Depto. COMPUTACION
INFOTECA
F. C. E. N. - U. B. A.

Tesis de Licenciatura

Miguel Santos
L.U. 1316/88
1994-1997

TESIS DE LICENCIATURA

Índice:

Prólogo..... 3

Introducción..... 4

Capítulo 1: ODBC..... 6
- Arquitectura..... 7
- Conformance Levels..... 9
- Funcionalidad nativa y extendida..... 10
- Gramática SQL..... 17
- Entonces, qué debe hacer una aplicación ?..... 19
- ODBC y VQL..... 20

Capítulo 2: OLE..... 22
- Linking and Embedding..... 22
- Arquitectura..... 22
- OLE Automation..... 24
- OLE y VQL..... 25

Capítulo 3: SQL Front Ends para Windows..... 29
- Tools..... 29
- Soporte a RDBMS's..... 29
- Objetos..... 30
- Resumen Comparativo..... 30
- Front Ends y VQL..... 31

Capítulo 4: RDBMS's..... 32
- IBM DB2..... 32
- Oracle..... 33
- Microsoft SQL Server..... 33
- Resumen comparativo..... 34
- RDBMS's y VQL..... 34

Capítulo 5: Visual Query Language..... 35
VQL Custom Query..... 35
VQL Preset Query..... 37
Preset Setup..... 37
Preset Runtime..... 43
Arquitectura y Diseño..... 44
Posibles Agregados y Mejoras..... 47

Conclusión..... 48

Apéndice A: Acrónimos..... 49

Apéndice B: Instalación..... 50

Apéndice C: Demo..... 51

Apéndice D: Demo..... 52

Apéndice E: Guía a los programas fuente..... 54

Bibliografía..... 56

Prólogo:

El trabajo de tesis que presento a continuación sienta sus bases en el estudio de los estándares y tendencias de desarrollo para aplicaciones de bases de datos cliente-servidor utilizando Microsoft Windows como front-end (cliente) y alguna fuente de datos, por ejemplo un RDBMS (Relational DataBase Management System) montado sobre Unix o MS Windows NT por ejemplo, como back-end (servidor). A lo largo del trabajo efectúo diversos análisis sobre los diferentes temas involucrados, culminando el mismo mediante la presentación del desarrollo de una herramienta nombrada Visual Query Language, para acceso visual a bases de datos, cuya idea principal es la generación en forma visual de consultas SQL (Structured Query Language) que luego son efectuadas a las distintas fuentes de datos. Dicho desarrollo involucra el estudio de los estándares de programación visual bajo MS Windows, como también de acceso a bases de datos, ODBC (Open DataBase Connectivity), SQL, de comunicación entre procesos en Windows, OLE (Object Linking & Embedding), DDE (Dinamic Data Exchange) o de utilización protocolos de comunicaciones y redes, IPX/SPX, TCP/IP, NETBEUI. Por otra parte, presentaré estudios de mercado y análisis herramientas, aplicaciones y tendencias en los usuarios, todos ellos expuestos a lo largo del desarrollo del trabajo.

Introducción:

Con el correr de los años, las aplicaciones de base de datos, así como también tantas otras, han sido soportadas por diferentes modelos de computación o procesamiento. En un principio no tan lejano, aún existen implementaciones de este modelo, las aplicaciones de base de datos eran totalmente centralizadas, más aún, era el aplicativo el que se encargaba de acceder a los datos directamente, no existía lo que hoy denominamos motor de base de datos. Avanzando en la historia se dio un paso muy importante, se empezó a hablar de encapsulamiento, con lo cual aparecieron primero bibliotecas de rutinas para acceder a distintas implementaciones de base de datos y más tarde lo hicieron los administradores de base de datos, procesos encargados de hacer todas las operaciones referentes a la base de datos. Sin embargo todavía continuaba vigente el modelo centralizado, es decir, tanto la aplicación como el motor de base de datos corrían en un computador central y único. También se evolucionaba en otros niveles, aparecían las bases de datos relacionales. Luego, con el devenir de las redes de computadoras y los diversos modelos de computación distribuidos, nació el concepto de cliente-servidor, modelo del cual hablaré en profundidad más adelante, el cual aplicado a base de datos arrojó excelentes dividendos, dado que combinaba las ventajas de modelos anteriores, mantenía la seguridad del modelo centralizado e incorporaba la performance del modelo distribuido, acceso centralizado y procesamiento distribuido.

Mientras tanto, por otra parte la informática continuaba avanzando por otros terrenos, entre ellos, el de la interfase con el usuario final, ya habían nacido los conceptos de sistema amigable e interfase gráfica, GUI de ahora en adelante. Aparecían en el mercado distintas implementaciones de GUI, entre ellas Microsoft Windows, fue tal el suceso que pronto se convirtió en un fenómeno universal, llevando a Microsoft a ser una de las empresas más poderosas en el campo de la informática.

Luego de esta breve reseña histórica llegamos a donde queríamos, por qué no combinar la potencia de los RDBMS con las bondades de una interfase gráfica? Un *servidor* de base de datos accedido desde un *cliente* Windows por ejemplo? La respuesta está a la vista, es el standard de estos días. Es por ello que resulta necesario para la toma de decisiones, poder acceder a los datos que generan las aplicaciones libremente, efectuar consultas no planificadas por la aplicación, mejor aún si podemos hacerlo en forma visual. Aquí nace **Visual Query Language**, una herramienta para la consulta visual de bases de datos. VQL está presentado en dos versiones **Custom** y **Preset**, Custom está dirigida hacia los usuarios experimentados o desarrolladores de software, que podrán acceder a todas las tablas de la base de datos, en forma totalmente libre. Preset, en cambio, está orientada al usuario final no necesariamente experimentado, conceptuado en forma de add-on para cualquier sistema de aplicación bajo Windows. Preset a su vez está compuesto por dos módulos, **Runtime**, que es el encargado de proveer la interfase, generar las consultas y efectuarlas, tomando la configuración desde un programa de inicialización .VQL generado por **Setup**, el otro módulo, donde el desarrollador de software genera el .VQL, que contiene la información necesaria para acceder a la base de datos, qué gráfico representa qué tabla, qué tablas son visibles y cuales no, los nombres para los campos, etc.

A continuación expondré un trabajo en el cual ahondaré sobre los conceptos descritos en esta breve introducción, además de analizar otros que irán surgiendo durante el desarrollo. El trabajo está organizado en forma de capítulos donde se analizan los temas estudiados, con el objeto de que el lector pueda saltear los temas que ya conoce, de todas formas, una sección de cada capítulo está destinada a exponer la relación que existe entre el tema pertinente y la herramienta en cuestión, Visual Query Language.

Capítulo 1: ODBC

El objetivo principal de Open Database Connectivity (ODBC), es proveer acceso a bases de datos heterogéneas, de una forma sencilla y universal.

Microsoft ODBC fue diseñado a partir de los estándares de la industria. El núcleo de ODBC, es decir su API (Application Programming Interface), está basado en la especificación de interfase X/Open and SQL Access Group Access Level. Esta especificación provee funciones de conexión y desconexión de fuentes de datos, setup y ejecución de consultas, descripción y fetch de los resultados, procesamiento de errores, procesamiento de transacciones, etc.

La gramática SQL de ODBC, está también basada en el standard X/Open and SQL Access Group CAE (1992). Esta especificación provee la sintaxis para crear y borrar tablas, views e índices, insertar, modificar y borrar campos, otorgar y revocar permisos, efectuar consultas y modificar la estructura de las tablas.

ODBC extiende ambos estándares aumentando su funcionalidad, acorde a los niveles de los motores de bases de datos relacionales de hoy en día. Como por ejemplo, funciones para usar diferentes modelos de conexión, establecer atributos sobre la conexión y las formas de procesamiento de las instrucciones como procesamiento asincrónico, utilizar datos de gran tamaño fuera del standard, traer el resultado sólo en bloques, scroll random de los registros del resultado, información sobre el driver o la fuente de datos, etc. Asimismo, en cuanto a la gramática de SQL, ODBC agrega instrucciones en modo batch, nuevos tipos de datos como long character, binary, date, time and timestamp, funciones escalares de procesamiento de strings, funciones numéricas, outer joins y ejecución de *stored procedures*.

De aquí en más describiré la funcionalidad de ODBC en detalle, especificando en cada caso si se trata de funciones nativas parte del núcleo o de extensiones. Para una descripción completa recomiendo acudir al ODBC Programmers Reference, o bien acceder mediante Internet a *http://www.microsoft.com/*msdn.

Arquitectura.

Una aplicación ODBC tiene cuatro componentes principales:

- La aplicación propiamente dicha.

  Puede ser una aplicación cualquiera, por ejemplo corriendo bajo Microsoft Windows, que realiza llamadas a funciones de ODBC para interactuar con un motor de base de datos. La aplicación debe conocer el ODBC API.

- Driver Manager.

  En general en forma de DLL, provee acceso a los drivers de ODBC. El driver manager se encarga de cargar los drivers de ODBC y rutear las funciones efectuadas por la aplicación al driver correspondiente. También hace control de errores y procesa algunas funciones de ODBC extendido.

- Driver.

  También en forma de DLL, procesa las funciones ODBC, es decir, se conecta con la fuente de datos, traduce las sentencias SQL y las envía a la fuente de datos, luego recibe la información correspondiente al resultado y la retorna a la aplicación. En el caso de que la fuente de datos no conozca SQL, es el driver el encargado de resolver las sentencias SQL. Por otra parte, el driver debe conocer los protocolos de transporte de la red para enviar y recibir los datos.

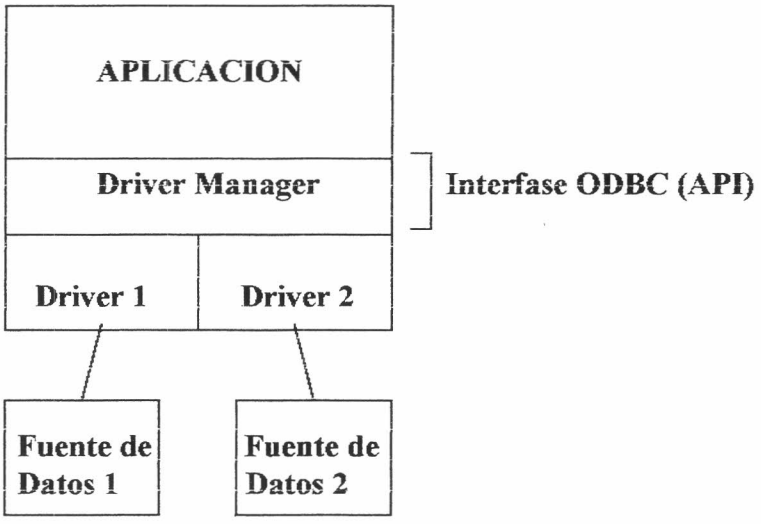
- Fuente de datos.

  Es la que contiene efectivamente la información que el usuario desea acceder, esta información está administrada por un DBMS.

  Ejemplos:

  Oracle System 7 (RDBMS) montado sobre Unix.
  MS SQL Server (RDBMS) montado sobre Windows NT.
  IBM DB2/2 (RDBMS) montado sobre MVS.

Cabe destacar que una aplicación puede acceder a más de una fuente de datos a la vez, las que pueden utilizar el mismo DBMS o no.



Tipos de Drivers.

ODBC define dos tipos de Drivers:

- Single Tier

  El driver no solo implementa el ODBC API (más precisamente el ODBC SPI o Service Provider Inteface), sino que además actúa como motor de SQL, es decir parsea, valida, optimiza la consulta y accede a los datos en forma directa. Típicamente son single tier los drivers utilizados para manipular archivos ISAM, por ejemplo Access, FoxPro o Paradox o bien archivos de texto, planillas de cálculo, etc.

