CSE 180, Final Exam, Fall 2021, Shel Finkelstein

Student Name: _____

Student ID: _____

UCSC Email: _____

ANSWERS

Part	Max Points	Points
I	40	
II	24	
III	36	
Total	100	

The first Section (Part I) of the Fall 2021 CSE 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 the 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.

This 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 this 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 your **Scantron sheet** for the first Section of the Exam and also **this second Section of the Exam**, and show your **UCSC id** when you hand them in. Also hand in your **8.5 x 11 "cheat sheet"**, with your name written in the upper right corner of the sheet.

Part II: (24 points, 6 points each)

Question 21: Here are statements creating two table S and T:

```
CREATE TABLE S (
   c INT PRIMARY KEY,
   d INT
    );
```

```
CREATE TABLE T (
   a INT PRIMARY KEY,
   b INT,
   FOREIGN KEY(b)
         REFERENCES S(c)
            ON UPDATE CASCADE
    );
```

Relation S(c,d) currently contains the four tuples, (2,60), (3,61), (4,62), and (5,63).
Relation T(a,b) currently contains the four tuples, (0,4), (1,5), (2,4), and (3,5).

Indicate all changes that happen to both S and T when each of the following SQL statements is executed upon the S and T instances that are shown above. That is, you should ignore changes made by earlier parts of this question when you answer later parts.

If a statement changes existing tuples in a relation, show both old and new values of those tuples. If a statement makes no changes to a relation, say that.

21a): UPDATE S set c=8 WHERE d=62;
Answer 21a):
Changes made to the original S:
(4,62) → (8,62)

Changes made to the original T:
(0,4) → (0,8) and (2,4)→(2,8)

21b): DELETE FROM T WHERE a=2;
Answer 21b):
Changes made to the original S:
No change

Changes made to the original T:
(2,4) is deleted

21c): DELETE FROM S WHERE c=3;
Answer 21c):
Changes made to the original S:
(3,61) is deleted

Changes made to the original T:
No change

Question 22: This question has two parts; be sure to answer both of them.

Assume that we have the following table in our database:

Employees(empNumber, empName, dept, salary)

In which empNumber is the Primary Key.

22a): Write a SQL statement that updates the salaries of all the employees who make more than the average salary, giving them a 12% raise. For example, the salary for an employee who makes more than the average salary and whose salary is 200 would become 224 after the update.

Answer 22a):

```
UPDATE Employees e
SET e.salary = e.salary * 1.12
WHERE salary > ( SELECT AVG(salary)
                 FROM Employees) ;
```

Okay not to have the tuple variable e for Employees, but it's clearer if e is used.
Okay to say e.salary = e.salary + 0.12 * e.salary
Okay to have a tuple variable in the subquery.

22b): Do you agree or disagree with the following statement? Justify your answer.

If the statement in part a) of this question is executed, the result depends on the order in which the tuples in Employees are processed, because average salary changes each time a tuple is updated.

Answer 22b):

The statement is not correct, so you should **Disagree** with it.

For a SQL modification statement, the database system determines all changes that should be made using the "old" state of the database before making any changes, then makes all those changes, which results in the new state of the database. Hence the order in which tuples are processed doesn't matter.

Question 23: We have the following relations:

Sailors(sid, sname, rating, age)
    // sailor id, sailor name, rating, age
Boats(bid, bname, color)
    // boat id, boat name, color of boat
Reserves(sid, bid, day)
    // sailor id, boat id, and date that sailor sid reserved that boat.

Codd's Relational Algebra included only 5 operators, (σ , π , x , U , and -). Write the following query using Codd's Relational Algebra (not SQL):

> *Find the sname of Sailors whose age is 21 and who have a reserved a red boat and have also reserved a blue boat.*

If you'd like, you may also use Rename (ρ) and Assignment ($\leftarrow$) in your answer.

To simplify notation, you may write SIGMA for σ and PI for π. You may also use <-- for Assignment and RHO for ρ (Rename). Also, although you can use subscripts, you can also use square brackets, for example writing:

PI[Boats.bid] (SIGMA[Boats.bname ='Titanic'] (Boats))

instead of writing:

$\pi_{\text{Boats.bid}}$ ($\sigma_{\text{Boats.bname ='Titanic'}}$ (Boats))

Answer 23: Here's a clear way to write it using ←.

