IDENTIFICACIÓN DE LA OPORTUNIDAD O PROBLEMA

DEFINICIÓN DE LA OPORTUNIDAD O PROBLEMA

"Actualmente no se cuenta con una metodología adecuada que muestre en forma clara la utilización e implementación de manejadores ODBC para bases de datos relacionales, mediante el uso de: el administrador de ODBC, herramientas incluidas en los DBMS y lenguajes de programación."

JUSTIFICACIÓN

Los distintos fabricantes de software cada día incluyen mayores ventajas y opciones en sus productos, para de esta forma satisfacer necesidades concretas de los usuarios. Por tal razón, han desarrollado productos que se especializan en la gestión de cada una de las áreas, tales como: finanzas, contabilidad, ventas, nómina, presupuestos, marketing, manejo de proyectos, entre otros. Esto ha permitido que una empresa pueda automatizar sus procesos utilizando diferentes herramientas de acuerdo a los requerimientos y presupuestos de sus áreas.

El uso de software de las distintas casas comerciales ha generado la necesidad de crear mecanismos que permitan la comunicación de datos entre estos productos, que en la mayoría de casos son bases de datos diferentes, tales como: ORACLE, VisualFox, Informix, Sybase, DB2, SQLServer entre otras.

En nuestro medio el desconocimiento del manejo de los mecanismos de intercambio de datos entre diferentes plataformas de bases de datos ha originado una serie de dificultades tales como:

- Automatizar todos los procesos de la empresa con un solo producto, exponiéndose así a los altos costos que se incurren en las actualizaciones, soporte técnico y mantenimiento.

- Limitar la posibilidad de aprovechamiento de herramientas especializadas en el manejo de las operaciones específicas de cada área de la empresa, por temor a que no se integren de forma óptima con los procesos existentes.

- Subutilizar o no utilizar los medios que se incluyen en cada herramienta de manejo de base de datos para el intercambio de información.

- Hacer migraciones innecesarias que toman mucho tiempo, riesgo y demandan costos elevados.

- Tener módulos informáticos aislados que dificultan a la empresa las intenciones de integración y obtención de resultados generales.

Como respuesta a lo anterior se ha propuesto el presente tema de tesis, en el que se desarrolla la metodología para el manejo del estándar de software ODBC (OPEN DATABASE CONECTIVITY) de Microsoft® que proporciona un camino para que los programadores puedan crear aplicaciones que permitan compartir datos entre una base de datos y otra.

Entre otras cosas este trabajo da a conocer

- La arquitectura del ODBC, en la que se muestra sus componentes y como interactúan unos con otros.
- La referencia a las funciones API ODBC , sus parámetros y usos.
- El manejo de los drivers ODBC incluidos en los DBMS, indicando la forma de configuración y conexión con las bases de datos.
- La metodología para crear un driver ODBC y su manejo mediante lenguaje Visual C++ (SDK).
- El proceso a seguir para la creación de un prototipo de prueba aplicado a Nómina de Personal mediante el uso de Visual Basic 6.0, Objetos y controles ADO, ODBC; y ASE Sybase 11.5 como DBMS.
- El proceso a seguir para la creación de un prototipo de prueba aplicado a Tesorería mediante el uso de Visual FoxPro, ODBC y ORACLE.

OBJETIVOS

Generales

- Analizar la comunicación entre Bases de Datos mediante el estándar ODBC de Microsoft® para crear una metodología que permita utilizarlas e implementarlas a través de manejadores normalizados y lenguajes de programación.

Específicos

- Elaborar un documento que sirva como instructivo para la administración de los manejadores ODBC mediante el estudio de los mismos incluidos en los Sistemas Administradores de Base de Datos.

- Plantear una metodología para implementar librerías dinámicas ODBC de acceso a datos a través de programación en Lenguajes visuales

- Construir librerías de enlace dinámico para acceso a datos a través de motores DBMS sobre el Sistema Operativo Microsoft Windows utilizando funciones API ODBC correspondientes.

- Construir un prototipo de Administración de Nómina de Personal para probar la metodología basada en los manejadores ODBC de Sybase y Visual Basic.

- Crear un prototipo para Gestión de Tesorería con la finalidad de probar la metodología basadas en los manejadores ODBC de ORACLE con Visual FoxPro.

CAPÍTULO I

INTRODUCCIÓN A LOS AMBIENTES DE DATOS COMPARTIDOS

Si se examina la estructura actual de cualquier organización y la forma en que las personas llevan a cabo sus trabajos, descubrirá que, en muchos casos, tanto la estructura como los procesos se diseñaron basándose en los recursos tecnológicos que disponen. Como ambiente compartido de datos se puede mencionar a la tecnología Cliente/Servidor, la cual trae consigo implicaciones muy importantes, al parecer este es el momento en que las organizaciones vuelvan a replantear la manera de como resuelven sus problemas.

1.1. Panorama Actual

Actualmente el mundo informático esta viviendo la globalización de la información, se desea compartir información entre todos los procesos internos de la empresa así como los procesos externos a la misma.

Brindar la posibilidad que el usuario pueda realizar las distintas operaciones con la empresa desde diferentes sitios remotos, implica el manejo de las últimas tecnologías que se presentan en el software moderno de administración de bases de datos. Desde este punto de vista, los departamentos de desarrollo de sistemas de las compañías están abandonando con rapidez las caras aplicaciones basadas en host y están desarrollando programas más flexibles que pueden ser creados, prototipados e implementados en plataformas de red LAN (LOCAL AREA NETWORK) de bajo precio y amplia aceptación.

El objetivo final de estos nuevos sistemas es simplificar y acelerar el proceso de los usuarios finales a la información almacenada en diversas plataformas dentro de la organización. En la actualidad las grandes corporaciones están utilizando tecnología Cliente/Servidor, en la cual una base de datos almacena gran cantidad de información que puede ser localizada por usuarios ejecutando aplicaciones front-end en modo gráfico. Estas aplicaciones preguntan o consultan a las bases de datos, que actúan

como servidor y extraen información que se puede insertar dentro de la aplicación del usuario, que actúa como cliente. Bill Gates explica así el concepto Cliente/Servidor: "Tú tienes un cliente que ejecuta la mayor parte del proceso, y un servidor, que corre una parte del código. Típicamente puedes tener una aplicación corriendo en un cliente, contra un sistema de Gestión de Base de Datos Relacional en el servidor, o un correo electrónico o cualquier otro sistema orientado al trabajo de grupo". De acuerdo con esta definición, una compañía orientada al mercado mainframe pensaría en Cliente/Servidor como una parte del código (la interfaz gráfica de usuario, por ejemplo) en el PC, y una gran cantidad de códigos en una computadora mainframe. Por otra parte, una empresa en el mercado microinformático defendería que Cliente/Servidor significa un poco de código en el servidor y grandes cantidades en la PC.

Para nuestros propósitos definiremos la informática Cliente/Servidor de la siguiente forma: sistemas informáticos en red, integrados, que utilizan aplicaciones que cooperan a lo largo de la red compartiendo sus recursos. El énfasis está en la palabra aplicación, porque el punto clave de adquirir y mantener un sistema informático es que cumpla un propósito. Definitivamente, lo primordial consiste en la manera de como se utilizan estos sistemas y como compartir los recursos computacionales.

Bien, luego de introducir el concepto de la tecnología Cliente/Servidor, se expone cómo y con qué se implementa prácticamente la misma. En un ambiente competitivo los distintos fabricantes de software cada día incluyen mayores ventajas y opciones en sus productos, para de esta forma satisfacer las necesidades de los usuarios. Por tal razón, han desarrollado productos que se especializan en la gestión de cada una de las áreas, tales como: finanzas, ventas, nómina, presupuestos, marketing, manejo de proyectos, entre otros. Esto ha permitido que una empresa pueda automatizar sus procesos utilizando diferentes herramientas de acuerdo a los requerimientos y presupuestos de sus áreas. El uso de software de las distintas casas comerciales ha generado la necesidad de crear mecanismos que permitan la comunicación de datos entre estos productos, que en la mayoría de casos son bases de datos diferentes, tales como: ORACLE, Fox, Informix, Sybase, DB2, SQLServer entre otras.

Muchas aplicaciones comerciales requieren de un acceso rápido a los registros individuales de una Base de Datos grande. Hasta ahora, la mayor parte de la programación de Sistemas de Base de Datos para microcomputadores se ha centrado en sistemas dBase/Xbase.

El estándar de software ODBC (OPEN DATABASE CONECTIVITY) de Microsoft® proporciona un camino para que los programadores puedan hoy crear aplicaciones de Base de Datos. Los programadores tendrán la ventaja tanto de la programación Orientada a Objetos como del lenguaje SQL (STRUCTURED QUERY LANGUAGE) de acceso a Bases de Datos. Incluso podrán acceder a archivos dBase.

Como ya se mencionó, en la actualidad es cada vez más necesario compartir recursos informáticos como Bases de Datos, Aplicaciones, etc.; razón por lo cual han proliferado en nuestro medio la instalación y uso de redes de área local LAN como también las conexiones a la red INTERNET tomando de esta manera mayor relevancia el uso compartido de grandes bancos de datos. De igual manera la programación de aplicaciones en ambiente gráfico se ha colocado a la vanguardia, de tal forma que los creadores de herramientas de programación visual no han escatimado esfuerzos para desarrollar controles como rutinas de conexión a bases de datos, combinando así la versatilidad de la interfaz gráfica con la seguridad de los DBMS, dentro de este estudio se va a analizar la manera de optimizar el uso de los controladores ODBC y el procedimiento a seguir para construir uno basado en funciones preestablecidas.

1.2. La Comunicación de datos en ambientes compartidos con arquitectura Cliente/Servidor

Muchas situaciones han llevado al lanzamiento de la arquitectura Cliente/Servidor. Algunas de las razones que tienen los usuarios para exigir las soluciones proporcionadas por esta tendencia son:

- Pueden usarse interfaces gráficas para presentar información gráfica.
- Comunicarse entre diferentes aplicaciones vía el portapapeles (Ej.: el texto procesador, hoja de cálculo).
- Usar PC's para transmitir todas las solicitudes al procesador central.

- Acceder simultáneamente a los servidores de bases de datos múltiples.
- Oportunidad de migrar a otras plataformas a costos de conversión bajos.

Además, es obvio que las aplicaciones actuales deben cubrir requisitos diferentes comparados a aquellos requeridos previamente, tales como los que se mencionarán a continuación.

1.2.1. Interfaces Gráficas

El uso de interfaces gráficas y la comunicación entre las aplicaciones es considerado muy necesario por muchos usuarios.

Hoy los usuarios están logrando estas metas al trabajar con procesadores de texto o hojas de cálculo electrónicas. No es fácil de explicarles que cuando invocan a su sistema no puede beneficiarse de las ventajas arriba expresadas y que ellos no pueden ver las respuestas a sus preguntas en formato gráfico cuando ellos estaban trabajando con hojas de cálculo electrónicas.

Las ventajas comparativas son:

- Las interfaces Gráficas dan lugar a diálogos que son mucho más flexibles amistosos
- La calidad de La presentación es superior.

Las aplicaciones a gran escala no pueden ofrecer interfaces gráficas en arquitecturas centralizadas. Las molestias principales son:

- Cada usuario necesita gran potencial de procesamiento.
- Un gran volumen de transmisión de datos se requerirá: aunque en texto el documento normalmente queda alrededor del tamaño de 2KB, es bastante común que tenga gráficos que exceden 350 KB.

1.2.2. Down-sizing o Right-sizing

Se refiere al proceso de migrar sistemas de mainframes grandes a los sistemas de tamaño medio (minis) y de los sistemas medianos a las redes de la computadora. Estos cambios estructurales han sido la fuerza cuya tendencia ha llevado al desarrollo de aplicaciones de Cliente/Servidor.

Es muy difícil para una empresa migrar todos sus sistemas de información de uno día al otro. El desarrollo de aplicaciones de Cliente/Servidor le permite que acceda a diferentes bases de datos ejecutando simplemente una consulta. Esto posibilita migrar parte del sistema y continuar con el desarrollo del resto.

1.2.3. Portabilidad – Adaptabilidad

Las empresas experimentan incrementos grandes en el volumen de información que ellas manejan y la calidad de esta información parece tener una importancia creciente.

No es posible pensar que el hardware que se compra cubrirá las necesidades futuras y esto incluso es más al referirse a la compra de software. Los requisitos de la empresa cambian de forma constante y rápida, además, bastante a menudo antes que el sistema realmente sea puesto en producción, los objetivos requeridos por el sistema cambiarán con respecto a lo que se descubrió previamente a su desarrollo.

Uno de los objetivos principales de Cliente/Servidor es ofrecer un buen nivel de aplicaciones portables, asegurando el mismo nivel de crecimiento indicado por la tecnología de hoy.

Sin embargo, es necesario cambiar con respecto a los cambios hechos a las plataformas de hardware, así como los sistemas deben adaptar los nuevos requisitos del usuario, hay que asegurar la portabilidad completa entre las plataformas apoyadas.

Actualmente las aplicaciones deben manejar de manera sencilla, eficiente y transparente los asuntos complejos, como niveles diferentes de protocolos de comunicación, SQL, restricciones, seguridades y otros rasgos técnicos.

Los usuarios están tendiendo a exigir sistemas más específicos y el suministro del mercado parece cubrir sus necesidades. Es por esta misma razón que los usuarios pueden escoger (para sus computadoras personales) de una variedad de sistemas operativos: DOS, Windows 3.1/95/98/2000/Me, OS/2, Windows NT, Apple Macintosh, etc.. Lo mismo aplica a la selección de servidores de bases de datos. Hay un gran número de servidores de bases de datos hoy en día, muchos de estos puede ser usados a través de varias plataformas: DB2, ORACLE, el Servidor de SQL, Sybase, el etc..

ODBC se ha definido como la interfase normalizada entre los clientes y servidores. Esto, por consiguiente, llevó a las herramientas de desarrollo de software a desarrollar rápidamente manejadores para ser utilizados por la mayoría de los servidores.

Desde este momento, las necesidades y objetivos están claros y las tecnologías tienden a reflejar su experiencia y crecimiento en estas materias conforme el tiempo pasa. Los sistemas de Cliente/Servidor no son, en la actualidad, la solución a todos estos problemas, pero ellos sin duda son una parte importante del mundo de la informática en el futuro.

1.3. La concepción Cliente/Servidor

Se oye a menudo sobre la facilidad para distribuir procesos y datos para racionalizar el uso de los recursos del sistema limitados utilizando Cliente/Servidor. Algunos de los ejemplos son:

- Client / File Server.
- Client / Print Server.
- Client / Mail Server.
- Client / Database Server.

Primero se revisará las características de una arquitectura centralizada para compararla con soluciones de Cliente/Servidor.

1.3.1. Arquitectura Centralizada

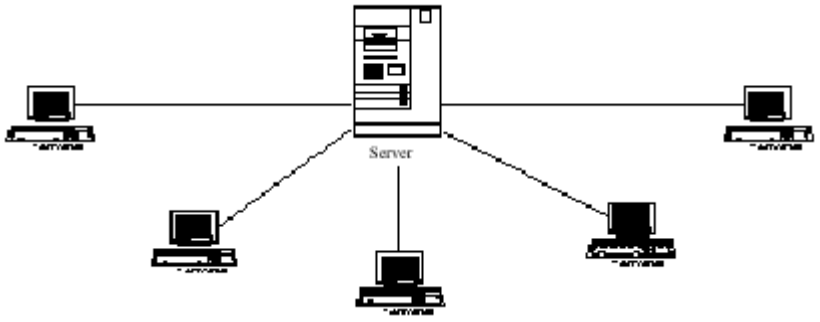


Figura 1.1

Los recursos en una arquitectura centralizada (Figura 1.1), son manejados por una computadora central. El CPU debe asignar recursos del sistema siempre que los usuarios soliciten ciertas operaciones. Para asistir a estos procesos el CPU debe realizar interrupciones, causando, un mal uso del recurso.

La transmisión de datos también es un factor crítico en este tipo de arquitectura, porque el sistema es diseñado para transmitir volúmenes de datos grandes por transacción, significando esto, que los sistemas llenarán despliegues de la pantalla con muchos datos difíciles de entender y con muy pocas oportunidades de lograr una interacción razonable con el usuario.

Por otro lado, la interfaz del usuario normalmente se restringe a las pantallas de texto.

Los puntos más favorables de los sistemas centralizados son: ofrecer integridad de la base de datos y mostrar la información en línea.

1.3.2. Client/File Server

Esto se refiere al caso cuando todo los datos se centralizan en el servidor de archivos y los datos a procesar son compartidos entre el Cliente y el Servidor (Figura 1.2).

La relación operativa es: el cliente pide datos, el servidor recibe esta petición y envía el bloque requerido de datos. Dado que el servidor no puede realizar selecciones de los datos inteligentes, no es posible perfeccionar traslados de los datos.

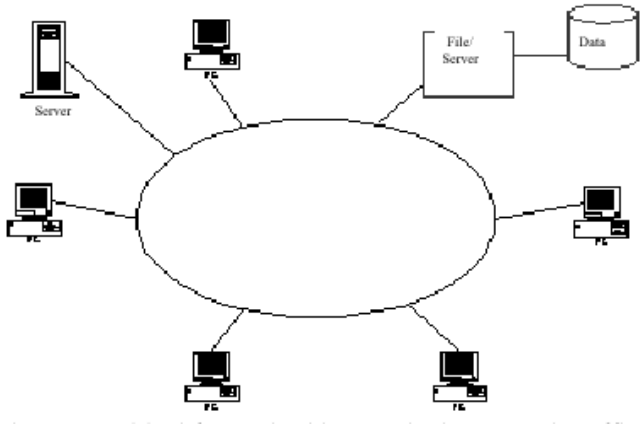


Figura 1.2

El factor más crítico, en este caso, es el tráfico de la red. También es difícil de lograr Integridad Transaccional completa.

1.3.3. Client/Print Server

El esquema Client/Print Server perfecciona el uso de recursos de impresión costosos, estos son compartido entre los múltiples usuarios, y también descentraliza el proceso requerido para generar la impresión exterior.

Mientras cada cliente procesa los datos que se imprimirán, el servidor proporciona servicios spooling similares a aquellos ya ofrecidos por esquemas centralizados. Un esquema básico se muestra en la Figura 1.3.

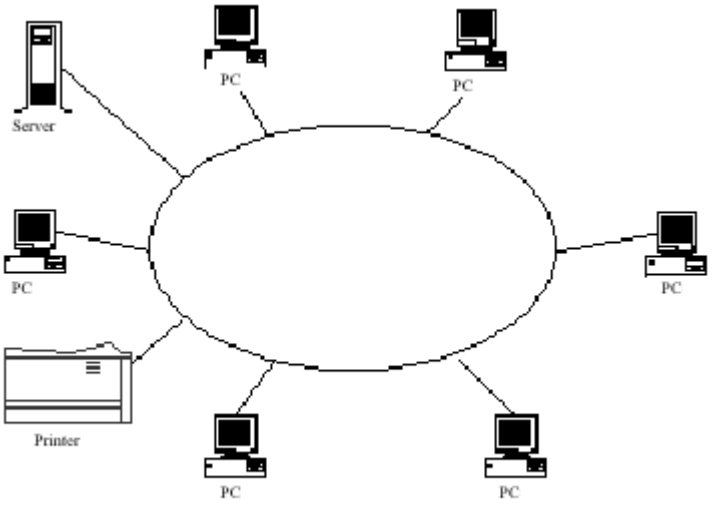


Figura 1.3

1.3.4. Client/Mail Server

El esquema Client/Mail Server (Figura 1.4) perfecciona el uso de recursos de comunicación (para casos de comunicación remotos) y maneja el correo. También descentraliza el proceso de generaciones del correo (cada cliente realiza esta tarea).

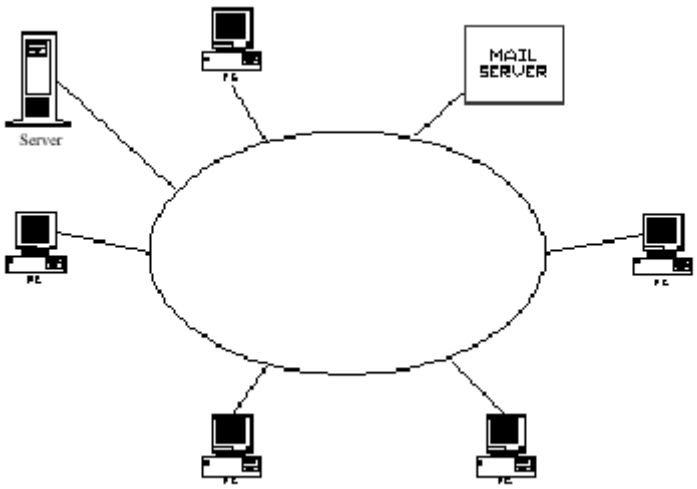


Figura 1.4

1.3.5. Client/Database Server

El esquema Client/Database Server (Figura 1.5) proporciona desde el punto de vista de base de datos, los mismos medios ofrecidos por un ambiente centralizado (información continua, actualizaciones, integridad, dirección, etc.). La característica que la diferencia de un Client/File Server es ser un servidor inteligente permitiéndoles a los clientes realizar consultas directas a los datos en lugar de las consultas archivo / registro (file / record queries).

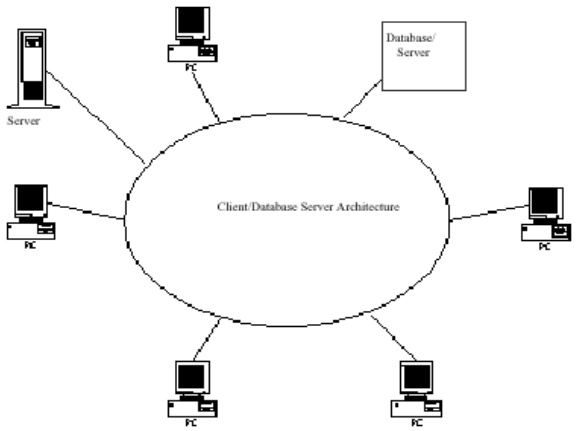


Figura 1.5

Las ventajas principales que ofrece este esquema son:

- Sólo se transmitirán los datos que estrictamente necesitan ser transmitidos. Esto significa que el tráfico de la red está muy reducido comparado al Servidor de Client/File.
- Usted deja de mantener la integridad transaccional completa porque las tomas de DBMS se hacen cargo de eso.

1.3.5.1 Arquitectura de Client/Database Server

Un asunto crucial en todas las arquitecturas arriba expresadas (Centralización, Client/File Server, Client/Database Server) es el diseño de la Base de datos. Sin embargo, el esquema de servidor de base de datos debe asegurar que todos los datos que transfiere afuera son sólo aquellos que son estrictamente necesarios. Esto significa que un buen diseño juega un papel importante en este esquema.

A continuación se presenta la Figura 1.6 de la arquitectura general Client/Database Server

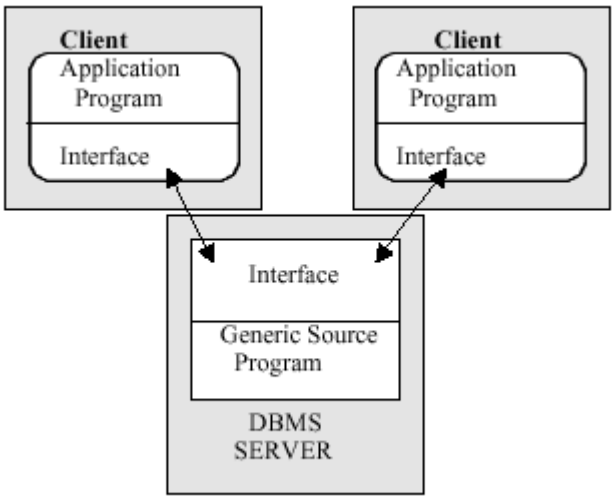


Figura 1.6

La Figura 1.7 despliega algunos de los niveles e interfaces con las que nosotros nos encontramos cuando queremos acceder a los datos en un ambiente Cliente/Servidor.

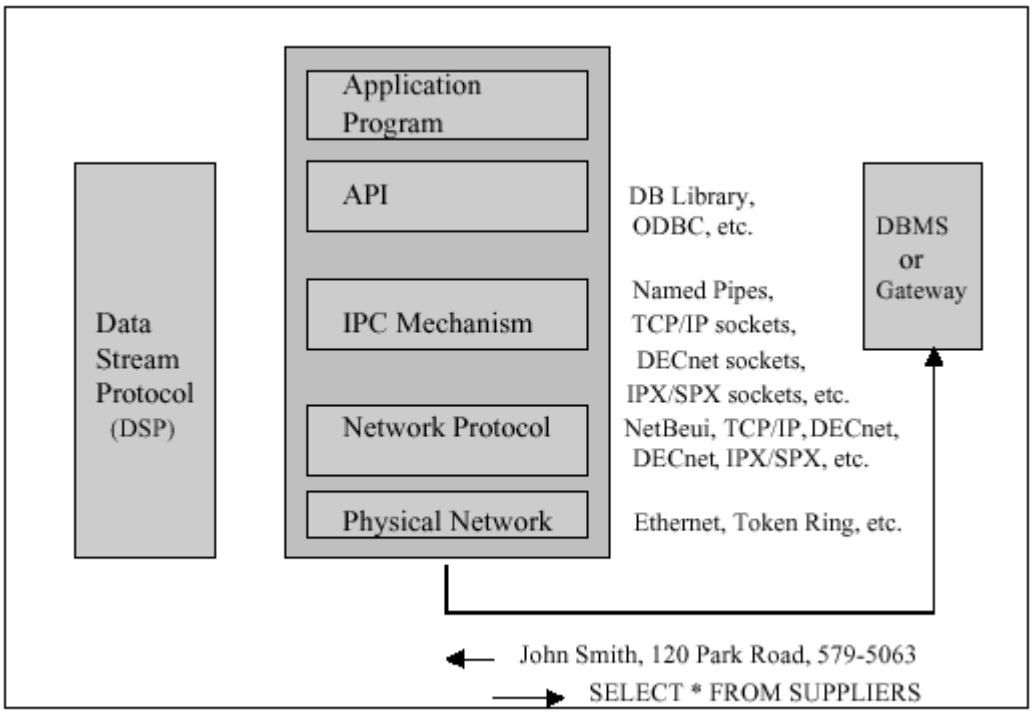


Figura 1.7

a. Application Programming Interfaces (API)

Cada DBMS tiene su propia Interfase de Programación de Aplicación API (Aplication Programming Interface), habilitando la comunicación con los Clientes. Hay por consiguiente, diferentes API para cada DBMS. Cada aplicación del Cliente debe, por consiguiente poder transformar demandas o requerimientos y los datos transfiere de manera que ellos puedan ser usados por la interfase de API apoyados por el acceso del DBMS.

b. Data Stream Protocols

Cada DBMS usa un Protocolo de Flujo de Datos (DSP), eso habilita el traslado de demandas, datos, el estado, mensajes del error, y así sucesivamente entre los DBMS y sus Clientes. Este puede considerarse como un "protocolo de comunicación lógico."

API usa mecanismos de comunicación Inter.-process (IPC), apoyado por el sistema operativo y la red, para empaquetar y transportar este protocolo lógico.

Generalmente cada DBMS ha desarrollado y ha perfeccionado su propio DSP, para trabajar exclusivamente con el DBMS. Esto, como una consecuencia, si la aplicación accede a Bases de datos múltiples, entonces debe usar múltiples DSP.

c. IPC Mechanisms

Hay diferentes mecanismos de comunicación usados entre los procesos (IPC) para transferir requisitos e información entre los DBMS y los Clientes. La selección del mecanismo de IPC que se usará directamente depende del Sistema Operativo y la red en la que la aplicación está ejecutándose.

d. Network Protocols

Se usan transporte o protocolos de la Red para transportar el Protocolo de Flujo de Datos sobre la red. Puede ser considerado como el plumbing este soporta el mecanismo de IPC eso se lleva a cabo con el protocolo de flujo de datos. También apoya las operaciones básicas de la red como transferencia de archivo y la impresión compartida.

Los protocolos de la red más populares son: NETBIOS, NetBEUI, TCP/IP, DECnet, IPX, y SPX. Puede acceder a bases de datos remotas vía LAN (Red del Area Local), WAN (Red de área Ampliada) o vía GETWAY. En todo caso, es posible que los protocolos de red usados para acceder a bases de datos remotas sean diferentes. Esto significa que una aplicación pueda requerir múltiples protocolos de red para acceder a varios DBMS back-ends.

Pero afortunadamente, todos los DBMS GENEXUS trabajan apoyados por todas las plataformas qué hace uso de todos los protocolos de comunicación de conexión de cliente que a su vez soportan Win 3.x, Win 95 y/o Win NT para que el servidor sea usado.

1.3.5.2. Arquitectura Client/Database Server con múltiples bases de datos

Una variación al Servidor de Bases de datos es acceder a Bases de datos múltiples de un Cliente, como se muestra en la Figura 1.8, incluso si ellos están en plataformas diferentes.

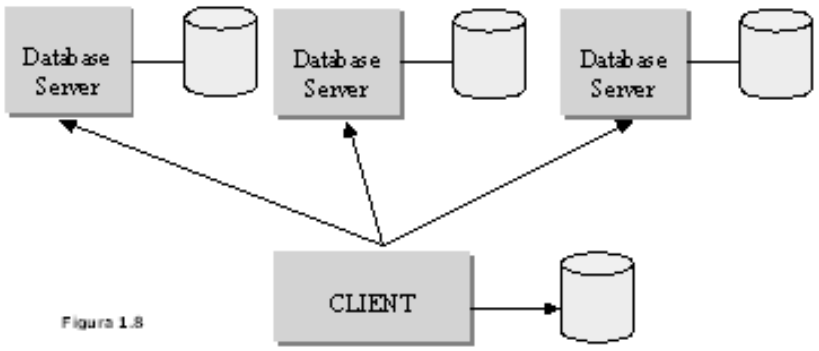


Figura 1.8

Los esfuerzos técnicos que se han llevado a cabo para soltar aplicaciones comerciales con acceso a Bases de datos múltiples no han sido suficientes. Los problemas experimentados por esta idea no han estado totalmente resueltos.
Algunos de estos problemas se refieren a:

- La administración de transacciones distribuidas.
- La replicación de datos.

- El acceso a diferentes plataformas, sistemas operativos, sistemas operativos de red, y los bases de datos diferentes.

1.4. Las bases de datos en el diseño de los sistemas de información

La rápida adopción por parte de las corporaciones de la tecnología de bases de datos está obligando a los fabricantes de software y departamentos de desarrollo a rediseñar sus respectivas arquitecturas de sistemas.

Las redes para PC tienen que evolucionar para proporcionar integración entre los múltiples sistemas operativos y aplicaciones utilizadas hoy en día fundamentadas principalmente en el poder de las bases de datos y las herramientas para administrarlas.

Existe un gran número de fabricantes que poseen bases de datos, herramientas de desarrollo de aplicaciones, compiladores, herramientas de consulta, etc., cada compañía ofrece un conjunto integrado de herramientas diseñadas para trabajar tanto con las nuevas aplicaciones Cliente/Servidor como con las aplicaciones ya existentes. Este es el marco donde se crea la relación entre las bases de datos y los sistemas de información.

Desde que se enunciaron las bases matemáticas del modelo relacional, no existía duda de que su éxito y su extensión iban a ser grandiosos, aunque, la intención real era la de convertirse en un sublenguaje universal de datos basado en el cálculo aplicado a predicados, la misma que no se ha cumplido totalmente, su universalidad incompleta, la imposibilidad de tratar con ciertos tipos de datos específicos es lo que ha incitado la creación de herramientas de administración de datos y también de las bases de datos orientadas a objetos.

El modelo relacional ha sido el primero que se ha sentado sobre los fundamentos matemáticos tan sólidos como son el álgebra relacional y el cálculo relacional. Su materialización es el llamado SQL, del cual existen varios dialectos como fabricantes, pero siempre todos respetando un conjunto de instrucciones estándar, definido por

un comité de normalización llamado ANSI. En entornos de trabajo según la arquitectura Cliente/Servidor, el lenguaje SQL es el intermediario principal y directo de todos los mensajes que gestionan las bases de datos, incluso indirectamente (SQL embébido o inmerso) que actúa desde el interior de las aplicaciones cliente, sin embargo, la mayoría de los administradores de bases de datos que implementan el SQL no son completamente relacionales, tal como sucede con los modelos teóricos, estos solo sirven como un ideal regulativo pero difícilmente se convertirán en realidad. Las Leyes de Codd se han convertido en auténticas normas del Modelo Relacional (Formas Normales), estableciendo una serie de reglas que debería satisfacer todo sistema de este tipo, así, algunos DBMS Relacionales, como DB2 u ORACLE, no superarían el 75% de cumplimiento de tales normas.

Las características del modelo relacional están basadas en almacenar los datos en tablas de dos dimensiones, sin grupos de repetición. Así, los registros se llaman filas o tuplas, y los campos, columnas o atributos. Cada tupla ha de ser única, no repetida y cada valor de la base de datos, diferente, esto constituye la llamada primera fórmula normal (NF1). Las relaciones entre las tablas se establecen mediante la conexión de atributos comunes, llamados claves. Las principales operaciones que pueden realizarse con tablas, son unión, diferencia, intersección, selección y proyección.

El mercado de software tiene una gran variedad de tecnología de bases de datos, herramientas de programación e interfases de aplicación diseñados para facilitar la compleja tarea de crear aplicaciones corporativas, generalmente en plataformas Cliente/Servidor.

Mientras sus productos y estrategias Cliente/Servidor difieren considerablemente, su objetivo es el mismo: capturar la base del mercado cliente/servidor y redirigir los esfuerzos de desarrollo futuros de las corporaciones hacia sus propias arquitecturas.

Por ejemplo, Microsoft enfoca el mercado Cliente/Servidor desde su posición como fabricante de sistemas operativos y aplicaciones para PC. Su estrategia es globalizar el sistema operativo Windows, y para promocionar esta idea ha creado la Arquitectura de Servicios Abiertos Windows (WOSA: Windows Open Services Architecture), un

conjunto de funciones llamadas API proporcionan una amplia variedad de servicios a las aplicaciones Windows.

El objetivo de diseño más allá de WOSA es permitir a diferentes tipos de aplicaciones Windows acceder de una forma coherente y consistente a servicios y datos desde computadores en red.

Microsoft estuvo durante mucho tiempo ausente del mercado de las bases de datos. Pero en los últimos años ha adquirido tecnología clave de otras compañías. Actualmente, Microsoft vende SQL Server (desarrollado por Sybase, Inc), una base de datos sobre OS/2 y LAN Manager para crear aplicaciones con soporte de decisiones, además de FoxPro (adquirido gracias a la absorción de FoxBase Inc) y el reciente Access, desarrollado internamente por la compañía. Estos dos últimos programas compiten directamente con dBASE y Paradox de Borland.

Para mejorar el intercambio y acceso de información desde las aplicaciones cliente al servidor de bases de datos, Microsoft creó un API o conjunto de funciones denominado ODBC está diseñado para permitir a Microsoft y a las aplicaciones de otras compañías acceder a las bases de datos Microsoft. Soportando ODBC en SQL Server y Access, estas dos bases de datos pueden intercambiar información entre ellas y entre otras.

