

Sistemas Informáticos 1

Practica 3

Cambios realizados en el fichero actualiza.sql:

Hemos creado 3 tablas adicionales como se pide en el enunciado, con el fin de eliminar atributos multivaluados (Countries, Genres, Languages). Todas ellas se componen de los campos id y nombre. Siendo la clave primaria el id.

Una vez hechas estas 3 tablas cambiamos las tablas imdb_moviecountries, imdb_moviegenres, imdb_movi_languages, en las que se ha cambiado el campo correspondiente por una referencia a las tablas anteriores. Además, imdb_movi_languages tiene un campo adicional, extrainfo, a la que se le han modificado las restricciones permitiendo NULL. Este diseño podría mejorarse debido a que es preferible evitar los campos con NULL.

En lo relacionado a actormovies, hemos eliminado de la clave primaria numarticipation siendo la clave primaria resultante la combinación de actorid y movieid. Estos campos han pasado a ser Foreign keys a la tablas imdb_actors y imdb_movies respectivamente. Por último, se permiten NULL en la columna asCharacter.

El único cambio en imdb_actor hemos optado por reducir el campo gender a una única letra, M o F para ser consistentes con la tabla customers.

En la imdb_directormovies se elimina de la primary key el campo numparticipations. De esta forma la clave primaria queda (movieid , directorid), que son referencias a la tablas imdb_movies y imdb_directors.

Los cambios realizados en imdb_movies se reducen a eliminar la restricción de NOT NULL movierelease.

En la tabla orderdetail faltaba la clave primaria. Consideramos que debería ser (orderid, prod_id) pero, si la utilizamos, tenemos claves primarias duplicadas. Para solucionarlo realizamos una query de las claves primarias repetidas agrupándolas por (orderid, prod_id). De esta forma conseguimos actualizar la tabla en la que no tenemos duplicados manteniendo la información inicial. Se han añadido dos referencias a orders (orderid) y products (prod_id).

En orders se ha añadido la clave extranjera customerid. Además la secuencia de orderid estaba en 1, y debido a la necesidad de insertar nuevos pedidos le hemos asignado el valor máximo de orderid.

En customers, se ha eliminado la restricción de NOT NULL en todos los atributos no utilizados por nuestra aplicación. Se ha añadido NOT NULL en email e income. Además email a pasado a ser UNIQUE debido a que sera usado para identificar a los usuarios de nuestra tienda. El campo income ahora es de tipo NUMERIC ya que servirá como saldo de los clientes de la aplicación. Además hemos cifrado el campo password en md5.

Al igual que en la tabla orders la secuencia del id de los usuarios estaba a 1 y la hemos puesto en su valor máximo.

Hemos creado 2 tablas adicionales:

- alertas, con campos prod_id y mgs. Siendo la clave primaria la combinación de los mismos. Actualmente solo se considera el caso 'producto agotado' pero podrían existir distintos tipos de alerta para un mismo producto. Inicialmente se rellena con todos aquellos productos cuyo stock esta a 0.
- topventas, esta tabla se crea con el fin de minimizar el tiempo de ejecución del procedimiento almacenado getTopVentas solicitado en el enunciado. Dicho procedimiento tardaba bastante tiempo en ser ejecutado (varios segundos), pero debido a que tendría escasa actualización hemos decidido añadirla a actualiza.sql.

SETPRICE.SQL:

Conseguimos la diferencia de años desde la venta de la película hasta hoy. Tras esto dividimos el precio actual entre 1,02 elevado a la diferencia de años. Actualizamos el precio con el resultado obtenido.

	orderid integer	prod_id integer	price numeric	quantity integer
1	181791	27	13	1
2	181791	5761	14	1
3	181792	6531	15.6	1
4	181793	6530	13	1
5	13	2321	14.1348350182057	1
6	28	2781	12.9338359643712	1
7	51	4109	17	1
8	98	1372	9.23845426026514	1
9	100	6519	15.8592848904268	1
10	147	2187	14.1348350182057	1

SETORDERAMOUNT.SQL:

Conseguimos los pedidos con netamount o totalamount a NULL y los completamos con sus correspondientes valores.

En el caso de netamount tomamos la suma de los precios de los diferentes productos multiplicados por su cantidad, mientras que totalamount lo actualizamos con $\text{netamount} * (\text{taxes}/100)$.

	orderid integer	orderdate date	customerid integer	netamount numeric	tax numeric	totalamount numeric	status character varying(10)
1	181791	2017-11-29	1	51	21	61.710000000000000000	Paid
2	108	2016-01-08	1	44.1176470588235	15	50.735294117647025	Paid
3	107	2012-10-05	1	128.0703365099502	15	147.280886986442730	Paid
4	104	2015-02-12	1	27.6816608996540	15	31.833910034602100	Paid
5	105	2012-11-06	1	11.9556466897549	15	13.748993693218135	Paid
6	106	2015-12-08	1	28.43137254901965	15	32.6960784313725975	Paid
7	103	2012-12-05	1	25.1285955879212	15	28.897884926109380	Paid
8	181792	2017-11-29	1	15.6	21	18.8760000000000000	Paid
9	181793	2017-11-29	1	13	21	15.7300000000000000	Paid
10	2026	2013-01-04	142	52.6591892835113	15	60.558067676037995	Shipped

GETTOPVENTAS.SQL:

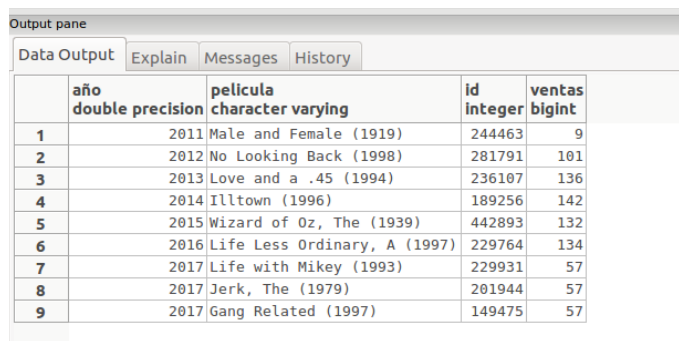
Debido a que hemos creado una tabla donde ya almacenamos los productos más vendidos. El producto almacenado se reduce a la búsqueda de los registros en los que el año sea mayor o igual al pasado como argumento.