- Multi Tier

  El driver implementa el ODBC SPI, en este caso transformando los calls de forma que los DBMS puedan interpretarlos adecuadamente, de la misma forma que debe manejar el data stram protocol utilizado por el DBMS para el inercambio de información. Ejemplos de este tipo de driver son los drivers de Oracle y SQL Server. Este tipo de configuración se denomina multi tier pues es posible que el driver deba comunicarse con algún Gateway, formandose en este caso una arquitectura de tres capas.

Conformance Levels

Así como ODBC define un API, define también la gramática SQL, basada en el standard X/Open and SQL Access Group CAE (1992). Tanto la gramática como el API están organizadas de forma jerárquica con varios niveles, los llamados *conformance levels*.

En cuanto a la gramática tenemos:

- Minimum SQL
- Core SQL
- Extended SQL

y en cuanto al API, cabe la aclaración que los niveles de gramática y API no son biyectivos respectivamente,

- Core API
- Level 1 API
- Level 2 API

Entonces, los drivers deben *conformar* cierto nivel de API y cierto nivel de gramática, por ejemplo el driver de SQL Server conforma el API Nivel 1 y la gramática SQL mínima.

Para más infomación sobre los *conformance levels* acudir al ODBC Programmers Reference.

Funcionalidad: Nativa y Extendida.

Como siempre, aquí nos encontramos con el problema de utilizar funciones extendidas, nuestra aplicación no será compatible con todos los sistemas ODBC disponibles. A su vez los desarrolladores de drivers deben hacer un análisis sobre qué funciones deberán soportar, como en general los drivers los provee el fabricante de RDBMS, la funcionalidad extendida va de acorde con las funciones del mismo.

Los drivers, según la especificación deben soportar por lo menos todas las funciones nativas, y por supuesto todas las que puedan del API extendida. Los drivers para DBMS que no soporten SQL, deben soportar por lo menos un subconjunto coherente de las funciones nativas.

Conexión a las fuentes de datos.

Antes de efectuar cualquier operación sobre los datos, la aplicación debe conectarse a la fuente de datos. Las fuentes de datos deben ser configuradas por los usuarios antes de ejecutar sus aplicaciones ODBC. ODBC ofrece varios modelos de conexión, siendo el más común de ellos el interactivo, donde se consulta al usuario la información necesaria para la conexión, además de mostrar las fuentes de datos disponibles.

Configuración de las fuentes de datos (Nativa).

Como se dijo antes, los usuarios deben configurar las fuentes de datos. Pueden hacerlo con un programa de administración que efectúa calls específicos a cada driver solicitando que datos deben pedirse al usuario para configurar esa fuente de datos. Por ejemplo, SQL Server de Microsoft, necesita entre otras cosas la dirección de red y el nombre de la librería de red para conectarse a un fuente de datos SQL Server. El programa de administración guarda los datos solicitados en un programa de inicialización de Windows (.INI). Por supuesto, los desarrolladores de aplicaciones pueden escribir sus propios programas de configuración en vez de utilizar el que viene en el ODBC SDK.

Conexión a una fuente de datos (Nativa).

Conectarse a una fuente de datos puede ser tan simple como abrir un archivo local o tan complejo como conectarse a una red, loggearse en un server, loggearse en un DBMS y seleccionar una base de datos. El camino es el siguiente, la aplicación pasa al Driver Manager el nombre de la fuente de datos, y opcionalmente el nombre de usuario y la password. El Driver Manager utiliza el nombre de la fuente de datos para extraer el driver correspondiente del archivo de inicialización de Windows (ODBC.INI), luego carga e inicializa la DLL del driver. A continuación el driver recoge la información restante del archivo de inicialización y se conecta con la fuente de datos.

Desconexión de la fuente de datos (Nativa).

Para desconectarse de una fuente de datos, una aplicación especifica que conexión debe cerrar. El driver se desconecta efectuando logout de los sistemas remotos, cerrando archivos y liberando recursos según corresponda. Luego, el Driver Manager descarga la DLL del driver en caso de que no tenga otras conexiones activas.

Información adicional para la Conexión (Extendida).

Los drivers, a veces requieren información adicional, por ejemplo como ya se dijo, una dirección de red. El driver puede proveer sus propios diálogos para conseguir dicha información o bien recibir de la aplicación un string con un formato determinado, que luego se encargará de parsear.

Pedido de información al usuario (Extendida).

Por ejemplo, si la aplicación no especifica el nombre d la fuente de datos, el driver puede obtener la lista de las fuentes del archivo de inicialización y mostrarla al usuario para que escoja.

Opciones de conexión (Extendida).

Cada driver, que posiblemente lo herede de su correspondiente DBMS, puede soportar opciones de conexión. Ejemplos típicos son sólo lectura, time outs, traces.

Ejecución de sentencias SQL.

Ejecución de una sentencia SQL (Nativa).

Para ejecuta una sentencia SQL, la aplicación pasa el string al driver, que en el caso de ser un driver de SQL DBMS, traduce el string al SQL nativo del DBMS si es que lo necesitara, y en el caso de ser un driver de un DBMS no SQL, procesa y ejecuta la sentencia. Notar aquí que la aplicación puede explotar funcionalidad específica del DBMS, para lo cual indica al driver que mande la sentencia sin traducirla, por supuesto esto indica que la aplicación será dependiente del DBMS utilizado.

Una sentencia SQL debe ser preparada antes de su ejecución, debe ser parseada, optimizada y compilada. Generalmente ésta es tarea del DBMS. Luego de que el DBMS prepara la sentencia, devuelve un identificador al driver, que pasa este identificador nuevamente al DBMS cuando debe ejecutarse la sentencia. La razón de dicho funcionamiento, es que una aplicación puede indicar que la sentencia sea preparada y ejecutada en un solo paso o en dos. La aplicación debería separar dichos pasos si va a ejecutar la misma sentencia más de una vez.

Parámetros SQL (Nativa)

Las sentencias SQL pueden contener parámetros. En el caso de que una aplicación asocie variables locales a dichos parámetros, el driver debe guardar las direcciones de dichas variables. La aplicación por otro lado debe indicar los tipos de cada parámetro, para que luego el driver los convierta si es necesario y pasa los valores al DBMS. Es posible que una aplicación quiera enviar una sentencia más de una vez con diferentes valores, con lo cual puede mandar a preparar la sentencia y luego enviarla para su ejecución tantas veces como sea necesario con distintos valores. Un ejemplo de sintaxis podría ser preparar la sentencia INSERT INTO AUTORES (ID, NOMBRE) VALUES (?, ?).

Arrays de parámetros (Extendida)

La aplicación puede también asociar arrays a los parámetros de una sentencia SQL. El driver recibe la dirección de memoria del array, su tipo, su longitud y los tipos de los parámetros de la sentencia mediante calls separados. Luego, cuando la aplicación envía la sentencia para su ejecución, el driver se encarga de acceder a los datos del array, convertir los tipos si es necesario y finalmente, dependiendo de la capacidad del DBMS, manda el array como parámetro de la sentencia al DBMS o bien particiona la misma en tantas sentencias como elementos tenga el array.

Parámetros en tiempo de ejecución, diferidos (Extendida).

Es posible, mediante esta función, que la aplicación pase sus parámetros al driver después de haber mandado la sentencia a ejecutar. En vez de asociar variables a los parámetros, la aplicación asocia flags, que son recibidos por el driver. Luego, cuando la aplicación envía la sentencia a ejecutar, el driver chequea los flags y solicita los datos correspondientes a la aplicación (mecanismo de callback), que ahora sí, pasa al driver los datos. Esto puede ser bastante útil a la hora de mandar grandes bloques de información por partes, como por ejemplo un bitmap.

Opciones Extendidas (Extendida)

Las opciones extendidas disponibles son por una parte la ejecución asincrónica de sentencias, la anulación de una ejecución asincrónica antes de su finalización y los timeouts para las sentencias, es el driver el encargado de mantener los timers y manejar la conversación con el DBMS.

Procesamiento de Transacciones

El modelo transaccional que usa ODBC es bastante simple, las transacciones se refieren a una fuente de datos a la vez, no existen transacciones anidadas, no requieren inicio explícito y sólo puede hacerse con ellas *commit* o *rollback*. En general son manejadas por el driver, que se ajusta a la capacidad del DBMS.

Inicio, commit y rollback de transacciones (Nativa).

Cuando un DBMS requiere inicio explícito de transacciones, el driver inicia una nueva transacción cada vez que:

- Una aplicación manda a ejecutar una sentencia SQL,
- esa sentencia puede estar en medio de una transacción, y
- no hay ninguna transacción pendiente.

Para hacer *commit* o *rollback* de una transacción, ODBC provee funciones, que otra vez son dependientes del DBMS, si el DBMS las implementa, no tiene más que invocar la función correspondiente, si no, debe implementarlas el mismo.

De nuevo, para obtener la mayor interoperabilidad, la aplicación debe invocar las funciones ODBC en vez de hacerlo explícitamente ejecutando funciones del DBMS.

Opciones Extendidas (Extendida).

La aplicación puede indicar al driver el modo de procesamiento de transacciones para una conexión ODBC, *manual-commit* o *auto-commit*. En *manual-commit*, la aplicación debe indicar al driver cuando hacer el commit, en *auto-commit*, el driver da commit a cada sentencia SQL que termina correctamente.

Existe también la posibilidad de especificar cuando una transacción permite concurrencia, es decir, que varias transacciones accedan simultáneamente a los datos que ésta utiliza.

Procesamiento de Resultados.

Típicamente cuando una aplicación ejecuta un SELECT, la fuente de datos retorna un resultado o *result set*, que debe ser procesado de alguna manera.

Procesamiento del result set (Nativa).

Antes de obtener un registro del resultado, la aplicación debe asociar variables locales a cada campo correspondiente al result set. El driver es el encargado de mantener las direcciones de esas variables, para luego copiar allí los datos. Cuando la aplicación pide un registro del resultado, el driver descarta el registro que tiene y pide el próximo al DBMS, luego hace la conversión de tipos si es necesario y copia los valores a las variables de la aplicación. Es responsabilidad de la aplicación indicar cuando deja de utilizar el result set, allí el driver cierra el result set y descarta los datos que hayan quedado pendientes.

Cursores (Nativa).

Un cursor es un puntero utilizado por el driver para mantener la posición dentro del result set. Cada cursor apunta a un único registro del result set y puede moverse solo hacia adelante un registro por vez. En otras palabras, la aplicación solo puede pedir un registro a la vez y moverse en el result set de a un registro hacia adelante.

L aplicación puede indicar el nombre del cursor al driver o dejar que este utilice un nombre generado, por otra parte mediante el nombre del cursor, puede borrar o actualizar el registro apuntado.

Obtención de datos de campos no asociados (Extendida).

Utilizando esta opción, la aplicación no necesita crear variables para todos los campos del result set, puede manejarlo dinámicamente, indicando al driver que asocie un campo por vez de izquierda a derecha, cada vez que lo hace, el driver devuelve un código de retorno que indica que hay más datos disponibles, es decir, se genera una conversación entre aplicación y driver para coordinar el pasaje y así ahorrar espacio. Esta función es útil para trabajar con bloques grandes de datos, por ejemplo bitmaps o sonido.

Opciones Extendidas del Result Set (Extendida).

La aplicación puede especificar el máximo tamaño en bytes o la máxima longitud en registros del result set.

Uso de arrays para almacenar resultados (Extendida).

Aquí también se puede especificar al driver que deje los resultados en arrays en vez de variables escalares, obviamente la aplicación debe indicar las direcciones de memoria, los tipos y los tamaños. Asimismo, el driver debe hacer las conversiones de tipos necesarias al recibir los datos del DBMS.

