

Projecte de base de dades

ClashSayale 2021-2022 – Fase 4

Llistat de membres (nom i correu):

Irina Aynés (irina.aynes@students.salle.url.edu)

Marc Geremias (marc.geremias@students.salle.url.edu)

Marc Valsells (marc.valsells@salle.url.edu)

Albert Tomàs (albert.tomas@students.salle.url.edu)

Data d'entrega: 27.05.2022

Taula de continguts

Introducció	4
Actualització del model entitat-relació i del model relacional	5
Actualització del model físic	6
Les cartes són la guerra, disfressada d'esport	7
TRIGGER 1	7
Solució	7
Explicació	9
Validació	9
Trigger 2	10
Solució	10
Explicació	10
Validació	10
Trigger 3	11
Solució	11
Explicació	12
Validació	12
No sóc un jugador, sóc un jugador de videojocs	13
Trigger 1	13
Solució	13
Explicació	14
Validació	14
Trigger 2	18
Solució	18
Explicació	19
Validació	20
Trigger 3	22
Solució	22
Explicació	23
Validació	24
Tingueu valor. Encara tenim el nostre clan. Sempre hi ha esperança	26
Funció f_selNewLeader	26
Solució	26
Explicació	26
Validació	27
Trigger 1	27
Solució	27
Explicació	28
Validació	28
Trigger 2	29

Solució	29
Explicació	30
Validació	30
Trigger 3	31
Solució	31
Explicació	31
Validació	31
M'agrada la competició. M'agraden els reptes ...	32
Trigger 1	32
Solució	32
Explicació	33
Validació	33
Trigger 2	35
Solució	35
Explicació	35
Validació	36
Trigger 3	37
Solució	37
Explicació	38
Validació	38
Conclusions	40
RECURSOS EMPRATS	40
Lliçons apreses i conclusions	41

1 Introducció

Arribem a final de curs i entreguem la memòria final de la fase 4 del projecte de ClashSayale. Després de tres fases on hem tocat diferents àmbits de programació de bases de dades, desde models conceptuals i relacionals, fins a models físics i inserció de dades junt amb consultes variades, arribem a la fase 4 i final del projecte, on l'objectiu és implementar una sèrie de triggers (disparadors) per demostrar el nostre coneixement sobre aquests.

Partint de la nostra base de dades ja implementada a les fases 1 i 2, la qual vam posar a prova durant la fase 3 amb les consultes i subconsultes, aquesta fase 4 ha servit per acabar de comprovar que la nostra base és òptima i està ben implementada.

L'objectiu d'aquesta fase, era demostrar les nostres habilitats amb triggers i stored procedures (si s'han utilitzat en alguns casos), després del temari vist a classe i les activitats d'avaluació contínua.

La fase està formada per 12 consultes en les que es demana implementar uns determinats triggers que es disparin quan es realitzi una acció determinada a la nostra base de dades, de manera que es pugui controlar errors i actualitzar dades de les nostres taules quan és necessari.

D'aquestes 12 consultes, s'han repartit tres per a cada membre del grup, de manera que tots els integrants treballem de forma equitativa. A més, també s'ha procurat repartir-nos cada mòdul aleatòriament per tal de conèixer el màxim possible totes les parts de la nostra base de dades i que tots s'hagin familiaritzat amb les taules i relacions diferents.

A continuació, s'exposa qui ha realitzat cada mòdul de la fase 4:

El mòdul 1 (Les cartes són la guerra, disfressada d'esport) l'ha realitzat Marc Geremias.

El mòdul 2 (No sóc un jugador, sóc un jugador de videojocs) l'ha realitzat Irina Aynés.

El mòdul 3 (Tingueu valor. Encara tenim el nostre clan. Sempre hi ha esperança) l'ha realitzat Marc Valsells.

El mòdul 4 (M'agrada la competició. M'agraden els reptes...) l'ha realitzat Albert Tomàs.

Dit això, ara comencen les explicacions dels (petits) canvis que hi ha hagut al model entitat-relació, model relacional i model físic, que com es veurà no són molt extensos ja que en anteriors fases vam acabar de perfeccionar aquests models i ara ja quedava petits detalls per arreglar.

Després començaran les explicacions dels triggers així com un breu resum sobre què tracta cada un d'ells i una sèrie de validacions on es demostra que aquests triggers funcionen de manera correcta.

2 Actualització del model entitat-relació i del model relacional

En aquesta última fase de la pràctica no ens ha fet falta realitzar cap modificació en el model entitat-relació ni en el model relacional, ja que hem utilitzat taules externes al model per guardar registres dels diferents triggers de la fase.

Així doncs, aquests registres no necessitaven cap referencia FK de les taules inicials només una copia d'informació sense relació.

3 Actualització del model físic

Per aquesta fase 4 s'han realitzat pocs canvis en el model físic. Seguidament s'expliquen aquests canvis:

S'ha creat una nova taula anomenada WARNINGS per poder-la utilitzar en totes les parts d'aquesta fase 4. Ja que molts enunciats d'aquesta fase demanaven insertar dades en aquesta taula per mostrar errors. Aquesta taula, conté la taula on hi ha l'error, el missatge d'error, la data d'aquest error i l'usuari que ha causat l'error.

```
CREATE TABLE Warnings (  
    affected_table VARCHAR(255),  
    error_message VARCHAR(255),  
    date DATE,  
    username VARCHAR(255)  
);
```

A més, per a la realització del trigger 3 del set 1 també s'ha creat la taula OPCardBlackList, aquesta taula s'utilitza per guardar les cartes que hagin guanyat més del 90% de les batalles on s'utilitzen. Finalment en aquesta taula afegim la data per així comprovar si anteriorment aquesta carta ja va ser col·locada a aquesta taula i en cas de ser així s'hauran de baixar tots els seus valors un 1%.

```
CREATE TABLE OPCardBlackList (  
    nom VARCHAR(255),  
    date DATE  
);
```

4 Les cartes són la guerra, disfressada d'esport

4.1 Trigger 1

4.1.1 Solució

```
CREATE OR REPLACE FUNCTION f_proporcionsRares()
RETURNS trigger AS $$
DECLARE
    n_total INT;
    n_common INT;
    n_rare INT;
    n_epic INT;
    nLegendary INT;
    n_champion INT;
BEGIN
    SELECT COUNT(c.nom) INTO n_total
    FROM carta AS c;

    SELECT COUNT(c.nom)*100 INTO n_common
    FROM carta AS c
    WHERE c.raresa LIKE 'Common';
    SELECT COUNT(c.nom)*100 INTO n_rare
    FROM carta AS c
    WHERE c.raresa LIKE 'Rare';
    SELECT COUNT(c.nom)*100 INTO n_epic
    FROM carta AS c
    WHERE c.raresa LIKE 'Epic';
    SELECT COUNT(c.nom)*100 INTO nLegendary
    FROM carta AS c
    WHERE c.raresa LIKE 'Legendary';
    SELECT COUNT(c.nom)*100 INTO n_champion
    FROM carta AS c
    WHERE c.raresa LIKE 'Champion';

    IF (n_common / n_total) != 31 THEN
        INSERT INTO warnings(affected_table, error_message, date, username)
        VALUES('Cartes',
            'Proporcions de raresa no respectades:' ||
            ' Common ' ||
            'la proporció actual és ' ||
            (n_common / n_total) ||
            ' quan hauria de ser ' ||
            '31',
            CURRENT_DATE,
            CURRENT_USER);
    END IF;

    IF (n_rare / n_total) != 26 THEN
        INSERT INTO warnings(affected_table, error_message, date, username)
        VALUES('Cartes',
            'Proporcions de raresa no respectades:' ||
```

```

        ' Rare ' ||
        'la proporció actual és ' ||
        (n_rare / n_total) ||
        ' quan hauria de ser ' ||
        '26',
        CURRENT_DATE,
        CURRENT_USER);
END IF;

IF (n_epic / n_total) != 23 THEN
    INSERT INTO warnings(affected_table, error_message, date, username)
    VALUES('Cartes',
        'Proporcions de raresa no respectades:' ||
        ' Epic ' ||
        'la proporció actual és ' ||
        (n_epic / n_total) ||
        ' quan hauria de ser ' ||
        '23',
        CURRENT_DATE,
        CURRENT_USER);
END IF;

IF (n_legendary / n_total) != 17 THEN
    INSERT INTO warnings(affected_table, error_message, date, username)
    VALUES('Cartes',
        'Proporcions de raresa no respectades:' ||
        ' Legendary ' ||
        'la proporció actual és ' ||
        (n_legendary / n_total) ||
        ' quan hauria de ser ' ||
        '17',
        CURRENT_DATE,
        CURRENT_USER);
END IF;

IF (n_champion / n_total) != 3 THEN
    INSERT INTO warnings(affected_table, error_message, date, username)
    VALUES('Cartes',
        'Proporcions de raresa no respectades:' ||
        ' Champion ' ||
        'la proporció actual és ' ||
        (n_champion / n_total) ||
        ' quan hauria de ser ' ||
        '3',
        CURRENT_DATE,
        CURRENT_USER);
END IF;

RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```



```
DROP TRIGGER IF EXISTS proporcionsRares ON carta;
CREATE TRIGGER proporcionsRares
  AFTER INSERT OR DELETE OR UPDATE ON carta
  EXECUTE FUNCTION f_proporcionsRares();
```

4.1.2 Explicació

Aquest primer trigger es dispara quan es realitza alguna modificació a la taula Carta, es a dir que s'executa en cas de fer un INSERT, DELETE o UPDATE.

Un cop es dispara calcula el percentatge de cartes de cada raresa i en cas que sigui diferent a l'especificat es fa un INSERT a la taula Warnings on s'apunta la taula Cartes, la raresa que no s'està respectant, el percentatge erroni i el percentatge que hauria de ser.

4.1.3 Validació

Per comprovar que el trigger està funcionant correctament es van realitzar algunes queries per modificar la taula de cartes. La primera va consistir en fer el següent INSERT:

```
INSERT INTO carta(nom, dany, velocitat_atac, raresa, arena)
VALUES ('xdxd', 12, 12, 'Champion', 54000000);
```

Un cop executat, es va fer un SELECT * FROM de la taula de Warning i el resultat va ser el següent:

	affected_table	error_message	date	username
1	Cartes	Proporcions de raresa no respectades: Common la proporció actual és 26 quan hauria de ser 31	2022-05-25	postgres
2	Cartes	Proporcions de raresa no respectades: Epic la proporció actual és 27 quan hauria de ser 23	2022-05-25	postgres
3	Cartes	Proporcions de raresa no respectades: Champion la proporció actual és 4 quan hauria de ser 3	2022-05-25	postgres

Com es pot comprovar a la imatge només es respecta el percentatge de raritat de Rare i Legendary, la resta al no complir els percentatges s'afegeixen a la taula Warnings amb la proporció actual, la data i l'usuari que ha fet la inserció.

4.2 Trigger 2

4.2.1 Solució

```
CREATE OR REPLACE FUNCTION f_setMaxlevel()
RETURNS trigger AS $$
BEGIN

    UPDATE pertany
    SET nivell = (SELECT nc.nivell
                  FROM nivellcarta AS nc
                  ORDER BY nc.nivell DESC
                  LIMIT 1)
    WHERE pertany.id_pertany = NEW.id_pertany;

RETURN NULL;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS triggerMaxlevel ON pertany;
CREATE TRIGGER triggerMaxlevel AFTER INSERT ON pertany
FOR EACH ROW
EXECUTE FUNCTION f_setMaxlevel();
```

4.2.2 Explicació

En aquest segon trigger s'ha fet que s'acció en cas que l'usuari aconseguixi una nova carta, de manera que cada vegada que es faci un INSERT a la taula pertany es dispararà. Aquest trigger simplement posarà al nivell màxim la nova carta que l'usuari hagi aconseguit, Per saber el nivell màxim actual es fa una subquery on es mira quin és el nivell més alt existent de la taula nivellcarta.

4.2.3 Validació

Per fer la validació primer es va fer una consulta per realitzar una inserció a la taula pertany per així afegir una nova carta a un jugador, de manera que es va executar l'ho següent.

```
INSERT INTO pertany(quantitat, data_desbolqueig, id_pertany, tag_jugador, nom_carta, nivell)
VALUES (637, '2021-04-04', 91943, '#QV2PYL', 'Battle Ram', 11);
```

Un cop executat l'insert es va realitzar la següent consulta per a veure la informació de la taula pertany on la id_pertany sigues el de la nova inserció.

```
SELECT *
FROM pertany
WHERE id_pertany = 91943;
```

	quantitat	data_desbolqueig	id_pertany	tag_jugador	nom_carta	nivell
1	637	2021-04-04	91943	#QV2PYL	Battle Ram	14

Com es pot veure, la inserció a la taula pertany es fa correctament però amb la modificació del trigger que modifica el nivell màxim.

4.3 Trigger 3

4.3.1 Solució

```
DROP TABLE IF EXISTS OPCardBlackList;
```

```
CREATE TABLE OPCardBlackList (  
    nom VARCHAR(255),  
    date DATE  
);
```

```
CREATE OR REPLACE FUNCTION f_targetesOp()  
RETURNS trigger AS $$  
BEGIN
```

```
    INSERT INTO OPCardBlackList(nom, date)  
    SELECT c.nom, CURRENT_DATE
```

```
    FROM carta AS c  
    JOIN formen AS f ON c.nom = f.nom_carta  
    JOIN pila AS p ON f.id_pila = p.id_pila  
    JOIN guanya AS g ON p.id_pila = g.id_pila
```

```
    GROUP BY c.nom  
    HAVING COUNT(id_batalla)*100 / (COUNT(id_batalla) + (SELECT COUNT(id_batalla)  
        FROM carta AS c2  
        JOIN formen AS f ON c2.nom = f.nom_carta  
        JOIN pila AS p ON f.id_pila = p.id_pila  
        JOIN perd ON p.id_pila = perd.id_pila  
        WHERE c.nom = c2.nom  
        GROUP BY c2.nom)) > 90;
```

```
    UPDATE carta  
    SET dany = dany * 0.99  
    WHERE nom IN (SELECT op.nom FROM OPCardBlackList AS op WHERE op.date + interval  
'7 day' < now());
```

```
    RETURN NULL;  
END;  
$$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS targetesOp ON batalla;  
CREATE TRIGGER targetesOp AFTER INSERT ON batalla  
FOR EACH ROW  
EXECUTE FUNCTION f_targetesOp();
```

4.3.2 Explicació

Aquest tercer trigger és l'encarregat de comprovar que les cartes que hi ha dins el joc estan balancejades. De manera que a cada insert a la taula batalles es fa la comprovació de que les cartes segueixin balancejades. Per fer-ho, primer es calcula el nombre de batalles guanyades i nombre de batalles perdudes que te cada carta i un cop obtingut el resultat es calcula el win rate de la següent manera:

$$\text{manera: } \frac{nPartidesGuanyades}{(nPartidesGuanyades + nPartidesPerdudes)}$$

En cas que el win rate sigui superior al 90% aquesta carta s'afegeix a la taula OPCardBlackList. A més també es comprova si la carta ha estat afegida a la taula OPCardBlackList fa menys d'una setmana, en cas de ser així es baixen tots els seus atributs un 1%.

4.3.3 Validació

Per realitzar la comprovació del trigger primer es van executar les subqueries que tenia dins per així saber quin era el winrate abans de les noves insercions, de manera que en la següent imatge es pot comprovar el winrate inicial.

10	Battle Ram	49	2022-05-27
11	Bomb Tower	49	2022-05-27
12	Bomber	48	2022-05-27
13	Bowler	51	2022-05-27
14	Cannon	50	2022-05-27
15	Cannon Cart	52	2022-05-27
16	Clone	49	2022-05-27
17	Dark Prince	48	2022-05-27
18	Dart Goblin	49	2022-05-27

Com es pot veure, la carta Clone comença amb un winrate de 49%. Un cop sabent el winrate actual es van fer diferents insercions a les taules batalles i guanyades per així comprovar que el win rate pujava. Un cop aquest winrate excedia el 90% el trigger s'executava i s'inseria una nova fila a la taula OPCardBlackList junt amb la data actual.

A més, per a veure si la carta ja havia estat afegida a la taula OPBlackList s'afegia de manera "hardcored" una fila a la taula i així es podia comprovar si es tenia en compte que ja hagués estat afegida fa una setmana o no.

5 No sóc un jugador, sóc un jugador de videojocs

5.1 Trigger 1

5.1.1 Solució

DROP FUNCTION if exists *suma_or_gemmes* CASCADE;

CREATE OR REPLACE FUNCTION *suma_or_gemmes*()

RETURNS trigger as \$\$

BEGIN

IF (SELECT id_bundle from bundle where *new.id_article* = id_bundle) = *new.id_article* then

UPDATE jugador

SET or_ = (CASE WHEN jugador.or_ IS NULL THEN 0 ELSE jugador.or_ END) +

(SELECT (CASE WHEN b.or_ IS NULL

THEN 0 ELSE b.or_ END) from bundle as b

WHERE *new.id_article* = b.id_bundle)

WHERE *new.tag_jugador* = jugador.tag_jugador;

UPDATE jugador

SET gemmes = (CASE WHEN jugador.gemmes IS NULL THEN 0 ELSE

jugador.gemmes END) +

(SELECT (CASE WHEN b.gemmes IS NULL THEN 0 ELSE b.gemmes END)

from bundle as b where *new.id_article* = b.id_bundle)

WHERE *new.tag_jugador* = jugador.tag_jugador;

end if;

IF (SELECT id_arena_pack from arena_pack where id_arena_pack = *new.id_article*) =
new.id_article then

UPDATE jugador

SET or_ = (CASE WHEN or_ IS NULL THEN 0 ELSE or_ END) + (SELECT (CASE

WHEN apa.or_ IS NULL THEN 0 ELSE apa.or_ END) from arena_pack_arena as apa join

arena a on apa.id_arena = a.id_arena

join arena_pack ap on apa.id_arena_pack = ap.id_arena_pack

join article a2 on ap.id_arena_pack = a2.id_article

join compren c on a2.id_article = c.id_article

join jugador j2 on c.tag_jugador = j2.tag_jugador

where *new.id_article* = ap.id_arena_pack and j2.tag_jugador = *new.tag_jugador* and

(SELECT(SUM(g.num_trofeus) + SUM(p.num_trofeus))

from jugador as j

join guanya g on j.tag_jugador = g.tag_jugador

join perd p on j.tag_jugador = p.tag_jugador

WHERE j.tag_jugador = *new.tag_jugador*

GROUP BY j.tag_jugador) >= a.nombre_min and (SELECT(SUM(g.num_trofeus) +
SUM(p.num_trofeus))

from jugador as j

join guanya g on j.tag_jugador = g.tag_jugador

join perd p on j.tag_jugador = p.tag_jugador

WHERE j.tag_jugador = *new.tag_jugador*

GROUP BY j.tag_jugador) <= a.nombre_max

and (a.titol LIKE '%Arena_L10 - Ultimate Champion%'

OR a.nombre_max < 32767)

```

        GROUP BY j2.tag_jugador, apa.or_)
    WHERE tag_jugador = new.tag_jugador;
end if;

RETURN NULL;

