UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

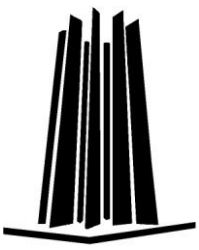
FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

“SISTEMA DE CONTROL DE DATOS PARA UNA CASA DE EMPEÑO”

T R A B A J O E S C R I T O
EN LA MODALIDAD DE SEMINARIOS
Y CURSOS DE ACTUALIZACIÓN Y
C A P A C I T A C I Ó N P R O F E S I O N A L
QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN
P R E S E N T A :

ABEL ISAEL TORRES CASTILLO

ASESOR: M. en I. Arcelia Bernal Díaz



MÉXICO, 2009.

AGRADECIMIENTOS

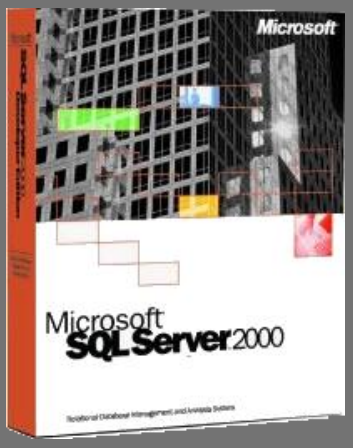
- A DIOS, Por llevarme en el camino correcto.
- A MIS PADRES, Abel Torres Acevedo y Gloria Castillo Sánchez, por su apoyo, cariño y confianza; sin ustedes no hubiera sido posible.
- A MIS HERMANAS, Alejandra Torres Castillo y Miriam Karla Torres Castillo, por su apoyo y ejemplo… las quiero mucho!
- A LA MAESTRA, M. en I. Arcelia Bernal Díaz, por su apoyo y llevar a buen término este trabajo.
- A MIS FAMILIARES Y AMIGOS, por mostrarme su apoyo, en especial al Dr. Mario Luís Flores Huerta, por ayudarme académicamente.
- A TODOS MIS MAESTROS, en especial al M. en C. Marcelo Pérez Medel, Ing. Hugo Portilla Vázquez, M. en C. Jesús Hernández Cabrera y al Ing. Enrique García Guzmán por apoyarme en la realización de este trabajo.
- A LA UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO y FES Aragón, por abrirme sus puertas y contribuir a mi formación.

2009

“SISTEMA DE CONTROL DE DATOS PARA UNA CASA DE EMPEÑO”

SCD-CE

2° DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE.



ABEL ISAEL TORRES CASTILLO

ÍNDICE

ÍNDICE

INTRODUCCIÓN

Las DB (Bases de Datos) son muy importantes, ya que permiten a cualquier sistema de información tener un buen almacenamiento de datos y una buena organización de ellos, por lo que es imprescindible, además de contar con el lenguaje, tener una buena planeación del diseño de la BD, ya que de ello dependerá el óptimo desempeño del sistema; para una casa de empeño de igual forma es fundamental, el contar con un sistema que sea capaz de organizar adecuadamente los datos y que estos regresen al mismo tiempo información sobre su estado. Para esto es necesario contar con una herramienta que permita la realización de éste sistema como SQL (Structured Query Language) que traducido *significa* Lenguaje de Consulta Estructurada, y es un lenguaje de alto nivel diseñado para la creación de sistemas de bases de datos relacionales. La ANSI (Instituto Americano de Normalización) y la ISO (Organismo Internacional de Normalización) fueron los encargados de que este lenguaje se estandarizara, y por ello este lenguaje se encuentra hoy en día en cualquier DBMS (Sistema Administrador de Base de Datos), donde se emplean programas como Oracle, SQL Server, MySQL, por mencionar algunos.

En este trabajo se planteará el desarrollo de la BD para una casa de empeño, de donde se retoma información y funcionamiento de 'la primera casa de empeño en México que fue "El Sacro y Real Monte de Piedad de Ánimas" que fue inaugurada por el Conde de Santa María de Regla, Don Pedro Romero de Terreros el sábado 25 de febrero de 1775, con una cantidad de 300 mil pesos oro para esta obra asistencial, dinero destinado del propio peculio del Fundador.

Después de que las autoridades eclesiásticas ofrecieran una misa, y se encontraran presentes importantes personalidades de la Corona española y sociedad en general para dar testimonio de esta importante obra, Juan Carabantes fue la primera persona en empeñar un aderezo de diamantes por el cual se le prestaron 40 pesos oro, recuperándola en agosto de ese mismo año.

Durante su primer año de operación el Nacional Monte de Piedad ya había realizado 17,000 operaciones de empeño, lo que equivalía a un cuarto de la población de la entonces Ciudad de México.

Hoy día esta Bicentenaria Institución respalda a la cuarta parte de los hogares del país; realiza un promedio de 20 millones de contratos prendarios cada año y recibe alrededor de 3 millones de artículos mensuales a través de sus diferentes sucursales distribuidas en toda la geografía nacional'[1]. De ahí la

[1] http://www.montepiedad.com.mx/ (historia)

complejidad de crear de un sistema que permita el manejo correcto y funcionalidad de los datos.

El presente proyecto tiene como principal objetivo agilizar diversas tareas que se realizan entorno a las prendas en una casa de empeño, ya que unas de las tareas que no realiza el sistema actual del Nacional Monte de Piedad, es el informe automático de las prendas que han expirado (estado vencido) y que ya no pueden ser rescatadas, además del reporte de ingresos y egresos por concepto de empeños, desempeños, refrendos y ventas, así como el poder automatizar otras operaciones como la generación de reportes de prendas para cada uno de sus estados utilizando procedimientos almacenados, también se generará la actualización de estados en automático, utilizando para la realización de ésta tarea los disparadores (triggers), sacando provecho de herramientas como el SQL Server que será el motor principal para este sistema, ya que ésta herramienta permite además del almacenamiento organizado de los datos, poder acceder a ellos de diversas formas según convenga a la funcionalidad o al provecho que se quiera sacar de estos datos.

Por otra parte se desean integrar todas las tareas que giran entorno a las prendas y objetos que se manejan, para crear una base de datos que contenga y realice el control de todos los productos, para la actualización de los datos se desea crear un programa (trigger) que sea capaz de actualizar el campo de estado, que es el que indicará si la prenda está vencida, refrendada, vendida, empeñada, o es cargo, por lo que el estado que se desea identificar y automatizar para este proyecto será el del estado vencido, para que al mismo tiempo éste impida que la prenda pueda ser refrendada o desempeñada, de ahí la importancia de generar este programa sobre nuestra tabla de prendas.

El presente trabajo estará compuesto de 5 capítulos, organizados de tal manera que se pueda comprender la metodología que se siguió para la generación del sistema (SCD-CE), partiendo de los siguientes puntos:

- Capítulo 1 Lenguaje de Consulta Estructurado (SQL): contendrá una reseña del SQL Server, además de una breve historia del programa y su estructura lógica.

- Capítulo 2 Análisis de requerimientos para el SCD-CE (Sistema de Control de Datos para una Casa de Empeño): mostrará los requerimientos mínimos para la realización del SCD-CE, contemplando el servidor donde se almacenará el sistema y el software que se utilizará.

- Capítulo 3 Diseño del SCD-CE: se explicará el modelo entidad relación y el diccionario de datos para comprender lo que es el desarrollo del sistema.

- Capítulo 4 Desarrollo del SCD-CE: mostrará como se realizó la BD desde las tablas hasta los procedimientos almacenados.

- Capítulo 5 Conclusiones y Pruebas a la BD: aquí se mencionarán los resultados que se obtuvieron en la utilización del sistema y en el proceso de su realización.

CAPÍTULO 1

1. Lenguaje de Consulta Estructurado (SQL)

'La historia de SQL empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba SEQUEL (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Los experimentos con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM y lo adoptaron algunos de sus clientes elegidos. Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo Oracle y Sybase, sólo por citar algunos) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho por lo que respecta a las bases de datos relacionales.

En 1986, el ANSI adoptó SQL (sustancialmente acogió el dialecto SQL de IBM) como estándar para los lenguajes relacionales y en 1987 se transformó en estándar ISO. Esta versión del estándar va con el nombre de SQL/86. En los años siguientes, éste ha sufrido diversas revisiones que han conducido primero a la versión SQL/89 y, posteriormente, a la actual SQL/92. El hecho de tener un estándar definido por un lenguaje para bases de datos relacionales abre potencialmente el camino a la intercomunicación entre todos los productos que se basan en él.

Actualmente, está en marcha un proceso de revisión del lenguaje por parte de los comités ANSI e ISO, que debería terminar en la definición de lo que en este momento se conoce como SQL3. Las características principales de esta nueva encarnación de SQL deberían ser su transformación en un lenguaje stand-alone (mientras ahora se usa como lenguaje hospedado en otros lenguajes) y la introducción de nuevos tipos de datos más complejos que permitan, por ejemplo, el tratamiento de datos multimediales'[1].

[1] http://www.wikilearning.com/curso_gratis/estructura_de_las_bases_de_datos_relacionales-lenguajes_de_consulta_a_bases_de_datos_relacionales/3623-4

CAPÍTULO 1

Para trabajar con la información de una base de datos, se debe utilizar un conjunto de comandos e instrucciones (lenguaje) definidos por el software del DBMS (Database Management System). En las bases de datos relacionales se pueden utilizar varios lenguajes ('Lenguajes Formales y Lenguajes Comerciales'[2]), de los que SQL es el más común. El American National Standards Institute (ANSI) y la International Standards Organization (ISO) definen estándares de software, incluidos los estándares para el lenguaje SQL. SQL Server 2000 admite el nivel de entrada de SQL-92, el estándar de SQL publicado por ANSI e ISO en 1992. El dialecto de SQL compatible con Microsoft SQL Server se llama Transact-SQL (T-SQL). T-SQL es el lenguaje principal utilizado por las aplicaciones de Microsoft SQL Server.

1.1 Ediciones de SQL Server 2000

Microsoft® SQL Server™ 2000 está disponible en las ediciones siguientes:

- SQL Server 2000 Enterprise

'Se utiliza como un servidor de base de datos de producción. Admite todas las características que están disponibles en SQL Server 2000 y es ampliable a los niveles de rendimiento que se requieren para ser compatible con los sitios Web más grandes, y el procesamiento de transacciones en línea (OLTP) y los sistemas de almacenamiento de datos corporativos'[3].

- SQL Server 2000 Standard

'Se utiliza como un servidor de base de datos para un pequeño grupo de trabajo o departamento'[4].

- SQL Server 2000 Personal

'La utilizan los usuarios móviles que están durante algún tiempo desconectados de la red, pero ejecutan aplicaciones que requieren un almacén de datos SQL Server. También se utiliza durante la ejecución de una aplicación independiente que requiere un almacén de datos SQL Server local en un equipo cliente'[5].

- SQL Server 2000 Developer

[3] Apuntes 2° DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE; Módulo2 pag. 4
[4] Idem
[5] Idem

'La utilizan los programadores que desarrollan aplicaciones que utilizan SQL Server 2000 como su almacén de datos. Aunque SQL Server 2000 Developer admite todas las características de SQL Server 2000 Enterprise, que permite a los programadores escribir y probar aplicaciones que pueden utilizar estas características, se autoriza el uso de SQL Server 2000 Developer sólo como un sistema de desarrollo y prueba, no como un servidor de producción'[6].

- SQL Server 2000 para Windows CE

'Microsoft® SQL Server 2000™ para Windows® CE (SQL Server CE) se utiliza como almacén de datos en los dispositivos Windows CE. Capaz de duplicar datos con cualquier versión de SQL Server 2000 para mantener los datos de Windows CE sincronizados con la base de datos principal.

- SQL Server 2000 Enterprise Evaluation

Versión con todas las características que está disponible para su descarga gratuita desde el Web. Está destinada sólo para utilizarla en la evaluación de características de SQL Server; esta versión dejará de funcionar cuando hayan transcurrido 180 días desde su descarga.

Estas versiones de SQL Server 2000 también incluyen un componente que permite a los desarrolladores de aplicaciones distribuir una copia del motor de la base de datos relacional con sus aplicaciones. Este componente tiene dos nombres:

- SQL Server 2000 Desktop Engine

- Microsoft Data Engine (MSDE) 2000

En los Libros en pantalla de SQL Server se hace referencia al componente como Desktop Engine. La información acerca de Desktop Engine también es aplicable a MSDE 2000.

Aunque la funcionalidad del motor de base de datos de SQL Server 2000 Desktop Engine es parecida a la del motor de las otras versiones de SQL Server, el tamaño de sus bases de datos no puede sobrepasar los 2 GB.

Tanto SQL Server 2000 Personal como SQL Server 2000 Desktop Engine disponen de un regulador de carga de trabajo simultáneo que limita el rendimiento del motor de la base de datos si se ejecutan más de 5 procesos por lote al mismo tiempo'[7].

[6] Idem
[7] Ibid pag. 5

CAPÍTULO 1

1.2 Arquitectura de la base de datos relacional

Los datos de Microsoft® SQL Server™ 2000 se almacenan en bases de datos. Los datos de una base de datos están organizados en los componentes lógicos visibles para los usuarios. Además, una base de datos está implementada físicamente como dos o más archivos de disco.

Al utilizar una base de datos, se trabaja principalmente con los componentes lógicos, como: tablas, vistas, procedimientos y usuarios. La implementación física de los archivos es casi enteramente transparente. Normalmente, sólo el administrador de la base de datos necesita trabajar con la implementación física.

'Cada instancia de SQL Server tiene cuatro bases de datos del sistema (master, model, tempdb y msdb) y tiene una o varias bases de datos de usuario. Algunas organizaciones sólo tienen una base de datos de usuario que contiene todos los datos de la organización. Otras organizaciones tienen bases de datos diferentes para cada grupo de la organización y, en algunas ocasiones, una base de datos sólo es utilizada por una única aplicación. Por ejemplo, una organización podría tener una base de datos para ventas, para nóminas, otra en aplicación de administración de documentos, etc. Algunas veces, la aplicación utiliza sólo una base de datos; otras aplicaciones pueden tener acceso a varias bases de datos'[8].

No es necesario ejecutar varias copias del motor de base de datos SQL Server para que varios usuarios puedan tener acceso a las bases de datos de un servidor. Una instancia de SQL Server Standard o Enterprise es capaz de controlar el trabajo de miles de usuarios sobre varias bases de datos al mismo tiempo. Cada instancia de SQL Server deja disponibles todas las bases de datos de la instancia para todos los usuarios que se conecten a la instancia, de acuerdo con los permisos de seguridad definidos.

Al conectarse a una instancia de SQL Server, la conexión se asocia a una base de datos concreta del servidor. A dicha base de datos se le llama base de datos actual. Normalmente, el usuario se conecta a una base de datos definida como la base de datos predeterminada por el administrador del sistema, aunque puede utilizar las opciones de conexión de las API de la base de datos para especificar otra base de datos. Puede cambiar de una base de datos a otra mediante la instrucción Transact-SQL USE *database_name* o mediante una función API que cambie el contexto de la base de datos actual.

SQL Server 2000 le permite separar las bases de datos de una instancia de SQL Server, volver a adjuntarlas a otra instancia e incluso adjuntar de nuevo la

[8] Apuntes 2° DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE; Modulo2 pag. 21

base de datos a la misma instancia. Si tiene un archivo de base de datos de SQL Server, al conectarse a SQL Server puede indicarle que adjunte a dicho archivo de base de datos un nombre de base de datos específico.

Para diseñar una base de datos es preciso comprender las funciones comerciales que desee modelar además de los conceptos y características de las bases de datos que se utilizan para representar esas funciones comerciales.

Es importante diseñar correctamente una base de datos para modelar las funciones comerciales, dado que la modificación posterior del diseño de una base de datos ya implementada puede ser muy laboriosa. Por otra parte, una base de datos bien diseñada ofrece un mejor rendimiento.

Al diseñar una base de datos, se considera lo siguiente:

- La finalidad de la base de datos y cómo afecta al diseño. Cree un plan para la base de datos que se ajuste a esa finalidad.
- Las reglas de normalización de la base de datos para evitar errores en el diseño de la base de datos.
- La protección de la integridad de los datos.
- Los requisitos de seguridad de la base de datos y los permisos de los usuarios.
- Los requisitos de rendimiento de la aplicación. Debe asegurarse que el diseño de la base de datos se beneficie con características de Microsoft® SQL Server™ 2000 que mejoran el rendimiento. El equilibrio entre el tamaño de la base de datos y la configuración del hardware es también importante para el rendimiento.
- El mantenimiento.
- La estimación del tamaño de la base de datos.

1.2.1 Bases de datos y datos del sistema

'Los sistemas Microsoft® SQL Server™ 2000 tienen cuatro bases de datos del sistema:

- Master: La base de datos master registra toda la información del sistema de SQL Server. Registra todas las cuentas de inicio de sesión y todas las opciones de configuración del sistema. La base de datos master registra la existencia del resto de bases de datos, incluida la ubicación de los archivos de base de datos. La base de datos master registra la información de inicialización de SQL Server, y siempre mantiene disponible una copia de seguridad reciente de master.

- Tempdb: La base de datos tempdb almacena todas las tablas y todos los procedimientos almacenados temporales. También satisface otras

necesidades de almacenamiento temporal, como las tablas de trabajo generadas por SQL Server. La base de datos tempdb es un recurso global; las tablas y los procedimientos almacenados temporales de todos los usuarios conectados al sistema se almacenan en ella; tempdb se vuelve a crear cada vez que se inicia SQL Server, de forma que el sistema se inicia con una copia limpia de la base de datos. Como las tablas y los procedimientos almacenados temporales se eliminan automáticamente al desconectar y cuando se cierra el sistema no hay conexiones activas, en tempdb nunca hay nada que se tenga que guardar de una sesión de SQL Server a otra. De forma predeterminada, tempdb crece automáticamente según sea necesario al mismo tiempo que se ejecuta SQL Server. Sin embargo, de forma distinta a otras bases de datos, se reinicia a su estado original cada vez que se inicia el motor de base de datos. Si el tamaño definido en tempdb es pequeño, parte de su carga de proceso del sistema puede llevarse con el crecimiento automático de tempdb hasta el tamaño necesario para admitir la carga de trabajo cada vez que se inicia SQL Server. Puede evitar esta sobrecarga si utiliza ALTER DATABASE para aumentar el tamaño de tempdb.

- model: La base de datos model se utiliza como plantilla para todas las bases de datos creadas en un sistema. Cuando se emite una instrucción CREATE DATABASE, la primera parte de la base de datos se crea copiando el contenido de la base de datos model, el resto de la nueva base de datos se llena con páginas vacías. Como tempdb se crea de nuevo cada vez que se inicia SQL Server, la base de datos model siempre tiene que existir en un sistema SQL Server.

- Msdb: La base de datos msdb la utiliza el Agente SQL Server para programar alertas y trabajos, además de registrar operadores'[9].

En SQL Server 2000 y SQL Server 7.0, cada base de datos, incluidas las bases de datos del sistema, tiene su propio conjunto de archivos y no los comparte con otras bases de datos.

Archivo de base de datos	Nombre del archivo físico.	Tamaño predeterminado, configuración típica
Datos principales de master	MASTER.MDF	11,0 MB
Registro de master	MASTLOG.LDF	1,25 MB

[9] Apuntes 2° DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE; Módulo2 pag. 23-24

Datos principales de tempdb	TEMPDB.MDF	8,0 MB
Registro de tempdb	TEMPLOG.LDF	0,5 MB
Datos principales de model	MODEL.MDF	0,75 MB
Registro de model	MODELLOG.LDF	0,75 MB
Datos principales de msdb	MSDBDATA.MDF	12,0 MB
Registro de msdb	MSDBLOG.LDF	2,25 MB

Los tamaños de estos archivos pueden variar ligeramente en las distintas ediciones de SQL Server 2000.

Cada base de datos de SQL Server 2000 contiene tablas del sistema que registran los datos que necesitan los componentes de SQL Server. El correcto funcionamiento de SQL Server depende de la integridad de información de las tablas del sistema, por lo que Microsoft no admite que los usuarios actualicen directamente la información de las tablas del sistema.

Microsoft proporciona un completo conjunto de herramientas administrativas que permite que los usuarios administren totalmente su sistema y administren todos los usuarios y los objetos de una base de datos. Los usuarios pueden utilizar las herramientas de administración, como el Administrador corporativo de SQL Server, para administrar directamente el sistema. Los programadores pueden utilizar la API SQL-DMO para incluir toda la funcionalidad de la administración de SQL Server en sus aplicaciones. Los programadores pueden utilizar los procedimientos almacenados del sistema e instrucciones DDL (Lenguaje de Definición de Datos) de Transact-SQL para admitir todas las funciones administrativas de sus sistemas.

Una importante función de SQL-DMO, los procedimientos almacenados del sistema y las instrucciones del lenguaje de definición de datos, es la de aislar las aplicaciones de los cambios de las tablas del sistema. En ocasiones, Microsoft tiene que cambiar las tablas del sistema en nuevas versiones de SQL Server para admitir las nuevas funciones que se agregan en dichas nuevas versiones. Las aplicaciones que utilicen instrucciones SELECT que hacen referencia directa a tablas del sistema suelen depender del formato anterior de las tablas del sistema. Puede que los sitios no puedan actualizar una versión de SQL a otra nueva hasta que todas las aplicaciones que seleccionen las tablas del sistema se hayan escrito otra vez. Microsoft tiene en cuenta los procedimientos almacenados del sistema, DDL y las interfaces SQL-DMO publicadas, e intenta mantener la compatibilidad con las versiones anteriores de dichas interfaces.

Microsoft no admite la definición de desencadenadores en las tablas del sistema; pueden alterar el funcionamiento del sistema.

Otra importante herramienta para consultar el catálogo de SQL Server es el conjunto de Vistas de esquema de información. Dichas vistas cumplen el esquema de información definido en el estándar SQL-92. Estas vistas proporcionan a las aplicaciones un componente basado en estándares para consultar el catálogo de SQL Server.

No se deben codificar instrucciones Transact-SQL que consulten directamente las tablas del sistema a menos que ése sea el único modo de obtener la información que requiere la aplicación. En la mayoría de los casos, las aplicaciones deberían obtener la información de catálogos y del sistema de:

- Las vistas de esquema de información SQL-92.
- SQL-DMO.
- La función de catálogo, métodos, atributos o propiedades de la API de datos utilizada en la aplicación, como ADO, OLE DB u ODBC.
- Procedimientos almacenados del sistema de Transact-SQL, instrucciones de catálogo y funciones integradas.

1.3 Arquitectura del Motor Base de Datos Relacional

El servidor es el componente de Microsoft® SQL Server™ 2000 que recibe instrucciones SQL de clientes y lleva a cabo todas las acciones necesarias para completar las instrucciones. Esta sección trata:

- Información general acerca de los componentes del servidor.
- Cómo el servidor compila cada lote de instrucciones SQL en un plan de ejecución que indica al servidor cómo procesar la instrucción.
- Cómo administra el servidor los recursos de Microsoft Windows® tales como memoria, subprocesos y tareas.
- Cómo el servidor determina qué parte de una consulta distribuida hace referencia a un servidor vinculado y qué petición se transmite al servidor para obtener los datos que se necesitan.
- Cómo el servidor transmite las llamadas de procedimientos almacenados remotos a los servidores remotos.

CAPÍTULO 1

- Cómo el servidor administra la simultaneidad y los problemas de transacciones.
- Cómo el servidor implementa los cursores del servidor.
- Las características que permiten a SQL Server ampliarse desde pequeños equipos portátiles a los grandes servidores que proporcionan el almacenamiento de datos primarios para las grandes organizaciones.
- Cómo el componente SQL Mail integra SQL Server con los servidores de correo electrónico para permitir que el servidor envíe correo y páginas cuando se produce un determinado suceso.

Los componentes de servidor de Microsoft® SQL Server 2000™ reciben instrucciones SQL de los clientes y las procesan.

El Flujo de datos tabulares, es donde los clientes envían las instrucciones SQL mediante un protocolo de nivel de aplicación específico de SQL Server llamado Flujo de datos tabulares (TDS, *Tabular Data Stream*). 'SQL Server 2000 acepta las siguientes versiones de TDS:

- TDS 8.0 enviado por clientes que ejecutan versiones de los componentes de cliente de SQL Server 2000. Los clientes de TDS 8.0 admiten todas las características de SQL Server 2000.
- TDS 7.0 enviado por clientes que ejecutan versiones de los componentes de cliente de SQL Server 7.0. Los clientes de TDS 7.0 no admiten las características introducidas por SQL Server 2000 y, a veces, el servidor tiene que ajustar los datos que devuelve a los clientes utilizando TDS 7.0. Por ejemplo, los clientes de TDS 7.0 no admiten el tipo de datos **sql_variant**; por tanto, SQL Server 2000 debe convertir los datos **sql_variant** a Unicode.
- TDS 4.2 enviado por clientes que ejecutan componentes de cliente de SQL Server 6.5, 6.0 y 4.21a. Los clientes de TDS 4.2 no admiten las características introducidas por SQL Server 2000 o SQL Server 7.0 y, a veces, el servidor tiene que ajustar los datos que devuelve a los clientes que utilizan TDS 4.2. Por ejemplo, los clientes de TDS 4.2 no admiten los tipos de datos Unicode, por lo que SQL Server 2000 debe convertir cualquier dato de tipo Unicode que envíe al cliente a datos de tipo carácter, con la posible pérdida de los caracteres extendidos. Los clientes de TDS 4.2 tampoco admiten valores de tipo **char**, **varchar**, **binary** o **varbinary** mayores de 255 bytes; por tanto SQL Server 2000 debe truncar cualquier valor mayor de 255 antes de enviarlo al cliente.
- Bibliotecas de red del servidor
- Los paquetes TDS se generan con el proveedor Microsoft OLE DB para SQL Server, el controlador ODBC de SQL Server o la DLL de DB-Library. Después, los paquetes TDS se pasan a una biblioteca de red del cliente SQL Server que encapsula los paquetes TDS en paquetes de protocolo de

red. En el servidor, los paquetes de protocolo de red son recibidos por una biblioteca de red de servidor que extrae el paquete TDS y lo pasa al servidor de la base de datos relacional.

- Este proceso se invierte cuando los resultados se devuelven al cliente.
- Motor de base de datos relacional'[10]

Cada servidor puede escuchar simultáneamente en varios protocolos de red y ejecutará una biblioteca de red de servidor por cada protocolo en el que esté escuchando. El servidor de base de datos procesa todas las peticiones que le pasan las bibliotecas de red de servidor. Compila todas las instrucciones SQL en planes de ejecución y, a continuación, utiliza los planes para tener acceso a los datos solicitados y crear el conjunto de resultados devuelto al cliente.

1.3.1 Componentes del Motor Relacional de Base de Datos

El servidor de base de datos relacional de Microsoft® SQL Server™ 2000 dispone de dos partes principales: el motor relacional y el motor de almacenamiento. Uno de los cambios de arquitectura más importantes realizados en SQL Server 7.0 fue la separación estricta de los componentes de motor relacional y de almacenamiento en el servidor, para hacer que al utilizar el API OLE DB puedan comunicarse el uno con el otro.

El procesamiento de una instrucción SELECT que hace referencia únicamente a tablas de bases de datos locales se puede resumir así:

- El motor relacional compila la instrucción SELECT en un plan de ejecución optimizado. El plan de ejecución define una serie de operaciones con conjuntos de filas básicos de tablas o índices individuales a los que se hace referencia en la instrucción SELECT.
- Un conjunto de filas es el término OLE DB para un conjunto de resultados. Los conjuntos de filas que solicita el motor relacional devuelven la cantidad de datos de una tabla o un índice que se necesita para realizar una de las operaciones que generarán el conjunto de resultados de la instrucción SELECT. Por ejemplo, esta instrucción SELECT requiere un recorrido de la tabla si hace referencia a una tabla sin índices:
- SELECT * FROM ScanTable
- El motor relacional implementa el recorrido de la tabla al solicitar un conjunto de filas que contenga todas las filas de ScanTable.
- Esta instrucción SELECT sólo necesita la información disponible en un índice:
- SELECT DISTINCT LastName FROM Northwind.dbo.Employees

[10] Apuntes 2° DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE; Modulo2 pag. 26-27

- El motor relacional implementa el recorrido del índice al solicitar un conjunto de filas que contiene las filas hoja del índice generado en la columna LastName.

Esta instrucción SELECT necesita la información de dos índices:

```
SELECT CompanyName, OrderID, ShippedDate
FROM Northwind.dbo.Customers AS Cst
JOIN Northwind.dbo.Orders AS Ord
ON (Cst.CustomerID = Ord.CustomerID)
```

- El motor relacional solicita dos conjuntos de filas, uno para el índice agrupado de Customers y otro para uno de los índices no agrupados de Orders.
- El motor relacional utiliza la API OLE DB para solicitar que el motor de almacenamiento abra los conjuntos de filas.
- Puesto que el motor relacional trabaja guiado por los pasos del plan de ejecución y necesita datos, utiliza OLE DB para recuperar las filas individuales de los conjuntos de filas que pidió al motor de almacenamiento que abriera. El motor de almacenamiento transfiere los datos desde los bufers de datos al motor relacional.
- El motor relacional combina los datos procedentes de los conjuntos de filas del motor de almacenamiento en el conjunto de resultados final que devuelve al usuario.

‘Las principales responsabilidades del motor relacional son:

- Analizar las instrucciones SQL.
- El analizador recorre una instrucción SQL y la separa en unidades lógicas tales como palabras clave, parámetros, operadores e identificadores. El analizador separa también toda la instrucción SQL en varias operaciones lógicas más pequeñas.
- Optimizar los planes de ejecución.
- Normalmente, hay muchas maneras en las que el servidor puede utilizar los datos de las tablas de origen para crear el conjunto de resultados. El optimizador de consultas determina cuáles son estos grupos de pasos, estima el costo de cada grupo (principalmente en términos de E/S de archivos) y elige el grupo de pasos con menor costo. Después, combina los pasos específicos con el árbol de la consulta para elaborar un plan de ejecución optimizado.
- Ejecutar el conjunto de operaciones lógicas definidas en el plan de ejecución.
- Una vez que el optimizador de consultas ha definido las operaciones lógicas necesarias para completar una instrucción, el motor relacional pasa por estas operaciones en la secuencia especificada en el plan de ejecución optimizado.

CAPÍTULO 1

- Procesar el lenguaje de definición de datos (DDL) y otras instrucciones.
- Estas instrucciones no son las instrucciones SELECT, INSERT, UPDATE o DELETE típicas, sino que tienen necesidades de procesamiento especiales. Ejemplos de esto son las instrucciones SET para establecer las opciones de conexión y las instrucciones CREATE para crear objetos en una base de datos.
- Resultados de formato.
- El motor relacional da formato a los resultados que se devuelven al cliente. Se da formato a los resultados como un conjunto de resultados tradicional o tabular, o como un documento XML. Después, los resultados se encapsulan en uno o más paquetes DTS y se devuelven a la aplicación.

Las principales responsabilidades del motor de almacenamiento incluyen:

- Administrar los archivos en los que se almacena la base de datos y administrar el uso del espacio de los archivos.
- Generar y leer las páginas físicas que se utilizan para almacenar los datos.
- Administrar los búferes de datos y toda la E/S hacia los archivos físicos.
- Controlar la simultaneidad. Administrar las transacciones y utilizar el bloqueo para controlar el acceso simultáneo de los usuarios a las filas de la base de datos.
- Registrar y recuperar.
- Implementar funciones de herramientas, como las instrucciones BACKUP, RESTORE, y DBCC, y la copia masiva.

SQL Server se instala con 33 idiomas naturales definidos en el servidor.

Las definiciones de cada idioma establecen cómo se interpretan los datos de las fechas:

- Los formatos en los que se presentan las fechas son:
    - dma (día, mes año)
    - mda (mes, día, año)
    - amd (año, mes, día)
- Nombres cortos y largos para cada mes.
- Nombres para cada día.
- Qué día se considera como el primero de la semana.
- Las definiciones de idioma se almacenan en **master.dbo.syslanguages** y cada idioma se identifica con un identificador (ID.) de idioma.
- Cada instancia de SQL Server utiliza un idioma predeterminado para todas las conexiones al servidor. Para obtener más información acerca de configurar las opciones, consulte **Opción default language.**
- La mayor parte de las conexiones utilizan el idioma predeterminado que se ha configurado para el servidor, pero cada conexión puede establecer individualmente un idioma de SQL Server para que lo utilice la conexión:

- Las aplicaciones de Microsoft ActiveX® Data Object y OLE DB pueden incluir la clave de idioma en una cadena de proveedor especificada cuando se conectan.
- Las aplicaciones OLE DB también pueden configurar la propiedad específica del proveedor SSPROP_INIT_CURRENTLANGUAGE antes de conectarse.
- Las aplicaciones de conectividad abierta de bases de datos (ODBC, *Open Database Connectivity*) pueden incluir la palabra clave LANGUAGE en una cadena de conexión que se especifica en **SQLDriverConnect**. Las aplicaciones ODBC también pueden especificar la configuración de idioma en una definición del origen de datos ODBC de SQL Server.
- Las aplicaciones de DB-Library pueden utilizar **dblogin** para asignar un registro de inicio de sesión y, a continuación, utilizar la macro DBSETNATLANG para especificar una configuración de idioma antes de llamar a **dbopen** para conectarse.
- Cualquier aplicación puede utilizar la instrucción SET LANGUAGE para especificar el idioma de SQL Server.