Cursores (Extendida).

Un cursor puede definirse como *scrollable*, es decir, puede moverse hacia adelante o hacia atrás en el result set, además de tener acceso directo a una determinada posición o bien movimiento relativo desde una posición hacia otra. La aplicación puede especificar además si el cursor es *dynamic, keyset-driven* o *mixed.* Cuando se define un cursor como dynamic, el orden de los registros y los valores de los campos reflejan el estado real en el DBMS, es decir, se detectan los cambios de otros usuarios en la fuente de datos, esta propiedad suele llamarse *autorefresh*. Por otro lado, si el cursor es keyset-driven, el driver utiliza claves para acceder a los registros, lo cual sólo permite detectar cambios de borrado o agregado de registros, además de ser fijo el orden de los registros una vez que se crea el cursor. Por último, un cursor es mixed cuando es keyset-driven para una parte del result set y dynamic para la otra.

Múltiples result sets (Extendida).

En el caso de utilizar un *store procedure*, un *batch* de SQL, un grupo de sentencias que contengan más de un SELECT, o un SELECT con arrays como parámetros, pueden generarse múltiples result sets. La aplicación puede procesar los result sets secuencialmente, siendo el driver el que se encarga de tener los result sets disponibles cuando se los necesita.

Formato de los datos.

Es perfectamente posible, y lo más común que la aplicación y el DBMS utilicen diferentes formatos para almacenar la información, inclusive puede requerirse también traducción a nivel de tipos . Como parte de la funcionalidad extendida, una aplicación puede indicar el nombre de la DLL que va a ser utilizada para la traducción, recordemos que es el driver el que debe efectuar los calls a dicha DLL para traducir parámetros, resultados, nombres, etc.

La especificación de ODBC no incluye DLL's de traducción, estas son responsabilidad de los desarrolladores de aplicación o de drivers.

Manejo de errores

Dentro de la funcionalidad nativa, ODBC provee procesamiento de los errores. Una vez procesada una sentencia SQL, el driver retorna un código a la aplicación que indica si la sentencia:

- Se ejecuto con éxito.
- Se ejecutó con un error no fatal.
- Canceló.
- El result set fue vacío.
- Faltan parámetros.
- Se está ejecutando asincrónicamente.

Cuando la sentencia termina con error, el driver guarda información extra proveniente del DBMS describiendo el error. Que puede ser consultada por la aplicación utilizando SQLSTATE como está descripto en la X/OPEN and SQL Access Group SQL CAE 1992.

Acceso a información del driver.

ODBC provee funcionalidad extendida para acceder a información de driver, sobre la fuente de datos, las conexiones, sentencias, result sets, etc.

Información sobre el Catalog (Extendida)

ODBC provee funciones para obtener información sobre el *Catalog* o estructura de la base de datos del sistema, provista por la fuente de datos. Se puede obtener información acerca de:

-Tablas: identificador, nombre, dueño, tipo, claves, índices, etc.
- Campos: nombre, tipo, longitud, privilegios, etc.
- *Procedures*: identificador, nombre, dueño, parámetros, descripción, etc.

Información sobre el driver y la fuente de datos (Extendida)

Muchos de los drivers y fuentes de datos no soportan todas las funciones de ODBC, por lo tanto deben existir formas de conocer las funciones que están disponibles de manera que la aplicación pueda actuar de distintas formas según el caso.

En general los drivers brindan información sobre:

- Funciones ODBC soportadas.
- Tipos de datos soportados.
- Gramática SQL soportada.
- Nombre y versión del driver.
- Nombre y versión de la fuente de datos.
- Límite de conexiones activas.
- Comportamiento de los cursores.
- Esquema de transacciones soportado.
- Máxima longitud de los nombres.
- *Case sensivity*, patrones de búsqueda, etc.

En todos los casos si el DBMS no puede dar la información, lo debe hacer el driver.

Información sobre conexiones (Extendida)

La aplicación puede consultar al driver sobre alguna conexión activa, por ejemplo, si la base de datos fue abierta *read-only*, en forma exclusiva, etc.

Información sobre sentencias SQL (Extendida)

Mediante esta función la aplicación puede saber:

- La sintaxis nativa de una sentencia, luego de traducida por el driver.
- La cantidad de parámetros de una sentencia.
- La cantidad de registros afectada por una sentencia.
- Las opciones extendidas de la sentencia que se está ejecutando.

Información sobre el result set (Extendida)

Si una aplicación no conoce la estructura del result set de una sentencia, por ejemplo si deja al usuario introducir la consulta manualmente, puede obtener la información de los campos del resultado, como su nombre, tipo y longitud.

Gramática SQL

ODBC define un standard para la gramática SQL, que luego cada driver debe traducir a la gramática nativa de su correspondiente DBMS. La gramática se divide en tres categorías:

Gramática Mínima

Es un subconjunto del la gramática Nativa, especificada especialmente para proveer un mínimo de funcionalidad a aquellas fuentes de datos no SQL, por ejemplo XBASE, planillas de cálculo, etc. Obviamente es el driver el encargado de resolver las sentencias SQL.

La gramática mínima contiene:

- Sentencias para crear y eliminar tablas.
- Seleccionar, insertar, borrar y modificar registros en forma simple.
- Expresiones simples.
- Datos de tipo caracter.

Gramática Nativa.

Corresponde, como ya se dijo al standard X/Open and SQL Access Group SQL CAE (1992). Todos los drivers para SQL DBMS deben soportar esta gramática.

En general contiene:

- La gramática mínima.
- Sentencias para crear, eliminar y modificar estructura de tablas, crear y eliminar índices, views, manejo de privilegios, opciones de selección, inserción y borrado de registros de mayor complejidad.
- Expresiones con *subquerys* y funciones como SUM, AVG, etc.
- Más tipos de datos.

Gramática extendida.

En general provee:

- La gramática nativa.
- *Outer Joins.*
- Expresiones con funciones escalares y formatos fecha, hora y *timestamp.*
- Más tipos de datos.
- Sentencias Batch.
- Store Procedure Calls.

Por último, repasemos qué debe hacer una aplicación

En esta sección comentaré brevemente los pasos a seguir para acceder a un DBMS desde una aplicación mediante ODBC:

- Conectarse a la fuente de datos. Típicamente es necesario proveer información como el usuario, password y nombre de la base de datos.
- Poner el string conteniendo la sentencia SQL en un buffer.
- Ejecutar una función del API para mandar la consulta, como por ejemplo SQLExecDirect con el buffer con la sentencia como parámetro.
- El DBMS procesa la sentencia y puede:
    1. Retornar resultados.
    2. No retornar, como un INSERT.
    3. Retornar un mensaje de error
- Si el caso es por ejemplo 1, la aplicación podría recibir los resultados en un cursos, consultar sobre los atributos del resultado, como columas o cantidad de registros, y finalmentee fetchear los resultados y procesarlos.
- Por ultimo, la conexión a la base de datos debe ser cerrada.

ODBC y VQL.

Visual Query Language fue desarrollado en Microsoft Visual Basic Versión 3.0, corre bajo MS Windows y mediante ODBC accede a fuentes de datos, en particular, centraré esta sección en la relación que existe entre ambos, la fuente de datos que usaré como ejemplo es Microsoft SQL Server.

Configuración del Cliente

Existen dos archivos de inicialización de Windows, que deben estar el directorio de windows y que contienen la información sobre los drivers y servers instalados, ODBCINST.INI y ODBC.INI. Mostraré un análisis de los mismos utilizando ejemplos.

ODBCINST.INI contiene información sobre los drivers de ODBC instalados:

[ODBC Drivers]
SQL Server = Installed

[SQL Server]
Driver = C:\WINDOWS\SYSTEM\SQLSRVR.DLL
Setup = C:\WINDOWS\SYSTEM\SQLSETUP.DLL

La sección [ODBC Drivers] indica al driver manager los nombres de los drivers instalados, mientras que la sección [SQL Server] le indica los nombres de las dll's del driver para SQL Server y del programa de Setup.

ODBC.INI contiene la información sobre las fuentes de datos accesibles y sus correspondientes drivers, fundamentalmente, indica para una fuente de datos, qué driver utilizar para acceder a qué server.

[ODBC Data Sources]
Demo=SQL Server

[Demo]
Driver = C:\WINDOWS\SYSTEM\SQLSRVR.DLL
Description = SQL Server el Server Demo
OemToAnsi = No
Network = dbnmp3
Address = \\demo\pipe\sql\query

La primera sección indica los nombres de las fuentes de datos, cuyos valores deben corresponderse con los definidos en el ODBCINST.INI, que más adelante tendrán una sección con la mínima información para acceder a dicha fuente.

DLL's para ODBC y Drivers.

Las siguientes DLL's deben estar en el path, el directorio WINDOWS\SYSTEM o en el directorio donde corre la aplicación.

ODBC.DLL:	Driver Manager.
ODBCINST.DLL:	Driver Setup.
VBDB300.DLL:	Capa de programación de Visual Basic .
SQLSRVR.DLL:	Driver.
SQLSETUP.DLL:	Driver Setup.
DBNMP3.DLL:	Named Pipes, requerimiento de SQL Server.

Información específica para SQL Server.

En el caso de Microsoft SQL Server, es necesario agregar algunos stored procedures al Catalog en función de que el server pueda soportar ODBC. Para ello debe ejecutarse el procedimiento INSTCAT.SQL, que instala los nuevos procedimientos en el server.

Notas sobre diseño y programación.

Dado que Visual Query Language es esencialmente una herramienta de consulta, he tratado de maximizar la interoperabilidad, por lo que he decidido utilizar la gramática mínima de SQL. Por otra parte, por una cuestión de performance, decidí no utilizar los drivers de ODBC para acceder a bases de datos DBASE, Paradox, Foxpro, etc., donde preferí utilizar el motor de bases de datos de Microsoft Access. De todas maneras, existe un único objeto "Database", que encapsula la base de datos, sea a través de ODBC o no. Sólo existen diferencias en el método de apertura de la base de datos, donde en el caso de XBASE es importante poder elegirla mediante *browse* de las unidades de disco disponibles.

Capítulo 2: OLE

El objetivo principal del standard (de facto, es de Microsoft) Object Linking and Embedding es lograr interoperabilidad entre las aplicaciones Windows, reutilizando código mediante un API abierta. Existen diferentes formas en las cuales aplicaciones pueden cooperar usando OLE. La forma más habitual, más orientada a documentos que a funciones, consiste en tener aplicaciones *container*, que procesen documentos compuestos, por ejemplo conteniendo distintos tipos de objetos, como texto, vídeo, sonido, gráficos, etc., que son creados y mantenidos por sus respectivas aplicaciones *server*. Es aquí donde a través de OLE, los servicios de las diferentes aplicaciones son integrados, de forma de verse a los ojos del usuario como una aplicación única. Otra forma de interacción es a través de OLE Automation, donde cada aplicación publica su jerarquía de clases y su funcionalidad, que puede ser utilizada por otras aplicaciones.

Linking and Embedding

Cuando un objeto es incorporado en un documento, mantiene una asociación con la aplicación server que lo creó, *linking* y *embedding* son las dos diferentes formas de asociación de los objetos con sus servers. Si un objeto es asociado mediante linking a un documento, el objeto reside fisicamente donde fue creado, es decir es una asociación virtual donde el documento container guarda solo una referencia al objeto. Establecer links es eficiente desde el punto de vista de que los objetos no están repetidos, los cambios hechos en el objeto fuente son reflejados en el container y pueden establecerse complejos links anidados. Por el contrario, se torna engorroso a la hora de transportar los documentos.

