|  |  |  |
| --- | --- | --- |
|  |  | PECL3  BBDD |

Miguel

Sergio

Juan Casado Ballesteros – 09108762A

Profesor: Sergio Caro

Laboratorio Martes 12:00 a 14:00

# índice

PECL3

[índice 1](#_Toc513805635)

[Análisis de las formas normales 2](#_Toc513805636)

[Primera forma normal: 2](#_Toc513805637)

[Segunda forma normal: 2](#_Toc513805638)

[Tercera forma normal: 2](#_Toc513805639)

[Ejemplos: 4](#_Toc513805640)

[Disparadores 5](#_Toc513805641)

[Calcular el precio del ticket: 5](#_Toc513805642)

[Calcular los puntos de repostaje: 5](#_Toc513805643)

[Control de empleados: 5](#_Toc513805644)

[Calcular PVP Artículo: 5](#_Toc513805645)

[Calcular puntos del empleado: 5](#_Toc513805646)

[Disparadores de dominio: 5](#_Toc513805647)

# Análisis de las formas normales

## Primera forma normal:

Todas nuestras relaciones están en primera forma normal ya que todos nuestros atributos son atómicos.

Para lograr la atomicidad de los atributos manteniendo la menor redundancia posible hemos descompuesto aquellos que eran multivariados, es decir, los hemos sacado a una tabla externa (teléfonos) que se relaciona con la tabla en la que estaba originalmente (empleado) a partir de la PK de esta (DNI). Esto lo hemos realizado cuando no conocíamos exactamente la cantidad de atributos multivaluados que iban a existir en cada instancia.

En el caso de si conocerlos y de ser fijos como ocurre con la Dirección del Empleado se los hemos incluido como atributos atómicos dentro de la relación.

## Segunda forma normal:

Todas nuestras relaciones cumplen la segunda forma normal ya que cumplen la primera y además cumplen que todas las dependencias funcionales de cada tabla son totales, es decir que todos los atributos no primos de cada relación dependen funcionalmente de forma total de la PK de la relación.

Para lograr esto hemos puesto como PK una fecha en relaciones como Canjea, Contiene, Reposta, Opinión de modo que todos los atributos de la relación dependan funcionalmente de forma total de dicha PK.

## Tercera forma normal:

Todas nuestras relaciones cumplen la tercera forma normal ya que cumplen la segunda y además cumple que no existen atributos no primos de las relaciones que dependan transitivamente de algún atributo que no sea clave de la relación.

Los atributos no primos se relacionan funcionalmente con toda la clave y exclusivamente con toda la clave, para lograrlo hemos tenido que modificar al empleado debido a que el CP se relacionaba funcionalmente con la localidad y el resto de los atributos del original atributo compuesto dirección con DNI.

Hemos sacado a una nueva relación la localidad y el CP de forma que dirección tome como FK el CP de esa relación y el DNI sea la PFK de la relación dirección.

También hemos cambiado como se guardan los sorteos en la base de datos, ahora cuando un ticket es premiado es almacenada la fecha de su premio en una tabla Sorteo, esto se hace porque el booleano que controla si el ticket ha sido canjeado ya depende de esta fecha.

Finalmente, el esquema relacional en 3FN quedaría:



## Ejemplos:

Mostramos las dependencias funcionales del Empleado como ejemplo de esquema en tercera forma normal.

El resto de sus “atributos” estará en las tablas Dirección y Localidades para mantener la BBDD en 3FN.

Podemos ver como todos los atributos no primos se relacionan funcionalmente de forma exclusiva con la PK y nada más que con ellas dependiendo totalmente de ella.

Podemos ver como la dirección que podría considerarse como un atributo multivalorado se ha convertido en un conjunto de atributos atómicos dentro de la relación.

En esta otra mostramos cómo descompuesto Empleado para separarlo de sus teléfonos ya que es un atributo multivalorado del que no sabemos cuantos puede llegar a tener, así seguimos respetando las teorías de normalización.



Cliente es otra relación en la que se aprecia claramente como se respeta la tercera forma normal ya que todos los atributos no primos relacionan exclusivamente con todas las claves y exclusivamente con las claves.

