CSE 180, Final Exam, Fall 2023, Shel Finkelstein

ANSWERS

Student Name: _____

Student ID: _____

Final Points:

Part I:	**40**
Part II:	**24**
Part III:	**36**
Total:	**100**

The first Section (Part I) of the Fall 2023 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 and Student ID filled in (including marking bubbles below) on that Scantron sheet. Please be sure to use a #2 pencil (not #3 or ink) to mark your choices on that Section of the Final.

This separate second Section (Parts II and III) of the Final is not multiple choice and is double-sided, so that you have extra space to write your answers on the facing page **after** each question. Please write your Name and Student ID on that second Section of the Exam, which you must hand in. You may use a #2 pencil or ink implement on this Section of the Exam, but make sure that your answers are clear.

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 **this second Section of the Exam**. You must also show your **UCSC ID** when you hand them in. Do not hand in your 8.5 x 11 sheet.

Part II: (24 points, 6 points each)

Question 21: Assume that we have previously created:

- a Persons table, whose Primary Key is SSN, and
- a Houses table, whose Primary Key is houseID.

Write a CREATE statement for another table:

Properties(<u>owner, houseID</u>, datePurchased)

which does everything that is described below:

A Properties tuple indicates that a person (owner) purchased a house (houseID) on the specified datePurchased. Attributes owner and SSN are integers. houseID is an integer in both Houses and Properties; datePurchased is a date.

The Primary Key of Properties is (owner, houseID). In the Properties table that you create, owner should be a Foreign Key corresponding to the Primary Key of Persons (SSN), and houseID should be a Foreign Key corresponding to the Primary Key of Houses (which is also called houseID).

When a person in Persons is deleted, tuples in Properties which that person owns should be deleted. When the Primary Key of a person in Persons is updated, then the tuples in Properties which that person owns should have their Foreign Key values updated the same way.

Deletion of a house in Houses is not permitted if there are any tuples in Properties for that houseID. Also, updating of the houseID for a house in Houses is not permitted if there are any tuples in Properties for that houseID.

Answer 21:

```
CREATE TABLE Properties (
        owner INTEGER,
        houseID INTEGER,
        datePurchased DATE,
        PRIMARY KEY (owner, houseID),
        FOREIGN KEY (owner) REFERENCES Persons (SSN)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (houseID) REFERENCES Houses (houseID)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT
        );
```

It's okay to write:

```
        FOREIGN KEY (houseID) REFERENCES Houses
```

without mentioning houseID again after Houses, since the attribute names are the same.

It's also okay to omit ON DELETE RESTRICT and ON UPDATE RESTRICT for the houseID FOREIGN KEY since RESTRICT is the default. Using NO ACTION instead of RESTRICT is also okay. But using REJECT instead of RESTRICT is not okay.

For either or both single attribute Foreign Keys, can specify Foreign Key next to the attribute, instead of as a schema element:

```
CREATE TABLE Properties (
        owner INTEGER REFERENCES Persons(SSN)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        houseID INTEGER REFERENCES Houses(houseID)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT,
        datePurchased DATE,
        PRIMARY KEY (owner, houseID) );
```

Question 22: 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 bid and color of the boats which have been reserved by a sailor whose rating is 2, and which have also been reserved by a sailor whose rating is 8.*

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[Sailors.sid] (SIGMA[Sailors.age = 21] (Sailors))

instead of writing:

$\pi_{\text{Sailors.sid}}$ ($\sigma_{\text{Sailors.age =21}}$ (Sailors))

Answer 22:

Here's one correct way to write this query in Relational Algebra.

$$\text{Reserved2} \leftarrow \pi_{\text{Reserved.bid}} \left(\sigma_{\text{Sailors.sid=Reserved.sid AND Sailors.rating=2}} (\text{Sailors X Reserved}) \right)$$

$$\text{Reserved8} \leftarrow \pi_{\text{Reserved.bid}} \left(\sigma_{\text{Sailors.sid=Reserved.sid AND Sailors.rating=8}} (\text{Sailors X Reserved}) \right)$$

$$\pi_{\text{Boats.bid, Boats.color}} \left(\sigma_{\text{Boats.bid=Reserved2.bid AND Boats.bid=Reserved8.bid}} (\text{Boats X Reserved2 X Reserved8}) \right)$$

Here are these same answers, written using words SIGMA and PI, instead of using σ and π.

