CMPS 180, Final Exam, Winter 2017, Shel Finkelstein

Student Name: _____

Student ID: _____

UCSC Email: _____

Final Points

Part	Max Points	Points
I	44	
II	30	
III	32	
Total	106	

The first Section (Part I) of the Winter 2017 CMPS 180 Final is multiple choice and is double-sided. Answer all multiple choice questions on your Scantron sheet. You do not have to hand in that first Section of the Exam, but you must hand in the Scantron sheet, with your name, email and student id on that Scantron sheet. Please be sure to use a #2 pencil to mark your choices on this Section of the Final.

The separate second Section (Parts II and III) of the Final is not multiple choice and is single-sided, so that you have extra space to write your answers. If you use that extra space, please be sure to write the number of the problem that you're solving next to your answer. Please write your name, email and student id on the second Section of the Exam, which you must hand in. You may use any writing implement on this Section of the Exam.

At the end of the Final, please be sure to hand in **both** your Scantron sheet for the first Section of the Exam and also the second Section of the Exam, and show your UCSC id when you hand them in.

Part I: (44 points, 2 points each)

Answer the questions in Part I on your Scantron sheets, which should have your name, email and UCSC id on them. Select the **best answer** for each of the following. For some questions, a choice is **"All of the Above"**, so read answer choices carefully.

Question 1: If an instance of relation R(A,B) has 10 different tuples in it, and an instance of relation S(B,C,D) has 6 different tuples in it, then how many tuples are there in the result if the following SQL query is executed on those instances?

```
SELECT *
FROM R, S
WHERE R.B = S.B;
```

a) 0
b) Exactly 16
c) Exactly 60
d) Between 0 and 16
e) Between 0 and 60

Question 2: We discussed ACID properties of transactions. What does Atomicity (the "A" in ACID) refer to for transactions?

a) Transaction execution is as if they were executed one at a time.
b) Transactions happen completely or not-at-all.
c) If a transaction commits, its changes are permanent, even if there are failures.
d) Business rules are always maintained by the database system.
e) Uncommitted (dirty) values from one transaction are never read by any other transaction.

Question 3: An instance of relation R(A,B) has m tuples in it, all the same, and an instance of relation S(A,B) has n tuples in it, which are all the same as the tuples in R(A,B). If R and S are Union-Compatible, and m and n are both at least 1, then how many tuples are there in the result of the following query?

```
( SELECT * FROM R )
    INTERSECT
( SELECT * FROM S );
```

a) 1
b) m + n
c) m – n (but not less than zero)
d) min(m, n)
e) max(m, n)

Question 4: Why might there be a runtime error when the following statement is executed on relations Customers(cname, age) and Activities(cname, slopeid, date)?

```
SELECT  Customers.cname
FROM Customers
WHERE Customers.cname =
        ( SELECT Activities.cname
          FROM Activities );
```

a) Some customer may have participated in no activities.
b) Some customer may have participated in exactly one activity.
c) Some customer may have participated in more than one activity.
d) For some cname in Activities, there might be no customer with that cname in Customers.
e) For some cname in Activities, there might be more than one customer with that cname in Customers.

Question 5: For the relation Employees(name, age, salary), suppose that no employees who are 65 or older, but there are employees who are under 65. Which employee names will appear in the result to the following query:

```
SELECT e1.name
FROM Employees e1
WHERE e1.salary < ALL
        ( SELECT e2.salary
          FROM Employees e2
          WHERE e2.age >= 65 );
```

a) There will be no employee names in the result.
b) All of the employee names will be in the result.
c) The result will be NULL.
d) The result will be UNKNOWN.
e) The query will cause a runtime error.