SQL Server admite también varias copias de mensajes de error específicas del idioma almacenado en **master.dbo.sysmessages**. Todas las instancias de SQL Server contienen el conjunto de mensajes en inglés. SQL Server está traducido al francés, alemán, español y japonés. Las instalaciones de las versiones traducidas de SQL Server instalan el conjunto de mensajes traducidos además del conjunto en inglés. Cuando SQL Server envía un mensaje a una conexión, utiliza el mensaje traducido si el Id del idioma de la conexión coincide con los Id del idioma presentes en **sysmessages**. Si no hay mensajes en **sysmessages** con el mismo Id del idioma, se envía la versión del mensaje en inglés'[11].

1.4 Ventajas de un sistema de Base de Datos de Servidor

El tener los datos almacenados y administrados en una ubicación central ofrece varias ventajas:

- Todos los elementos de datos están almacenados en una ubicación central donde todos los usuarios pueden trabajar con ellos.
- No se almacenan copias separadas del elemento en cada cliente, lo que elimina los problemas de hacer que todos los usuarios trabajen con la misma información. El sistema no necesita garantizar que todas las copias

[11] Apuntes 2° DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE; Modulo2 pag. 28-30

de los datos se actualicen con los valores actuales, porque hay una única copia en la ubicación central.

- Las reglas de empresa y de seguridad se pueden definir una sola vez en el servidor para todos los usuarios. La exigencia de las reglas se puede llevar a cabo en una base de datos utilizando restricciones, procedimientos almacenados y desencadenadores. Las reglas se pueden exigir también en una aplicación de servidor, puesto que estas aplicaciones son, asimismo, recursos centrales a los que tienen acceso muchos clientes reducidos.
- Los servidores de base de datos relacionales optimizan el tráfico de la red al devolver sólo los datos que la aplicación necesita.

  Por ejemplo, si una aplicación que trabaja con un servidor de archivos tiene que presentar la lista de los representantes comerciales de Oregón, debe obtener el archivo de empleados completo. Si la aplicación trabajara con un servidor de bases de datos relacionales, se enviaría este comando:

  SELECT firstname, lastname
  FROM employees
  WHERE Title = 'Sales Representative'
  AND Region = 'oregon'

  La base de datos relacional sólo devuelve los nombres de los representantes comerciales de Oregón y no toda la información de todos los empleados.

- Los gastos en hardware se pueden minimizar.

  Como los datos no están almacenados en los clientes, éstos no tienen que dedicar espacio de disco para almacenarlos. Los clientes tampoco necesitan la capacidad de procesamiento para administrar los datos localmente y el servidor no tiene que dedicar capacidad de procesamiento para presentar los datos.

  El servidor se puede configurar para optimizar la capacidad de E/S de disco necesaria para obtener los datos y los clientes se pueden configurar para optimizar el formato y presentación de los datos obtenidos desde el servidor.

  El servidor puede estar almacenado en una ubicación relativamente segura y estar equipado con dispositivos como sistemas de alimentación ininterrumpida, lo que resulta más económico que si se protegieran todos los clientes.

- Las tareas de mantenimiento como las copias de seguridad y restauración de los datos son más sencillas porque están concentradas en el servidor central.

CAPÍTULO 1

Microsoft SQL Server puede proporcionar los servicios de base de datos necesarios para sistemas extremadamente grandes. Los servidores de gran tamaño pueden tener miles de usuarios conectados a una instancia de SQL Server 2000 al mismo tiempo. SQL Server 2000 dispone de protección total para estos entornos, con medidas de seguridad que evitan problemas como tener varios usuarios intentando actualizar los mismos datos al mismo tiempo. SQL Server 2000 asigna también de manera muy eficaz los recursos disponibles, como memoria, ancho de banda de la red y E/S del disco, entre los distintos usuarios.

Los sitios de Internet extremadamente grandes pueden dividir sus datos entre varios servidores, extendiendo la carga de procesamiento entre varios equipos y permitiendo que el sitio sirva a miles de usuarios simultáneos.

En un único equipo se pueden ejecutar varias instancias de SQL Server 2000. Por ejemplo, una organización que proporcione servicios de base de datos a muchas otras organizaciones puede ejecutar una instancia de SQL Server 2000 para cada organización cliente, todas ellas en un solo equipo. Esto aísla los datos para cada organización cliente, a la vez que permite a la organización que ofrece el servicio reducir costos teniendo que administrar únicamente un equipo servidor.

Las aplicaciones de SQL Server 2000 se pueden ejecutar en el mismo equipo que SQL Server 2000. La aplicación se conecta a SQL Server 2000 utilizando los componentes de comunicaciones entre procesos (IPC) de Windows, tales como memoria compartida, en lugar de una red. Esto permite utilizar SQL Server 2000 en sistemas pequeños en los que una aplicación debe almacenar los datos localmente.

Los sitios Web de mayor tamaño y los sistemas de procesamiento de datos a nivel corporativo generan a menudo un mayor procesamiento de base de datos del que puede admitir un único equipo. En estos grandes sistemas, los servicios de base de datos vienen proporcionados por un grupo de servidores de base de datos que forman un nivel de servicios de base de datos. SQL Server 2000 no es compatible con una forma de equilibrio de carga de agrupación para crear un nivel de servicios de base de datos, pero sí admite un mecanismo que se puede utilizar para dividir los datos entre un grupo de servidores autónomos. Si bien cada servidor se administra individualmente, los servidores cooperan para expandir la carga de procesamiento de la base de datos entre el grupo. Un grupo de servidores autónomos que comparten una carga de trabajo se denomina federación de servidores.

CAPÍTULO 1

1.5 Modos de Autenticación de SQL Server

Cuando SQL Server 2000 se está ejecutando en Windows NT o Windows 2000, los miembros de la función fija del servidor **sysadmin** pueden especificar uno de los dos modos de autenticación:

- Modo de autenticación de Windows

  Sólo se permite la autenticación de Windows. Los usuarios no pueden especificar un ID de inicio de sesión de SQL Server 2000. Éste es el modo de autenticación predeterminado para SQL Server 2000. No puede especificar Modo de autenticación de Windows para una instancia de SQL Server que se esté ejecutando en Windows 98, puesto que el sistema operativo no es compatible con la autenticación de Windows.

- Modo mixto

  Si los usuarios proporcionan un ID de inicio de sesión de SQL Server 2000 cuando inician una sesión, se autentican utilizando la autenticación de SQL Server. Si no proporcionan un ID de inicio de sesión de SQL Server 2000 o si piden autenticación de Windows, se autentican mediante la autenticación de Windows 2000.

Dichos modos se especifican durante la instalación o en el Administrador corporativo de SQL Server.

1.6 Implementar una Base de Datos en SQL Server

Una base de datos de Microsoft® SQL Server™ 2000 consta de una colección de tablas que contienen datos y otros objetos, como vistas, índices, procedimientos almacenados y desencadenadores, que se definen para poder llevar a cabo distintas operaciones con datos. Los datos almacenados en una base de datos suelen estar relacionados con un tema o un proceso determinados como, por ejemplo, la información de inventario para el almacén de una fábrica.

SQL Server admite muchas bases de datos. Cada base de datos puede almacenar datos interrelacionados o sin relacionar procedentes de otras bases de datos. Por ejemplo, un servidor podría tener una base de datos que almacene datos del personal y otra que almacene datos relacionados con los productos. Por otra parte, puede utilizarse una base de datos para almacenar datos acerca de

pedidos actuales de los clientes y otra base de datos relacionada puede almacenar pedidos anteriores de los clientes que se utilicen para la elaboración de los informes anuales.

Antes de crear una base de datos, es importante entender las partes que la componen y cómo diseñarlas para asegurar que la base de datos funcione correctamente una vez implementada.

Importante Se recomienda no crear objetos de usuario, como tablas, vistas, procedimientos almacenados o desencadenadores en la base de datos master. La base de datos master contiene las tablas del sistema con la información del sistema que utiliza SQL Server, como por ejemplo los valores de las opciones de configuración.

1.6.1 Componentes de una base de datos SQL

Una base de datos de Microsoft® SQL Server™ 2000 consta de una colección de tablas en las que se almacena un conjunto específico de datos estructurados. Una tabla contiene una colección de filas (denominadas tuplas o registros) y columnas (denominadas atributos). Cada columna de la tabla se ha diseñado para almacenar un determinado tipo de información (por ejemplo, fechas, nombres, importes en moneda o números). Las tablas contienen diversos tipos de controles (restricciones, reglas, desencadenadores, valores predeterminados y tipos de datos personalizados por los usuarios) que garantizan la validez de los datos. Las tablas pueden presentar índices, similares a los de los libros, que permiten localizar las filas rápidamente. Se puede agregar restricciones de integridad referencial declarativa (DRI) a las tablas con el fin de asegurar la coherencia de los datos interrelacionados que se encuentran en tablas distintas. Asimismo, una base de datos puede almacenar procedimientos que utilicen código de programación de Transact-SQL para realizar operaciones con los datos que contiene la base de datos, por ejemplo, almacenar vistas que permiten el acceso personalizado a los datos de las tablas.

1.6.2 Crear un plan de base de datos

El primer paso en la creación de una base de datos consiste en elaborar un plan que sirva de guía para la implementación de la base de datos y, al mismo tiempo, de especificación funcional después de su implementación. La complejidad y los detalles de diseño de una base de datos dependen de la complejidad y el tamaño de la aplicación, así como de los usuarios.

La naturaleza y complejidad de una aplicación de la base de datos, así como el proceso de diseño, pueden variar considerablemente. Una base de datos puede ser relativamente sencilla y estar diseñada para que la utilice una sola

CAPÍTULO 1

persona, o bien ser grande y compleja, y estar diseñada para tratar, por ejemplo, las transacciones bancarias de cientos de miles de clientes. En el primer caso, el diseño de la base de datos puede consistir en poco más que unas anotaciones en un papel. En el segundo caso, el diseño puede ser un documento formal, con cientos de páginas que contengan todos y cada uno de los detalles relacionados con la base de datos.

Al diseñar la base de datos, independientemente de su tamaño y complejidad, se deben llevar a cabo los siguientes pasos básicos:

- Recopilar información.
- Identificar los objetos (tablas).
- Modelar los objetos.
- Identificar los tipos de información para cada objeto.
- Identificar las relaciones entre los objetos.

Antes de crear una base de datos, deberá conocer con detalle el cometido previsto para la base de datos. Si la base de datos va a reemplazar a un sistema de información manual o en papel, dicho sistema le proporcionará la mayor parte de la información que necesita. Es importante consultar a todas las personas involucradas en el sistema para saber lo que hacen y qué necesitan de la base de datos. También es importante identificar qué es lo que desean que haga el nuevo sistema, así como los problemas, las limitaciones y los puntos de congestión del sistema existente. Recopile copias de las instrucciones del cliente, listas de inventarios, informes de administración y, en general, de todos aquellos documentos que formen parte del sistema existente, porque le servirán para diseñar la base de datos y las interfaces.

Durante el proceso de recopilación de información, deberá identificar los objetos o las entidades más importantes que vayan a ser administrados por la base de datos. El objeto puede ser tangible (una persona o un producto, por ejemplo), o bien intangible (como una transacción empresarial, un departamento de una compañía o un período de liquidación de nóminas). Normalmente hay pocos objetos principales. Después de ser identificados, se evidencian los elementos relacionados. Cada elemento diferenciado en la base de datos debe tener su tabla correspondiente.

El objeto principal en la base de datos de ejemplo **pubs** incluida en Microsoft® SQL Server™ 2000 es una libreria. Los objetos relacionados con los libros en las operaciones empresariales de esta compañía son los autores que escriben los libros, los editores que los producen, las tiendas que los venden y las transacciones comerciales realizadas con las tiendas. Cada uno de estos objetos es una tabla de la base de datos.

CAPÍTULO 1

1.7 Estructura de un sistema completo

'Un sistema de bases de datos se divide en módulos que se encargan de cada una de las responsabilidades del sistema completo. Algunas de estas fusiones del sistema de base de datos las pueden proporcionar el sistema operativo de la computadora. En la mayoría de los casos, los sistemas operativos de la computadora proporcionan solo los servicios más básicos y los sistemas de bases de datos deben constituirse sobre esta base. Así, el dueño de una base de datos debe incluir consideraciones de la interfaz entre el sistema de bases de datos y el sistema operativo.

Los componentes funcionales de un sistema de bases de datos se pueden dividir a grandes rasgos en componentes de procesamiento de consultas y componentes de gestión de almacenamiento. Los componentes de procesamiento de consultas incluyen:

- Compilador del LMD (Lenguaje de Manipulación de Datos), que traduce las instrucciones del LMD en lenguaje de consultas a instrucciones a bajo nivel que entiende el motor de evaluación de consultas.

- Precompilador del LMD incorporado, que convierte las instrucciones del LMD incorporadas en un programa de aplicación en llamadas a procedimientos normales en el lenguaje anfitrión.

- Intérprete del LDD (Lenguaje de Definición de Datos), que interpreta las instrucciones del LDD y las registra en un conjunto de tablas que contienen metadatos.

- Motor de evaluación de consultas que ejecuta las instrucciones a bajo nivel generadas por el compilador del LMD.

Los componentes de gestión de almacenamiento (fig.1) proporcionan la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y envío de consultas al sistema. El gestor de almacenamiento tiene:

- Gestor de autorización e integridad que comprueba que se satisfagan las ligaduras de integridad y la autorización de los usuarios para acceder a los datos.

- Gestor de transacciones que asegura que la base de datos quede en un estado consistente a pesar de los fallos del sistema y que las ejecuciones de transacciones concurrentes ocurran sin conflictos.

CAPÍTULO 1

- Gestor de archivos que gestiona la reserva de espacio de almacenamiento de disco y las estructuras de datos usadas para representar la información almacenada en el disco.

- Gestor de memoria intermedia que es responsable de traer los datos del disco de almacenamiento a memoria principal y decidir que datos traer a memoria caché.

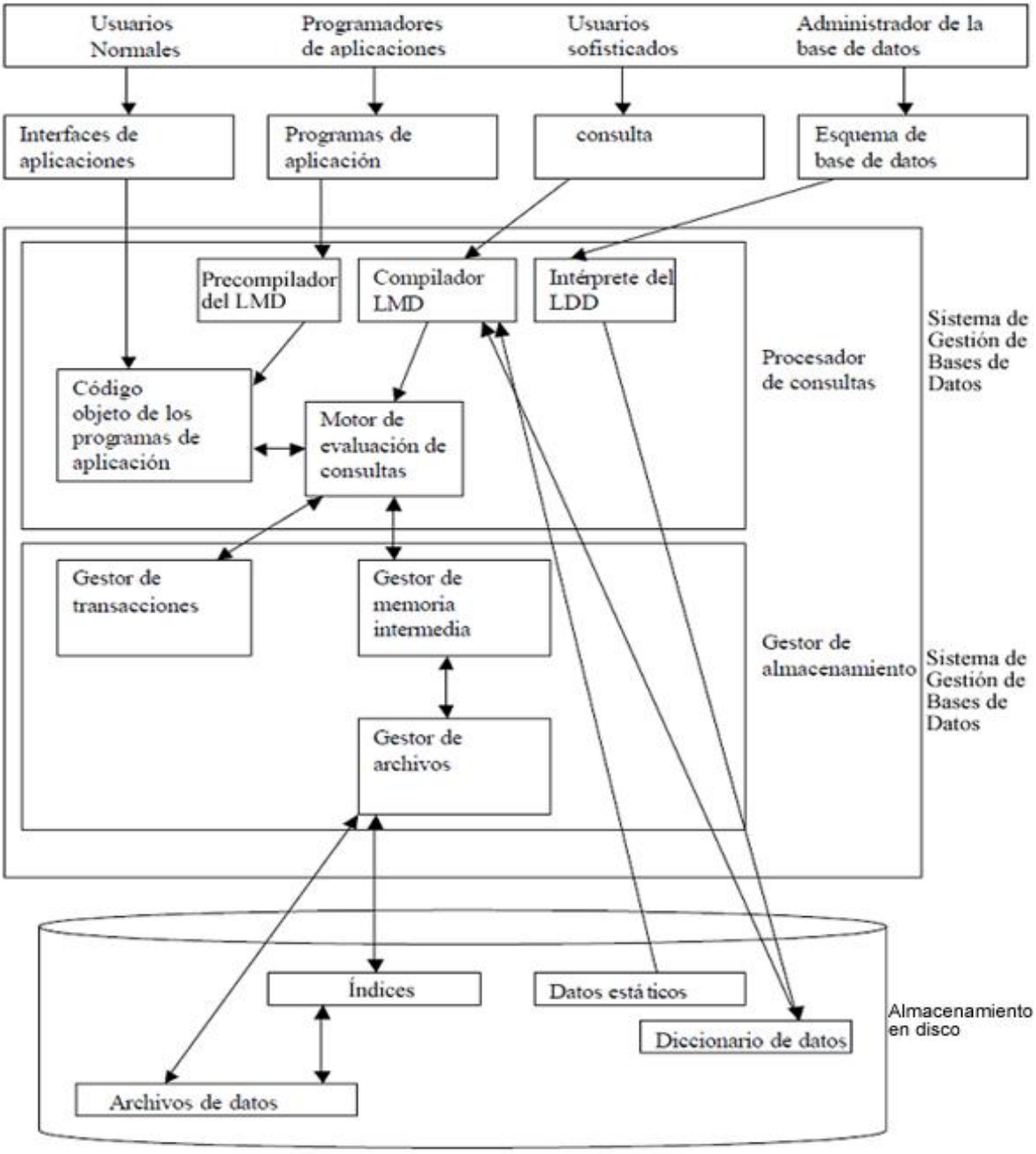


Figura 1. Esquema global de un sistema manejador de bases de datos.

También se necesitan diversas estructuras de datos como parte de la implementación física del sistema:

- Archivos de datos que almacenan la base de datos en sí.
- Diccionario de datos que almacena metadatos acerca de la estructura de la base de datos.
- Índices que proporcionan acceso rápido a elementos de datos que tienen valores particulares.
- Datos estáticos que almacenan información estadística sobre los datos en la base de datos, el procesador de consultas usa esta información para seleccionar las formas eficientes para ejecutar una consulta'[12].

Para el siguiente capítulo se mencionarán los requerimientos mínimos necesarios para crear el SCD-CE (Sistema de Control de Datos para una Casa de Empeño) tomando en cuenta las necesidades de hardware y software para el SQL Server 2000, detallando cada una de ellas con el fin de saber las diversas opciones que tenemos para crear sistemas de información con SQL Server 2000.

CAPÍTULO 1

[12] http://www.cs.buap.mx/~dpinto/bd/bdintro.pdf

CAPÍTULO 2

2. Análisis de requerimientos para el SCD-CE

Para la realización de cualquier sistema es fundamental realizar un estudio en cuanto a los requerimientos funcionales (tareas que se desean realizar) y los no funcionales como los requerimientos de hardware y software, que se requiere para la realización del proyecto, ya que de esto dependerá el óptimo funcionamiento del sistema.

2.1 Requerimientos funcionales

Para ésta parte se enlistarán las funcionalidades que el sistema debe de realizar:

1. El sistema deberá permitir el registro de todas las prendas u objetos que sean aceptadas por el valuador.

2. El sistema deberá evaluar si el cliente existe o no, para evitar datos repetidos.

3. La Base de Datos debe permitir realizar las tareas de empeño, desempeño, refrendo y ventas.

4. Deducir automáticamente el valor de los campos *fecha de entrada* y *fecha de vencimiento*, además de los costos por concepto de desempeño y refrendo, para agilizar la captura de los datos.

5. El sistema deberá de ser capaz de actualizar automáticamente el campo de *estado* de cada una de las prendas, a los diferentes estados (vencido, refrendado, baja vendido, cargo y en venta) según sea el caso, para agilizar la actualización de estados de los objetos.

6. Se requerirá que el sistema pueda generar reportes de las prendas para cada uno de sus estados, para tener un control de las diferentes operaciones que se realizan.

2.2 Requerimientos no funcionales:

2.2.1 Características del hardware

- Características del procesador:
  - ✓ Procesador Intel o compatible Pentium, Pentium Pro o Pentium II en adelante, la velocidad mínima del procesador puede ser de 166 MHz.

- Características de memoria (RAM):
  - ✓ Enterprise — 64 MB como mínimo; se recomiendan 128 MB.
  - ✓ Standard — 64 MB como mínimo.
  - ✓ Personal — 64 MB en Windows 2000, 32 MB en otros sistemas operativos.
  - ✓ Developer — 64 MB como mínimo.
  - ✓ Desktop Engine — 64 como mínimo en Windows 2000, 32 MB en otros sistemas operativos.

- Los requisitos de disco duro para SQL Server 2000 varían en función de los componentes y la opción de instalación seleccionados:
  - ✓ Componentes de bases de datos de 95 a 270 MB, 250 MB típica.
  - ✓ Analysis Services 50 MB mínima, 130 MB típica.
  - ✓ English Query 80 MB.
  - ✓ Sólo Desktop Engine 44 MB.

- SQL Server 2000 requiere un monitor con resolución VGA. Las herramientas gráficas de SQL Server requieren un monitor con una resolución de 800x600 o superior.

- SQL Server 2000 requiere una unidad de CD-ROM y Microsoft Mouse o compatible.

2.2.2 Características del software

El SCD-CE será realizado con SQL Server 2000 y para esto ya se especificó en el punto anterior los requerimientos específicos para este software pero también es importante saber es que SO (sistema operativo) es posible instalar SQL Server 2000 por lo que se enlistan los siguientes SO:

- ✓ Windows XP Service Pack 2
- ✓ Windows 2000
- ✓ Microsoft Windows NT versión 4.0 Service Pack 5 o posterior
- ✓ Windows Millennium
- ✓ Windows 98
- ✓ Windows 95 (sólo la opción de conectividad de clientes)

- Además SQL Server 2000 requiere Internet Explorer 5.0 o posterior.

CAPÍTULO 3

3. Diseño del SCD-CE

En la realización de sistemas de información es muy importante que antes de crear la BD se tenga un análisis profundo de la estructura que tendrá nuestro sistema como por ejemplo la realización de un modelo entidad relación (E/R) que nos permite crear un bosquejo de lo que será la estructura de la BD, éste tipo de diseño previo fue propuesto por 'Chen a mediados de los años setenta como medio de representación conceptual de los problemas y para representar la visión de un sistema de forma global. Físicamente adopta la forma de un grafo[13] escrito en papel al que se denomina **diagrama Entidad-Relación**'[14]. Sus elementos fundamentales son las entidades y las relaciones.

- 'Una **entidad** caracteriza a un tipo de objeto, real o abstracto, del problema a modelizar. Toda entidad tiene existencia propia, es distinguible del resto de las entidades, tiene nombre y posee **atributos** definidos en un dominio determinado. Una entidad es todo aquello de lo que se desea almacenar información. En el diagrama E-R las entidades se representan mediante rectángulos.
- Una **relación** es una asociación o relación matemática entre varias entidades. Las relaciones también se nombran. Se representan en el diagrama E-R mediante flechas y rombos. Cada entidad interviene en una relación con una determinada **cardinalidad**. La cardinalidad (número de instancias o elementos de una entidad que pueden asociarse a un elemento de la otra entidad relacionada) se representa mediante una pareja de datos, en minúsculas, de la forma *(cardinalidad mínima, cardinalidad máxima)*, asociada a cada uno de las entidades que intervienen en la relación. Son posibles las siguientes cardinalidades: *(0,1), (1,1), (0,n), (1,n), (m,n)*. También se informa de las cardinalidades máximas con las que intervienen las entidades en la relación.

El **tipo de relación** se define tomando los máximos de las cardinalidades que intervienen en la relación. Hay cuatro tipos posibles:

1. Una a una (1:1). En este tipo de relación, una vez fijado un elemento de una entidad se conoce la otra. Ejemplo: nación y capital.
2. Una a muchas (1:N). Ejemplo: cliente y pedidos.
3. Muchas a una (N:1). Simetría respecto al tipo anterior según el punto de vista de una u otra entidad.
4. Muchas a muchas (N:N). Ejemplo: personas y viviendas.

[13] Ver GLOSARIO.
[14] http://www.cs.us.es/cursos/bd-2001/temas/diseno.html

CAPÍTULO 3

Toda entidad debe ser unívocamente identificada y distinguible mediante un conjunto de atributos (quizás un solo atributo) denominado **identificador** o **clave principal o primaria**. Puede haber varios posibles identificadores para una misma entidad, en cuyo caso se ha de escoger uno de ellos como identificador principal siendo el resto identificadores alternativos.

Hay unas normas de sentido común a seguir cuando se dibuja un diagrama E-R: emplear preferentemente líneas rectas en las relaciones y evitar en lo posible que estas líneas se crucen. Se suele usar nombres para describir las entidades y verbos para las relaciones. Esto es lógico ya que las entidades se ponen en común cuando se realiza alguna acción. Los verbos empleados no necesariamente tienen que ser siempre infinitivos'[15].

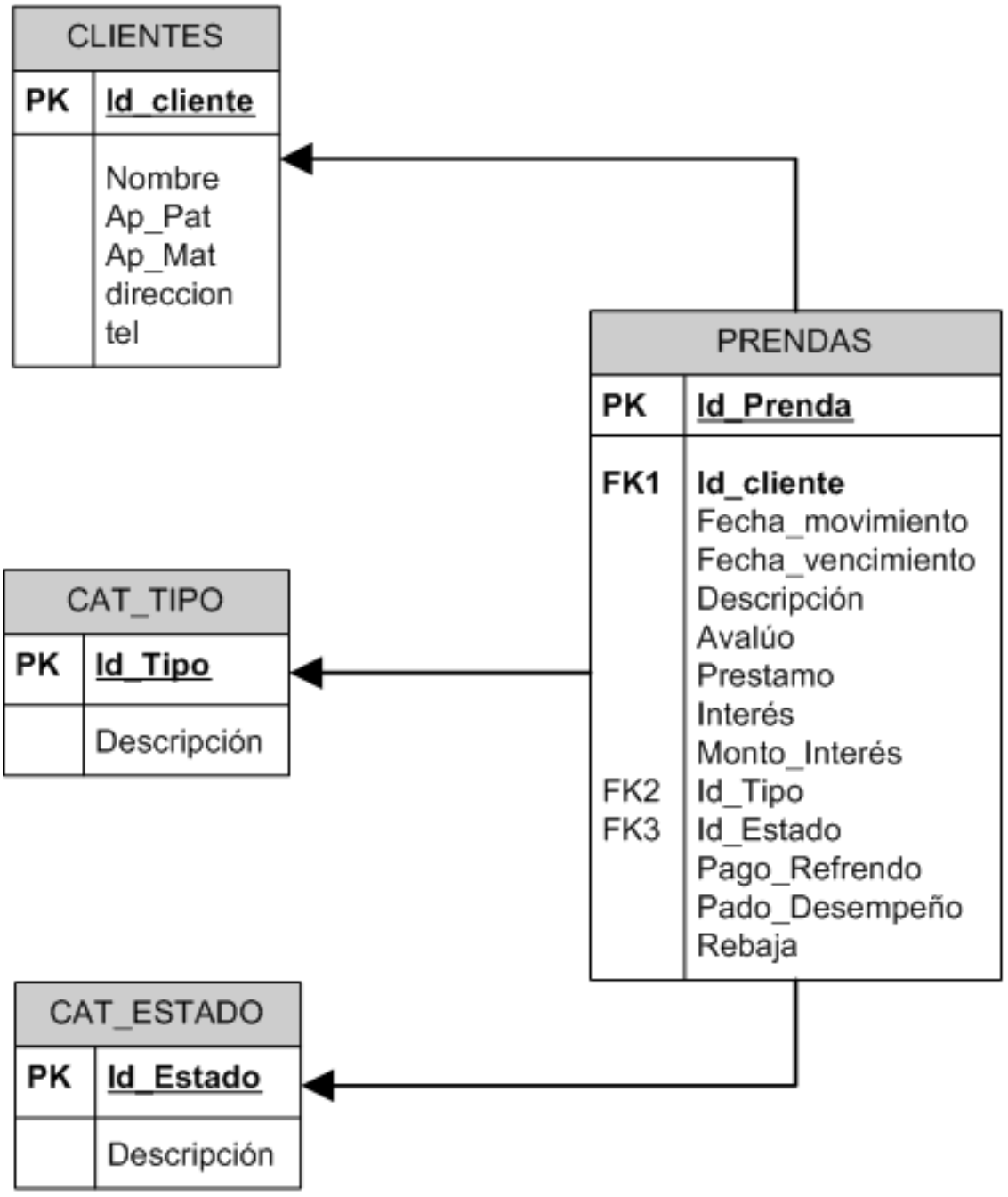
3.1 Modelo Entidad Relación (E/R)

El siguiente modelo E/R representa la manera en que está estructurada la base de datos para este sistema el cual nos muestra que habrá una serie de catálogos que servirán como referencias para que no se introduzca información que no este definida o en su defecto permitida, uno de esos catálogos es el catálogo de tipo de prendas (CAT_TIPO) que contiene los tipo de prendas que pueden almacenarse o de alguna manera establece también la clasificación a la que pertenece cada una de las prendas que podrán recibirse en la empresa, ya sea por ejemplo una cama que entraría en la clasificación de *muebles*, una colección de cd's o una moto que entrarían en la clasificación de *varios*, un juego de aretes que entraría en la clasificación de *alajas* y por otra parte los electrodomésticos que es una clasificación en donde entrarían las televisiones, modulales, refrigeradores, etc; también existe un catálogo de estados (CAT_ESTADO), que sirve para establecer las diferentes etapas en que las prendas podrán estar ya sea empeñadas, refrendadas, desempeñadas, vencidas o vendidas y poder así saber que se puede hacer con cada una de ellas dependiendo de su estado; además se tiene la tabla clientes que de acuerdo a su definición está hecha con el fin de almacenar información de los clientes pero a su ves nos servirá para tener un catalogo de clientes que conforme a su definición esto permitirá que un cliente pueda tener muchas prendas ya que ésta tabla a su ves se encuentra ligada a la tabla prendas que de acuerdo a su diseño contiene o esta relacionada a un y solo un cliente, por lo que una prenda solo puede tener un cliente y un cliente puede contener muchas prendas. Cabe señalar que cada uno de los campos se encuentra definido en el diccionario de datos.

[15] Ibid pag. 27

CAPÍTULO 3

MODELO ENTIDAD RELACIÓN



CAPÍTULO 3

3.2 Diccionario de Datos

NOMBRE	ALIAS	TIPO	TAMAÑO	DESCRIPCIÓN
Clave del cliente	Id_Cliente	int	11	Sirve como identificador único para cada uno de los clientes.
Nombre del cliente	Nombre	varchar	10	Es el nombre del cliente que sirve para tener un registro de él.
Apellido paterno	Ap_pat	varchar	10	Complementa al nombre del cliente.
Apellido materno	Ap_mat	varchar	10	Complementa al nombre del cliente.
Dirección del cliente	Direccion	varchar	50	Permite saber el domicilio del cliente.
Teléfono del cliente	Tel	int	11	Número telefónico en donde se puede localizar al cliente.
Identificador de la prenda	Id_Prenda	int	11	Sirve como identificador único para cada una de las prendas.
Fecha en que se realizó el movimiento	Fecha_movimiento	varchar	11	Es importante para saber los movimientos del día y para determinar la fecha de vencimiento.
Fecha en que se vencerá la prenda	Fecha_vencimiento	varchar	11	Es fundamental para los reportes de prendas vencidas y que pueden pasar a la venta; o en su defecto que pasa a cargo.
Características de la prenda y estado en que se encuentra	Descripción	varchar	50	Es importante para el cálculo del avalúo, y nos da una descripción esencial de la prenda.
Precio actual estimado	Avalúo	int	11	Es el precio que se le asigna a la prenda según sus características y estado
Préstamo	Préstamo	int	11	Es el 50% del avalúo, y es lo que se le presta al cliente.
Interés (impuesto)	Interés	Smallin	6	Es el porcentaje de impuesto que se le

				añade al préstamo.
Monto con impuesto	Monto interés	int	11	Es la cantidad que se le incrementará al préstamo.
Clasificación de la prenda	Tipo	varchar	1	Es el campo que especificará el tipo de prenda, ya sea alhaja, mueble, electrodoméstico ó varios.
Estado en que se encuentra la prenda	Estado	varchar	1	Muestra si la prenda esta actualmente empeñada, desempeñada, refrendada, vendida o paso a cargo.
Refrendo	Pago_Refrendo	int	11	Es la cantidad a pagar para obtener una prorroga y que la prenda no se venza.
Desempeño	Pago_Desempeño	int	11	Es la cantidad a paga para rescatar la prenda.
Descuento	Rebaja	Smallint	6	Porcentaje que se descontará a la prenda mensualmente cuando ésta, este en venta.
Identificador del estado	Id_estado	varchar	1	Es un identificador único del catalogo estados.
Descripción del estado	Descripción	varchar	20	Es la etiqueta o nombre del estado.
Identificador del tipo	Id_tipo	varchar	1	Es un identificador único del catalogo tipo.
Descripción del tipo	Descripción	varchar	20	Es la etiqueta o nombre del tipo.