Reserved2 ← Pi[Reserved.bid] (SIGMA[Sailors.sid=Reserved.sid AND Sailors.rating=2]
(Sailors X Reserved))

Reserved8 ← Pi[Reserved.bid] (SIGMA[Sailors.sid=Reserved.sid AND Sailors.rating=8]
(Sailors X Reserved))

Pi[Boats.bid, Boats.color] (SIGMA[Boats.bid=Reserved2.bid AND Boats.bid=Reserved8.bid]
(Boats X Reserved2 X Reserved8))

It's also possible to write this out as one long Relational Algebra expression, although that's uglier. Here's one such solution, with the definitions of Reserved2 and Reserved8 written in place.

$$\pi_{\text{Boats.bid, Boats.color}} \left(\sigma_{\text{Boats.bid=Reserved2.bid AND Boats.bid=Reserved8.bid}} \right.$$
(Boats
X Pi[Reserved2.bid] (SIGMA[Sailors.sid=Reserved2.sid AND Sailors.rating=2]
(Sailors X $\rho_{\text{Reserved2}}$(Reserved))
X Pi[Reserved8.bid] (SIGMA[Sailors.sid=Reserved8.sid AND Sailors.rating=8]
(Sailors X $\rho_{\text{Reserved8}}$(Reserved))
))

Question 23: Answer each of these questions with either the full word **TRUE** or the full word **FALSE**, writing your answers clearly above the blanks using capital letters. No explanation is required, and there will be no part credit.

23a): If myRateFunction is a PL/pgsql Stored Function that takes an integer parameter and returns an integer, you can execute the following at the PostgreSQL psql prompt to see the result of myRateFunction on the value 12.

```
SELECT myRateFunction(12);
```

Answer 23a): _____**TRUE**_____

23b): If your PL/pgSQL Stored Function includes the declaration:

```
DECLARE c CURSOR FOR <query>;
```

where <query> is a SQL query, then you must OPEN cursor c before accessing the results of the query by using FETCH commands.

Answer 23b): _____**TRUE**_____

23c): In a C program using libpq to access a PostgreSQL database, the following would be a legal statement, assuming that Sells(bar, beer, price) is a table in that database.

```
PGresult *res = PQexec("SELECT price FROM Sells WHERE beer = 'Bud' ");
```

Answer 23c): _____**FALSE**_____ **Needs a connection**

23d): In a C program using libpq to access a PostgreSQL database, assume that res is the result set from executing a SQL query on that database. We would write:

```
PQgetvalue(res, 3, 1)
```

to get the value of the first attribute of the third row in that result set,

Answer 23d): _____**FALSE**_____ **2, 0 rather than 3, 1**

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

Suppose that you have a relation Departments(deptName, building, floor, manager), which has just the following non-trivial Functional Dependencies.

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

24a): Is the relation Departments with these Functional Dependencies in **Boyce-Codd Normal Form**? Give a clear detailed proof of your answer.

Answer 24a)

Departments(deptName, building, floor, manager) is not in BCNF.

We could use either of the FD's

- deptName → building
- manager, building → floor

to prove that it's not in BCNF. One suffices; don't have to use both.

Neither of the FDs is trivial, since the attribute on the right-hand side doesn't appear on the left-hand side.

For the FD deptName → building, the attribute closure of the left-hand side deptName is just {deptName}, not all the attributes of Departments, so the left-hand side is not a superkey.

For the FD manager, building → floor, the attribute closure of the left-hand side {manager, building} is just {manager, building, floor}, missing deptName, so the left-hand side is not a superkey.

As mentioned earlier, it suffices to show that one of the FDs is non-trivial, and does its left-hand side is not a superkey. Don't have to do this for two FDs, but as long as your proof is correct, it's okay to demonstrate this for both FDs.

[This is the same relation Departments that was on the previous page, with the same Functional Dependencies.]

Departments(deptName, building, floor, manager)

with just the following non-trivial Functional Dependencies:

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

24b): Is the relation Departments with these Functional Dependencies in **Third Normal Form**? Give a clear detailed proof of your answer.

Answer 24b)

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 and 2: 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 being a superkey is enough.)

3: deptName → building 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.

4: manager, building → floor. Its 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}.

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

Part III: (36 points, 9 points each)

Some familiar tables appear below, with Primary Keys underlined. These tables appear briefly at the top of all 4 questions in Part III of the Final.