Varios fabricantes independientes han anunciado que soportan esta especificación ODBC y se espera que todos la integren en sus productos a un costo casi insignificante.

Borland se introdujo en el mercado Cliente/Servidor a través de bases de datos y herramientas de programación que no incorporaban SQL. dBASE de Borland es un líder en el mercado, seguido por la última versión de la base de datos relacional Paradox y la avanzada ingeniería de base de datos Interbase, que Borland adquirió al absorber Ashton-Tate.

La visión del futuro de Borland toma forma bajo la arquitectura BOCA (Borland Object Component Architecture), una arquitectura software multinivel que intenta romper los límites tradicionales entre las aplicaciones permitiendo a cualquiera de una gran variedad de aplicaciones compartir funciones y datos a través de una serie de interfases.

En el centro de esta arquitectura, se sitúa la ingeniería Interbase, que actúa como administrador central del código de programación y los datos de la aplicación.

Al igual que la estrategia de Microsoft, BOCA está diseñado para integrar los diversos productos Borland a la vez que contempla a los fabricantes independientes.

Los componentes primarios de BOCA son Borland Desktop, que comparte objetos con otras aplicaciones, y Borland Database Engine, una librería DLL (realmente, un motor de base de datos), que residirá eventualmente en el núcleo de todos los productos Borland de gestión de datos.

Ambas compañías están también planeando soportar la librería API definida por SQL Access Group, un enfoque a la interfase Cliente/Servidor a un nivel más bajo.

El SQL Access Group está compuesto por la mayoría de los principales fabricantes de bases de datos y LAN, y ya han creado un conjunto estándar de llamadas de programación que, al menos en teoría, permitirán a cualquier front-end acceder de forma consistente a cualquier servidor.

Indudablemente, los sistemas de información en la actualidad deben mantener una relación estrecha con las bases de datos, un buen diseño de datos seguramente significa que las decisiones que se tomaran a partir del mismo estarán debidamente sustentadas.

La mayoría de diseñadores de software consideran el almacenamiento de los datos como la esencia de los sistemas de información. Los objetivos generales del diseño

de la organización del almacenamiento y administración de los datos se muestran en la Figura 1.9.

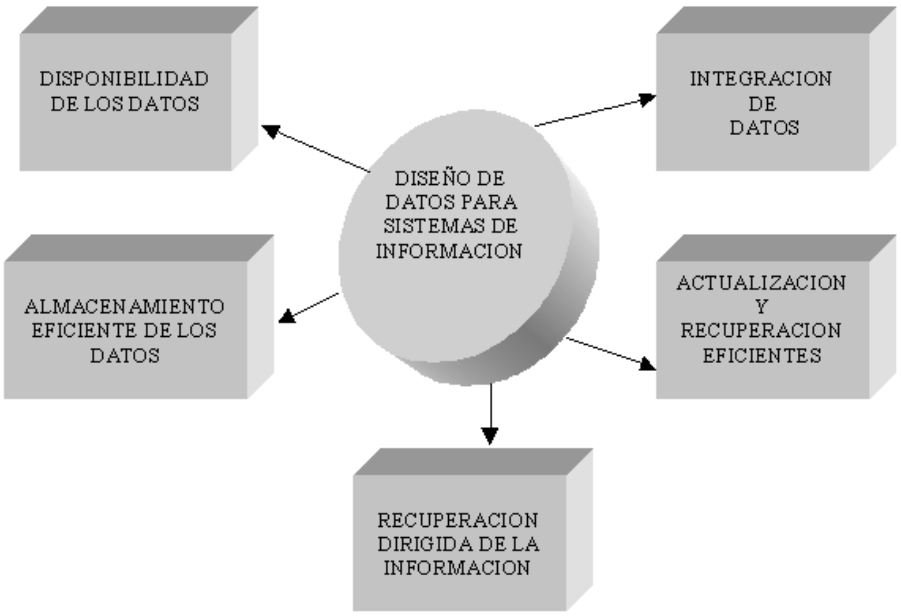


Figura 1.9

Primero, los datos deben estar dispuestos para cuando el usuario desee utilizarlos. Segundo, los datos deben ser precisos y consistentes (deben poseer una integridad). Mas allá de esto, dentro d los objetivos se incluye un almacenamiento eficiente de los datos, así como su actualización y grabados eficientes. Finalmente, es necesario que el acceso a la información tenga un propósito. La información obtenida de los datos almacenados debe contar con un formato útil que facilite la administración, la planeación, el control o la toma de decisiones.

En un sistema de información basado en computadora se cuenta con varios enfoques para el almacenamiento de los datos, pero mencionaremos los dos más convencionales. El primer método consiste en almacenar los datos en archivos individuales, siendo estos exclusivos para una aplicación en particular. Este método implica ciertas desventajas, tales como: que la aplicación es propietaria a perpetuidad de los datos, debido a que es la única que puede administrarlos en ese formato de su exclusivo reconocimiento, el esfuerzo para recuperar o introducir la información se

multiplica por tantas veces se repitan las instancias de datos en los diferentes archivos, de igual manera un cambio exigiría la actualización de varios archivo al mismo tiempo, la forma de relacionar estos archivos de datos se volvería compleja; en fin, esto solo por mencionar algunas desventajas, sin desmerecer que en ciertas circunstancias especiales (seguridad, por ejemplo) esta podría ser una alternativa válida.

El segundo enfoque para el almacenamiento de datos en un sistema basado en computadora, involucra el diseño lógico y físico de una base de datos. Una base de datos es un almacenamiento de datos formalmente definido, controlado centralmente para intentar servir a múltiples, diferentes y dispersas aplicaciones

Como conclusión, puede afirmarse que está en auge una auténtica revolución para acercar a la informática todos los tipos de datos existentes, muchos de ellos, aislados por la industria durante algún tiempo. En ella, se exploran múltiples soluciones de alto nivel, pero el modelo relacional, es el que tiene más futuro.

1.5. Uso de los lenguajes de consulta

Inevitablemente se ha tenido que mencionar al SQL en algunos párrafos anteriores a esta sección, pero, es tanta la relación existente entre este y las bases de datos que casi al hablar de uno se habla del otro.

SQL es la abreviatura de Structured Query Language (lenguaje de consulta estructurado). Se trata de un lenguaje estándar para acceder a los datos almacenados en una base de datos. Su utilidad radica en que casi todas las bases de datos se manejan de forma más o menos distinta, pero la mayoría de los gestores de bases de datos más potentes permiten que se les hagan consultas en este lenguaje.

Parece razonable suponer que quien quiere aprender SQL es porque ya tiene unos conocimientos elementales de bases de datos, y que entiende conceptos como tabla, campo, registro, etc. Aun así, las ideas básicas, presindiendo de todo tipo de formalismos (incluso demasiado) son:

- Una tabla es un conjunto de datos con una cierta homogeneidad.
- Un registro sería cada uno de los bloques de datos.
- Un campo es cada uno de los datos que componen un registro.

La forma habitual de representar una tabla de datos es como se muestra en la Tabla 1.1:

Tabla 1.1

NOMBRE	Héctor Armas	José Luis Armas	Manuel Cárdenas
DOMICILIO	Bolívar y Flores	La Victoria	Paraíso y El Cielo
TELÉFONO	06955248	099840734	06951405
FECHA DE NACIMIENTO	10/11/1972	02/04/1980	25/12/1920

En esta representación:

- El conjunto de todos los datos es la tabla.
- Cada fila representa un registro de la tabla.
- Cada columna son los distintos valores que toma cada uno de los campos.

Se indican a continuación sólo unas nociones básicas de lo que se puede realizar con SQL, dejando de lado muchos aspectos más avanzados como la creación de bases de datos y de tablas, las consultas agrupadas y los sub-queries (subconsultas).

- Consultar datos de una tabla.
- Consultar datos que cumplan ciertas condiciones.
- Mostrar datos de varias tablas.
- Operaciones elementales con funciones agregadas.
- Buscar un cierto texto en una tabla.
- Actualizaciones repetitivas.

Las acciones descritas se pueden realizar con el grupo de palabras reservadas básicas que se detallan en la Tabla 1.2 .

Tabla 1.2

Palabra Reservada SQL	Aplicación ...
SELECT	Para identificar las columnas que se desea desplegar
WHERE	Para aplicar un filtro en el despliegue de columnas
ORDER BY	Para generar un orden en el conjunto de registros a desplegar
INSERT	Para adicionar nuevos registros en la tabla
DELETE	Para eliminar registros de una tabla
UPDATE	Para modificar los campos de una tabla

En el Capítulo 2 "Manejo de datos compartidos usando SQL" se tratará con mas detalle el estándar SQL.

1.6. Los lenguajes de programación como herramientas de apoyo a los procesos de comunicación de información entre bases de datos

La preocupación de las empresas hacia el desarrollo de productos que permitan el intercambio de información entre diversos DBMS's sin conflictos para el usuario final, motivaron que surgieran varias tecnologías nuevas para el programador, tales como las especificaciones ODBC de Microsoft, IDAPI de Borland y RDA de ISO.

1.6.1. Microsoft ODBC

ODBC es una especificación para una base de datos API. Este API es independiente de cualquier DBMS o sistema operativo; el ODBC API es también independiente del lenguaje de programación. El ODBC API se basa en las especificaciones CLI de X/Open e ISO/IEC. ODBC 3.x cumple totalmente ambas especificaciones –las versiones preliminares de ODBC estaban basadas en las versiones preliminares de estas especificaciones pero no se implementaron totalmente y agrega aspectos usualmente necesarios para desarrolladores de aplicaciones de base de datos basadas en pantallas, tales como cursores de despliegue.

Las funciones del ODBC API son implementadas por desarrolladores de drivers específicos DBMS. Las aplicaciones llaman a las funciones con estos drivers para acceder a los datos en un DBMS de manera independiente. Un administra la comunicación entre las aplicaciones y los drivers.

Aunque Microsoft provea un Driver Manager para computadoras que corren Microsoft Windows NT® Server, Windows 2000 Server, Microsoft Windows NT Workstation, Windows 2000 Profesional, y Microsoft Windows® 95/98, ha escrito varios drivers ODBC, y llama a funciones ODBC desde algunas de sus aplicaciones, cualquier persona puede escribir aplicaciones ODBC y drivers. De hecho, la extensa mayoría aplicaciones ODBC y drivers disponibles para computadoras que corren Windows NT Server, Windows 95/98 son producidas por otras compañías no Microsoft. Además, existen drivers ODBC y aplicaciones sobre Macintoch® y una variedad de plataformas UNIX.

Para ayudar a los desarrolladores de drivers y aplicaciones, Microsoft ofrece un ODBC Software Development Kit (SDK) para computadoras que corren Windows NT Server, Windows 2000 Server, Windows NT Workstation, Windows 2000 Profesional, y Windows 95/98.

Es importante comprender que ODBC se diseñó para exponer las capacidades de la base de datos, no para complementarla. Así, los escritores de aplicaciones no deberían esperar que el uso de ODBC transforme repentinamente una base de datos simple en un motor de base de datos totalmente relacional. Ni los escritores de drivers esperaron implementar funcionalidades no encontradas en su base de datos subyacente. Una excepción a esto es que los desarrolladores que escribieron drivers que accedan directamente a los archivos de datos (tales como datos en un archivo Xbase) se requieran para diseñar un motor de base de datos que se apoye por lo menos en funcionalidades mínimas de SQL. Otra excepción es que el driver ODBC de Microsoft® Data Access Component (MDAC) SDK provee una biblioteca de cursores que emula a cursores de despliegue para drivers que implementan un nivel seguro de funcionalidad.

1.6.2. Borland IDAPI

Integrated Database API (IDAPI), es la API empleada por el motor BDE (Borland Database Engine) en sus productos Delphi, dBase y Paradox para Windows. Unifica los tipos de acceso de datos ISAM y los orientados a consultas (SQL o QBE) en un modelo de cursor consistente. Los principales objetivos de IDAPI son:

- La separación del lenguaje de acceso de datos y el formato de archivo de datos.
- El acceso directo contra la importación o exportación de datos.
- Una API única para las fuentes de datos nativas y no nativas.
- La unificación del acceso a la PC y a las bases de datos SQL de otras plataformas.
- El desarrollo simplificado de aplicaciones para acceso a bases de datos.

El proyecto inició en 1990 como ODAPI (Open Database API) 1.0, incluida en el producto Quattro Pro 1.0 y en Paradox 1.0 para Windows y constaba del motor y controladores para Paradox y dBase y del motor para QBE (Query By Example). En 1993, fue liberada la versión 1.1 incluida en Quattro Pro 5.0 y Paradox 4.5, que incluía los controladores SQL para Interbase, Oracle y Sybase. En 1994 fue liberado el controlador para Informix SQL. La siguiente versión fue el BDE 2.0, el cual aportaba el trabajo realizado por el Comité Técnico de IDAPI (conformado por Borland, IBM, Novell y WordPerfect) y ODBC Sockets, que convertían un controlador ODBC en un controlador IDAPI. La más reciente versión es la 2.5, incluida en todas las versiones de Delphi, y conteniendo el SQL Links 2.5.

IDAPI, tal como ODBC, es una arquitectura basada en controladores; para cada fuente de datos distinta, debe haber un controlador separado. Una de las características importantes es que IDAPI está orientado a objetos, haciendo su infraestructura altamente extensible y personalizable. También proporciona un rico conjunto de servicios tales como el administrador de buffers, ordenamiento, controladores de lenguajes, entre otros, los cuales son compartidos por todos los controladores. La Figura 1.10 muestra esta arquitectura.

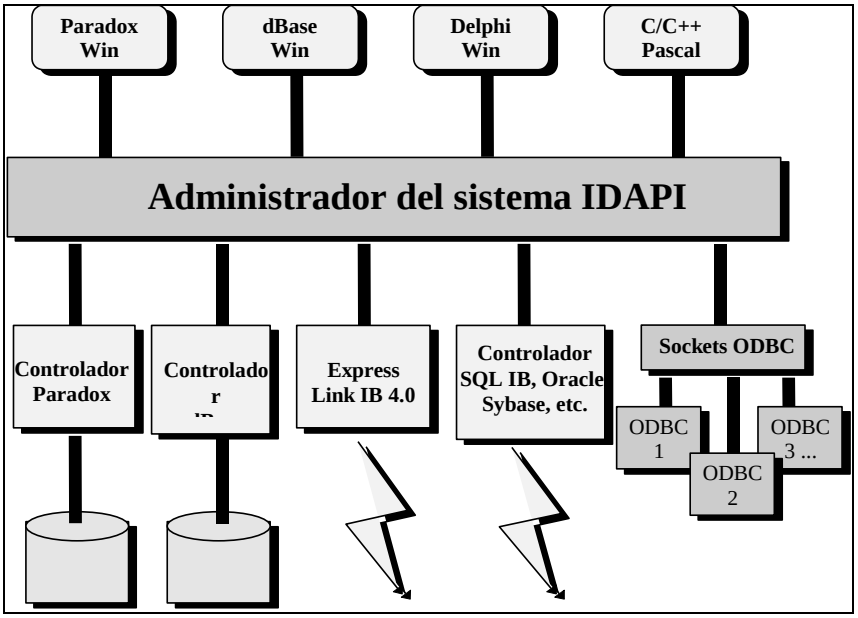


Figura 1.10

El Administrador del sistema IDAPI es el que realiza todas las tareas de control: carga los controladores a demanda, administra su configuración, guarda la pista de los recursos tal como las bases de datos abiertas, los cursores y otros. Además, cuando termina la aplicación cliente, libera y limpia todos los recursos empleados por la aplicación. Para maximizar la reutilización de código y mantener al sistema modular y portable, la infraestructura IDAPI proporciona un número de servicios, que los controladores llaman para hacer sus acciones. Todos los controladores basados en SQL están construidos usando estos servicios, que constituyen cerca del 80% de la funcionalidad del controlador, lo que indica que solo un 20% de código para un nuevo controlador debe ser escrito. Algunos de estos servicios incluyen la API de mapeo navegacional hacía SQL, el almacenamiento local de registros, los servicios de requerimientos de esquema y la lógica de manejo de campos Blob (Binary large objects), entre otros.

Claramente se observa que el desarrollo de esta tecnología fue para competir directamente con el mercado de ODBC. Reconociendo esto, se proporcionan los llamados ODBC Sockets, que son módulos agregados a los controladores ODBC para trasladar la API de ODBC a la API de IDAPI, incrementando con ello la variedad de fuente de datos disponibles para conexión.

1.6.3. ISO Remote Database Access (RDA)

El RDA es un protocolo de comunicaciones para el acceso a bases de datos remotas, que fue adoptado por la ISO y por la International Electrotechnical Commission (IEC). Está basado en el modelo Cliente/Servidor, en donde el cliente actúa como un programa de aplicación, mientras que el servidor se auxilia de un intermediario el cual es un proceso que controla la transferencia de datos de y hacia la base de datos. El principal propósito es proporcionar la ínter conectividad entre ambientes heterogéneos. La especificación RDA consta de dos partes:

1. RDA Genérico, el cual es un modelo, servicio y protocolo básicos para la conexión a una base de datos arbitraria.
2. SQL a la medida, que añade protocolos específicos para la conexión a bases de datos que estén de acuerdo al SQL.

Un cliente RDA es un proceso, dentro de un sistema abierto, que hace peticiones a servicios de otro proceso llamado el servidor de bases de datos. Este servidor de bases de datos proporciona facilidades de almacenamiento y, mediante una estructura de comunicación, servicios de bases de datos a los clientes RDA. La comunicación entre el cliente y el servidor RDA se realiza mediante un servicio RDA, soportado por un proveedor de servicios RDA, como se muestra en la Figura 1.11 .

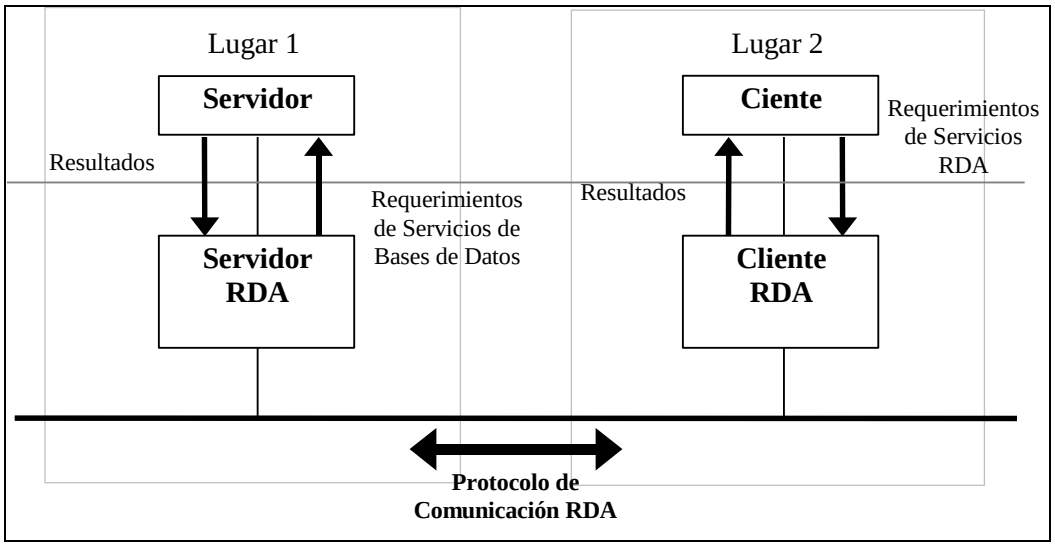


Figura 1.11

Una sentencia SQL es ejecutada por el servidor tal y como si estuviera incrustada dentro de un lenguaje huésped local al recurso de bases de datos SQL. Cualquier excepción o condición de terminación surgida en el servidor es enviada al cliente RDA. Durante la inicialización del diálogo en un ambiente SQL, el cliente y el servidor RDA acuerdan el nivel de conformidad al cual operarán.

Una transacción RDA es una unidad lógica de procesamiento determinada por el cliente RDA. Las transacciones son manejadas como unidades atómicas. Cuando una transacción RDA es terminada, todo el conjunto de cambios a los datos es aplicado totalmente o no. El cliente RDA envía una petición de terminación de una transacción RDA cuando envía una petición de compromiso o aborto al servidor RDA. Los cambios realizados durante una transacción no están disponibles a otros clientes RDA hasta que la transacción es terminada por el servidor, por lo que garantizan el aislamiento entre transacciones. RDA proporciona dos métodos de manejo de transacciones:

1. Contexto básico de aplicación para el compromiso a una fase.
2. Contexto de Procesamiento Transaccional de aplicación para el compromiso de dos fases.

Un cliente RDA puede requerir uno de cinco servicios RDA:

1. Administración de Diálogo RDA.- para iniciar y terminar un diálogo RDA.
2. Administración de Transacciones RDA.- para iniciar o terminar transacciones RDA.
3. Control RDA.- reporta estados o cancela operaciones existentes.
4. Manejo de Recursos.- para habilitar o deshabilitar el acceso a recursos de datos RDA.
5. Lenguaje de Base de Datos.- para acceder y modificar los recursos de datos.

El estándar RDA de 1993, conocido como RDA-3, aún no soporta todas las facilidades del estándar SQL/92. Al final de 1994, no había muchos productos que soportaran el estándar RDA, mientras que otras implantaciones fueron realizadas por el Gobierno de E. U. basadas en este protocolo. RDA es apropiado en lugares donde los protocolos a nivel de transporte ya estén establecidos, tal como en TCP/IP y OSI. Se

espera que RDA sea empleado para la interconexión entre productos de bases de datos SQL de diferentes vendedores. La intercomunicación entre productos del mismo vendedor continuará utilizando formas de comunicación e intercambio específicas del vendedor.

CAPÍTULO II

MANEJO DE DATOS COMPARTIDOS USANDO SQL

Este capítulo discute algunos de los conceptos e historia detrás el desarrollo de SQL, incluyendo:

- SQL y las diversas maneras que las aplicaciones lo usan.
- Como se realiza el acceso a bases de datos de red en el mundo real y cuáles partes de este proceso pueden ser fácilmente normalizadas.

2.1. La estandarización de SQL

En la década de los años setenta, y bajo la línea de investigación de la empresa IBM, E. F. Codd publicó el artículo "A Relational Model for Large Shared Data Banks", en donde sentó las bases de lo que denominó el modelo relacional, que intentaba ser un modelo conceptual para la administración de bases de datos. Varias empresas se interesaron en esos conceptos y desarrollaron algunas implementaciones de lenguajes que trataban de seguir este modelo, siendo el más significativo el Structured English Query Language (SEQUEL), definido por Donald Chamberlain y su equipo en 1974, e implantado en el prototipo SEQUEL-XRM. Una segunda revisión produjo el SEQUEL/2 en 1976, que por razones legales se llamó SQL (Structured Query Language), y que fue implantado en el proyecto System R, el cual inició su funcionamiento en 1977 y dentro de las instalaciones de IBM. El primer producto comercial que incluyó al SQL fue Oracle, de Relational Software Inc. Debido a la rápida aceptación de los productos comerciales que empleaban al SQL, la American National Standards Institute (ANSI), lo aprobó como estándar en 1986, siendo la primera versión denominada informalmente como SQL/86. En 1987, también fue aceptado como un estándar por la International Standards Organization (ISO). En 1989 se extendió para soportar características de integridad de datos y se desarrolló el SQL incrustado, lo que dio lugar al SQL/89. Se siguieron añadiendo funcionalidades al lenguaje y en 1992 se presentó otra versión, el SQL/92, ratificado como estándar en el documento de la ISO/IEC 9075:1992 y siendo soportada rápidamente por la mayoría de los productos

comerciales de bases de datos basados en el modelo relacional. Actualmente está en revisión una nueva versión, denominada SQL3, la cual se espera sea publicada como estándar ANSI e ISO en 1999, e incluye nuevas capacidades, además de las presentadas en las anteriores versiones, como nuevos tipos de datos del lenguaje (BOOLEAN, BLOB y CLOB) y definidos por el usuario (CREATE TYPE); soporte para subtablas y supertablas manejadas por herencia, vistas temporales, agregados a sentencias para manejar cursores (cláusulas SENSITIVE y WITH HOLD), expresiones recursivas para tablas (cláusula RECURSIVE), cuantificadores nuevos (cláusulas FOR SOME | ALL), agregados a condiciones (cláusulas SIMILAR y DISTINCT), facilidades para la creación de tabulaciones cruzadas, mejoras para la integridad y seguridad de los datos , así como la capacidad de continuar o terminar una transacción en un punto determinado (cláusula SAVEPOINT), entre otras características.

El SQL es soportado por la mayoría de los DBMS, aunque cada uno adiciona ciertas características que incrementan su funcionalidad y rendimiento. Igualmente, para la creación de aplicaciones, cada uno proporciona su lenguaje de programación, que no está en conformidad con algún estándar. Esto implica que para crear una aplicación que acceda a un DBMS, hay que aprender la sintaxis del lenguaje de programación de cada uno de los diferentes DBMS del mercado, lo que ocasiona que los programadores expertos en lenguajes como C, COBOL, Pascal, Ada, RPG, PL/1 y otros, tuvieran que aprender cada lenguaje nativo. Era necesario establecer una forma de crear aplicaciones que accederán varios DBMS sin que implicara mucho trabajo para el programador y con una especificación común y accesible que fuera aceptada por los fabricantes de los diferentes.

2.2. Gramática del SQL/92

La gramática SQL mostrada a continuación es la empleada por la especificación ODBC

Tabla 2.1

Sentencia SQL	Nivel de Conformidad
alter-table-statement ::= ALTER TABLE base-table-name { ADD column-identifier data-type | ADD (column-identifier data-type [, column-identifier data-typel]...) }	Esencial
alter-table-statement ::= ALTER TABLE base-table-name	Extendida

Sentencia SQL	Nivel de Conformidad
{ ADD column-identifier data-type | ADD (column-identifier data-type [, column-identifier data-type]...) | DROP [COLUMN] column-identifier [CASCADE | RESTRICT] }	
create-index-statement ::= CREATE [UNIQUE] INDEX index-name ON base-table-name (column-identifier [ASC | DESC] [, column-identifier [ASC | DESC]]...)	Esencial
create-table-statement ::= CREATE TABLE base-table-name (column-element [, column-element] ...) column-element ::= column-definition | table-constraint-definition column-definition ::= column-identifier data-type [DEFAULT defau1t - va1ue] [column-constraint-definition[column-constraint-definition] ...] default - va1ue ::= 1iteral | NULL | USER column-constraint-definition ::= NOT NULL | UNIQUE | PRIMARY KEY | REFERENCES ref-tab1e-name referenced-columns | CHECK (search-condition) table-constraint-definition ::= UNIQUE (column-identifier [, column-identifier] ...) | PRIMARY KEY (column-identifier [, column-identifier] ...) | CHECK (search-condition) | FOREIGN KEY referencing-columns REFERENCES ref-table-name referenced-columns	Mínimo
create-view-statement ::= CREATE VIEW viewed-table-name [(column-identifier [, column-identifier]...)] AS query-specification	Esencial
delete-statement-positioned ::= DELETE FROM table-name WHERE CURRENT OF cursor-name	Esencial
delete-statement-searched ::= DELETE FROM table-name [WHERE search-condition]	Mínimo
drop-index-statement::= DROP INDEX index-name	Esencial
drop-table-statement ::= DROP TABLE base-table-name [CASCADE | RESTRICT]	Mínimo
drop-view-statement ::= DROP VIEW viewed-table-name [CASCADE | RESTRICT]	Esencial
grant-statement ::= GRANT { ALL | grant-privilege [, grant-privilege]... } ON table-name TO {PUBLIC | user-name [, user-name]... } grant-privilege ::= DELETE | INSERT | SELECT | UPDATE [(column-identifier [, column-identifier]...)] | REFERENCES [(column-identifier [, column-identifier]...)]	Esencial
insert-statement ::= INSERT INTO table-name [(column-identifier [, column-identifier]...)] VALUES (insert-value[, insert-value]...)	Mínimo

Sentencia SQL	Nivel de Conformidad
insert-statement ::= INSERT INTO table-name [(column-identifier [, column-identifier]...)] { query-specification | VALUES (insert-value [, insert-value]...)]	Esencial
ODBC-procedure-extension ODBC-std-esc-initiator [?=] call procedure ODBC-std-esc-terminator | ODBC-ext-esc-initiator [?=] call procedure ODBC-ext-esc-terminator	Extendido
revoke-statement ::= REVOKE {ALL | revoke-privilege [, revoke-privilege]... } ON table-name FROM { PUBLIC | user-name [, user-name]... } [CASCADE | RESTRICT] revoke-privilege ::= DELETE | INSERT | SELECT | UPDATE | REFERENCES	Esencial
select-statement ::= SELECT [ALL | DISTINCT] select-list FROM table-reference-list [WHERE search-condition] [order-by-clause]	Mínimo
select-statement ::= SELECT [ALL | DISTINCT] select-list FROM table-reference-list [WHERE search-condition] [GROUP BY column-name [, column-name]...] [HAVING search-condition] [order-by-clause]	Esencial
select-statement ::= SELECT [ALL | DISTINCT] select-list FROM table-reference-list [WHERE search-condition] [GROUP BY column-name [, column-name]...] [HAVING search-condition] [UNION [ALL] select-statement]... [order-by-clause]	Extendido
select-for-update-statement ::= SELECT [ALL | DISTINCT] select-list FROM table-reference-list [WHERE search-condition] FOR UPDATE OF [column-name [, column-name] ...]	Esencial
statement ::= create-table-statement | delete-statement-searched | drop-table-statement | insert-statement | select-statement | update-statement-searched	Mínimo
statement ::= alter-table-statement | create-index-statement | create-table-statement | create-view-statement | delete-statement-searched | drop-index-statement | drop-table-statement | drop-view-statement | grant-statement | insert-statement | revoke-statement | select-statement | update-statement-searched	
statement ::= alter-table-statement | create-index-statement | create-table-statement | create-view-statement | delete-statement-positioned	Extendido

Sentencia SQL	Nivel de Conformidad
| delete-statement-searched | drop-index-statement | drop-table-statement | drop-view-statement | grant-statement | insert-statement | ODBC-procedure-extension | revoke-statement | select-statement | select-for-update-statement | statement-list | update-statement-positioned | update-statement-searched	
statement-list ::= statement | statement; statement-list	Extendido
update-statement-positioned ::= UPDATE table-name SET column-identifier = { expression | NULL } [, column-identifier = {expression | NULL}]... WHERE CURRENT OF cursor-name	Esencial
update-statement-searched ::= UPDATE table-name SET column-identifier = {expression | NULL } [, column-identifier = {expression | NULL}]... [WHERE search-condition]	Mínimo

2.3. Técnicas de consulta con SQL

Un DBMS típico permite a sus usuarios almacenar, acceder, y modificar datos de una manera eficiente y organizada. Originalmente, los usuarios de DBMS's eran los programadores. El acceso a los datos almacenados requirieron escribir programas en un lenguaje de programación tal como COBOL. Mientras estos programas se escribieron frecuentemente para presentar una interfase amistosa al usuario no-técnico, el acceso a los datos requería de conocimientos de programación. El acceso casual o al instante a los datos era algo complejo.

Los usuarios no estaban completamente contentos con esta situación. Y cuando ellos necesitaban acceder a los datos lo frecuente era requerir que el programador del DBMS escribiera software especial. Por ejemplo, si un departamento de ventas quería ver las ventas totales en el mes previo por cada uno de sus vendedores y quería que esta información figure en orden por el volumen de ventas de cada vendedor de la compañía, tenía dos de opciones: Utilizar un programa que ya existe y que permita que la información sea recuperada exactamente como se la necesita, o el departamento tenía que pedir a un programador que escriba tal programa. En muchos casos, este trabajo era el que se requería, y además, era una solución costosa tanto en tiempo (desarrollo, pruebas, etc.) como en recursos económicos

(programadores, software, etc.). Como muchos usuarios necesitaron acceso fácil y diferentes reportes el problema creció más y de forma más compleja.

Las aplicaciones desarrolladas que emplean el SQL, pueden realizar consultas a un servidor de base de datos mediante tres técnicas comunes:

2.3.1. SQL Incrustado

También llamado SQL incrustado, en donde las sentencias SQL son incrustadas dentro del código fuente de un lenguaje huésped, como C, COBOL, Ada, Pascal u otro, permitiendo que un programador pueda tener acceso a un DBMS. Este código fuente es precompilado mediante un programa el cual traduce las sentencias SQL a llamadas de funciones nativas del DBMS, lo cual genera un nuevo archivo con código fuente en C puro, que será compilado y ligado con la librería de funciones nativas del DBMS, para finalmente crear una versión ejecutable de la aplicación. Una ventaja de este método es que la aplicación es altamente personalizable dependiendo de las capacidades ofrecidas por el lenguaje huésped.

2.3.2. Módulos SQL

Los módulos, que son grupos de procedimientos, son compilados y almacenados por el DBMS, pero son llamados desde el código fuente de un lenguaje huésped. Cada procedimiento contiene una sola sentencia SQL y los datos son transmitidos hacia cada procedimiento mediante sus parámetros. Puede pensarse que los módulos son parecidos a las librerías de objetos que son ligados al código de la aplicación; sin embargo, la implementación de los módulos es dependiente de la aplicación y del DBMS. La principal ventaja de este método es que las sentencias SQL están separadas completamente del código fuente de la aplicación; si alguno cambia, no es necesario (en teoría) modificar al otro.

2.3.3. Interfaz a Nivel de Llamadas

También conocido como Call Level Interface, o CLI, en donde las sentencias SQL son enviadas al DBMS mediante una serie de funciones, que constituyen una Interfaz de Programación de Aplicaciones (Aplication Programming Interface, API). Es similar al funcionamiento del SQL incrustado en el aspecto que se emplean llamadas a funciones, pero con la salvedad de que no es necesario el proceso de precompilación del código fuente, sino que la inclusión de las funciones de librería hacen que un módulo (proceso o aplicación) separado y residente en memoria, inicie la administración de las peticiones enviadas de la aplicación hacia el servidor, y de los resultados y mensajes de error del servidor hacia la aplicación, en tiempo de ejecución. Tiene como ventaja que el código fuente de las aplicaciones ya no es dependiente de las librerías nativas del DBMS, sino que usando la API estándar, el módulo residente se encarga de la conexión, procesamiento y desconexión con la fuente de datos, obteniendo un alto grado de interoperabilidad entre DBMS heterogéneos.

Aunque cada una de estas técnicas proporciona sus pros y sus contras, la más apta para la creación de aplicaciones genéricas ínter operables que empleen la arquitectura Cliente/Servidor es la Interfaz a Nivel de Llamada.

Realmente, el término Call Level Interface se usa en vez de Aplication Programming Interface (API), que es otro término para la misma cosa. En el mundo de las bases de datos. SQL es el API de los DBMS.

De las opciones anteriores, **SQL Incrustado** es el más usado usualmente, aunque la mayoría de DBMS's importantes soporten CLI's propietarios.