CAPÍTULO 4

4. Desarrollo del SCD-CE

Para la creación de éste sistema se emplearon diversos comandos del lenguaje de programación de SQL Server denominado **Transact-SQL** tales como triggers, cursores, funciones y procedimientos almacenado que fueron esenciales para la funcionalidad que se buscaba para éste proyecto.

Los siguientes puntos explicarán la manera en que cada parte del sistema fue desarrollado, mencionando desde su diseño hasta el código empleado para su creación, tomando en cuenta las partes funcionales de una casa de empeño para poder lograr la emulación de éstas tareas mediante el programa SQL Server 2000.

4.1 Creación de la BD

La construcción de la BD en el SQL Server, es crear una nueva base de datos y los archivos que se utilizarán para el almacenamiento de la base de datos propia. Para la realización de la base de datos mediante las herramientas del programa SQL Server 2000 es necesario realizar los siguientes pasos:

- En primer lugar se abre el SQL Enterprise Manager, para después desglosar el árbol de la consola de root y colocar el cursor sobre Databases para que ahí mismo con un clic derecho se seleccione New Database como se muestra en la figura 4.1:

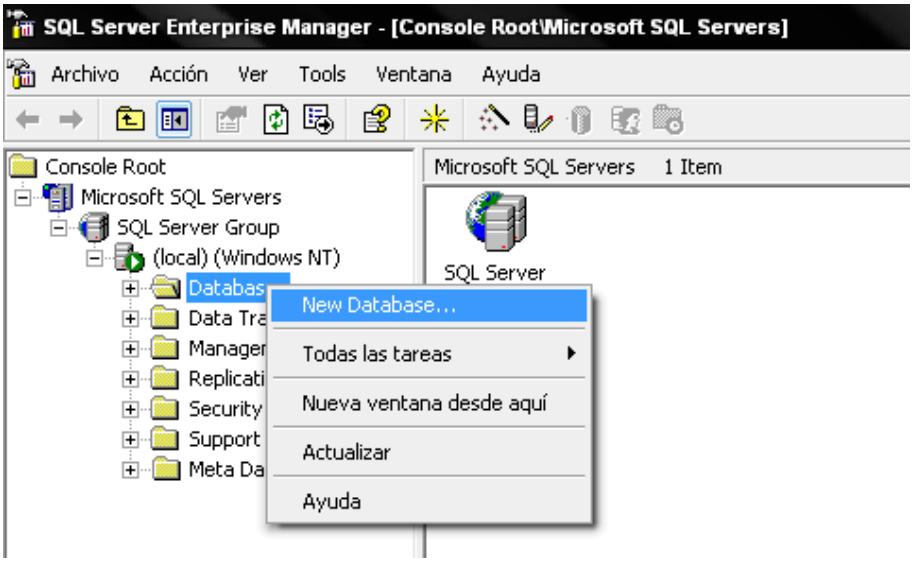


Figura 4.1 (creación de una Base de Datos nueva)

- Después solamente se introduce el titulo de la base de datos como se describe en la siguiente imagen (Figura 4.2):

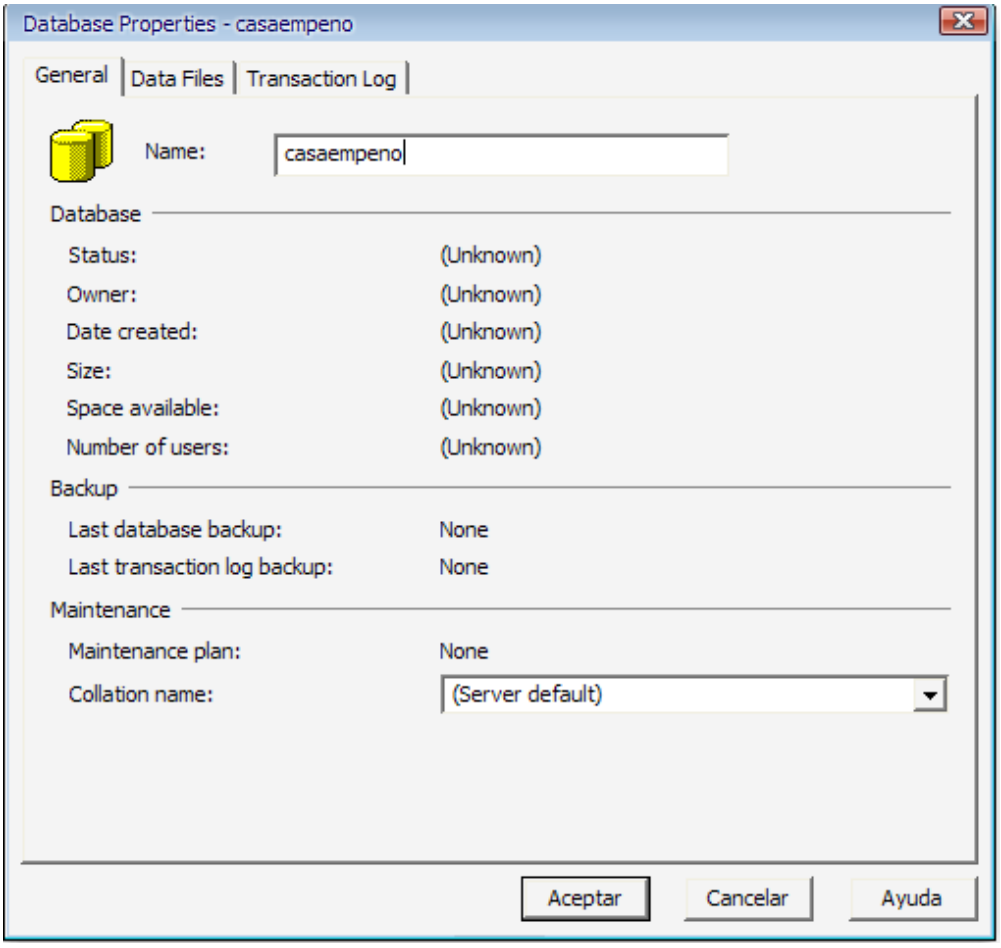


Figura 4.2 (propiedades de la Base de Datos)

De esta manera la base de datos (espacio ó compartimiento de datos) se genera y se puede visualizar que la base de datos fue creada de la siguiente forma:

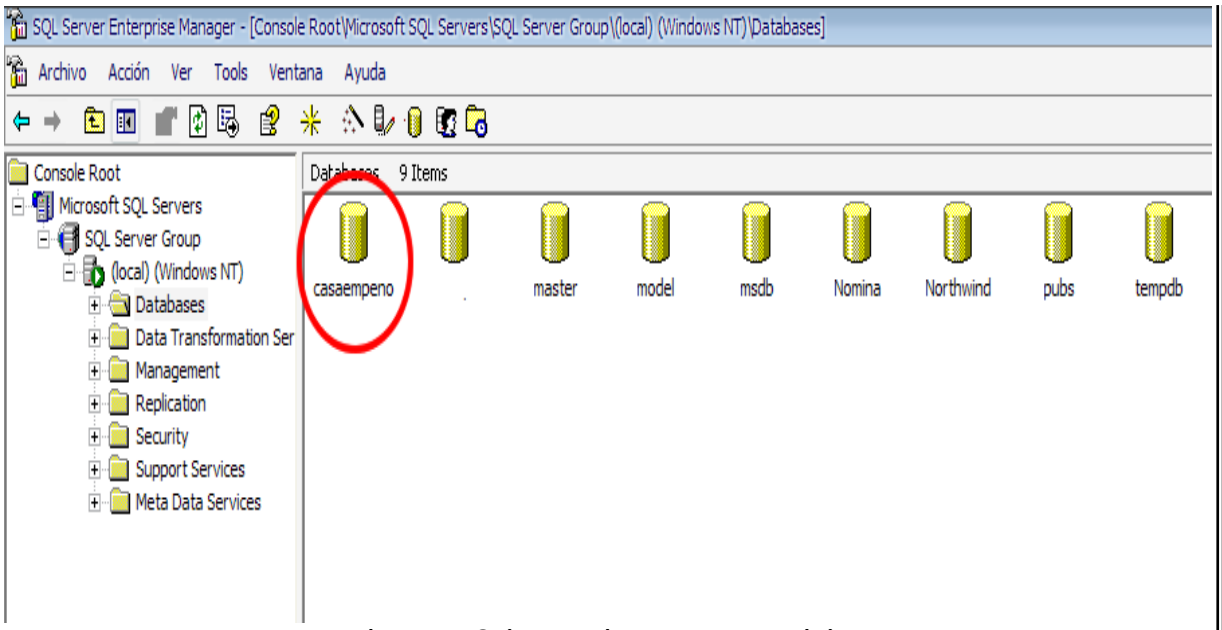


Figura 4.3 (Base de Datos creada)

Otra forma de generar la BD podría ser mediante la instrucción SQL “create database” desde el Query Analyzer como se muestra en la siguiente imagen (Figura 4.4) y en donde se visualiza la ejecución del comando y se muestra la manera en que se puede visualizar que la BD fue generada:

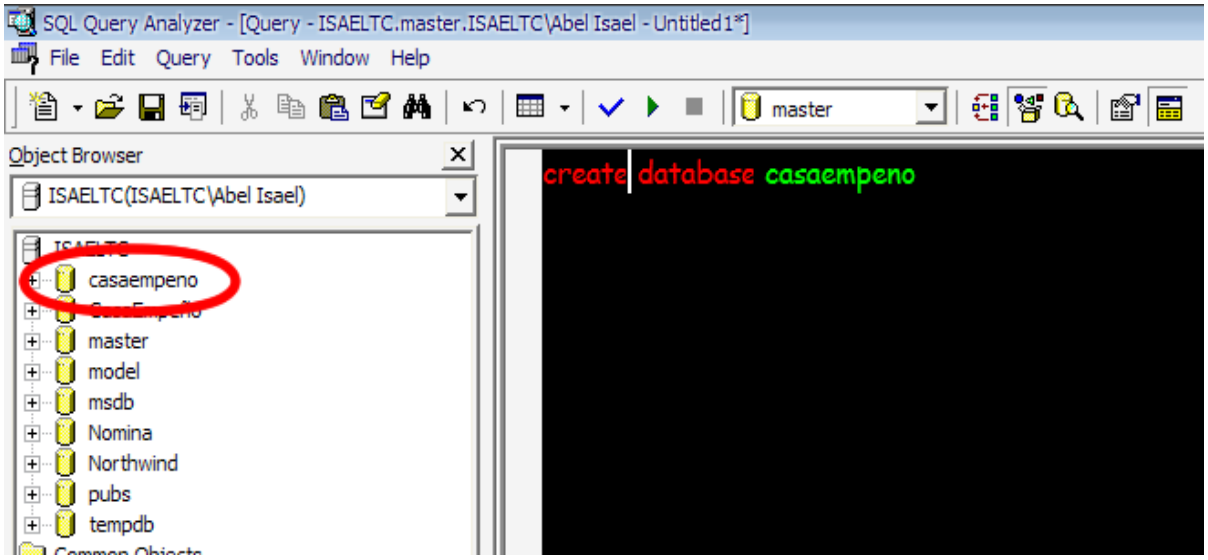


Figura 4.4 (creación de la Base de Datos con código SQL)

La sintaxis completa de la instrucción CREATE DATABASE es la siguiente:

CAPÍTULO 4

```
CREATE DATABASE database_name
    [ ON
        [ PRIMARY ] [ <filespec> [ ,...n ]
        [ , <filegroup> [ ,...n ] ]
    [ LOG ON { <filespec> [ ,...n ] } ]
    ]
    [ COLLATE collation_name ]
    [ WITH <external_access_option> ]
]
[;]

To attach a database
CREATE DATABASE database_name
    ON <filespec> [ ,...n ]
    FOR { ATTACH [ WITH <service_broker_option> ]
        | ATTACH_REBUILD_LOG }
[;]

<filespec> ::=
{
(
    NAME = logical_file_name ,
        FILENAME = { 'os_file_name' | 'filestream_path' }
        [ , SIZE = size [ KB | MB | GB | TB ] ]
        [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] |
UNLIMITED } ]
        [ , FILEGROWTH = growth_increment [ KB | MB | GB | TB
| % ] ]
) [ ,...n ]
}

<filegroup> ::=
{
FILEGROUP filegroup_name [ CONTAINS FILESTREAM ] [ DEFAULT ]
    <filespec> [ ,...n ]
}

<external_access_option> ::=
{
  [ DB_CHAINING { ON | OFF } ]
  [ , TRUSTWORTHY { ON | OFF } ]
}
<service_broker_option> ::=
{
    ENABLE_BROKER
  | NEW_BROKER
  | ERROR_BROKER_CONVERSATIONS
```

```
}

Create a database snapshot
CREATE DATABASE database_snapshot_name
    ON
        (
        NAME = logical_file_name,
        FILENAME = 'os_file_name'
        ) [ ,...n ]
    AS SNAPSHOT OF source_database_name
[;]
```

Argumentos

database_name

Es el nombre de la nueva base de datos. Los nombres de base de datos deben ser únicos en una instancia de SQL Server y cumplir las reglas de los identificadores.

database_name puede tener 128 caracteres como máximo, a menos que no se especifique un nombre lógico para el archivo de registro. Si no se especifica un nombre de archivo de registro lógico, SQL Server genera *logical_file_name* y *os_file_name* para el registro, anexando un sufijo a *database_name*. Esto limita *database_name* a 123 caracteres, por lo que el nombre de archivo lógico generado tiene como máximo 128 caracteres.

Si no se especifica un nombre de archivo de datos, SQL Server utiliza *database_name* como *logical_file_name* y *os_file_name*.

ON

Especifica que los archivos de disco utilizados para almacenar las secciones de datos de la base de datos (archivos de datos) se definen explícitamente. ON es obligatorio cuando va seguido de una lista de elementos <filespec> separados por comas que definen los archivos de datos del grupo de archivos principal. Detrás de la lista de archivos del grupo de archivos principal se puede colocar una lista opcional de elementos <filegroup> separados por comas que definan los grupos de archivos de usuario y sus archivos.

PRIMARY

Especifica que la lista de elementos <filespec> asociada define el archivo principal. El primer archivo especificado en la entrada <filespec> del grupo

de archivos principal se convierte en el archivo principal. Una base de datos sólo puede tener un archivo principal.

Si no se especifica PRIMARY, el primer archivo enumerado en la instrucción CREATE DATABASE se convierte en el archivo principal.

LOG ON

Especifica que los archivos de disco utilizados para almacenar el registro de la base de datos (archivos de registro) se definen explícitamente. LOG ON va seguido de una lista de elementos <filespec> separados por comas que definen los archivos de registro. Si no se especifica LOG ON, se crea automáticamente un archivo de registro cuyo tamaño es el 25 por ciento de la suma de los tamaños de todos los archivos de datos de la base de datos, o 512 KB, lo que sea mayor. LOG ON no se puede especificar en una instantánea de base de datos.

COLLATE *collation_name*

Especifica la intercalación predeterminada de la base de datos. El nombre de la intercalación puede ser un nombre de intercalación de Windows o un nombre de intercalación de SQL. Si no se especifica, se asigna a la base de datos la intercalación predeterminada de la instancia de SQL Server. No se puede especificar un nombre de intercalación en una instantánea de base de datos.

No se puede especificar un nombre de intercalación con las cláusulas FOR ATTACH o FOR ATTACH_REBUILD_LOG.

FOR ATTACH [WITH <service_broker_option>]

Especifica que la base de datos se crea adjuntando un conjunto existente de archivos de sistema operativo. Debe haber una entrada <filespec> que especifique el archivo principal. Las demás entradas <filespec> que son necesarias son las correspondientes a los archivos con una ruta de acceso diferente de la que tenían cuando la base de datos se creó por primera vez o se adjuntó por última vez. Debe especificarse una entrada <filespec> para estos archivos.

FOR ATTACH tiene los siguientes requisitos:

- Todos los archivos de datos (MDF y NDF) deben estar disponibles.
- Si hay varios archivos de registro, todos ellos deben estar disponibles.

Si una base de datos de lectura/escritura tiene un único archivo de registro que no está disponible actualmente y si la base de datos se cerró sin

usuarios o transacciones abiertas antes de la operación de adjuntar, FOR ATTACH regenera automáticamente el archivo de registro y actualiza el archivo principal. En cambio, en el caso de una base de datos de sólo lectura, el registro no se regenera, ya que el archivo principal no se puede actualizar. Por tanto, cuando se adjunta una base de datos de sólo lectura cuyo registro no está disponible, es necesario suministrar el archivo o los archivos de registro en la cláusula FOR ATTACH.

Nota:

Una base de datos creada por una versión más reciente de SQL Server no puede adjuntarse en versiones anteriores. La versión de la base de datos de origen debe ser como mínimo la versión 80 (SQL Server 2000) para adjuntar a SQL Server 2008. Al adjuntar bases de datos SQL Server 2000 o SQL Server 2005 con un nivel de compatibilidad inferior a 80, se establecerán con una compatibilidad de 80.

En SQL Server, los archivos de texto completo que formen parte de la base de datos que se va a adjuntar, se incluirán con la base de datos. Para especificar una nueva ruta de acceso al catálogo de texto, escriba la nueva ubicación sin incluir el nombre de archivo de texto del sistema operativo.

FOR ATTACH no se puede especificar en una instantánea de base de datos.

Si la base de datos utiliza Service Broker, use WITH

<service_broker_option> en la cláusula FOR ATTACH:

<service_broker_option>

Controla la entrega de mensajes de Service Broker y el identificador de Service Broker para la base de datos. Las opciones de Service Broker solo se pueden especificar cuando se utiliza la cláusula FOR ATTACH.

ENABLE_BROKER

Indica que se habilite Service Broker para la base de datos especificada. En otras palabras, se inicia la entrega de mensajes y se establece **is_broker_enabled** en True en la vista de catálogo **sys.databases**. La base de datos conserva el identificador de Service Broker existente.

NEW_BROKER

Crea un valor **service_broker_guid** en **sys.databases** y en la base de datos restaurada y finaliza todos los extremos de conversación con limpieza.

El broker se habilita, pero no se envía ningún mensaje a los extremos de conversación remotos. Cualquier ruta que haga referencia al identificador de Service Broker anterior debe volverse a crear con el nuevo identificador.

ERROR_BROKER_CONVERSATIONS

Finaliza todas las conversaciones con un error que indica que la base de datos está adjunta o restaurada. El broker está deshabilitado hasta que finaliza esta operación y, después, se habilita. La base de datos conserva el identificador de Service Broker existente.

Cuando adjunte una base de datos replicada que se copió en lugar de separarse, considere los siguientes puntos:

- Si adjunta la base de datos a la misma versión e instancia de servidor que la base de datos original, no se requieren pasos adicionales.
- Si adjunta la base de datos a la misma instancia de servidor, pero con una versión actualizada, debe ejecutar sp_vupgrade_replication para actualizar la replicación una vez completada la operación de adjuntar.
- Si adjunta la base de datos a una instancia de servidor diferente, sin importar la versión, debe ejecutar sp_removedbreplication para quitar la replicación una vez completada la operación de adjuntar.

Nota:

Adjunte los trabajos con el formato de almacenamiento **vardecimal**, pero el SQL Server Database Engine (Motor de base de datos de SQL Server) se debe actualizar al menos al Service Pack 2 de SQL Server 2005. No puede adjuntar ninguna base de datos que utilice el formato de almacenamiento vardecimal a una versión anterior de SQL Server.

Nota de seguridad:

Se recomienda no adjuntar bases de datos de orígenes desconocidos o que no sean de confianza. Estas bases de datos pueden contener código dañino que podría ejecutar código Transact-SQL no deseado o provocar errores debido a la modificación del esquema o de la estructura de la base de datos física. Antes de utilizar una base de datos de un origen desconocido o que no sea de confianza, ejecute DBCC CHECKDB en la base de datos en un servidor que no sea de producción y examine el código de la base de datos, como procedimientos almacenados u otro código definido por el usuario.

FOR ATTACH_REBUILD_LOG

Especifica que la base de datos se crea adjuntando un conjunto existente de archivos de sistema operativo. Esta opción está limitada a las bases de datos de lectura/escritura. Si no se encuentran uno o varios archivos de registro de transacciones, se vuelve a generar el archivo de registro. Debe haber una entrada <*filespec*> que especifique el archivo principal.

Nota:

Si los archivos de registro están disponibles, el Database Engine (Motor de base de datos) utilizará esos archivos en lugar de volver a generar los archivos de registro.

FOR ATTACH_REBUILD_LOG tiene los requisitos siguientes:

- Un cierre limpio de la base de datos.
- Todos los archivos de datos (MDF y NDF) deben estar disponibles.

Importante:

Esta operación interrumpe la cadena de copias de seguridad del registro. Se recomienda realizar una copia de seguridad completa de la base de datos después de finalizar la operación.

Normalmente, FOR ATTACH_REBUILD_LOG se utiliza cuando se copia una base de datos de lectura/escritura con un registro grande en otro servidor donde la copia se va a utilizar en la mayoría de los casos o únicamente para operaciones de lectura y, por tanto, necesitará menos espacio de registro que la base de datos original.

FOR ATTACH_REBUILD_LOG no se puede especificar en una instantánea de base de datos.

<filespec>

Controla las propiedades de archivo.

NAME *logical_file_name*

Especifica un nombre lógico para el archivo. NAME es obligatorio si se especifica FILENAME, excepto cuando se especifica una de las cláusulas FOR ATTACH. Un grupo de archivos FILESTREAM no se puede denominar PRIMARY.

logical_file_name

Es el nombre lógico que se utiliza en SQL Server cuando se hace referencia al archivo. *Logical_file_name* debe ser único en la base de datos

y ajustarse a las reglas de los identificadores. El nombre puede ser una constante de caracteres o Unicode, o un identificador regular o delimitado.

FILENAME { '*os_file_name*' | '*filestream_path*' }

Especifica el nombre de archivo (físico) del sistema operativo.

' *os_file_name* '

Es la ruta de acceso y el nombre de archivo que el sistema operativo utiliza cuando se crea el archivo. El archivo debe residir en uno de los siguientes dispositivos: el servidor local en el que se ha instalado SQL Server, una red de área de almacenamiento (SAN basada en Canal de fibra) o una red basada en ISCSI (con tarjeta de red). La ruta de acceso especificada debe existir antes de ejecutar la instrucción CREATE DATABASE. Para obtener más información, vea "Archivos y grupos de archivos de base de datos" en la sección Notas.

Los parámetros SIZE, MAXSIZE y FILEGROWTH no se pueden establecer cuando se especifica una ruta UNC para el archivo.

Si el archivo se encuentra en una partición sin procesar, *os_file_name* sólo debe indicar la letra de unidad de una partición sin procesar existente. Sólo se puede crear un archivo de datos en cada partición sin procesar.

Los archivos de datos no deben guardarse en sistemas de archivo comprimidos a menos que se trate de archivos secundarios de sólo lectura o que la base de datos sea de sólo lectura. Los archivos de registro no se deben almacenar en sistemas de archivo comprimidos.

' *filestream_path* '

Para un grupo de archivos FILESTREAM, FILENAME hace referencia a la ruta de acceso donde se almacenarán los datos FILESTREAM. La ruta de acceso hasta la última carpeta debe existir y la última carpeta no debe existir. Por ejemplo, si especifica la ruta de acceso C:\MyFiles\MyFilestreamData, C:\MyFiles debe existir antes de ejecutar ALTER DATABASE, pero la carpeta MyFilestreamData no debe existir.

El grupo de archivos y el archivo (<filespec>) se deben crear en la misma instrucción. Sólo puede haber un archivo, <filespec>, para un grupo de archivos FILESTREAM.

Las propiedades SIZE, MAXSIZE y FILEGROWTH no se aplican a un grupo de archivos FILESTREAM.

SIZE *size*

Especifica el tamaño del archivo.

SIZE no se puede especificar si se especifica *os_file_name* como ruta UNC. SIZE no se aplica a un grupo de archivos FILESTREAM.

size

Es el tamaño inicial del archivo.

Cuando no se suministra *size* para el archivo principal, el Database Engine (Motor de base de datos) utiliza el tamaño del archivo principal de la base de datos **model**. Cuando se especifica un archivo de datos secundario o un archivo de registro, pero no se especifica el argumento *size* para ese archivo, el Database Engine (Motor de base de datos) hace que el tamaño del archivo sea de 1 MB. El tamaño especificado para el archivo principal debe tener al menos el tamaño del archivo principal de la base de datos **model**.

Se pueden utilizar los sufijos kilobyte (KB), megabyte (MB), gigabyte (GB) o terabyte (TB). El valor predeterminado es MB. Especifique un número entero; no incluya decimales. *Size* es un valor entero. Para valores mayores que 2147483647, utilice unidades más grandes.

MAXSIZE *max_size*

Especifica el tamaño máximo que puede alcanzar el archivo. MAXSIZE no se puede especificar si se especifica *os_file_name* como ruta UNC. MAXSIZE no se aplica a un grupo de archivos FILESTREAM.

max_size

Es el tamaño máximo del archivo. Se pueden utilizar los sufijos KB, MB, GB y TB. El valor predeterminado es MB. Especifique un número entero; no incluya decimales. Si no se especifica *max_size*, el archivo aumenta hasta que el disco esté lleno. *Max_size* es un valor entero. Para valores mayores que 2147483647, utilice unidades más grandes.

UNLIMITED

Especifica que el archivo crecerá hasta que el disco esté lleno. En SQL Server, si se especifica un crecimiento ilimitado para un archivo de registro, su tamaño máximo será de 2 TB, y para un archivo de datos será de 16 TB.

FILEGROWTH *growth_increment*

Especifica el incremento de crecimiento automático del archivo. El valor FILEGROWTH de un archivo no puede exceder del valor MAXSIZE. FILEGROWTH no se puede especificar si se especifica *os_file_name* como ruta UNC. FILEGROWTH no se aplica a un grupo de archivos FILESTREAM.

growth_increment

Es la cantidad de espacio que se agrega al archivo siempre que se necesita más espacio.

El valor se puede especificar en MB, KB, GB, TB o como porcentaje (%). Si se especifica un número sin los sufijos MB, KB o %, el valor predeterminado es MB. Cuando se especifica %, el incremento de crecimiento es el porcentaje especificado del tamaño del archivo en el momento en que tiene lugar el incremento. El tamaño especificado se redondea a la cifra más próxima a 64 KB.

El valor 0 indica que el crecimiento automático está desactivado y no se permite más espacio.

Si no se especifica FILEGROWTH, el valor predeterminado es 1 MB para los archivos de datos y el 10% para los archivos de registro, y el valor mínimo es 64 KB.

Nota:

En SQL Server, el incremento de crecimiento predeterminado para los archivos de datos ha cambiado del 10% a 1 MB. El valor predeterminado para el archivo de registro (10%) permanece invariable.

<filegroup>

Controla las propiedades del grupo de archivos. Filegroup no se puede especificar en una instantánea de base de datos.

FILEGROUP *filegroup_name*

Es el nombre lógico del grupo de archivos.

filegroup_name

filegroup_name debe ser único en la base de datos y no puede ser los nombres PRIMARY ni PRIMARY_LOG suministrados por el sistema. El nombre puede ser una constante de caracteres o Unicode, o un

identificador regular o delimitado. El nombre debe cumplir las reglas de los identificadores.

CONTAINS FILESTREAM

Especifica que el grupo de archivos almacena objetos binarios grandes (BLOB) de FILESTREAM en el sistema de archivos.

DEFAULT

Especifica que el grupo de archivos indicado es el grupo de archivos predeterminado de la base de datos.

<external_access_option>

Controla el acceso externo a la base de datos y desde ésta.

DB_CHAINING { ON | OFF }

Si se especifica ON, la base de datos puede ser el origen o destino de una cadena de propiedad entre bases de datos.

Si es OFF, la base de datos no puede participar en encadenamientos de propiedad entre bases de datos. El valor predeterminado es OFF.

Importante:
La instancia de SQL Server reconocerá este valor si la opción de servidor **cross db ownership chaining** es 0 (OFF). Si **cross db ownership chaining** es 1 (ON), todas las bases de datos de usuario pueden participar en cadenas de propiedad entre bases de datos, independientemente del valor de esta opción. Esta opción se establece mediante sp_configure.

Para establecer esta opción, debe pertenecer a la función fija de servidor **sysadmin**. La opción DB_CHAINING no se puede establecer en estas bases de datos del sistema: **master**, **model**, **tempdb**.

TRUSTWORTHY { ON | OFF }

Cuando se especifica ON, los módulos de base de datos (por ejemplo, vistas, funciones definidas por el usuario o procedimientos almacenados) que utilicen un contexto de suplantación pueden tener acceso a recursos externos a la base de datos.

Si es OFF, los módulos de base de datos en un contexto de suplantación no pueden tener acceso a recursos externos a la base de datos. El valor predeterminado es OFF.

TRUSTWORTHY se establece en OFF siempre que la base de datos se adjunta.

De forma predeterminada, el valor TRUSTWORTHY se establece en OFF en todas las bases de datos de sistema, excepto **msdb**. El valor no se puede cambiar en las bases de datos **model** ni **tempdb**. Se recomienda no establecer la opción TRUSTWORTHY en ON en la base de datos **maestra**.

Para establecer esta opción, debe pertenecer a la función fija de servidor **sysadmin**.

database_snapshot_name

Es el nombre de la nueva instantánea de base de datos. Los nombres de instantánea de base de datos deben ser únicos en una instancia de SQL Server y deben cumplir las reglas de los identificadores. *database_snapshot_name* puede tener 128 caracteres como máximo.

ON **(** NAME **=** *logical_file_name***,** FILENAME **= '***os_file_name***')** [**,**... *n*]

Para la creación de una instantánea de base de datos, especifica una lista de archivos de la base de datos de origen. Para que la instantánea funcione, todos los archivos de datos deben especificarse individualmente. Sin embargo, no se permiten archivos de registro para las instantáneas de base de datos.

Para obtener las descripciones de NAME y FILENAME y sus valores, vea las descripciones de los valores equivalentes de <filespec>.

Nota:

Cuando se crea una instantánea de base de datos, no se admiten las demás opciones de <filespec> ni la palabra clave PRIMARY.

AS SNAPSHOT OF *source_database_name*

Indica que la base de datos que se va a crear es una instantánea de la base de datos de origen especificada en *source_database_name*. La instantánea y la base de datos de origen deben estar en la misma instancia'[16].