# Disparadores

Adicionalmente a los disparadores indicados en la práctica hemos hecho otros adicionales, los explicaremos todos a continuación:

## Calcular el precio del ticket:

Este disparador salta cuando se crea o se modifica un ticket, buscará cada artículo y multiplicará su PVP por la cantidad de este, posteriormente sumará todos esos resultados.

* [Trigger](Triggers/SQL_Triggers/Calcular_Precio_Ticket.txt)
* [Llamada](Triggers/SQL_Triggers/Calcular_Precio.txt)

## Calcular los puntos de repostaje:

Este disparador tomará los litros y el precio de los mismos en el momento de ser creado para calcular los punto a punto el euro.

* [Trigger](Triggers/SQL_Triggers/Calcular_Puntos_Repostaje.txt)
* [Llamada](Triggers/SQL_Triggers/Calcular_Puntos.txt)

## Control de empleados:

Este disparador se asegura de que un trabajador no pueda trabajar en un surtidor y una tienda a la vez.

## [Trigger](Triggers/SQL_Triggers/Comprobar_Puesto_Empleado.txt)

* [Llamada](Triggers/SQL_Triggers/Comprobar_Puesto.txt)

## Calcular PVP Artículo:

Cuando un artículo es insertado calculamos su PVP en función del iva.

* [Trigger](Triggers/SQL_Triggers/Calcular_PVP_Articulo.txt)
* [Llamada](Triggers/SQL_Triggers/Calcular_PVP.txt)

## 

## Calcular puntos del cliente:

Toma los puntos ganados al repostar y los gastados al canjear y le pone al empleado los puntos que tiene. Adicionalmente este disparador no dejará al empleado canjear puntos si no tiene suficientes.

Si modificamos los artículos:

* [Trigger](Triggers/SQL_Triggers/Modificar_Puntos_Canjea.txt)
* [Llamada](Triggers/SQL_Triggers/Modificar_Puntos_Canjea.txt)

Si insertamos artículos:

* [Trigger](Triggers/SQL_Triggers/Sumar_Puntos_Clientes.txt)
* [Llamada](Triggers/SQL_Triggers/Restar_Puntos.txt)

Si repostamos:

* [Trigger](Triggers/SQL_Triggers/Sumar_Puntos_Clientes.txt)
* [Llamada](Triggers/SQL_Triggers/Sumar_Puntos.txt)

Si modificamos un repostaje:

* [Trigger](Triggers/SQL_Triggers/Modificar_Puntos_Cliente_Repostaje.txt)
* [Llamada](Triggers/SQL_Triggers/Modificar_Puntos_Repostaje.txt)

## Disparadores de dominio:

Hemos hecho unos cuantos disparadores para comprobar que ciertos atributos que no pueden ser negativos o que obligatoriamente tengan que ser mayor que 1. Esto debería haberse hecho creando dominios que así lo regularan, pero debido a que en esta práctica se pedían disparadores hemos decidido hacerlo así.

* En la carpeta ./Triggers/SQL\_Triggers/ los que empiezan por comprobar o exception.

# Roles, grupos y usuarios

# Caso práctico JAVA

Hemos creado un programa JAVA desde el cual nos conectamos a la BBDD y podemos realizar bajo distintas cuentas de usuario consultas cuyo resultado luego mostramos.

Desde una clase ConectionManager nos conectamos a la BBDD con un usuario que se haya seleccionado previamente en la interface gráfica. Una vez seleccionado uno no hace falta volver a seleccionarlo para hacer otra consulta si no se desea cambiar de usuario.

Posteriormente mediante la clase Query utilizaremos la clase anterior para conectarnos y desconectarnos de la BBDD en cada consulta. Tendremos unas consultas modelo ya hechas (las de la PECL2) que podremos seleccionar y visualizar. Adicionalmente podremos hacer cualquier consulta personalizada.

Para mostrar los resultados iteramos sobre cada fila de datos obtenida y dentro de la fila sobre cada atributo mostrándolos todos.

Podemos comprobar como solo si tenemos permisos para hacer cada consulta podremos realizarla. De no tenerlos se nos denegará el acceso.

* ConnectionManager
* Query
* Asker (nos permite seleccionar las consultas)
* Logger (nos permite seleccionar los usuarios)

## Ejemplos: