UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"ARAGON"

FUNDAMENTOS PARA LA ELABORACIÓN DE APLICACIONES CLIENTE-SERVIDOR UTILIZANDO JAVA - JDBC.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A N :

PATRICIA PEREZ RUIZ
LISANDRO LOPEZ VILLATORO

DIRECTOR: ING. ERNESTO PEÑALOZA ROMERO

MEXICO D.F.

JUNIO 2002

Con todo mi amor y respeto a mis padres:
Delfina Ruiz García
Lucio Pérez Acevedo
Y con el cariño a mis hermanos:
Melitón Pérez Ruiz
Juan Manuel Pérez Ruiz
Alberto Pérez Ruiz

Con todo mi amor y respeto a mis padres:
María Esthela Villatoro Morales
Lizandro López Morales
Y con el cariño a mis hermanos:
Romeo López Villatoro
José Antonio López Villatoro

Licenciado

AGRADECIMIENTOS

A DIOS

Por haberme dado la dicha de cumplir la más bella de sus creaciones: la vida.

A MIS PADRES

Con gran respeto y admiración por toda una vida de lucha, sacrificio y esfuerzo que realizaron para que pudiera concluir una carrera profesional.

Gracias por el cariño que siempre han depositado en mi, por sus consejos y sobre todo por haber sido mi aliento y mi fuerza en todos los momentos de mi vida.

A MIS HERMANOS

Por haber crecido juntos y haber compartido toda una vida con el apoyo incondicional que siempre me han brindado.

A LA UNAM

Por haberme dado la oportunidad de ser profesionista y por todo lo que aprendí en mi estancia en ella.

Con la elaboración de este documento y haciendo una retrospección estudiantil, me siento realmente contento de haber concluido mis estudios a nivel **Licenciatura**.

Por lo que quiero darle las gracias a todos mis profesores de todas mis escuelas, que depositaron en mi sus conocimientos y experiencia.

A todos mis compañeros de salón que me dieron su amistad y apoyo.

A la Máxima Casa de Estudios: **Universidad Nacional Autonoma de México**, por permitirme crecer y creer en ella.

A **mi familia**, por todo el apoyo, cariño y comprensión que me ayudo a llegar aquí.

A **Dios** por darme la oportunidad de disfrutar la vida.

Por último a todas las personas que directa o indirectamente me ayudaron a concluir mis estudios.

INDICE

OBJETIVO

Debido al crecimiento acelerado del uso de Java como nuevo lenguaje capaz de solucionar aspectos que hace tiempo ni se pensaba. Se pretende elaborar un documento en el cual se plasme los fundamentos para la elaboración de una aplicación que permita la conexión a bases de datos. Dando a conocer los aspectos importantes de JDBC y probando su correcto funcionamiento de la teoría con la realidad, de la documentación existente de JDBC.

CAPÍTULO I

LENGUAJE JAVA

Introducción

El lenguaje de programación Java es uno de los progresos mas importantes en la computación, gano aceptación mundial con una rapidez sin precedentes. En los últimos años Java ha asumido un papel crucial en el desarrollo de tecnologías de red y en el perfil del web, la promesa de tener un entorno de programación que no es especifico de la plataforma han hecho de Java un lenguaje poderoso y popular.

El presente capitulo relata la historia de Java como un sistema que surgió de la frustración de un grupo de investigadores como medio de conseguir un fin, en este caso los fines no solo justificaban los medios , sino que los medios se convirtieron en el objetivo final.

Posteriormente se explica que es Java y para que sirve, además de hablar sobre el ambiente de desarrollo de Java y sus principales componentes.

Para terminar este capitulo se enlistan las caracteristicas principales que han hecho que este lenguaje de programación reciba una publicidad sin precedentes.

1.1 Historia de Java

La creación del lenguaje de Programación Java tiene una historia peculiar: a finales de 1991 el programador *James Gosling*[1] inicio un proyecto de investigación para la empresa *Sun Microsystem*, llamado *Green* que consistía en desarrollar programas para aparatos electrodomésticos. En este proyecto, Gosling pretendía que artículos como tostadores, lámparas, hornos de microondas, videocaseteras, etc., tuvieran un grado mínimo de inteligencia y se comunicaran unos con otros, sobre todo, tratando de hacerlos más sencillos y libres de errores.

Una vez concluido el proyecto *Green*, se obtuvo un prototipo llamado *Star 7 (* 7)*, el cual era un control remoto en el que se presentaba una interfaz basada en la representación de una casa en forma animada y se operaba mediante una pantalla sensible al tacto, con lo que el usuario del *Star 7* podía navegar con sólo la yema del dedo por un universo de salas y objetos. En este sistema aparecía un simpático personaje llamado "Duke", quien es ahora la mascota de Java (figura 1.1). La capacidad más destacada de un dispositivo *Star 7*, era comunicarse con otro dispositivo *Star 7*,

FIGURA 1.1

En un principio el software de *Star 7* estaba planeado para que fuera desarrollado en *C++*, pero James Gosling, como miembro del equipo con mayor experiencia en lenguajes de programación, se negó, argumentando que la eficiencia de *C++* no compensaba el gran costo en pruebas y depuración. Por otra parte, los programas creados en *C* o *C++* deben ser compilados para un chip o procesador y si ese chip es cambiado, todo el programa debe compilarse nuevamente, lo que hace que los desarrollos sean más caros.

[1]Creador del editor EMACS UNIX y el sistema de ventanas NeW

Y tomando en cuenta que en el campo de la electrónica la reducción de costos es primordial, surge un gran problema debido a que la aparición de chips más baratos y generalmente más eficientes, conduce inmediatamente a los fabricantes a incluirlos en las nuevas series de producción. Por pequeña que sea la diferencia en el precio, al ser multiplicada por la tirada masiva de aparatos, da lugar a un considerable ahorro, por lo que se buscó un lenguaje de programación que generara programas que pudieran trabajar en cualquier sistema; es decir, un lenguaje independiente de la plataforma.

A partir de entonces, Games Gosling trabajo en un nuevo lenguaje de programación, partiendo de la sintaxis de C++, por ser ya conocido por la mayoría, pero intentando remediar las deficiencias que él había observado cuando programaba en ese lenguaje.

Desde el principio, el lenguaje de Gosling fue diseñado para ser simple y fácil de aprender y sobre todo de usar. A este nuevo lenguaje le llamó *Oak* (Roble), inspirado en un árbol que podía verse desde la ventana de su oficina.

Finalmente el proyecto *Green* contaba con un impresionante dispositivo de demostración, un sistema operativo *(Green*OS) y su lenguaje de programación (*Oak*); pero aún así no obtuvo el éxito esperado.

En Noviembre de 1992, el equipo de desarrollo se incorpora al proyecto *FirstPerson, Inc.*, el cual enfoco sus esfuerzos a la industria de la Televisión interactiva, Pero entonces perdieron un curso para producir un equipo de televisión de alta calidad para Time-Warner.

En el año de 1993, cuando *FirstPerson* aún trabajaba para la televisión interactiva, se liberó el primer visualizador para *World Wide Web*, *Mosaic 1.0* desarrollado por *Marc Andreesen*, un estudiante universitario que era empleado del Centro Nacional de aplicaciones de Súper Computo.

Como ninguno de los programas desarrollados en *Oak* se convirtió en un sistema comercial, *Sun* se dio cuenta de que la televisión interactiva no sería jamás un gran éxito, por lo que *FirstPerson* se dedico a desarrollar con rapidez estrategias que les produjeran beneficios económicos, pero a pesar de todos los esfuerzos realizados, *FirstPerson* cierra sus puertas en 1994.

El *Mosaic 1.0* llegó a ser un fenómeno internacional y ayudó a que el *World Wide Web* llegará a ser un medio masivo y por otro lado, la tecnología utilizada en *Oak* era bien favorecida para este medio, especialmente a causa de su capacidad para correr sobre múltiples plataformas, por lo que *Sun* se percató de que tenía un negocio prometedor en sus manos; sin embargo, de pronto se encontró con un pequeño problema: no podía usar, el nombre de *Oak* porque ya había un producto más antiguo con ese nombre.

Después de múltiples jornadas para sustituir el nombre *Oak* por otro de igual atractivo, en Enero de 1995, *Oak* fue renombrado como *Java*. Existen varias versiones del origen del nombre de Java: la primera dice que se inspiraron en una importante y bella isla de Indonesia con ese nombre; otros argumentan que es un acrónimo que significa: Simplemente Otro Acrónimo Vago (Just Another Vague Acronym); y otros, que le pusieron así simplemente porque se escucha agradable.

Pero éste lenguaje proviene de un gran esfuerzo y de cuatro principales alcances del proyecto *Green*: el lenguaje Oak, el sistema operativo *GreenOS*, la interface para el usuario y el prototipo *Star 7*.

De las jornadas de trabajo para escoger el nombre de Java llamadas *coffee breakes* surge también la figura de la tasa de café que caracteriza a Java .

A mediados de 1994 el crecimiento de la popularidad de la *Web* atrajo la atención del equipo. Decidieron que podían construir un excelente visualizador utilizando la tecnología Java. Con el objetivo

de llevar al *Web* su sistema de programación en tiempo real independiente de la plataforma, construyeron un visualizador *Web*. Este visualizador, llamado *WebRunner*, se programó utilizando Java y se terminó a finales de 1994, los ejecutivos de *Sun* quedaron impresionados y vislumbraron las posibilidades tecnológicas y comerciales que podrían derivarse del nuevo visualizador: herramientas, servidores, ambientes de desarrollo, etc.. El 23 de Mayo de 1995, *Sun Microsystem* presentó formalmente a Java y *HotJava* en la exposición *SunWorld '95* de San Francisco.

Ya con su nuevo nombre, Java se encontró con un admirador: Andreesen, quien para esos tiempos era vicepresidente de Netscape; empresa que licenció a Java para usarlo en sus visualizadores o navegadores, poniendo el lenguaje ante millones de usuarios. La primer versión "Beta" de Java, se hizo disponible en la red, en el mismo año de 1995.

Por su parte, la empresa *Sun* hizo un kit para desarrolladores y código fuente, para que su producto fuera disponible para todo aquel que quisiera usarlo.

La conclusión de esta historia es que a pesar de que hoy en día los electrodomésticos no son más inteligentes que en 1991, a cambio tenemos un poderoso lenguaje de programación entre nosotros.

1.2 ¿Qué Es Java? Y ¿Para Qué Sirve?

Java es un poderoso lenguaje de programación Orientado a Objetos muy parecido a C++ con el cual se pueden desarrollar aplicaciones completas como hojas de cálculo, procesadores de texto, programas para redes, bases de datos, etc., pero eso no es todo, Java aparte de ser un lenguaje de propósito general permite ir más allá de la naturaleza estática de las páginas Web ya que con él se desarrollan programas llamados *applets* que son insertados en páginas HTML, los cuales brindan gran interactividad al world wide web y permiten el uso intenso de gráficos, sonidos, multimedia, o grandes cantidades de datos.

Frecuentemente a Java se le describe como "C++ menos" a causa de que algunos de sus elementos se omitieron y esto debido a que *James Gosling* quiso evitar los problemas que el proyecto "Green" había encontrado al usar C++ cuando desarrollaba el prototipo *Star 7*, por lo que las partes más complejas de C++ se excluyeron de Java, tal como gestión de memoria y los apuntadores, los cuales son muy complicados de usar, la gestión de memoria ocurre automáticamente en Java, los programadores no tiene que escribir su propio recolector de basura ya que cuenta con rutinas especiales que liberan la memoria.

A los programadores experimentados se les puede facilitar el aprendizaje de Java, debido a su similitud con C, pero pueden tener problema al ajustarse a algunos de los cambios y reducciones de "C++", sin embargo, los arquitectos de Java se propusieron hacer un lenguaje más fácil de escribir y aprender, lo más importante es que lo lograron.

Para desarrollar un programa, lo primero es obtener el ambiente de desarrollo Java y es suficiente tener nada más el Java Development Kit (JDK), el cual esta disponible para bajarse de la red en cualquier momento, aunque es importante señalar que únicamente es una interfaz de línea de comandos, por lo que se necesitará un editor de texto por separado para realizar los archivos del código

fuente. Existen otros ambientes de desarrollo Java, todos tienen un ambiente gráfico de desarrollo y que incluyen su JDK, que también pueden ser bajados de la red en versiones limitadas o de prueba y se describen en la siguiente tabla (Ver tabla 1.1).

El sitio de donde se puede obtener el JDK es: *java.sun.com*

Producto	Fabricante	Plataforma	Sitio
Java Workshop	Sun	Solaris, Windows NT y Windows 95	www.sun.com/sunsoft/Developer-products/java/index.html
Café	Symantec	Windows NT y Windows 95 y Macintosh	cafe.symantec.com/
Visual J++	Microsoft	Windows NT y Windows 95	www.microsoft.com/visualj/
Technologies' Roaster	Technologies' Roaster	Macintosh	http://www.roaster.com/roaster/
SuperCede	Asymetrix	Windows NT y Windows 95	www.asymetrix.com/products/supercede/
NetCraft	SourceCraft	Windows NT 3.5 y Windows 95	www.sourcecraft.com:4800/about/netcraft/
WinGEN for Java	Pro-C	Windows NT 3.5 y Windows 95	www.pro-c.com/products/wfj/java.html
Jfactory	Rogue Wave	Windows 95, Windows NT, SPARC Solaris 2.4 or 2.5, HP-UX 10.01, IBM OS/2 Warp 3.0	www.roguewave.com/products/jfactory/jfactory.html
CodeWarrior	Metrowerks	Windows NT y Windows 95 y Macintosh	www.metrowerks.com/products/

Tabla 1.1

Es importante señalar que estos ambientes de desarrollo, aunque se basan en los mismos principios, no son compatibles entre si al 100% ya que cada fabricante puso algunos estándares o elementos adicionales, por ejemplo Visual J++, además de agregar el uso de objetos ActiveX, efectúa sus conexiones a bases de datos con ODBC (DAO) mientras que los otros se fundamentan en JDBC.

Cuando se habla de JDK 1.1 o JDK 1.2 nos referimos a un entorno completo de aplicaciones (la máquina virtual de Java, o Java Virtual Machine más las librerías de clases) que se emplea para dos propositos distintos:

Ejecución de Aplicaciones: Las aplicaciones son programas independientes con los mismos derechos y responsabilidades que los programas en otros lenguajes. Como los programas C++, los programas Java independientes comienzan con una llamada a main() y finalizan, normalmente con una

llamada exit(). Por regla general, un programa independiente se ejecuta invocando a la JVM y ejecutando un archivo de clases.

Ejecución de Applets: Los applets se ejecutan en navegadores, incrustados en páginas web, normalmente bajo el control de un entorno de ejecución Java (JRE Java Runtime Environment) construido dentro del navegador.

Java También define otros entornos mas sencillos para uso de aplicaciones mas reducidas.

