Titulación: GRADO INGENIERIA INFORMATICA Año Académico: 2019/2020 -- Curso: 2º Asignatura: Ficheros y Bases de Datos Título: Memoria Práctica 2 – Consultas, vistas y disparadores	uc3m uc3m | Universidad Carlos III de Madrid

Profesor:	FRANCISCO JAVIER CALLE GOMEZ	**Grupo**	81
Alumno/a:	JORGE RODRÍGUEZ FRAILE	**NIA:**	100405951
Alumno/a:	CARLOS RUBIO OLIVARES	**NIA:**	100405834

1. Introducción

El objetivo de esta práctica es, a partir de una base de datos ya creada y estructurada, crear diferentes elementos que nos ayuden a operar con ella y que sea más fácil acceder a diferentes partes de la misma.

Para este cometido utilizaremos elementos como vistas, consultas y triggers. Es importante recalcar que estos cambios también modificarán la manera en la que se insertarán datos futuros, sobre todo gracias a la implementación de los disparadores.

2. Consultas

Tragicomic:

a. A $\equiv \pi_{title,director}(\sigma_{title,director}(genre = 'Drama'))\ (Genres\_movies)$

B $\equiv \pi_{title,director}(\sigma_{title,director}(genre = 'Comedy'))\ (Genres\_movies)$

$\pi_{actor}\ (Cast\ *_{title,director}\ (A\ *_{title,director}\ B))(Casts)$

b.

```
1. WITH A AS (SELECT title, director FROM genres_movies WHERE
   genre='Drama'),
2. B AS (SELECT title, director FROM genres_movies WHERE genre='Comedy')
3. SELECT DISTINCT actor FROM casts JOIN (SELECT title, director FROM A
   JOIN B USING (title, director)) USING (title, director);
```

c. Pruebas:

Primero hemos probado por separado si se muestran correctamente las películas de drama (A) y las de comedia (B), comprobando si un conjunto de películas cumplía dicho criterio.

Para seguir con las pruebas hemos comprobado si el join de A y B funcionaba mostrando todas las tragicomedias utilizando el mismo método.

Para finalizar, hemos hecho el último join para mostrar los actores y comprobar que la consulta funcionaba en su totalidad.

Burden-user:

a) $C \equiv \pi_{nick}\ (Membership)$

$D \equiv \pi_{citizenID}\ (Contracts)$

$\pi_{nick,reg_{date},citizenID}\sigma_{months\_between(sysdate,reg\_date)>=6,citizenID\ IS\ NULL\ OR\ NOT\ IN\ D,nick\ NOT\ IN\ C}$

$(Users \mid * Profiles)$

b)

```
1. SELECT
2.     nick,
3.     reg_date,
4.     citizenID
5. FROM
6.     users
7.     LEFT JOIN profiles USING (nick)
8. WHERE (months_between (sysdate, reg_date) >= 6
9.     AND nick NOT IN (
10.             SELECT
11.                 nick
12.             FROM
13.                 membership)
14.             AND (citizenID IS NULL
15.                 OR citizenID NOT IN (
16.                     SELECT
17.                         citizenID
18.                     FROM
19.                         contracts)));
```

c) Pruebas:

Para probar esta consulta lo que hicimos fue imprimir además del nombre, citizenID y reg_date, la fecha actual y de esta manera podremos contrastar que entre la fecha de registro y la actual haya 6 meses. Además, buscamos un sujeto que cumpliese esas condiciones, más de 6 meses registrado, que no haya contratado nunca un servicio y sin clubs, el sujeto elegido fue el usuario con nick braulito. Y ese usuario con un where aparecía. Otra prueba fue buscar un usuario que no llevase 6 meses, perteneciera a un club o que hubiese tenido un contrato, esos usuarios ha sido kiki y 60386681C. Ambos miembros si los limitábamos con un where en la consulta no aparecían.