It's okay to write red instead of 'red', and the same is true for blue.
And it okay to use PI instead of π and SIGMA instead of σ, as indicated above.

RedBoatSailors ←

$$\pi_{\text{Reserves.sid}} (\sigma_{\text{Boats.bid = Reserves.bid AND Boats.color ='red'}} (\text{Boats x Reserves}))$$

BlueBoatSailors ←

$$\pi_{\text{Reserves.sid}} (\sigma_{\text{Boats.bid = Reserves.bid AND Boats.color ='blue'}} (\text{Boats x Reserves}))$$

Answer:

$$\pi_{\text{Sailors.sname}} (\sigma_{\text{Sailors.age = 21 AND Sailors.sid = RedBoatSailors.sid AND Sailors.sid = BlueBoatSailors.sid}} (\text{Sailors x RedBoatSailors x BlueBoatSailors}))$$

There are other correct ways to do this. However, you'll need to have separate appearances of Boats and Reserves (for the red boat reservations and the blue boats reservations) to do this, and you'll have to do some sort of renaming, using either ρ or ←, to avoid confusing the red and the blue boat reservations.

Obviously no boat be both red and blue, and a reservation of a red boat isn't a reservation of a blue boat.

Question 24: This question has two parts; be sure to answer both of them.

24a): You have a relation Departments(deptName, building, floor, manager), which has just the following non-trivial Functional Dependencies, no others:

deptName → building
manager, building → floor
building, floor → deptName
building, floor → manager

Is the Departments relation in 3NF? Justify your answer clearly.

Answer 24a) Yes, it's in 3NF. Let's look at the 4 non-trivial FDs that we're given, and see if each satisfies one of the 3NF properties.

1: deptName → building is an FD. (Note that we haven't been told that deptName is a key.)

That FD is not trivial, since building doesn't appear on the left.

The attribute closure of deptName is just {deptName, building}, so the left-hand side of that FD is not a superkey for Departments.

Is building part of a key for Departments? Yes, {building, floor} is a key since {building, floor}$^+$ is all the attributes of Departments. But building$^+$ is just building, and floor$^+$ is just floor.

2: manager, building → floor. It's left-hand side is a superkey, since {manager, building}$^+$ is {manager, building, floor, deptName}. For 3NF we could instead just repeat our observation that the right-hand side of the FD, floor, is part of a key {building, floor}.

3 and 4: The left-hand side of both of these FDs is building, floor, and as we've already shown, {building floor} is a superkey. (In fact, it's even a key, but it's been a superkey is enough.)

Since all of the non-trivial FDs satisfy on of the 3NF requirements, Departments is in 3NF.

[Okay to use my cat, dog, horse terminology, as long as you use do the proof correctly.]

24b): In Design Theory, what is the Update Anomaly? In your answer, explain why it is a problem by giving a clear example.

Answer 24b)

The Update Anomaly occurs when there is a FD: X → Y on a relation R, and there can be multiple tuples in an instance of R that have the same value for X.

This FD is a constraint saying that all tuples in R that have the same value for the attributes in X must also have the same value for the attribute (or attributes) in Y.

Hence if an instance of R has multiple tuples that have the same value for X, then if the value of Y is updated for one of those tuples, the value of Y also has to be updated for the the other tuples that have the same value of X. If that doesn't happen, the constraint isn't satisfied; that's the Update Anomaly.

For Example: Iif we have a table Employees(eid, name, addr, rank, salary_scale) with the FD: rank → salary_scale and we update the salary_scale of one employee whose rank is 3, then we also have to update the salary_scale of every other employee whose rank is 3. Otherwise, the FD rank → salary has been violated.

Part III: (36 points, 9 points each)

Some familiar tables appear below, with Primary Keys underlined. **These tables also appear on the last page of the Final, which you can tear off to help you do questions in Part III of the Final.** You don't have to turn in that last page at the end of the Exam.

