**Tres Exercicis triggers:**

1. Utilitzant triggers per mantenir regles de negoci.

Suposem que Middlesex Transport Authority (MTA) té una regla que estableix que el salari d’un conductor d’autobus no pot ser modificat per un import que superi el 20% del salari actual.

Crear un trigger que es dispari cada vegada que es produeixi una modificació en la columna salari de la taula d’empleats de manera que si modifiquem el salari d’un conductor que no respecti aquesta regla mostri un missatge d’error a l’usuari.

create or replace function f\_salary\_change\_monitoring()

returns trigger

language plpgsql

as

$$

begin

if ((OLD.salary <> 0 and ((NEW.salary/OLD.salary) >= 1.2)) or

( NEW.salary <>0 and (OLD.salary/NEW.salary) >= 1.2)) then

RAISE EXCEPTION 'Large percentage change in salary prohibited.';

end if;

return NEW;

end;$$;

create trigger tr\_salary\_change\_monitoring

before update on employees

for each row

execute procedure f\_salary\_change\_monitoring();

Update employees

set salary = 3600

where last\_name = 'Hunold';

2. Triggers per prevenir esborrats o modificacions.

A la base de dades Empleats tenim una columna Department\_id que fa referència al departament on treballen els empleats. Tot i aquesta clau forana enacara podriem posar un trigger a la taula Departments per aturar qualsevol canvi en el nom del departament o bé per aturar qualsevol acció d’eliminar una fila de la taula. Tot i que és força irreal imaginem que tenim una regla de negoci que estableix que una vegada creat un departament aquest s’ha de mantenir ‘per sempre’ i no pot canviar el seu nom. El trigger que crearem és un trigger de sentència i actuarà sobre els events DELETE I UPDATE department\_name.

create or replace function f\_dept\_trigger()

returns trigger

language plpgsql as

$$

begin

RAISE EXCEPTION 'You are not allowed to change the value of dname or delete a dept row';

end;$$;

create trigger tr\_dept\_trigger

before update of department\_name or delete

on departments

execute procedure f\_dept\_trigger();

update departments

set department\_name = 'Hola que ase'

where department\_name = 'Marketing';

3. Crear triggers per validació de dades.

Les Check constraint són similars a la validació de dades i és una opció que podem utilitzar en la sentència CREATE TABLE on podem especificar quin tipus de dades podem entrar en una columna en particular. Imaginem que tenim una regla de negoci que ens diu que els salaris no poden ser negatius o zero, ni majors de 30000€. Podriem crear una check constraint com aquesta:

*check (salary >0 and salary <30000)*

Així qualsevol salari que sigui més petit o igual a 0 o més gran de 30000 causarà una violació de la constraint. Aplicant aquesta constraint, no obstant, no podrem saber si el salari era més gran de 30000€ o bé era negatiu o zero.

Podem crear un trigger en comptes de la check constraint, que ens digui com s’ha infringit la restricció sobre aquesta column. D’aquesta manera si el salari està fora del rang (0 – 30000], el trigger generarà un missatge d’error informant a l’usuari si el salari és major de 30000 o bé és negatiu.

Aquest serà un trigger que haurà d’actuar a nivell de fila i només per a les operacions Insert i Update de la columna salary.

create or replace function f\_verifica\_salary\_trigger()

returns trigger language plpgsql

as

$body$

begin

if NEW.salary < 0 then

RAISE EXCEPTION 'The salary cannot be negative.';

end if;

if NEW.salary > 5000 then

RAISE EXCEPTION 'The salary cannot exceed 30000.';

end if;

end; $body$;

create trigger tr\_verifica\_salary\_trigger

before update of salary or insert

on employees

for each row

when((NEW.salary < 0) or (NEW.salary > 30000))

execute procedure f\_verifica\_salary\_trigger();

UPDATE employees

SET salary = -9000

WHERE employee\_id = '114';