2.4. El procesamiento de una sentencia SQL

Antes de discutir las técnicas para usar SQL en programación, es necesario discutir como se procesa una sentencia SQL. Los pasos que involucra este proceso son comunes a las tres técnicas ya mencionadas, aunque cada técnica los desempeñe en

tiempos diferentes. La Figura 2.1 muestra los pasos que involucra procesar una sentencia SQL, y que serán discutidos a lo largo del resto de esta sección.

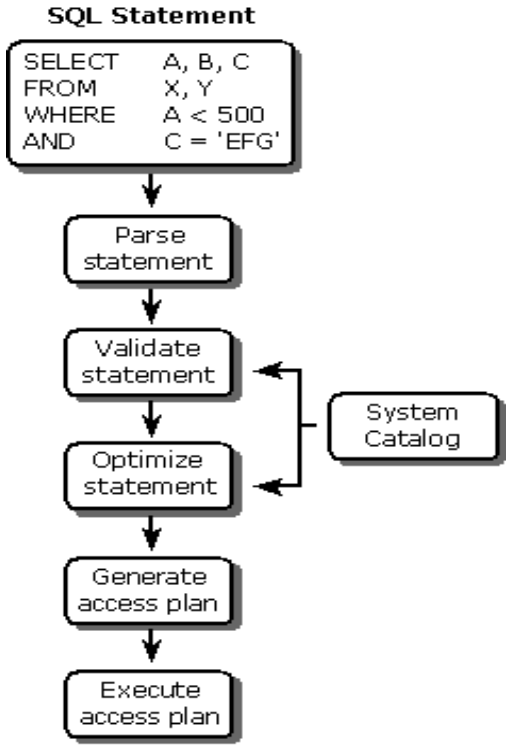


Figura 2.1

Para procesar una sentencia SQL, un DBMS sigue los siguientes cinco pasos:

1. El DBMS primero pasa la sentencia SQL por un parser. Esto es, divide la sentencia en palabras individuales, llamadas tokens, asegura que la sentencia tenga un verbo válido y sobre todo cláusulas válidas. Los errores de sintaxis y las palabras mal escritas pueden detectarse en este paso.

2. El DBMS valida la sentencia. Verifica la sentencia contra el catálogo de sistema. ¿Existen en la base de datos todas las tablas nombradas en la sentencia?, ¿Existen todas las columnas y sus nombres de columnas no son ambiguos?, ¿Tiene el usuario los privilegios requeridos para ejecutar la sentencia?. Ciertos errores semánticos pueden detectarse en este paso.

3. El DBMS genera un acceso plano acerca de la sentencia. El acceso plano es una representación binaria de los pasos que se requieren que efectúe la sentencia; para el DBMS es equivalente a código ejecutable.

4. El DBMS perfecciona el acceso plano. Explora diversas maneras para efectivizar el acceso plano. ¿Puede usarse un índice para acelerar una búsqueda?, ¿Debe el DBMS primero efectuar una búsqueda condicionada a la tabla A y entonces relacionarla con la tabla B, o debería primero relacionarlas y luego realizar la búsqueda condicionada?, ¿Puede evitar una búsqueda secuencial mediante una tabla o se reduce al subconjunto de la tabla?. Después de explorar las alternativas, el DBMS escoge una de ellas.

5. El DBMS ejecuta la sentencia ejecutando el acceso plano.

Los pasos usados para procesar una sentencia SQL varían por el volumen de acceso a la base de datos que sea requerido y por la cantidad de tiempo que estos tomen. Pasar por el Parser una sentencia SQL no requiere de acceso a la base de datos y puede hacerse muy rápidamente. La optimización, por otra parte, es un proceso intenso de CPU y requiere de acceso al catálogo de sistema. Para una consulta compleja, una consulta multitabla por ejemplo, el optimizador puede explorar millares de maneras diferentes de efectuar la misma consulta. Sin embargo, el costo de ejecutar la consulta ineficientemente es comúnmente tan alto como el tiempo gastado en la optimización que es recobrado al aumentar la velocidad de ejecución de las consultas. Esto es aun más importante si el mismo plano de acceso perfeccionado se puede usar varias veces para desempeñar consultas repetitivas.

2.5. SQL Incrustado

La primera técnica para enviar sentencias SQL al DBMS es incrustarlo en el cuerpo del programa anfitrión. Esto es factible debido a que SQL no usa variables ni sentencias de control de flujo, esto se usa frecuentemente como un sublenguaje de base de datos que puede agregarse a un programa escrito en un lenguaje de programación convencional, tal como C o COBOL. Esta es la idea central del SQL incrustado: poner

sentencias SQL en un programa escritas en el lenguaje de programación anfitrión. Las técnicas siguientes se usan para incrustar sentencias SQL en un lenguaje anfitrión:

- Las sentencias SQL incrustado son procesadas por un precompilador SQL especial. Todas las sentencias SQL comienzan con un introductor y finalizan con un terminador, ambos son las banderas de la sentencia SQL para el precompilador. El introductor y terminador varían con el lenguaje del anfitrión. Por ejemplo, el introductor es "EXEC SQL" en el C y "(SQL)" en MUMPS, y el terminador es un punto y coma (;) en lenguaje C y un paréntesis derecho en MUMPS.
- Las variables del programa de aplicación, llamadas variables host, pueden usarse en sentencias Embedded dondequiera que las constantes así lo permitan. Esto puede usarse para adaptar una sentencia SQL a una situación particular (enviar variables, por ejemplo) y para recibir los resultados de una consulta(y utilizarlos para otros propósitos).
- Las consultas que retornan una simple fila de datos se manejan con una sentencia simple SELECT; esta sentencia especifica ambas partes, la consulta y las variables de anfitrión en las que se devolverán los datos de respuesta.
- Las consultas que devuelven filas múltiples de datos se manejan con cursores. Un cursor guarda la ubicación actual de la fila dentro de un conjunto de resultados. La sentencia DECLARE CURSOR define la consulta, la sentencia OPEN inicia el proceso de la consulta, la sentencia FETCH recupera las filas de datos, y los sentencia CLOSE finaliza el proceso de consulta.
- Mientras un cursor está ubicado para actualización o eliminación, las sentencias pueden usarse para actualizar o borrar la fila actualmente seleccionada por el cursor.

2.5.1. Ejemplo de SQL Incrustado

El código siguiente es un programa simple de SQL incrustado, escrito en C. El programa ilustra muchas, pero no todas, las técnicas de SQL incrustado. El programa desea recuperar datos a partir del número de orden preguntado al usuario, los números de cliente, vendedores, la condición de las órdenes, y muestra la información recobrada sobre la pantalla.

```
main()
{
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    int OrderID;          /* ID Orden (del usuario)          */
    int CustID;           /* ID Customer recuperado          */
    char SalesPerson[10]  /* Nombre del vendedor recuperado  */
    char Status[6]        /* Estado de la orden recuperado   */
  EXEC SQL END DECLARE SECTION;

  /* Coloca un error de procesamiento */
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;

  /* Pregunta al usuario por el número de orden */
  printf ("Ingrese numero de orden: ");
  scanf ("%d", &OrderID);

  /* Ejecuta la consulta SQL */
  EXEC SQL SELECT CustID, SalesPerson, Status
    FROM Orders
    WHERE OrderID = :OrderID
    INTO :CustID, :SalesPerson, :Status;

  /* Despliega los resultados */
  printf ("Número del Customer:  %d\n", CustID);
  printf ("Vendedor : %s\n", SalesPerson);
  printf ("Estado   : %s\n", Status);
  exit();

query_error:
  printf ("SQL error: %ld\n", sqlca->sqlcode);
  exit();

bad_number:
  printf ("Número de orden inválido.\n");
  exit();
```

}

Note lo siguiente acerca de este programa:

- **Variables Host**. Las variables host se declaran en una sección encerrados entre las palabras claves BEGIN DECLARE SECTION y END DECLARE SECTION. Cada nombre de variable host es antepuesto por dos puntos (:) cuando aparece en un SQL incrustado. Los dos puntos permiten al precompilador distinguir entre variables host y objetos de base de datos, tales como tablas y columnas, que tienen el mismo nombre.

- **Tipos de datos**. Los tipos de datos soportados por el DBMS y el lenguaje del host pueden ser bastante diferentes. Esto afecta a las variables host porque ellas juegan un papel dual. Primeramente, los variables host son variables de programa, declaradas y manipuladas por sentencias del lenguaje anfitrión. Por otra parte, estas se usan en sentencias SQL incrustado para recuperar datos de una base de datos. Si el tipo de datos del lenguaje de anfitrión no corresponde al tipo de datos del DBMS, el DBMS automáticamente convierte los datos. Sin embargo, porque cada DBMS tiene su idiosincrasias y sus reglas propias asociadas con el proceso de conversión, los tipos de datos de las variables host deben elegirse cuidadosamente.

- **Manejo de errores.** El DBMS reporta los errores de tiempo de ejecución al programa de aplicación mediante el Area de Comunicaciones SQL, o SQLCA. En el código de ejemplo anterior, la primera sentencia Embedded SQL es INCLUDE SQLCA. Esto le dice al precompilador que se va a para incluir el SQLCA en la estructura en el programa. Este se requiere para procesar los errores devueltos por el DBMS. La sentencia WHENEVER...GOTO le dice al precompilador que genere un código manejador de error que ramifica o enlaza a una etiqueta específica cuando ocurre un error.

- **Simple SELECT**. La sentencia usada para devolver los datos es una simple sentencia SELECT; esto es, devuelve una simple fila de datos. Por lo tanto, el código el ejemplo no declara o usa cursores.

2.5.2. Compilación de un programa con SQL Incrustado

Ya que un programa con SQL incrustado contiene una mezcla de sentencias SQL y sentencias en lenguaje anfitrión este no puede ser compilado directamente en el lenguaje del anfitrión. Entonces debe ser compilado a través de múltiples procesos. Además, los procesos difieren de un producto a otro, pero los pasos son los mismos para todos los productos.

La Figura 2.2 presenta los pasos necesarios para compilar un programa SQL incrustado:

Pasos para compilar un programa con SQL incrustado:

1. El programa con SQL incrustado es entregado al precompilador SQL, una herramienta de programación. El precompilador revisa el programa, encuentra la sentencia SQL incrustada y la procesa. Se requiere un precompilador diferente para cada lenguaje de programación soportado por el DBMS. Los DBMS ofrecen precompiladores para cada uno de los lenguajes de programación y lenguajes ensambladores.

2. El precompilador genera dos archivos de salida. El primero es un archivo fuente, despojado de las sentencias SQL incrustado. En su lugar, el precompilador las sustituye por llamadas a rutinas propiedad del DBMS que proveen el enlace en tiempo de ejecución entre la aplicación y el DBMS. Típicamente, los nombres y las secuencias de llamada de estas rutinas son conocidas únicamente por el precompilador y el DBMS, ellas no son interfase pública para el DBMS. El segundo archivo es una copia de todas las sentencias SQL incrustado usadas en el programa. Este archivo es a veces llamado un Database Request Module o DBRM.

3. El archivo fuente que sale del precompilador es sometido al compilador estándar del lenguaje de programación anfitrión (tal como C o el compilador

COBOL). El compilador procesa el código fuente y produce código objeto en su salida. Note que en este paso no tuvo nada que ver el DBMS ni el SQL.

4. El enlazador acepta los módulos objeto generados por el compilador, los enlaza con varias librerías de rutinas, y produce un programa ejecutable. Las librerías de rutinas enlazadas en el programa ejecutable incluyen las rutinas propiedad del DBMS descritas en el paso 2.

5. El Database Request Module DBRM generado por el precompilador es sometido a una obligatoria utilidad especial. Esta utilidad examina la sentencia SQL, la pasa por un parser, la valida y la optimiza, y entonces produce un acceso plano para cada sentencia. El resultado es un acceso plano combinado para todo el programa, representando una versión ejecutable de las sentencias SQL incrustado. La utilidad obligatoria almacena el plano en la base de datos, usualmente asignándole el nombre del programa de aplicación que lo usará.

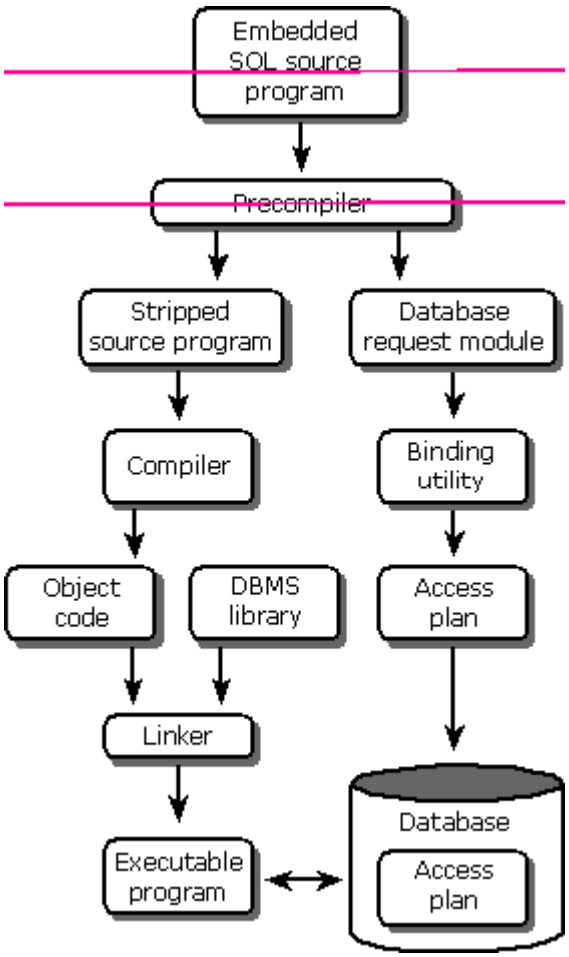


Figura 2.2

Nótese que los pasos usados para compilar un programa con SQL incrustado tienen cercana relación con los pasos descritos en la sección "Procesando una sentencia SQL. Particularmente, note que el precompilador separa la sentencia SQL del código fuente anfitrión, y la utilidad obligatoria revisa y valida la sentencia SQL y crea los accesos planos. En el DBMS, cuando el paso 5 toma lugar en tiempo de compilación, los cuatro primeros pasos de proceso de la sentencia SQL toman lugar en tiempo de compilación, mientras que el último paso (ejecución) toma lugar en tiempo de ejecución. Esto genera el efecto de hacer que las consultas sean muy rápidas para el DBMS.

CAPÍTULO III

ESTÁNDAR Y ARQUITECTURA ODBC

Se ha tomado en cuenta para este capitulo las razones que llevaron a su estandarización, modelo de funcionamiento, librerías API y la arquitectura interna de la norma.

3.1. Noción inicial

Open Data Base Conectivity ODBC es un desarrollo de la compañía Microsoft ® que se ha convertido en estándar, para el acceso a través de él a gran cantidad de tipos de datos. Básicamente, por tanto, cualquier aplicación simple que emplee ODBC puede acceder a las bases de datos soportadas por este estándar. Para que funcione la aplicación sólo necesita comunicarse con un paquete de archivos ODBC, e instantáneamente puede trabajar con cualquier tipo de datos soportados por este paquete.

El campo de aplicación y alcance que la tecnología ODBC proporciona es una interfaz común para acceder a bases de datos SQL heterogéneas. ODBC se basa en SQL como estándar de acceso a los datos. Esto permite a los desarrolladores construir y distribuir aplicaciones Cliente/Servidor no ligadas específicamente a una base de datos determinada. Cada usuario añade luego el driver correspondiente, el cual se encarga de enlazar la aplicación con el DBMS de su elección. Los drivers aíslan a la aplicación de llamadas específicas para una base de datos determinada.

Cabe añadir que, las fuentes de datos ODBC deben estar configuradas correctamente en el ODBC.INI y en el ODBCINST.INI antes de poder ser usada. Si se intenta crear un informe en un sistema y se trata de abrirlo con otro sistema, probablemente no empleen el mismo ODBC Data Source, y por tanto no se establecerá la conexión. Además hay que asegurarse que el SQL usado en el ODBC ha de estar basado en los estándares establecidos por el American National Standards Institute (ANSI) para el lenguaje SQL.

El Paquete de Archivos ODBC se compone de una serie de DLLs y archivos INI que posibilitan su correcto funcionamiento. Se integran en el entorno Windows, y realmente funcionan como pasarela entre las peticiones a las bases de datos y los datos que éstas muestran. Cualquier base de datos que deba usarse bajo ODBC debe estar configurada como fuente de datos ODBC.

3.2. Razones para su creación

Históricamente, las compañías usaron un DBMS único. Todo el acceso de base de datos se hizo mediante aplicaciones escritas para trabajar exclusivamente con ese sistema. Sin embargo, como el uso de computadoras creció y más software de computadora llegó a estar disponible, las compañías comenzaron a adquirir DBMS diferentes. Las razones eran muchas: La gente compró lo que era más barato, lo que era más rápido, lo que ellos ya conocieron, lo que era último sobre el mercado, lo que funcionaba mejor para una aplicación única. Las otras razones eran las reorganizaciones y fusiones, donde departamentos que anteriormente habían usado un DBMS único ahora ya tenían varios.

La situación creció aun más compleja con el advenimiento de computadoras personales. Estas computadoras traían en un sinfín de herramientas para consultar, analizar y desplegar datos, conjuntamente con varias de bases de datos fáciles de usar y baratas. De ahí en adelante, cualquier corporación frecuentemente trabajaba con datos esparcidos a través de una gran cantidad de desktops, servidores y mini computadoras, además, la información estaba almacenada en una variedad de bases de datos incompatibles, y eran accedidas por un número extenso de herramientas diferentes, pocas de las cuales podían acceder a todos los datos.

El desafío final vino con la llegada del concepto Cliente/Servidor, que busca hacer uso más eficiente de los recursos computacionales. Las computadoras personales baratas (los clientes) proveen una interfaz gráfica a los datos y un sinnúmero de herramientas baratas, las tales como hojas de cálculo, diagramadores, constructores de reportes, entre otras. Las computadoras centrales y mini computadoras (los servidores) alojan

el DBMS, donde los clientes pueden usar todo su poder computacional y ubicación central para proveer una rápida y mejor coordinación para acceder a los datos.

Un problema similar encararon los vendedores independientes de software. Ellos construyeron software de base de datos para mini computadoras y computadoras centrales se forzaron a escribir una versión de cada aplicación para cada DBMS o escribir código específico DBMS para cada base de datos que ellos querían acceder. Los vendedores que escribían el software para computadoras personales tuvieron que escribir rutinas de acceso de datos para cada DBMS diferente con los cuales ellos querían trabajar. Esto frecuentemente significó una cantidad enorme de recursos que se gastaron para escribir y mantener las rutinas de acceso a datos antes que en desarrollo de aplicaciones, y además, las aplicaciones se vendieron no por su calidad, sino por si estas podían acceder a los datos de un DBMS determinado.

Los desarrolladores necesitaron una manera especial para acceder a datos en DBMS diferentes. Los clientes y servidores necesitaron una manera para combinar datos desde DBMS diferentes en una misma aplicación, se deseaba encontrar una forma de programar aplicaciones independientes de cualquier DBMS. En resumen, se necesitaba una manera ínter operable de acceso a datos; es decir, conectividad abierta de base de datos.

3.3. El estándar ODBC

La empresa Microsoft desarrolló, basada en los estándares X/Open CLI e ISO, la arquitectura llamada Open Data Base Conectivity (ODBC), de la cual comercializó la primera versión para la plataforma Windows en 1992. La principal característica de ODBC es la de permitir que las aplicaciones cliente desarrolladas bajo una arquitectura Cliente/Servidor puedan tener interoperabilidad con DBMSs relacionales de alto rendimiento (tales como Oracle, Informix, Sybase, DB2, y otros más), sin que los programadores tengan que aprender la sintaxis adicional SQL proporcionada por los fabricantes, sino solamente una, que sea estándar.

La primera especificación de ODBC (versión 1.0) tuvo poca aceptación entre los desarrolladores y los usuarios de aplicaciones clientes de bases de datos. Pero no fue hasta la versión 2.0 (que se incluyó con el paquete integrado Office 4.0 de Microsoft y en algunas aplicaciones de bases de datos propietarias) que se comprobó su funcionalidad. Las herramientas de programación visual que se volvieron muy populares (como Visual Basic, Visual C/C++, Borland C/C++, Delphi, FoxPro para Windows, y más) ahora incluían el soporte para conexión a una fuente de datos mediante ODBC. Igualmente los DBMS ofrecían controladores de conectividad, codificados por ellos o por terceros fabricantes, para sus productos de bases de datos; entre los principales desarrolladores de controladores ODBC están Microsoft, Intersolv, Visigenic, Simba Technologies y Syware.

Microsoft, como ayuda para los desarrolladores de aplicaciones cliente y de controladores, ofrece el SDK (Software Development Kit, Juego de Desarrollo de Software) que contiene las librerías, código fuente de ejemplo y manuales impresos con la información necesaria para la construcción de un controlador para ODBC. No obstante, la versión 2.0 fue diseñada para el ambiente Windows 3.1 y WfW (Windows for Workgroups) 3.11, que son de 16 bits. Con la llegada y el aumento de popularidad del sistema operativo Windows NT y la liberación de Windows 95, se diseñó la versión 2.5 de ODBC, la cual ofrece la misma funcionalidad que la versión anterior, pero ahora incluyendo controladores de 16 y 32 bits. Fue hasta la liberación de la versión 3.0 (en octubre de 1996), completamente de 32 bits, que se incluyeron algunas grandes mejoras como: secuencias de escape para uniones externas (outer joins), ejecución asíncrona, soporte para procedimientos almacenados (stored procedures), cursores de desplazamiento, y otras más, lo que aumentó la funcionalidad y el rendimiento de las aplicaciones. La última versión de ODBC es la 3.5, que agrega algunas nuevas funciones y corrige algunos errores presentados en versiones anteriores.

La arquitectura ODBC está diseñada para la máxima interoperabilidad con diferentes DBMS, permitiendo que una aplicación realice llamadas a funciones estándares con la interfase de ODBC, implementada mediante módulos específicos llamados controladores. El uso de estos controladores aísla a las aplicaciones de las llamadas

específicas al DBMS. Debido a que los controladores son cargados en memoria en tiempo de ejecución, el usuario solo tiene que agregar un nuevo controlador para el DBMS respectivo, sin tener que compilar nuevamente a la aplicación. Cada controlador ODBC específico se encarga del manejo de los protocolos para la conexión, transmisión de datos y desconexión entre el cliente y el servidor, así como del tratamiento de los mensajes de error que pudieran ocurrir. Este protocolo funciona de igual forma que otros controladores, por ejemplo, los controladores de Windows para impresión: si es necesario instalar un controlador, éste se instala mediante el Administrador de Impresoras y entonces cualquier aplicación puede enviar los datos a tal impresora mediante la API para impresión proporcionada por Windows y por lo tanto el programador no necesita modificar el código de la aplicación.

La arquitectura básica de ODBC se muestra en la Figura 3.1. El Administrador de Controladores, que es proporcionado con la instalación de cualquier versión de ODBC, se encarga de aceptar las llamadas a la API de ODBC realizadas por una aplicación cliente para la conexión a una fuente de datos. El controlador específico de dicha fuente de datos es levantado en memoria por el Administrador de Controladores y se encarga de transmitir las sentencias SQL hacia la fuente de datos, transformándolas (si es necesario) a la gramática nativa de la fuente de datos, así como establece la comunicación con el protocolo de red empleado, y maneja el conjunto de resultados y los posibles mensajes de error generados por la fuente de datos, regresándolos a la aplicación para su tratamiento. La fuente de datos puede ser un DBMS relacional, un archivo ISAM o un sistema anfitrión que funciona como capa intermedia para la conexión hacia otros DBMS y puede residir en la misma máquina que la aplicación (acceso local) o en una máquina conectada en una red local (acceso remoto).

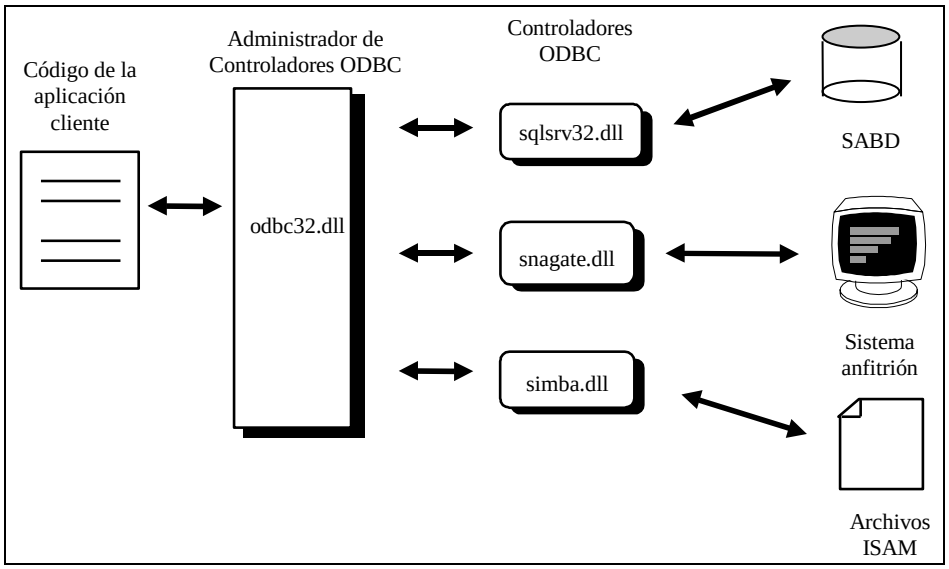


Figura 3.1

En la actualidad, la especificación ODBC es la más popular para la interoperabilidad entre aplicaciones cliente para la plataforma Windows y los DBMS relacionales para cualquier plataforma.

3.4. Drivers

Cuando se utilizan interfaces estándarizadas como ODBC, es necesario hacer uso de un software que convierte la llamada ODBC en una llamada asociada al DBMS API.

Este software se llama ***Database ODBC Driver***.

ODBC define dos tipos de drivers:

- **Single-tier** – O hilera simple, el driver procesa todas las llamadas hechas al ODBC y sentencias SQL . En este caso el driver se encarga de alguna de las funciones de la fuente de datos. Ejemplos: FoxPro .DBF driver, y el Visual Basic .MDB Driver.

- **Múltiple-tier** – O hilera múltiple, el driver procesa las llamadas hechas a ODBC y pasa todas las sentencias SQL a la fuente de datos. Ejemplos: ORACLE, DB2, SQL Server y INFORMIX.

Normalmente, los drivers (sobre todo drivers de tipo Multiple-tier) necesitan otros productos para ser capaces de operar correctamente en un cliente. Esto es porque todas las compañías que desarrollan drivers no sustituyen el estándar DBMS API, ellos simplemente lo complementan. Esto significa que si un software API es requerido para acceder a ciertos DBMS sin ODBC, API resolverá ese requerimiento aunque use ODBC.

3.5. La arquitectura Open DataBase Conectivity (ODBC)

El propósito principal de la especificación ODBC es proporcionar una interfaz universal para el acceso de datos almacenados en una base de datos. Con ODBC los desarrolladores pueden permitir a una aplicación el acceso concurrente, la observación y la modificación de información desde múltiples y diversos DBMS. Básicamente, la interfaz de desarrollo de ODBC consta de:

- Una librería de llamadas a funciones ODBC que permiten a una aplicación conectarse a un DBMS, ejecutar sentencias SQL y obtener resultados.
- La sintaxis estándar del SQL. Para enviar una sentencia SQL, se debe proporcionar una cadena de caracteres conteniendo la sentencia como argumento a una función de ODBC; no es necesario modificar la sentencia para un DBMS específico. Sin embargo, la sintaxis SQL debe ser la estándar soportada por ODBC.
- Un conjunto estandarizado de códigos de error.
- Una forma estándar de conectar y llevar un registro de un DBMS.
- Una representación estandarizada de tipos de datos.

Además del soporte de funcionalidad implementado por el SQL ANSI/89, la especificación ODBC incluye algunas otras capacidades, como:

- Resultados multirenglón en una sola llamada a función.
- Soporte para secuencias de sentencias SQL, ya sea en procedimientos almacenados o como una secuencia de sentencias SQL ejecutadas mediante **SQLExecute** o **SQLExecDirect**.
- Cuenta exacta o aproximada de los renglones de un cursor.
- Operaciones de actualización y eliminación posicionadas por cursor, y actualización y eliminación secuencial por llamada a una función **SQLSetPos**.
- Funciones de catálogo que extraen la información del esquema sin la necesidad de soportar vistas del esquema de información.
- Secuencias de escape para uniones externas (outer joins), funciones escalares, literales de fecha-tiempo (datetime), literales de intervalo y procedimientos almacenados (stored procedures).
- Reporte del nivel de compatibilidad ANSI y del soporte SQL de un controlador.
- Tipos para aplicaciones: fecha-tiempo, intervalo, numérico/decimal, y enteros de 64 bits
- Ejecución asíncrona

Aunque estas características no están en el nivel Esencial de conformidad de la API de ODBC (ver Niveles de Conformidad), es posible implementar la mayoría para que una aplicación que se conecte a un DBMS que no tenga estas características pueda soportarlas.

3.5.1 Modelo de Funcionamiento

La internase común para ODBC es un API de conexión de base de datos, basado en la Interfase de Nivel de Llamada CLI. ODBC permite a las aplicaciones acceder en forma transparente a datos en diferentes DBMS vía sentencias SQL.

Muchos elementos están envueltos en conexiones de ODBC. La Figura 3.2 despliega los componentes de una aplicación que accede a una base de datos vía ODBC.

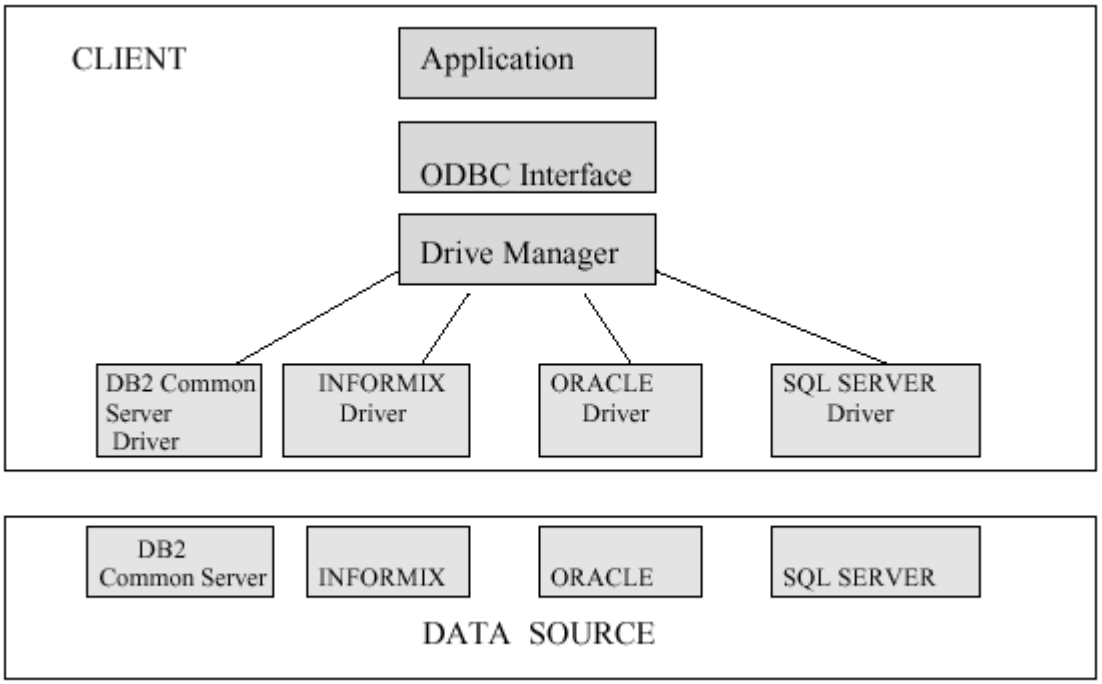


Figura 3.2

- **Aplicación .**- Procesa y llama a las funciones ODBC para someter sentencias SQL y recupera resultados.
- **Driver Manager o Administrador de controladores** - Su trabajo es cargar los drivers y ejecutar las funciones del driver común.
- **Database Driver o Driver ODBC** - Es un DLL que procesa las llamadas a funciones ODBC, somete pedidos SQL en los DBMS y retorna los resultados a la aplicación. Si es necesario, el driver modifica los pedidos de la aplicación para que ellos puedan ser aceptados por los Datos Fuente (Data Source).
- **Data Source u Origen de datos** - Consiste en el DBMS, el sistema operativo, y el servidor de datos de plataforma de red. Procesa los requisitos del driver y envía los resultados.

A continuación se presenta un esquema (Figura 3.3) con los elementos que se agregarán al presentado anteriormente, cuando el ODBC está incorporado.

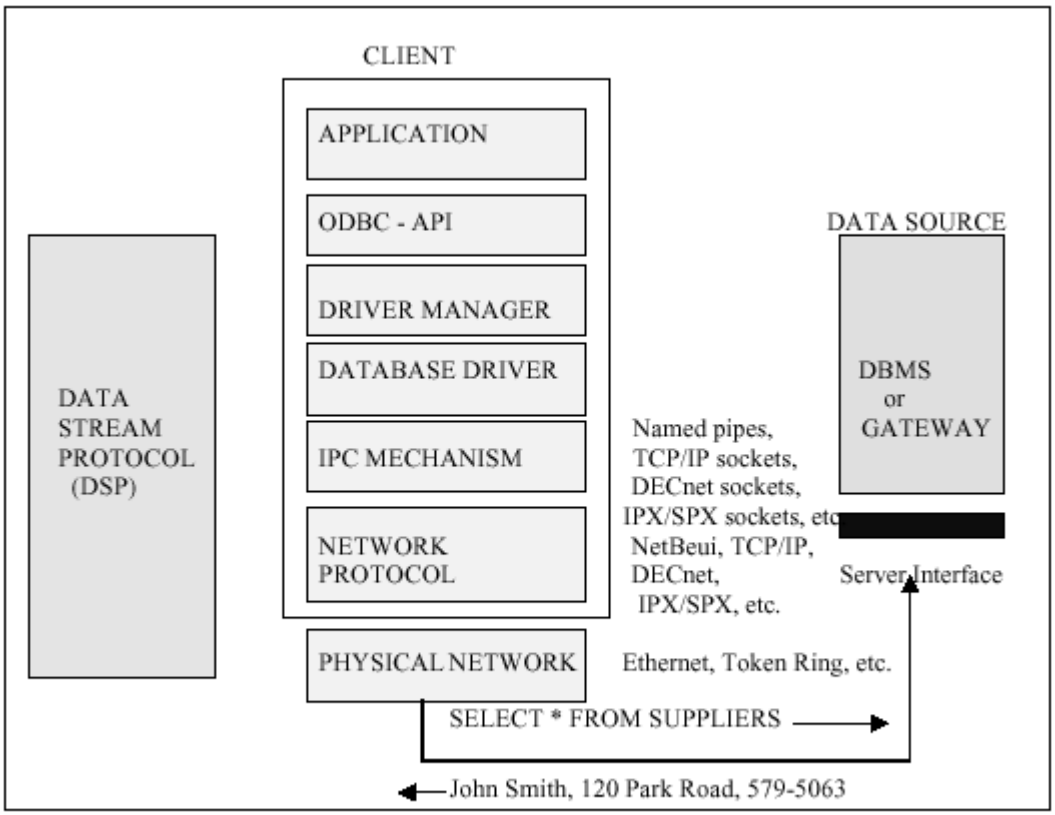


Figura 3.3

La especificación ODBC proporciona las siguientes características para que una aplicación pueda ser ínter operable con un determinado DBMS:

- Establecimiento de una conexión con una fuente de datos, ejecución de sentencias SQL y regreso de un conjunto de resultados.
- Recepción de mensajes de error estándar.
- Establecimiento de una interfaz estándar para el usuario final.
- Uso de un conjunto de tipos de datos estándar definidos por ODBC.
- Uso de una gramática SQL estándar definida por ODBC.