Persons(personID, lastName, firstName, email, affiliation, isStudent)

Conferences(conferenceName, year, conferenceDate, regularAttendeeCost, studentAttendeeCost, submissionDueDate, reviewDueDate, importance)

Submissions(conferenceName, year, submissionID, numPages, submitDate, wasAccepted, submissionTitle, dateAccepted, datePublished)

Authors(authorID, conferenceName, year, submissionID, authorPosition)

Reviewers(reviewerID, conferenceName, year, reliability)

Reviews(reviewerID, conferenceName, year, submissionID, reviewDate, rating)

Attendees(attendeeID, conferenceName, year)

The Primary Key in each table is shown underlined. Assume that there aren't any UNIQUE or NOT NULL constraints specified for these tables. Data types aren't shown to keep thing simple. There aren't any trick questions about data types.

You should assume Foreign Keys as follows:

- Every (conferenceName, year) in Submissions appears as a Primary Key in Conferences.
- Every authorID in Authors appears as a Primary Key in Persons.
- Every (conferenceName, year) in Authors appears as a Primary Key in Conferences.
- Every reviewerID in Reviewers appears as a Primary Key in Persons.
- Every (conferenceName, year) in Reviewers appears as a Primary Key in Conferences.
- Every (reviewerID, conferenceName, year) in Reviews appears as a Primary Key in Reviewers.
- Every (conferenceName, year, submissionID) in Reviews appears as a Primary Key in Submissions.
- Every attendee in Attendees appears as a personID in Persons.
- Every (conferenceName, year) in Attendees appears as a Primary Key in Conferences.

Write legal SQL queries for Questions 25-28. If you want to create and then use views to answer these questions, that's okay, but views are not required unless the question asks for them.

Don't use DISTINCT in your queries unless it's necessary, 1 point will be deducted if you use DISTINCT when you don't have to do so. And some points may be deducted for queries that are very complicated, even if they are correct.

For all questions:

- Okay to use other tuple variables, or use table names instead, or use neither if the attribute is unambiguous.
- Spacing isn't important, unless it's illegal SQL.
- Dates can be written using any legal format.
- Using AS in the SELECT clause isn't necessary, although we do it.
- Using AS in the FROM clause is okay, although we don't do it.
- Correct use of JOIN ON is okay.

Question 25: Find all the reviewers who are students, who wrote a review whose review date is December 7, 2021 and whose reliability **(associated with the conference for which they wrote that December 7, 2021 review)** isn't NULL. (isStudent is a Boolean attribute in the Persons table which indicates whether the person is a student.)

The attributes in the result should be the first name of the Reviewer, the last name of the reviewer and the reliability of the reviewer. These attributes should appear as theFirstName, theLastName and theReliability.

No duplicates should appear in your result.

Answer 25: Here are two solutions.

```
SELECT DISTINCT p.firstName AS thefirstName, p.lastName AS thelastName,
                r.reliability AS theReliability
FROM Persons p, Reviewers r, Reviews review,
WHERE p.isStudent
    AND p.personID = r.reviewerID
    AND r.reviewerID = review.reviewerID
    AND r.conferenceName= review.conferenceName
    AND r.year = review.year
    AND r.reliability IS NOT NULL
    AND review.reviewDate = DATE '12/07/2021';
```

- This solution needs DISTINCT, because the student review might have written more than one review on that date.
- Okay to write "DATE '2021-12-07'" instead of "DATE '12/07/21'" in either answer.
- Okay to write"p.isStudent = TRUE" or even "p.isStudent IS TRUE" (which we didn't discuss in class), instead of "p.isStudent". You can also write T instead of TRUE.

[Second solution on next page.]

```
SELECT DISTINCT p.firstName AS thefirstName, p.lastName AS thelastName,
                        r.reliability AS theReliability
FROM Persons P, Reviewers r
WHERE p.isStudent
    AND p.personID = r.reviewerID
    AND EXISTS ( SELECT *
                    FROM Reviews review
                    WHERE r.reviewerID = review.reviewerID
                        AND r.conferenceName= review.conferenceName
                        AND r.year = review.year
                        AND r.reliability IS NOT NULL
                        AND review.reviewDate = DATE '12/07/2021' );
```

- DISTINCT is still needed, even though no personID can appear twice, because the attributes in the SELECT clause are firstName, lastName and reliability, and multiple people can have the same values for all three attributes.
- This can also be written correctly using a form of IN or using = ANY instead of EXISTS.