END
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS suma_trig ON compren;
CREATE TRIGGER suma_trig AFTER INSERT ON compren
FOR EACH ROW
EXECUTE FUNCTION suma_or_gemmes();

```

5.1.2 Explicació

Aquest trigger consisteix en actualitzar les dades de la compra. És a dir, s'ha sumar l'or i les gemmes del paquet de bundles i l'or del paquet arena al jugador si n'ha comprat. Primer de tot, s'ha creat una funció per fer els UPDATE a la taula jugador. Per poder distingir entre els articles de la taula bundle i els de l'arena pack s'ha fet un IF comprovant que la id de l'article estigui a un dels dos paquets. El primer UPDATE, actualitzar l'or del jugador que acaba de comprar si aquest ha comprat al paquet bundle. En el segon, segueix la mateixa metodologia que el primer, només canviant gemmes per or. I finalment el tercer, realitza una actualització a la taula jugador per aquells que hagin comprat un article del paquet arena. Per aquest últim update, primer s'ha hagut de sumar els trofeus del jugador (els guanyats i perduts) i mirar a quina arena puntua aquell jugador. Un cop s'ha identificat l'arena aleshores amb els joins s'ha connectat les taules arena, arena_pack_arena i arena_pack i s'ha agafat l'or i sumat al jugador.

5.1.3 Validació

La validació d'aquest trigger s'ha fet en 2 parts, una per cada paquet d'article.

La primera part consisteix en comprovar el paquet bundle, s'ha buscat primer un jugador aleatori i també s'ha buscat una id aleatoria de la taula bundle. Seguidament s'ha fet un INSERT a la taula compren amb aquestes dades aleatòries:

- S'ha agafat com a jugador aleatori el : #QV2PYL
- S'ha agafat com a id_bundle aleatoria : 9

	id_bundle	or_	gemmes
1	9	8288	117

```

INSERT INTO compren (tag_jugador, num_targeta, id_article, data_, descompte)
VALUES ('#QV2PYL', '0626997669324072', 9, now(), 0);

```

Per mostrar el jugador amb l'or i les gemmes s'ha fet la següent consulta:

```

SELECT tag_jugador, jugador.or_, jugador.gemmes from jugador WHERE tag_jugador like '#QV2PYL';

```

Seguidament podem comprovar el funcionament del trigger:

Abans del trigger:

	tag_jugador	or_	gemmes
1	#QV2PYL	6442	18

Després del trigger:

	tag_jugador	or_	gemmes
1	#QV2PYL	14730	135

Podem observar com l'or i les gemmes del jugador s'han actualitzat correctament i s'han sumat amb la quantitat del bundle.

I la segona part es comprovar que l'or del paquet arena s'afegeix correctament al jugador, s'ha buscat un jugador aleatori, aleshores a partir d'aquest s'ha fet una consulta semblant al trigger, on s'ha mirat quin número de trofeus tenia i així veure a quina arena pertany.

- S'ha agafat com a jugador aleatori el : #QV2PYL

Consulta:

```
SELECT(SUM(g.num_trofeus) + SUM(p.num_trofeus) ) as trofeus
from jugador as j join guanya g on j.tag_jugador = g.tag_jugador
join perd p on j.tag_jugador = p.tag_jugador
WHERE j.tag_jugador like '#QV2PYL'
GROUP BY j.tag_jugador
```

Resultat:

	trofeus
1	328

Un cop trobat els trofeus s'ha buscat a la taula arena a quina pertany:

```
SELECT *
from arena;
```

Resultat:

	id_arena	titol	nombre_min	nombre_max
1	54000000	TrainingCamp - Training Camp	0	32767
2	54000001	Arena1 - Goblin Stadium	0	299
3	54000021	ArenaPvE - Training Camp	0	32767
4	54000022	ArenaTvE - Training Camp	0	32767
5	54000023	Arena_TouchdownTest - Touchdown Arena	0	32767
6	54000025	Arena_KingOfTheHillTest - King of the Hill Arena Info	0	32767
7	54000026	Arena_KingOfTheHillTest2 - King of the Hill Arena Info	0	32767
8	54000032	Arena_Obstacles - Touchdown Arena	0	32767
9	54000002	Arena2 - Bone Pit	300	599
10	54000003	Arena3 - Barbarian Bowl	600	999
11	54000004	Arena4 - P.E.K.K.A's Playhouse	1000	1299

Observem que l'arena del jugador es la 54000002, per tant ara busquem quines id article correspon a aquesta arena:

```
SELECT *
  from arena_pack_arena
  where id_arena = 54000002;
```

	id_arena	id_arena_pack	or_
1	54000002	86	6201
2	54000002	40	3688
3	54000002	36	1128
4	54000002	60	2804

- Com a mostra s'ha agafat la id d'arena pack 60.

Posteriorment, s'ha fet un INSERT INTO a la taula compren afegint el jugador seleccionat amb la seva targeta de credit i la id de l'article agafat com a mostra anteriorment.

```
INSERT INTO compren (tag_jugador, num_targeta, id_article, data_, descompte)
VALUES ('#QV2PYL', '0626966543536722', 60, now(), 0);
```

Finalment es pot comprovar com el trigger retorna la suma de l'or que tenia el jugador més l'or que ha comprat.

Abans del trigger:

	tag_jugador	or_	gemmes
1	#QV2PYL	10638	562

Després del trigger:

	tag_jugador	or_	gemmes
1	#QV2PYL	19542	562

Per comprovar que aquests resultats son correctes s'ha mirat que l'or del jugador corresponent a l'arena segons els trofeus que té sigui la suma del trigger abans més aquest resultat. Per validar això s'ha fet la següent consulta:

```
SELECT apa.or_ from arena_pack_arena as apa join
  arena a on apa.id_arena = a.id_arena join arena_pack ap on apa.id_arena_pack =
ap.id_arena_pack
  join article a2 on ap.id_arena_pack = a2.id_article join compren c on a2.id_article = c.id_article
  join jugador j2 on c.tag_jugador = j2.tag_jugador
  where a2.id_article = 60 and j2.tag_jugador like '#QV2PYL' and (SELECT(SUM(g.num_trofeus)
+ SUM(p.num_trofeus) )
  as trofeus from jugador as j join guanya g on j.tag_jugador = g.tag_jugador
  join perd p on j.tag_jugador = p.tag_jugador
  WHERE j.tag_jugador like '#QV2PYL'
  GROUP BY j.tag_jugador) >= a.nombre_min and (SELECT(SUM(g.num_trofeus) +
SUM(p.num_trofeus))
  from jugador as j join guanya g on j.tag_jugador = g.tag_jugador
  join perd p on j.tag_jugador = p.tag_jugador
  WHERE j.tag_jugador like '#QV2PYL'
  GROUP BY j.tag_jugador) <= a.nombre_max
  and (a.titol LIKE '%Arena_L10 - Ultimate Champion%'
  OR a.nombre_max < 32767 );
```

Resultat:

	or_
1	8904

Per acabar, podem observar com el trigger ens ha retornat correctament la suma de l'or.

5.2 Trigger 2

5.2.1 Solució

```
DROP TABLE IF EXISTS missatges_prohibits;
CREATE TABLE missatges_prohibits(
  id_paraula SERIAL,
  paraula varchar(255),
  PRIMARY KEY (id_paraula)
);

INSERT INTO missatges_prohibits (paraula)
VALUES ('stupid'), ('silly'),('idiot');

DROP FUNCTION if exists missatges_ofensius CASCADE;
CREATE OR REPLACE FUNCTION missatges_ofensius()
RETURNS trigger as $$
  DECLARE i INTEGER:= 1;
  DECLARE trobat boolean:= false;
BEGIN
  WHILE i <= (SELECT COUNT(paraula) from missatges_prohibits) AND trobat = FALSE
  LOOP
    IF (SELECT COUNT(m.cos) from missatge as m WHERE new.id_missatge =
m.id_missatge
      and m.cos LIKE '%||(SELECT paraula from missatges_prohibits WHERE id_paraula =
i)||%' )
      >= 1 then
      trobat = true;
    end if;
    i = i +1;
  END LOOP;
  IF trobat then
    INSERT INTO Warnings (affected_table, error_message, date, username)
    VALUES ('missatge','Missatge d"odi enviat amb paraula/s ' || (Select paraula from
missatges_prohibits where id_paraula = i-1)
      || ' a l'usuari ' || new.tag_rep, (SELECT m.data_ from missatge as m where m.id_missatge
= new.id_missatge) ,(SELECT j.nom FROM jugador as j where j.tag_jugador =
new.tag_envia));

    UPDATE jugador
      SET nom = '_banned_' || nom
      WHERE new.tag_envia = tag_jugador;

    end if;
    return null;
  END
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS ofen_trig ON conversen;
CREATE TRIGGER ofen_trig AFTER INSERT ON conversen
FOR EACH ROW
EXECUTE FUNCTION missatges_ofensius();
```

```

DROP FUNCTION if exists missatges_clans CASCADE;
CREATE OR REPLACE FUNCTION missatges_clans()
RETURNS trigger as $$
    DECLARE i INTEGER:= 1;
    DECLARE trobat boolean:= false;
BEGIN
    WHILE i <= (SELECT COUNT(paraula) from missatges_prohibits) AND trobat = FALSE
    LOOP
        IF (SELECT COUNT(m.cos) from missatge as m WHERE new.id_missatge =
m.id_missatge
        and m.cos LIKE '%||(SELECT paraula from missatges_prohibits WHERE id_paula =
i)||%'')
            >= 1 then
            trobat = true;
        end if;
        i = i +1;
    END LOOP;

    IF trobat then
        INSERT INTO Warnings (affected_table, error_message, date, username
VALUES ('missatge','Missatge d"odi enviat amb paraula/s ' || (Select paraula from
missatges_prohibits where id_paula = i-1)
||' al clan '|| new.tag_clan, (SELECT m.data_ from missatge as m where
m.id_missatge = new.id_missatge) ,(SELECT j.nom FROM jugador as j where
j.tag_jugador = new.tag_jugador));