En cambio, en caso de que el objeto sea embedded, una copia del objeto original es guardada en el documento container, así como también toda la información que posea ese objeto para su respectivo server. Es decir, el objeto embedded pasa a formar fisicamente parte del documento container. Las ventajas y desventajas, se las pueden imaginar los lectores.

Arquitectura

La arquitectura de OLE está basada en un diseño orientado a objetos. En términos de objetos, OLE define interfaces mediante una jerarquía de clases y conjuntos de métodos que describen el comportamiento de los objetos sin mostrar su implementación. En función de definir una interfase común, muchas de las clases definidas por OLE son abstractas, de donde las aplicaciones server derivarán sus propias clases e implementarán sus propios métodos. Por supuesto, dentro de esta familia de clases básicas existen métodos implementados, como por ejemplo los referentes al sistema de comunicación entre objetos.

Interfaces

Las interfaces o clases, pueden estar implementadas por OLE en sí, la aplicación server o la aplicación container, dependiendo de los servicios que otorguen. Los servicios standard para todas las aplicaciones, están implementados directamente en interfaces provistas por OLE, mientras que servicios que son específicos para determinado documento, objeto o aplicación, son implementados en interfaces de las aplicaciones server y container. Por ejemplo, cuando un usuario de una aplicación container edita un objeto embedded, el container invoca los métodos de la interfase provista por el server para hacer la edición, mientras que cuando el server termina una acción como aumentar el objeto de tamaño, invoca métodos del container para avisar al container que el tamaño del objeto cambió. La comunicación entre el container y el objeto en cuestión se realiza invocando métodos en las librerías de OLE propiamente dicho.

Los servicios provistos pueden clasificarse en cuatro áreas: comunicación entre objetos, soporte de infraestructura, linking y embedding básico y funciones avanzadas.

Objetos OLE

Existen dos niveles de objetos OLE, el más general llamado *component object* implementa todo el soporte de comunicaciones entre objetos. Los component objects son los que se utilizan para acceden a las interfaces de bajo nivel de OLE. Por ejemplo cuando una aplicación container efectúa un call a una función de *in place activation*, un component object es el que pasa la información al server apropiado, la aplicación no necesita saber a cerca como se hace la comunicación o como se ve el objeto. Todo esto es implementado por los component objects.

El otro nivel de objetos OLE es el que usan los documentos compuestos. Los objetos de documento compuesto soportan además de la comunicación, las funciones básicas para linking o embedding. En general poseen dos tipos de datos, los nativos, que son los que necesita el server para editar el objeto y los de presentación, que se necesitan para mostrar el objeto.

Sistema de almacenamiento

OLE implementa un sistema de almacenamiento jerárquico similar a un file system donde existe un nivel de *storage objects*, que pueden ser vistos como directorios, y otro nivel, los *stream objects*, que se asemejan a los archivos. En otras palabras, un *storage object* puede contener *streams* y otros *storage objects*, un *stream* solo puede contener datos. A cada objeto OLE se le asigna su propio *storage object*. El acceso a los objetos se hace a través de una interfase provista por OLE, llamada *Docfiles*.

Aunque las aplicaciones OLE no necesariamente deben usar Docfiles, es lo que se recomienda, pues es un sistema de almacenamiento especialmente diseñado para objetos OLE. Por ejemplo una de sus ventajas es que maneja transparentemente la carga parcializada de objetos, útil al tratarse de objetos de gran tamaño como vídeo, sonido u objetos compuestos, luego a medida que la aplicación lo necesita, Docfiles entrega más porciones del objeto. Además, puede funcionar también en modo transaccional mediante two-phase

commit, en función de soportar aplicaciones con mantenimiento de versiones o varios niveles de undo.

Comunicación entre procesos

La comunicación entre procesos está totalmente encapsulada por OLE, con lo cual tanto servers como clientes no deben preocuparse por esto. La comunicación se efectúa por medio de paquetes donde OLE agrupa mensajes y datos en función de optimizar la comunicación, esto no parece muy útil si cliente y server están dentro de una misma máquina, pero si lo es a la hora de implementar Network OLE, el próximo paso.

OLE Automation

OLE Automation es el camino a un nuevo standard de interoperabilidad de aplicaciones propuesto por Microsoft, básicamente consiste en publicar una interfase o jerarquía de clases en función de que otras aplicaciones puedan utilizar sus servicios. De esta manera las aplicaciones diseñadas mediante OLE Automation pueden además de utilizar sus propias partes u *objetos programables*, exponerlos para que otras aplicaciones los utilicen.

Objetos Programables

En la jerga se llama objeto programable a una instancia de una clase que la aplicación publica para su utilización. Estas pueden ser clases propias de la aplicación, o bien clases que estructuren las cosas de forma que tengan sentido para ser utilizadas por otros. Entonces, un objeto es *programable*, cuando la aplicación expone sus miembros, es decir sus métodos y sus propiedades. Los métodos son los servicios que expone el objeto mientras que las propiedades son pares de funciones que informan o setean estados internos del objeto. Por ejemplo algunos métodos del objeto visual *Window* son: *Activate, Restore, Minimize*, etc., mientras que algunas propiedades son: *Height, Width, WindowState*, etc.

Otro concepto muy común para OLE Automation son las *collections*, clases que nos permiten agrupar objetos, como por ejemplo la clase *Windows*, que agrupa objetos *Window*.

Partes de OLE Automation

OLE Automation tiene por un lado interfaces para hacer aplicaciones que expongan sus objetos programables y por otro, interfaces para que herramientas de programación, Visual Basic por ejemplo, puedan acceder dichos objetos programables.

El grupo de interfaces *Dispatch*, permite a los programadores la exposición de sus clases además de proveer una implementación standard de funciones para acceder a los objetos expuestos. Esto incluye la interfase *IDispatch* e *IEnumVariant*, encargada de la manipulación de datos, ambas implementadas en OLE2DISP.DLL.

Las interfaces *Type* proveen la información de tipos necesaria para acceder a los objetos, incluye las interfaces *ITypeLib* e *ITypeInfo* y están implementadas en TYPELIB.DLL.

OLE Automation también incluye National Language Support, estas funciones permiten a los programadores exponer sus interfaces en distintos idiomas, están implementadas en OLE2NLS.DLL.

Problemas

Por el momento no existe un standard que indique formas de organizar la jerarquía, de nombrar los objetos, miembros, etc., con lo cual cada desarrollador de una aplicación OLE Automation hace las cosas como quiere. Esto es un problema a la hora de programar un cliente, dado que por ejemplo, la jerarquía de Excel, Quattro o Lotus no tienen por qué ser la misma, si es que la tienen, por lo tanto el programador debe desarrollar una interfase para cada server. Como conclusión, OLE parece funcionar muy bien mientras nos movamos dentro de Microsoft, y dado que OLE es la arquitectura en que Microsoft desarrolla sus nuevos sistemas operativos (léase Cairo), queda claro que es un paso más de Microsoft hacia el monopolio.

Más información

Para más información sobre OLE es conveniente consultar los Microsoft Developer's Network CD's, asimismo se encontrará información de último momento en Internet, bajo la dirección http://www.microsoft.com/msdn.

OLE y VQL

Visual Query Language hace uso de OLE Automation para comunicarse con distintas aplicaciones de mercado, en este caso se ha previsto la utilización de Microsoft Excel, para lo cual se ha estudiado su jerarquía de clases y los servicios necesarios en función de hacer una transferencia de datos entre VQL y Excel.

OLE Automation VQL - Excel

Dentro de la funcionalidad de VQL, tanto en su versión Custom como en Runtime, se encuentra la posibilidad de pasar a Excel la información seleccionada en la ventana de resultado, ventana que contiene los registros resultado del query a la base de datos, donde luego el usuario puede seleccionar un grupo de registros o campos para se transferidos a la planilla de cálculo.

Mostraré aquí el código reducido para efectuar esta transferencia.

Declaración de variables:

Se declara una variable tipo *object*, que la que apunte al objeto Application de Excel que se creará más adelante.

Dim oleobj As object
Dim iselstartcol, iselendcol, iselstartrow, iselendrow as integer
Dim col, row as integer

Definición del rectángulo seleccionado por el usuario.

```
iselstartcol = Me.cResultGrid.SelStartCol
iselendcol = Me.cResultGrid.SelEndCol
iselstartrow = Me.cResultGrid.SelStartRow
iselendrow = Me.cResultGrid.SelEndRow
```

Creación del objeto Application de Excel, que debe existir en la tabla de registración de objetos OLE de Windows, la *registry* implementada en el archivo REG.DAT.

```
Set oleobj = CreateObject("Excel.Application")
```

Se le indica al objeto Application que agregue un nuevo libro de trabajo, que contendrá una nueva WorkSheet por default.

```
oleobj.workbooks.add
```

Transferencia de los nombres de los campos, se accede abreviadamente a la propiedad cells, pues se utilizan propiedades default, es lo mismo:

oleobj.cells que,

oleobj.workbooks.activeworkbook.worksheets.activeworksheet.cells

```
For col = 0 To Me.cResultGrid.Cols - 1
    oleobj.cells(1, col + 1).value = Me.cResultGrid.ColHeading(col)
Next
```

Transferencia de los registros

```
For Row = iselstartrow To iselendrow
   Me.cResultGrid.Row = Row
   For col = 0 To Me.cResultGrid.Cols - 1
     oleobj.cells(Row - iselstartrow + 3, col + 1).value = Me.cResultGrid.ColText(col)
   Next
Next
```

Se hace visible al objeto, con lo cual se le pasa el control.

```
oleobj.Visible = True
```

Como se ve, en el caso de Visual Basic, el código para OLE Automation queda perfectamente integrado, por supuesto existen también dificultades, especialmente a la hora de hacer debugging, donde lo único que vemos es un *OLE Automation Error!*.

Ejemplo: Jerarquía de Clases de Excel (Abreviada).

Application
- *MenuBar*
  - *Menu*
    - *Menuitem*
- *ToolBar*
  - *ToolBarButton*
- *AddIn*
- *Debug*
- *Dialog*
- *Window*
  - *Pane*
  - *PageSetup*
- *WorkBook*
  - *WorkSheet*
  - *Chart*
  - *DialogSheet*
  - *Module*
  - *Style*
  - *Window*
    - *Pane*
  - *Name*
  - *RoutingSlip*
  - *Mailer*

Capítulo 3: SQL Front Ends para Windows

Como ya se anticipó en la introducción, la arquitectura cliente-servidor combina los beneficios de los poderosos RDBMS's corriendo sobre hardware de gran potencia, inclusive sobre minis o mainframes, con la facilidad de uso que proveen las nuevas interfaces gráficas, como Windows por ejemplo. Como en general casi todos los manejadores de bases de datos utilizan SQL como lenguaje de consulta, comúnmente se refiere a las herramientas de programación para aplicaciones cliente como SQL Front Ends.

En esta oportunidad se han analizado tres front ends, PowerBuilder de PowerSoft, SQL Windows de Gupta y Visual Basic de Microsoft.

Todos los productos incluyen lenguajes de programación que complementan la programación visual, que en general es asistida por editores especializados en cada caso. En cuanto a diseño general sobre el manejo de bases de datos los dos primeros fueron concebidos específicamente para la arquitectura cliente servidor, es decir para acceder a bases de datos en forma remota, y luego incorporaron manejadores propios de bases de datos en forma local para prototipación, mientras que en el caso de Visual Basic, la evolución fue al revés, comenzó con su sistema de archivos propio y luego incorporó el soporte para acceder, primero a SQL Server mediante DB Library y luego expandió sus horizontes soportando ODBC.