FilmMaster:

a) $\pi_{rownum=1} \top_{avg(c)} (\pi_{dic,avg(c)}(\pi_{a.director\ AS\ dic,a.title,count('x')\ AS\ c}(Movies\ AS\ a) \Phi_{a.director=b.director} (Comments\ AS\ b)\ \Gamma_{a.director,a.title})\ \Gamma_{dic}$

b)

```
1. SELECT
2.     dic,
3.     avg(c)
4. FROM (
5.     SELECT
6.         a.director dic,
7.         a.title,
8.         COUNT('x') c
9.     FROM
10.             movies a
11.             JOIN comments b ON (a.director = b.director)
12.         GROUP BY
13.             a.director,
14.             a.title)
15.   GROUP BY
16.       dic
17.   ORDER BY
18.       avg(c)
19.       DESC fetch FIRST ROW ONLY;
```

c) Pruebas:

Hemos obtenido como resultado el director Woody Allen con una media de 750 películas, para comprobar que el resultado es correcto, simplemente hemos accedido en la base de datos a la file de Woody Allen y hemos hecho manualmente las operaciones necesarias, al obtener los mismos resultados y al comparar con la tabla resultante sin elegir solo el primero, hemos confirmado que, en efecto, Woody Allen es el director con más comentarios por película.

3. Vista

Leader:

a. $\pi_{first\ 10,D.member,club,\frac{Z}{M}AS\ media-comentarios}(\pi_{A.member,B.club,count\ ('x')AS\ Z}\ ((Proposals\ AS\ A\ \Phi_{A.title=B.title\ AND\ A.director\ =B.director}\ Comments\ AS\ B\ )\ AS\ C\ \Phi_{C.member\ =\ D.member}\ (\pi_{member,count('x)AS\ M}\ (Proposals)\ \mathcal{G}_{member}))$

b.

```
1. CREATE VIEW LEADER (nickname, club, media_comentarios) AS
2.     SELECT D.member, club, Z/M media_comentarios FROM (
3.     SELECT A.member, B.club, COUNT ('x') Z FROM PROPOSALS A JOIN
   COMMENTS B ON (A.title = B.title AND A.director = B.director) GROUP BY
   (A.member, B.club)) C JOIN
4.     (SELECT member, COUNT('x') M FROM PROPOSALS GROUP BY (member)) D
   ON (C.member = D.member)
5.     ORDER BY (media_comentarios) DESC
6.     FETCH FIRST 10 ROWS ONLY;
```

c. Pruebas:

La prueba básica es ver si la vista está bien implementada y tiene todos los elementos que les corresponde. Esto se comprueba simplemente haciendo un select y viendo si unos cuantos usuarios concretos cumplen los requisitos que hemos puesto en la vista. Esta prueba se cumple, por lo que podemos pasar a la siguiente.

En lo que respecta a la inserción, hemos elegido a un usuario llamado martillo y le hemos incrementado la media de proposals para ver si se insertaba en la vista. No se ha podido insertar ya que en efecto este usuario no cumple los requisitos por mucho que nosotros le incrementemos la media mediante un insert.

insert into leader values('martillo', 'The Sorority of Hamlet or all', '1001');

En lo que respecta al borrado en la vista, intentamos borrar al usuario ecl, que forma parte de la vista, el resultado es que no nos deja eliminar dicha fila de la vista.

delete from leader where (nickname = 'ecl' AND club = 'Brotherhood of Best');

En cuanto a la modificación, tratamos de cambiar el nickname al usuario ecl por eclecl, pero no permite columnas virtuales, por lo que no realiza la operación.

UPDATE leader SET club='eclecl' WHERE (nickname ='ecl');

Captain Aragna:

a.

$A \equiv \pi_{member,count('x')AS\ N}(\sigma_{title,director\ member\ NOT\ IN\ (\pi_{title,director,nick}(Comments))})(Proposals)Ç_{member}$

$B \equiv \pi_{member,count('x')\ AS\ T}(Proposals)Ç_{member}$

$\pi_{member,(N/T)*100\ AS\ porcentaje}(A *_{member} B)$

b.

```
1. CREATE OR REPLACE VIEW CaptainAragna AS
2. WITH A AS (SELECT member, COUNT('x') N FROM Proposals WHERE (title,
   director, member) NOT IN (SELECT title, director, nick FROM comments)
   GROUP BY member),
3. B AS (SELECT member, COUNT('x') T FROM proposals GROUP BY member)
4. SELECT member, (N/T)*100 Porcentaje FROM A JOIN B USING (member);
```

c. Pruebas:

La consulta funciona correctamente, probamos con un usuario del que hemos mirado sus comentarios y propuestas, y el resultado es el esperado.

select * from CaptainAragna where member='alc';

No es posible insertar, modificar o borrar un registro de la vista, ya que la vista necesita muchos datos que no muestra y que el usuario no le proporciona. Supondría muchas operaciones y no realiza la operación.

insert into CaptainAragna values('adori','75');

update CaptainAragna set porcentaje='25' where member='alc';

delete from CaptainAragna where member='hini';

4. Tablas Históricas

View clubs_vigentes y pertenencia_vigentes

a. $\pi_*(\sigma_{end-date\ is\ null}(Clubs))$

$\pi_*(\sigma_{club\ IN\ name}(Clubs\_vigentes))$ (Membership)

b.

```
1. CREATE VIEW clubs_vigentes AS SELECT * FROM clubs WHERE (end_date IS
   NULL);
2. CREATE OR REPLACE VIEW pertenencia_vigente AS SELECT * FROM membership
   WHERE (club IN (SELECT name FROM clubs_vigentes));
```

c. Pruebas:

Primero probamos si están todos los elementos que se corresponden a esta vista, simplemente hacemos un select con la condición de que el club tenga una fecha de cierre null (esta columna de fecha se ha insertado para la prueba, ya que para esta vista no tiene sentido almacenar fecha de cierre), como no se enseña ninguna fila, asumimos que está correcto.

select * from clubs_vigentes where end_date is not null;

Probamos a consultar qué club que aún está abierto ha fundado el usuario berti, y el resultado es el esperado berti fundó el club The Soldier's Community el 29/03/19.

select * from clubs_vigentes where founder='berti';

En lo que respecta a la inserción, hemos insertado un club con fecha de cierre, se inserta correctamente, aunque no se haga la inserción en la vista, si no en la tabla clubs debido a que salta el trigger.

insert into clubs_vigentes values('The Trimph Stadium', 'aranz', '07/02/19', null, 'Wanna join us?', 'O');

Probando la eliminación, elimina de la manera que se espera, no en la vista, si no que pasa a ser un club histórico.

delete from clubs_vigentes where name='The Trimph Stadium';

Por último, en la actualización nos deja modificar dependiendo del atributo que se modifique, si por ejemplo se modifica el nombre del club, dará error ya que las membresías desaparecerían, pero si cambiamos slogan, se modifica sin problema.

update clubs_vigentes set slogan='AAAAAAA' where slogan='Lots of movie talk and fun. ' and founder='aaj';

Para probar la consulta de pertenencia vigente, escogimos unos cuantos clubes de vigentes e históricos y miramos que no aparecieran los miembros de los históricos, pero sí de los vigentes.

select * from pertenencia_vigente where club='Alliance of Hamlet';

En cuanto a la inserción

insert into pertenencia_vigente values('jaime1', 'Alliance of Hamlet','carti','I', to_date('24/03/19','DD/MM/YY'),to_date('27/03/19','DD/MM/YY'), null, 'Wanna join?','Wanna join?');

delete from pertenencia_vigente where nick='jaime1' and club='Alliance of Hamlet';

Update pertenencia_vigente set acc_msg='He entrado' where nick='aldunate' and club='Alliance of Hamlet';

View clubs_historicos y pertenencia_historica

a. $\pi_*(\sigma_{end-date\ is\ not\ null}(Clubs))$
$\pi_*(\sigma_{club\ IN\ name}(Clubs\_historicos))$ (Membership)

b.

```
1. CREATE VIEW clubs_historicos AS SELECT * FROM clubs WHERE (end_date IS
   NOT NULL);
2. CREATE OR REPLACE VIEW pertenencia_historica AS SELECT * FROM
   membership WHERE (club IN (SELECT name FROM clubs_historicos));
```

c. Pruebas:

Las pruebas hechas en este club son las mismas hechas en la anterior, y se obtienen exactamente los mismos resultados, excepto en la inserción, debido a que el disparador creado salta.

