1. (4 points) What are the four ACID properties?

   Atomicity
   Consistency
   Isolation
   Durability

2. (4 points) Choose one of the acid properties and provide a definition of the property and explain why that property is important. (Hint: What can go wrong if that property doesn’t hold.)

   Atomicity – Without this, as result of a failure, partially completed transactions might be applied to the database.
   Consistency – Without this, the database might be left in an inconsistent state. Or, an intermediate state, as perceived by another transaction might be inconsistent.
   Isolation – Without this, transactions executing in parallel might interact in surprising and unfortunate ways.
   Durability – Without this, completed transactions might be lost as a result of a failure, leaving the users not knowing what had and had not been done.

3. (8 points) In terms of the Relational Data Model explain the difference between a Superkey, a Candidate Key, and a Primary Key.

   Superkey – a subset of the fields that uniquely determines the tuples in a relation.
   Candidate Key – a minimal superkey in the sense that no field can be removed from the key and have it still remain a superkey.
   Primary Key – a chosen, single, Candidate Key.

4. (4 Points) Provide an example of “Impedance Mismatch” between SQL and ordinary programming languages.

   There are a large number of possibilities here. Some examples:
   Database types (e.g. dates, numbers) don't exist in the host language or don't have the same ranges, for example, as the types in the host language.
   The host language uses objects, the database has tuples and rows but no objects.
Consider the following Entity Relationship diagram:

5. (20 Points) Provide a relational schema (named relations and attributes for those relations) for this ER diagram. Make sure to show keys and foreign key relationships.
For questions 6 to 13, consider the following relational schema:

<table>
<thead>
<tr>
<th>Table</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>SSN</td>
</tr>
<tr>
<td>Department</td>
<td>DeptNum</td>
</tr>
<tr>
<td>Department_Locations</td>
<td>DeptNum</td>
</tr>
<tr>
<td>Project</td>
<td>ProjName</td>
</tr>
<tr>
<td>Works_On</td>
<td>SSN</td>
</tr>
</tbody>
</table>

Write SQL for the following. (Hint: You may have to use EXISTS or NOT EXISTS.)

6. (3 points) Retrieve the names of all employees in Department 6. If there are multiple employees with the same name, the name should be repeated as many times as there are employees with that name.

```sql
SELECT Name
FROM Employee
WHERE DeptNum = 6;
```

7. (3 points) Retrieve the names of all employees in Department 6. If there are multiple employees with the same name, the name should only appear once.

```sql
SELECT DISTINCT Name
FROM Employee
WHERE DeptNum = 6;
```

8. (3 points) Retrieve the numbers and names of all departments with a location that begins with ‘H’.

```sql
SELECT D.DeptName, D.DeptNum
FROM Department D JOIN Department_Locations DL
ON D.DeptNum = DL.DeptNum
WHERE DL.DeptLoc LIKE 'H%';
```
9. (4 points) Retrieve the names of all employees in Department 6 who do not work on Project XYZ.

My choice (others accepted):

```sql
SELECT Name
FROM Employee
WHERE DeptNum = 6
    AND NOT EXISTS
        (SELECT *
         FROM Project P JOIN Works_On W ON P.ProjNum = W.ProjNum
         WHERE P.ProjName = 'XYZ' AND W.SSN = Employee.SSN);
```

10. (4 points) For each project list the project name and the total hours per week (by all employees) spent on the project.

```sql
SELECT ProjName, SUM(Hours)
FROM Project P JOIN Works_On W ON P.ProjNum = W.ProjNum
GROUP BY ProjName;
```

11. (5 points) Retrieve the names of all employees who work on at least one project located in Houston but whose department has no location in Houston.

My choice:

```sql
SELECT Name
FROM Employee E
WHERE
    EXISTS
        (SELECT *
         FROM Project P JOIN Works_On W ON P.ProjNum = W.ProjNum
         WHERE P.ProjName = 'XYZ' AND W.SSN = Employee.SSN)
    AND NOT EXISTS
        (SELECT *
         FROM Department D JOIN Department_Locations L
         ON D.DeptNum = L.DeptNum
         WHERE L.DeptLoc = 'Houston' AND D.DeptNum = E.DeptNum)
```
12. (3 points) Add an employee named “John Doe” with SSN 123456789, Sex “M”, and Department Number 5 to the database.

\[
\text{INSERT INTO Employee VALUES (123456789, 'John Doe', 'M', 5);}
\]

\textit{or, for example,}

\[
\text{INSERT INTO Employee\nSET Name='John Doe', SSN=123456789, Sex='M', DeptNum=5;}
\]

13. (3 points) Increase the number of hours for all employees working on Project Number 123 by 10%.

\[
\text{UPDATE Works_On\nSET Hours = Hours * 1.1\nWHERE ProjNum = 123;}
\]
Consider the following relation with the indicated Functional Dependencies:

<table>
<thead>
<tr>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
</tr>
</tbody>
</table>

{SSN} → Name, DeptNo, DName
{DeptNo} → DName
{ProjNo} → DeptNo, PName, Budget, PMgrSSN
{DeptNo, PName} → ProjNo, Budget, PMgrSSN
{PMgrSSN} → DeptNo
{SSN, ProjNo} → Hours

For each of the tables shown below answer one of:
- N (No or None) if the table is not in Second Normal Form
- 2 (2NF or Second Normal Form) if the table is Second Normal Form but not Third Normal Form
- 3 (3NF or Third Normal Form) if the table is in Third Normal Form.

14. ______ 3 _______ (4 points)
   
   | SSN | Name |

15. ______ 2 _______ (4 points)
   
   | SSN | Name | DeptNo | DName |

16. ______ N _______ (4 points)
   
   | SSN | Name | ProjNo | Hours |
   Key is (SSN, ProjNo)

17. ______ 3 _______ (4 points)
   
   | ProjNo | DeptNo | PName | Budget |

18. ______ 3 _______ (4 points)
   
   | ProjNo | DeptNo | PName | PMgrSSN | Budget |

Note: #18 really is in third normal form. The fact that PMgrSSN determines DeptNo does not invalidate this since (DeptNo, Pname) is an alternate key and that makes DeptNo a prime attribute. This is very similar to the example shown in Figure 10.12 in the book. This relation is third normal form but not Boyce-Codd normal form.
19. *(12 Points)* Provide an outline (show the major steps, not just a SELECT statement) for a host language program that runs a query with parameters (e.g. Dept Number is a user specified input) and outputs the results of the query.

Example: An outline for reading a file and outputting it might look something like:

1. Get the file name from the user.
2. OPEN the file using the file name.
3. For each line in the file do steps 3.1 and 3.2:
   3.1 READ the line
   3.2 Output the line
4. CLOSE the file

Something along the lines of:

1. Open a CONNECTION to the database.
2. Get the parameter from the user.
3. Create a CURSOR.
4. Execute the query using the cursor, attaching the parameter to the query.
5. Until there are no more rows to be fetched:
   1. FETCH the next row from the CURSOR.
   2. Output the row.
6. Close the CURSOR.
7. Close the CONNECTION.

There were a lot of variations accepted for this one.