        UPDATE jugador
        SET nom = '_banned_' || nom
        WHERE new.tag_jugador = tag_jugador;
    end if;

    return null;
END
$$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS ofen_trig2 ON envia;
CREATE TRIGGER ofen_trig2 AFTER INSERT ON envia
FOR EACH ROW
EXECUTE FUNCTION missatges_clans();

```

5.2.2 Explicació

En aquest trigger es demana revisa si els missatges contenen una paraula d'odi, si aquest la conte s'ha de baneja l'usuari que ha enviat el missatge i afegir-lo a la taula Warnings, també s'ha d'afegir un missatge d'error indicant el receptor del missatge (clan/jugador) i la taula on hi ha aquest missatge. Per fer això, primer s'ha creat la taula missatges_prohibits per afegir les paraules que es vol detectar com a odi. Posteriorment s'ha creat dues funcions amb la mateixa metodologia, ja que no és podia fer només una, tal i com s'ha creat el nostre model relacional. Aquesta funció primer amb el WHILE LOOP comprova que en el missatge hi hagi una paraula com la de missatges_prohibits si aquest LOOP retorna una variable anomenada trobat com a TRUE aleshores s'ha fet un condicional (IF) per insertar, en cas de ser trobat, a la taula warnings i fer un UPDATE al nom del jugador que ha enviat el missatge per banejar-lo.

5.2.3 Validació

Per poder validar correctament els dos triggers, primer es validara un i posteriorment l'altre per comprovar millor el seu funcionament.

Primer s'ha fet un INSERT a la taula missatge indicant una id aleatoria, un missatge que contingui una de les paraules afegides com a odi i la data actual. Després s'ha fet un altre INSERT a la taula conversen indicant un tag del jugador, un altre tag del jugador i la mateixa id de la taula missatge.

```
INSERT INTO missatge(id_missatge,cos, data_)
VALUES (3024,'You ara very stupid',CURRENT_DATE);
INSERT INTO conversen (tag_envia, tag_rep,id_missatge)
VALUES ('#P8CJYJ02', '#2V20QJVR', 3024);
```

Amb la consulta:

```
SELECT *
from warnings;
```

S'ha fet un SELECT a tota la taula Warning per comprovar si el trigger ha insertat les dades correctament. A continuació es mostra una imatge del resultat d'aquesta consulta:

	affected_table	error_message	date	username
1	missatge	Missatge d'odi enviat amb paraula/s stupid a l'usuari #2V20QJVR	2022-05-24	nick is fat

Seguidament per comprovar que l'UPDATE a la taula jugador s'hagi fet correctament, s'ha realitzat la següent consulta:

```
SELECT nom
from jugador
where tag_jugador like '%#P8CJYJ02%';
```

	nom
1	_banned_ nick is fat

En el segon trigger s'ha fet dos INSERT a la taula missatge amb la mateixa metodologia que l'anterior i a la taula envia afegint la id del missatge, el jugador que envia el missatge i el clan que rep el missatge.

```
INSERT INTO missatge(id_missatge,cos, data_)
VALUES (3061,'Hi idiot I hate you',CURRENT_DATE);
INSERT INTO envia (id_missatge, tag_clan, tag_jugador)
VALUES (3061, '#8LGRYC', '#2V20QJVR');
```

Per comprovar que la inserció fos correcta s'ha fet el mateix SELECT que anteriorment a la taula warnings

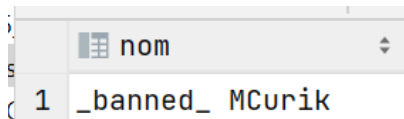
missatge	Missatge d'odi enviat amb paraula/s idiota al clan #8LGRYC	2022-05-27	MCurik
----------	--	------------	--------

I com s'ha fet anteriorment també s'ha fet un select mostrant el nom del jugador per comprovar que s'ha bannejat:

- Consulta:

```
SELECT nom  
from jugador  
where jugador.tag_jugador like '#2V20QJVR';
```

- Resultat:



	nom
1	_banned_ MCurik

Com es pot observar la paraula d'odi enviada era "idiota" tal i com mostra el resultat, la data també surt correcte i el nom del clan i del jugador també corresponent als tag enviat (jugador) i rebut (clan).

5.3 Trigger 3

5.3.1 Solució

```
DROP TABLE IF EXISTS ranquing;
```

```
CREATE TABLE ranquing(  
  id_ranquing SERIAL,  
  tag_jugador VARCHAR(255),  
  arena VARCHAR(255),  
  num_trofeus INTEGER,  
  id_temp VARCHAR(255),  
  PRIMARY KEY (id_ranquing)  
);
```

```
DROP FUNCTION if exists actualitza_ranquing CASCADE;
```

```
CREATE OR REPLACE FUNCTION actualitza_ranquing()
```

```
RETURNS trigger as $$
```

```
  DECLARE temp_anterior VARCHAR (255) := (SELECT id_temporada from temporada where  
temporada.data_fi <> new.data_fi ORDER BY temporada.data_fi desc LIMIT 1);
```

```
  BEGIN
```

```
    INSERT INTO ranquing (tag_jugador, arena, num_trofeus, id_temp)
```

```
    SELECT j2.tag_jugador,a.id_arena,(SELECT(SELECT (CASE WHEN SUM(g.num_trofeus)  
IS NULL THEN 0 ELSE SUM(g.num_trofeus) END)
```

```
      from jugador as j join guanya g on j.tag_jugador = g.tag_jugador
```

```
      join batalla on g.id_batalla = batalla.id_batalla
```

```
      WHERE batalla.id_temporada = temp_anterior
```

```
      and j.tag_jugador = j2.tag_jugador) + (SELECT (CASE WHEN SUM(p.num_trofeus) IS  
NULL THEN 0 ELSE SUM(p.num_trofeus) END)
```

```
      from jugador as j join perd p on j.tag_jugador = p.tag_jugador
```

```
      join batalla as b2 on p.id_batalla = b2.id_batalla
```

```
      WHERE b2.id_temporada = temp_anterior
```

```
      and j.tag_jugador = j2.tag_jugador)
```

```
      from jugador as j3
```

```
      where j3.tag_jugador = j2.tag_jugador
```

```
      GROUP BY j3.tag_jugador),temp_anterior
```

```
    from jugador as j2
```

```
    join arena as a on
```

```
    (SELECT(SELECT (CASE WHEN SUM(g.num_trofeus) IS NULL THEN 0 ELSE  
SUM(g.num_trofeus) END)
```

```
      from jugador as j join guanya g on j.tag_jugador = g.tag_jugador
```

```
      join batalla on g.id_batalla = batalla.id_batalla
```

```
      WHERE batalla.id_temporada = temp_anterior
```

```
      and j.tag_jugador = j2.tag_jugador) + (SELECT (CASE WHEN SUM(p.num_trofeus) IS  
NULL THEN 0 ELSE SUM(p.num_trofeus) END)
```

```
      from jugador as j join perd p on j.tag_jugador = p.tag_jugador
```

```
      join batalla as b2 on p.id_batalla = b2.id_batalla
```

```
      WHERE b2.id_temporada = temp_anterior
```

```
      and j.tag_jugador = j2.tag_jugador)
```

```
      from jugador as j3
```

```
      where j3.tag_jugador = j2.tag_jugador
```

```

GROUP BY j3.tag_jugador) >= a.nombre_min and (SELECT((SELECT (CASE WHEN
SUM(g.num_trofeus) IS NULL THEN 0 ELSE SUM(g.num_trofeus) END)
    from jugador as j join guanya g on j.tag_jugador = g.tag_jugador
    join batalla on g.id_batalla = batalla.id_batalla
    WHERE batalla.id_temporada = temp_anterior
    and j.tag_jugador = j2.tag_jugador) + (SELECT (CASE WHEN
SUM(p.num_trofeus) IS NULL THEN 0 ELSE SUM(p.num_trofeus) END)
    from jugador as j join perd p on j.tag_jugador = p.tag_jugador
    join batalla as b2 on p.id_batalla = b2.id_batalla
    WHERE b2.id_temporada = temp_anterior
    and j.tag_jugador = j2.tag_jugador))
    from jugador as j3
    where j3.tag_jugador = j2.tag_jugador
    GROUP BY j3.tag_jugador) <= a.nombre_max
and (a.titol LIKE '%Arena L10 - Ultimate Champion%'
OR a.nombre_max < 32767)
WHERE j2.tag_jugador IN (SELECT tag_jugador from participen where id_temporada =
temp_anterior)
GROUP BY j2.tag_jugador,a.id_arena,temp_anterior;

RETURN NULL;
END
$$ LANGUAGE plpgsql;

```

```

DROP TRIGGER IF EXISTS rank_trig ON temporada;
CREATE TRIGGER rank_trig AFTER INSERT ON temporada
FOR EACH ROW
EXECUTE FUNCTION actualitza_ranquing();

```

5.3.2 Explicació

Per aquest trigger, s'havia de crear un rànquing de jugadors de la temporada anterior a la que insertem, ja que ens diu que s'ha de considerar que al final d'una temporada tingui lloc sempre que una temporada comenci de la darrera i, per tant, sempre que s'afegeix a la base de dades. Primer de tot, s'ha declarat una variable anomenada `temp_anterior` de tipus `VARCHAR` que se li assigna la temporada anterior a la que acaban d'insertar. Posteriorment s'ha fet un `INSERT` a la taula rànquing. Per afegir els valors s'ha fet un `SELECT`, seleccionant els que pertanyen a la temporada anterior tal i com s'indica al `WHERE`. Per suma els trofeus s'han fet dos subconsultes i s'ha indicat un `CASE WHEN` per verificar que el valor que surt no sigui null, ja que si es null i es suma amb l'altre valor el resultat serà null i la consulta no seria vàlida. Per tant, perquè això no passes s'ha posat aquesta funció i així el valor a retornar quan sigui null serà 0. Per trobar l'arena en que s'ha classificat el jugador s'ha mirat que la suma anterior no sigui més gran o més petita que els números d'arena i que sigui més petita que el valor màxim o igual a l'arena més gran, d'aquesta manera és descartava totes les arenes no vàlides i es troba la del jugador indicat per la temporada. Per últim, abans de fer el trigger s'ha creat la taula `RANQUING` per poder afegir tots els valors indicats anteriorment.

5.3.3 Validació

Per validar correctament aquest exercici, primer s'ha fet varios inserts per després comprovar que el trigger funciona correctament. S'ha creat primer de tot una temporada amb unes dates inventades, posteriorment s'ha afegit aquesta temporada i un jugador agafat aleatòriament a la taula participen. Seguidament s'ha fet un insert a la taula Batalla indicant la temporada anterior, la data anterior i una duració aleatòria. I finalment s'ha afegit el mateix jugador aleatori que anteriorment a la taula perd i guanya i també s'ha afegit la id de la batalla generat anteriorment amb l'insert, una id de pila aleatòria i un número de trofeus per provar.