PersonalJava

Un subconjunto de JDK1.1 para dispositivos personales como los PDA, o asistentes digitales personales.

EmbeddedJava

Un subconjunto de JDK1.1 para uso en controladores integrados, con extensiones dirigidas a entornos de tiempo real. EmbeddedJava es un tema conflictivo en estos momentos, ya que un cierto numero de vendedores con experiencia en sistemas de tiempo real quedaron tan descontentos con el trabajo EmbeddedJava de Sun, que formaron el J-Consortium a comienzos de 1999 para trabajar en pos de unas mejores extensiones de Java en tiempo real, independientes del fabricante.

JavaCard

Un entorno para uso con tarjetas inteligentes, diseñadas para soportar los requisitos de transaccion y aplicación de este mercado.

JavaTV

Un entorno Java para uso con aplicaciones a través de la televisión, como progración interactiva y video bajo demanda.

JavaPhone

Un conjunto de extensiones de la API, sobre PersonalJava o Embedded Java, para desarrollo de aplicaciones de telefonia.

Sun tiene otros componentes enfocados a Java, al margen de la plataforma central, entre ellas es conveniente citar:

Java3D. Soporte para imagenes 3D.

JavaMedia Framework. Soporte de multimedia.

Java Servlets. Java para servidores Web.

JavaCryptographyExtensions. Un marco para criptografia de clave privada y publica.

JavaHelp. Un sistema completo de ayuda.

Jini. Un marco para crear comunidades de dispositivos inteligentes, incluyendo configuración automática de redes y busqueda de recursos.

JavaSpeech. Una Api para sintesis y reconocimiento de voz.

Java 2 Enterprise Edition. Una colección de tecnologías, directorio, bases de datos, correo electrónico, mensajeria, transacciones orientadas al despliegue en entorno empresarial.

El Java Development Kit cuenta con las herramientas necesarias para desarrollar poderosos programas y se listan en la tabla 1.2

Executable	Nombre de la Herramienta	Descripción
appletviewer	Java applet viewer	Despliega applets.
Java	Java interprete	Corre Java bytecode.
Javac	Compilador Java	Compila programas Java a bytecodes
javadoc	Generador de documentos Java	Crea documentación en forma HTML del código fuente Java.
Javah	Generador de archivos stubs o de declaraciones	Crea archivos de declaraciones (cabecera o *.h*) como los de C y crea objetos compilados para el manejo de RMI.
Javap	Desensamblador	Convierte archivos bytecode Java a una representación de código fuente
Jdb	Debugger	Se encarga de ir barriendo el código para ver la forma en que se comporta el código al estarse ejecutando.

Tabla 1.2

A continuación se describen con mayor detalle dichas utilerías:

Todos las utilerías que en seguida se mencionan, son llamados desde el modo de comandos del sistema operativo, a excepción del sistema Macintosh.

appletviewer

Esta herramienta sirve como prueba mínima del *applet* que se esté construyendo, se puede utilizar en lugar de utilizar todo un navegador más extenso y que al estarle realizando pruebas pudiera ser muy lento cargar el *applet* varias veces. Para poderlo utilizar, se debe de crear una página HTML que contenga el código suficiente para llamar el *applet*. El único inconveniente es que, si en la misma página HTML existe más código para presentar más cosas o formatos, como pudieran ser tablas, imágenes, etc. no serán mostrados, únicamente se mostrará el o los *applets* que contenga. Se puede llamar como sigue:

appletviewer *Opciones URL*
URL. Es la ubicación en donde se encuentra la página HTML que contiene incrustado un *applet*, si se está en el directorio en donde esta la página, sólo se deberá de proporcionar el nombre de la página, sin importar mayúsculas o no.
Opciones.
-debug Inicia el *appletviewer* en el *debugger*
-jdb Permite *debugear* el *applet* dentro del documento HTML.

Java

Esta herramienta sirve como interprete en tiempo de ejecución de las aplicaciones independientes (*stand alone*) siempre y cuando ya estén compiladas en *bytecode*, ésta herramienta funciona como un comando en modo consola, las aplicaciones pueden tener o no parte gráfica la cual manejará por si misma sus propias ventanas. Se puede llamar como sigue:

java Opciones ArchivoClass Argumentos

Opciones:

-help	Despliega la ayuda de las posibles opciones que se le pueden aplicar al interprete de Java.
-version	Muestra la versión del JDK que esta instalado
-v (-verbose)	Mostrará todas las clases que sean cargadas en la ejecución de la aplicación.
-cs (-checksource)	Verifica si la version del código fuente es más nueva que la de la clase, de ser así se compila el código fuente para generar una nueva clase.
-noasyncgc	Apaga la asincronización del recolector de basura.
-verbosegc	Mostrará mensajes de cada vez que sea utilizado algo del recolector de basura.
-verify	Verifica que todas las clases sean cargadas.
-noverify	Apaga la verificación anterior.
-verifyremote	Verifica las clases importadas o inherentes. Esta opción se maneja por omisión.
-mx val	Establece el tamaño máximo de la pila Java especificado por *val*. El valor mínimo permitido es 1K (-mx 1k), el valor por omisión es de 16MB (-mx 16m). Se debe de utilizar "m" o "k" para especificar *megabytes* o *kilobytes* respectivamente
-ms val	Establece el tamaño inicial de la pila Java especificado por *val*. El valor mínimo permitido es 1K (-mx 1k), el valor por omisión es de 1MB (-ms 1m). Se debe de utilizar "m" o "k" para especificar *megabytes* o *kilobytes* respectivamente.
-ss val	Establece el valor del tamaño de la memoria reservada (*stack*) para algún proceso en C especificado por *_val*. El valor permitido debe ser mayor a 1K (-ss 1k). Se debe de utilizar "m" o "k" para especificar *megabytes* o *kilobytes* respectivamente
-oss val	Establece el valor del tamaño de la memoria reservada (*stack*) para procesos en Java especificado por *val*. Se debe de utilizar "m" o "k" para especificar *megabytes* o *kilobytes* respectivamente
-debug	Permite ejecutar la aplicacion en forma de poderla *debugear*
-prof	Genera la información del perfil de los datos en forma de archivo dentro de *\JAVA.PROF*
-classpath dirs	Indica en que directorio deberá de buscar los archivos *.class*, para separar la indicación de varios directorios se deberá de utilizar una coma (en UNIX) o punto y coma (en ambientes DOS)

ArchivoClass: Indica que archivo compilado (*.class*) será la aplicación a ejecutar, si la clase se encuentra dentro de un paquete se deberá de indicar el paquete completo, sin incluir la extensión del archivo

Argumentos: Indica que la aplicación puede esperar parametros para su ejecución, es decir, como toda aplicación ejecutada de esta manera son clases que cuentan con el metodo *main()* y este metodo sera el que tome los argumentos adicionales y opcionales que se le manden a la aplicacion al ejecutarla

javac

Esta herramienta sirve para compilar los archivos con código fuente (.java) en archivos ejecutables (utilizando el interprete java) con la extensión *.class*, Si el archivo con código fuente incluyera más de una clase, al compilarse se generarán los archivos *.class*, uno por cada clase que se crea. Si dentro del código fuente se esta creando un paquete, en el momento de compilarse se crea la

estructura de directorios necesarios y en el último dejará el archivo o archivos *.class* pudiéndosele indicar que también los cree en el directorio CLASSPATH, para que estén disponibles de inmediato. Se puede llamar como sigue:

javac Opciones ArchivoJava
Opciones:

-g	Guarda la información integra del depurado en los archivos .class
-O	Optimiza el código para un mejor rendimiento en tiempo de ejecución.
-nowarn	Anula los mensajes de aviso
-verbose	Imprime mensajes informando de que archivos de clases estan compilandose
-classpath path	Establece la ruta de acceso a las clases donde hay que buscar durante la compilación.
-deprecation	Genera errores fatales si se utilizan clases deprecadas
-d dir	Especifica un directorio a donde van a ir los archivos .class
-J<runtime flag>	Especifica las opciones que se van a pasar al ejecutor de aplicaciones

ArchivoJava: Indica que archivo con código fuente (*.java*) se compilará, se le debe incluir la extensión del archivo.

Javadoc

Esta herramienta sirve para crear documento HTML el cual se basa en los comentarios marcados, con los caracteres correspondientes, dentro del código fuente (.java) el propósito es poder consultar una ayuda de la caracteristicas que contiene la clase y dicha ayuda se puede distribuir o publicar en la *Web*. Se puede llamar como sigue:

javadoc Opciones ArchivoJava
Opciones

-verbose	Muestra más información del archivo que séesta documentando.
-d directory	Especifica en que directorio se dejara el documento generado.
-classpath dirs	Indica en que directorio debera de buscar las clases que se incluyen dentro del código fuente

***ArchivoJava:* Indica que archivo con código fuente (*.java*) se documentará, se le debe incluir la extensión del archivo.**

Los comentarios deben de tener las etiquetas reservadas para ello, las cuales son buscadas dentro del código fuente y son tomadas para incluirlas dentro del documento de ayuda

@see clase	Pone una liga que dice *See Also* y relaciona el documento HTML correspondiente de la clase mencionada
@see clase#método	Pone una liga que dice *See Also* y relaciona el documento HTML correspondiente de la clase mencionada, en un método o apartado del documento para particularizar.
@param param descr	Describe los parámetros que contiene un método
@version ver	Especificará la versión que se esta creando del código fuente
@author name	Especificara el nombre de la persona que esta creando el código
@return descr	Describe el valor que esta regresando el método
@exception class	Crea una liga de las excepciones que la clase puede llegar a disparar

javah

Esta herramienta sirve para crear archivos de cabecera de C (*.h*) o archivos guía (stub) y genera el código fuente para implementar métodos Java en C. Se puede llamar como sigue:

javah Opciones ArchivoClase

Opciones

-stubs	Crea un archivo guía (stub) en lugar del de cabecera que es el generado por omisión.
-d dir	Indica en que directorio deberá dejar el archivo que cree
-v	Despliega mensajes de lo que realiza cuando generá el archivo
-o archivo	Toda salida que genera al crear el archivo lo deposita en el archivo especificado puede ser un texto regular, un archivo de cabecera (archivo.H) o un guia (stub) (archivo.C).
-version	Muestra la version del *javah*

ArchivoClase: **Indica que archivo (*.class*) del que se generará el archivo, no se le debe incluir la extensión del archivo.**

javap

Esta herramienta sirve para desensamblar o traducir archivos o clases compiladas y que están en *bytecode*. La información que genera es mostrar los datos o métodos públicos o privados, generalmente se utiliza cuando por alguna circunstancia no se cuenta con el código fuente y se necesita conocer un poco más para la implementación o uso de dicha clase. Se puede llamar como sigue:

javap Opciones ArchivosClase

Opciones

-version	Muestra la version del *javap*
-p	Genera informacion de las variables miembro y métodos privados y públicos, por omisión, sólo se mostrarán los públicos.
-c	Desensambla codigo fuente y muestra el *bytecode* producido por el compilador
-h	Genera informacion de una clase particular que puede ser usada en archivos de cabecera de C para utilizarse en programas C.
-classpath dirs	Indica en que directorio debera de buscar los archivos *.class*, para separar la indicación de varios directorios se deberá de utilizar una coma (en UNIX) o punto y coma (en ambientes DOS)
-verify	Verifica que el código fuente no tenga errores y que las clases que hace referencia sean cargadas
-v	Despliega información detallada del código fuente cuando se desensabla

ArchivosClase: Indica que archivo o archivos (*.class*) del que se obtendrá la información, si se desearán revisar varias clases se deberán de separar por un solo espacio, no se le debe incluir la extensión del archivo.

jdb

Esta herramienta sirve para revisar a detalle en tiempo de ejecución el comportamiento del código, comúnmente llamado *debugger*. Se puede llamar como sigue:

jdb Opciones

Opciones

-host comp	Le indica al *jdb* a que computadora *comp* deberá de conectarse para poder revisar el programa o la clase, el nombre de la computadora puede ser por su nombre DNS o su dirección IP.
-password contraseña	Indica con que contraseña se le permitirá al *jdb* la entrada al sistema de archivos.

La explicación detallada de los comandos propios del *jdb* queda fuera del alcance de este documento.

Para desarrollar un programa Java se necesitan fundamentalmente dos partes: un compilador y un interprete, funcionando de la siguiente manera: primero, con la ayuda de un editor, se crea un archivo de texto con la extensión .java (almacenado como archivo de texto plano o sin formato) y el compilador crea uno o más archivos *bytecodes* (conjunto de instrucciones parecidas al código de máquina, pero con la diferencia de que no son específicos para un procesador en particular) con extensión .class. los cuales son ejecutados con el interprete de código de *bytes* también conocido como máquina virtual de Java o interprete de ejecución de Java.

1.3 Características de Java

Java ha sido un poderoso y exitoso lenguaje de programación gracias a las caracteristicas que se describen a continuación:

Arquitectura Neutral

La característica más importante de Java es que es un lenguaje de programación de arquitectura neutral, esto quiere decir que es capaz de generar programas que trabajen en sistemas operativos diferentes, a esta propiedad también se le conoce como Independencia de la Plataforma.

La mayoría del software de computadoras se desarrolla para un Sistema Operativo especifico, por ejemplo si se quisiera que un software corriera tanto en windows como en macintosh se tendrían que desarrollar 2 versiones, una para cada plataforma, lo que implicaría un esfuerzo y un gasto considerable. Java permite que el mismo código se utilice en sistemas distintos, únicamente implementando un interprete para cada plataforma con la ventaja de que el código no se tiene que compilar más de una vez, de esta manera Java logra ser un lenguaje que no depende de una arquitectura computacional definida.

Simplicidad

Java ofrece todas las funcionalidades de un lenguaje potente pero sin las caracteristicas confusas que suelen tener y como es muy similar a C++ un lenguaje mundialmente conocido, Java resulta ser muy sencillo.

Java cuenta con un conjunto de palabras reservadas y sintaxis muy reducidas, por lo que es más fácil de escribir, compilar, depurar y sobre todo de aprender.

Otra decisión de diseño para hacer Java más simple son sus elementales objetos y tipos de datos ya que obliga a la declaración explícita de métodos, reduciendo las posibilidades de error. Maneja

la memoria para eliminar las preocupaciones por parte del programador de liberación o corrupción de memoria. También implementa los array auténticos con comprobación de limites en vez de listas enlazadas de punteros para evitar la posibilidad de sobre escribir o corromper la memoria resultado de punteros que señalan a zonas equivocadas. Java permite que los errores en el uso de variables sean encontrados cuando el programa se compila, más bien que dejarlos en un programa que este corriendo donde ellos son más duros encontrar. Como resultado, los programas comportan en una manera más predecible.

