Procediments emmagatzemats i disparadors, per què són necessaris?

NOM I COGNOMS: JOSEP ANDREU MIRALLES

EXERCICI 1 (20%)

1: Afegim una restricció en la columna birt_dt per què no accepti valors nuls.

```
-- Apartat 1. Hem tindrem prou afegint una restricció en la columna birt_dt per què no accepti valors nuls.

ALTER TABLE clinical.tb_patient ALTER COLUMN birth_dt SET NOT NULL;
```

En la següent captura s'observa com s'ha volgut inserir un nou registre amb un valor NULL en el camp birt_dt, i com hem obtingut una resposta d'error degut a la restricció introduïda.

```
INSERT INTO clinical.tb_patient(patient_id, ehr_number, name, sex, birth_dt, residence, insurance)
    VALUES(34, 1033, 'Ricard', 'M', NULL, 'Carrer Lateral', 'Muface');
```

```
ERROR:  null value in column "birth_dt" violates not-null constraint
DETAIL:  Failing row contains (34, 1033, Ricard, M, null, Carrer Lateral, Muface).
SQL state: 23502
```

2: Afegim una restricció en la qual indiquem que la data d'arribada ha de ser igual o anterior a la data d'alta.

```
-- Apartat 2. Afegim una restricció en la qual indiquem que la data d'arribada ha de ser igual o anterior a la data d'alta.

ALTER TABLE clinical.tb_encounter
    ADD CONSTRAINT ck_dates_arribada_alta
    CHECK (arrival_dt < discharge_dt);
```

Un cop executat l'script passem a comprovar el seu funcionament inserint un nou registre on la data d'arribada és posterior a la data d'alta, i com podem observar obtenim un missatge d'error degut a la restricció introduïda.

```
INSERT INTO clinical.tb_encounter(encounter_id, patient_id, encounter_type, arrival_dt, discharge_dt, med_service_id)
    VALUES(21989443,9,'Ingreso Programado','2017-02-12 07:32:00','2016-02-14 19:12:00',3);
```

```
ERROR:  new row for relation "tb_encounter" violates check constraint "ck_dates_arribada_alta"
DETAIL:  Failing row contains (21989443, 9, Ingreso Programado, 2017-02-12 07:32:00, 2016-02-14 19:12:00, 3).
SQL state: 23514
```

3: Creem una funció que retorni l'error i un disparador que detecti les actualitzacions dels valors de la columna created_dt.

```
-- Apartat 3. Creem una funció que retorni la reposta i un disparador que detecti les actualitzacions dels valors creted_dt.

    --- Funció disparador

CREATE OR REPLACE FUNCTION no_actualitzacions()
    RETURNS TRIGGER AS
    $BODY$
    BEGIN
    RAISE EXCEPTION 'No s´admeten modificacions del valor created_dt';
END;
$BODY$
LANGUAGE plpgsql

    --- Disparador

CREATE TRIGGER check_actualitzacions
    BEFORE UPDATE OF created_dt ON clinical.tb_orders
    FOR EACH ROW
    EXECUTE PROCEDURE no_actualitzacions();
```

Un cop executada la funció i el disparador provem de modificar el camp creted_dt i veiem com obtenim un missatge d'error on ens indica que no s'admeten modificacions del camp created_dt.

```
UPDATE clinical.tb_orders SET created_dt = '2021-01-02 10:53' WHERE order_id = 442;
```

```
ERROR:  No s´admeten modificacions del valor created_dt
CONTEXT:  PL/pgSQL function no_actualitzacions() line 3 at RAISE
SQL state: P0001
```

EXERCICI 2 (40%)

2a:

```
-- Exercici 2 a)

CREATE or REPLACE FUNCTION catalog_yearly_orders (tria_anys INTEGER, tria_codi INTEGER)
RETURNS INTEGER AS '

DECLARE
    newtable INTEGER :=0;

BEGIN
    IF EXISTS (SELECT * FROM clinical.tb_orders WHERE order_code = tria_codi AND EXTRACT (YEAR FROM created_dt) = tria_anys) THEN
        SELECT COUNT(order_id) INTO newtable
            FROM clinical.tb_orders
            WHERE EXTRACT (YEAR FROM created_dt) = tria_anys AND order_code = tria_codi
        GROUP BY order_code, EXTRACT (YEAR FROM created_dt);
    END IF;
    RETURN newtable;
END;
' LANGUAGE plpgsql;
```