```
INSERT INTO temporada (id_temporada, data_inici, data_fi)
VALUES ('T11', '2022-01-01', '2022-08-20');
INSERT INTO participen (tag_jugador, id_temporada)
VALUES ('#QV2PYL', 'T11');
INSERT INTO batalla (data, durada, id_temporada)
VALUES ('2022-01-02', '03:52:00', 'T11');
INSERT INTO guanya (tag_jugador, id_batalla, num_trofeus, id_pila)
VALUES ('#QV2PYL', 9920, 80, 102);
INSERT INTO perd (tag_jugador, id_batalla, id_pila, num_trofeus)
VALUES ('#QV2PYL', 9920, 1113, -67);
```

Un cop insertades totes les dades, s'ha executat la funció i el trigger i posteriorment s'ha afegit un nou INSERT d'una nova temporada.

```
INSERT INTO temporada (id_temporada, data_inici, data_fi)
VALUES ('T12', '2023-01-01', '2023-08-20');
```

Finalment per obtenir el resultat del rànding s'ha fet un SELECT d'aquesta taula i així comprovar si s'ha fet correctament el trigger

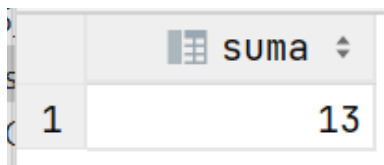
```
SELECT *
FROM ranquing;
```

	id_ranquing	tag_jugador	arena	num_trofeus	id_temp
1	1	#QV2PYL	54000001	13	T11

Per comprovar que aquests resultats son correctes s'han realitzat les següents consultes:

Primer s'ha comprovat que la suma dels trofeus fos correcta:

```
(SELECT((SELECT (CASE WHEN SUM(g.num_trofeus) IS NULL THEN 0 ELSE SUM(g.num_trofeus) END)
from jugador as j join guanya g on j.tag_jugador = g.tag_jugador
join batalla on g.id_batalla = batalla.id_batalla
WHERE batalla.data >= '2022-01-02' and batalla.data <= '2022-08-20'
and j.tag_jugador = '#QV2PYL') + (SELECT (CASE WHEN SUM(p.num_trofeus) IS NULL THEN 0 ELSE SUM(p.num_trofeus) END)
from jugador as j join perd p on j.tag_jugador = p.tag_jugador
join batalla as b2 on p.id_batalla = b2.id_batalla
WHERE b2.data >= '2022-01-01' and b2.data <= '2022-08-20'
and j.tag_jugador = '#QV2PYL')) as suma
from jugador as j
where j.tag_jugador = '#QV2PYL'
GROUP BY j.tag_jugador);
```

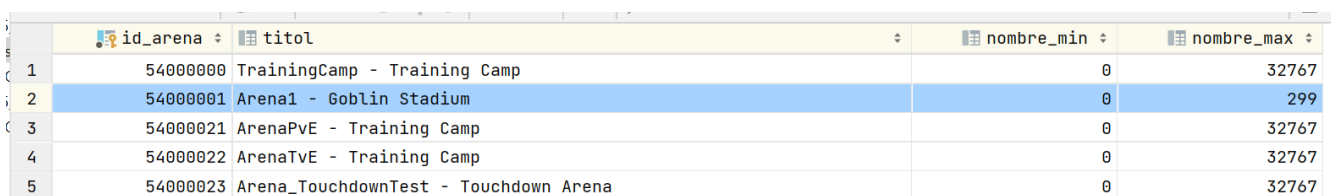


	suma
1	13

Seguidament s'ha comprovat que el jugador escollit aleatòriament, es classifiqui a l'arena 54000001 tal i com mostra el rànkung:

- Com la suma de trofeus es 13. Fent la consulta:

```
SELECT *
FROM arena;
```



	id_arena	titol	nombre_min	nombre_max
1	54000000	TrainingCamp - Training Camp	0	32767
2	54000001	Arena1 - Goblin Stadium	0	299
3	54000021	ArenaPvE - Training Camp	0	32767
4	54000022	ArenaTvE - Training Camp	0	32767
5	54000023	Arena_TouchdownTest - Touchdown Arena	0	32767

Obtenim que l'arena corresponent al número de trofeus del jugador escollit aleatoriament es la 54000001 tal i ens mostra el resultat del trigger.

6 Tingueu valor. Encara tenim el nostre clan. Sempre hi ha esperança

6.1 Funció *f_selNewLeader*

Donat que el trigger 1 i 2 tenen una funcionalitat compartida, assignar un leader, s'ha creat una funció en la qual li passes el tag d'un clan aquesta fa la lògica per escollir el nou leader.

6.1.1 Solució

```
CREATE OR REPLACE FUNCTION f_selNewLeader(VARCHAR)
RETURNS void AS $$
DECLARE
    leaderID INTEGER = (SELECT id_rol FROM rol WHERE nom = 'leader');
    coLeaderID INTEGER = (SELECT id_rol FROM rol WHERE nom = 'coLeader');
    memberID INTEGER = (SELECT id_rol FROM rol WHERE nom = 'member');
    randColeader VARCHAR;
    randMember VARCHAR;
    countColeader INTEGER;
BEGIN
    -- Setting values to variables
    SELECT count(id_forma_part) INTO countColeader
    FROM forma_part WHERE id_rol = coLeaderID AND tag_clan = $1;
    SELECT tag_jugador INTO randColeader
    FROM forma_part
    WHERE id_rol = coLeaderID AND tag_clan = $1
    OFFSET floor(random() * countColeader)
    LIMIT 1;
    SELECT tag_jugador INTO randMember
    FROM forma_part
    WHERE id_rol = memberID AND tag_clan = $1
    OFFSET floor(random() * (SELECT count(id_forma_part) FROM forma_part WHERE id_rol
= memberID AND tag_clan = $1))
    LIMIT 1;

    -- Function logic
    IF countColeader > 0
    THEN
        UPDATE forma_part
        SET id_rol = leaderID
        WHERE tag_clan = $1 AND tag_jugador = randColeader AND id_rol = coLeaderID;
    ELSE
        UPDATE forma_part
        SET id_rol = leaderID
        WHERE tag_clan = $1 AND tag_jugador = randMember AND id_rol = memberID;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

6.1.2 Explicació

Per tal d'evitar la repetició de consultes i així optimitzar el SGBD s'han creat diverses variables amb els ids dels rols i amb la quantitat de coleaders. Al principi es mira quin és el nombre de coleaders, si aquest és més gran de 0 (hi ha coleaders) s'escull un coleader de forma aleatoria, mitjançant un OFFSET el qual comença a mostrar a partir d'una fila en concret i obviarà les

anteriors. Per escollir aquest valor s'utilitza la funció random la qual ens dona un nombre aleatori entre 0 i 1, aquest és multiplicat pel nombre de coleaders i arrodonit a un nombre enter amb floor(). En el cas de que no hi hagin coleaders executa al mateix amb els membres.

6.1.3 Validació

Cridant aquesta funció de forma manual posant-hi un clan en concret s'ha comprovat que actualitza el id_rol d'un coleader a leader de manera aleatoria i en el cas de que no hi hagin coleaders actualitza el rol d'un membre.