La decisión de implementar la mayoría o todas estas características para la construcción de una aplicación cliente determinará su grado de complejidad; esto es, a una mayor interoperabilidad del sistema, es necesario escribir más código y establecer más funciones de la API.

3.5.2. Arquitectura interna

3.5.2.1. Referencia de la API ODBC

ODBC sigue las normas y especificaciones siguientes que se reparten con el Call-Level-Interface (CLI). ODBC es un súper conjunto de estas normas.

- La especificación X/Open CAE "Data Management: SQL Call-Level-Interface (CLI)"
- ISO / IEC 9075-3:1995 (E) Call–Level-Interface (SQL / CLI)

Como resultado de esta alineación, lo siguiente es verdadero:

- Una aplicación escrita para la especificación X/Open e ISO CLI trabajará con el driver ODBC 3.x o un driver que cumpla estas normas cuando se recompile con archivos de cabecera ODBC 3.x y se enlace con librerías ODBC 3.x, y cuando accede al driver mediante el ODBC 3.x Driver Manager.
- Un driver escrito para la especificación X/Open e ISO CLI trabajará con el driver ODBC 3.x o un driver que cumpla estas normas cuando se recompile con archivos de cabecera ODBC 3.x y se enlace con librerías ODBC 3.x, y cuando accede al driver mediante el ODBC 3.x Driver Manager.

El CORE interface conformance level comprende todos los aspectos en el ISO CLI y todas las características no opcionales en el X/Open CLI. Los aspectos opcionales del X/Open CLI aparece en niveles mas altos de conformidad de interfaz. Porque todos los drivers ODBC 3.x se requieren para soportar las características en el Core interface conformance level, lo siguiente es cierto:

- Un driver ODBC 3.x soportará todas las características usadas por una aplicación que cumpla con los estándares.
- Una aplicación ODBC 3.x que usa solamente las características ISO CLI y las características no opcionales de X/Open CLI trabajará con cualquier driver que cumpla con los estándares.

Además de las especificaciones de interfase de nivel de llamada (Call-Level-Interfase) contenidas en los estándares ISO/IEC y X/Open CLI, ODBC implementa los aspectos siguientes. (Algunas de estas características existieron en versiones de ODBC anteriores a ODBC 3.x.)

- Recuperación de múltiples filas con un llamado único de función.
- Incrustación a un arreglo de parámetros.
- Soporte bookmark incluyendo recuperación por bookmark, bookmarks de longitud variable; y actualización y borrado por operaciones bookmark sobre filas no secuenciales.
- Incrustación Row-wise.
- Los desplazamientos obligatorios.
- Soporte para lotes de sentencias SQL, o en un procedimiento almacenado o como una sucesión de sentencias SQL de declaraciones ejecutadas mediante **SQLExecute** o **SQLExecDirect**.
- Cuenta de filas exacta o aproximada.
- Operaciones de actualización y eliminación posicionadas; y actualización y eliminación en batch llamando a la función **SQLSetPos**.
- Funciones de catálogo que recuperan información desde el esquema de información sin la necesidad que soporte vistas de esquema de información.
- Secuencias de escape para relaciones externas, funciones escalares, literales fecha / hora, intervalos de literales, y procedimientos almacenados.
- Librerías de traducción de Code-Page.
- Reporte del nivel de conformidad ANSI y el soporte SQL del driver.
- Replicación automática sobre demanda del descriptor de parámetros de implementación.
- Diagnósticos mejorados y arreglos del estado de filas y parámetros.
- Fecha / hora, intervalos, numérico/decimal, enteros de 64 bits tipos de buffer de aplicación.
- Ejecución asíncrona.
- Soporte de procedimientos almacenados, incluyendo secuencias de escape, envío al exterior de parámetros y funciones de catálogo.

Conexiones mejoradas incluyendo el soporte para atributos de conexión y atributos de examen.

La API de ODBC para la conexión y el tratamiento de sentencias consta de las siguientes funciones (entre corchetes se incluye la versión de ODBC en que fue introducida la función y en algunos casos se señalan los cambios con respecto a otras funciones) :

Tabla 3.1.

Tarea	Nombre de la función	Propósito
Conexión a una fuente de datos	SQLAllocHandle [3.0]	Obtiene un handler para el ambiente, la conexión, las sentencias y los descriptores.
	SQLAllocEnv [1.0]	Obtiene un handler para el ambiente.
	SQLAllocConnect [1.0]	Obtiene un handler para la conexión.
	SQLAllocStmt [1.0]	Obtiene un handler para la sentencia.
	SQLConnect [1.0]	Hace la conexión a una fuente de datos específica, con el ID del usuario y su contraseña.
	SQLDriverConnect [1.0]	Hace la conexión a una fuente de datos específica mediante una cadena de conexión o solicita que el Administrador de Controladores muestre un cuadro de diálogo al usuario.
	SQLBrowseConnect [1.0]	Regresa los niveles sucesivos de atributos de conexión y valores de atributos válidos. Cuando un valor ha sido especificado para cada atributo de conexión, se conecta a la fuente de datos.
Obtener información acerca de controladores y fuentes de datos	SQLDataSources [1.0]	Regresa la lista de fuentes de datos disponibles.
	SQLDrivers [2.0]	Regresa la lista de controladores instalados y sus atributos.
	SQLGetInfo [1.0]	Regresa información acerca de un controlador específico y de una fuente de datos.
	SQLGetFunctions [1.0]	Regresa las funciones soportadas por el controlador.
	SQLGetTypeInfo [1.0]	Regresa información acerca de tipos de datos soportados.
Configurar y obtener atributos del controlador	SQLSetConnectAttr [3.0] (reemplaza a SQLSetConnectOption [1.0])	Asigna los atributos de conexión, tales como el modo de acceso, ejecución asíncrona, autocommit, tiempos de conexión, aislamiento de transacciones, entre otros.
	SQLGetConnectAttr [3.0] (reemplaza a SQLGetConnectOption [1.0])	Obtiene los valores de los atributo de conexión.
	SQLSetEnvAttr [3.0]	Asigna los atributos de ambiente, tales como el modo de escrutinio para la conexión, la versión de ODBC soportada por el controlador, entre otras.

	SQLGetEnvAttr [3.0]	Obtiene los valores de los atributos de ambiente.
	SQLSetStmtAttr [3.0] (reemplaza a SQLSetStmtOption [1.0])	Asigna los atributos de sentencia, tales como el tipo de ligado, el tamaño y el estado de procesamiento de parámetros, entre otros.
	SQLGetStmtAttr [3.0] (reemplaza a SQLGetStmtOption [1.0] y a SQLParamOptions [1.0])	Obtiene los valores de los atributos de sentencia.
Configurar y obtener campos descriptores	SQLGetDescField [3.0]	Regresa el valor actual de un campo de un registro descriptor, que consta de los atributos: nombre, tipo, intervalo fecha/tiempo, longitud del octeto, precisión, escala y nulabilidad de una columna.
	SQLGetDescRec [3.0]	Regresa los valores de los campos de un registro descriptor.
	SQLSetDescField [3.0]	Asigna un valor a un campo descriptor.
	SQLSetDescRec [3.0]	Asigna valores a los campos de un registro descriptor.
Prepara sentencias	SQLPrepare [1.0]	Prepara una sentencia SQL para ejecución posterior.
	SQLBindParameter [2.0]	Asigna almacenamiento de parámetros a variables huésped en una sentencia SQL.
	SQLGetCursorName [1.0]	Regresa el nombre del cursor asociado con un handler de sentencia.
	SQLSetCursorName [1.0]	Especifica un nombre de cursor.
	SQLSetScrollOptions [1.0]	Asigna opciones que controlan el comportamiento del cursor.
Envía sentencias	SQLExecute [1.0]	Ejecuta una sentencia preparada.
	SQLExecDirect [1.0]	Ejecuta una sentencia en forma directa.
	SQLNativeSql [1.0]	Regresa el texto de una sentencia SQL sin ser traducida por el controlador.
	SQLDescribeParam [1.0]	Regresa la descripción de un parámetro específico en una sentencia.
	SQLNumParams [1.0]	Regresa el número de parámetros en una sentencia.
	SQLParamData [1.0]	Usado en conjunto con SQLPutData proporciona datos de parámetros en tiempo de ejecución (usado para valores largos de datos).
	SQLPutData [1.0]	Envía parte o todos los valores de datos para un parámetro (usado para valores largos de datos).
Regresa resultados e información acerca de los resultados	SQLRowCount [1.0]	Regresa el número de renglones afectados por un requerimiento de inserción, actualización o eliminación.
	SQLNumResultCols [1.0]	Regresa el número de columnas del conjunto de resultados.
	SQLDescribeCol [1.0]	Describe una columna del conjunto de resultados.
	SQLColAttribute [3.0] (renombrada de SQLColAttributes [1.0])	Describe atributos de una columna del conjunto de resultados.

	SQLBindCol [1.0]	Asigna almacenamiento para una columna de resultados y especifica el tipo de dato.
	SQLFetch [1.0]	Regresa múltiples renglones de resultado.
	SQLFetchScroll [3.0] (reemplaza a SQLExtendedFetch [1.0])	Regresa renglones de resultado deslizables.
	SQLGetData [1.0]	Regresa parte o todas las columnas de un renglón de un conjunto de resultados (usado para valores largos de datos).
	SQLSetPos [1.0]	Posiciona un cursor dentro de un bloque de datos obtenidos, y permite a una aplicación actualizar los datos en el conjunto de renglones, o actualizar o borrar datos del conjunto de resultados.
	SQLBulkOperations [3.0]	Ejecuta inserciones y operaciones voluminosas, incluyendo actualización, borrado y traída por marcas (bookmarks).
	SQLMoreResults [1.0]	Determina cuando hay más conjuntos de resultados disponibles y, si puede, inicializa el procesamiento para el siguiente conjunto de resultados.
	SQLGetDiagField [3.0]	Regresa información de diagnóstico adicional (un campo de la estructura de diagnóstico de datos).
	SQLGetDiagRec [3.0]	Regresa información de diagnóstico adicional (múltiples campos de la estructura de diagnóstico de datos).
	SQLError [1.0]	Regresa información adicional sobre condiciones de error o de estado actual.
Obtiene información sobre las tablas del sistema de la fuente de datos (funciones de catálogo)	SQLColumnPrivileges [1.0]	Regresa una lista de columnas y privilegios asociados para una o más tablas.
	SQLColumns [1.0]	Regresa la lista de nombres de columnas en una tabla específica.
	SQLForeignKeys [1.0]	Regresa una lista de nombres de columnas que asignan llaves foráneas, si éstas existen, para una tabla específica.
	SQLPrimaryKeys [1.0]	Regresa una lista de nombres de columnas que asignan la llave primaria para una tabla.
	SQLProcedureColumns [1.0]	Regresa la lista de parámetros de entrada y salida, así como las columnas que colocan el conjunto de resultados para los procedimientos específicos.
	SQLProcedures [1.0]	Regresa la lista de nombres de procedimientos almacenados en una fuente de datos específica.
	SQLSpecialColumns [1.0]	Regresa información acerca del conjunto óptimo de columnas que identifican un renglón en una tabla específica, o las columnas que son automáticamente actualizadas cuando cualquier valor en el renglón es actualizado por una transacción.

	SQLStatistics [1.0]	Regresa estadísticas acerca de una tabla y la lista de los índices asociados con la tabla.
	SQLTablePrivileges [1.0]	Regresa la lista de tablas y privilegios asociados con cada tabla.
	SQLTables [1.0]	Regresa la lista de los nombres de las tablas almacenadas en una fuente de datos específica.
Término de una sentencia	SQLFreeStmt [1.0]	Termina el procesamiento de sentencias, descarta los resultados pendientes, y opcionalmente libera todos los recursos asociados al handler de sentencia.
	SQLCloseCursor [3.0]	Cierra un cursor que ha sido abierto en un handler de sentencia.
	SQLCancel [1.0]	Cancela una sentencia SQL.
	SQLEndTran [3.0] (reemplaza a SQLTransact [1.0])	Compromete o aborta una transacción.
Término de una conexión	SQLDisconnect [1.0]	Cierra una conexión.
	SQLFreeHandle [3.0]	Libera los recursos asociados a un handler de ambiente, de conexión, de sentencia o de descriptores.
	SQLFreeConnect [1.0]	Libera los recursos asociados a un handler de conexión.
	SQLFreeEnv [1.0]	Libera los recursos asociados a un handler de ambiente.

Las siguientes funciones no pertenecen propiamente a la API de ODBC, ni están definidas en el estándar ISO SQL/CLI, sino que proporcionan una ayuda al desarrollador para la instalación y configuración de los controladores, las fuentes de datos y los traductores en el ambiente Windows, por lo que no es estrictamente necesario que sean implementadas. La mayoría de estas funciones ya están implementadas y solo necesitan ser llamadas con los parámetros correspondientes.

Tabla 3.2.

Tarea	Nombre de la función	Propósito
Configura fuentes de datos y traductores	ConfigDriver [2.5]	Instala o desinstala un controlador
	ConfigDSN [1.0]	Agrega, modifica o elimina una fuente de datos.
	ConfigTranslator [2.0]	Regresa una opción de traducción por omisión.

Tabla 3.3

Tarea	Nombre de la función	Propósito
Instalación de ODBC	SQLConfigDriver [2.5]	Carga una dll de configuración específica del controlador.
	SQLGetInstalledDrivers [1.0]	Regresa una lista de controladores instalados.
	SQLInstallDriverEx [3.0]	Agrega un controlador al sistema.
	SQLInstallDriverManager [1.0]	Regresa el directorio destino para el Administrador de Controladores
	SQLInstallerError [3.0]	Regresa información sobre errores o sobre el estado para las funciones de instalación.
	SQLInstallTranslatorEx [3.0]	Agrega un traductor al sistema.
	SQLPostInstallerError [3.0]	Permite reportar errores a una librería de instalación de un controlador o traductor.
	SQLRemoveDriver [3.0]	Remueve un controlador del sistema.
	SQLRemoveDriverManager [3.0]	Remueve los componentes principales de ODBC del sistema.
	SQLRemoveTranslator [3.0]	Remueve un traductor del sistema.
Configuración de las fuentes de datos	SQLConfigDataSource [1.0]	Llama a la dll de configuración específica del controlador.
	SQLCreateDataSource [2.0]	Muestra un cuadro de diálogo para añadir fuentes de datos.
	SQLGetConfigMode [3.0]	Regresa el modo de configuración que indica dónde está la lista de entradas de valores DSN de odbc.ini en el sistema.
	SQLGetPrivateProfileString [2.0]	Escribe un valor al sistema.
	SQLGetTranslator [2.0]	Muestra un cuadro de diálogo para seleccionar un traductor.
	SQLManageDataSources [2.0]	Muestra un cuadro de diálogo para configurar las fuentes de datos y controladores.
	SQLReadFileDSN [3.0]	Lee información de un archivo DSN.
	SQLRemoveDSNFromIni [1.0]	Remueve una fuente de datos.
	SQLSetConfigMode [3.0]	Asigna el modo de configuración que indica dónde está la lista de entradas de valores DSN de odbc.ini en el sistema.
	SQLValidDSN [2.0]	Verifica la longitud y validez del nombre de una fuente de datos.
	SQLWriteDSNToIni [1.0]	Agrega una fuente de datos.
	SQLWriteFileDSN [3.0]	Escribe información al archivo DSN.
	SQLWritePrivateProfileString [2.0]	Obtiene un valor del sistema.

Tabla 3.4

Tarea	Nombre de la función	Propósito
Traduce datos	SQLDataSourceToDriver	Traduce todos los flujos de datos de la fuente de datos al

		controlador.
	SQLDriverToDataSource	Traduce todos los flujos de datos del controlador a la fuente de datos.

3.5.2.2 Niveles de Conformidad

Cuando se especifica un ODBC, se establecen niveles de conformidad a un nivel API y SQL.

Las funciones ODBC se componen de funciones CORE (definidas como X/Open, y como una especificación de Interfase de nivel de llamada del SQL Access Group), y de funciones extendidas de niveles (nivel 1 y nivel 2), así que ODBC puede extender esta especificación.

Cada aplicación determina las capacidades que implementará usando ODBC. La interfaz de ODBC proporciona dos clases de conformidad que las aplicaciones pueden usar: la conformidad de la API de funciones ODBC y la conformidad de la gramática SQL, con el fin de que una aplicación establezca el grado de interoperabilidad que desea obtener.

a. Conformidad a nivel de API

Existen tres niveles de conformidad: el nivel Esencial debe ser implementado por cualquier controlador que soporte ODBC, ya que satisface los requerimientos mínimos de una aplicación genérica para interoperabilidad definidas por la especificación ISO SQL/CLI, y los niveles 1 y 2 que cubren las características soportadas por el nivel Esencial, más algunas otras, tal como control de transacciones, cursores deslizables, obtención de llaves primarias, uso de procedimientos almacenados, uso de marcas (bookmarks) y del diccionario de datos, ejecución asíncrona, entre otras. Los niveles de conformidad de la API proporcionan la siguiente funcionalidad:

API Esencial

- Asigna y libera handlers de ambiente, conexión y sentencia.

- Realiza la conexión a las fuentes de datos. Usa múltiples sentencias en una conexión.
- Prepara y ejecuta sentencias SQL. Ejecuta sentencias SQL de forma inmediata.
- Asigna almacenamiento para parámetros en una sentencia SQL y para columnas de resultados.
- Obtiene datos de un conjunto de resultados. Obtiene información acerca del conjunto de resultados.
- Compromete o aborta las transacciones.
- Obtiene información de errores.

API de Nivel 1

- Cubre la funcionalidad de la API Esencial.
- Hace la conexión a las fuentes de datos con cuadros de diálogos específicos del controlador.
- Establece y consulta valores de las opciones de conexiones y sentencias.
- Envía parte o todo el valor de un parámetro (usado para datos largos).
- Obtiene parte o todo el valor de una columna de resultados (usado para datos largos).
- Obtiene información del catálogo (columnas, columnas especiales, estadísticas y tablas).
- Obtiene información acerca de las capacidades del controlador y de la fuente de datos, tales como los tipos de datos soportados, las funciones escalares y las funciones ODBC.

API de Nivel 2

- Cubre la funcionalidad Esencial y de Nivel 1.
- Indaga sobre la información de la conexión y sobre la lista de fuentes de datos disponibles.
- Envía arreglos con los valores de parámetros. Obtiene arreglos con los valores de columnas de resultados.
- Obtiene el número de parámetros y describe parámetros individuales.

- Usa cursores deslizables.
- Obtiene la forma nativa de una sentencia SQL.
- Obtiene información del catálogo (privilegios, llaves y procedimientos).
- Llama a una dll de traducción.

Las funciones pertenecientes a los niveles de conformidad de la API de ODBC son mostradas en la tabla 3.4:

Tabla 3.5

Nivel Esencial		**Nivel 1**	**Nivel 2**
SQLAllocConnect	SQLFreeHandle[#]	SQLBulkOperations	SQLBrowseConnect
SQLAllocEnv	SQLFreeStmt	SQLColumns	SQLColumnPrivileges
SQLAllocHandle[#]	SQLGetConnectAttr[+]	SQLDriverConnect	SQLDataSources
SQLAllocStmt	SQLGetCursorName	SQLGetConnectOption	SQLDescribeParam
SQLBindCol	SQLGetDescField[+]	SQLGetData	SQLExtendedFetch
SQLBindParameter[1]	SQLGetDescRec[+]	SQLGetFunctions	SQLForeignKeys
SQLCancel[1]	SQLGetDiagField[+]	SQLGetInfo	SQLNativeSql
SQLCloseCursor[+]	SQLGetDiagRec[+]	SQLGetStmtOption	SQLNumParams
SQLColAttribute[1]	SQLGetEnvAttr[+]	SQLGetTypeInfo	SQLParamOptions
SQLConnect	SQLGetStmtAttr[+]	SQLMoreResults	SQLSetScrollOptions
SQLCopyDesc[+]	SQLNumResultCols	SQLParamData	SQLTablePrivileges
SQLDescribeCol[1]	SQLPrepare	SQLPrimaryKeys	
SQLDisconnect	SQLRowCount	SQLProcedureColumns	
SQLDrivers	SQLSetConnectAttr[2+]	SQLProcedures	
SQLEndTran[1+]	SQLSetCursorName	SQLPutData	
SQLExecDirect	SQLSetDescField[1+]	SQLSetPos[1]	
SQLExecute	SQLSetDescRec[+]	SQLSpecialColumns[1]	
SQLFetch	SQLSetEnvAttr[2+]	SQLStatistics	
SQLFetchScroll[1+]	SQLSetStmtAttr[2+]	SQLTables	
SQLFreeConnect	SQLTransact		
SQLFreeEnv			

[1] Algunas características significativas de esta función están disponibles solo en los niveles más altos de conformidad.

[2] La configuración de ciertos atributos a valores que no sean por omisión depende del nivel de conformidad.

[+] Estas funciones, que pertenecían al nivel 1, han sido incluidas en el nivel Esencial a partir de la versión 3.x de ODBC. Algunas son nuevas para la versión 3.x y otras son actualizaciones a la versión 2.x.

[#] SQLAllocHandle encapsula a SQLAllocConnect, SQLAllocEnv y SQLAllocStmt, mientras que SQLFreeHandle encapsula a SQLFreeConnect, SQLFreeEnv y parte de SQLFreeStmt.

b. Conformidad a nivel de gramática SQL

Para la conformidad a nivel de la gramática SQL existen tres niveles, que corresponden a la especificación del X/Open y SAG CAE de 1992. A continuación se mencionan los elementos que deben estar presentes en cada nivel de conformidad de dicha gramática:

Gramática SQL Mínima

- Lenguaje de Definición de Datos: CREATE TABLE y DROP TABLE.
- Lenguaje de Manipulación de Datos: SELECT simple, INSERT, UPDATE SEARCHED y DELETE SEARCHED.
- Expresiones: simples (como A > B + C).
- Tipos de Datos: CHAR, VARCHAR o LONG VARCHAR.

Gramática SQL Esencial

- Gramática SQL y tipos de datos de la gramática Mínima.
- Lenguaje de Manipulación de Datos: ALTER TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, GRANT y REVOKE.
- Lenguaje de Manipulación de Datos: SELECT completo.
- Expresiones: subconsultas, funciones de conjuntos como SUM y MIN.
- Tipos de Datos: DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, FLOAT, DOUBLE PRECISION.

Gramática SQL Extendida

- Gramática SQL y tipos de datos de la gramática Mínima y Esencial.
- Lenguaje de Manipulación de Datos: uniones externas, UPDATE posicionado, DELETE posicionado, SELECT FOR UPDATE y uniones.
- Expresiones: funciones escalares como SUBSTRING y ABS, y literales de fechas, tiempo y estampilla de tiempo (timestamp).
- Tipos de Datos: BIT, TINYINT, BIGINT, BINARY, VARBINARY, LONG VARBINARY, DATE, TIME, TIMESTAMP.
- Sentencias SQL en lotes.
- Llamadas a procedimientos.

Las aplicaciones que usen ODBC requieren de una gramática mínima SQL que debe ser cubierta por los controladores. La sintaxis descrita a continuación es un subconjunto de la sintaxis del SQL/92.

declaración-crear-tabla ::=

CREATE TABLE nombre-tabla-base

(identificador-columna tipo-dato [,identificador-columna tipo-dato]…)

declaracion-eliminar-buscado ::=

DELETE FROM nombre-tabla [WHERE condicion-busqueda]

declaración-eliminación-tabla ::=

DROP TABLE nombre-tabla-base

declaración-insertar ::=

INSERT INTO nombre-tabla [(identificador-columna [, identificador-columna]...)]

VALUES (valor-insertar[, valor-insertar]...)

declaración-selección ::=

SELECT [ALL | DISTINCT] lista-selección

FROM lista-referencia-tabla

[WHERE condición-búsqueda]

[cláusula-ordenar-por]

declaración ::= declaración-crear-tabla

| declaración-eliminar-buscado

| declaración-eliminación-tabla

| declaración-insertar

| declaración-selección

| declaración-actualización-buscado

declaración-actualización-buscado

UPDATE nombre-tabla

SET identificador-columna = {expresión | NULL }

[, identificador-columna = {expresión | NULL}]...

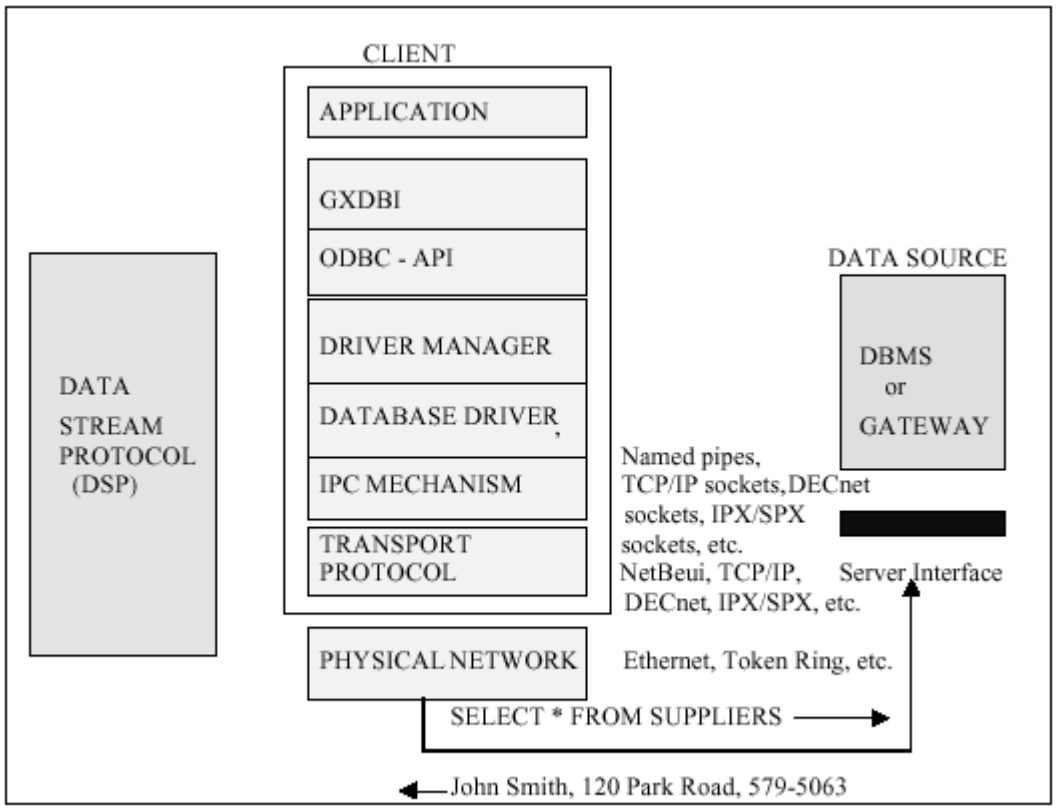
[WHERE condición-búsqueda]

Las herramientas CASE son capaces de realizar generaciones de código estándar, porque agregan un nivel más como muestra la Figura 3.3. Este nivel consiste en un conjunto de funciones denominado, Database Interfase (GXDBI).

Las funciones de GXDBI son invocadas por los programas generados (aplicación) y actúan como la interfase ODBC (vea Figura 3.4).

Las funciones de GXDBI se distribuyen como DLLs.

Figura 3.4



CAPÍTULO IV

METODOLOGÍA PARA UTILIZACIÓN DE DRIVERS ODBC INCLUIDOS EN LOS DBMS

ODBC es un intermediario entre bases de datos y aplicaciones, cuya tarea es sostener una conversación de preguntas y respuestas entre dos entidades que no hablan el mismo idioma y que gestionan sus recursos de forma diferente. Este intermediario se proporciona con la mayoría de DBMS comerciales.

4.1. Componentes de conexión ODBC

Se identificar tiene cuatro componentes de una conexión ODBC, tal como se muestra en la Figura 4.1:

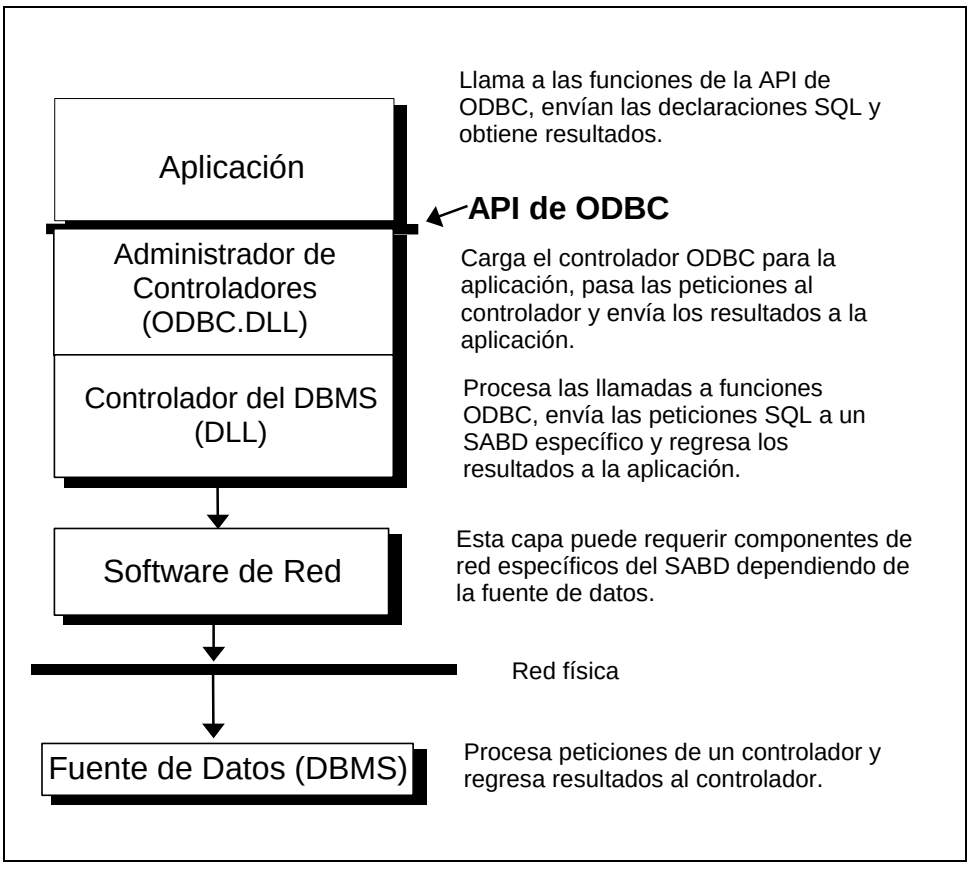


Figura 4.1

Aplicación. Ejecuta el procesamiento y las llamadas a funciones ODBC para enviar sentencias SQL y obtener resultados. Puede estar construida en cualquier lenguaje de programación. Por ejemplo: un programa de rol de pagos, de control de ventas, de inventario, etc.

Administrador de Controladores. Carga los controladores que se utilizarán en la aplicación y realiza algunas comprobaciones de errores. En la plataforma Windows, este se instala junto con el sistema operativo. Consiste en la librería ODBC.DLL .

Controlador. Procesa las llamadas a funciones ODBC, envía peticiones SQL a una fuente específica de datos y regresa resultados a la aplicación. Si es necesario, el controlador transforma la petición de la aplicación para que cumpla con la sintaxis SQL soportada por el DBMS asociado. Cuando se instala un nuevo DBMS este automáticamente instala su driver ODBC correspondiente, el driver consiste en una librería DLL. En caso de no instalarse junto con el DBMS, ciertas empresas se especializan en crear drivers ODBC para DBMS relacionales, no relacionales y otro tipo de fuentes de datos, entre los fabricantes más importantes de drivers ODBC tenemos a **Intersolv**, **Simba Technologies**, **Syware**, **Microsoft**, **Visigenic**, etc.

Fuente de datos. Consiste en los datos que el usuario desea acceder en el sistema operativo asociado y la plataforma de red (si hay alguna) usada para acceder el DBMS. Básicamente es un almacenamiento de datos.

4.2. ODBC Data Source Administrator

Es la interfaz gráfica que permite administrar los drivers ODBC, los orígenes de datos DSN y en general, el seguimiento de una conexión. Esta interfaz se muestra en la Figura 4.2.

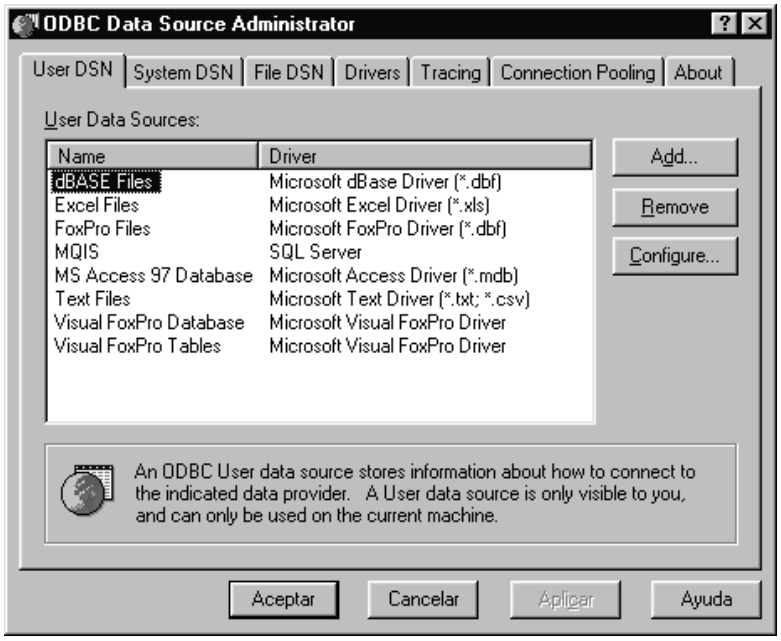


Figura 4.2

Consta de las partes detalladas en la tabla 4.1 :

Tabla 4.1

Componentes de la Ventana de diálogo ODBC Data Source Administrator

CONTROL	DESCRIPCION
User Data Sources	Un DSN de usuario almacena información acerca de cómo conectarse a un proveedor de datos indicado. Un DSN de usuario es visible solo para ese usuario y puede ser utilizado solamente en esa computadora. Enumeran todos los DSN del usuario, incluyendo el nombre y driver asociado de cada DSN. Haciendo doble click en un DSN de usuario se muestra la caja de diálogo de configuración de fuente de datos del driver especificado.
Add	Haga Click para agregar una nueva fuente de datos de usuario, y aparece la caja de diálogo **Create New Data Source**, escoja el driver que desea para una fuente de datos de usuario y, haga click en **Finish** y aparecerá c
Remove	Elimina un DSN de usuario.
Configure	Cambia la configuración del DSN de usuario seleccionado aquí se desplegará la caja de diálogo de configuración de fuente de datos del driver específicado.
OK	Cierra la caja de dialogo **ODBC Data Source Administrator**. No se hace click sobre **OK** aceptar los cambios en la lista de **User Data Sources**. Los cambios de la lista son aceptados una vez que se haya hecho click sobre **OK** en la caja de diálogo de configuración de fuente de datos del driver especificado.
Cancel	Cierra la caja de diálogo ODBC Data Source Administrator y deshace los cambos realizados.
Help	Despliega la ayuda en pantalla
System Data Sources	Un DSN de sistema es visible a todos los usuarios de esa computadora incluyendo los servicios NT. Enumeran todos los DSN del sistema, incluyendo el nombre y driver asociado de cada DSN. Haciendo doble click en un DSN de sistema se muestra la caja de diálogo de configuración de fuente de datos del driver especificado.
Add	Haga Click para agregar una nueva fuente de datos de sistema, y aparece la caja de diálogo **Create New Data Source**, escoja el driver que desea para una fuente de datos de sistema y, haga click en **Finish** y aparecerá la caja de diálogo de configuración de fuente de datos del driver especificado.
Remove	Elimina un DSN de sistema.