A su vez, en cuanto a las facilidades para desarrollo en grupo, PowerBuilder y SQL Windows cumplen bien, en cambio Visual, aún en la versión 4, es escaso en este propósito. Por otra parte, todos los productos son capaces de generar aplicaciones ejecutables, es decir no necesitan runtime, vale la pena aclarar que fue Microsoft el que de alguna manera obligó a la competencia a permitir generar aplicaciones ejecutables *royalty free*, puesto que los otros distribuían un runtime con costo por cada cliente.

Tools

Los tres productos poseen diversas herramientas como por ejemplo manejadores de proyectos, diseñadores de forms o windows, generadores de reportes, manejadores de bases de datos, manejadores de objetos, etc. En general todas son bastante intuitivas, aunque se nota una mayor profesionalidad en PowerBuilder. Los tres productos se venden en tres diferentes versiones, desktop, network y enterprise o similares, las diferencias entre las versiones son justamente las herramientas con que vienen, e inclusive en algunos casos su funcionalidad.

Soporte a RDBMS's

Este es un punto donde se nota una clara superioridad de PowerBuilder y SQL Windows sobre Visual Basic, puesto que aunque los tres soportan ODBC, los dos primeros acceden directamente a los drivers de las bases de datos líderes, como Oracle, SyBase, DB2, Informix, haciendo uso específico de las ventajas de cada uno, mientras que Visual sólo opera en forma directa con Microsoft SQL Server.

Objetos

SQL Windows es el que tiene en mi opinión el mejor modelo de objetos, soportando la definición de clases en forma visual, manteniendo el modelo en un sentido más *puro* que los demás, inclusive soportando herencia Multiple.

PowerBuilder maneja el concepto de clase, aunque disfrazado en lo que llama user object, por ejemplo, es imposible derivar de un botón, sino que debe crearse un user object que contenga un botón y agregarle la funcionalidad necesitada.

Visual Basic es el mas pobre en cuanto a orientación a objetos, aunque a impuesto lo que se ha dado en llamar *component programming*, un paradigma, si es que se lo puede calificar así, que está basado en la reutilización de componentes de programación (*vbx* en un primer momento, *ocx* luego) que poseen propiedades, métodos y son estimulados por eventos, pero sin embargo no es posible derivar de ese componente, no existe la noción de clase. Recién en la versión 4, se incorporan la herencia y el concepto de clase, aunque no completamente integrados en el modelo. Vale la pena decir que nuevamente, VB ha impuesto este standard y ha obligado a los dos anteriores a soportar dentro de su modelo vbx y ocx.

Resumen Comparativo

	PowerBuilder	SQL Windows	Visual Basic 3
RAM Mínimo	**8 MB**	**4 MB**	**4 MB**
ODBC	**Si**	**Si**	**Si**
Soporte Drivers Esp.	**Si**	**Si**	**No**
Generación Autom.	**Si**	**Si**	**No***
Soporte a VBX	**Si**	**Si**	**Si**
Herencia	**Si**	**Si**	**No***
Proc. de eventos	**Si**	**Si**	**Si**
OLE	**Si**	**Si**	**Si**
Drag & Drop	**Si**	**Si**	**Si**
Generac. de Reportes	**Si**	**Si**	**Si**
Control de Versión	**Si**	**Si**	**No***
Desarrollo en grupo	**Si**	**Si**	**No***
Repositorio de Apps.	**No**	**Si**	**No**
Calls a DLLs	**Si**	**Si**	**Si**
Generac. de EXE	**Si**	**Si**	**Si**

* Puntos soportados en Visual Basic 4.

Front Ends y VQL

VQL fue codificado enteramente en Visual Basic v3.0, la elección de la herramienta estuvo basada principalmente en dos razones, por un lado la disponibilidad y el costo del producto, que es sensiblemente más económico, aunque por supuesto las prestaciones no son las mismas, y por otra parte, aunque no estuviera orientado a desarrollo corporativo, VB ya se perfilaba como un standard importante de mercado, la prueba de ello es que con la aparición de la versión 4, la brecha entre los tres front ends ha disminuido. Personalmente hubiera preferido utilizar un mejor modelo de objetos, aunque es importante destacar que la curva de aprendizaje fue muy rápida. Cabe aclarar también que, como es usual en Microsoft, el producto original estaba plagado de bugs y problemas no documentados.

Capítulo 4: RDBMSs

Gran parte de las aplicaciones de hoy en día utilizan Relational DataBase Management Systems como fuentes de datos. Se nota ya una tendencia fuerte hacia el modelo cliente-servidor, utilizando como *front-end* Microsoft Windows, de forma de separar lo más posible la interface del procesamiento de la información.

No es mi intención en este capítulo repetir historias conocidas, sino más bien mostrar un análisis de tres RDBMS's del mercado, IBM DB2/2 2.1, Oracle 7.1 y Microsoft SQL Server 6.0.

Los tres sistemas ofrecen *stored procedures*, *triggers* y *constraints*. Stored procedures son rutinas almacenadas en el RDBMS, que se ejecutan en el servidor y son activados por el cliente mediante un call, obviamente estos procedimientos o rutinas, ya se encuentran compilados y optimizados, con lo cual la performance es muy superior a la ejecución de sentencias SQL dinámicas desde el cliente. Un trigger es un store procedure que se activa cuando ocurre un determinado evento, como por ejemplo la inserción de un nuevo registro o la modificación de algún campo. Constraints son reglas que especifican por ejemplo los valores aceptables de algún campo o bien determinadas relaciones entre campos y tablas, mediante constraints, se puede asegurar la integridad referencial de una base de datos, los tres sistemas traen conjuntos de reglas default para distintos niveles de integridad.

IBM DB2.

IBM se ha esforzado por ofrecer DB2 en diversas plataformas, PC's con OS/2, RS6000 con AIX, AS400 y mainframes. A medida que pasa el tiempo, los distintos DB2 son más parecidos entre sí, pero todavía existen diferencias significativas entre las versiones, fundamentalmente en lo que se refiere al lenguaje de definición de datos.

En el caso de DB2/2, soporta multiprocesadores simétricos, provee API's para diversos lenguajes como COBOL, FORTRAN, C y C^{++} , como también Visual Age o VX-REXX, también existe drivers de ODBC para DB2/2. Es posible configurar DB2/2 para soportar replicación de datos, inclusive funciona bien con DDCS (Distributed Database Connection Services).

La administración es sencilla, puede hacerse local o remota a través de software GUI provisto en el paquete.

Es posible indicar el nivel de optimización para SQL, existen nueve niveles configurables por la aplicación o por el administrador de la base de datos, de esta forma puede hacerse un tunning a nivel de encontrar el óptimo balance para cada aplicación.

Por otra parte, DB2/2 soporta SQL estático, provee un API para que los compiladores puedan mandar a precompilar las sentencias que pueden ser conocidas en tiempo de compilación, que DB2/2 almacenará en el server para su uso posterior. Cabe notar la diferencia con los stored procedures, que deben ser llamados explícitamente desde la aplicación.

Oracle

Oracle corre en más de 90 plataformas diferentes, 60 de las cuales son distintos ambientes UNIX. Quizá el punto más fuerte de Oracle es la portabilidad, ya que el software de base de datos es esencialmente el mismo código para todas las plataformas. Mover una base de datos de una plataforma a otra es tan fácil como correr un programa de transporte que viene con el paquete. La administración de la base de datos es igualmente consistente en todos los ambientes.

También Oracle sabe aprovechar los ambientes con múltiples procesadores, siempre que el sistema operativo lo administre, como es el caso de versiones de OS/2, Windows NT, AIX o SCO Unix. La versión previa ya soportaba replicación de datos estricta mediante two-phase commit, ahora se incorporan nuevos esquemas de replicación, con time-stamps, en versiones pesimistas y optimistas.

En cuanto a la administración, también se hace a través de herramientas GUI, que corren en los sistemas nativos, en el caso de los sistemas operativos no provean GUI, como Novell Netware por ejemplo, Oracle provee herramientas Windows para hacer la administración en forma Remota desde una estación de trabajo.

Oracle no es tan fácil de programar como DB2 /2 y tampoco aprovecha la estructura de multitrheading del sistema operativo donde está instalado (tiene una propia), pero por otra parte, recalcando lo que ya se dijo, los mejores puntos de Oracle son portabilidad, escalabilidad e interoperabilidad.

Microsoft SQL Server

Originalmente, Microsoft compró la licencia de SQL Server a SyBase, que funcionaba bajo OS/2. Luego de poco tiempo, y con el advenimiento de Windows NT, los productos comenzaron a diferenciarse, cada uno siguió su camino. Esta última versión de SQL Server contiene cambios substanciales con respecto a la arquitectura, no así con respecto a la interfase y administración, donde sigue siendo similar a sus anteriores versiones, inclusive al producto de SyBase.

SQL Server, que corre solo bajo Windows NT, tiene una nueva herramienta GUI de administración llamada SQL Enterprise Manager, mientras que versiones anteriores usaban ISQL una herramienta basada en texto.

Esta versión agrega *scrollable cursors*, manejo distribuido de los objetos y *extended store procedures*. Esto último es interesante pues no limita a los desarrolladores a suscribirse a las limitaciones de Transact SQL, el lenguaje de programación nativo de SQL Server. Otra ventaja, o desventaja según se lo vea, es la alto nivel de integración que posee con el sistema operativo en cuanto a arquitectura, sistemas de seguridad y administración. También implementa replicación de datos y hace uso de múltiples procesadores asignando un thread NT a cada procesador. En cuanto a interoperabilidad, Microsoft propone la utilización de ODBC.

Depto. COMPUTACION
INFOTECA
F.C.E.N. - U.B.A.

Resumen Comparativo

	DB2/2	Oracle	SQL Server
Precio por usuario(50)	U$s 123	U$s 101	U$s 25
Plataforma Server	PC OS/2 RS/6000 AIX AS/400 OS/400 Mainframe MVS, VM	PC OS2,Netware,NT Varios UNIX Mainframe MVS, VM	NT
RAM	16 MB	16 MB	32 MB
SMP	Si	Si	Si
Campos por tabla.	256	256	250
Lockeo de registros	Solo Página	Si	Solo Página
Replicación	Si	Si	Si
Adm. Centralizada	Si	Si	Si
Integridad Referencial	Si	Si	Si
Store Procedures	Si	Si	Si
Triggers	Si	Si	Si
Constraints	Si	Si	Si
Transporte	IPX, NetBIOS, SNA (APPC), TCP/IP	IPX, NetBIOS, TCP/IP	IPX, NetBIOS, TCP/IP, NetBEUI
SQL Estático	Si	No	No
ODBC Driver	Si (No Nativo)	Si	Si
Alertas SNMP	Si	No	No

RDBMS y VQL.

Dado su concepción y arquitectura, VQL debe soportar la mayor cantidad de RDBMSs posible por lo cual se ha optado por utilizar el standard ODBC, que provee conectividad con una gran mayoría. Puesto que VQL es un sistema enteramente de consulta, no utiliza funciones avanzadas de los RDBMSs, de esta forma fue posible lograr una uniformidad sin perder funcionalidad.

En cuanto la implementación demo se eligió SQL Server por una mera cuestión de disponibilidad. Inclusive las pruebas se hicieron con una versión anterior del motor.

Capítulo 5: Visual Query Language

VQL está presentado en dos versiones, **Custom** y **Preset**, Custom está dirigida hacia los usuarios experimentados o desarrolladores de software, que pueden acceder a todas las tablas de la base de datos, en forma totalmente libre. Preset, en cambio, está orientada al usuario final no necesariamente experimentado, conceptuado en forma de add-on para cualquier sistema de aplicación bajo Windows. Preset está compuesto a su vez por dos módulos, **Runtime**, que es el encargado de proveer la interfase, generar las consultas y efectuarlas, tomando la configuración desde un programa de inicialización .VQL generado por **Setup**, el otro módulo, donde el desarrollador de software genera el .VQL, que contiene la información necesaria para acceder a la base de datos, qué gráfico representa qué tabla, qué tablas son visibles y cuales no, los nombres para los campos, etc.