Question 6: Which statement is true for Relational Algebra Operations, where R is a relation and C1 and C2 are conditions on the attributes of R?

a) σ is Commutative: $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$
b) $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C1 \text{ AND } C2}(R)$
c) Union is Commutative (for union-compatible relations): $R \cup S = S \cup R$
d) Union is Associative (for union-compatible relations): $(R \cup S) \cup T = R \cup (S \cup T)$
e) All of the Above

Question 7: Employees(<u>name</u>, age, salary) is a relation, and the salary for an employee named Smith is 5000. What is Smith's salary after the following, assuming that this is the only transaction executing?

```
BEGIN TRANSACTION;
UPDATE Employees SET salary = salary + 1000 WHERE name='Smith';
UPDATE Employees SET salary = 2 * salary WHERE name='Smith';
ROLLBACK TRANSACTION;
```

a) 5000
b) 6000
c) 12000
d) Could be 5000, 6000 or 12000
e) None of the Above

Question 8: Employees(name, age, salary) is a relation, and the salary for an employee named Smith is 5000. Two different transactions T1 and T2 are executed by different people at approximately the same time, with Isolation Level Serializable. Both transactions commit. T1 and T2 are the only transactions executing. What is Smith's salary afterwards?

T1:

```
BEGIN TRANSACTION;
UPDATE Employees SET salary = salary + 1000 WHERE name='Smith';
COMMIT TRANSACTION;
```

T2:

```
BEGIN TRANSACTION;
UPDATE Employees SET salary = 2 * salary WHERE name='Smith';
COMMIT TRANSACTION;
```

a) Must be 5000
b) Must be 11000
c) Must be 12000
d) Must be either 11000 or 12000
e) Could be something other than 11000 or 12000

Question 9: The Employees(name, age, salary) table has been created with name as the primary key, salary having a default value of 9000, and with age and salary both allowed to be NULL. What is in the tuple with name 'Chou' after the following statement is executed, assuming that there was no tuple for 'Chou' before the statement was executed?

```
INSERT INTO EMPLOYEES(name, age)
  VALUES ('Chou', 25);
```

a) 'Chou', 25
b) 'Chou', 25, NULL
c) 'Chou', 25, 9000
d) 'Chou', NULL, 9000
e) There will be no tuple for Chou because salary was not supplied in the INSERT.

Question 10: For the relations Customers(cname, age) and Activities(cname, slopeid, date), what does the following Relational Algebra query do?

$$\Pi_{\text{Customers.cname, Customers.age}}(\sigma_{\text{Customers.cname=Activities.cname}}(\text{Customer X Activities}))$$

a) Finds cname and age for the customers who participated in at least one activity.
b) Finds cname and age for the customers who didn't participate in any activity.
c) Finds cname and age for each customer.
d) For each customer, gives cname, age and counts the number of activities that the customer participated in.
e) None of the Above

Question 11: For a table Movies(title, year, length, studio), which index will probably help the most for executing the following query?

```
SELECT title, year, length
FROM Movies
WHERE year > 2000 AND studio = 'Disney";
```

a) An index on year
b) An index on studio
c) An index on (year, studio) in that order
d) An index on (studio, year) in that order
e) Indexes on (year, studio) and (studio, year) are both the most helpful, and they're equally good.

Question 12: Assume that name is the primary key in the Beers table. This use of a CHECK constraint is legal in many SQL systems (but not in PostgreSQL):

```
CREATE TABLE Sells (
        bar     CHAR(20),
        beer    CHAR(20)      CHECK ( beer IN
                                        (SELECT name FROM Beers) ),
        price   REAL
);
```

For SQL systems in which this use of CHECK is legal, which is the best answer?

a) If you INSERT a row into the Sells table whose beer attribute value doesn't appear as a name in the Beers table, then there will be an error.
b) If you DELETE a row from the Beers table, and there is a row in the Sells table whose beer attribute value equals the beer name in the deleted row, then there will be an error.
c) If you UPDATE a row in the Sells table, changing its beer attribute value to a different beer whose name appears in the Beers table, then there will be an error.
d) Answer a) and b) are both correct, but answer c) is not correct.
e) Answer a), b) and c) are all correct.