Configure	Cambia la configuración del DSN de sistema seleccionado aquí se desplegará la caja de diálogo de configuración de fuente de datos del driver especificado.
OK	Cierra la caja de dialogo **ODBC Data Source Administrator**. No se hace click sobre **OK** aceptar los cambios en la lista de **System Data Sources**. Los cambios de la lista son aceptados una vez que se haya hecho click sobre **OK** en la caja de diálogo de configuración de fuente de datos del driver especificado.
Cancel	Cierra la caja de diálogo **ODBC Data Source Administrator** y deshace los cambos realizados.
Help	Despliega la ayuda en pantalla
File Data Sources	Un DSN de archivo es posible compartirlo entre los usuarios que tienen el mismo driver, se crea un archivo con la extensión DSN y se lo reparte entre los usuarios. Enumeran todos los DSN del archivo, incluyendo el nombre y driver asociado de cada DSN. Haciendo doble click en un DSN de archivo se muestra la caja de diálogo de configuración de fuente de datos del driver especificado.
Add	Haga Click para agregar una nueva fuente de datos de archivo, y aparece la caja de diálogo **Create New Data Source**, escoja el driver que desea para una fuente de datos de archivo y, haga click en **Finish** y aparecerá la caja de diálogo de configuración de fuente de datos del driver especificado.
Remove	Elimina un DSN de archivo.
Configure	Cambia la configuración del DSN de archivo seleccionado aquí se desplegará la caja de diálogo de configuración de fuente de datos del driver especificado.
OK	Cierra la caja de dialogo **ODBC Data Source Administrator**. No se hace click sobre **OK** aceptar los cambios en la lista de **System Data Sources**. Los cambios de la lista son aceptados una vez que se haya hecho click sobre **OK** en la caja de diálogo de configuración de fuente de datos del driver especificado.
Cancel	Cierra la caja de diálogo **ODBC Data Source Administrator** y deshace los cambos realizados.
Help	Despliega la ayuda en pantalla
Drivers	Es de carácter informativo, despliega información acerca de los drivers ODBC instalados sobre la computadora actual.
ODBC drivers	Nombre, compañía, versión, fecha de creación del archivo de cada driver.
OK	Cierra el cuadro de diálogo **ODBC Data Source Administrator**.
Cancel	Cierra el cuadro de diálogo **ODBC Data Source Administrator**.
Help	Despliega la ventana de ayuda
Tracing Tab	Permite crear un log de las llamadas a los drivers ODBC para soporte del programador o para ayudar en la depuración de aplicaciones. Especifica como se realizara el seguimiento de las llamadas a funciones ODBC por parte del **ODBC Driver Manager**, este puede realizar el seguimiento continuamente o solamente una por conexión, puede realizar seguimiento dinámico o puede permitir que el seguimiento la realice una DLL de usuario.
Start Tracing Now/Stop Tracing Now (toggle)	Habilita el seguimiento dinámico continuo. Cuando una conexión se realizado o no, mientras el cuadro de diálogo **ODBC Data Source Administrator** está desplegado o hasta que se haga clic en **Stop Tracing Now.**
Start Visual Studio Analyzer/Stop Visual Studio Analyzer (toggle)	Habilita el **Visual Studio Analyzer** que permanece habilitado hasta que se haga click en **Stop Visual Studio Analyzer**. Esta es una herramienta que se puede utilizar para depurar y analizar la aplicación distribuida.
Log file Path	Despliega la ruta y el nombre de archivo donde la información de seguimiento será almacenada. Utilice la ruta y el nombre de archivo por defecto (\sql.log) o alternativamente ingrese uno nuevo haciendo clic en el botón **Browse.**
Browse	Permite seleccionar la ruta y el nombre de archivo para el archivo de log explorando los directorios de la computadora.
Custom Trace DLL	Permite seleccionar un archivo de DLL diferente al Odbctrac.dll para realizar el seguimiento. Ingrese la ruta y el nombre de archivo de la Dll de usuario, o haga click sobre **Select DLL** para explorar el directorio. El archivo Odbctrac.dll que es suministrado con el **MDAC SDK** puede ser reemplazado por una DLL de usuario.
Select DLL	Permite al usuario explorar la estructura de directorios para encontrar una DLL de usuario para realizar el seguimiento de las llamadas a funciones ODBC. La ruta y el nombre del archivo de la DLL escogida aparecerán en el cuadro de diálogo **Custom Trace DLL**.
OK	Acepta los cambios para las configuraciones del **Log file Pat.** y **Custom Trace DLL** y cierra la caja de diálogo **ODBC Data Source Administrator**.
Cancel	Cierra la caja de diálogo **ODBC Data Source Administrator** sin aceptar los cambios o nueva configuración.
Apply	Acepta los cambios a la configuración de seguimiento y deja abierto el cuadro de diálogo **ODBC Data Source Administrator**.
Help	Despliega la ayuda en línea

Tracing Tab	Specifies how the ODBC Driver Manager traces calls to ODBC functions. The Driver Manager can trace calls continuously or for one connection only, can perform tracing dynamically, or can allow tracing to be performed by a custom trace .dll.
Start Tracing Now/Stop Tracing Now (toggle)	Enables continuous dynamic tracing, whether or not a connection has been made, as long as the **ODBC Data Source Administrator** dialog box is displayed or until you click **Stop Tracing Now**.
Start Visual Studio Analyzer/Stop Visual Studio Analyzer (toggle)	Enables Visual Studio Analyzer, which remains enabled until you click **Stop Visual Studio Analyzer**. Visual Studio Analyzer is a tool you can use for debugging and analyzing your distributed application. For more information about Visual Studio Analyzer, see the documentation in the MSDN Library.
Log file Path	Despliega la ruta y el nombre de archivo donde la información de seguimiento será almacenada. Displays the path and file name where the tracing information will be stored. Use the default path and file name (\sql.log), or alternatively, specify a new file either by entering a new path and file name or by clicking **Browse** and selecting a directory and file.
Browse	Permite seleccionar una ruta y nombre de usuario para el archivo de log, explorando la estructura de directorios de la computadora.
Custom Trace DLL	Permite seleccionar u archivo DLL diferente a Odbctrac.dll para realizar el seguimiento. Ingrese la ruta y el nombre de archivo de la DLL de usuario o haga clic en **Select DLL** para explorar los directorios. El archivo Odbctrac.dll que ha sido proporcionado con el MDAC SDK puede ser reemplazado por la DLL de usuario que ha escogido.
Select DLL	Permite al usuario explorar la estructura de directorios para encontrar una DLL de seguimiento de usuario. La ruta y el nombre de archivo escogidos aparecerán en la caja de texto **Custom Trace DLL**.
OK	Acepta los cambios de configuración en el **Log file path** y **Custom Trace DLL** y cierra el cuadro de diálogo **ODBC Data Source Administrator**.
Cancel	Cierra la caja de diálogo **ODBC Data Source Administrator** sin aceptar los cambios o nueva configuración.
Apply	Acepta los cambios para la configuración de seguimiento y mantiene abierto el cuadro de diálogo **ODBC Data Source Administrator**.
Help	Habilita la ayuda en línea.
Connection Pooling Tab	Agrupa conexiones y permite a una aplicación reutilizar manejadores de conexión abiertos. Que guarda round-trips en el servidor. Permite alterar el periodo de tiempo para reintentar una conexión y el período de time-out para el driver seleccionado cuando se esta utilizando una agrupación de conexiones, también permite habilitar y deshabilitar el monitoreo de rendimiento que registra estadísticamente el número de conexiones. El grupo de conexiones habilita a una aplicación para que utilice una conexión del grupo, la misma que no necesita ser reestablecida para cada uso. Una vez que se ha hecho una conexión y se ha colocado en el grupo, una aplicación puede reutilizar esa conexión sin necesidad de realizar el proceso de conexión completo, mejorando de esta manera el rendimiento.
ODBC drivers	Lista el nombre de cada driver instalado con la opción de time out para el grupo de conexiones. Para establecer la opción time-out, haga doble clic sobre el nombre del driver.
Connection Pooling Time out	Establece connection pooling time-out en segundos para el driver seleccionado. Para establecer los atributos del connection pooling haga doble clic sobre el driver seleccionado. Debe aparecer entonces el cuadro de diálogo **Set Connection Pooling Attributes**. Se puede establecer el periodo de tiempo que las conexiones que no se usan permanecen en el grupo y también activa el grupo de conexiones para el driver seleccionado.
Enable	Habilita el monitor de rendimiento para el grupo de conexiones.
Disable	Deshabilita el monitor de rendimiento para el grupo de conexiones.
Retry Wait Time	Especifica en segundos y hasta 6 dígitos, el tiempo que el ODBC Drive manager espera antes de reintentar una conexión hacia el DBMS.
OK	Acepta los cambios de la configuración de la agrupación de conexiones y cierra el cuadro de diálogo **ODBC Data Source Administrator** .
Cancel	Cierra la caja de diálogo **ODBC Data Source Administrator** sin aceptar los cambios o nueva configuración.
Apply	Acepta los cambios de la configuración de la agrupación de conexiones y deja abierto el cuadro de diálogo **ODBC Data Source Administrator**.
Help	Despliega la ayuda en línea.
About Tab	Despliega información acerca de los componentes core ODBC, incluyendo el Drive Manager, la librería de cursores, la DLL de instalación y otros archivos que constituyen el componente core.
Core component list	Descripción, versión, nombre de archivo de cada componente core ODBC

OK	Cierra la caja dad diálogo **ODBC Data Source Administrator**.
Cancel	Cierra la caja de diálogo **ODBC Data Source Administrator** sin aceptar los cambios o nueva configuración.
Help	Despliega la ayuda en línea

4.3. Metodología para crear un origen de datos (DSN)

Para crear un nuevo origen de datos, siga el procedimiento siguiente:

Paso 1. Haga clic en el menú **INICIO (START)** de Windows.

Paso 2. Escoja la opción **CONFIGURACIÓN (SETTINGS)** y escoja la opción **PANEL DE CONTROL (CONTROL PANEL).** Tal como se muestra en la Figura 4.3 .

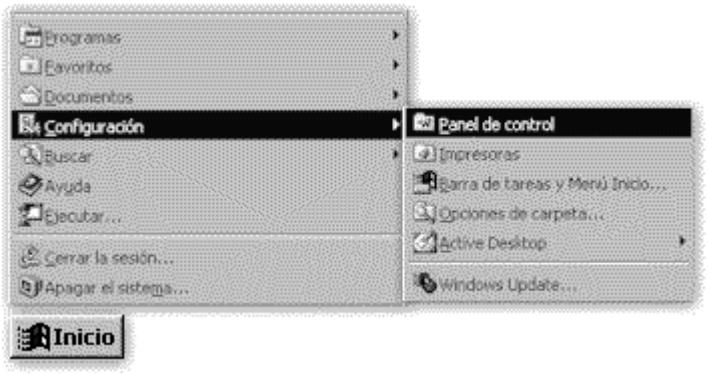


Figura 4.3

Paso 3. **En el PANEL DE CONTROL** ubique el ícono del **ODBC Data Source Administrator**, el mismo de la figura 4.4.

Figura 4.4

Paso 4. Al hacer clic en este ícono, deberá aparecer el **ODBC Data Source Administrator**, el mismo de la Figura 4.5.

Figura 4.5

Para conocer mas acerca de los componentes de esta ventana de diálogo, revise la Tabla 4.1 "*Componentes de la Ventana de diálogo ODBC Data Source Administrator"*

Paso 5. Seleccionar una solapa de **DSN** (User, system o file), la misma que en principio mostrará los orígenes de datos existentes y presione el botón **Add**. Se muestra la Figura 4.6.

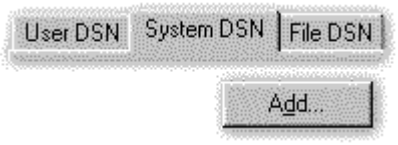


Figura 4.6

Paso 6. Se selecciona aquí el driver ODBC que se va a utilizar y presiona luego el botón **Finalizar**. Tal como se muestra en la Figura 4.7.

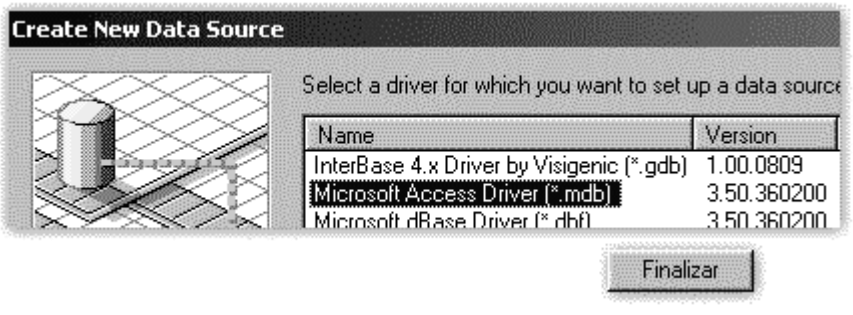


Figura 4.7

Paso 7. Finalmente, como ya sabemos cada driver ODBC tiene su propia interfaz de configuración, y en este momento se disparará el especificado.

Para probar lo indicado, a continuación se graficará y explicará un ejemplo con **MS ACCESS**.

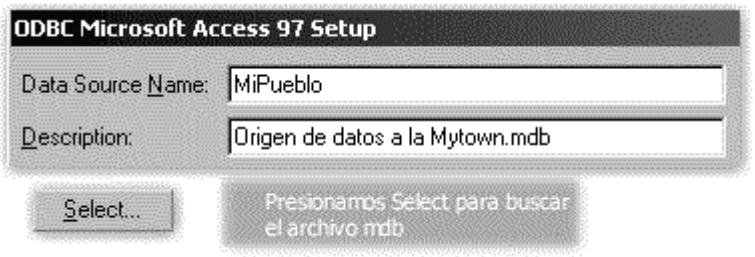


Figura 4.8

- Se ha escogido el driver de MS ACCESS.
- La interfaz del driver aparece y solicita ciertos datos para establecer una conexión, es este caso el DSN toma el nombre de **MiPueblo** y se lo describe como **Origen de datos a la Mytown.mdb** que en realidad es una base de datos en Access (básicamente no es necesaria una descripción).
- Luego se presiona el botón **Select** para iniciar la búsqueda del archivo mdb, se muestra en la Figuras 4.8 y 4.9 .

Figura 4.9

- Luego de ubicado el archivo mdb, debe aparecer el nombre y la ruta de localización del mismo, tal como se muestra en la figura 4.9 . En esta parte se presiona OK y ya esta listo un DSN de sistema para un archivo de Access.

Figura 4.10

Paso 8. Finalizada la configuración particular del driver ODBC, es necesario que se presione **ACEPTAR** en la ventana **ODBC Data Source Administrator** para finalizar y tener listo un DSN.

CAPÍTULO V

METODOLOGÍA PARA CREAR UN DRIVER ODBC MEDIANTE UN SDK

Es posible, mediante una herramienta de programación crear un driver ODBC, se deberá seguir una metodología básica de diseño y programación.

Microsoft afirma que un controlador que trabaje bajo la norma ODBC debe de cumplir al menos con el nivel Esencial de la gramática SQL y con el nivel 1 de la API de los niveles de conformidad propuestos en su especificación. Sin embargo, esto no es una regla ya que existen en el mercado controladores que mencionan la falta de funcionalidad en algunos aspectos de los niveles antes mencionados, dadas las características de su producto. Por esto, primeramente se deben determinar las características primordiales de funcionalidad del DBMS o del almacén de datos al que se desea conectar para determinar los niveles de conformidad que es posible aplicar.

5.1. Diseño de un driver ODBC

Tomando en consideración lo anterior, se pretende que el driver ODBC permita las siguientes funciones básicas:

- Establecimiento de la conexión a la fuente de datos
- Configuración de los atributos del controlador
- Preparación y ejecución de sentencias (basadas en la gramática soportada por el DBMS y/o en la necesaria para acceder a un almacén de datos particular no relacional)
- Obtención de la información del regreso de resultados
- Terminación de transacciones
- Terminación de la conexión
- Obtención de información y características del controlador
- Tratamiento de los errores más comunes
- Instalación del controlador

Como un ejemplo del método en que una aplicación cliente puede emplear la API de ODBC para la interconexión a un DBMS, se muestra el siguiente código, que es explicado a continuación:

```
#include "SQL.H"
#include "SQLEXT.H"
#include <string.h>

#ifndef NULL
#define NULL   0
#endif

int print_err(HDBC hdbc, HSTMT hstmt);

int main(UCHAR *server, UCHAR *uid, UCHAR *pwd)
{
HENV   henv;          /* crea los handlers de ambiente,
                         conexión y sentencia  */
HDBC   hdbc;
HSTMT  hstmt;

SDWORD  id;
UCHAR   name[51];
SDWORD  namelen;
UWORD   scale = 0;

/* EXEC SQL CONNECT TO :server USER :uid USING
:authentication_string; */
SQLAllocEnv(&henv); /* establece un handler de ambiente */
SQLAllocConnect(henv, &hdbc);  /* establece un handler de
conexión */

/* conexión a la base de datos */
if (SQLConnect(hdbc, server, SQL_NTS, uid, SQL_NTS, pwd,
SQL_NTS) != SQL_SUCCESS)
    return( print_err(hdbc, SQL_NULL_HSTMT) );

SQLAllocStmt(hdbc, &hstmt);  /* establece un handler de
sentencia */

/* EXEC SQL CREATE TABLE NAMEID (ID integer, NAME varchar(50));
*/
{
   UCHAR create[] ="CREATE TABLE NAMEID (ID integer, NAME
varchar(50))";

/* executa la sentencia SQL */
   if(SQLExecDirect(hstmt, create, SQL_NTS) != SQL_SUCCESS)
       return(print_err(hdbc, hstmt));
}
```

```
/* EXEC SQL COMMIT WORK; */
SQLTransact(henv, hdbc, SQL_COMMIT);  /* compromete la creación
de la tabla */

/* EXEC SQL INSERT INTO NAMEID VALUES ( :id, :name ); */
{
     UCHAR insert[]= "INSERT INTO NAMEID VALUES (?, ?)";
 /* se muestra el uso del método de SQLPrepare/SQLExecute  */
/* prepara la inserción */
   if(SQLPrepare(hstmt, insert, SQL_NTS) != SQL_SUCCESS)
      return(print_err(hdbc, hstmt));
   SQLSetParam(hstmt, 1,SQL_C_LONG, SQL_INTEGER,
(UDWORD)sizeof(UDWORD),
        scale, (PTR)&id, (SDWORD *)NULL);
   SQLSetParam(hstmt, 2,SQL_C_CHAR, SQL_VARCHAR,
(UDWORD)sizeof(name),
        scale, (PTR)name, (SDWORD *)NULL);

/* ahora se asignan los valores a los parámetros y se ejecuta
la inserción */
   id=500;
   (void)strcpy(name, "Babbage");
   if(SQLExecute(hstmt) != SQL_SUCCESS)
      return(print_err(hdbc, hstmt));
}
/* EXEC SQL COMMIT WORK; */
SQLTransact(hdbc, SQL_COMMIT);  /* se comprometen las
inserciones */

/* EXEC SQL DECLARE c1 CURSOR FOR SELECT ID, NAME FROM NAMEID;
*/
/* EXEC SQL OPEN c1; */
{
   UCHAR select[]= "select ID, NAME from NAMEID";
   if(SQLExecDirect(hstmt, select, SQL_NTS) != SQL_SUCCESS)
   return(print_err(hdbc, hstmt));
}
/* EXEC SQL FETCH c1 INTO :id, :name; */
/* uso del ligado de columnas con SQLBindCol */
SQLBindCol(hstmt, 1,SQL_C_LONG, (PTR)&id,
(SDWORD)sizeof(SDWORD), (SDWORD *)NULL);
SQLBindCol(hstmt, 2,SQL_C_CHAR, (PTR)name,
(SDWORD)sizeof(name), &namelen);

SQLFetch(hstmt);  /* ahora se ejecuta la obtención de
resultados */

/* finalmente se compromete la transacción y se desconecta */
/* EXEC SQL COMMIT WORK; */
SQLTransact(hdbc, SQL_COMMIT);  /* compromete la transacción
*/

/* EXEC SQL CLOSE c1; */
```

```
SQLFreeStmt(hstmt, SQL_DROP);    /* libera el handler de
sentencia */

/* EXEC SQL DISCONNECT; */
SQLDisconnect(hdbc);  /* se desconecta de la base de datos */

SQLFreeConnect(hdbc); /* libera el handler de conexión */
SQLFreeEnv(henv);     /* libera el handler de ambiente */

return(0);
}
```

Como primer paso, es necesaria la creación de los handlers que almacenan el estado actual del procesamiento de la consulta, así como de los atributos de la conexión y de la transacción, empleando las funciones de la API de ODBC ***SQLAllocEnv*** y ***SQLAllocConnect***. La conexión a la fuente de datos se realiza con ***SQLConnect*** y si es exitosa, se crea un handler de sentencia mediante ***SQLAllocStmt***. En el caso de que se elija el método de ejecución directa, se llama a la función ***SQLExecuteDirect*** con la sentencia SQL como parámetro. En caso de error, se muestra un mensaje con tipo de falla; en caso de éxito, se compromete la transacción usando ***SQLTransact***.

Para emplear el método de ejecución preparada, es necesario llamar a la función ***SQLPrepare***, la cual envía la sentencia SQL al DBMS para su preparación. En el ejemplo, se realiza una inserción usando variables locales, lo que permite al usuario establecer los valores a insertar al momento de la ejecución. La función ***SQLSetParam*** liga una variable local a un marcador definido dentro del texto de la sentencia (el símbolo ?), al momento de asignar posteriormente el valor a insertar en la variable, automáticamente se sustituye en la sentencia a ejecutar. La ejecución de la sentencia ya preparada se realiza con ***SQLExecute*** y se compromete con ***SQLTransact***.

La obtención del conjunto de resultados por la ejecución de una sentencia de selección se realiza empleando las funciones ***SQLBindCol***, la cual establece una liga entre las columnas del conjunto de resultados y una variable en donde se almacenarán los valores de cada renglón. Para cada columna es necesario llamar a esta función. La obtención de un renglón del conjunto de resultados se efectúa con la llamada a la función ***SQLFetch*** y en caso de varios renglones es necesario llamar a

dicha función tantas veces como sea necesario. Cada vez que se ejecuta ***SQLFetch***, se actualizan los valores en las variables locales ligadas con ***SQLBind***, por lo que es necesario almacenarlas en un arreglo o usando algún otro método de almacenamiento.

Cuando ya no se desean ejecutar más sentencias SQL, se libera el handler de sentencia con **SQLFreeStmt**, y se termina la conexión a la fuente de datos con **SQLDisconnect**. La liberación de los handlers de conexión y de ambiente se hacen con **SQLFreeConnect** y **SQLFreeEnv**, lo que termina el procesamiento de la aplicación.

La construcción de un controlador que cumpla con ODBC implica el conocimiento de varias tecnologías, tales como la programación de aplicaciones y la construcción de librerías de enlace dinámico para el ambiente Windows, incluyendo el soporte para la comunicación remota usando el protocolo de sockets. Ya más detalladamente, la creación de un driver ODBC en particular implica:

- ***Determinar la versión implementada de ODBC***.- Ya que la actualización de la versión 2.x a la versión 3.x de ODBC presentó un cambio representativo en el número de funciones y características mejoradas, es necesaria la elección de la versión que implique compatibilidad hacia atrás (esto es, que un controlador de versiones anteriores pueda funcionar con las nuevas versiones, con el mínimo de modificaciones).

- ***Determinar los niveles de conformidad de ODBC para el driver***.- Como se detalló anteriormente, un controlador deberá cumplir como mínimo con el nivel Esencial para la API y con la gramática Mínima SQL establecida por ODBC. A continuación se listan las funciones que deberán estar presentes como mínimo en el driver ODBC :

Tabla 5.1. Funciones de la API de ODBC de implementación mínima en el driver ODBC

SQLAllocConnect	SQLEndTran	SQLGetConnectAttr	SQLPrepare
SQLAllocEnv	SQLError	SQLGetConnectOption	SQLRowCount
SQLAllocHandle	SQLExecDirect	SQLGetCursorName	SQLSetConnectAttr
SQLAllocStmt	SQLExecute	SQLGetEnvAttr	SQLSetConnectOption

SQLBindCol	SQLFetch	SQLGetFunctions	SQLSetEnvAttr
SQLConnect	SQLFreeConnect	SQLGetInfo	SQLSetStmtAttr
SQLDescribeCol	SQLFreeEnv	SQLGetStmtAttr	SQLSetStmtOption
SQLDisconnect	SQLFreeHandle	SQLGetStmtOption	SQLTransact
SQLDriverConnect	SQLFreeStmt	SQLNumResultCols	

Aunque la funcionalidad básica de ODBC está proporcionada solo por algunas de estas funciones, las restantes son informativas o de configuración del controlador, permitiendo a una aplicación cliente más flexibilidad en su manejo.

No se tendrá que realizar ningún tipo de conversión de tipos de datos o una verificación exhaustiva de las construcciones gramaticales para traducirlas a la empleada por la fuente de datos. Se asegura que una aplicación cliente que utiliza la gramática SQL/92 para sus consultas, funcionará correctamente en un ambiente DBMS actual.

- ***Determinar el protocolo de comunicación remota***.- Es necesaria la implementación de rutinas que permitan establecer la comunicación remota entre el cliente y el servidor. El protocolo de *Berkeley sockets v4.3* es uno de los más empleados actualmente para la arquitectura Cliente/Servidor, de tal forma que se ha portado a la plataforma Windows® (actualmente en las versiones 1.1 y 2.0 de WinSock, aunque la versión 2.0 añade varias mejoras con respecto a la versión 1.1, está última es la más óptima en aplicaciones Cliente/Servidor,). Al momento de ejecutar la aplicación cliente, el driver es levantado en memoria para hacer la conexión al almacén de datos, así como también la librería de enlace dinámico **wsock32.dll** (en la versión de 32 bits), que administra la comunicación entre dos aplicaciones que utilicen el protocolo de sockets. Esto se hace una sola vez; si hay más conexiones a la misma fuente de datos o a alguna otra, solo se crean instancias de la librería de comunicación para sockets.

  Existen dos tipos de conexión dentro del protocolo de sockets: el orientado a conexión (TCP), que funciona de forma similar a una llamada telefónica, y el no orientado a conexión (UDP), que funciona similarmente al envió de cartas por correo. Sin embargo, el entorno Windows se basa en objetos, que se comunican entre sí mediante el envío de mensajes, de forma asíncrona, lo que hace más eficiente el multiprocesamiento. Para adaptar este modo de funcionamiento nativo

es necesario hacer adaptaciones al protocolo de sockets con el fin de incluir el modo orientado a conexión pero manejando mensajes de forma asíncrona. Para tratar con esta situación, fue ampliada la API de Winsock añadiendo algunas constantes y funciones, las cuales están definidas al final del archivo **winsock.h** (incluido en Visual C++) con el prefijo WSA.

Por la confiabilidad que presenta en el transporte de datos y por ser el más eficiente en ambientes Windows, se eligió el tipo de conexión TCP para la comunicación Cliente/Servidor, usando la API de WinSock del lado del cliente y la de Berkeley sockets estándar del lado del servidor.

- ***Determinar la complejidad implementada en el Administrador de Conectividad***.- La función primordial del Administrador de Conectividad es aceptar las peticiones que le envíe el driver ODBC de alguna aplicación cliente en Windows y hacer la traducción hacia la API nativa del DBMS. Este agente estará diseñado de tal forma que reciba las peticiones de varios clientes, y para cada una, genera un proceso hijo independiente, pero con las mismas características heredadas del padre, encargado exclusivamente de ejecutar tal petición. Esta implementación necesita que el padre se haga cargo en caso de algún error o falla del tratamiento o terminación de los procesos hijos creados por él.

  El planteamiento básico es que el servidor se haga cargo de la mayor parte del trabajo; es decir, si es necesario un análisis semántico de una sentencia SQL, ésta será enviada, tal como esté escrita, hacía el Administrador de Conectividad, y posteriormente hacía el Administrador de Transacciones del DBMS, quien iniciará el procesamiento de la consulta y regresará los resultados obtenidos, en lugar de implantar un analizador semántico de SQL en el Controlador o en el Administrador de Conectividad, lo cual incrementaría la complejidad de la solución. Es importante que el Administrador de Conectividad maneje correctamente el tipo de sentencia SQL recibida, para realizar el procesamiento en el caso de ejecuciones preparadas o ejecuciones inmediatas, con las correspondientes llamadas a las funciones de la API nativa del DBMS.

5.2. Metodología de desarrollo del driver

El desarrollo del driver tomó como base la metodología de creación de prototipos, en la cual se crea un prototipo funcional mínimo que cumpla las características principales requeridas por el proyecto, y posteriormente se pueda implementar otras características hasta alcanzar niveles más funcionales de acuerdo a los estándares del mercado.

Se distinguieron en el desarrollo del controlador tres etapas generales:

1. ***Investigación documental y análisis de requerimientos.***- Como punto de partida del proyecto, se recabó el máximo posible de información concerniente a los temas involucrados, como la arquitectura ODBC, el funcionamiento de los DBMS, posibles formatos de almacenes de datos, la creación de librerías dinámicas en ambiente Windows para el empleo de las funciones de conectividad del controlador y el desarrollo de aplicaciones empleando el compilador del SDK Visual C++. También se definieron las características que cubrirá el driver, sus entradas, los procesos que realiza, sus salidas y el tratamiento de errores. En esta etapa se evalúan la factibilidad de construcción de la aplicación, sus alcances y limitaciones, y sus posibles vías alternas de mejoramiento.

2. ***Construcción de la aplicación***.- Teniendo conocimiento de las técnicas y de los requerimientos necesarios, se comenzó la construcción del prototipo mínimo funcional, que permita la comprobación del correcto funcionamiento del driver para la ejecución de consultas sobre archivos de datos en formato CSV y la solución de problemas que se presenten durante la codificación. Se determinaron las acciones a seguir en los casos no previstos o en los que la funcionalidad puede ser extensible o mejorada para algunas otras características particulares.

3. ***Verificación y comprobación del funcionamiento de la aplicación.***- Cuando el grado de depuración de la aplicación se considera suficiente como para ponerla en funcionamiento, se realizan las pruebas que muestren la correcta ejecución de las funciones del controlador. Se deberá tomar en consideración que una mala

ejecución o una prueba insuficiente que no se detecte en las primeras etapas, será más difícil de detectar en las etapas posteriores.

Cabe destacar que durante todas las etapas de desarrollo de la aplicación, se documentaron todos las acciones efectuadas, ya sea de codificación y construcción, como de cambios, actualizaciones y mejoras hechas a la aplicación y que afectaron su funcionamiento. Cada función desarrollada o modificada deberá tener una correcta explicación de funcionamiento y de implantación, así como de los parámetros de entrada requeridos y de los valores de salida obtenidos.

5.3. Secuencia de construcción de una aplicación que use ODBC

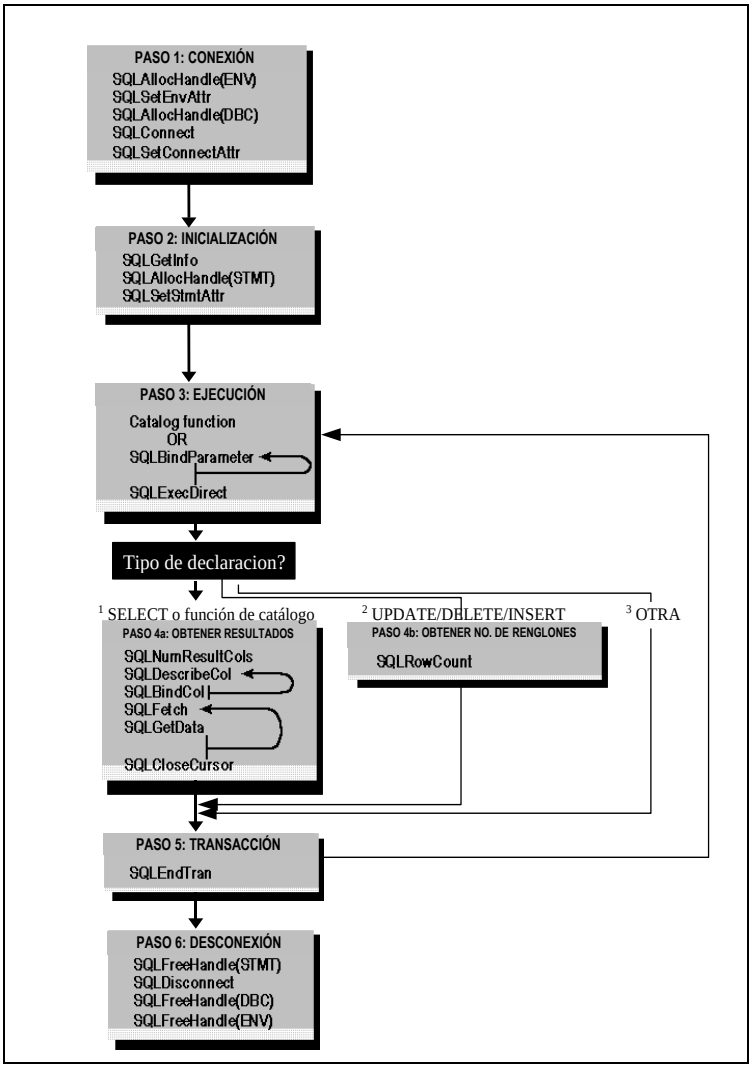


Figura 5.1

Para determinar las funciones implementadas en el driver ODBC, es necesario analizar la secuencia de procesamiento hecha por una aplicación cliente que use ODBC, como muestra la Figura 5.1.