Ademas destacamos que en la tabla creada hemos añadido el id de las películas, para simplificar la programación de la pagina web.

Para la creación de la tabla agrupamos el JOIN de (orders, orderdetail, products) por fecha y película, sumando las cantidades de las ventas. Sobre este resultado seleccionamos el máximo.

Somos conscientes de que se realizan 2 consultas muy parecidas, pero no hemos sido capaces de obtener una solución mas óptima. Una de las opciones que intentamos fue la creación de un procedimiento almacenado en PLPGSQL con instrucciones de control de flujo pero el resultado, a pesar de ser el mismo, mostraba un peor rendimiento.

Nótese que en caso de tener para varias películas con el mismo número de ventas las mostramos todas.



	año	pelicula	id	ventas
	double precision	character varying	integer	bigint
1	2011	Male and Female (1919)	244463	9
2	2012	No Looking Back (1998)	281791	101
3	2013	Love and a .45 (1994)	236107	136
4	2014	Illtown (1996)	189256	142
5	2015	Wizard of Oz, The (1939)	442893	132
6	2016	Life Less Ordinary, A (1997)	229764	134
7	2017	Life with Mikey (1993)	229931	57
8	2017	Jerk, The (1979)	201944	57
9	2017	Gang Related (1997)	149475	57

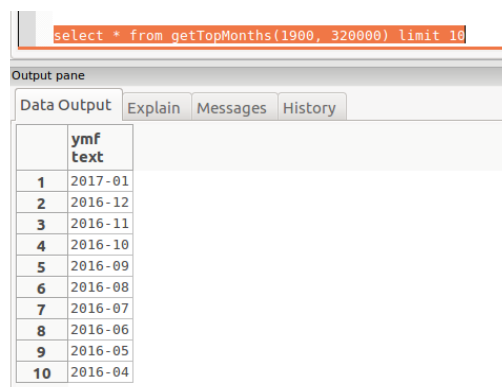
GETTOPMONTHS.SQL:

Este procedimiento almacenado puede entenderse como la unión de dos consultas distintas.

La primera de ellas devuelve los meses-años en los que los ingresos obtenidos superan el argumento pasado a la función. Para ellos simplemente agrupamos por a mes-año y realizamos la operación de suma de totalamount.

La segunda devuelve los meses-años en los que el número de productos vendidos supera el umbral pasado como parámetro. Para ellos simplemente sumamos el número de ventas de ese producto a lo largo de ese mes.

Es importante notar que la unión de dos consultas debe tener las mismas columnas, condición que se cumple. Además destacamos que la operación UNION opera con conjuntos, de manera que no aparecerán resultados repetidos.



The screenshot shows a SQL query execution interface. At the top, a query is entered in a text box: `select * from getTopMonths(1900, 320000) limit 10`. Below the query box is an "Output pane" with tabs for "Data Output", "Explain", "Messages", and "History". The "Data Output" tab is selected, displaying a table with 10 rows of results. The first column is labeled "ymf" and the second column is labeled "text". The results are as follows:

	ymf	text
1	2017-01	
2	2016-12	
3	2016-11	
4	2016-10	
5	2016-09	
6	2016-08	
7	2016-07	
8	2016-06	
9	2016-05	
10	2016-04	

UPDORDERS.SQL:

Se consideran tres casos distintos dependiendo de la operación que se haga sobre orderdetail:

Caso INSERT:

Se supone que el carrito ya está creado, puesto que se hará desde php cuando el usuario haga login en la aplicación.

Actualizamos el carrito con el nuevo objeto que se desea comprar, sumando en netamount la cantidad por el precio del nuevo artículo.

Caso UPDATE:

Para obtener el número de unidades que debemos modificar restamos el valor quantity de los registros NEW y OLD que nos proporciona PL/PGSQL. Si la cantidad llega a cero no permitimos que se guarde este valor, si no que borramos el registro de

la tabla. Sea la cantidad obtenida positiva o negativa, solo tendremos que multiplicar este valor por el precio y sumar al totalamount anterior.

Caso DELETE:

Necesitamos saber cuantas unidades vamos a eliminar, obtenemos este valor del registro OLD. Si este número resulta ser igual al que había antes tenemos que borrar el registro. Si no, simplemente restamos la cantidad a la que hubiese antes.

UPDINVENTORY.SQL:

Este trigger solo debe ejecutarse cuando se pase confirme una compra. Esto se traduce en cambiar de NULL a otro valor el campo status. Para que se comporte de esta manera añadimos la instrucción WHEN a la creación del trigger.

Hemos decidido que en caso de que no podamos vender todas las unidades de alguno de los productos no se realizara la compra, manteniendo el estado del order a NULL(todavía es un carrito) e informando al cliente de que no se ha podido terminar su compra.

NOTA: Como el diagrama Entidad-Relación no cabe en el documento se entrega fuera de este, de manera que se puede modificar su tamaño para su mejor visualización.