UNIZAR – UNIVERSIDAD DE ZARAGOZA

BASES DE DATOS

Universidad Zaragoza

DANIEL RUEDA
JORGE SANZ
ALVARO MONTEAGUDO

INGENIERIA INFORMATICA

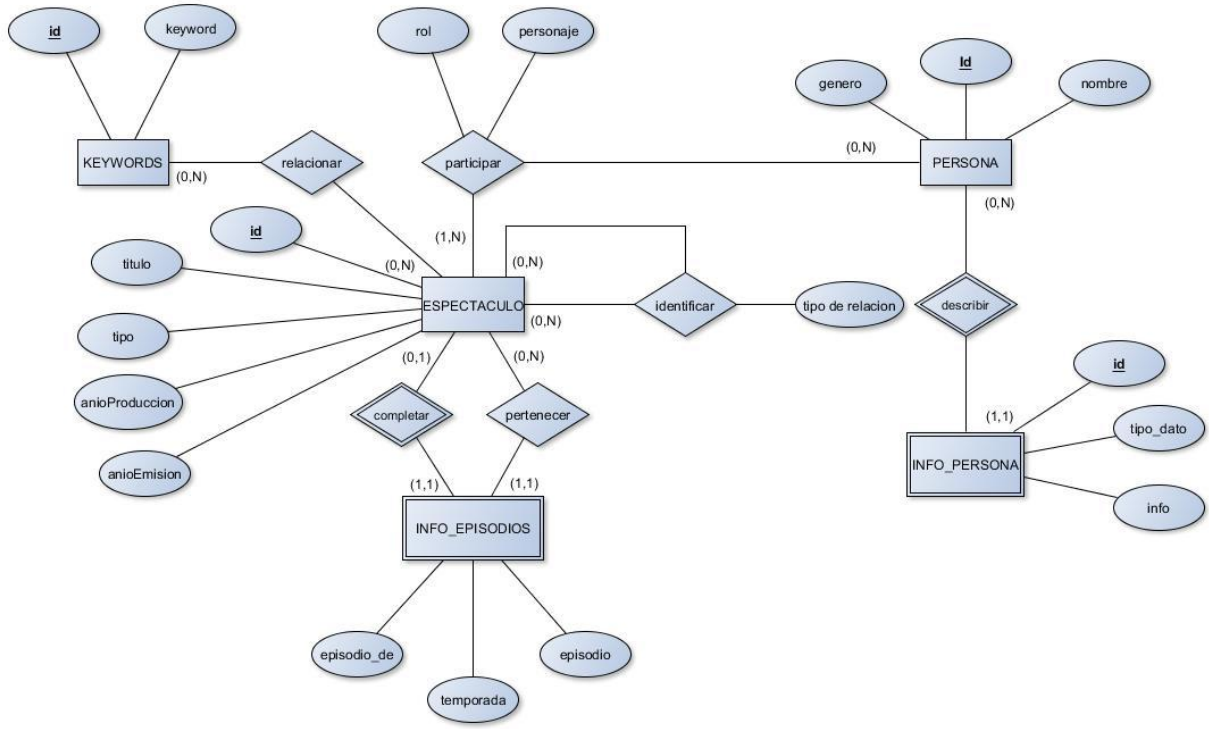
SEGUNDO AÑO

SEGUNDO CUATRIMESTRE

10/09/2015

PARTE 1: DISEÑO DE LA BASE DE DATOS

MODELO ENTIDAD RELACIÓN



El modelo entidad relación se basa en una entidad principal ESPECTACULO y entidades que completan información sobre los espectáculos guardados en la misma. La información básica de las personas se guarda en una entidad PERSONA, para guardar información extra de las personas tenemos una entidad débil de PERSONA, cada tupla de la entidad débil guarda el tipo y el texto de la información (ej. fecha de muerte, nacimiento…).

Cada espectáculo está identificado con una serie de palabras clave que se guardan en KEYWORDS, se trata de una relación N a M y por lo tanto se crea una tabla con las claves primarias del espectáculo y de la palabra clave.

ESPECTACULO puede ser desde una película, una serie o un episodio de una serie, por eso la entidad débil INFO EPISODIOS dependiente de ESPECTÁCULO tiene una doble interrelación.

La entidad ESPECTACULO tiene una auto-interrelación N a M, se genera otra tabla que guarda dos identificadores de espectáculos y el tipo de relación que hay entre ellos (ej. remakes, versiones, secuelas…).

Entre PERSONA y ESPECTACULO tenemos otra cardinalidad N a M, la tabla generada guardará tanto el id de la película como el de la persona así como el rol que cumplió en la peli y una cadena con el personaje que interpreto en caso de que su rol fuese actriz o actor.

RESTRICCIONES DEL MODELO

Personaje decidimos que sea una cadena no nula, a pesar de que haya datos de diferentes roles que no sean actor o actriz porque a la hora de insertar datos había muchísimos actores que tenían varios papeles en una misma película y necesitábamos un identificador extra, para resolver el problema, incluimos personaje en la clave primaria y hicimos una sustitución de todos los personajes = NULL por una cadena' No es personaje' para no tener problemas de inserción. La clave primaria está formada por todos los atributos de la tabla.

Los datos no deben tener acentos, diéresis o & para no tener errores de inserción.

El nombre de las personas no puede ser NULL, toda persona incluida en la base de datos tiene que tener un nombre, independientemente de cuanta o que información se guarde de la misma. Género solo puede tomar dos valores: m para masculino y f para femenino.

Al igual que nombre de persona, el titulo y el tipo de cada espectáculo no pueden ser NULL.

