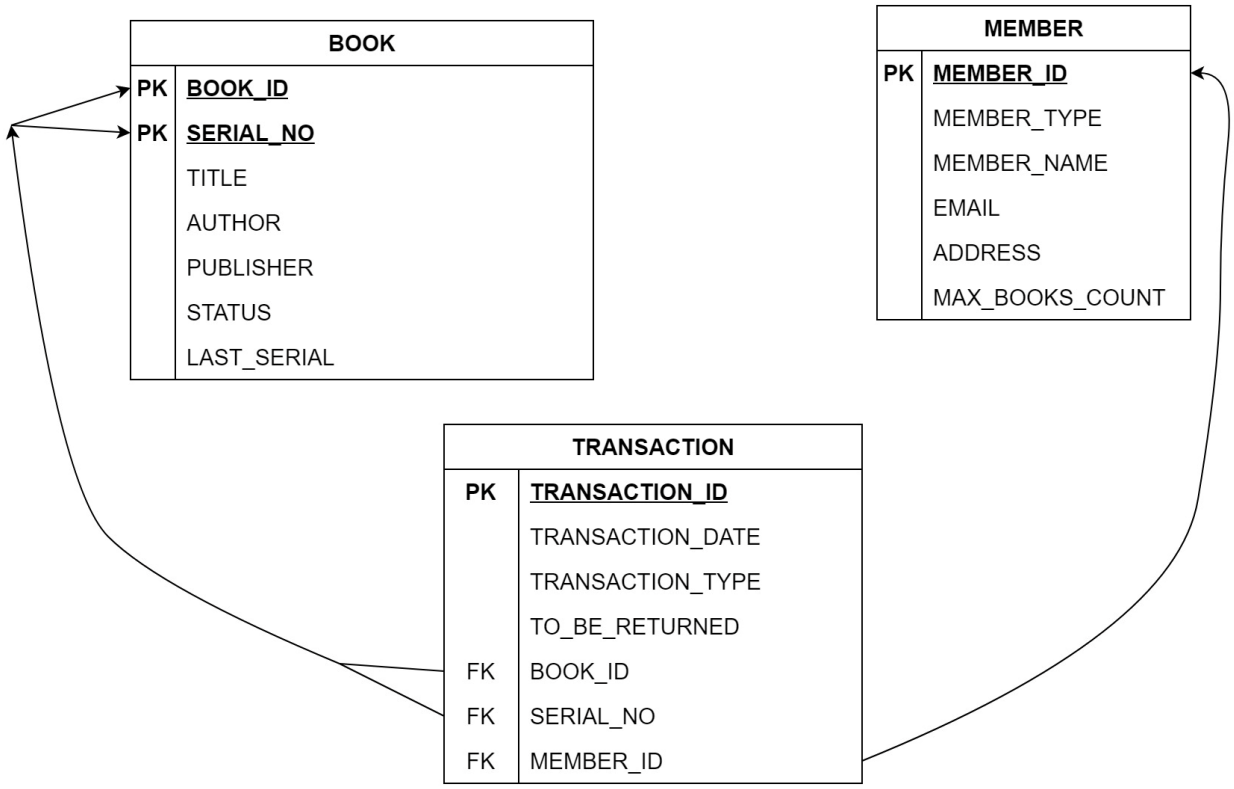
The ER Diagram is given below,



Design and implement Tables(relations):

According to the ER diagram shown above, we create three tables – Book, Transaction and Member. Book has primary key as book_id and serial_no together, and some other attributes are there. Transaction and Member have primary key as transaction_id and member_id respectively. Other attributes are also present. Also, we export book_id, serial_no into Transaction table, and they work as foreign key to refer the Book table. Similarly, we include member_id in the Transaction table. This is used as foreign key to refer the Member table.