6.2 Trigger 1

6.2.1 Solució

```
DROP TABLE IF EXISTS logDeletes;
```

```
CREATE TABLE logDeletes(  
  id SERIAL PRIMARY KEY ,  
  tag_removed VARCHAR,  
  tag_clan VARCHAR,  
  id_rol INTEGER,  
  tag_leader VARCHAR,  
  removed_date date  
);
```

```
CREATE OR REPLACE FUNCTION f_CopdEfecte()
```

```
RETURNS trigger AS $$
```

```
DECLARE
```

```
  newestLeader INTEGER = (SELECT id_forma_part FROM forma_part
```

```
                        WHERE tag_clan = OLD.tag_clan AND id_rol = (SELECT id_rol FROM rol
```

```
WHERE nom = 'leader')
```

```
                        ORDER BY data DESC LIMIT 1);
```

```
  item INTEGER;
```

```
BEGIN
```

```
  IF NEW.id_rol IS NULL
```

```
  THEN
```

```
    INSERT INTO logDeletes(tag_removed, tag_clan, id_rol, tag_leader, removed_date) VALUES  
(OLD.tag_jugador, OLD.tag_clan, OLD.id_rol, newestLeader, now());
```

```
    UPDATE forma_part
```

```
      SET jugadors_eliminars = jugadors_eliminars + 1
```

```
      WHERE id_forma_part = newestLeader;
```

```
    IF ((SELECT data + interval '1 day' FROM forma_part WHERE id_forma_part =  
newestLeader) > now())
```

```
    THEN
```

```
      IF ((SELECT jugadors_eliminars FROM forma_part WHERE id_forma_part =  
newestLeader) > 5)
```

```
      THEN
```

```
        -- Desfer canvis
```

```
        -- Deleting logs older than 24h that aren't relevant and it is unnecessary storing them
```

```
        DELETE FROM logDeletes
```

```
        WHERE (removed_date - interval '24 hours') > now();
```

```
      FOR item IN (SELECT id FROM logDeletes WHERE tag_leader = (SELECT tag_leader  
FROM forma_part WHERE id_forma_part = newestLeader) AND tag_clan = OLD.tag_clan)
```

```
      LOOP
```

```

        UPDATE forma_part
        SET id_rol = (SELECT id_rol FROM logDeletes WHERE id = item)
        WHERE tag_clan = OLD.tag_clan AND tag_jugador = (SELECT tag_removed FROM
logDeletes WHERE id = item);
    END LOOP;

    -- Downgrade
    UPDATE forma_part
    SET id_rol = NULL, jugadors_eliminats = 0
    WHERE tag_clan = OLD.tag_clan AND id_forma_part = newestLeader;
    PERFORM f_selNewLeader(OLD.tag_clan);

    END IF;
    END IF;
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS CopdEfecte ON forma_part;
CREATE TRIGGER CopdEfecte
AFTER UPDATE ON forma_part
FOR EACH ROW
WHEN (pg_trigger_depth() = 0) -- Per evitar que el trigger salti al fer updates de forma part dins
del mateix trigger
EXECUTE FUNCTION f_CopdEfecte();

```

6.2.2 Explicació

Aquest trigger s'executarà després de que hi hagi una actualització a la taula forma_part (relació clan - rol - jugador) i sempre que aquesta modificació no s'hagi realitzat dins d'un trigger. Això s'ha fet donat que dins de la funció actualitzem el rol dels jugador i de no posar-ho és cridaria a ell mateix de forma recursiva.

A la funció es comença per buscar quin és el líder actual (filtran per clan, tipus de rol i ordenant per data d'actualització). A continuació, si el nou rol és nul (ha sigut eliminat) guardem l'antic rol i qui ha realitzat el canvi en una taula auxiliar per si després s'han de desfer els canvis recuperar-los, també s'incrementa el nombre d'usuaris que ha eliminat el líder. Després es comprova si fa menys d'un dia que el jugador es líder comparant la data actual amb la data d'assignació del rol més un dia. En el cas de que es compleixin les dues condicions s'eliminen els logs de la taula auxiliar més antics de 24h donats que ja no són útils i amb un bucle, per cada element restant a la taula auxiliar que hagi eliminat el líder actual, es retorna el seu valor original mitjançant un UPDATE. Un cop acabat de restaurar tots els elements el líder actual passa a tenir rol NULL i s'executa la procedure per seleccionar el nou líder.

6.2.3 Validació

Per a la validació d'aquest trigger s'ha escollit un clan a l'atzar i eliminat varis usuaris, donat que el líder d'aquest clan portava més d'un dia com a líder el nombre de jugadors eliminats ha anat incrementant sense restablir cap canvi al posar un valor NULL a jugadors del mateix clan.

WHERE tag_clan = '#L0YJ02YL'						ORDER BY id_rol	
	id_forma_part	tag_clan	tag_jugador	id_rol	data	jugadors_eliminats	
5	133	#L0YJ02YL	#89UV8ULY	2	2021-05-07	8	

A posteriori s'ha canviat la data d'actualització del rol i s'ha posat jugadors eliminats a 0 de nou. S'ha realitzat 5 UPDATES posant el id_rol a NULL sense problema, però al 5é el trigger ha entrat en acció i els usuaris eliminats s'ha restaurat, el id_rol del líder ha passat a valdre NULL i s'ha escollit un nou líder, els jugadors que havien sigut expulsats abans d'aquest líder segueixen expulsats.

WHERE tag_clan = '#L0YJ02YL'				ORDER BY id_rol DESC		
	id_forma_part	tag_clan	tag_jugador	id_rol	data	jugadors_eliminats
1	120	#L0YJ02YL	#26RQQU02	<null>	2021-09-15	0
2	133	#L0YJ02YL	#89UV8ULY	<null>	2022-05-27	0
3	122	#L0YJ02YL	#CCQCU90L	<null>	2021-09-10	0
4	110	#L0YJ02YL	#89RYUPQ0V	<null>	2021-10-23	0
5	114	#L0YJ02YL	#Q88YUVLG	<null>	2021-05-05	0
6	112	#L0YJ02YL	#98R0LCGG	<null>	2022-05-12	0

6.3 Trigger 2

6.3.1 Solució

```

DROP function IF EXISTS f_minTrofeus;
CREATE OR REPLACE FUNCTION f_minTrofeus()
RETURNS trigger AS $$
DECLARE
    currentClan VARCHAR;
    idFormaPart INTEGER;
BEGIN
    SELECT tag_clan INTO currentClan
    FROM forma_part
    WHERE tag_jugador = NEW.tag_jugador
    ORDER BY data desc
    LIMIT 1;

    SELECT id_forma_part INTO idFormaPart
    FROM forma_part
    WHERE tag_jugador = NEW.tag_jugador
    ORDER BY data desc
    LIMIT 1;

    INSERT INTO dummylog VALUES (CONCAT('Jug: ', NEW.tag_jugador, ' T: ',
NEW.trofeus, ' Clan: ', currentClan), now());

    INSERT INTO dummylog VALUES (CONCAT('min clan: ', (SELECT trofeus_minims
FROM clan WHERE tag_clan = currentClan), ' T: ', NEW.trofeus ));

    IF (SELECT trofeus_minims FROM clan WHERE tag_clan = currentClan) > NEW.trofeus
    THEN
        INSERT INTO dummylog VALUES ('A eliminar', now());
        IF (SELECT id_rol FROM forma_part WHERE id_forma_part = idFormaPart) = (SELECT
id_rol FROM rol WHERE nom = 'leader')
        THEN
            INSERT INTO dummylog VALUES ('A escollir lider', now());

```

```

        PERFORM f_selNewLeader(currentClan);
    end if;
    UPDATE forma_part
    SET id_rol = NULL, data = now()
    WHERE id_forma_part = idFormaPart;
END IF;
RETURN NULL;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS minTrofeus ON jugador;
CREATE TRIGGER minTrofeus
    AFTER UPDATE OF trofeus ON jugador
    FOR EACH ROW
    EXECUTE FUNCTION f_minTrofeus();

```

6.3.2 Explicació

Aquest trigger s'accionarà per cada cop que s'actualitzi els trofeus d'un jugador. La primera part del trigger consta en guardar en dues variables el tag del clan el qual està actualment, el que té la relació data-rol-clan més recent, i quin id te en aquesta taula (forma_part).

A continuació es comprova que el nou valor de trofeu sigui inferior al mínim del clan, en cas contrari no es fa res, acte seguit és comprova si el jugador es un líder, en cas que sí s'executa el procedured f_selNewLeader(). Tant si era líder com si no es posa el id_rol d'aquest jugador a NULL i s'actualitza la data d'assignació del rol.

6.3.3 Validació

Per realitzar la validació s'ha seleccionat un clan en concret i mostrat el seu líder, els colíders i els jugadors expulsats. A continuació s'ha realitzat un update posant un nombre de trofeus inferior al mínim del clan al lider, això ha fet que aquest passes a ser expulsat i s'escull un nou líder aleatori.

Abans del update

WHERE tag_clan = '#L0YJ02YL' and (id_rol = 2 OR id_rol = 3 OR id_rol IS NULL)

	id_forma_part	tag_clan	tag_jugador	id_rol	data
1	114	#L0YJ02YL	#Q88YUVLG	2	2022-05-27
2	142	#L0YJ02YL	#L00C8Y9Q2	3	2021-08-24
3	149	#L0YJ02YL	#QL8CJUYQ	3	2021-09-18

Després del update

WHERE tag_clan = '#L0YJ02YL' and (id_rol = 2 OR id_rol = 3 OR id_rol IS NULL)

	id_forma_part	tag_clan	tag_jugador	id_rol	data
1	114	#L0YJ02YL	#Q88YUVLG	<null>	2022-05-27
2	142	#L0YJ02YL	#L00C8Y9Q2	3	2021-08-24
3	149	#L0YJ02YL	#QL8CJUYQ	2	2021-09-18

6.4 Trigger 3

6.4.1 Solució

```
CREATE OR REPLACE FUNCTION f_malsPerdedors()
RETURNS trigger AS $$
DECLARE
    warningMsg VARCHAR;
BEGIN
    IF NOT current_user = 'admin'
    THEN
        SELECT CONCAT('S''ha intentat esborrar la batalla ', OLD.id_batalla, ' on l''usuari ',
        (SELECT tag_jugador FROM perd WHERE id_batalla = OLD.id_batalla), ' va perdre ', (SELECT
        num_trofeus FROM perd WHERE id_batalla = OLD.id_batalla), ' trofeus') INTO warningMsg;
        INSERT INTO warnings (affected_table, error_message, date, username) VALUES ('batalla',
        warningMsg, now(),current_user);
    END IF;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS malsPerdedors ON jugador;
CREATE TRIGGER malsPerdedors
BEFORE DELETE ON batalla
FOR EACH ROW
EXECUTE FUNCTION f_malsPerdedors();
```

6.4.2 Explicació

Es crea un trigger el qual s'acciona abans de fer un DELETE a la taula de batalla i s'executa per cada fila donat que s'han de guardar elements de cada fila eliminada a la taula d'avisos. El primer que es realitza és comprovar que l'usuari actual (el que ha executat la sentència) sigui diferent a admin, si és admin no fa falta fer res. A continuació es realitza un CONCATCAT per guardar en una variable VARCHAR el text del missatge a introduir a la taula Warnings. Finalment es fa un INSERT a aquesta taula amb la cadena anterior, la taula afectada, la data i el usuari que ha executat la sentència.