[16] http://msdn.microsoft.com/es-es/library/ms176061.aspx (create database)

4.2 Elaboración de las tablas

4.2.1 ***Creación de la tabla cat_tipo:*** una vez teniendo la Base de Datos generada se puede empezar a generar cada una de las tablas, empezando por la de cat_tipo que es la tabla que contendrá los tipos de prendas que se podrán manejar en la casa de empeño, como por ejemplo alhajas, muebles, electrodomésticos y varios. Ésta tabla servirá para que a la hora de introducir datos en el campo "tipo" de la tabla prendas, no se acepten valores que no estén definidos previamente en esta tabla. La tabla se generó con las siguientes líneas de código, empezando por la línea create table:

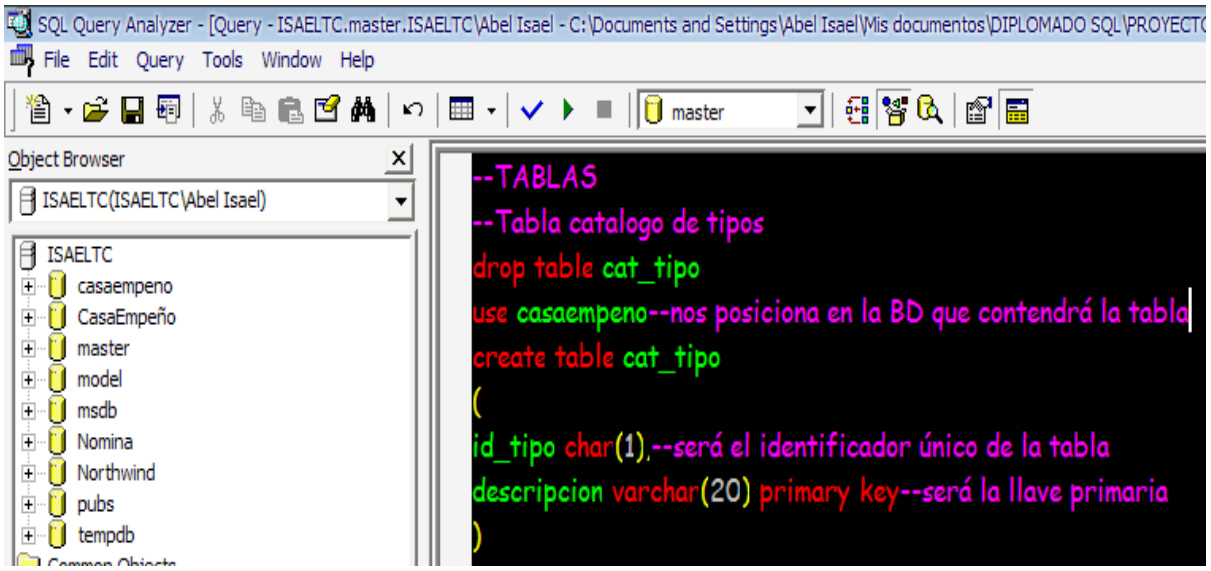


Figura 4.5 (código para la creación de la tabla *cat_tipo*)

- Para generar la tabla con las herramientas del SQL Server 2000 se realizan los siguientes pasos:

- primero se abre el programa Enterprise Manager del SQL Server y se posiciona el cursor sobre la base de datos que contendrá la tabla, para darle click derecho y seleccionar la opción que se muestra en la Figura 4.6:

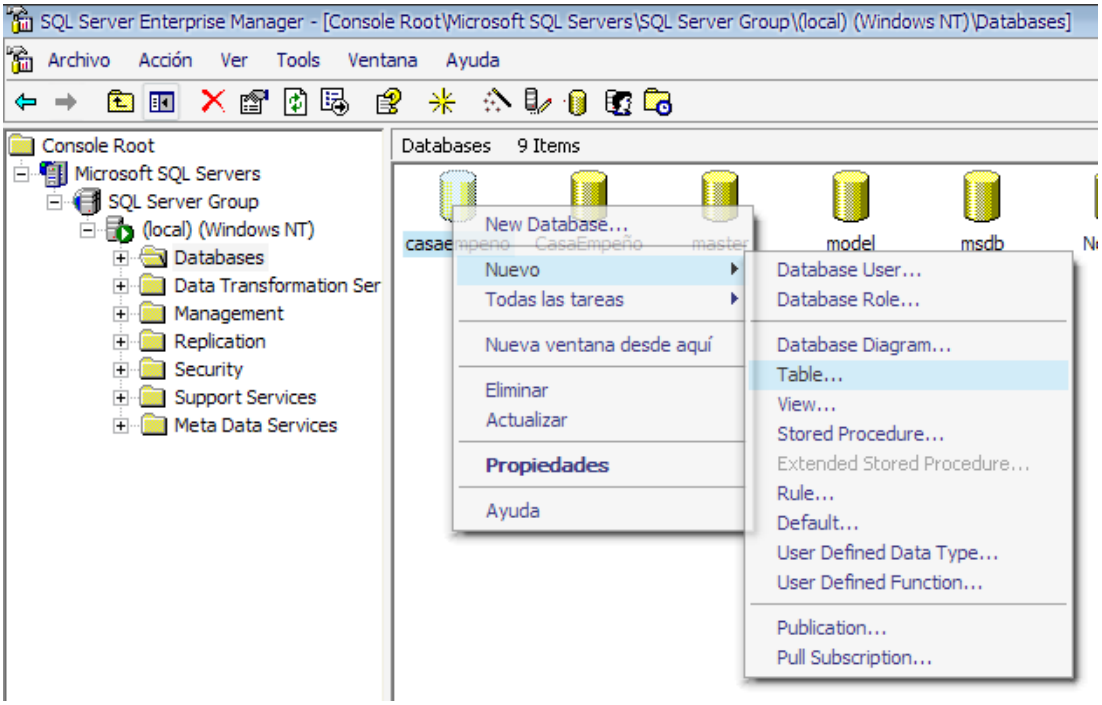


Figura 4.6 (creación de tablas con herramientas SQL Server)

- Haciendo el paso anterior aparecerá la siguiente ventana en donde especificaremos los campos de la tabla:

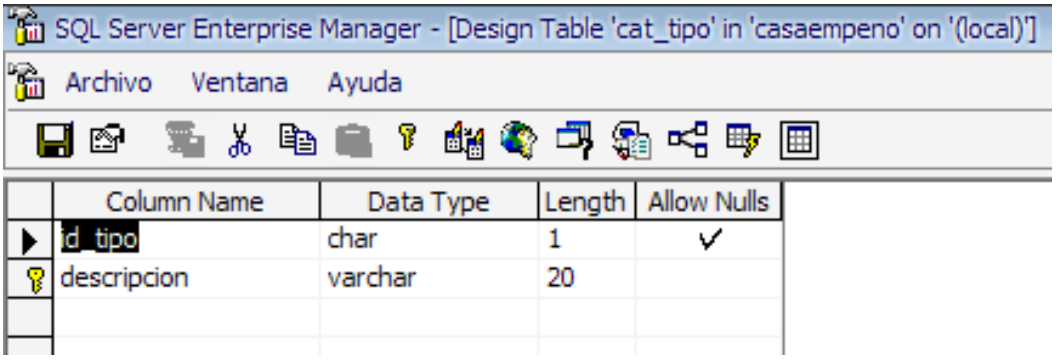


Figura 4.7 (campos de la tabla *cat_tipo*)

- para poderla guardar solo debe darse un click sobre el ícono guardar:

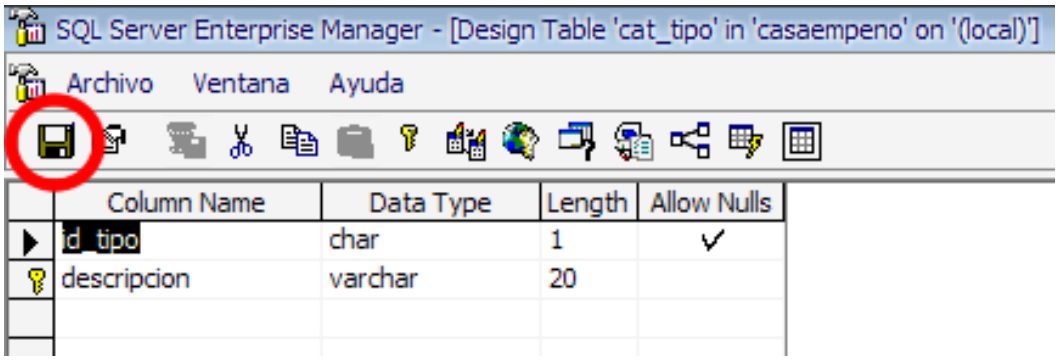


Figura 4.8 (guardar cambios de tabla *cat_tipo*)

De las dos maneras de generar la tabla se obtienen los mismos resultados, por lo tanto de ambas formas e independiente mente del método utilizado la tabla se ha generado.

4.2.2 ***Creación de la tabla cat_estado:*** Esta tabla sirve para tener un catalogo de los diferentes estados en los que pueden estar las prendas, ya sea empeñadas, desempeñadas, refrendadas, vencidas (en venta), vendidas ó cargo que es cuando la prenda no pudo ser vendida en un determinado tiempo y por lo tanto el valuador tiene que pagar el monto que prestó; su creación se realizó de la siguiente manera:

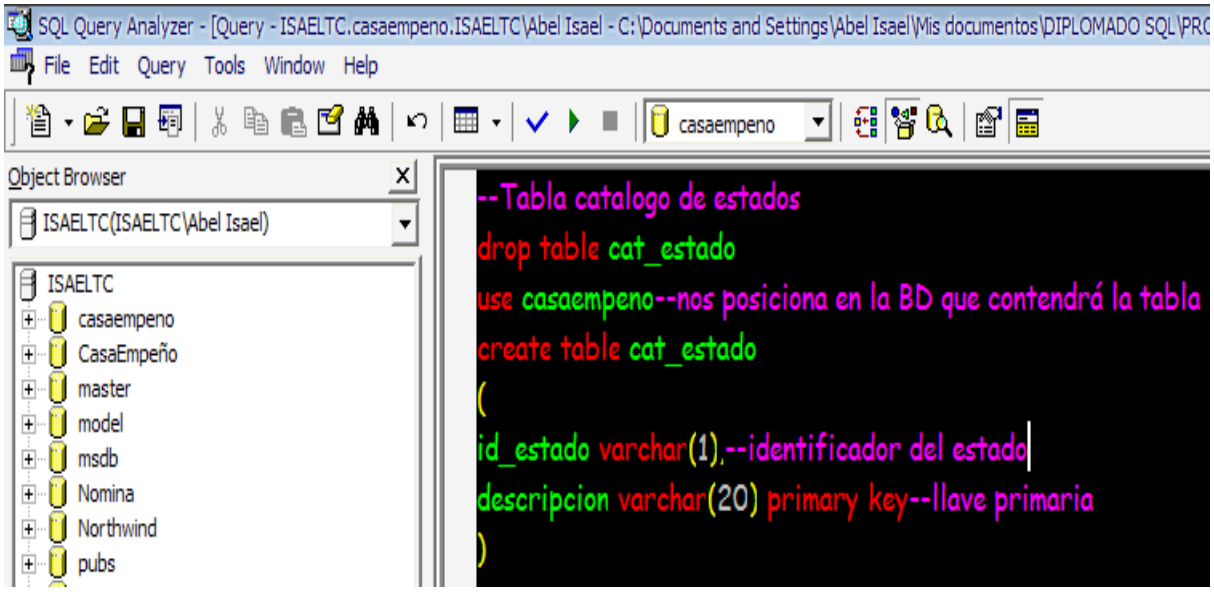


Figura 4.9 (creación de la tabla *cat_estado* mediante código SQL)

4.2.3 ***Creación de la tabla Clientes:*** La tabla clientes se decidió generar por separado y no dentro de la tabla prendas debido al manejo de los datos, ya que al querer consultar clientes o productos podría generar “sobre carga” a la hora de buscar estos datos debido a que la fila tendría demasiadas columnas, la otra razón

y la mas importante es que el tener los datos del cliente dentro de la tabla prendas provocaría una mala normalización de la BD ya que se repetiría información como los datos del cliente por lo que al tener la información de los clientes por separado permite que un cliente pueda tener mas de una prenda sin que se tenga que volver a capturar su información y de ésta forma se evita la repetición de los datos, por lo que tener los datos del cliente en una tabla y los datos de la prenda en otra permite tener nuestra BD mejor diseñada tanto en cuestiones de normalización como en optimización.

La tabla "clientes" esta compuesta por seis campos que contempla el id_prenda que será la llave primaria, nombre, ap_pat, ap_mat, que contendran el nombre completo del cliente y por ultimo los campos direccion y tel que será información complementaria de los clientes. Para la realización de esta tabla se generaron las siguientes líneas de código:

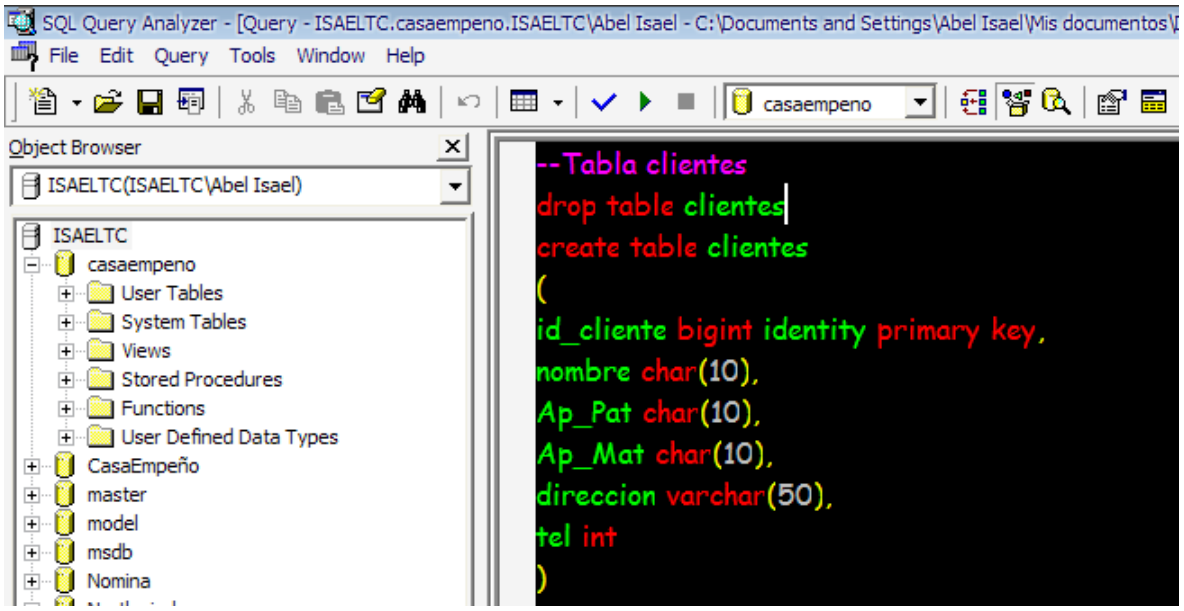


Figura 4.10 (creación de la tabla *clientes* mediante código SQL)

4.2.4 ***Creación de la tabla Prendas:*** para esta tabla se tomaron en cuenta diversos factores para su diseño, debido a que sobre esta tabla se realizarán la mayoría de las consultas, ya que ésta contendrá todas las prendas que se empeñen; de ahí la importancia de un buen diseño para ésta tabla y lograr así el correcto funcionamiento del sistema, para esto se generaron campos como el tipo de la prenda (tipo) que servirá para la clasificación de las prendas que se tendrán en la BD, lo que indicará si se tienen alhajas (joyas, relojes, pulseras, etc.), electrodomésticos (televisiones, modulares, hornos de microondas, etc.), muebles (recamaras, libreros, comedores, salas, etc.) o varios (CD's, herramienta, equipos de cómputo, etc); y poder así realizar consultas referentes a los tipos de prendas que se tienen o para la realización de estadísticas de tipos de prendas que más se empeñan, además de que éste campo irá referenciado con la tabla cat_tipo,

específicamente con el campo “descripción” para que se verifiquen los datos que se introducen en el campo “tipo” de la tabla prendas y en caso de que se introdujera un campo no definido en la tabla cat_tipo el SQL Server mandará un mensaje de que el datos que se introdujo no se encuentra en el catálogo o que esta mal escrito, para que de esta forma de alguna manera se eviten errores de captura. Por otra parte se generó un campo llamado estado que de igual forma será de vital importancia ya que permitirá generar las diversas consultas como reporte de prendas vencidas, reporte de prendas empeñadas, reporte de prendas refrendadas, reporte de prendas en venta, reporte de prendas vendidas y reporte de prendas a cargo, por lo que se vuelve un campo importantísimo en la tabla y en general en toda la Base de Datos, éste campo también esta referenciado pero con la tabla cat_estado con el campo “descripción” para cumplir con los mismos objetivos de la referencia anterior. El campo id_cliente está referenciado con la tabla clientes con el objetivo de referenciar a un cliente en diferentes prendas siendo además una llave foránea (FK) al igual que las dos referencias anteriores.

Es importante señalar que en las líneas 9, 12, 15 y 16 los campos se encuentran referenciados a una función y no a un campo, donde lo importante a destacar es que para definir esto, las funciones tuvieron que ser creadas antes de referenciarlas ya que si no se hace de ésta manera en el momento de querer generar la tabla prendas el SQL Server mandaría un mensaje especificando que la función referenciada no existe.

El campo Interes y Rebaja tendrán un valor constante, y no fueron definidos estos valores con el atributo “default” dentro de la definición de la tabla, debido a que esto implicaría que cuando se quisiera cambiar uno de estos valores, se tuviera que crear un “alter table“ a toda la tabla, por lo que con la definición empleada solo se tendrá que actualizar la propia variable, por lo que la actualización de esto se llevará a cabo mas fácilmente.

Para los demás campos, el nombre es bastante descriptivo como son el Identificador único de la prenda (Id_prenda), fecha_movimiento que servirá para saber cuando se realizó un empeño, desempeño, refrendo o alguna venta; por otra parte la fecha_vencimiento describe cuando es el último día para rescatar una prenda o para que ésta pase a estado vencido o en su defecto a la pase a estado venta. Los campos Descripción y Prestamo están descritos en el diccionario de datos por lo que no es necesario agregarles más comentarios.

Para el desarrollo de esta tabla se generó el siguiente código:

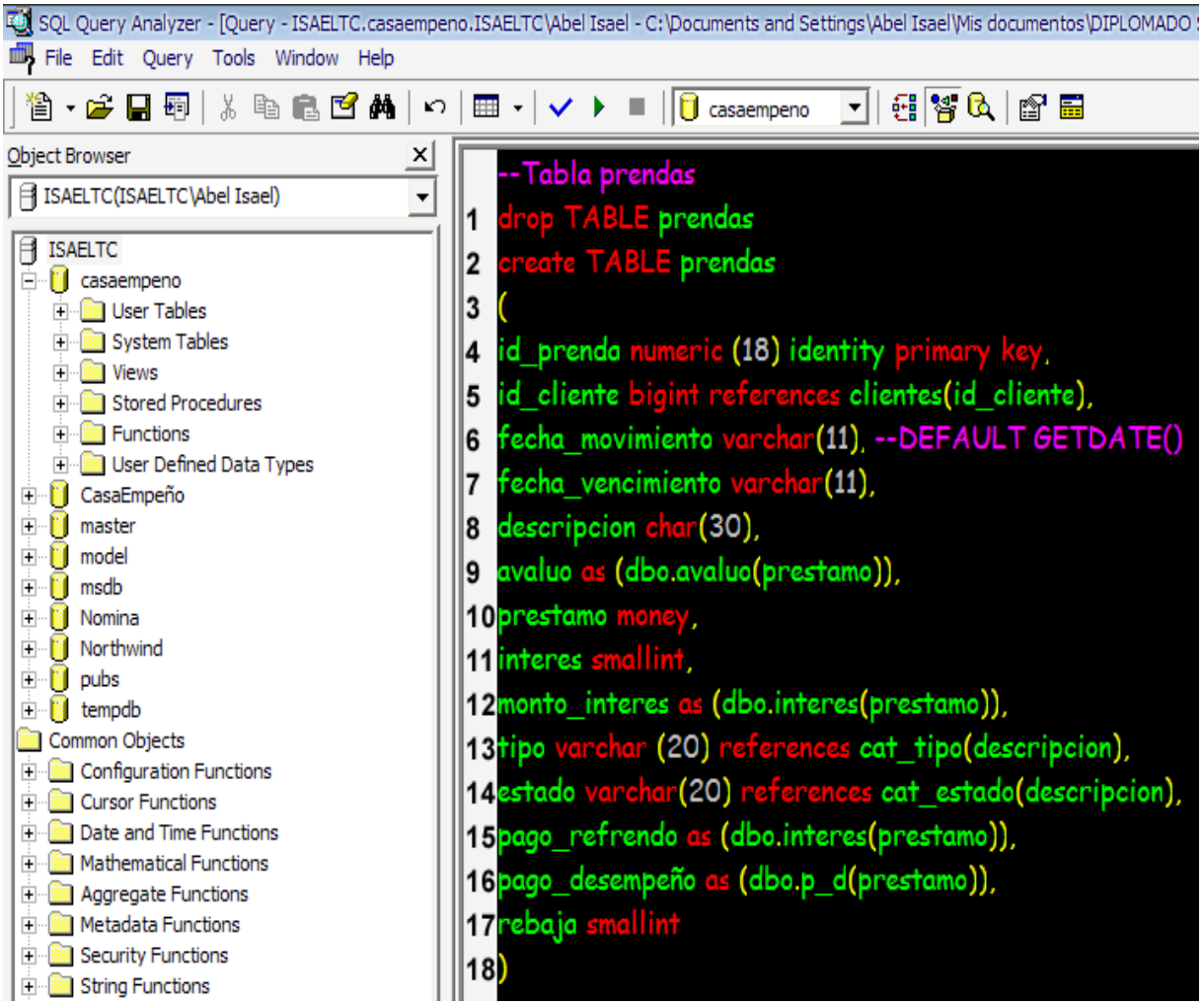


Figura 4.11 (creación de la tabla *prendas* mediante código SQL)

Con esto se tiene finalizado la creación de todas las tablas y para poder visualizarlas se tienen dos opciones:

Opción 1: desde el Query Analizer desglosando el árbol de la propia BD como se muestra en la figura 4.12:

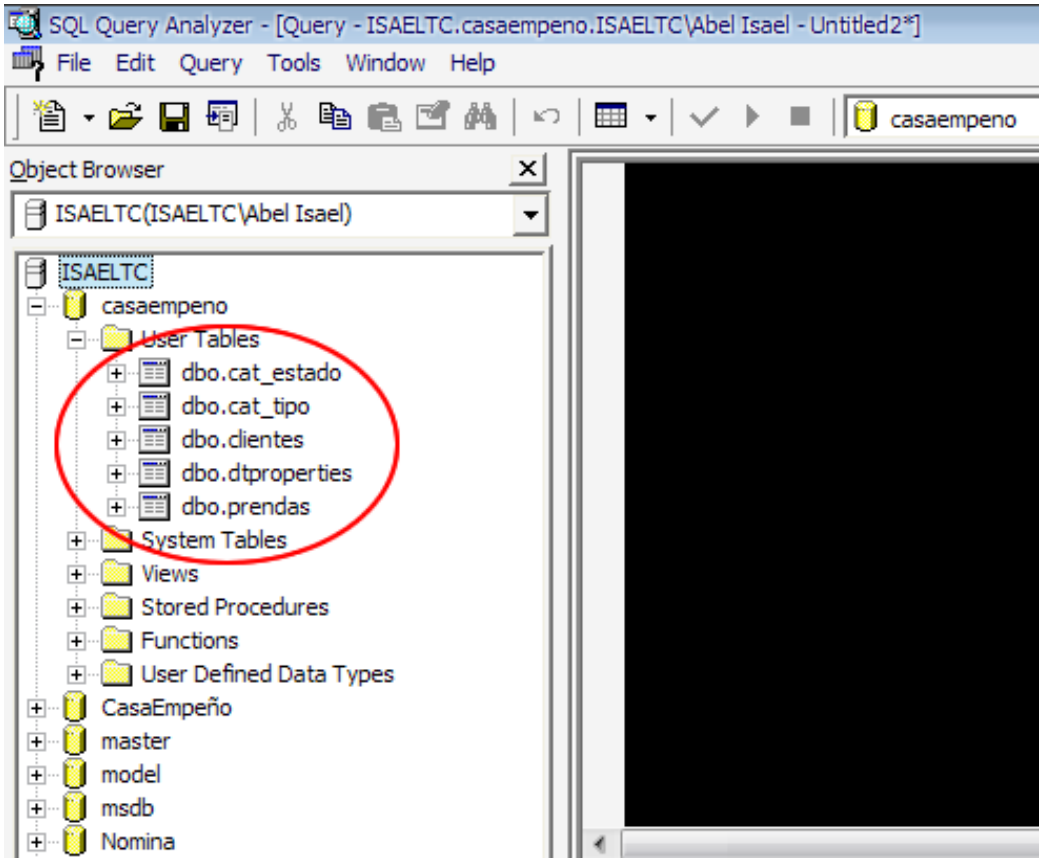


Figura 4.12 (árbol de tablas del sistema)

Opción 2: con el Enterprise Manager desglosando el árbol de “Databases” y dando click sobre la BD deseada como se muestra en la figura 4.13:

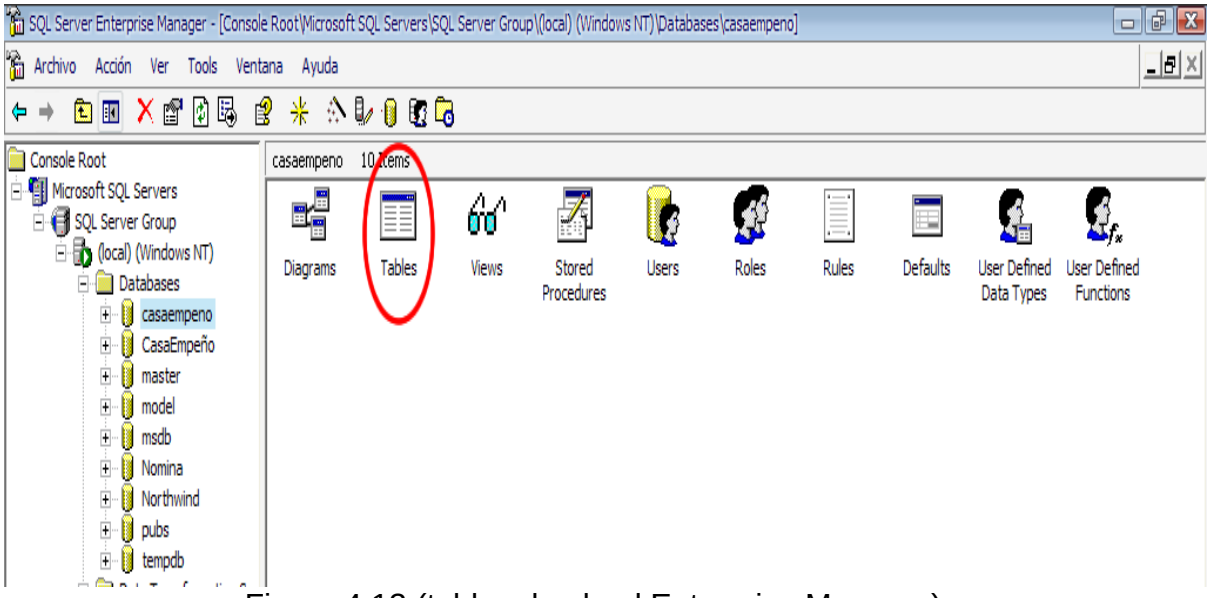


Figura 4.13 (tablas desde el Enterprise Manager)

En el lado derecho se observa la opción “Tables”, por lo que para poder ver las tablas es necesario dar doble click sobre el ícono de “Tables” donde se mostrará lo siguiente:

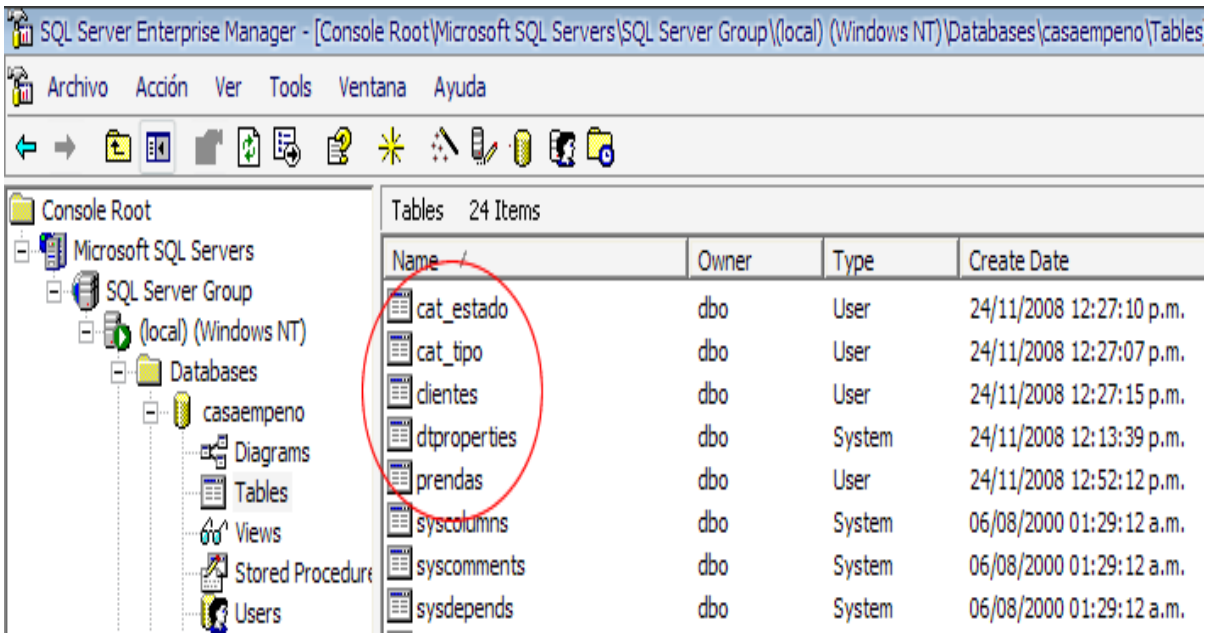


Figura 4.14 (visualización de las tablas)

Con esto podemos comprobar que las tablas fueron realizadas satisfactoriamente.

4.3 Definición de funciones

La función en SQL ‘es una rutina guardada de Transact-SQL o de Common Language Runtime (CLR) que devuelve un valor. Las funciones definidas por el usuario no se pueden utilizar para realizar acciones que modifican el estado de la base de datos. Las funciones definidas por el usuario, como las funciones de sistema, se pueden llamar desde una consulta’[17].

Para éste proyecto las funciones servirán para el cálculo de valores como el avalúo, el monto_interes, el pago_refrendo y el pago_desempeño por lo que es importante señalar que para estos campos no será necesario introducir el valor manualmente debido a que las funciones deducirán e insertarán el valor de acuerdo a los valores introducidos para el movimiento en cuestión (empeño).

[17] http://msdn.microsoft.com/es-es/library/ms186755.aspx

La sintaxis para las funciones que se utilizaron es la siguiente:

```
CREATE FUNCTION nombre_función
(
@parametro
)
RETURNS tipo_de_dato
AS
BEGIN        --comienzo de la operación
RETURN       --operación
END          --fin de la operación
```

4.3.1 *Función para cálculo del avalúo*:

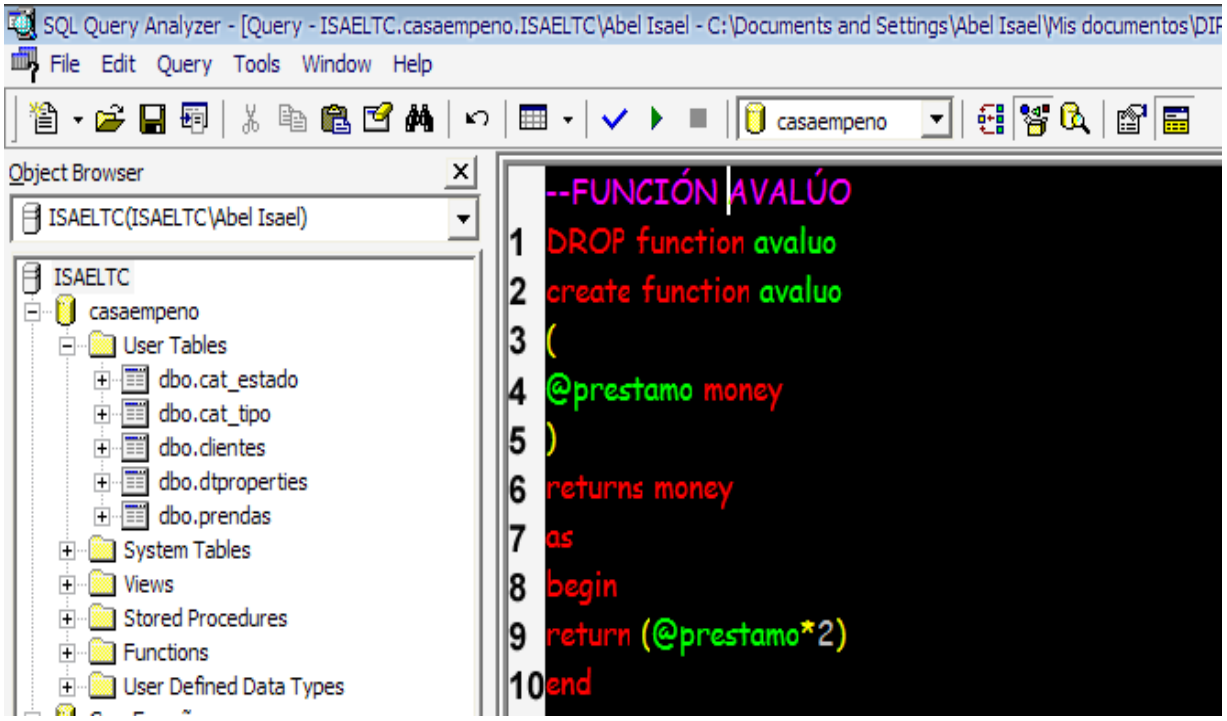


Figura 4.15 (creación de la función *avaluo* mediante código SQL)

Como se puede observar la variable “prestamo“ es el valor introducido en la tabla (campo “prestamo”) por el usuario (valuador) y éste valor es recuperado por la función y multiplicado por 2 para devolver el monto del avalúo que como fue especificado en la tabla de *prendas,* el campo avalúo esta referenciado a ésta función por lo que el valor generado de ésta es insertado en el campo. Es importante señalar que el préstamo que se le da al cliente por su objeto es del 50% del valor real de su objeto.

4.3.2 *Función para el agregado interés*:

Ésta función regresará el impuesto a pagar y será insertado tanto en el campo “monto_interes” como en el campo “refrendo” ya que el primero especifica el impuesto a pagar de acuerdo al préstamo obtenido y el refrendo es la cantidad a pagar para obtener una prorroga, para que la prenda no venza y para que ésta pueda ser rescatada posteriormente. Como se puede notar es el mismo valor para ambos campos, de ahí que en la definición de la tabla prendas ésta función haya sido referenciada en ambos campos (“monto_interes” y ”refrendo”).

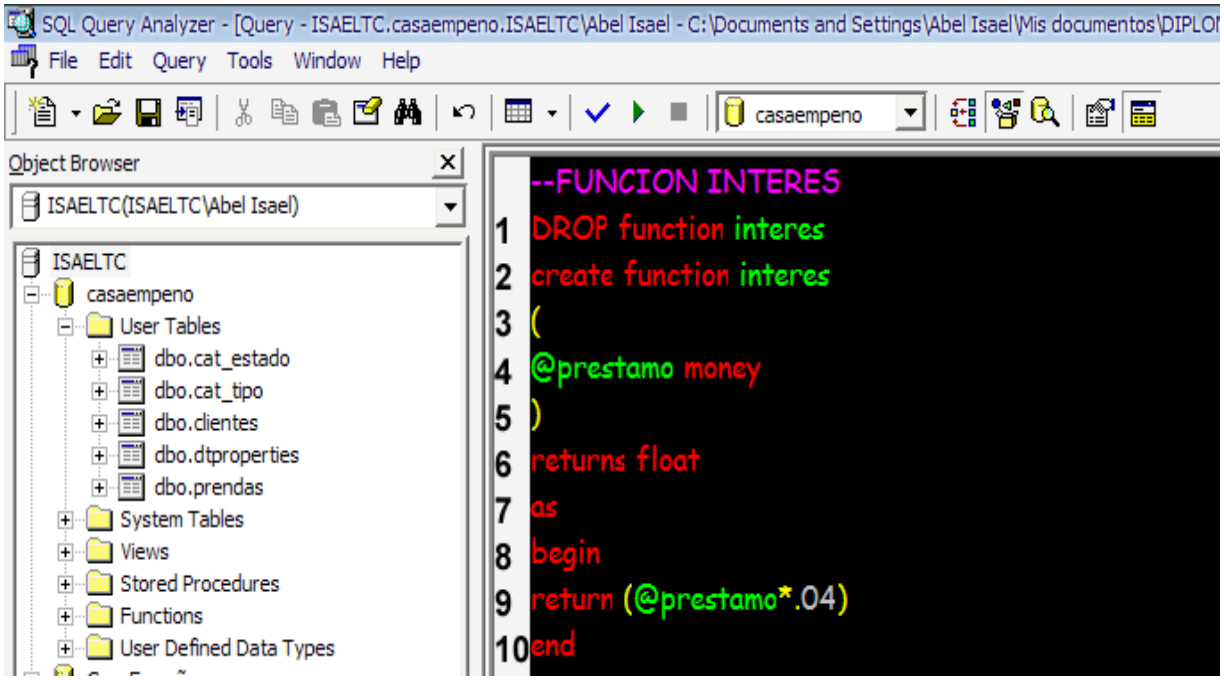


Figura 4.16 (creación de la función *interes* mediante código SQL)

Como puede observarse, para calcular el monto deseado es obteniendo el valor del préstamo y multiplicarlo por el interés (4%) que en realidad es .04 para obtener el impuesto a pagar que es interpretado como refrendo en caso de querer una prorroga al vencimiento de la prenda o en su defecto para mostrar el monto del interés (monto_interes) que no son mas que los intereses a pagar por el préstamo obtenido.

NOTA: En caso de necesitar cambiar el monto del interés, se actualizará el cambio desde la función.

4.3.3 *Función para el cálculo del pago por desempeño*:

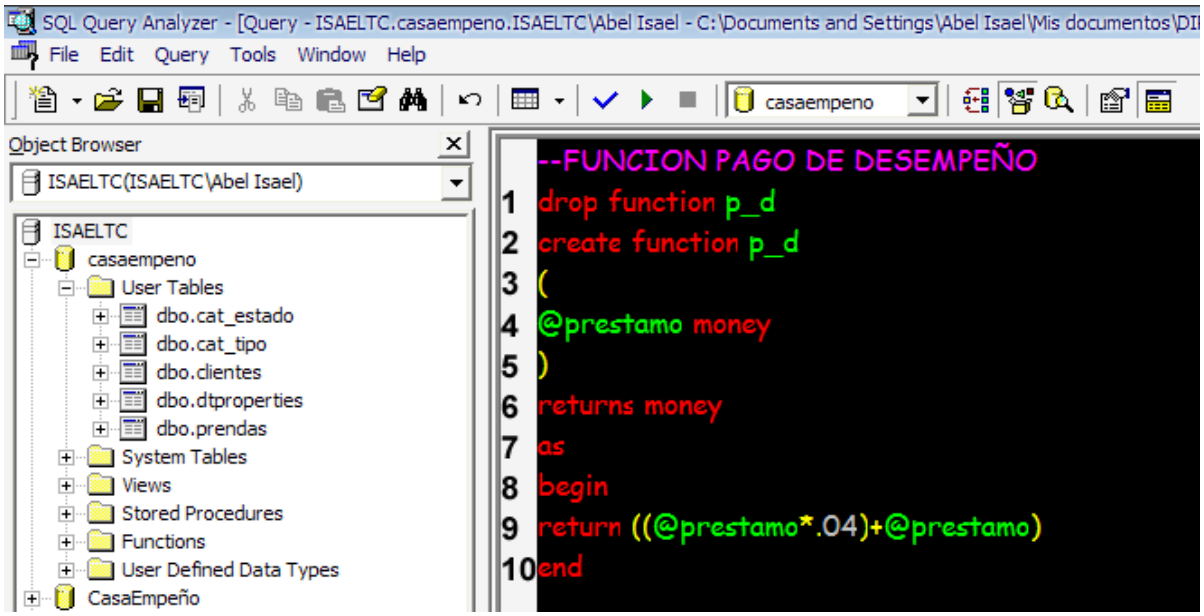


Figura 4.17 (creación de la función *p_d* (Pago de Desempeño) con código SQL)

Como se puede observar en la función para poder desempeñar una prenda es necesario pagar los impuestos del préstamo más el propio préstamo, lo cual gracias a ésta función la cantidad por concepto de desempeño se obtiene automáticamente además de que al mismo tiempo inserta el parámetro.

4.4 Construcción de los trigger's

Los trigger's para éste proyecto juegan un papel esencial en el aspecto funcional, siendo más específicos en la automatización de las tareas, ya que éstos son programas que actúan por si solos cuando su parámetro definido de levantamiento (disparo) es realizado, ya sea un DELETE, INSERT o DELETE que son los parámetros que los trigger's aceptan para realizar una tarea definida dentro de ellos si uno de éstos parámetros es realizado dentro del sistema (dentro de la misma BD).

En otras palabras los trigger's 'crean un desencadenador DML, DDL o logon. Un desencadenador es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos. Los desencadenadores DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.

Nota:

Estos desencadenadores se activan cuando se desencadena cualquier evento válido, con independencia de que las filas de la tabla se vean o no afectadas.

Los desencadenadores DDL se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL. Los desencadenadores logon se activan en respuesta al evento LOGON que se genera cuando se establece la sesión de un usuario. Los desencadenadores se pueden crear directamente a partir de instrucciones Transact-SQL o de métodos de ensamblados que se crean en Microsoft .NET Framework Common Language Runtime (CLR) y se cargan en una instancia de SQL Server. SQL Server permite crear varios desencadenadores para una instrucción específica'[18].

La sintaxis general de los trigger's utilizados es la siguiente:

```
CREATE   TRIGGER nombre_del_trigger
ON nombre_tabla_o_vista
FOR INSERT, UPDATE ó DELETE
AS
{sentencia SQL}
```

4.4.1 *Trigger pasar prenda a estado vencido*:

El trigger llamado pasar_a_vencido como se muestra en la figura, es el encargado de actualizar todas aquellas prendas que tienen el campo fecha_vencimiento expirado, lo cual quiere decir que para todas aquellas prendas que se encuentren en esta circunstancia el trigger actualizará su campo "estado" a *"vencido"* teniendo en cuenta que el estado previo puede ser ya sea *empeñado* ó *refrendado.*

[18] http://msdn.microsoft.com/es-es/library/ms189799.aspx

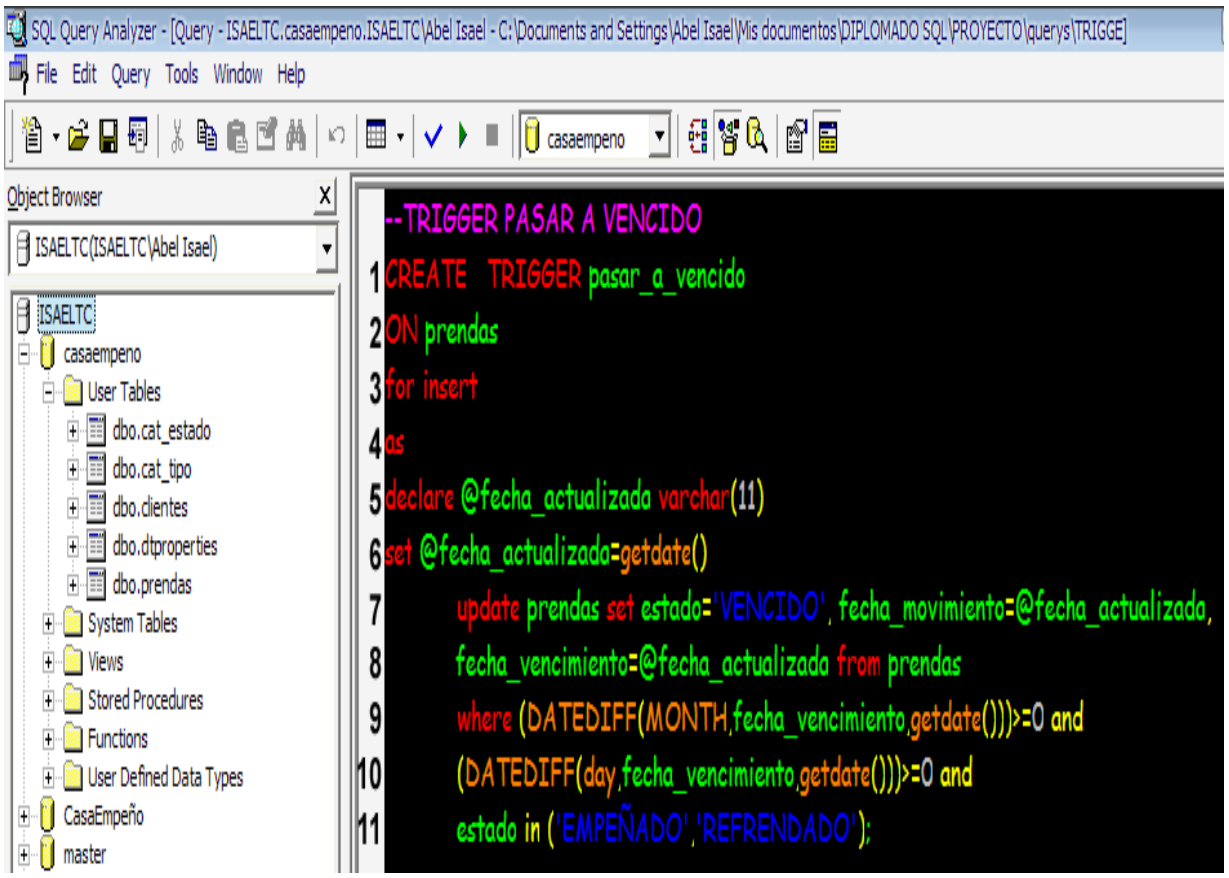


Figura 4.18 (creación del trigger *pasar_a_vencido* mediante código SQL)

Como puede observarse el trigger actuará sobre la tabla “prendas” (línea 2) y se disparará cuando se realice un insert (línea 3) además de que la sentencia SQL (línea 5 a 11) se encuentra compuesta en primera instancia por la declaración de una variable local (línea 5) llamada fecha_actualizada, que será la fecha actualizada (día en que la prenda venció o el día en que la prenda fue actualizada) que se define el la línea 6 con el comando SET, después de la línea 7 a la 11 se encuentra el query ó sentencia que actualizará la prenda y en donde se encuentra especificada la tarea a realizar (UPDATE línea 7) y los campos a actualizar con sus respectivos valores nuevos (SET estado='VENCIDO', fecha_movimiento=@fecha_actualizada, fecha_vencimiento=@fecha_actualizada línea 7 y 8) donde los valores nuevos se encuentran después del signo igual y la función que nos permite definir sus valores es la función SET; por último tenemos las condiciones que se deben cumplir para que se realice la actualización y esto se hace mediante el comando WHERE (línea 9-11) de donde se puede resaltar la función DATEDIFF (línea 9 y 10) que devuelve la diferencia ya sea de los meses (MONTH línea 9) o de los días (DAY línea 10) entre dos fechas establecidas lo cual si nos devuelve un número superior a 0 (definido con los operadores >= en líneas 9 y 10) querrá decir que las fecha a evaluar (fecha_vencimiento) supera el parámetro a cumplir (GETDATE() que es la fecha actual) lo que determina si una prenda de acuerdo a su campo “fecha_vencimiento” ha expirado o no, para que la

prenda sea actualizada si su fecha expiró o no sea actualizada si su fecha no cumple con las condiciones especificadas. La línea final del código "estado IN ('EMPEÑADO','REFRENDADO');" define que la prenda debe estar en estado "EMPEÑADO" ó "REFRENDADO" para que pueda ser actualizada al estado "VENCIDO".

4.4.2 *Trigger pasar prenda a estado cargo*:

El trigger pasar_a_cargo tiene el objetivo de actualizar el campo "estado" de la tabla "prendas" a *CARGO*, que es el estado en donde si la prenda no se ha podido vender en un lapso no mayor a 5 meses pasará a éste estado por lo que el trigger pasar_a_cargo es el responsable de buscar todas aquellas prendas que no se han podido vender y actualizar su campo "estado" a *CARGO*. La definición de éste trigger es muy similar al trigger pasar_a_vencido, con la diferencia en que éste se disparará cuando haya un UPDATE (definido en línea 3) y no en un INSERT como el trigger anterior, además de que solo se toman en cuenta los meses (línea 9) y el campo "estado" a buscar debe de ser solo para las prendas que se encuentres en estado *VENCIDO* (línea 10).

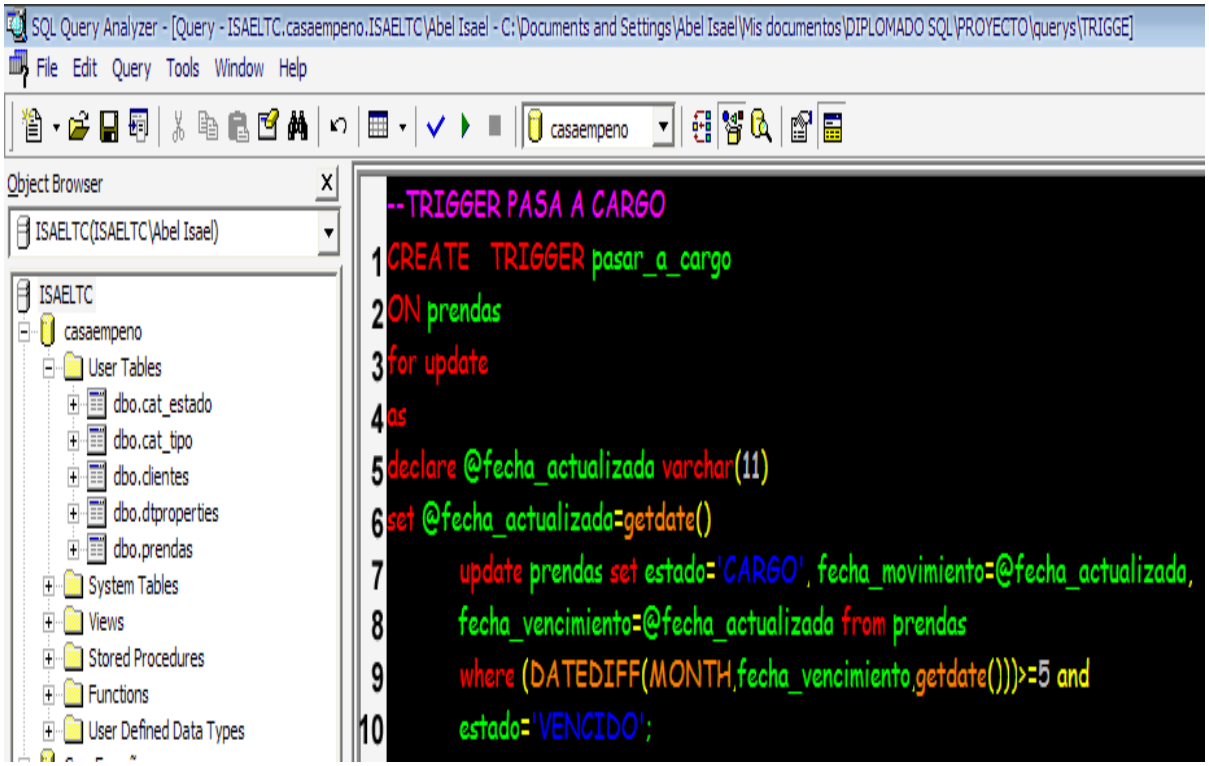


Figura 4.19 (creación del trigger *pasar_a_cargo* mediante código SQL)

CAPÍTULO 4

4.5 Elaboración de la vista ingresos_egresos

El objetivo de ésta vista es mostrar un reporte de los ingresos y egresos totales por cada uno de los diferentes movimientos que se llegan a hacer en una casa de empeño, ya que por un lado los egresos estarán definidos por los empeños en donde para ésta tarea se realiza un préstamo de dinero, por lo que se considera egreso (-) y por otro lado están los ingresos (+) que están dados por los refrendos, ventas, desempeños y cargos, por lo que la vista mostrara el total de cada uno de ellos a manera de lista y al final de ella mostrara un total que será la suma total de los ingresos y egresos. La vista se construyó con las siguientes líneas de código:

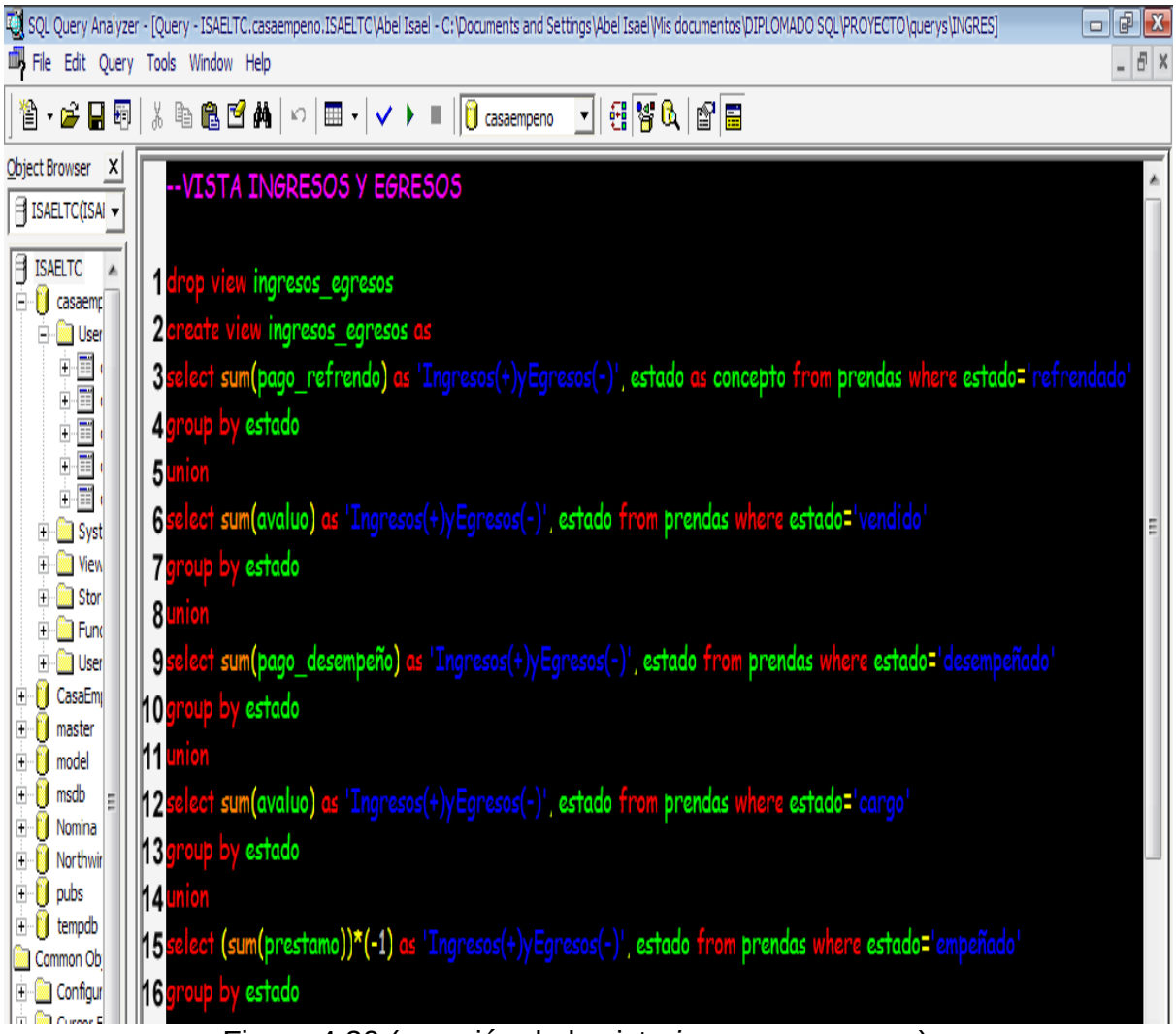


Figura 4.20 (creación de la vista *ingresos_egresos*)

En el código de la vista se puede observar que es generada por el comando CREATE VIEW (línea 2) seguida del AS que es el que indica el comienzo de la

definición de la vista; después del AS vienes las diferentes consultas que fueron realizadas para obtener los diferentes conceptos por ingresos y egresos que a su ves fueron unidas por el comando UNION que combina los resultados de dos o más consultas en un solo conjunto de resultados y en donde la función SUM es utilizada en todas las consultas para sumar el campo de interes para cada uno de los conceptos. Es importante mencionar que la vista sólo mostrará el listado de los totales de cada uno de los conceptos y no el súper total (la suma de todos los conceptos), ya que para esto es necesario utilizar la función COMPUTE que es la que mandará el súper total especificando dentro de ella la columna a sumar como se muestra a continuación: **compute sum ([Ingresos(+) y Egresos(-)])** donde COMPUTE es la función, SUM es la función que especifica la operación a realizar (sumar) y [Ingresos(+) y Egresos(-)] es la columna a evaluar; por lo que así se obtiene el resultado deseado.

4.6 Realización de los procedimientos almacenados

'Un procedimiento almacenado es una colección guardada de instrucciones de Transact-SQL o una referencia a un método de Common Language Runtime (CLR) de Microsoft .NET Framework que puede aceptar y devolver los parámetros proporcionados por el usuario. Los procedimientos se pueden crear para uso permanente o para uso temporal en una sesión, un procedimiento local temporal, o para su uso temporal en todas las sesiones, un procedimiento temporal global'[19].

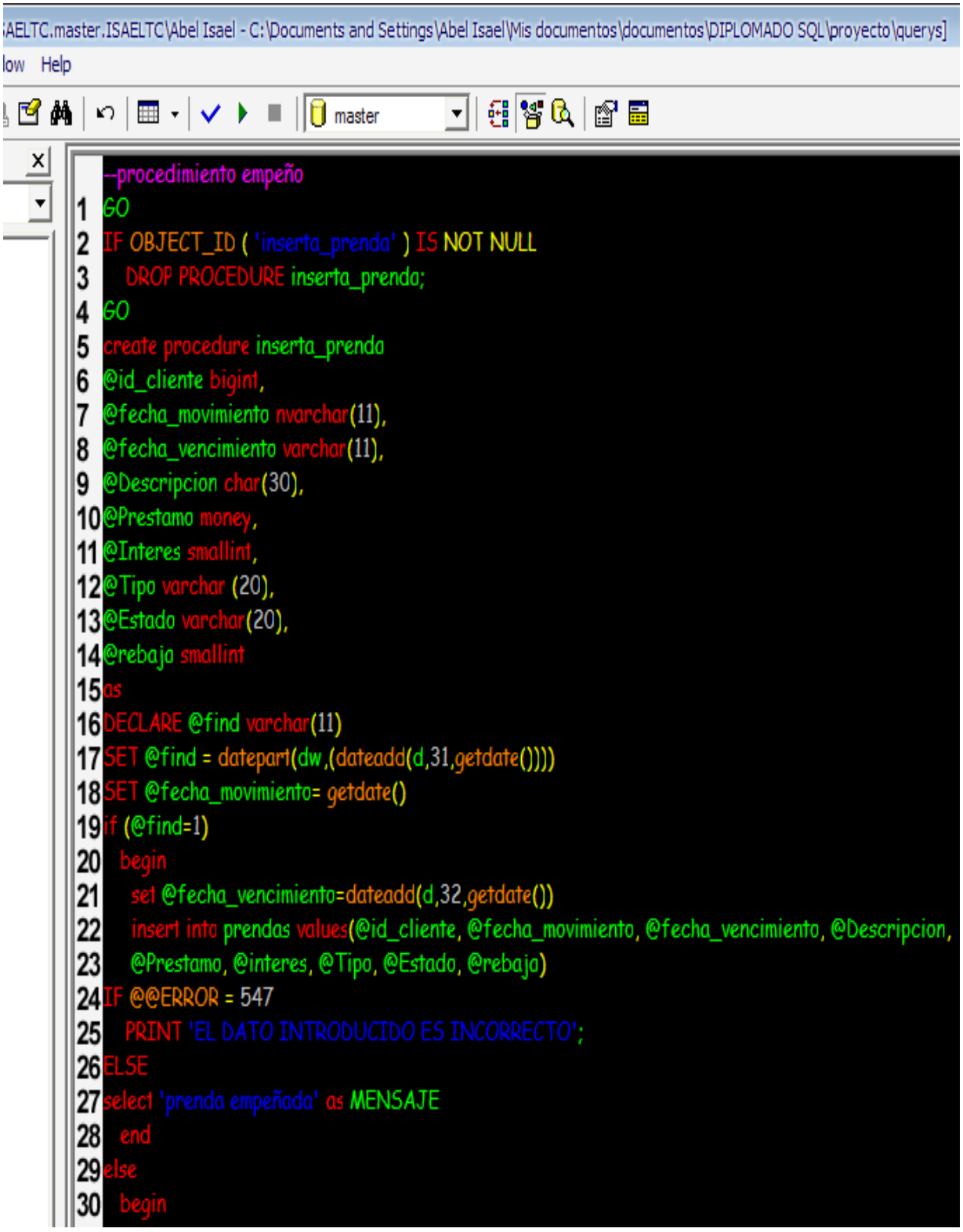
4.6.1 *Procedimiento almacenado empeño*

Para éste punto se crearon dos procedimientos almacenados en donde el procedimiento inserta_prenda será el encargado de permitir la inserción de las prendas, y el procedimiento almacenado llamado inserta_cliente será el que permita registrar a un cliente en el sistema.

El procedimiento inserta_prenda se realizó mediante el siguiente código:

[19] http://msdn.microsoft.com/es-es/library/ms187926.aspx

CAPÍTULO 4




```
--procedimiento empeño
GO
IF OBJECT_ID ( 'inserta_prenda' ) IS NOT NULL
   DROP PROCEDURE inserta_prenda;
GO
create procedure inserta_prenda
@id_cliente bigint,
@fecha_movimiento nvarchar(11),
@fecha_vencimiento varchar(11),
@Descripcion char(30),
@Prestamo money,
@Interes smallint,
@Tipo varchar (20),
@Estado varchar(20),
@rebaja smallint
as
DECLARE @find varchar(11)
SET @find = datepart(dw,(dateadd(d,31,getdate())))
SET @fecha_movimiento= getdate()
if (@find=1)
  begin
    set @fecha_vencimiento=dateadd(d,32,getdate())
    insert into prendas values(@id_cliente, @fecha_movimiento, @fecha_vencimiento, @Descripcion,
    @Prestamo, @interes, @Tipo, @Estado, @rebaja)
IF @@ERROR = 547
   PRINT 'EL DATO INTRODUCIDO ES INCORRECTO';
ELSE
select 'prenda empeñada' as MENSAJE
  end
else
  begin
```

```
31    set @fecha_vencimiento=dateadd(d,31,getdate())
32    insert into prendas values(@id_cliente, @fecha_movimiento, @fecha_vencimiento, @Descripcion,
33    @Prestamo, @interes, @Tipo, @Estado, @rebaja)
34  IF @@ERROR = 547
35     PRINT 'EL DATO INTRODUCIDO ES INCORRECTO';
36  ELSE
37  select 'prenda empeñada' as MENSAJE
38   end;
39  select prendas.id_prenda,nombre+ap_pat+ap_mat as nombre, direccion, tel, prendas.fecha_movimiento,
40  prendas.fecha_vencimiento, prendas.descripcion, prendas.prestamo, prendas.pago_desempeño
41  from clientes join prendas on (prendas.id_cliente = clientes.id_cliente)
42  where prendas.id_prenda = ident_current('prendas')
```

Figura 4.21(código del procedimiento almacenado *inserta_prenda*)

El código escrito para éste procedimiento almacenado se traduce de la siguiente manera:

- **línea 1-4**: se verifica si el procedimiento existe y de serlo así lo borra.
- **línea 5**: es el comando que genera los procedimientos almacenados y va seguido del nombre del procedimiento a realizar (inserta_prenda).
- **líneas 6-14**: son los parámetros o variables del procedimiento almacenado y serán los parámetros que el usuario introducirá cuando se empeñe una prenda. Cabe señalar que los parámetros fecha_movimiento y fecha_vencimiento serán insertados y definidos por el propio procedimiento almacenado y los parámetros interés y rebaja serán valores constantes.
- **línea 15**: nos indica el como, indicándonos que a continuación vendrá las sentencias SQL que definirán al procedimiento almacenado.
- **línea 16**: se declara una variable local (find) que servirá para identificar si una fecha cae en domingo., por lo que si fuera el caso la fecha_vencimiento será recorrida a un día después.
- **línea 17**: se define el valor de inicio de la variable *find* y en donde datepart es la función que permite conocer un día o mes para poderlo validar, y en donde para este caso el día que nos interesa es el domingo que para ésta función este día es relacionado con un 1.
- **línea 18**: se le asigna la fecha actual a la variable fecha_movimiento con la función getdate().
- **línea 19**: comienza la validación de la búsqueda del domingo (1) mediante un IF que se interpreta como “si la fecha cae en domingo?”
- **línea 20**: es el comando que indica el comienzo de la sentencia SQL, para el caso de que la fecha caiga en domingo.
- **línea 21**: cuando la fecha cae en domingo la variable fecha_vencimiento se recorre un día, por lo que en ésta línea se asigna el valor (la fecha recorrida) con la función DATEADD que lo que hace es agregar días o

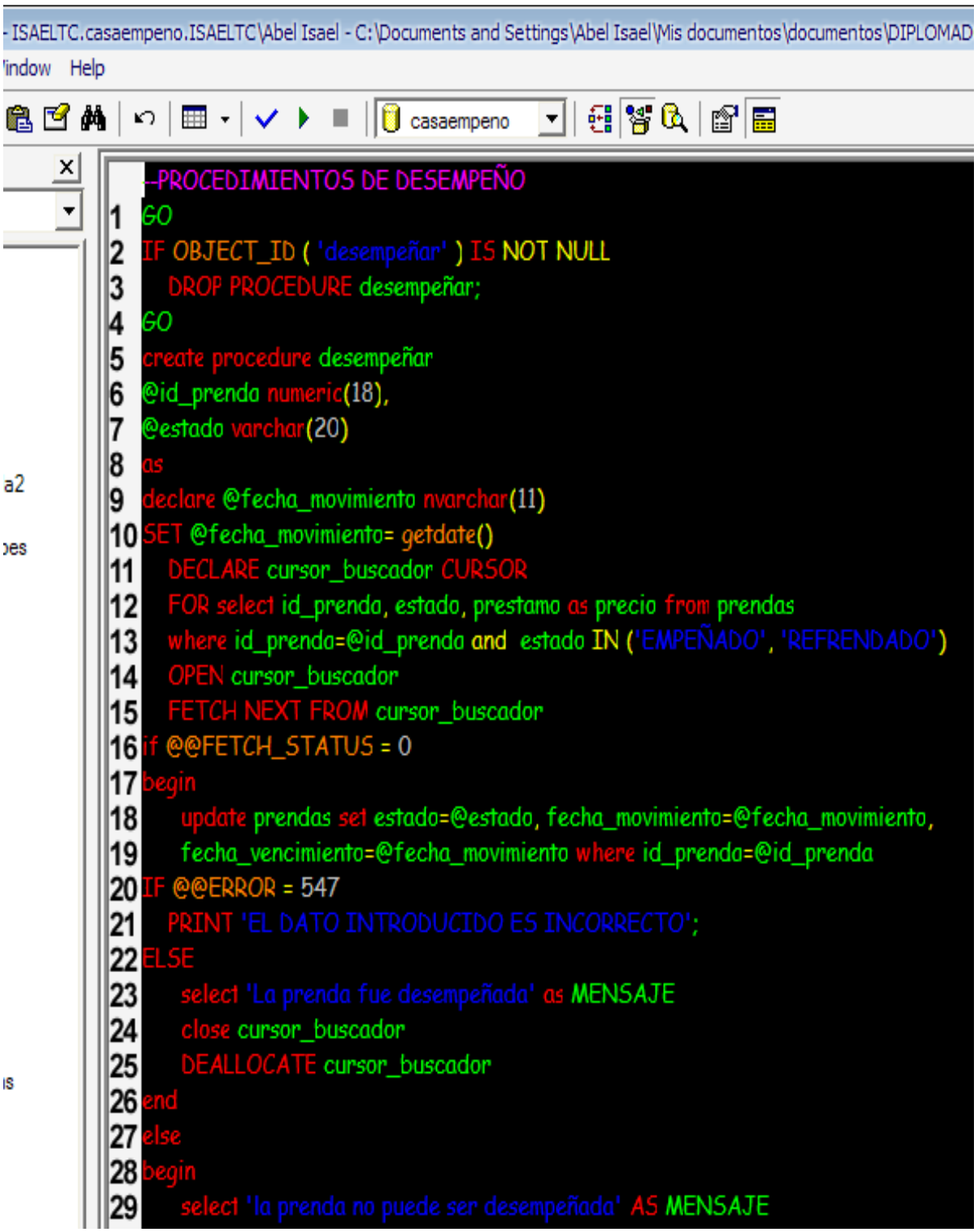
meses dependiendo de lo que se quiera logar. Para éste caso se agregaron días.