De la Figura 5.1, se observan las funciones básicas que un controlador ODBC debe proporcionar al desarrollador de aplicaciones para que exista la interconexión a la fuente de datos y que por ende deberán estar presentes en el controlador ODBC para el sistema DBMS. Sin embargo, para otro almacén de datos tal vez sea necesario incrementar las funciones empleadas (tales como las de catálogo) en los niveles de conformidad superiores, para aprovechar sus características particulares y mejorar la eficiencia en el uso del sistema.

5.3.1. Codificación del Driver ODBC

Definidas las funciones de la API de ODBC que serán soportadas, el primer paso para

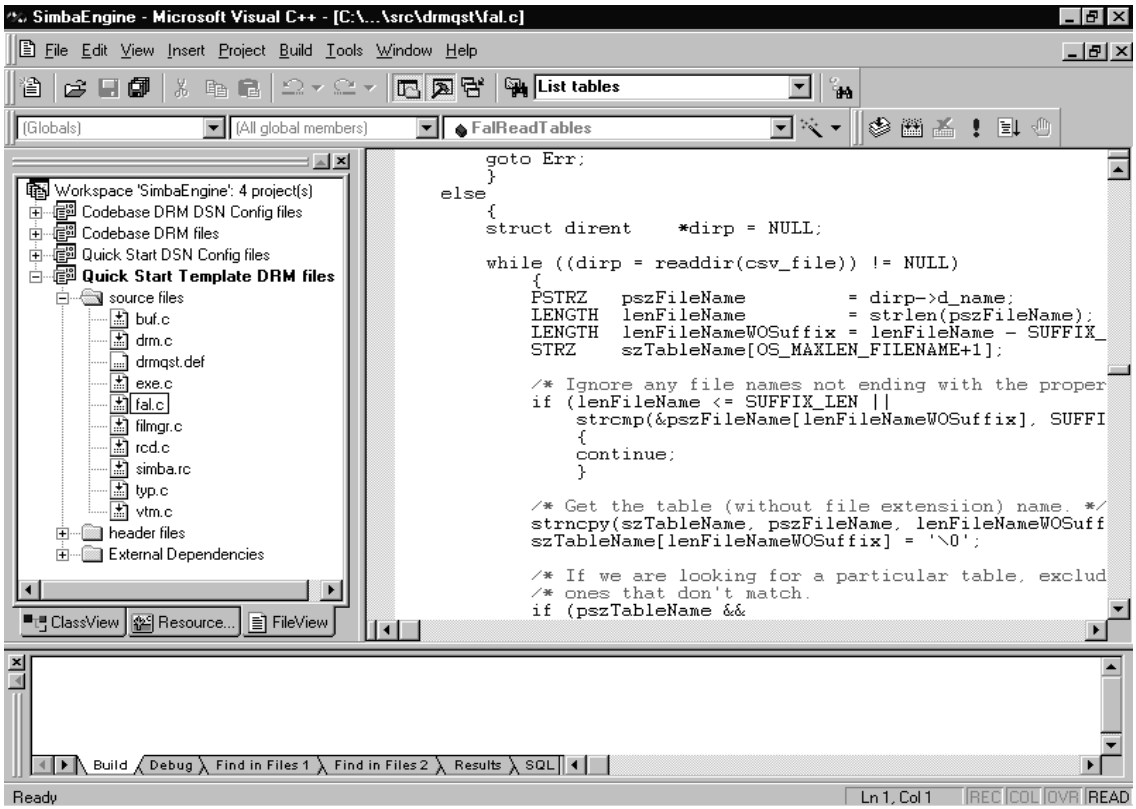


Figura 5.2

la construcción del Driver ODBC es la creación de un proyecto en el cual se incluyan todos los archivos necesarios para la construcción de la librería de enlace dinámico, usando el ambiente de programación del Microsoft Visual C++ y el QST del Simba Software Development Kit, como se muestra en la Figura 5.2.

Se observan en el panel izquierdo los archivos incluidos en el proyecto, entre los cuales están los archivos de cabecera con las definiciones de estructuras y prototipos de funciones de ODBC (buf.h, fal.h, vtm.h, etc.), así como el código fuente de inicialización de la librería dinámica (drmqst.def) y del Driver ODBC.

Las fases del proceso de construcción del Driver ODBC se indican en la siguiente figura:

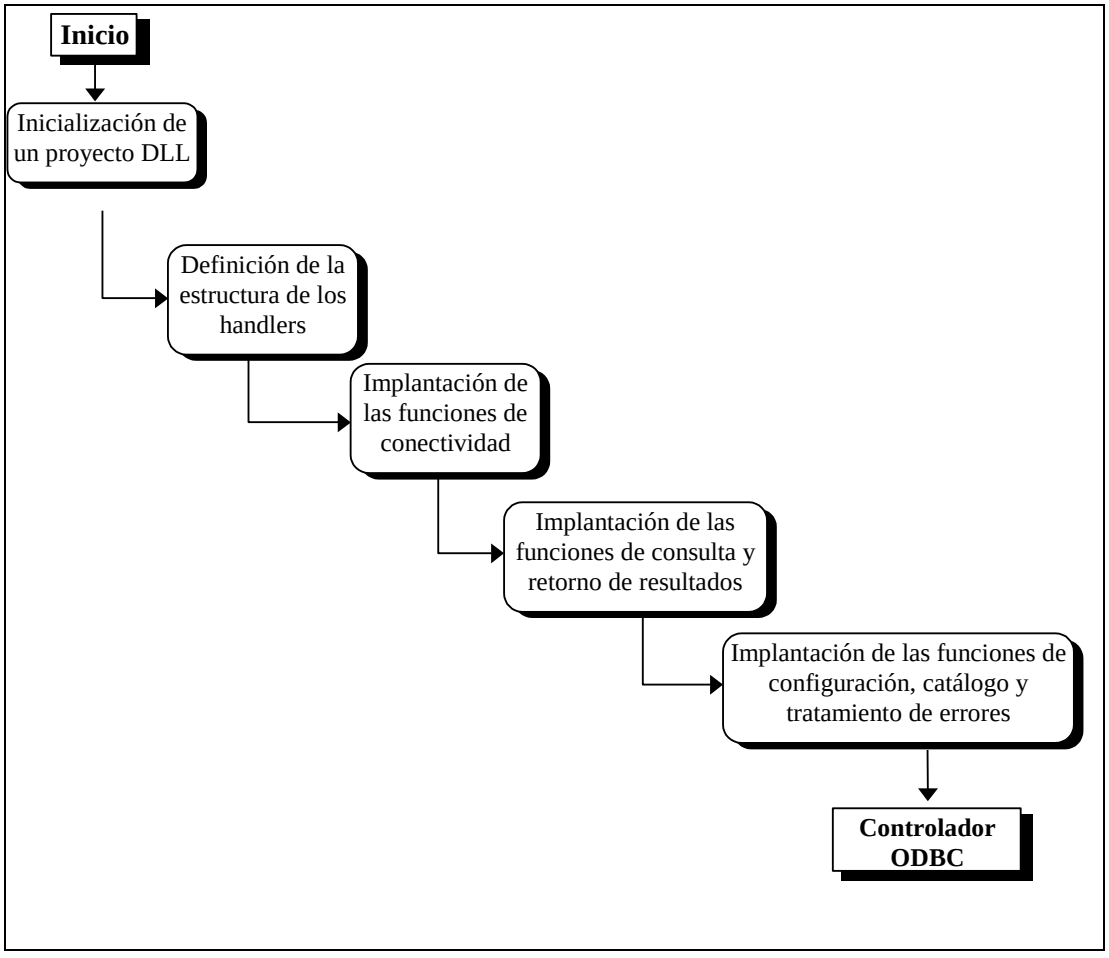


Figura 5.3

5.4. Introducción al SDK SimbaEngine

SimbaEngine en un SDK (Software Development Kit), que permite construir de manera modular un driver ODBC para acceso de datos.

SimbaEngine también permite proveer conectividad Cliente/Servidor para fuentes de datos SQL y no SQL, desde un simple servidor. Este motor incluye componentes de red para permitir una comunicación directa entre los computadores clientes y el servidor SimbaEngine usando el protocolo TCP/IP.

Un driver ODBC generado con SimbaEngine rinde tal como un driver de cualquier DBMS comercial, es decir permite crear, actualizar una base de datos relacional, además este motor convierte relativamente una base de datos propietaria (no necesariamente relacional), en un DBMS relacional totalmente operativo, permitiéndole a este realizar tareas como: abrir, recuperar, guardar o actualizar tablas en un almacén de datos.

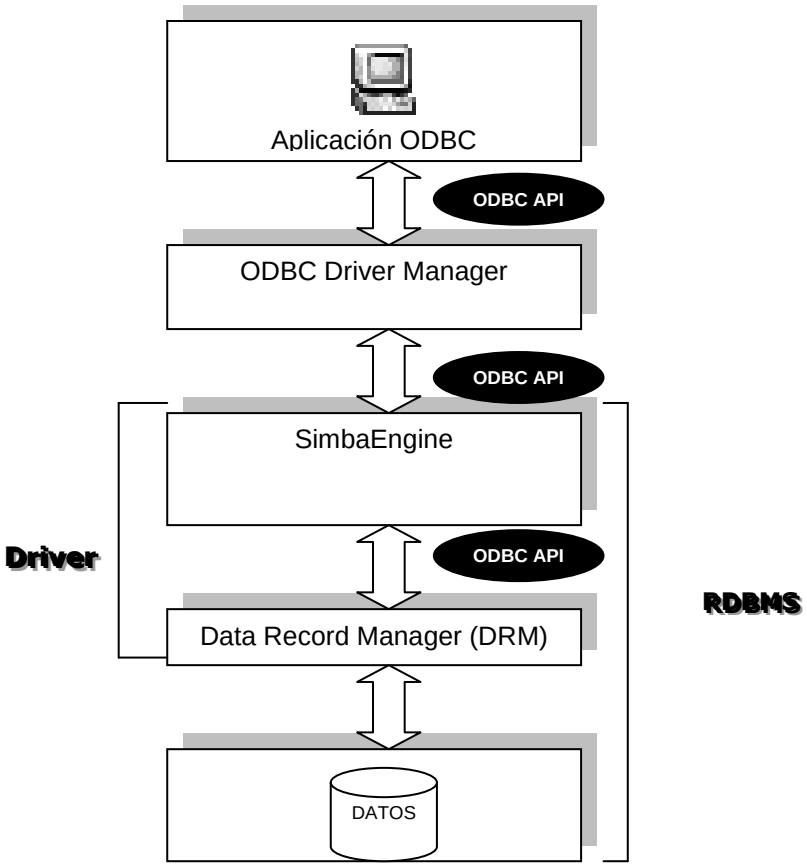


Figura 5.4

La Figura 5.4 presenta la relación entre una aplicación ODBC, un driver ODBC creado con SimbaEngine y un almacén de datos.

ODBC Compliant Application.- Esta es una aplicación que necesita acceder a un almacén de datos a través de estándares abiertos, estas pueden ser programas como: Microsoft Excel, Cristal Reports, Visual Basic, etc.

ODBC Driver Manager.- Es una interfaz que permite conectar aplicaciones Windows estándar con un driver. Una aplicación accede a un almacén de datos a través de llamadas ODBC API, algunas de las cuales incluyen sentencias SQL con argumentos. El ODBC Driver Administrator carga el driver ODBC para la aplicación y le pasa los requerimientos. Los resultados son pasados a la aplicación a través de llamadas a funciones ODBC.

SimbaEngine.- Es un motor SQL, que procesa llamadas a funciones ODBC, analiza sentencias SQL, y genera un plan óptimo para acceso de datos. Este realiza algunas llamadas de bajo nivel al Data Record manager (DRM) a través de la DRM API. SimbaEngine revisa los tipos de datos SQL y los traduce a tipos de datos accesibles a la capa DRM.

Data Record Manager (DRM) y DRM API.- El DRM es un conjunto de funciones de bajo nivel que proveen una interfaz para mapeo e interpretación de datos. Este presenta a SimbaEngine una vista relacional de una base de datos nativa, accede directa o indirectamente a cada uno de los archivos de base de datos, y retorna resultados hacia el SimbaEngine, soportando cualquier condición de error. Para construir un driver ODBC con SimbaEngine, se debería escribir el DRM modificando las funciones existentes en el Quick Start Template (QST). Este creara un único nexo entre los datos y el motor SimbaEngine.

Data Store.- El último componente son los datos en formato nativo. Los datos pueden ser accedidos por el DRM, a través de una API, para acceso de datos o DBMS, sin que esto afecte la arquitectura del driver ODBC creado con SimbaEngine.

5.4.1. SimbaEngine y el DRM

El DRM es un componente clave de un driver ODBC creado con SimbaEngine, este es un conjunto de funciones de bajo nivel para acceso de datos que provee una interfaz a estos. La capa DRM es el código que implementa la DRM API.

Cuando el motor SimbaEngine realiza llamadas a las funciones DRM API, este esta llamando a la capa DRM, para que acceda a los datos. Para que el motor pueda realizar su trabajo, los datos son presentados a través de la DRM API. El motor SimbaEngine solo entiende los datos que le devuelve las llamadas a las funciones de la DRM API.

El motor SimbaEngine requiere de la DRM API acciones como: abrir una tabla, moverse hacia la primera fila de una tabla abierta, leer una columna de una fila, moverse a la siguiente fila en la tabla, cerrar una tabla, y crear un diccionario de datos o tabla virtual.

La DRM API es orientada hacia tablas. Lo que hace que un driver ODBC sea más rápido y simple de implementar en contraste con otro SDK que no sea orientado a tablas.

Debido a que el motor SimbaEngine utiliza tablas para crear los diccionarios de datos solamente necesita implementar una forma para recorrer y recuperar información. Este motor refresca su diccionario de datos cada vez que un almacén de datos a sido accedido, asegurando la información si por acaso este almacén de datos vuelve a ser cambiado.

Para implementar un DRM, es necesario determinar la forma de presentación de los almacenes de datos, es decir, presentarlos como una serie de tablas en el caso de que no este en este formato. De todas maneras, la información de los diccionarios de datos serán presentados al motor SimbaEngine por el DRM API como una serie de tablas bien definidas. Muchos tipos diferentes de almacenamientos de datos pueden ser representados como tablas por SimbaEngine, por ejemplo la capa DRM accede a

archivos de datos planos, archivos de datos ISAM, archivos de datos Cobol, archivos de datos jerárquicos y archivos de datos multidimensionales, y en general, a todos los representa como tablas. Para implementar una capa DRM, es necesario crear un puente único entre los datos y el motor SimbaEngine.

5.4.2. Relación del programador con el motor SimbaEngine

Obviamente la capa DRM debe ser construida por el programador. Para construir un driver ODBC completo con el motor SimbaEngine, se recomienda seguir los siguientes pasos:

- Construir un prototipo de driver, con capacidad de lectura escritura usando el QST (Quick Start Template).
- Optimizar el driver y prepararlo para su distribución.
- Implementar funcionalidades avanzadas.

Construcción de un prototipo de driver con capacidad de lectura escritura.- Usando el QST es posible escribir la capa DRM al modificar un conjunto de funciones adicionando funcionalidades de lectura y posteriormente de escritura. QST es una plantilla de driver para acceder a archivos en formato CSV (Comma Separated Values). Este incluye funcionalidades básicas que al ser modificadas permiten construir un prototipo de driver**.**

Optimizar el driver y prepararlo para su distribución.- Una vez terminado el prototipo de driver sencillo es posible mejorarlo para que acceda a un almacén de datos, esto es adicionando índices, manejando mensajes de error y construyendo un instalador propio para propósitos de distribución.

Implementar funcionalidades avanzadas.- Cuando el driver ODBC ya ha sido implementado se puede adicionarle funciones avanzadas, por ejemplo procedimientos almacenados (Stored Procedures), predicados extendidos, entre otros para mejorar su rendimiento.

5.4.3. Tablas reales y tablas virtuales

El motor SimbaEngine reconoce tablas virtuales y tablas reales. Una tabla real contiene datos actuales almacenados localmente por sobre un servidor de archivos, en un formato nativo especifico a la base de datos.

Una tabla virtual es creada por la capa DRM a través de una llamada a función DRM API y contiene información en el diccionario de datos requerida por el motor SimbaEngine. Esta información consiste en la estructura de la tabla, nombre de columnas, nombre de tipo de datos, longitud de columnas, tipo de columnas, etc. Esta información en el diccionario de datos esta arreglada en un formato tabular para que pueda ser accedida fácilmente por el motor SimbaEngine. Este motor refresca el diccionario de datos cada vez que un almacenamiento de datos ha sido accedido, para asegurar los cambios en la estructura del almacén de datos. Una tabla virtual es solamente un arreglo de información del diccionario de datos, no es una tabla de base de datos actual o un archivo actual.

Incluido con el QST esta el VTM (Virtual Table Manager), esta herramienta permite crear tablas virtuales. Consiste de funciones que pueden ser llamadas para crear y adicionar datos a tablas virtuales. Si se utiliza el VTM, no es necesario escribir el código para manejar tablas virtuales.

5.4.4. Tareas del DRM

Cuando se implementa una capa DRM, se crea una plataforma para que ocurran ciertas tareas. Un driver ODBC completo generado por SimbaEngine (el motor realiza llamadas DRM API en la capa DRM) realizará las siguientes tareas:

- Conectar a un origen de datos
- Abrir y recuperar la información acerca de tablas
- Crear y eliminar tablas e índices
- Recuperar información de tablas
- Actualizar tablas

- Desconectarse del origen de datos
- Realizar funciones de utilidad general

5.5. Teoría del motor SimbaEngine

Esta sección presenta información acerca del motor SimbaEngine, que permitirá un mejor entendimiento del mismo.

5.5.1. Los objetos reconocidos por SimbaEngine

Con cada tarea que el driver pueda realizar el motor asocia cuatro tipos de objetos: sesiones, bases de datos, tablas y columnas. Estos objetos son ordenados en una estructura jerárquica bajo cada aplicación ODBC (ver figura 5.5).

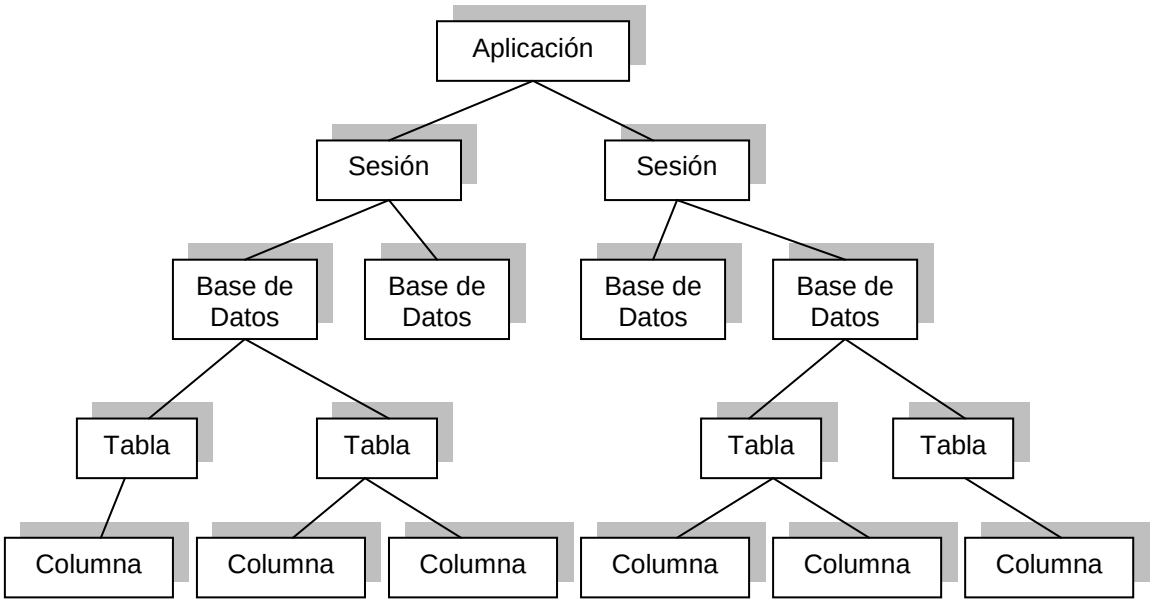


Figura 5.5

Sesión.- Una sesión es una conexión a la capa DRM. Cuando una aplicación establece una conexión con el driver a través de la llamada ODBC API: **SQLConnect** o **SQLDriverConnect**, el motor realiza algunas llamadas DRM API en la capa DRM y se establece una sesión. Una aplicación puede iniciar múltiples sesiones a través de múltiple conexiones, y múltiples aplicaciones pueden estar activas al mismo tiempo.

Base de Datos (DataBase).- Una base de datos es la agrupación de tablas que una aplicación accede durante una sesión activa. Cada sesión puede acceder a múltiples bases de datos. Una base de datos generalmente es un directorio de tablas, un archivo conteniendo tablas o cualquier otra agrupación de tablas.

***Tabla (Table)*.-** Una tabla contiene una o más columnas. El motor SimbaEngine reconoce tablas reales y tablas virtuales.

***Columna (Column)*.-** Cada columna en una tabla real es almacenada en un tipo de dato específico a un almacén de datos. La capa DRM revisa este tipo de datos especifico y lo interpreta en un tipo de datos DRM. Cada columna en una tabla virtual es recuperada por el motor SimbaEngine, en un formato especifico.

El motor SimbaEngine obtiene identificadores únicos para estos objetos desde la capa DRM a través de llamadas especificas a funciones DRM API. Estos identificadores son usados como argumentos en las llamas subsecuentes a funciones DRM API. Los identificadores son localizados por la capa DRM y son implementados como punteros a memoria que la capa DRM usa internamente para almacenar información sobre un objeto asociado.

Por ejemplo, un identificador de columna apunta a una estructura conteniendo el nombre de columna, en tipo de dato de la columna y el ancho de la columna. El motor SimbaEngine no modifica los identificadores, o nada de lo direccionado por los identificadores. El motor SimbaEngine requiere identificadores únicos de tabla y columna, sobre múltiples bases de datos e identificadores únicos de bases e datos sobre múltiples sesiones.

5.5.2. Tareas del DRM y la DRM API

Esta sección incluye las llamadas DRM API involucradas en cada tarea. También se describe como el motor SimbaEngine implementa la interfaz ODBC usando la DRM API y la secuencia que SimbaEngine utiliza para llamar las funciones.

El motor SimbaEngine realiza llamadas DRM API en la capa DRM, para realizar lo siguiente:

- Conectar a un origen de datos
- Abrir y recuperar la información acerca de tablas
- Crear y eliminar tablas e índices
- Recuperar información de tablas
- Actualizar tablas
- Desconectarse de un origen de datos
- Realizar funciones de utilidad general

***Conexión a una fuente de datos*.-** Cuando se conecta a un origen de datos SimbaEngine realizará las siguientes llamadas DRM API para inicializar los parámetros de sesión y de base de datos requeridos por la capa DRM (ver Figura 5.6).

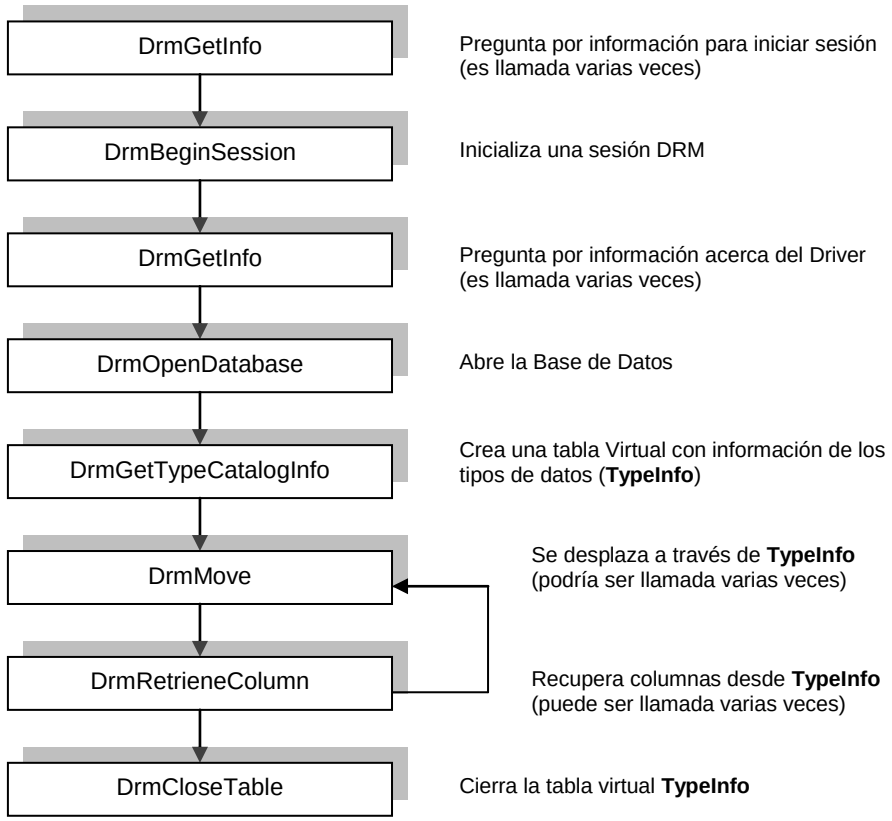


Figura 5.6

Cuando el motor se conecta a un almacén de datos requiere primero información acerca de la sesión, con una llamada a **DrmGefInfo.** Realizado esto el motor inicia una sesión DRM con una llamada a **DrmBeginSession** y requiere además información del driver con otra llamada **DrmGetInfo**. Una vez que SimbaEngine tiene la información del driver abre la base de datos con una llamada a **DrmOpenDataBase** y crea una tabla virtual de información de tipos de datos con una llamada a **DrmGetTypeCatalogInfo.** Entonces el motor recorre la tabla virtual **TypeInfo** y recupera sus columnas. Cuando el motor finaliza con la tabla virtual **TypeInfo** llama al **DrmCloseType.**

Abrir y recuperar la información acerca de tablas.- Cuando el motor SimbaEngine accede a los datos en una tabla, primero abre la tabla con una llamada a **DrmOpenTable.** Una vez que la tabla esta abierta el motor llama a **DrmGetTableInfo** para recuperar mas información a cerca de la tabla, entonces se llama a **DrmGetTableColumnInfo** y **DrmGetTableIndexInfo** para crear tablas virtuales que contienen información a cerca de la estructura de la tabla actual. El motor revisa cada tabla virtual llamando a **DrmMove** y recupera la información de las columnas de la tabla virtual llamando a **DrmRetrieveColumn**. Cuando el motor ha terminado con cada tabla virtual, son cerradas con una llama a **DrmCloseTable** . Las tablas virtuales pueden ser administradas por el VTM.

La Figura 5.7 ilustra una secuencia de llamadas DRM API que el motor SimbaEngine realiza para abrir y recuperar la información acerca de las tablas de la base de datos.

Recuperar información de tablas.- El motor SimbaEngine recupera datos desde las tablas de una base de datos en respuesta a una sentencia **Select** utilizada por una aplicación a través de la ODBC API (por ejemplo **SQLPrepare**, **SQLExecute**, **SQLFetch**, y **SQLGetData**).

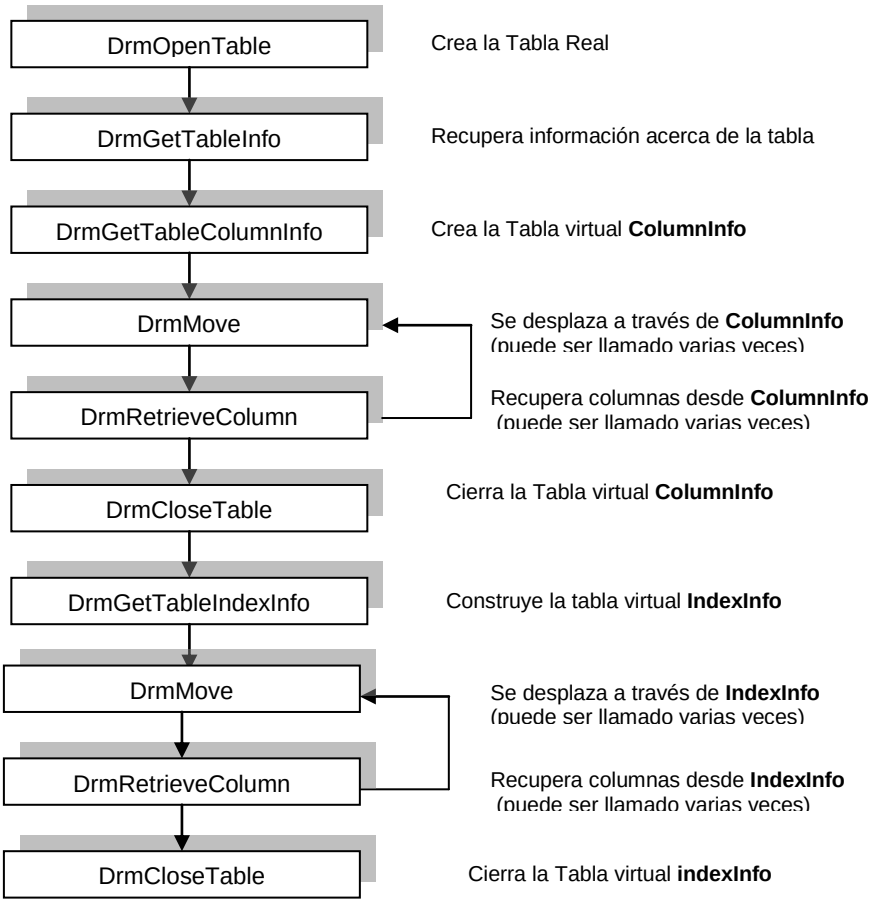


Figura 5.7

Para fuentes de datos que soportan índices el motor utiliza los índices existentes para optimizar las consultas SQL. El motor obtiene el nombre y características de los índices asociados con una tabla a través de una llamada a **DrmGetTableIndexInfo.** Utilizando esta información es posible optimizar las consultas. Luego de esto el motor SimbaEngine llega a un optimo QEP (Query Execution Plan) para la sentencia **Select**. Una vez que el QDP está en su lugar el motor realiza las subsecuentes llamadas en la capa DRM. En esta parte el motor SimbaEngine esta listo para recorrer la tabla (referirse a la Figura 5.8).

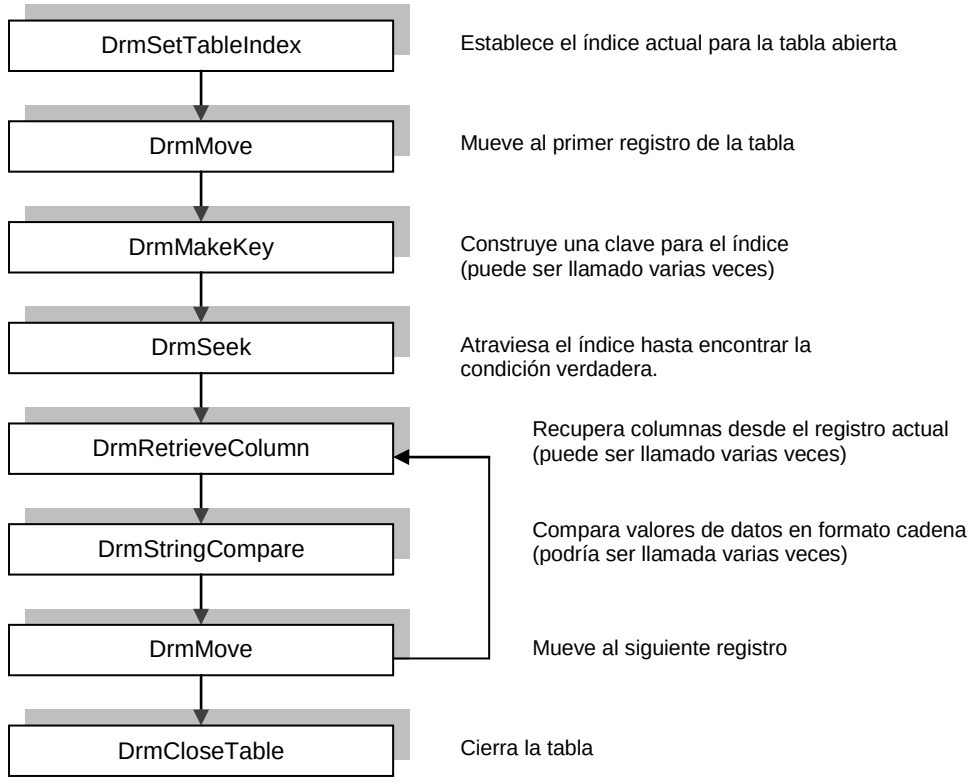


Figura 5.8

Primero el motor inicializa el índice apropiado de la tabla a través de la llamada a **DrmSetTableIndex** y **DrmMakeKey**.

Luego, el motor SimbaEngine recorre los registros de la tabla con llamadas a **DrmSeek** y **DrmMove**. El motor llama a **DrmRetrieveColumn** para recuperar información de una columna específica en el registro actual y, para algunas operaciones de relación, el motor llama a **DrmStringCompare** para comparar dos cadenas.

Cuando el motor SimbaEngine a terminado con la tabla realiza una llamada sobre la capa DRM para cerrar la tabla con **DrmCloseTable**.

A veces la sentencia **Select** contiene funciones agregadas (por ejemplo **order by**, **group by**) sobre un campo de texto largo, el motor SimbaEngine necesitará tener un marcador de posición (**bookmark**) para el registro actual desde la capa DRM.

Cuando SimbaEngine llama a **DrmGetBookmark** la capa DRM retorna un identificador único para el **bookmark**. Para colocar la posición del registro actual en

el **bookmark** obtenido con **DrmGetBookmark**, el motor llama al **DrmgetBookmark.**

Creación y eliminación de tablas e índices .- Cuando una aplicación invoca una sentencia **create table** o **drop table** a través de las funciones ODBC API **SQLPrepare** y **SQLExecute** (o **SQLExecDirect**) el motor SimbaEngine llama a una de las funciones DRM API (se muestra en la Figura 5.9).

DrmCreateTable	Crea una nueva tabla en la base de datos
DrmDeleteTable	Elimina una tabla de la base de datos

Figura 5.9

Cuando una sentencia **create index** o **drop index** es preparada y ejecutada por medio de funciones ODBC API **SQLPrepare SQLExecute** (o **SQLExecDirect**) el motor SimbaEngine llama a una de las funciones DRM API (ver Figura 5.10).

DrmCreateIndex	Construye un índice para una tabla especifica
DrmDeleteIndex	Elimina un índice especifico para una tabla

Figura 5.10

Actualización de tablas .- En una base de datos se puede requerir adicionar nuevos registros, eliminar registros existentes o actualizar registros en una tabla, en SQL estas operaciones corresponden a las sentencias **Insert**, **delete**, **update**.

Para insertar un nuevo registro o actualizar un existente en una tabla, el motor SimbaEngine realiza las siguientes llamas sobre la DRM API (ver la Figura 5.11).

