2020

Database Systems

ASSIGNMENT: 4

FOR ALL SECTIONS

Question No: 1

a. Consider the **bank** database:
branch(*branch_name*, *branch city*, *assets*)
customer (*customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*customer_name*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*customer_name*, *account_number*)

Let us define a view ***branch_cust*** as follows:

create view *branch_cust* **as**
select *branch_name, customer_name*
from *depositor, account*
where *depositor.account_number = account.account_number*

Suppose that the view is materialized; that is, the view is computed and stored. Write triggers to maintain the view, that is, to keep it up-to-date on **insertions** from depositor or account. Do not bother about updates.

b. Consider the bank database given in part a. Write an SQL trigger to carry out the following action: On **delete** of an account, for each owner of the account, check if the owner has any remaining accounts, and if she does not, delete her from the depositor relation.

c. Consider a database that includes the following relations:
salaried-worker (name, office, phone, salary)
hourly-worker (name, hourly-wage)
address (name, street, city)

Suppose that we wish to require that every name that appears in address appear in either salaried-worker or hourly-worker, but not necessarily in both. Write a syntax for expressing such constraints

d. Using the relations of our sample bank database given in part a, write an SQL expression to define the following views:
   a. A view containing the account numbers and customer names (but not the balances) for all accounts at the Deer Park branch.
   b. A view containing the names and addresses of all customers who have an account with the bank, but do not have a loan.
   c. A view containing the name and average account balance of every customer of the Rock Ridge branch.

e. For each of the views that you defined in part d, explain how updates would be performed (if they should be allowed at all).

Question No: 2

a. Define DDL and explain.

b. Define views and explain its uses.

c. Define triggers and explain its uses.

Question No: 3

a. Create Table **StudentDetails**, where ID is unique.

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

b. Create Table **StaudentMarks**, where ID is unique.

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

c. Create a View named **DetailsView** from the table StudentDetails, which will have name and address of StudentDetails if which S_ID < 5.

d. Display **DetailsView.**

e. Create a view named **StudentNamesView** from the table StudentDetails, which will have s_ID and Name. Order the **StudentNames** by name.

f. Display **StudentNamesView.**

g. Create a View named **MarksView** from two tables **StudentDetails** and **StudentMarks**, which will have name, address and marks of the students.

h. Display **MarksView.**

i. On **MarksView** write a query to find students who got marks 80 or above.

j. On **MarksView** write a query to find student who got highest marks.

k. Drop **MarksView.**

Question No: 4

a. Create Table **StaudentMarks**, where ID is unique.

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

b. Delete tuple where NAME is Ram.

c. Update marks=92 in tuple where ID is 1.

Question No: 5

a. Create Table **StaudentMarks**, where ID is unique.

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

b. Creates a **row-level** trigger for the **StaudentMarks** table that would fire for INSERT or UPDATE or DELETE operations performed on the **StaudentMarks** table. This trigger will display the marks difference between the old values and new values