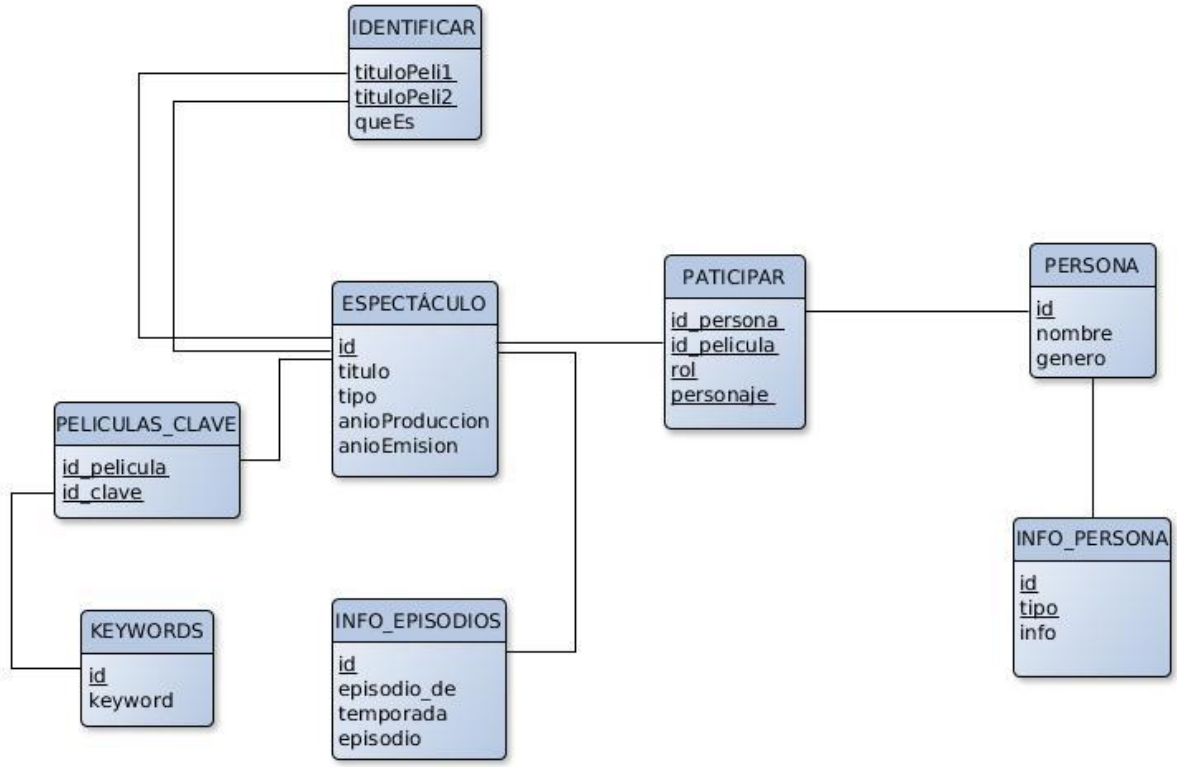
Todas las tuplas de INFO EPISODIOS deben tener el campo episodio de distinto de NULL pues es el identificador de la serie a l que pertenece el episodio (será un id existente de la tabla ESPECTACULO).

No hay ninguna tupla en IDENTIFICAR que tenga tipo de relación a NULL, si existe la relación entre dos espectáculos tiene que tener un tipo definido.

NORMALIZACIÓN

Esta en primera formal porque no tiene atributos multivaluados, también está en segunda porque cualquier atributo no primario de las tablas depende completamente de la clave primaria de la misma, así como ningún atributo primario depende parcial o totalmente de otro atributo primario por lo que también cumple la tercera forma normal. No existen dependencias multivaluadas por lo que cumple la cuarta forma normal y por lo tanto la forma normal de Boyce Codd también, una forma normal se cumple siempre si demuestras que la siguiente se cumple para tu modelo.

MODELO RELACIONAL



CREACIÓN DE TABLAS

```
CREATE TABLE PERSONA (
id NUMBER(11) PRIMARY KEY,
nombre CHAR(40) NOT NULL,
genero CHAR(1)
);

CREATE TABLE INFO_PERSONA (
id NUMBER(11),
tipo_dato NUMBER(3) NOT NULL,
info CHAR(120),
FOREIGN KEY (id) REFERENCES PERSONA(id),
PRIMARY KEY (id,tipo_dato)
);

CREATE TABLE ESPECTACULO (
id NUMBER(11) PRIMARY KEY,
titulo CHAR(100) NOT NULL,
tipo NUMBER(1) NOT NULL,
anioProduccion CHAR(5),
anioEmision CHAR(10)
);
```

```
CREATE TABLE INFO_EPISODIOS (
id NUMBER(11),
FOREIGN KEY (id) REFERENCES ESPECTACULO(id),
PRIMARY KEY (id),
episodio_de NUMBER(11) NOT NULL,
temporada NUMBER(2),
episodio NUMBER(3)
);

CREATE TABLE IDENTIFICAR(
peliculaOriginal NUMBER(11) NOT NULL,
peliculaRelacionada NUMBER(11) NOT NULL,
tipo_relacion NUMBER(2) NOT NULL,
FOREIGN KEY (peliculaOriginal) REFERENCES ESPECTACULO(id),
FOREIGN KEY (peliculaRelacionada) REFERENCES ESPECTACULO(id),
PRIMARY KEY (peliculaOriginal,peliculaRelacionada,tipo_relacion)
);

CREATE TABLE KEYWORDS(
id NUMBER(11) PRIMARY KEY,
keyword CHAR(100), /*palabras clave*/
PRIMARY KEY (id,keyword)
);

CREATE TABLE PELICULAS_CLAVE(
id_pelicula NUMBER(11),
id_palabraClave NUMBER(11),
FOREIGN KEY (id_pelicula) REFERENCES ESPECTACULO(id),
FOREIGN KEY (id_palabraClave) REFERENCES KEYWORDS(id)
);

CREATE TABLE PARTICIPAR(
id_persona NUMBER(11),
id_pelicula NUMBER(11),
rol NUMBER(13),
personaje CHAR(100),
FOREIGN KEY (id_persona) REFERENCES PERSONA(id),
FOREIGN KEY (id_pelicula) REFERENCES ESPECTACULO(id),
PRIMARY KEY (id_persona,id_pelicula,rol,personaje)
);
```

PARTE 2: POBLACIÓN BASE DE DATOS Y CONSULTAS

Para realizar las consultas a la base de datos miniIMDB decidimos utilizar un canal ssh desde el que ejecutar un comando en la base de datos y redireccionar la salida a un fichero que nosotros pudiéramos modificar, con los ficheros resultantes aplicábamos un filtro para eliminar caracteres indeseados o que generan problemas con SQL, en nuestro caso, tildes, &, comillas simples y alguno más. Todos las tildes se sustituían por la vocal sin tilde, los & por and y las comillas simples eran eliminadas para facilitar lo máximo posible la inserción. Uno vez depurados los datos, pasamos los ficheros por un programa en JAVA en el que escribamos las sentencias SQL de inserción en un fichero con extensión .SQL listo para insertar.

ESTADÍSTICAS DE LAS TABLAS

ESPECTACULO: 5680 tuplas, 445 Kb. Tuplas de ejemplo:

```
      ID
----------
TITULO
--------------------------------------------------------------------------------
    TIPO ANIOPRODUCCION  ANIOEMISION
---------- --------------- ------------------------------
      961
003 y medio
       2 1979          1979-1980
    1612
10 + 2
      ID
----------
TITULO
--------------------------------------------------------------------------------
    TIPO ANIOPRODUCCION  ANIOEMISION
---------- --------------- ------------------------------
       2 1994          1994-2004
    4617
13 x 13
       2 1987          1987-1988
      ID
----------
TITULO
--------------------------------------------------------------------------------
    TIPO ANIOPRODUCCION  ANIOEMISION
---------- --------------- ------------------------------
    4627
Stress de primavera
       7 1989
    4644
```

PERSONA: 38978 tuplas, 2292 kB. Tuplas de ejemplo:

```
ID
----------
NOMBRE
--------------------------------------------------------------------------------
GEN
---
       31
 El Frances , Jose
m

       33
 El Gato , Felix
m

       ID
----------
NOMBRE
--------------------------------------------------------------------------------
GEN
---

      734
Aames, Willie
m

      882
Aaron, Jack

       ID
----------
NOMBRE
--------------------------------------------------------------------------------
GEN
---
m

     1135
Abad de Santillan, Diego
m
```

INFO_PERSONA: 22735 tuplas, 1504kB. Tuplas de ejemplo:

```
ID  TIPO_DATO
---------- ----------
INFO
--------------------------------------------------------------------------------
      31        20
Montpellier, Herault, France


      31        21
1971


  2894729        20

      ID  TIPO_DATO
---------- ----------
INFO
--------------------------------------------------------------------------------
Valladolid, Valladolid, Castilla y Leon, Spain


  2894729        21
1971


      734        20
Los Angeles, California, USA

      ID  TIPO_DATO
---------- ----------
INFO
--------------------------------------------------------------------------------

      734        21
15 July 1960


      882        20
New York City, New York, USA
```

PARTICIPAR: 97011 tuplas, 6305kB. Tuplas de ejmplo:

```
ID_PERSONA ID_PELICULA       ROL
---------- ----------- ----------
PERSONAJE
--------------------------------------------------------------------------------
      33    1584953         1
Himself

    1135    1999660         1
Himself

    1135    2465724         1

ID_PERSONA ID_PELICULA       ROL
---------- ----------- ----------
PERSONAJE
--------------------------------------------------------------------------------
Himself

    6003    2387806         1
Himself

   13011    1935123         1
Himself
```

KEYWORDS: 132716 tuplas, 7197 kB. Tuplas de ejemplo:

```
ID
----------
KEYWORD
--------------------------------------------------------------------------------
       1
number-in-title

       2
twitter-hashtag-in-title
       3
      ID
----------
KEYWORD
--------------------------------------------------------------------------------
censored-profanity-in-title

       4
father

       5
misanthrope
```

IDENTIFICAR: 370 tuplas, 19 kB. Tuplas de ejemplo:

PELICULAORIGINAL	PELICULARELACIONADA	TIPO_RELACION
90773	1811428	6
125308	1811840	6
125308	2032575	6
125308	2033484	6
125308	2046499	6
175875	175888	1

INFO_EPISODIOS: 162 tuplas, 10kB. Tuplas de ejemplo:

ID	EPISODIO_DE	TEMPORADA	EPISODIO
4627	4617		
77581	77577	1	1
196402	196397	5	12
196409	196397	2	4
196410	196397	2	8
196411	196397	1	4
196412	196397	6	2

PELICULAS_CLAVE: 22760 tuplas, 1134 kB. Tuplas de ejemplo:

ID_PELICULA	ID_PALABRACLAVE
961	1
1612	636
1612	637
1612	638
1612	639
1612	1
1612	640

Las consultas realizadas a la base de datos proporcionadas son:

```
/*Personas*/
SELECT *
FROM name;

/*Información de las personas*/
SELECT *
FROM person_info;

/*Obtiene información de los espectáculos excluyendo la información extra en el caso de que sean episodios de serie*/
SELECT id,title,kind_id,production_year,series_years
FROM title;

/*Muestra la información extra de los episodios*/
SELECT id,episode_of_id,season_nr,episode_nr
FROM title
WHERE kind_id=7;

/* Películas y relación con otras */
SELECT *
FROM movie_link;

/* Palabras clave de películas */
SELECT *
FROM keyword;

/*Tabla que se crea a partir de una relación N a M entre palabras clave y películas*/
SELECT *
FROM movie_keyword;

/* Sentencia para conseguir los datos de la tabla participar de nuestro modelo */
SELECT P.person_id,P.movie_id,P.role_id,Q.name
FROM cast_info P,char_name Q                    /* Actores y actrices, personaje ¡= NULL */
WHERE P.person_role_id=Q.id
UNION
SELECT person_id, movie_id, role_id, NULL       /* Los que no son actores ni actrices*/
FROM cast_info;
```