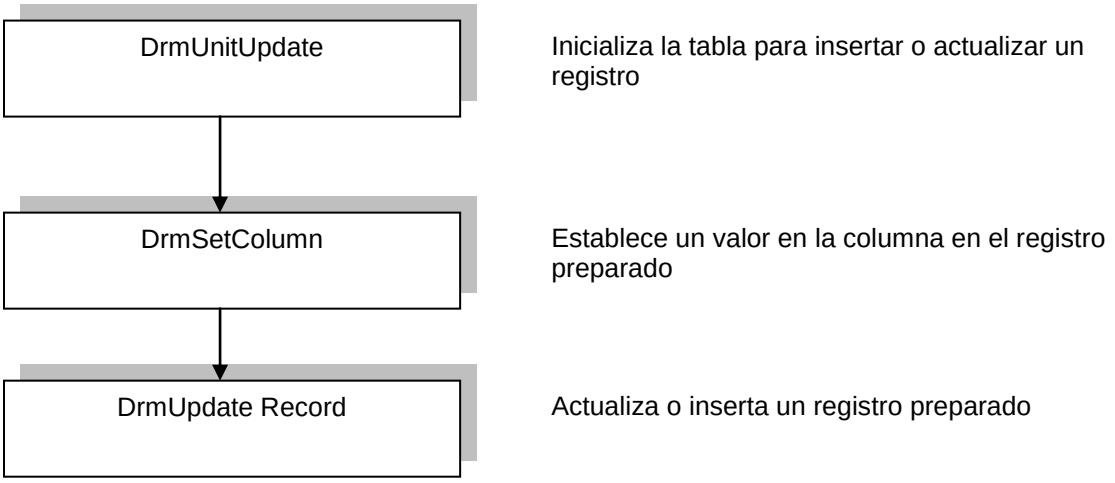


Figura 5.11

Dada esta secuencia de comandos es decisión del programador implementar la inserción o actualización de registros. Por ejemplo, la capa DRM puede sobrescribir un registro existente en una tabla o escribir un nuevo registro en una tabla, una columna a la vez cuando **DrmSetColumn** es llamada. En ambos casos, **DrmUpdateRecord** no debería hacer nada.

Como una alternativa la capa DRM puede almacenar una nueva columna de datos en un registro de datos preparado en memoria cuando **DrmSetColumn** es llamada, y sobrescribe el registro existente o inserta el nuevo registro en la tabla cuando se llama a **DrmUpdateRecord**. Un registro preparado es una copia del registro actual que el motor SimbaEngine actualiza y que lo inserta al final de la tabla para reemplazar un registro existente que esta marcado para borrar.

Los nuevos registros son insertados en la tabla en una posición determinada por la capa DRM. Los registros existentes son sobrescritos o borrados en la posición actual. Para eliminar el registro actual SimbaEngine llama a **DrmDeleteRecord** (se muestra en la Figura 5.12).

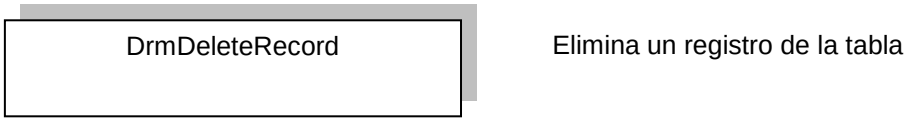


Figura 5.12

Desconexión de una fuente de datos.- Para desconectarse de una fuente de datos, el motor SimbaEngine realiza las siguientes llamadas en la DRM API (referirse a la Figura 5.13).

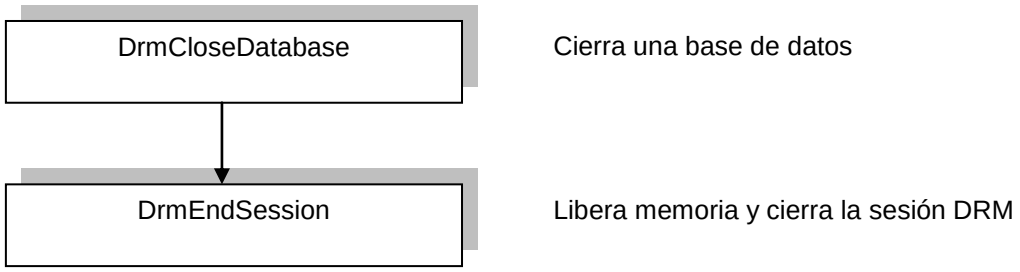


Figura 5.13

Desarrollo de funciones de utilidad general.- El motor SimbaEngine puede desarrollar tareas generales que no están relacionadas directamente a la fuente de datos. Por ejemplo SimbaEngine llama a **DrmStringCompare** para requerir que el driver DRM compare dos cadenas de acuerdo a una secuencia definida por el DRM. El motor utiliza esta función para ordenar nombres de tablas, nombres de columnas, datos en formato carácter, etc.

Otro ejemplo de una función de utilidad general es **DrmGetExtErrStr.** El motor llama a esta función para recuperara una cadena de mensaje de error asociado con un código extendido de error.

5.6. Teoría del QST (Quick Start Template)

En esta sección se podrá conocer mas acerca de la manera de trabajar del prototipo de driver que se personalizara para cumplir con los requerimientos de los almacenes de datos.

5.6.1. Concepto de QST

El Quick Start Template es una codificación inicial de un driver sobre el cual es necesario realizar una serie de adiciones y modificaciones de código para construir un prototipo de driver con capacidad de lectura-escritura. Una vez logrado esto es

posible modificar el código fuente para mejorar el rendimiento o adicionar mas utilidades.

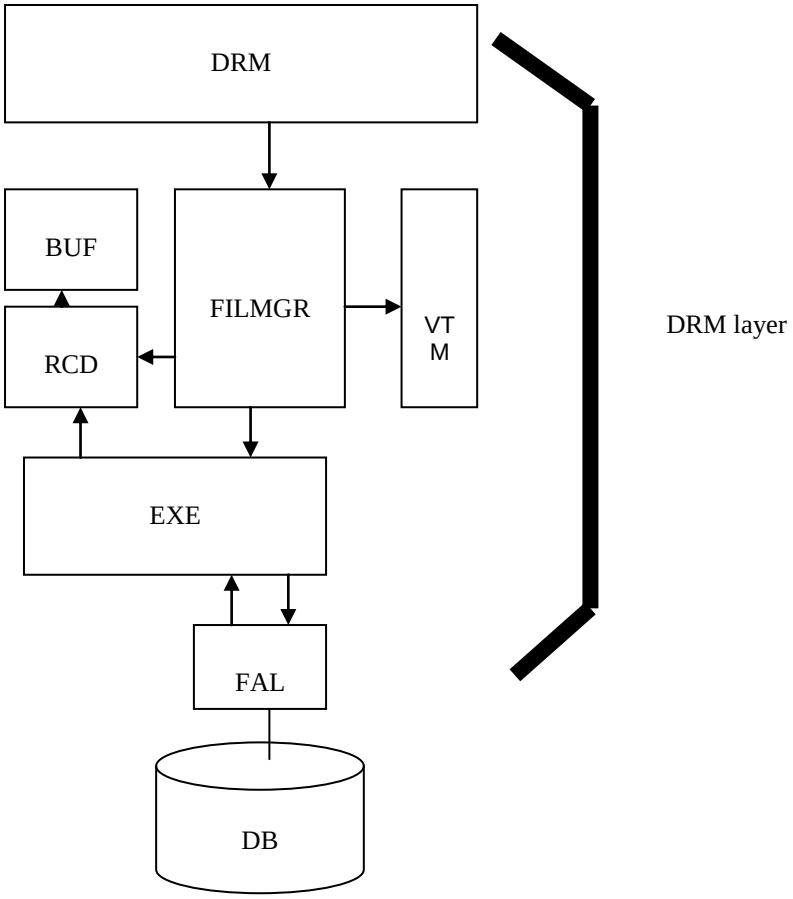
5.6.2. Arquitectura del QST

La arquitectura del QST consta de 8 componentes, de los cuales los 7 superiores se presentan a continuación con su respectivo acrónimo y se aprecia su relación en la Figura 5.14:

Tabla 5.2

DRM	Data Record Manager
VTM	Virtual Table Manager
BUF	Buffer API
RCD	Record Set manager
EXE	Executor
FAL	File Access Layer
FILMGR	File Manager

Figura 5.14



5.6.3. DRM (Data Record Manager)

DRM es la capa que el motor SimbaEngine llama directamente y, adicionalmente utiliza sus componentes para crear el QST. El código del DRM es contenido en el archivo **drm.c**.

Si se requiere escribir un driver ODBC sin utilizar el QST, se necesita implementar las funciones DRM y todos los componentes necesarios dependiendo del tipo de formato de datos que se requiera acceder.

En el QST, todas las funciones DRM llaman a otras funciones en el FILMGR (File Manager).

5.6.4. FILMGR (File Manager)

Este componente realiza llamadas a funciones para cumplir tareas de acceso a datos. El FILMGR fue diseñado para acceder a archivos binarios y archivos basados en texto, de igual manera que para registros de longitud variable y fija. Además este se encarga de llamar a las funciones necesarias para crear tablas virtuales. El código del FILMGR esta contenido en los archivos **FilMgr.c** y **FilMgr.h** . El File Manager utiliza los siguientes componentes:

RCD (Record set manager).- Este componente utiliza el componente BUF (Buffer API) para manejar el buffer y almacenar las estructuras de registros, estas contienen instancias de estructuras de campo que apuntan a los datos. El RCD puede crear, destruir, adicionar y remover registros.

BUF (Buffer API).- Este componente es un conjunto de funciones utilizadas principalmente para almacenar estructuras de registros, estas contienen estructuras de campo que apuntan a los datos. Un **Recordset** considerado como un conjunto de registros forma una tabla y puede ser tomado como un buffer que contiene estructuras de registro. El Recordset es manejado por el RCD.

El BUF se encarga de la manipulación del buffer para almacenar y acceder a memoria guardando la información del almacén de datos. El código del BUF esta contenido en el archivo **buf.c .**

VTM (Virtual Table Manager).- Es un conjunto de funciones que se utilizan en la capa DRM para crear y publicar tablas virtuales con información del diccionario de datos. El QST utiliza el VTM e incluye código en varias funciones para manejar automáticamente el paso de la información del VTM al SimbaEngine. El código del VTM esta contenido en el archivo **vtm.h** .

EXE (Executor).- Este componente es una capa que separa el FAL del FILMGR, además modera las relaciones entre ellos. EXE también contienen todas las funciones necesarias para extraer la información de la estructura de las tablas y las estructuras de las bases de datos. El código del componente EXE esta contenido en el archivo **exe.h**.

FAL (File Access Layer).- Este componente es un almacén de datos especifico. El código fuente de este componente se encuentra contenido en los archivos **fal.c** y **fal.h** dentro del proyecto QST y puede ser modificado para que un driver acceda a datos específicos.

FAL es totalmente independiente del FILMGR y del resto de componentes. La capa FAL puede tener muchas implementaciones dependiendo del almacén de datos y del formato en que estos se encuentren.

Inicialmente, este componente que es parte del QST accede a archivos en formato CSV. Este código puede abrir y cerrar tablas, recuperar información de tablas e inicializar buffers.

El componente FAL puede ser implementado de acuerdo a los requerimientos, es decir, se puede obtener un driver ODBC para cualquier forma de almacenamiento de datos sin cambiar en ninguna otra parte de la codificación.

5.6.5. VTM (Virtual Table Manager)

Este componente permite crear tablas virtuales. Una tabla virtual no es una tabla actual de base de datos, pero el motor SimbaEngine la mira como si lo fuera, esto es útil para recorrer y extraer información de las mismas. En conclusión, las tablas virtuales contienen diccionarios de datos acerca de tablas, columnas, tipos de datos e índices.

El motor SimbaEngine utiliza directamente estos diccionarios de datos para acceder a las bases de datos. La capa DRM es responsable de la organización de la información de estos diccionarios de datos en un formato de tabla que el motor SimbaEngine pueda entender, es decir, la capa DRM necesita crear las tablas virtuales para el motor.

La razón principal para que el motor SimbaEngine utilice tablas virtuales radica en que la definición de la información de los diccionarios de datos es simple, y esta información de los diccionarios de datos en la mayoría de las bases de datos cambia. Utilizando diccionarios de datos se asegura que todos los usuarios posean una información actualizada de la estructura de la base de datos.

El VTM oculta la complejidad de la creación de tablas virtuales, además esta diseñado para crear un sinnúmero de tipos de tablas virtuales diferentes, las mismas que son requeridas para diseñar e implementar la capa DRM. VTM utiliza dos librerías Simba de rutinas genéricas, **lst** para administración de listas y **sbm** para administración de memoria. El código del VTM se encuentra dentro del QST y no es necesario modificarlo. Si acaso no se quiere utilizar el VTM se deberá escribir el código para crear tablas virtuales dentro de las siguientes funciones DRM:

- **DrmGetDatabaseTableInfo . -** El motor SimbaEngine llama a esta función para requerir que la capa DRM construya una tabla virtual con información de todas las tablas en la base de datos.

- **DrmGetTypeCatalogInfo .-** El motor SimbaEngine llama a esta función para requerir que la capa DRM construya una tabla virtual con información de catalogo acerca de todos los tipos de datos soportados por la capa DRM.

- **DrmTableColumnInfo .-** El motor SimbaEngine llama a esta función para requerir que la capa DRM construya una tabla virtual con información de las columnas en la tabla.

- **DrmTableIndexInfo .-** El motor SimbaEngine llama a esta función para requerir que la capa DRM construya una tabla virtual con información sobre los índices asociados con una tabla especifica.

Estas son funciones adicionales del DRM que utilizan tablas virtuales.

- DrmTableForeignKeyInfo
- DrmTablePrivilegeInfo
- DrmTableColumnPrivilegeInfo
- DrmDatabaseProcedureInfo
- DrmProcedureColumnInfo

5.6.6. Modificaciones al QST

Aquí se describe la razón por la cual es necesario realizar modificaciones al QST y como el motor SimbaEngine utiliza esa información en las funciones FAL.

5.6.6.1. Modificaciones para obtener un driver con capacidad de lectura

Los pasos a seguir para construir un driver ODBC con capacidad de solo lectura de datos (SELECT), se detallan a continuación:

Paso 1 : Indicar el tipo de datos que el driver soportará

Se debe modificar la estructura **STTYPECATALOG** para indicar que tipo de datos soportara el driver.

Cuando se realiza una conexión al driver, SimbaEngine llama a **DrmBeginSession**, **DrmOpenDatabase** y **DrmGetTypeCatalogInfo**. **DrmGetTypeCatalogInfo**. Además utilizara la función VTM **MakeTypeCatVirtualTable** para crear una tabla virtual con información acerca del tipo de datos. Esta información se recuperara desde la estructura **STTYPECATALOG** que ya ha sido previamente modificada.

Paso 2 : Leer una lista de tablas de datos

Se debe modificar la función **FalReadTables** para indicar que tablas existen en el almacén de datos a tratar.

Una aplicación puede obtener una lista de tablas a través de su driver ODBC, entonces este llama a SQLTables. Este llama a los triggers del motor SimbaEngine para llamar a **DrmGetDatabaseTableInfo**, **DrmGetDatabaseTableInfo** utilizará la función VTM **MakeDatabaseInfoVirtualTable** para obtener la lista. **FalReadTables** será llamada para recuperar esa lista y pasarla a la aplicación.

Paso 3 : Extraer información de las columnas

Primero se debe modificar la función **FalOpen** para localizar memoria, crear una ruta a los datos y abrir un archivo de datos. Luego se debe modificar la función **FalReadColInfo**, para leer información de columnas de una tabla específica. Luego se modifica la función **FalClose** para revertir cualquier cosa que la función **FalOpen** haya realizado.

Para recibir información de todas las columnas en las tablas de un almacén de datos una aplicación deberá realizar llamadas **SQLColumns**. Este disparará una llamada de SimbaEngine, **DrmGetDataBaseTableInfo** (como se hace para **SQLTables**). Una vez que la lista de tablas haya sido recuperada se llamará a **FalReadTables**; **FalOpen**, **FalReadColInfo** y **FalReadData**, se llamarán por cada una de las tablas para recoger toda la información de las columnas. La información de cada tabla se compilará en una tabla virtual a

través de las llamadas a la función VTM **MakeTableColInfoVirtualTable**. Luego cada una de las tablas llamará a la función **FalClose.**

Paso 4 : Lectura de datos reales con el drive ODBC

Primero se debe modificar la función **FalGetInfo** para extraer cierta información en tiempo de ejecución, luego se debe modificar la función **FalReadData** para leer datos en un buffer. Finalmente se modifica la función **FalPackBuffer** para empaquetar los datos en un **recordset** que puede ser fácilmente revisado.

Cuando una consulta es ejecutada sobre del driver, el motor SimbaEngine, llama a **DrmOpenTable**, esto causará tres llamadas FAL, **FalOpen** , **FalReadColInfo** y **FalReadData**, para abrir la tabla y obtener toda la información requerida, cuando SimbaEngine llama a **DrmGetTableColumnInfo**, la VTM (**MakeTableColInfoVirtualTable**) se utilizará para crear una tabla virtual que contiene toda la información de las columnas para una tabla.

En esta parte, **FalReadData** y **FalPackBuffer** se llamarán para dar información de registro en una tabla y **FalGetInfo** será invocada varias veces para chequear el valor de las banderas.

5.6.6.2. Modificaciones para obtener un driver con capacidad de escritura

Paso 5 : Adición de funcionalidad para eliminación de registros

En este paso se modifica la función **FalDeleteRecord** para permitir eliminación de registros.

Cuando una consulta de eliminación es ejecutada sobre el driver todas las acciones mencionadas en el paso 4 para leer datos reales se repiten. Entonces SimbaEngine llama a **DrmDeleteRecord**, que a su vez llama a **FalDeleteRecord** y **FalDeleteInfo** para afectar las banderas requeridas.

Paso 6 : Adición de funcionalidad para actualización e inserción de registros

Aquí se debe modificar la función **FalUpdateRecord** y la función **FalInsertRecord.** Cuando se ejecuta una consulta de actualización en un driver ODBC todas las acciones mencionadas en el paso 4 se repiten. Siguiendo estos pasos SimbaEngine llamará a **DrmInitUpdate**, **DrmSetColumn** y **DrmUpdateRecord** esto disparará una llamada a **FalUpdateRecord** para actualizar el registro en la tabla real.

En el caso de un archivo en formato CSV, para actualizar registros **FalUpdateRecord** llama a **FalDeleteRecord** y **FalInsertRecord. FalGetInfo** será llamada también para recuperar las banderas requeridas.

Cuando una consulta de inserción es ejecutada sobre el driver todas las acciones mencionadas en el paso 4 se repiten. El motor SimbaEngine entonces llamará a **DrmInitUpdate**, **DrmSetColumn** y **DrmUpdateRecord** esto disparará una llamada a **FalInsertRecord**, para insertar el registro en una tabla real.

5.7. Desarrollo de un driver

En esta parte se tratarán cuatro:

- Instalación del motor SimbaEngine
- Archivos instalados
- Construcción de drivers SimbaEngine
- Importación de la configuración por defecto de SimbaEngine

5.7.1. Instalación del motor SimbaEngine

Requerimientos de software y hardware

- IBM PC o compatible (Pentium I o superior)
- 50 MB libres en disco

- 64 MB RAM
- Microsoft Visual C++ (32-bit Edition) Version 5.0 o superior
- Windows 95/98/2000/Me o Windows NT 4.0

Instalación

1. Inserte el CD-ROM.
2. En el explorador de Windows haga doble clic sobre **SimbaEngine61_SDK.exe**
3. Siga las instrucciones de instalación
    - Lea el acuerdo de licencia
    - Seleccione la ubicación de instalación
4. Haga clic en Finalizar

5.7.2. Archivos instalados

A continuación se lista los archivos y subdirectorios que son creados en el directorio de instalación. El directorio de instalación es **c:\SimbaTechnologies\SimbaSDK**.

Tabla 5.3

Directorio / Archivo	Descripción
\BIN	Contiene objetos y librerías compartidas y archivos ejecutables binarios.
\CODEBASE	Archivos de cabecera específicos para CODEBASE.
\DBF	Archivos de muestra usados para prueba con el CodeBase Testing Driver.
\DOCS	Archivo de documentación del SimbaEngine en formato PDF y SimbaClient for JDBC Online Help en formato HTML.
\IODBC-2.50.3	Código fuente del ODBC driver manager
\LIB	Contiene las librerías estáticas de lo construido.
License.txt	Información del licenciamiento.
readme.htm	Contiene información del paquete de instalación.
sample_odbc.ini	Archivo que permite una configuración rápida de las fuentes de datos sobre una plataforma Linux®.
sample_client.reg	Entrada de registro de prueba para **SimbaClient** que permite una configuración rápida de las entradas requeridas para una plataforma Windows®.

sample_odbc.reg	Entrada de registro de prueba para **ODBC** que permite una configuración rápida de las entradas requeridas, cuando se trabaja sobre una plataforma Windows®.
Simple_server.reg	Entrada de registro de prueba para **SimbaServer** que permite una configuración rápida de las entradas requeridas, cuando se trabaja sobre una plataforma Windows®.
Simbatst.ini	Archivo de muestra que permite una configuración rápida de la herramienta **SimbaTestTools** sobre una plataforma UNIX.
\SRC	Contienen directorios con código fuente de SimbaEngine.
SimbaEngine.dsw	Archivo de WorkSpace en Visual C++, para los componentes SimbaEngine.
\DRMCB	Ejemplo del DRM para **CodeBase**
\DRMQST	Quick Star Template DRM
\DRMQST	Motor SQL (SimbaEngine)
\ENGINE	Archivos de cabecera compartidos con variedad de componentes.
\INCLUDE	SimbaClient for ODBC para JDX 1.3. x (ODBC 2.1)
\JAVACLN2.1.x	Clases cliente
\JDBCTestSuite	Suite de prueba para clientes Java
\TESTSCR	Contiene scripts de prueba para **SimbaTestTools**
\DRMCB	Suite de prueba para el driver de ejemplo de **CodeBase**
\DRMQST	Suite de prueba para el driver QST
\TEXTDATA	Archivos de datos para probar el driver QST
\WEB	Todos los archivos relacionados con el SimbaServer Control Center y el SimbaServer ODBS Administrador.

Si el sistema operativo es Windows /95/98/2000/Me ® o Windows 2000®, se instalarán los siguientes archivos en el directorio **system**, o en el directorio **system32** si la plataforma es Windows NT:

Tabla 5.4

Archivo	Descripción
Ctl3d32.dll	Microsoft 3-D control DLL
mfc42.dll	Microsoft DLL redistribuida requerida por aplicaciones construidas por MFC
Odbc16gt.dll	Microsoft 16-bit ODBC generic thunking DLL
Odbc32.dll	Microsoft ODBC driver manager DLL
Odbc32gt.dll	Microsoft 32-bit ODBC generic thunking DLL
Odbccp32.cpl	Microsoft control panel applet
Odbccp32.dll	Microsoft 32-bit driver
Odbccr32.dll	Microsoft cursor DLL

Odbccint.dll	Microsoft language library
Odbcinst.hlp	Microsoft ayuda para la instalación de la ODBC DLL
Odbctrac.dll	Microsoft ODBC trace DLL
Simcbc32.dll	DLL para seteo y configuración del SimbaEngine CodeBase ODBC Testing Driver
Simeng32.dll	DLL para SimbaEngine CodeBase ODBC Testing Driver
Simspy32.dll	DLL requerida para ejecutar SimbaEngine DRM Spy

Cuando se instala el motor SimbaEngine, se instalan además los siguientes archivos del QST que corresponden al código fuente, estos son colocados en el directorio **c:\\SimbaTechnologies\SimbaSDK\src\Drmqst:**

Tabla 5.5

Archivo	Área funcional
Buf.c Buf.h	Rutinas para funcionalidad relacionadas con el buffer.
Drm.c	Implementación de funciones del DRM
Exe.c Exe.h	Funciones que llama el FAL API
Fal.c Fal.h	Funciones para acceso de archivos (implementación para archivos en formato CSV)
FilMgr.c FilMgr.h	Funciones para tabla base y rutinas funcionales relacionadas con tablas virtuales.
Drmqst.dsp	Archivo de proyecto del QST
Rcd.c Rcd.h	Rutinas de funcionalidad relacionadas con los recordset
Ses.h	Central type definition repositorio
Template.h	Central type definition repositorio
Typ.c Typ.h	Funciones que definen el soporte de tipo de datos.
Vtm.c Vtm.h	Funciones para construir la funcionalidad de tablas virtuales.

5.7.3. Construcción de un driver SimbaEngine

Los archivos a construir serán ubicados en **C:\SimbaTechnologies\bin.** Para reconstruir el código fuente siga los siguientes pasos.

1. Abrir Microsoft Visual C++ ®

2. Seleccione **File** | **Open Workspace**. Abra el archivo de WorkSpace para SimbaEngine: **SimbaEngine.dsw**, que se encuentra localizado en la ruta **c:\SimbaTechnologies\SimbaSDK\src**.
3. Desde la ventana de WorkSpace haga clic en el tab FileView.
4. Para cada entrada en la tabla, haga clic sobre el nombre del proyecto para seleccionarlo. Luego con el botón derecho del mouse haga clic y selecciones la opción "Set as Active Project", para seleccionar el proyecto a construir (referirse a la siguiente tabla de proyectos).

Tabla 5.6

DLL o EXE	Proyecto	Descripción
Drmcb.dll	CodeBase DRM	Driver de muestra para CodeBase DRM
Drmcb.cfg	CodeBase DRM DSN config	DLL de configuración del DSN para CodeBase DRM
Drmqst.dll	Quick Start template DRM	Driver QST.
Drmqst.cfg	Quick Stara DSN Config	DLL de configuración del DSN para el QST.

5. Seleccione la opción **Build | Build** <archivo destino> o presione F7 para construir el componente. Donde <archivo destino> es el nombre del componente DLL o el archivo EXE a construir.

5.7.4. Importación de la configuración por defecto de SimbaEngine

Los archivos de registro presentados como ejemplo (**sample_client.reg**, **sample_odbc.reg**, **sample_server.reg**), contienen la configuración por defecto que puede ser importada hacia el registro de Windows®, para el SimbaEngine, **SimbaServer** y el **SimbaClient**.

Para importar la configuración por defecto de **SimbaServer**:

1. Abra un archivo de registro (el cual vaya a registrar) en un editor de texto.
2. Reemplace todas las instancias de **<InstallDir>** con el directorio donde esta instalado SimbaEngine, para este caso **c:\ SimbaTechnologies\SimbaSDK**
3. Guarde el archivo de registro.

4. Desde el menú **Inicio** seleccione **Ejecutar**.
5. Escriba **REGEDIT** y presione aceptar para ejecutar el editor de registro.
6. Desde el menú **registry** seleccione "**Import registry file**".

5.8. Construcción de un prototipo de driver ODBC

Esta sección describe como construir un prototipo de driver ODBC con capacidad de Lectura/Escritura, utilizando el QST, esto se divide en dos partes.

- Construcción de un Driver con capacidad de lectura
- Construcción de un Driver con capacidad de escritura

5.8.1. Construcción de un Driver con capacidad de lectura

El QST es un driver funcional que permite acceder a archivos en formato CSV, este incluye funcionalidades básicas que al ser modificadas generan un prototipo de driver.

Se recomienda crear primero un driver con capacidad de lectura si la intención es obtener uno con capacidad de Lectura/Escritura. Se la realiza un proceso de cuatro pasos:

Paso 1: Indicar los tipos de datos que el driver soportará
Paso 2: Leer la lista de tablas de datos
Paso 3: Extraer la información de las columnas
Paso 4: Leer datos reales con el driver

En cada paso se deberá modificar algunas funciones del QST y luego realizar una prueba usando la herramienta **QuickTest Tool** para asegurarse que las funciones modificadas trabajen adecuadamente. La modificación de las funciones se las realizará usando Microsoft Visual C++ de Microsoft Visual Studio. Para obtener capacidad de lectura en el driver se deberá modificar 7 funciones y una estructura. Para esto será necesario:

1. Abrir Microsoft Visual C++

2. Seleccione **File** | **Open Workspace**. Abra el archivo de WorkSpace para SimbaEngine: SimbaEngine.dsw, localizado en **c:\SimbaTechnologies\SimbaSDK\src**.
3. Desde la ventana de WorkSpace haga clic en el tab. FileView.
4. Haga clic sobre el QST DRM Project, luego con el botón derecho del mouse haga clic y selecciones la opción "**Set as Active Project**", para seleccionar el proyecto a construir.

Paso 1 : Indicar el tipo de datos que el driver soportará

El QST soporta cuatro tipos de datos nativos: Char, Integer, Timestamp y Float. El código que se presenta permite acceder a almacenes de datos compuestos por archivos de datos con formato CSV, es decir todos los accesos serán específicamente para ese tipo de formato.

Se recomienda construir el prototipo con esos cuatro tipos de datos básicos y luego de haberlo probado adicionar soporte para otros tipos de datos.

Si estos cuatro tipos de datos son adecuados, no es necesario modificar nada y se debe avanzar al paso 2 para probar la conexión. Si se requiere otro tipo de dato, en primera instancia se puede usar el tipo Char para representarlo, en lo posterior se puede adicionar un tipo de dato, de la siguiente manera:

1. Modificar la estructura **STTYPECATALOG**.
   - Hacer doble clic en el archivo **typ**.**h**. Aquí se encuentra una lista de tipos de datos soportados.
   - Comentar los tipos de datos que no se requerirán o agregar los que se requieran.

   A continuación se presenta una muestra del código para esta estructura.

```
STTYPECATALOG
Static TYPE_CAT stTypeCatalog[] =
/* DRM type, Type name, Length required, Type attribute, Maximum Length, Maximum
```

```
Scale, Maximum Precision, Local type name
*/
{
{DRM_coltypText, "CHAR", True, 0, 254, 0, 0, "CHAR"},
{DRM_coltypLong, "LONG", False, 0, 0, 0, 0, "LONG"},
{DRM_coltypDateTime, "TIMESTAMP", False, 0, 0, 0, 0, "TIMESTAMP"},
{DRM_coltypEEESingle, "FLOAT", False, 0, 0, 0, 0, "FLOAT"},
};
```

2. Utilice la herramienta **QuickTest Tool** para realizar una prueba de conexión. Esta es una prueba preliminar para asegurarse que el driver puede conectarse correctamente a un almacén.

   - En Visual Studio, construya el driver seleccionando **Build | Rebuild All.**
   - En el directorio **c:\SimbaTechnologies\SimbaSDK\bin** haga doble clic en el archivo **QuickTest.exe**
   - En la sección **DATA SOURCES** del programa, seleccione **QuickStar Template DSN** desde el control de desplazamiento **Data Source Name**.
   - En un cuadro de texto adjunto aparecerá el path de la ubicación del archivo de datos. Este path es tomado del archivo de registro de Windows, para esta prueba no cambie el path.
   - En la sección **TESTS** seleccione **Connect Tool Data Source**.
   - Haga clic en la opción **Test.**

Los resultados de esta prueba aparecerán en la sección **RESULT** de la ventana. Al ejecutarse esta prueba debería aparecer el mensaje "**connect successful**". Si la prueba retorna algún error, revise los cambios que ha realizado en la estructura **STTYPECATALOG**.

Para información extra acerca de los tipos de datos que SimbaEngine soporta, refiérase a la sección "Data Types" en el "*SimbaEngine and DRM API referente Guide"*. Se puede revisar también la función **DrmGetTypeCatalogIngo** en la misma guía de referencia.

Para información sobre tipos de datos ODBC, refiérase al apéndice D, del *"ODBC Programmer's Reference"* que se puede encontrar en el sitio web de Microsoft.

Paso 2 : Leer la lista de tablas de datos

Ahora se debe listar las tablas que existen en el almacén de datos. Para esto realice lo siguiente:

1. Modificar la función **FalReadTables.**

   Haga doble clic en el archivo fal.c. Busque la función **FalReadTables,** que lee información a cerca de cada tabla en el almacén de datos y la almacena como una lista de estructuras. En la estructura de la base de datos.

   El siguiente pseudo código describe el código en esta función:

```
FalReafTables (TableList, Database Name)
    Para cada tabla en la base de datos:
        Crear una estructura de tabla;
        Colocar el valor del nombre en la estructura de tabla;
        Adicionar la estructura de tabla a TableList;
    Retornar error o éxito;
```

   - Escribir el código que recupera los nombres de las tablas en el almacén de datos.
   - Editar el código de la plantilla para asegurarse que **TableList** tiene estructuras de tablas.
   - Para cada tabla, asignar un valor al miembro **SZTABLENAME.** Se recomienda utilizar el nombre de la tabla.

   Lo que se necesita realmente es el nombre de la tabla. El código del QST se encarga del resto de ítems.

   El motor de SimbaEngine utiliza la función **AppendToList** que se encuentra en el archivo Fal.c para adicionar estructuras a la lista.

2. Utilice la herramienta **QuickTest Tool** para desplegar la lista de tablas del almacenamiento de datos.

   - Señale el proyecto **QuickTest**. Y haga clic con el botón derecho del mouse y escoja la opción "**Set as active project**".
   - Construya la herramienta **QuickTest Tool** seleccionando la opción **Build**.
   - Construya el driver seleccionando **Build | Rebuild All.**
   - Actualice la información del DSN en la entrada de registro para usar el driver.
   - Abrir el programa **QuickTest.exe** que se encuentra en **c:\SimbaTechnologies\SimbaSDK\bin**.
   - Verifique que el Data Source Name está correcto.
   - Cambiar la ubicación de los archivos de datos. Haga clic en **Browse** y seleccione el **Path** hacia el directorio donde se encuentra los archivos de datos.
   - En la sección **TESTS** de la ventana **QuickTest**, seleccione **List Tables**.
   - Haga clic en **TEST**.

   El resultado de esta prueba aparecerá en la sección **RESULT** de la ventana **QuickTest**, si la prueba retorna algún error, revise la información que se cambio en la función **FalReadTables**. Si existe algún problema, revise el código con el **Debugger**.

Paso 3 : Extraer información de Columnas

En esta parte es necesario modificar tres funciones.

1. Modificar la función **FalOpen**.
   Edite el archivo **Fal.c** y busque la función **FalOpen**, este código permite separar memoria crear un path al archivo de datos y abrirlo.

   El siguiente pseudo código describe el código en esta función:

```
FalOpen (Database structure, table structure)
    Separa memoria para la estructura Fal;
```

Encuentra la tabla;
Abre la tabla;
Almacena la estructura fal en la estructura tabla;
Retorna error o éxito;

No es necesario remover nada de código, en caso de que este no sea aplicable al driver que se está construyendo. Algunas de las estructuras dentro de esta función son requeridas en otras funciones en lo posterior.

Para extraer el nombre de la base de datos y el nombre de la tabla desde la estructura de tabla, se utiliza las funciones **GetDataBaseName () y GetDataBaseTable().**

2. Modificar la función **FalReadColInfo**.
Dentro del archivo Fal.c ubique la función **falReadColInfo** que lee la información de columna para una tabla especificada.

El siguiente pseudo código describe el código en esta función

FalReadColInfo (Table structure)
Para cada columna en la tabla:
Crear una estructura columna;
Establecer el nombre de columna, tipo de columna
para cada estructura columna;
adicionar la estructura columna a la lista de columnas
en la estructura de tabla;
Retorna error o éxito;

Edite el código para publicar la estructura d columna y adicionarla a la lista.