Existen muchos lenguajes de programación que se enorgullecen de su versatilidad a la hora de proporcionar formas de realizar lo mismo de distintas maneras, aunque puede ser una característica poderosa para el programador, en Java debido a su simplicidad, ofrece un número reducido de maneras de realizar una tarea dada.

Orientado A Objetos

La programación orientada a objetos (OOP) es una manera poderosa de organizar y desarrollar software que facilita la construcción de sistemas completos a partir de componentes llamados objetos, los cuales existen independientemente y tienen reglas para comunicarse unos con otros.

Java presenta todas las características necesarias para la programación orientada a objetos como la herencia, el polimorfismo y la encapsulación.

Muchos de los conceptos de la programación orientada a objetos son heredados de "C++", pero Java también toma conceptos prestados de otros lenguajes de ese tipo como "Smalltalk" que es un lenguaje de programación muy rico en conceptos orientados a objetos, en "Smalltalk" todo es un objeto incluyendo clases y tipos base.

Como la mayoría de los lenguajes orientados a objetos, Java incluye un conjunto de bibliotecas de clase que ofrecen tipos de datos básicos, capacidad para el manejo de entradas y salidas del sistema, conectividad para redes, protocolos para Internet y otras funciones de gran utilidad y como estas bibliotecas están escritas en Java, se pueden transportar a todas las plataformas como sucede con todas las aplicaciones Java.

Las técnicas de programación orientadas a objetos permiten construir módulos y bibliotecas de software que son reutilizables, lo que permite que posteriormente sean interconectados y de esta manera desarrollar grandes proyectos facilitando el análisis de sistemas.

Seguridad

Java provee seguridad en varios niveles diferentes, en primer lugar, el lenguaje se diseñó para hacer sumamente difícil ejecutar código dañoso y la eliminación de punteros es un paso grande al respecto, ya que los punteros en conjunción con operaciones aritméticas permiten al programador que un puntero haga referencia a un lugar conocido de la memoria y pueda sumar o restar algún valor y referirse a áreas de memoria que son inalterables; otro nivel de seguridad es el verificador de bytecodes, ya que como describimos anteriormente los programas Java se compilan en un conjunto de instrucciones llamadas bytecodes, antes de que un programa Java sea corrido, un verificador checa cada bytecode para asegurarse de que nada sospechoso sucede.

Si los bytecodes pasan la verificación, sin generar ningún mensaje de error, entonces sabemos que:

- El código no produce desbordamientos de operandos en la pila
- Los tipos de parámetros de todos los códigos de operacion son conocidos y correctos
- No ha ocurrido ninguna conversion ilegal de datos, tal como convertir enteros en punteros
- El acceso a los campos de un objeto se sabe que es legal. *public*, *private* o *protected*
- No hay ningún intento de violar las reglas de acceso o y seguridad establecidas

Además de estas medidas, Java tiene varias reglas de seguridad que se aplican a applets, ya que para impedir que un programa pueda realizar actos de violencia contra unidades de disco del usuario, un applet no puede de ninguna manera abrir, leer o escribir archivos sobre el sistema del usuario, y debido a esto, cuando se realizan operaciones con archivo, se trabajan sobre el disco duro de la máquina de donde partió el applet.

Multihilado

Java es Multithreaded (multi-hilo) característica que le ha servido para sobresalir en el campo de la programación, ya que los threads (hilos también llamados procesos ligeros, representan una manera para que un programa de computadora haga más de una tarea a la vez.

El beneficio de ser Multithreaded consiste en un mejoramiento de comportamiento en tiempo real, el cual esta limitado a las capacidades del sistema operativo subyacente, pero independientemente de todo, el Multithreaded supera a los entornos de flujo único de programa (Single-threaded) tanto en facilidad de desarrollo como en rendimiento.

Java posibilita la creación y control de hilos de ejecución explícitamente. La utilización de hilos (*threads*) en Java, permite una enorme flexibilidad a los programadores a la hora de desarrollar aplicaciones. La simplicidad para crear, configurar y ejecutar hilos de ejecución, permite que se puedan implementar muy poderosas y portables aplicaciones/applets que no se puede con otros lenguajes de tercera generación. En un lenguaje orientado a Internet como es Java, esta herramienta es vital.

En los visualizadores que soportan Java, se puede ver el uso de múltiples hilos ya que dos applets se pueden ejecutar al mismo tiempo, o que se puede desplazar la página del navegador mientras el applet continúa ejecutándose. Esto no significa que el applet utilice múltiples hilos, sino que el navegador es multihilo, multihilvanado o multithreaded.

Los visualizadores utilizan diferentes hilos ejecutándose en paralelo para realizar varias tareas, "*aparentemente*" concurrentemente. Por ejemplo, en muchas páginas web, se puede *desplazar* la página e ir *leyendo* el texto antes de que todas las imágenes estén presentes en la pantalla. En este caso, el navegador está trayéndose las imágenes en un hilo de ejecución y soportando el desplazamiento de la página en otro hilo diferente.

Las aplicaciones (y applets) multihilo utilizan muchos contextos de ejecución para cumplir su trabajo. Hacen uso del hecho de que muchas tareas contienen subtareas distintas e independientes. Se puede utilizar un hilo de ejecución para cada subtarea.

Mientras que los programas de flujo único pueden realizar su tarea ejecutando las subtareas secuencialmente, un programa multihilo permite que cada thread comience y termine tan pronto como sea posible. Este comportamiento presenta una mejor respuesta a la entrada en tiempo real.

Familiaridad

Java se dice que es familiar por la similitud que tiene con el lenguaje de programación C. Por lo tanto se supone que es más facil de aprender y aplicar Java como lenguaje de programación. Por lo que además se da por hecho que la curva de aprendizaje de dicho lenguaje, para aquellos que conocen C, es muy corta.

Robustez

Java es robusto ya que contiene todos los elementos necesarios de un lenguaje completo, si se supone que Java surge de los fundamentos de C y además a este se le quitan las cosas inconcistentes o inseguras, como los apuntadores, lo hace un lenguaje con mayor potencial y seguridad que su antecesor.

Portabilidad

Sun ha desarrollado toda una filosofía y por consecuente todo lo necesario para que una aplicación desarrollada en Java se de por hecho que puede ejecutarce en todas (o casi) todas las máquinas virtuales desarrolladas para varios sistemas operativos. Con lo que el programador si necesita migrar cierto código, no se preocupe en todos los cambios que tendria que sufrir su código, si lo hubiera desarrollado en otro lenguaje de programación. A esto Sun le dice: Escribelo una vez y correlo en donde sea.

Interpretado

Java es un lenguaje interpretado, ya que en el momento de la compilación del código, se genera un archivo con instrucciones de la máquina virtual de Java, por lo que en el momento de la ejecución de dicho código, lo que realmente pasa es que la MV, va leyendo el archivo compilado paso a paso, con la finalidad de interpretar que instrucciones se le estan indicando a dicha máquina virtual, con esto se dice que el código Java se puede ejecutar en donde sea.

Resumen

El lenguaje de Programación Java fue desarrollado en *Sun Microsystem* por James Gosling a finales de 1995.

Java es un poderoso lenguaje de programación Orientado a Objetos muy parecido a C++ con el cual se pueden desarrollar aplicaciones completas ademas de desarrollar programas llamados *applets* que son insertados en páginas HTML, los cuales brindan gran interactividad al world wide web y permiten el uso intenso de gráficos, sonidos, multimedia, o grandes cantidades de datos.

Java ha sido un poderoso y exitoso lenguaje de programación gracias a las características que se describen a continuación:

Java es Multithreaded (multi-hilo) característica que le ha servido para sobresalir en el campo de la programación, ya que los threads representan una manera para que un programa de computadora haga más de una tarea a la vez.

Otra característica importante de Java son los niveles de seguridad con que cuenta, en primer lugar, el lenguaje se diseño para hacer sumamente difícil ejecutar código dañoso y la eliminación de punteros es un paso grande al respecto, otro nivel de seguridad es el verificador de bytecodes el cual checa cada bytecode para asegurarse de que nada sospechoso sucede.

La característica más importante de Java es que es un lenguaje de programación de arquitectura neutral, esto quiere decir que es capaz de generar programas que trabajen en diferentes sistemas operativos, Java utiliza programación orientada a objetos pero aun asi, resulta ser muy sencillo, se dice que es familiar por la similitud que tiene con el lenguaje de programación C, aunque se omitieron cosas

inconcistentes o inseguras, como los apuntadores, por lo que Java es un lenguaje con mayor potencial y seguridad que su antecesor.

CAPÍTULO II

BASES DE DATOS RELACIONALES

Introducción

Las computadoras nacieron de la necesidad de manipular y guardar información. Esta información normalmente se accesa desde un ambiente de bases de datos, en la actualidad es muy importante trabajar en el diseño y análisis de sistemas para disponer de información precisa en el momento en que se requiera es por esto que en el presente capítulo se trata de los conceptos y terminología de bases de datos relacionales, comparándolo con otros modelos de datos como el de red y el jerárquico.

El conocimiento de bases de datos relacionales es requerido para usar JDBC, por lo que este capítulo es muy importante para el tema principal de este trabajo

Para finalizar se tratan los comandos básicos de SQL por ser un lenguaje de bases de datos relacionales y que ha llegado a ser popular junto con el modelo relacional de bases de datos

En general, este capítulo brinda un breve resumen de bases de datos relacionales y SQL.

2.1 Historia

Las bases de datos relacionales tienen como principales antecesores a los sistemas de gestión de archivo, las bases de datos jerárquicas y las bases de datos de red, los cuales los describiremos brevemente:

Sistemas de Gestión de Archivos

El sistema de gestión de archivos básicamente no tenía modelo de datos, ya que se almacenaban en archivos individuales y eran procesados por programas que contenían una descripción de los archivos (DA) en la que se describía la estructura de datos, pero se tenía la desventaja de que si esta estructura cambiaba, todos los programas que tenían relación con el archivo tenían que modificarse, por lo que era cada vez más difícil mantener las aplicaciones existentes.

Bases De Datos Jerárquicas

En este modelo, cada registro de la base de datos representa una pieza específica, los registros tenían relaciones padre/hijo que ligaban cada elemento con sus componentes y así sucesivamente, para tener acceso a los datos lo único que se tenía que hacer era:

- Encontrar un elemento en particular a partir de su número
- Descender al primer hijo
- Ascender hasta su padre
- Moverse de lado hasta el siguiente hijo.

Por lo que lo único que se hacía es navegar a través de los registros moviéndose hacia los cuatro lados (arriba, abajo, izquierda, derecha)

Uno de los sistemas de gestión de bases de datos jerárquica más popular fue el *Information Management System* (IMS) de IBM

Bases De Datos En Red

La base de datos jerárquica se convirtió en un gran problema cuando los datos tenían una estructura más compleja por lo que se desarrollo el modelo de datos en red el cual extendía el modelo jerárquico permitiendo que cada registro participara en múltiples relaciones padre/hijo llamadas conjuntos.

En este modelo de datos un programa podría:

- Hallar un registro especifico mediante una clave
- Descender al primer hijo de un conjunto particular
- Moverse lateralmente de un hijo al siguiente dentro del mismo conjunto.
- Ascender de un hijo a su padre en otro conjunto.

Las relaciones de conjunto y la estructura de los registros tenían que ser especificadas de antemano por lo que modificar la estructura de la base de datos requería generalmente la reconstrucción completa de la base de datos.

Modelo de Datos Relacional.

Las desventajas de los modelos jerárquicos y de red provocaron gran interés en el modelo de datos descrito en 1970 por el Dr. Edgar F. Codd en un artículo titulado "A Relational Model of Data for Large Shared Data Banks", este modelo fue favorecido en instituciones académicas debido a sus fundaciones matemáticas en las cuales se eliminaban las estructuras padre/hijo.

Desdichadamente no fue clara la definición matemática del artículo del Doctor Codd por lo que muchas de las bases de datos relacionales no lo eran en realidad, por lo que en 1985, E.F. Codd escribió un nuevo artículo en donde estableció doce reglas a seguir por cualquier base de datos para poder llamarse verdaderamente relacional, las cuales se describen a continuación:

Regla 1. Información-Representación. Toda la información en una base de datos relacional se representa explícitamente a nivel lógico y únicamente mediante valores en tablas.
Regla 2. Acceso garantizado. Se garantiza que cada dato esta accesible lógicamente mediante la combinación nombre de tabla, clave primaria y nombre de columna.

Regla 3 Tratamiento sistemático de valores nulos: Los valores nulos se utilizan para representar información perdida e información inaplicable de una manera sistemática, independientemente del tipo de datos.
Regla 4 Catalogo Dinamico en Linea con base en el modelo relacional: El esquema de base de datos debe almacenarse de la misma manera que los datos para permitir el acceso mediante el mismo lenguaje de consulta relacional
Regla 5. Sublenguaje Global de Datos Al menos uno de los lenguajes de manipulación de datos incluido en una base de datos relacional debe permitir las siguientes operaciones

- Definición de datos
- Definiciones de vistas
- Manipulación de datos
- Restricciones de integridad
- Autorización
- Limite de transacción

Regla 6. Actualización de vistas. Cuando se actualiza la vista (insertar, borrar o actualizar) y el cambio se deriva de las tablas de base de la vista, la base de datos debe activar también la actualización para efectuar el mismo cambio en las tablas de la vista
Regla 7 Inserción a alto nivel. Se debe mantener la capacidad para manejar actualizaciones, inserciones y eliminaciones como operaciones de grupo
Regla 8 Independencia Fisica de los Datos Los cambios en los métodos de acceso o almacenamiento en una base de datos han de ser transparentes a las operaciones ejecutadas sobre los datos
Regla 9 Independencia lógica de los Datos Todo cambio en una base que no implique perdida de datos no afectara a las operaciones que estén siendo ejecutadas en la base de datos en ese momento
Regla 10 Independencia de la Integridad Las restricciones especificas a la integridad de una determinada base de datos relacional deben ser definibles en el sublenguaje relacional y almacenables en el catalogo, no en el programa de aplicacion
Regla 11 Independencia de distribucion El nivel y el tipo de distribución sera claro para los lenguajes de manipulación de datos de este motor de bases de datos
Regla 12 No transtornar Todos los lenguajes de una base de datos relacional debe basar todas las operaciones únicamente en el esquema; No se permite que ninguna operación pueda desviarse de estas definiciones y limitaciones

En la actualidad el modelo relacional se ha establecido como el principal modelo de datos para las aplicaciones de procesamiento de datos.

2.2 Conceptos De Bases De Datos

Una base de datos consiste en una colección de tablas

Una tabla es un arreglo de dos dimensiones compuesto por renglones y columnas, cada tabla en la base de datos representa una entidad de datos.

Una identidad es un objeto identificable o un sustantivo (personas, lugares o cosas) del cual se registra información y cada entidad es una colección de atributos.

Un atributo es una caracteristica o propiedad asociada con las entidades, los atributos se modelan como columnas de la entidad.