Un cop emmagatzemat el procediment passem a executar-lo, i veiem com obtenim correctament el nombre de prestacions segons l'opció de catàleg i any introduït, també podem observar que en cas que no hi hagi coincidència entre les variables o que els valors introduïts no existeixen en el registre de la taula, el procediment ens retorna un valor 0 enlloc de NULL.

```
SELECT * FROM catalog_yearly_orders (2009, 1454)
```

	catalog_yearly_orders integer
1	4

```
SELECT * FROM catalog_yearly_orders (2017, 2084)
```

	catalog_yearly_orders integer
1	1

```
SELECT * FROM catalog_yearly_orders (2016, 2114)
```

	catalog_yearly_orders integer
1	2

```
SELECT * FROM catalog_yearly_orders (2001, 2114)
```

	catalog_yearly_orders integer
1	0

```
SELECT * FROM catalog_yearly_orders (2021, 3114)
```

	catalog_yearly_orders integer
1	0

2b:

```
-- Exercici 2 b)

CREATE or REPLACE FUNCTION yearly_orders (tria_anys INTEGER)
RETURNS INTEGER AS '

DECLARE
    newtable2 INTEGER :=0;

BEGIN
    IF tria_anys IN (SELECT EXTRACT (YEAR FROM created_dt) FROM clinical.tb_orders) THEN
        SELECT COUNT(order_id) INTO newtable2
            FROM clinical.tb_orders
            WHERE EXTRACT (YEAR FROM created_dt) = tria_anys
        GROUP BY EXTRACT (YEAR FROM created_dt);
    END IF;
    RETURN newtable2;
END;
' LANGUAGE plpgsql;

SELECT * FROM yearly_orders (2004)
SELECT * FROM yearly_orders (2016)
```

Un cop emmagatzemat el procediment passem a executar-lo, i veiem com obtenim correctament el nombre de prestacions segons l'any introduït, també podem observar que en cas que no hi hagi coincidència amb l'any introduït, el procediment ens retorna un valor 0 enlloc de NULL.

```
SELECT * FROM yearly_orders (2015)
```

	yearly_orders integer
1	9

```
SELECT * FROM yearly_orders (2016)
```

	yearly_orders integer
1	279

```
SELECT * FROM yearly_orders (2018)
```

	yearly_orders integer
1	3

```
SELECT * FROM yearly_orders (2004)
```

	yearly_orders integer
1	0

```
SELECT * FROM yearly_orders (2006)
```

	yearly_orders integer
1	0

2c:

EXERCICI 3 (30%)

3a:

```
-- Exercici 3 a)

CREATE FUNCTION nou_status()
RETURNS trigger AS '
BEGIN
    IF (OLD.status = ''Cancelada'' AND NEW.status = ''Realizada'') OR (OLD.status = ''Realizada'' AND NEW.status = ''Cancelada'') THEN
        RAISE EXCEPTION ''No s´admet la modificació del valor "Cancelada" pel de "Realizada"'';
    ELSE RETURN NEW;
    END IF;
RETURN NEW;
END'
LANGUAGE 'plpgsql'

CREATE TRIGGER impedeix_canvis_status
BEFORE UPDATE OF status ON tb_orders
FOR EACH ROW EXECUTE PROCEDURE nou_status();
```

Un cop creat el disparador, el testem, i veiem que els registres on s'ha intentat modificar el ser status de "Cancelado" a "Realizado" o vice versa donen error i com en el cas de la fila amb order_id = 360 on es vol modificar l'status de "Solicitada" a "Realizada" si que accepta i emmagatzema la modificació.

259	358	1454	416315	Cancelada	2009-06-02 09:09:00	2009-06-02 09:09:00	5
260	359	1454	750284	Cancelada	2009-09-16 13:05:00	2009-09-16 13:05:00	5
261	360	1454	691529	Solicitada	2009-08-28 10:42:00	2009-09-08 11:36:00	5
262	361	2216	952783	Cancelada	2009-11-13 11:59:00	2009-11-13 11:59:00	5
263	362	1454	1087486	Cancelada	2009-12-23 10:49:00	2009-12-23 10:49:00	5
264	363	2144	952783	Realizada	2009-11-13 11:59:00	2009-11-13 11:59:00	6
265	364	2216	11687014	Realizada	2015-08-28 10:11:00	2015-08-28 11:48:00	6
266	365	2216	11466315	Realizada	2016-01-08 11:15:00	2016-01-09 09:57:00	6

Registres de la taula tb_orders abans de la modificació

```
UPDATE clinical.tb_orders SET status = 'Realizada' WHERE order_id = 359;
```

```
ERROR:  No s´admet aquesta modificació del registre en el camp status
CONTEXT:  PL/pgSQL function nou_status() line 4 at RAISE
SQL state: P0001
```

```
UPDATE clinical.tb_orders SET status = 'Cancelada' WHERE order_id = 363;
```

```
ERROR:  No s´admet aquesta modificació del registre en el camp status
CONTEXT:  PL/pgSQL function nou_status() line 4 at RAISE
SQL state: P0001
```

```
UPDATE clinical.tb_orders SET status = 'Realizada' WHERE order_id = 360;
```