The diagram for table design is given –



SQL for table design:

```
CREATE TABLE BOOK (
      BOOK_ID CHAR(3),
      SERIAL_NO NUMBER(3,0),
      TITLE CHAR(100),
      AUTHOR CHAR(100),
      PUBLISHER CHAR(100),
      STATUS CHAR(10) CHECK (STATUS IN ('issued','available'))
      LAST_SERIAL NUMBER(3,0),
      PRIMARY KEY(BOOK_ID, SERIAL_NO)
```

```
);

CREATE TABLE MEMBER (
      MEMBER_ID CHAR(3) PRIMARY KEY,
      MEMBER_TYPE CHAR(7) CHECK (MEMBER_TYPE IN ('student','faculty')),
      MEMBER_NAME CHAR(20) NOT NULL,
      EMAIL CHAR(50),
      ADDRESS CHAR(100),
      MAX_BOOKS_COUNT NUMBER(1,0)
);

CREATE TABLE TRANSACTION (
      TRANSACTION_ID CHAR(3) PRIMARY KEY,
      TRANSACTION_DATE DATE,
      TRANSACTION_TYPE CHAR(6) CHECK (TRANSACTION_TYPE IN ('issue','return')),
      TO_BE_RETURNED DATE DATE,
      BOOK_ID CHAR(3),
      SERIAL_NO NUMBER(3,0),
      MEMBER_ID CHAR(3),
      FOREIGN KEY(BOOK_ID, SERIAL_NO) REFERENCES BOOK(BOOK_ID,SERIAL_NO),
      FOREIGN KEY(MEMBER_ID) REFERENCES MEMBER(MEMBER_ID)
);
```

Related Questions:

A)) Display total number of copies (irrespective of issued or not) for each book in the library andnumber of such copies issued

```
SELECT B.BOOK_ID, B.TITLE, COUNT(*) AS TOTAL_COPIES, COUNT(T.BOOK_ID) AS
ISSUED_COPIES
FROM BOOK B
LEFT JOIN TRANSACTION T
ON B.BOOK_ID = T.BOOK_ID AND B.SERIAL_NO = T.SERIAL_NO
GROUP BY B.BOOK_ID, B.TITLE;
```

B)) Find the members holding the books even after due date

```
SELECT m.member_id, m.member_name, t.book_id, t.serial_no
```

```
FROM MEMBER AS m
INNER JOIN TRANSACTION AS t
ON m.member_id = t.member_id
WHERE t.transaction_type = ‘issue’
AND t.to_be_returned < CURRENT_DATE;
```

C)) Find the transaction details for delayed book returns and delay in terms of number of days.

```
SELECT T.TRANSACTION_ID, T.TRANSACTION_DATE, T.BOOK_ID, T.SERIAL_NO,
T.MEMBER_ID, T.TO_BE_RETURNED,
       B.TITLE, DATEDIFF(DAY, T.TO_BE_RETURNED, GETDATE()) AS DAYS_DELAYED
FROM TRANSACTION AS T
INNER JOIN BOOK AS B
ON T.BOOK_ID = B.BOOK_ID AND T.SERIAL_NO = B.SERIAL_NO
WHERE T.TRANSACTION_TYPE = 'return' AND T.TO_BE_RETURNED < GETDATE();
```

D)) Find the student members not making any transaction and do the same for faculty members.

```
SELECT * FROM MEMBER
LEFT JOIN TRANSACTION
ON MEMBER.MEMBER_ID = TRANSACTION.MEMBER_ID
WHERE TRANSACTION.TRANSACTION_ID IS NULL
AND MEMBER.MEMBER_TYPE = 'student';

SELECT * FROM MEMBER
LEFT JOIN TRANSACTION
ON MEMBER.MEMBER_ID = TRANSACTION.MEMBER_ID
WHERE TRANSACTION.TRANSACTION_ID IS NULL
AND MEMBER.MEMBER_TYPE =’faculty’;
```

E)) Find the count of issue for each book (not the specific copy).

```
SELECT BOOK_ID, COUNT(*) AS ISSUE_COUNT
FROM TRANSACTION
WHERE TRANSACTION_TYPE = 'issue'
GROUP BY BOOK_ID;
```