Las entidades pueden tener asociaciones *(Relationships)* las cuales son conexiones entre los objetos o entidades modeladas.

Las asociaciones tienen diferentes formas: uno a uno (1:1), uno a muchos (1:M) y muchos a muchos (M:M)

- Asociación Uno a Uno: Indica que por una entidad en particular existe una y solo una entidad relacionada
- Asociación uno a muchos: Indica que por multiples registros en una entidad dada existen una o más de una entidad relacionada
- Asociación Muchos a Muchos: Indica que por multiples registros en una entidad dada existen uno o más registros en una entidad relacionada

Una base de datos es frecuentemente dibujada en un diagrama entidad -relación (ERD), este diagrama provee una serie de lineas, cajas y simbolos que representan las asociaciones (Relationships) entre cantidades.

Normalización

Una vez que los atributos y entidades de las bases de datos han sido identificadas, existe un proceso de normalización que completa el desarrollo de la base de datos.

La normalización es la eliminación de atributos repetidos en una entidad, el desarrollador de la base de datos debe revisar y modificar el apropiado nivel de normalización.

Varios niveles de normalización han sido definidos por el Dr. Codd y otros investigadores, cada nivel examina las entidades y sus atributos y determina a donde pertenece y a donde no pertenece cada entidad.

La normalización de una base de datos relacional es una parte importante del proceso de diseño de base de datos, ya que reduce la redundancia de datos y con esto se reduce el espacio de almacenamiento requerido para los datos. Un diseño normalizado también simplifica el proceso de hacer modificaciones a las bases de datos.

Creación De Tablas

El fin del proceso de diseño es cuando las entidades llegan a ser tablas de la base de datos y los atributos llegan a ser columnas de las tablas. Estas tablas y columnas son luego manipuladas usando SQL, cada tabla puede contener uno o más renglones de datos.

Llaves (Key)

Se denomina llave o clave al atributo que permite localizar de manera única a una entidad.

Llave Primaria (Primary Key, PK)

La llave primaria de una tabla identifica en forma única a cada renglón de la tabla.

Llave Foráneas (Foreign Key, FK)

Es la llave primaria de una tabla y al mismo tiempo parte de otra tabla únicamente como atributo.

Relación

El término relación es un término matemático para una tabla. Una base de datos con 3 tablas sería una base de datos con tres relaciones. El término relación y tabla son frecuentemente usados intercambiablemente.

Dominio

Es frecuentemente usado para describir un rango de valores que son apropiados para un atributo o columna.

Joins

En términos técnicos un Join es una operación que toma 2 tablas como operando y produce una nueva tabla concatenando los renglones con los valores de las columnas, no es necesario que los nombres de las columnas sean los mismos pero si es necesario que tengan el mismo dominio.

Tupla

El termino tupla es equivalente a un registro, una tupla es una colección ordenada de uno o más elementos de datos que forman un registro.

2.3 Comandos Básicos SQL

En los últimos años SQL ha tenido impacto en la gestión de base de datos y ha sido utilizado en mainframes, estaciones de trabajo, computadoras personales y especialmente en redes de área local cliente/servidor.

A continuación se describen las principales sentencias de SQL.

Manipulación De Datos

SELECT	Recupera datos de la base de datos
INSERT	Añade nuevas filas de datos a la base de datos
DELETE	Suprime filas de datos de la base de datos
UPDATE	Modifica datos existentes en la base de datos.

Definición De Datos

CREATE TABLE	Añade una nueva tabla a la base de datos
DROP TABLE*	Suprime una tabla de la base de datos
ALTER TABLE*	Modifica la estructura de una tabla existente
CREATE VIEW*	Añade una nueva vista a la base de datos
DROP VIEW*	Suprime una lista de la base de datos
CREATE INDEX*	Construye un índice para una columna
DROP INDEX*	Suprime el índice para una columna
CREATE SYNONYM*	Define un alias para un nombre de tabla
DROP SYNONYM*	Suprime un alias para un nombre de tabla
COMMENT*	Define comentarios para una tabla
LABEL*	Define el título de una columna

Control De Acceso

GRANT	Concede privilegios de acceso a usuarios
REVOKE	Suprime privilegios de acceso a usuarios

Control De Transacciones

COMMIT	Finaliza la transacción actual
ROLLBACK	Aborta la transacción actual

SQL Programático

DECLARE	Define un cursor para una consulta
EXPLAIN*	Describe el plan de acceso datos para una consulta
OPEN	Abre un cursor para recuperar resultados de consulta
FETCH	Recupera una fila de resultados de consulta
CLOSE	Cierra un cursor
PREPARE*	Prepara una sentencia SQL para ejecución dinámica
EXECUTE*	Ejecuta dinámicamente una sentencia
SQLDESCRIBE*	Describe una consulta preparada

*No forma parte del estándar SQL pero se encuentra en la mayoría de los productos más populares basadas en SQL.

Resumen

Las bases de datos relacionales surgen en 1970 con un articulo titulado "A Relational Model of Data for Large Shared Data Banks", escrito por el Dr. Edgar F. Coudd.

Antes de que surgiera el modelo de datos relacional se trabajaba con los modelos de datos jerárquicos y de red.

Una base de datos relacional consiste en una colección de tablas. Cada tabla en la base de datos representa una entidad de datos y cada entidad es una colección de atributos.

Las entidades se pueden asociar de diferentes formas: uno a uno (1:1), uno a muchos (1:M) y muchos a muchos (M:M).

El desarrollo de la base de datos se completa con la normalización que es la eliminación de atributos repetidos en una entidad.

El fin del proceso de diseño es cuando las entidades llegan a ser tablas de la base de datos y los atributos llegan a ser columnas de las tablas. Estas tablas y columnas son luego manipuladas usando SQL.

Dominio es usado para describir un rango de valores que son apropiados para un atributo o columna.

En términos técnicos un Join es una operación que toma 2 tablas como operando y produce una nueva tabla. El termino tupla es equivalente a un registro.

CAPÍTULO III

EL MODELO CLIENTE/SERVIDOR

Introducción

La tecnología informática juega un papel cada vez más importante en la empresa moderna, pues permite no sólo disminuir el papeleo y en general agilizar las operaciones, sino también lanzar más rápidamente productos al mercado y en general aumentar la competitividad de la empresa. En los últimos tiempos se ha venido haciendo un énfasis cada vez mayor en este aspecto, lo cual ha modificado substancialmente el papel que juega la informática en la empresa, pues además de ser un elemento de apoyo a las operaciones básicas se ha constituido en un medio de obtener ventajas competitivas.

Las tecnologías computacionales modernas buscan responder a las necesidades de las empresas de los 90s y para ello plantean nuevas formas de hacer las cosas. Entre ellas una de las más importantes es el llamado esquema Cliente / Servidor, cuyos principios más importantes se definen a continuación.

En este capítulo se mencionan los pasos necesarios para efectuar una conexión Cliente/Servidor en Lenguaje Java.

3.1 Esquema Cliente Servidor

La arquitectura cliente/servidor es un modelo de computación en el que el procesamiento requerido para ejecutar una aplicación o conjunto de aplicaciones relacionadas se divide entre dos o más procesos que cooperan entre si . Usualmente la mayoria del trabajo pesado se hace en el proceso llamado servidor y el proceso cliente sólo se ocupa de la interacción con el usuario.

Los principales componentes de la arquitectura cliente-servidor son entonces los Clientes, los Servidores y la infraestructura de comunicaciones.

Los Clientes interactúan con el usuario, usualmente en forma gráfica. Frecuentemente se comunican con procesos auxiliares que se encargan de establecer conexión con el servidor, enviar el pedido, recibir la respuesta, manejar las fallas y realizar actividades de sincronización y de seguridad.

Los Servidores proporcionan un servicio al cliente y devuelven los resultados. En algunos casos existen procesos auxiliares que se encargan de recibir las solicitudes del cliente, verificar la protección, activar un proceso servidor para satisfacer el pedido, recibir su respuesta y enviarla al cliente. Además deben manejar los interbloqueos, la recuperación ante fallas, y otros aspectos afines. Por las razones anteriores la plataforma computacional asociada con los servidores es más poderosa que la de los clientes. Por esta razón se utilizan PCs poderosos, estaciones de trabajo, minicomputadores o sistemas grandes. Además deben manejar servicios como administración de la red, mensajes, control y administración de la entrada al sistema ("login"), auditoria , recuperación y contabilidad. Usualmente en los servidores existe algún tipo de servicio de bases de datos.

Para que los clientes y los servidores puedan comunicarse se requiere una infraestructura de comunicaciones, la cual proporciona los mecanismos básicos de direccionamiento y transporte. La mayoria de los sistemas Cliente/Servidor actuales se basan en redes locales y por lo tanto utilizan

protocolos no orientados a conexión, lo cual implica que las aplicaciones deben hacer las verificaciones. La red debe tener características adecuadas de desempeño, confiabilidad, transparencia y administración.

Como ejemplos de clientes pueden citarse interfaces de usuario para enviar comandos a un servidor, APIs para el desarrollo de aplicaciones distribuidas, herramientas en el cliente para hacer acceso a servidores remotos (por ejemplos servidores de SQL) o aplicaciones que solicitan acceso a servidores para algunos servicios.

Como ejemplos de servidores pueden citarse servidores de ventanas como X-windows, servidores de archivos como NFS, servidores para el manejo de bases de datos, como los servidores de SQL, servidores de diseño y manufactura asistidos por computadora, etc.

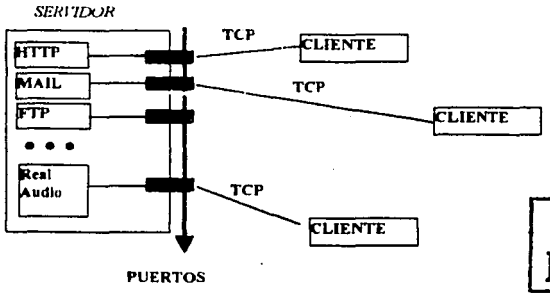
3.2 Conceptos Básicos De La Comunicación De Datos

APLICACION (HTTP,FTP,TELNET,...)
TRANSPORTE (TCP,UDP,...)
REDES (IP,IPX,...)
CONEXIÓN (Device Drivers, Winsock,...)

TCP: (*Transport Control Protocol*) Es un protocolo basado en la conexión punto a punto que provee un flujo confiable de datos entre dos computadoras.

UDP: (*User Datagram Protocol*) Es un protocolo que envía paquetes de datos de manera independiente, llamados datagramas, de una computadora a otra sin garantizar su llegada

Puertos:



Cada computadora conectada a Internet es identificada con un número llamado IP, que está formado 4 bytes. A cada aplicación que se encuentra registrada en el Servidor se le asigna además un número de 16 bits llamado Puerto.

De tal manera que para establecer una conexión con el servidor es necesario especificar su dirección que está formada por su IP y por el número de puerto, por ejemplo...

254.90.128.56:8889

El rango de número de puerto es del 0 al 65535.

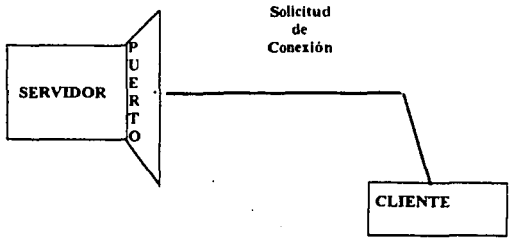
Los primeros números 0 al 1023 se encuentran reservados para servicios del sistema como HTTP, FTP, TELNET, MAIL, etc...

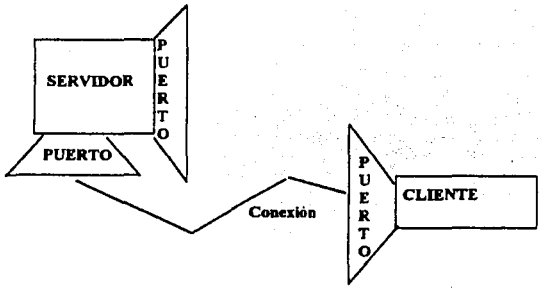
TCP utiliza el concepto de socket para comunicar el cliente con el puerto del servidor

SOCKET

TCP establece una conexión punto a punto, ¿qué pasa cuando hay más de un cliente que requiere el mismo servicio (la misma aplicación)?

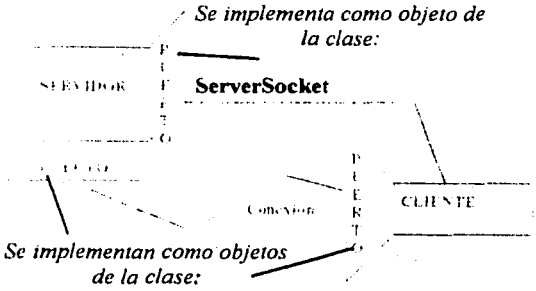
TCP asigna un puerto a cada una de las aplicaciones que está a la espera de que una maquina cliente solicite su conexión. Una vez aceptada la conexión, TCP crea un nuevo puerto de comunicación en el servidor, exclusivo para comunicarse con el cliente,





Un Socket es la representación en programación de un puerto de conexión punto a punto de TCP.

Clases en Java para el manejo de Sockets

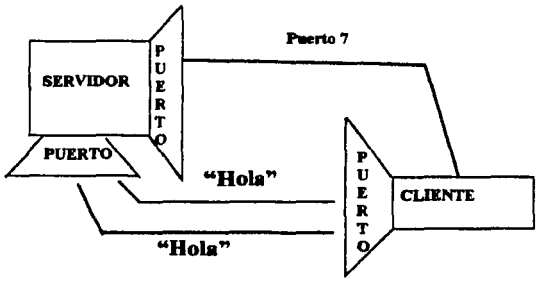


Las clases ServerSocket y Socket se encuentran definidas en la libreria:

*java.net.**

Ejemplo De Un Socket En El Lado Cliente

Cliente Echo



Paso 1: Definir y abrir el Socket

```
Socket  echoSocket = null;
...
try {
   echoSocket = new Socket("academ01.qro.itesm.mx",7);
      ...
} catch (Exception e) {System.out.println("error:"+e)};
```

Paso 2: Crear los streams de entrada y de salida

```
Socket  echoSocket = null;
DataInputStream in = null;
DataOutputStream out = null;
...
try {
   echoSocket = new Socket("academ01.qro.itesm.mx",7);
   in = new DataInputStream (echoSocket.getInputStream( ));
   out = new DataOutputStream (echoSocket.getoutputStream( ));
} catch (Exception e) {System.out.println("error"+e));
```

Paso 3: Leer y escribir los datos

```
String texto;
texto = System.in.readLine();
while (texto != null) {
   out.writeUTF(texto);
   System.out.println("echo: "+in.readUTF());
   texto = System.in.readLine();
}
```

Paso 4 : Cerrar Streams y Socket

```
out.close();
in.close();
echoSocket.close();
```

Ejemplo De Un Socket En El Lado Servidor

Un Echo Server sencillo...