259	358	1454	416315	Cancelada	2009-06-02 09:09:00	2009-06-02 09:09:00	5
260	359	1454	750284	Cancelada	2009-09-16 13:05:00	2009-09-16 13:05:00	5
261	360	1454	691529	Realizada	2009-08-28 10:42:00	2009-09-08 11:36:00	5
262	361	2216	952783	Cancelada	2009-11-13 11:59:00	2009-11-13 11:59:00	5
263	362	1454	1087486	Cancelada	2009-12-23 10:49:00	2009-12-23 10:49:00	5
264	363	2144	952783	Realizada	2009-11-13 11:59:00	2009-11-13 11:59:00	6
265	364	2216	11687014	Realizada	2015-08-28 10:11:00	2015-08-28 11:48:00	6
266	365	2216	11466315	Realizada	2016-01-08 11:15:00	2016-01-09 09:57:00	6

Registres de la taula tb_orders després de la modificació

3b:

```
-- Exercici 3 b)

CREATE TABLE tb_orders_status_changelog (
    order_id INTEGER NOT NULL,
    old_status VARCHAR(50) NOT NULL,
    new_status VARCHAR(50) NOT NULL,
    changelog_dt TIMESTAMP NOT NULL,
    CONSTRAINT fk_tb_orders_status_changelog FOREIGN KEY(order_id) REFERENCES tb_orders(order_id)
);
```

Un cop executat l'script passem a comprovar si la nova taula tb_orders_status_changelog s'ha creat correctament.

order_id integer	old_status character varying (50)	new_status character varying (50)	changelog_dt timestamp without time zone

3c:

```
-- Exercici 3 c)

CREATE FUNCTION insert_canvi_status() RETURNS trigger AS '
BEGIN
    INSERT INTO clinical.tb_orders_status_changelog
        VALUES (OLD.order_id, OLD.status, NEW.status, current_timestamp);
RETURN NULL;
END;
' LANGUAGE plpgsql;

CREATE TRIGGER detecta_canvis_status
AFTER UPDATE OF status ON tb_orders
FOR EACH ROW EXECUTE PROCEDURE insert_canvi_status();
```

Un cop creat el disparador, el testem, i veiem com en realitzar modificacions en el camp status de diferents files aquests queden emmagatzemats com a noves files en la nova taula tb_orders_status_changelog. També observem que si la modificació és realitza en altres camps (status_dt o order_code) les modificacions no creen nous registres en la nova taula tb_orders_status_changelog.

1	100	2084	458151	Solicitada	2009-06-16 09:12:00	2011-06-08 14:08:00	1
2	101	2216	458151	Cancelada	2009-06-16 09:12:00	2011-06-08 14:08:00	2
3	102	2144	458151	Solicitada	2009-06-16 09:12:00	2011-06-08 14:08:00	3
4	103	4459	1017014	Solicitada	2009-12-03 09:50:00	2013-02-06 14:35:00	4
5	104	2216	580497	Solicitada	2010-02-24 12:21:00	2013-03-18 03:37:00	5
6	105	2216	580497	Solicitada	2010-03-27 23:54:00	2019-02-28 02:00:00	6
7	106	2216	580497	Solicitada	2010-03-22 12:43:00	2011-06-16 10:11:00	7
8	107	2144	580497	Solicitada	2010-02-24 12:21:00	2013-03-18 03:37:00	8
9	108	2216	580497	Realizada	2010-03-04 08:55:00	2010-03-04 08:55:00	8

Registres de la taula tb_orders abans de la modificació

```
UPDATE clinical.tb_orders SET status = 'Realizada' WHERE order_id = 100;
UPDATE clinical.tb_orders SET status_dt = '2021-03-08 11:53' WHERE order_id = 101;
UPDATE clinical.tb_orders SET status = 'Realizada' WHERE order_id = 102;
UPDATE clinical.tb_orders SET order_code = 2216 WHERE order_id = 103;
UPDATE clinical.tb_orders SET status = 'Cancelada' WHERE order_id = 104;
```

	order_id integer	old_status character varying (50)	new_status character varying (50)	changelog_dt timestamp without time zone
1	100	Solicitada	Realizada	2021-05-09 16:04:03.881685
2	102	Solicitada	Realizada	2021-05-09 16:04:16.800355
3	104	Solicitada	Cancelada	2021-05-09 16:04:23.447481

Nous registres creats en la nova taula tb_orders_status_changelog després de les modificacions

EXERCICI 4 (10%)

Cada funció té una classificació de volativilitat, per defecte, si no especifiquem el contrari, a Postgresql totes les funcions són declarades com a **"Volatile"**. **"Volatile"** indica que el valor de la funció pot canviar fins i tot en una sola simple exploració d'una taula, de manera que no es poden fer optimitzacions. Relativament poques funcions de les base de dades són volàtils en aquest sentit; alguns exemples són **"random ()", "currval ()", "timeofday ()"**. Hem de tenir en compte que qualsevol funció que tingui efectes secundaris s'ha de classificar com a volàtil, encara que el resultat sigui previsible, per evitar que les consultes no s'optimitzin, un exemple d'això és "setval ()".