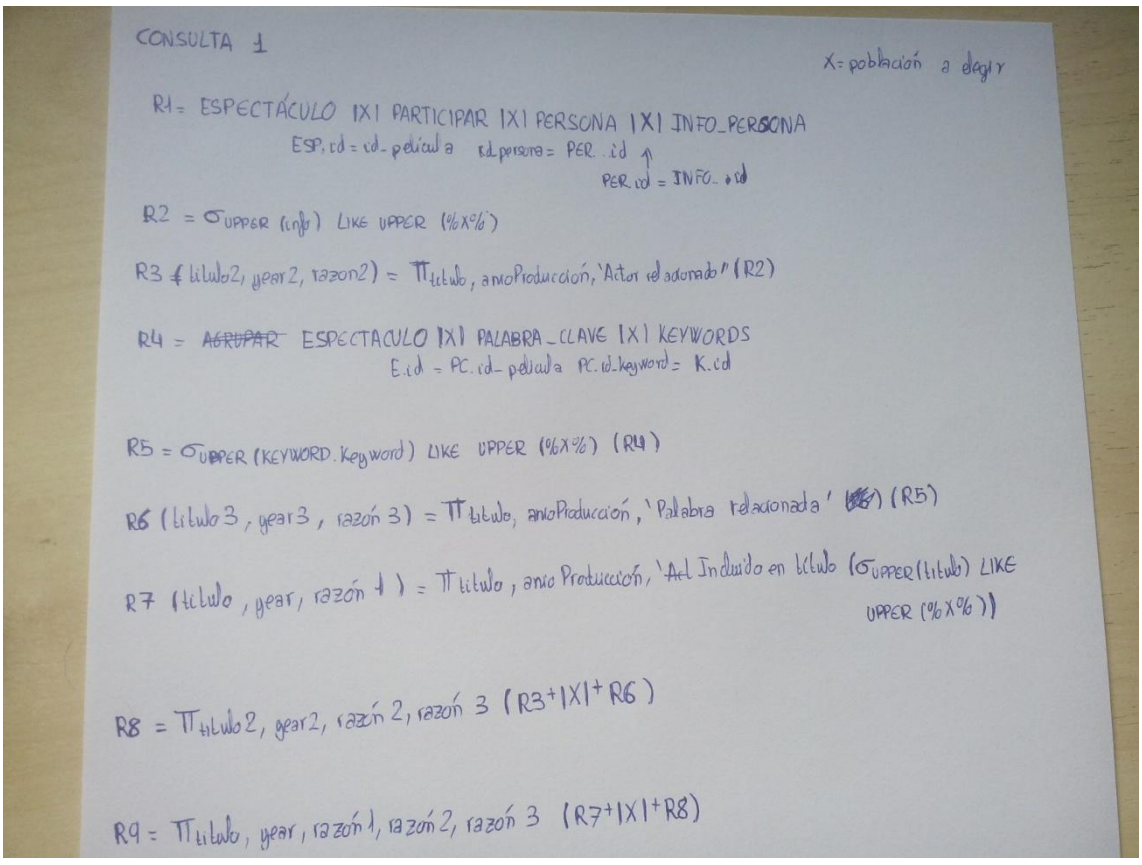
CONSULTAS

CONSULTA OBLIGATORIA 1

Dado el nombre de cualquier población del mundo, busca las películas relacionadas con esa población, ya sea en su título, por el lugar de nacimiento de alguna de las personas que participan en esa película o que se comente algo de esa ciudad en los keywords de la película. La respuesta incluirá el título de la película, el año, y la causa por la que se incluye en esa lista. Deben estar ordenadas de más antiguas a más modernas.

Para esta consulta se han creado tres vistas que guardarán las tres diferentes razones y los títulos y años de las películas identificadas por la razón. A partir de estas tres vistas, para solapar las tres razones identificando que películas están repetidas se ha creado una vista adicional para juntar las vistas que guardan las películas incluidas por un participante y las guardadas por tener una palabra clave relacionada con la población, para unir ambas vistas se ha utilizado un full outer join para incluir todas las tuplas de ambas vistas. Con esta vida adicional y con el uso de otro full outer join, se juntan las dos razones anteriores con la tercera que faltaba para completar la consulta.

ALGEBRA RELACIONAL:

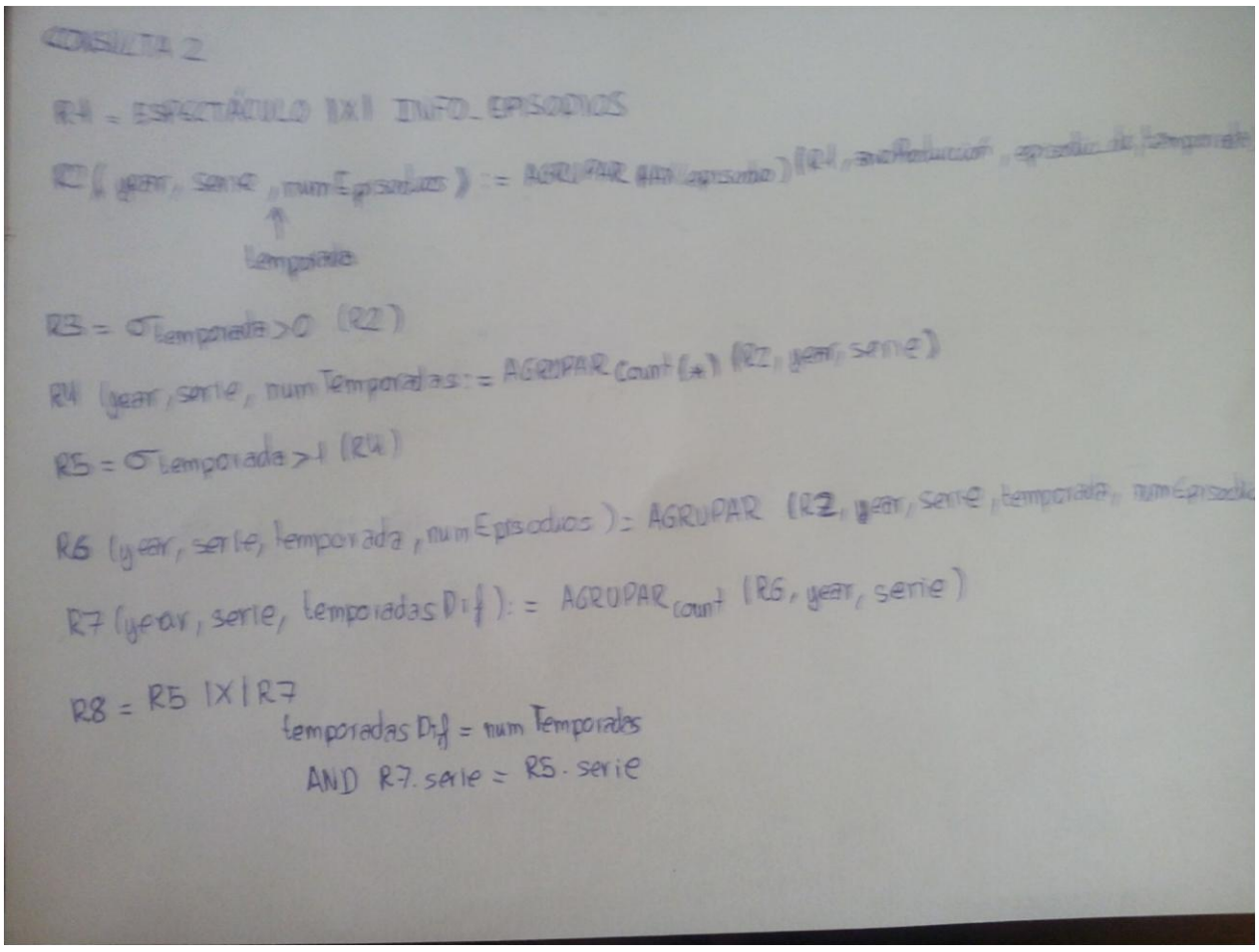


CONSULTA OBLIGATORIA 2

Ordena de más antiguas a más modernas aquellas series con dos o más temporadas en las que ninguna temporada tenga el mismo número de episodios que el resto de temporadas de la serie.