Paso 1: Definir y abrir ServerSocket

```
ServerSocket  serverSocket = null;
...
try {
   serverSocket = new ServerSocket(1234);
        ...
} catch (Exception e) {System.out.println("error:"+e));
```

Paso 2: Conectar con el cliente y crear el socket de comunicación

```
ServerSocket  serverSocket = null;
Socket  clienteSocket = null;
...
try {
   serverSocket = new ServerSocket(1234);
   clienteSocket = serverSocket.accept();
        ...
} catch (Exception e) {System.out.println("error:"+e));
```

Paso 3: Crear los streams de entrada y de salida

```
ServerSocket  serverSocket = null;
Socket  clienteSocket = null;
DataInputStream in = null;
DataOutputStream out = null;
...
try {
   serverSocket = new ServerSocket(1234);
   clienteSocket = serverSocket.accept();
   in = new DataInputStream (clienteSocket.getInputStream( ));
   out = new DataOutputStream (clienteSocket.getoutputStream( ));
} catch (Exception e) {System.out.println("error:"+e));
```

Paso 4: Leer y escribir los datos

```
String texto;
texto = in.readLine();
while (texto != null) {
   out.writeUTF(texto);
   texto=in.readUTF();
}
```

Paso 5: Cerrar Streams y Socket

```
out.close();
in.close();
clienteSocket.close();
serverSocket.close();
```

3.3 Ventajas E Inconvenientes De La Arquitectura Cliente Servidor

Se mencionan a continuación algunas de las ventajas de la utilización de la arquitectura Cliente/Servidor.

Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor es la existencia de plataformas de hardware cada vez más baratas. Esta constituye a su vez una de las más palpables ventajas de este esquema, la posibilidad de utilizar máquinas considerablemente más baratas que las requeridas por una solución centralizada, basada en sistemas grandes. Además de lo anterior, se pueden utilizar componentes, tanto de hardware como de software, de varios fabricantes, lo cual contribuye considerablemente a la reducción de costos y favorece la flexibilidad en la implantación y actualización de soluciones.

Además de lo anterior, la arquitectura Cliente/Servidor permite compartir información y facilita la integración entre sistemas diferentes, permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas pero utilizando interfaces más amigables al usuario. De esta manera podemos, por ejemplo, integrar PCs con sistemas medianos y grandes, sin que todas las máquinas tengan que utilizar el mismo sistema operacional.

Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos con este esquema tienen una interacción más intuitiva con el usuario. Si se utilizan interfaces gráficas para interactuar con el usuario, la arquitectura Cliente/Servidor presenta la ventaja, con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues esta puede residir en el cliente, lo cual permite aprovechar mejor el ancho de banda de la red.

Una ventaja adicional del uso dla arquitectura Cliente / Servidor es que es más rápido el mantenimiento y el desarrollo de aplicaciones pues se pueden emplear las herramientas existentes (por ejemplo los servidores de SQL o las herramientas de más bajo nivel como los sockets o el RPC).

La estructura inherentemente modular facilita además la integración de nuevas tecnologías y el crecimiento de la infraestructura computacional, favoreciendo así la escalabilidad de las soluciones.

La arquitectura Cliente/Servidor contribuye además a proporcionar a los diferentes departamentos de una empresa soluciones locales, pero permitiendo además la integración de la información relevante a nivel global.

La arquitectura cliente/servidor tiene algunos inconvenientes que se mencionan a continuación.

Por una parte, el mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y de software, distribuidas por distintos proveedores, lo cual dificulta el diagnóstico de fallas.

Además de lo anterior, se cuenta con muy escasas herramientas para la administración y ajuste del desempeño de los sistemas.

En el desarrollo de aplicaciones Cliente/Servidor se deben tener en cuenta diferentes aspectos, que se mencionan a continuación.

Por un lado, es importante que los clientes y los servidores utilicen el mismo mecanismo (por ejemplo sockets o RPC), lo cual implica que se deben tener mecanismos generales que existan en diferentes plataformas.

Además hay que tener estrategias para el manejo de errores y para mantener la consistencia de los datos.

La seguridad de un esquema Cliente/Servidor es otra preocupación importante. En este caso los mecanismos son distintos que en el caso de los sistemas centralizados. Por ejemplo, se deben hacer verificaciones en el cliente y en el servidor. También se puede recurrir a otras técnicas como el encriptamiento.

El desempeño es otro de los aspectos que se deben tener en cuenta en la arquitectura Cliente/Servidor. Problemas de este estilo pueden presentarse por congestión en la red, dificultad de predecir el tráfico, etc.

Un aspecto directamente relacionado con el anterior es el de cómo distribuir los datos en la red. En el caso de una empresa, por ejemplo, este puede ser hecho por departamentos, geográficamente, o de otras maneras. Además hay que tener en cuenta que en algunos casos, por razones de confiabilidad o eficiencia se pueden tener datos replicados, y que puede haber actualizaciones simultáneas.

A otro nivel, una de las decisiones que deben tomar las organizaciones es la de si compran o desarrollar los diferentes componentes.

3.4 Ventajas Que Puede Aportar La Arquitectura Cliente Servidor A Las Empresas.

Anteriormente se trato la arquitectura Cliente/Servidor haciendo énfasis en los aspectos técnicos. Ahora se trata cómo puede beneficiar a la empresa.

Como una primera ventaja podemos mencionar que con el uso de este esquema se reducen los costos de producción de software y se disminuyen los tiempos requeridos. Esto es así pues, para la construcción de una nueva aplicación pueden usarse los servidores que haya disponibles reduciéndose el desarrollo a la elaboración de los procesos del cliente, según los requerimientos deseados. Lo anterior disminuye los costos internos del área de sistemas. Además, se pueden obtener ventajas importantes al reducir el costo del hardware requerido, llevando las aplicaciones a plataformas más baratas, aprovechando el poder de cómputo de los diferentes elementos de la red, y facilitando la interacción entre las distintas aplicaciones de la empresa. La arquitectura Cliente/Servidor también contribuye a una disminución de los costos de entrenamiento de personal pues favorecen la construcción de interfaces gráficas interactivas, las cuales son más intuitivas y fáciles de usar por el usuario final.

Otra de las ventajas de la arquitectura Cliente/Servidor es que facilita el suministro de información a los usuarios. Esto es así porque, por un lado proporciona una mayor consistencia a la información de la empresa al contar con un control centralizado de los elementos compartidos, y por otro, porque facilita la construcción de interfaces gráficas interactivas, las cuales pueden hacer que los "datos" se conviertan en "información".

Además de lo anterior, la arquitectura Cliente/Servidor permite llevar más fácilmente la información a donde se necesita, además de que contribuye a aumentar su precisión pues se puede obtener de su fuente (el servidor) y no de una copia en papel o en medio magnético.

La habilidad de integrar sistemas heterogéneos es inherente al modelo Cliente/Servidor, pues los clientes y los servidores pueden existir en múltiples plataformas y hacer acceso a datos de cualquier sitio de la red. Además un cliente puede integrar datos de diferentes sitios para presentarlos, a su manera, al usuario final.

Al favorecer la construcción de interfaces gráficas interactivas y el acceso transparente a diferentes nodos de la red se facilita el uso de las aplicaciones por parte de los usuarios, lo cual aumenta su productividad.

La arquitectura Cliente/Servidor también favorece la adaptación a cambios en la tecnología pues facilita la migración de las aplicaciones a otras plataformas y, al aislar claramente las diferentes funciones de una aplicación hace más fácil incorporar nuevas tecnologías en esta.

Resumen

Los principales componentes de la arquitectura cliente/servidor son: los Clientes que interactúan con el usuario, usualmente en forma gráfica y los Servidores que proporcionan un servicio al cliente y devuelven los resultados, para que los clientes y los servidores puedan comunicarse se requiere una infraestructura de comunicaciones, la cual proporciona los mecanismos básicos de direccionamiento y transporte.

Pasos Para Crear Un Socket Lado Cliente En Java:

Paso 1: Definir y abrir el Socket
Paso 2: Crear los streams de entrada y de salida
Paso 3: Leer y escribir los datos
Paso 4 : Cerrar Streams y Socket

Pasos Para Crear Un Socket Lado Servidor En Java:

Paso 1: Definir y abrir ServerSocket
Paso 2: Conectar con el cliente y crear el socket de comunicación
Paso 3: Crear los streams de entrada y de salida
Paso 4: Leer y escribir los datos
Paso 5: Cerrar Streams y Socket

Uno de los aspectos que más ha promovido el uso de sistemas Cliente/Servidor es la existencia de plataformas de hardware cada vez más baratas, además permite compartir información y facilita la integración entre sistemas diferentes, aunque los sistemas Cliente/Servidor poseen muchas ventajas, el mantenimiento de los sistemas es más difícil pues implica la interacción de diferentes partes de hardware y de software, distribuidas por distintos proveedores, lo cual dificulta el diagnóstico de fallas.

CAPÍTULO IV

JDBC

Introducción

Un buen sistema de información se ha convertido en una ficha clave para el buen funcionamiento organizacional y para crear ventajas competitivas reales en los tiempos actuales. Obtener información fiable, actualizada y completa en el momento en que se necesita y, en algunos casos, en el lugar preciso, marca diferencias reales ante los clientes, los competidores e incluso dentro de departamentos de una misma empresa.

Con el progreso de la computacion, la necesidad de manejar mas informacion incrementa, este crecimiento necesita una interface para el acceso de bases de datos desde Java, es por eso que nace JDBC (Java Database Connectivity). Una de las ventajas de usar Java sobre otros lenguajes es la independencia de la plataforma. JDBC ha logrado la misma flexibilidad en el mundo de las bases de datos , este capitulo trata del origen de JDBC, que es JDBC para que sirve y como funciona. En el mundo de la computacion existen muchos estandares para el acceso a bases de datos, el mas popular es el SQL (Structured Query Language) el cual define un conjunto de comandos para crear, almacenar, modificar, recuperar y manejar informacion de una base de datos, en este capitulo se indica como desarrollar aplicaciones que se conecten a bases de datos SQL .En este capítulo también se trata de las diferencias y similitudes de JDBC y ODBC y de los modelos para el acceso de bases de datos (modelo de dos y tres capas). Hasta antes de JDBC, el único camino para comunicarse con bases de datos desde Java era por medio de llamadas a programas externos llamados CGIs (Commond Gateway Interface), por lo que para finalizar este capítulo se menciona a JDBC como alternativa a usar CGI's. La Seguridad JDBC es un tema que también trata este capítulo, ya que la seguridad es siempre un problema importante, sobre todo cuando se trata de bases de datos.

Para concluir este capítulo, se mencionan los tipos de manejadores o drivers con que cuenta JDBC

4.1 Origen JDBC

El surgimiento de JDBC, se debe a la necesidad de desarrollar aplicaciones que se conecten a bases de datos SQL, para este fin existía ODBC que era usado como estándar para el acceso a bases de datos relacionales, pero tenía grandes inconvenientes al quererlo usar en aplicaciones Java y lo que sucedía era que es una interfaz desarrollada en lenguaje C, por lo que no era convertible a Java.

El factor más importante es que al utilizar ODBC, se perdían independencia de la plataforma. Lo que se necesitaba era una interface Java que fuera consistente con el resto del sistema Java, es decir que cumpliera con sus principales características, que fuera orientado a objetos e independiente de la plataforma. Por lo tanto la meta de los desarrolladores de JDBC era crear una API de bajo nivel que soportar el acceso SQL, para esto basaron su trabajo en interface X/Open CLI (Call Level Interface). Cabe señalar que ODBC también está pasando en dicha interfase.

JavaSoft introdujo formalmente JDBC desde junio de 1996, esta interface permitía realizar aplicaciones independientes de la base de datos y de la plataforma, con lo que se cumplió con la necesidad de tener un interface natural Java.

4. 2 ¿Qué Es JDBC? Y ¿Para Qué Sirve?

Es una API de Java para ejecutar declaraciones o sentencias (*querys*) SQL. JDBC es una marca registrada, y no son siglas; no obstante, JDBC es frecuentemente nombrado como "Java Data Base Connectivity" consiste en un conjunto de clases e interfaces escritos en el lenguaje de programación Java. JDBC provee una API estándar para los desarrollos Bruce de herramientas para base de datos, permitiendo desarrollar dichas aplicaciones en Java 100% puro.

JDBC es un conjunto de clases o interfaces escritas en lenguaje Java que permiten realizar aplicaciones con conexión a base de datos en Java 100% puro, JDBC usa el estándar SQL y facilita el acceso uniforme a una amplia gama de base de datos relacionadas, por lo que, con la API JDBC se pueden enviar fácilmente declaraciones SQL a casi cualquier base de datos relacional, es decir, se puede escribir un programa único usando JDBC y el programa es capaz de enviar declaraciones SQL a la base de datos correspondiente.

JDBC es una interfase de bajo nivel, lo que permite invocar por llamar comandos SQL directamente y se diseñó para ser una base sobre la que se pueden conseguir interfaces de alto nivel, es decir, amigables con los usuarios.

Usando JDBC es fácil enviar declaraciones SQL a casi cualquier base de datos relacional. En otras palabras, con la API JDBC, no es necesario escribir un programa para accesar una base del datos de Sybase, otro para accesar Oracle, otro para Informix y uno para cualquier otra base datos. Uno puede escribir un programa único que use la API JDBC y el programa será capaz de enviar declaraciones SQL a la base de datos apropiada. Y si la aplicación esta escrita con el lenguaje de programación Java, no se deberá preocupar sobre las diferentes plataformas en donde será ejecutada. La combinación de Java y JDBC permite a programadores escribir una vez y ejecutarlo en donde sea.

Java, siendo robusto, seguro, fácil de usar y de comprender, teniendo la posibilidad de bajarlo u obtenerlo de Internet, es un buen punto de partida para creer aplicaciones de base de datos, sin preocuparse de la base de datos a la que se accesar y qué plataforma se desarrollará y ejecutará.

JDBC extiende lo que se puede hacer en Java. Por ejemplo, con Java y la API JDBC, es posible publicar una página en el Web que contenga un Applet que usa información obtenida desde una base de datos remota. Una empresa puede usar JDBC para conectar a todos sus empleados (aún cuando ellos usan una computadora Windows, Macintosh o UNIX) a una o más bases de datos por medio de una Intranet. Con más y más desarrolladores que usan el lenguaje de programación Java, la necesidad de un acceso fácil a una base de datos desde Java continuará en crecimiento.

Los negocios pueden continuar usando sus bases de datos instaladas, actualmente, accesando a la información fácilmente aún cuando se almacena sobre sistemas diferentes de gestión de base que datos. El tiempo del desarrollo para nuevas aplicaciones es corto. El control de versión e instalación se simplifican mucho. Por ejemplo, de las ventajas, un programador puede escribir una aplicación o una actualización, poniéndolo en un servidor y todos los usuarios tendrán acceso a la última versión. Para empresas con servicio de venta de información, Java y JDBC ofrecen una mejor manera de conseguir información actualizada por clientes externos.