- **línea 22-23**: Se define la inserción.
- **línea 24**: Se valida el error de integridad que es la parte en donde el SQL Server verificará si el valor introducido está definido en los catálogos cat_tipo y cat_estado; y en el caso en que se introduzca un valor no valido o un valor mal escrito el programa mandará un mensaje de error por lo que con la función @@ERROR se cacha el error, para posteriormente realizar una interpretación del mensaje enviado por el SQL Server.
- **línea 25**: Se define el mensaje a publicar mediante el comando PRINT.
- **línea 26**: (ELSE) especifica que de no haber un error se realice la tarea siguiente.
- **línea 27**: Envía un mensaje de que “la prenda fue empeñada”, y no hubo ningún error de captura.
- **línea 28**: termina el IF ELSE de la validación del error.
- **línea 29**: comienza el ELSE de la validación de la fecha (de no ser domingo).
- **línea 30**: se especifica el comienzo del ELSE anterior con el comando BEGIN.
- **línea 31**: se le define el valor a la variable fecha_vencimiento con el comando SET y se emplean las mismas funciones con la diferencia de que en ves de sumarle a la fecha actual 32 días solo se le agregarán 31.
- **línea 32-33**: se realiza la inserción de los datos mediante el comando INSERT.
- **línea 34**: se valida de nuevo el error de integridad.
- **línea 35**: se define el mensaje para el caso de que ocurra el error.
- **línea 36**: es el ELSE que significa: en caso de que no ocurra un error.
- **línea 37**: se envía el mensaje de prenda empeñada.
- **línea 38**: termina el IF ELSE de la validación de la fecha.
- **línea 39-42**: se define el SELECT que mandará en pantalla la prenda que acaba de ser empeñada de donde destaca la función IDENT_CURRENT que es la función que devuelve el último registro ó valor de identidad generado.

4.6.2 *Procedimiento almacenado desempeño*

Es el encargado de llevar a cabo la tarea del desempeño y se definió mediante el siguiente código:

CAPÍTULO 4




```
--PROCEDIMIENTOS DE DESEMPEÑO
GO
IF OBJECT_ID ( 'desempeñar' ) IS NOT NULL
   DROP PROCEDURE desempeñar;
GO
create procedure desempeñar
@id_prenda numeric(18),
@estado varchar(20)
as
declare @fecha_movimiento nvarchar(11)
SET @fecha_movimiento= getdate()
   DECLARE cursor_buscador CURSOR
   FOR select id_prenda, estado, prestamo as precio from prendas
   where id_prenda=@id_prenda and  estado IN ('EMPEÑADO', 'REFRENDADO')
   OPEN cursor_buscador
   FETCH NEXT FROM cursor_buscador
if @@FETCH_STATUS = 0
begin
    update prendas set estado=@estado, fecha_movimiento=@fecha_movimiento,
    fecha_vencimiento=@fecha_movimiento where id_prenda=@id_prenda
IF @@ERROR = 547
   PRINT 'EL DATO INTRODUCIDO ES INCORRECTO';
ELSE
    select 'La prenda fue desempeñada' as MENSAJE
    close cursor_buscador
    DEALLOCATE cursor_buscador
end
else
begin
    select 'la prenda no puede ser desempeñada' AS MENSAJE
```

```
30    SELECT id_prenda, estado from prendas where id_prenda=@id_prenda
31    close cursor_buscador
32    DEALLOCATE cursor_buscador
33 end;
```

Figura 4.22 (código del procedimiento almacenado *desempeñar*)

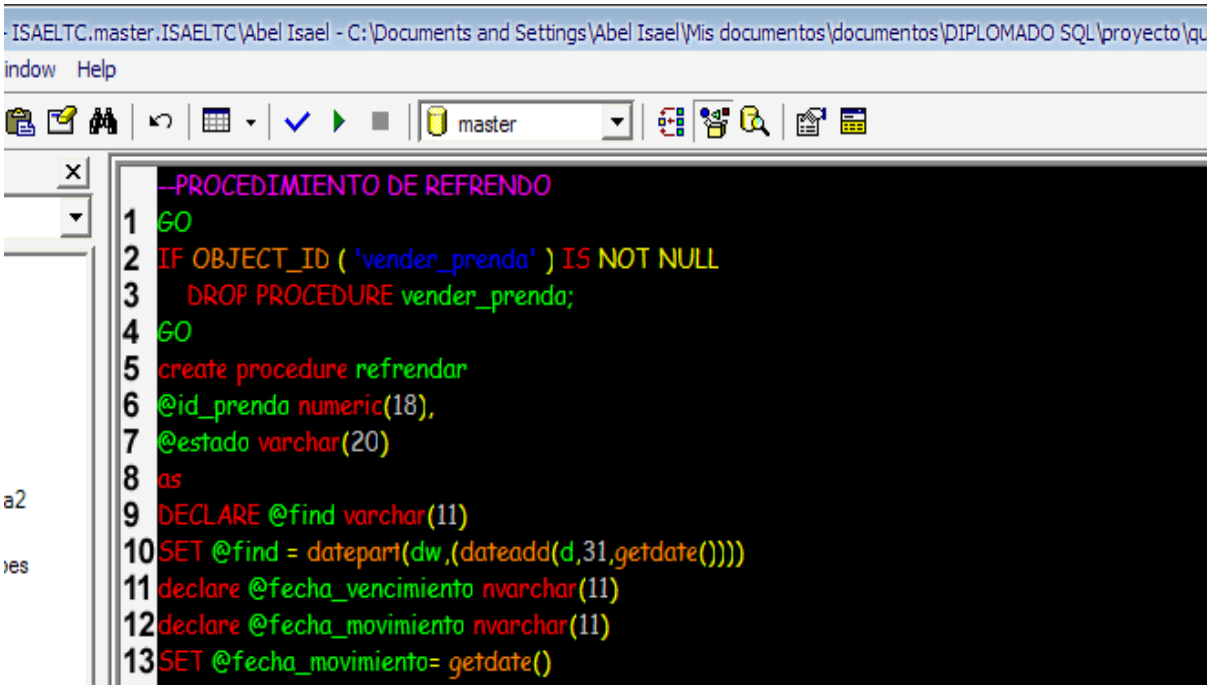
El código escrito para éste procedimiento almacenado se traduce de la siguiente manera:

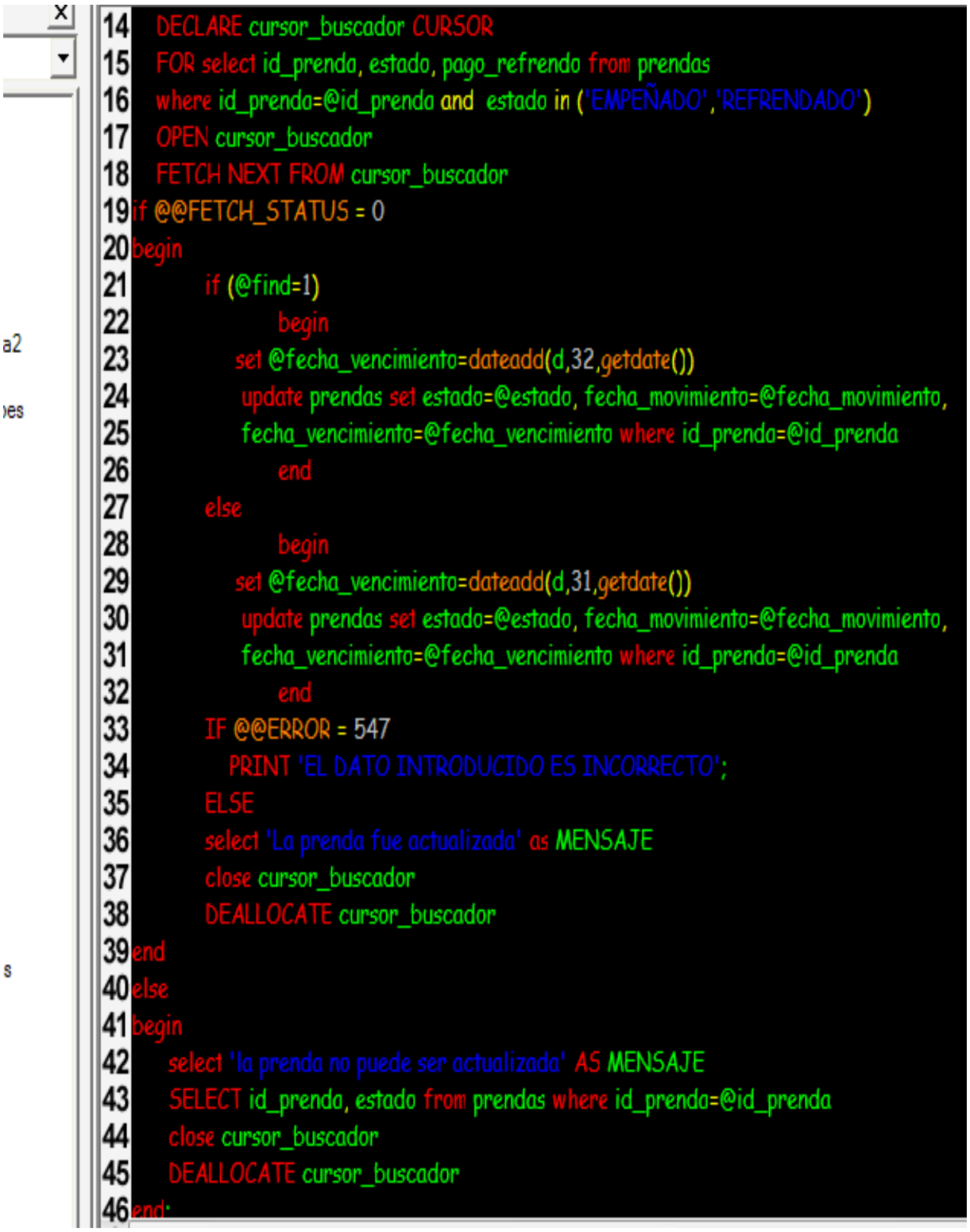
- **línea 1-4**: se verifica si el procedimiento existe y de serlo así lo borra.
- **línea 5**: es el comando que genera los procedimientos almacenados y va seguido del nombre del procedimiento a realizar (desempeñar).
- **línea 6-7**: se defines las variables ó parámetros del procedimiento almacenado, que para éste caso son dos y en donde id_prenda servirá como parámetro para actualizar solo la prenda que tenga ese identificador único de prenda (id_prenda) mientras que la variable estado contendrá un valor constante igual a “DESEMPEÑADO”.
- **línea 8**: AS éste comando especifica el comienzo de la sentencia SQL del procedimiento almacenado.
- **línea 9**: se declara una variable local (fecha_movimiento) que contendrá la fecha actual y que servirá para definírsela a los campos fecha_movimiento y fecha_vencimiento de la prenda a desempeñar; se les asocia la misma fecha debido a que el objeto ya no está sujeto a un plazo por ser una prenda desempeñada.
- **línea 10**: se le asigna la fecha actual a la variable local (fecha_movimiento).
- **línea 11-13**: se declara un **CURSOR** (de nombre cursor_buscador) que servirá para validar que la prenda que se pretende desempeñar este en estado “EMPEÑADO” ó ”DESEMPEÑADO” ya que de no ser así la prenda no podrá ser desempeñada debido a que pudiera estar la prenda en estado “VENCIDO”, “VENDIDO” ó “CARGO “, por lo que éste cursor es imprescindible para no cometer errores de querer desempeñar principalmente una prenda que ya está vencida.
- **línea 14**: se abre el CURSOR.
- **línea 15**: `FETCH NEXT` es la única búsqueda disponible ya que no se ha especificado la opción SCROLL.
- **línea 16**: se valida si el cursor a devuelto un valor, y en donde si la función @@FETCH_STATUS devuelve el valor 0 (cero) significará que la instrucción FETCH se ejecutó correctamente, de lo contrario querrá decir que no se encontró ninguna prenda con la consulta especificada, por lo que se deduce que la prenda no puede ser desempeñada.
- **línea 17**: el comando BEGIN hace referencia a que si la validación es correcta la prenda podrá ser desempeñada y por lo tanto señala el inicio de la sentencia SQL de actualización (UPDATE).
- **línea 18-19**: definición de la sentencia UPDATE.

- **línea 20**: se valida el error de integridad.
- **línea 21**: se publica el mensaje en caso de haber un error de integridad.
- **línea 22**: declaración del ELSE que señala “en caso de no haber un error”
- **línea 23**: se publica que la prenda fue empeñada con la etiqueta MENSAJE.
- **línea 24**: se cierra el cursor con el comando CLOSE “nombre_del_cursor” en donde el nombre_del_cursor es el cursor a cerrar.
- **línea 25**: se libera el espacio en memoria que ocupa el cursor con el comando DEALLOCATE “nombre_del_cursor”.
- **línea 26**: se cierra el IF de la validación del error.
- **línea 27**: se señala el ELSE de la validación del cursor.
- **línea 28**: se especifica el comienzo del ELSE.
- **línea 29**: se publica el mensaje “la prenda no puede ser desempeñada”.
- **línea 30**: se realiza una consulta que mostrara los campos “id_prenda” y el “estado” para visualizar a partir del campo “estado” el porque la prenda no puede ser desempañada.
- **línea 31**: se cierra el cursor con el comando CLOSE “nombre_del_cursor” en donde el nombre_del_cursor es el cursor a cerrar.
- **línea 32**: se libera el espacio en memoria que ocupa el cursor con el comando DEALLOCATE “nombre_del_cursor”.
- **línea 33**: se cierra el IF ELSE de la validación del cursor con el comando END.

4.6.3 Procedimiento almacenado refrendo

Es el procedimiento almacenado encargado de llevar a cabo la tarea refrendo y se definió mediante el siguiente código:





```
DECLARE cursor_buscador CURSOR
FOR select id_prenda, estado, pago_refrendo from prendas
where id_prenda=@id_prenda and  estado in ('EMPEÑADO','REFRENDADO')
OPEN cursor_buscador
FETCH NEXT FROM cursor_buscador
if @@FETCH_STATUS = 0
begin
        if (@find=1)
                begin
           set @fecha_vencimiento=dateadd(d,32,getdate())
            update prendas set estado=@estado, fecha_movimiento=@fecha_movimiento,
            fecha_vencimiento=@fecha_vencimiento where id_prenda=@id_prenda
                end
        else
                begin
           set @fecha_vencimiento=dateadd(d,31,getdate())
            update prendas set estado=@estado, fecha_movimiento=@fecha_movimiento,
            fecha_vencimiento=@fecha_vencimiento where id_prenda=@id_prenda
                end
        IF @@ERROR = 547
           PRINT 'EL DATO INTRODUCIDO ES INCORRECTO';
        ELSE
        select 'La prenda fue actualizada' as MENSAJE
        close cursor_buscador
        DEALLOCATE cursor_buscador
end
else
begin
    select 'la prenda no puede ser actualizada' AS MENSAJE
    SELECT id_prenda, estado from prendas where id_prenda=@id_prenda
    close cursor_buscador
    DEALLOCATE cursor_buscador
end
```

Figura 4.23 (código del procedimiento almacenado *refrendar*)

CAPÍTULO 4

El código escrito para éste procedimiento almacenado se traduce de la siguiente manera:

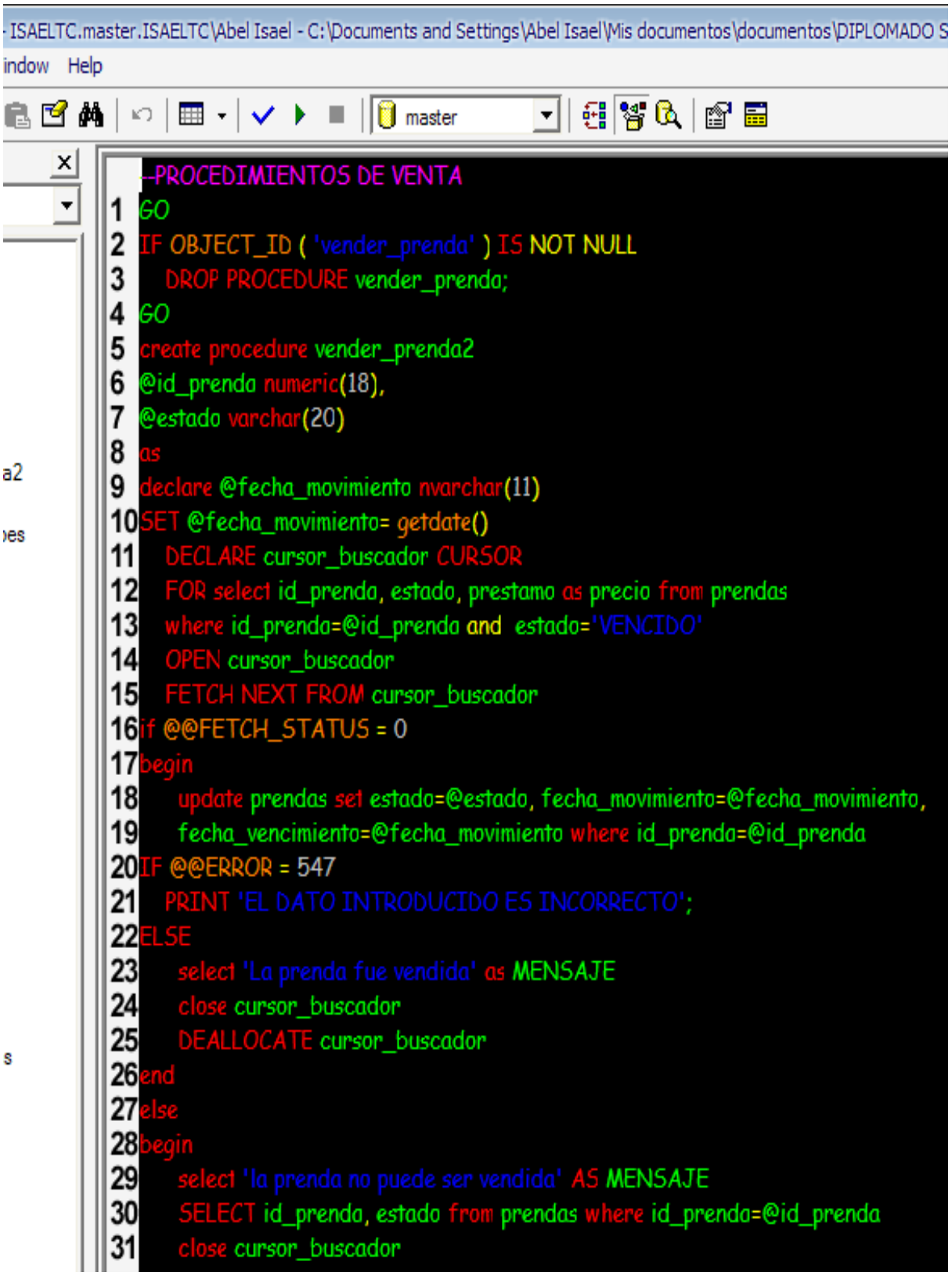
- **línea 1-4**: se verifica si el procedimiento existe y de serlo así lo borra.
- **línea 5**: se crea el procedimiento almacenado refrendar.
- **línea 6-7**: se definen las variables ó parámetros del procedimiento almacenado, que para éste caso son dos y en donde id_prenda servirá como parámetro de filtración para actualizar solo la prenda que tenga ese identificador único de prenda (id_prenda) mientras que la variable estado contendrá un valor constante igual a “REFRENDADO”.
- **línea 8**: AS éste comando especifica el comienzo de la sentencia SQL del procedimiento almacenado.
- **línea 9**: se declara una variable local (find) que servirá para identificar si una fecha cae en domingo., por lo que si fuera el caso la fecha_vencimiento será recorrida a un día después.
- **línea 10**: se define el valor de inicio de la variable *find* y en donde datepart es la función que permite conocer un día o mes para poderlo validar, por lo que para este caso el día que nos interesa encontrar es el domingo donde el día esta relacionado con un 1 y será éste el parámetro a evaluar.
- **línea 11**: es declarada la variable fecha_vencimiento para asignarla al campo del mismo nombre de la tabla “prendas” y el donde ésta fecha tendrá un valor de 31 o 32 días después de la fecha actual.
- **línea 12**: se declara la variable local fecha_movimiento que será introducido en el campo del mismo nombre de la tabla “prendas” al objeto que sea señalado.
- **línea 13**: se le asigna la fecha actual a la variable fecha_movimiento con el comando SET.
- **línea 14**: se declara un cursor de nombre cursor_buscador.
- **línea 15-16**: se declara la sentencia SQL SELECT del cursor para verificar si la prenda especificada puede ser REFRENDADA ya que para que la prenda u objeto pueda ser REFRENDADO es necesario que la prenda esté en estado EMPEÑADO ó REFRENDADO ya que de otro modo la prenda de no cumplir con éstas características no podrá ser REFRENDADA.
- **línea 17**: se abre el CURSOR.
- **línea 18**: `FETCH NEXT` es la única búsqueda disponible ya que no se ha especificado la opción SCROLL.
- **línea 19**: se valida si el cursor a devuelto un valor, y en donde si la función @@FETCH_STATUS devuelve el valor 0 (cero) significará que la instrucción FETCH se ejecutó correctamente, de lo contrario querrá decir que no se encontró ninguna prenda con la consulta especificada, por lo que se deduce que la prenda no puede ser REFRENDADA.
- **línea 20**: comienza el la instrucción derivada del IF en caso de que la prenda a REFRENDAR cumpla con las características para poder ser refrendada.
- **línea 21**: comienza la validación de la búsqueda del domingo (1) mediante un IF que se interpreta como “si la fecha cae en domingo?”

- **línea 22**: es el comando que indica el comienzo de la sentencia SQL, para el caso de que la fecha caiga en domingo.
- **línea 23**: cuando la fecha cae en domingo la variable fecha_vencimiento se recorre un día, por lo que en ésta línea se asigna el valor (la fecha recorrida) con la función DATEADD que lo que hace es agregar días o meses dependiendo de lo que se quiera logar. Para éste caso se agregaron días.
- **línea 24-25**: es definida la actualización de la prenda (UPDATE).
- **línea 26**: termina la sentencia SQL del IF.
- **línea 27**: se coloca el ELSE que para éste caso significa: en caso de que la fecha de vencimiento no sea domingo.
- **línea 28**: se coloca el BEGIN para señalar que comienza la sentencia SQL del ELSE.
- **línea 29**: se le asigna el valor a la variable fecha_vencimiento para el caso de que ésta fecha no caiga en domingo.
- **línea 30-31**: se define el UPDATE.
- **línea 32**: se cierra el ELSE.
- **línea 33**: se valida el error de integridad.
- **línea 34**: se publica el mensaje de error para cuando éste es identificado.
- **línea 35**: el ELSE para éste caso significa: en caso de no haber errores de integridad.
- **línea 36**: se publica un mensaje, señalando que la prenda pudo ser actualizada correctamente.
- **línea 37**: se cierra el cursor.
- **línea 38**: se libera el espacio en memoria que ocupa el cursor.
- **línea 39**: se cierra el IF correspondiente al de la validación de la línea 16.
- **línea 40**: se especifica el ELSE, que significa para éste caso: en caso de no cumplir con las especificaciones para poder ser REFRENDADA.
- **línea 41**: se especifica el comienzo de la sentencia SQL correspondiente al ELSE.
- **línea 42-43**: se definen dos SELECT, en donde el primero es para publicar que la prenda no pudo ser REFRENDADA y el segundo es para publicar la prenda que se quería REFRENDAR y poder visualizar el estado en que se encuentra, para poder comprobar el porque no puede ser refrendada.
- **línea 44**: se cierra el cursor.
- **línea 45**: se libera el espacio en memoria que ocupa el cursor.
- **línea 46**: se cierra el ELSE.

4.6.4 Procedimiento almacenado ventas

Es el procedimiento almacenado utilizado para realizar las ventas de los objetos y se definió mediante el siguiente código:

CAPÍTULO 4




```
--PROCEDIMIENTOS DE VENTA
GO
IF OBJECT_ID ( 'vender_prenda' ) IS NOT NULL
   DROP PROCEDURE vender_prenda;
GO
create procedure vender_prenda2
@id_prenda numeric(18),
@estado varchar(20)
as
declare @fecha_movimiento nvarchar(11)
SET @fecha_movimiento= getdate()
   DECLARE cursor_buscador CURSOR
   FOR select id_prenda, estado, prestamo as precio from prendas
   where id_prenda=@id_prenda and  estado='VENCIDO'
   OPEN cursor_buscador
   FETCH NEXT FROM cursor_buscador
if @@FETCH_STATUS = 0
begin
    update prendas set estado=@estado, fecha_movimiento=@fecha_movimiento,
    fecha_vencimiento=@fecha_movimiento where id_prenda=@id_prenda
IF @@ERROR = 547
   PRINT 'EL DATO INTRODUCIDO ES INCORRECTO';
ELSE
    select 'La prenda fue vendida' as MENSAJE
    close cursor_buscador
    DEALLOCATE cursor_buscador
end
else
begin
    select 'la prenda no puede ser vendida' AS MENSAJE
    SELECT id_prenda, estado from prendas where id_prenda=@id_prenda
    close cursor_buscador
```

```
32    DEALLOCATE cursor_buscador
33 end;
```

Figura 4.24 (código del procedimiento almacenado *vender_prenda*)

El código escrito para éste procedimiento almacenado se traduce de la siguiente manera:

- **línea 1-4**: se verifica si el procedimiento existe y de serlo así lo borra.
- **línea 5**: se crea el procedimiento almacenado vender_prenda.
- **línea 6-7**: se definen las variables ó parámetros del procedimiento almacenado, que para éste caso son dos y en donde id_prenda servirá como parámetro de filtración para actualizar solo la prenda que tenga ese identificador único de prenda (id_prenda) mientras que la variable estado contendrá un valor constante igual a “VENDIDO”.
- **línea 8**: AS éste comando especifica el comienzo de la sentencia SQL del procedimiento almacenado.
- **línea 9**: se declara una variable local (fecha_movimiento) que contendrá la fecha actual y que servirá para definírsela a los campos fecha_movimiento y fecha_vencimiento de la prenda a vender; se les asocia la misma fecha debido a que el objeto ya no está sujeto a un plazo por ser una prenda vendida.
- **línea 10**: se le asigna la fecha actual a la variable local (fecha_movimiento).
- **línea 11-13**: se declara un **CURSOR** (de nombre cursor_buscador) que servirá para validar que la prenda que se pretende vender este en estado “VENCIDO” ya que de no ser así la prenda no podrá ser VENDIDA debido a que pudiera estar la prenda en estado “EMPEÑADO”, “REFRENDADO” ó “CARGO“, por lo que éste cursor es imprescindible para no cometer errores de querer desempeñar principalmente una prenda que está en estado “EMPEÑADO”, “REFRENDADO” ó “CARGO“.
- **línea 14**: se abre el CURSOR.
- **línea 15**: `FETCH NEXT` es la única búsqueda disponible ya que no se ha especificado la opción SCROLL.
- **línea 16**: se valida si el cursor a devuelto un valor, y en donde si la función @@FETCH_STATUS devuelve el valor 0 (cero) significará que la instrucción FETCH se ejecutó correctamente, de lo contrario querrá decir que no se encontró ninguna prenda con la consulta especificada, por lo que se deduce que la prenda no puede ser vendida.
- **línea 17**: el comando BEGIN hace referencia a que si la validación es correcta la prenda podrá ser vendida y por lo tanto señala el inicio de la sentencia SQL de actualización (UPDATE).
- **línea 18-19**: definición de la sentencia UPDATE.
- **línea 20**: se valida el error de integridad.
- **línea 21**: se publica el mensaje de error para cuando éste es identificado.
- **línea 22**: el ELSE para éste caso significa: en caso de no haber errores de integridad.

- **línea 23**: se publica un mensaje, señalando que la prenda pudo ser actualizada correctamente.
- **línea 24**: se cierra el cursor.
- **línea 25**: se libera el espacio en memoria que ocupa el cursor.
- **línea 26**: se cierra el IF correspondiente al de la validación de la línea 13.
- **línea 27**: se especifica el ELSE, que significa para éste caso: en caso de no cumplir con las especificaciones para poder ser VENDIDA.
- **línea 28**: se especifica el comienzo de la sentencia SQL correspondiente al ELSE.
- **línea 29-30**: se definen dos SELECT, en donde el primero es para publicar que la prenda no pudo ser VENDIDA y el segundo es para publicar la prenda que se quería VENDER y poder visualizar el estado en que se encuentra, para poder comprobar el porque no puede ser vendida.
- **línea 31**: se cierra el cursor.
- **línea 32**: se libera el espacio en memoria que ocupa el cursor.
- **línea 33**: se cierra el ELSE.

4.6.5 Procedimiento almacenado reportes

Con éste procedimiento almacenado se generarán los siguientes reportes: reporte de prendas vencidas, reporte de prendas empeñadas, reporte de prendas refrendadas, reporte de prendas vendidas, reporte de prendas desempañadas y el reporte de prendas que pasaron a cargo; por lo que es un procedimiento bastante funcional debido a que con un solo procedimiento se pueden generar todos los reportes y no hacerlos de manera separada como seria el caso de haberlos hecho mediante vistas.