insert into clubs_historicos values('The Trimph Stadium', 'aranz', '07/02/19', '17/02/19', 'Wanna join us?', 'O');

delete from clubs_historicos where name='The Trimph League';

update clubs_historicos set slogan='AAAAAAA' where slogan='Wanna join us?' and founder='aranz';

En cuanto a la vista pertenencia_historica, se puede insertar, borrar y actualizar, como se ha comprobado con las siguientes querys:

insert into pertenencia_historica values('felico', 'Need Club','esmeralda','I', to_date('24/03/19','DD/MM/YY'),to_date('27/03/19','DD/MM/YY'), null, 'Wanna join?','Wanna join?');

delete from pertenencia_historica where nick='felico' and club='Need Club';

Update pertenencia_historica set acc_msg='He entrado' where nick='cagnoaldo' and club='The Club of Begin';

Trigger Borrar Clubs_historicos

a. Cambiamos la acción de borrado en la vista de clubs_historicos por una de borrado en la tabla clubs, para que no se rompa la integridad deben también borrarse todos los usuarios que sean candidatos a ese club, además de los propios miembros, donde identificamos la fila a borrar mediante :OLD.name

b.

```
1. CREATE OR REPLACE TRIGGER TG_delete_historicos instead OF DELETE ON
   clubs_historicos
2.   BEGIN
3.     DELETE FROM candidates WHERE (club = :OLD.name);
4.     DELETE FROM membership WHERE (club = :OLD.name);
5.     DELETE FROM clubs WHERE(name=:OLD.name);
6.   END TG_delete_historicos;
```

c. Ejecutando el mandato: delete from clubs_historicos where name='The Trimph League' and founder='aranz'; que trata de eliminar un club histórico, en vez de eliminar en la view, elimina en la tabla de clubs.

Para probar que no de error de tabla mutante en el borrado probamos a introducir clubes que sean históricos mediante clubs, para después eliminarlos a todos con un mismo mandato(Los introducidos son los clubes actuales, pero con el prefijo Prueba).

insert into clubs select concat('Prueba',name), founder, cre_date, sysdate, slogan, open from clubs;

delete from clubs_historicos where name like 'Prueba%';

Trigger No permitir insertar en Clubs_historicos

a. En este disparador, para evitar que se inserten en históricos levantamos una excepción cada vez que se intente ejecutar una inserción. Para mostrar el mensaje de la excepción utilizamos raise_application_error, con código -20001, ya que los errores de usuario empiezan en -20000.

b.

```
1. CREATE TRIGGER TG_not_insert_historicos instead OF INSERT ON
   clubs_historicos
2.   DECLARE
3.     notInsert Exception;
4.   BEGIN
5.     raise notInsert;
6.   EXCEPTION
7.     WHEN notInsert
8.       THEN raise_application_error(-20001, 'No se puede insertar en
   historicos');
9.     END TG_not_insert_historicos;
```

c. Pruebas:

insert into clubs_historicos values('The Trimph Stadium', 'aranz', '07/02/19', '17/02/19', 'Wanna join us?', 'O');

Tras este mandato no inserta el club en la vista, nos salta la excepción ‘ No se puede insertar en históricos’ que es el resultado esperado.

Trigger Borrar en Clubs_vigentes

a. En vez de borrar en la vista clubs_vigentes, lo que hace es ponerle una fecha de cierre al club que se está intentado borrar, de esta manera sale de la vista clubs_vigentes y pasa a estar en la vista clubs_historicos

b.

```
1. CREATE TRIGGER TG_delete_vigentes
2.   instead OF DELETE ON clubs_vigentes
3.   BEGIN
4.     UPDATE clubs SET end_date=sysdate WHERE (name =:OLD.name);
5.   END TG_delete_vigentes;
```

c. Pruebas:

Como se ven en las pruebas anteriores, se actualiza el club, no se borra.