4. 3 ¿Cómo Trabaja JDBC?

Como sabemos SQL es un buen lenguaje para manipular o tratar datos almacenados de alguna forma organizada (en bases de datos, por lo tanto JDBC provee cierta funcionalidad para favorecer dicha manipulación de datos. La funcionalidad la permite gracias a que incluye clases Java, para representar a los elementos necesarios para ello, como son: la conexión a la base de datos, tratamiento y ejecución de sentencias SQL, conjuntos de resultados e información detallada de la base o del conjunto de resultados (metadata). Las clases e interfaces están escritas en Java para permitir una integración natural de la API JDBC y la aplicación que se desarrolle en Java.

JDBC hace posible tres cosas

- Establece una conexion con una base datos
- Envia declaraciones SQL
- Procesa resultados

El fragmento siguiente de código da un ejemplo básico de estos tres pasos

```
Connection con = DriverManager.getConnection (
"jdbc:odbc:wombat", "login", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next()) {
int x = rs.getInt("a");
String s = rs.getString("b");
float f = rs.getFloat("c"); }
```

JDBC Es Una API De Bajo Nivel Y Una Base Para APIs De Alto Nivel

JDBC es una interface de "bajo nivel", significa que se usa para invocar (o "llamar"), comandos SQL directamente. Es más fácil su uso para otras APIs de conectividad a bases de datos, pero se diseño también para ser una base sobre la que se puede construir interfaces de alto nivel y herramientas. Una interfaz de alto nivel es amigable al usuario, usando una más comprensible o más conveniente API que se comunica con interfaces de bajo nivel, tal como JDBC.

Actualmente se construyen dos tipos de que APIs de alto nivel sobre JDBC:

1.- Una incrustación de SQL para Java. Por lo menos un vendedor planea construir ésto. Un DBMS implementará SQL, un lenguaje diseñado específicamente para el uso con bases de datos. JDBC requiere que los declaraciones SQL sean pasadas como String a los métodos de Java. Un procesador SQL incluido permitido a un programador mezclar declaraciones SQL directamente con Java: por ejemplo, una variable de Java puede usarse en una declaración SQL para recibir o proveer valores SQL. El procesador SQL incluido producirá llamados Java/SQL mezclados.

2.- Mapeado directo entre tablas de base se datos relacionales con clases de Java. JavaSoft y otras empresas han anunciado planes para incrementar ésto. En este "objeto relacional" mapeado, cada fila de la tabla llega a ser una instancia de esa clase y cada valor de esa columna corresponde al atributo de esa instancia. Los programadores pueden operar entonces directamente sobre objetos de Java; los llamados SQL traen y almacenan datos que se generan automáticamente "bajo esta capa". Mapeados más sofisticados también son permitidos, por ejemplo, filas de tablas múltiples son combinadas en una clase Java.

Como el interés de JDBC ha crecido, más desarrolladores han trabajado con herramientas basadas en JDBC para construir programas más fáciles. Los programadores han escrito también aplicaciones que han accesado una base que datos de forma más fácil para el usuario final. Por ejemplo, 1 aplicación podría concentrar un menú de protesta clases de datos con opciones para escoger un estudio seleccionar una tarea se pueden presentar un *prompt* y/o espacios en blanco (casillas) para ser llenadas con información necesaria para la tarea seleccionada. Con ayuda de estas aplicaciones, los usuarios pueden desempeñar las tareas de bases de datos aún con poco o ningún conocimiento de la sintaxis SQL.

JDBC Contra ODBC Y Otras APIs

ODBC de Microsoft es una de las interfase más usada para accesar a bases de datos relacionadas brindando la capacidad de conectarse a casi todas las bases de datos sobre casi todas las plataformas, pero no es apropiado su uso directo desde Java debido a cuatro aspectos fundamentales:

1.- Debido a que ODBC está realizada en C; una aplicación realizada en Java presenta desventajas en seguridad, implementación y robustez.

2.- Java no tiene apuntadores y ODBC utiliza demasiados.

3.- ODBC mezcla aspectos simples y complejos, JDBC se diseñó para mantener las cosas tan simples como lo permitieran sus capacidades más avanzadas.

4.- Cuando se usa ODBC, sus controladores deben instalarse en cada máquina cliente. el controlador JDBC está escrito completamente en Java y es automáticamente instalable.

Existen otras APIs diferentes a JDBC y ODBC como RDO, ADO y OLE DB, las cuales siguen la misma dirección de JDBC, es decir una interfase de base de datos orientado a objetos. Con clases que se pueden implementar sobre ODBC.

La API ODBC de Microsoft es probablemente la más usada por programadores como interface para accesar base se datos relacionales. Ofrece la capacidad para conectar a casi todos los bases del datos sobre casi todas las plataformas. Pero, ¿por que no simplemente usar ODBC desde Java?

La respuesta es que usted puede usar ODBC desde Java, pero este es mejor si usa JDBC en forma del JDBC-ODBC como puente, como se verá más adelante.

Una buena pregunta es ¿Porqué necesita usted JDBC?. Hay varias respuestas para esta pregunta:

1.- ODBC no es apropiado es su uso directo desde Java porque usa a una interface de C. Los llamados desde Java al código nativo C, tiene número de desventajas en la seguridad, implementación, robustez portabilidad automática de aplicaciones.

2.- una traducción literaria de la API C de ODBC a una API Java no sería deseable. Por ejemplo, Java no tiene apuntadores y ODBC utiliza demasiados.

3.- ODBC esa fácil de aprender. Mezclan aspectos simples con avanzados y da opciones complejas junto con simples. JDBC, por otra parte, se diseñó para mantener las cosas tan simples como lo permitieran sus capacidades más avanzadas.

4.- una API Java como JDBC es necesaria a fin de permitir una solución "Java 100% puro". Cuando ODBC se usado, los controladores ODBC deben ser instalados manualmente en cada máquina cliente. El manejador JDBC esta escrito completamente en Java, el código JDBC es automáticamente instalado, portátil y seguro sobre todas las plataformas de Java desde los computadores de red hasta Mainframes.

En resumen, la API JDBC es una interface natural Java para las abstracciones y conceptos SQL básicos. Programadores familiarizados con ODBC encontrarán fáciles de aprender a JDBC. JDBC mantiene los aspectos básicos del diseño de ODBC; de hecho, ambas interfaces están basadas en X/Open SQL CLI (Call Lavel Interface). La principal diferencia es que JDBC construye y refuerza el estilo y virtudes de Java y por supuesto, es fácil de usar.

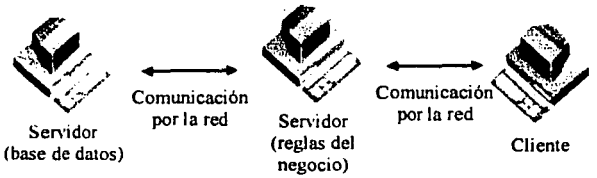
Más recientemente, Microsoft ha introducido nuevas APIs más allá de ODBC, RDO, ADO y OLE DB. Estos diseños se mueven a la misma dirección de JDBC, por ejemplo, en ser una interface de base de datos orientado objetos con una base de clases que pueden implementarse sobre ODBC. Sin embargo, no se ven con la funcionalidad obligatoria en ninguna interface para ser alternativa de ODBC. Con estas interfaces sólo se tiene un revestimiento sobre ODBC. JDBC necesita evolucionar desde su liberación; sin embargo, la mayoría de la nueva funcionalidad pertenece en APIs de alto nivel, así como la relación de objetos mapeados e incluir un intérprete SQL mencionado anteriormente.

Modelos De Dos Y Tres Capas

La API JDBC apoya ambos modelos para el acceso de bases de datos.

El modelos de dos capas, un applet o una aplicación Java se comunica directamente a la base de datos. Esto requiere algún manejador JDBC que puede comunicarse con el sistema particular de gestión de base datos a ser accesar. Un usuario envía declaraciones SQL a la base de datos y los resultados de estas declaraciones se envían de regreso al usuario. La base de datos puede ubicarse en otra máquina a la que el usuario se conecta por medio de la red. A esto se le conoce como una configuración cliente/servidor, con la máquina del usuario, el cliente y la máquina que aloja a la base de datos como servidor.

En el modelo de tres capaz, los comandos se envían a un "nivel medio" de servicios, que entonces envía declaraciones SQL a la base de datos. La base de datos procesan las declaraciones SQL y envía los resultados de regreso a nivel medio, que los envía al usuario. Algunas personas encuentran el modelo de tres capaz muy atractivo, porque el nivel medio hace lo posible para mantener el control sobre el acceso y los tipos de actualizaciones que pueden hacerse a los datos. Otra ventaja es que cuando hay un nivel medio, el usuario puede emplear de forma fácil una API de bajo nivel el cual traduce llamados de bajo nivel apropiados. Finalmente, en muchos casos la arquitectura de tres niveles puede proveer ventajas del desempeño.



Hasta ahora el nivel medio típicamente se escribe en lenguajes como C o C++, que ofrece ayuda en el desempeño. Sin embargo, con la introducción de compiladores optimizados que producen *bytecode* Java en código máquina eficientemente, llega a ser práctico para implementar el nivel medio en Java. Esto es bueno, haciendo lo posible por aprovechar de robustez de Java, multihilado y aspectos de seguridad. JDBC es importante para permitir el acceso de base de datos desde un nivel medio de Java.

Conformidad Con SQL

El Lenguaje Estructurado de Consultas (SQL) es el lenguajes estándar para accesar o consultar bases de datos relacionales. Un problema es que, aunque la mayoría de los sistemas de gestión de bases de datos (DBMSs) usa una forma estándar de SQL para la funcionalidad básica, ellos no adoptan

la más reciente definición estándar de SQL en cuanto a sintaxis o semántica para una funcionalidad que más avanzada. Por ejemplo, no todas las bases de datos soportan *stored procedures* u *outer joins*.

Es importante mencionar que independientemente JDBC estandariza el mecanismo para conectarse a bases de datos de ninguna manera intenta estandarizar la sintaxis SQL. Se espera que los manejadores SQL, que en verdad son estándares, se expanden para incluir más y más funcionalidad. Entretanto, la API JDBC debe soportar a SQL como es.

Una forma de mantener la idea estándar de SQL es que toda sentencias se envíe al manejador de bases de datos, pero se correrá el riesgo de recibir mensajes del error en el momento de ejecutar ésta sentencia, sin embargo, la mayoría de las consultas siguen una sintaxis de SQL estándar por lo que JDBC permite que se cambie el host de la base de datos, los puertos y drivers con solo pequeños cambios en el código.

Otro camino para mantener la conformidad con SQL es que se provea cláusulas de escape ODBC.

La sintaxis de escape provee una sintaxis estándar JDBC para los seres más comunes SQL huele de divergencia. Por ejemplo, hay cláusulas escape para literales derecha que para llamados a stored procedures.

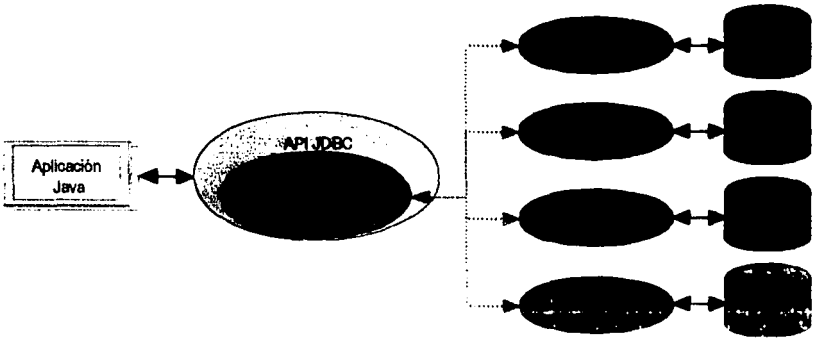
Para aplicaciones complejas, JDBC provee conformidad con SQL en una tercera manera. Provee información descriptiva sobre el DBMS por medio de la interface DatabaseMetaData para que las aplicaciones pueden adaptarse a los requerimientos y capacidades se cada DBMS.

La API JDBC se usará como una base API para desarrollar herramientas de acceso a bases de datos y APIs de alto nivel, también tiene que dirigirse al problema de conformidad. La designación "JDBC COMPLIANT TM" se creó para colocar un nivel estándar JDBC de funcionalidad en el que el usuario puede confiar. Para poder usar ésta designación, un manejador desde soportar ANSI SQL-2 Entry Lavel (ANSI SQL-2 se refiere a las normas adoptadas por el Instituto de Normas Nacionales Estadounidenses de 1992. *Entry Lavel* se refiere a una lista especifica de capacidades SQL). Los desarrolladores de manejadores pueden comprobar que sus librerías se encuentran con la designación, probándolas con las herramientas que vienen con el JDBC.

La designación "JDBC COMPLIANT TM" indica que un vendedor de aplicaciones JDBC ha pasado las pruebas de conformidad provistas por JavaSoft. Estas pruebas de conformidad verifican la existencia de todas las clases y métodos definidos en la API JDBC y verifica como es posible que el SQL *Entry Lavel* esta disponible. Las pruebas no son exhaustivas, por supuesto, ésta designación de cumplimiento proveer algún grado de confianza en implementación JDBC. Con la aceptación más y más amplia de la API JDBC por vendedores de bases de datos, vendedores se conexiones, vendedores de servicios en Internet y programadores, JDBC rápidamente llegará a ser la norma para Java en el exceso de bases de datos.

4. 4 Estructura JDBC

En lugar de utilizar métodos con una gran cantidad de parámetros para lograr el manejo de una posible y amplio comportamiento, las clases base de Java fueron diseñadas para utilizar diferentes métodos para mejorar diferentes tareas; por lo tanto JDBC permite la misma funcionalidad.



La API JDBC

Como ya se ha mencionado JDBC se creo como un interface SQL de Java, la cual ejecuta las sentencias y obtiene los resultados. Para el ejemplo anterior se posiciona en la parte de la comunicación de los *Drivers* y la aplicación Java, esto lo permite por medio del Administrador de Manejadores.

Manejadores

Es el elemento que permite realmente la comunicación entre la aplicación Java y la base de datos, enviando y recibiendo datos, de manera "transparente" para el programador. Obviamente se neceita un *Driver* adecuado a la base de datos con la que se quiera trabajar.

Base De Datos

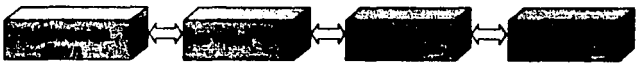
Elemento que sirve como repositorio de los datos, son los que la aplicación Java manipulara.