Question 13: Which operation can an ALTER statement do?

a) Create a table
b) Drop a table
c) Delete all rows from a table
d) Add additional columns to a table
e) None of the Above

Question 14: Which of these is/are advantages of Stored Procedures?

a) Can make performance better by enabling operations to be performed on data without moving that data from the database to the client.
b) Can make programming easier by allowing code to be written once in a procedure, and then reused by anyone authorized to execute the procedure.
c) Can improve security by allowing people to be authorized to execute a procedure without authorizing them to access all the data used by the procedure.
d) All of the Above
e) None of the Above

Question 15: Sells(bar, beer, price) is a table, and RipoffBars(bar) is another table. What does this Trigger do, assuming that price represents a dollar amount?

```
CREATE TRIGGER PriceTrig
AFTER UPDATE OF price ON Sells
REFERENCING
        OLD ROW AS old_row
        NEW ROW AS new_row
FOR EACH ROW
WHEN(new_row.price – old_row.price > 8.00)
INSERT INTO RipoffBars
        VALUES(new_row.bar);
```

a) If any row in Sells has a beer that costs more than 8 dollars, then the bar in that row is inserted into RipoffBars.
b) If a row is inserted into Sells that has a beer that costs more than 8 dollars, then the bar in that inserted row is inserted into RipoffBars.
c) If price is updated for a row in Sells, and the new price is more than 8 dollars higher than the old price, then the bar in that updated row is inserted into RipoffBars.
d) If price is updated for a row in Sells, and the old price is more than 8 dollars higher than the new price, then the bar in that updated row is inserted into RipoffBars.
e) If price is updated for a row in Sells, and the new price is more than 8 dollars higher than the old price, then the update returns an error.

Question 16: Sells(bar, beer, price) is a table, where bar and beer are CHAR(20) and price is FLOAT. Assuming that myCon is a connection, what error appears in the following JDBC, which is supposed to print the beers and prices for Joe''s Bar?

```
Statement stmt = myCon.createStatement();
ResultSet Menu =
 stmt.executeQuery(" SELECT beer, price FROM Sells WHERE bar = 'Joe''s Bar' ");
while (Menu.fetch()) {
   // For each value in result, get values of beer and price, and print them
   System.out.println(Menu.getString(1), Menu.getFloat(2));
```

a) Should use *PreparedStatement*, not *Statement*
b) Should use *Menu.next*, not *Menu.fetch*
c) Should use *System.out.println(ResultSet.Float(1), ResultSet.getString(2));* not *System.out.println(Menu.getString(1), Menu.getFloat(2));*
d) Should use *stmt.updateQuery*, not *stmt.executeQuery*
e) In the SELECT statement, 'Joe''s Bar' has the single quote symbol appearing twice (between Joe and s) in *'Joe''s Bar'*, but it should only appear once, replaced by *'Joe's Bar'*

Question 17: R(A,B) and S(A,B) are union-compatible tables which may have duplicates. Here are two queries, Q1 (on the left) and Q2 (on the right) on those tables. Which statement is always correct for the results of those queries?

```
(SELECT DISTINCT *
FROM R
WHERE A > 10)

UNION ALL

(SELECT DISTINCT *
FROM S
WHERE B < 300);
```

```
(SELECT *
FROM R
WHERE A > 10)

UNION

(SELECT *
FROM S
WHERE B < 300);
```

a) The results of Q1 and Q2 must always be the same.
b) Everything in the result of Q1 must always be in the result of Q2, but the results aren't always the same.
c) Everything in the result of Q2 must always be in the result of Q1, but the results aren't always the same.
d) The result of Q1 can never have duplicates.
e) None of the Above

Question 18: In On-Line Analytic Processing (OLAP), what does the term "roll-up" mean?

a) Separating attributes in a Fact table into Dimension attributes and Dependent attributes.
b) Joining a Fact table with its underlying Dimension tables.
c) Doing an Outer Join to obtain facts that are not in the Fact Table because values for those facts are 0 or NULL.
d) Aggregating values along one or more dimensions, e.g., taking a sum.
e) Disaggregating a value into its constituent parts, e.g., breaking a sum down into the values that led to the sum.