Para éste procedimiento se generó el siguiente código:

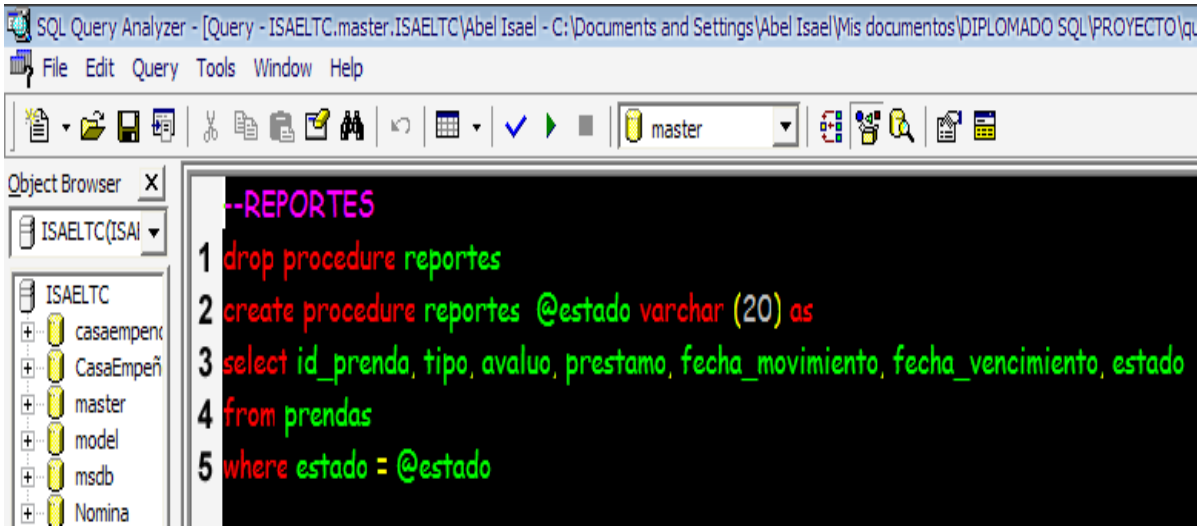


Figura 4.25 (código del procedimiento almacenado *reportes*)

CAPÍTULO 4

El código escrito para éste procedimiento almacenado se traduce de la siguiente manera:

- **línea 1**: se verifica si el procedimiento existe y de serlo así lo borra.
- **línea 2**: se crea el procedimiento almacenado “reportes”, y se define la única variable “estado” que servirá para indicarle al procedimiento que tipo de reporte se desea visualizar.
- **línea 3-5**: se define la sentencia SQL SELECT, y en donde se puede resaltar que para lograr la funcionalidad de especificarle el tipo de reporte, la variable @estado es metida dentro del filtro WHERE para que a la hora de introducirle el parámetro, el procedimiento almacenado sepa que tipo de reporte debe de enviar.

CAPÍTULO 5

5. Conclusiones y Pruebas a la BD

El presente capítulo está diseñado para que al mismo tiempo que se vaya comprobando la funcionalidad de la estructura (tablas) y programas (*trigger*'s, funciones, procedimientos almacenados, cursores y vistas), se argumenten las conclusiones de cada una de las partes que componen el sistema; para poder ir probando éstas partes, fue necesario cambiar un poco el orden del desarrollo (cap 4), ya que por ejemplo, para poder probar las funciones es necesario primero insertar algún dato con el procedimiento almacenado "empeño", por lo que el presente capítulo se fue construyendo de acuerdo a la mejor manera de ir demostrando las funcionalidades del SCD-CE.

5.1 Primera conclusión (creación de la BD)

El crear una BD en SQL Server 2000 implica crear una instancia de BD, donde la instancia no es más que un recurso dedicado del servidor, hacia el programa manejador de BD (SQL Server 2000), pero esto no quiere decir que la Base de Datos tendrá una instancia individual, ya que en SQL Server 2000, la BD creada (casaempeno) es adjuntada a Bases de Datos creadas anteriormente en el programa y a los archivos de sistema del SQL Server 2000, lo cual quiere decir que todas las bases de datos almacenadas en SQL Server 2000 comparten una sola instancia; por lo tanto se puede concluir que el crear una BD en éste programa estará sujeto de alguna manera a compartir recursos del servidor, por lo que el decidir en que programa se generaría ésta BD dependió en gran medida de lo que se pretendía construir, tomando en cuenta diversos factores como el tamaño de la BD y la estructura lógica del programa manejador de BD y las funcionalidades que éste programa facilitaba, por lo que al evaluar esos factores SQL Server 2000 fue la opción a elegir debido a que en el servidor solo se generaría ésta BD y no sería un servidor compartido, otra de las razones por las que se decidió utilizar éste programa fue que las funciones podían ser definidas en un campo de cualquier tabla lo cual permitió en ese sentido resolver partes de automatización. Es importante mencionar que en otros programas como Oracle, Postgres o MySQL, ésta funcionalidad también podría lograrse, pero en algunos casos (Oracle y Postgres) el hacerlo era consumir mas recursos de memoria ya que esto podría lograrse mediante un *Trigger* lo cual implica que al hacer la inserción no solo se realizaría ésta petición si no que también se levantaría un *query* más largo donde se tendía que evaluar que dato se insertaría y en la

manera que se resolvió éste problema las funciones eran individuales para cada campo.

A final de cuentas cada manejador de Bases de Datos tiene críticas a favor y en contra, y es imposible decir cual es mejor que otro, ya que desde mi punto de vista el mejor programa es el que mejor se tenga conocimiento para la tarea que se te encomienda.

5.2 Segunda conclusión (Elaboración de las tablas)

En ésta parte se puede concluir que el diseño entidad-relación es óptimo para las funcionalidades que se pretendían obtener, debido a que todas las tareas fueron enfocadas entorno a las prendas y no a otras tareas administrativas como reclamos, nóminas, etcétera, por lo que si bien éste modelo cumplió con las expectativas, también es importante decir que cualquier cosa puede ser mejorada, por lo que en éste trabajo nunca se pretendió decir, que las cosas solo se pueden hacer de una manera, al contrario siempre será mejor el contar con sugerencias que permitan tener un mayor número de alternativas para hacer las cosas y enriquecer nuestra manera de analizar los problemas.

5.3 Comprobación y Conclusiones de los Procedimientos Almacenados

5.3.1 Realización de inserciones a través de procedimientos almacenados y verificación de las funciones y catálogos.

Para éste punto se comprobará que las funciones realmente hagan lo que se especificó en el capítulo anterior, donde se mencionaba que iban a calcular y al mismo tiempo insertar el valor de los campos “avaluo”, “monto_interes”, “pago_refrendo” y “pago_refrendo”, también se justificará la creación de las tablas “cat_tipo” y “cat_estado” ya que se verificará que realmente se cheque la correcta captura de los datos y por otra parte se comprobará que el procedimiento almacenado “inserta_prenda” (empeño) haga las validaciones y funciones especificadas en su definición además de verificar que el procedimiento almacenado “inserta_cliente” valide que un cliente exista o no.

- ✓ Antes de probar el procedimiento almacenado, se mostrará el error que se genera cuando se quiere crear la tabla “prendas” sin haber generado primero las funciones:

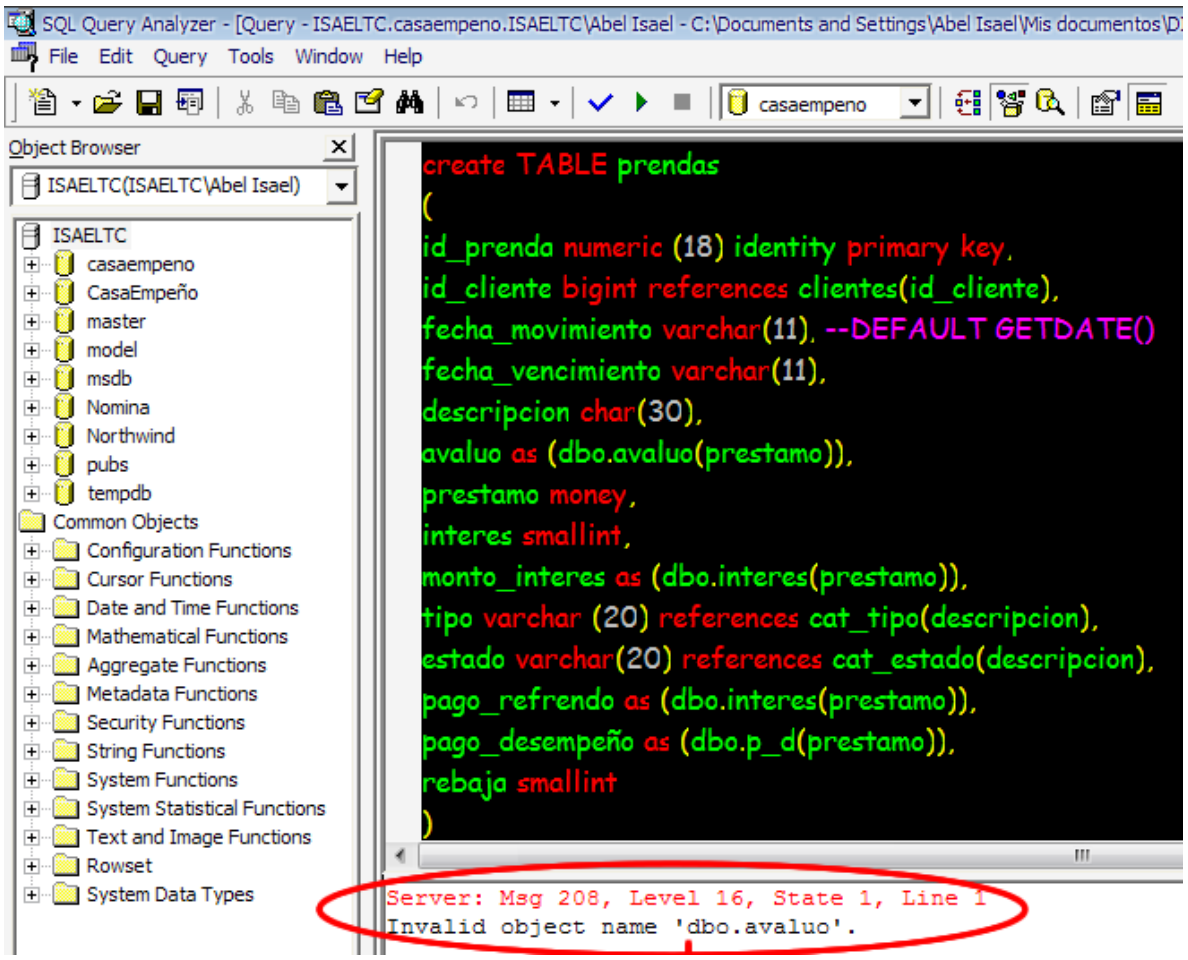


Figura 5.1 (Error de inexistencia de función)

- ✓ En el siguiente código se realizarán algunas inserciones para empezar a trabajar con la BD y para llenar los datos de los catálogos definidos:

```
--INSERTS

--INSERCIONES DE CATALOGO TIPO
INSERT INTO cat_tipo VALUES  ('A','ALAJA')
INSERT INTO cat_tipo VALUES  ('E','ELECTRODOMESTICO')
INSERT INTO cat_tipo VALUES  ('M','MUEBLES')
INSERT INTO cat_tipo VALUES  ('V','VARIOS')
```

```
--INSERCIONES DE CATALOGO ESTADO
INSERT INTO cat_estado VALUES  ('1','EMPEÑADO')
INSERT INTO cat_estado VALUES  ('2','REFRENDADO')
INSERT INTO cat_estado VALUES  ('3','VENCIDO')
INSERT INTO cat_estado VALUES  ('4','VENDIDO')
INSERT INTO cat_estado VALUES  ('5','DESEMPEÑADO')
INSERT INTO cat_estado VALUES  ('6','CARGO')

--INSERTS PRENDAS
INSERT INTO PRENDAS VALUES  (1,'Jan  2 2008', 'Feb  2 2008', 'en buen estado', 800, 4, 'ALAJA', 'VENCIDO', 9)
INSERT INTO PRENDAS VALUES  (2,'Feb  5 2008', 'Mar  5 2008', 'en buen estado', 4000, 4, 'VARIOS', 'VENCIDO', 9)
INSERT INTO PRENDAS VALUES  (3,'Mar  3 2008', 'Apr  3 2008', 'en buen estado', 5000, 4, 'ELECTRODOMESTICO', 'REFRENDADO', 9)
INSERT INTO PRENDAS VALUES  (4,'Apr 12 2007', 'May 12 2007', 'en buen estado', 6000, 4, 'ALAJA', 'VENDIDO', 9)
INSERT INTO PRENDAS VALUES  (5,'May  8 2008', 'Jun  8 2008', 'en buen estado', 7000, 4, 'MUEBLES', 'VENDIDO', 9)
INSERT INTO PRENDAS VALUES  (6,'Jun  9 2007', 'Jul  9 2007', 'en buen estado', 8000, 4, 'MUEBLES', 'CARGO', 9)
INSERT INTO PRENDAS VALUES  (7,'Jul 11 2007', 'Aug 11 2007', 'en buen estado', 9000, 4, 'VARIOS', 'CARGO', 9)
INSERT INTO PRENDAS VALUES  (8,'Aug 12 2007', 'Sep 12 2007', 'en buen estado', 2000, 4, 'ELECTRODOMESTICO', 'DESEMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (9,'Sep 13 2007', 'Oct 13 2007', 'en buen estado', 1000, 4, 'ELECTRODOMESTICO', 'DESEMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (10,'Oct 14 2007', 'Nov 14 2007', 'en buen estado', 500, 4, 'MUEBLES', 'VENCIDO', 9)
INSERT INTO PRENDAS VALUES  (11,'Nov 15 2007', 'Dec 15 2007', 'en buen estado', 100, 4, 'VARIOS', 'VENCIDO', 9)
INSERT INTO PRENDAS VALUES  (12,'Dec 16 2007', 'Jan 16 2008', 'en buen estado', 100, 4, 'VARIOS', 'VENCIDO', 9)

INSERT INTO PRENDAS VALUES  (1,'Jan  2 2008', 'Feb  2 2008', 'en buen estado', 800, 4, 'ALAJA', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (2,'Feb  5 2008', 'Mar  5 2008', 'en buen estado5', 4000, 4, 'VARIOS', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (3,'Mar  3 2008', 'Apr  3 2008', 'en buen estado', 5000, 4, 'ELECTRODOMESTICO', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (4,'Apr 12 2007', 'May 12 2007', 'en buen estado', 6000, 4, 'ALAJA', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (5,'May  8 2008', 'Jun  8 2008', 'en buen estado', 7000, 4, 'MUEBLES', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (6,'Jun  9 2007', 'Jul  9 2007', 'en buen estado', 8000, 4, 'MUEBLES', 'EMPEÑADO', 9)
```

```
INSERT INTO PRENDAS VALUES  (7,'Jul 11 2007', 'Aug 11 2007', 'en buen estado', 9000, 4, 'VARIOS', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (8,'Aug 12 2007', 'Sep 12 2007', 'en buen estado', 2000, 4, 'ELECTRODOMESTICO', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (9,'Sep 13 2007', 'Oct 13 2007', 'en buen estado', 1000, 4, 'ELECTRODOMESTICO', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (10,'Oct 14 2007', 'Nov 14 2007', 'en buen estado', 500, 4, 'MUEBLES', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (11,'Nov 15 2007', 'Dec 15 2007', 'en buen estado', 100, 4, 'VARIOS', 'EMPEÑADO', 9)
INSERT INTO PRENDAS VALUES  (12,'Dec 16 2007', 'Jan 16 2008', 'en buen estado', 100, 4, 'VARIOS', 'EMPEÑADO', 9)

--INSERCIÓN DE CLIENTES
INSERT INTO CLIENTES VALUES  ('cliente1', 'AP1', 'AM1','d1',21325648)
INSERT INTO CLIENTES VALUES  ('cliente2', 'AP2', 'AM2','d2',32568978)
INSERT INTO CLIENTES VALUES  ('cliente3', 'AP3', 'AM3','d3',56897845)
INSERT INTO CLIENTES VALUES  ('cliente4', 'AP4', 'AM4','d4',45895632)
INSERT INTO CLIENTES VALUES  ('cliente5', 'AP5', 'AM5','d5',78452132)
INSERT INTO CLIENTES VALUES  ('cliente6', 'AP6', 'AM6','d6',10325689)
INSERT INTO CLIENTES VALUES  ('cliente7', 'AP7', 'AM7','d7',32320578)
INSERT INTO CLIENTES VALUES  ('cliente8', 'AP8', 'AM8','d8',20457856)
INSERT INTO CLIENTES VALUES  ('cliente9', 'AP9', 'AM9','d9',21035678)
INSERT INTO CLIENTES VALUES  ('cliente10', 'AP10', 'AM10','d10',32568978)
INSERT INTO CLIENTES VALUES  ('cliente11', 'AP11', 'AM11','d11',56320278)
INSERT INTO CLIENTES VALUES  ('cliente12', 'AP12', 'AM12','d12',45023205)
```

Para realizar un empeño es necesario crear primero el registro del cliente donde se verifica si el número telefónico está en la BD o no, para deducir si el cliente existe o no, de tal forma que si el procedimiento almacenado “inserta_cliente” manda un mensaje de que el cliente fue agregado, significará que el cliente no existía y fue agregado, de lo contrario el cliente no se agregará, pero el programa enviará los datos de ese cliente para poder saber que “id_cliente” agregar en el nuevo registro algún objeto, ya que éste dato se debe agregar en los datos de la prenda y es el que permitirá que los clientes puedan tener varias prendas en empeño y que sus datos no se encuentren repetidos en la BD.

- ✓ La siguiente imagen mostrará la manera en que se ejecuta el procedimiento almacenado “inserta_cliente” y la forma en que se introduce a un nuevo cliente:

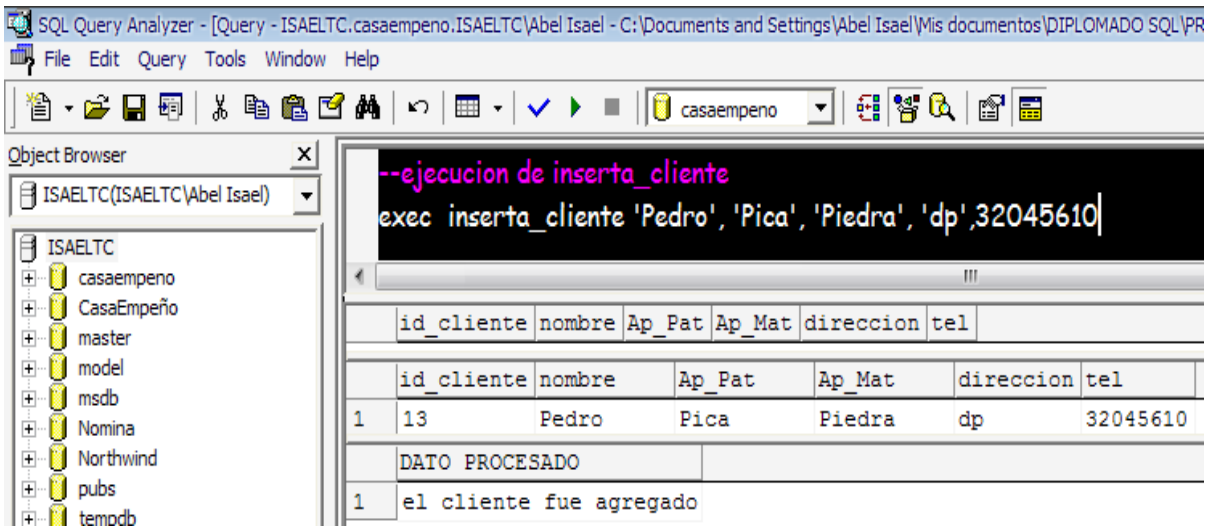


Figura 5.2 (ejecución del procedimiento almacenado *inserta_cliente*)

En la imagen se puede apreciar el mensaje “el cliente fue agregado” lo cual quiere decir que el cliente no existía.

- A continuación se muestra el otro caso donde el cliente ya existe y en donde el mensaje es “el cliente ya existe”

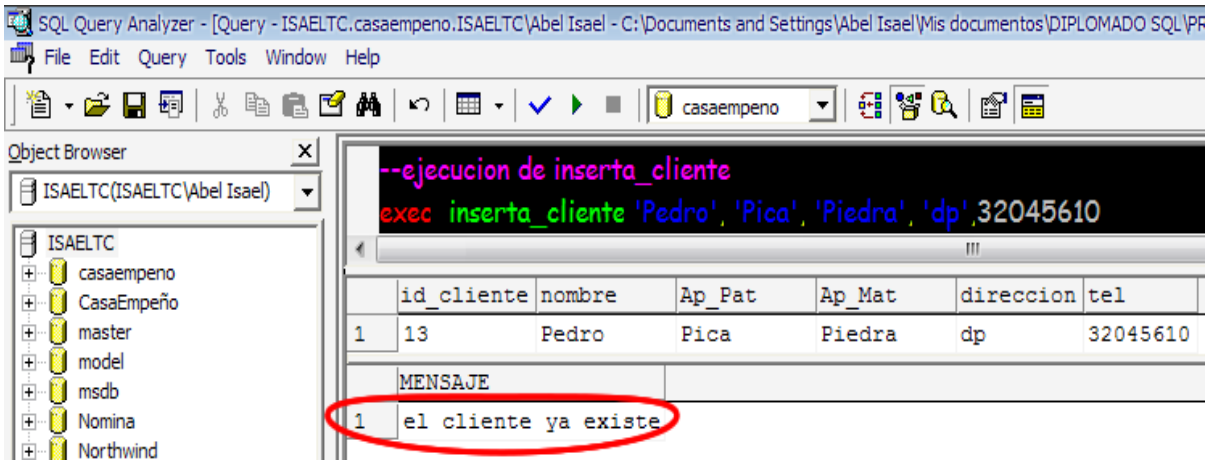


Figura 5.3 (mensaje de existencia)

Con esto se concluye que la funcionalidad especificada trabaja correctamente, por lo tanto el procedimiento almacenado es capaza de detectar cuando un cliente ya existe lo que impide que los datos puedan repetirse.

- ✓ A continuación se mostrara en la imagen, la manera en que el procedimiento almacenado “inserta_prenda” correspondiente a la tarea “empeño” se manda a llamar y como se introducen los datos:

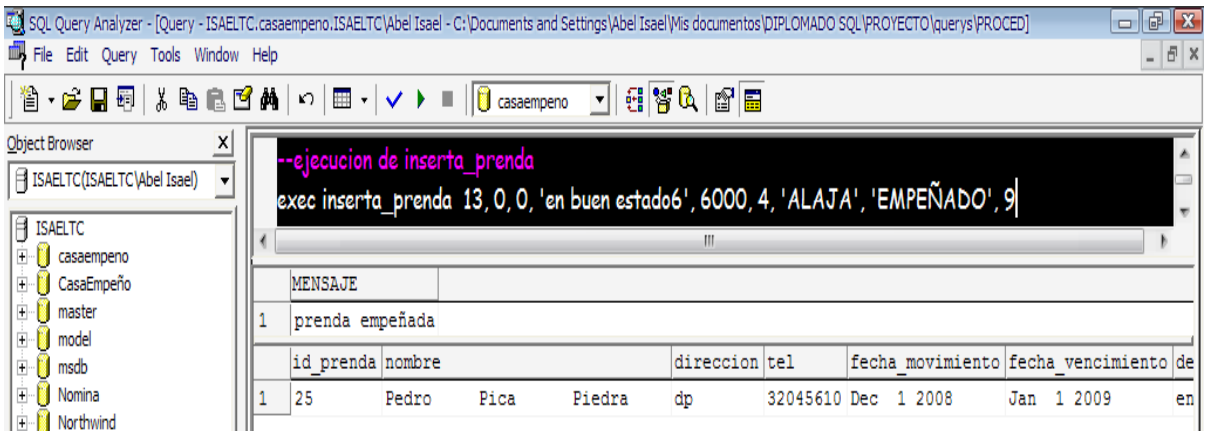


Figura 5.4 (empeño de una prenda)

Lo importante a destacar en esta parte son las fechas definidas por el sistema, ya que como puede observarse no fue necesario capturar las fechas, ni el avalúo, monto_interés, pago_refrendo y pago_desempeño por lo que de alguna manera se automatizan esas inserciones lo cual agiliza el trabajo. En la imagen de abajo se muestran los campos mencionados:

	id_prenda	prestamo	avaluo	monto_interes	pago_refrendo	pago_desempeño
1	25	6000.0000	12000.0000	240.0	240.0	6240.0000

Figura 5.5 (datos de una prenda)

Se pude observar que las operaciones o montos, son calculados e insertados perfectamente por el sistema, por lo que se puede ver el funcionamiento final de las funciones.

- ✓ Ahora se mostrará que los catálogos definidos y las referencias hechas en la tabla prendas funciones en conjunto para validar que los datos introducidos estén escritos correctamente:

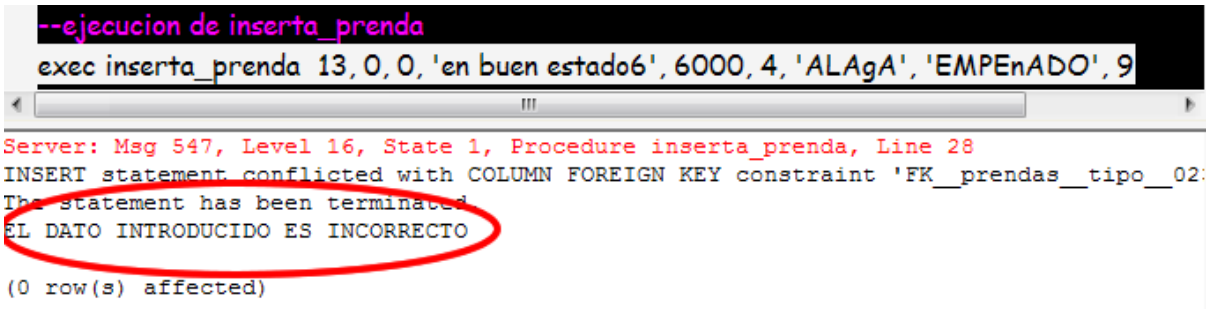


Figura 5.6 (error de sintaxis)

Donde se puede observar que al querer agregar 'ALAgA' y EMPEnADO el sistema detecta que los datos a agregar están mal escritos y automáticamente manda el mensaje de error y no permite la inserción de los datos, por lo que esto impedirá que se capturen mal los datos además de impedir que se generen errores y demuestra que la definición de la tabla prendas en conjunto con los catalogos cumplen con los objetivos señalados.

5.3.2 Actualizaciones con procedimientos almacenados

En éste punto se comprobarán que las tareas desempeño, refrendo, y venta se lleven a cabo correctamente con la utilización de los procedimientos almacenados tomando en cuenta que para realizar éstas tareas será necesario tomar en cuenta algunas validaciones como el estado en que se encuentran, para que cada una de estas tareas se lleve a cabo satisfactoriamente.

- Procedimiento almacenado desempeño

Como ya fue señalado anteriormente éste procedimiento almacenado será el encargado de realizar la tarea de desempeño, donde para poder realizar ésta tarea, la prenda deberá estar en estado EMPEÑADO, por lo que la fecha de vencimiento deberá estar al mismo tiempo vigente.

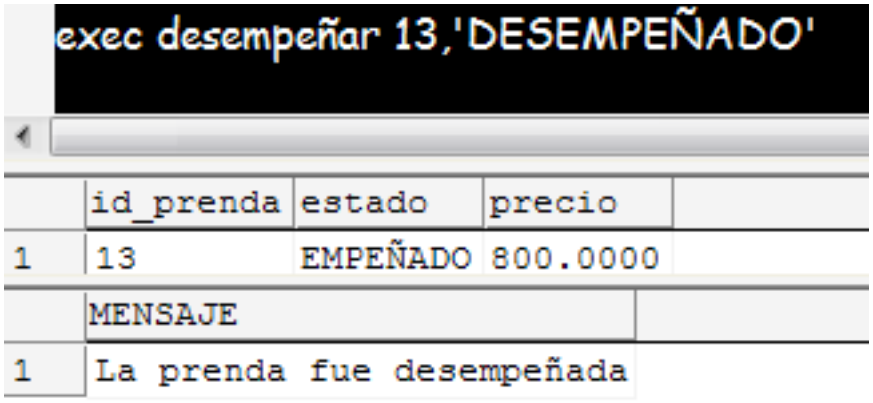


Figura 5.7 (desempeño de una prenda)

En la imagen de arriba podrá observarse que la prenda pudo ser desempeñada ya que como el sistema muestra la prenda se encontraba en estado EMPEÑADO y el costo de desempeño era de $200; para demostrar que el procedimiento hace las validaciones correspondientes para impedir ésta tarea se muestra el siguiente ejemplo:

```
exec desempeñar 13,'DESEMPEÑADO'
```

	id_prenda	estado	precio

	MENSAJE
1	la prenda no puede ser desempeñada

	id_prenda	estado
1	13	DESEMPEÑADO

Figura 5.8 (error al desempeñar)

Como se puede observa, al tratar de querer desempeñar la prenda que ya habíamos desempeñado el sistema nos arroja un mensaje, señalando que la prenda se encuentra en estado DESEMPEÑADO y que por lo tanto la prenda no puede ser desempeñada, por lo que se comprueba que el procedimiento realiza su trabajo de manera correcta impidiendo que una prenda no pueda desempeñarse si el estado de la misma no es EMPEÑADO ó REFRENDADO.

- Procedimiento almacenado refrendo

Para éste punto primero se tratará de refrendar una prenda que no cumpla con las características para poder ser refrendada buscando el id_prenda que corresponda a una prenda que se encuentre en estado VENCIDO:

		avaluo	prestamo	interes	monto_interes	tipo	estado	
1		1600.0000	800.0000	4	32.0	ALAJA	VENCIDO	
2		8000.0000	4000.0000	4	160.0	VARIOS	VENCIDO	
3		10000.0000	5000.0000	4	200.0	ELECTRODOMESTICO	REFRENDADO	
4		12000.0000	6000.0000	4	240.0	ALAJA	VENDIDO	
5		14000.0000	7000.0000	4	280.0	MUEBLES	VENDIDO	
6		16000.0000	8000.0000	4	320.0	MUEBLES	CARGO	
7		18000.0000	9000.0000	4	360.0	VARIOS	CARGO	
8		4000.0000	2000.0000	4	80.0	ELECTRODOMESTICO	DESEMPEÑADO	
9		2000.0000	1000.0000	4	40.0	ELECTRODOMESTICO	DESEMPEÑADO	
10		1000.0000	500.0000	4	20.0	MUEBLES	VENCIDO	
11		200.0000	100.0000	4	4.0	VARIOS	VENCIDO	

Figura 5.9 (estado de una prenda)

La prenda que se tratará de desempeñar será la correspondiente al id_prenda = 1.

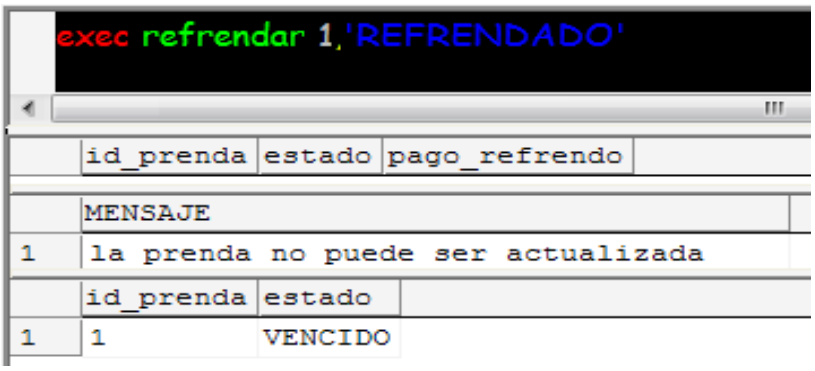


Figura 5.10 (error de refrendo)

Como se muestra en la imagen la prenda no pudo ser refrendada debido a que no cumple con las características para poderla actualizar por lo que se demuestra que el procedimientos almacenada hace las validaciones perfectamente. Ahora se mostrara un ejemplo donde se refrendará una prenda que está en estado refrendado y que corresponde al id_prenda = 3 (se puede corroborar en la figura 5.9):

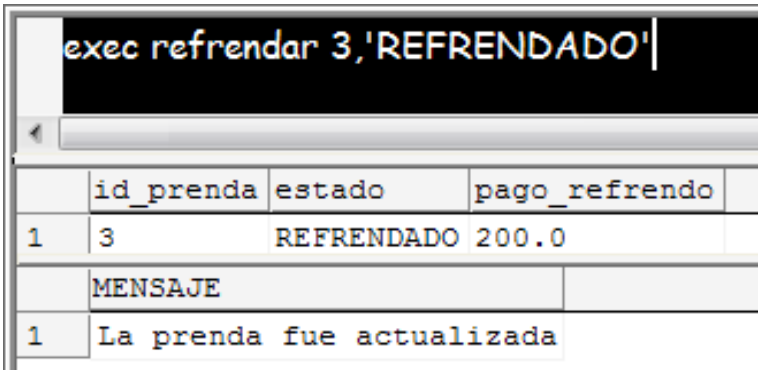


Figura 5.11 (refrendo de una prenda)

Como puede observarse, al intentar refrendar una prenda con estado apropiado el procedimiento almacenado permite hacer la actualización por lo que se confirma la buena funcionalidad del procedimiento “refrendar”.