VQL Custom Query

Funcionalidad

Los usuarios experimentados en Windows y con dominio de bases de datos relacionales pueden saltear esta sección, mi recomendación es instalar el producto y utilizarlo, es muy intuitivo.

Generalidades

VQL Custom Query está diseñado en forma totalmente MDI (Multiple Document Interface), con lo cual el usuario puede tener abiertas múltiples querys a bases de datos heterogéneas. La interfase tiene un look 3D, correspondiente con las nuevas aplicaciones para Windows 95 y funcionalmente es similar a cualquier aplicación Windows del mercado, lo cual asegura una utilización intuitiva (para usuarios Windows). Contiene un *dynamic toolbar*, nuevo concepto de toolbar donde sólo encontramos visibles a los botones disponibles en cada instante, según los estados en que se puede encontrar el sistema. Por otra parte posee un *status bar*, donde se va desplegando la ayuda en forma automática y sensible al contexto.

Menú Bar

Archivos.

Abrir BD Local:	Contiene las opciones para abrir bases de datos dBase III, dBase IV, MS Access, MS Foxpro 2.5 y Paradox.
Abrir BD ODBC:	Abrir bases de datos ODBC, invoca el diálogo de conexión con una fuente de datos.
About:	Información sobre el autor.
Salir:	Sale del sistema.
Ultimas sesiones:	Aquí se muestran las cuatro últimas sesiones, de forma que el usuario acceda rápidamente.

Preferencias.

Login Time Out:	Establece el máximo tiempo de tolerancia para establecer una conexión con una fuente de datos.
Query Time Out:	Establece el máximo tiempo de tolerancia para ejecutar una consulta.
Default DBM:	Especifica el Database Manager que se va a utilizar por defecto, es decir el tipo de base de datos que se va a intentar abrir cuando el usuario pulse el botón *abrir* de la base de datos puede ser dBase III, dBase IV, MS Access, MS Foxpro 2.5, Paradox u ODBC.

Ventanas.

Cascada:	Organiza los querys abiertos en forma de cascada.
Mosaico:	Organiza los querys abiertos en forma de mosaico.
Organizar íconos:	Ordena los íconos de querys que estén minimizados.

Una vez descripto el menú, objeto de por sí estático, seguiré con la descripción de la funcionalidad de la ventana de query, a la que se llega después de abrir la base de datos que se consultará. En el caso de ODBC deben ingresarse los parámetros de conexión, como *DataSource, UserId y Password*, en el caso de bases de datos locales el sistema permite hacer browse de las unidades disponibles, utilizando el diálogo standard de Windows para apertura de archivos.

Ventana de Query.

Como *title bar*, de la ventana tenemos la base de datos que ha sido abierta, y en una lista a la derecha las tablas que la componen. Una vez que se eligen las tablas de donde se obtendrá información, se actualizan los paneles de información sobre orden, agrupamientos y campos a mostrar. Luego deben seleccionarse los campos que se desean ver, puede ingresarse información sobre juntas, criterios de selección y agrupación. Por otra parte, haciendo click con el botón derecho sobre los distintos campos se obtiene su funcionalidad en forma de *popup*. Existe la posibilidad de calcular funciones de los campos según del tipo que sean, están disponibles suma, promedio y cantidad para tipos numéricos y solo cantidad para otros tipos.

Con una ventana de query en el foco de la aplicación se tienen en el toolbar las funciones de abrir otra base de datos, cerrar la actual, blanquear todos los campos, mostrar el código SQL generado hasta el momento y ejecutar la consulta.

Si uno ejecuta la consulta se crea un nuevo *child window*, que contendrá el resultado triado desde la fuente de datos.

Ventana de Resultado.

Como title bar tenemos el título elegido para la consulta, luego vemos en forma de browse interactivo los registros resultado. Existen botones para avanzar y retroceder en el resultado, ir al principio y al final, avanzar o retroceder páginas y poner marcas e ir hacia ellas. Con la ventana de resultado activa, vemos en el toolbar los botones correspondientes a

abrir una nueva base de datos y pasar los datos del resultado a una planilla Excel. Una vez seleccionado el grupo de campos o registros que queremos transferir, activamos el botón y comienza el intercambio de datos por OLE Automation.

Ventana de Junta

Contiene un lista con las tablas seleccionadas hasta el momento en la ventana de Query, una vez que se seleccionan dos tablas, se actualizan dos nuevas listas con los campos correspondientes a cada tabla seleccionada. Luego deben elegirse los campos que formen la junta, y pulsar el botón de *Agregar*, para efectivamente agregar la junta a la lista en la ventana de Query.

Ventana de Selección de Criterio.

Útil para la selección de criterios de selección o agrupación, que se corresponderán con cláusulas WHERE y HAVING respectivamente, la ventana de selección posee campos para elegir campo operando, operador y valor de referencia, además de indicar si se agrega a la lista de criterios con un operador lógico AND u OR.

En todo momento, por supuesto es posible activar cualquiera de las ventanas que existan en la aplicación y trabajar sobre ellas sin perder los datos ingresados. En todo momento tenemos ayuda en línea en el status bar. Al cambiar las preferencias, quedan almacenadas en un archivo de inicialización llamado CUSTOM.INI.

VQL Preset Query

Como ya se anticipó, Preset está compuesto por dos módulos, uno para el desarrollador, llamado Setup, y otro el que accede el usuario final, Runtime. El producto de Setup es un programa .VQL, que puede ser editado llegado el caso con cualquier editor ASCII, que luego es interpretado por Runtime, que arma de acuerdo a él la interfase con el usuario final.

Preset Setup

También completamente MDI, Setup es capaz de editar varios archivos .VQL al mismo tiempo. Está organizado en forma de *wizard*, va dando al desarrollador una serie de pasos a seguir, que van desde la prototipación hasta los últimos detalles de conexión. Al igual que Custom posee un *dynamic toolbar*, un menú de opciones generales y una *status bar*, donde continuamente da mensajes y ayuda al usuario.

El objetivo de Setup es generar un archivo .VQL para que sea interpretado por Runtime. El archivo tiene la siguiente estructura.

Una primera sección donde se indican las preferencias y path para los íconos.

[Preferencias]
LoginTimeOut = *n*
QueryTimeOut = *n*
IconDevPath = < *Directorio para almacenar los íconos que se utilicen en desarrollo*>
IconProdPath = < *Directorio donde estarán los íconos en producción* >

Luego, los datos referentes a aplicaciones que se manejen desde runtime por OLE Automation.

[OLE]
Excel = <*habilita conexión?*>

Otra sección con la información sobre la base de datos.

[Base de Datos]
DBM = < *Manejador de Base de datos que utiliza* >
Tablas = < *Cantidad de tablas* >
Nombre = < *Nombre* >

Luego, una sección para las tablas y a continuación otra para cada tabla.

[Tablas]
Tabla0 = *Nombre0, Titulo0, Icono0, visible?, cantidad de campos 0*
Tabla1 = *Nombre0, Titulo0, Icono0, visible?, cantidad de campos 1*

-
-
-

TablaN = *.NombreN, TituloN, IconoN, visible?, cantidad de campos* N

[Tabla0]
Campo0 = *Nombre0, Título0, visible?, tipo 0*
Campo1 = *Nombre1, Título1, visible?, tipo 1*

•
•
•

CampoM = *NombreM, TítuloM, visible?, tipo M*

•
•
•

[TablaN]
Campo0 = *Nombre0, Título0, visible?, tipo 0*
Campo1 = *Nombre1, Título1, visible?, tipo 1*

•
•
•

CampoP = *NombreP, TítuloP, visible?, tipo P*

Ahora veamos como se construye un archivo .VQL con Setup. Empecemos con una descripción de las funciones incluidas en el menú.

Menú Bar

Archivos.

Nuevo:	Crea un nuevo archivo .VQL, iniciando el proceso de prototipación.
Abrir:	Abre un archivo .VQL para su edición.
Salvar:	Salva el archivo actual.
Salvar Como:	Permite salvar el archivo actual con otro nombre.
About:	Información sobre el autor.
Salir:	Sale del sistema.
Ultimas sesiones:	Aquí se muestran las cuatro últimas sesiones, de forma que el usuario acceda rápidamente.

Preferencias.

Login Time Out:	Establece el máximo tiempo de tolerancia para establecer una conexión con una fuente de datos.
Query Time Out:	Establece el máximo tiempo de tolerancia para ejecutar una consulta.

Ventanas.

Cascada:	Organiza las ventanas en forma de cascada.
Mosaico:	Organiza las ventanas en forma de mosaico.
Organizar íconos:	Ordena los íconos de ventanas que estén minimizadas.

Describiré la funcionalidad de Setup guiando a los lectores a través de la generación desde el inicio de un archivo .VQL.

Fase 1:

Mediante el botón o la opción de menú de crear archivo, iniciamos la fase 1, que solicita los datos para la prototipación del archivo:

- El tipo de la base de datos (Database Manager) puede ser dBase III, dBase IV, MS Access, MS Foxpro 2.5, Paradox u ODBC.
- La base de datos, que se abre con el botón *Abrir* que dependiendo del tipo de base de datos, activa un browse de las unidades de disco para que el usuario seleccione la base de datos a utilizar para las bases de datos no ODBC, o activa el diálogo standard de conexión a fuentes de datos ODBC. Esta es la base de datos que se usará para la prototipación del programa.
- El directorio donde Setup almacenará los íconos utilizados.

Una vez completados los datos, pasamos a la fase 2 con el botón *Próximo*.

Fase 2:

- Una vez abierta la base de datos, Setup prototipa el programa .VQL manteniendo para ello una estructura de datos interna, la que llena con una serie de defaults provenientes algunos de ellos de la estructura de la base de datos. A partir de esta etapa, ya es posible salvar el programa con un nuevo nombre, ya que Setup establece automáticamente nombres para los programas nuevos que se generan. De hecho, una vez que superamos la fase 1, ya no se utiliza información sobre la base de datos, es por ello que la edición de un programa VQL existente comienza indefectiblemente por la fase 2. Es aquí donde para cada tabla especificamos:
- Visible: Indica si esa tabla podrá ser accedida por el usuario final. El default es que si pueda.
- Título: El título en lenguaje natural para esa tabla, en función de no confundir al usuario con los nombres físicos de las tablas. El default es el nombre físico de la tabla, informado por el DBMS.
- Icono: El gráfico que representará la tabla en Runtime. Con el botón de *Browse*, podemos seleccionar el ícono de las unidades disponibles, una vez seleccionado, Setup lo almacena en el directorio de íconos indicado en la fase 1. Existe un ícono default que simboliza un tabla, se llama DEFTABLE.ICO.
- Información sobre los campos de la tabla:

  Campo: Nombre físico del campo, es read only y lo actualiza Setup en la prototipación.

  Visible: Indica si el campo será visible para el usuario final. El default es que sí lo sea.
- Título: El título en lenguaje natural para ese campo. El default es el nombre físico del campo, informado por el DBMS.

Una vez completados todos los datos, en realidad puede hacerse en cualquier momento, puesto que Setup propone defaults para todas las opciones, se pasa mediante el botón próximo a la fase 3.

Fase 3:

En esta fase final, se indica información adicional que necesita Runtime sobre el ambiente de producción, según el tipo de base de datos que se utilice, Setup permite completar algunos campos y otros no.

- Base de datos: Ubicación y nombre físico de la base de datos, este ítem se aplica si la fuente de datos es no ODBC.
- Directorio de íconos: Es el directorio donde Runtime debe buscar los íconos en el ambiente de producción.