Para probar que no error de tabla mutante al eliminar múltiples columnas lo que hacemos es introducir todos los clubes de clubs con el prefijo Prueba y sin fecha de cierre en la tabla de clubes vigentes y después los eliminamos buscando que el nombre comience con Pruebas... . Los mandatos ejecutados son:

insert into clubs_vigentes select concat('Prueba',name), founder, cre_date, null, slogan, open from clubs;

delete from clubs_vigentes where name like 'Prueba%';

delete from clubs_historicos where name like 'Prueba%';

5. Diseño Externo

Vista OpenPub, actividad clubes, solo accesible para usuarios registrados

a.

$$\pi_{name,num-users,floor(month-between\ (sysdate,cre-date))\ AS\ months,\frac{propo}{floor(months-between(sysdate,cre-date))}\ AS\ avg_{prop}}\ \pi_{\frac{num_{comment}}{propo}}(((club\_vigentes\ *_{name}\ (\pi_{club,name,count('x'),num-users}\sigma_{club\ IN\ (\pi_{name}\ (clubs-vigentes))}(Membership)\ G_{club})\ *_{name}\ (\pi_{club,name,count('x')AS\ propo}\ (Proposals)G_{club}))\ *_{name}\ (\pi_{club,name,count('x')\ AS\ num-comment}\ (Comments)\ G_{club}))$$

b.

```
1. CREATE VIEW OPENPUB AS
2.     SELECT name, NUM_USERS, FLOOR(MONTHS_BETWEEN(sysdate,
   cre_date)) MONTHS, PROPO/FLOOR(MONTHS_BETWEEN(sysdate,cre_date))
   AVG_PROP, NUM_COMMENT/PROPO COMMENT_PER_PROP
3.     FROM ((CLUBS_VIGENTES JOIN (SELECT CLUB NAME ,COUNT('X')
   NUM_USERS FROM MEMBERSHIP WHERE (CLUB IN (SELECT NAME FROM
   CLUBS_VIGENTES)) GROUP BY CLUB) USING (NAME))
4.      JOIN (SELECT CLUB NAME, COUNT('X') PROPO FROM PROPOSALS GROUP
   BY (CLUB))USING (NAME))
5.      JOIN (SELECT CLUB NAME, COUNT('X') NUM_COMMENT FROM COMMENTS
   GROUP BY (CLUB)) USING (NAME);
```

c. Pruebas:

Para ver si se han insertado correctamente los elementos de la vista, hacemos la siguiente query de select y comprobamos sobre algunas filas si se cumplen los requisitos dados, con todas las columnas que se pedían. El resultado es positivo, por lo que podemos avanzar a la siguiente prueba.

select * from openpub where name='The Club of the Patio';

Para la inserción, como es obvio, no nos permite insertar valores directamente en la view.

insert into openpub values('fifo','15','16','17','18');

En cuanto a la eliminación, teniendo en cuenta lo visto anteriormente, no debería de dejar hacerlo, probamos con la siguiente query, y , en efecto, nos salta error.

delete from openpub where name='Friendship of Patio';

Y para la actualización obtenemos el mismo resultado, no podemos alterar la vista.

update openpub set name='AAAAAAAFriendship of Patio' where name='Friendship of Patio';

Vista Anyone_Goes, top clubes que aceptan más solicitudes, solo accesible para usuarios registrados

a. $\sigma_{first(1)}(\pi_{club\ AS\ name,count('x')AS\ num_{solicitudes}}\sigma_{type='I',rej_{date}IS\ NULL}\ {}_{club}\mathcal{G}_{count('x')}(Candidates))$

b.

```
1. CREATE VIEW ANYONE_GOES AS
2. SELECT club name, COUNT('x') NUM_SOLICITUDES FROM candidates WHERE
   (TYPE = 'I' AND REJ_DATE IS NULL) GROUP BY club
3. ORDER BY COUNT('X') DESC
4. FETCH FIRST 5 ROWS ONLY;
```

c. Pruebas:

Para comprobar los datos, hacemos la siguiente query, donde los datos que hemos introducido tendrían que cumplir supuestamente las condiciones de la view y estar ahí, el resultado es ese, por lo que pasa esta prueba.

select * from anyone_goes where name='How Community ';

Pasando a la inserción, no nos deja insertar, al igual que en la anterior view.

insert into anyone_goes values('NombClub','15');

