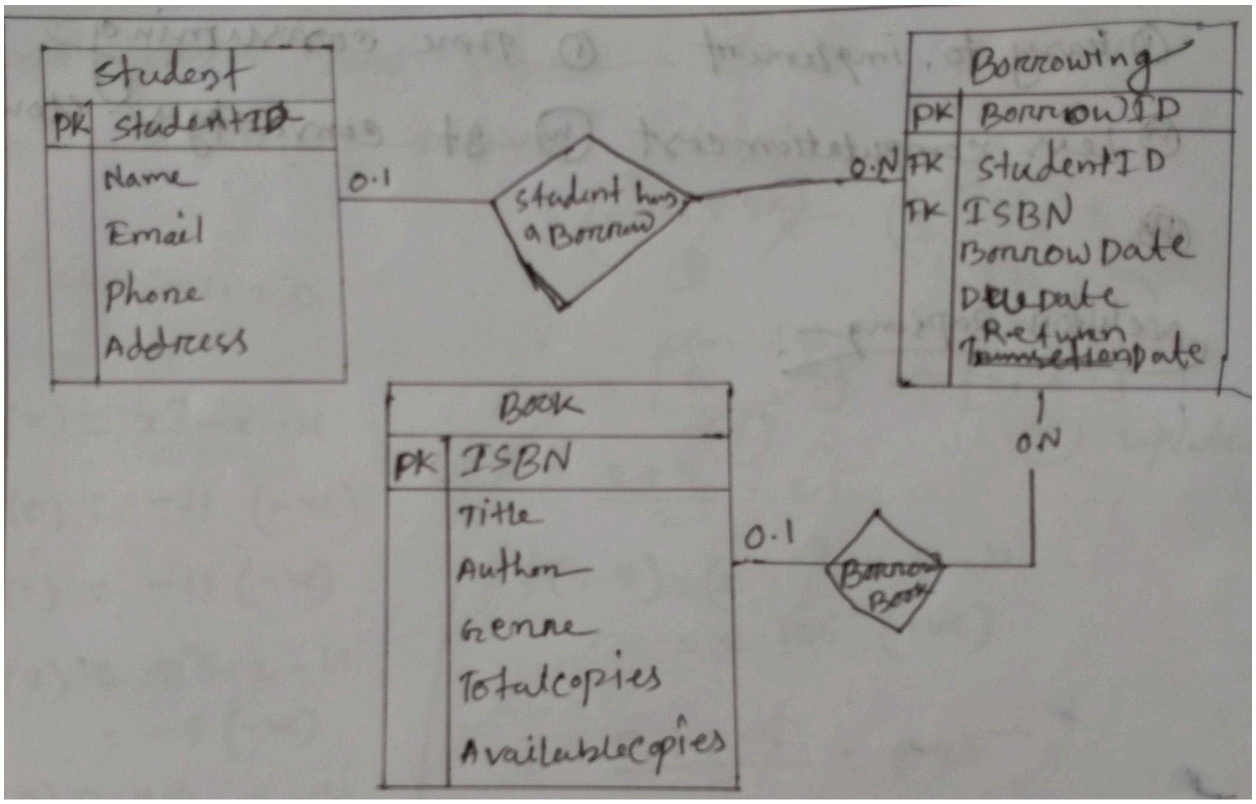
Answer to the question no: 1



Answer to the question no: 2

```
INSERT INTO BORROWING(STUDENTID, ISBN, BORROWDATE, DUEDATE)
VALUES (3, (SELECT ISBN
            FROM BOOK
            ORDER BY AVAILBLECOPIES DESC
            LIMIT  1),
            '2024-02-11', '2024-02-12'
           );
```

Answer to the question no: 3

```
UPDATE BOOK
SET AVAILBLECOPIES = AVAILBLECOPIES - 1
WHERE ISBN = '9781234567890';
```