- ODBC: Información necesaria para acceder directamente a la fuente de datos, es decir, Server, Database, Username y Password.
- Ole Automation: Indica si se permitirá operar con Microsoft Excel a través de OLE Automation.

Una vez concluida la carga de datos se pulsa *Fin*, donde Setup salva el programa y concluye la sesión de edición para ese programa.

En cualquier momento es posible volver a la etapa anterior, salvar los cambios, salvar el programa con otro nombre, cancelar la edición de ese programa, editar uno ya existente, crear uno nuevo, etc.

Preset Runtime

Runtime es la herramienta para el usuario final, que puede ser utilizada *stand alone*, pero fue diseñada para actuar como *add-on* de otras aplicaciones Windows que interactúen con bases de datos.

Runtime recibe como parámetro en la línea de comandos el nombre del archivo .VQL que debe interpretar. En caso de no recibir parámetros despliega un diálogo de browse por las unidades de disco disponibles, inclusive de red, con el objeto de abrir un archivo .VQL. Al operar recibiendo en la línea de comandos, es posible registrar bajo Windows el tipo de archivo .VQL como de Visual Query Language, asociado al Runtime, de modo que cuando se activen o editen programas .VQL, automáticamente se arranque el Runtime, es decir ejecutando los archivos .VQL desde las aplicaciones, se activará el runtime que interpretará ese archivo.

Aunque funcionalmente es similar a la pantalla de Query de Custom, Runtime presenta las cosas de otra manera, tomando la información del .VQL, asigna un gráfico a cada tabla, solo muestra las tablas que se programaron como visibles y muestra los títulos que se asignaron en vez de los nombres físicos de las tablas.

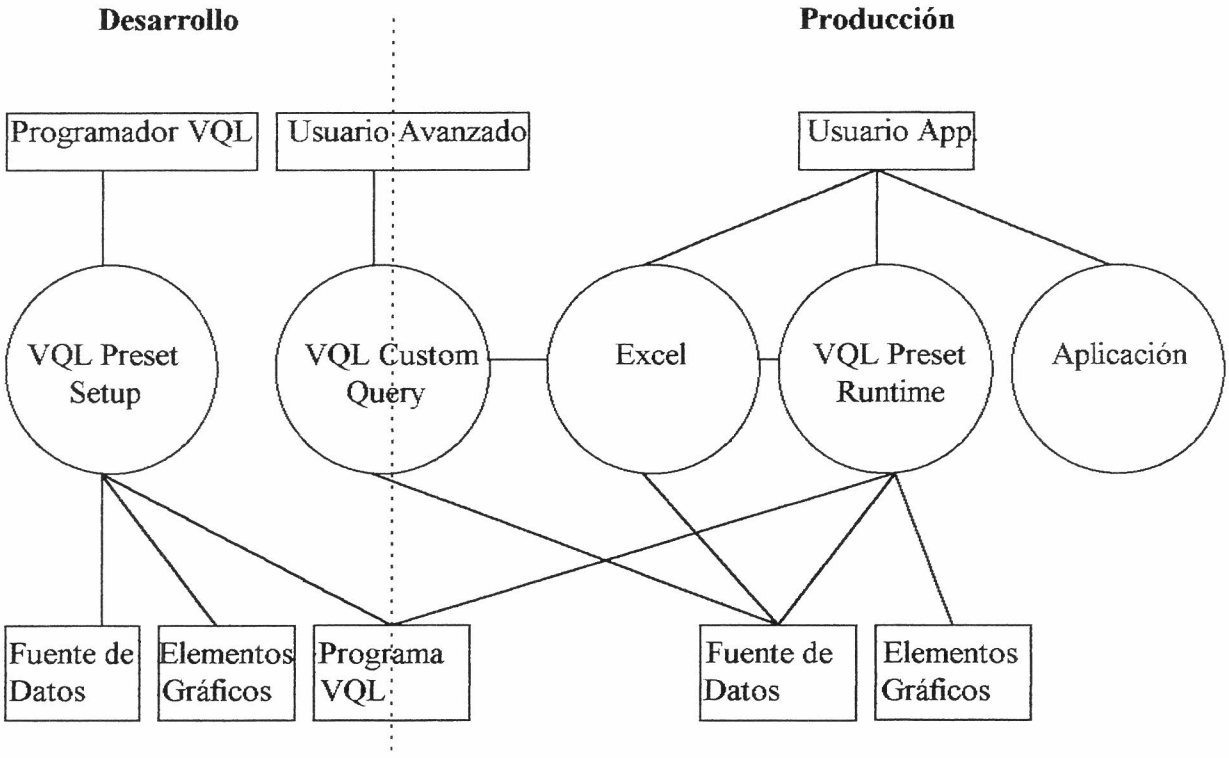
Al seleccionar un gráfico, se refrescan los paneles referentes de campos, que esta vez contienen los títulos en lenguaje natural. Cuando seleccionamos más de una tabla, se habilita la opción de Junta, que nos permite establecer una junta natural entre tablas, activando la ventana de selección de Juntas. Similarmente, es posible establecer criterios de selección o de agrupación.

Una vez que se programa visualmente el query, puede verse el código SQL mediante el botón *SQL* del toolbar, o bien ejecutarse la consulta mediante el botón *?*. Aquí aparece un diálogo pidiendo el nombre de la consulta, una vez ingresado éste se procede a efectuar la consulta con el botón de *OK*. Si la consulta no produce ningún error en la fuente de datos, se despliega una ventana de resultado, completamente independiente de la ventana principal, con los títulos de cada campo, los registros del resultado y botones para avanzar y retroceder en el resultado, ir al principio y al final, avanzar o retroceder páginas y poner marcas e ir hacia ellas, además de un botón para interactuar con Excel mediante OLE Automation, si es que en el programa VQL está habilitada la función.

Cabe aclarar, que pueden tenerse varias ventanas de resultado activas al mismo tiempo, pudiendo pasar de una a otra sin interferir en los punteros o cursores a los resultados, inclusive, puede hacerse un nuevo query utilizando la información del anterior, todo ello sin modificar en absoluto los resultados obtenidos.

Arquitectura y Diseño

El Escenario



Como vemos en este diagrama, el programador VQL combina elementos gráficos con las estructuras de datos, generando mediante Preset Setup el programa VQL en forma asistida, que será interpertado en tiempo de producción por VQL Runtime, permitiendo al Usuario acceder a los datos en forma visual. En cuanto a VQL Custom, puede ser utilizado por un usuario avanzado en cualquier circunstancia, como herramienta visual de consulta. En cualquiera de los casos, el usuario puede transferir sus resultados a una planilla Excel.

Las Estructuras

Estructura de VQL Preset Setup

<table>
<tr><td colspan="9">MDI Main Container</td></tr>
<tr><td colspan="3">Wizard 1</td><td rowspan="2">o o o o o o o o o</td><td colspan="3">Wizard N</td></tr>
<tr><td>Fase 1</td><td>Fase 2</td><td>Fase 3</td><td>Fase 1</td><td>Fase 2</td><td>Fase 3</td></tr>
<tr><td colspan="3">VQL Prototyper</td><td colspan="3">VQL File Manager & Parser</td><td colspan="3">Rutinas de Soporte</td></tr>
</table>

Estructura de VQL Preset Runtime

<table>
<tr><td colspan="6">Visual Query Builder</td><td>Results Manager</td></tr>
<tr><td>VQL File Manager</td><td>Selection & Grouping Builder</td><td>Join Builder</td><td>SQL Generator</td><td>ODBC Interface</td><td>Rutinas de Soporte</td><td>OLE Interface</td></tr>
</table>

Estrucura de VQL Custom Query

<table>
<tr><td colspan="8">MDI Main Container</td></tr>
<tr><td colspan="2">Visual Query Builder 1</td><td>o o o o</td><td>VQM N</td><td colspan="2">Results Manager 1</td><td>o o o</td><td>RM M</td></tr>
<tr><td>VQL File Manager</td><td>Selection & Grouping Builder</td><td>Join Builder</td><td>SQL Generator</td><td>ODBC Interface</td><td>Rutinas de Soporte</td><td colspan="2">OLE Interface</td></tr>
</table>

Como vemos, Custom Query es en cierta forma una colección de Visual Query Builders aunque con algunas diferencias respecto de Runtime, ya que los diferentes componentes de Custom acceden directamente a la información de la estructura de la base dse datos, provista por el driver de ODBC. En cambio, Runtime toma toda la información del programa VQL.

Posibles Agregados y Mejoras

A continuación enumeraré una serie de ítems indicando las posibles mejoras y/o agregados que pueden incorporarse a futuro (tal vez un próximo tesista?), que no fueron implementados por exceder el alcance de este trabajo de tesis.

- Instalador de VQL.
  Desarrollo de un instalador para windows.

- Ayuda
  Al margen del modelo de autoayuda incorporado (ayuda automática sensible al contexto que se despliega en el status bar), sería interesante un poco de ayuda explicativa.

- Control de errores.
  El sistema efectua un control de errores básico, aunque si se deseara poner el producto en producción debería profundizarse en este punto.

- Look features.
  Agregar una sección ***Look*** al programa VQL para customizar el look del runtime, de forma de ser similar al sistema en que se utilice. Algunas variables podrían ser: color de fondo, 3D shading, tipo de borde, fonts, literales, etc.

- Install Kit.
  Desarrollo de un instalador para el programa VQL, runtime y librerías para el usuario final. Puede agregarse como ultima fase del Setup la generación de los diskettes de instalación para el Runtime que incluirían el programa instalador.

- Runtime OLE control.
  Proveer a runtime en forma de OLE control para ser incorporado en las aplicaciones en vez de ser ejecutado externamente.

Conclusión:

La investigación y el desarrollo efectuado, me han aportado conocimentos sobre los estandares propuestos por Microsoft, ya que la plataforma elegida para el desarrollo fue Windows, por una cuestión de mercado, puesto que es el sistema de desktop más popular y utilizado por la mayoría de los usuarios potenciales de una herramienta de este tipo.

Por otra parte, he adquirido una interesante experiencia en programación visual, aprendiendo Visual Basic, que además se a convertido en un estandar para el desarrollo de aplicaciones front-end bajo Windows. Otro de los puntos importantes que destaco fue haber analizado los diferentes motores de bases de datos relacionales disponibles en el mercado, lo cual me brindó una visión amplia al respecto.

En el caso particular de ODBC, encuentro interesante analizar una vez más el fenómeno Microsoft, aunque desde un punto de vista diferente. En este caso, los desarrolladores de RDBMS se han visto obligados a implemetar drivers de ODBC para sus motores, ya que de otra forma hubieran quedado afuera de muchas elecciones, puesto que a la hora de decidir un sistema para front-end, la alternativa número uno (sino la única) es Windows en cualquiera de sus versiones, lo mismo sucede con el Visual Basic, que de una forma u otra se ha ido transformando en la herramienta de desarrollo a elegir, más que por prestaciones tecnológicas por una cuestión de estandares y de disponibilidad de recursos, sin dejar de observar la innumerable cantidad de software escrito para Visual Basic (VBX, OCX). Es por ello que encuentro valioso haber obtenido conocimientos en este aspecto.

De forma similar, OLE se ha convertido en un standard para la comunicación entre procesos en Windows (no solo en Windows, ahora esta amenazando la posición de Java en el desarrollo de Internet Applications, con su implementación en forma de ActiveX...), de esta manera las aplicaciones que pretendan prestar servicios a otras en Wiundows, deben pensar seriamente en hacerlo a través de OLE, ya que en un futuro Nashville (Windows '97) y Cairo (próximo NT), estarán basados completamente en OLE.