4.5 JDBC En Lugar De CGI'S

Hasta antes de JDBC, el único camino para comunicarse con bases de datos desde Java era por medio de llamadas a programas externos llamados CGIs (Commond Gateway Interface), lo que permitía de forma separada realizar consultas y la devolución o respuesta de datos.

Esta es una manera muy lenta y produce o sería muy suceptible a tener errores muy severos dentro de la aplicación. También ocasiona a tener que desarrollar los dos módulos en dos lenguajes diferentes la aplicación en Java y el CGI en C, perl, shell o culaquier otro, con lo cual se propicia más complejidad en el momento de desarrollo.

Otra razón para usar JDBC es la rapidez de obtención de los datos, ya que con CGIs, él es el que procesa los datos y los devuelve a Java por medio de *Streams*. Además de tener que esperar los resultados más tiempo de lo normal, debido a que también interviene el servidor de Web. Como se representa acontinuación:



Por otro lado si se utiliza JDBC en lugar de CGIs se tienen consultas más rápidas hacia la base de datos y lo más importante es que es en menos pasos. Como se muestra acontinuación:

4.6 Seguridad JDBC

La seguridad es siempre un problema importante, sobre todo cuando se trata de bases de datos. JDBC sigue el modelo de seguiridad estándar, en el cual los applets pueden conectarse solamente al servidor donde ellos están alojados; los applets remotos no pueden conectarse a bases de datos locales. Las aplicaciones no tienen ninguna restricción de conexión. Los applets y las aplicaciones necesitan una política de seguridad y código que den fuerza a esa política, aveces es difícil saber por adelantado que nivel de seguiridad se requerirá en una situación dada y a menudo existe un intercambio entre conveniencia y seguridad aunque cualquier aplicación interesante debe permitir a usuarios finales configurar algunos de los parámetros de seguridad para satisfacer sus necesidades

Se tienen algunos aspectos muy importantes relacionados a la seguridad y JDBC, que a continuación se explican.

Restricciones De Seguridad En Applets Con Conexión A La Base De Datos²

Antes de la liberación del JDK 1.1, los applets son obtenidos de un servidor http, el cual se restringe en dos maneras: El applet no puede utilizar código no nativo de Java, que incluye no poder utilizar controladores escritos en C o C++, tampoco el puente JDBC-ODBC. La otra restricción de los applets, es que no pueden utilizar bases de datos que no están localizadas en la misma computadora del servidor Web, lo que ocasiona algunos problemas con la mayoría de arquitecturas tecnológicas recientes, ya que dicha localización generalmente no se cumple.

Otra limitación de los applets, es debido a que en las librerías o clases principales que utilizan los navegadores no se habían incluido las clases necesarias de JDBC; a demás de que dichos navegadores generalmente no copian al cliente clases que empiecen con la palabra reservada "java", debido a que se quiere evitar que alguien dañino pueda crear una clase similar o sobrescribirlas, para acciones no deseadas por el navegador, con lo que afecta poder copiar las clases JDBC.

Con la liberación del JDK 1.1, se solucionaron algunos de estos problemas: ya se incluye JDBC, en las clases principales del JDK; adicionalmente un applet puede ser firmado digitalmente (dichas firmas salen del propósito de este documento), con lo que al firmarse, se dice que es un applet confiable, además, al ser confiable, puede utilizar bases de datos localizadas en diferente ubicación que el servidor web.

Tipos De Codigo[3]

Código Confiable:

En todo tipo de desarrollo de Java incluyendo aplicaciones y applets de fuentes amigables (por "amigables" se entiende como servidores de alguna empresa, servidores personales o de desarrollo), se pueden firmar con una llave encriptada para asegurar su integridad durante su descarga al cliente.

Código No Confiable:

A applets no confiables encontrados en Internet, no se les permite utilizar dispositivos locales del cliente. ni el sistema de archivos, además de sólo permitirle conectarse al servidor del que se descargo, incluyendo conexiones a base de datos.

Cambios[4]

Con la versión 3 de JDBC, próxima a su liberación. Se ha implementado aún más detalle en el manejo de la seguridad como se puede ver a continuación.

La Clase SQLPermission

La clase representa un conjunto de permisos que el código base (codebase) puede tener. Actualmente sólo un permiso se ha definido setLog. El objeto SecurityManager verifica el permiso setLog cuando un applet uno de los métodos setLogWriter y setLogStream del objeto DriverManager, si éste permiso no se ha establecido se dispara la excepción java.lang.SecurityEsception .

---

[2] Java 1.1 Unleashed, Professional Reference Edition ***by Shelley Powers***
[3] Sun Sep 8 16:22:50 CDT 1996 ***by Jerrid Hamann***
[4] JDBC ™ 3.0 Specification Proposed Final October 2000, Sun Microsystems, Inc. ***by Jon Ellis & Linda Ho with Maydene Fisher***

Convenios del Sistema

Los convenios del sistema definidos en las definiciones de la arquitectura Connector, describe la interfase entre el servidor de la aplicación y uno o más adaptadores de recursos. Esta interface le permite a los adaptadores de los recursos para ser relacionados de una manera que pueda ser usado por cualquier servidor de la aplicación que soporte este tipo de convenios de sistema. Los tipos de contratos son los siguientes:

- Un convenio del manejo de una conexión que habilita los componentes de una aplicación para conectarse al sistema de fuente de datos. Esto es equivalente a la interfase JDBC *DataSource* y *ConnectionPoolDataSource*.
- Un convenio del manejo de una transacción entre el manejador de transacciones y el sistema que es la fuente de datos, permitiendo el manejo de transacciones en sus recursos. Esto es equivalente a la interfase *DataSource*.
- Un convenio de seguridad que permite el seguro acceso a sistemas fuentes de datos; esta autentificación siempre se maneja como la proporción de un usuario y su contraseña.

4.7 Drivers JDBC

Los *Drivers* o manejadores JDBC actuales caen en una de estas cuatro categorías:

1.- El puente JDBC-ODBC más el manejador ODBC: JavaSoft provee productos JDBC para accesar bases de datos por medio de manejadores ODBC. Una advertencia importante es que el cliente ODBC, y en muchos casos el cliente de la base que datos, deben ser instalados en cada máquina cliente que usen este manejador. Como resultado, que ese tipo de manejadores son apropiados sobre redes corporativas donde la instalación no es un problema importante o para usar aplicaciones escritos en Java deben usar la arquitectura que tres niveles.

2.- API metido parcialmente manejador Java: que este tipo de manejador convierte llamados JDBC en llamados a la API cliente para Oracle, Sybase, Informix, DB2 u otro DBMS. También es necesario que este tipo de manejadores se sale en cada máquina cliente.

3.- JDBC-Red manejador Java puro: es que se manejador traduce llamados JDBC dentro de un DBMS independiente al protocolo de la red, que se traduce entonces el protocolo DBMS. Ese servidor en la red es capaz de conectarse con clientes Java puros de muchos bases de datos diferentes. En general, ésta es la más flexible alternativa JDBC. Es probable que todos los vendedores de estas soluciones provean productos apropiados para uso en Internet, debiendo manejar requerimientos adicionales para la seguridad seguridad, acceso mediante *firewall*, etc.

4.- Protocolo nativo - manejador Java puro: este tipos de conductor convierte llamados JDBC dentro del protocolo de la red usados por los DBMSs directamente. Esto permite un llamado directo desde el cliente al servidor DBMS y es una solución práctica para acceso a Internet. Dichos pro tocolo son patentados por que los vendedores de bases de datos, proclives tendrán la última palabra.

Eventualmente, se espera que los manejadores de categorías 3 y 4 serán la manera preferida de acceso bases de datos desde JDBC. Los manejadores se categorías 1 y 2 son las soluciones iniciales, donde manejadores Java no son aún disponibles. Las categorías 3 y 4 ofrecen todas las ventajas de Java, incluyendo la instalación automática. La siguiente tabla muestra las cuatro categorías y sus propiedades:

Categoría del manejador	¿Todo Java?	Protocolo de Red
puente JDBC-ODBC	No	Directo
API nativa como base	No	Directo
JDBC-red	Si	requiere colector
Protocolo nativo como base	Si	Directo

Manejadores JDBC

En este momento existen docenas de manejadores de la categoría 1: manejadores ODBC que pueden usarse con el puente de JavaSoft. Hay actualmente cerca de una docena de la categorías 2: construidos por APIs nativas por DBMSs. Hay algunos manejadores de la categoría 3 y actualmente hay por lo menos dos manejadores de la categoría 4, pero por el final de 1997, se esperaban más manejadores. Para más información consultar:

http://java.sun.com/products/jdbc

Los primeros vendedores de la categoría 3 ya disponibles son SCO, Open Horizon, Visigenic y WebLogic. JavaSoft e Intersolv sus líderes en conectividad de bases de datos, trabajaron juntos para reproducir el puente JDBC-ODBC y el Grupo de Prueba JDBC de Manejadores (JDBC driver test suite).

Resumen

Es una API de Java que permiten ejecutar sentencias SQL, JDBC no es un acrónimo pero comúnmente es conocido como Java DataBase Connectivity.

La combinación de Java y JDBC permite realizar aplicaciones sin necesidad de preocuparse de la base de datos a la que se accesará ni de las plataformas de desarrollo y ejecución.

CAPÍTULO V

API JDBC

Introducción

La API de JDBC esta integrada fundamentalmente por interfaces, objetos y excepciones con las cuales se pueden conectar a una base de datos para ejecutar instrucciones de SQL y obtener resultados. Primeramente se describen las interfaces de JDBC con las cuales se realiza la conexión, se realizan transacciones y se obtienen resultados

5.1 Interfaces

Java proporciona interfaces para conectarse a la base de datos y ejecutar instrucciones de SQL mediante interfaces de la API de JDBC en la cual se ejecutan instrucciones normales de SQL, instrucciones dinámicas y procedimientos almacenados.

CallableStatement

Proporciona métodos para ejecutar procedimientos almacenados que regresan valores del parámetro OUT. EL objeto CallableStatement hereda el objeto PreparedStatement, pero también agrega varios métodos para registrar parámetros para que sean parámetros OUT y además proporciona métodos para que los parámetrós sean devueltos desde el procedimiento almacenado

Un objeto *CallableStatement* posee una forma para llamar procedimientos almacenados (*stored procedure*) siendo un estándar actual para todos los DBMSs. Un *stored procedure* son sentencias almacenadas en una base de datos en una estructura similar a la de una función y que son llamadas de forma similar. Este llamado se escribe en una sintaxis de escape que puede tomarse de una de las dos formas siguientes: con un parámetro de resultado y la otra sin él. Un parámetro de resultado, un tipo de parámetro *OUT*, regresara algún resultado del *stored procedure*. Ambas formas pueden tener un número variable de parámetros de entrada (parámetros *IN*), de salida (*OUT*) o ambos (parámetros *INOUT*).

Connection

Es el objeto que proporciona a su aplicación java una conexión a la base de datos. Este objeto puede emplearse para crear los distintos objetos Statement para ejecutar instrucciones de SQL y procedimientos almacenados, también permite establecer las propiedades de transacción para la conexión.

Un objeto *Connection* representa una conexión una base de datos desde un aplicación Java. Una sesión de conexión incluye declaraciones o sentencias SQL que son enviadas, ejecutadas y los resultados se regresan sobre esa misma conexión. Una aplicación puede tener una con más conexiones a una o más bases de datos.

DatabaseMetaData

Proporciona diversos métodos para obtener información especifica sobre la base de datos, esta interfaz proporciona métodos para obtener el listado de las tablas de una base de datos especifica, así como de las llaves primarias, las columnas, etc.

La interface *DatabaseMetaData* provee varios métodos importantes para obtener información acerca de la base de datos y sus objetos. Por ejemplo, con esos métodos se puede obtener información sobre las llaves primarias de un tabla, determinar si la base de datos acepta *outer joins*, obtener todas las tablas actuales de la base, etc. Al ser una interface no se puede crear directamente, se debe de crear utilizando *Connection.getMetaData()*.

Driver

El objeto de la interfaz driver es un objeto Driver para una base de datos especifica y contiene información sobre la conexión de su aplicación java y proporciona información de la base de datos por ejemplo la version.

Un *driver* o controlador de bases de datos, es un objeto especifico de una base de datos que define como las sentencias son ejecutadas en una base de datos en particular. El primer objeto JDBC provee la interface *Driver*.La interface *Driver* provee varios métodos para obtener información acerca del actual controlador de la base de datos, así como provee un método que provee la conexión, utilizado para cominicarce con la base de datos.

PreparedStatement

Permite ejecutar instrucciones dinámicas de SQL y procedimientos almacenados. Las instrucciones dinámicas de SQL difieren de las instrucciones normales de SQL porque no se conocen los valores al momento de la creación.

La internase *PreparedStatement* hereda de *Statement* y difieren en:

La instancia de *PreparedStatement* contienen una declaración SQL que ha sido compilada. Esto es lo qué hace una declaración "preparada".

La declaración SQL contenida en un objeto *PreparedStatement* puede tener uno o más parámetros *IN*. Un parámetro *IN* es un parámetro cuyo valor no se a especificado cuando la declaración SQL se crea. En su lugar la declaración tiene un signo de interrogación ("?") cómo una variable a sustituir por un valor para cada parámetro *IN*. El valor para cada signo de interrogación debe ser proporcionado por su apropiado método *setXXX* antes de ejecutar la declaración.

Como el objeto *PreparedStatement* esta precompilado, su ejecución puede ser más rápida que los objetos *Statement*. Consiguientemente, una declaración SQL que se puede o necesita ejecutar muchas veces se recomienda crearla como un objeto *PreparedStatement* para aumentar la eficiencia.

Al ser una subclase de *Statement*, *PreparedStatement* hereda toda la funcionalidad de su antecesor. Además, agrega un conjunto de métodos que se necesita para colocar los valores para que se enviado a la base de datos, en el lugar de las variables (signos de interrogación) para los parámetros *IN*. También, los tres métodos *execute*, *executeQuery* y *executeUpdate* se modificaron para que ninguno de ellos tomen argumentos. Las formas de tomar los parámetros de esos métodos en *Statement* nunca deberían ser usadas con un objeto *PreparedStatement*.

Resulset

Es el objeto que se crea y utiliza para obtener información de una instrucción select de SQL, esta instrucción regresa un cursor que se puede utilizar por la interfaz resulset para navegar a través de todos los resultados regresados de select. Esta interfaz tiene varios métodos para obtener información de las diferentes columnas contenidas en el cursor.

Un *ResultSet* contiene todas las filas (registros) que cumplieron las condiciones en una declaración SQL y provee el acceso a los datos en esas filas mediante un conjunto de métodos. El método ResultSet.next se usa para moverse al registro siguiente del ResultSet, manejado el registro siguiente como el registro actual.

La forma general de un conjunto de resultados es una tabla con columnas y sus valores correspondientes regresados por un query.