Para realizar esta consulta se han utilizado dos vistas. La primera para guardar el número de capítulos por temporada de las series incluidas en la base de datos, calculado con la función MAX en función de temporada. La segunda guarda las series y el número de temporadas de todas las series que tienen más de una temporada, función COUNT (DISTINCT temporada). Para obtener el resultado de esta consulta se listan todas las series de la primera vista cuyo número de episodios distintos en función de la temporada es igual al número de temporadas de la serie, función COUNT (DISTINCT num_episodios).

ALGEBRA RELACIONAL:

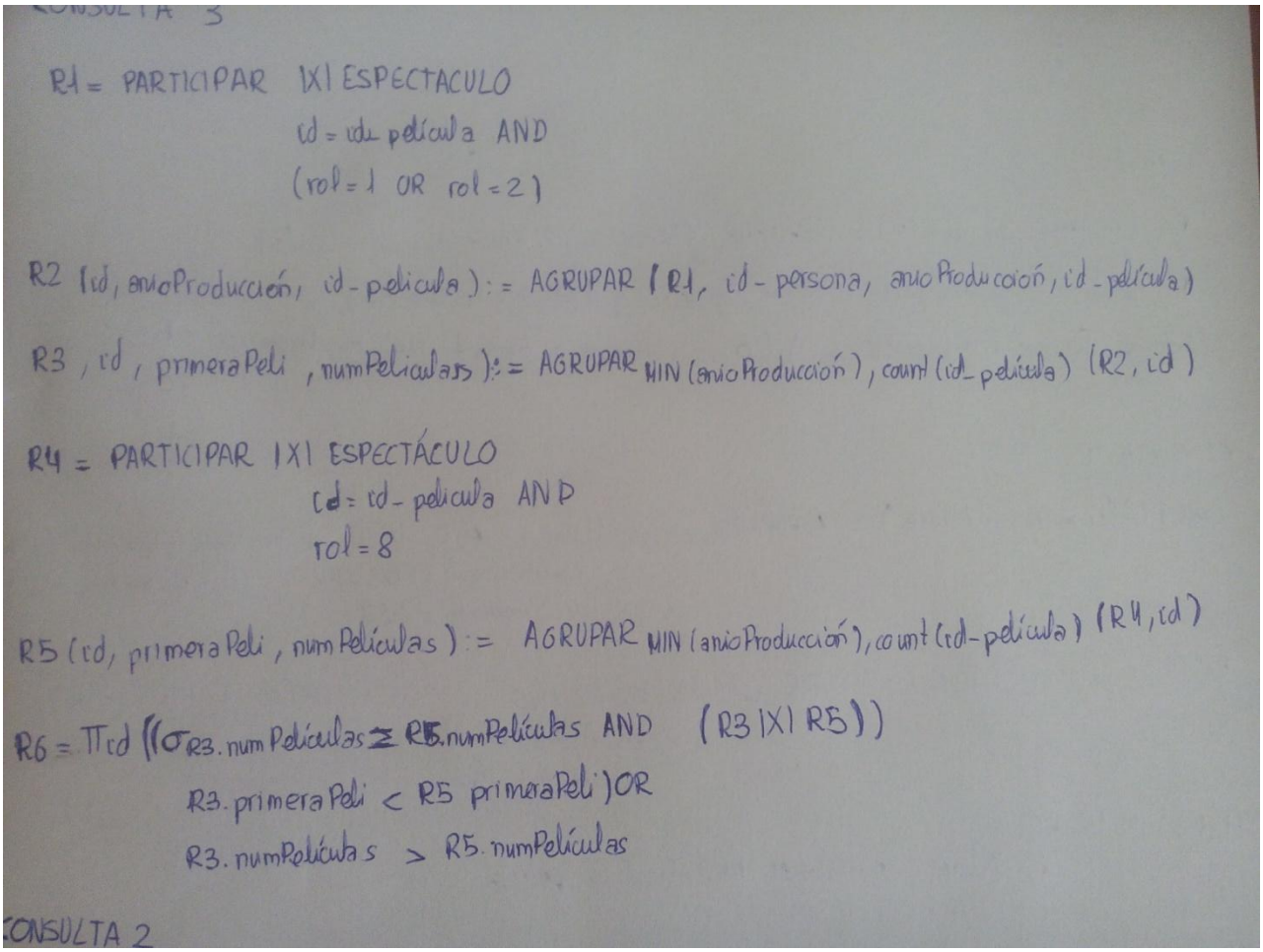


CONSULTA OBLIGATORIA 3

Personas que han tenido una época como director más larga y más prolífica a su época como actor. Ambas épocas se pueden solapar, pero primero comenzó de actor.

Para obtener el resultado de esta consulta se han creado dos vistas idénticas, una para directores y otra para actores, estas, guardan el id de la persona, el número de películas en las que ha participado o ha dirigido y el año de su primera película. Función MIN para obtener el menor año para cada persona, y COUNT (DISTINCT id_pelicula) para contar el número de películas). La consulta lista los ids de los actores que han hecho más películas como actor que las que hicieron como director, o si hicieron las mismasw comenzaron antes de actor. Se ha utilizado un INNER JOIN para conseguir todas las tuplas comunes a las dos vistas, que es lo que nos interesa.

ALGEBRA RELACIONAL:

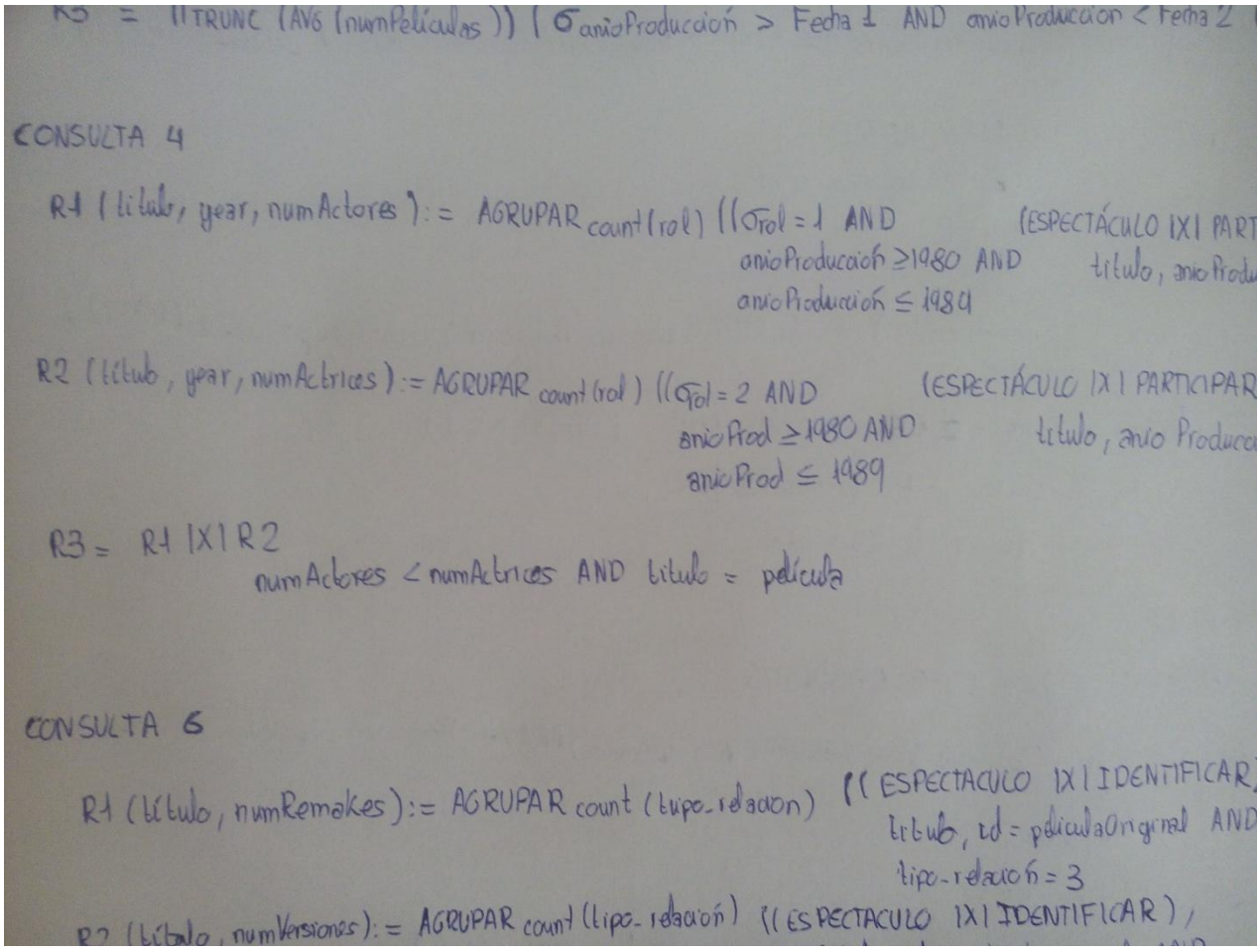


CONSULTA OPCIONAL 1

Lista de las películas, año de la producción, número de actores y actrices que participaron en ellas en las que el número de actrices es mayor que el número de actores ordenados de más antiguas a más modernas en la década de los 80 (1980-1989).

Se han utilizado dos vistas idénticas, una para actores y otra para actrices, estas vistas guardan, titulo, año de producción y el número de actores/actrices que aparecen en cada una de las películas, función COUNT (rol) con rol el número identificativo de actor o actriz para contar el número de los mism@s que participan en la película y año de producción comprendido entre 1980 y 1989. Con estas dos vistas se realiza un INNER JOIN para obtener las comunes pero que además cumplan la condición que el número de actrices sea mayor que el de actores.

ALGEBRA RELACIONAL:

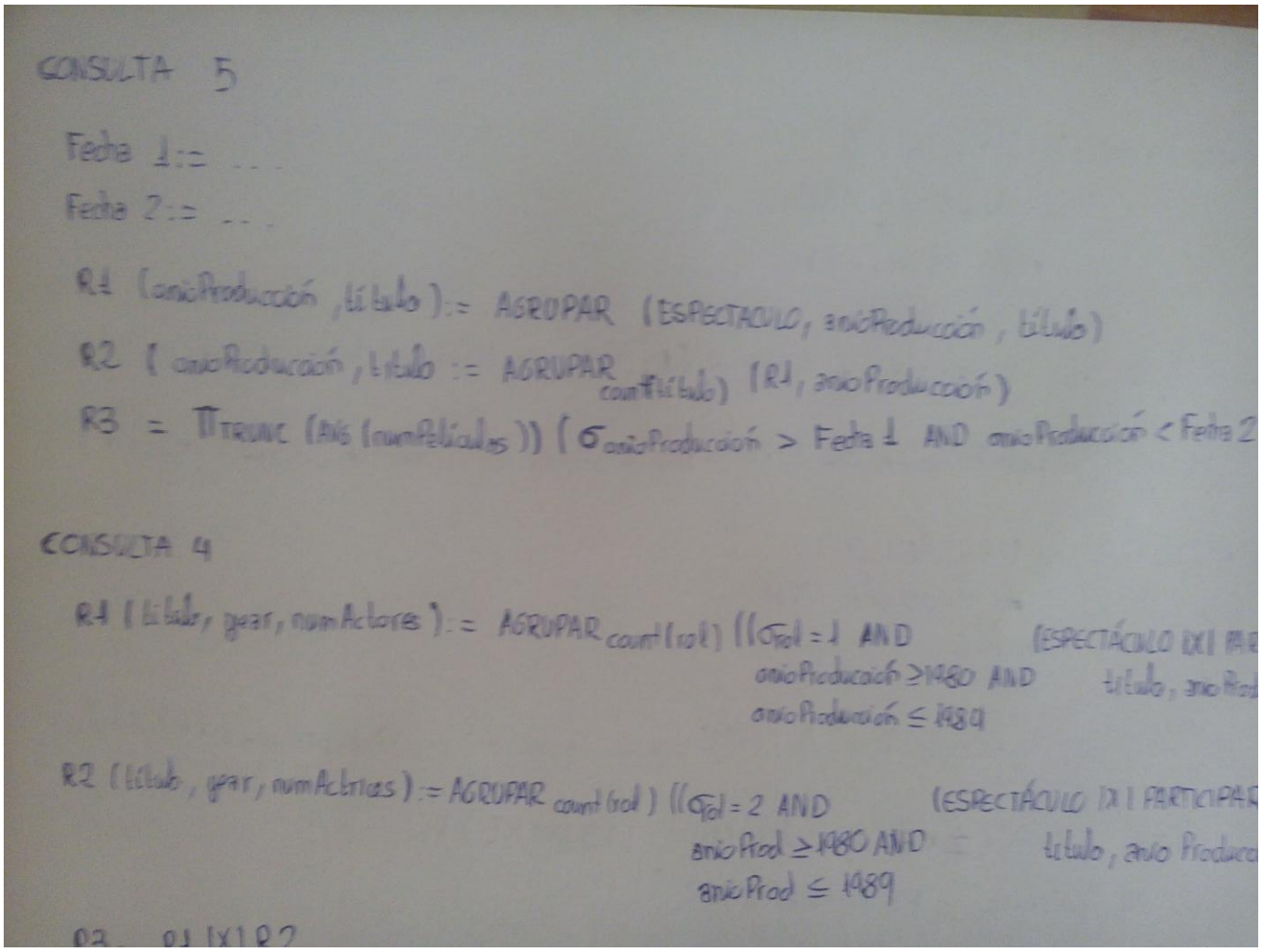


CONSULTA OPCIONAL 2

Número de medio de películas producidas por año en un período entre dos años introducidos.