Con el borrado obtenemos el mismo resultado con la inserción.

delete from anyone_goes where name='How Community';

Por último, la actualización nos vuelve a retornar un error.

update anyone_goes set name='AAAAAAAHow Community' where name='How Community';

Vista Report, solo accesible para usuarios registrados

a. $\pi_{name,founder,cre-date,end-date,slogan,open}(Clubs\ *_{name}\ (\pi_{nick,club,name}((\pi_{nick,club}(\sigma_{nick=user})\ (Membership))\cup\ (\pi_{nick,club}\ (\sigma_{nick=user})\ (Candidates)))$

b.

```
1. CREATE  VIEW REPORT AS
2. SELECT DISTINCT name, founder, cre_date, end_date, slogan, OPEN
3. FROM clubs
4.     JOIN
5.     (SELECT DISTINCT nick, club name
6.     FROM
7.         (SELECT nick, club FROM membership WHERE (nick = USER))
8.         UNION
9.         (SELECT nick, club FROM candidates WHERE (nick = USER))
10.          ) USING (name);
```

c. Pruebas:

Haciendo la prueba de la consulta, utilizamos el mismo modus operandi que en las anteriores vistas, obtenemos el resultado esperado.

```
select * from report;
```

Las opciones de borrado, insertado y actualización no son permitidas.

Inserción:

```
insert into report values('nombUser','Funda',to_date('10/01/19', 'DD/MM/YY'), to_date('13/01/19', 'DD/MM/YY'),'Ven','O');
```

Borrado:

```
delete from report where open='C';
```

Actualización:

```
update report set name='ramon' where name=user;
```

Role usuarios_registrados

a. Creamos un rol con un nombre representativo, usuarios_registrados, y que no requiere contraseña.

b.

```
1. CREATE ROLE usuarios_registrados NOT IDENTIFIED;
```

c. Pruebas:

Nos creamos un usuario y le concedemos este rol, y vemos si en efecto se le otorgan los permisos de las vistas.

Conceder permiso a usuarios_registrados

a. Le concedemos acceso a OpenPub, Anyone_Goes y Report al role usuarios_registrados, de tal manera que los que tengan esos permisos puedan acceder a las mismas.

b.

```
1. GRANT SELECT ON openpub TO usuarios_registrados;
2. GRANT SELECT ON any_goes TO usuarios_registrados;
3. GRANT SELECT ON REPORT TO usuarios_registrados;
```

6. Disparadores

Application

a. Este disparador se activará antes de hacer un insert en la tabla candidates, y con una granularidad de for each row. Primero definimos una excepción y un booleano, este último nos servirá para ver si el club al que intentamos acceder está cerrado o no. Una vez hecho esto, si nuestra acción es una application y l_exist no es nulo significa que estamos intentando aplicar a un club cerrado, por lo que levantamos la excepción. En los demás casos no hacemos nada y simplemente dejamos que se inserte normalmente.

b.

```
1. CREATE OR REPLACE TRIGGER APPLICATION BEFORE INSERT ON candidates FOR
   EACH ROW
2. DECLARE
3.     notInsert Exception;
4.     l_exist varchar2(1);
5.     BEGIN
6.     SELECT MAX('Y') INTO l_exist
7.             FROM CLUBS
8.             WHERE (name = :NEW.club AND OPEN = 'C');
9.
10.          IF :NEW.type = 'A' THEN
11.              IF (l_exist IS NOT NULL) THEN
12.                  raise notInsert;
13.              END IF;
14.              END  IF;
15.          EXCEPTION
16.          WHEN notInsert
17.            THEN raise_application_error(-20002, 'No se puede
   solicitar acceso a un club cerrado sin invitacion');
18.          END;
```

c. Pruebas:

Salta el disparador:

insert into candidates (nick, club, type, req_date, req_msg, rej_date, rej_msg) values ('cmp', 'Guild of the Heart geraniums', 'A', to_date('10/01/19', 'DD/MM/YY'), 'dkhwizxcgqrjrfbf mudevgkdkj chwuka.',to_date('13/02/19', 'DD/MM/YY'),'dkhwizxcgqrjrfbf mudevgkdkj chwuka.');