- Procedimiento almacenado ventas

Para éste procedimiento se realizará una prueba con la prenda que se acaba de refrendar, para que el sistema nos envíe el mensaje de que la prenda no puede ser vendida ya que para que una prenda pueda llegar ser vendida, es necesario que ésta este sólo en estado VENCIDO, de otra manera ninguna prenda podrá ser vendida.

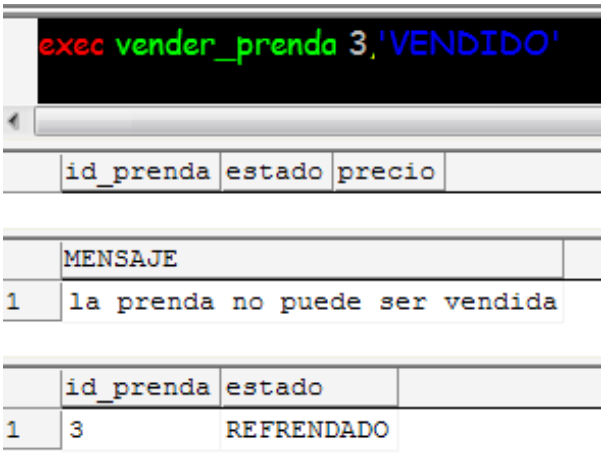


Figura 5.12 (error de venta)

Como se puede ver, efectivamente la prenda no pudo ser vendida, pero ahora se intentará vender la prenda que corresponde al id_prenda=1 y que se puede corroborar su estado en la figura*.

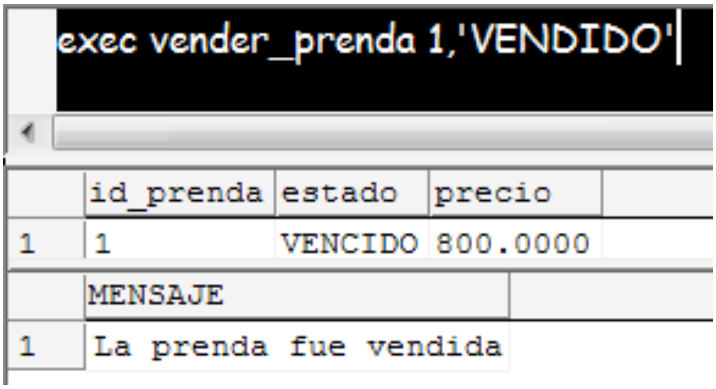


Figura 5.13 (venta de una prenda)

Con la imagen anterior se corrobora que realmente el procedimiento realiza la tarea correctamente haciendo las validaciones y actualizaciones correspondientes.

5.3.3 Verificación de los reportes con procedimientos almacenados

En éste procedimiento, lo importante a destacar es que a diferencia de hacerlo con vistas, con éste procedimiento almacenado fue posible generar todos los reportes señalados en el capítulo anterior, mediante un solo procedimiento, el cual se presenta a continuación para cada uno de sus reportes.

- Reporte de prendas empeñadas

```
exec reportes 'empeñado'
exec reportes 'refrendado'
exec reportes 'vencido'
exec reportes 'vendido'
exec reportes 'desempeñado'
exec reportes 'cargo'
```

	id_prenda	tipo	avaluo	prestamo	fecha_movimiento	fecha_vencimiento	estado
1	14	VARIOS	8000.0000	4000.0000	Feb 5 2008	Mar 5 2008	EMPEÑADO
2	15	ELECTRODOMESTICO	10000.0000	5000.0000	Mar 3 2008	Apr 3 2008	EMPEÑADO
3	16	ALAJA	12000.0000	6000.0000	Apr 12 2007	May 12 2007	EMPEÑADO
4	17	MUEBLES	14000.0000	7000.0000	May 8 2008	Jun 8 2008	EMPEÑADO
5	18	MUEBLES	16000.0000	8000.0000	Jun 9 2007	Jul 9 2007	EMPEÑADO
6	19	VARIOS	18000.0000	9000.0000	Jul 11 2007	Aug 11 2007	EMPEÑADO
7	20	ELECTRODOMESTICO	4000.0000	2000.0000	Aug 12 2007	Sep 12 2007	EMPEÑADO
8	21	ELECTRODOMESTICO	2000.0000	1000.0000	Sep 13 2007	Oct 13 2007	EMPEÑADO
9	22	MUEBLES	1000.0000	500.0000	Oct 14 2007	Nov 14 2007	EMPEÑADO
10	23	VARIOS	200.0000	100.0000	Nov 15 2007	Dec 15 2007	EMPEÑADO
11	24	VARIOS	200.0000	100.0000	Dec 16 2007	Jan 16 2008	EMPEÑADO
12	25	ALAJA	12000.0000	6000.0000	Dec 1 2008	Jan 1 2009	EMPEÑADO

Figura 5.14 (reporte de prendas empeñadas)

- Reporte de prendas desempeñadas

```
exec reportes 'empeñado'
exec reportes 'refrendado'
exec reportes 'vencido'
exec reportes 'vendido'
exec reportes 'desempeñado'
exec reportes 'cargo'
```

	id_prenda	tipo	avaluo	prestamo	fecha_movimiento	fecha_vencimiento	estado
1	8	ELECTRODOMESTICO	4000.0000	2000.0000	Aug 12 2007	Sep 12 2007	DESEMPEÑADO
2	9	ELECTRODOMESTICO	2000.0000	1000.0000	Sep 13 2007	Oct 13 2007	DESEMPEÑADO
3	13	ALAJA	1600.0000	800.0000	Dec 1 2008	Dec 1 2008	DESEMPEÑADO

Figura 5.15 (reporte de prendas desempeñadas)

- Reporte de prendas refrendadas

```
exec reportes 'empeñado'
exec reportes 'refrendado'
exec reportes 'vencido'
exec reportes 'vendido'
exec reportes 'desempeñado'
exec reportes 'cargo'
```

	id_prenda	tipo	avaluo	prestamo	fecha_movimiento	fecha_vencimiento	estado
1	3	ELECTRODOMESTICO	10000.0000	5000.0000	Dec 1 2008	Jan 1 2009	REFRENDADO

Figura 5.16 (reporte de prendas refrendadas)

- Reporte de prendas vendidas

```
exec reportes 'empeñado'
exec reportes 'refrendado'
exec reportes 'vencido'
exec reportes 'vendido'
exec reportes 'desempeñado'
exec reportes 'cargo'
```

	id_prenda	tipo	avaluo	prestamo	fecha_movimiento	fecha_vencimiento	estado
1	1	ALAJA	1600.0000	800.0000	Dec 1 2008	Dec 1 2008	VENDIDO
2	4	ALAJA	12000.0000	6000.0000	Apr 12 2007	May 12 2007	VENDIDO
3	5	MUEBLES	14000.0000	7000.0000	May 8 2008	Jun 8 2008	VENDIDO

Figura 5.17 (reporte de prendas vendidas)

- Reporte de prendas vencidas

```
exec reportes 'empeñado'
exec reportes 'refrendado'
exec reportes 'vencido'
exec reportes 'vendido'
exec reportes 'desempeñado'
exec reportes 'cargo'
```

	id_prenda	tipo	avaluo	prestamo	fecha_movimiento	fecha_vencimiento	estado
1	2	VARIOS	8000.0000	4000.0000	Feb 5 2008	Mar 5 2008	VENCIDO
2	10	MUEBLES	1000.0000	500.0000	Oct 14 2007	Nov 14 2007	VENCIDO
3	11	VARIOS	200.0000	100.0000	Nov 15 2007	Dec 15 2007	VENCIDO
4	12	VARIOS	200.0000	100.0000	Dec 16 2007	Jan 16 2008	VENCIDO

Figura 5.18 (reporte de prendas vencidas)

- Reporte de prendas cargo

```
exec reportes 'refrendado'
exec reportes 'vencido'
exec reportes 'vendido'
exec reportes 'desempeñado'
exec reportes 'cargo'
```

	id_prenda	tipo	avaluo	prestamo	fecha_movimiento	fecha_vencimiento	estado
1	6	MUEBLES	16000.0000	8000.0000	Jun 9 2007	Jul 9 2007	CARGO
2	7	VARIOS	18000.0000	9000.0000	Jul 11 2007	Aug 11 2007	CARGO

Figura 5.19 (reporte de cargos)

Con esto podemos concluir que realmente se logra la funcionalidad que se quería, por lo que hacerlo mediante un procedimiento almacenado y no con visas nos ahorramos espacio en disco y de igual forma se logra la funcionalidad deseada.

5.4 Conclusiones Triggers

Los *trigger*s en éste proyecto juegan un papel muy importante debido a que de ellos dependerá que se automatice la tarea de verificar que la fecha de vencimientos de cada una de las prendas no esté expirada, ya que de ser así el

trigger pasa_a_vencido actualizará su campo “estado” a estado VENCIDO, por otra parte las prendas que estén en estado VENCIDO y no se hallan vendido en un lapso mayor a cinco meses serán actualizadas a estado CARGO por el *trigger* pasar a cargo. Para demostrar la funcionalidad de éstos *trigger*’s es necesario primero enseñar las prendas que serán afectadas por cada uno de los disparadores, para poder después visualizar los cambios que se generaron, cuando éstos están en funcionamiento. En la siguiente imagen se muestra las prendas que serán afectadas con el *trigger* pasar_a_vencido donde se observará que las prendas que tienen el id_prenda del 14-24 serán afectadas por el *trigger* pasar_a_vencido; mientras que las prendas que serán afectadas por el *trigger* pasar_a_cargo serán las prendas que tienen el id_prenda igual a 2,10 y 11.

	id_prenda	fecha_movimiento	fecha_vencimiento	estado
1	1	Dec 1 2008	Dec 1 2008	VENDIDO
2	2	Feb 5 2008	Mar 5 2008	VENCIDO
3	3	Dec 1 2008	Jan 1 2009	REFRENDADO
4	4	Apr 12 2007	May 12 2007	VENDIDO
5	5	May 8 2008	Jun 8 2008	VENDIDO
6	6	Jun 9 2007	Jul 9 2007	CARGO
7	7	Jul 11 2007	Aug 11 2007	CARGO
8	8	Aug 12 2007	Sep 12 2007	DESEMPEÑADO
9	9	Sep 13 2007	Oct 13 2007	DESEMPEÑADO
10	10	Oct 14 2007	Nov 14 2007	VENCIDO
11	11	Nov 15 2007	Dec 15 2007	VENCIDO
12	12	Dec 16 2007	Jan 16 2008	VENCIDO
13	13	Dec 1 2008	Dec 1 2008	DESEMPEÑADO
14	14	Feb 5 2008	Mar 5 2008	EMPEÑADO
15	15	Mar 3 2008	Apr 3 2008	EMPEÑADO
16	16	Apr 12 2007	May 12 2007	EMPEÑADO
17	17	May 8 2008	Jun 8 2008	EMPEÑADO
18	18	Jun 9 2007	Jul 9 2007	EMPEÑADO
19	19	Jul 11 2007	Aug 11 2007	EMPEÑADO
20	20	Aug 12 2007	Sep 12 2007	EMPEÑADO
21	21	Sep 13 2007	Oct 13 2007	EMPEÑADO
22	22	Oct 14 2007	Nov 14 2007	EMPEÑADO
23	23	Nov 15 2007	Dec 15 2007	EMPEÑADO
24	24	Dec 16 2007	Jan 16 2008	EMPEÑADO
25	25	Dec 1 2008	Jan 1 2009	EMPEÑADO

Figura 5.20 (búsqueda de fechas vencidas)

A continuación se presenta como actúan cada uno de los *trigger*s mostrando los cambios que éstos producen al ser llamados (disparados):

5.4.1 Trigger pasar prenda a estado vencido

Éste *trigger* se dispara cuando se realiza una inserción y será comprobado mediante la inserción de una prenda nueva:

```
exec inserta_prenda 14, 0, 0, 'en buen estado6', 6000, 4, 'ALAJA', 'EMPEÑADO', 9
```

	id_prenda	fecha_movimiento	fecha_vencimiento	estado
1	1	Dec 1 2008	Dec 1 2008	VENDIDO
2	2	Dec 1 2008	Dec 1 2008	CARGO
3	3	Dec 1 2008	Jan 1 2009	REFRENDADO
4	4	Apr 12 2007	May 12 2007	VENDIDO
5	5	May 8 2008	Jun 8 2008	VENDIDO
6	6	Jun 9 2007	Jul 9 2007	CARGO
7	7	Jul 11 2007	Aug 11 2007	CARGO
8	8	Aug 12 2007	Sep 12 2007	DESEMPEÑADO
9	9	Sep 13 2007	Oct 13 2007	DESEMPEÑADO
10	10	Dec 1 2008	Dec 1 2008	CARGO
11	11	Dec 1 2008	Dec 1 2008	CARGO
12	12	Dec 1 2008	Dec 1 2008	CARGO
13	13	Dec 1 2008	Dec 1 2008	DESEMPEÑADO
14	14	Dec 1 2008	Dec 1 2008	VENCIDO
15	15	Dec 1 2008	Dec 1 2008	VENCIDO
16	16	Dec 1 2008	Dec 1 2008	VENCIDO
17	17	Dec 1 2008	Dec 1 2008	VENCIDO
18	18	Dec 1 2008	Dec 1 2008	VENCIDO
19	19	Dec 1 2008	Dec 1 2008	VENCIDO
20	20	Dec 1 2008	Dec 1 2008	VENCIDO
21	21	Dec 1 2008	Dec 1 2008	VENCIDO
22	22	Dec 1 2008	Dec 1 2008	VENCIDO
23	23	Dec 1 2008	Dec 1 2008	VENCIDO
24	24	Dec 1 2008	Dec 1 2008	VENCIDO
25	25	Dec 1 2008	Jan 1 2009	EMPEÑADO

Figura 5.21 (fechas actualizadas)

Como puede verse las prendas señaladas fueron actualizadas satisfactoriamente.

5.4.2 Trigger pasar prenda a estado cargo

Éste *trigger* se dispara cuando se realiza una actualización y será comprobado mediante la actualización (refrendo):

```
exec refrendar 28,'REFRENDADO'
```

	id_prenda	fecha_movimiento	fecha_vencimiento	estado
1	1	Dec 1 2008	Dec 1 2008	VENDIDO
2	2	Dec 1 2008	Dec 1 2008	CARGO
3	3	Dec 1 2008	Jan 1 2009	REFRENDADO
4	4	Apr 12 2007	May 12 2007	VENDIDO
5	5	May 8 2008	Jun 8 2008	VENDIDO
6	6	Jun 9 2007	Jul 9 2007	CARGO
7	7	Jul 11 2007	Aug 11 2007	CARGO
8	8	Aug 12 2007	Sep 12 2007	DESEMPEÑADO
9	9	Sep 13 2007	Oct 13 2007	DESEMPEÑADO
10	10	Dec 1 2008	Dec 1 2008	CARGO
11	11	Dec 1 2008	Dec 1 2008	CARGO
12	12	Dec 1 2008	Dec 1 2008	CARGO
13	13	Dec 1 2008	Dec 1 2008	DESEMPEÑADO
14	14	Dec 1 2008	Dec 1 2008	VENCIDO
15	15	Dec 1 2008	Dec 1 2008	VENCIDO
16	16	Dec 1 2008	Dec 1 2008	VENCIDO
17	17	Dec 1 2008	Dec 1 2008	VENCIDO
18	18	Dec 1 2008	Dec 1 2008	VENCIDO
19	19	Dec 1 2008	Dec 1 2008	VENCIDO
20	20	Dec 1 2008	Dec 1 2008	VENCIDO
21	21	Dec 1 2008	Dec 1 2008	VENCIDO
22	22	Dec 1 2008	Dec 1 2008	VENCIDO
23	23	Dec 1 2008	Dec 1 2008	VENCIDO
24	24	Dec 1 2008	Dec 1 2008	VENCIDO
25	25	Dec 1 2008	Jan 1 2009	EMPEÑADO
26	28	Dec 1 2008	Jan 1 2009	REFRENDADO

Figura 5.22 (fechas)

Como se muestra en la figura de arriba el *trigger* pasar a cargo de igual forma funciona correctamente de acuerdo a su definición y objetivos.

5.4.3 Ventajas que proveen los Triggers

Con esto podemos concluir que la utilización de los *trigger*'s es fundamental para cuando se quiere automatizar algún proceso, pero también es importante señalar que el abuso de ellos podría generar que el sistema se haga lento, ya que los *trigger*'s se disparan automáticamente, lo cual quiere decir que en vez de hacer una petición se estarán realizando dos; pero para éste proyecto los *trigger*'s fueron definidos de distinta manera para evitar este tipo de problema, ya que un *trigger* se disparará cuando se genere una inserción, mientras que el otro se disparará cuando se haga alguna actualización, para que de ésta forma los *trigger*'s funcionen correctamente y no compliquen la velocidad del sistema.

5.5 Comprobación y conclusión de la vista ingresos_egresos.

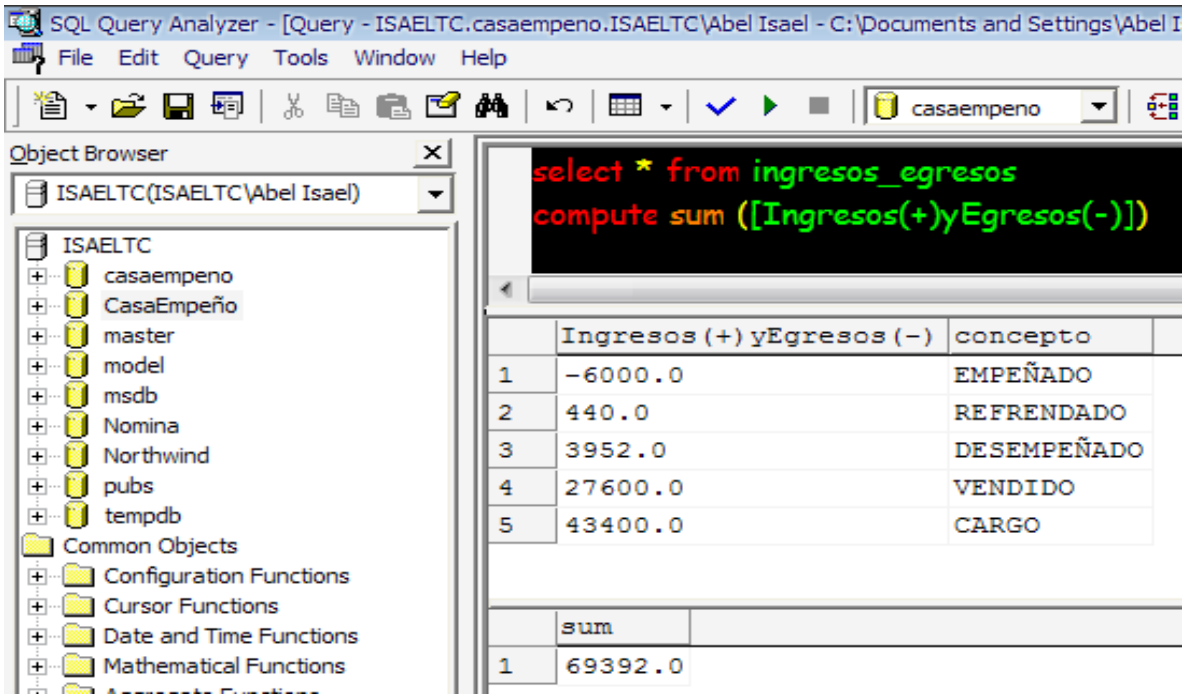


Figura 5.23 (totales por concepto)

Como puede observarse al manda a llamar la vista ingresos_egresos, el sistema envía una tabla en donde se muestran dos columnas, una donde se

muestra el conjunto de los ingresos y egresos (-) y otra en donde se muestra el concepto por el cual se señala la cantidad. Para hacer la suma total, como fue señalado en el capítulo anterior es necesario utilizar el comando COMPUTE que es el que arrojará el total (ganancias o perdidas).

Con esto se demuestra que la vista funciona y que verdaderamente se obtiene el cálculo de los ingresos totales de la casa de empeño que fue descrita durante todo el proyecto.

5.6 Conclusiones generales

El utilizar SQL Server 2000 para la realización de éste proyecto, permitió realizar todo lo imaginado para la funcionalidad que se requería, permitiendo automatizar tareas y validar datos que se ingresaban, ayudando a mejorar los procedimientos que se hacen en éste sistema, además de prevenir y corregir posibles errores que se pudieran presentar. Por otra parte el sistema cumplió con los objetivos planteados al inicio del proyecto; es importante señalar que durante el desarrollo de éste sistema se presentaron dudas en cuanto al como realizar los problemas a resolver, ya que por ejemplo, los reportes pudieron haberse hecho mediante vistas y no con el procedimiento almacenado que fue la solución para ésta parte, pero se dedujo que era mejor mediante la generación de un solo procedimiento almacenado, ya que en vez de realizar esta tarea con 20 líneas de código que es lo que se hubiera codificado mediante vistas, se resolvió el problema con solo 5 líneas, lo cual fue una de las razones que influyó a que se haya programado de la manera en que se hizo; también la definición de las tablas fue otra de las cuestiones que surgieron y que fue resuelta satisfactoriamente, pero no se hace a un lado la idea de que el sistema puede mejorar. Es importante mencionar que la utilización de *TRIGGER*'s y las validaciones que se hicieron dentro de los procedimientos almacenados es sólo una forma de resolver el problema, ya que existen otras posibilidades que permiten la optimización en cuanto a la velocidad de respuesta, como por ejemplo, programar las validaciones desde un lenguaje como PHP (que se ejecuta del lado del servidor) o javascript (lenguaje que se ejecuta del lado del *cliente*), para liberar ejecuciones en el manejador de BD y se pueda distribuir la carga de trabajo con otras herramientas. Por otro lado los *trigger*'s son procesos que se ejecutan cuando se presenta el evento esperado, por lo que en ves de ejecutarse solo el *query* especificado, se ejecuta también el *trigger*, lo cual genera una petición más pesada. Para determinar cuando utilizar un *trigger* o no, es importante tener en cuenta la funcionalidad del sistema, además de otros factores que permitan hacer una valoración de su uso, para poder deducir si es adecuada o no la utilización de ellos.

El desarrollo de éste proyecto permitió ilustrar la mayor parte de los temas que se vieron durante el diplomado, ya que se utilizaron diversas opciones como

cursores, funciones, procedimientos almacenados, *trigger* y caracteres especiales como @@ERROR, por mencionar solo algunas de las posibilidades que el SQL Server 2000 provee para para la realización de sistemas. En general, todos los objetivos planteados pudieron resolverse satisfactoriamente, ilustrando los temas vistos en el 2° *DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE*.

CAPÍTULO 5

GLOSARIO

- Almoneda

Es el lugar a donde pasan las prendas que no fueron reclamadas y en donde son mostradas al público en general para su venta.

- Cajero de desempeño

Es el encargado de realizar las operaciones de refrendo y desempeño, este trabajador obtiene el estado de la prenda introduciendo en el sistema el número de prenda (#boleta).

- Cargo

Es la prenda que tiene que pagar el valuador después de que ésta no se haya vendido, en el tiempo establecido por la empresa (para este caso se define un lapso no mayor a tres meses).

- Cursor

En bases de datos, el término **cursor** se refiere a una estructura de control utilizada para el recorrido (y potencial procesamiento) de los registros del resultado de una **consulta**.

- DB (Data Base)

BD ó Base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

- Demasía

Es el remanente que queda a favor del pignorante, después de que la Institución descuenta del monto de la venta, el préstamo, los intereses generados, los gastos de almacenaje y los gastos de operación. Las demasías pueden cobrarse dentro de un plazo de 6 meses nominales después de la fecha de venta, presentando su billete de empeño y una identificación oficial.

- Depositario

Es el que se encarga de almacenar las prendas y resguardarlas, ya que es el que se encarga de llevar a cabo el control de lo que entra y sale.

- Desempeño

Es el proceso mediante el cual, el interesado o pignorante, cumpliendo con lo pactado en el contrato de Prenda y de acuerdo a las condiciones del billete de empeño, puede recuperar la prenda depositada en garantía, pagando la suma de dinero que se le prestó más los intereses generados y lo correspondiente a gastos de almacenaje. Una vez realizado el pago, se le entregará la prenda que dejó en garantía. Deberá presentarse un día hábil antes de la fecha límite para realizar la operación de desempeño y evitar así la comercialización de la prenda.

- Empeño

Proceso en el que el usuario o pignorante recibe en forma inmediata una suma de dinero en efectivo a cambio de dejar en depósito - y a modo de garantía- una prenda de su propiedad. Al usuario se le entrega un billete o boleta de empeño que incluye la descripción de la prenda depositada y las condiciones para su recuperación basadas en el Contrato de Prenda. El valor de la prenda a empeñar deberá ser de 50 pesos mínimo. El término de plazo de empeño es de 4 meses con opción de desempeño o refrendo en el período del quinto mes nominal.

- Entregador de prendas

Este empleado, visualiza en el sistema las prendas que han sido desempeñadas y es el que se encarga de mandarlas a pedir con el auxiliar de deposito que al mismo tiempo con el depositario revisan los datos de la prenda solicitada con los datos de la prenda almacenada, en donde una vez revisado la prenda, se da de baja del deposito.

- Expendedor

Es el que se encarga del resguardo de las prendas en la almoneda (tienda) y también es el que las vende.

- Grafo

Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.

Típicamente, un grafo se representa gráficamente como un conjunto de puntos (vértices o nodos) unidos por líneas (aristas).

Desde un punto de vista práctico, los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras. Por ejemplo, una red de computadoras puede representarse y estudiarse mediante un grafo, en el cual los vértices representan terminales y las aristas representan conexiones (las cuales, a su vez, pueden ser cables o conexiones inalámbricas).

Prácticamente cualquier problema puede representarse mediante un grafo, y su estudio trasciende a las diversas áreas de las ciencias duras y las ciencias sociales.

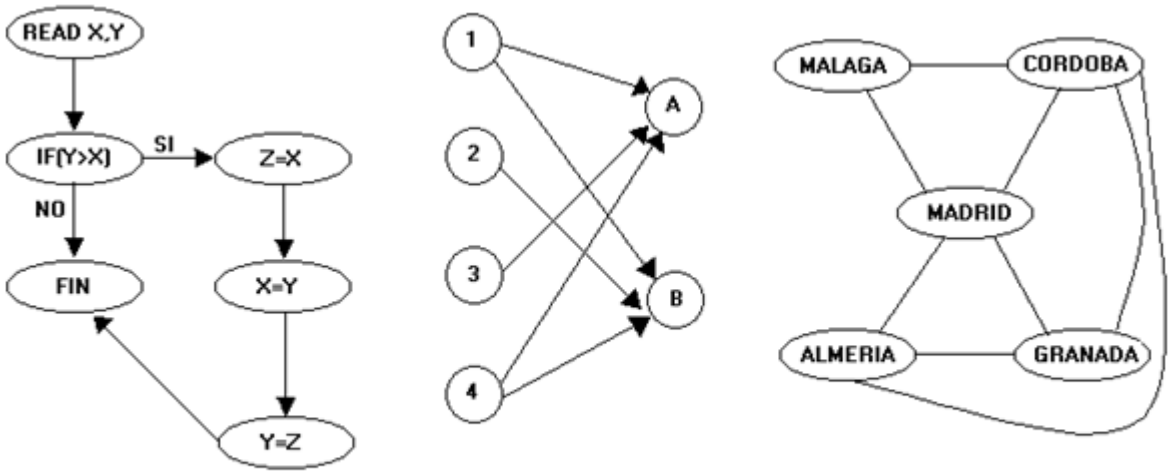


Figura :Grafos simples

- Hardware

Se refiere a todas las partes físicas y tangibles de una computadora (sus componentes eléctricos, electrónicos, electromecánicos y mecánicos).

- Jefe de almoneda

Es el que coordina al auxiliar de almoneda, auxiliar de deposito, expendedor de prendas (vendedor) y el cajero de almoneda. En almoneda se llevan a cabo las

ventas de las prendas que no fueron reclamadas (desempeñadas o refrendadas), además de productos nuevos.

- Metadatos

Son datos que describen otros datos. En general, un grupo de metadatos se refiere a un grupo de datos, llamado *recurso*. El concepto de metadatos es análogo al uso de índices para localizar objetos en vez de datos.

- Procedimiento Almacenado (Stored Procedure)

Es un programa (o procedimiento) el cual es almacenado físicamente en una base de datos.

- Refrendo (sólo aplica en alhajas y relojes)

El interesado o pignorante, una vez que ha cumplido con lo pactado en el Contrato de Prenda y de acuerdo a las condiciones de la boleta de empeño, puede optar por un nuevo plazo en el contrato de prenda. Es decir, si carece de todo el dinero para desempeñar su prenda, puede sólo pagar los intereses generados y lo correspondiente a los gastos de almacenaje, y tener un nuevo plazo para poder realizar su pago y desempeñar su prenda. Solo se permiten tres refrendos por prenda, de una duración de 4 meses cada uno. Al término de cada refrendo, el pignorante deberá presentarse, dos días hábiles antes de su fecha de vencimiento, en el área de refrendos, para establecer su próxima operación.

- SCD-CE

Son las siglas que dan título a éste trabajo y significan Sistema de Control de Datos Para una Casa de Empeño.

- Servidor

En informática, un **servidor** es una computadora que, formando parte de una red, provee servicios a otras computadoras denominadas clientes.

- Software

Es el equipamiento lógico o soporte lógico de una computadora, y comprende el conjunto de los componentes lógicos necesarios para hacer posible la realización

de una tarea específica. Tales componentes lógicos incluyen, entre otros, aplicaciones informáticas tales como procesador de textos, que permite al usuario realizar todas las tareas concernientes a edición de textos; software de sistema, tal como un sistema operativo, el que, básicamente, permite al resto de los programas funcionar adecuadamente, facilitando la interacción con los componentes físicos y el resto de las aplicaciones, también provee una interface ante el usuario.

- SQL

Lenguaje de consulta estructurado (**S**tructured **Q**uery **L**anguage) es el lenguaje estándar para trabajar con bases de datos.

- Tabla

Tabla en Bases de Datos es el lugar donde se almacenan los datos recogidos por un programa. Una tabla esta compuesta por Campos (columnas de la tabla) y Registros (renglones de la tabla).

- Trigger

Un **trigger** (o disparador) en una Base de datos , es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación de inserción (INSERT), actualización (UPDATE) o borrado (DELETE).

- Valuador

Es el que se encarga de asignarle un valor a la prenda que se desea empeñar, según las condiciones y tipo de producto a evaluar, el cual dependiendo del precio asignado, el valuador otorgará el 50% del valor (evaluado) de la prenda como préstamo.

BIBLIOGRAFÍA

• LIBROS

Mario Gerardo, Piattini Velthuis, Esperanza Marcos Martínez; *Diseño de Bases de Datos Relacionales*; RA-MA, 1999.

Peter Rob, Carlos Coronel; *Sistemas de Bases de Datos (Diseño, Implementación y Administración)*; Thomson, 5ª Edición.

Esperanza Marcos Martínez, Coral Calero Muñoz, Belén Vela Sánchez; *Tecnología y Diseño de Bases de datos*; Alfa Omega RA-MA, 1ª Edición-Abril 2007.

Adoración de Miguel, Mario Piattini; *Fundamentos y Modelos de Bases de Datos*; Alfa Omega RA-MA, 2ª Edición.

Dolores Cuadra, Elena Castro, Ana Iglesias, Paloma Martínez, Francisco Javier Calle, César de Pablo, Lourdes Moreno; *Desarrollo de bases de Datos (casos prácticos desde el análisis hasta la implementación)*; Alfa Omega RA-MA, 1ª Edición, México-Abril 2008.

• REFERENCIAS ELECTRÓNICAS

http://www.montepiedad.com.mx/ (historia)

http://www.wikilearning.com/curso_gratis/estructura_de_las_bases_de_datos_relacionales-lenguajes_de_consulta_a_bases_de_datos_relacionales/3623-4
(Lenguajes de Consulta a Bases de Datos Relacionales)

http://www.cs.buap.mx/~dpinto/bd/bdintro.pdf (Concepto Base de Datos)

http://www.htmlpoint.com/sql/sql_04.htm (Historia SQL)

http://es.geocities.com/lenguajesde_recuperacion/historia.htm (Historia SQL)

http://www.cs.us.es/cursos/bd-2001/temas/diseno.html (Modelo Entidad-Relación)

http://msdn.microsoft.com/es-es/library/ms186755.aspx (function)

http://msdn.microsoft.com/es-es/library/ms189799.aspx (Trigger)
http://msdn.microsoft.com/es-es/library/ms187926.aspx (store procedure)

http://es.wikipedia.org/wiki/Metadato (Concepto Metadatos)

Apuntes 2° DIPLOMADO EN DISEÑO DE SISTEMAS DE INFORMACIÓN ORIENTADO A NEGOCIOS CON SQL SERVER Y SQL ORACLE; Modulo2