Se ha utilizado una vista que almacena el número de películas por año de producción de todos los años de producción incluidos en la base de datos, función COUNT (DISTINCT titulo) para obtener el número de películas, y GROUP BY para ordenarlas por año. La consulta calcula una media truncada del número medio de películas entre dos años señalados, en nuestro caso 1979 y 2000.

ALGEBRA RELACIONAL:

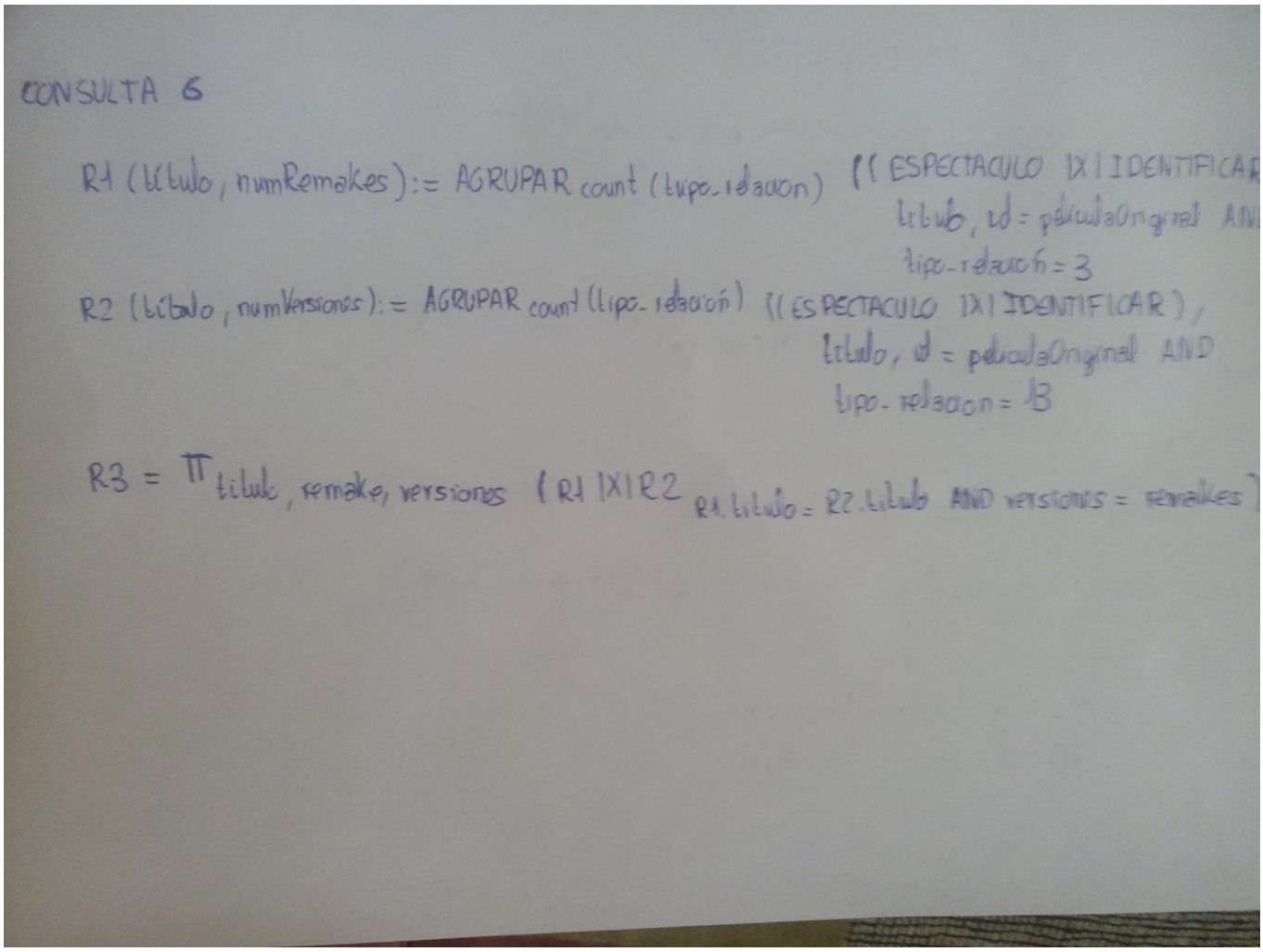


CONSULTA OPCIONAL 3

Titulo, número de remakes y versiones de todas las películas que tienen el mismo número de remakes que de versiones.

Dos vistas, una que calcula el número de versiones de las películas guardadas en la tabla IDENTIFICAR, que relaciona una película con otra; y otro que calcula el número de versiones. Funciones utilizadas COUNT en función del tipo de relación (3 para remakes y 13 para versiones). La consulta devuelve el titulo, número de remakes y número de versiones de todas las películas que tengan el mismo número de remakes como de versiones, para ello se usa un INNER JOIN entre las dos vistas creada sobre el titulo de la película y que cumplan la condición.

ALGEBRA RELACIONAL:



Los resultados de las consultas se adjuntan en una carpeta a parte en la práctica para no cargar la memoria de páginas.

DATOS SOBRE REUNIONES

Durante el mes de julio nos reunimos casi a diario para poblar las tres bases de datos del enunciado de las prácticas. El filtrado y la inserción correspondientes a esta práctica suman un total de 24 horas aproximadamente.

El proceso de consultas nos llevo un total de 20 horas, a esto hay que añadirle obtención de resultados, algebra relacional y verificación, así que en total unas 25 horas dedicadas al proceso de consultas a la base datos proporcionada y creación de las nuestras propias y las pedidas por le enunciado.