Question 19: For the following addressbook DTD:

```
<!DOCTYPE addressbook [
  <!ELEMENT addressbook (person*)>
  <!ELEMENT person
     (name, address+, ( homephone | ( workphone, mobile )* ), email?)>
  <!ELEMENT name          (#PCDATA)>
  <!ELEMENT address       (#PCDATA)>
  <!ELEMENT homephone     (#PCDATA)>
  <!ELEMENT workphone     (#PCDATA)>
  <!ELEMENT mobile        (#PCDATA)>
  <!ELEMENT email         (#PCDATA)>
]>
```

Does the following data correspond to that DTD?

```
<addressbook>
        <person>
                <name> Abraham Lincoln </name>
                <address> Washington D.C. 20500 </address>
                <workphone> (312) 555 9876 </workphone>
                <mobile> (773) 555 2543 </mobile>
        </person>
</addressbook>
```

a) Yes, the data corresponds to the DTD.
b) No, because there's only one person.
c) No, because there's no homephone.
d) No, because there's both a workphone and a mobile.
e) No, because there's no email.

Question 20: For the table R(A, B, C), which is a lossless join decomposition? R_1(A, B) is the projection of R on attributes A, B, and R_2(B,C) is the projection of R on attributes B, C.

a) R_1(A, B) and R_2(B, C), if there are no functional dependencies
b) R_1(A, B) and R_2(B, C), if we know that A → B
c) R_1(A, B) and R_2(B, C), if we know that B → A
d) R_1(A, B) and R_2(B, C), if we know that A → C
e) R_1(A, B) and R_2(B, C), if we know that C → A

Question 21: If you want to check whether attributes salary1 and salary2 in a relation are both NULL, which way (or ways) can you write a condition in a SQL WHERE clause to test for that?

a) salary1 = NULL AND salary2 = NULL
b) salary1 IS NULL AND salary1 = salary2
c) salary1 IS NULL AND salary2 IS NULL
d) All of the Above
e) None of the Above

Question 22: Suppose that R(A, B ,C, D) is a relation, r is an instance of R, and F is a non-trivial Functional Dependency on the attributes of R. Here are two statements:

i. If r satisfies F, then F must hold for R.
ii. If r does not satisfy F, then F does not hold for R.

You're asked to supply the correct answer about these two statements.

a) Both statements are True.
b) The first statement is True and the second statement is False.
c) The first statement is False and the second statement is True.
d) Both statements are False.
e) All of the Above

Part II: (30 points, 6 points each)

Question 23: Assume we have the following tables:

- Stores table, with Primary Key store_id
- Customers table, with Primary Key customer_id
- Products table, with Primary Key product_id

These tables have other columns that aren't important for this question.

Here's the Create Statement for another table, the Sales Table, which gives information about products sold to customers in stores:

```
CREATE TABLE Sales
  ( store_id INT,
    customer_id INT,
    product_id INT,
    quantity INT,
PRIMARY KEY(store_id, customer_id, product_id) );
```

Rewrite this CREATE statement (don't do an ALTER) so that customer_id and product_id are Foreign Keys that correspond to the Primary keys of Customers and Products, respectively. Policies for Referential Integrity should be:

- A row in Customers can't be deleted if there are any Sales rows mentioning that customer.
- Deleting a row from Products deletes all Sales rows mentioning that product.

Also, the quantity for any row in Sales must be at least 1, but can't be more than 25.

Answer 23:

```
CREATE TABLE Sales (
  store_id INT,
  customer_id INT REFERENCES Customers(customer_id),
  product_id INT REFERENCES Products(product_id) ON DELETE CASCADE,
  quantity INT CHECK (quantity >=1 AND quantity <=25),
  PRIMARY KEY(store_id, customer_id, product_id)
);
```

The check could also be written as a tuple-based check at the end of the CREATE.

Question 24a): Why have any indexes in a database? Give two different clear reasons.

Answer 24a): Any two of the following reasons are good.