Les funcions del tipus **"Volatile"** poden fer qualsevol cosa, fins i tot modificar la base de dades, i poden retornar resultats diferents en successives consultes amb els mateixos inputs. En resum una funció **"Volatile"** és:

- Tota funció amb efectes secundaris.
- Tota funció que escriu a la base de dades.
- Qualsevol funció que consulti dades externes no gestionades per "Snapshots" de Postgresql.

En general, es desitjable deixar les funcions com a **"Volatile"** tret que hi hagi una bona raó per no fer-ho.

Altres opcions disponibles per a declarar les funcions són **"Immutable"** i **"Stable"**. Els tres serveixen per donar informació a l'optimizador de consultes, de manera que aquest sàpiga si el resultat retornat per la funció serà constant i així aquest pugui prendre les decisions oportunes.

"Immutable" indica que la funció no pot modificar la base de dades i sempre retorna el mateix resultat quan es donen els mateixos inputs; és a dir, no fa cerques de bases de dades ni utilitza informació que no estigui directament present a la seva llista de variables. Si triem aquesta opció de volativilitat, qualsevol execució de la funció amb totes les variables es pot substituir pel valor de la funció.

Una funció **"Immutable"** ha de ser una funció pura , els resultats de la qual depenen només dels seus inputs. Aquest és un requisit molt estricte, ja que no pot cridar a altres funcions no **"Immutable"**, tampoc no pot accedir a taules, ni pot accedir als valor de les propietats de la configuració.

I finalment **"Stable"** indica que la funció no pot modificar la base de dades i que en una simple consulta de la taula, retornarà constantment el mateix resultat per les mateixes variables, però que el seu resultat podria canviar entre les sentències SQL. Aquesta és la selecció adequada per a les funcions els resultats dels quals depenen de les cerques en bases de dades, de paràmetres variables (com ara la zona horària actual), etc... Es desaconsella el seu ús en "triggers" del tipus AFTER que hagin de consultar modificacions de files per la comanda actual. També s'ha de tenir

en compte que la família de funcions del tipus **"statement_timestamp"** estan qualificades com **"Stable"**, ja que els seus valors no canvien dins d'una transacció.

Una funció **"Stable"** pot utilitzar qualsevol entrada que sigui "Stable": com són altres funcions "Stable" o bé consultes de taules. És segur fer consultes de taules perquè la vista de la funció d'aquestes taules no canviarà amb l'"Snapshot" actual de la consulta.

Criteris de valoració

En l'enunciat s'indica el pes/valoració de cada exercici.

Per aconseguir la puntuació màxima en els exercicis, cal explicar amb claredat la solució que es proposa.

Format i data de lliurament

El format de l'arxiu que conté la vostra solució pot **ser .pdf, .doc i .docx**. **Per a altres opcions, si us plau, contacteu prèviament amb el vostre consultor.** El nom de l'arxiu ha de contenir el codi de l'assignatura, el vostre cognom i el vostre nom, així com el nombre d'activitat (PAC3).

El fitxer .zip que contingui tots els fitxers de la PAC (tant els fitxers .sql com el document que mostra els resultats de les vostres solucions) l'heu d'enviar a la bústia de Lliurament i registre d'AC disponible a l'aula (apartat Avaluació).

La data límit per lliurar la PAC3 és el **09/05/2021**.

Nota: **Propietat intel·lectual**

Al presentar una pràctica o PAC que faci ús de recursos aliens, s'ha de presentar juntament amb ella un document en el qual es detallin tots ells, especificant el nom de cada recurs, el seu autor, el lloc on es va obtenir i el seu estatus legal: si l'obra està protegida pel copyright o s'acull a cap altra llicència d'ús (*Creative Commons*, llicència GNU, GPL etc.).

L'estudiant s'haurà d'assegurar que la llicència que sigui no impedeixi específicament el seu ús en el marc de la pràctica o PAC. En el cas de no trobar la informació corresponent haurà d'assumir que l'obra està protegida pel copyright. A més, serà necessari adjuntar els fitxers originals quan les obres utilitzades siguin digitals, i el seu codi font si així correspon.