SubscriptionKinds(subscriptionMode, subscriptionInterval, rate, stillOffered)

Editions(editionDate, numArticles, numPages)

Subscribers(subscriberPhone, subscriberName, subscriberAddress)

Subscriptions(subscriberPhone, subscriptionStartDate, subscriptionMode, subscriptionInterval, paymentReceived)

Holds(subscriberPhone, subscriptionStartDate, holdStartDate, holdEndDate)

Articles(editionDate, articleNum, articleAuthor, articlePage)

ReadArticles(subscriberPhone, editionDate, articleNum, readInterval)

Assume that no attributes can be NULL, and that there are no UNIQUE constraints. Data types aren't shown to keep things simple. There aren't any trick questions about data types.

You should assume Foreign Keys as follows:

- Every subscriberPhone in Subscriptions appears as a subscriberPhone in Subscribers.
- Every (subscriptionMode, subscriptionInterval) in Subscriptions appears as a (subscriptionMode, subscriptionInterval) in SubscriptionKinds.
- Every (subscriptionStartDate, subscriptionMode) in Holds appears as a (subscriptionStartDate, subscriptionMode) in Subscriptions.
- Every editionDate in Articles appears as an editionDate in Editions.
- Every subscriberPhone in ReadArticles appears as a subscriberPhone in Subscribers.
- Every (editionDate, articleNum) in ReadArticles appears as an (editionDate, articleNum) in Articles.

Write legal SQL 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. (Of course, points will also be deducted if DISTINCT is needed and you don't use it.) And some points may be deducted for queries that are over-complicated, even if they are correct.

- **Okay to use tuple variables in queries, or use table names if the same table doesn't appear twice, or omit table names in attributes aren't ambiguous.**
- **Spacing doesn't matter in your answers.**
- **Capitalization doesn't matter, except in character string constants or patterns.**
- **No deduction if semi-colon is missing from end of statement.**
- **There is a deduction if you use DISTINCT and it's not needed, or if you don't use DISTINCT and it is needed.**
- **Order in which tables appear in the FROM clause doesn't matter.**
- **Order in which AND'ed conditions appear in the WHERE clause doesn't matter.**

There may be many different correct ways to write a SQL query, not just the ones that we show. However, there may be a deduction if you include unnecessary parts in a SQL query, even if it is correct.

SubscriptionKinds(subscriptionMode, subscriptionInterval, rate, stillOffered)

Editions(editionDate, numArticles, numPages)

Subscribers(subscriberPhone, subscriberName, subscriberAddress)

Subscriptions(subscriberPhone, subscriptionStartDate, subscriptionMode, subscriptionInterval, paymentReceived)

Holds(subscriberPhone, subscriptionStartDate, holdStartDate, holdEndDate)

Articles(editionDate, articleNum, articleAuthor, articlePage)

ReadArticles(subscriberPhone, editionDate, articleNum, readInterval)

Question 25: paymentReceived is an attribute in Subscriptions which indicates whether payment has been received for that subscription.

Write a SQL statement that defines a view PaidSubscribers. There should be a tuple in PaidSubscribers for each subscriber who has at least one subscription, if payment has been received for every subscription of that subscriber.

The attributes in your view should be called theSubscriberPhone (which should be the phone of the subscriber) and theSubscriberAddress (which should be the address of the subscriber).

No duplicates should appear in your result.

Answer 25:

```
CREATE VIEW PaidSubscribers AS
      SELECT DISTINCT sr.subscriberPhone AS theSubscriberPhone,
                                    sr.subscriberAddress AS theSubscriberAddress
      FROM Subscribers sr, Subscriptions sn
      WHERE sr.subscriberPhone = sn.subscriberPhone
          AND NOT EXISTS
                      ( SELECT * FROM  Subscriptions sn2
                        WHERE sn2.paymentReceived = FALSE
                            AND sr.subscriberPhone = sn2.subscriberPhone );
```

- DISTINCT is needed because a subscriber may have many subscriptions.
- Can write NOT sn2.paymenbtReceived, instead of sn2.paymentReceived = FALSE. Okay to use F or f instead of FALSE. Okay to say "IS FALSE".
- In the subquery, okay to have the last clause to use sn instead of sr:
  AND sn.subscriberPhone = sn2.subscriberPhone)