1. Query execution may be faster using indexes, rather than scanning entire tables to find matching tuples. For example, an index on age may improve performance of a query selecting employees whose age is 21.
2. Checking uniqueness of primary keys (or UNIQUE attributes) is much faster for a database system with indexes (on the primary keys, or on the UNIQUE attributes) than without them.
3. Checking Referential Integrity (ensuring that children have parents when there's a constraint) is much faster for a database system with indexes than without them.

Question 24b): Why not have indexes on every column in a database? Give two different reasons.

Answer 24b): Any two of the following reasons are good.

1. Indexes have to be maintained correctly when there are inserts, deletes and updates affecting the indexes. That makes those statements take longer to execute.
2. Indexes occupy additional storage on disk (or SSD).
3. Accessing indexes uses additional memory on processors.

Question 25a): For the relation Employees(name, age, salary), write a SQL UPDATE statement that finds every employee whose name begins with 'SHEL' and increases that employee's salary by 35.

Answer 25a):

```
UPDATE Employees
SET salary = salary + 35
WHERE name LIKE 'SHEL%';
```

25b): For the relation Employees(name, age, salary), write a SQL DELETE statement that deletes all employees who receive the highest salary.

Answer 25b): Here are 3 solutions that work in standard SQL.

```
DELETE FROM Employees
WHERE salary =
    ( SELECT MAX(salary) FROM Employees );

DELETE FROM Employees
WHERE salary >= ALL
    ( SELECT salary FROM Employees );

DELETE FROM Employees e1
WHERE NOT EXISTS
     ( SELECT * FROM Employees e2
       WHERE e2.salary > e1.salary );
```

Question 26a): Codd's relational algebra for sets included only the 5 operators Selection, Projection, Product, Union and Difference. But other operators can be defined using those operators.

Show that **Intersection** can be defined using these 5 operators.

Answer 26a): For two Union-Compatible Relations R and S, either of the following works. Only Difference is needed.

$R \cap S = R - (R - S)$

$R \cap S = S - (S - R)$

26b): Armstrong's Axioms for Functional Dependencies are Reflexivity, Augmentation and Transitivity. Using those axioms, we proved other Rules for Functional Dependencies, including the Union Rule.

The Union Rule is: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.

Here's the proof of the Union Rule, but with explanations missing for 3 steps in that proof. Write one of *Reflexivity*, *Augmentation* or *Transitivity* on each of the 3 empty underlines (starting with **?**) below to supply the missing explanations in the proof. It's okay to write the same explanation more than once, if that's correct.

Proof Step	**Explanation**
$X \rightarrow Z$	*Given*
So we get $XY \rightarrow YZ$	**?** *Augmentation*
$X \rightarrow Y$	*Given*
So we get $X \rightarrow XY$	**?** *Augmentation*
Therefore $X \rightarrow YZ$	**?** *Transitivity*

Question 27: Suppose that you have a relation

Inventory(StoreID, StoreName, Part, Quantity)

with the following functional dependencies:

StoreID, Part → Quantity
StoreName, Part → Quantity
StoreName → StoreID
StoreID → StoreName

27a) Is Inventory in BCNF? Justify your answer fully and clearly.
Answer 27a):

No, Inventory is not in Boyce-Codd Normal Form (BCNF).

The Functional Dependency StoreName → Store_Id is not trivial. And the left-hand side of that FD is not a superkey:

$\{StoreName\}^+ = \{StoreName, Store\_Id\}$, and that's not all the attributes of Inventory.

You can also show that Inventory is not in BCNF the same way using the Functional Dependency StoreID → StoreName. (You only had to do this for one FD.)

But the left-hand sides of the first two FDs are superkeys, so you can't use those FDs to prove that Inventory is not in BCNF.

27b) Is Inventory in 3NF? Justify your answer fully and clearly.
Answer 27b):

Yes, Inventory is in Third Normal Form (3NF).