Question 26: Find the authors of submissions whose datePublished is after August 26, 1990 and whose highest review rating **for that submission** is greater or equal to 3. The attributes in your result should be authorID, datePublished, submissionTitle and authorPosition.

The tuples in your result should be in alphabetical order based on submissionTitle; tuples that have the same submissionTitle should appear with bigger authorPosition values appearing before smaller authorPosition values.

No duplicates should appear in your result.

Answer 26:

Here are 3 different ways to do this. In all of them, the equality for all 3 tables on conferenceName, year and submissionID has to be expressed for two pairs of tables, since that will imply equality of those attributes for the third pair of tables.

- Okay to write "DATE '2021-12-07'" instead of "DATE '12/07/21'" in either answer.
- Okay to write"p.isStudent = TRUE" or even "p.isStudent IS TRUE" (which we didn't discuss in class), instead of "p.isStudent". You can also write T instead of TRUE.
- Okay to use other tuple variables, or use table names instead, or use neither if the attribute is unambiguous.

```
SELECT DISTINCT a.authorID, s.datePublished, s.submissionTitle, a.authorPosition
FROM Authors a, Submissions s, Reviews review
WHERE a.conferenceName = s.conferenceName
    AND a.year = s.year
    AND a.submissionID = s.submissionID
    AND s.datePublished > DATE '08/26/1990'
    AND s.conferenceName = review.conferenceName
    AND s.year = review.year
    AND s.submissionID = review.submissionID
    AND review.rating >= 3
ORDER BY s.submissionTitle, a.authorPosition DESC;
```

- Need DISTINCT, because there might be multiple reviews of the author's paper with rating greater than 3.

[Other solutions on next page.]

```
SELECT DISTINCT a.authorID, s.datePublished, s.submissionTitle, a.authorPosition
FROM Authors a, Submissions s
WHERE a.conferenceName = s.conferenceName
    AND a.year = s.year
    AND a.submissionID = s.submissionID
    AND s.datePublished > DATE '08/26/1990'
    AND EXISTS ( SELECT *
                 FROM Reviews review
                 WHERE s.conferenceName = review.conferenceName
                     AND s.year = review.year
                     AND s.submissionID = review.submissionID
                     AND review.rating >= 3 )
ORDER BY s.submissionTitle, a.authorPosition DESC;
```

- Still need DISTINCT, because the author might have written multiple papers that agree on all four of the SELECT clause attributes. (Yes, even submissionTitle.)
- This can also be written correctly using a form of IN or using = ANY instead of EXISTS.

```
SELECT a.authorID, s.datePublished, s.submissionTitle, a.authorPosition
FROM Authors a, Submissions s, Reviews review
WHERE a.conferenceName = s.conferenceName
    AND a.year = s.year
    AND a.submissionID = s.submissionID
    AND s.datePublished > DATE '08/26/1990'
    AND s.conferenceName = review.conferenceName
    AND s.year = review.year
    AND s.submissionID = review.submissionID
GROUP BY a.authorID, s.datePublished, s.submissionTitle, a.authorPosition
HAVING MAX (review.rating) >= 3
ORDER BY s.submissionTitle, a.authorPosition DESC;
```

- DISTINCT is not needed because the GROUP BY gets rid of duplicates.
- In the HAVING clause, instead of MAX (review.rating) > 3 could say SOME (review.rating > 3) or ANY (review.rating > 3).
  Note the difference sin the parentheses.
- Could have GROUP BY with all the attributes in the Primary Key of Submissions, namely {conferenceName, year, submissionID), in addition to authorID. But although the same attributes could appear in the SELECT clause, for this variation you'd need to have SELECT DISTINCT, not just SELECT.