6.4.3 Validació

Si es selecciona les dades del perdedor d'una batalla (1) i s'elimina amb un usuari diferent a admin s'afegeix el log corresponent a la taula d'avisos. Pel contrari si la petició la fa l'usuari admin no queda registrat.

	tag_jugador	id_batalla	id_pila	num_trofeus
1	#P99JPCLQ8	1	716	-39

	affected_table	error_message	date	username
1	batalla	S'ha intentat esborrar la batalla 1 on l'usuari #P99JPCLQ8 va perdre -39 trofeus	2022-05-27	postgres

7 M'agrada la competició. M'agraden els reptes ...

7.1 Trigger 1

7.1.1 Solució

```
DROP FUNCTION IF EXISTS update_gold_experience CASCADE;
```

```
CREATE OR REPLACE FUNCTION update_gold_experience ()
```

```
RETURNS trigger AS $$
```

```
BEGIN
```

```
    IF ((SELECT id_missio2 FROM depen
```

```
        WHERE id_missio1 = NEW.id_missio) IN (SELECT id_missio FROM completen
                                                WHERE tag_jugador = NEW.tag_jugador))
```

```
    THEN
```

```
        UPDATE jugador SET
```

```
        or_ = or_ + NEW.or_;
```

```
        experiència = experiència + NEW.experiència
```

```
        WHERE tag_jugador = NEW.tag_jugador;
```

```
    ELSE
```

```
        INSERT INTO warnings (affected_table, error_message, date, username)
```

```
        VALUES ('completen',
```

```
                'L'entrada de la quest per a "' || (SELECT titol FROM missio
```

```
                    WHERE id_missio = NEW.id_missio) ||
```

```
                "' s'ha realitzat sense completar el "' || (SELECT titol FROM missio AS m
```

```
                    JOIN depen AS d ON m.id_missio = d.id_missio2
```

```
                    WHERE d.id_missio1 = NEW.id_missio) ||
```

```
                "' prerequisit',
```

```
                CURRENT_DATE,
```

```
                NEW.tag_jugador);
```

```
    END IF;
```

```
    RETURN NULL;
```

```
END $$
```

```
LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS missionComplete ON completen CASCADE;
```

```
CREATE TRIGGER missionComplete AFTER INSERT ON completen
```

```
FOR EACH ROW
```

```
EXECUTE FUNCTION update_gold_experience();
```


7.1.2 Explicació

Bé, primer de tot, creem la funció que executarà el trigger. Aquesta, resumidament, comprovar que es compleix el prerequisit de la missió, que és que s'hagi completat una altra missió abans de la que es vol completar ara. Un cop ha comprovat que es compleix el prerequisit, fa un update de l'or i l'experiència del jugador. En cas que no es compleixi el prerequisit comentat anteriorment, s'afegeix a la taula de warnings el missatge d'error corresponent.





7.1.3 Validació

Per a fer la validació, s'ha utilitzat una sèrie de subconsultes les quals s'explicaran amb ordre a continuació:



Fem un SELECT d'un jugador per veure l'or i l'experiència que té, d'aquesta manera podrem comparar si es fa la inserció desitjada:

tag_jugador	nom	experiencia	trofeus	targeta_credit	or_	gemmes
1	#202C2CU0U	Manoel	126601	6869 0626381901632479	89089	740

Ara fem un SELECT per veure les missions que ha fet aquest mateix jugador:

	 id_missio ▾	 titol ▾	 descripcio ▾	 requeriment ▾
1	50	Jacqueline	Funcheon	Donate 689 gems
2	103	Hanna	Deares	Donate 988 gems
3	190	Lavena	Scarth	Donate 528 gold

A continuació, mirem si aquestes missions depenen d'alguna altra missió:

	 id_missio1	 id_missio2
1	50	103

Veiem que la missió 50 depèn de la missió 103, és a dir, que s'ha d'haver completat la missió 103 abans de realitzar la 50.

Ara insertem a la taula "completen" amb la missió 50 ja que sabem que ja ha completat la missió 103 i per tant ens hauria d'inserir l'or i experiència que tingui aquella missió:

```
INSERT INTO completen (id_missio, id_arena, tag_jugador, or_, experiencia, desbloqueja)
VALUES (50, 54000057, '#202C2CU0U', 111, 3399, CURRENT_DATE);
```

Tornem a fer SELECT del jugador per veure si s'ha inserit tot correctament:

tag_jugador	nom	experiencia	trofeus	targeta_credit	or_	gemmes
1	#202C2CU0U	Manoel	130000	6869 0626381901632479	89200	740

Com podem observar, es valors d'or i experiència s'han actualitzat correctament.

Ara, farem la validació amb la taula warnings per veure si, en cas d'error, s'insereix correctament:

Per fer-ho, s'ha fet SELECT d'un altre jugador i s'ha mirat quines missions ha completat:

	id_missio	titol	descripcio	requeriment
1	170	Hugibert	Janczak	Donate 1293 gems

Veiem que aquest jugador només ha completat una missió (amb ID = 170).

Ara, mirant la taula “depen” (que ens indica de quina missió depèn una missió), veiem que la missió 170 depèn de la missió 191, i veiem que aquest jugador no ha realitzat la missió 191, és per això que, en cas d'inserir a la taula “completen”, hauria d'inserir l'error a la taula “warnings” ja que no ha completat el prerequisit. Per tant, anem a comprovar-ho inserint la missió 170 a “completen”:

```
INSERT INTO completen (id_missio, id_arena, tag_jugador, or_, experiencia, desbloqueja)
VALUES (170, 54000057, '#J8QPJY8Q', 100, 2000, CURRENT_DATE);
```

Quan fem l'insert, automàticament s'insereix l'error a la taula de “warnings” indicant que la missió 170 (“Hugibert”) s'ha realitzat abans de realitzar la missió 191 (“Regan”), i per tant, veiem que el trigger funciona correctament:

	affected_table	error_message	date	username
1	completen	L'entrada de la quest per a "Hugibert" s'ha realitzat sense completar el "Regan" prerequisit	2022-05-23	#J8QPJY8Q

7.2 *Trigger 2*

7.2.1 *Solució*

```
DROP FUNCTION IF EXISTS updateTrophies CASCADE;
```

```
CREATE OR REPLACE FUNCTION updateTrophies()  
RETURNS trigger AS $$  
BEGIN  
UPDATE jugador  
SET  
    trofeus = trofeus + NEW.num_trofeus  
WHERE tag_jugador = NEW.tag_jugador;  
RETURN NULL;  
END $$  
LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS battleWon ON guanya CASCADE;
```

```
CREATE TRIGGER battleWon AFTER INSERT ON guanya  
FOR EACH ROW  
EXECUTE FUNCTION updateTrophies();
```

```
DROP TRIGGER IF EXISTS battleLost ON perd CASCADE;
```

```
CREATE TRIGGER battleLost AFTER INSERT ON perd  
FOR EACH ROW  
EXECUTE FUNCTION updateTrophies();
```

7.2.2 *Explicació*

Aquesta consulta està composta per dos triggers ja que ens interessa que s'actualitzin les dades quan se'ns insereix a la taula "guanya" o a la taula "perd". Per tant, quan hi ha una inserció en alguna d'aquestes taules, cridem a una funció que simplement ens modifica els trofeus del jugador depenent de si guanya o perd i li suma o resta trofeus, respectivament. En aquest trigger no es demanava modificar la taula "warnings".

7.2.3 Validació

Primer de tot, inserim una batalla nova a la taula batalles. El id, al ser un SERIAL, es posa sol:

```
INSERT INTO batalla (data, durada, id_temporada)
VALUES (CURRENT_DATE, '02:45:00', 'T4');
```

Ara, comprovem els trofeus que tenen dos jugadors per més endavant veure si es modifiquen:

	tag_jugador	nom	experiencia	trofeus	targeta_credit	or_	gemmes
1	#VGR9CL0G	*L*0*R*D*	252745	6900	0626868393116342	79863	688
2	#LRUQQPVU	B Host	217012	6570	0626354846476302	308	197

A continuació, farem que el jugador amb tag = '#VGR9CL0G' perdi la partida que juga amb el jugador amb tag = '#LRUQQPVU' i, per tant, que aquest sigui el guanyador.

Per fer-ho inserirem a la taula guanya i a la taula perd els jugadors comentats i comprovarem que els modifiqui els trofeus:

```
INSERT INTO guanya (tag_jugador, id_batalla, id_pila, num_trofeus)
VALUES ('#LRUQQPVU', 9923, 193, 30);
/* Inserim un perdedor a la taula perd*/
INSERT INTO perd (tag_jugador, id_batalla, id_pila, num_trofeus)
VALUES ('#VGR9CL0G', 9923, 1760, -30);
```

Si tot funciona, s'hauria de sumar 30 trofeus al jugador '#LRUQQPVU' i s'hauria de restar 30 al jugador '#VGR9CL0G':

	tag_jugador	nom	experiencia	trofeus	targeta_credit	or_	gemmes
1	#VGR9CL0G	*L*0*R*D*	252745	6870	0626868393116342	79863	688
2	#LRUQQPVU	B Host	217012	6600	0626354846476302	308	197

Com podem observar, s'ha modificat correctament els trofeus de cada jugador.