No salta el disparador:

insert into candidates (nick, club, type, req_date, req_msg, rej_date, rej_msg) values ('cmp', 'Guild of the Heart geraniums', 'I', to_date('10/01/19', 'DD/MM/YY'), 'dkhwizxcgqrjrfbf mudevgkdkj chwuka.', to_date('13/02/19', 'DD/MM/YY'),'dkhwizxcgqrjrfbf mudevgkdkj chwuka.');

Prueba de tabla mutante, insertamos múltiples candidatos para que pasen por el trigger, esperamos que los introduzca o salte error:

insert into candidates select nick, club, member, type, to_date('07/05/20','DD/MM/YY'), req_msg, to_date('08/05/20','DD/MM/YY'), rej_msg from candidates;

delete from candidates where req_date=to_date('07/05/20','DD/MM/YY');

Overwrite

a. Este disparador actuará sobre la tabla de Comments, antes de realizar una inserción se comprobará si ya existe un comentario para ese usuario y película, si es así, se borrará el comentario anterior para dar paso al comentario nuevo sobre la misma, y si no existía previamente se insertará con normalidad.

b.

```
1.    CREATE OR REPLACE TRIGGER OVERWRITE after INSERT ON comments
2.     BEGIN
3.         DELETE FROM comments WHERE((club, nick, title, director,
   msg_date) IN (SELECT club, nick, title, director, MIN(msg_date) fecha
   FROM (SELECT club, nick, title, director, msg_date FROM
4.     (SELECT club, nick, title, director, COUNT('x') cuenta
5.         FROM comments
6.         GROUP BY nick, club,title, director)
7.     JOIN
8.     comments USING (club, nick, title, director)
9.     WHERE cuenta>1)
10.          GROUP BY(club, nick, title, director)));
11.          END overwrite;
```

c. Pruebas:

Salta el disparador:

insert into comments values('Association of Skin', 'aabc', TO_DATE('2029-05-0718:45','YYYY-MM-DDHH24:MI'), 'Epic Movie', 'Jason Friedberg', 'Heaven want','Prueba','6');

No salta el disparador:

insert into comments values('The Group of Stork', 'aacg', TO_DATE('2010-05-0718:45','YYYY-MM-DDHH24:MI'), 'Epic Movie', '0Jason Friedberg', 'Taste and pictures','Prueba','6');

Prueba de tabla mutante: Tras un mandato de inserción masiva, todos los comentarios de aab, pero en la fecha 07/05/29, se queda solo con los comentarios más nuevos.

insert into comments select club, nick, to_date('07/05/29','DD/MM/YY'), title, director, subject, message, valoration from comments where nick='aab';

Supera la prueba.

Trigger No comments

a. Este trigger es un before insert con una granularidad for each row. Lo primero que hacemos es comprobar si existe alguna fila que tenga la misma fecha que la tupla que vamos a insertar. Si esto se cumple, simplemente cambiamos la msg_date de la fila a insertar sumándole un segundo. Hemos tenido que hacer 2 ya que tenemos que actuar sobre dos tablas diferentes.

b.

```
1. CREATE OR REPLACE TRIGGER NoCommentsC BEFORE INSERT ON Comments FOR
   each ROW
2. DECLARE
3.     l_exist varchar2(1);
4. BEGIN
5.     SELECT MAX('Y') INTO l_exist
6.             FROM COMMENTS
7.             WHERE (MSG_DATE = :NEW.MSG_DATE);
8.     IF(l_exist IS NULL) THEN
9.       :NEW.MSG_DATE := (:NEW.MSG_DATE + INTERVAL '1' SECOND);
10.         END IF;
11.     END;
12.
13.     CREATE OR REPLACE TRIGGER NoCommentsP BEFORE INSERT ON Proposals
   FOR each ROW
14.     DECLARE
15.         l_exist varchar2(1);
16.     BEGIN
17.         SELECT MAX('Y') INTO l_exist
18.                 FROM Proposals
19.                 WHERE (PROP_DATE = :NEW.PROP_DATE);
20.         IF(l_exist IS NULL) THEN
21.           :NEW.PROP_DATE := (:NEW.PROP_DATE + INTERVAL '1' SECOND);
22.         END IF;
23.     END;
```

c. Pruebas:

La inserción se lleva a cabo con éxito tras ejecutar los siguientes mandatos.

insert into comments values ('The Community of Guitar', 'aab', to_date('07/05/29','DD/MM/YY'), 'The Hours', 'Stephen Daldry', 'Jar and weakness', 'Prueba', '5');

insert into Proposals values ('The Hours', 'Stephen Daldry', 'The Community of Guitar', 'aab', to_date('17/06/19','DD/MM/YY'), 'Masterpiece', 'Mensaje');

Trigger Reachable Clients

a. Para este trigger, lo primero que hemos hecho es comprobar si el perfil que tiene el contrato que vamos a insertar tiene teléfono o no, una vez hecho esto comprobamos si nuestro contrato tiene enddate o no, si ambas condiciones se cumplen, lanzamos nuestra excepción.

b.

```
1. CREATE OR REPLACE TRIGGER ReachableClients BEFORE INSERT ON CONTRACTS
   FOR each ROW
2. DECLARE
3.     l_exist varchar2(1);
4.     notInsert Exception;
5. BEGIN
6.     SELECT MAX('Y') INTO l_exist
7.             FROM PROFILES
8.             WHERE (CITIZENID = :NEW.CITIZENID AND mobile IS NULL);
9.
10.         IF (l_exist IS NOT NULL AND :NEW.enddate IS NULL) THEN
11.           raise notInsert;
12.         END IF;
13.     EXCEPTION
14.         WHEN notInsert
15.           THEN raise_application_error(-20003, 'No puede haber un
   contrato vigente sin un numero de telefono asociado');
16.     END;
```

c. Pruebas:

Primero probamos a crear un contrato para un perfil que no tenga teléfono, que es en el caso en que debe saltar el disparador. Se ejecuta el mandato de a continuación y los resultados son los esperados, salta una excepción indicando que es necesario que tenga teléfono.

insert into contracts values('Test', '48649111V', 'Short Timer', to_date('28/10/15', 'DD/MM/YY'), null, '112 State Street, Third floor, door B', 'Springtown', '78201', 'Finland');

También se prueba en caso de que el perfil tenga un teléfono o que el contrato haya expirado, en estos casos no debe hacer nada y los inserta con normalidad.

insert into contracts values('Test', '61478893P', 'Short Timer', to_date('28/10/15', 'DD/MM/YY'), null, '112 State Street, Third floor, door B', 'Springtown', '78201', 'Finland');

delete contracts where idcontract='Test';

insert into contracts values('Test', '48649111V', 'Short Timer', to_date('28/10/15', 'DD/MM/YY'), to_date('28/10/16', 'DD/MM/YY'), '112 State Street, Third floor, door B', 'Springtown', '78201', 'Finland');

delete contracts where idcontract='Test';

7. Conclusiones

En este proyecto hemos conseguido hacer que todas nuestras implementaciones sean lo más efectivas posibles, intentando eliminar código innecesario o que no nos diera un resultado tan efectivo como deseáramos.

Uno de los apartados que más nos ha costado ha sido el de los disparadores, ya que había que tener en cuenta muchos factores a la hora de hacer cualquier tipo de operación dentro del disparador, sobre todo para evitar tablas mutantes, o que por ejemplo se creará una recursividad dentro del trigger. También hemos tenido dificultades en la parte de las consultas, ya que al empezar el proyecto no teníamos todavía la vista general del trabajo y de la base de datos, al conseguir tener esta perspectiva, el resto del trabajo ha resultado más llevadero.

El tiempo dedicado a este trabajo ha sido de unas 2-3 horas diarias durante 2 semanas, donde la gran mayoría del tiempo ha sido invertido en pensar una solución óptima más que codificar el problema. No nos hemos dividido el trabajo realmente, si no que entre los dos hemos intentado buscar una solución, implementando las ideas de ambos en estas.

De esta práctica hemos aprendido a crear elementos como triggers o views, además de tener una mayor seguridad a la hora de establecer la solución a un problema sobre una base de datos.