Question 27: Find all the conferences whose conferenceName has the string SIG anywhere in it (uppercase), and for which no submissions were accepted. Use the Boolean attribute wasAccepted to determine if a submission was accepted.

The attributes in your result should be conferenceName, year and the number of submissions to the conference, which should appear as theConferenceName, theConferenceYear and theSubmissionCount. (Yes, you should count all the submissions, even though none of them was accepted.)

No duplicates should appear in your result.

Answer 27: Here are two solutions. DISTINCT is not needed in either because these are GROUP BY attributes.

Here's an elegant way to do this:

```
SELECT c.conferenceName, c.year, COUNT(*) AS theSubmissionCount
FROM Conferences c, Submissions s
WHERE c.conferenceName = s.conferenceName
    AND c.year = s. year
    AND c.conferenceName LIKE '%SIG%'
GROUP BY c.conferenceName, c.year
HAVING EVERY (NOT s.wasAccepted);
```

- Okay to write "s.wasAccepted = FALSE" instead of "NOT s.wasAccepted". Could also use F instead of FALSE.

Here's another correct way which will get full credit, although it is more complicated.

```
SELECT c.conferenceName, c.year, COUNT(*) AS theSubmissionCount
FROM Conferences c, Submissions s
WHERE c.conferenceName = s.conferenceName
    AND c.year = s. year
    AND c.conferenceName LIKE '%SIG%'
    AND NOT EXISTS ( SELECT *
                     FROM Submissions s2
                     WHERE s2.conferenceName = s.conferenceName
                         AND s2.year = s.year
                         AND s2.wasAccepted );
GROUP BY c.conferenceName, c.year;
```

- Okay to write "s.wasAccepted = TRUE" instead of "s.wasAccepted". Could also use T instead of TRUE.

Note that this this doesn't capture conferences which had no submissions. Those could either with a UNION or with a LEFT OUTER JOIN.

Question 28: This question has two parts; be sure to answer both.

Sometimes a person can be an author, a reviewer and also an attendee, all for the same conference. We'll say that such a person is a "triply-involved" person for that conference. A person might be triply-involved in more than one conference.

28a): Create a view called TriplyInvolved that finds all triply-involved persons. The attributes in your result should be the ID of a triply involved person and conferenceName and year for which that person is triply involved.

No duplicates should appear in your view.

28b): Write a query that uses the TriplyInvolved view to determine triplyInvolvedCount for each conference, the number of people who are triply-involved in that conference. The attributes in your result should be conferenceName, year and triplyInvolvedCount. In your result, include only the conferences for which there is at least one triply involved person.

Be sure to use the TriplyInvolved view to do this; you may also use the original tables, if necessary.

No duplicates should appear in your result.

Answer 28a):

```
CREATE VIEW TriplyInvolved AS
        SELECT DISTINCT r.reviewerID, r.conferenceName, r.year
        FROM Reviews r, Authors au, Attendees at
        WHERE r.reviewerID = au.authorID
            AND r.conferenceName = au.conferenceName
            AND r.year = au.year
            AND r.reviewerID = at.attendeeID
            AND r.conferenceName = at.conferenceName
            AND r.year = at.year;
```

- DISTINCT is needed, because someone might be a reviewer or an author multiple times for the same conference.
- In the SELECT clause, could use ID, conferenceName and year from any of the three relations, as long as it's the correct ID for that relation.
- In the WHERE clause, only need to check for equality of the values of ID, conferenceName and year twice among the 3 relations.
- It is correct (but redundant) to also check whether the IDs all match personID, because of Referential Integrity.

Answer 28b):

```
SELECT t.conferenceName, t.year, COUNT(*) AS triplyInvolvedCount
FROM TriplyInvolved t
GROUP BY t.conferenceName, t.year;
```

- DISTINCT is not needed because these are GROUP BY attributes.
- Could use COUNT of any attribute that can't be NULL, and that's t.conferenceName or t.year.
- Should not include a check that the COUNT is greater than 0 (or is 1 or more). You won't get a group if the COUNT is 0. Checking that is unnecessary, and it reflects a misunderstanding of GROUP BY, so there will be a deduction for that.