En cuanto a Visual Query Language en sí, quiero destacar que como herramienta tuvo muy buena aceptación en una prueba piloto que se hizo distribuyéndola en forma gratuita a los clientes de la empresa para la cual trabajo. Una confirmación de que la senda era buena, es el hecho de que han salido al mercado herramientas de consulta a bases de datos en forma visual. A fines de 1994 Microsoft presentó Microsoft Query, cuya funcionalidad es similar a VQL Custom Query. Los RDBMS's comenzaron a incluir en sus paquetes herramientas para Windows. Por lo general todas estas herramientas han sido concebidas para ser utilizadas stand alone, no se ha tomado en cuenta la posibilidad de verlas como add-on para las aplicaciones, que es la idea principal de VQL, que se ve plasmada en VQL Runtime. que se ve como parte de la aplicación y su función principal es permitir al usuario efectuar consultas no planificadas.

En cuanto a la contribución académica, creo que este trabajo es una buena síntesis de los temas a estudiar para las personas que deban desarrollar aplicaciones bajo Windows consultando bases de datos relacionales en forma cliente - servidor. Por otra parte resulta una guía para el aprendizaje de los estándares ODBC y OLE, aportando lo necesario para un buen start-up. De igual manera, destaco una comparación útil a la hora de elegir un motor de base de datos. Llevado a la actualidad, sería un interesante proyecto migrar VQL a una arquitectura de Web Application, que parece ser el modelo que se viene.

Apéndice A: Acrónimos.

API: Application Programming Interface
DBMS: DataBase Management System
DDE: Dinamic Data Exchange
DLL: Dynamic Link Library
GUI: Graphical User Interface.
MDI: Multiple Document Interface
ODBC: Open DataBase Connectivity.
OLE: Object Linking & Embedding
RDBMS: Relational DataBase Management System
SDK: Software Development Kit
SQL: Structured Query Language

Apéndice B: Instalación.

Pasos a seguir para la instalación de VQL, tanto en programas ejecutables y librerías como fuentes y documentación:

1. Ejecutar el programa instalar *d:* en el disco VQL 1, donde *d:* es el drive destino donde se instalará el software que contiene comprimidos todos los componentes.

2. Una vez instalado se genera una estructura de directorios:
   VQL
      BIN (ejecutables y librerías)
      DEMO (archivo VQL para demostración del runtime).
         ICONS (íconos de la demo)
      DOCS (documentación)
      ICONS (biblioteca de íconos)
      SOURCE (programas fuente, Visual Basic Projects)
         fuentes comunes
         BITMAPS (bitmaps e íconos)
         CUSTOM (Custom Query)
         LIB (vbx no standard utilizados)
         RUNTIME (VQL Runtime)
         SETUP (Preset Setup)

3. Crear el grupo de programas para los ejecutables:
   VQL\BIN\CUSTOM.EXE
   VQL\BIN\SETUP.EXE
   VQL\BIN\RUNTIME.EXE

4. Enjoy!

Runtime en el usuario final:

1. Ejecutar el programa instalar *d:* en el disco VQL Runtime 1, donde *d:* es el drive destino donde se instalará el software que contiene comprimidos todos los componentes.

Una vez instalado se genera el directorio VQL.

3. Instalar el archivo .VQL con su directorio de íconos (el path debe ser el mismo que se incluyó en el Setup (variable *IconProdPath).*

4. Asociar mediante el File Manager la extensión .VQL con el d:\VQL\RUNTIME.EXE.

4. Listo para ejecutar .VQL's.

Apéndice C: Códigos de errores fatales del runtime

Parsing del archivo VQL

10 : No se ha encontrado la variable LoginTimeOut en la sección Preferencias.
11 : No se ha encontrado la variable QueryTimeOut en la sección Preferencias.
12 : No se ha encontrado la variable IconProdPath en la sección Preferencias.
13 : No se ha encontrado la variable Nombre en la sección Base de Datos.
14 : No se ha encontrado la variable DBM en la sección Base de Datos.
15 : No se ha encontrado la variable Tablas en la sección Base de Datos.
16 : No se ha encontrado una variable Tabla*xx* en la sección Tablas.
17 : No se ha encontrado la variable Campo*xx* en la sección *tabla*.

Apéndice D: Demo.

Se utiliza para el demo la base de datos biblio que viene como ejemplo en los productos de Microsoft, base de datos Access. Existe un archivo llamado *biblio.vql*, generado mediante Preset Setup, que puede ser interpretado por el Runtime o bien accedido con Preset Setup para hacer alguna modificación y observar el comportamiernto.

Resulta además interesante asociar para Windows mediante el File Manager por ejemplo, la extención VQL al VQL Runtime, de modo de acceder automáticamente el Runtime cuando se ejecuta un archivo VQL.

He aquí biblio.vql

[Preferencias]
LoginTimeOut= 20
QueryTimeOut= 60
IconDevPath=.\icons
IconProdPath=.\icons

[OLE]
Excel= -1

[Base de Datos]
DBM= 2
Tablas= 4
Nombre=.\BIBLIO.MDB

[Tablas]
Tabla0=Authors, Autores, AUTORES.ICO,-1, 2
Tabla1=Publisher Comments, Editor Comments, COMMENTS.ICO,-1, 3
Tabla2=Publishers, Editores, EDITORES.ICO,-1, 9
Tabla3=Titles, Libros, LIBROS.ICO,-1, 5

[Authors]
Campo0=Au_ID, Identificador,-1, 0
Campo1=Author, Nombre,-1, 0

[Publisher Comments]
Campo0=PubID, Identificador,-1, 0
Campo1=Publisher, Nombre,-1, 0
Campo2=Comments, Comentario,-1, 0

[Publishers]
Campo0=PubID, Identificador,-1, 0
Campo1=Name, Nombre,-1, 0
Campo2=Company Name, Empresa,-1, 0
Campo3=Address, Dirección,-1, 0
Campo4=City, Ciudad,-1, 0
Campo5=State, Estado,-1, 0
Campo6=Zip, Codigo Postal,-1, 0
Campo7=Telephone, Telefono,-1, 0
Campo8=Fax, Fax,-1, 0

[Titles]
Campo0=Title, Titulo,-1, 0
Campo1=Year Published, Año de Edición,-1, 0
Campo2=Au_ID, Autor ID,-1, 0
Campo3=ISBN, ISBN,-1, 0
Campo4=PubID, Editor ID,-1, 0

Apéndice E: Guía a los programas fuente.

VQL ha sido desarrollado en Microsoft Visual Basic 3.0 Profesional Edition, por lo tanto es necesario disponer del mismo en su instalación completa en función de acceder al código fuente, que está grabado en formato binario.

El diskette *Fuentes* contiene un directorio *source*, con tres archivos:

vbconst.bas:	definición de constantes generales.
dataconst.bas:	definición de constantes para bases de datos.
utils.bas:	rutinas de utilería general.

Por otra parte existen cuatro subdirectorios de *source*:

custom:	VQL Custom Query, el archivo que contiene la organización de proyecto es *custom.mak*.
runtime:	VQL Preset Runtime, el archivo que contiene la organización de proyecto es *runtime.mak*.
setup:	VQL Preset Setup, el archivo que contiene la organización de proyecto es *setup.mak*.
bitmaps:	Contiene íconos y bitmaps.
lib:	Contiene los vbx utilizados.

En todos los casos, abriendo con Visual Basic los archivos de proyecto (.MAK), es posible acceder a los componentes de cada uno, he aquí una breve explicación:

Custom:

MAIN.BAS	: Definiciones y rutinas de uso general para este programa, incluyendo la sub main que consiste en el punto de entrada al programa.
..\VBCONST.BAS	: Definición de constantes generales.
..\DATACONS.BAS	: Definición de constantes para bases de datos.
..\UTILS.BAS	: Rutinas de utilería general.
MDIMAIN.FRM	: Form principal.
ABOUT.FRM	: Form con información del sistema.
QUERYFOR.FRM	: Form de armado de consulta.
JOINFORM.FRM	: Form de armado de junta.
RESULTFE.FRM	: Form de display del resultado.
SELFORM.FRM	: Form de armado de selección.
MSOLE2.VBX	: VBX para acceder a OLE 2.0.
CMDIALOG.VBX	: VBX para *common dialogs* (abrir archivo, etc.)
THREED.VBX	: VBX para controles standard de windows.
..\LIB\SSDATA1.VBX	: VBX para grilla de datos.
..\LIB\VSVBX.VBX	: VBX confunciones de interfase gráfica.

Runtime:

MAIN.BAS	: Definiciones y rutinas de uso general para este programa, incluyendo la sub main que consiste en el punto de entrada al programa.
..\VBCONST.BAS	: Definición de constantes generales.
..\DATACONS.BAS	: Definición de constantes para bases de datos.
..\UTILS.BAS	: Rutinas de utilería general.
MDIMAIN.FRM	: Form principal.
QUERYFOR.FRM	: Form de armado de consulta.
JOINFORM.FRM	: Form de armado de junta.
RESULTFE.FRM	: Form de display del resultado.
SELFORM.FRM	: Form de armado de selección.
MSOLE2.VBX	: VBX para acceder a OLE 2.0.
CMDIALOG.VBX	: VBX para *common dialogs* (abrir archivo, etc.)
THREED.VBX	: VBX para controles standard de windows.
..\LIB\SSDATA1.VBX	: VBX para grilla de datos.
..\LIB\SSDATA2.VBX	: VBX para combo box de datos.
..\LIB\VSVBX.VBX	: VBX con funciones de interfase gráfica.
..\LIB\BMPLST1.VBX	: VBX para lista de bitmaps.

Setup:

MAIN.BAS	: Definiciones y rutinas de uso general para este programa, incluyendo la sub main que consiste en el punto de entrada al programa.
..\VBCONST.BAS	: Definición de constantes generales.
..\DATACONS.BAS	: Definición de constantes para bases de datos.
..\UTILS.BAS	: Rutinas de utilería general.
MDIMAIN.FRM	: Form principal.
ABOUT.FRM	: Form con información del sistema.
FASE1.FRM	: Form para la fase 1 del setup wizard.
FASE2.FRM	: Form para la fase 2 del setup wizard.
FASE3.FRM	: Form para la fase 3 del setup wizard.
CMDIALOG.VBX	: VBX para *common dialogs* (abrir archivo, etc.)
THREED.VBX	: VBX para controles standard de windows.
..\LIB\SSDATA1.VBX	: VBX para grilla de datos.
..\LIB\VSVBX.VBX	: VBX con funciones de interfase gráfica.

En todos los forms, las rutinas privadas se encuentran definidas en la sección *general* del form, por otra parte puede analizarse el código correspondiente a los diversos eventos, tanto de los forms como de los diferentes controles ubicados en cada form.

Bibliografía:

Jenings, "Database developers guide with Visual Basic 3", Sams Publishing, 1993.

Hursch, "SQL, Structured Query Language", Rama, 1989.

Slalemy, "Client-Server computing with Oracle", Ziff Babies, 1993.

MSDN Level II Vol. 7, Developer's Library, Microsoft, 1994.

MSDN Level II Vol. 8, Developer's Library, Microsoft, 1194.

MSDN Level II Vol. 9, Developer's Library, Microsoft, 1195.

MSDN Level II Vol. 10, Developer's Library, Microsoft, 1195.

MS Technet January, Microsoft, 1995.

MS Technet March, Microsoft, 1995.

MS Technet June, Microsoft, 1995.

MS Technet September, Microsoft, 1995.

MS Office Development Kit, Microsoft, 1995.

MS Windows Resource Kit, Microsoft, 1995.

MS ODBC Programmer's Reference, Microsoft, 1995.

Manuales de los siguientes productos:

SCO Unix System V.
Novell Netware v. 3.12.
Microsoft Windows NT v. 3.5.
Oracle 7.