7.3 Trigger 3

7.3.1 Solució

DROP FUNCTION IF EXISTS comprova CASCADE;

```
CREATE OR REPLACE FUNCTION comprova()
RETURNS trigger AS $$
DECLARE totOK boolean:= true;
BEGIN
IF (NEW.quantitat IS NULL)
THEN
    totOK = false;
    INSERT INTO warnings (affected_table, error_message, date, username)
    VALUES ('dona',
            'S"ha intentat fer una donació nul·la',
            CURRENT_DATE,
            NEW.tag_jugador);
END IF;
IF (NEW.tag_clan <> (SELECT tag_clan FROM forma_part
                    WHERE tag_jugador = NEW.tag_jugador))
THEN
    totOK = false;
    INSERT INTO warnings (affected_table, error_message, date, username)
    VALUES ('dona',
            'S"ha realitzat una donació de ' || NEW.quantitat ||
            'd"or a ' || NEW.tag_clan ||
            'sense pertànyer al clan',
            CURRENT_DATE,
            NEW.tag_jugador);
END IF;
IF ((SELECT jugadors_eliminats FROM forma_part
      WHERE tag_jugador = NEW.tag_jugador) = 1)
THEN
    totOK = false;
    INSERT INTO warnings (affected_table, error_message, date, username)
    VALUES ('dona',
            'S"ha realitzat una donació de ' || NEW.quantitat ||
            'd"or a ' || NEW.tag_clan ||
            'i el jugador va ser expulsat d"aquest',
            CURRENT_DATE,
            NEW.tag_jugador);
END IF;
IF (totOK = false)
THEN
    DELETE FROM dona
    WHERE id_donacio = NEW.id_donacio;
END IF;
RETURN NULL;
END $$
LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS comprovaDonacio ON dona CASCADE;
```

```
CREATE TRIGGER comprovaDonacio AFTER INSERT OR UPDATE ON dona  
FOR EACH ROW  
EXECUTE FUNCTION comprova();
```

7.3.2 Explicació

Aquest trigger, és probablement el més llarg d'aquesta secció. El que es demana, és que controlem les inconsistències a la taula de donacions. És a dir, que controlem que no se'ns faci donacions amb valors nuls, o que es facin donacions si un usuari no pertany a un clan, etc.. Per fer-ho, s'ha fet una funció que es crida cada cop que es fa un INSERT a la taula "dona". Aquesta funció està composta per quatre IFs que controlen cada error que pot haver-hi. Un per controlar que la quantitat que es dona no sigui nul·la, un altre per controlar que el jugador que faci la donació pertany al clan, el tercer per comprovar que el jugador que fa la donació no hagi estat eliminat del clan anteriorment i, finalment, l'últim IF comprova que si s'ha produït un sol error de tots els comentats anteriorment, eliminarà aquella donació de la taula, ja que aquesta és corrupte.

7.3.3 Validació

Per a fer aquesta validació simplement insertarem valors incorrectes per que es comprovi que el trigger elimina de la taula aquelles donacions corruptes i, després farem una donació correcta per a que es vegi que en cas d'una bona donació, tot funciona correctament.

Comencem inserint una donació amb quantitat nul·la:

```
INSERT INTO dona (tag_jugador, tag_clan, quantitat, data)  
VALUES ('#QV2PYL', '#8LGRYC', null, CURRENT_DATE);
```

Veiem que la taula "warnings" ens retorna l'error desitjat:

	affected_table	error_message	date	username
1	dona	S'ha intentat fer una donació nul·la	2022-05-23	#QV2PYL

Ara, inserim una donació d'un jugador que no pertanyi al clan (s'ha comprovat que el tag_clan introduït és un al que el jugador no pertany):

```
INSERT INTO dona (tag_jugador, tag_clan, quantitat, data)  
VALUES ('#QV2PYL', '#2CQQVQCU', 10, CURRENT_DATE);
```

Veiem que ara warnings conté l'error anterior i el nou error de que no pertany al clan:

	affected_table	error_message	date	username
1	dona	S'ha intentat fer una donació nul·la	2022-05-23	#QV2PYL
2	dona	S'ha realitzat una donació de 10d'or a #2CQQVQCU sense pertànyer al clan	2022-05-23	#QV2PYL

Finalment, fem una donació correcte per a què es mostri que no s'afegeix res a la taula warnings:

```
INSERT INTO dona (tag_jugador, tag_clan, quantitat, data)
VALUES ('#QV2PYL', '#8LGRYC', 10, CURRENT_DATE);
```

En aquest cas, comprovem que el jugador pertanyi al clan i a més fa una donació amb valor de quantitat 10, veiem que a la taula warnings no s'insereix res i es queda igual que anteriorment i, si anem a la taula dona, s'haurà inserit correctament (és la última fila de la captura):

affected_table	error_message	date	username
1 dona	S'ha intentat fer una donació nul·la	2022-05-23	#QV2PYL
2 dona	S'ha realitzat una donació de 10d'or a #2CQQVQCUsense pertànyer al clan	2022-05-23	#QV2PYL

	id_donacio	tag_jugador	tag_clan	quantitat	data
5395	5395	#YLRVGPJP	#8VUVGPVC	50	2021-08-25
5396	5396	#YLRVGPJP	#8VUVGPVC	40	2021-09-13
5397	5397	#YLRVGPJP	#8VUVGPVC	30	2021-08-08
5398	5398	#YLRVGPJP	#8VUVGPVC	20	2021-10-11
5399	5399	#8L200J88G	#8VUVGPVC	40	2021-10-21
5400	5400	#8L200J88G	#8VUVGPVC	35	2021-10-12
5401	5401	#8L200J88G	#8VUVGPVC	20	2021-11-07
5402	5402	#8L200J88G	#8VUVGPVC	45	2021-10-29
5403	5403	#8L200J88G	#8VUVGPVC	15	2021-09-09
5404	5404	#8L200J88G	#8VUVGPVC	45	2021-10-17
5405	5405	#8L200J88G	#8VUVGPVC	20	2021-08-04
5406	5406	#8L200J88G	#8VUVGPVC	35	2021-08-28
5407	5413	#QV2PYL	#8LGRYC	10	2022-05-23

8 Conclusions

8.1 Recursos emprats

Eta pa	Marc Geremias	Marc Valsells	Irina Aynés	Albert Tomàs	Total
Actualització dels models Entitat-Relació, Relacional i/o Físic	30min	10min	0min	2min	42min
Implementació dels triggers	5h	7h	6h	5h	23h
Memòria	3h	2h	3h	4h	12h
Total:	8h 30min	9h 10min	9h	9h 02min	35h 42min

Com es pot observar, les actualitzacions dels models entitat-relació i el relacional han estat molt poques, ja que en les anteriors fases de la pràctica ja es va anar modificant tot de manera que ja per aquesta última fase, estava tot ben implementat i no feia falta gairebé canvis. En el model físic l'únic que s'ha fet és crear la taula “warnings” que era necessària per aquesta fase exclusivament.

En quant a la resta, veiem que el temps dedicat als triggers, no ha estat tant com per exemple el temps que es va dedicar a la fase 3 amb les consultes, o a la fase 3 amb la implementació i inserció de dades a la base. Ha estat una fase més tranquil·la.

Finalment en quant al temps de memòria, sempre porta una estona explicar les consultes que es fan (en aquest cas els triggers) i les seves respectives validacions, que és el que més temps ens sol portar, però com veiem, els números no s'allunyen de la fase 3, 2 i 1 de la pràctica.

8.2 *Lliçons apreses i conclusions*

Bé, ara que s'acaba la pràctica després d'aquesta entrega, cal explicar les conclusions apreses al llarg de la pràctica de ClashSayale.

Per començar, cal recalcar que aquesta pràctica és la demostració de la feina que porta crear una base de dades funcional desde zero, desde que es fa un esquema principal, passant per els models entitat-relació, relacionals i físics, fins a la implementació i inserció de dades, tenint en compte els esquemes realitzats i respectant les claus primàries, forànies i relacions que s'han creat per intentar fer una base de dades més òptima.

Durant aquest curs i al llarg de les fases, ens n'hem adonat que hi ha hagut fases potser amb un nivell de dificultat més elevat, com per exemple la fase 3, la qual tractava de consultes per validar que la nostra base de dades tingués una estructura ben feta. Aquesta sens dubte va ser la fase que requeria més coneixement de l'assignatura. També les primeres fases eren molt importants ja que tot passa per tenir un bon model entitat-relació i relacional, ja que si la base de dades no està ben estructurada, no hi ha manera de prosseguir després.

Durant aquesta fase 4 que tractava de triggers, hem posat a la pràctica els coneixements apresos a classe i durant les activitats d'avaluació continuada per veure si hem adquirit els coneixements. Al principi els triggers poden semblar una mica enrevessats, però un cop et fas a la idea de com funcionen i les estructures que segueixen, simplement és seguir el patró i, es demostra amb els temps que hem emprat per aquesta fase 4 que, en comparació amb la fase 3 o la fase 2, ha estat més reduït.

També s'ha de dir que hem tingut molta sort amb el grup que som, ja que cadascú ha fet la seva part sempre i no hem faltat a cap entrega per molt apretats de feina que anéssim, per tant, bona feina. I un agraïment especial a la Laura Pascual, becària de l'assignatura, la qual ens ha ajudat a tots amb dubtes i a més ens ha ajudat a entendre coses que no ens havien quedat clares.

Per acabar, dir que hem après molt durant el curs i estem molt contents de la feina que hem fet, tot i que sempre es pot millorar, podem dir que ara mateix disposem dels coneixements bàsics amb pràctica per poder implementar bases de dades en PostgreSQL, MySQL, SQLite i Neo4j gràcies a la pràctica general i a les activitats d'avaluació continuada realitzades durant el curs.