Decidimos realizar la inserción y depuración de datos de las tres bases en conjunto y dividirnos cada una de las prácticas para cada miembro, asi cada uno se encargaba de las consultas, diseño físico y memoria de su práctica. Esta práctica fue asignada a Alvaro Monteagudo Moreno con NIP: 681060.

MEMORIA PARTE 3: DISEÑO FÍSICO

ESTADISTICAS PRE_DISEÑO

1)
Statistics
---
763 recursive calls
0 db block gets
43409 consistent gets
1398 physical reads
247016 redo size
517015 bytes sent via SQL*Net to client
3607 bytes received via SQL*Net from client
295 SQL*Net roundtrips to/from client
61 sorts (memory)
0 sorts (disk)
4410 rows processed

2)
Statistics
---
103 recursive calls
0 db block gets
620 consistent gets
0 physical reads
116 redo size
328 bytes sent via SQL*Net to client
373 bytes received via SQL*Net from client
1 SQL*Net roundtrips to/from client
13 sorts (memory)
0 sorts (disk)
0 rows processed

3)-----

Statistics
---
27 recursive calls
0 db block gets
3722 consistent gets
0 physical reads
0 redo size
4116 bytes sent via SQL*Net to client
560 bytes received via SQL*Net from client
18 SQL*Net roundtrips to/from client

```
          6  sorts (memory)
          0  sorts (disk)
        241  rows processed
4 )

Statistics
----------------------------------------------------------
         28  recursive calls
          0  db block gets
       3722  consistent gets
          0  physical reads
          0  redo size
      25248  bytes sent via SQL*Net to client
        527  bytes received via SQL*Net from client
         15  SQL*Net roundtrips to/from client
          4  sorts (memory)
          0  sorts (disk)
        198  rows processed
5 )

Statistics
----------------------------------------------------------
         12  recursive calls
          0  db block gets
        173  consistent gets
          0  physical reads
          0  redo size
        419  bytes sent via SQL*Net to client
        384  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          1  rows processed
6 )

Statistics
----------------------------------------------------------
        104  recursive calls
          0  db block gets
        577  consistent gets
          0  physical reads
        268  redo size
       1191  bytes sent via SQL*Net to client
        395  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          8  sorts (memory)
          0  sorts (disk)
         17  rows processed
```

DISEÑO FÍSICO

Las claves numéricas como id en espectáculo y en persona deberían sustituirse por otras claves cortas pero que aporten información sobre el titulo o sobre el nombre. Sin embargo, Las tablas espectáculo y persona son demasiado numerosas como para modificarlas.

Podría haberse realizado particiones horizontales en la tabla participar para distinguir actores, directores y el resto. No se ha hecho debido al coste de actualización y a que tampoco suponen un aumento relevante en el rendimiento de las consultas.

Ya se realizo una partición vertical en la fase de diseño conceptual sobre espectáculo, debido a que la información muchos atributos de la tabla (información acerca de los episodios) no se utilizaban con el resto normalmente.

No se ha realizado precalculo de joins, porque las tablas son muy grandes y el coste de actualización sería excesivo.

No se han creado vistas materializadas puesto que ninguna favorecería lo suficiente a las consultas como para compensar su coste.

Se han empleado índices sobre la tabla espectáculo debido a que algunos atributos son utilizados habitualmente en condiciones de selección.

```
CREATE INDEX titulo_idx ON ESPECTACULO(titulo);
CREATE INDEX anioProd_idx ON ESPECTACULO(anioProduccion);
```

ESTADISTICAS POST_DISEÑO

1)

```
Statistics
----------------------------------------------------------
          172  recursive calls
            0  db block gets
        39859  consistent gets
           95  physical reads
            0  redo size
       517015  bytes sent via SQL*Net to client
3607  bytes received via SQL*Net from client
          295  SQL*Net roundtrips to/from client
           51  sorts (memory)
            0  sorts (disk)
         4410  rows processed
```

2)

```
Statistics
----------------------------------------------------------
           39  recursive calls
            0  db block gets
          613  consistent gets
            0  physical reads
            0  redo size
          328  bytes sent via SQL*Net to client
          373  bytes received via SQL*Net from client
            1  SQL*Net roundtrips to/from client
           13  sorts (memory)
            0  sorts (disk)
            0  rows processed
```

3)

```
Statistics
----------------------------------------------------------
           27  recursive calls
            0  db block gets
         3722  consistent gets
            0  physical reads
            0  redo size
         4116  bytes sent via SQL*Net to client
          560  bytes received via SQL*Net from client
           18  SQL*Net roundtrips to/from client
            6  sorts (memory)
            0  sorts (disk)
          241  rows processed
```

4)

Statistics
---
28 recursive calls
0 db block gets
3722 consistent gets
0 physical reads
0 redo size
25248 bytes sent via SQL*Net to client
527 bytes received via SQL*Net from client
15 SQL*Net roundtrips to/from client
4 sorts (memory)
0 sorts (disk)
198 rows processed

5)

Statistics
---
12 recursive calls
0 db block gets
173 consistent gets
0 physical reads
0 redo size
419 bytes sent via SQL*Net to client
384 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
1 rows processed

6)

Statistics
---
40 recursive calls
0 db block gets
568 consistent gets
0 physical reads
0 redo size
1191 bytes sent via SQL*Net to client
395 bytes received via SQL*Net from client
3 SQL*Net roundtrips to/from client
8 sorts (memory)
0 sorts (disk)
17 rows processed

TRIGGERS

Para la consistencia de la base de datos se ha diseñado un trigger que compruebe cada vez que se inserte un episodio de una temporada de una serie que no esta ya introducido en la base de datos.

Para ello hemos creado un trigger después de insrción o actualización sobre la tabla INFO_EPISODIOS. Para gestionar el trigger hemos utilizado un cursor sobre los atributos temporada, episodio_de e id de la tabla INFO_EPISODIOS. Abrimos el cursor, renombramos sus parámetros y comprobamos si los nuevos parámetros no están introducidos en la tabla, en caso contrario alta un mensaje de error informando sobre ello.

Como trigger de tarea no trivial se ha creado uno que añada una columna a la tabla ESPECTÁCULO que almacene el número de palabras clave que están relacionadas con la misma y la incrementa cuando toque, quizás en un futuro pudiera ser necsario por algún usuario que se conecte a la base de datos.

Para ello hemos utilizado una variable contador que fuera contando las tuplas coincidentes entre dos tablas or método de un INNER JOIN, al terminar hace SET sobre la columna definida en ALTER TABLE mencionado anteriprmente