The NOT EXISTS can be rewritten "NOT IN" (or "<> ALL"); DISTINCT is still needed.

```
CREATE VIEW PaidSubscribers AS
        SELECT DISTINCT sr.subscriberPhone AS theSubscriberPhone,
                                        sr.subscriberAddress AS theSubscriberAddress
        FROM Subscribers sr, Subscriptions sn
        WHERE sr.subscriberPhone = sn.subscriberPhone
            AND sr.subscriberPhone NOT IN
                        ( SELECT sn2.subscriberPhone FROM  Subscriptions sn2
                          WHERE sn2.paymentReceived = FALSE );
```

We can also use EXISTS to say that there's at least one subscription, instead of having Subscriptions in the FROM clause. In these versions, DISTINCT isn't needed.

```
CREATE VIEW PaidSubscribers AS
        SELECT sr.subscriberPhone AS theSubscriberPhone,
                                sr.subscriberAddress AS theSubscriberAddress
        FROM Subscribers sr
        WHERE EXISTS
                    ( SELECT *
                      FROM Subscriptions sn
                      WHERE sn.subscriberPhone = sr.subscriberPhone )
            AND NOT EXISTS
                    ( SELECT * FROM  Subscriptions sn2
                      WHERE sn2.paymentReceived = FALSE
                          AND sn2.subscriberPhone = sr.subscriberPhone );
```

The EXISTS can also be rewritten using "IN" or "= ANY":

```
CREATE VIEW PaidSubscribers AS
        SELECT sr.subscriberPhone AS theSubscriberPhone,
                                sr.subscriberAddress AS theSubscriberAddress
        FROM Subscribers sr
        WHERE EXISTS
                    ( SELECT *
                      FROM Subscriptions sn
                      WHERE sn.subscriberPhone = sr.subscriberPhone )
            AND sr.subscriberPhone NOT IN
                    ( SELECT * FROM  Subscriptions sn2
                      WHERE sn2.paymentReceived = FALSE );
```

Finally, here's an elegant answer that using GROUP BY with EVERY in the HAVING clause:

```
CREATE VIEW PaidSubscribers AS
        SELECT sr.subscriberPhone AS theSubscriberPhone,
                                sr.subscriberAddress AS theSubscriberAddress
        FROM Subscribers sr, Subscriptions sn
        WHERE sr.subscriberPhone = sn.subscriberPhone
        GROUP BY sr.subscriberPhone
        HAVING EVERY( sn.paymentReceived = TRUE );
```

- There will be no group in the result for subscribers who have no subscriptions.
- DISTINCT is not needed because there will only one tuple for each subscriber.
- Okay to write T or t instead of TRUE, and okay to write IS TRUE.
- Also okay to just say HAVING EVERY(sn.paymentReceived);
- The GROUP BY could also include sr.subscriberAddress, but that's not necessary because sr.subscriberPhone is the Primary Key of Subscribers.

SubscriptionKinds(subscriptionMode, subscriptionInterval, rate, stillOffered)

Editions(editionDate, numArticles, numPages)

Subscribers(subscriberPhone, subscriberName, subscriberAddress)

Subscriptions(subscriberPhone, subscriptionStartDate, subscriptionMode, subscriptionInterval, paymentReceived)

Holds(subscriberPhone, subscriptionStartDate, holdStartDate, holdEndDate)

Articles(editionDate, articleNum, articleAuthor, articlePage)

ReadArticles(subscriberPhone, editionDate, articleNum, readInterval)

Question 26: articlePage is an attribute in Articles which indicates the page on which an article appears. readInterval is an attribute in the ReadArticles table which tells us how long a particular subscriber spent reading a particular article.

Write a SQL statement that finds all the articles in Articles:`

a) which appear on page 1 in their edition, and
b) where the average time spent by readers of that article is 5 minutes or more.

Your result should include attributes editionDate, articleNum and articleAuthor from the Articles table. Result tuples should appear in reverse alphabetical order based on articleAuthor. If tuples have the same author, tuples with earlier editionDate values should appear before later editionDate values.

No duplicates should appear in your result.