La lista de columnas es publicada utilizando la función **AppendToList** y es extraída utilizando la función **GetTableColumnList**.

3. Modificar la función **FalClose**.
Dentro del archive Fal.c, ubique la función **FalClose**, aquí el código permite liberar memoria y cerrar los archivos de datos.

El siguiente pseudo código describe el código en esta función:

```
FalClose (DataBase structure, Tabla structure)
        Liberar de memoria la estructura fal;
        Cerrar la tabla;
    Retorna error o éxito;
```

Edite el código para cerrar la tabla.

Para información adicional sobre lectura o extracción de la estructura de tabla refiérase a la función **DrmGetTableColumnInfo** en el *"SimbaEngine and DRM API Reference Guide"*.

4. Utilice la herramienta **QuickTest Tool** para actualizar una prueba del listado d e columnas para determinar si el código modificado abre y cierra las tablas del almacén de datos y además recupera la información de columnas.

   - Construir el driver seleccionando **Build | Rebuild All**
   - Ejecute la herramienta **QuickTest Tool**.
   - Verificar que el DSN y la ubicación de los archivos de datos son correctas.
   - En la sección **TEST** de la ventana **QuickTest**, seleccione **List Columns of Table**. En la lista desplegable seleccione en la tabla de la cual se desea listar las columnas. Si se desea listar todas las columnas de todas las tablas en el almacén de datos, seleccione **<All Tables>** desde la lista desplegable.
   - Haga clic en **TEST**.

El resultado de esta prueba aparecerá en la sección **RESULT** de la ventana **QuickTest**, si la prueba retorna algún error, revise la información que se cambio en la función **FalOpen, FalReadColInfo, y FalClose**. Si existe algún problema, revise el código con el **Debugger**.

Paso 4 : Leer datos reales con el driver

Si ya se ha verificado que las modificaciones del paso anterior funcionan, el siguiente paso consiste en leer datos reales con el driver. El procedimiento a seguir se detalla a continuación.

1. Modificar la función **FalGetInfo**

   Dentro del archivo **fal.c** ubique la función **FalGetInfo**, la cual extrae información relevante a cerca de los datos durante el tiempo de ejecución.

   Dentro de la función **FalGetInfo** existen dos banderas que pueden ser establecidas a **TRUE** o **FALSE.** Una bandera, **INFO_CONVERT** indica cuando los datos están almacenados en formato carácter dentro de los archivos de datos. La otra bandera **INFO_MOREDATA**, indica si los archivos de datos son muy grandes para colocarlos en el buffer uno a la vez, y si se debe llamar varias veces **a FalReadData**. **FalRealData** se modificará en el paso siguiente.

   El siguiente pseudo código describe el código de la función **FalGetInfo.**

```
FalGetInfo (Tablestructure, FLAG)
    Si FLAG es INFO_CONVERT retornar True o False;
    Si FLAG es INFO_MOREDATA retornar True o False;
Retornar error o éxito;
```

   El valor por defecto de la bandera **INFO_CONVERT** es **TRUE**, que significa que los datos están almacenados en formato **CHAR** y todas las conversiones pueden estar manejadas por el código **QST**. Si se retorna **FALSE,** se deberá adicionar algo de código en **FalReaData** para manejar la conversión.

   El valor por defecto de la bandera **INFO_MOREDATA** es **FALSE**, que indica si el archivo de datos en su totalidad o la tabla de datos serán leídos en memoria. Si la tabla es muy grande la lectura se realizará por partes y se colocará la bandera

INFO_MOREDATA a **TRUE.** Cuando no existen mas datos para leer, se debe colocar la bandera **INFO_MOREDATA** a **FALSE.**

2. Modificar la función **FalReadData**

   Dentro del archivo **Fal.c** ubique la función **FalReadData**, la misma que lee datos desde una tabla en un buffer de memoria.

   El pseudo código siguiente describe el código de esta función:

```
FalReadData (Table structure)
    Separar memoria para guardar todo o parte de la tabla;
    Leer los datos en el buffer;
    Almacenar el buffer de datos en la estructura fal;
Retorna error o éxito;
```

   Se debe editar el código para localizar memoria para los datos dentro de esta función. Existe la opción de leer el contenido completo o parte de la tabla en memoria. Si se ha leído únicamente una parte de la tabla se debería retornar **TRUE** cuando **FalGetInfo** es llamada con la bandera **INFO_MOREDATA**.

   Si la tabla esta vacía entonces retorna éxito. Para archivos en formato CSV esta tarea es fácil porque los valores están separados por comas y los registro están separados por caracteres de nueva línea.

   Si el tratamiento de los datos se los realiza en el buffer no es necesario modificar la función **FalPackBuffer** en el paso 3 siguiente.

3. Modificar en la función **FalPackBuffer**.

   Dentro del archivo **fal.c** ubique la función **FalPackBuffer.** El empaquetado de los datos dará como resultado un recordset lleno de registros y campos que pueden ser fácilmente explorados para retornar datos específicos cuando sea necesario.

   - Use **ExeCreateRecord** para crear la estructura **RECORD**

- Use **ExeInsertField** para crear e insertar una estructura **FIELD** en una estructura **RECORD**.
- Use **ExeInsertRecord** para insertar la estructura **RECORD** en la estructura **RECORDSET** que es contenida en la estructura **TABLE.**

Estas son funciones para cada miembro de la estructura **TABLE**. Esta podría ser la mejor ocasión para almacenar cualquier información especifica a las ubicaciones de los registros en la tabla. La estructura **TABLE** tiene un miembro que puede ser utilizado para almacenar la posición del archivo (Record Number) para que su ubicación en el archivo se conozca. Esta información puede ser utilizada también para eliminar o actualizar funciones.

El siguiente pseudo código describe el código en esta función.

```
FalPackbuffer (Table structure)
    Extraer cada registro del buffer creado en FalReadData;
    Por cada registro:
            Crear una estructura registro,
            Insertar cada campo del registro de datos en la estructura registro;
            Insertar la estructura registro en el Recordset;
Retorna error o éxito;
```

- Editar el código para crear registros y llenar los campos con datos.
- Editar el código para insertar los registros en el **recordset**.

4. Utilice la herramienta **QuickTest Tool** para realizar una prueba de consulta para determinar si el código adicionado o modificado recuperará datos desde las tablas.

- Construya el driver seleccionado **Build** | **Rebuild All.**
- Verifique que el **DSN** y la ubicación de los archivos de datos este correcta.
- En la sección **TEST** de la ventana **QuickTest**, seleccione **Execute Query**
- En la ventana junto a la selección **Execute Query**, ingrese una sentencia **Select** sin la cláusula **group by**.

- La herramienta **QuickTest Tool** puede realizar múltiples consultas. Cada sentencia de consulta se puede separar con un caracter **<Enter>**.
- Haga clic en **TEST.**

El resultado de esta prueba aparecerá en la sección **RESULT** de la ventana **QuickTest**, si la prueba retorna algún error, revise la información que se cambio en la función **FalOpen, FalReadColInfo, y FalClose**. Si existe algún problema, revise el código con el **Debugger** para ubicar el problema.

Con los pasos anteriores se ha completado la construcción de un driver ODBC con capacidad de solo lectura. Este driver puede conectar un almacén de datos, abrir, cerrar tablas, y recuperar datos. Se verá mas adelante la manera como adicionar capacidades a este driver.

5.8.2. Construcción de un driver con capacidad de escritura

En esta parte se supone que ya se ha construido un driver con capacidad de solo lectura. Adicionando o modificando cierto código al mismo driver se lo podrá dotar de funcionalidades tales como inserción, actualización y eliminación de datos sobre tablas. Este proceso requiere de dos pasos:

Paso 5: Adicionar funcionalidad para eliminar registros.
Paso 6: Adicionar funcionalidad para insertar registros.

Se deberá modificar el código del QST, cambiando tres funciones **FalInsertRecord, FalDeleteRecord y FalUpdateRecord**. En cada paso se modificará una o dos funciones y luego se deberá realizar una prueba utilizando la herramienta **QuickTest Tool** para asegurarse que la nueva información adicionada o modificada a las funciones trabajen correctamente.

Paso 5 : Adicionar funcionalidad para eliminar registros.

El primer paso para adicionar capacidad de escritura es dotar de la función para eliminación de registros de las tablas de datos:

1. Modificación de la función **FalDeleteRecord**.
   Dentro del archivo fal.c ubique la función **FalDeleteRecord** la misma que eliminará un registro de una tabla de datos.

   El siguiente pseudo código describe el código en la función:

   FalDeleteRecord (Table structure)
   Consiga el registro actual de la estructura table;
   Elimine el registro actual de la tabla real;
   Retorna error o éxito;

   Edite el código para obtener el registro de la estructura tabla entonces eliminar el registro de la tabla.

2. Utilice la herramienta **QuickTest Tool** para realizar una prueba **Execute Query** y determinar si el driver puede eliminar registros de las tablas da datos.

   - Construya el driver seleccionado **Build** | **Rebuild All.**
   - Abra la herramienta **QuickTest Tool**.
   - Verifique que el **DSN** y la ubicación de los archivos de datos este correcta.
   - En la sección **TEST** de la ventana **QuickTest**, seleccione **Execute Query**
   - En la ventana junto a la selección **Execute Query**, ingrese una sentencia SQL para eliminación.
   - La herramienta **QuickTest Tool** puede realizar múltiples consultas. Cada sentencia de consulta se puede separar con un caracter **<Enter>**.
   - Haga clic en **Test.**

El resultado de esta prueba aparecerá en la sección **RESULT** de la ventana **QuickTest**, si la prueba retorna algún error, revise la información que se cambio en

la función **FalOpen, FalReadColInfo, y FalClose**. Si existe algún problema, revise el código con el **Debugger** para ubicar el problema.

Paso 6: Adicionar funcionalidad para actualización e inserción de registros.

Este procedimiento consta de dos partes:

1. Modificar la función **FalInsertRecord** dentro del archivo **fal.c** ubique la función **FalInsertRecord** la misma que inserta un nuevo registro en una tabla, el siguiente pseudo código describe el código de esta función:

```
FalInsertRecord (table structure, record structure)
    Obtenga el registro a insertar desde la estructura registro;
    Adicione el registro a la tabla real;
Retornar error o éxito;
```

   Edite el código para insertar un registro en la tabla.

2. Modificar la función **FalUpdateRecord**.

   Dentro del archivo fal.c ubique la función **FalUpdateRecord** la misma que actualiza algunos o todos los campos de un registro.

   El siguiente pseudo código describe el código de esta función:

```
FalUpdateREcord (Table structure)
    Obtenga la estructura registro para actualizar desde la estructura tabla;
    Reemplace el actual registro en la tabla con 1 desde la estructura tabla;
    Retornar error o éxito;
```

   El QST localizará y almacenará una estructura RECORD en la estructura TABLE. Entonces **FalUpdateRecord** será llamada. Dentro de **FalUpdateRecord**, el código elimina el registro antiguo y adiciona un registro nuevo a la tabla real esta implementación sirve solamente para archivos en formato CSV.

Utilice la función **ExeExtractField** para extraer estructuras FIELD, desde la estructura RECORD y la función **GetFieldData** para extraer los datos de la estructura FIELD.

3. Utilice la herramienta **QuickTest Tool** para realizar una prueba **Execute Query** y determinar si el driver puede insertar y actualizar.

   - Construya el driver seleccionado **Build** | **Rebuild All.**
   - Abra la herramienta **QuickTest Tool**.
   - Verifique que el **DSN** y la ubicación de los archivos de datos este correcta.
   - En la sección **TEST** de la ventana **QuickTest**, seleccione **Execute Query**
   - En la ventana junto a la selección **Execute Query**, ingrese una sentencia SQL para inserción o actualización.
   - La herramienta **QuickTest Tool** puede realizar múltiples consultas. Cada sentencia de consulta se puede separar con un caracter **<Enter>**.
   - Haga clic en **Test.**

El resultado de esta prueba aparecerá en la sección **RESULT** de la ventana **QuickTest**, si la prueba retorna algún error, revise la información que se cambio en la función **FalOpen, FalReadColInfo, y FalClose**. Si existe algún problema, revise el código con el **Debugger** para ubicar el problema.

Hasta esta parte se ha construido un prototipo funcional de driver ODBC con capacidad de lectura escritura, es decir con este driver se puede realizar lecturas, inserciones, actualizaciones y eliminaciones de datos en tablas.

5.9. Configuración de Orígenes de Datos

Las opciones de configuración para el motor SimbaEngine son colocadas en el archivo de entradas de registro llamado **ODBC.INI**, dentro de **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\odbc.ini.**

Estas entradas contienen el comportamiento especifico del driver y la información del origen de datos. La sección [**ODBC DRIVERS**] contiene una lista de los orígenes de datos disponibles sobre esa máquina. En la sección especifica de orígenes de datos, llamados después de cada entrada en la sección ODBC driver se colocará el driver opcional y/o requerido.

5.9.1. Adición de un driver al registro

La clave de registro **odbcinst.ini** se encuentra bajo **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC**. Aquí se registran todos los drivers ODBC instalados sobre la máquina. Esta información es utilizada por el **ODBC Driver Manager** (en **ODBC32.dll**) y el motor SimbaEngine.

Bajo **HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\odbcinst.ini.** se encuentran los siguientes valores:

Tabla 5.7

Setting	Descripción
[Nombre del Drive]	Crea una nueva clave con el nombre del driver.
Driver=< Ruta completa de la dll del driver con directorio incluido> Ej. Drive=c:\winnt\system32\driver.dll	Crea un nuevo valor de cadena bajo la clave [nombre del driver] para indicar la dll del driver ODB. Si se está ejecutando Windows 95 o 98, esto deberá estar bajo el directorio \system. Si la plataforma es Windows NT esta se encontrará bajo el directorio \system32.
Setup=< Ruta completa de la dll de configuración con directorio incluido > Ej. setup= c:\winnt\system32\driver.cfg	Crea un Nuevo valor de cadena bajo la clave [nombre del driver] para indicar la dll de configuración de driver (localizada en el directorio \system32)
[ODBC Drivers]	
<Nombre de driver>="Installed" Ej. Mi_Driver="Installed"	Crea un Nuevo valor de cadena para el driver bajo [ODBC drivers] para indicar que el driver está instalado en la máquina.

5.9.2. Creación de un origen de datos para el driver

A continuación se describen los pasos necesarios para crear un origen de datos (DSN) para el driver ODBC. Primero se debe editar el archivo **odbc.ini**. Entonces se debe crear un origen de datos de usuario para el driver. A continuación se describen dos formas de realizar esta tarea: Usando **Microsoft ODBC Administrator** (se lo puede

encontrar en el paquete **Microsoft ODBC SDK**), y actualizando el registro manualmente.

5.9.2.1. El archivo Odbc.ini

Esta clave de registro esta bajo **HKEY_CURRENT_USER\SOFTWARE\ODBC.** En esta clave se registra toda la información disponible acerca del origen de datos para el usuario.

Los parámetros del archivo ODBC.ini son usados para configurar un origen de datos. Específicamente los parámetros son usados por:

- El **ODBC Driver Manager** (ODBC32.DLL) y SimbaEngine para información de configuración.
- Para definir orígenes de datos. Note que no es necesario editar directamente los valores de ODBC.INI para definir orígenes de datos; se puede usar el programa de Windows.

5.9.3. Creación de un DSN utilizando ODBC Data Source Administrator

Este programa es fácil de usar y permite al usuario crear orígenes de datos siguiendo algunos pasos. Para usar el ODBC Administrator y crear un origen de datos de usuario, se debe:

1. Desde el menú **Inicio** selecciones **Configuración | Panel de Control | ODBC Data Sources** o desde el directorio de Windows ejecute **odbcad32.exe** inmediatamente aparecerá la interfaz del **ODBC Data Source Administrator**.
2. Haga clic en el tab **DSN de usuario**, luego haga clic en el botón **<ADD>**.
3. Desde la lista de drivers disponibles seleccione el nombre del driver que fue creado en **odbcinst.ini**.
4. Para el **Data Source Name**, ingrese un nombre que identifique al origen de datos.

5. En el **Data Directory** ingrese la ruta completa donde se encuentran los archivos de datos.

5.9.4. Creación de un DSN modificando las entradas del registro manualmente

La actualización del registro tomará algún tiempo, será necesario asegurarse que el siguiente procedimiento se lo realice por completo. Los pasos que se describirán se aplica a todas las plataformas Windows.

Para actualizar manualmente el registro, se debe:

1. Ejecutar el editor de registros **regedit** y luego proceda a ubicar **HKEY_CURRENT_USER\software\ODBC\odbc.ini.**
2. Seleccionar la clave **odbc.ini**.
3. En la opción **Menú**, seleccione **New** | **Key**.
4. Ingrese la nueva clave para el origen de datos y haga clic sobre el botón **OK.** El nombre de la clave aparecerá bajo la clave **odbc.ini.**
5. Seleccione la nueva clave creada. Desde el menú **Edit** , escoja **New|Strting value.**
6. Escriba el valor para el nombre del driver. Luego presione **<Enter>.**
7. Para el **Driver Value**, ingrese la <**ruta>\driver.dll**, donde ruta es el directorio **<dir_instalacion>|SimbaSDK|bin**, luego presione **OK**.
8. Con la recién creada clave para origen de datos iluminada, desde el menú **Edit** escoja **New|String Value**.
9. Escriba **DBQ** como el valor para el nombre y presione **<Enter>**. **DBQ** es el calificador de base de datos, es un término de Simba que describe la ruta a la base de datos que se utilizara cuando se conecte al driver.
10. Para el valor **DBQ** , ingrese la ruta completa de los archivos de datos.
11. Mientras se mantiene posicionado en la clave recién creada, desde el menú **Edit** escoja **New|String Value**.
12. Para el valor de nombre, ingrese la descripción y presione <**Enter**>.
13. Ingrese la descripción del origen de datos y presione **OK**.

14. Seleccione la clave **ODBC Data Sources** bajo la clave **odbc.ini**, entonces escoja **New|String Value** desde el menú **Edit**.
15. Escriba el nombre del origen de datos creado en el paso 4 . Este nombre debe coincidir exactamente con el nombre del paso 4. Haga click en **OK** y presione <**Enter**>.
16. Para el valor de datos, ingrese el nombre del driver creado en el paso 7 y luego haga click en **OK**.

5.9.5. Adicionando entradas a odbc.ini

Es posible crear parámetros propios para utilizarlos por la capa DRM. La DRM Setup DLL podría crear estos parámetros en **odbc.ini** , o crearlos en un archivo de inicialización con esos parámetros por separado, a continuación se presentan algunos ejemplos:

NetworkAccess = On
PageTimeout=600
CollatingSequence= Ascci

Si se necesita información adicional acerca del archivo **odbc.ini**, se puede referir a *Microsoft's ODBC Programmers Reference*.

La tabla siguiente describe las entradas del archivo odbc.ini bajo la clave CURRENT_USER.

Tabla 5.8

PARÁMETRO	DESCRIPCIÓN
[ODBC Data Sources]	Requerido. Aquí se listan los orígenes de datos que el ODBC Driver Manager y otras aplicaciones pueden acceder.
DataSourceName=	Requerido. Un origen de datos. Se puede colocar una descripción opcional al lado derecho del signo =.
[DataSourceName]	Requerido. Debe existir una sección por cada origen de datos en ODBC Data Sources.
Driver=path\drive.dll	Requerido. Indica al ODBC Driver Manager el nombre completo del driver a cargar. Esta es la DLL que se ha creado enlazando las librerías del SimbaEngine con la

	capa DRM.
Description= descripción del origen de datos	Opcional. Es una descripción del origen de datos.
DBQ=databasequalifier	Requerido. Es la base de datos que la capa DRM utiliza; a veces esto se refiere a la ubicación de la base de datos del DRM
UID=UserID	Opcional, depende del almacén de datos o DBMS.
QryPlan=0	Si 1, los planes de ejecución de la consulta son desplegados en la vista en el archivo especificado en QryPlanOutput.
QryPlanOutput=filename.qpf	Para definir donde grabar la información del query Plan Ouput.
FiltSpy=0	Si 1, Push-Down Spy es llamada y la saluda desplegada en la ventana Push-Down Spy.
NoJoinOpt=0	Si 1, SimbaEngine no presionará las uniones a la capa DRM.
NoFiltOpt=0	Si 1, SimbaEngine no presionará los filtros a la capa DRM
NoIndexes=0	Si 1, SimbaEngine no utilizará índices.
MaxColSupport=n	SimbaEngine soporta sobre las 5000 columnas en una tabla simple. Se debe colocar el límite de columnas utilizando esta opción. Si MaxColSupport no está definida, o si n es menor o igual que 256, SimbaEngine coloca por defecto el límite a 256. Si n es mayor que 256 y menor o igual a 256, SimbaEngine coloca el limite igual a n. SimbaEngine permite tablas con un numero mayor a 5000 columnas para ser abiertas y revisadas. Algunas aplicaciones no manejan tablas con 5000 columnas de manera sencilla. El establecer este valor previene a SimbaEngine para que presente mas de MaxColSupport columnas para la aplicación, esto es entregando las primeras MaxColSupport filas, desde la tabla virtual DrmGetTableColumnInfo.
CODEPAGE=n	Especifica el código de pagina a ser usado Ejemplo. 1250- Windows Eastern European 1251- Windows Cyrillic 1252- Windows US
[SimbeEngineOptions]	(Opcional) Cabecera de la sección de opciones para SimbaEngine.
SpyOutput=OutputFileName	Si esta entrada aparece en ODBC.ini las salidas DRM Spy y Push-Down Spy serán direccionadas hacia OutPutFileName; las ventanas DRM Spy y Push-Down Spy no serán desplegadas una alerta de beep ocurre al

	momento de conectarse como un recuerdo de que esta opción esta siendo utilizada. Para deshabilitar esta opción se debe removerlo totalmente no solo eliminar el archivo OutPutFileName.
TempfileDirectory=dir	Especifica un directorio valido en el cual los archivos temp serán creados y destruidos. Esta opción no se establece se utilizara la variable de ambiente Windows TEMP
ViewFileDirectory=dir	Especifica el directorio en el cual SimbaEngine creara archivo de vista si la capa DMR reporta que no quiere guardar las vistas. Si el parámetro ViewFileDirectory no se ha especificado, para el origen de datos en ODBC.ini ODBC utilizará el parámetro DBQ, como el directorio en el cual se crearan los archivos de vistas.

5.9.6. Informacion del Registro

Poco después de iniciar una **DrmBeginSession** durante una llamada ODBC a **SQLAllConnect**, el motor SimbaEngine, llama a la **DrmGetInfo** con el argumento **ulGrBit** establecido a **DRM_infoOdbcIniTokens**.

- Si el DRM requiere infirmación extra del ODBC.ini para el origen de datos actual, para el nombre del parámetro requerido hacia SimbaEngine, a través del buffer referenciado por **PvInfo**.
- El motor SimbaEngine llama al **DrmGetInfo** hasta que la capa DRM retorne **DRM_errNoCurrentRecord**. Pasa los nombres de todos los posibles parámetros que la capa DRM requiere y entonces retorna **DRM_errNoCurrentRecord**. No sera realizaran llamadas futuras. A **DrmGetInfo (DRM_InfoOdbcIniTonkens).**

En Microsoft Windows, esto es opcional para la capa DRM para reportar entradas a esta bandera porque la capa DRM puede llamar al registro de Windows **GetPrivateProfileString** para manejar los tokens privados INI esta bandera se provee para compatibilidad de código con la Api de SimbaEngine si la capa DRM esta trabajando sobre la red, sino, es recomendable retornar **DRM_errNoCurrentRecord** desde la llamada con esta bandera.

Inmediatamente después que todas las llamas **DRMGetInfo** han sido realizadas al driver, durante una llamada a la ODBC **SQLConnect** o **SQLDriverConnect**, SimbaEngine, envía el valor del parámetro a la capa DRM, para una o más llamadas a **DRM_SetIniOption**. Con el argumento **lOption** con el argumento **DRM_SetIniValue**. Los pares parámetro/valor son enviados uno a la vez en formato delimitado por comas como Token, valor, donde Token es la cadena retornada desde una de las llamadas **DRMGetInfo** mencionadas y Valor es el valor de cadena asociada encontrada en el archivo de inicialización.

5.10 . Personalizacion de drivers

Esta seccion incluye información a cerca de optimización de código, incremento de la eficiencia del driver, personalización de mensajes de error y otras características adicionales, esta sección se divide en dos partes:

- Adición de funcionalidades
- Incremento de rendimiento

5.10.1. Adición de funcionalidades

Aquí se describirá como adicionar funciones generales al driver de lectura escritura, implementando funciones para crear y eliminar tablas, y adiciona funcionalidades de bookmark.

SimbaEngine contiene muchas características avanzadas que pueden ser implementadas en el driver.

5.10.1.1. Establecimiento de las banderas DrmGetInfo

Algunas banderas pueden ser configuradas dentro de las banderas **DRMGetInfo** , las mismas que al ser cambiadas permiten personalizar el driver para conocer la

configuración correcta del **DRMGetInfo** se puede referir al SimbaEngine and DRM API Reference .

Las siguientes banderas necesitan ser cambiadas:

1. **DRM_infoDriverName** .- Cambiar esta bandera para retornar el nombre del Driver DLL.
2. **DRM_infoDBMSName** .- Cambiar esta bandera para retornar el nombre de la base de datos que esta siendo accesada.
3. **DRM_infoIsamVersion** .- cambiar esta bandera para retornar la versión del DMS del almacén de datos siendo accesado.

Es posible personalizar luego el driver cambiando otra configuración **DrmGetInfo**.

5.10.1.2. Adición de soporte para otros tipos de datos

Aquí se describe como adicionar soporte para otros tipos de datos en el driver. El prototipo de driver puede soportar cuatro tipos de datos: char, timestamp, integer, y float.

Para adicionar soporte para otros tipos de datos es necesario editar el código en algunos lugares.

1. Abra el archivo **typ.h** y busque por ***"data type supported"***. Esto le ubicara en la estructura **StTypeCatalog**. Aquí se puede encontrar la lista de tipos de datos soportados. Adicione los tipos de datos que necesita en esta lista.
2. Abra el archivo **filMgr.c** y busque por ***"data type supported"***. Aquí se ubicara una sentencia switch, sobre **(pcolumn->columntype)**. Adicione cada nuevo tipo de dato a la sentencia switch. Establezca el valor de **MaxLen** (Longitud máxima) o la máxima longitud de datos esperados por cada tipo de dato adicional.
3. Dentro del archivo **mgr.c** y busque por ***"data type supported"***. Aquí se ubicara una sentencia switch, sobre **(pcolumn->columntype)**. Adicione cada nuevo

tipo de dato a la sentencia switch. Entonces adicione el código para convertir el tipo de datos nativo al tipo DRM apropiado.

4. Abra el archivo **fal.c** y busque por ***"data type supported"***. Aquí se ubicara una sentencia switch, sobre **(pcolumn.columntype)**. Adicione cada nuevo tipo de dato a la sentencia switch. Entonces adicione el código para establecer la longitud escala y posición como sea necesario.
5. Dentro del archivo **fal.c** buscar la función **FalReadColInfo**. Los comentarios en el código describen la **State Machine**. La figura describe el rendimiento de **State Machine**.
6. Dentro del archivo **fal.c** buscar por **DataTypeSupport**, encontrara una sentencia switch sobre **((pcolumn) != DRM_coltypText)**. Aquí se debe adicionar el código para convertir los tipos de datos DRM a su tipo de dato nativo. Para hacer esto es necesario escribir las funciones de conexión.
7. Dentro del archivo **fal.c** buscar por **DataTypeSupport**, encontrara una sentencia switch sobre **(GetColumnType(pcolumn))**. Si es necesario se adicionara el código para calcular la longitud del buffer de caracteres para guardar los datos en memoria. En la figura 5.15 se ve la correlacion entre tipos de datos.

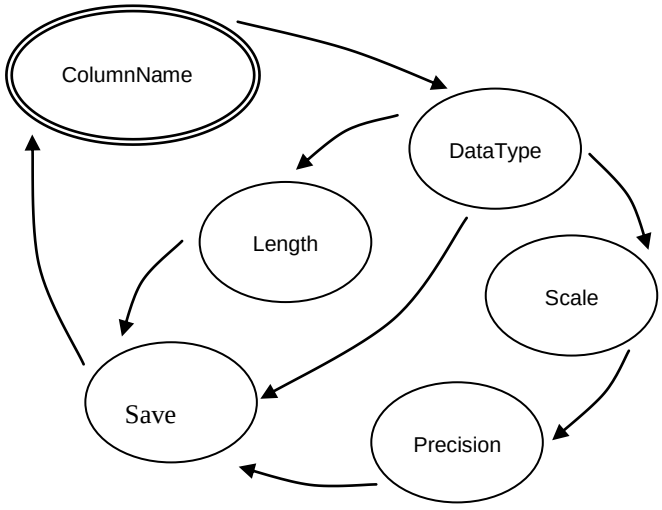


Figura 5.15

5.10.1.3. Control de comparaciones de cadena

La función DRM **DrmStringCompare** es llamada para comparar valores de cadena tales como nombres de tablas, nombres de columnas y valores de cadena en filtros. Esto permite determinar como serán realizadas las comparaciones de cadena en el driver. Se puede definir la secuencia de intercalado de a dos cadenas dentro de **DrmStringCompare**.

5.10.1.4. Personalización de mensajes de error

Utiliza el mismo procedimiento para personalizar los errores de mensaje y adicionar errores de mensaje extendido para el driver.

1. Abra el archivo **simba.rc** en el directorio resourse del proyecto SimbaEngine.
2. Buscar las cadenas conteniendo la palabra "company Name" y reemplazarla con el nombre de su compañía.
3. Buscar por el comentario de cabecera "Custom" para localizar otras áreas que se podían cambiar.
4. Grabe y cierre el archivo.

Los mensajes de error extendidos permiten adicionar por sobre 2000 mensajes de error en el driver para permitir definir errores sobre almacenes de datos específicos.

El proceso a seguir para adicionar mensajes de error extendidos en el driver es el siguiente:

1. Abra el archivo **simba.rc**.
2. Adicione la cadenas de recursos u otros recursos que la implementación del DRM requiera. Las cadenas de recursos pueden tener un valor de **DRM_RES_STR_START** o mayor. Se debe insertar las cadenas de error aquí.
3. Grabar y cerrar el archivo.

El proceso siguiente indica como utilizar los mensajes de error extendidos:

1. Retornar el numero de error extendido (ext err #), desde la función DRM.
2. Cuando **DrmGetExtErrStr**, es llamada con el ext err #, esta retorna la cadena asociada desde los recursos.

5.10.1.5. Adicion de bookmarks

Un bookmark es un marcador de posición en el código. Aquí se describirá como operan en un driver.
Para ordenar los resultados de datos (por ejemplo: con la sentencia orden by y group by) que contienen una columna **LONGVARBINARY** o **LONGVARCHAR**, la capa DRM preservará un bookmark por cada fila. Para el resto de resultados de datos solamente el ultimo boormark necesita ser preservado.

SimbaEngine asume que un bookmark es persistente para cada tabla durante todo el tiempo que pertenece abierta. Esto puede ser un problema cuando dos aplicaciones están accediendo a la misma tabla.

Por ejemplo, las aplicaciones 1 y 2 están accediendo a la misma tabla. La aplicación 1 realiza una sentencia select sobre la tabla, que es una tarea de dos pasos: un **SQLExecute** y un **SQLGetData**. Durante el tiempo que la aplicación esta procesando **SQLExecute**, la aplicación 2 puede acceder y actualizar la tabla. Durante la actualización de la tabla, el bookmark dejado por la aplicación 1 puede perderse o aparecer invalido. Cuando la aplicación 2 entorna a la tabla para mirar el bookmark y realizar el **SQLGetData**, el bookmark no puede ser encontrado y los datos pueden ser agrupados incorrectamente.

Las funciones que se necesitan implementar son: **DrmGetBookmark DrmGoToBookmark**.

5.10.1.6. Adición de soporte para creación y eliminación de tablas.

Para adicionar código al prototipo de driver del prototipo de driver de lectura - escritura que permita eliminar y crear tablas, se debe adicionar código a las siguiente funcione DRM:

- DrmCreateTable
- DrmDeleteTable
- DrmCreateColumn

Se puede encontrar mas información a cerca de estas funciones refiriendo se a la SimbaEngine and DRM API Reference Guide.

5.10.2. Como mejorar el funcionamiento del driver odbc

En esta parte se indicara la manera d como incrementar la eficiencia del driver ODBC incrementado soporte para índice optimizando el código.

5.10.2.1. Incremento del soporte para índices

Si el almacén de datos que esta accediendo utilizando índice es posible adicionar soportes para estos en el DRM. Esto incrementará la velocidad en la que se puede recuperar datos con consultas que incorporan filtros. Es necesario incrementar 5 funciones:

- DrmGetTableIndexInfo
- DrmSetTableIndex
- DrmMakeKey
- DrmSeek
- DrmStringCompare

Si es que acceso el almacén de datos que se esta accediendo ya maneja índice no es necesario habilitar la creación y eliminación de los mismos, entonces no se debe implementar **DrmCreateIndex** y **DrmDeleteIndex**.

En caso de que se desee habilitar estas funcionalidades se debe implementar dos funciones:

- DrmCreateIndex
- DrmDeleteIndex

5.10.2.2. Como hacer el código más eficiente

El código provisto en el QST es simple y fácil de entender. Es posible adicionar mejoras en el rendimiento optimizando el código para mejorar la interacción con el almacén de datos.
Aquí se presentan algunas formas para darle mas eficiencia al código. El código del QST es de propósito general y permite adaptarlo aun sin números de almacenes de datos, es decir que se puede mejorar el rendimiento haciendo cambios específico para un almacén de datos en particular.

Dos componentes del QST pueden no ser necesarios para el driver y el almacén de datos, entonces este código podrían ser removido, estos componentes son: **Record Set Manager** y el **Virtual Table Manager**.

El RSD contiene código para separar memoria que ayuda en la recuperación de datos, el código RSD puede ser removido y cualquier dato puede ser directamente accesado desde el buffer API.

La VTM contiene código para crear tablas virtuales de la información del diccionario de datos. Cualquier dato que ha sido extraído usando el VTM el cual no cambia después del tiempo de compilación, puede ser directamente extraído sin generar una tabla virtual. Por ejemplo, el tipo de dato recuperado, cuando el tipo de dato no cambio durante el tiempo de ejecución, podría no requerir el uso de la VTM.

5.10.2.3. Seguridad de datos en ambiente multiusuario

Si los datos son accesados por dos copias diferentes del driver al mismo tiempo, estos son aislados por el mecanismo interno del DBMS, es decir, los mismos datos no son accesibles por otros mientras otro usuario esta editándolos. Si el almacén de datos no tiene un modelo de bloqueo o aislamiento es posible implementar uno para asegurar que los datos no se corrompan. Este modelo de bloqueo podría especificar aislamiento sobre ciertos niveles del almacén de datos, por ejemplo se podría bloquear los datos a nivel de tabla, a nivel de pagina, o a nivel de fila.