Fundamentals of Database Systems

Assignment: 2

Due Date: 18th August, 2017

Instructions

This question paper contains 10 questions in 6 pages.

Q1: Consider the following schema for an office payroll system, where the primary keys are underlined and the foreign keys are italicized.

Person(pid, fname, lname)
Employee(*pid*, desig, salary)

Write a relational algebra query to update the salary of all employees to the average salary for their designation.

A. $\boldsymbol{Employee \leftarrow \pi_{pid,desig,salary=avg(salary)}(Employee * (_{desig}\mathcal{G}_{avg(salary)}(Employee)))}$

B. $\pi_{pid,desig,salary=avg(salary)}\sigma_{salary \neq avg(salary)}(Employee * (_{desig}\mathcal{G}_{avg(salary)}(Employee)))$

C. $Employee \leftarrow \pi_{pid,salary=avg(salary)}(Employee * (_{desig}\mathcal{G}_{avg(salary)}(Employee)))$

D. $Employee \leftarrow \sigma_{salary \neq avg(salary)}(Employee * (_{desig}\mathcal{G}_{avg(salary)}(Employee)))$

Explanation: Updation of a set of tuples require the tuples to be selected, some function/operation on the required attribute of the selected tuple and assignment to relation without changing the schema.

Q2: Consider the following schema for an office payroll system, where the primary keys are underlined and the foreign keys are italicized.

Person(pid, fname, lname)
Employee(*pid*, desig, salary)

Write an SQL query for the following relational algebra query.

$$Employee \leftarrow \pi_{pid,desig,salary=avg(salary)}(Employee * (_{desig}\mathcal{G}_{avg(salary)}(Employee)))$$

A.
```
UPDATE Employee
SET salary=avg(salary)
WHERE salary != avg
GROUP BY desig;
```

B.
```
SELECT salary=avg(salary), desig
FROM Employee
GROUP BY desig;
```

C.
```
WITH t as (SELECT desig as d, avg(salary) as avg
FROM Employee
GROUP BY desig)
UPDATE Employee, t
SET salary=avg
WHERE desig = d;
```

D. SELECT pid, desig, salary
FROM Employee
WHERE salary = avg(salary)
GROUP BY desig;

Explanation: Option C is correct, as it groups designations calculating average salary for each, and then updates the table appropriately.
A is incorrect since GROUP BY cannot be used in an UPDATE query.
B and D are incorrect because they do not perform any update, as well as logical mistakes.

Q3: Consider the following schema, where the primary key is underlined.

Person(pid, first_name, last_name, address)

Consider the following queries.

- SQL Query: SELECT first_name, last_name FROM Person;
- Relational Algebra Query: $\pi_{first\_name,last\_name}(Person)$

Can the two queries have different number of tuples in their outputs?

A. Yes

B. No

C. Not enough information

D. The two queries are not the same.

Explanation: The two queries are same and can have different number of tuples in their answer sets based on data, since relational algebra queries output distinct tuples or a set while SQL queries are by default multi-set in nature.

Q4: From the following options, select the command syntax that suggests that SQL is a Data Manipulative Language (DML).

A. INSERT INTO *table-name*
VALUES (*comma separated list of values*);

B. DROP TABLE *table-name*;

C. ALTER TABLE *table-name*
DROP *attribute-name*;

D. None of the above

Explanation: INSERT command is one of the DML commands in SQL while DROP and ALTER commands are part of DDL commands.

Q5: From the following options, select the command syntax that suggests that SQL is a Data Definition Language (DDL).

A. UPDATE *table-name*
SET *attribute-name = value*;

B. DELETE FROM *table-name*
WHERE *predicate*;

C. CREATE TABLE *table-name* (*attribute-name datatype*);

D. All of the above

Explanation: CREATE TABLE is a DDL command while UPDATE and DELETE are DML commands.

Q6: Given below is the database schema of a hotel, where the primary keys are underlined and the foreign keys are italicized.

Room(room_no, intercom_no, tariff)
Customer(cid, name, contact, address, staying)
Checkin(chid, *cid*, *room_no*, checkinTimestamp, checkoutTimestamp)

The *staying* attribute of the Customer table stores indicates if the customer is currently staying; if yes, the value is 'Yes', otherwise, it is 'No'. The system needs a trigger that changes the value of *staying* to 'No' when the check-out time is updated in the *Checkin* table. Assuming that the check-out time is updated one at a time, select the best possible answer that defines the above trigger.

A. CREATE TRIGGER update_staying
AFTER UPDATE of checkoutTimestamp on Checkin
BEGIN
UPDATE Customer as C
SET staying = 'No'
WHERE C.cid = cid
END

B. CREATE TRIGGER update_staying
BEFORE INSERT on Checkin
BEGIN
UPDATE Customer as C
SET staying = 'No'
WHERE C.cid = cid
END

C. CREATE TRIGGER update_staying
BEFORE UPDATE of checkoutTimestamp on Checkin
BEGIN
UPDATE Customer as C
SET staying = 'No'
WHERE C.cid = cid
END

D. CREATE TRIGGER update_staying
AFTER INSERT on Checkin
BEGIN
UPDATE Customer as C
SET staying = 'No'
WHERE C.cid = cid
END

Explanation: The query to record check-out time will be an update query and the staying status of a customer should be changed only after the check-out time has been updated from null to some value.