First, let's note that both {StoreID, Part} and {StoreName, Part} are keys. We can see that they're both superkeys, since:

{StoreID, Part}$^+$ = {StoreID, Part, StoreName, Quantity}
using StoreID $\rightarrow$ StoreName and StoreID, Part $\rightarrow$ Quantity

{StoreName, Part}$^+$ = {StoreName, Part, StoreID, Quantity}
using StoreName $\rightarrow$ StoreID and StoreName, Part $\rightarrow$ Quantity

But they're also keys, since they aren't superkeys if they're "put on a diet". That is, {StoreName}$^+$ = {StoreName, Store_Id}, {StoreID}$^+$ = {StoreID, Store_Name}, and {Part}$^+$ = {Part}, and none of those closures contain all the attributes of Inventory.

None of the FDs is trivial. Now let's look at each of the FDs in turn and show that either the left-hand side is a superkey or the right-hand side is part of a key.

- For StoreID, Part $\rightarrow$ Quantity, the left-hand side is a superkey. Check.
- For StoreName, Part $\rightarrow$ Quantity, the left-hand side is a superkey. Check
- For StoreName $\rightarrow$ Store_ID, the left-hand side isn't a superkey, but the right-hand side is part of the key (StoreID, Part). Check.
- For StoreID $\rightarrow$ Store_Name, the left-hand side isn't a superkey, but the right-hand side is part of the key (StoreName, Part). Check.

Since all the FDs check out, Inventory with these FDs is in 3NF.

Part III: (32 points, 8 points each)

The familiar relations Persons, Houses and Tenants are shown below, with Primary Keys underlined. Assume Referential Integrity as follows:

- Every HouseID that appears in Persons, Ownerships or Tenants also appears as a HouseID in Houses.
- Every SSN (OwnerSSN in Landlords, LeaseTenantSSN in Tenants) also appears as an SSN in Persons.
- Every LandlordID that appears in Ownerships also appears as a LandlordID in Landlords.

Persons (SSN, Name, HouseID, ApartmentNumber, Salary)

Houses (HouseID, HouseAddress, ApartmentCount, Color)

Landlords (LandlordID, OwnerSSN, LandlordAddress)

Ownerships (LandlordID, HouseID, PurchaseDate, PropertyTax)

Tenants (HouseID, ApartmentNumber, LeaseTenantSSN, LeaseStartDate,
LeaseExpirationDate, Rent, LastRentPaidDate, RentOverdue)

Assume that no attributes can be NULL, and that there are no UNIQUE constraints.

Write SQL queries for each of the following questions. If you want to create and then use views to answer these questions, that's okay, but views are not required unless a question asks you for them.

Don't use DISTINCT in your queries unless it's necessary.

Question 28: Find the names (with no duplicates) of all persons who live in houses whose color is 'Red'. Names should appear in the result in reverse alphabetical order, so that names starting with 'Z' come before names starting with 'Y', etc.

Answer 28:

```
SELECT DISTINCT p.name
FROM Persons P, Houses H
WHERE P.HouseID = H.HouseID AND H.Color = 'Red'
ORDER BY P.name DESC;
```

Question 29: If a landlord owns a house, then there will be a row for that LandlordID and HouseID in the Ownership relation. Find the LandlordID (with no duplicates) of all landlords in the Landlord relation who don't own any houses.

Answer 29: Here are two correct answers.

```
SELECT L.LandlordID
FROM Landlords L
WHERE NOT EXISTS ( SELECT O.LandlordID
                   FROM Ownerships O
                   WHERE O.LandlordID = L.LandlordID );
```

```
SELECT L.LandlordID
FROM Landlords L
WHERE L.LandlordID <> ALL ( SELECT O.LandlordID
                            FROM Ownerships O );
```

Question 30: There can be multiple rows in the Tenants table that have the same LeaseTenant. For each person who is a LeaseTenant, find the total of the Rent for all the Tenant rows in which that person is the LeaseTenant, but only include rows in your result if the total Rent is more than 2000.

Your result should include the LeaseTenant's SSN, his name, his personal address, and the total Rent for all the rows in Tenants in which he is LeaseTenant. In your result, call the total Rent TotalRent. And, as we emphasized above, only include rows in your result if TotalRent is more than 2000.