Answer 26:

```
SELECT a.editionDate, a.articleNum, a.articleAuthor
FROM Articles a, readArticles ra
WHERE a.articlePage = 1
    AND ( ra.editionDate, ra.articleNum ) = ( a.editionDate, .a.articleNum )
GROUP BY ra.editionDate, ra.articleNum
HAVING AVG(readInterval) >= INTERVAL '5 minutes'
ORDER BY a.articleAuthor DESC, a.editionDate;
```

- DISTINCT is not needed due to the GROUP BY
- ASC can be used in the ORDER BY for a.editionDate, but that's the default.
- Can write ra.editionDate = a.editionDate AND ra.articleNum .a.articleNum instead of (ra.editionDate, ra.articleNum) = (a.editionDate, .a.articleNum)
- Okay to write INTERVAL '00:05:00' for the 5 minutes.

Here's another way to write this that's okay.

```
SELECT a.editionDate, a.articleNum, a.articleAuthor
FROM Articles a
WHERE a.articlePage = 1
    AND (a.editionDate, a.articleNum) IN
                ( SELECT ra.editionDate, ra.articleNum
                  FROM ReadArticles ra
                  GROUP BY ra.editionDate, ra.articleNum
                  HAVING AVG(ra.readInterval) >= INTERVAL '5 minutes' )
ORDER BY a.articleAuthor DESC, a.editionDate;
```

SubscriptionKinds(subscriptionMode, subscriptionInterval, rate, stillOffered)

Editions(editionDate, numArticles, numPages)

Subscribers(subscriberPhone, subscriberName, subscriberAddress)

Subscriptions(subscriberPhone, subscriptionStartDate, subscriptionMode, subscriptionInterval, paymentReceived)

Holds(subscriberPhone, subscriptionStartDate, holdStartDate, holdEndDate)

Articles(editionDate, articleNum, articleAuthor, articlePage)

ReadArticles(subscriberPhone, editionDate, articleNum, readInterval)

Question 27: editionDate identifies an edition in the Editions table. numArticles is another attribute in the Editions table.

editionDate is also an attribute in the Articles table. For a particular edition, we can count the number of articles there are in the Articles table for that particular edition. Let's call that the *Computed Article Count* of that edition. It's possible that the *Computed Article Count* for an edition isn't equal to the numArticles value for that edition in the Editions table.

Write a query that outputs just the editions for which the *Computed Article Count* isn't equal to numArticles for that edition. The attributes in your result should be editionDate, numArticles, and computedArticleCount, where computedArticleCount is the *Computed Article Count* for that edition.

No duplicates should appear in your result.

Careful: *There might be some editions for which* the *Computed Article Count is 0, but numArticles is not 0. Since these values aren't equal for such editions, those editions should appear in the result of your query.*

Answer 27:

```
SELECT e.editionDate, e.numArticles, COUNT(*) AS articleCount
FROM Editions e, Articles a
WHERE e.editionDate = a.editionDate
GROUP BY e.editionDate, e.numArticles
HAVING COUNT(*) != e.numArticles

UNION

SELECT e.editonDate, e.numArticles, 0 AS articleCount
FROM Editions e
WHERE e.numArticles != 0
    AND NOT EXISTS
            ( SELECT *
              FROM Articles a
              WHERE e.editionDate = a.editionDate );
```

- DISTINCT is not needed.
- In the above solution, could have COUNT of any attribute that can't be NULL, instead of COUNT(*), in both that HAVING and SELECT clause.
- Can't use articleCount in the HAVING clause, because it's not defined until the SELECT clause is evaluated.
- In the SELECT clause after UNION, could have anything legal after the SELECT.
- Without the query after the UNION, the situation where there are no articles in an edition, but e.numArticles isn't 0 would not be handled correctly.
- Would be fine to do this with a view that computes the *Computed Article Count* value for an edition, as long as it always gets the correct value (including when that value is 0). Could use the view in a query that determine non-matching values of numArticles and *Computed Article Count*.

Here's another correct solution using LEFT OUTER JOIN.

```
SELECT e.editionDate, e.numArticles, COUNT(a.editionDate) AS articleCount
FROM Editions e, LEFT OUTER JOIN Articles a
        ON e.editionDate = a.editionDate
GROUP BY e.editionDate, e.numArticles
HAVING COUNT(a.editionDate) != e.numArticles;
```

- In the COUNT that appears in SELECT and HAVING clauses, could use any attribute from the Articles table. But can't use COUNT(*), which will give the wrong count value (it will be 1) when there are no articles in an edition.
- Okay to write LEFT JOIN instead of LEFT OUTER JOIN.