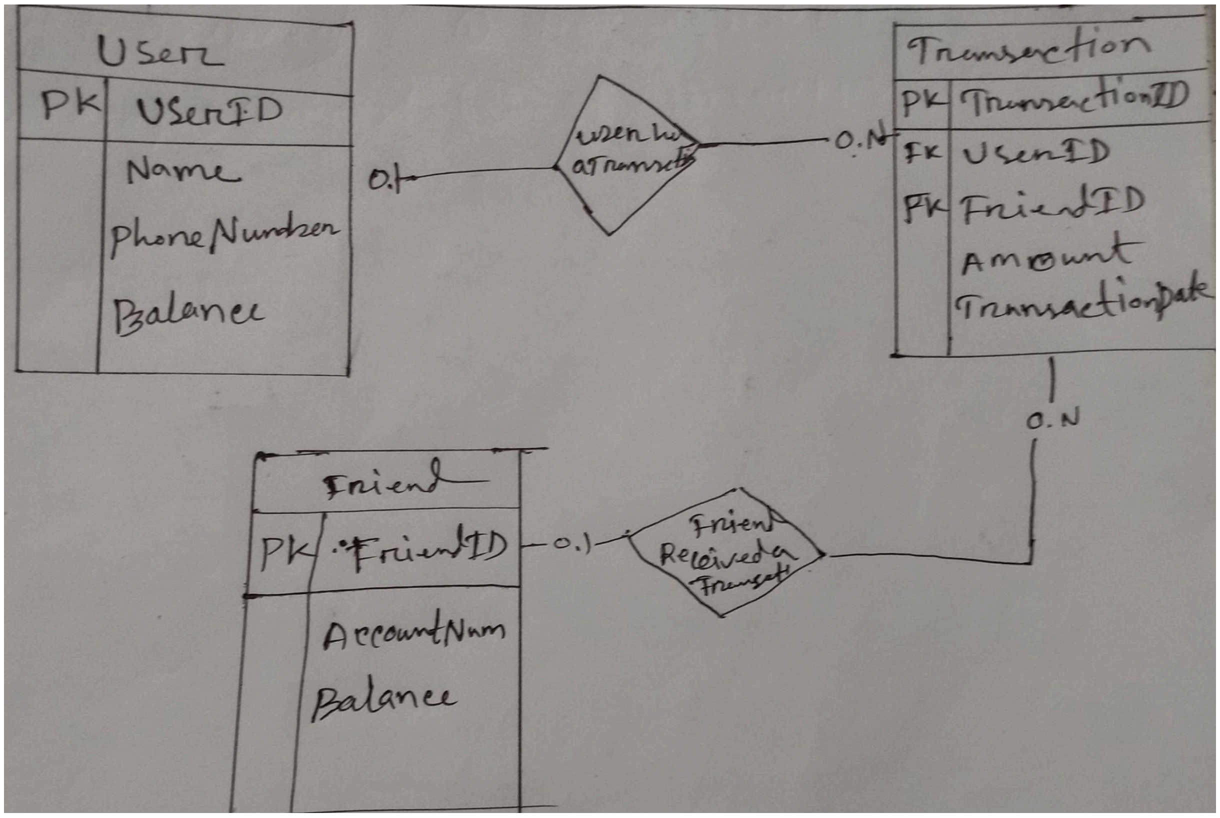
Answer to the question no: 4

```
SELECT STUDENT.NAME, COUNT(BORROWING.STUDENTID) AS BORROWCOUNT
FROM STUDENT
LEFT JOIN BORROWING ON STUDENT.STUDENTID = BORROWING.STUDENTID
GROUP BY STUDENT.STUDENTID
ORDER BY BORROWCOUNT DESC
LIMIT 2;
```

<u>**Answer to the question no: 5**</u>

```
SELECT BOOK.TITLE, BOOK.ISBN, BORROWING.BORROWDATE, BORROWING.DUEDATE,
BORROWING.RETURNDATE
FROM BOOK
INNER JOIN BORROWING ON BOOK.ISBN = BORROWING.ISBN
WHERE BORROWING.RETURNDATE < CURDATE();
```

<u>**Answer to the question no: 6**</u>



<u>**Answer to the question no: 7**</u>

Both UNION and UNION ALL are set operations used in MySQL to combine results from multiple SELECT statements into a single result set. However, they differ in how they handle duplicate rows:

UNION:

1. Combines the results of two or more SELECT statements.
2. Removes duplicate rows from the combined result set.
3. Requires the number of columns and their data types to be identical in all SELECT statements involved.
4. Generally slower than UNION ALL due to the duplicate removal process.

UNION ALL:

1. Combines the results of two or more SELECT statements.
2. Includes all rows, even duplicates, from the combined result set.
3. Does not require any specific match in the number of columns or data types, but compatibility issues could arise if they differ significantly.
4. Generally faster than UNION as it doesn't perform duplicate removal.

Answer to the question no: 8

```
SELECT *
FROM EMPLOYEES
WHERE SALARY = (SELECT MAX(SALARY)
                FROM EMPLOYEES
                 WHERE SALARY < (SELECT MAX(SALARY)
                                 FROM EMPLOYEES)
                );
```

Answer to the question no: 9

```
CREATE TABLE Department
(
  DepartmentID INT PRIMARY KEY,
  DepartmentName VARCHAR(60)
);

CREATE TABLE Employee
(
   EmployeeID INT PRIMARY KEY,
   FirstName VARCHAR(60),
   LastName VARCHAR(60),
   DepartmentID INT,
         FOREIGN KEY (DepartmentID)
         REFERENCES Department(DepartmentID)
         ON DELETE SET NULL
);

CREATE TABLE JobHistory
(
   JobID INT PRIMARY KEY,
   EmployeeID INT,
   JobTitle VARCHAR(60),
         FOREIGN KEY (EmployeeID)
         REFERENCES Employee(EmployeeID)
         ON DELETE CASCADE
);
```

Answer to the question no: 10

These things are very interesting to me

Basic Queries: These are like asking simple questions to get specific answers from the database. For example, asking "What's the salary of John?" or "How much money does the Sales department make?"

Functions: For Example, finding the total amount of money earned by a department or figuring out the average salary of all employees.

Subqueries: This is like asking a question within a question. Imagine I am trying to find out who has the second-highest salary in the Sales department. First, you'd ask the database to list all the salaries in Sales, then I pick out the second-highest one from that list.
That's it.