UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

"A R A G O N"

INGENIERIA EN COMPUTACION

EL PRECOMPILADOR PRO*C

T E S I S

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A :

MANUEL ANTONIO LOPEZ GUERRERO

ASESOR DE TESIS:

ING. LUIS LORENZO JIMENEZ GARCIA

SAN JUAN DE ARAGON, EDO. DE MEX. JULIO, 1995

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

A mis padres, con todo mi amor, respeto y cariño porque gracias a ellos he alcanzado una de mis mayores metas.

A mis hermanos Carmen, Irma, Armando, Eduardo Martha y Moisés por su cariño, apoyo y comprensión.

A mis abuelitos y tíos porque siempre he recibido sus palabras de aliento.

A Silvia con todo mi amor, por tu apoyo y porque en ti encontré una pareja maravillosa.

A mis sobrinos Pabel y Nadia porque los quiero como dos hermanos mas.

A todos mis amigos y compañeros de la escuela que siempre estuvieron conmigo.

A la Escuela Nacional de Estudios Profesionales Aragón por darme la oportunidad de lograr ser un profesionista.

ÍNDICE

INTRODUCCIÓN

En el presente trabajo se muestran las características del precompilador Pro*C que lo han convertido en una herramienta muy poderosa en el desarrollo de programas de diversa índole.

En general, el precompilador Pro*C nos permite combinar dos aspectos importantes en la programación, como lo son las Bases de Datos y los lenguajes de alto nivel. El objetivo es mostrar como se lleva a cabo esta vinculación entre los argumentos y funciones de SQL con las instrucciones de un programa escrito en lenguaje C.

El precompilador es una herramienta de programación de la Base de Datos Relacional ORACLE. Dentro de esta existen precompiladores para varios lenguajes de alto nivel, por ejemplo C, COBOL o FORTRAN llamados Pro*C, Pro*COBOL y Pro*FORTRAN respectivamente. Por ser el tema del presente trabajo solo se desarrollo el tema correspondiente al precompilador Pro*C.

Al ser el precompilador Pro*C una herramienta de la Base de Datos ORACLE, en el capítulo I se dan las características principales de esta Base de Datos, como los son sus estructuras de archivos y de memoria, y su Sistema Manejador de Base de Datos Relacional RDBMS.

Como todo ORACLE tiene varias limitaciones, como por ejemplo, el número de procesos simultáneos, número de cursores abiertos, número de archivos de la Base de Datos que se pueden crear son definidos en el archivo init.ora y no pueden pasar de ciertos valores. También se tienen limitaciones por el sistema operativo como puede ser que el tamaño máximo de un archivo en UNIX es de 2Gb, el número de usuarios que depende de la licencia que se tenga.

En el capítulo II se trata el tema del Lenguaje Estructurado de Consulta (SQL), como el lenguaje através del cual se accesa a la información contenida en la Base de Datos. Este capítulo es importante, ya que los argumentos y funciones de SQL son parte fundamental en la construcción de un Pro*C, ya que através de estos se extrae la información de la Base de Datos y es utilizada por las instrucciones en lenguaje C para realizar procesos y entregar información.

En el capítulo III se trata más a fondo el funcionamiento del Pro*C, se mencionan sus características principales, se dan cinco ejemplos básicos para la comprensión de los principales comandos utilizados, se muestran las rutinas que los programas utilizan para el manejo de errores y por último los procesos de compilación y precompilación. Este capítulo sirve como base para el siguiente capítulo donde se muestran programas mas elaborados.

El capítulo IV es una aplicación del Pro*C, es decir, se muestran los programas desarrollados en Pro*C que componen la parte fundamental de un sistema. Este sistema está formado por programas en Pro*C, programas en C, pantallas desarrolladas en sqlforms, que es otra herramienta de ORACLE y programas en lenguaje de control de UNIX llamado shell que son básicamente programas con comandos de Sistema Operativo. En este capítulo se muestran los programas en Pro*C y en el Apéndice B se muestran todos los demás programas.

En el Apéndice A se muestran las estructuras de las tablas utilizadas por el sistema del capítulo IV, y que atraves de las cuales se obtiene toda la información para los procesos y los reportes.

Para poder trabajar con el Precompilador Pro*C es necesario tener instalada la Base de Datos ORACLE en nuestro equipo y tener instalado el compilador de C.

A continuación se muestra un cuadro con las plataformas donde puede ser instalado ORACLE y los requerimientos de espacio y memoria necesarios.

PLATAFORMA	SISTEMA OPERATIVO	ESPACIO EN DISCO	MEMORIA
DATA GENERAL DG/UX Aviion	DG/UX 5.4.3	65 Mb	6.5 Mb
DEC ALPHA AXP Open VMS	Open VMS alpha 1.0	270000 blocks	64 Mb
DEC RISC Serie Ultrix 32	Ultrix 4.3	126.3 Mb	29.8 Mb
DEC VAX Open VMS	VMS 4.7 o posterior	170000 blocks	16 Mb
DESKTOP MS-DOS	DOS 3.3 o posterior	10.3 Mb	3.5 Mb de Mem. exten.
DESKTOP:OS/2	OS/2 1.21 o posterior	11 Mb	8 Mb de RAM
HP 9000 Series 7XX/8XX HP-UX	HP-UX 9.0 o posterior	65Mb	> 16 Mb
IBM RS/6000	AIX 3.2.3 y posterior	155 Mb	40 Mb
NCR System 3000	MultiProcessor 2.03	65 Mb	8 Mb
Netware	Netware V3.11	Configuración server 35 Mb	16 Mb
SEQUENT DYNIX/ptx	DYNIX/ptx V 1.4	86 Mb	64 Mb
Sun 4/SPARC	Sun OS 4.1.3	74.3 Mb	12 Mb

CAPITULO I

Los Precompiladores como una Herramienta de ORACLE

I.1 Qué es un Precompilador?

Un precompilador es una interfase entre un lenguaje de alto nivel (como el C, FORTRAN, Pascal, Cobol, etc.) y la Base de Datos Relacional ORACLE (que proviene de la palabra oráculo).

Usando un precompilador es posible desarrollar una aplicación en un lenguaje de alto nivel que contenga argumentos escritos en lenguaje SQL. El Lenguaje Estructurado de Consulta (Structured Query Language SQL), es un lenguaje para accesar a la información que se encuentra en una Base de Datos Relacional.

ORACLE provee una interfase para programadores de aplicaciones que es la interfase de precompiladores. El precompilador traduce los argumentos de SQL que aparecen en el programa dentro del código fuente, que puede ser compilado y ligado con las apropiadas librerías para generar un código ejecutable.

Usando la herramienta de ORACLE que es el precompilador, se pueden desarrollar aplicaciones que combinen la potencia de un lenguaje de alto nivel, con la facilidad de un lenguaje de base de datos, en este caso SQL, que accesa y manipula datos de ORACLE.

El precomilador acepta el programa fuente como entrada, traduce los argumentos de SQL y genera y modifica el programa fuente, el cual puede ser compilado ligado y ejecutado, como se muestra en la figura I.1

Si se quiere trabajar con datos de una Base de Datos Relacional ORACLE, es necesario usar un lenguaje para manipular esos datos, este lenguaje es el SQL, este fue seleccionado porque es flexible, poderoso y fácil de entender. El lenguaje SQL es un lenguaje no estructurado, se puede especificar que es lo que se quiere sin especificar como hacerlo. La mayoría de los comandos son muy parecidos al inglés y es fácil de manipular un dato o muchos a la vez.

Para poder realizar una aplicación basada en precompiladores es necesario:

- Conocer conceptos y terminología sobre la Base de Datos Relacional ORACLE.
- Tener conocimiento sobre Lenguaje Estructurado de Consulta SQL.
- Tener conocimiento de un lenguaje de programación de alto nivel como C, FORTRAN, Pascal, etc.

A lo largo de este capítulo se exponen algunos concepto relacionados a ORACLE y algunos aspectos de su funcionamiento que ayudaran a comprender como trabajan los precompiladores y algunas tareas que realizan.

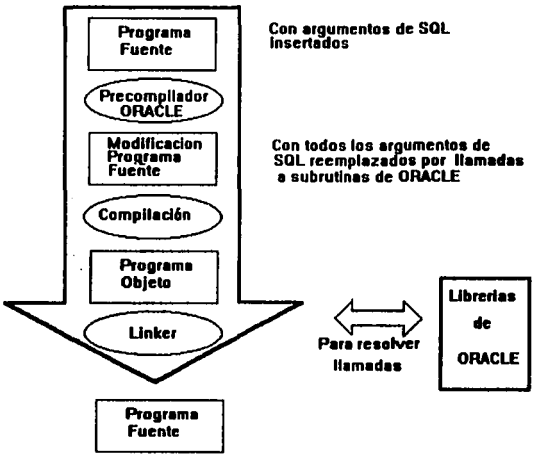

Figura I.1

I.2 Introducción al ORACLE

Con el paso del tiempo, el sistema manejador de base de datos relacional se ha convertido en el camino más aceptado para manejar datos. Los sistema relacionales ofrecen los siguientes beneficios:

- fácil acceso a todos los datos
- flexibilidad en el modelado de los datos
- reducido almecenamiento de datos y redundancia
- independencia de almacenamiento físico y diseño lógico de datos
- alto nivel en la manipulación de datos

ORACLE fue la primer compañía en ofrecer un Manejador de Base de Datos (DBMS) comercialmente, y apartir de entonces ha realizando continuamente innovaciones en el campo de los Manejadores de Bases de Datos (RDBMS). ORACLE ofrece un RDBMS que es portable, compatible y conectable que lo convierte en una poderosa herramienta para los usuarios.

Una base de datos es una colección de datos que puede ser tratada como una unidad. Una base de datos consiste de archivos de sistema operativo. Físicamente, son archivos de la base de datos y archivos llamados de redo log. En forma lógica, los archivos de base de datos contienen el diccionario de tablas y los usuarios de estas, y los archivos de redo log contienen los datos recuperados. Una base de datos requiere una o más copias de los archivos de control, estos archivos de control contienen la información que identifica y describe el resto de la base de datos. La figura I.2 muestra las partes que se requieren en una base de datos ORACLE.

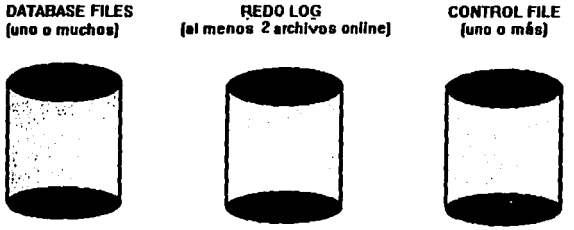

Figura I.2

Un sistema de base de datos ORACLE puede ser configurado para proveer un alto rango de servicios. En todas las configuraciones , un usuario puede accesar la base de datos (la base de datos y los archivos de redo log). La figura I.3 muestra un sistema de base de datos diseñado para ser accesado por muchos usuarios concurrentemente.

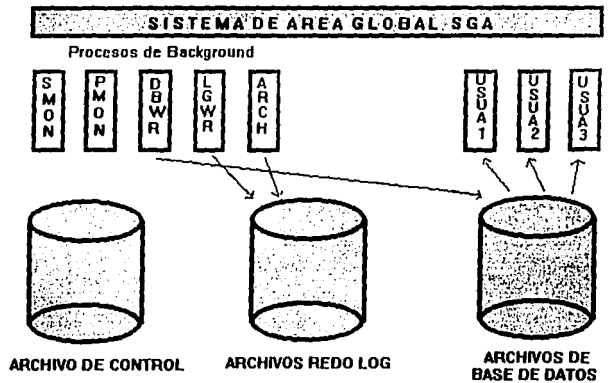

Figura I.3

Una instancia de ORACLE son los mecanismos para accesar y controlar la base de datos. Una instancia puede ser inicializada independientemente de una base de datos (esto es que puede inicializarse sin tener montada o abierta una base de datos). Una instancia puede abrir más de una base de datos.

Una instancia consiste de :

- una área de memoria compartida llamada Area Global del Sistema (System Global Area SGA) que permite la comunicación entre los procesos.
- cinco procesos de background (procesos DBWR,LGWR,SMON,PMON y ARCH) que son compartidos por todos los usuarios.

Todas las instancias necesitan accesar el código del RDBMS de ORACLE. Este código puede ser comparido por multiples instancias. Todas las instancias que se encuentren en un mismo CPU deben tener distintos nombres. Los nombres de las instancias se especifican cuando el software de ORACLE es instalado. Una base de datos puede ser accesada por una instancia, o con un mismo sistema operativo, por multiples instancias simultáneamente.

I.3 Sistema Manejador de Base de Datos RelacionalORACLE (RDBMS)

Un Manejador de Base de Datos Realacional RDBMS es un programa de software que : almacena, recupera y modifica datos; mantiene la consistencia de los datos; resuelve problemas de concurrencia; regula el acceso a los datos.

Un RDBMS tiene las siguientes características:

- Representación de datos en forma de tablas
- Utiliza el lenguaje de cuarta generación SQL
- Capacidades relacionales completas, establece automáticamente todas las relaciones
- Maneja todos los operadores relacionales
- Flexibilidad, fácil modificación de datos y facilidad al cambiar la estructura de los datos
- Diccionario de datos integrado

El Sistema Manejador de Base de Datos Relacional (Relational Database Management System RDBMS) es la parte más importante de ORACLE. Incluye el manejador de la base de datos y varias herramientas para proporcionar ayuda a los usuarios y al administrador de la base de datos en el mantenimiento, monitoreo y uso de los datos .

El núcleo del RDBMS es el kernel. El kernel realiza las siguientes tareas:

- maneja el almacenamiento y definición de los datos
- controla y limita el acceso a los datos
- permite respaldar y recuperar los datos
- interpreta los argumentos de SQL.

Una parte del kernel es el optimizador. El optimizador examina rutas de acceso alternativas a los datos para buscar la ruta optima que resolvera la consulta.

I.4 Estructuras de archivos.

Generalmente solo al administrador de la base de datos (DBA) le concierne lo referente a los archivos físicos; los usuarios de la base de datos raramente necesitan conocer acerca de los archivos físicos. Los archivos físicos son los archivos del sistema operativo que permiten operar el sistema de base de datos de ORACLE.

El mínimo número de grupos de archivos que se requieren para una base de datos de ORACLE se muestran en la figura I.4

TIPO DE ARCHIVO	NUMERO MINIMO	TAMAÑO MINIMO
Archivo de Base de Datos (Database File)	1	500 Kbytes
Online Redo Logs	2	50 Kbytes
Archivo de Control (Control File)	1	Determinado en la instalación
Archivo de programas del RDBMS de ORACLE	muchos	Depende del sistema operativo (estimado en 17400 Kbytes

Figura I.4

Las estructuras físicas son como los archivos del sistema operativo, son almacenados en un medio de almacenamiento tangible, como cinta magnética, unidad de disco o disco flexible. A cada archivo le

corresponde una localidad de espacio asignado por el sistema operativo. El DBA es el encargado de las unidades de espacio físicas. Por ejemplo, es el encargado de asegurarse que los archivos puedan almacenar todos los datos de la base de datos, y añadir otro archivo cuando es necesario.

Las estructuras lógicas también corresponden a unidades de espacio, pero son independientes de las localidades de espacio físico. Una tabla lógica, por ejemplo puede ocupar varios archivos . Un ejemplo de estructuras lógicas en ORACLE son las tablas y los tablespaces.

Las estructuras lógicas están mas relacionadas con los datos,por ejemplo la rapidez con la cual un dato es regresado en una consulta a una tabla puede estar en función de en cuál tablespace se encuentra la tabla y cuantos índices han sido creados de la tabla.

Archivos de Base de Datos (Database Files)

Una base de datos de ORACLE consiste en uno o más archivos de base de datos. Estos archivos contienen todos los datos de la base de datos. Algunas características de los archivos de base de datos son:

- El archivo puede ser asociado con una o más bases de datos.
- Uno o más archivos físicos forman la unidad lógica de almacenamiento de la base de datos llamada tablespace.
- Todos los archivos de base de datos deben estar accesibles cuando una instancia está activa.
- El funcionamiento de la base de datos es más óptimo si cada archivo de base de datos está localizado de forma contigua en el espacio del disco; sin embargo, los archivos de base de datos no necesitan estar contiguos.
- Una vez creados los archivos de base de datos no deben cambiar de tamaño.

El primer archivo de base de datos creado en el tablespace se llamada SYSTEM. Multiples archivos de base de datos pueden añadirse al tablespace.
El archivo de base de datos llamado SYSTEM debe ser de al menos 500 Kbytes para contener:

- el diccionario de datos inicial
- el segmento de rollback inicial

El RDBMS maneja los archivos de base de datos en unidades conocidas como blocks. Los blocks de la base de datos típicamente estan enytre 2 Kbytes o 4 Kbytes. A menos que la mayoría de los registros sean muy largos o muy cortos , el tamaño del block no debe cambiarse.

Todos los archivos de base de datos en un tablaspace pueden ser borrados de la base de datos usando el argumento de SQL DROP TABLESPACE. Después de borrar el tablespace se pueden borrar los archivos de base de datos usando los camandos del sistema operativo.

Archivos de Control (Control Files)

Siempre que una base de datos esta abierta, uno o más archivos de control estan trabajando. Un archivo de control es un pequeño archivo en binario llamado ORACLE.DCF. Un archivo de control es asociado a solo una base de datos.

Los archivos de control son creados durante la creación de la base de datos. Los archivo de control deben estar siempre accesibles, este es requerido cuando una instancia empieza a trabajar y accesa a la base de datos.

Los archivos de control contienen información acerca de la base de datos que es requerida por la base de datos para ser accesada por una instancia. Los archivos de control contienen la siguiente información:

- el nombre de la base de datos
- el nombre de los archivo redo log
- fecha de creación de la base de datos

El nombre de la base de datos aparece en el parámetro DB_NAME del archivo llamado INIT.ORA o es el que se da cuando se usa en el comando de creación de la base de datos CREATE DATABASE. El archivo INIT.ORA es el archivo donde se guardan todos los parámetros de la base de datos y es leido cada vez que la base de datos es inicializada.

Los archivos de control son automáticamente modificados por ORACLE; no pueden ser editados porque son actualizados continuamente cuando se esta usando la base de datos, estos deben estar disponibles en cualquier momento siempre que la base de datos este abierta.

Se recomienda que se tengan multiples copias de los archivos de control. Se recomienda también tenerlos en discos separados, esto reduce el riesgo de que cuando ocurra una falla se pierdan todos los archivos de control. La única desventaja que se tiene cuando existen varios archivos de control es que cuando se realizan ciertas operaciones que actualizan los archivos de control, estas actualizaciones deben hacerse en todas las copias afectando el óptimo rendimiento de la base de datos. Lo más recomendable es que se tengan dos copias de los archivos de control en dos diferentes discos.

Los nombres de los archivos de control se encuentran en el parámetro CONTROL_FILES del archivo INIT.ORA. Este parámetro da una lista uno o más de los archivos de control, por default la lista contiene solo un nombre. Se puede cambiar el parámetro solo cuando la base de datos está inactiva. Esto se hace para añadir mas archivos de control, para cambiarles los nombres o para cambiar su localización.
En el proceso de inicializar una instancia ésta reconoce y da mantenimiento a los archivos de control, pero no puede crear uno nuevo. Para añadir un archivo de control, se debe copiar el archivo de control en un nuevo archivo y añadir los nombres de los archivos de control en la lista del parámetro CONTROL_FILE. Dar de baja la base de datos (shutdown) e inicializarla (startup) para que reconozca los nuevos archivos de control.

Por los regular los archivos de control son pequeños. Lo que determina su tamaño son dos parámetro del INIT.ORA el LOG_FILES y el DB_FILES.

Archivos Redo Log

Los archivos redo log son una serie de archivos del sistema operativo, externos a la base de datos y que registran los cambios en la base de datos durante sus transacciones. Existen dos tipos de redo logs: los archivos redo log en línea (online) y los archivos redo log fuera de línea (offline)

Los archivos redo log online son un conjuntos de archivos en los cuales son escritas las transacciones cada vez que una transacción de la base de datos es aceptada (commit). Los archivos redo log son usados en las operaciones de recuperación para restablecer archivos de la base de datos perdidos. Si la base de datos no puede escribir en los archivos redo log online, el RDBMS manda un mensaje de error.

Los archivos redo log offline son copias de los archivos redo log online que han sido salvados o archivados en disco o cinta. Ambos los archivos online y offline son utilizados en la recuperación de archivos de la base de datos en caso de falla. El uso de los archivos redo log offline es opcional.

Es recomendable que cada base de datos tenga al menos dos archivos redo log online.

Los archivos red log online son una parte escencial del RDBMS del ORACLE porque son los únicos archivos en los cuales las transacciones son escritas al ser aceptadas (commit). Es decir, una transacción es aceptada completamente hasta que la transacción es escrita en el archivo redo log online.

Al guardar todas las transacciones aceptadas de la base de datos los archivos redo log online son muy útiles cuando una instancia falla y se quiere restablecer.

El archivo redo log siempre contiene al menos dos archivos de sistema operativo llamados online redo log. Los archivos redo log online son creados al mismo tiempo que la base de datos es creada. Cuando se usa el comando CREATE DATABASE, se pueden especificar:

- el número de de archivos redo logs online que se quieran usar
- el tamaño de los archivos
- el lugar donde estaran los archivos.

En la sintáxis de la creación de la base de datos se especifica el lugar, el nombre y el tamaño de los archivos redo log como se muestra en la figura Y.5

```
CREATE DATABASE nombre ...
LOGFILE 'path/ONELOG.RDO' SIZE 50000, 'path/TWOLOG.RDO'
    SIZE 50000;
```

Figura I.5

Donde "nombre" es el nombre de la base de datos, "path" es la ruta donde se econtrara el archivo redo log, ONELOG y TWOLOG son los nombres del primer archivo redo log y del segundo respectivamente y SIZE especifica el tamaño del archivo en bytes.

Hay dos factores que limitan el número de archivos redo log que se pueden añadir a la base de datos:

- MAXLOGFILES, es un argumento opcional que se puede usar cuando se crea la base de datos con CREATE DATABASE

- el parámetro LOG_FILES del archivo INIT.ORA.

El parámetro MAXLOGFILES especifica el máximo número aboluto de archivos redo log, este máximo está sujeto al límite de números de archivos que tiene ORACLE y que es de 256.

El actual número máximo de archivos redo log es establecido en el parámetro LOG_FILES del archivo INIT.ORA. Este límite tiene efecto siempre que una instancia esta abierta hasta que es cerrada, el default es 16. El parámetro LOG_FILES puede reducir temporalmente el límite establescido en MAXLOGFILES pero nunca aumentarlo.

El uso de LOG_FILES y MAXLOGFILES es opcional. Si ninguno de los dos es usado ORACLE toma el límite de 256 archivos. El mínimo de archivos redo log es dos.

Los archivos redo log no necesariamente tienen que ser del mismo tamaño. Sin embargo, para garantizar que los puntos de chequeo ocurran a intervalos regulares y el funcionamiento del sistema sea el óptimo, todos los archivos redo log online deben ser del mismo tamaño. El tamaño de los archivos redo log esta dado en bytes. El tamaño mínimo de un archivo redo log es de 50 kilobytes.

Los archivos redo log en una base de datos deben estar en diferentes discos por dos razones:

- para reducir el riesgo de perder toda la base de datos y los archivos redo log en caso de una falla
- para reducir la contención del disco (durante la alta actividad de la base de datos los archivos redo log son accesados frecuentemente).

Cada vez que una transacción es aceptada, el Log Writer (LGWR) escribe la información acerca de las transacciones, de los buffers redo log del SGA para activar los archivos redo log online. Todos los cambios de la base de datos están almacenados en un archivo redo log online en forma de entradas de redo.

Una entrada de redo contiene información que puede ser usada para reconstruir todos los cambios hechos en la base de datos, estos cambios son los de los bloque de datos y de los bloque de rollback.

El RDBMS usa los archivos redo log online en forma circular; cuando un archivo se llena el LGWR comienza a escribir en el siguiente archivo redo log. Cuando el último archivo redo log disponible se llena, el RDBMS regresa al primer archivo redo log y comienza a escribir en el, y el ciclo se repite nuevamente. El punto en el que el RDBMS termina de escribir en un archivo redo log y comienza a escribir en el siguiente es llamado log switch. El archivio redo log online que está siendo accesado por el LGWR es llamado archivo activo redo log.

La escritura circular de los archivos redo log se muestra en la figura I.6

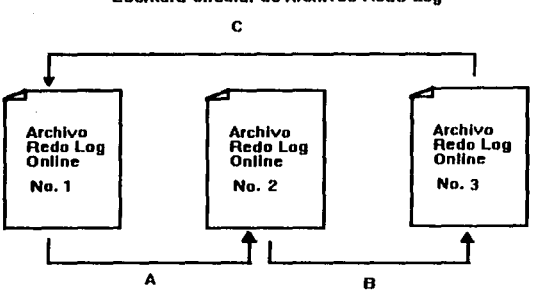

Figura I.6

Cuando el archvio Redo Log Online No. 1 se llena, ocurre un log switch (A) y el LGWR comienza a escribir en el Redo Log Online No. 2. Cuando el archivo Redo Log Online No.2 se llena, otro log switch ocurre (B) y el LGWR comienza a escribir en el archvio Redo Log Online No.3 , cuando este es llenado, ocurre el tercer log switch (C) y el LGWR comienza a sobreescribir el archivo Redo Log Online No.1

Segmentos de Rollback

Un segmento de rollback puede almacenar un número límite de transacciones. Este número está en función de la información almacenada de las transacciones activas en los segmentos de rollback. Cuando una transacción comienza se empieza a escribir la información en los segmentos de rollback. Esta información no es borrada de los segmentos de rollback hasta que se acepta la transacción o se da la orden de rollback.

Cuando se crea una base de datos usando el comando CREATE DATABASE, un segmento de rollback inicial es creado usando los parámetros por default. Un solo segmento es apropiado para sistemas pequeños . por ejemplo, bases de datos que tienen un solo tablespace. Sin embargo, para la mayoría de los sistemas de base de datos, es deseable que se añadan mas segmentos de rollback para dar servicio a todos los usuarios, para soportar la actividad de las transacciones, o para permitir múltiples tablespaces.

CREATE ROLLBACK SEGMENT;

Una vez que un segmento de rollback es creado la base de datos se debe desactivar y restaurar para que el segmento de rollback pueda ser usado.

Para determinar los segmentos de rollback que tendrá la base de datos el DBA debe considerar:

- cuantos segmentos de rollback usara
- cuanto espacio es reservado para cada segmento
- en que tablespace deberá residir el segmento de rollbacK
- cuanto espacio es o será dedicado a los segmentos de rollback
- cual es el porcentaje y el máximo numero de transacciones concurrentes
- cuales son los requierimientos de optimización

Al incrementar el número de segmentos de rollback se reduce la contención para los bloques de segmentos de rollback, es decir entre mas segmentos de rollback se tengan se puede almacenar mas información de las transacciones que no han sido aceptadas (no han dado commit).

El espacio reservado para los segmentos de rollback es fijo. El número esperado del total de transacciones concurrentes está en función de los usuarios de la base de datos y del volúmen de actividad. Así el número total de segmentos de rollback debe ser el necesario para dar soporte al máximo número de transacciones simultáneas. El número de segmentos de rollback se puede calcular mediante la siguiente expresión:

#segmentos= 2x[max # transacciones / (tamaño del bloque) /32]

Esta expresión es el resultado de multiples pruebas y es recomendable para el cálculo de segmentos de rollback en una base de datos.

Los segmentos de rollback son agrupados en bloques llamados extents. Todos los segmentos de rollback deben ser del mismo tamaño y se debe procurar tener el número de segmentos de rollback necesarios para contener la información de las transacciones. En el caso de que el espacio disponible no alcance a guardar toda la información de las transacciones la transacción terminara con error.

I.6 Estructuras de Memoria

Las operaciones del RDBMS de ORACLE dependen de las estructuras de memoria y de los procesos. Todas las estructuras de memoria residen en la memoria principal del sistema en el que se encuentra la base de datos. Los procesos son trabajos o tareas que realizan las operaciones en memoria. Estas operaciones incluyen los procesos de los usuarios y los cinco procesos de background: el Escritor de Archivos de Base de Datos (Database Writer DBWR), el Escritor de Archivos Redo Log (Log Writer LGWR), el Monitor de Sistema (System Monitor SMON), el Monitor de Procesos (Process Monitor PMON), y el Archivador (Archiver).

Las estructuras de memoria, los procesos de usuario y los cinco procesos de background se muestran en la figura I.7.

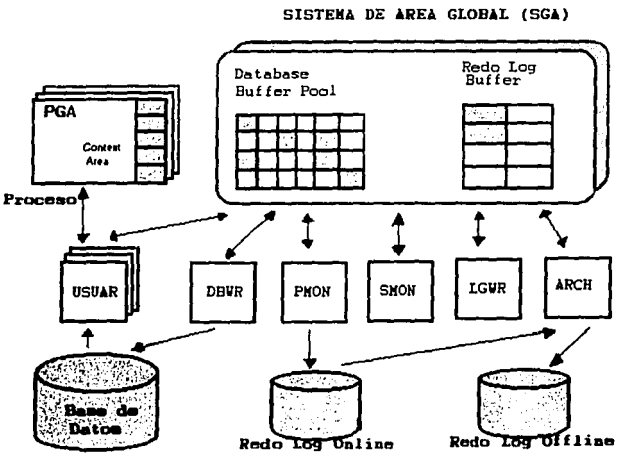

Figura 1.7

Las estructuras de memoria tienen los siguientes propósitos:

- almacenar el código de programas ejecutables
- almacenar los datos necearios durante la ejecución de los programas (por ejemplo, los renglones regresados en una consulta)
- almacenar información que es transferida entre procesos y memoria periférica (por ejemplo, bloques de la base de datos que se transfieren del disco alprograma)
- almacenar información que es compartida y comunicada entre los procesos de ORACLE

Existen varias estructuras básicas de memoria asociadas con el RDBMS de ORACLE. Estas incluyen el Sistema de Area Global (System Global Area SGA), los Buffers de la Base de Datos y los Buffers de Redo Log, el Area Global de Programas (Program Global Areas PGA) y el Area de Contexto (Context Area).

Sistema de Area Global (SGA)

El Sistema de Area Global (System Global Area SGA) es un conjunto de buffers compartidos que contienen datos y control de información para una base de datos ORACLE. El SGA es escrito solo por el RDBMS, y no por los usuarios. El SGA contiene:

- buffers de la base de datos
- buffers de redo log
- el diccionario de la base de datos

El RDBMS de ORACLE reserva un SGA cada vez que una instancia es inicializada, y termina de usarla cuando la instancia termina. Es decir, cada vez que una instancia es inicializada cuenta con su propio SGA. Los datos del SGA son compartidos por todos los usuarios que esten conectados en ese momento a la base de datos.

El tamaño del SGA es determinado cuando la instancia se inicializa; una vez que el SGA es reservado, su tamaño no cambia. El comando SHOW SGA nos muestra el tamaño del SGA. El tamaño del SGA depende de los parámetros establecidos en el archivo INIT.ORA.

Buffers de la Base de Datos y Buffers Redo Log

El SGA contiene dos áreas de buffres: un buffer de base de datos de reserva y un buffer de redo log. El buffer de base de datos de reserva contiene a su vez dos tipos de bloques: los bloques de segmentos de datos y los bloque de segmentos de rollback.

Durante el curso de una transacción, los cambios en los datos no son escritos inmediatamente en los archivos de la base de datos (database files). Antes de esto hay tres acciones que se realizan:

1. Cada argumento ejecutado en la transacción modifica un bloque de segmento de datos en el buffer de base de datos de reserva. Cada bloque de datos que reside en los buffers de reserva corresponde al bloque de datos de los archivos de la base de datos.

2. Mientras tanto, la información que puede ser usada para deshacer una transacción es almacenada en un bloque de rollback dentro del buffer de base de datos de reserva. Así, ésta información puede ser utilizada despues de producirse una lectura de otro usuario a esos datos (lo que se conoce como consistencia de datos), o esos datos pueden usarse en caso de que la transacción necesite un rollback.

3. Un registro de cada cambio hecho al bloque de datos y a los bloques de rollback durante los pasos 1 y 2 es guardado en un buffer redo log. Cada vez que una transacción es aceptada, la información de los buffers de redo log es escrita en los archivos redo log.

Como la cantidad de espacio en los buffers de reserva es limitado, los buffers deben volver a usarse, por eso, su contenido es escrito en el disco por el DBWR. Cuando no hay espacio en los buffers de la base de datos, los buffers que han sido menos recientemente usados son escritos en los archivos de la base de datos tanto si han sido aceptados o no. Los bloque mas recientemente usados no son escritos en la base de datos hasta que son aceptados.

segmento de rollback durante las transacciones. Cada vez que una transacción es aceptada la información de los buffers redo log es escrita en los archivos de redo log.

Area Global de Programa (PGA)

El Area Global de Programa (Program Global Area PGA) es un buffer de memoria que contiene datos e información de control para un solo proceso de usuario.

Cada proceso de usuario tiene un PGA . Usualmente, el RDBMS de ORACLE crea un PGA cuando un usario se conecta a ORACLE. El PGA contiene información acerca de la conexión del usuaro con la base de datos ORACLE, y mantiene la información para que el usuario pueda comunicarse con la base de datos.

El PGA es un área de memoria no compartida. Un PGA es reservada para cada usuario que se conecta a ORACLE. El PGA es exclusivo para el proceso de usuario y es leido y escrito solo por el código de ORACLE en nombre del usuario.

Usalmente, el PGA es reservado al tiempo de conexión; si no hay suficiente memoria al tiempo que se conecta el usuario ocurre un error, al ocurrir esto el usuario no se conecta a la base de datos hasta que haya memoria disponible. El tamaño del PGA varía de acuerdo al sistema operativo.

Area de Contexto (Context Area)

Todos los argumentos de SQL usados en un proceso de un usuario requiere de un área de contexto. Un área de contexto es un buffer de memoria creado para retener y procesar información acerca de la ejecución de un argumento de SQL.

El RDBMS de ORACLE automáticamente abre un área de contexto cada vez que un argumento de SQL requiere de uno, y lo cierra cuando el argumento es ejecutado. Además, los programas pueden explicitamente abrir y cerrar áeas de contexto.

Cada proceso de usuario tienen un grupo de áreas de contexto, estas son asociadas con el PGA. Las áreas de contexto contienen toda la estructura de los datos necesaria para ejecutar un argumento de SQL, incluyendo:

- el texto del argumento de SQL
- una traducción del argumento de SQL
- un registro del resultado y algunos valores inmediatos
- un informe del estado de la información usada para ejecutar el argumento
- control de información usada para el ordenamiento de la información

Un cursor es un puntero a un área de contexto, o el nombre para un área de contexto especifica. Las interfases de programación como los precompiladores ofrecen un mayor control de los cursores. En el desarrolo de programas en precompiladores, un cursor es un recurso disponible para programar y puede ser especificamente usado para el análisis gramatical de argumentos de SQL en una aplicación.

Las áreas de contexto comienzan tomando un tamaño y crecen dinámicamente si se necesita más espacio. La cantidad de espacio inicialmente reservado para el área de contexto se establece con el parámetro CONTEXT_INCR del archivo INIT.ORA.

Una vez que el área de contexto es reservada, esta puede ser usada al hacer el análisis gramatical de subsecuentes argumentos de SQL en un particular cursor. El cursor no se libera hasta que es cerrado.

I.6 Estructura de Procesos

Un proceso es un mecanismo en el sistema operativo que puede ejecutar una serie de pasos. Algunos sistemas operativos usan el termino tarea o trabajo.

La estructura de procesos de un sistema, por ejemplo, la del RBDMS de ORACLE es importante porque éste define como pueden ocurrir múltiples actividades y como son llevadas a cabo.

Proceso de Usuario.

Los procesos de usuarios existen en todos los sistemas de bases de datos. Un proceso de usuario es invocado al momento de correr un programa de aplicación, como un programa en Pro*C, o una herramienta como SQL*DBA. Un proceso de usuario es creado normalmente cunado un usuario se conecta y comienza a trabajar.

Los procesos de usuario se comunican con la base de datos através del programa de interfase. El programa de interfase es una interfase entre los programas del usuario y los programas de ORACLE. El programa de interfase realiza las siguientes tareas:

- provee una barrera de seguridad, previeniendo accesos destructivos de los usuarios a la base de datos
- actua como un mecanismo de comunicación, para formatear requisiciones de información, pasar datos, detectar y regresar mensajes de errores.
- realiza conversión y transformación de datos, particularmente entre diferentes tipos de computadoras o para tipos de datos de programas de usuarios externos.

Los programas de ORACLE realizan tareas en nombre de los programas del usuario, por ejemplo, extraen registros de un bloque de datos.

Procesos de una Tarea.

En los sistemas de una tarea, el programa del usuario, el programa del kernel de ORACLE y el programa de interfase corren en el mismo proceso. En el sistema de una tarea, el programa de interfase es el responsable de la separación y protección del código de ORACLE, y es el responsable del paso de datos entre el usuario y ORACLE.

El sistema de una tarea de ORACLE se utiliza en sistemas operativos que puedan soportar una separación entre los programas de usuario y los programas de ORACLE en un solo proceso. Esta separación es requerida para asegurar la integridad de los datos.
Algunos sistemas operativos no pueden preveer esta separación y tienen que correr en procesos de dos tareas para prevenir datos a ORACLE por programas de usuarios.

Proceso de dos Tareas

En los procesos de dos tareas ORACLE ubica los procesos del usuario y los procesos de ORACLE en procesos separados. En los sistemas de una tarea, el programa de interfase realiza la comunicación entre los dos programas, el de usuario y el de ORACLE. Sin embargo, en los sistemas de dos tareas se usan los procesos del sistema operativo en el que corre la base de datos para realizar la comunicación entre los programas de usuario y los programas de ORACLE.

El método de separación de procesos para correr los programas de usuario es llamado proceso "shadow". En un sistema de dos tareas, todos los procesos conectados a ORACLE tienen un proceso "shadow".

El sistema de dos tareas también se usa cuando los programas del usuario corre en una computadora local y los programas de ORACLE y de la base de datos estan en una computadora remota.

Un RDBMS de ORACLE puede trabajar en proceso simple o en multiproceso. Si en el archivo INIT.ORA, el parámetro SINGLE_PROCESS esta en estado verdadero (TRUE), el sistema trabaja como proceso simple; si es falso (FALSE) trabaja en multiproceso.

En la mayoría de los sistemas, una instancia ORACLE puede inicializarse como proceso simple o como multiproceso independientemente de como fue instalada o de como fue inicializada la última vez. Sin embargo en los sistemas que no soportan múltiples procesos o memoria compartida, ORACLE puede correr solo como proceso simple (por ejemplo, ORACLE instalado en MS-DOS basado en computadoras personales).

Proceso Simple.

Un proceso simple de ORACLE (llamado single user) es un sistema de base de datos que puede ser accesado por solo un usuario; múltiples usuarios no pueden accesar la base de datos concurrentemente.

ORACLE corre en computadoras personales como proceso simple, porque el sistema operativo MS-DOS no es capaz de correr como multiusuario. Los procesos simples implican instancias simples; la base de datos no puede estar abierta para otras instancias.

Los procesos simples no pueden usar los cinco procesos de background que son usadas en el multiproceso. Los procesos simples son menos óptimos y menos poderosos que el multiproceso.

Multiproceso.

Un sistema ORACLE de multiproceso (llamado multi user) puede ser usado por más de un usuario o aplicación. La mayoría de las bases de datos trabajan como multiproceso, porque uno de los principales beneficios de las bases de datos es el manejo de los datos por múltiples usuarios al mismo tiempo.

Un sistema multiproceso requiere de los cinco procesos de background para cada usuario de la base de datos.

Procesos de Background

Para atender a todos los usuarios, el sistema ORACLE de multiproceso utilizan algunos procesos adicionales llamados procesos de background. Su proceso es mejorar el funcionamiento de la base de datos por la consolidación de algunas funciones que deben ser tomadas por los múltiples programas de ORACLE que corren para cada programa de usuario.

Los procesos de background asíncronamente realizan lecturas y escrituras a la base de datos, y realizan el proceso de monitoreo para proveer un gran paralelismo para mejorar el funcionamiento y seguridad de la base de datos.

Cada instancia de ORACLE puede usar los cinco procesos de background. El nombre de estos procesos son : el Escritor de Archivos de Base de Datos (Database Writer DBWR), el Escritor de Archivos Redo Log (Log Writer LGWR), el Monitor de Sistema (System Monitor SMON), el Monitor de Procesos (Process Monitor PMON), y el Archivador (Archiver). Si múltiples instancias comparten la base de datos, cada instancia tiene su propio grupo de procesos de background.

En muchos sistemas operativos los procesos de background son creados automáticamente cuando la instancia es inicializada. En otros sistemas operativos los procesos son creados como parte de la instalació de ORACLE.

Cuando es deseable que muchas instancias de ORACLE esten corriendo en una misma computadora, ORACLE cuenta con un mecanismo para nombrar las instancias. Los nombres de los procesos de background son prefijados con un identificador de instancia para distingir el grupo de procesos para cada instancia. Por ejemplo para una instancia llamada INSUNO existen los procesos llamados:

- ORA_INSUNO_DBWR
- ORA_INSUNO_LGWR
- ORA_INSUNO_SMON
- ORA_INSUNO_PMON
- ORA_INSUNO_ARCH

Escritor de Base de Datos (DBWR)

El propósito del DBWR es escribir los bloques modificados de los buffers de la base de datos a la base de datos. Los bloques son escritos en el orden correcto pasra mantener la integridad de la base de datos. El proceso para escribir asegura que siempre existan buffers disponibles para escribir, si un buffer se necesita, los buffers menos recientemente usados son escritos en la base de datos.

El DBWR no necesita escribir los bloque cuando una transacción es aceptada, la integridad de los datos esta garantizada por la escritura de los bloques aceptados en los archivos redo log online, en vez de que se escriban los datos a la base de datos cada vez que una transacción es aceptada.

Escritor de Archivos Redo Log (LGWR)

El proceso LGWR escribe las entradas a los archivos redo log en el disco. La información en los redo log es generada en los buffers del SGA. Cuando una transacción es aceptada, el LGWR escribe la información de los redo log en los archivos redo log online (algunas veces, si se necesitan mas buffers, el LGWR escribe la información de los redo log aunque la transacción no haya sido aceptada; esta información se hace permanete solo si la información es aceptada después). El LGWR asegura el fácil

flujo de los datos del buffer al disco sin retrasar los procesos que generan la nueva información que esta entrando a los redo log.

Monitor del Sistema (SMON)

El propósito del SMON es realizar la recuperación de la instancia. Es usado cuando se inicializa la instancia y realiza las apropiadas recuperaciones en caso de una falla de CPU. El SMON también es responsable de limpiar los segmentos temporales que no estan en uso y para la recuperación de los pasos de la transacción perdida durante la falla de la instancia.

El SMON se activa regularmente para checar la base de datos, y puede ser llamada por otro proceso detecta la necesidad de utilizar el SMON.

Monitor de Procesos (PMON)

El propósio del PMON es el de realizar el proceso de recuperación cuando un proceso de usuario falla. El PMON es responsable de limpiar y liberar los recursos del proceso que fueron usados. Por ejemplo, inicializa a condiciones iniciales las tablas de la transacción activa, libera candados, borra los identificadores de proceso de la lista de procesos activos de ORACLE.

Igual que el SMON, el PMON se activa regularmente para realizar el chequeo y puede ser llamado por otro proceso que detecte la necesidad de usarlo.

Archivador (ARCH)

El proceso ARCH copia el archivo redo log online a disco cuando esta lleno. El ARCH se activa solo cuando los redo log son usados.

CAPITULO II

Argumentos de SQL en un Precompilador

II.1 Introducción al SQL

El SQL es un lenguaje para accesar a la información almacenada en Bases de Datos Relacionales. La palabra SQL está formada por las siglas de Structured Query Language, es decir, Lenguaje Estructurado de Consulta.

El concepto de Base de Datos Relacional surgió de la necesidad de sustentar los conceptos de un modelo relacional de datos y de un sublenguaje para acceder a ellos basados en el cálculo de predicados. Estas ideas fueron desarrolladas posteriormente en los laboratorios de IBM dando lugar a un lenguaje llamado SEQUEL y a un prototipo de sistema relacional llamado system R. Al evolucionar SEQUEL su nombre fue cambiado por SQL. A partir de entonces aparecen múltiples productos de bases de datos basados en SQL, esta tendencia hacia su uso generalizado en la industria se vio consolidado al ser adoptado por el Instituto Americano de Normas (ANSI, American National Standards Institute), que ha desarrollado y publicado unas especificaciones para este lenguaje. Después fueron aceptadas por la ISO (International Standars Organization).

El SQL es un lenguaje no estructurado porque:
- procesa grupos de registro en vez de uno solo a la vez
- provee navegación automática de los datos

SQL permite trabajar con datos estructurados . En vez de manipular un solo registro, se pueden manejar grupos de registros, la forma más común de grupos de registros son las tablas. Todos los argumentos de SQL aceptan grupos de datos de entrada y regresan grupos de datos de salida. Las propiedades de SQL permiten que el resultado de un argumento de SQL sea usado como la entrada de otro argumento de SQL.

SQL no requiere que se especifique el modo de acceso a los datos. Todos los argumentos de SQL utilizan el optimizador de consulta, que es una porción del RDBMS, que determina la forma más rápida de accesar los datos. El optimizador de consulta conoce que índices existen y usa estos índices de la manera más apropiada cuando accesa una tabla, para el usuario no es necesario conocer si una tabla tiene índices o que tipo de índices tiene esa tabla.

SQL provee argumentos para una variedad de tareas que incluyen:
- consultas de datos
- insertar, modificar, y borrar renglones en una tabla
- crear, modificar, y borrar objetos de la base de datos
- controlar el acceso a la base de datos y a los objetos de la base de datos
- garantizar la consistencia de la base de datos

En un programa en Pro*C estan presentes instrucciones del lenguaje de alto nivel en el que se este trabajando (ya sea C, COBOL,Pascal,etc) y argumentos de SQL que permiten accesar y manipular los datos de la Base de Datos.

En este capítulo se da una introducción a lo que es el Lenguaje Estructurado de Consulta SQL , los tipos de datos que maneja, las funciones y comandos con los que cuenta, y que en el cuerpo de un programa Pro*C estan presentes para interactuar con las instrucciones de un lenguaje de alto nivel, y que luego será compilado para generar un código ejecutable el cual se corre de la misma forma que un programa en C.

II.2 Conceptos Básicos

El modelo Entidad-Relación categoriza todos los elementos de un sistema como entidades (una persona, un lugar , o una cosa) o una relación entre entidades. Ambas construcciones son representadas por una misma estructura llamada tabla. Los argumentos de SQL actuan sobre los datos de la base de datos y estos se encuentran almacenados en las tablas.

Una tabla es una estructura de datos que contienen los datos en una base de datos relacional. Una tabla está compuesta por columnas y renglones. Esta puede representar una simple entidad, por ejemplo, una tabla puede representar una lista de productos de una empresa. Las tablas pueden representar también una relación entre dos entidades, por ejemplo, puede representar una relación entre empleados y los trabajos que realizan.

Con un buen diseño las tablas pueden representar una entidad y describir la relación entre una entidad y otra. La tabla PROFESOR describe los profesores que laboran en una escuela, en esta tabla se incluye la columna GPO que representa la relación entre los profesores y los grupos donde imparten las clases. La figura II.1 nos muestra la tabla de profesores con sus columnas y renglones.

NUM_PROF	NOM_PROF	MATERIA	GPO
00001	Leonardo Ruiz	Biología	1101
00002	José Pérez	Matemáticas	1103
00010	Irma Jiménez	Matemáticas	1201
00015	Jorge López	Física	1104
00018	Carmen Guerrero	Química	1206

Figura II.1

Las caractristicas de las tablas son:

- El formato columna/renglón de una tabla resulta familiar para visualizar los datos.

para representar la relación entre los datos.
- Las tablas están bien relacionadas, porque la teoría relacional está fundada en el álgebra relacional y cálculos relacionales.

Cada columna en una tabla representa uno y solo un atributo de una entidad. El nombre de una columna debe indicar su significado. Una columna no debe tener un significado confuso, por lo tanto no se debe usar una columna para representar múltiples atributos.

Los renglones almacenan los datos de una tabla. Cada renglón representa una ocurrencia de una entidad o la relación representada por la tabla, no debe haber renglones duplicados en una tabla, ya que eso significa que se tiene información redundante.

Además de las tablas hay otros objetos de la base de datos sobre los que pueden actuar los comandos de SQL. Uno de estos objetos son las vistas las cuales son una consulta a ciertas columnas de una tabla. Una vista contiene solo ciertas columnas de una tabla y en ella no se almacenan registros ya que solo es una consulta a la tabla de la cual se forma. Frecuentemente es usada de la misma forma que una tabla. Una vista es una tabla virtual, tiene las siguientes características:

- Se ve como una tabla, pero no existe como tal
- Sus datos se derivan de las tablas
- No almacenan datos
- Provee simplicidad, para ver exactamente lo que se necesita
- Provee seguridad para prevenir que usuarios no autorizados vean ciertas columnas o renglones de las tablas

Otros objetos de la base de datos son los sinónimos que son imágenes de las tablas. y hacen referencia a ellas. Los sinónimos por sí mismos tampoco almacenan registros. Si una tabla tiene que ser usada por varios usuarios, se les crea a cada uno un sinónimo que es usado como si fuera una tabla.

II.3 Tipos de Datos Manejados en SQL

Los términos literal, constante y valor tienen el mismo significado en ORACLE, ellos representan una parte de la información. Los caracteres y literales de fechas son encerradas entre apóstrofes. Las apóstrofes habilitan a ORACLE para distinguir los nombres de los objetos de la base de datos de los caracteres y literales de fecha.

Las variables son utilizadas para almacenar una literal. Las variables se manejan de la misma manera que cuando se programa en algún lenguaje y en cualquier momento una variable puede estar vacía o puede contener una literal.

Cada variable o literal manipulada por el RDBMS de ORACLE tiene un tipo de dato, los tipos de datos establecen ciertas propiedades de los valores. Estas propiedades causan que ORACLE trate un tipo de dato diferente de otra tipo de dato. Por ejemplo se pueden realizar operaciones aritméticas con valores tipo numéricos y no con valores de tipo caracter.

Los tipos de datos también restringen los rangos de valores que las variables pueden contener.

Un valor en la base de datos asume el tipo de dato basado en el tipo de dato de la columna a la cual pertenece. A cada columna de una tabla se le asigna un tipo de dato y consecuentemente,cada valor perteneciente a una columna asume el tipo de dato al que pertenece la columna. ORACLE asigna automáticamente el tipo de dato a valores que no pertenecen a la base de datos basado en el contexto en el que se este usando la variable. Por ejemplo, si se inserta el siguiente dato '01-ENE-93' dentro de una columna de una tabla, ORACLE toma la cadena de caracteres como un valor tipo DATE después de que ha verificado que es una forma válida de fecha.

Generalmente una expresión no puede mezclar tipos de datos, es decir, no se pueden multiplicar dos variables numéricas y restarle un variable tipo caracter. Sin embargo, ORACLE puede convertir un tipo de dato en otro si es posible, por ejemplo si la variable '50' fue declarada como tipo caracter, ORACLE puede convertirla automáticamente a tipo numérica si esta es usada en una expresión numérica.

Tipo de Dato Caracter (CHAR)

El tipo de dato CHAR es usado para manipular palabras y cadenas de texto. Todos los tipos de textos pueden ser almacenados en variables tipo CHAR, pero solo numeros pueden ser almacenados en variables numéricas (NUMBER).

El dato es almacenado en una variable de caracter con valores de ASCII o EBCDIC, el uso de caracteres no estandares puede reducir la portabilidad de los datos.

Actualmente, los tipos de datos CHAR y VARCHAR son equivalentes. Estos tipos de datos son usados para almacenar datos de tipo CHAR (alfanuméricos). Se pueden usar las palabras CHAR o VARCHAR indistintamente, con el mismo efecto.

El máximo número de caracteres que pueden ser almacenados en una columna definida como CHAR o VARCHAR es 255. La máxima longitud que puede tomar una columna definida como CHAR es especificada en la creación de la tabla.

Algunos ejemplos de variables tipo CHAR son : 'NOMBRE','EDAD', '123', etc.

Tipo de Dato Numérico (NUMBER)

El tipo de dato NUMBER es usado para almacenar números. Números o casi cualquier magnitud puede ser almacenada con una presición arriba de 38 digitos . El rango de los números que puede almacenar es de 1.0×10^{-129} a 9.99×10^{124}.

Cuando se especifica un campo numérico es recomendable especificar el máximo número de digitos y de lugares decimales. El especificar el máximo número de digitos a ser almacenados en una columna no implica que todos los valores tengan una cierta longitud. Si el valor excede la máxima presición regresa un error. Si un valor excede la máxima escala, este valor es redondeado. Un tipo de dato NUMBER puede ser especificado con los parámetros de presición y escala (número de digitos a la derecha del punto decimal), la escala está en el rango de -84 a 127. La figura II.1 muestra los datos que pueden ser almacenados en una variable tipo NUMBER con diferentes factores de escala.

DATO	ESPECIFICADO COMO	ALMACENADO COMO
7,465,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.8	NUMBER(6)	excede presición
7,456,123.8	NUMBER(15,1)	7456123.8
7,456,123.89	NUMBER(7,-2)	7456100

Figura II.1

Si la escala es negativa , el dato es redondeado el número de lugares especificados a la izquierda del punto decimal, si se especifica por ejemplo (10,-2) significa que se redondea a cien.

ORACLE también maneja el concepto de número de punto flotante con el tipo de dato FLOAT. Un valor del tipo de dato FLOAT puede especificar el punto decimal en cualquier lugar desde el primero hasta el último dígito.

Un número puede ser representado en forma exponencial. Un número exponencial es un número seguido por una E, un signo y un dígito de 1 a 28. Por ejemplo el número 347289 se representa como 3.47289E+5, el número 0.0635 se representa como 6.35E-2.

Por compatibilidad con otros sistemas de bases de datos , ORACLE permite la sintáxis de casi cualquier variedad de tipos de datos incluyendo DECIMAL, INTEGER, REAL y DOUBLE_PRECISION.Todas las formas de tipos de datos numéricos estan almacenados en el mismo formato interno de ORACLE.

Tipo de Dato Fecha (DATE)

El tipo de dato DATE es usado para almacenar una fecha e información de tiempo. La fecha y el tiempo se pueden representar en tipos de datos CHAR o NUMBER, ORACLE asocia propiedades especiales a los valores de DATE.

Para cada valor de fecha la siguiente información se puede almacenar:
* Centuria
* Año
* Mes
* Día
* Hora
* Minuto
* Segundo.

El tiempo es almacenado en el formato de 24 horas. Por default, el tiempo en un campo DATE es 12:00:00 am que sería media noche. La fecha por default en un campo DATE es el primer dia del actual mes. La función SYSDATE da la fecha y la hora actual. El formato estandard para la fecha es DD-

MON-YY, como por ejemplo 01-ENE-93. Para insertar valores que no tienen el formato estandard de ORACLE se utiliza la función TO_DATE.

No se pueden multiplicar o dividir fechas, pero se pueden sumar o restar constantes de números a las fechas, como por ejemplo SYSDATE + 1 es mañana, y SYSDATE - 7 es una semana antes. ORACLE maneja varias funciones para operaciones comunes entre fechas, como la función ADD_MONTHS, que permite sumar los meses de una fecha, o la función MONTHS_BETWEEN que da el número de meses entre dos fechas, la porción fraccional representa los 31 días del mes.

Tipo de Dato LONG

Una columna definida como LONG puede almacenar variables de cadenas de caracteres con una longitud de arriba de 65,535 caracteres. Los datos tipo LONG son iguales a los tipo VARCHAR excepto que almacenan valores mas grandes. Los datos tipo LONG pueden ser usados para almacenar arreglos de caracteres.

Los datos de tipo LONG estan sujetos a unas restricciones. Solo una columna tipo LONG es permitida por tabla. No se pueden usar columnas tipo LONG en:

* índices
* clausulas WHERE, GROUP BY,CONNECT BY o DISTINCT
* funciones (como SUBSTR o INSTR)
* expresiones.

Tipo de Dato ROWID

Asociada con todos los renglones de una tabla en una base de datos esta una pseudo-columna que corresponde a la dirección de cada renglón. Esta dirección puede ser obtenida mediante una consulta usando la palabra reservada ROWID. Como se muestra en la figura II.2 , seleccionando el ROWID en una consulta, se obtiene una representación en hexadecimal de la dirección de cada renglón selecionado.

SELECT ROWID,NOMBRE FROM EMP WHERE SEPARTAMENTO = 20;	
ROWID	**NOMBRE**
0000000F.000.0002	MIGUEL
0000000F.003.0002	IRMA
0000000A.007.0002	CARMEN
0000000F.00A.0003	EDUARDO
0000000F.00C.0001	ARMANDO

Figura II.2

El ROWID está conformado por tres partes que son necesarias para localizar el renglón:
- el bloque en el archivo de la base de datos
- el renglon en el bloque (el primer renglón es el 0)
- en que archivo de la base de datos está (el primero es el 1).

El ROWID tiene muchos usos importantes, son una forma rápida para accesar un renglón particular, pueden usarse para ver como son almacenados los renglones y son el identificador único que puede tomar un renglón en una tabla.

Los identificadores ROWID permanecen contantes durante el tiempo de vida del renglón. Sin embargo, no debe tomarse el ROWID como una llave primaria, es decir, como un campo de la tabla que no cambia. Un ROWID puede cambiar si se borra el renglón y reinserta otro, también al borrar un renglón ORACLE puede reasignar el ROWID a un nuevo renglón que se inserte después.

Los ROWID pueden usarse en las clausulas SELECT y WHERE de una consulta, estos no son almacenados en la base de datos y por lo tanto no modifican el dato. No se puede modificar, insertar o borrar un valor de ROWID.

II.4 Conversión de Tipos de Datos

La cláusula WHERE y expresiones de consultas pueden hacer referencia a diferentes tipos de datos en las comparaciones. En este caso, ORACLE puede convertir los tipos de datos para poder realizar la consulta. Por ejemplo en los siguientes casos ORACLE realiza las conversiones automáticamente:

```
SELECT NOMBRE FROM EMP WHERE NOMBRE = 135

SELECT NOMBRE FROM EMP WHERE NUM_EMP = '345'

SELECT NOMBRE FROM EMP WHERE FECHA = '20-APR-93'

SELECT NOMBRE FROM EMP WHERE ROWID = '00002514.0001.0001'
```

Para realizar la conversión ORACLE puede convertir una constante al tipo de dato definido para la columna de la tabla, convertir los valores de la columna al tipo de dato de la constante o convertir el tipo de dato de una columna al tipo de dato definido para otra columna.

La figura II.3 muestra las funciones más comunes para convertir un valor de un tipo de dato a otro.

	A		
DE	**CHAR**	**NUMBER**	**DATE**
CHAR	No necesario	TO_NUMBER	TO_DATE
NUMBER	TO_CHAR	No necesario	TO_DATE
DATE	TO_CHAR	Inválido	No necesario

Figura II.3

La conversión de datos depende del contexto en el que se este trabajando. Algunas veces se puede hacer una conversión automática y algunas veces se deberá especificar explícitamente la conversión usando las funciones de SQL por las siguientes causas:

- Los argumentos de SQL son más fáciles de comprender cuando se usa una conversión de datos explícita.
- La conversión automática puede tener efectos negativos, particularmente si un tipo de dato de una columna es convertido al mismo que al de alguna constante.

- Los algoritmos para conversiones implícitas estan sujetos a cambio entre las versiones de los productos de ORACLE.

II.5 Funciones de SQL

Las funciones nos permiten modificar valores, combinar valores para crear nuevos valores, cambiar formato de valores. Se pueden utilizar en cualquier tipo de consulta, incluyendo las más complejas. Una ventaja de una función es que puede regresar un valor basado en múltiples argumentos.

Una función tiene el siguiente formato:

FUNCION (argumento)

Una función puede tener más de un argumento:

FUNCION (arg 1, arg 2, ..., arg N)

Por ejemplo:

substr(nombre,1,8)

El primer argumento de una función de SQL es siempre el valor al cual se le aplica la función. Existen varios tipos de funciones.

De acuerdo a los renglones que afectan las funciones se clasifican en:

Funciones INDIVIDUALES
Funciones GRUPALES

En las funciones individuales cada renglón es evaluado por separado, la función se aplica a cada renglón de la tabla.

Las funciones de grupo evalúan conjuntos de renglones y regresan un valor único.

De acuerdo a su tipo de valor las funciones se clasifican en:

Funciones CHAR
Funciones NUMERICAS
Funciones DATE

Funciones Tipo CHAR

Algunas de las funciones tipo CHAR (caracter) mas comunes de SQL son:

INITCAP(argumento)

Pone en mayúscula la primera letra de cada palabra. Por ejemplo, si se aplica la funcion a una columna que contenga nombres se obtendria lo siguiente:

INITCAP(nombre)

josé lópez se convierte en José López
EDUARDO JIMENEZ se convierte en Eduardo Jiménez
IRMA guerrero se convierte en Irma Guerrero

LENGHT(argumento)

Calcula el número de caracteres de una cadena. Por ejemplo:

LENGHT(ALUMNO) = 6
LENGHT(AUTOMOVIL) = 9

SUBSTR(arg1,arg2,arg3)

Muestra una subcadena dependiendo de los parámetros arg2 y arg3. Si la función se aplica a una columna de estados nos daría los siguientes resultados.

SUBSTR(estado,1,3)

Chihuahua regresa la cadena Chi
Morelos regresa la cadena Mor

Lo que nos dice la función es que de la columna estados nos regrese a partir del primer caracter los siguientes tres.

SUBSTR(estado,3,3)

Chihuahua regresa la cadena ihu
Morelos regresa la cadena rel

Esta vez la función nos regresa a partir del tercer caracter nos regresa los siguientes tres.

LOWER(argumento)

Convierte todos los caracteres de la cadena en minúsculas. Aplicando esta función a la columna de ocupaciones tenemos lo siguiente:

LOWER(ocupación)

OBRERO regresa la cadena obrero
Oficinista regresa la cadena oficinista

UPPER(argumento)

Convierte todos los caracteres de la cadena en mayúsculas. Aplicando esta función a la misma columna de ocupaciones obtenemos los siguientes resultados.

UPPER(ocupación)

obrero regresa la cadena OBRERO
Oficinista regresa la cadena OFICINISTA

TRANSLATE(char, from, to)

Regresa la cadena de caracteres char reemplazando todas las ocurrencias de from con el correspondiente caracter en to.

TRANSLATE('HOLA! AQUI!','!','-')

Regresa la siguiente cadena: HOLA- AQUI-

REPLACE(arg1, arg2, arg3)

Regresa la cadena de caracteres especificada por arg1, con todas las ocurrencias de arg2 reemplazadas por arg3, si no se especifica arg3 todas las ocurrencias de arg2 son borradas. Si ambos arg1 y arg2 no son especificados, la función regresa un nulo.

REPLACE('JACK & JUE','J','BL')

La función entrega la siguiente cadena de caracteres:

BLACK & BLUE

Funciones Tipo NUMBER

Utilizando funciones numéricas se pueden efectuar operaciones sobre un conjunto de valores de una tabla o vista. Algunos ejemplos de funciones numéricas son:

ABS(argumento)

Retorna el valor absoluto del argumento entre los paréntesis, este valor puede ser una variable, una columna o una operacion. Por ejemplo, en una tabla de empleados donde se tengan columnas de comisiones y salarios se puede aplicar esta función.

ABS(COM-SAL)

En este ejemplo la función ABS nos da el valor aboluto de la diferencia entre la comisión y el salario.

ROUND(arg1,arg2)

ROUND regresa arg1 redondeado a arg2 lugares a la derecha del punto decimal; si arg2 es omitido, son 0 lugares, arg2 puede ser negativo para redondear los dígitos a la izquierda del punto decimal. arg2 debe ser entero

ROUND(15.193,1) = 15.2

SIGN(argumento)

Esta función regresa un -1, 0 o +1 dependiendo del valor del dato. Por ejemplo:

SIGN(COM-SAL)

Puede regresar lo siguiente:

SIGN = -1 si COM-SAL < 0
SIGN = 0 si COM-SAL = 0
SIGN = +1 si COM-SAL > 0

NVL(arg1,arg2)

NVL permite un manejo apropiado de valores nulos. Si no se usa la función NVL al sumar un valor nulo con una cantidad el resultado será nulo. El parámetro arg1 es el nombre de una columna. Si arg1 no es nulo NVL regresa su valor. Si arg1 es nulo NVL regresa el valor de arg2.

NVL(COM,0) + SAL

Si el valor de COM no es nulo NVL regresa la suma de COM+SAL, si COM es nulo NVL regresa la suma de 0 + SAL.

CEIL(arg1)

Redondea al número inmediato mayor o igual a arg1). Por ejemplo:

CEIL(15.7) = 16

FLOOR(arg1)

Redondea al número inmediato igual o menor que arg1

FLOOR(15.7) = 15

MOD(arg 1, arg2)

Regresa el residuo que resulta de dividir arg1 entre arg2. Si arg2 es 0 regresa arg1.

MOD(7,5) = 2

POWER(arg1, arg2)

POWER regresa el arg 1 elevado a la potencia especificada por arg2. arg2 debe ser entero, de lo contrario regresa un mensaje de error.

POWER(3,2) = 9

SQRT(arg1)

Regresa la raíz cuadrada de arg1; si arg1 < 0 regresa un nulo.

SQRT(25) = 5

Funciones Tipo DATE

Las funciones tipo DATE se especifican para valores de tiempo, estas funciones se especifican de la misma forma que las funciones CHAR, con el nombre de la función seguido por sus argumentos entre paréntesis.

ADD_MONTHS(arg1, arg2)

Esta función suma un número determinado de meses arg2 a una fecha arg1. Por ejemplo:

ADD_MONTHS(fec_ini,6)

Suma 6 meses a la fecha especificada en fec_ini

MONTHS_BETWEEN(arg1 , arg2)

Calcula el numero de meses entre dos fechas arg1 y arg2.

MONTHS_BETWEEN(fec_ini,fec_fin)

NEXT_DAY(arg1, arg2)

Encuentra la fecha del siguiente dia arg2 a partir de la fecha arg1. Por ejemplo:

NEXT_DAY(fec_ini,'friday')

Encuentra la fecha del siguiente viernes a partir de la fecha de inicio (fec_ini).

TO_CHAR(arg1, arg2)

Las fechas son desplegadas en el formato default de ORACLE, (por ejemplo, '12-JUN-93') a menos que se utilice la función TO_CHAR.

La fecha va a ser representada como una cadena de caracteres de acuerdo al formato dado en arg2. Los formatos son los siguientes:

DIA		
dd	número	12

dy	abreviado	fri
day	deletreado	friday
ddspth	deletreado ordinal	twelfth

MES

mm	número	3
mon	abreviado	mar
month	deletreado	march

AÑO

yy	año	93
yyyy	año y siglo	1993

Si los formatos se presentan en mayúsculas la salida lo refleja.

Un ejemplo de la función TO_CHAR es:

TO_CHAR(fec_ini,'Dy Mon dd,yyyy')

La función regresará un valor como Tue Jun 09,1993

TO_DATE(arg1, arg2)

La representación interna de una fecha va desde el siglo hasta el segundo. La función TO_DATE convierte una cadena de caracteres de una gran variedad de formatos a un dato tipo DATE de ORACLE

TO_DATE('070387083000','MMDDYYHHMISS')

Funciones Grupales

Las funciones grupales regresan un resultado basado en un grupo de renglones. Las funciones grupales pueden aplicarse a cualquier valor numérico y a algunos valores tipo char y date.

AVG([DISTINCT|ALL] arg1)

Regresa el porcentaje de arg1, ignorando valores nulos. Las opciones de DISTINCT y ALL son opcionales y son para valores únicos y para todos los valores respectivamente

AVG(SAL) = 2073.1237 Es el porcentaje de la columna de salarios

COUNT(arg1)

Obtiene el número de renglones de una tabla, incluyendo los duplicados y aquellos que son nulos. Puede combinarse con las opciones DISTINCT para eliminar los duplicados, el asterisco (*) implica todos los renglones.

MAX(arg1)

Regresa el máximo valor de arg1

MAX(SAL) = 50000 Obtiene el máximo salario de la columna de salarios

MIN(arg1)

Regresa el mínimo valor de arg1

MIN(SAL) = 10000 Obtiene el mínimo salario de la columna de salarios.

STDDEV(arg1)

Regresa la desviación estandard de arg1, ignorando los valores nulos

STDDEV(SAL) = 1182.50322 Calcula la desviación estandard de la columna de salarios.

SUM(arg1)

Obtiene la suma de arg1

SUM(SAL) = 29025 Calcula la suma de los salarios de la columna de salarios.

VARIANCE(arg1)

Regresa la varianza de arg1, ignorando valores nulos.

VARIANCE(SAL) = 1389313.67 Obtiene la varianza de la columna de salarios.

II.6 Comandos de SQL

Los comandos de SQL se utilizan para crear, almacenar, cambiar, recuperar y mantener la información de la Base de Datos ORACLE.

Un comando de SQL se guarda en una parte de la memoria llamada SQL Buffer, en donde se queda hasta que un nuevo comando es introducido.

Comando CREATE TABLE

El comando CREATE TABLE es usado para crear una tabla que es la estructura básica para almacenar información en una Base de Datos ORACLE. La sintáxis de este comando es:

```
CREATE TABLE nombre
(columna1  Tipo_columna,
 columna2  Tipo_columna,
 .
 .
 .
 columnan  Tipo_columna)
 PCTFREE n
 PCTUSED n
 INITRANTS n
 MAXTRANTS n
 TABLESPACE tablespace
 STORAGE(INITIAL n NEXT n MINEXTENTS n MAXEXTENTS n
      PCTINCEASE n)
```

El parámetro PCTFREE se refiere a la cantidad de espacio reservado en todos los bloques de una tabla para posibles modificaciones de los datos de una tabla. El PCTFREE es especificado como un entero positivo de 0 a 100 que indica el porcentaje de bloques que deben ser reservados para futuras modificaciones de los renglones en los bloques.El valor por default es 10.

PCTUSED especifica el espacio mínimo que ORACLE puede reservar para cada bloque de una tabla. El PCTUSED se especifica como un entero positivo de 0 a 100, el default es 40. Un valor alto de PCTUSED puede resultar en un uso más eficiente del espacio en una tabla, pero reduce el óptimo funcionamiento al reservar más espacio.

MAXTRANS es el máximo número de transacciones que pueden modificar un bloque de datos concurrentemente. Cada transacción que puede modificar un bloque de datos requiere una entrada de transacción. El espacio usado por una entrada de transacción depende del sistema operativo, y es usualmente 23 bytes. MAXTRANS tiene un rango de 1 a 225 y el default es 225.

INITRANS es el número inicial de entradas de transacción que puden ser reservadas para cada bloque. Tiene un rango de 1 a 225 y el default es 1. Cuando el número de transacciones concurrentes excede el número especificado en el INITRANS, las entradas de transacciones son manejadas por el parámetro MAXTRANS.

TABLESPACE es el nombre del tablespace en el cual la tabla será creada. Si no se especifica el tablespace la tabla es creada en el tablespace especificado por default para cada usuario.

STORAGE controla el espacio reservado para cada tabla. El STORAGE debe ser tal no se tenga que no sea necesario añadir espacio frecuentemente. El STORAGE tiene varias cláusulas que son: INITIAL, es el tamaño en bytes del primer extent cuando el objeto es creado, el INITIAL por default es 10240 bytes, el mínimo INITIAL es 4095 bytes y el máximo 4095 megabytes. El NEXT es el tamaño en bytes de los subsecuentes extents, el default es 10240, el mínimo es 2048 bytes y el máximo 4095 megabytes. MAXEXTENTS especifica el número máximo de extents incluyendo el primero, que puede ser reservado. El default es 99 extents. El MINEXTENTS es el número mínimo de extents que pueden ser reservados. El PCTINCREASE es el porcentaje en que los extents van a crecer, el PCTINCREASE por default es 50.

Por ejemplo para crear una tabla donde se lleve el registro de los empleados de una empresa el comando CREATE sería como se muestra a continuación.

```
CREATE TABLE EMPLEADO
(FICHA    NUMBER(8) NOT NULL,
NOMBRE    CHAR(15),
DEPTO     CHAR(15),
FECHA     DATE,
SALARIO   NUMBER(10,2),
COMISION  NUMBER(10,2))
PCTFREE 10
PCTUSED 40
INITRANS 1
MAXTRANS 225
TABLESPACE personal
STORAGE(INITIAL 30K NEXT 10K MINEXTENTS 1 MAXEXTENTS 5
PCTINCREASE 5)
```

Esta tabla tiene cinco columnas la primera de ellas es la FICHA y debe ser no nula, es decir, siempre debe existir, los parámetros de PCTFREE, PCTUSED, INITRANS y MAXTRANS son los de default ya que se considera que la tabla no tendrá muchos cambios en cuanto al tamaño de los datos. Los parámetro del STORAGE no son muy grandes ya que para este ejemplo se considera que la tabla no crecerá mucho, es decir, que no se insertarán muchos registros que sobrepasen el espacio inicial de 30K, en caso contrario se tendría el NEXT de 10K y un máximo de 5 extents con un porcentaje de crecimiento de 5 por ciento.

Comando SELECT

Utilizando SQL se puede consultar la Base de Datos de muchas formas. A pesar de que SQL es flexible, su sintáxis es muy específica. La sintáxis de una consulta es:

SELECT columna1, columna2, coulumna3, ... , columnaN
FROM tabla1, tabla2, tabla3, ... , tablaN
[WHERE condiciones]
[GROUP BY columna1, columna2, columna3, ... , columnaN]
[HAVING condiciones grupales]
[ORDER BY columna1, columna2, columna3, ... , columnaN]

SELECT especifica las columnas de una tabla, vista o sinónimo seleccionadas. Utilizando el asterisco (*) se seleccionan todas las columnas de una tabla. Si se quiere alterar el order de las columnas regresadas se especifica en el orden en que aparecen en el SELECT.

FROM especifica las tablas, vistas o sinónimos de donde se va a realizar la consulta. Mas de una tabla especifica un join, es decir una consulta donde se toman datos de varias tablas.

WHERE especifica las condiciones para ser usadas en la selección. Estas condiciones pueden contener subconsultas (subqueries) es decir SELECT dentro de un WHERE. Esta clausula puede no ir, en este caso la conulta no tendría condiciones.

GROUP BY, HAVING son usadas para desplegar una información sumarizada de un grupo de renglones que tienen los mismos valores en uno o más campos.

ORDER BY especifica el orden en el cual los renglones de una tabla van a ser regresados o desplegados.

A continuación se describe la tabla EMPLEADO figura II.4 a partir de la cual se ejemplifican varias formas de hacer una consulta.

EMPLEADO					
FICHA	NOMBRE	DEPTO	FECHA	SALARIO	COMISION
304811	MARTHA	VENTAS	17-DEC-94	1,000	
288832	ELOISA	CONTABILIDAD	01-JUN-95	2,400	300
663553	EDUARDO	PERSONAL	09-DEC-94	1,000	500
993873	ARMANDO	ANALISTA	28-MAY-95	800	120
037643	IRMA	ANALISTA	21-MAR-95	3,400	
152533	MOISES	VENTAS	13-JUL-95	1,500	100
845282	LUIS	PERSONAL	01-APR-91	400	600

Figura II.4

Para seleccionar todas las columnas y todos los renglones de la tabla EMPLEADO el comando SELECT queda como:

SELECT *

```
FROM EMPLEADO
```

Si se quiere seleccionar solo el nombre y el departamento la consulta queda como:

```
SELECT NOMBRE, DEPTO
FROM EMPLEADO
```

Para seleccionar los nombres, departamento y salario de los empleados que trabajan como analistas la consulta queda como:

```
SELECT NOMBRE, DEPTO, SALARIO
FROM EMP
WHERE DEPTO = 'ANALISTA'
```

Si se quiere seleccionar todos los datos de los empleados que ganen menos de 1,000, que los agrupe y ordene por número de empleado

```
SELECT *
FROM EMPLEADO
WHERE SALARIO > 1000
ORDER BY NUMERO
GROUP BY NUMERO
```

Seleccionar los nombre de los empleados que comiencen con A

```
SELECT NOMBRE
FROM EMPLEADO
WHERE NOMBRE like 'A%'
```

Para obtener el número de trabajadores y la suma de los salarios la consulta queda como:

```
SELECT COUNT(*), SUM(SALARIO)
FROM EMPLEADO
```

Seleccionar el nombre y la fecha de contratación de los empleados que no tengan comisión, y que los ordene por orden alfabético.

```
SELECT NOMBRE, FECHA, COMISION
FROM EMPLEADO
WHERE COMISION is NULL
```

```
ORDER BY NOMBRE
```

Si se quiere seleccionar la suma de salarios por departamento que sea mayores a 10,000, y el nombre de los empleados por departamento, la consulta queda como:

```
SELECT NOMBRE,DEPTO, SUM(SALARIO)
FROM EMPLEADO
GROUP BY DEPTO, NOMBRE
HAVING SUM(SALARIO) > 10000
```

Se usa la cláusula HAVING por que se utiliza la función grupal SUM como una condición.

Otra forma de consulta es cuando se tienen varias condiciones, por ejemplo para seleccionar los trabajadores que ganen mas de 1,500 y que hayan sido contratados en mayo., ordenarlos en forma decreciente por su número.

```
SELECT NUMERO, NOMBRE, SALARIO, FECHA
FROM EMPLEADO
WHERE SALARIO > 1500
AND FECHA like '%MAY%'
ORDER BY NUMERO DEC
```

Comando INSERT

El comando INSERT añade registros a una tabla específica. La sintáxis del comando es:

```
INSERT INTO nombre_tabla [col1, col2, col3, ... , coln]
    VALUES(val1, val2, val3, ..., valn)
```

Los valores a ser insertados deben estar separados por coma, los valores deben coincidir con el tipo de datos de la columnas de la tabla a la que se insertan. Los valores tipo CHAR y DATE deben estar entre apóstrofes.

Las columnas se especifican en caso de que los datos se inserten solamente en algunas columnas de la tabla, o para introducir los datos en alguna secuencia específica. En caso de que se vaya a insertar en todas las columnas y en el orden en que estan en la tabla no se especifican las columnas, solo la tabla.

Por ejemplo para insertar un dato en la tabla EMPLEADO de la figura II.4, el comando INSERT quedaría como:

```
INSERT INTO EMPLEADO
VALUES(241288,'JORGE','VENTAS','23-DEC-93',3000,210)
```

Se puede utilizar el comando INSERT con un query para seleccionar renglones de una tabla e insertarlos en otra, el query sustituye la cláusula VALUES. Por ejemplo si se tuviera otra tabla llamada EMP con las columnas NUMERO, NOMBRE, SAL se puede utilizar el comando INSERT como:

```
INSERT INTO EMPLEADO (FICHA,NOMBRE,SALARIO)
SELECT NUMERO,NOMBRE,SAL
FROM EMP
WHERE SAL < 1000
```

Se pueden insertar valores nulos a columnas de una tabla (a menos que NOT NULL este especificado para esa columna cuando la tabla fue creada).

En la tabla EMPLEADO existen trabajadores sin comisión, es decir esta es nula. Para insertar un empleado sin comisión el comando queda de la siguiente forma:

```
INSERT INTO EMPLEADO
VALUES(123764,'MANUEL','PERSONAL','12-JUL-93',20000,NULL)
```

Comando DELETE

El comando DELETE borra los renglones de una tabla dejando la estructura de la misma. Con este comando se pueden borrar todos los renglones de una tabla o renglones específicos.

El comando DELETE tiene la siguiente sintáxis:

```
DELETE FROM nombre_tabla
[WHERE condicion]
```

Por ejemplo para borrar todos los registros de la tabla EMPLEADO de la figura II.4 el comando quedaría de la siguiente forma:

```
DELETE FROM EMPLEADO
```

Se pueden borrar renglones que cumplan ciertas condiciones, por ejemplo para borrar todos los empleados del departamento de personal quedaría como:

```
DELETE FROM EMPLEADO
WHERE DEPTO = 'PERSONAL'
```

Comando DROP

Con el comando DROP se borran todos los registros de una tabla y también la estructura de la tabla. El espacio que se tenía reservado para la tabla queda disponible para ser usado nuevamente.

El comando tiene la sintáxis que se muestra a continuación.

DROP TABLE nombre_tabla

Para dropear la tabla EMPLEADO el comando sería:

DROP TABLE EMPLEADO

Comando UPDATE

Este comando nos sirve para modificar los datos en una tabla. La sintáxis es la siguiente:

UPDATE nombre_tabla
SET columna = valor [, columna = valor ...]
[WHERE condiciones]

Tomando la tabla EMPLEADO de la figura II.4, si quisieramos cambiar todas las comisiones de los empleados y que ninguno tuviera comisión, el comando UPDATE quedaría de la siguiente forma:

UPDATE EMPLEADO
SET COMISION IS NULL

Con la cláusula WHERE se modifican renglones específicos, por ejemplo para cambiar el salario de los trabajadores de la tabla EMPLEADO y que este sea igual a 3000 siempre y cuando hayan sido contratados en diciembre de 1993. El comando quedaría como:

UPDATE EMPLEADO
SET SALARIO = 3000
WHERE FECHA LIKE '%DEC-94'

Para cambiar el salario y la comisión de los trabajadores del departamento de ventas y que su ficha comience con 3,el nuevo salario será igual a 2500 y la comisión igual a 300. El comando UPDATE se usará de la siguiente forma.

UPDATE EMPLEADO

```
SET SALARIO = 2500, COMISION = 300
WHERE DEPTO = 'VENTAS'
AND FICHA LIKE '3%'
```

Comando COMMIT

El comando COMMIT se utiliza para salvar todos los cambios de una transacción y con esto marcar el fin de la transacción.

Una transacción (o unidad lógica de trabajo) es una secuencia de comandos de SQL que ORACLE trata como una unidad. Una transacción comienza después de un COMMIT, ROLLBACK o después de la conexión a la Base de Datos. Una transacción termina con un COMMIT, ROLLBACK o cuando se desconecta de la Base de Datos.

Una inserción (INSERT),un borrado (DELETE) o un cambio (UPDATE) a una tabla no se hace permanente hasta que el trabajo es salvado en la Base de Datos (COMMIT). Hasta que el trabajo es salvado, únicamente el usuario que hizo los cambios los puede ver, todos los demás usuarios ven los datos como estaban al momento del último COMMIT. La sintáxis es la siguiente:

COMMIT [WORK]

La palabra WORK puede o no ir.

Comando ROLLBACK.

El comando ROLLBACK cancela todos los cambios pendientes regresando al estado en que estaba la información al momento del último COMMIT. La sintáxis es:

ROLLBACK

CAPITULO III

El Precompilador Pro*C

III.1 Caracteristicas de un Precompilador

Un precompilador es una herramienta de programación que nos permite combinar argumentos de SQL con un programa de alto nivel. Existen seis precompiladores disponibles en ORACLE que soportan los siguientes lenguajes de alto nivel.

-Ada
-C
-FORTRAN
-Pascal
-PL/I
-COBOL

Los precompiladores combinan el poder y la flexibilidad del lenguaje SQL dentro de los programas de aplicación. Además, los precompiladores permiten el monitoreo de los recursos, la ejecución de los argumentos de SQL, e indicadores de tiempo de ejecución.

Através de la precompilación se añaden pasos al proceso de desarollo de aplicaciones, este paso es el de traducir cada argumento de SQL en varias llamadas ORACLE al lenguaje de alto nivel.

En general los precompiladores tienen ciertas características que permiten:

- Desarrollar una aplicación en cualquiera de los seis lenguajes de programación mencionados anteriormente.
- Utilizar los estándares ANSI para los argumentos de SQL en un lenguaje de alto nivel.
- Tomar las ventajas de los argumentos de SQL en técnicas avanzadas de programación.
- Diseñar y desarrollar aplicaciones para usuarios
- Conversión automática entre tipos de datos de ORACLE y los tipos de datos de los lenguajes de alto nivel.
- Provee utiles opciones de programación en línea y comandos en línea.
- Precompila por separado múltiples módulos del programa, y los liga dentro de un programa ejecutable.
- Provee un chequeo completo de sintáxis y semántica para argumentos de SQL.
- Manejo de errores y advertencias en el Area de Cominicación de SQL (SQLCA) y el argumento WHENEVER

III.2 Introducción al Pro*C

El término de "argumentos de SQL" se refiere a comandos o sentencias de SQL que están dentro de un programa de aplicación. El programa host es el programa donde se desarrolla la aplicación, y el lenguaje en el que es escrito el programa es el lenguaje host y por último una variable host es aquella variable donde se almacenara o de donde se tomara algun valor dentro de un Pro*C.

El término de "argumentos de SQL" se refiere a comandos o sentencias de SQL que están dentro de un programa de aplicación. El programa host es el programa donde se desarrolla la aplicación, y el lenguaje en el que es escrito el programa es el lenguaje host y por último una variable host es aquella variable donde se almacenara o de donde se tomara algun valor dentro de un Pro*C.

Cualquier argumento valido de SQL puede ser ejecutado en un programa en Pro*C. Las lineas de programación pueden aparecer en cualquier parte del programa.

Para minimizar las dificultades que se puedan presentar para identificar argumentos de SQL dentro de un programa, todos los argumentos de SQL comienzan con las palabras EXEC SQL .

Usando el precompilador se añade un paso dentro del proceso de programación normal. La secuencia normal de eventos en la escritura y ejecución de un programa en C es la siguiente :

1. Escribir el programa en C.
2. Compilar el programa, dando como resultado un archivo objeto de salida.
3. Ligar el archivo objeto (link), resultando el archivo ejecutable.
4. Correr el programa para realizar el trabajo deseado.

Si se incluyen argumentos en Pro*C en el programa original se debe incluir un paso adicional, como se muestra a continuación.

1. Escribir el programa en Pro*C.
2. Precompilar el programa usando Pro*C, resultando un archivo de salida.
3. Compilar el programa, resultando un archivo objeto.
4. Ligar (link) el archivo objeto, resultando el archivo ejecutable.
5. Correr el programa para realizar el trabajo deseado.

La figura III.1 muestra un esquema del desarrollo de un programa en Pro*C.

Como se puede observar en la figura, el resultado de la precompilación es un programa fuente modificado que puede ser compilado normalmente.

III.3 Elementos de un Programa en Pro*C

Un programa en Pro*C está formado por dos partes, ambas requeridas por el precompilador:

- El Prólogo de la Aplicación
- El Cuerpo de la Aplicación

El Prólogo de la Aplicación define variables y realiza una preparación general del programa en Pro*C. El Cuerpo de la Aplicación comprende básicamente las llamadas de Pro*C, incluyendo los argumentos de SQL para manipular los datos de la base de datos.

El Prólogo de la Aplicación

Al principio de todos los programas en Pro*C está el Prólogo de la Aplicación, el cual consta de tres partes:

1. La sección DECLARE, para declarar las variables host.
2. El argumento INCLUDE SQLCA, que declara el Area de Cominicaciones de SQL, el cual provee el manejo de errores.
3. El argumento CONNECT, para conectarse al RDBMS de ORACLE

Solo los argumentos de programación en C pueden precedir estos argumentos en un programa en Pro*C.

En la sección DECLARE (sección de Declaración) todas las variables host que pueden ser usadas en el programa en C son declaradas. La sección DECLARE comienza con el argumento:

EXEC SQL BEGIN DECLARE SECTION;

y termina con el argumento

EXEC SQL END DECLARE SECTION;

Entre estos dos argumentos se declaran las variables host que van a ser ocupadas en el programa.

Solo una sección BEGIN/END DECLARE es permitida por unidad de precompilación, pero un programa puede tener varias unidades de precompilación independientes.

Si se hace referencia a una variable host dentro de un argumento EXEC SQL en el programa y no ha sido declarada en la DECLARE SECTION aparecerá el siguiente mensaje cuando el programa es precompilado:

Undeclared host variable a at line b in file c

donde a es el nombre de la variable, b es el número de la línea en la cual fue usada la variable, y c es el nombre del archivo.

Una variable host debe ser declarada para cualquier valor que pueda tomar, ya sea en un argumento SQL o en algún argumento de programación en C. Los tipos de datos deben ser declarados usando el lenguaje host en la DECLARE SECTION, en este caso el lenguaje host es el C. Estos tipos de datos no necesariamente deben ser iguales a los tipos de datos usados en ORACLE cuando el dato fue definido; ORACLE puede realizar todas las posibles conversiones entre los tipos de datos de C y los tipos de datos de ORACLE.

Un ejemplo de la DECLARE SECTION usando columnas de la tabla EMPLEADO (Ver APENDICE A) se muestra a continuación.

```
EXEC SQL BEGIN DECLARE SECTION

int sal;          /* salario de empleado */
char nombre[20];  /* nombre del empleado */
int comm;         /* comisión del empleado */

EXEC SQL END DECLARE SECTION
```

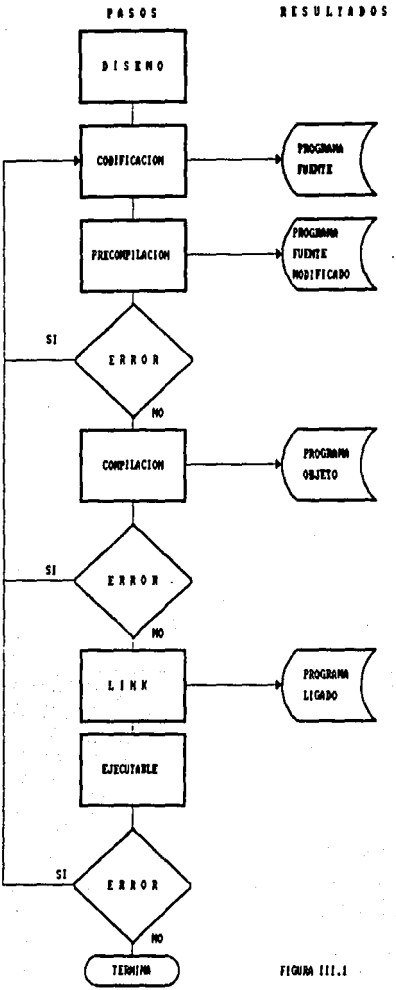
PASOS
RESULTADOS
DISEÑO
CODIFICACION
PROGRAMA FUENTE
PRECOMPILACION
PROGRAMA FUENTE MODIFICADO
SI
ERROR
NO
COMPILACION
PROGRAMA OBJETO
SI
ERROR
NO
LINK
PROGRAMA LIGADO
EJECUTABLE
SI
ERROR
NO
TERMINA

FIGURA III.1

En el ejemplo se declararon tres variables host sal, nombre y comm, las cuales pueden aparecer en un argumento SQL en cualquier parte del programa (en el cuerpo de la aplicación). Por ejemplo:

```
EXEC SQL SELECT SALARIO, NOMBRE, COMISION
     INTO :sal,:nombre, :com
     FROM EMPLEADO
     WHERE NOMBRE = :nombre;
```

Una variable host:

- debe ser explícitamente declarada en la sección DECLARE.
- debe ser usada en el mismo formato, mayúscula o minúscula en el que fue declarada.
- debe ser precedida por dos puntos (:) en el argumento SQL.
- no debe ser precedida por dos puntos (:) en un argumento de C.
- no debe ser una palabra reservada.
- puede tener asociado un indicador de variable.

Un indicador de variable es una variable opcional que corresponde uno a uno con una variable host declarada en la sección DECLARE. Este identificador de variable es usado principalmente para tratar con valores nulos. Estos son usados para almacenar campos individuales que regresan código que indican "valor nulo regresado" o "campo de caracter truncado".

Pueden declararse punteros a variables en la sección DECLARE de la misma forma que se declaran en C, usando un asterísco (*) y el nombre de la variable, como se muestra en el siguiente ejemplo:

```
EXEC SQL BEGIN DECLARE SECTION

int i, j, *ptr;
char *cp;

EXEC SQL END DECLARE SECTION
```

Cuando se usa en un argumento SQL ,el nombre de la variable debe ser precedida por dos puntos (:) no por un asterisco, el asterístico es implícito.

```
SELECT INTFIELD
 INTO :ptr
FROM ....;
```

Para los punteros tipo caracter, el tamaño es tomado del tamaño del tipo de base especificado en su declaración. Para punteros tipo caracter, el objeto al que se hace referencia es asumido para ser una cadena terminada en nulo, cuyo tamaño es determinado por la función de longitud de cadena strlen (string length).

Si un puntero a una variable caracter es usado como una variable host de entrada, su longitud es determinada por una llamada a la función strlen.

El Pro*C permite usar el pseudotipo VARCHAR para cadenas de caracter con longitud variable. Esta es solo referida en la sección DECLARE . Un ejemplo de una variable declarada como VARCHAR se muestra a continuación.

```
EXEC SQL BEGIN DECLARE SECTION

VARCHAR jobDesc[40];

EXEC SQL END DECLARE SECTION
```

Las variables VARCHAR nunca están terminadas en nulo. Una cadena de caracteres puede definirse de dos formas:

```
char str[100];     /*como arreglo de caracter */
char *str          /*como puntero de cadena */
```

Cuando se define como arreglo, la longitud es determinada por la dimensión de la cadena (100 en el ejemplo).

Las variables tipo VARCHAR son redefinidas en los argumentos de SQL poniendo el nombre de la variable precedida por dos puntos, de la misma forma que las demas variables. Por ejemplo la variable declarada anteriormente puede usarse como:

```
EXEC SQL SELECT jobDesc
     INTO :jobDesc
     FROM EMPLEADO;

jobDesc.len=strlen(jobDesc.arr);

ó

EXEC SQL SELECT ...
     INTO ...
     FROM EMPLEADO
     WHERE jobDesc= :jobDesc;
```

Cuando una variable VARCHAR es usada como salida de una variable host (por ejemplo la variable de la cláusula INTO), la longitud del campo es establecida por el RDBMS de ORACLE. Cuando es usada como una variable host de entrada (por ejemplo en la cláusula WHERE) la longitud debe especificarse en el programa.

Area de Comunicaciones de SQL (SQLCA)

Todos los programas en Pro*C deben incluir en el Prólogo de la Aplicación una referencia al Area de Comunicaciones de SQL, esto se hace incluyendo la siguiente línea:

```
EXEC SQL INCLUDE SQLCA;
```

El argumento EXEC SQL INCLUDE es usado para incluir otros archivos en el programa en Pro*C. Este argumento reemplaza el #include de C para el manejo de compatibilidad de lenguaje.

El programa en Pro*C debe estar habilitado para localizar el archivo SQLCA al momento en que se lleva a cabo la precompilación. El SQLCA contiene definiciones de variables usadas para comunicarse con ORACLE. Cuando el programa es precompilado, ORACLE sustituye algunas variables del lenguaje

host en lugar del argumento INCLUDE. Estas declaraciones habilitan a ORACLE para funcionar como interfase con el programa, pasando información durante la ejecución del programa acerca del estado de varias operaciones.

El SQLCA incluye información como:

- banderas de advertencia
- códigos de error
- texto de diagnóstico.

Por ejemplo, se puede checar si el argumento de SQL se ejecutó satisfactoriamente, y si así sucedio, cuantos renglones fueron insertados, borrados o modificados. O si por el contrario un argumento de SQL terminó con error, obtener información sobre lo que está pasando.

Con el SQLCA el programador tiene la opción de tomar diferentes acciones en varios puntos del programa, basado en la retroalimentación recivida de ORACLE hacerca del trabajo que se esté realizando.

Por default, un programa en Pro*C puede ignorar errores y continuar procesando cuando es posible. Usando variables incluidas en el SQLCA, un programador puede tener el control de la acción tomada bajo ciertas circunstancias.

Por ejemplo, con el argumento WHENEVER se puede indicar la acción a tomar después de detectar un error, una advertencia, o un evento en particular. La acción puede incluir detener el programa, tomar otra acción, o continuar si es posible.

Dos de las variables más importantes del SQLCA son:

- sqlca.sqlcode. Esta variable almacena el código que resulta, después de la ejecución de todo argumento de SQL. El código es un entero que indica cómo se está ejecutando un argumento. Un cero en el sqlca.sqlcode indica una ejecución exitosa. Un valor negativo indica un error de ORACLE. Un valor positivo indica ejecución exitosa con un código de estado (por ejemplo, fin de archivo). Actualmente está disponible un código positivo; 1403 que indica "registro no encontrado", o "último registro seleccionado".

- sqlca.sqlwarn. Esta variable es actualmente un arreglo de 8 banderas de advertencia. El primer elemento, sqlca.sqlwarn[0], el segundo como sqlca.sqlwarn[1], etc., y son usadas como indicadores de cualquier problema.

ORACLE utiliza otra libreria que es el Area de Comunicaciones ORACLE ORACA, el ORACA solo puede ser usada si la definición del ORACA es incluida dentro del argumento EXEC SQL INCLUDE, y la opción ORACA = YES es seleccionada.

El ORACA incluye la siguiente información:

- orastxt. Texto del Actual Argumento de SQL. Usado especialmente cuando un error es encontrado. Así el programador puede revisar el texto del argumento que es analizado por el RDBMS de ORACLE, el texto que es analizado por el precompilador (como el CONNECT, FETCH, COMMIT). Al menos los primeros 70 caracteres del argumento de SQL serán desplegados en el ORACA.

- orasfnm. Nombre del Archivo con Error. Si múltiples archivos son precompilados para una aplicación, el ORACA puede identificar en cuál archivo ocurrió el error.

orasInr. Número de Linea con Error. Identifica el número de la línea con error en el archivo.

El uso del ORACA es opcional, para usar el ORACA se debe realizar lo siguiente:

1. Usar la opción de la línea de comando:
   ORACA = YES (El default es no).
2. Añadir la siguiente línea en el programa:
   EXEC SQL INCLUDE ORACA
3. Definir las banderas deseadas en las opciones de ORACA.

Conexión a ORACLE (CONNECT)

Todos los programas en Pro*C deben conectarse al RDBMS de ORACLE antes de poder accesar los datos de ORACLE. Para la conexión se utiliza el argumento:

```
EXEC SQL CONNECT :oracleid
     IDENTIFIED BY :oraclepassword;
```

ó

```
EXEC SQL CONNECT :oracleid;
```

- El argumento CONNECT debe ser el primer argumento del SQL ejecutable que debe ser incluido en el Pro*C. Solo argumentos declarativos o código de lenguaje C pueden precedir lógicamente al argumento CONNECT.

- Si el password es proporcionado por el usuario deben utilizarse variables host para el usuario y el password de ORACLE (:oracleid, :oraclepasswd).

- Ambas variables host deben ser declaradas como de longitud fija o como variables de cadena de caracter. Las cadenas de longitud fija deben estar rellenadas a la derecha con blancos (hasta 20 caracteres).

- Ambas variables host deben ser inicializadas antes de que el argumento CONNECT se ejecute.

- A pesar de que el CONNECT es el primer argumento ejecutable no indica que sea el principio de una unidad de trabajo. El próximo argumento encontrado en el programa puede ser el comienzo de la unidad de trabajo.

Para inicializar las variables, el programador puede hacelo dentro del programa o aceptando los valores desde el teclado.

El Cuerpo de la Aplicación

El Cuerpo de la Aplicación contiene argumentos de SQL para consultar y manipular los datos almacenados en la base de datos ORACLE. Estos argumentos son llamados argumentos de Manipulación de Datos (Data Manipulation Langiage DML). El Cuerpo de la Aplicación puede contener

también varios argumentos DDL (Data Definition Language), los cuales son usados para crear o definir estructuras, como tablas, vistas o índices.

Cuando es ejecutado un argumento de manipulación de datos como un INSERT,UPDATE o DELETE nos importa el grupo de valores de cualquier variable host de entrada. Cuando se ejecuta un argumento SELECT sin embargo, también es de interes el número de registros que regresa la consulta.

Las consultas de SQL pueden ser clasificadas como: consultas que sólo regresan un valor y consultas que regresan más de un valor.

Las consultas que regresan más de un valor requieren cursores explícitamente declarados. Un cursor identifica los renglones que actualmente esté regresando la consulta.

Los siguientes argumentos de SQL nos permiten definir y manipular un cursor explícito.

DECLARE. Declara el nombre del cursor
OPEN .Ejecuta la consulta e identifica el grupo de columnas sobre las que se hace la consulta
FETCH .Regresa cada renglón y los intriduce en las variables de salida uno por uno
CLOSE .Deshabilita el cursor

Realizar una consulta es muy común dentro de una aplicación en Pro*C, para realizar una consulta se ejecuta un argumento SELECT. Por ejemplo para una consulta sobre la tabla EMPLEADO de la figura II.4 del Capítulo 2 se utiliza la siguiente sintáxis:

```
EXEC SQL SELECT NOMBRE,DEPTO,SALARIO
     INTO :nom_emp,nom_dpto,:sal
     FROM EMPLEADO
     WHERE FICHA = :ficha;
```

Los nombres de las columnas y las expresiones que siguen a la palabra SELECT forman la lista de selección. En este ejemplo la lista de selección contiene tres datos. Bajo las condiciones especificadas en la cláusula WHERE, ORACLE regresa valores de columnas a las variables hosts en la cláusula INTO.

El número de datos en la lista de selección debe ser igual al número de variables hosts en la cláusula INTO, las variables host son utilizadas para almacenar los valores regresados en el SELECT.

Este es el caso simple, cuando una consulta regresa solo un renglón. Sin embargo, si una consulta regresa más de un renglón, debe efectuarse un FETCH de registros usando un cursor.

Se puede utilizar el argumento INSERT para añadir renglones a una tabla. En el siguiente ejemplo se añade un renglón a la tabla EMPLEADO.

```
EXEC SQL INSERT INTO EMPLEADO (NOMBRE,DEPTO,SALARIO)
     VALUES (:nom_emp,:nom_dpto,:sal);
```

Cada columna especificada en la lista de columnas debe pertenecer a la tabla nombrada en la cláusula INTO. La cláusula VALUES especifica el renglón de valores a ser insertados. Los valores pueden ser constantes, variables host, expresiones de SQL, o pseudocolumnas como USER o SYSDATE.

El número de valores en la cláusula VALUES debe ser igual al número de columnas en la lista de columnas. Se puede omitir la lista de columnas si la cláusula VALUES contiene valores para cada una de las columnas de la tabla.

Un subquery o subconsulta es una variación del argumento SELECT. Los subqueries son usados para:

- suplir valores para comparaciones en las cláusulas WHERE,HAVING, o argumentos como SELECT,INSERT o DELETE.
- definir el grupo de renglones a ser insertados por un argumento CREATE TABLE, o INSERT.
- definir valores para la cláusula SET de un argumento UPDATE.

A continuación se muestra el uso de un subquery en lugar de la cláusula VALUES de un INSERT para copiar los renglones de una tabla a otra.

```
EXEC SQL INSERT INTO EMPLEADO2(NOMBRE,DEPTO,SALARIO)
     SELECT NOMBRE,DEPTO,SALARIO
     FROM EMPLEADO
     WHERE DEPTO = :num_dpto;
```

Se puede utilizar el argumento UPDATE para cambiar los valores de columnas específicas en una tabla. Para modificar el salario de un empleado en especial se utiliza el siguiente argumento UPDATE.

```
EXEC SQL UPDATE EMPLEADO
     SET SALARO = :sal, COMISION = :com
     WHERE NOMBRE = :nom_emp;
```

Se utiliza la cláusula opcional WHERE para especificar las condiciones bajo las cuales un renglón es modificado. La cláusula SET lista el nombre de una o más columnas las cuales serán modificadas. Se puede usar un subquery para proveer estos valores, como se muestra a continuación.

```
EXEC SQL UPDATE EMPLEADO
   SET SALARIO = (SELECT AVG(SALARIO)*1.1
          FROM EMPLEADO
          WHERE DEPTO = :nom_dpto)
     WHERE NOMBRE = :nom_emp;
```

De la misma forma que el SELECT o el INSERT se utiliza el argumento DELETE para borrar renglones de una tabla.

Por ejemplo, para borrar los empleados de un departamento específico se podría utilizar el siguiente argumento.

```
EXEC SQL DELETE FROM EMPLEADO
    WHERE DEPTO = :nom_dpto;
```

La cláusula WHERE es utilizada para seleccionar, modificar o borrar renglones de una tabla que cumplan con ciertas condiciones. Si se omite la cláusula WHERE todos los renglones de la tabla son procesados. Es decir, si se quiere borrar toda la información de la tabla EMPLEADO se tendría lo siguiente:

```
EXEC SQL DELETE FROM EMPLEADO;
```

Cuando un subquery regresa múltiples renglones se debe definir explícitamente un cursor que permita:

- procesar todos los renglones regresados por el query
- tener conocimiento de que renglón se está proceasando actualmente.

Un cursor identifica el renglón actual del grupo de renglones regresados por un query, esto nos permite procesar un renglón a la vez. Los siguientes argumentos nos permiten definir y manipular un cursor:

- **DECLARE**
- **OPEN**
- **FETCH**
- **CLOSE**

Primero se utiliza el argumento DECLARE para nombrar el cursor y asociarlo con el query.

El argumento OPEN ejecuta el query e identifica todos los renglones que cumplen con la condición del query. Estos renglones forman un grupo llamado grupo activo del cursor. Después de abrir el cursor, este se puede usar para recuperar los renglones regresados por el query asociado.

Los renglones del grupo activo son recuperados uno por uno. El argumento FETCH es usado para recuperar el actual renglón en el grupo activo. Se puede ejecutar el FETCH repetitivamente hasta que todos los renglones sean recuperados.

Cuando el FETCH haya terminado de recuperar todos los renglones el cursor se deshabilita con el arcumento CLOSE.

El argumento DECLARE define un cursor por su nombre y le asocia un query, como en el siguiente ejemplo:

```
EXEC SQL DECLARE emp_cursor CURSOR FOR

SELECT NOMBRE,DEPTO,SALARIO
FROM EMP
WHERE DEPTO = nom_dpto;
```

El nombre del cursor es un identificador usado por el precompilador y no debe estar definido en la DECLARE SECTION. El nombre del cursor no debe tener guiones, puede ser de cualquier longitud, pero solo los treinta primeros caracteres son significativos. Para ser compatibles con ANSI, los nombres de los cursores no deben ser mayores de 18 caracteres.

El argumento SELECT asociado con el cursor no debe incluir la cláusula INTO, ya que esta cláusula y la lista de variables host de salida son parte del argumento FETCH.

Por ser declarativo, el argumento DECLARE debe preceder físicamente (no solo lógicamente) a cualquier otro argumento SQL que haga referencia al cursor, una referencia anterior no esta permitida.

Los argumentos de control del cursor (DECLARE, OPEN, FETCH y CLOSE) deben estar todos en la misma unidad de precompilación, es decir no se puede declarar un cursor en un archivo A y abrirlo en el archivo B.

En un archivo se pueden declarar muchos cursores si es necesario, pero no deben declararse dos cursores con el mismo nombre.

El argumento OPEN ejecuta el query e identifica el grupo activo . El cursor llamado emp_cursor queda abierto con el siguiente argumento:

```
EXEC SQL OPEN emp_cursor;
```

La posición de un cursor cuando es abierto (OPEN) es justamente antes del grupo activo, sin embargo ningún renglón es regresado en este punto.

Una vez que el cursor esta abierto, las variables host de entrada no son reexaminadas hasta que se vuelva a abrir el cursor.

Generalmente un cursor debe cerrarse (CLOSE) después de haberlo abierto. Sin embargo si se especifica MODE= ORACLE (valor por default), no es necesario cerrar un cursor despues de reabrirlo.

Se utiliza el argumento FETCH para recuperar los renglones del grupo activo y especifica las variables host de salida que pueden contener el resultado de la consulta.

La cláusula INTO y la lista de valores forman parte del argumento FETCH. En el siguiente ejemplo se realiza el FETCH INTO de tres variables host:

```
EXEC SQL FETCH emp_cursor
```

INTO :nom_emp, :nom_dpto, :sal;

El cursor debe haber sido previamente declarado (DECLARE) y abierto (OPEN).

La primera vez que se ejecuta el FETCH, el cursor mueve el primer renglón en el grupo activo y éste se convierte en el renglón actual. Cada vez que se ejecuta el FETCH , el cursor avanza al siguiente renglón del grupo activo, cambiando el renglón actual.

El cursor solo puede moverse hacia adelante en el grupo activo. Para recuperar un renglón que ya paso por el FETCH, debe reabrirse el cursor, y comenzar nuevamente con el primer renglón del grupo activo.

El FETCH puede ejecutarse varias veces, éste debe estar asociado solo a un grupo de variables host de salida. Si se intenta asignar un FETCH a diferentes variables host de salidase pueden obtener resultados no esperados.

Si se quiere cambiar el grupo activo, primero deben asignarse nuevos valores a las variables host de netrada, y entonces reabrir el cursor.

Si el grupo activo esta vacio, o no contiene más renglones, el FETCH regresará el código de error "no data found". En un programa el argumento WHENEVER NO DATA FOUND detecta esta condición de que no existen mas renglones, y de acuerdo a esto se pueden tomar diferentes caminos, como por ejemplo, cerrar el cursor o detener el programa.

A continuación se presentan varios ejemplos sencillos de programas escritos en Pro*C, donde se muestran la mayoría de los conceptos vistos hasta ahora.

```
/* Ejemplo 1 Programa en Pro*C que realiza una conexión
   a la Base de Datos */

#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;

VARCHAR uid[20];
VARCHAR pwd[20];

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

main(argc,argv)
 int argc;
 char *argv[];
  {
   strcpy(uid.arr,argv[1]);     /*Copia el Usuario*/
   uid.len=strlen(uid.arr);
   strcpy(pwd.arr,argv[2]);     /*Copia el Password*/
```

```
    pwd.len=strlen(pwd.arr);

    EXEC SQL CONNECT :uid  IDENTIFIED BY :pwd;

    printf(" Conectado a ORACLE usuario: %s\n",uid.arr);

    EXEC SQL COMMIT WORK RELEASE;    /*Realiza un
                          commit*/

    exit(0);                /*Termina el programa*/
    }


/* Ejemplo 2 Programa en Pro*C que crea una tabla */


 #include <stdio.h>

 EXEC SQL BEGIN DECLARE SECTION;   /*Inicia seccion de
                    declaracion de variables*/

 VARCHAR uid[20];
 VARCHAR pwd[20];

 EXEC SQL END DECLARE SECTION;    /*Termina Seccion de
                    declaracion */

 EXEC SQL INCLUDE SQLCA;

 main(argc,argv)
  int argc;
  char *argv[];
  {
   strcpy(uid.arr,aregv[1]);
   uid.len=strlen(uid.arr);
   strcpy(pwd.arr,argv[2]);
   pwd.len=strlen(pwd.arr);

   EXEC SQL CONNECT :uid  IDENTIFIED BY :pwd;

   printf(" Conectado a ORACLE usuario: %s\n",uid.arr);

   EXEC SQL CREATE TABLE EMPLEADO
         ( FICHA   number(6),
           NOMBRE  char(15),
           DEPTO   char(15),
           FECHA   date,
           SALARIO  number(10,2),
           COMISION number(10,2));

   printf("Tabla creada \n");
```

```
    EXEC SQL COMMIT WORK RELEASE;
    exit(0);
    }


/* Ejemplo 3 Programa en Pro*C que inserta datos en
   la tabla EMPLEADO */

 #include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;

 VARCHAR uid[20];
 VARCHAR pwd[20];
 VARCHAR nombre[15];
 VARCHAR depto[15];
 VARCHAR fecha[15];
 int ficha;
 int respuesta;
 float salario;
 float comision;

 EXEC SQL END DECLARE SECTION;

 EXEC SQL INCLUDE SQLCA;

 main(argc,argv)
  int argc;
  char *argv[];
  {
   strcpy(uid.arr,aregv[1]);
   uid.len=strlen(uid.arr);
   strcpy(pwd.arr,argv[2]);
   pwd.len=strlen(pwd.arr);

   EXEC SQL CONNECT :uid  IDENTIFIED BY :pwd;

   printf(" Conectado a ORACLE usuario: %s\n",uid.arr);

   while(1)
    {
     printf("Dar la ficha del empleado (o 0 para salir): ");
     respuesta = scanf("%d",&ficha);
     if(respuesta == EOF || respuesta == 0 || ficha == 0)
       break;

     printf("Dar el nombre del empleado:  ");
     scanf("%s",nombre.arr);
     nombre.len=strlen(nombre.arr);
```

```
    printf("Dar el departamento del empleado: ");
    scanf("%s",depto.arr);
    depto.len= strlen(depto.arr);

    printf("Dar la fecha de contratacion: ");
    scanf("%s",fecha.arr);
    fecha.len=strlen(fecha.arr);

    printf("Dar el salario del empleado: ");
    scanf("%f",&salario);

    printf("Dar la comision del empleado: ");
    scanf("%f",&comision);


    /* Inserta los datos en la tabla EMPLEADO */

    EXEC SQL INSERT INTO EMPLEADO
          VALUES(:ficha,:nombre,:depto,:fecha,
              :salario,:comision);
    EXEC SQL COMMIT WORK;

    printf(" Empleado %s insertado \n\n",nombre.arr);
    }

  EXEC SQL COMMIT WORK RELEASE;
  exit(0);
  }
```

/* Ejemplo 4 Pro*C que entrega un reporte de la tabla EMPLEADO. Despliega el nombre, el departamento y el salario, ordenado por departamento y salario. El departamento debe ser dado como dato desde el teclado . En este ejemplo se utiliza un cursor*/

```
#include <stdio.h>

FILE *salida;

EXEC SQL BEGIN DECLARE SECTION

  VARCHAR uid[20];
  VARCHAR pwd[20];
  int ficha;
  VARCHAR nombre[15];
  VARCHAR depto[15];
  VARCHAR fecha[15];
  float salario,comision;
  int respuesta;

EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL INCLUDE SQLCA;

main(argc,argv)
  int argc;
  char *argv[];
  {
   strcpy(uid.arr,argv[1]);
   uid.len=strlen(uid.arr);
   strcpy(pwd.arr,argv[2]);
   pwd.len=strlen(pwd.arr);

   /* Cuando encuentra un error va a la etiqueta
   error_oracle*/

   EXEC SQL WHENEVER SQLERROR GOTO error_oracle;

   EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

   printf(" Conectado a ORACLE usuario: %s\n",uid.arr);

   /* Abre el archivo de salida reporte.lis */

   if((salida=fopen("reporte.lis","w"))==NULL)
    {
     printf("NO PUEDO ABRIR ARCHIVO DE SALIDA reporte.lis\n");
     goto error_oracle;
    }

   printf("Dar el departamento del empleado:  ");
   scanf("%s",depto.arr);
   depto.len=strlen(depto.arr);

   /* Declara el nombre del cursor como cursor1 */

   EXEC SQL DECLARE cursor1 CURSOR FOR
      SELECT NOMBRE,DEPTO,SALARIO
      FROM EMPLEADO
      WHERE DEPTO = :depto
      ORDER BY DEPTO,SALARIO;

   EXEC SQL OPEN cursor1;      /* Abre el cursor */

   /* Cuando no encuentra mas datos detiene termina el
   programa */

   EXEC SQL WHENEVER NOT FOUND STOP;

   fprintf(salida,"NOMBRE DEL EMPLEADO \t\t DEPARTAMENTO \t\t SALARIO \n\n");
   for(;;)
     {
      /*Regresa los renglones del cursor y los introduce
      en las variables host */
```

```
    EXEC SQL FETCH cursor1 INTO :nombre,:depto,
                        :salario;
    fprintf(salida,"%-10s \t\t\t %-10s \t\t %6.2f \n",
          nombre.arr,depto.arr,salario);
    }

  EXEC SQL CLOSE CURSOR curso1;    /*Cierra el cursor*/
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  EXEC SQL COMMIT WORK RELEASE;
  exit(0);

  error_oracle:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("Error %s \n",sqlca.sqlerrm.sqlerrmc);
    EXEC SQL ROLLBACK WORK
    fclose(salida);
    exit(0)
}
```

El reporte quedaria como:

NOMBRE DEL EMPLEADO	DEPARTAMENTO	SALARIO
Armando	Informatica	1200.00
Moises	Informatica	1400.00
Eduardo	Ventas	800.50

III.4 Detección y Corrección de Errores.

Un programa de aplicación es más seguro si contiene rutinas para el manejo y corrección de errores.

Una parte importante de todo programa debe ser dedicada al menajo de errores. El principal beneficio de realizar un manejo de errores es que le permite al programa continuar con su operación cuando se presente un error.

Los errores se presentan por fallas en el diseño, por código erroneo, fallas de hardware, por conectarse con un usuario inválido, etc.

Nunca es posible anticiparse a todos los posibles errores, pero se debe realizar un manejo de los errores que se puedan presentar para conocer su significado y poder tomar una desición en caso de que se presenten.

Para el precompilador Pro*C de ORACLE el manejo de errores significa detección y recuperación de argumentos de SQL que tengan un error en su ejecución.

Se puede realizar también el manejo de mensajes de advertencia, como "valor truncado" y cambios de estado como "fin de datos". Esto es especialmente importante cuando se raliza un chequeo de errores y advertencias después de la ejecución de cualquier argumento de SQL. Es decir, un argumento INSERT,UPDATE o DELETE pueden fallar después de procesar todos los datos elegibles en una tabla, y al realizar un chequeo de los mensajes de error o advertencia se toma una desición del camino a seguir dependiendo de estos mensajes.

ORACLE modifica el SQLCA (Area de Comunicaciones de SQL) después de ejecutar un argumento de SQL. ORACLE regresa el código almacenado en el SQLCA y de acuerdo a este código en el programa se puede determinar el resultado del argumento SQL.

El resultado del argumento de SQL se puede revisar de dos formas:

- con un chequeo implícito a través del argumento WHENEVER
- con un chequeo explícito de las variables del SQLCA

El argumento WHENEVER realiza un chequeo de las variables del SQLCA. Generalmente se utiliza el argumento WHENEVER porque es más fácil de usar y es más portable.

Cuando se necesita más información sobre los errores en tiempo de ejecución que la que proporciona el SQLCA puede usarse el ORACA (Area de Comunicación ORACLE). El ORACA contiene información hacerca de .los argumento de SQL, grupos de opciones, etc.

Todos los argumentos ejecutables de SQL regresan un código de estado que es almacenado en la variable SQLCODE y la cual es leida implícitamente con el argumento WHENEVER.

Un código de estado cero significa que ORACLE ejecutó el argumento sin detectar un error o una advertencia. Un código de estado positivo significa que ORACLE ejecutó el argumento pero detecto una advertencia. Y, un código de estado negativo significa que ORACLE no pudo ejecutar el argumento porque encontró un error.

Los mensajes de error de ORACLE están compuestos de los siguientes elementos:

Banderas de Advertencia. Las banderas de advertencia son almacenadas en las variables de SQLCA, de la SQLWARN(1) hasta la SQLWARN(8), la cual puede ser leída explícitamente o implícitamente. Estas banderas son usadas para condiciones en tiempo de ejecución y que ORACLE no las considera como errores. Por ejemplo, ORACLE asigna una bandera de advertencia cuando se presenta un valor truncado de una columna a una variable host.

Conteo de Renglones Procesados. El número de renglones procesados por un argumento SQL ejecutable es almacenado en la variable SQLERRD(3) de SQLCA. Estrictamente hablando esta variable no significa que exista un error, pero puede ser de gran ayuda para realizar ciertas tareas en el programa. Por ejemplo, si se espera que de una tabla se borren 10 renglones, y despues de realizar el borrado, y al revisar la variable SQLERRD(3) se observa que en realidad se borraron 75 renglones, se pueden tomar diferentes alternativas, como dar un rollback al borrado o revisar la cláusula WHERE para determinar si esta correcta la condición para el borrado.

Texto del Mensaje de Error. El código y el mensaje de error de ORACLE son almacenados en la variable SQLERRMC. Al menos los primeros 70 caracteres del texto son almacenados en esta variable.

Todos los programas en Pro*C deben utilizar el SQLCA para proveer un análisis y manejo de errores. Los campos del SQLCA contienen errores, advertencias y estados de información actualizados por ORACLE siempre que un argumento de SQL es ejecutado. Así, el SQLCA siempre refleja el resultado de la última operación de SQL. Para determinar el resultado se revisan las variables en el SQLCA, estas variables son:

SQLCAID. Este campo tipo caracter es inicializado por SQLCA para identificar el Area de Comunicaciones de SQL.

SQLABC. Es un campo tipo entero donde se guarda la longitud, en bytes, de la estructura de datos de SQLCA.

SQLCODE. Es un campo tipo caracter y es donde se guarda el código de estado del argumento SQL más recientemente ejecutado. Este código de estado, que indica el resultado de la operación de SQL puede ser cualquiera de los siguientes:

- 0 significa que ORACLE ejecuto el argumento sin detectar un error o una advertencia.
- >0 significa que ORACLE ejecuto el argumento pwero detecto una advertencia. Esto ocurre cuando ORACLE no puede encontrar algún renglón que cumpla con la condición especificada en la cláusula WHERE, o cuando un SELECT INTO o un FETCH no regresan ningún renglón.
- <0 Significa que ORACLE no pudo ejecutar el argumento porque hubo un error, ya sea en la base de datos, en el sistema, en la red o en la aplicación. Como puede ser un error grave, cuando ocurre, por lo regular se debe dar un rollback a la transacción.

SQLERRM.Es un subregistro que contiene los siguientes dos campos:
- SQLERRML. Es un campo entero que almacena la longitud del texto de mensaje almacenado en SQLERRMC.
- SQLERRMC. Es un campo tipo caracter que almacena el mensaje correspondiente al código de error almacenado en SQLCODE. Este campo puede almacenar hasta 70 caracteres.

SQLERRD.Es un arreglo tipo entero de 6 elementos, de los cuales solo el SQLERRD(3) esta disponible, y los restantes estan reservados para uso futuro.
- SQLERRD(3). Este campo almacena el número de renglones procesados por el argumento de SQL más recientemente ejecutado. El contador es cero después del argumento OPEN y se incrementa al ejecutar el FETCH. Para un argumento INSERT, UPDATE, DELETE o SELECT INTO, el contador refleja el número de renglones procesados

SQLWARN. Es un arreglo tipo caracter de 8 elementos, el cual asigna una 'W' como bandera a las variables. Por ejemplo, una bandera de advertencia 'W' se asigna a una variable cuando ORACLE asigna un valor truncado a una variable host de salida. Los campos de esta variable son los siguientes:
- SQLWARN(1). Esta bandera se activa cuando alguna otra bandera de advertencia es activada.
- SQLWARN(2). Es una bandera que se activa si un valor de una columna es truncado y se le asigna a una variable host de salida. Esto se aplica a datos tipo caracter. ORACLE trunca ciertos datos numéricos sin activar esta bandera de advertencia.
- SQLWARN(3). Esta bandera no esta actualmente en uso.
- SQLWARN(4). Esta bandera se activa si el número de columnas de una lista de selección no es igual al número de variables host de salida de la cláusula INTO.
- SQLWARN(5). Esta bandera se activa si todos los renglones de una tabla son procesados por un argumento UPDATE o DELETE sin una cláusula WHERE. Una argumento UPDATE o DELETE es llamado incondicional si no existe una condición que restrinja el número de renglones procesados. Por lo regular aplicar un UPDATE o DELETE a toda la tabla es inusual, por lo que ORACLE activa una bandera de advertencia. Con esta información se puede efectuar un rollback a la transacción si es necesario.

Las bandera SQLWARN(6), SQLWARN(7) y SQLWARN(8) no estan actualmente disponibles.

Por default, el precompilador Pro*C ignora condiciones de error y advertencia, y continúa procesando si es posible. Para una revisión automática y un posible manejo de errores, es necesario utilizar el argumento WHENEVER.

Con el argumento WHENEVER es posible detectar condiciones anormales y tomar acciones apropiadas basadas en condiciones específicas. El argumento WHENEVER tiene la siguiente sintáxis:

EXEC SQL WHENEVER condición acción;

El argumento WHENEVER permite tomar diferentes acciones cuando ORACLE detecta un error, una advertencia, o una condición "not found". Esta acciones implican detener el programa, ir a un argumento etiquetado, o continuar con la ejecución del programa si es posible.

ORACLE realiza un chequeo automático del SQLCA para cualquiera de las siguientes condiciones:

- SQLERROR. Se activa cuando el sqlca.sqlcode es negativo
- SQLWARNING. Se activa cuando el sqlca.warn[0] es 'W'.
- NOT FOUND. Es regresado cuando ORACLE no encuentra renglones que cumplan con la condición del WHERE.

Cuando ORACLE detecta alguna de estas condiciones el programa toma alguna de las siguientes acciones:

- que vaya a la etiqueta que *cierre* el cursor (close_cursor) si no encuentra datos que cumplan con las condiciones del WHERE.
- que continúe con el siguiente argumento si detecta una bandera de advertencia.
- que vaya a la etiqueta que contiene la rutina para el manejo de errores (rutina_error).

En el cuerpo del programa se deben tener los siguientes argumentos despúes del argumento ejecutable.

```
...
EXEC SQL WHENEVER NOT FOUND GOTO close_cursor;
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER SQLERROR GOTO rutina_error;
...
```

También es posible utilizar el argumento WHENEVER para detectar un error al conectarse a la base de datos, por ejemplo:

```
...
EXEC SQL WHENEVER SQLERROR GOTO error_conexion;
EXEC SQL CONNECT :username IDENTIFIED BY :password;
...

error_conexion:
    stop;
    exit;
```

Como el argumento WHENEVER es un argumento declarativo, su alcance es determinado por su posición física dentro del programa, no por su posición lógica, es decir, no por el flujo lógico del programa. El argumento WHENEVER va al principio del primer argumento ejecutable sobre el cual se quiera realizar algun chequeo de error.

Un argumento WHENEVER tiene efecto hasta que se encuentre otro argumento WHENEVER que realize un chequeo por la misma condición.

En el siguiente ejemplo, el primer argumento WHENEVER SQLERROR se aplica solo al CONNECT. El segundo argumento WHENEVER SQLERROR se aplica tanto al UPDATE como al DROP, sin importar el GOTO del paso1 al psao3.

```
paso1:
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL CONNECT :username IDENTIFIED BY :password;
...

GOTO paso3;

paso2:
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL UPDATE EMPLEADO SET SALARIO = salario*1.1;
...

paso3:
EXEC SQL DROP TABLE EMPLEADO1;
...
```

El programa debe estar preparado para una condición de fin de datos cuando se utiliza un cursor. Si el FETCH no regresa datos, el programa deberá saltar a la etiqueta especificada donde se utiliza el argumento CLOSE con lo cual se cierra el cursor.

```
EXEC SQL WHENEVER NOT FOUND GOTO cierra_cursor;
...


cierra_cursor:
...
EXEC SQL CLOSE cierra_cursor;
```

Cuando se necesita más información acerca de los errores en tiempo de ejecución y los cambios de estado , se utilizan las variables de ORACA.

El ORACA contiene grupos de opciones, estadísticas de sistemas, el nombre del archivo donde ocurrio el error, localización del error en el archivo, etc.

Las variables del ORACA a través de las cuales se realiza un diagnóstico son:

ORACAID. Es un campo tipo caracter que asigna un identificador al Area de Comunicaciones del ORACLE (ORACA).

ORACCHF. Si se define la variable ORADBGF, esta variable habilita el chequeo de la consistencia del cursor antes de cualquier operación de éste.
Estas variables pueden tomar los siguientes valores:
0 . Deshabilita el chequeo de la consistencia del cursor (valor por default)
1 . Habilita el chequeo de la consistencia del cursor.

ORADBGF. Esta variable permite escoger todas las opciones de depuración de errores y toma uno de los siguientes valores:
0 . Deshabilita las operaciones de DEBUG (opción por default)
1 . Habilita todas las operaciones de DEBUG.

ORATXTF. Esta variable nos permite especificar en que momento guardar el texto del argumento de SQL que se este manajando. Puede tomar los siguientes valores:

0 . Nunca guarda el texto del argumento de SQL (valor por default)
1 . Guarda el texto del argumento de SQL solo en la variable SQLERROR.
2 . Guarda el texto del argumento de SQL en las variables SQLERROR o SQLWARNING.
3 . Siempre guarda el texto del argumento de SQL que se este procesando.

El texto del argumento de SQL es guardado en un registro del ORACLA llamado ORASTXT.

ORAHOC. Es un campo tipo entero que guarda el máximo número de cursores que pueden ser abiertos (MAXOPENCURSORS) durante la ejecución del programa.

ORAMOC. Registra el máximo número de cursores abiertos requeridos por el programa. Este número puede ser tan grande como el ORAHOC.

ORANPR. Es un campo tipo entero que registra el número de argumentos de SQL a los que se requiere hacer un análisis gramatical (parseo) de parte del programa.

ORANEXT. Registra el número de argumentos del SQL ejecutados por el programa.

En resumen, se puede realizar un a revisión de errores con las variables del SQLCA, el valor de sus variables. O también utilizando el argumento WHENEVER preguntando por una condición de error.

A continuación, algunos ejemplos donde se muestra el uso de variables que nos ayudan a detectar y corregir errores en programas Pro*C.

```
/* Ejemplo 5  modificación al ejemplo 4. En caso de existir un error despliega la descripción del error, el
argumento de SQL  donde ocurrio el error, la linea del error y el identificador de proceso que se le asigna
al programa */

#include <stdio.h>

FILE *salida;
```

```
EXEC SQL BEGIN DECLARE SECTION

  VARCHAR uid[20];
  VARCHAR pwd[20];
  int ficha;
  VARCHAR nombre[15];
  VARCHAR depto[15];
  VARCHAR fecha[15];
  float salario,comision;
  int respuesta;
  int a;

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

main(argc,argv)
  int argc;
  char *argv[];
   {
    strcpy(uid.arr,argv[1]);
    uid.len=strlen(uid.arr);
    strcpy(pwd.arr,argv[2]);
    pwd.len=strlen(pwd.arr);

    EXEC SQL WHENEVER SQLERROR GOTO error_oracle:

    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

    printf(" Conectado a ORACLE usuario: %s\n",uid.arr);

    EXEC ORACLE OPTION (ORACA=YES);

    oraca.orahchf=1;     /*Habilita variables para el */
    oraca.oradbgf=1;     /*manejo de errores */
    oraca.oracchf=1;
    oraca.oraslxtf=3;

    if((salida=fopen("reporte.lis","w"))==NULL)
     {
      printf("NO PUEDO ABRIR ARCHIVO DE SALIDA reporte.lis\n");
      goto error_oracle;
     }

    printf("Dar el departamento del empleado:  ");
    scanf("%s",depto.arr);
    depto.len=strlen(depto.arr);

    EXEC SQL DECLARE cursor1 CURSOR FOR
       SELECT NOMBRE,DEPTO,SALARIO
       FROM EMPLEADO
```

```
    WHERE DEPTO = :depto
    ORDER BY DEPTO,SALARIO;

  EXEC SQL OPEN cursor1;


  EXEC SQL WHENEVER NOT FOUND no_mas;

  fprintf(salida,"NOMBRE DEL EMPLEADO \t\t DEPARTAMENTO \t\t SALARIO \n\n");
  for(;;)
    {
    EXEC SQL FETCH cursor1 INTO :nombre,:depto,
                        :salario,:a;

    /* Se agrego la variable "a" para provocar el error */

    fprintf(salida,"%-10s \t\t\t %-10s \t\t %6.2f \n",
          nombre.arr,depto.arr,salario);
    }

  no_mas:
   EXEC SQL CLOSE CURSOR curso1;
   EXEC SQL WHENEVER SQLERROR CONTINUE;
   EXEC SQL COMMIT WORK RELEASE;
   exit(0);

  error_oracle:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]='\0';
    oraca.orastxt.orastxtc[oraca.orastxt.orastxtl]='\0';
    oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml]='\0';
    printf("\n\nError encontrado %s \n\n",
          sqlca.sqlerrm.sqlerrmc);
    printf("Argumento de SQL con error:\n \"%s...\"\n\n",
          oraca.orastxt.orastxtc);
    printf("en la linea %d del programa con id: %s.\n",
          oraca.orasInr,oraca.orasfnm.orasfnmc);
    EXEC SQL ROLLBACK WORK
    fclose(salida);
    exit(0)
  }
```

Al agregar en el FETCH la variable a que no ha sido seleccionada, se provoca un error que debe ser detectado por el WHENEVER desplegando en la pantalla el siguiente mensaje

```
Error encontrado ORA-01007: variable not in select list

Argumento SQL con error:
```

"SELECT NOMBRE,DEPTO,SALARIO FROM EMPLEADO WHERE DEPTO=:b1 ORDER BY DEP..."

en la linea 77 del programa con id: 18269.pc.

El número del error ORA-01007 es el código del error y la descripción que despliega nos dice el error que se presento.

```
/* Ejemplo 6 Modificacion al ejemplo 5. Entrega el reporte de la tabla EMPLEADO.(reporte.lis)
Si no se presenta un error manda a un archivo de salida aparte (resultado.lis) el último argumento de
SQL ejecutado por el programa y el número de renglones procesados.
En caso de existir un error manda al archivo de salida (resultado.lis) el argumento de SQL donde se
presento el error, la linea del error y el identificador de proceso que se le asigna al programa */
#include <stdio.h>

FILE *salida;
FILE *estadis;

EXEC SQL BEGIN DECLARE SECTION

  VARCHAR uid[20];
  VARCHAR pwd[20];
  int ficha;
  VARCHAR nombre[15];
  VARCHAR depto[15];
  VARCHAR fecha[15];
  float salario,comision;
  int respuesta;

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

main(argc,argv)
  int argc;
  char *argv[];
  {
   strcpy(uid.arr,argv[1]);
   uid.len=strlen(uid.arr);
   strcpy(pwd.arr,argv[2]);
   pwd.len=strlen(pwd.arr);

   EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

   EXEC ORACLE OPTION (ORACA=YES);

   oraca.orahchf=1;
   oraca.oradbgf=1;
```

```
    oraca.oracchf=1;
    oraca.orastxtf=3;

    printf(" Conectado a ORACLE usuario: %s\n",uid.arr);

    if((salida=fopen("reporte.lis","w"))==NULL)
      {
       printf("NO PUEDO ABRIR ARCHIVO DE SALIDA reporte.lis\n");
       goto error_oracle;
      }

    if((salida=fopen("resultado.lis","w"))==NULL)
      {
       printf("NO PUEDO ABRIR ARCHIVO DE SALIDA resultado.lis\n");
       goto error_oracle;
      }

    printf("Dar el departamento del empleado:  ");
    scanf("%s",depto.arr);
    depto.len=strlen(depto.arr);

    EXEC SQL DECLARE cursor1 CURSOR FOR
        SELECT NOMBRE,DEPTO,SALARIO
        FROM EMPLEADO
        WHERE DEPTO = :depto
        ORDER BY DEPTO,SALARIO;

    EXEC SQL OPEN cursor1;

    EXEC SQL WHENEVER NOT FOUND no_mas;

    fprintf(salida,"NOMBRE DEL EMPLEADO \t\t DEPARTAMENTO \t\t SALARIO \n\n");
    for(;;)
      {
       EXEC SQL FETCH cursor1 INTO :nombre,:depto,
                                   :salario;
       fprintf(salida,"%-10s \t\t\t %-10s \t\t %6.2f \n",
            nombre.arr,depto.arr,salario);
      }
no_mas:
    fprintf(estadis,"\n\nUltimo argumento de SQl ejecutado:\n%s \n",oraca.orastxt.orastxtc);
    fprintf(estadis,"Consulta %d renglon%s \n",sqlca.sqlerrd[2],(sqlca.sqlerrd[2] == 1) ? "" : "es");

    EXEC SQL CLOSE CURSOR curso1;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL COMMIT WORK RELEASE;
    exit(0);

  error_oracle:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]='\0';
    oraca.orastxt.orastxtc[oraca.orastxt.orastxtl]='\0';
```

```
oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml]='\0';
fprintf("\n\nError encontrado %s \n\n",
    sqlca.sqlerrm.sqlerrmc);
fprintf("Argumento de SQL con error:\n \"%s...\"\n\n",
    oraca.orastxt.orastxtc);
fprintf("en la linea %d del programa con id: %s.\n",          oraca.orasInr,oraca.orasfnm.orasfnmc);
EXEC SQL ROLLBACK WORK
fclose(salida);
exit(0)

  }
```

Cuando no se presenta ningún error, el programa deja de salida el archivo reporte.lis con el resultado de la selección de datos de la tabla EMPLEADO. Además crea el archivo resultado.lis con la siguiente información:

```
Ultimo argumento de SQL ejecutado
"SELECT NOMBRE,DEPTO,SALARIO FROM EMPLEADO WHERE DEPTO=:b1 ORDER BY DEP..."

Consulta 3 renglones
```

Si se presenta algún error no manda nada al archivo reporte.lis y en el archivo resultado.lis manda lo siguiente:

```
Error encontrado ORA-01007: variable not in select list

Argumento SQL con error:
 "SELECT NOMBRE,DEPTO,SALARIO FROM EMPLEADO WHERE DEPTO=:b1 ORDER BY DEP..."

en la linea 77 del programa con id: 18269.pc.
```

III.5 Requisitos para Ejecutar un Pro*C

Durante la precompilación, el Pro*C genera una secuencia de código de lenguaje C que reemplaza los argumentos de SQL que se encuentran en el programa. El Pro*C puede desplegar mensajes de error y tomar alguna acción de acuerdo al tipo de error detectado.

El principal requisito para correr un Pro*C es tener el archivo de entrada para ser precompilado. El unico argumento requerido es el nombre del archivo, esto asumiendo que se está usando la convensión normal para los nombres de los archivos, es decir, que la extención del programa es *.pc*. Si se utiliza otra extensión se debe especificar el lenguaje host (C) que se esta empleando.

Si el sistema operativo utiliza directorios o rutas (paths) se debe asegurar que estos esten disponibles, que existan y que se tengan privilegios de acceso y escritura sobre ellos.

Pro*C crea archivos temporales durante su ejecución, esta ejecución sera abortada si no puede abrir los archivos temporales. Por ejemplo, si existe poco espacio en el directorio donde se esta llevando a cabo la precompilación se pueden tener problemas para abrir los archivos temporales.

El comando para desplegar las opciones de precompilación es PCC, con lo cual se desplegaran los comandos, su sintáxis y los valores por default. En general su sintáxis es :
PCC INAME = filename (opción= valor (opción = valor))

Como se dijo anteriormente solo un argumento es obligatorio: INAME = filename, si no ese esta usando la extención ".pc" se debe usar el argumento HOST = C.

La figura III.2 muestra las extenciones por default para diferentes sistemas operativos.

SISTEMA OPERATIVO	EXTENCION DE ARCHIVO DE ENTRADA	EXTENCION DE ARCHIVO DE SALIDA
VAX/VMS	.pc	.c
IBM/VM/CMS	.csql	.c
UNIX	.pc	.c

Figura III.2

Pro*C tiene varias opciones que pueden ser utilizadas mientras se esta realizando el proceso de precompilación, por ejemplo:

AREASIZE. Este parámetro especifica el tamaño del cursor que es abierto por el programa. Este valor es dado como parámetro también en el archivo INIT.ORA. Si no se especifica un tamaño en especial en el programa toma el que se especifico en el INIT.ORTA. El AREASIZE puede ser utilizado varias veces en un programa para cambiar el tamaño a diferentes cursores. El AREASIZE se especifica en k-bytes y el default es 16.

ERRORS. Esta opción especifica donde se enviaran los mensajes de error. El valor por default es "yes". Las opciones son "yes" y "no". Si la opción es "yes" los errores son enviados a la terminal y a un archivo de salida.

HOST. Esta opción especifica el lenguaje host del programa de entrada, en este caso C. Las opciones son C, cobol y fortran. Esta opción es requerida cuando el archivo de engtrada no tiene extención.

INCLUDE. Esta opción especifica la ruta para los archivos que seran precompilados, por ejemplo:

PCC INAME = abc INCLUDE = xxx/s/b/c.

Por lo genaral no se utiliza este parámetro. También puede especificarse INCLUDE = dirA INCLUDE = dirB etc., lo que significa que se buscara primero en el directorio local, después en dirA, y finalmente en dirB.

LNAME. Esta opción es usada para dar el nombre del archivo de mensajes de salida. Este archivo es escrito en el adirectorio donde se este realizando la precompilación. Por ejemplo:

PCC INAME = prog1 LNAME = salida.lis

MAXLITERAL. Esta opción es usada para declarar ciertas restricciones de compilación con respecto a la máxima longitud de cadenas. El valor por default es 100 para lenguaje C.

MAXOPENCURSORS. Nos permite fijar el número máximo de cursores abiertos simultáneamente en el programa. El máximo valor de MAXOPENCURSORS depende del sistema operativo, por lo regular su valor es de 10.

XREF. Esta opción especifica una sección de referencia que puede ser incluida en el archivo de salida. El valor por default es "yes". En esta sección de referencia se incluyen variables host, nombres de cursores y nombres de argumentos.

CAPITULO IV

Aplicación del Pro*C a un Sistema de Costos

IV.I Características del Sistema

Tomando como ejemplo un sistema de costos, en este capítulo se muestra como se aplica el precompilador Pro*C en programas que realizan procesos propios del sistema y en programas que nos entregan reportes.

Este sistema denominado Sistema de Gastos de Origen (SIGOR) se encuentra en producción en la Gerencia de Ductos y Terminales de PEMEX Gas y Petroquímica Básica, donde se realizó la conversión a Pro*C.

El Sistema de Gastos de Origen (SIGOR) es un subsistema del Sistema Institucional de Contabilidad de PEMEX. Surge de la necesidad de hacer mas eficiente los procedimientos de administración y control interno de la información contable-financiera en cada centro de trabajo de Petróleos Mexicanos.

El objetivo de este sistema es conocer el gasto que se tiene en el centro de trabajo. A través de este sistema se puede saber cuanto gasta el centro de trabajo y cada departamento, y en que fue ese gasto.

El SIGOR presenta una información analítica del origen de los gastos aplicados a las cuentas de mayor operacionales.

Una cuenta de mayor es donde se realiza el registro del activo, pasivo o capital y se clasifican de acuerdo a su operación.

Las cuentas de mayor operacionales que forman parte del SIGOR son las que a continuación se indican:

5206 Gastos en Ventas Nacionales.
5213 Gastos Generales de Administración.
5219 Operación Gasoductos de Transportación.
5220 Gastos de Distribución.
5221 Operación de Telecomunicaciones.
5223 Operación de Servicios Médicos.

En PEMEX todo gasto esta clasificado por concepto de origen. El concepto de origen es una clasificación específica del gasto, es decir, con el concepto de origen se identifica plenamente en que se efectuó el gasto.

Como se menciono anteriormente el SIGOR tiene como fuente de información el Sistema Institucional de Contabilidad de PEMEX, este sistema provee todos los registros de las cuentas de gastos de operación que fueron capturados durante el mes correspondiente.

Requerimientos de Equipo.

Este sistema está instalado en un equipo HP 9000/842, pudiendo ser instalado en equipos NCR o UNISYS. Las necesidades de instalación de este sistema son las siguientes:

- Sistema Operativo UNIX versión 7.08 o posteriores
- Espacio en disco 23 Mbytes (como mínimo).
- Dos cuentas de acceso. Una para el usuario administrador y la otra para el usuario operativo.
- Tener instalada la base de datos ORACLE versión 6.0.33 o posterior.
- Son necesarios dos TABLESPACES, uno para datos y otro para índices. El de datos de 20 Mbytes y el de índices de 5 Mbytes.
- Precompilador Pro*C versión 1.3.18 o posterior
- SQL*Plus versión 3.0.9 o posterior.
- SQL*Forms versión 2.3

El sistema de Gastos de Origen está desarrollado principalmente en Pro*C, estos programas realizan la mayoría de los procesos y todos los reportes.

Además de los programas en Pro*C se realizaron shells, que son programas con comandos de UNIX, estos programas se utilizan principalmente para la ejecución de varios programas en forma encadenada.

Las pantallas de menú fueron realizadas en lenguaje C.

Las pantallas de consulta están desarrolladas en SQL*Forms que es una herramienta de ORACLE . Esta herramienta nos permite realizar consultas de la información contenida en las tablas, además de poder consultar, podemos insertar, modificar y borrar información.

Por ser el tema del presente trabajo todos los Pro*C que realizan los procesos y los reportes son explicados en los puntos IV.2 y IV.3 respectivamente, los shell y programas en C se incluyen en el Apéndice A y en este capítulo solo se explican brevemente.

A continuación se muestra la secuencia que se sigue cada mes para obtener los gastos de cada departamento a través del Sistema de Gastos de Origen.

1. El primer paso es accesar al sistema, para esto se requiere introducir la cuenta (LOGIN) y la clave personal de acceso (PASSWORD), presionando la tecla ENTER. Una vez efectuados estos pasos, aparecerá en la pantalla el menú principal del SIGOR, figura IV.1.

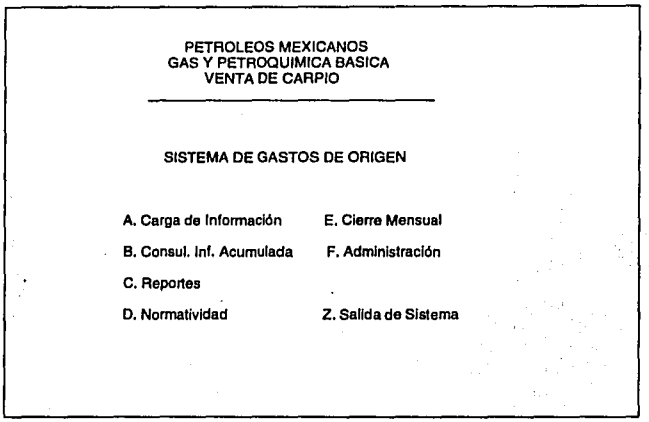

Figura IV.2 Menú Principal del SIGOR

2. Al seleccionar en el menú principal (figura IV.1) la opción "A" se despliega la pantalla de parámetros, figura IV.2, donde se pide al usuario la fecha inicial del mes que se va a procesar con el formato DDMMAA y la fecha final del mes que se va a procesar con el mismo formato.

Con este proceso se carga al sistema todos los gastos de operación registrados durante el mes en el Sistema Institucional de Contabilidad de PEMEX (SIC). En este paso se lleva a cabo la ejecución de la mayoría de los programas de procesos.

3. Concluida la carga del SIC al sistema se realiza la consulta de la información, seleccionando en el menú principal del SIGOR la opción "B" Consulta de Información Acumulada se despliega la pantalla de consultas, figura IV.3.

Esta pantalla nos permite consultar y verificar las cifras por cuenta de mayor de la balanza mensual que corresponda contra las columnas de importe acumulado, importe del mes (por diferencia entre cargo y abono) y total acumulado.

Carga del S.I.C. al Sistema

PARAMETROS

Fecha Inicial del Mes (DDMMAA) ==> DDMMAA

Fecha Final del Mes (DDMMAA) ==> DDMMAA

Figura IV.2 Menú de Parámetros de Selección de Información

PEMEX GOR			SISTEMA DE GASTOS DE ORIGEN CONSULTA DE SALDOS			01-APR-94 GOR5001	
CTA	CTRO	DEPTO	CONOR	RGAS	IMP. ACUM.	IMP. MENS.	TOTAL
				TOTAL			

IV.3. Consulta de saldos

4. Después de verificar que los importes del SIGOR sean los mismos que los de la balanza mensual obtenida del SIC, seleccionar del menú principal la opción "D" que es la pantalla de reportes, figura IV.4.

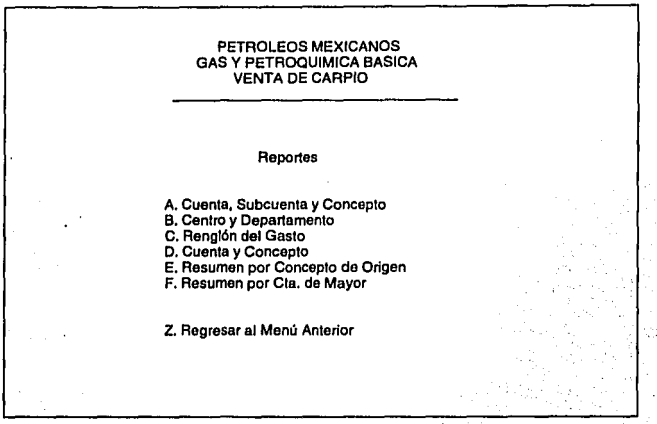

Figura IV.4 Menú de Reportes

En esta opción se pueden obtener los siguientes reportes:

- Reporte por Cuenta Subcuenta y Concepto de Origen.
- Reporte por Centro y Departamento.
- Reporte por Renglón del Gasto.
- Reporte por Cuenta y Concepto de Origen.
- Reporte de Resumen por Concepto de Origen.
- Reporte de Resumen por Cuenta de Mayor.

Para generar cada uno de los reportes el sistema pide los parámetro del mes correpondiente, figura IV.5. En esta pantalla se pide la fecha de cierre del mes en el formato DDMMAA.

5. Antes de ejecutar la opción de Cierre Mensual se debe hacer un respaldo de la información. Para esto se elige la opción de Administración, figura IV.10, y después la opción "B" Exportación de la Información del mes. Un export es una utilería de ORACLE la cual nos permite realizar un respaldo de la información que se tiene en las tablas. Este respaldo se guarda en un archivo físico, este archivo no es

editable y con este se realiza el import, que es una utilería de ORACLE que nos permite recuperar la información de las tablas.

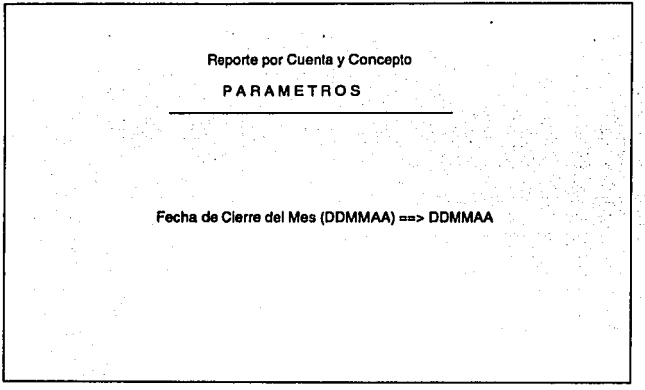

Figura IV.5 Pantalla de parámetros para la generación de reportes

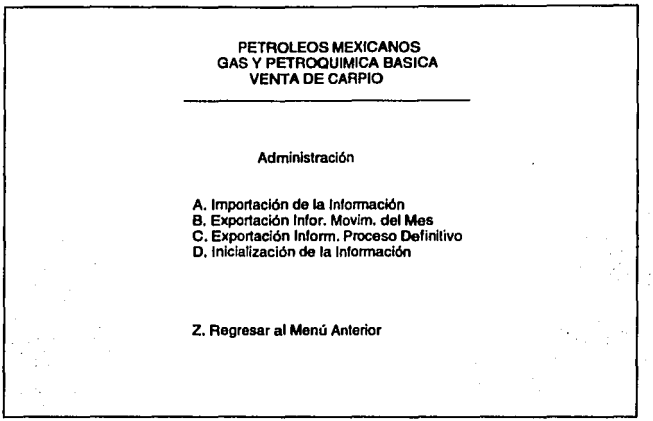

Figura IV.10 Menú de Administración

Teniendo el archivo que se genera con el export (extención ".dmp") se tiene la estructura de las tablas y los datos que contienen. En caso de que se pierda la información de la base de datos, ésta se recupera realizando un import con el archivo ".dmp" del export. El export se realiza sobre la tabla T_GOA que es la que contiene la información acumulada de los meses.

Este respaldo se realiza para tener la información antes de realizar el proceso definitivo por si se debe hacer un reproceso de la información.

Al seleccionar la opción "B" aparece la pantalla de parámetros que pide la fecha de cierre del mes, figura IV.11. Esta pantalla es la misma para las opciones "A", "B" y "C".

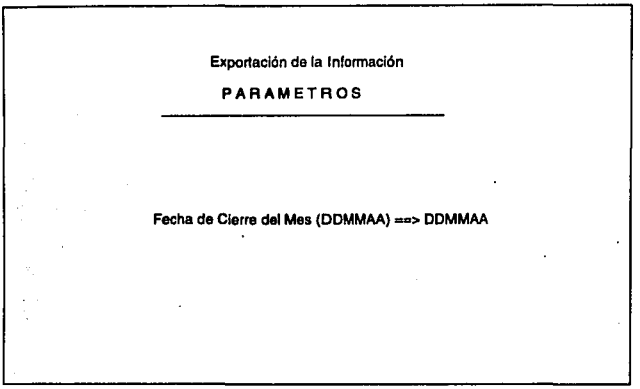

Figura IV.11 Pantalla de Parámetros para el respaldo de Información

6. Cierre Mensual, este proceso deberá ejecutarse después de haber generado todos los reportes y validado las cifras de control. Para realizar el cierre mensual seleccionar la opción "E" del menú principal.

Al seleccionar esta opción solo aparece una pantalla que pregunta si se desea ejecutar el proceso de cierre mensual, figura IV.6.

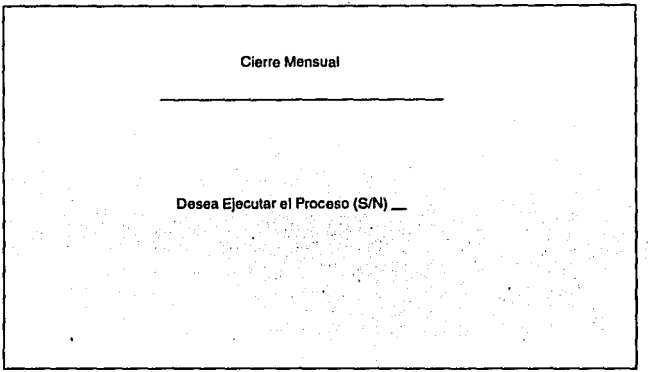

Figura IV.6 Pantalla para ejecutar el proceso de cierre mensual.

Al realizar este proceso se actualizarán los saldos acumulados, incluyendo los movimientos generados durante el mes.

Una vez ejecutado este proceso, el sistema no podrá ejecutar ningún otro, incluyendo la opción de repoortes.

7. Después de realizar el Cierre Mensual se realiza otro respaldo de la base de datos. En la opción "C" del menú de Administración se tiene la opción de Exportación de la Información de Proceso Definitivo. Con esta opción se respaldan la información de la tabla donde se registran los acumulados de la información que es T_GOA, figura IV.13.

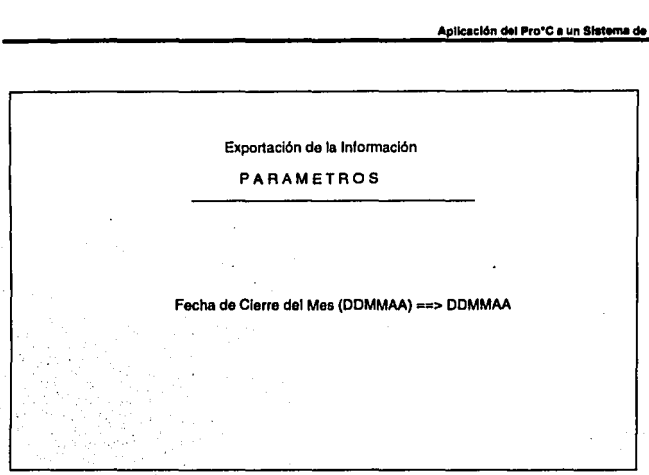

Figura IV.13 Pantalla de parámetros para el respaldo de la información del proceso definitivo

IV.2 Programas de Proceso

Como se explicó en el punto IV.1 después de accesar al sistema de Gastos de Origen, se despliega en pantalla el menú principal,figura IV.1. Este menú es un programa en C llamado menu0.c (ver Apéndice). Al seleccionar la opción "A. Carga de Información", se despliega una pantalla de parámetros donde se le pide al usuario la fecha inicial y final del mes a procesar con el formato DDMMAA, figura IV.2.

Después de dar los parámetros que se piden el realiza la carga de información. El programa menu0.c ejecuta el shell llamado REGBA3001.exe (ver Apéndice).

El shell REGBA3001.exe se utiliza para ejecutar en forma encadenada los programas que realizan la carga de información.

El primer programa que corre este shell es REGBA305.pc el cual carga la información proveniente de SIC a una tabla temporal. El programa REGBA3001.exe le pasa el parámetro del mes y año de proceso en el formato MMAA, por ejemplo para enero el parámetro es 0194.

El programa REGBA305 se muestra a continuación, en el cual se añadieron comentarios en las partes más importantes.

```
/* REGBA305.pc Realiza la carga de la tabla T_129 */
#include <stdio.h>
FILE *mescp;

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR nomarch[9];
VARCHAR cv_us[12];
VARCHAR passwd[12];
VARCHAR scta_db[10],n_doc[9],cv_proy[9],cv_aut[12];
VARCHAR f_venc[10], n_doch[9],cv_proyh[9],cv_auth[12];
VARCHAR f_vench[10], scta_cr[10],mmyy[5],mmyy1[5];
VARCHAR mmyy2[5],afet[4],mm[3],yy[3],mes[2],mes_anio[5];
int cta_dbb,t_doc,r_sec,n_cenafo,n_dep,n_conori,n_cta;
int t_doch,r_sech,n_cenafoh,n_deph,n_conorih;
double i_op,i_oph,cta_crb;
char letra,letrah c;
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;

decodifica()
 {
/*La función decodifica toma el parámetro de entrada mes-año mmyy y lo decodifica obteniendo por
separado el año y el mes. Dependiendo del número de mes le asigna la letra que le corresponde.
Teniendo el año y la literal correspondiente se la asigna a la variable mes_anio*/

  EXEC SQL SELECT substr(:mmyy,1,2)  /*Obtiene el mes*/
        INTO :mm
        FROM DUAL;

  EXEC SQL SELECT substr(:mmyy,3,4)  /*Obtiene el año*/
        INTO :yy
        FROM DUAL;

   if(strcmp(mm.arr,"01")==0)
     strcpy(mes.arr,"A");
   if(strcmp(mm.arr,"02")==0)
     strcpy(mes.arr,"B");
   if(strcmp(mm.arr,"03")==0)
     strcpy(mes.arr,"C");
   if(strcmp(mm.arr,"04")==0)
     strcpy(mes.arr,"D");
   if(strcmp(mm.arr,"05")==0)
     strcpy(mes.arr,"E");
   if(strcmp(mm.arr,"06")==0)
     strcpy(mes.arr,"F");
   if(strcmp(mm.arr,"07")==0)
     strcpy(mes.arr,"G");
   if(strcmp(mm.arr,"08")==0)
     strcpy(mes.arr,"H");
   if(strcmp(mm.arr,"09")==0)
```

```
    strcpy(mes.arr,"J");
  if(strcmp(mm.arr,"10")==0)
    strcpy(mes.arr,"K");
  if(strcmp(mm.arr,"11")==0)
    strcpy(mes.arr,"L");
  if(strcmp(mm.arr,"12")==0)
    strcpy(mes.arr,"M");

  strcpy(mes_anio.arr,yy.arr);
  strcat(mes_anio.arr,mes.arr);
  strcat(mes_anio.arr,"%");
  mes_anio.len=strlen(mes_anio.arr);

}

main(argc,argv)
 int argc;
 char *argv[];
 {
  strcpy(cv_us.arr,argv[1]);
  cv_us.len=strlen(cv_us.arr);
  strcpy(passwd.arr,argv[2]);
  passwd.len=strlen(passwd.arr);
  strcpy(mmyy.arr,argv[3]);
  mmyy.len=strlen(mmyy.arr);

  EXEC SQL WHENEVER SQLERROR GOTO error_oracle;
  EXEC SQL CONNECT :cv_us IDENTIFIED BY :passwd;

  EXEC ORACLE OPTION (ORACA=YES);

   oraca.orahchf=1;    /*Habilita variables para el */
   oraca.oradbgf=1;    /*manejo de errores */
   oraca.oracchf=1;
   oraca.orastxtf=3;

  decodifica();

  EXEC SQL DELETE FROM T_129; /*Borra la tabla T_129*/

  /*Declaración del cursor cargo129*/

  EXEC SQL DECLARE cargo129 CURSOR FOR
     SELECT :mmyy,t_006.cta_dbb,t_006.scta_db,t_006.n_doc,
          t_006.f_doc,t_006.r_sec,t_007.n_cenafo,
          t_007.n_dep,t_007.n_conori,t_007.cv_proy,
          t_007.cv_aut,t_006.i_op,'D',t_007.f_venc
     FROM t_003,t_006,t_007
     WHERE t_006.n_doc like :mes_anio
      AND t_006.cta_dbb in (select n_cta from t_145
                  where n_cont = 9801)
      AND t_006.n_doc = t_003.n_doc
```

```
    AND t_006.r_sec = t_003.r_sec
    AND t_007.n_doc = t_003.n_docori
    AND t_007.r_sec = t_003.r_secori;

EXEC SQL OPEN cargo129;  /*Abre el cursor*/

EXEC SQL WHENEVER NOT FOUND GOTO fin_cargo129;
for(;;)
{
 /*Extrae los datos del cursor y los introduce en las
   variables host*/

 EXEC SQL FETCH cargo129 INTO :mmyy1,:cta_dbb,:scta_db,
      :n_doc,:n_cenafo,:n_dep,:n_conori,:cv_proy,
      :cv_aut,:i_op,:letra,:f_venc;

 mmyy1.arr[mmyy1.len]='\0';
 scta_db.arr[scta_db.len]='\0';
 n_doc.arr[n_doc.len]='\0';
 cv_proy.arr[cv_proy]='\0';
 cv_aut.arr[cv_aut.len]='\0';
 f_venc.arr[f_venc.len]='\0';

 EXEC SQL INSERT INTO T_129 VALUES (:mmyy1,:cta_dbb,
          :scta_db,:n_doc,:t_doc,:r_sec,
          :n_cenafo,:n_dep,:n_conori,:cv_proy,
          :cv_aut,:i_op,'D',:f_venc);
}

fin_cargo129:
  EXEC SQL CLOSE cargo129;  /* Cierra el cursor*/
  EXEC SQL COMMIT;

EXEC SQL DECLARE debe129 CURSOR FOR
   SELECT :mmyy,t_006.cta_crb,t_006.scta_cr,t_006.n_doc,
       t_006.t_doc,t_006.r_sec,t_007.n_cenafo,
       t_007.n_dep,t_007.n_conori,t_007.cv_proy,
       t_007.cv_aut,(t_006.i_op)*(-1),'H',
       t_007.f_venc
   FROM t_003,t_006,t_007
   WHERE t_006.n_doc like :mes_anio
    AND t_006.cta_dbb in (select n_cta from t_145
              where n_cont = 9801)
    AND t_006.n_doc = t_003.n_doc
    AND t_006.r_sec = t_003.r_sec
    AND t_007.n_doc = t_003.n_docori
    AND t_007.r_sec = t_003.r_secori;

EXEC SQL OPEN debe129;  /*Abre el cursor*/

EXEC SQL WHENEVER NOT FOUND GOTO fin_debe129;
for(;;)
```

```
{
  EXEC SQL FETCH cargo129 INTO :mmyy2,:cta_crb,:scta_cr,
       :n_doch,:n_cenafoh,:n_deph,:n_conorih,
       :cv_proyh,:cv_auth,:i_oph,:letrah,:f_vench;

  mmyy2.arr[mmyy2.len]='\0';
  scta_cr.arr[scta_cr.len]='\0';
  n_doch.arr[n_doch.len]='\0';
  cv_proyh.arr[cv_proyh.len]='\0';
  cv_auth.arr[cv_auth.len]='\0';
  f_vench.arr[f_vench.len]='\0';

  EXEC SQL INSERT INTO T_129 VALUES (:mmyy2,:cta_crb,
           :scta_cr,:n_doch,:t_doch,:r_sech,
           :n_cenafoh,:n_deph,:n_conorih,
           :cv_proyh,:cv_auth,:i_oph,'H',
           :f_vench);
}

fin_debe129:
  EXEC SQL CLOSE debe129;  /*Cierra cursor debe129*/
  EXEC SQL COMMIT;
  exit(0);

error_oracle:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]='\0';
    oraca.orastxt.orastxtc[oraca.orastxt.orastxtl]='\0';
    oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml]='\0';
    printf("\n\nError encontrado %s \n\n",
        sqlca.sqlerrm.sqlerrmc);
    printf("Argumento de SQL con error:\n \"%s...\"\n\n",
        oraca.orastxt.orastxtc);
    printf("en la linea %d del programa con id: %s.\n",
        oraca.oraslnr,oraca.orasfnm.orasfnmc);
    EXEC SQL ROLLBACK WORK
    exit(0)

}
```

El programa REGBA305.pc toma el parámetro del año y mes de operacion MMAA, el cual es decodificado de la siguiente forma: se toman los dos primeros dígitos, los cuales corresponden al mes, y de acuerdo a éste se le asigna una letra. Las letras que lo asigna representan a cada uno de los meses y son "A", "B", "C", "D", "E", "F", "G", "H", "J", "K", "L" y "M" (por convensión la letra I es omitida), para enero, febrero y así respectivamente hasta diciembre. Por ejemplo si el mes de operación es 01 (enero) se le asigna la letra "A".

Esta asignación de una letra para cada mes se realiza porque el registro que se lleva en el sistema de contaduría SIC se realiza a través de un identificador que se llama número de documento, por ejemplo,

cuando se captura una cuenta por pagar en el SIC se le asigna un número de documento 94A152961, el cual está compuesto del año 94, el mes "A" y un consecutivo asignado por el sistema 152961.

Teniendo el mes y año decodificado, se procede a seleccionar la información de las tablas de SIC. Estas tablas son la T_003 Movimientos Contable/Presupuestal, T_006 Movimientos Contables (por cuenta y subcuenta) y T_007 Movimientos Presupuestales Se seleccionan los documentos de cargo y de crédito identificandolos por sus cuentas de cargo y crédito (cta_ddd y cta_crb respectivamente).

Una vez seleccionada la información se inserta en la tabla temporal T_129 del SIGOR.

El siguiente programa que es ejecutado por GOR3001.exe es actsic.pc. Este programa corrige el centro de trabajo, el número de departamento, la cuenta de operación y la subcuenta de operación de los documentos que se cargan al SIGOR y que provienen del sistema de contaduría (SIC). Esta corrección se realiza porque algunos documentos de SIC son capturados en el sistema con los datos correctos y en el programa estos datos se comparan con un catálogo. El parámetro que se toma para realizar la comparación es la clave de autorización, esta clave determina que centro, departamento, cuenta y subcuenta le corresponde a cada documento.

El programa actsic.pc selecciona los datos de la tabla T_129, de la clave de autorización toma los últimos cuatro dígitos. Estos cuatro dígitos representan un rango, este rango se compara con la tabla T_CONDEP que es un catálogo de rangos. Con el rango tomado de cada número de documenros se valida el centro, departamento, la cuenta y la subcuenta. En caso de que los datos de la tabla T_129 no coincidan con el catálogo (T_CONDEP) se corrigen los datos.

```
/*actsic.pc  Actualiza la tabla T_129 con catálogos*/

#include<stdio.h>
FILE *mescp;

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR nomarch[9];
VARCHAR cv_us[12];
VARCHAR passwd[12];
VARCHAR n_doc[9],cv_scta[10],n_scta[10],idrenglon[18];
int r_sec,n_cenafo,n_centr,cv_aut,n_cta,n_dep,n_conori;
int n_depeco,n_cta1;
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
int argc;
 char *argv[];
  {
   strcpy(cv_us.arr,argv[1]);
   cv_us.len=strlen(cv_us.arr);
   strcpy(passwd.arr,argv[2]);
   passwd.len=strlen(passwd.arr);

   EXEC SQL WHENEVER SQLERROR GOTO error_oracle;
```

```
EXEC SQL CONNECT :cv_us IDENTIFIED BY :passwd;

EXEC ORACLE OPTION (ORACA=YES);

oraca.orahchf=1;    /*Habilita variables para el */
oraca.oradbgf=1;    /*manejo de errores */
oraca.oracchf=1;
oraca.orastxtf=3;

EXEC SQL DECLARE clave CURSOR FOR
     SELECT n_doc,r_sec,n_cenafo,n_dep,n_cta,cv_scta,
        n_conori,substr(cv_aut,8,11),rowid
     FROM t_129
     ORDER BY n_doc,r_sec,n_dep,n_conori;

EXEC SQL OPEN clave;

EXEC SQL WHENEVER NOT FOUND GOTO fin_clave;
for(;;)
 {
  EXEC SQL FETCH clave INTO :n_doc,:r_sec,:n_cenafo,
       :n_dep,:n_cta,:cv_scta,:n_conori,:cv_aut,
       :idrenglon;

  n_doc.arr[n_doc.len]='\0';
  cv_scta.arr[cv_scta.len]='\0';
  idrenglon.arr[idrenglon.len]='\0';

  EXEC SQL DECLARE depto CURSOR FOR
     SELECT to_number(n_centr),n_depeco,n_cta,n_scta
      FROM t_condep
     WHERE  :cv_aut between ran_ini and ran_fin;

  EXEC SQL OPEN depto;

  EXEC SQL WHENEVER NOT FOUND CONTINUE;

  EXEC SQL FETCH depto INTO :n_centr,:n_depeco,:n_cta,
                  :n_scta;

  n_scta.arr[n_scta.len]='\0';

  EXEC SQL UPDATE T_129 SET
       n_cenafo = :n_centr,
       n_dep = :n_depeco,
       n_cta = :n_cta,
       cv_scta = :n_scta
     WHERE :idrenglon = rowid;
 }

fin_clave:
```

```
    EXEC SQL CLOSE clave;
    EXEC SQL CLOSE depto;
    EXEC SQL COMMIT;
    exit(0);

error_oracle:
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]='\0';
     oraca.orastxt.orastxtc[oraca.orastxt.orastxtl]='\0';
     oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml]='\0';
     printf("\n\nError encontrado %s \n\n",
          sqlca.sqlerrm.sqlerrmc);
     printf("Argumento de SQL con error:\n \"%s...\"\n\n",
          oraca.orastxt.orastxtc);
     printf("en la linea %d del programa con id: %s.\n",
          oraca.orasinr,oraca.orasfnm.orasfnmc);
     EXEC SQL ROLLBACK WORK
     exit(0);
}
```

Después de corregir los documentos que están erroneos el programa GOR3001.exe ejecuta el programa GOR3001.pc.

El programa GOR3001.pc selecciona la información contenida en la tabla T_129 y de acuerdo a la fecha de inicio que se da como parámetro le pone la fecha de movimiento (F_MOV) a cada número de documento.

Los datos seleccionados son insertados en la tabla T_GOR, que contiene la información de todos los meses procesados. Después de ser insertados los datos y por convensión a nivel institucional, los conceptos de origen (N_CONORI) que comiencen con 00 se cambian a 06.

```
/*GOR3001.pc  Inserta de T_129 en T_GOR*/

#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;

VARCHAR cv_us[12];
VARCHAR passwd[12];
VARCHAR n_scta[10],f_mov[10];
VARCHAR fecfin[10];
int n_cta,n_cenafo,n_dep,n_conori,ren_gas;
double i_op;

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
```

```
int argc;
char *argv[];
{
 strcpy(cv_us.arr,argv[1]);
 cv_us.len=strlen(cv_us.arr);
 strcpy(passwd.arr,argv[2]);
 passwd.len=strlen(passwd.arr);
 strcpy(fecfin.arr,argv[3]);
 fecfin.len=strlen(fecfin.arr);

 EXEC SQL WHENEVER SQLERROR GOTO error_oracle;
 EXEC SQL CONNECT :cv_us IDENTIFIED BY :passwd;

 EXEC ORACLE OPTION (ORACA=YES);

  oraca.orahchf=1;     /*Habilita variables para el */
  oraca.oradbgf=1;     /*manejo de errores */
  oraca.oracchf=1;
  oraca.orastxtf=3;

 EXEC SQL DECLARE lee_t_gor CURSOR FOR
   SELECT n_cta, cv_scta, n_cenafo,n_dep,
     substr(lpad(nvl(n_conori,'000000'),6,'0',1,4), null,
     sum(i_op),to_date(:fecfin,'ddmmyy')
   FROM t_129
   GROUP BY n_cta,cv_scta, n_cenafo, n_dep,
     substr(lpad(nvl(n_conori,'000000'),6,'0',),1,4);

 EXEC SQL OPEN lee_t_gor;

 EXEC SQL WHENEVER NOT FOUND GOTO fin_t_gor;

 for(;;)
 {
  EXEC SQL FETCH lee_t_gor INTO :n_cta, :n_scta,
            :n_cenafo, :n_dep, :n_conori,:
            :ren_gas, :i_op, :f_mov;

  n_scta.arr[n_scta.len]='\0';
  f_mov.arr[f_mov.len]='\0';

  EXEC SQL INSERT INTO t_gor VALUES (:n_cta, :n_scta,
            :n_cenafo, :n_dep, :n_conori, :ren_gas,
            :i_op, :f_mov);

 }

 fin_t_gor:
   EXEC SQL UPDATE t_gor
     SET n_conori='06'||substr(n_conori,3,2)
     WHERE substr(n_conori,1,2)='00'
       OR substr(n_conori,1,2)='01';
```

```
    EXEC SQL CLOSE lee_t_gor;
    EXEC SQL COMMIT;
    exit(0);

error_oracle:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]='\0';
    oraca.orastxt.orastxtc[oraca.orastxt.orastxtl]='\0';
    oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml]='\0';
    printf("\n\nError encontrado %s \n\n",
        sqlca.sqlerrm.sqlerrmc);
    printf("Argumento de SQL con error:\n \"%s...\"\n\n",
        oraca.orastxt.orastxtc);
    printf("en la linea %d del programa con id: %s.\n",
        oraca.orasinr,oraca.orasfnm.orasfnmc);
    EXEC SQL ROLLBACK WORK
    exit(0)

}
```

El programa GOR7011 selecciona la información de T_GOR y la compara con la información de T_GOA donde está la información acumulada de los meses anteriores. Si la cuenta, subcuenta, centro, departamento y concepto de origen son iguales, el importe de operación del registro en T_GOA pasa a ser el importe del mes del registro en T_GOA, es decir, los sustituye. En caso de no cumplir con esta condición el registro es insertado en T_GOA.

En el cursor tgoa se seleccionan los datos de T_GOA y se selecciona la pseudocolumna rowid que identifica a cada renglón de una tabla, el rowid es utilizado para identificar el registro que será actualizado

```
/*GOR7011.pc  Compara y actualiza T_GOA*/
#include <stdio.h>
EXEC SQL BEGIN DECLARE SECTION;

VARCHAR cv_us[12];
VARCHAR passwd[12];
VARCHAR n_scta[10],n_conori[5],f_mov[10];
VARCHAR n_scta2[10],n_conori2[5],f_mov2[10];
int n_cta,n_cenafo,n_dep,ren_gas;
double i_op_acu,i_op_mes;
int n_cta2,n_cenafo2,n_dep2,ren_gas2;
double i_op_acu2,i_op_mes2;
VARCHAR fecini[10];
VARCHAR idreng[18];

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
```

```
int argc;
char *argv[];
{
 strcpy(cv_us.arr,argv[1]);
 cv_us.len=strlen(cv_us.arr);
 strcpy(passwd.arr,argv[2]);
 passwd.len=strlen(passwd.arr);
 strcpy(fecini.arr,argv[3]);
 fecini.len=strlen(fecini.arr);

 EXEC SQL WHENEVER SQLERROR GOTO error_oracle;
 EXEC SQL CONNECT :cv_us IDENTIFIED BY :passwd;

 EXEC ORACLE OPTION (ORACA=YES);

 oraca.orahchf=1;     /*Habilita variables para el */
 oraca.oradbgf=1;     /*manejo de errores */
 oraca.oracchf=1;
 oraca.orastxtf=3;

EXEC SQL DECLARE t_goap CURSOR FOR
  SELECT n_cta, n_scta, n_cenafo,n_dep,
       n_conori,i_op_mes
  FROM t_goap;


EXEC SQL DECLARE tgoa CURSOR FOR
  SELECT n_cta, n_scta, n_cenafo,n_dep,
       n_conori,rowid

EXEC SQL OPEN tgoap;

EXEC SQL WHENEVER NOT FOUND GOTO fin_tgoap;

for(;;)
 {
 EXEC SQL FETCH tgoap INTO :n_cta, :n_scta,
          :n_cenafo, :n_dep, :n_conori,
          :i_op_mes;

 n_scta.arr[n_scta.len]='\0';
 n_conori.arr[n_conori.len]='\0';

 EXEC SQL OPEN tgoa;
 EXEC SQL WHENEVER NOT FOUND CONTINUE;
 EXEC SQL FETCH tgoa INTO :n_cta2, :n_scta2,
          :n_cenafo2, :n_dep2, :n_conori2,
          :i_op_mes2,:idreng;

 n_scta2.arr[n_scta2.len]='\0';
 n_conori2.arr[n_conori2.len]='\0';
 idreng.arr[idreng.len]='\0';
```

```
    if(n_cta==n_cta2 && (strcmp(n_scta.arr,n_scta2.arr)==0)
     && n_cenafo==n_cenafo2 && n_dep==n_dep2 &&
     (strcmp(n_conri.arr,n_conori2.arr)==0)
     {
      EXEC SQL UPDATE t_goa
          SET i_op_mes = :i_op_mes
          WHERE :idreng = rowid;
     }
     else
      {
       EXEC SQL INSERT INTO t_goa VALUES (:n_cta, :n_scta,
              :n_cenafo, :n_dep, :n_conori, null,
              0, :i_op_mes, to_date(:fecini,'ddmmyy'));
      }
   }

   fin_tgoap:
     EXEC SQL CLOSE tgoap;
     EXEC SQL CLOSE tgoa;
     EXEC SQL COMMIT;
     exit(0);

error_oracle:
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]='\0';
     oraca.orastxt.orastxtc[oraca.orastxt.orastxtl]='\0';
     oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml]='\0';
     printf("\n\nError encontrado %s \n\n",
          sqlca.sqlerrm.sqlerrmc);
     printf("Argumento de SQL con error:\n \"%s...\"\n\n",
          oraca.orastxt.orastxtc);
     printf("en la linea %d del programa con id: %s.\n",
          oraca.orasinr,oraca.orasfnm.orasfnmc);
     EXEC SQL ROLLBACK WORK
     exit(0)
}
```

Después de ejecutar el programa REGBA7011.pc se tiene la información de los movimientos del mes y de los movimientos acumulados. En este momento es cuando se pueden obtener los reportes, los cuales se explican en el punto IV.3.

Al obtener los reportes y verificar que la información está correcta se procede a realizar el cierre mensual. Con lo que se corre el programa GOR7002.pc. Al realizar el cierre mensual los saldos del mes se acumulan, dejando solo los saldos acumulados.

```
/*GOR7002.pc. Realiza el cierre y acumulación de saldos*/
#include <stdio.h>

EXEC SQL BEGIN DECLARE SECTION;
int xncta,xncenafo,xndep,encta,encenafo,endep;
```

```
VARCHAR xnscta[10],enscta[10],xconori[7],econori[7];
VARCHAR xfmov[15];
double xiopacu,xiopmes;
VARCHAR cv_us[12];
VARCHAR passwd[12];
VARCHAR numreg[18];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;

main(argc,argv)
 int argc;
 char *argv[];
 {
  strcpy(cv_us.arr,argv[1]);
  cv_us.len=strlen(cv_us.arr);
  strcpy(passwd.arr,argv[2]);
  passwd.len=strlen(passwd.arr);
  strcpy(fecini.arr,argv[3]);
  fecini.len=strlen(fecini.arr);

  EXEC SQL WHENEVER SQLERROR GOTO error_oracle;
  EXEC SQL CONNECT :cv_us IDENTIFIED BY :passwd;

  EXEC ORACLE OPTION (ORACA=YES);

  oraca.orahchf=1;     /*Habilita variables para el */
  oraca.oradbgf=1;     /*manejo de errores */
  oraca.oracchf=1;
  oraca.orastxtf=3;

  EXEC SQL DECLARE lee_goa1 CURSOR FOR
    SELECT n_cta,n_scta,n_cenafo,n_dep,n_conori,i_op_acu,
        i_op_mes,rowid
    FROM t_goa;

  EXEC SQL OPEN lee_goa1;
  EXEC SQL WHENEVER NOT FOUND GOTO fin_goa1;
  for(;;)
  {
   EXEC SQL FETCH lee_goa1 INTO :xncta,:xnscta,:xncenafo,
                    :xndep,:xconori,:xiopacu,
                    :xiopmes,:numreg;

   xnscta.arr[xnscta.len]='\0';
   xconori.arr[xconori.len]='\0';
   numreg.arr[numreg.len]='\0';

   EXEC SQL UPDATE t_goa
     SET i_op_acu = :xiopacu + :xiopmes,
       i_op_mes = 0
     WHERE :numreg = rowid;
```

```
    EXEC SQL COMMIT;
   }
  fin_goa1:
   EXEC SQL CLOSE lee_goa1;
   EXEC SQL COMMIT;
   exit(0);


error_oracle:
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml]='\0';
     oraca.orastxt.orastxtc[oraca.orastxt.orastxtl]='\0';
     oraca.orasfnm.orasfnmc[oraca.orasfnm.orasfnml]='\0';
     printf("\n\nError encontrado %s \n\n",
          sqlca.sqlerrm.sqlerrmc);
     printf("Argumento de SQL con error:\n \"%s...\"\n\n",
          oraca.orastxt.orastxtc);
     printf("en la linea %d del programa con id: %s.\n",
          oraca.orasInr,oraca.orasfnm.orasfnmc);
     EXEC SQL ROLLBACK WORK
     exit(0)

 }
```

IV.3 Programas de Reportes

Como se vio en el punto IV.1 se pueden obtener los siguientes reportes. Al seleccionar la opción "C" del menú principal (figura IV.2), se pueden obtener los siguientes reportes:

- Reporte por Cuenta Subcuenta y Concepto de Origen.
- Reporte por Centro y Departamento.
- Reporte por Renglón del Gasto.
- Reporte por Cuenta y Concepto de Origen.
- Reporte de Resumen por Concepto de Origen.
- Reporte de Resumen por Cuenta de Mayor.

Al seleccionar la opción "A" del menú de reportes (figura IV.4) obtenemos el reporte de cuenta,subcuenta y concepto de origen. Este reporte nos entrega un reporte donde agrupa las cuentas de operación y sus subcuentas a través del cual se puede observar en que cuenta y subcuenta se realizó un gasto mayor y en cuales un gasto menor.

Al seleccionar esta opción se ejecuta el programa GOR8001.exe (ver apéndice A)que es un shell que le pasa los parámetros de usuario, password y fecha de cierre al programa GOR8001.pc, el cual se muestra a continuación

```
/* ============================================================
GOR8001.pc  - Reporte a nivel cuenta,subcuenta,centro y concepto de origen.

Accesos:          T_GOA[S]  CONOR[S]  CETRO[S]  T_041[S]
                  T_047[S]  T_052[S]  T_050[9]
```

Parametros: GOR8001 <Usuario> <Clave> <DDMMAA>

```
=============================================================*/
#include <stdio.h>
char des_mes[20];
int ren=80,primero=1,hoja=0;
FILE *sal;

EXEC SQL BEGIN DECLARE SECTION;          /*Inicia declaración de variables*/

VARCHAR usuario[10],clave[15];
VARCHAR des_cpto[41],n_grupo_lei[3],n_grupo_ant[3],n_conori_lei[7];
VARCHAR mes[3],ano[3],f_cierre[7],imp_importe[20],nom_conta[50];
VARCHAR n_scta_ant[9],n_scta_lei[9],des_ctro[50];
VARCHAR nom_cta[40],nom_scta[27];
int   n_cta_ant,n_cta_lei,n_cenafo_ant,n_cenafo_lei;
double importe_acumulado,importe_mensual,importe_total;
double total_cenafo_acumulado=0,total_cenafo_mensual=0,total_cenafo_total=0;
double total_grupo_acumulado=0,total_grupo_mensual=0,total_grupo_total=0;
double total_cuenta_acumulado=0,total_cuenta_mensual=0,total_cuenta_total=0;
double total_subcta_acumulado=0,total_subcta_mensual=0,total_subcta_total=0;
double total_general_acumulado=0,total_general_mensual=0,total_general_total=0;

VARCHAR wimporte_acumulado[15],wimporte_mensual[15],wimporte_total[15];
VARCHAR wtotal_cenafo_acumulado[15],wtotal_cenafo_mensual[15];
VARCHAR wtotal_cenafo_total[15];
VARCHAR wtotal_grupo_acumulado[15],wtotal_grupo_mensual[15];
VARCHAR wtotal_grupo_total[15];
VARCHAR wtotal_cuenta_acumulado[15],wtotal_cuenta_mensual[15];
VARCHAR wtotal_cuenta_total[15];
VARCHAR wtotal_subcta_acumulado[15],wtotal_subcta_mensual[15];
VARCHAR wtotal_subcta_total[15];
VARCHAR wtotal_general_acumulado[15],wtotal_general_mensual[15];
VARCHAR wtotal_general_total[15];

EXEC SQL END DECLARE SECTION;          /*Termina declaración de variables*/

EXEC SQL INCLUDE sqlca.h;
EXEC SQL DECLARE GOA  CURSOR FOR
Select n_cta,
    n_scta,
    n_cenafo,
    n_conori,
    substr(n_conori,1,2),          /*Selecciona los 2 promeros dígitos de n_conori)*/
    sum(nvl(i_op_acu,0)),          /*Si el importe es nulo pone ceros*/
    sum(nvl(i_op_mes,0)),
    sum(nvl(i_op_acu,0)+nvl(i_op_mes,0))
From  t_goa
Group  by n_cenafo,n_cta,n_scta,n_conori
order  by n_cenafo,n_cta,n_scta,n_conori ;

titulo()
```

```
  {
    hoja++;
    fprintf(sal,"\f%84s","P E T R O L E O S  M E X I C A N O S");
    fprintf(sal,"%30s  "," GOR8001  ");
    fprintf(sal,"  HOJA.- %3i\n ",hoja);
    fprintf(sal,"%84s\n","      SUBDIRECCION DE FINANZAS      ");
    fprintf(sal,"%84s\n","    GERENCIA DE CONTABILIDAD      ");
    fprintf(sal,"%84s\n",nom_conta.arr);
    fprintf(sal,"%77s","GASTOS DE ORIGEN  DEL MES DE ");
    fprintf(sal," %-20s\n\n",des_mes);
    fprintf(sal,"%47s","CUENTA  ");
    fprintf(sal,"       %i",n_cta_ant);
    fprintf(sal,"   %-40s\n",nom_cta.arr);
    fprintf(sal,"%47s","SUBCUENTA  ");
    fprintf(sal,"   %9s",n_scta_ant.arr);
    fprintf(sal,"   %-27s\n\n\n",nom_scta.arr);
    fprintf(sal,"   %3.0d  %-50s\n\n",n_cenafo_ant,des_ctro.arr);
    fprintf(sal,"   Descripcion del Concepto              ");
    fprintf(sal," Concepto       Saldo Anterior          ");
    fprintf(sal,"Movimientos          Saldo actual     \n\n");
    ren=13;
  }

imprime_registro()
  {
    if (ren >55)
      {
        titulo();
      }
    EXEC SQL SELECT to_char(:importe_acumulado,'999,999,999.99'),
                    to_char(:importe_mensual,'999,999,999.99'),
                    to_char(:importe_total,'999,999,999.99')
             INTO :wimporte_acumulado,:wimporte_mensual,:wimporte_total
             FROM DUAL;
    fprintf(sal," %40-s  %6-s",des_cpto.arr,n_conori_lei.arr);
    fprintf(sal,"   %20s",wimporte_acumulado.arr);
    fprintf(sal,"    %20s",wimporte_mensual.arr);
    fprintf(sal,"    %20s\n",wimporte_total.arr);
    ren++;
  }

acumula()
  {
    total_cenafo_acumulado+=importe_acumulado;
    total_cenafo_mensual+=importe_mensual;
    total_cenafo_total+=importe_total;
    total_grupo_acumulado+=importe_acumulado;
    total_grupo_mensual+=importe_mensual;
    total_grupo_total+=importe_total;
    total_subcta_acumulado+=importe_acumulado;
    total_subcta_mensual+=importe_mensual;
    total_subcta_total+=importe_total;
```

```
    total_cuenta_acumulado+=importe_acumulado;
    total_cuenta_mensual+=importe_mensual;
    total_cuenta_total+=importe_total;
    total_general_acumulado+=importe_acumulado;
    total_general_mensual+=importe_mensual;
    total_general_total+=importe_total;
  }

corte_cenafo()
  {
    EXEC SQL SELECT to_char(:total_cenafo_acumulado,'999,999,999.99'),
                    to_char(:total_cenafo_mensual,'999,999,999.99'),
                    to_char(:total_cenafo_total,'999,999,999.99')
             INTO :wtotal_cenafo_acumulado,:wtotal_cenafo_mensual,
                  :wtotal_cenafo_total
             FROM DUAL;
    fprintf(sal,"\n        Total Centro  ");
    fprintf(sal,"%27s  %20s"," ",wtotal_cenafo_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_cenafo_mensual.arr);
    fprintf(sal,"    %20s\n\n",wtotal_cenafo_total.arr);
    ren=60;
    total_cenafo_acumulado=0;
    total_cenafo_mensual=0;
    total_cenafo_total=0;
    n_cenafo_ant=n_cenafo_lei;
    obtiene_cetro();
  }

corte_grupo()
  {
    EXEC SQL SELECT to_char(:total_grupo_acumulado,'999,999,999.99'),
                    to_char(:total_grupo_mensual,'999,999,999.99'),
                    to_char(:total_grupo_total,'999,999,999.99')
             INTO :wtotal_grupo_acumulado,:wtotal_grupo_mensual,
                  :wtotal_grupo_total
             FROM DUAL;
    fprintf(sal,"\n        Total Grupo  ");
    fprintf(sal,"%28s  %20s"," ",wtotal_grupo_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_grupo_mensual.arr);
    fprintf(sal,"    %20s\n\n",wtotal_grupo_total.arr);
    ren+=4;
    total_grupo_acumulado=0;
    total_grupo_mensual=0;
    total_grupo_total=0;
    strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
    if (ren >55)
      {
        titulo();
      }
  }

corte_subcta()
```

```
  {
    EXEC SQL SELECT to_char(:total_subcta_acumulado,'999,999,999.99'),
                    to_char(:total_subcta_mensual,'999,999,999.99'),
                    to_char(:total_subcta_total,'999,999,999.99')
             INTO :wtotal_subcta_acumulado,:wtotal_subcta_mensual,
                  :wtotal_subcta_total
             FROM DUAL;
    fprintf(sal,"\n          Total Subcuenta   ");
    fprintf(sal,"%23s  %20s"," ",wtotal_subcta_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_subcta_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_subcta_total.arr);
    ren+=4;
    total_subcta_acumulado=0;
    total_subcta_mensual=0;
    total_subcta_total=0;
    strcpy(n_scta_ant.arr,n_scta_lei.arr);
    n_scta_ant.len=strlen(n_scta_ant.arr);
    n_scta_ant.arr[n_scta_ant.len]='\0';
    obtiene_scta();
    if (ren >55)
     {
       titulo();
     }
  }

corte_cuenta()
  {
    EXEC SQL SELECT to_char(:total_cuenta_acumulado,'999,999,999.99'),
                    to_char(:total_cuenta_mensual,'999,999,999.99'),
                    to_char(:total_cuenta_total,'999,999,999.99')
             INTO :wtotal_cuenta_acumulado,:wtotal_cuenta_mensual,
                  :wtotal_cuenta_total
             FROM DUAL;
    fprintf(sal,"\n          Total Cuenta   ");
    fprintf(sal,"%26s  %20s"," ",wtotal_cuenta_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_cuenta_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_cuenta_total.arr);
    ren+=4;
    total_cuenta_acumulado=0;
    total_cuenta_mensual=0;
    total_cuenta_total=0;
    n_cta_ant=n_cta_lei;
    obtiene_cta();
    strcpy(n_scta_ant.arr,n_scta_lei.arr);
    n_scta_ant.len=strlen(n_scta_ant.arr);
    n_scta_ant.arr[n_scta_ant.len]='\0';
    obtiene_scta();
    if (ren >55)
     {
       titulo();
     }
  }
```

```
total_general()
  {
    EXEC SQL SELECT to_char(:total_general_acumulado,'999,999,999.99'),
                    to_char(:total_general_mensual,'999,999,999.99'),
                    to_char(:total_general_total,'999,999,999.99')
             INTO :wtotal_general_acumulado,:wtotal_general_mensual,
                  :wtotal_general_total
             FROM DUAL;
    fprintf(sal,"\n          Total Contaduria  ");
    fprintf(sal,"%23s  %20s"," ",wtotal_general_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_general_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_general_total.arr);
    ren++;
    if (ren >55)
      {
        titulo();
      }
  }

obtiene_cetro()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_ctro;   /*Prueba condiciones de error*/
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_ctro;
    EXEC SQL SELECT des_ctro
          INTO   :des_ctro
          FROM   cetro
          WHERE  cve_ctro=:n_cenafo_ant;
          des_ctro.arr[des_ctro.len]='\0';
    return(0);
    sqlerr_ctro:
        fprintf(sal," ERROR DE ORACLE AL OBTENER EL CENTRO   \n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    err_obt_ctro:
        strcpy(des_ctro.arr,"*******    NO EXISTE CENTRO    *********");
        return(0);
  }

obtiene_cta()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_cta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_cta;
    EXEC SQL SELECT substr(nom_cta,1,40)
          INTO   :nom_cta
          FROM   t_047
          WHERE  n_cta=:n_cta_ant;
          nom_cta.arr[nom_cta.len]='\0';
    return(0);
    sqlerr_cta:
        fprintf(sal," ERROR DE ORACLE AL OBTENER NOMBRE CTA\n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
```

```
        return(1);
    err_obt_cta:
        strcpy(nom_cta.arr,"******* NO EXISTE CUENTA        *********");
        return(0);
  }

obtiene_scta()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_scta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_scta;
    EXEC SQL SELECT nom_scta
             INTO :nom_scta
             FROM t_052
            WHERE n_cta=:n_cta_ant
              AND cv_scta=:n_scta_ant;
            nom_scta.arr[nom_scta.len]='\0';
    return(0);
    sqlerr_scta:
        fprintf(sal," ERROR DE ORACLE AL OBTENER NOMBRE SCTA\n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    err_obt_scta:
        strcpy(nom_scta.arr,"**** NO EXISTE SUBCUENTA ****");
        return(0);
  }

obtiene_contaduria()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_conta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_contaduria;
    EXEC SQL SELECT nom_cont
          INTO   :nom_conta
          FROM   t_059
          WHERE  n_cont=(SELECT n_cont FROM t_041);
    nom_conta.arr[nom_conta.len]='\0';
    return(0);
    sqlerr_conta:
        fprintf(sal," ERROR DE ORACLE AL OBTENER CONTADURIA\n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    err_obt_contaduria:
        strcpy(nom_conta.arr,"******* NO EXISTE CONTADURIA    *********");
        return(0);
  }

obtiene_concepto()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr;
    EXEC SQL WHENEVER NOT FOUND GOTO busca_01;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
          INTO   :des_cpto
          FROM   CONOR
```

```
        WHERE  decode(substr(CVE_CPTO,1,2),
                  '00','06'||substr(CVE_CPTO,3),
                   CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
    sqlerr:
        fprintf(sal," ERROR DE ORACLE AL OBTENER CONCEPTO\n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    busca_01:
        EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
        EXEC SQL WHENEVER NOT FOUND GOTO err_obt_con;
        EXEC SQL SELECT substr(DES_CPTO,1,40)
              INTO  :des_cpto
              FROM  CONOR
              WHERE  decode(substr(CVE_CPTO,1,2),
                    '00','06'||substr(CVE_CPTO,3,4),
                     CVE_CPTO)=:n_conori_lei;
        des_cpto.arr[des_cpto.len]='\0';
        return(0);
    err_obt_con:
        strcpy(des_cpto.arr,"******  NO EXISTE CONCEPTO  **************");
        return(0);
}

proceso()
 {
    EXEC SQL WHENEVER SQLERROR GOTO falla;
    EXEC SQL OPEN GOA;
    if ((sal=fopen("GOR8001.lst","w"))==0)
      {
        printf("ERROR: Al Abrir el Archivo GOR8001.lst\n");
        exit(0);
      }
    EXEC SQL WHENEVER NOT FOUND GOTO fin_proceso;
    for( ; ; )
      {
        EXEC SQL FETCH GOA
        INTO :n_cta_lei,:n_scta_lei,:n_cenafo_lei,:n_conori_lei,
           :n_grupo_lei,:importe_acumulado,
           :importe_mensual,:importe_total;

           n_conori_lei.arr[n_conori_lei.len]='\0';
           n_grupo_lei.arr[n_grupo_lei.len]='\0';
           n_scta_lei.arr[n_scta_lei.len]='\0';
        obtiene_concepto();
        if (primero == 1)
          {
           n_cenafo_ant=n_cenafo_lei;
           strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
           n_cta_ant=n_cta_lei;
           strcpy(n_scta_ant.arr,n_scta_lei.arr);
```

```
    n_scta_ant.len=strlen(n_scta_ant.arr);
    n_scta_ant.arr[n_scta_ant.len]='\0';
    obtiene_scta();
    obtiene_cta();
    obtiene_cetro();
    primero=0;
   }
if(n_cenafo_ant==n_cenafo_lei)
 {
  if(n_cta_ant==n_cta_lei)
   {
    if(strcmp(n_scta_lei.arr,n_scta_ant.arr) == 0 )
     {
      if(strcmp(n_grupo_lei.arr,n_grupo_ant.arr) == 0 )
       {
        imprime_registro();
        acumula();
       }
      else
       {
        corte_grupo();
        imprime_registro();
        acumula();
       }
     }
    else
     {
      corte_grupo();
      corte_subcta();
      imprime_registro();
      acumula();
     }
   }
  else
   {
     corte_grupo();
     corte_subcta();
     corte_cuenta();
     imprime_registro();
     acumula();
   }
  }
 else
  {
     corte_grupo();
     corte_subcta();
     corte_cuenta();
     corte_cenafo();
     imprime_registro();
     acumula();
  }
}
```

```
falla:
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      printf("\n**************************************************");
      printf("\nERROR (Proceso): %s",sqlca.sqlerrm.sqlerrmc);
      printf("\n**************************************************\n");
      EXEC SQL ROLLBACK WORK
      RELEASE;
      exit(0);
fin_proceso:
      EXEC SQL CLOSE GOA;
      EXEC SQL COMMIT WORK;
      EXEC SQL RELEASE;
      corte_grupo();
      corte_subcta();
      corte_cuenta();
      corte_cenafo();
      total_general();
      fclose(sal);
      exit(0);
  }
main(argc,argv)
   int argc;
   char *argv[];
   {
      strcpy(usuario.arr,argv[1]);
      usuario.len=strlen(usuario.arr);
      usuario.arr[usuario.len]='\0';
      strcpy(clave.arr,argv[2]);
      clave.len=strlen(clave.arr);
      clave.arr[clave.len]='\0';
      strcpy(f_cierre.arr,argv[3]);
      f_cierre.len=strlen(f_cierre.arr);
      f_cierre.arr[f_cierre.len]='\0';
      mes.arr[0]=f_cierre.arr[2];
      mes.arr[1]=f_cierre.arr[3];
      mes.arr[2]='\0';
      ano.arr[0]=f_cierre.arr[4];
      ano.arr[1]=f_cierre.arr[5];
      ano.arr[2]='\0';
      mes.len=2;
      ano.len=2;
      switch (atoi(mes.arr))
        {
          case 1: strcpy(des_mes,"ENERO"); break;
          case 2: strcpy(des_mes,"FEBRERO"); break;
          case 3: strcpy(des_mes,"MARZO"); break;
          case 4: strcpy(des_mes,"ABRIL"); break;
          case 5: strcpy(des_mes,"MAYO"); break;
          case 6: strcpy(des_mes,"JUNIO"); break;
          case 7: strcpy(des_mes,"JULIO"); break;
          case 8: strcpy(des_mes,"AGOSTO"); break;
          case 9: strcpy(des_mes,"SEPTIEMBRE"); break;
```

```
        case 10: strcpy(des_mes,"OCTUBRE"); break;
        case 11: strcpy(des_mes,"NOVIEMBRE"); break;
        case 12: strcpy(des_mes,"DICIEMBRE"); break;
      }
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT :usuario IDENTIFIED BY :clave;
    obtiene_contaduria();
    proceso();
error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("\n**************************************************");
    printf("\nERROR (Inicio): %s",sqlca.sqlerrm.sqlerrmc);
    printf("\n**************************************************\n");
    EXEC SQL COMMIT WORK;
    EXEC SQL RELEASE;
    exit(0);
}
```

Una muestra del reporte del programa GOR8001.pc se muestra en la figura IV.14. En este reporte se muestran las columnas de saldo anterior, movimientos del mes y saldo actual, a través de las cuales se realiza una comparación de cuanto se está gastando.

El reporte por centro y departamento se obtiene al seleccionar la opción "B" del menu de reportes (figura IV.4) con este reporte podemos observar los gastos que hubo por centro de trabajo y departamento mensualmente.

El programa que se corre es GOR8002.exe (ver apéndice A) que le pasa los parámetros de usuario ,password y fecha de cierre al programa GOR8002.pc , el cual se muestra a continuación

```
/* ================================================================
   GOR8002.pc   - Reporte a nivel centro,departamento y concepto de origen.

   Accesos:          T_GOA[S]  CONOR[S]  DEPTO[S]  T_041[S]
                     T_047[S]  T_052[S]  T_059[S]

   Parametros:   GOR8002  <Usuario> <Clave> <DDMMAA>

 ================================================================*/
#include <stdio.h>
char des_mes[20];
int  ren=80,primero=1,hoja=0;
FILE *sal;
EXEC SQL BEGIN DECLARE SECTION;
   VARCHAR usuario[10],clave[15];
   VARCHAR des_cpto[41],n_conori_lei[7],des_depto[50];
   VARCHAR mes[3],ano[3],f_cierre[7],imp_importe[20],nom_conta[50];
   VARCHAR n_scta_ant[9],n_scta_lei[9];
   VARCHAR nom_cta[40],nom_scta[27];
   int    n_cta_ant,n_cta_lei,n_cenafo_ant,n_cenafo_lei,n_dep_ant,n_dep_lei;
   double  importe_acumulado,importe_mensual,importe_total;
double total_depto_acumulado=0,total_depto_mensual=0,total_depto_total=0;
```

```
double total_cenafo_acumulado=0,total_cenafo_mensual=0,total_cenafo_total=0;
double total_cuenta_acumulado=0,total_cuenta_mensual=0,total_cuenta_total=0;
double total_subcta_acumulado=0,total_subcta_mensual=0,total_subcta_total=0;
double total_general_acumulado=0,total_general_mensual=0,total_general_total=0;
VARCHAR wimporte_acumulado[15],wimporte_mensual[15],wimporte_total[15];
VARCHAR wtotal_depto_acumulado[15],wtotal_depto_mensual[15];
VARCHAR wtotal_depto_total[15];
VARCHAR wtotal_cenafo_acumulado[15],wtotal_cenafo_mensual[15];
VARCHAR wtotal_cenafo_total[15];
VARCHAR wtotal_cuenta_acumulado[15],wtotal_cuenta_mensual[15];
VARCHAR wtotal_cuenta_total[15];
VARCHAR wtotal_subcta_acumulado[15],wtotal_subcta_mensual[15];
VARCHAR wtotal_subcta_total[15];
VARCHAR wtotal_general_acumulado[15],wtotal_general_mensual[15];
VARCHAR wtotal_general_total[15];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;
EXEC SQL DECLARE GOA  CURSOR FOR
Select n_cta,
    n_scta,
    n_cenafo,
    n_dep,
    n_conori,
    sum(nvl(i_op_acu,0)),
    sum(nvl(i_op_mes,0)),
    sum(nvl(i_op_acu,0)+nvl(i_op_mes,0))
From  t_goa
Group  by n_cta,n_scta,n_cenafo,n_dep,n_conori
order  by n_cta,n_scta,n_cenafo,n_dep,n_conori;

titulo()
  {
    obtiene_depto();
    hoja++;
    fprintf(sal,"\f%84s","P E T R O L E O S  M E X I C A N O S");
    fprintf(sal,"%30s  "," GOR8002  ");
    fprintf(sal,"  HOJA.- %3i\n ",hoja);
    fprintf(sal,"%84s\n","      SUBDIRECCION DE FINANZAS      ");
    fprintf(sal,"%86s\n","      GERENCIA DE CONTABILIDAD      ");
    fprintf(sal,"%77s\n",nom_conta.arr);
    fprintf(sal,"%77s","GASTOS DE ORIGEN  DEL MES DE ");
    fprintf(sal," %-20s\n\n",des_mes);
    fprintf(sal,"%47s","CUENTA  ");
    fprintf(sal,"       %i",n_cta_ant);
    fprintf(sal,"   %-40s\n",nom_cta.arr);
    fprintf(sal,"%47s","SUBCUENTA  ");
    fprintf(sal,"   %9s",n_scta_ant.arr);
    fprintf(sal,"   %-27s\n\n\n",nom_scta.arr);
    fprintf(sal,"   %3.0d  %5.0d  %-50s\n\n",n_cenafo_ant,n_dep_ant,des_depto.arr);
    fprintf(sal,"    Descipcion del Concepto              ");
    fprintf(sal," Concepto       Saldo Anterior           ");
```

```
    fprintf(sal,"Movimientos          Saldo actual     \n\n");
    ren=12;
  }
imprime_registro()
  {
    if (ren >50)
     {
       titulo();
     }
    EXEC SQL SELECT to_char(:importe_acumulado,'999,999,999.99'),
                    to_char(:importe_mensual,'999,999,999.99'),
                    to_char(:importe_total,'999,999,999.99')
             INTO :wimporte_acumulado,:wimporte_mensual,:wimporte_total
             FROM DUAL;
    fprintf(sal," %40-s    %6-s",des_cpto.arr,n_conori_lei.arr);
    fprintf(sal,"  %20s",wimporte_acumulado.arr);
    fprintf(sal,"    %20s",wimporte_mensual.arr);
    fprintf(sal,"    %20s\n",wimporte_total.arr);
    ren++;
  }
corte_depto()
  {
    EXEC SQL SELECT to_char(:total_depto_acumulado,'999,999,999.99'),
                    to_char(:total_depto_mensual,'999,999,999.99'),
                    to_char(:total_depto_total,'999,999,999.99')
             INTO :wtotal_depto_acumulado,:wtotal_depto_mensual,
                  :wtotal_depto_total
             FROM DUAL;
    fprintf(sal,"\n          Total Departamento  ");
    fprintf(sal,"%22s  %20s"," ",wtotal_depto_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_depto_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_depto_total.arr);
    ren=80;
    total_depto_acumulado=0;
    total_depto_mensual=0;
    total_depto_total=0;
    n_dep_ant=n_dep_lei;
  }
corte_cenafo()
  {
    EXEC SQL SELECT to_char(:total_cenafo_acumulado,'999,999,999.99'),
                    to_char(:total_cenafo_mensual,'999,999,999.99'),
                    to_char(:total_cenafo_total,'999,999,999.99')
             INTO :wtotal_cenafo_acumulado,:wtotal_cenafo_mensual,
                  :wtotal_cenafo_total
             FROM DUAL;
    fprintf(sal,"\n          Total Centro  ");
    fprintf(sal,"%27s  %20s"," ",wtotal_cenafo_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_cenafo_mensual.arr);
    fprintf(sal,"    %20s\n\n",wtotal_cenafo_total.arr);
    ren=80;
    total_cenafo_acumulado=0;
```

```
    total_cenafo_mensual=0;
    total_cenafo_total=0;
    n_cenafo_ant=n_cenafo_lei;
  }
corte_cuenta()
  {
    EXEC SQL SELECT to_char(:total_cuenta_acumulado,'999,999,999.99'),
                    to_char(:total_cuenta_mensual,'999,999,999.99'),
                    to_char(:total_cuenta_total,'999,999,999.99')
             INTO :wtotal_cuenta_acumulado,:wtotal_cuenta_mensual,
                  :wtotal_cuenta_total
             FROM DUAL;
    fprintf(sal,"\n          Total Cuenta  ");
    fprintf(sal,"%29s  %20s"," ",wtotal_cuenta_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_cuenta_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_cuenta_total.arr);
    ren=80;
    total_cuenta_acumulado=0;
    total_cuenta_mensual=0;
    total_cuenta_total=0;
    n_cta_ant=n_cta_lei;
    obtiene_cta();
    strcpy(n_scta_ant.arr,n_scta_lei.arr);
    n_scta_ant.len=strlen(n_scta_ant.arr);
    n_scta_ant.arr[n_scta_ant.len]='\0';
    obtiene_scta();
  }
corte_subcta()
  {
    EXEC SQL SELECT to_char(:total_subcta_acumulado,'999,999,999.99'),
                    to_char(:total_subcta_mensual,'999,999,999.99'),
                    to_char(:total_subcta_total,'999,999,999.99')
             INTO :wtotal_subcta_acumulado,:wtotal_subcta_mensual,
                  :wtotal_subcta_total
             FROM DUAL;
    fprintf(sal,"\n          Total Subcuenta  ");
    fprintf(sal,"%25s  %20s"," ",wtotal_subcta_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_subcta_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_subcta_total.arr);
    ren++;
    ren=80;
    total_subcta_acumulado=0;
    total_subcta_mensual=0;
    total_subcta_total=0;
    strcpy(n_scta_ant.arr,n_scta_lei.arr);
    obtiene_scta();
    if (ren >55)
     {
       titulo();
     }
  }
total_general()
```

```
  {
    EXEC SQL SELECT to_char(:total_general_acumulado,'999,999,999.99'),
                    to_char(:total_general_mensual,'999,999,999.99'),
                    to_char(:total_general_total,'999,999,999.99')
             INTO :wtotal_general_acumulado,:wtotal_general_mensual,
                  :wtotal_general_total
             FROM DUAL;
    fprintf(sal,"\n          Total Contaduria  ");
    fprintf(sal,"%25s  %20s"," ",wtotal_general_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_general_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_general_total.arr);
  }
obtiene_depto()
  {
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr_depto;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_depto;
    EXEC SQL SELECT des_dep
             INTO  :des_depto
             FROM  t_dep
             WHERE n_cenafo=:n_cenafo_ant
              AND  n_dep=:n_dep_ant;
    des_depto.arr[des_depto.len]='\0';
    return(0);
sqlerr_depto:
    fprintf(sal," ERROR DE ORACLE AL OBTENER DEPARTAMENTO\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
err_obt_depto:
    strcpy(des_depto.arr,"******* NO EXISTE DEPARTAMENTO *********");
    return(0);
  }
obtiene_cta()
  {
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr_cta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_cta;
    EXEC SQL SELECT substr(nom_cta,1,40)
             INTO  :nom_cta
             FROM  t_047
             WHERE n_cta=:n_cta_ant;
             nom_cta.arr[nom_cta.len]='\0';
    return(0);
sqlerr_cta:
    fprintf(sal," ERROR DE ORACLE AL OBTENER NOMBRE CTA\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
err_obt_cta:
    strcpy(nom_cta.arr,"******* NO EXISTE CUENTA       *********");
    return(0);
  }
obtiene_scta()
  {
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr_scta;
```

```
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_scta;
    EXEC SQL SELECT nom_scta
             INTO :nom_scta
             FROM t_052
             WHERE n_cta=:n_cta_ant
              AND cv_scta=:n_scta_ant;
             nom_scta.arr[nom_scta.len]='\0';
    return(0);
sqlerr_scta:
    fprintf(sal," ERROR DE ORACLE AL OBTENER NOMBRE SCTA\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
err_obt_scta:
    strcpy(nom_scta.arr,"**** NO EXISTE SUBCUENTA ****");
    return(0);
  }
obtiene_contaduria()
  {
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr_conta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_contaduria;
    EXEC SQL SELECT nom_cont
         INTO   :nom_conta
         FROM   t_059
         WHERE  n_cont=(SELECT n_cont FROM t_041);
    nom_conta.arr[nom_conta.len]='\0';
    return(0);
sqlerr_conta:
    fprintf(sal," ERROR DE ORACLE AL OBTENER CONTADURIA\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
err_obt_contaduria:
    strcpy(nom_conta.arr,"******* NO EXISTE CONTADURIA *********");
    return(0);
  }
obtiene_concepto()
  {
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
    EXEC SQL WHENEVER NOT FOUND GOTO busca_01;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
         INTO   :des_cpto
         FROM   CONOR
         WHERE  decode(substr(CVE_CPTO,1,2),
                '00','06'||substr(CVE_CPTO,3,4),
                CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
sqlerr:
    fprintf(sal," ERROR DE ORACLE AL OBTENER CONCEPTO\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
busca_01:
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
```

```
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_con;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
          INTO  :des_cpto
          FROM  CONOR
          WHERE  decode(substr(CVE_CPTO,1,2),
                    '00','06'||substr(CVE_CPTO,3,4),
                     CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
err_obt_con:
    strcpy(des_cpto.arr,"*****  NO EXISTE CONCEPTO  ************");
    return(0);
  }
proceso()
  {
    EXEC SQL WHENEVER SQLERROR GOTO falla;
    EXEC SQL OPEN GOA;
    if ((sal=fopen("GOR8002.lst","w"))==0)
      {
        printf("ERROR: Al Abrir el Archivo GOR8002.lst\n");
        exit(0);
      }
    EXEC SQL WHENEVER NOT FOUND GOTO fin_proceso;
    for( ; ; )
      {
        EXEC SQL FETCH GOA
        INTO :n_cta_lei,:n_scta_lei,:n_cenafo_lei,:n_dep_lei,
          :n_conori_lei,:importe_acumulado,:importe_mensual,
          :importe_total;
        n_conori_lei.arr[n_conori_lei.len]='\0';
         n_scta_lei.arr[n_scta_lei.len]='\0';
        obtiene_concepto();
        if (primero == 1)
          {
           n_cta_ant=n_cta_lei;
           strcpy(n_scta_ant.arr,n_scta_lei.arr);
           n_scta_ant.len=strlen(n_scta_ant.arr);
           n_scta_ant.arr[n_scta_ant.len]='\0';
           obtiene_cta();
           obtiene_scta();
           n_cenafo_ant=n_cenafo_lei;
           n_dep_ant=n_dep_lei;
           primero=0;
          }
        if(n_dep_ant!=n_dep_lei)
          corte_depto();
        if(n_cenafo_ant!=n_cenafo_lei)
          corte_cenafo();
        if(strcmp(n_scta_lei.arr,n_scta_ant.arr) != 0 )
          corte_subcta();
        if(n_cta_ant!=n_cta_lei)
          corte_cuenta();
```

```
        imprime_registro();
        total_cenafo_acumulado+=importe_acumulado;
        total_cenafo_mensual+=importe_mensual;
        total_cenafo_total+=importe_total;
        total_depto_acumulado+=importe_acumulado;
        total_depto_mensual+=importe_mensual;
        total_depto_total+=importe_total;
        total_subcta_acumulado+=importe_acumulado;
        total_subcta_mensual+=importe_mensual;
        total_subcta_total+=importe_total;
        total_cuenta_acumulado+=importe_acumulado;
        total_cuenta_mensual+=importe_mensual;
        total_cuenta_total+=importe_total;
        total_general_acumulado+=importe_acumulado;
        total_general_mensual+=importe_mensual;
        total_general_total+=importe_total;
      }
falla:
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      printf("\n**********************************************");
      printf("\nERROR (Proceso): %s",sqlca.sqlerrm.sqlerrmc);
      printf("\n**********************************************\n");
      EXEC SQL ROLLBACK WORK
      RELEASE;
      exit(0);
fin_proceso:
      EXEC SQL CLOSE GOA;
      EXEC SQL COMMIT WORK;
      EXEC SQL RELEASE;
      corte_depto();
      corte_cenafo();
      corte_subcta();
      corte_cuenta();
      total_general();
      fclose(sal);
      exit(0);
  }
main(argc,argv)
   int argc;
   char *argv[];
   {
      strcpy(usuario.arr,argv[1]);
      usuario.len=strlen(usuario.arr);
      usuario.arr[usuario.len]='\0';
      strcpy(clave.arr,argv[2]);
      clave.len=strlen(clave.arr);
      clave.arr[clave.len]='\0';
      strcpy(f_cierre.arr,argv[3]);
      f_cierre.len=strlen(f_cierre.arr);
      f_cierre.arr[f_cierre.len]='\0';
      mes.arr[0]=f_cierre.arr[2];
      mes.arr[1]=f_cierre.arr[3];
```

```
    mes.arr[2]='\0';
    ano.arr[0]=f_cierre.arr[4];
    ano.arr[1]=f_cierre.arr[5];
    ano.arr[2]='\0';
    mes.len=2;
    ano.len=2;
    switch (atoi(mes.arr))
      {
        case 1: strcpy(des_mes,"ENERO"); break;
        case 2: strcpy(des_mes,"FEBRERO"); break;
        case 3: strcpy(des_mes,"MARZO"); break;
        case 4: strcpy(des_mes,"ABRIL"); break;
        case 5: strcpy(des_mes,"MAYO"); break;
        case 6: strcpy(des_mes,"JUNIO"); break;
        case 7: strcpy(des_mes,"JULIO"); break;
        case 8: strcpy(des_mes,"AGOSTO"); break;
        case 9: strcpy(des_mes,"SEPTIEMBRE"); break;
        case 10: strcpy(des_mes,"OCTUBRE"); break;
        case 11: strcpy(des_mes,"NOVIEMBRE"); break;
        case 12: strcpy(des_mes,"DICIEMBRE"); break;
      }
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT :usuario IDENTIFIED BY :clave;
    obtiene_contaduria();
    proceso();
error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("\n**************************************************");
    printf("\nERROR (Inicio): %s",sqlca.sqlerrm.sqlerrmc);
    printf("\n**************************************************\n");
    EXEC SQL COMMIT WORK;
    EXEC SQL RELEASE;
    exit(0);
}
```

En la figura IV.5 se muestra parte del reporte del programa GOR8002.pc donde se muestran los cortes por departamento, por centro y por contaduría, con las columnas de saldo anterior, movimientos del mes y saldo actual.

El reporte de renglón del gasto, opción "D" es una agrupación de conceptos de origen y con el se identifican los gastos en una forma más genaral. El programa que pasa los parámetros de usuario, password y fecha de cierre es GOR8005.exe (ver apéndice A). El programa del reporte es GOR8005.pc que se muestra a continuación.

```
/* ============================================================
   GOR8005.pc    - Reporte  a nivel de Renglon del Gasto.

   Accesos:       T_GOA[S]  CONOR[S]  CETRO[S]  T_041[S]
                  T_047[S]  T_052[S]  T_050[9]

   Parametros:    GOR8005  <Usuario> <Clave> <DDMMAA>
```

```
 ================================================== ==================*/
#include <stdio.h>
char des_mes[20];
int  ren=60,primero=1,hoja=0;
FILE *sal;

EXEC SQL BEGIN DECLARE SECTION;
double total_cta_acumulado=0,total_cta_mensual=0,total_cta_total=0;
double total_general_acumulado=0,total_general_mensual=0,total_general_total=0;
VARCHAR usuario[10],clave[15];
VARCHAR des_cplo[41],n_grupo_lei[3],n_grupo_ant[3],n_conori_lei[7];
VARCHAR mes[3],ano[3],f_cierre[7],imp_importe[20],nom_conta[50];
double importe_acumulado,importe_mensual,importe_total;
VARCHAR wimporte_acumulado[15],wimporte_mensual[15],wimporte_total[15];
int ren_gas,n_cta,ren_gas_ant,n_cta_ant;
VARCHAR wtotal_cta_acumulado[15],wtotal_cta_mensual[15],wtotal_cta_total[15];
VARCHAR wtotal_general_acumulado[15],wtotal_general_mensual[15];
VARCHAR wtotal_general_total[15];

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca.h;
EXEC SQL DECLARE GOA  CURSOR FOR
Select n_cta,ren_gas,
     sum(nvl(i_op_acu,0)),
     sum(nvl(i_op_mes,0)),
     sum(nvl(i_op_acu,0)+nvl(i_op_mes,0))
From  t_goa
Group  by n_cta,ren_gas
order  by n_cta,ren_gas ;

titulo()
  {
    hoja++;
    fprintf(sal,"\f%84s","P E T R O L E O S  M E X I C A N O S");
    fprintf(sal,"%30s  "," GOR8005  ");
    fprintf(sal,"  HOJA.- %3i\n ",hoja);
    fprintf(sal,"%84s\n","     SUBDIRECCION DE FINANZAS      ");
    fprintf(sal,"%84s\n","     GERENCIA DE CONTABILIDAD      ");
    fprintf(sal,"%75s\n",nom_conta.arr);
    fprintf(sal,"%77s","GASTOS DE ORIGEN  DEL MES DE ");
    fprintf(sal," %-20s\n\n",des_mes);
    fprintf(sal,"  Cuenta   Renglon del Gasto            ");
    fprintf(sal,"     Saldo Anterior          ");
    fprintf(sal,"Movimientos          Saldo actual     \n\n");
    ren=13;
  }
imprime_registro()
  {
    if (ren >55)
     {
       titulo();
     }
```

```
    EXEC SQL SELECT to_char(:importe_acumulado,'999,999,999.99'),
                    to_char(:importe_mensual,'999,999,999.99'),
                    to_char(:importe_total,'999,999,999.99')
             INTO :wimporte_acumulado,:wimporte_mensual,:wimporte_total
             FROM DUAL;

    fprintf(sal,"    %4d          %3d",n_cta,ren_gas);
    fprintf(sal,"%22s%s"," ",wimporte_acumulado.arr);
    fprintf(sal,"%10s%s"," ",wimporte_mensual.arr);
    fprintf(sal,"%9s%s\n"," ",wimporte_total.arr);
    ren++;
  }

corte_cta()
  {
    EXEC SQL SELECT to_char(:total_cta_acumulado,'999,999,999.99'),
                    to_char(:total_cta_mensual,'999,999,999.99'),
                    to_char(:total_cta_total,'999,999,999.99')
             INTO :wtotal_cta_acumulado,:wtotal_cta_mensual,:wtotal_cta_total
             FROM DUAL;

    fprintf(sal,"\n        Total Cuenta    ");
    fprintf(sal,"%20s%s"," ",wtotal_cta_acumulado.arr);
    fprintf(sal,"%10s%s"," ",wtotal_cta_mensual.arr);
    fprintf(sal,"%9s%s\n\n"," ",wtotal_cta_total.arr);
    ren=ren+4;
    total_cta_acumulado=0;
    total_cta_mensual=0;
    total_cta_total=0;
    n_cta_ant=n_cta;
    if (ren >55)
     {
       titulo();
     }
  }

corte_general()
  {
    EXEC SQL SELECT to_char(:total_general_acumulado,'999,999,999.99'),
                    to_char(:total_general_mensual,'999,999,999.99'),
                    to_char(:total_general_total,'999,999,999.99')
             INTO :wtotal_general_acumulado,:wtotal_general_mensual,
                  :wtotal_general_total
             FROM DUAL;

    fprintf(sal,"\n        Total Contaduria    ");
    fprintf(sal,"%16s%s"," ",wtotal_general_acumulado.arr);
    fprintf(sal,"%10s%s"," ",wtotal_general_mensual.arr);
    fprintf(sal,"%9s%s\n"," ",wtotal_general_total.arr);
    ren++;
    if (ren >55)
     {
```

```
        titulo();
      }
  }

obtiene_contaduria()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_conta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_contaduria;
    EXEC SQL SELECT nom_cont
          INTO  :nom_conta
          FROM  t_059
          WHERE  n_cont=(SELECT n_cont FROM t_041);
    nom_conta.arr[nom_conta.len]='\0';
    return(0);

    sqlerr_conta:
        fprintf(sal," ERROR DE ORACLE AL OBTENER CONTADURIA\n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    err_obt_contaduria:
        strcpy(nom_conta.arr,"******* NO EXISTE CONTADURIA  ********");
        return(0);
  }

obtiene_concepto()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr;
    EXEC SQL WHENEVER NOT FOUND GOTO busca_01;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
          INTO  :des_cpto
          FROM  CONOR
          WHERE  decode(substr(CVE_CPTO,1,2),
                    '00','06'llsubstr(CVE_CPTO,3,4),
                     CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
    sqlerr:
        fprintf(sal," ERROR DE ORACLE AL OBTENER CONCEPTO\n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    busca_01:
        EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
        EXEC SQL WHENEVER NOT FOUND GOTO err_obt_con;
        EXEC SQL SELECT substr(DES_CPTO,1,40)
              INTO  :des_cpto
              FROM  CONOR
              WHERE  decode(substr(CVE_CPTO,1,2),
                    '00','06'llsubstr(CVE_CPTO,3,4),
                     CVE_CPTO)=:n_conon_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
    err_obt_con:
```

```
        strcpy(des_cpto.arr,"*****  NO EXISTE CONCEPTO  ************");
        return(0);
    }

proceso()
    {
     EXEC SQL WHENEVER SQLERROR GOTO falla;
     EXEC SQL OPEN GOA;
     if ((sal=fopen("GOR8005.lst","w"))==0)
       {
         printf("ERROR: Al Abrir el Archivo GOR8005.lst\n");
         exit(0);
       }
     EXEC SQL WHENEVER NOT FOUND GOTO fin_proceso;
     for( ; ; )
       {
         EXEC SQL FETCH GOA
         INTO :n_cta,:ren_gas,:importe_acumulado,
            :importe_mensual,:importe_total;

         if (primero == 1)
           {
            n_cta_ant=n_cta;
            primero=0;
           }

       if(n_cta_ant==n_cta )
         {
            total_cta_acumulado+=importe_acumulado;
            total_cta_mensual+=importe_mensual;
            total_cta_total+=importe_total;
            total_general_acumulado+=importe_acumulado;
            total_general_mensual+=importe_mensual;
            total_general_total+=importe_total;
            imprime_registro();
            n_cta_ant=n_cta;
         }
       else
        {
         corte_cta();
            total_cta_acumulado+=importe_acumulado;
            total_cta_mensual+=importe_mensual;
            total_cta_total+=importe_total;
            total_general_acumulado+=importe_acumulado;
            total_general_mensual+=importe_mensual;
            total_general_total+=importe_total;
         imprime_registro();
        }
      }
falla:
     EXEC SQL WHENEVER SQLERROR CONTINUE;
     printf("\n*************************************************");
```

```
        printf("\nERROR (Proceso): %s",sqlca.sqlerrm.sqlerrmc);
        printf("\n**********************************\n");
        EXEC SQL ROLLBACK WORK
        RELEASE;
        exit(0);
fin_proceso:
        EXEC SQL CLOSE GOA;
        EXEC SQL COMMIT WORK;
        EXEC SQL RELEASE;
        corte_cta();
        corte_general();
        fclose(sal);
        exit(0);
    }
main(argc,argv)
    int argc;
    char *argv[];
    {
        strcpy(usuario.arr,argv[1]);
        usuario.len=strlen(usuario.arr);
        usuario.arr[usuario.len]='\0';
        strcpy(clave.arr,argv[2]);
        clave.len=strlen(clave.arr);
        clave.arr[clave.len]='\0';
        strcpy(f_cierre.arr,argv[3]);
        f_cierre.len=strlen(f_cierre.arr);
        f_cierre.arr[f_cierre.len]='\0';
        mes.arr[0]=f_cierre.arr[2];
        mes.arr[1]=f_cierre.arr[3];
        mes.arr[2]='\0';
        ano.arr[0]=f_cierre.arr[4];
        ano.arr[1]=f_cierre.arr[5];
        ano.arr[2]='\0';
        mes.len=2;
        ano.len=2;
        switch (atoi(mes.arr))
          {
            case 1: strcpy(des_mes,"ENERO"); break;
            case 2: strcpy(des_mes,"FEBRERO"); break;
            case 3: strcpy(des_mes,"MARZO"); break;
            case 4: strcpy(des_mes,"ABRIL"); break;
            case 5: strcpy(des_mes,"MAYO"); break;
            case 6: strcpy(des_mes,"JUNIO"); break;
            case 7: strcpy(des_mes,"JULIO"); break;
            case 8: strcpy(des_mes,"AGOSTO"); break;
            case 9: strcpy(des_mes,"SEPTIEMBRE"); break;
            case 10: strcpy(des_mes,"OCTUBRE"); break;
            case 11: strcpy(des_mes,"NOVIEMBRE"); break;
            case 12: strcpy(des_mes,"DICIEMBRE"); break;
          }
        EXEC SQL WHENEVER SQLERROR GOTO error;
        EXEC SQL CONNECT :usuario IDENTIFIED BY :clave;
```

```
    obtiene_contaduria();
    proceso();
error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("\n**************************************************");
    printf("\nERROR (Inicio): %s",sqlca.sqlerrm.sqlerrmc);
    printf("\n**************************************************\n");
    EXEC SQL COMMIT WORK;
    EXEC SQL RELEASE;
    exit(0);
  }
```

En la figura IV.6 se muestra el reporte del programa GOR8005.pc.

El siguiente reporte es por cuenta y concepto y que se optiene con la opción "D". A diferencia del reporte de la opción "A" este reporte no toma en cuenta la subcuenta de operación. Los programas que se corren en esta opción son el GOR8004.exe, el cual corre a su vez el programa CTACON.pc.

```
* ================================================================
  CTACON.pc   - Reporte a nivel cuenta y concepto de origen.

  Accesos:      T_GOA[S]  CONOR[S]  CETRO[S]  T_041[S]
                T_047[S]  T_052[S]  T_050[9]

  Parametros:   CTACON   <Usuario> <Clave> <DDMMAA>

 ================================================================*/
#include <stdio.h>
char des_mes[20];
int  ren=80,primero=1,hoja=0;
FILE *sal;
EXEC SQL BEGIN DECLARE SECTION;
  VARCHAR usuario[10],clave[15];
  VARCHAR des_cpto[41],n_grupo_lei[3],n_grupo_ant[3],n_conori_lei[7];
  VARCHAR mes[3],ano[3],f_cierre[7],imp_importe[20],nom_conta[50];
  VARCHAR n_scta_ant[9],n_scta_lei[9],des_ctro[50];
  VARCHAR nom_cta[40],nom_scta[27];
  int   n_cta_ant,n_cta_lei,n_cenafo_ant,n_cenafo_lei;
double  importe_acumulado,importe_mensual,importe_total;
double total_cenafo_acumulado=0,total_cenafo_mensual=0,total_cenafo_total=0;
double total_grupo_acumulado=0,total_grupo_mensual=0,total_grupo_total=0;
double total_cuenta_acumulado=0,total_cuenta_mensual=0,total_cuenta_total=0;
double total_subcta_acumulado=0,total_subcta_mensual=0,total_subcta_total=0;
double total_general_acumulado=0,total_general_mensual=0,total_general_total=0;
VARCHAR wimporte_acumulado[15],wimporte_mensual[15],wimporte_total[15];
VARCHAR wtotal_cenafo_acumulado[15],wtotal_cenafo_mensual[15];
VARCHAR wtotal_cenafo_total[15];
VARCHAR wtotal_grupo_acumulado[15],wtotal_grupo_mensual[15];
VARCHAR wtotal_grupo_total[15];
VARCHAR wtotal_cuenta_acumulado[15],wtotal_cuenta_mensual[15];
VARCHAR wtotal_cuenta_total[15];
```

```
VARCHAR wtotal_subcta_acumulado[15],wtotal_subcta_mensual[15];
VARCHAR wtotal_subcta_total[15];
VARCHAR wtotal_general_acumulado[15],wtotal_general_mensual[15];
VARCHAR wtotal_general_total[15];
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE sqlca.h;
EXEC SQL DECLARE GOA  CURSOR FOR
Select n_cta,
     n_conori,
     substr(n_conori,1,2),
     sum(nvl(i_op_acu,0)),
     sum(nvl(i_op_mes,0)),
     sum(nvl(i_op_acu,0)+nvl(i_op_mes,0))
From  t_goa
Group  by n_cta,n_conori
order  by n_cta,n_conori
;
titulo()
  {
    hoja++;
    fprintf(sal,"\f%84s","P E T R O L E O S  M E X I C A N O S");
    fprintf(sal,"%30s  "," CTACON   ");
    fprintf(sal,"  HOJA.- %3i\n ",hoja);
    fprintf(sal,"%84s\n","      SUBDIRECCION DE FINANZAS      ");
    fprintf(sal,"%86s\n","    GERENCIA DE CONTABILIDAD       ");
    fprintf(sal,"%77s\n",nom_conta.arr);
    fprintf(sal,"%77s","GASTOS DE ORIGEN  DEL MES DE ");
    fprintf(sal," %-20s\n\n",des_mes);
    fprintf(sal,"%47s","CUENTA  ");
    fprintf(sal,"       %i",n_cta_ant);
    fprintf(sal,"   %-40s\n",nom_cta.arr);
    fprintf(sal,"   %3.0d  %-50s\n\n",n_cenafo_ant,des_ctro.arr);
    fprintf(sal,"   Descripcion del Concepto              ");
    fprintf(sal," Concepto       Saldo Anterior          ");
    fprintf(sal,"Movimientos          Saldo actual      \n\n");
    ren=13;
  }
imprime_registro()
  {
    if (ren >55)
     {
       titulo();
     }
   EXEC SQL SELECT to_char(:importe_acumulado,'999,999,999.99'),
              to_char(:importe_mensual,'999,999,999.99'),
              to_char(:importe_total,'999,999,999.99')
         INTO :wimporte_acumulado,:wimporte_mensual,
            :wimporte_total
         FROM DUAL;

    fprintf(sal," %40-s     %6-s",des_cpto.arr,n_conori_lei.arr);
    fprintf(sal,"  %20s",wimporte_acumulado.arr);
```

```
    fprintf(sal,"   %20s",wimporte_mensual.arr);
    fprintf(sal,"   %20s\n",wimporte_total.arr);
    ren++;
  }
corte_grupo()
  {
   EXEC SQL SELECT to_char(:total_grupo_acumulado,'999,999,999.99'),
              to_char(:total_grupo_mensual,'999,999,999.99'),
              to_char(:total_grupo_total,'999,999,999.99')
         INTO :wtotal_grupo_acumulado,:wtotal_grupo_mensual,
            :wtotal_grupo_total
         FROM DUAL;

    fprintf(sal,"\n         Total Grupo  ");
    fprintf(sal,"%29s  %20s"," ",wtotal_grupo_acumulado.arr);
    fprintf(sal,"   %20s",wtotal_grupo_mensual.arr);
    fprintf(sal,"   %20s\n\n",wtotal_grupo_total.arr);
    ren=ren+4;
    total_grupo_acumulado=0;
    total_grupo_mensual=0;
    total_grupo_total=0;
    strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
    if (ren >55)
     {
       titulo();
     }
  }
corte_cuenta()
  {
   EXEC SQL SELECT to_char(:total_cuenta_acumulado,'999,999,999.99'),
              to_char(:total_cuenta_mensual,'999,999,999.99'),
              to_char(:total_cuenta_total,'999,999,999.99')
         INTO :wtotal_cuenta_acumulado,:wtotal_cuenta_mensual,
            :wtotal_cuenta_total
         FROM DUAL;

    fprintf(sal,"\n         Total Cuenta ");
    fprintf(sal,"%29s  %20s"," ",wtotal_cuenta_acumulado.arr);
    fprintf(sal,"   %20s",wtotal_cuenta_mensual.arr);
    fprintf(sal,"   %20s\n",wtotal_cuenta_total.arr);
    ren++;
    ren=80;
    total_cuenta_acumulado=0;
    total_cuenta_mensual=0;
    total_cuenta_total=0;
    n_cta_ant=n_cta_lei;
    obtiene_cta();
    if (ren >55)
     {
       titulo();
     }
  }
```

```
total_general()
  {
   EXEC SQL SELECT to_char(:total_general_acumulado,'999,999,999.99'),
                   to_char(:total_general_mensual,'999,999,999.99'),
                   to_char(:total_general_total,'999,999,999.99')
            INTO :wtotal_general_acumulado,:wtotal_general_mensual,
                 :wtotal_general_total
            FROM DUAL;

    fprintf(sal,"\n          Total Contaduria  ");
    fprintf(sal,"%24s  %20s"," ",wtotal_general_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_general_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_general_total.arr);
    ren++;
    if (ren >55)
     {
       titulo();
     }
  }
obtiene_cta()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_cta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_cta;
    EXEC SQL SELECT substr(nom_cta,1,40)
          INTO   :nom_cta
          FROM   t_047
          WHERE  n_cta=:n_cta_ant;
          nom_cta.arr[nom_cta.len]='\0';
    return(0);
sqlerr_cta:
    fprintf(sal," ERROR DE ORACLE AL OBTENER NOMBRE CTA\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
err_obt_cta:
    strcpy(nom_cta.arr,"******  NO EXISTE CUENTA       *********");
    return(0);
  }
obtiene_contaduria()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_conta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_contaduria;
    EXEC SQL SELECT nom_cont
          INTO   :nom_conta
          FROM   t_059
          WHERE  n_cont=(SELECT n_cont FROM t_041);
    nom_conta.arr[nom_conta.len]='\0';
    return(0);
sqlerr_conta:
    fprintf(sal," ERROR DE ORACLE AL OBTENER CONTADURIA\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
err_obt_contaduria:
```

```
    strcpy(nom_conta.arr,"******* NO EXISTE CONTADURIA  *********");
    return(0);
  }
obtiene_concepto()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr;
    EXEC SQL WHENEVER NOT FOUND GOTO busca_01;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
          INTO  :des_cpto
          FROM  CONOR
          WHERE  decode(substr(CVE_CPTO,1,2),
                      '00','06'llsubstr(CVE_CPTO,3,4),
                       CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
sqlerr:
    fprintf(sal," ERROR DE ORACLE AL OBTENER CONCEPTO\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
busca_01:
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_con;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
          INTO  :des_cpto
          FROM  CONOR
          WHERE  decode(substr(CVE_CPTO,1,2),
                      '00','06'llsubstr(CVE_CPTO,3,4),
                       CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
err_obt_con:
    strcpy(des_cpto.arr,"******  NO EXISTE CONCEPTO  ************");
    return(0);
  }
proceso()
  {
    EXEC SQL WHENEVER SQLERROR GOTO falla;
    EXEC SQL OPEN GOA;
    if ((sal=fopen("GOR8004.lst","w"))==0)
      {
        printf("ERROR: Al Abrir el Archivo GOR8004.lst\n");
        exit(0);
      }
    EXEC SQL WHENEVER NOT FOUND GOTO fin_proceso;
    for( ; ; )
      {
        EXEC SQL FETCH GOA
        INTO :n_cta_lei,:n_conori_lei,
           :n_grupo_lei,:importe_acumulado,
           :importe_mensual,:importe_total;
           n_conori_lei.arr[n_conori_lei.len]='\0';
           n_grupo_lei.arr[n_grupo_lei.len]='\0';
```

```
        obtiene_concepto();
        if (primero == 1)
          {
           strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
           n_cta_ant=n_cta_lei;
           obtiene_cta();
           primero=0;
          }
        if(strcmp(n_grupo_lei.arr,n_grupo_ant.arr) != 0 )
          corte_grupo();
        if(n_cta_ant!=n_cta_lei)
          corte_cuenta();
        imprime_registro();
        total_grupo_acumulado+=importe_acumulado;
        total_grupo_mensual+=importe_mensual;
        total_grupo_total+=importe_total;
        total_cuenta_acumulado+=importe_acumulado;
        total_cuenta_mensual+=importe_mensual;
        total_cuenta_total+=importe_total;
        total_general_acumulado+=importe_acumulado;
        total_general_mensual+=importe_mensual;
        total_general_total+=importe_total;
      }
falla:
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      printf("\n**************************************************");
      printf("\nERROR (Proceso): %s",sqlca.sqlerrm.sqlerrmc);
      printf("\n**************************************************\n");
      EXEC SQL ROLLBACK WORK
      RELEASE;
      exit(0);
fin_proceso:
      EXEC SQL CLOSE GOA;
      EXEC SQL COMMIT WORK;
      EXEC SQL RELEASE;
      corte_grupo();
      corte_cuenta();
      total_general();
      fclose(sal);
      exit(0);
  }
main(argc,argv)
  int argc;
  char *argv[];
  {
      strcpy(usuario.arr,argv[1]);
      usuario.len=strlen(usuario.arr);
      usuario.arr[usuario.len]='\0';
      strcpy(clave.arr,argv[2]);
      clave.len=strlen(clave.arr);
      clave.arr[clave.len]='\0';
      strcpy(f_cierre.arr,argv[3]);
```

```
    f_cierre.len=strlen(f_cierre.arr);
    f_cierre.arr[f_cierre.len]='\0';
    mes.arr[0]=f_cierre.arr[2];
    mes.arr[1]=f_cierre.arr[3];
    mes.arr[2]='\0';
    ano.arr[0]=f_cierre.arr[4];
    ano.arr[1]=f_cierre.arr[5];
    ano.arr[2]='\0';
    mes.len=2;
    ano.len=2;
    switch (atoi(mes.arr))
      {
        case 1: strcpy(des_mes,"ENERO"); break;
        case 2: strcpy(des_mes,"FEBRERO"); break;
        case 3: strcpy(des_mes,"MARZO"); break;
        case 4: strcpy(des_mes,"ABRIL"); break;
        case 5: strcpy(des_mes,"MAYO"); break;
        case 6: strcpy(des_mes,"JUNIO"); break;
        case 7: strcpy(des_mes,"JULIO"); break;
        case 8: strcpy(des_mes,"AGOSTO"); break;
        case 9: strcpy(des_mes,"SEPTIEMBRE"); break;
        case 10: strcpy(des_mes,"OCTUBRE"); break;
        case 11: strcpy(des_mes,"NOVIEMBRE"); break;
        case 12: strcpy(des_mes,"DICIEMBRE"); break;
      }
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT :usuario IDENTIFIED BY :clave;
    obtiene_contaduria();
    proceso();
error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("\n*************************************************");
    printf("\nERROR (Inicio): %s",sqlca.sqlerrm.sqlerrmc);
    printf("\n*************************************************\n");
    EXEC SQL COMMIT WORK;
    EXEC SQL RELEASE;
    exit(0);
  }
```

La figura IV.7 muestra parte del reporte generado por el programa CTACON.pc, donde se observan las columnas con los movimientos acumulados y mensuales, y los cortes por grupo, cuenta y contaduría.

Con la opción "E" se obtiene un reporte con un resumen de conceptos de origen. En esta opción se corre el programa GOR8000.exe que pasa los parámetros de usuario, password y fecha de cierre al programa GOR8000.pc, el cual se muestra a continuación.

```
/* ========================================================================
   GOR8000.pc    - Reporte Global a nivel concepto de Origen.

   Accesos:      T_GOA[S]  CONOR[S]  CETRO[S]  T_041[S]
                 T_047[S]  T_052[S]  T_050[9]
```

```
  Parametros:    GOR8000  <Usuario> <Clave> <DDMMAA>

==========================================================================*/
#include <stdio.h>
char des_mes[20];
int  ren=80,primero=1,hoja=0;
FILE *sal;

EXEC SQL BEGIN DECLARE SECTION;

   VARCHAR usuario[10],clave[15];
   VARCHAR des_cpto[41],n_grupo_lei[3],n_grupo_ant[3],n_conori_lei[7];
   VARCHAR mes[3],ano[3],f_cierre[7],imp_importe[20],nom_conta[50];
   double  importe_acumulado,importe_mensual,importe_total;
   VARCHAR  wimporte_acumulado[15],wimporte_mensual[15],wimporte_total[15];
   int n_cenafo, n_cenafo_ant;
   double total_grupo_acumulado=0,total_grupo_mensual=0,total_grupo_total=0;
   double total_general_acumulado=0,total_general_mensual=0;
   double total_general_total=0;
   double total_ctro_acumulado=0,total_ctro_mensual=0,total_ctro_total=0;
   VARCHAR wtotal_grupo_acumulado[15],wtotal_grupo_mensual[15];
   VARCHAR wtotal_grupo_total[15];
   VARCHAR wtotal_general_acumulado[15],wtotal_general_mensual[15];
   VARCHAR wtotal_general_total[15];
   VARCHAR wtotal_ctro_acumulado[15],wtotal_ctro_mensual[15];
   VARCHAR wtotal_ctro_total[15];

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;
EXEC SQL DECLARE GOA  CURSOR FOR
Select n_conori,
     substr(n_conori,1,2),
     sum(nvl(i_op_acu,0)),
     sum(nvl(i_op_mes,0)),
     sum(nvl(i_op_acu,0)+nvl(i_op_mes,0)),n_cenafo
From  t_goa
Group  by n_cenafo,n_conori
order  by n_cenafo,n_conori ;

titulo()
  {
    hoja++;
    fprintf(sal,"\f%84s","P E T R O L E O S  M E X I C A N O S");
    fprintf(sal,"%30s  "," GOR8000  ");
    fprintf(sal,"  HOJA.- %3i\n ",hoja);
    fprintf(sal,"%84s\n","     SUBDIRECCION DE FINANZAS      ");
    fprintf(sal,"%86s\n","        GERENCIA DE CONTABILIDAD      ");
    fprintf(sal,"%76s\n",nom_conta.arr);
    fprintf(sal,"%77s","GASTOS DE ORIGEN  DEL MES DE ");
    fprintf(sal," %-20s\n\n",des_mes);
    fprintf(sal,"Centro   Descripcion del Concepto              ");
```

```
    fprintf(sal,"  Concepto       Saldo Anterior          ");
    fprintf(sal,"Movimientos        Saldo actual    \n\n");
    ren=13;
  }
imprime_registro()
  {
   if (ren >55)
     {
       titulo();
     }
    EXEC SQL SELECT to_char(:importe_acumulado,'999,999,999.99'),
                to_char(:importe_mensual,'999,999,999.99'),
                to_char(:importe_total,'999,999,999.99')
           INTO :wimporte_acumulado,:wimporte_mensual,:wimporte_total
           FROM DUAL;

    fprintf(sal,"%d%7s%40-s  %7s ",n_cenafo," ",des_cpto.arr,n_conori_lel.arr);
    fprintf(sal,"  %20s",wimporte_acumulado.arr);
    fprintf(sal,"   %20s",wimporte_mensual.arr);
    fprintf(sal,"   %20s\n",wimporte_total.arr);
    ren++;
  }

acumula()
  {
   total_grupo_acumulado+=importe_acumulado;
   total_grupo_mensual+=importe_mensual;
   total_grupo_total+=importe_total;
   total_general_acumulado+=importe_acumulado;
   total_general_mensual+=importe_mensual;
   total_general_total+=importe_total;
   total_ctro_acumulado+=importe_acumulado;
   total_ctro_mensual+=importe_mensual;
   total_ctro_total+=importe_total;
  }

corte_grupo()
  {
    EXEC SQL SELECT to_char(:total_grupo_acumulado,'999,999,999.99'),
                to_char(:total_grupo_mensual,'999,999,999.99'),
                to_char(:total_grupo_total,'999,999,999.99')
           INTO :wtotal_grupo_acumulado,:wtotal_grupo_mensual,
             :wtotal_grupo_total
           FROM DUAL;
    fprintf(sal,"\n         Total Grupo  ");
    fprintf(sal,"%37s  %20s"," ",wtotal_grupo_acumulado.arr);
    fprintf(sal,"   %20s",wtotal_grupo_mensual.arr);
    fprintf(sal,"   %20s\n\n",wtotal_grupo_total.arr);
    ren=ren+4;
    total_grupo_acumulado=0;
    total_grupo_mensual=0;
    total_grupo_total=0;
```

```
    strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
    if (ren >55)
     {
       titulo();
     }
  }

corte_centro()
  {
    EXEC SQL SELECT to_char(:total_ctro_acumulado,'999,999,999.99'),
                    to_char(:total_ctro_mensual,'999,999,999.99'),
                    to_char(:total_ctro_total,'999,999,999.99')
             INTO :wtotal_ctro_acumulado,:wtotal_ctro_mensual,
                  :wtotal_ctro_total
             FROM DUAL;
    fprintf(sal,"\n          Total Centro ");
    fprintf(sal,"%37s %20s"," ",wtotal_ctro_acumulado.arr);
    fprintf(sal,"   %20s",wtotal_ctro_mensual.arr);
    fprintf(sal,"   %20s\n\n",wtotal_ctro_total.arr);
    ren=ren+4;
    total_ctro_acumulado=0;
    total_ctro_mensual=0;
    total_ctro_total=0;
    n_cenafo_ant = n_cenafo;
    if (ren >55)
     {
       titulo();
     }
  }

total_general()
  {
    EXEC SQL SELECT to_char(:total_general_acumulado,'999,999,999.99'),
                    to_char(:total_general_mensual,'999,999,999.99'),
                    to_char(:total_general_total,'999,999,999.99')
             INTO :wtotal_general_acumulado,:wtotal_general_mensual,
                  :wtotal_general_total
             FROM DUAL;
    fprintf(sal,"\n          Total Contaduria ");
    fprintf(sal,"%33s  %20s"," ",wtotal_general_acumulado.arr);
    fprintf(sal,"   %20s",wtotal_general_mensual.arr);
    fprintf(sal,"   %20s\n",wtotal_general_total.arr);
    ren++;
    if (ren >55)
     {
       titulo();
     }
  }

obtiene_contaduria()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_conta;
```

```
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_contaduria;
    EXEC SQL SELECT nom_cont
          INTO  :nom_conta
          FROM  t_059
          WHERE  n_cont=(SELECT n_cont FROM t_041);
    nom_conta.arr[nom_conta.len]='\0';
    return(0);
sqlerr_conta:
    fprintf(sal," ERROR DE ORACLE AL OBTENER CONTADURIA\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
err_obt_contaduria:
    strcpy(nom_conta.arr,"******* NO EXISTE CONTADURIA  ********");
    return(0);
  }

obtiene_concepto()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr;
    EXEC SQL WHENEVER NOT FOUND GOTO busca_01;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
          INTO  :des_cpto
          FROM  CONOR
          WHERE  decode(substr(CVE_CPTO,1,2),
                   '00','06'||substr(CVE_CPTO,3,4),
                    CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
sqlerr:
    fprintf(sal," ERROR DE ORACLE AL OBTENER CONCEPTO\n");
    fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
    return(1);
busca_01:
    EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_con;
    EXEC SQL SELECT substr(DES_CPTO,1,40)
          INTO  :des_cpto
          FROM  CONOR
          WHERE  decode(substr(CVE_CPTO,1,2),
                   '00','06'||substr(CVE_CPTO,3,4),
                    CVE_CPTO)=:n_conori_lei;
    des_cpto.arr[des_cpto.len]='\0';
    return(0);
err_obt_con:
    strcpy(des_cpto.arr,"******  NO EXISTE CONCEPTO  ************");
    return(0);
  }

proceso()
  {
    EXEC SQL WHENEVER SQLERROR GOTO falla;
    EXEC SQL OPEN GOA;
```

```
    if ((sal=fopen("GOR8000.lst","w"))==0)
      {
        printf("ERROR: Al Abrir el Archivo GOR8000.lst\n");
        exit(0);
      }
    EXEC SQL WHENEVER NOT FOUND GOTO fin_proceso;
    for( ; ; )
      {
        EXEC SQL FETCH GOA
        INTO :n_conori_lei,:n_grupo_lei,:importe_acumulado,
           :importe_mensual,:importe_total,:n_cenafo;
           n_conori_lei.arr[n_conori_lei.len]='\0';
           n_grupo_lei.arr[n_grupo_lei.len]='\0';
        obtiene_concepto();

        if (primero == 1)
          {
            n_cenafo_ant = n_cenafo;
            strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
            primero=0;
          }

        if(n_cenafo_ant == n_cenafo)
        {
         if(strcmp(n_grupo_lei.arr,n_grupo_ant.arr) == 0 )
          {
           imprime_registro();
           acumula();
           strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
          }
          else
           {
            corte_grupo();
           imprime_registro();
           acumula();
           }
         }
        else
        {
          corte_grupo();
          corte_centro();
          imprime_registro();
          acumula();
        }
      }
falla:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("\n**********************************************");
    printf("\nERROR (Proceso): %s",sqlca.sqlerrm.sqlerrmc);
    printf("\n**********************************************\n");
    EXEC SQL ROLLBACK WORK
    RELEASE;
```

```
    exit(0);
fin_proceso:
    EXEC SQL CLOSE GOA;
    EXEC SQL COMMIT WORK;
    EXEC SQL RELEASE;
    corte_grupo();
    corte_centro();
    total_general();
    fclose(sal);
    exit(0);
 }
main(argc,argv)
  int argc;
  char *argv[];
  {
    strcpy(usuario.arr,argv[1]);
    usuario.len=strlen(usuario.arr);
    usuario.arr[usuario.len]='\0';
    strcpy(clave.arr,argv[2]);
    clave.len=strlen(clave.arr);
    clave.arr[clave.len]='\0';
    strcpy(f_cierre.arr,argv[3]);
    f_cierre.len=strlen(f_cierre.arr);
    f_cierre.arr[f_cierre.len]='\0';
    mes.arr[0]=f_cierre.arr[2];
    mes.arr[1]=f_cierre.arr[3];
    mes.arr[2]='\0';
    ano.arr[0]=f_cierre.arr[4];
    ano.arr[1]=f_cierre.arr[5];
    ano.arr[2]='\0';
    mes.len=2;
    ano.len=2;
    switch (atoi(mes.arr))
      {
        case 1: strcpy(des_mes,"ENERO"); break;
        case 2: strcpy(des_mes,"FEBRERO"); break;
        case 3: strcpy(des_mes,"MARZO"); break;
        case 4: strcpy(des_mes,"ABRIL"); break;
        case 5: strcpy(des_mes,"MAYO"); break;
        case 6: strcpy(des_mes,"JUNIO"); break;
        case 7: strcpy(des_mes,"JULIO"); break;
        case 8: strcpy(des_mes,"AGOSTO"); break;
        case 9: strcpy(des_mes,"SEPTIEMBRE"); break;
        case 10: strcpy(des_mes,"OCTUBRE"); break;
        case 11: strcpy(des_mes,"NOVIEMBRE"); break;
        case 12: strcpy(des_mes,"DICIEMBRE"); break;
      }
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT :usuario IDENTIFIED BY :clave;
    obtiene_contaduria();
    proceso();
error:
```

```
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("\n**************************************************");
    printf("\nERROR (Inicio): %s",sqlca.sqlerrm.sqlerrmc);
    printf("\n**************************************************\n");
    EXEC SQL COMMIT WORK;
    EXEC SQL RELEASE;
    exit(0);
}
```

En la figura IV.8 se muestra parte del reporte generado por el programa GOR8000.pc

Finalmente con la opción "F" se obtiene el reporte con el resumen de las cuentas de mayor o cuentas de operación . Para esta opción se corren los programas GOR8003.exe (ver apéndice A) el cual pasa los parámetros de usuario, password y fecha de cierre al programa ctagor.pc, el cual se muestra a continuación.

```
/* ================================================================
   ctagor.pc     - Reporte Global a nivel Numero de Cuenta.

   Accesos:       T_GOA[S]  CONOR[S]  CETRO[S]  T_041[S]
                  T_047[S]  T_052[S]  T_059[S]

   Parametros:    ctagor   <Usuario> <Clave> <DDMMAA>
================================================================*/
#include <stdio.h>
char des_mes[20];
int  ren=80,primero=1,hoja=0;
FILE *sal;

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR usuario[10],clave[15];
VARCHAR des_cpto[41],n_grupo_lei[3],n_grupo_ant[3],n_cta_lei[4];
VARCHAR des_ctro[50];
VARCHAR mes[3],ano[3],f_cierre[7],imp_importe[20],nom_conta[50];
double  importe_acumulado,importe_mensual,importe_total;
int     n_cenafo_ant,n_cenafo_lei;
double total_grupo_acumulado=0,total_grupo_mensual=0,total_grupo_total=0;
double total_general_acumulado=0,total_general_mensual=0,total_general_total=0;
double total_cenafo_acumulado=0,total_cenafo_mensual=0,total_cenafo_total=0;
VARCHAR wimporte_acumulado[15],wimporte_mensual[15],wimporte_total[15];
VARCHAR wtotal_grupo_acumulado[15],wtotal_grupo_mensual[15];
VARCHAR wtotal_grupo_total[15];
VARCHAR wtotal_general_acumulado[15],wtotal_general_mensual[15];
VARCHAR wtotal_general_total[15];
VARCHAR wtotal_cenafo_acumulado[15],wtotal_cenafo_mensual[15];
VARCHAR wtotal_cenafo_total[15];
EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca.h;
EXEC SQL DECLARE GOA  CURSOR FOR
Select n_cenafo, n_cta,
    sum(nvl(i_op_acu,0)),
```

```
    sum(nvl(i_op_mes,0)),
    sum(nvl(i_op_acu,0)+nvl(i_op_mes,0))
From  t_goa
Group  by n_cenafo, n_cta
Order  by n_cenafo, n_cta;

titulo()
  {
    hoja++;
    fprintf(sal,"\f%17s","CTAGOR.lst");
    fprintf(sal,"%60s","P E T R O L E O S  M E X I C A N O S");
    fprintf(sal,"%30s  "," ctagor.pc");
    fprintf(sal,"  HOJA.- %3i\n ",hoja);
    fprintf(sal,"%76s\n","     SUBDIRECCION DE FINANZAS      ");
    fprintf(sal,"%78s\n","    GERENCIA DE CONTABILIDAD      ");
    fprintf(sal,"%69s\n",nom_conta.arr);
    fprintf(sal,"%70s","GASTOS DE ORIGEN  DEL MES DE ");
    fprintf(sal," %-23s\n\n",des_mes);
    fprintf(sal,"                                    ");
    fprintf(sal,"%2s","CENTRO :  ");
    fprintf(sal,"%3.0d  %-50s\n\n",n_cenafo_ant,des_ctro.arr);
    fprintf(sal,"                                    ");
    fprintf(sal," Cuenta       Saldo Anterior          ");
    fprintf(sal,"Movimientos          Saldo actual     \n\n");
    ren=16;
  }
imprime_registro()
  {
    if (ren >55)
     {
       titulo();
     }
    EXEC SQL SELECT to_char(:importe_acumulado,'999,999,999.99'),
                to_char(:importe_mensual,'999,999,999.99'),
                to_char(:importe_total,'999,999,999.99')
             INTO :wimporte_acumulado,:wimporte_mensual,:wimporte_total
             FROM DUAL;

    fprintf(sal," %40-s     %4-s",des_cpto.arr,n_cta_lel.arr);
    fprintf(sal,"  %20s",wimporte_acumulado.arr);
    fprintf(sal,"    %20s",wimporte_mensual.arr);
    fprintf(sal,"    %20s\n",wimporte_total.arr);
    ren++;
  }

corte_cenafo()
  {
    EXEC SQL SELECT to_char(:total_cenafo_acumulado,'999,999,999.99'),
                to_char(:total_cenafo_mensual,'999,999,999.99'),
                to_char(:total_cenafo_total,'999,999,999.99')
             INTO :wtotal_cenafo_acumulado,:wtotal_cenafo_mensual,
                :wtotal_cenafo_total
```

```
            FROM DUAL;
    fprintf(sal,"\n          Total Centro  ");
    fprintf(sal,"%26s  %20s"," ",wtotal_cenafo_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_cenafo_mensual.arr);
    fprintf(sal,"    %20s\n\n",wtotal_cenafo_total.arr);
    ren=80;
    total_cenafo_acumulado=0;
    total_cenafo_mensual=0;
    total_cenafo_total=0;
    n_cenafo_ant=n_cenafo_lei;
    obtiene_cetro();
  }

corte_cuenta()
  {
    EXEC SQL SELECT to_char(:total_grupo_acumulado,'999,999,999.99'),
                    to_char(:total_grupo_mensual,'999,999,999.99'),
                    to_char(:total_grupo_total,'999,999,999.99')
             INTO :wtotal_grupo_acumulado,:wtotal_grupo_mensual,
                  :wtotal_grupo_total
             FROM DUAL;
    fprintf(sal,"\n          Total Contaduria  ");
    fprintf(sal,"  %20s",wtotal_grupo_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_grupo_mensual.arr);
    fprintf(sal,"    %20s\n\n",wtotal_grupo_total.arr);
    ren=80;
    total_grupo_acumulado=0;
    total_grupo_mensual=0;
    total_grupo_total=0;
    strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
    if (ren >55)
      {
        titulo();
      }
  }

total_general()
  {
    EXEC SQL SELECT to_char(:total_general_acumulado,'999,999,999.99'),
                    to_char(:total_general_mensual,'999,999,999.99'),
                    to_char(:total_general_total,'999,999,999.99')
             INTO :wtotal_general_acumulado,:wtotal_general_mensual,
                  :wtotal_general_total
             FROM DUAL;

    fprintf(sal,"\n          Total Reporte ");
    fprintf(sal,"%26s  %20s"," ",wtotal_general_acumulado.arr);
    fprintf(sal,"    %20s",wtotal_general_mensual.arr);
    fprintf(sal,"    %20s\n",wtotal_general_total.arr);
  }

obtiene_cetro()
```

```
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_ctro;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_ctro;
    EXEC SQL SELECT des_ctro
          INTO  :des_ctro
          FROM  cetro
          WHERE  cve_ctro=:n_cenafo_ant;
          des_ctro.arr[des_ctro.len]='\0';
    return(0);
    sqlerr_ctro:
        fprintf(sal," ERROR DE ORACLE AL OBTENER EL CENTRO  \n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    err_obt_ctro:
        strcpy(des_ctro.arr,"*******   NO EXISTE CENTRO   *********");
        return(0);
  }

obtieno_contaduria()
  {
    EXEC SQL WHENEVER SQLERROR  GOTO sqlerr_conta;
    EXEC SQL WHENEVER NOT FOUND GOTO err_obt_contaduria;
    EXEC SQL SELECT nom_cont
          INTO  :nom_conta
          FROM  t_059
          WHERE  n_cont=(SELECT n_cont FROM t_041);
    nom_conta.arr[nom_conta.len]='\0';
    return(0);
    sqlerr_conta:
        fprintf(sal," ERROR DE ORACLE AL OBTENER CONTADURIA\n");
        fprintf(sal," ERROR NUMERO %s \n",sqlca.sqlerrm.sqlerrmc);
        return(1);
    err_obt_contaduria:
        strcpy(nom_conta.arr,"******* NO EXISTE CONTADURIA   *********");
        return(0);
  }

proceso()
  {
    EXEC SQL WHENEVER SQLERROR GOTO falla;
    EXEC SQL OPEN GOA;
    if ((sal=fopen("CTAGOR.lst","w"))==0)
      {
        printf("ERROR: Al Abrir el Archivo CTAGOR.lst\n");
        exit(0);
      }
    EXEC SQL WHENEVER NOT FOUND GOTO fin_proceso;
    for( ; ; )
      {
        EXEC SQL FETCH GOA
        INTO :n_cenafo_lei,:n_cta_lei,:importe_acumulado,
           :importe_mensual,:importe_total;
```

```
        n_cta_lei.arr[n_cta_lei.len]='\0';
        n_grupo_lei.arr[n_grupo_lei.len]='\0';

        if (primero == 1)
          {
           strcpy(n_grupo_ant.arr,n_grupo_lei.arr);
           n_cenafo_ant=n_cenafo_lei;
           obtiene_cetro();
           primero=0;
          }

        if(strcmp(n_grupo_lei.arr,n_grupo_ant.arr) != 0 )
          corte_cuenta();
        if(n_cenafo_ant!=n_cenafo_lei)
          corte_cenafo();
        imprime_registro();
        total_cenafo_acumulado+=importe_acumulado;
        total_cenafo_mensual+=importe_mensual;
        total_cenafo_total+=importe_total;
        total_grupo_acumulado+=importe_acumulado;
        total_grupo_mensual+=importe_mensual;
        total_grupo_total+=importe_total;
        total_general_acumulado+=importe_acumulado;
        total_general_mensual+=importe_mensual;
        total_general_total+=importe_total;
      }
falla:
      EXEC SQL WHENEVER SQLERROR CONTINUE;
      printf("\n**************************************************");
      printf("\nERROR (Proceso): %s",sqlca.sqlerrm.sqlerrmc);
      printf("\n**************************************************\n");
      EXEC SQL ROLLBACK WORK
      RELEASE;
      exit(0);
fin_proceso:
      EXEC SQL CLOSE GOA;
      EXEC SQL COMMIT WORK;
      EXEC SQL RELEASE;
      corte_cenafo();
      total_general();
      fclose(sal);
      exit(0);
  }
main(argc,argv)
   int argc;
   char *argv[];
   {
      strcpy(usuario.arr,argv[1]);
      usuario.len=strlen(usuario.arr);
      usuario.arr[usuario.len]='\0';
      strcpy(clave.arr,argv[2]);
```

```
    clave.len=strlen(clave.arr);
    clave.arr[clave.len]='\0';
    strcpy(f_cierre.arr,argv[3]);
    f_cierre.len=strlen(f_cierre.arr);
    f_cierre.arr[f_cierre.len]='\0';
    mes.arr[0]=f_cierre.arr[2];
    mes.arr[1]=f_cierre.arr[3];
    mes.arr[2]='\0';
    ano.arr[0]=f_cierre.arr[4];
    ano.arr[1]=f_cierre.arr[5];
    ano.arr[2]='\0';
    mes.len=2;
    ano.len=2;
    switch (atoi(mes.arr))
      {
        case 1: strcpy(des_mes,"ENERO"); break;
        case 2: strcpy(des_mes,"FEBRERO"); break;
        case 3: strcpy(des_mes,"MARZO"); break;
        case 4: strcpy(des_mes,"ABRIL"); break;
        case 5: strcpy(des_mes,"MAYO"); break;
        case 6: strcpy(des_mes,"JUNIO"); break;
        case 7: strcpy(des_mes,"JULIO"); break;
        case 8: strcpy(des_mes,"AGOSTO"); break;
        case 9: strcpy(des_mes,"SEPTIEMBRE"); break;
        case 10: strcpy(des_mes,"OCTUBRE"); break;
        case 11: strcpy(des_mes,"NOVIEMBRE"); break;
        case 12: strcpy(des_mes,"DICIEMBRE"); break;
      }
    EXEC SQL WHENEVER SQLERROR GOTO error;
    EXEC SQL CONNECT :usuario IDENTIFIED BY :clave;
    obtiene_contaduria();
    proceso();
error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    printf("\n**************************************************");
    printf("\nERROR (Inicio): %s",sqlca.sqlerrm.sqlerrmc);
    printf("\n**************************************************\n");
    EXEC SQL COMMIT WORK;
    EXEC SQL RELEASE;
    exit(0);
}
```

La figura IV.9 muestra el reporte del programa ctagor.pc, que es el resumen por número de cuenta.

Los reportes mencionados anteriormente permite obtener toda la información necesaria en lo que se refiere a gastos efectuados en los centros de trabajo. Estos reportes ayudan a identificar en que se gasta mas o en que se gasta menos, ya sea a un nivel particular como es el concepto de origen hasta un nivel mas general como lo es un centro de trabajo

CONCLUSIONES

En la actualidad existen muchas opciones para el desarrollo de aplicaciones y sistemas, hay una gran variedad de bases de datos, lenguajes de programación y paquetería en general. El precompilador Pro*C de la Base de Datos ORACLE es una de estas opciones.

Esta Base de Datos posee características muy importantes como: representación de datos en forma de tablas, utiliza un lenguaje de cuarta generación que es el SQL, tiene capacidades relacionales completas, flexibilidad al cambiar datos y facilidad para modificar estructuras, diccionario de datos integrado, mantiene en todo momento la integridad de información y es una Base de Datos con sistema multiproceso ya que puede ser accesada por más de un usuario al mismo tiempo.

Un programa hecho en Pro*C utiliza argumentos de SQL , que es un lenguaje flexible, poderoso y fácil de entender, por lo que su uso dentro de un programa en C resulta fácil de entender.

El precompilador Pro*C es una herramienta orientada a diseñar y desarrollar aplicaciones para usuarios. Realizar un programa en Pro*C no difiere mucho a realizar un programa en lenguaje C, en general la diferencia es que maneja argumentos de SQL, pero con un Pro*C se tiene la posibilidad de interactuar con la Base de Datos directamente seleccionando, borrando, insertando, modificando datos, alterando tablas o incluso crearlas o borrarlas. Las instrucciones de C y los argumentos de SQL están bien definidos dentro de lo que es el cuerpo del programa.

Cuando cambian las versiones de ORACLE y de Pro*C los programas desarrollados en una versión anterior varían poco y pueden ser compilados nuevamente, pueden variar algunas características, como por ejemplo, aumenta el número de cursores abiertos, maneja mas variables para errores o aumenta el tipo de variables que puede manejar.

Algunas opciones se vuelven obsoletas, como las opciones Areasize o Rebind ya que las áreas de contexto son automáticamente inicializadas, y las variables son reubicadas cuando es necesario.

Otras instrucciones son mejoradas, la intruccion WHENEVER ahora tiene la opción de ejecutar una rutina, cuando se presenta la condición definida por dicha instrucción.

La tendencia actual es dejar un poco la programación con lenguajes como el FORTRAN o COBOL y desarrollar con lenguajes estructurales como C o PASCAL. En el presente trabajo se mostraron las características del Pro*C como un camino mas que se puede tomar para la programación, y que ha sido poco difundido por las necesidades de equipo que necesita, pero que es muy importante conocer.

APENDICE A

Los programas que se muestran a continuación están realizados en lenguaje C y shell. Estos programas fueron mencionados en el capítulo IV y forman parte del sistema explicado.

```
## GOR3001.exe  Shell que corre los programas para la carga de información ##
clear
echo '\033"[11;20H"\033[1m EJECUTANDO PROCESO DE CARGA DE SIC \033[0m'
sleep 4
fecini=$1
fecfin=$2
mmyy=`echo $2|cut -c3-6`
  mm=`echo $2|cut -c3-4`
  yy=`echo $2|cut -c5-6`
yymm=`echo $yy$mm`
echo '_____'                                                    >>job
echo '|    Inicio de Carga de SIC a Gastos de Origen          |' >>job
echo '|            GOR3001.exe                                 |' >>job
date '+|   Fecha: %d-%m-%y      Hora: %H:%M:%S                 |' >>job
echo '|_____|'                                                  >>job
fecini=$fecini
fecfin=$fecfin
mmyy=$mmyy
mm=$mm
yy=$yy
yymm=$yymm
echo ' Fecha de Inicial de Carga = '$fecini                     >>job
echo ' Fecha de Final de Carga   = '$fecfin                     >>job
echo ' Fecha de Cierre Mensual   = '$fecfin                     >>job
echo ' '                                                        >>job
echo '  Ejecucion del SQL REGBA305.sql          '               >>job
echo '  Generacion de la Informacion del SIC para el GOR'       >>job
echo ' '                                                        >>job
echo ' '                                                        >>job
REGBA305 $GOR_USR $GOR_PWD $mmyy                    >>job 2>>job
echo ' '                                                        >>job
echo ' '                                                        >>job
echo ' '                                                        >>job
actsic $GOR_USR $GOR_PWD                            >>job 2>>job
echo ' '                                                        >>job
echo '  Ejecucion del SQL GOR3001.pc            '               >>job
echo '  Carga de SIC al Sistema de Gastos de Origen '           >>job
echo ' '                                                        >>job
GOR3001 $GOR_USR $GOR_PWD $fecfin                   >>job 2>>job
echo ' '                                                        >>job
echo ' Actualizacion de Acumulados en T_GOA'                    >>job
echo ' '                                                        >>job
echo '  Ejecucion del SQL GOR7001.pc            '               >>job
GOR7001 $GOR_USR $GOR_PWD  $fecini $fecfin          >>job 2>>job
echo ' '                                                        >>job
echo '  Ejecucion del SQL GOR7011.pc            '               >>job
echo ' '                                                        >>job
GOR7011 $GOR_USR $GOR_PWD $fecfin                   >>job 2>>job
echo ' '                                                        >>job
echo '_____'                                                    >>job
```

```
echo '|     Fin Carga de SIC a Gastos de Origen                    |'    >>job
date '+|  Fecha: %d-%m-%y       Hora: %H:%M:%S                    |'    >>job
echo '_____________________________________________________________'    >>job
cat $HOME/tmp/job > $HOME/tmp/day_GOR3001 2>> $HOME/tmp/day_GOR3001
lp -t\"day_GOR3001\" $HOME/tmp/day_GOR3001


## GOR0001.exe Shell que ejecuta el proceso de inicialización de información ##
clear
echo '\033"[11;20H"\033[1m  EJECUTANDO PROCESO DE INICIALIZACION\033[0m'
sleep 4
fecini=$1 fecfin=$2
echo '_____________________________________________________________'    >>job
echo '|   Inicio de Inicializacion de Tablas para Reproceso         |'  >>job
echo '|            GOR0001.exe                                      |'  >>job
date '+|  Fecha: %d-%m-%y       Hora: %H:%M:%S                    |'    >>job
echo '_____________________________________________________________'    >>job
fecini=$fecini fecfin=$fecfin
echo 'fecha de inicio='$fecini' fecha final='$fecfin                    >>job
echo ' '                                                                >>job
echo '  Ejecucion del sql GOR0001.sql                '                  >>job
echo ' '                                                                >>job
echo \"start $GOR_AMB/bin/GOR0001 $fecini $fecfin\"|sqlplus -silent $GOR_USR/$GOR_PWD
echo '_____________________________________________________________'    >>job
echo '|     Fin de Inicializacion de Tablas para Reproceso          |'  >>job
date '+|  Fecha: %d-%m-%y       Hora: %H:%M:%S                    |'    >>job
echo '_____________________________________________________________'    >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR0001
lp -t\"day GOR0001\" $HOME/tmp/day_GOR0001


## GOR1103.exe. Shell que ejecuta el proceso de importación de la información ##
clear
echo '\033"[11;18H"\033[1m EJECUTANDO PROCESO DE IMPORTACION\033[0m'
sleep 4
echo '_____________________________________________________________'    >>job
echo '|   Inicio del Import de la Informacion                       |'  >>job
echo '|            GOR1103.exe                                      |'  >>job
date '+|  Fecha: %d-%m-%y       Hora: %H:%M:%S                    |'    >>job
echo '_____________________________________________________________'    >>job
echo ' '                                                                >>job
        if test ! -f $HOME/GOR$1.dmp
        then
          echo 'No existe el archivo $HOME/GOR$1.dmp '
                break;exit
        fi
cp $HOME/GOR$1.dmp $GOR_AMB/bin/.
echo \"delete T_GOR;\"|  sqlplus -silent tj40010/adm_gor          >>job
echo \"delete T_GOA;\"|  sqlplus -silent tj40010/adm_gor          >>job
imp tj40010/adm_gor <<-!                                        >>job 2>>job
GOR$1
40960
N
Y
Y
Y
```

```
Y
!
rm GOR$1.dmp
echo '_____'  >>job
echo '|    Fin  del Import de la Informacion                    |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                    |'  >>job
echo '_____'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR1103
lp -t\"day GOR1103\" $HOME/tmp/day_GOR1103


### GOR1104.exe . Shell que ejecuta el proceso de exportación de la información ##
clear
echo '\033"[11;18H"\033[1m EJECUTANDO PROCESO DE EXPORTACION\033[0m'
sleep 4
echo '_____'  >job
echo '|   Inicio del Export de la Informacion  ( Movimientos )   |' >>job
echo '|              GOR1104.exe                                  |' >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                    |' >>job
echo '_____' >>job
echo ' ' >>job
exp t|40010/adm_gor <<-!                                  >>job 2>>job
40960
GOR$1
T
Y
Y
t_goa
!
#mv GOR$1.dmp $HOME/.
mv GOR$1.dmp /prd/gor/t|40010/exp/goa$1.dmp                 >>job
echo '_____'  >>job
echo '|    Fin  del Export de la Informacion                    |' >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                    |' >>job
echo '_____' >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR1104
lp -t\"day GOR1104\" $HOME/tmp/day_GOR1104


## GOR7002.exe . Shell que ejecuta el programa de cierre mensual ##
clear
echo '\033"[11;18H"\033[1m EJECUTANDO PROCESO MENSUAL (definitivo)\033[0m'
sleep 4
echo '_____'  >job
echo '|   Inicio de Proceso Mensual Definitivo                   |'  >>job
echo '|              GOR7002.exe                                  |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                    |'  >>job
echo '_____'  >>job
echo ' '                                                          >>job
echo ' Ejecucion del PRO*C GOR7002.pc                              >>job
echo ' Actualiza la Tabla T_GOA '                                  >>job
echo ' '                                                          >>job
$GOR_AMB/bin/GOR7002 $GOR_USR $GOR_PWD
echo '_____'  >>job
echo '|    Fin  de Proceso Mensual Definitivo                    |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                    |'  >>job
```

```
echo '_____'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR7002
lp -t\"day GOR7002\" $HOME/tmp/day_GOR7002


## GOR8000.exe . Shell que ejecuta el programa GOR8000.pc de reportes ##
clear
echo '\033"[11;22H"\033[1m EJECUTANDO REPORTE POR CONCEPTO DE ORIGEN\033[0m'
sleep 4
fecha=$1
echo '_____'  >>job
echo '|   Inicio de Reporte Resumen por Concepto de Origen                |'  >>job
echo '|              GOR8000.exe                                           |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                               |'  >>job
echo '_____'  >>job
fecha=$fecha
echo ' Fecha de Cierre Mensual = '$fecha                                    >>job
echo ' '                                                                    >>job
echo ' Ejecucion del PC GOR8000.pc                                          >>job
echo ' Carga de SIC al Sistema de Gastos de Origen '                        >>job
echo ' '                                                                    >>job
$GOR_AMB/bin/GOR8000 $GOR_USR $GOR_PWD \$fecha                              >>job 2>>job
mv GOR8000.lst $HOME/tmp/.
lp -t\"sigor-vc\" $HOME/tmp/GOR8000.lst
echo '_____'  >>job
echo '|     Fin de Reporte Resumen por Concepto de Origen                 |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                               |'  >>job
echo '_____'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR8000
lp -t\"day GOR8000\" $HOME/tmp/day_GOR8000


## GOR8001.exe . Shell que ejecuta el programa GOR8001 de reportes ##
clear
echo '\033"[11;22H"\033[1m EJECUTANDO REPORTE POR CUENTA Y CONCEPTO\033[0m'
sleep 4
fecha=$1
echo '_____'  >>job
echo '|   Inicio de Reporte por Cuenta,Subcuenta y Cpto.de Origen          |'  >>job
echo '|              GOR8001.exe                                           |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                               |'  >>job
echo '_____'  >>job
fecha=$fecha
echo ' Fecha de Cierre Mensual = '$fecha                                    >>job
echo ' '                                                                    >>job
echo ' Ejecucion del PC GOR8001.pc                                          >>job
echo ' Carga de SIC al Sistema de Gastos de Origen '                        >>job
echo ' '                                                                    >>job
$GOR_AMB/bin/GOR8001 $GOR_USR $GOR_PWD \$fecha                              >>job 2>>job
mv GOR8001.lst $HOME/tmp/.
lp -t\"sigor-vc\" $HOME/tmp/GOR8001.lst
echo '_____'  >>job
echo '|     Fin de Reporte por Cuenta,Subcuenta y Cpto.de Origen          |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                               |'  >>job
echo '_____'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR8001
```

```
lp -t\"day GOR8001\" $HOME/tmp/day_GOR8001


## GOR8002.exe . Shell que ejecuta el programa GOR8002 de reportes ##
clear
echo'\033"[11;20H"\033[1m EJECUTANDO REPORTE POR CENTRO Y DEPARTAMENTO\033[0m'
sleep 4
fecha=$1
echo '_____'                                                    >>job
echo '|   Inicio de Reporte por Centro y Departamento          |'  >>job
echo '|            GOR8002.exe                                 |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                  |'  >>job
echo '_____'                                                    >>job
fecha=$fecha
echo ' Fecha de Cierre Mensual = '$fecha
echo ' '                                                       >>job
echo '  Ejecucion del PC GOR8002.pc                             >>job
echo '  Carga de SIC al Sistema de Gastos de Origen '           >>job
echo ' '                                                       >>job
$GOR_AMB/bin/GOR8002 $GOR_USR $GOR_PWD \$fecha                  >>job 2>>job
mv GOR8002.lst $HOME/tmp/.
lp -t\"sigor-vc\" $HOME/tmp/GOR8002.lst
echo '_____'                                                    >>job
echo '|     Fin de Reporte por Centro y Departamento           |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                  |'  >>job
echo '_____'                                                    >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR8002
lp  -t\"day GOR8002\" $HOME/tmp/day_GOR8002

## GOR8003.exe . Shell que ejecuta el programa GOR8003 de reportes ##
clear
echo '\033"[11;22H"\033[1m EJECUTANDO REPORTE DE RESUMEN POR CUENTA\033[0m'
sleep 4
fecha=$1
echo '_____'                                                    >>job
echo '|   Inicio de Reporte de Resumen por Cuenta              |'  >>job
echo '|            GOR8003.exe                                 |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                  |'  >>job
echo '_____'                                                    >>job
fecha=$fecha
echo ' Fecha de Cierre Mensual = '$fecha                       >>job
echo ' '                                                       >>job
echo '  Ejecucion del PC ctagor.pc                              >>job
echo '  Carga de SIC al Sistema de Gastos de Origen '           >>job
echo ' '                                                       >>job
$GOR_AMB/bin/ctagor $GOR_USR $GOR_PWD \$fecha                  >>job 2>>job
mv CTAGOR.lst $HOME/tmp/.
lp  -t\"sigor-vc\" $HOME/tmp/CTAGOR.lst
echo '_____'                                                    >>job
echo '|     Fin de Reporte de Resumen por Cuenta               |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                  |'  >>job
echo '_____'                                                    >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR8003
lp  -t\"day GOR8003\" $HOME/tmp/day_GOR8003

## GOR8004.exe . Shell que ejecuta el programa GOR8004 de reportes ##
```

```
clear
echo '\033"[11;22H"\033[1m EJECUTANDO REPORTE POR CUENTA Y CONCEPTO\033[0m'
sleep 4
fecha=$1
echo '_____'  >>job
echo '|   Inicio de Reporte por Cuenta y Concepto                    |'  >>job
echo '|            GOR8004.exe                                       |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                         |'  >>job
echo '|______________________________________________________________|'  >>job
fecha=$fecha
echo ' Fecha de Cierre Mensual = '$fecha                              >>job
echo ' '                                                              >>job
echo '  Ejecucion del PC CTACON.pc                '                   >>job
echo '  Carga de SIC al Sistema de Gastos de Origen '                 >>job
echo ' '                                                              >>job
$GOR_AMB/bin/CTACON $GOR_USR $GOR_PWD \$fecha                      >>job 2>>job
mv GOR8004.lst $HOME/tmp/.
lp -t\"sigor-vc\" $HOME/tmp/GOR8004.lst
echo '_____'  >>job
echo '|     Fin de Reporte por Cuenta y Concepto                    |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                         |'  >>job
echo '|______________________________________________________________|'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR8004
lp -t\"day GOR8004\" $HOME/tmp/day_GOR8004


## GOR8005.exe . Shell que ejecuta el programa GOR8005 de reportes ###
clear
echo '\033"[11;22H"\033[1m EJECUTANDO REPORTE POR CUENTA Y CONCEPTO\033[0m'
sleep 4
fecha=$1
echo '_____'  >>job
echo '|   Inicio de Reporte por Cuenta y Renglon del Gasto           |'  >>job
echo '|            GOR8005.exe                                       |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                         |'  >>job
echo '|______________________________________________________________|'  >>job
fecha=$fecha
echo ' Fecha de Cierre Mensual = '$fecha                              >>job
echo ' '                                                              >>job
echo '  Ejecucion del PC GOR8005.pc                '                  >>job
echo '  Carga de SIC al Sistema de Gastos de Origen '                 >>job
echo ' '                                                              >>job
$GOR_AMB/bin/GOR8005 $GOR_USR $GOR_PWD \$fecha                     >>job 2>>job
mv GOR8005.lst $HOME/tmp/.
lp -t\"sigor-vc\" $HOME/tmp/GOR8005.lst
echo '_____'  >>job
echo '|     Fin de Reporte por Cuenta y Renglon del Gasto           |'  >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                         |'  >>job
echo '|______________________________________________________________|'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR8005
lp -t\"day GOR8005\" $HOME/tmp/day_GOR8005


## GOR8006.exe ##
clear
echo '\033"[11;20H"\033[1m EJECUTANDO REPORTE POR RENGLON DEL GASTO\033[0m'
```

```
sleep 4
fecha=$1
echo '_____'  >job
echo '|  Inicio de Reporte por Renglon del Gasto                    |' >>job
echo '|              GOR8006.exe                                    |' >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                       |' >>job
echo '_____'  >>job
fecha=$fecha
echo ' Fecha de Cierre Mensual = '$fecha                             >>job
echo ' '                                                            >>job
echo '  Ejecucion del PC GOR8006.pc                '                 >>job
echo '  Carga de SIC al Sistema de Gastos de Origen '                >>job
echo ' '                                                            >>job
$GOR_AMB/bin/GOR8006 $GOR_USR $GOR_PWD \$fecha                      >>job 2>>job
mv GOR8006.lst $HOME/tmp/.
lp -t\"sigor-vc\" $HOME/tmp/GOR8006.lst
echo '_____'  >>job
echo '|     Fin de Reporte por Renglon del Gasto                    |' >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                       |' >>job
echo '_____'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_GOR8006
rm $HOME/tmp/job
lp -t\"day GOR8006\" $HOME/tmp/day_GOR8006


### gor1104.exe . Shell que ejecuta el proceso de exportación de la información ##
clear
echo '\033"[11;18H"\033[1m EJECUTANDO PROCESO DE EXPORTACION\033[0m'
sleep 4
echo '_____'  >job
echo '|  Inicio del Export de la Informacion  ( Definitivo )        |' >>job
echo '|           gor1104.exe                                       |' >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                       |' >>job
echo '_____'  >>job
echo ' '                                                            >>job
exp tj40010/adm_gor <<-! >>$job 2>>job
40960
GOR$1
T
Y
Y
t_goa
!
mv GOR$1.dmp /prd/gor/tj40010/exp/goA$1.dmp
echo '_____'  >>job
echo '|     Fin  del Export de la Informacion                       |' >>job
date '+|  Fecha: %d-%m-%y      Hora: %H:%M:%S                       |' >>job
echo '_____'  >>job
cat $HOME/tmp/job >> $HOME/tmp/day_gor1104
rm $HOME/tmp/job
lp  -t\"day gor1104\" $HOME/tmp/day_gor1104


/* ================================================================*/
/*adminis.c. Programa en C que muestra el menu de administración */
/* ================================================================*/
```

```
#include "men.c"
char fe1[7],fe2[7],oper[50];

menu001()
  {
    setraw();
    i=1;
    for (j=1;j<=4;j++)
       matriz[i][j] = 0;
    dimx = 1;
    dimy = 8;
    renglon = 1;
    columna = 1;
    funcion = 'A';
    matriz[1][1] = 'A';
    matriz[1][2] = 'B';
    matriz[1][3] = 'C';
    matriz[1][4] = 'D';
    matriz[1][6] = 'Z';
    strcpy(texto[1].c,"A. Importacion de la Informacion");
    strcpy(texto[2].c,"B. Export. Inf. de Movimientos del mes");
    strcpy(texto[3].c,"C. Export. Inf. de Proceso Definitivo");
    strcpy(texto[4].c,"D. Iniciacion de la Informacion");
    strcpy(texto[6].c,"Z. Regresar al Menu Principal");
    titulos();
    descripcion("Administracion");
    gotoxy(10,21);printf(texto[1].c);
    gotoxy(11,21);printf(texto[2].c);
    gotoxy(12,21);printf(texto[3].c);
    gotoxy(13,21);printf(texto[4].c);
    gotoxy(15,21);printf(texto[6].c);
    BRI;
    gotoxy(19,18);printf(" Seleccione una opcion ");
    NOR;
    cursor(21,10,6);
    switch (funcion)

     {
       case 'A':
          restore();
          opcion_A();
          restore();
          break;
       case 'B':
          restore();
          opcion_B();
          restore();
          break;
       case 'C':
          restore();
          opcion_C();
          restore();
          break;
       case 'D':
          restore();
```

```
            opcion_D();
            restore();
            break;
          case 'Z':
            clear();
            restore();
            gotoxy(1,1);
            exit(0);
      }
   }

opcion_A()
{
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Importacion de la informacion");
    gotoxy(9,17); printf("Fecha de cierre del mes(DDMMAA) ===> DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR1103.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

     sino_datos();
  }
    sino_job_int();
}
opcion_B()
{
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Exportacion de la informacion (Movimientos)");
    gotoxy(9,17); printf("Fecha de cierre del mes(DDMMAA) ===> DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR1104.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

     sino_datos();
  }
    sino_job_int();
}
opcion_C()
{
  resp = 'n';
  while ( resp != 's' )
   {
```

```
    procesos("Exportacion de la informacion (Definitivo)");
    gotoxy(9,17); printf("Fecha de cierre del mes(DDMMAA)  ===>  DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"gor1104.exe ");

      valida_ddmmaa(fe1,9,56);
      strcat(comando,fe1); strcat(comando," ");

      sino_datos();
   }
    sino_job_int();
}
opcion_D()
{
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Inicializacion de informacion");
    gotoxy(9,17); printf("Fecha inicial del mes  (DDMMAA)  ===>  DDMMAA");
    gotoxy(10,17); printf("Fecha final del mes   (DDMMAA)  ===>  DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR0001.exe ");

      valida_ddmmaa(fe1,9,56);
      strcat(comando,fe1); strcat(comando," ");

      valida_ddmmaa(fe2,10,56);
      strcat(comando,fe2); strcat(comando," ");

      sino_datos();
   }
    sino_job_int();
}
/*****************************************************************/
/*                    PROGRAMA PRINCIPAL                         */
/*****************************************************************/
main()
   {
     while (1)
      {
       menu001();
      }
   }


/* ================================================================*/
/*cierre.c. Programa en C que muestra el menu de cierre mensual */
/* ================================================================*/

#include "menu.uti"
```

```
    procesos("Exportacion de la informacion (Definitivo)");
    gotoxy(9,17); printf("Fecha de cierre del mes(DDMMAA) ===> DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"gor1104.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

     sino_datos();
  }
    sino_job_int();
}
opcion_D()
{
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Inicializacion de informacion");
    gotoxy(9,17); printf("Fecha inicial del mes  (DDMMAA) ===> DDMMAA");
    gotoxy(10,17); printf("Fecha final del mes   (DDMMAA) ===> DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR0001.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

     valida_ddmmaa(fe2,10,56);
     strcat(comando,fe2); strcat(comando," ");

     sino_datos();
  }
    sino_job_int();
}
/*****************************************************************/
/*                  PROGRAMA PRINCIPAL                           */
/*****************************************************************/
main()
   {
    while (1)
     {
      menu001();
     }
   }

/* ================================================================*/
/*cierre.c. Programa en C que muestra el menu de cierre mensual */
/* ================================================================*/

#include "menu.util"
```

```
char ano[3],mes[3], dia[3] ,mesant[3],impresion[2];
char cinta[7],num[3],pob1[6],pob2[6],pob3[6];
char pob4[6],pob5[6],pob6[6],po1[6],po2[6],po3[6],fe1[7],fe2[7],fe3[7];
char ci1[7],ci2[7],ci3[7];
int status;

cierre()
  {
     setraw();
       i=1;
       for (j=1;j<=5;j++)
           matriz[i][j] = 0;
     dimx = 1;
     dimy = 5;
     renglon = 1;
     columna = 1;
     funcion = 'A';
     matriz[1][1] = 'A';
     matriz[1][4] = 'Z';
     strcpy(texto[1].c," A. Actualizacion de Acumulados");
     strcpy(texto[4].c," Z. Regresar al menu anterior");
     titulos();
     descripcion("Cierre Mensual");
     gotoxy(10,25);printf(texto[1].c);
     gotoxy(13,25);printf(texto[4].c);
     BRI;
     gotoxy(21,22);printf(" Seleccione la opcion deseada  ");
     NOR;
     cursor(25,10,5);
     switch (funcion)

      {
        case 'A':
           restore();
           opcion_A();
           strcpy(oper,path);
           strcat(oper,"cierre");
           restore();
           execl(oper,oper,path,0);
        case 'Z':
           clear();
           restore();
           gotoxy(1,1);
           exit(0);
     }
  }


opcion_A()
{
   procesos("Actualizacion de acumulados");
     strcpy(comando,path);
     strcat(comando,"GOR7002.exe ");
 sino_job_int();
}
```

```
main(argc,argv)
  int argc;
  char *argv[];
  {
   while (1)
   {
    strcpy(path,argv[1]);
    cierre();
   }
  }


/*menu0.c. Programa en C que muestra el menú princupal del sistema de Gastos de Origen*/
#include "menuc.c"
char fe1[7],fe2[7],oper[50];

menu001()
  {
    setraw();
    for (i=1;i<=2;i++)
    for (j=1;j<=4;j++)
        matriz[i][j] = 0;
    dimx = 2;
    dimy = 8;
    renglon = 1;
    columna = 1;
    funcion = 'A';
    matriz[1][1] = 'A';
    matriz[1][2] = 'B';
    matriz[1][3] = 'C';
    matriz[1][4] = 'D';
    matriz[2][1] = 'E';
    matriz[2][2] = 'F';
    matriz[2][4] = 'Z';
    strcpy(texto[1].c,"A. Carga de Informacion ");
    strcpy(texto[2].c,"B. Consulta de Inf. Acumulada ");
    strcpy(texto[3].c,"C. Reportes ");
    strcpy(texto[4].c,"D. Normatividad ");
    strcpy(texto[9].c,"E. Cierre Mensual");
    strcpy(texto[10].c,"F. Administracion");
    strcpy(texto[12].c,"Z. Salida del Sistema");
    titulos();
    descripcion("Sistema de Gastos de Origen");
    gotoxy(10,14);printf(texto[1].c);
    gotoxy(11,14);printf(texto[2].c);
    gotoxy(12,14);printf(texto[3].c);
    gotoxy(13,14);printf(texto[4].c);
    gotoxy(10,48);printf(texto[9].c);
    gotoxy(11,48);printf(texto[10].c);
    gotoxy(13,48);printf(texto[12].c);
    BRI;
    gotoxy(19,18);printf(" Seleccione una opcion ");
    NOR;
    cursor(14,48,10,8);
    switch (funcion)
```

```
      {
        case 'A':
           restore();
           opcion_A();
           restore();
           break;
        case 'B':
           restore();
           system("runform30  GOR5001 tj40010/adm_gor");
           restore();
           break;
        case 'C':
           restore();
           system("reportes");
           restore();
           break;
        case 'D':
           restore();
           system("normatividad");
           restore();
           break;
        case 'E':
           restore();
           opcion_E();
           restore();
           break;
        case 'F':
           restore();
           system("adminis");
           restore();
           break;
        case 'Z':
           clear();
           restore();
           gotoxy(1,1);
           exit(0);
    }
  }

opcion_A()
{
 resp = 'n';
 while ( resp != 's' )
  {
   procesos("Carga de sic al sistema");
   gotoxy(9,17); printf("Fecha inicial del mes  (DDMMAA)  ===>  DDMMAA");
   gotoxy(10,17); printf("Fecha final del mes   (DDMMAA)  ===>  DDMMAA");
   BRI;
   gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
   NOR;

     strcpy(comando,"GOR3001.exe ");

    valida_ddmmaa(fe1,9,56);
    strcat(comando,fe1); strcat(comando," ");
```

```
    valida_ddmmaa(fe2,10,56);
    strcat(comando,fe2); strcat(comando," ");

   sino_datos();
 }
  sino_job_int();
}

opcion_E()
{
   procesos("Actualizacion de acumulados");
     strcpy(comando,"GOR7002.exe ");
  sino_job_int();
}
/*****************************************************************/
/*                PROGRAMA PRINCIPAL                             */
/*****************************************************************/
main()
   {
    while (1)
     {
      menu001();
     }
   }

/* ================================================================*/
/*normatividad.c. Programa en C que muestra el menu de catálogos*/
/* ================================================================*/

#include "men.c"
char fe1[7],fe2[7],oper[50];

menu001()
  {
    setraw();
    i=1;
    for (j=1;j<=4;j++)
        matriz[i][j] = 0;
    dimx = 1;
    dimy = 6;
    region = 1;
    columna = 1;
    funcion = 'A';
    matriz[1][1] = 'A';
    matriz[1][3] = 'B';
    matriz[1][6] = 'Z';
    strcpy(texto[1].c,"A. Catalogo de Clas. y Subctas. por Depto.");
    strcpy(texto[3].c,"B. Catalogo de Centro y Deptos. Economicos");
    strcpy(texto[6].c,"Z. Regresar al Menu Principal");
    titulos();
    descripcion("Normatividad");
    gotoxy(10,18);printf(texto[1].c);
    gotoxy(12,18);printf(texto[3].c);
    gotoxy(15,18);printf(texto[6].c);
    BR!;
    gotoxy(19,18);printf(" Seleccione una opcion ");
```

```
    NOR;
    cursor(18,10,6);
    switch (funcion)
      {
        case 'A':
           restore();
           system("runform30  GOR1113 tj40010/adm_gor");
           restore();
           break;
        case 'B':
           restore();
           system("runform30  GOR1114 tj40010/adm_gor");
           restore();
           break;
        case 'Z':
            clear();
            restore();
            gotoxy(1,1);
            exit(0);
      }
  }

/*****************************************************************/
/*                PROGRAMA PRINCIPAL                             */
/*****************************************************************/
main()
  {
   while (1)
    {
     menu001();
    }
  }


/* ================================================================*/
/*reportes.c. Programa en C que muestra el menú de reportes */
/* ================================================================*/

#include "menuc.c"
char fe1[7],fe2[7],oper[50];

menu001()
  {
    setraw();
    for (i=1;i<=2;i++)
    for (j=1;j<=4;j++)
       matriz[i][j] = 0;
    dimx = 2;
    dimy = 8;
    renglon = 1;
    columna = 1;
    funcion = 'A';
    matriz[1][1] = 'A';
    matriz[1][2] = 'B';
    matriz[1][3] = 'C';
    matriz[1][4] = 'D';
```

```
matriz[2][1] = 'E';
matriz[2][2] = 'F';
matriz[2][4] = 'Z';
strcpy(texto[1].c,"A. Cuenta, Subcta. Concepto");
strcpy(texto[2].c,"B. Centro y Departamento ");
strcpy(texto[3].c,"C. Renglon del Gasto ");
strcpy(texto[4].c,"D. Cuenta y Concepto de Origen");
strcpy(texto[9].c,"E. Resumen por Conpto. de Origen");
strcpy(texto[10].c,"F. Resumen por Cuenta");
strcpy(texto[12].c,"Z. Regresar al Menu Principal");
titulos();
descripcion("Reportes");
gotoxy(10,9);printf(texto[1].c);
gotoxy(11,9);printf(texto[2].c);
gotoxy(12,9);printf(texto[3].c);
gotoxy(13,9);printf(texto[4].c);
gotoxy(10,45);printf(texto[9].c);
gotoxy(11,45);printf(texto[10].c);
gotoxy(13,45);printf(texto[12].c);
BRI;
gotoxy(19,18);printf(" Seleccione una opcion ");
NOR;
cursor(9,45,10,8);
switch (funcion)

 {
   case 'A':
      restore();
      opcion_A();
      restore();
      break;
   case 'B':
      restore();
      opcion_B();
      restore();
      break;
   case 'C':
      restore();
      opcion_C();
      restore();
      break;
   case 'D':
      restore();
      opcion_D();
      restore();
      break;
   case 'E':
      restore();
      opcion_E();
      restore();
      break;
   case 'F':
      restore();
      opcion_F();
      restore();
      break;
```

```
        case 'Z':
            clear();
            restore();
            gotoxy(1,1);
            exit(0);
      }
   }

opcion_A()
{
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Reporte por Cuenta y Concepto");
    gotoxy(9,17); printf("Fecha de cierre de mes (DDMMAA)  ===>  DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR8001.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

   sino_datos();
  }
   sino_job_int();
}
opcion_B()
{
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Reporte por Centro y Departamento");
    gotoxy(9,17); printf("Fecha de cierre de mes (DDMMAA)  ===>  DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR8002.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

   sino_datos();
  }
   sino_job_int();
}

opcion_C()
{
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Reporte por Renglon del Gasto");
    gotoxy(9,17); printf("Fecha de cierre de mes (DDMMAA)  ===>  DDMMAA");
```

```
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR8006.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

   sino_datos();
  }
   sino_job_int();
 }

 opcion_D()
 {
  resp = 'n';
  while ( resp != 's' )
   {
    procesos("Reporte por Cuenta y Concepto");
    gotoxy(9,17); printf("Fecha de cierre de mes (DDMMAA) ===> DDMMAA");
    BRI;
    gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
    NOR;

      strcpy(comando,"GOR8004.exe ");

     valida_ddmmaa(fe1,9,56);
     strcat(comando,fe1); strcat(comando," ");

   sino_datos();
  }
   sino_job_int();
}

opcion_E()
{
 resp = 'n';
 while ( resp != 's' )
  {
   procesos("Reporte del Resumen por Concepto");
   gotoxy(9,17); printf("Fecha de cierre de mes (DDMMAA) ===> DDMMAA");
   BRI;
   gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
   NOR;

     strcpy(comando,"GOR8000.exe ");

    valida_ddmmaa(fe1,9,56);
    strcat(comando,fe1); strcat(comando," ");

  sino_datos();
 }
  sino_job_int();
}
opcion_F()
```

```
{
 resp = 'n';
 while ( resp != 's' )
  {
   procesos("Reporte del Resumen por Cuenta");
   gotoxy(9,17); printf("Fecha de cierre de mes (DDMMAA)  ===>  DDMMAA");
   BRI;
   gotoxy(16,4);printf("FAVOR DE PROPORCIONAR LOS DATOS");
   NOR;

     strcpy(comando,"GOR8003.exe ");

    valida_ddmmaa(fe1,9,56);
    strcat(comando,fe1); strcat(comando," ");

  sino_datos();
 }
  sino_job_int();
}

/*****************************************************************/
/*                  PROGRAMA PRINCIPAL                           */
/*****************************************************************/
main()
  {
    while (1)
     {
      menu001();
     }
  }


/* ============================================================*/
/*rutinas.uti. Programa en C que contiene rutinas para los programas de menú*/
/* ============================================================*/

char mensaje[65];
char comando[300];
char resp,*c[2];
static int day_tab [2] [13] = {
      {0,31,28,31,30,31,30,31,31,30,31,30,31},
      {0,31,29,31,30,31,30,31,31,30,31,30,31}};

mensajes(mensaje)
char mensaje[65];
{
 gotoxy(23,10);
 BRI;
 printf("%65-s",mensaje);
 NOR;
}

limpia_mensaje()
{
 gotoxy(23,10);
```

```
 printf("                                                    ");
}

lectura(longitud,valor)
int longitud;
char *valor[];
{
int i;
char captura[80];
char c;
for(i=0;i<longitud;i++)
   captura[i] = ' ';
captura[0] = '\0';
i=0;
c='\0';
while( (c = getchar()) != '\n')
    {
    if(i<=longitud)
      {
      if(i == longitud)
        captura[longitud]='\0';
      else
        captura[i] = c;
      }
    i++;
    }
strncpy(valor,captura,longitud);
}


sino_job_int()
{
 gotoxy(23,42);
 printf(" ");
 BRI;
 gotoxy(23,10);
 printf("Desea Ejecutar el Proceso (s/n) ");
 NOR;
 resp=getchar(); gets(c);
 if(resp == 's'|| resp == 'S' )
   system(comando);
}

sino_datos()
{
 gotoxy(23,10);
 printf("Estan correctos los datos (s/n) ");
 resp=getchar(); gets(c);
 limpia_mensaje();
}


 numeros(longitud,valor,status)
int longitud;
char valor[];
int *status;
```

```
{
int i;
int j=0;
for(i=0;i<longitud;i++)
  {
    if( isdigit(valor[i]) )
       j++;
  }
if( j == longitud )
   *status=0;
else
   *status=1;
}

valida_ddmmaa(valor,x,y)
char valor[];
int x,y;
{
int dia,mes,ano,bis,status;
char diac[3],mesc[3],anoc[3];

 status = 1;
 while ( status == 1 )
 {
  gotoxy(x,y);
  lectura(6,valor);
  numeros(6,valor,&status);
 }
}


/* ==========================================================================*/
/*men.c*. Programa en C que contiene rutinas para los programas de menú /
/* ==========================================================================*/

#define INV printf("\033[7m")
#define NOR printf("\033[0m")
#define BLI printf("\033[5m")
#define BRI printf("\033[1m")
#include <termio.h>
#include <errno.h>
#include <stdio.h>
#include <ctype.h>
#include "rutinas.uti"

/* variables globales del sistema */

 int dimx;
 int dimy;
 int renglon;
 int columna;
 int t;
 int i;
 int j;
 int opcion;
 int matriz[5][15];
```

```
 int funcion;
 int x;
 int y;

 struct ren {
        char  c[75];
        };
 struct ren texto[30];


 clear()
    {
     printf("\033[2J");      /* limpia pantalla  */
    }


 /* == procedimiento para posicionarse en la pantalla ==
    == x - renglon de la pantalla                    ==
    == y - columna de la pantalla                    == */

 gotoxy(x,y)
   int x,y;
    {
      char xx[3],yy[3];
      sprintf(xx,"%d\0",x);
      sprintf(yy,"%d\0",y);
      printf("\033[%s;%sH",xx,yy);
    }

procesos(letrero)
char letrero[70];
 {
  int i,columna,longitud;
  clear();
  box(1,1,24,79);
  longitud=strlen(letrero);
  columna=(80 - longitud) / 2;
  gotoxy(2,columna);
  INV;
  printf(letrero);
  NOR;
  gotoxy(4,30);
  printf("P A R A M E T R O S");
  if(longitud < 19 )
   {
    longitud = 19;
    columna = 30;
   }
  gotoxy(5,columna);
  printf("\033(0");
  for(i=1;i<=longitud;i++)
     printf("q");
  printf("\033(B");
}

/*=================================================
```

```
==  Dibuja un rectangulo                                    ==
== x1 - renglon de la esquina superior                      ==
==      izquierda del rectangulo                            ==
== y1 - columna de la esquina superior                      ==
==      izquierda del rectangulo                            ==
== x2 - renglon de la esquina inferior                      ==
==      derecha del rectangulo                              ==
== y2 - columna de la esquina inferior                      ==
==      derecha del rectangulo                              ==
================================================*/

box(x1,y1,x2,y2)
 int x1,y1,x2,y2;
 {
   register int j;
   printf("\033(0");              /* inicia modo grafico      */
   gotoxy(x1,y1);                 /* punto de inicio  */
   printf("l\033[D\033[B");       /* esquina superior derecha  */
   for (j=1;j<x2-x1;j++)          /* linea derecha            */
       printf("x\033[B\033[D");
       printf("m");               /* esquina inferior derecha  */
   for (j=1;j<y2-y1;j++)          /* linea inferior           */
       printf("q");
       printf("j");               /* esquina inferior izquierda */
   for (j=1;j<x2-x1;j++)          /* linea izquierda          */
       printf("\033[A\033[Dx");
       printf("\033[A\033[Dk");   /* esquina superior izquierda */
   for (j=1;j<y2-y1;j++)          /* linea superior           */
       printf("\033[2Dq");
       printf("\033(B");          /* quita modo grafico       */
 }

/* Despliega titulos en Pantalla */

titulos()
 {
  int j;
  clear();
  box(1,1,24,79);
  gotoxy(2,30);printf("PETROLEOS MEXICANOS");
  gotoxy(3,27);printf("GAS Y PETROQUIMICA BASICA");
  gotoxy(4,32);printf("Venta de Carpio");
  printf("\033(0");

  /* Dibuja linea intermedia  */

  gotoxy(5,17);
  for (j=1;j<47;j++)
      printf("q");
      printf("\033(B");
}

/* Obtiene la longitud del letrero y lo centra */


descripcion(letrero)
```

```
char letrero[70];
{
 int columna,longitud;
 longitud=strlen(letrero);
 columna=(80 - longitud) / 2;
 gotoxy(6,columna);
 BRI;
 printf(letrero);
 NOR;
}

/*================================================================
== procedimiento para encontrar la posicion en pantalla (x,y)    ==
== en donde inicia la opcion deseada, ademas de calcular la      ==
== posicion del texto dentro de la matriz de textos de opciones  ==
== parametros:                                                   ==
==          pc1 - columna de la pantalla en donde                ==
==                se encuentra la opcion de la                   ==
==                primera columna de opciones                    ==
==          pr  - renglon de la pantalla en donde                ==
==                inician las opciones                           ==
================================================================*/

transforma(pc1,pr)
  int pc1,pr;
  {
    if (columna == 1)
       y = pc1;
    x = renglon + pr - 1;
    t = columna * dimy - dimy + renglon; /*Calcula numero de texto*/
  }


/*================================================================
== procedimiento para detectar el movimiento deseado, ya         ==
== sea por el movimiento de las teclas de 'flechas' o por        ==
== oprimir la tecla correspondiente a la opcion                  ==
== parametros:                                                   ==
==          poscol1 - columna de la pantalla en donde            ==
==                se encuentra la opcion de la                   ==
==                primera columna de opciones                    ==
==          posren  - renglon de la pantalla en donde            ==
==                inician las opciones                           ==
==          nopcion - numero de opciones en el menu              ==
================================================================*/

cursor(poscol1,posren,nopcion)
  int poscol1,posren,nopcion;
  {
    transforma(poscol1,posren);
    gotoxy(x,y);
    printf("\033[7m");
    printf(texto[t].c);
    printf("\033[0m");
    while((opcion = getchar())!=13)
      {
```

```
opcion = toupper(opcion);
if(opcion == 88)
  {
   funcion=opcion;
   break;
  }
if(opcion == 27)
  {
   opcion = getchar();
   if (opcion == 91)
     {
      transforma(poscol1,posren);
      gotoxy(x,y);
      printf(texto[t].c);
      opcion = getchar();
      switch (opcion)
           {
            case 65:
               renglon--;
               if (renglon <= 0)
                 renglon = dimy;
                 while (matriz[columna][renglon] == 0)
                      renglon--;
             break;

            case 66:
               renglon++;
               if (renglon > dimy)
                 renglon = 1;
                 while (matriz[columna][renglon] == 0)
                      {
                       if (renglon > dimy)
                          renglon = 1;
                        else
                          renglon++;
                       }
             break;

            case 68:
               columna--;
               if (columna <= 0)
                 columna = dimx;
                 while (matriz[columna][renglon] == 0)
                      columna--;
             break;

            case 67:
               columna++;
               if (columna > dimx)
                  columna = 1;
                  while (matriz[columna][renglon] == 0)
                      columna--;
             break;

           }
        funcion = matriz[columna][renglon];
```

```
            }
          }
        else
          {
          if (((opcion > 64) && (opcion < 65 + nopcion)) ||
             (opcion == 88) || (opcion == 90))
             {
              transforma(poscol1,posren);
              gotoxy(x,y);
              printf(texto[t].c);
              funcion = toupper(opcion);
              for (i=1;i<=dimx;i++)
                 for (j=1;j<=dimy;j++)
                    if (matriz[i][j] == funcion)
                       {
                       columna = i;
                       renglon = j;
                       goto found;
                       }
             }
          }
      found:
          transforma(poscol1,posren);
          gotoxy(x,y);
          printf("\033[7m");
          printf(texto[t].c);
          printf("\033[0m");
      }
  }

/* ========================================================================== */
/*      rutinas  setraw()  y  restore()                                       */
/* ========================================================================== */

/* Imprime mensaje de Error */

void syserr(msg)
char *msg;
 {
  extern int errno, sys_nerr;
  extern char *sys_errlist[];
  gotoxy(21,7);
  fprintf(stderr, "ERROR: %s (%d", msg, errno);
  if (errno > 0 && errno < sys_nerr)
    {
    gotoxy(21,20);
    fprintf(stderr,": %s)\n", sys_errlist[errno]);
    }
  else
    {
     gotoxy(21,20);
     fprintf(stderr,")\n");
    }
  exit(1);
}
```

```
static struct termio tbufsave;

void setraw()
   {
    struct termio tbuf;
    if (ioctl(1, TCGETA, &tbuf) == -1)  /* se guarda estado actal */
       syserr("ioctl ");
    if (ioctl(1, TCGETA, &tbufsave) == -1)  /* se guarda estado actal */
       syserr("ioctl ");

     /* se respalda estado actual y se cambia a modo raw */
     tbuf.c_lflag &= ~(INLCR | ICRNL);
     tbuf.c_lflag &= ~(ICANON | ECHO);
     tbuf.c_cc[4] = 1; /* MIN */
     tbuf.c_cc[5] = 0;  /* TIME */

     if (ioctl(0, TCSETAF, &tbuf) == -1) /* se establecen modos raw */
        syserr("ioctl2 ");
  }

/* Restaura banderas*/

void restore()
 {
   if (ioctl(0, TCSETAF, &tbufsave) == -1)
      syserr("ioctl3 ");
 }


/* ================================================================*/
/*menuc.c. Programa en C con rutinas para programas de menú */
/* ================================================================*/

#define INV printf("\033[7m")
#define NOR printf("\033[0m")
#define BLI printf("\033[5m")
#define BRI printf("\033[1m")
#include <termio.h>
#include <errno.h>
#include <stdio.h>
#include <ctype.h>
#include "rutinas.util"

/* Declaracion de Variables */

 int dimx;
 int dimy;
 int renglon;
 int columna;
 int t;
 int i;
 int j;
 int opcion;
 int matriz[5][15];
 int funcion;
 int x;
```

```
int y;

struct ren {
        char  c[75];
         };
struct ren texto[30];


/* procedimiento para limpiar la pantalla */
clear()
  {
     printf("\033[2J");
  }

/* ================================================
== procedimiento para posicionarse en la pantalla      ==
== x - renglon de la pantalla                          ==
== y - columna de la pantalla                          ==
================================================*/

gotoxy(x,y)
  int x,y;
   {
     char xx[3],yy[3];
     sprintf(xx,"%d\0",x);
     sprintf(yy,"%d\0",y);
     printf("\033[%s;%sH",xx,yy);
   }


/* ============================================================
== procedimiento para dibujar un rectangulo en pantalla        ==
== x1 - renglon de la esquina superior                         ==
==      izquierda del rectangulo                               ==
== y1 - columna de la esquina superior                         ==
==      izquierda del rectangulo                               ==
== x2 - renglon de la esquina inferior                         ==
==      derecha del rectangulo                                 ==
== y2 - columna de la esquina inferior                         ==
==      derecha del rectangulo                                 ==
============================================================*/

box(x1,y1,x2,y2)
 int x1,y1,x2,y2;
 {
     register int j;
     printf("\033(0");              /* inicia modo grafico      */
     gotoxy(x1,y1);                 /* punto de inicio  */
     printf("l\033[D\033[B");       /* esquina superior derecha  */
     for (j=1;j<x2-x1;j++)          /* linea derecha            */
         printf("x\033[B\033[D");
         printf("m");               /* esquina inferior derecha  */
     for (j=1;j<y2-y1;j++)          /* linea inferior           */
         printf("q");
         printf("j");               /* esquina inferior izquierda */
     for (j=1;j<x2-x1;j++)          /* linea izquierda          */
```

```
        printf("\033[A\033[Dx");
        printf("\033[A\033[Dk");     /* esquina superior izquierda */
      for (j=1;j<y2-y1;j++)          /* linea superior             */
        printf("\033[2Dq");
        printf("\033(B");           /* quita modo grafico         */
}


/* Despliega titulos en Pantalla */

titulos()
 {
  int j;
  clear();
  box(1,1,24,79);
  gotoxy(2,30);printf("PETROLEOS MEXICANOS");
  gotoxy(3,27);printf("GAS Y PETROQUIMICA BASICA");
  gotoxy(4,32);printf("Venta de Carpio");
  printf("\033(0");

  /* Dibuja linea intermedia  */

  gotoxy(5,17);
  for (j=1;j<47;j++)
      printf("q");
      printf("\033(B");
}

/* Obtiene la longitud del letrero y lo centra */

descripcion(letrero)
char letrero[70];
{
 int columna,longitud;
 longitud=strlen(letrero);
 columna=(80 - longitud) / 2;
 gotoxy(6,columna);
 BRI;
 printf(letrero);
 NOR;
}

procesos(letrero)
char letrero[70];
 {
  int i,columna,longitud;
  clear();
  box(1,1,24,79);
  longitud=strlen(letrero);
  columna=(80 - longitud) / 2;
  gotoxy(2,columna);
  INV;
  printf(letrero);
  NOR;
  gotoxy(4,30);
  printf("P A R A M E T R O S");
```

```
 if(longitud < 19 )
   {
    longitud = 19;
    columna = 30;
   }
 gotoxy(5,columna);
 printf("\033(0");
 for(i=1;i<=longitud;i++)
    printf("q");
 printf("\033(B");
 }
/* ================================================================
== procedimiento para encontrar la posicion en pantalla (x,y)     ==
== en donde inicia la opcion deseada, ademas de calcular la       ==
== posicion del texto dentro de la matriz de textos de opciones   ==
== pc1 - columna de la pantalla en donde                          ==
==       se encuentra la opcion de la                             ==
==       primera columna                                          ==
== pc2 - columna de la pantalla en donde                          ==
==       se encuentra la opcion de la                             ==
==       segunda columna                                          ==
== pr  - renglon de la pantalla en donde                          ==
==       inician las opciones                                     ==
================================================================*/

transforma(pc1,pc2,pr)
     int pc1,pc2,pr;
  {
     if (columna == 1)
         y = pc1;
     if (columna == 2)
         y = pc2;
     x = renglon + pr - 1;
     t = columna * dimy - dimy + renglon;
  }

/* ================================================================
== procedimiento para detectar el movimiento deseado, ya          ==
== sea por el movimiento de las teclas de 'flechas' o por         ==
== oprimir la tecla correspondiente a la opcion                   ==
==          poscol1 - columna de la pantalla en donde             ==
==                    se encuentra la opcion de la                ==
==                    primera columna de opciones                 ==
==          poscol2 - columna de la pantalla en donde             ==
==                    se encuentra la opcion de la                ==
==                    segunda columna de opciones                 ==
==          posren  - renglon de la pantalla en donde             ==
==                    inician las opciones                        ==
==          nopcion - numero de opciones en el menu               ==
================================================================*/

cursor(poscol1,poscol2,posren,nopcion)
  int poscol1,poscol2,posren,nopcion,
  {
     transforma(poscol1,poscol2,posren);
     gotoxy(x,y);
```

```
printf("\033[7m");
printf(texto[t].c);
printf("\033[0m");
while((opcion = getchar())!=13)
  {
   opcion = toupper(opcion);
   if(opcion == 88)
    {
     funcion=opcion;
     break;
    }
   if (opcion == 27)
     {
      opcion = getchar();
      if (opcion == 91)
        {
         transforma(poscol1,poscol2,posren);
         gotoxy(x,y);
         printf(texto[t].c);
         opcion = getchar();
         switch (opcion)
              {
           case 65:
                renglon--;
                if (renglon <= 0)
                   renglon = dimy;
                 while (matriz[columna][renglon] == 0)
                       renglon--;
           break;

           case 66:
                renglon++;
                if (renglon > dimy)
                   renglon = 1;
                 while (matriz[columna][renglon] == 0)
                      {
                       if (renglon > dimy)
                          renglon = 1;
                        else
                          renglon++;
                      }
           break;

           case 68:
                columna--;
                if (columna <= 0)
                   columna = dimx;
                while (matriz[columna][renglon] == 0)
                      columna--;
           break;

           case 67:
                columna++;
                if (columna > dimx)
                   columna = 1;
                while (matriz[columna][renglon] == 0)
```

```
                    columna--;
                break;

                }
              funcion = matriz[columna][renglon];
            }
          }
        else
          {
           if (((opcion > 64) && (opcion < 65 + nopcion)) ||
              (opcion == 88) || (opcion == 90))
             {
               transforma(poscol1,poscol2,posren);
               gotoxy(x,y);
               printf(texto[t].c);
               funcion = toupper(opcion);
               for (i=1;i<=dimx;i++)
                 for (j=1;j<=dimy;j++)
                   if (matriz[i][j] == funcion)
                     {
                      columna = i;
                      renglon = j;
                      goto found;
                      }
             }
          }
       found:
          transforma(poscol1,poscol2,posren);
          gotoxy(x,y);
          printf("\033[7m");
          printf(texto[t].c);
          printf("\033[0m");
       }
  }

/* ========================================================================== */
/*      rutinas  setraw() y  restore()                                        */
/* ========================================================================== */

/* Imprime mensajes de error */
void syserr(msg)
char *msg;
  {
   extern int errno, sys_nerr;
   extern char *sys_errlist[];
   gotoxy(21,7);
   fprintf(stderr, "ERROR: %s (%d", msg, errno);
   if(errno > 0 && errno < sys_nerr)
    {
     gotoxy(21,20);
     fprintf(stderr,"; %s)\n", sys_errlist[errno]);
    }
   else
    {
     gotoxy(21,20);
     fprintf(stderr,")\n");
```

```
    }
    exit(1);
  }

static struct termio tbufsave;

void setraw()
{
      struct termio tbuf;
      if (ioctl(1, TCGETA, &tbuf) == -1)      /* se guarda estado actal */
        syserr("ioctl ");
      if (ioctl(1, TCGETA, &tbufsave) == -1)  /* se guarda estado actal */
        syserr("ioctl ");

       /* se respalda estado actual y se cambia a modo raw */

      tbuf.c_iflag &= ~(INLCR | ICRNL);
      tbuf.c_lflag &= ~(ICANON | ECHO);
      tbuf.c_cc[4] = 1;                  /* MIN */
      tbuf.c_cc[5] = 0;                  /* TIME */

      if (ioctl(0, TCSETAF, &tbuf) == -1)      /* se establecen modos raw */
        syserr("ioctl2 ");
}

/* Restablece banderas */
void restore()
   {
    if (ioctl(0, TCSETAF, &tbufsave) == -1)
       syserr("ioctl3 ");
   }
```

APENDICE B

T_GOA		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_CTA	Numérico (4) no nulo	Número de cuenta de operación
N_SCTA	Char (9) no nulo	Número de subcuenta
N_CENAFO	Numérico (3) no nulo	Número de centro afectado
N_DEP	Numérico (5) no nulo	Número de departamento
N_CONORI	Char (6) no nulo	Número de concepto de origen
REN_GAS	Numérico (3)	Número de renglon del gasto
I_OP_ACU	Numérico (16,2) no nulo	Importe de operación acumulado
I_OP_MES	Numérico (16,2) no nulo	Importe de operación mensual
F_MOV	Fecha no nulo	Fecha de movimiento

T_GOR		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_CTA	Numérico (4) no nulo	Número de cuenta de operación
N_SCTA	Char (9) no nulo	Número de subcuenta
N_CENAFO	Numérico (3) no nulo	Número de centro afectado
N_DEP	Numérico (5) no nulo	Número de departamento
N_CONORI	Char (6) no nulo	Número de concepto de origen
REN_GAS	Numérico (3)	Número de renglon del gasto
I_OP	Numérico (16,2) no nulo	Importe de operación
F_MOV	Fecha no nulo	Fecha de movimiento

CETRO		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
CVE_CTRO	Numérico (3) no nulo	Clave de centro de trabajo
CVE_ZONA	Numérico (2) no nulo	Clave de zona
DES_CTRO	Char (50) no nulo	Descripción del centro de trabajo
TIP_REGI	Char (1) no nulo	Tipo de región

CONOR		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
CVE_CPTO	Char (6) no nulo	Clave del concepto de origen
DES_CPTO	Char (55) no nulo	Descripcion del concepto de origen
TIP_REGI	Char (1) no nulo	Tipo de región

CORGT		
NOMBRE DEL CAMPO	TIPO DE CAMPO	DESCRIPCION
CVE_RENG	Numérico (3) no nulo	Clave del renglón del gasto
CVE_CPTO	Char (6) no nulo	Clave del concepto de origen
TIP_REGI	Char (1) no nulo	Tipo de región

DEPTO		
NOMBRE DEL CAMPO	TIPO DE CAMPO	DESCRIPCION
CVE_CTRO	Numérico (3) no nulo	Clave del centro de trabajo
CVE_DPTO	Numérco (5) no nulo	Clave de departamento
CVE_CTRG	Numérco (3) no nulo	Clave de centro gerencia
CVE_GCIA	Numérico (5) no nulo	Clave de la gerencia
CVE_UDCO	Numérco (3) no nulo	Clave de unidad de control
NUM_CTAM	Numérico (4) no nulo	Número de cuenta de mayor
NUM_CTAO	Numérco (6) no nulo	Numéro de cuenta de operación
DES_DPTO	Char (50) no nulo	Descripción del departamento
CVE_NIVL	Numérico (2) no nulo	Clave de nivel
TIP_REGI	Char (1) no nulo	Tipo de región
CVE_OFIP	Numérico (3) no nulo	Clave de oficina pagadora

T_003		
NOMBRE DEL CAMPO	TIPO DE CAMPO	DESCRIPCION
N_DOC	Char (8) no nulo	Número de documento contabilizador
T_DOC	Numérico (1) no nulo	Tipo de documento
R_SEC	Numérco (5) no nulo	Número consecutivo
N_DOCORI	Char (8) no nulo	Número de documento contabilizador original
T_DOCORI	Numérico (1) no nulo	Tipo de documento original
R_SECORI	Numérco (5) no nulo	Número consecutivo original

T_004		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_DOC	Char (8) no nulo	Número de documento contabilizador
T_DOC	Numérico (1) no nulo	Tipo de documento
F_ELAB	Date no nulo	Fecha de elaboración
N_CENAFR	Numérico (3) no nulo	Número de centro afectador
AR_FUN	Numérico (3) no nulo	Area funcional
T_MON	Char (1) no nulo	Tipo de moneda
T_CAMOP	Numérco (11,6)	Tipo de cambio de operación
I_TTDOC	Numércio (18,2)	Importe total del número de documento
I_TTME	Numérico (16,2)	Importe total moneda extranjera
DES_CON	Char (100) no nulo	Descripción del concepto de origen
N_ERROR	Numérico (4)	Número de error
CV_ACT	Char (1)	Clave de actualización
F_MOV	Date no nulo	Fecha de movimiento
NOM_BEN	Char (30)	Nombre del beneficiario

T_008		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_DOC	Char (8) no nulo	Número de documento contabilizador
T_DOC	Numérico (1) no nulo	Tipo de documento
R_SEC	Numérco (5) no nulo	Número consecutivo
I_ME	Numérico (16,2)	Importe en moneda extranjera
I_OP	Numérico (18,2) no nulo	Importe de operación
CV_OP	Char (5)	Clave de operación
CTA_DBB	Numércio (4) no nulo	Cuenta de cargo
SCTA_DB	Char (9) no nulo	Subcuenta de cargo
CTA_DBO	Numérico (4) no nulo	Cuenta de cargo de operación
CTA_CRB	Numénco (4) no nulo	Cuenta de crédito
SCTA_CR	Char (9) no nulo	Subcuenta de crédito
CTA_CRO	Numénco (4) no nulo	Cuenta de crédito de operación
CV_DH	Char (1) no nulo	Clave de cargo/crédito
DIG_ESP	Char (1)	Digito especial
N_AVI	Char (15)	Número de aviso
CV_VAL	Char (1)	Clave de validación
DIG_AVI	Char (1)	Digito de aviso
CV_AVITRA	Char (1)	Clave de aviso transmitido
DIG_AJ	Char (1)	Digito de ajuste
CV_ACTAVI	Char (1)	Clave de actualización de aviso

T_007		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_DOC	Char (8) no nulo	Número de documento contabilizador
T_DOC	Numérico (1) no nulo	Tipo de documento
R_SEC	Numérico (5) no nulo	Número consecutivo
DOC_FTE	Char (15) no nulo	Documento fuente
FOL_ADE	Char (8)	Folio de adefas
CV_PROY	Char (8)	Clave de proyecto
CV_AUT	Char (11) no nulo	Clave de autorización
F_VENC	Date no nulo	Fecha de vencimiento
N_CENAFO	Numérico (3) no nulo	Número de centro afectado
N_DEP	Numérico (5) no nulo	Número de departamento
N_CONORI	Numérico (6) no nulo	Número de concepto de origen
DIG_ADE	Char (1)	Dígito de adefas

T_041		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_CONT	Numérico (4) no nulo	Número de la contaduría
N_CENAFR	Numérico (3) no nulo	Número de centro afectador

T_042		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_CENAFR	Numérico (3) no nulo	Número de centro afectador
N_DEP	Numérico (5) no nulo	Número de departamento

T_047		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_CTA	Numérico (4) no nulo	Número de cuenta
NOM_CTA	Char (50) no nulo	Descripción de la cuenta
NOM_COR	Char (20) no nulo	Descripción del concepto de origen

T_052		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
N_CTA	Numérico (4) no nulo	Número de cuenta
CV_SCTA	Char (9) no nulo	Clave de subcuenta
NOM_SCTA	Char (27) no nulo	Descripción de la subcuenta
N_SCTACT	Numérico (3) no nulo	Número de subcuenta actual

N_CENPAGR	Numérico (3) no nulo	Número de centro pagador

T_129		
NOMBRE DEL CAMPO	**TIPO DE CAMPO**	**DESCRIPCION**
MMYY_OP	Numérico (4) no nulo	Mes y año de operación
N_CTA	Numenco (4) no nulo	Número de cuenta
CV_SCTA	Char (9) no nulo	Clave de subcuenta
N_DOC	Char (8) no nulo	Número de documento contabilizador
T_DOC	Numérico (1) no nulo	Tipo de documento
R_SEC	Numérco (5) no nulo	Número consecutivo
N_CENAFO	Numérico (3) no nulo	Numero de centro afectado
N_DEP	Numénco (5) no nulo	Número de departamento
N_CONORI	Numérico (5) no nulo	Número de concepto de origen
CV_PROY	Char (8)	Clave de proyecto
CV_AUT	Char (11) no nulo	Clave de autonzación
I_OP	Numérico (18.2) no nulo	Importe de operación
ID_MOVDH	Char (1) no nulo	Identificador de movimiento cargo/crédito
F_VENC	Date no nulo	Fecha de vencimiento

BIBLIOGRAFÍA

- Introducción a ORACLE para Desarrolladores
  ORACLE
  Edición 1991

- ORACLE Precompilers
  Programer's Guide
  ORACLE
  Versión 1.3
  Edición 1990

- ORACLE RDBMS
  Database Administrator's Guide
  ORACLE
  Versión 6.0
  Edición 1990

- ORACLE RDBMS
  Performance Tuning Guide
  ORACLE
  Versión 6.0
  Edición 1990

- ORACLE RDBMS
  Utilities User's Guide
  ORACLE
  Versión 6.0
  Edición 1994

- Pro*C Supplement to the ORACLE
  Precompiler's Guide
  ORACLE
  Versión 1.6
  Edición 1994

- Pro*C User's Guide
  ORACLE
  Versión 1.1
  Edición 1990