SubscriptionKinds(subscriptionMode, subscriptionInterval, rate, stillOffered)

Editions(editionDate, numArticles, numPages)

Subscribers(subscriberPhone, subscriberName, subscriberAddress)

Subscriptions(subscriberPhone, subscriptionStartDate, subscriptionMode, subscriptionInterval, paymentReceived)

Holds(subscriberPhone, subscriptionStartDate, holdStartDate, holdEndDate)

Articles(editionDate, articleNum, articleAuthor, articlePage)

ReadArticles(subscriberPhone, editionDate, articleNum, readInterval)

Question 28: A tuple in Holds is a hold on a particular subscription in Subscriptions. A subscription in Subscriptions corresponds to a subscriber in Subscribers, and to a subscriber kind in SubscriberKinds.

Write a SQL statement that finds all the holds which have all of the following properties:

a) The end date for the hold is December 12, 2023.
b) The end date of the subscription is after January 5, 2024. (You'll have to calculate the subscription's end date.)
c) The name of the subscriber corresponding to that hold has 'usk' (with that capitalization) appearing anywhere in it.
d) The rate of the subscription being held is more than '56.78'.

The attributes which should appear in your result for a hold are the subscriberName, holdStartDate and holdEndDate for that hold. No duplicates should appear in your result.

Answer 28:

```
SELECT DISTINCT sr.subscriberName, h.holdStartDate, h.holdEndDate
FROM Subscribers sr, Holds h, Subscriptions sn, SubscriptionKinds sk
WHERE h.holdEndDate = DATE('2023-12-12')
    AND sn.subscriptionStartDate + sn.subscriptionInterval
            > DATE('2024-01-05')
    AND sr.subscriberName LIKE '%usk%'
    AND sk.rate > 56.78
    AND ( sk.subscriptionMode, sk.subscriptionInterval )
            = ( sn.subscriptionMode, sn.subscriptionInterval )
    AND sr.subscriberPhone = sn.subscriberPhone
    AND ( sn.subscriberPhone, sn.subscriptionStartDate )
            =(  h.subscriberPhone, h.subscriptionStartDate );
```

- DISTINCT is needed for this query, since there could be multiple holds for the same subscriber which have the send holdEndDate and satisfy the other conditions in this query.
- Okay to write (DATE '12/12/23') and DATE '01/04/24', although that format isn't universally accepted.
- The keyword DATE is needed in CSE 180 before a date constant.
- Okay to write " DATE(sn.subscriptionStartDate + sn.subscriptionInterval)" with the keyword DATE at the front although DATE isn't needed.
- Okay to write either or both of the equality tests on pairs of attributes as two conditions AND'ed together:

```
    AND sk.subscriptionMode = sn.subscriptionMode
    AND sk.subscriptionInterval = sn.subscriptionInterval

    AND sn.subscriberPhone = h.subscriberPhone
    AND sn.subscriptionStartDate = h.subscriptionStartDate
```

- There are 4 tests comparing attributes to constants, and 5 tests involving attribute equality for join conditions, assuming that the equality tests on pairs of attributes count as 2 tests.

The only two tables that appear in the SELECT clause are Subscribers and Holds, so either or both of the other two tables could be handled using EXISTS (or even rewriting using IN). Here's an example of that.

```
SELECT DISTINCT sr.subscriberName, h.holdStartDate, h.holdEndDate
FROM Subscribers sr, Holds h,
WHERE h.holdEndDate = DATE('2023-12-12')
    AND sr.subscriberName LIKE '%usk%'
    AND EXISTS
            ( SELECT *
              FROM Subscriptions sn, SubscriptionKinds sk
              WHERE sk.rate > 56.78
                 AND sn.subscriptionStartDate + sn.subscriptionInterval
                                > DATE('2024-01-05')
                 AND ( sk.subscriptionMode, sk.subscriptionInterval )
                            = ( sn.subscriptionMode, sn.subscriptionInterval )
                 AND sr.subscriberPhone = sn.subscriberPhone
                 AND ( sn.subscriberPhone, sn.subscriptionStartDate )
                            =(  h.subscriberPhone, h.subscriptionStartDate );
```

- DISTINCT is still needed, since there could be multiple holds for the same subscriber which have the send holdEndDate and satisfy the other conditions in this query.
- It would even be legal to move the conditions on Subscribers and Holds into the subquery.