COMP3311 21T1

Week 07
Constraints, Triggers, and Aggregates

Database Systems

[Show with no answers] [Show with all answers]

1. Consider a schema for an organisation

```
Employee(id:integer, name:text, works_in:integer, salary:integer, ...)
Department(id:integer, name:text, manager:integer, ...)
```

Where `works_in` is a foreign key indicating which Department an Employee works in, and `manager` is a foreign key indicating which Employee is the manager of a Department.

A manager must work in the Department they manage. Can this be checked by standard SQL table constraints? If not, why not?

If it cannot be checked by standard SQL constraints, write an assertion to ensure that each manager also works in the department they manage. Define using a standard SQL assertion statement like:

```
create assertion manager_works_in_department
check  ...
```

[show answer]

2. Using the same schema from the previous question, write an assertion to ensure that no employee in a department earns more than the manager of their department. Define using a standard SQL assertion statement like:

```
create assertion employee_manager_salary
check  ...
```

[show answer]

3. What is the SQL command used in PostgreSQL to define a trigger? And what is the SQL command to remove it?

[show answer]

4. Trigers can be defined as `BEFORE` or `AFTER`. What exactly are they *before* or *after*?

[show answer]

5. Give examples of when you might use a trigger `BEFORE` and `AFTER` ...

    a. an insert operation

[show answer]

b. an update operation

[show answer]

c. a delete operation

[show answer]

6. Consider the following relational schema:

```
create table R(a int, b int, c text, primary key(a,b));
create table S(x int primary key, y int);
create table T(j int primary key, k int references S(x));
```

State how you could use triggers to implement the following constraint checking (hint: revise the material on Constraint Checking from the Relational Data Model and ER-Relational Mapping extended notes)

a. primary key constraint on relation `R`

[show answer]

b. foreign key constraint between `T.j` and `S.x`

[show answer]

7. Explain the difference between these triggers

```
create trigger updateS1 after update on S
for each row execute procedure updateS();

create trigger updateS2 after update on S
for each statement execute procedure updateS();
```

when executed with the following statements. Assume that `S` contains primary keys (1,2,3,4,5,6,7,8,9).

a. `update S set y = y + 1 where x = 5;`

[show answer]

b. `update S set y = y + 1 where x > 5;`

[show answer]

8. What problems might be caused by the following pair of triggers?

```
create trigger T1 after insert on Table1
for each row execute procedure T1trigger();

create trigger T2 after update on Table2
for each row execute procedure T2trigger();

create function T1trigger() returns trigger
as $$
begin
update Table 2 set Attr1 = ...;
end; $$ language plpgsql;

create function T2trigger() returns trigger
as $$
begin
insert into Table1 values (...);
end; $$ language plpgsql;
```

[show answer]

9. Given a table:

```
Emp(empname:text, salary:integer, last_date:timestamp, last_usr:text)
```

Define a trigger that ensures that any time a row is inserted or updated in the table, the current user name and time are stamped into the row. The trigger should also ensure that an employee's name is given and that the salary has a positive value.

The two PostgreSQL builtin functions `user()` and `now()` will provide the values that you need for the "stamp".

[show answer]

10. Consider the following relational schema:

```
Enrolment(course:char(8), sid:integer, mark:integer)
Course(code:char(8), lic:text, quota:integer, numStudes:integer);
```

Define triggers which keep the `numStudes` field in the `Course` table consistent with the number of enrolment records for that course in the `Enrolment` table, and also ensure that new enrolment records are rejected if they would cause the quota to be exceeded.

[show answer]

11. Consider the following (partial) relational schema:

```
Shipments(id:integer, customer:integer, isbn:text, ship_date:timestamp)
Editions(isbn:text, title:text, publisher:integer, published:date, ...)
Stock(isbn:text, numInStock:integer, numSold:integer)
Customer(id:integer, name:text, ...)
```

Define a PLpgSQL trigger function `new_shipment()` which is called after each `INSERT` or `UPDATE` operation is performed on the `Shipments` table.

The `new_shipment()` function should check to make sure that each added shipment contains a valid customer ID number and a valid ISBN. It should then update the stock information:

- for an `INSERT`, subtract one from the total amount of stock and add one to the number sold
- for an `UPDATE`, if the change involves the book, then restore the `Stock` entry for the old book and update the `Stock` entry for the new book

It should also calculate a new shipment ID (one higher than the previous highest) and ensure that it is placed in the `shipment_id` field of the new tuple. It should also set the time-stamp for the new tuple to the current time.

Under this scheme, tuples would be inserted into the `Shipments` table as:

```
insert into Shipments(customer,isbn) values (9300035,'0-8053-1755-4');
```

[show answer]

12. Suggest a PostgreSQL `CREATE TABLE` definition that would ensure that all of the effects in the previous question happened automatically.

    [show answer]

13. Consider the following typical components of a PostgreSQL user-defined aggregate definition:

```
CREATE AGGREGATE AggName(BaseType) (
    stype     = ...,
    initcond  = ...,
    sfunc     = ...,
    finalfunc = ...,
);
```

    Explain what each one does, and how they work together to produce the aggregation result.

    [show answer]

14. Imagine that PostgreSQL did not have an `avg()` aggregate to compute the mean of a column of `numeric` values. How could you implement it using a PostgreSQL user-defined aggregation definition (called `mean`)? Assume that it ignores `null` values. If the column is

empty (has no values) return `null`.

[show answer]

15. How could you get the mean of a column of values without having to write your own aggregation operation? Assume a simple table with schema: `R(...,a:integer,...)`.

[show answer]