Q7: Given below is the database schema of a hotel, where the primary keys are underlined and the foreign keys are italicized.

Room(room_no, intercom_no, tariff)
Customer(cid, name, contact, address, staying)
Checkin(chid, *cid*, *room_no*, checkinTimestamp, checkoutTimestamp)

What is the result of the following query?

```
UPDATE Room
SET tariff = 1.15*tariff
WHERE 1;
```

A. Tariff of some rooms is increased by 15% for 'room_no' 1.

B. Tariff of 1 room is increased by 15%.

C. Tariff of all rooms is increased by 15%.

D. None of the above.

Explanation: C is correct.
Condition '1' evaluates to TRUE, thus tariff is increased for all rooms by 15%.

Q8: Which is the keyword used in SQL to remove a table?

A. REMOVE

B. DELETE

C. ALTER

D. DROP

Explanation: SQL specifications.

Q9: Consider the following database schema for an online game, where the primary keys are underlined and the foreign keys are italicized.

Account(id, *username*, email, password)
Player(username, *cid*, attack-value, attack-rank, defense-value, defense-rank, gold, rank)
Attack(aid, *attacker*, *defender*, result, gold_stolen)
Clan(cid, cname)

Each account is tied to a player name. Players attack each other to steal gold and raise their *attack-value*, *defense-value* to earn better rank.

Players can spend their gold to change their *attack-value*, *defense-value* at any point. Given a state of the game at any point, *attack-value* and *defense-value* can be ranked individually. Highest *attack-value* results in the *attack-rank* 1 and so on. At every hour mark, the ranks are updated based on values at that point.

Which SQL query would be used to update the *attack-rank* and *defense-rank* into the Player table?

Note: You may assume that "Rank()" is a function that simply gives the rank of each row of a query with respect to the other rows. Syntax of Rank() is

```
Rank() OVER ( [ query_partition_clause] ORDER BY clause )
```

A. WITH t1 as (
(SELECT username, attack-value AS arank FROM Player ORDER BY attack-value)
NATURAL JOIN
(SELECT username, defense-value AS drank FROM Player ORDER BY defense-value)
)
UPDATE Player
SET attack-rank = arank, defense-rank = drank;

B. WITH t1 as
(SELECT username, attack-value AS arank FROM Player ORDER BY attack-value),
t2 as
(SELECT username, defense-value AS drank FROM Player ORDER BY defense-value)
UPDATE Player
SET attack-rank = t1.arank, defense-rank = t2.drank;

C. WITH t1 as (
(SELECT username, Rank() OVER (ORDER BY attack-value DESC) AS arank FROM Player)
NATURAL JOIN
(SELECT username, Rank() OVER (ORDER BY defense-value DESC) as drank FROM Player)
)
UPDATE Player NATURAL JOIN t1
SET attack-rank = arank, defense-rank = drank;

D. WITH t1 as
(SELECT username, Rank() OVER (ORDER BY attack-value DESC) AS arank FROM Player)
t2 as
(SELECT username, Rank() OVER (ORDER BY defense-value DESC) as drank FROM Player)
UPDATE t1 NATURAL JOIN t2
SET attack-rank = t1.arank, defense-rank = t2.drank;

Explanation: C
*t*1 is a natural join on ordering on attack and defense values. Rank on such ordering is taken from this table and inserted in the Player table.

Q10: Consider the following database schema for an online game, where the primary keys are underlined and the foreign keys are italicized.

Account(id, *username*, email, password)
Player(username, *cid*, attack-value, attack-rank, defense-value, defense-rank, gold, rank)
Attack(aid, *attacker*, *defender*, result, gold_stolen)
Clan(cid, cname)

Each account is tied to a player name. Players attack each other to steal gold and raise their *attack-value*, *defense-value* to earn better rank.

The summation of *attack-rank* and *defense-rank* decides a player's total rank, with the lowest sum yielding best rank. For example, someone having rank #1 on *attack-value* and rank #100 on *defense-value* (sum 101) will be ranked worse than someone who has rank #25 of both (sum 50).

What would be the SQL query to update current rank of the players (after the *attack-rank* and *defense-rank* has been updated)?

Note: You may assume that "Rank()" is a function that simply gives the rank of each row of a query with respect to the other rows. Syntax of Rank() is
`Rank() OVER ( [ query_partition_clause] ORDER BY clause )`

A. UPDATE Player
NATURAL JOIN
(SELECT username, Rank() OVER (ORDER BY (attack-rank + defense-rank) ASC) AS overall_rank FROM Player) SET rank = overall_rank;

B. UPDATE Player
SET rank = attack-rank + defense-rank;

C. UPDATE Player
NATURAL JOIN
(SELECT username, Rank() OVER (ORDER BY attack-value ASC) AS a_rank, Rank() OVER (ORDER BY defense-value ASC) AS d_rank FROM Player)
SET rank = a_rank + d_rank;

D. UPDATE Player
NATURAL JOIN
(SELECT username, Rank() OVER (ORDER BY (attack-value + defense-value) ASC) AS overall_rank FROM Player) SET rank = overall_rank;

Explanation: A is correct.
Since *attack-rank* and *defense-rank* are already present, we order on their sum and assign rank thereafter. Ordering on the *attack-value* and *defense-value* as in opion D, will not be reliable since their ranks need not be directly proportional to values. B and C are clearly wrong since they are basically setting the final rank as sum of individual ranks, instead of ordering on that sum.