Answer 30: Here are three correct answers; you only needed to provide one.

Answer #1 to Question 30:

```
SELECT P.SSN, P.name, H.HouseAddress, SUM(T.Rent) AS TotalRent
FROM Persons P, Houses H, Tenants T
WHERE P.SSN = T.LeaseTenantSSN AND P.HouseId = H.HouseID
GROUP BY P.SSN, P.name, H.HouseAddress
HAVING SUM(T.Rent) > 2000;
```

You could use T.LeaseTenantSSN instead of P.SSN if that appears in both the GROUP BY and the SELECT, since P.SSN = T.LeaseTenantSSN.

Answer #2 to Question 30:

This answer uses a view that doesn't group by P.name because P.name isn't in its SELECT clause.

```
CREATE VIEW BigRent(SSN, HouseAddress, TotalRent)  AS
   SELECT P.SSN, H.HouseAddress, SUM(T.Rent)
   FROM Persons P, Houses H, Tenants T
   WHERE P.SSN = T.LeaseTenantSSN AND P.HouseId = H.HouseID
   GROUP BY P.SSN, H.HouseAddress
   HAVING SUM(T.Rent) > 2000;

SELECT P.SSN, P.name, B.HouseAddress, B.TotalRent
FROM Persons P, BigRent B
WHERE P.SSN = B.SSN;
```

Answer #3 to Question 30:

This answer uses a SELECT in the FROM clause with a tuple variable.

```
SELECT P.SSN, P.name, H.HouseAddress, T2.TotalRent
FROM Persons P, Houses H, ( SELECT T.LeaseTenantSSN, SUM(T.Rent) AS TotalRent
                            FROM Tenants T
                            GROUP BY T.LeaseTenantSSN
                            HAVING SUM(T.Rent) > 2000 ) T2
WHERE P.SSN = T2.LeaseTenantSSN AND P.HouseId = H.HouseID;
```

Answer #3 could also be handled using a VIEW, and Answer #2 could also be handled using a SELECT in the FROM clause with a tuple variable **.**

Question 31: This question has two parts; be sure to answer both.

31a) Create a SQL view called Rented. For each house in the Houses relation, the Rented view should give the HouseID and the number of apartments in that house that have rows in Tenants. In the result of your view, the second attribute should be called RentedCount.

(Hint: Doing this correctly requires an Outer Join, since there could be Houses that have no Tenants. You will still get partial credit if you do a Join instead of an Outer Join.]

31b) Write a SQL query that finds all the houses that appear in the Rented view whose RentedCount in the Rented view does not equal the ApartmentCount in Houses for that house. Your result should include the HouseID, the RentedCount and the ApartmentCount for each house in Rented whose RentedCount doesn't equal its ApartmentCount.

Answers 31a) and 31b):

31a)

```
CREATE VIEW Rented AS
SELECT H.HouseID, COUNT(T.ApartmentNumber) AS RentedCount
FROM Houses H LEFT OUTER JOIN Tenants T ON H.HouseID = T.HouseID
GROUP BY H.HouseID;
```

You can use COALESCE if you want to, but it's not needed, since NULL is ignored when COUNT is computed (except when you do COUNT(*)), and COUNT on the empty set is 0.

```
CREATE VIEW Rented AS
SELECT H.HouseID, COUNT(COALESCE(T.ApartmentNumber, 0)) AS RentedCount
FROM Houses H LEFT JOIN Tenants T ON H.HouseID = T.HouseID
GROUP BY H.HouseID;
```

In this case OUTER JOIN (same as FULL OUTER JOIN) would be equivalent because in the schema, you were told to assume Referential Integrity (every HouseID in Tenants appears in Houses).

Also, you can use COUNT(*), or COUNT of other attributes, instead of COUNT(T.ApartmentNumber).

31b)

```
SELECT R.HouseID, R.RentedCount, H.ApartmentCount
FROM Rented R, Houses H
WHERE R.HouseID = H.HouseID AND R.RentedCount <> H.ApartmentCount;
```