ResulSetMetaData

Permite obtener información sobre un conjunto de resultados, el objeto ResulSetMetaData se crea a partir de un objeto ResulSet y permite obtener el numero de columnas, los nombres y tipos de columnas de un conjunto de resultados.

El objeto *ResulSetMetaData* provee varios métodos para obtener información acerca de los datos regresados dentro del objeto *ResulSet*. Esos métodos proveen una forma de obtener información como: el nombre y tipo de la columna, a que tabla pertenece dicha columna y si dicha columna acepta nulos; permitiendo con esto hacer un a aplicación más robusta.

tatement

Se crea a partir del objeto Connection y se puede usar para ejecutar instrucciones SQL estándar y procedimientos almacenados. El objeto proporciona dos métodos principales executeQuery() y xecuteUpdate(). Estos métodos le permiten ejecutar consultas de SQL .

Un objeto *Statement* se usa para enviar sentencias SQL a una base de datos. Hay realmente tres tipos de objetos *Statement*, todos actúan como recipientes (contenedores) para ejecutar sentencias SQL Sobre la conexión. *Statement.PreparedStatement*, que hereda de *Statement* y Callable*Statement* que hereda de *PreparedStatement*. Ellos se especializan para tipos particulares de envio de sentencias SQL. En objeto *Statement* se usa para ejecutar una sentencia simple sin parámetros. El objeto *PreparedStatement* se utiliza para enviar sentencias SQL precompiladas son o sin parámetros *IN* y el Objeto *CallableStatement* se usa para hacer llamados a *stored procedures* contenidos en la base de Datos.

La interface *Statement* provee métodos básicos para ejecutar sentencias y recobrar resultados, la Interface *PreparedStatement* agrega métodos para utilizar parámetros *IN* y el objeto *CallableStatement* agrega métodos para utilizar parámetros *OUT*.

5.2 Objetos

Se emplean para proporcionar a Java los tipos de datos específicos para la mayoría de bases de Datos.

Date

Proporciona un objeto que puede aceptar valores de tipo Date de base de datos y es heredado del Objeto Date normal de java

DriverManager

Proporciona una forma de realizar una conexión a la base de datos y se usa principalmente para administrar objetos Driver, tiene varios métodos para el registro de controladores, obtención de Conexiones y envió de información al flujo de salida de la base de datos.

Esta clase controla la interface entre la aplicación y el conjunto de Manejadores (*Drivers*) JDBC, adicionalmente provee mecanismos para su manipulación de cada uno de ellos. En el momento de que se inicia el objeto, trata de cargar todos los controladores que están registrados en la variable de sistema "jdbc.drivers", la cual puede contener una lista de ellos separados por dos puntos (:). Modificando esta propiedad el usuario puede determinar que manejaron utilizará su aplicación. Adicionalmente la clase de un manejador, puede ser cargado explícitamente, si se conoce el nombre, esto por medio del mecanismo ya conocido `Class.forName()`, Por ejemplo si se necesitará cargar la clase *imaginario.sql.MiBaseDatos* se debería de codificar como sigue:

Class.forName("imaginario.sql.MiBaseDatos")

DriverPropertyInfo

Es utilizado para manejar propiedades especificas de un objeto Driver

Esta clase se considera importante para programadores expertos, los cuales necesitan conocer ciertas propiedades de los manejadores, con lo que se puede conocer más detalle de ellos, esto se obtiene al llamar el método *getPropertyInfo* de la interface *Driver*, que devolverá un arreglo de éstos objetos. Como ejemplo podríamos ver que un objeto tiene la propiedad de "TipoDeTransacción", que de inicio dijera "AutoCommit", que fuera requerido que tuviera un valor al inicio de la conexión y que otros posibles valores fueran "AutoRollBack" y "EsperaUsuario".

Time

Proporciona un objeto que puede aceptar valores de tipo Time de base de datos y hereda del objeto Date

Timestamp

Proporciona un objeto que puede aceptar valores del tipo Timestamp de la base de datos

Types

Contiene una lista de valores enteros predefinidos que identifican cada uno de los tipos de datos disponibles

5.3 Excepciones JDBC

Cuando ocurre un error en Java se lanza una excepción, la API de JDBC contiene tres nuevas excepciones que pueden atraparse identificando varios errores en la ejecución de métodos JDBC

DataTruncation

Es lanzada cuando JDBC trunca un valor de datos en forma inesperada, esta excepción tiene métodos que brindan información sobre el valor de datos que fue truncado así como información sobre el error de truncamiento.

SQLException

Es lanzada por casi todos los métodos en la API de JDBC y contiene métodos que brindan información de el error y de el estado actual de la transacción SQL

Esta clase extiende de *java.lang.Exception* y permite conocer los errores que se presenten al accesar o trabajar con una base de datos. Cada que ocurre una excepción ofrece la siguiente información:

Una cadena que describe el error. Esta cadena también se conoce y maneja como el mensaje JavaException y se puede consultar con su método *getMessage()*.

Un código de error especificado por el vendedor de la base de datos y generalmente es un entero. Que generalmente es el número de error que regresa la base de datos actual al presentarse algún error.

Una cadena que maneja la convención XOPEN conocido como SQLstate. El posible valor de ésta cadena, se describe en las especificaciones XOPEN SQL

SQLWarning

Se genera cuando la base de datos emite una advertencia.

5.4 Otros Objetos JDBC

DataTruncation

DataTruncation extiende de la clase *java.sql.SQLWarning*. Cuando JDBC inesperadamente trunca (corta) un valor de un dato, éste emite una advertencia *DataTruncation* en procesos de lectura o emite un error crítico *DataTruncation* en procesos de escritura. El valor de *SQLstate* para un *DataTruncation* es "01004".

La siguiente lista muestra todos los métodos disponibles de éste objeto:

DataTruncation

Es el constructor de éste, el cúal crea un objeto de este tipo. Recordando que *DataTruncation* extiende de *java.sql.SQLWarning* y a su vez éste de *java.sql.SQLException* al llamar el constructor al mismo tiempo se actualizan las variables o propiedades *vendorCode* y *SQLState*, ya que serán actualizados al generarse un error. Sus valores son 0 y 01004, respectivamente. La declaración es la siguiente:

public DataTruncation(int index, boolean parameter, boolean read, int dataSize, int transferSize)

parámetro:

int index	El índice del parámetro o valor de la columna
Boolean parameter	Cierto si el valor del parámetro fue truncado
Boolean read	Cierto si la lectura fue truncada
int dataSize	Tamaño original del dato
int transferSize	Tamaño del dato después de ser truncado

getIndex

Regresa el índice de la columna o parámetro que fue truncado, se tendrá un valor de regreso de –1 si el índice de la columna o parámetro se desconoce, en este caso el *parameter* y *read* son ignorados. La declaración es la siguiente:

public int getIndex()

getParameter

Regresa cierto si el valor truncado fue un parámetro y falso si se trata de una columna. La declaración es la siguiente:

```
public boolean getParameter()
```

getRead

Regresa cierto si el valor se truncó al estar realizando una lectura de la base de datos y falso si se trataba de una escritura. La declaración es la siguiente:

```
public boolean getRead()
```

getDataSize

Regresa el número de *bytes* que debieron ser transferidos, en el proceso, si se esta realizando una conversión de datos, éste valor puede ser una aproximación. Si el valor es de –1 quiere decir que se desconoce el valor. La declaración es la siguiente:

```
public int getDataSize()
```

getTransferSize

Regresa el número de *bytes* que fueron transferidos, en el proceso. Si el valor es de –1 quiere decir que se desconoce el valor. La declaración es la siguiente:

```
public int getTransferSize()
```

Date

Date extiende de la clase *java.util.Date*. Esta clase es una adicional que permite a JDBC valores SQL que son del tipo SQL DATE. Agrega formatos y otras operaciones para soportarlos. Por lo que JavaSoft, recomienda en el manejo de JDBC, el uso de *java.sql.Date* en lugar de *java.util.Date*.

La siguiente lista muestra todos los métodos disponibles de éste objeto:

Date

Éste tiene dos formas distintas como constructores.

Es el constructor de éste, el cúal crea un objeto de este tipo. La declaración es la siguiente:

public Date(int year, int month, int day)	
parametro	
int year	Su valor inicia puede ser de 1900
int month	Sus posibles valores van de 0 a 11
int day	Sus posibles valores van de 1 a 31

Es el constructor de éste, el cúal crea un objeto de este tipo. Este tiene como objetivo manejar la diferencia en milisegundos desde "January 1, 1970, 00:00:00 GMT", por lo que hace uso de los métodos de su objeto padre. También se puede ver la documentación del objeto `java.lang.System#currentTimeMillis()`, la explicación de éste queda fuera del alcance de este documento. La declaración es la siguiente:

public Date(long date)	
parámetro	
long date	milliseconds since January 1, 1970, 00:00:00 GMT

valueOf

Permite convertir una cadena a un objeto *Date*, pero es indispensable que este en el formato "yyyy-mm-dd". Esté disparará o emitirá un error del tipo `java.lang.IllegalArgumentException` al presentarse algún valor incorrecto dentro de la cadena, como, que se pase una cadena vacia, que los valores del año, mes y día no sean validos con respecto a los mencionados en el primer constructor. La declaración es la siguiente:

public static Date valueOf(String s)	
parámetro	
String s	Cadena a convertir en formato "yyyy-mm-dd"

toString

Permite convertir el valor manejado de un objeto *Date* a una cadena con el formato "yyyy-mm-dd". La declaración es la siguiente:

public String toString ()

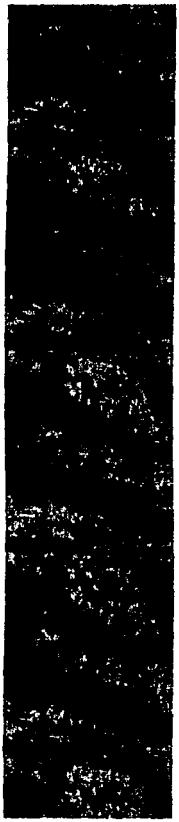
Los siguiente métodos existen dentro del objeto *Date*, pero están sobrecargados, por lo que directamente no se puede manejar horas con éste objeto, para ello se tendrían que llamar los métodos del objeto padre (refiérase a dicha información para más detalle). Si se consultan directamente emitirá un error del tipo `java.lang.IllegalArgumentException`.

getHours
getMinutes
getSeconds
setHours
setMinutes
setSeconds

Resumen

JDBC es una API de Java que permiten ejecutar sentencias SQL, JDBC no es un acrónimo pero comúnmente es conocido como Java DataBase Connectivity.

La combinación de Java y JDBC permite realizar aplicaciones sin necesidad de preocuparse de la base de datos a la que se accesará ni de las plataformas de desarrollo y ejecución.



CAPÍTULO VI

GUIA PARA USAR JDBC

Introducción

JDBC otorga una biblioteca estándar para acceder a una base de datos relacionales. Mediante la API JDBC se puede acceder a una amplia diversidad de base de datos SQL con exactamente la sintaxis de Java. Es importante destacar que aunque JDBC estandariza el mecanismo para conectarse a base de datos, la sintaxis para enviar consultas y confirmar transacciones, así como la estructura de datos que representa al resultado no intenta estandarizar la sintaxis SQL. Por ello puede utilizar cualquier extensión SQL que utilice determinado proveedor. No obstante, dado que la mayoría de las consultas siguen una sintaxis SQL estándar, utilizar JDBC permite cambiar los host de la base de datos, puertos e incluso proveedores de bases de datos con pequeños cambios en el código.

En el capítulo anterior se mencionaron las interfaces, objetos y excepciones JDBC, en este capítulo se trata de la forma en la que se utilizan para realizar una conexión de base de datos.

6.1 Pasos Para Usar JDBC

Existen siete pasos normales para acceder a una base de datos:

1.Cargar el controlador JDBC

2.Definir el URL de la conexión

3.Establecer la conexión

4.Crear un objeto statement

5.Ejecutar una consulta o actualización

6.Procesar los resultados

7.Cerrar la conexión

Cargar El Controlador

El controlador es el software necesario para realizar la comunicación con el servidor de bases de datos, Para cargar el controlador, todo lo que necesita es cargar la clase adecuada utilizando el método Class.forName que toma una cadena que representa a un nombre de clase completamente calificado (que incluya nombre de paquetes) y carga la clase correspondiente. Esta llamada podria arrojar una ClassNotFoundException, por lo que se deberá incluir dentro de un bloque try/cath.

Ejemplo

```
try{
  Class.forName("connect.Microsoft.MicrosoftDriver");
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Class.forName("com.sybase.jdbc.SybDriver");
```

```
}catch (ClassNotFoundException cnfe){
  System.err.println("Error al cargar el controlador" + cnfe);
}
```

Definición Del URL

Un URL proporciona la manera de identificar una base de datos, en el momento que ya tenemos cargado el controlador JDBC se necesita especificar la ubicación del servidor de bases de datos. Los URLs que apuntan a bases de datos están formados por tres cosas:

```
jdbc:<subprotocol>:<subname>
```

El protocolo en un URL es siempre JDBC

<subprotocol> el nombre del driver o el nombre de l mecanismo de conectividad de base de datos. Un nombre común de un subprotocolo es "odbc"

<subname> Manera de identificar la base de datos. El subname puede variar dependiendo del subprotocolo.

Ejemplo

```
jdbc:odbc:fred
```

En este ejemplo el subprotocolo es "odbc", y el subname "fred" es la fuente de datos odbc.

Establecimiento De La Conexión

La manera estándar de establecer una conexión con una base de datos es llamar el método *DriverManager.getConnection*. Este método toma una cadena que contiene un URL. La clase

DriverManager, es referida como un manejador de capas JDBC, los intentos de ubicar un manejador que pueda conectarse a la base de datos representada por ese URL. La clase *DriverManager* mantiene una lista de clases *Driver* registradas y cuando el método *getConnection* se llama, verifica cada manejador en una lista hasta que encuentre uno que pueda conectarse a la base que datos especificada en el URL. El método *connect* de la clase *Driver*, usa éste URL para establecer la conexión realmente.

Esto puede ser útil en el raro caso que dos manejadores pueden conectarse a una base de datos y el usuario quiere explícitamente seleccionar un manejador particular. Sin embargo, es mucho más fácil, simplemente dejar que el *DriverManager* abra una conexión. Es siguiente código ejemplifica la apertura de una conexión a una base de datos ubicada en el URL "jdbc:odbc:wombat" con identificados del usuario "mi-usuario" y contraseña "java123".

```
String url = "jdbc:odbc:wombat";
Connection con = DriverManager.getConnection(url, "mi-usuario", "java123");
```

Para realizar la conexión real a la red, es necesario pasar el URL, el nombre de usuario de la base de datos y la contraseña al método getConnection de la clase DriverManager, cabe señalar que el método getConnection arroja una SQLException, por lo que es necesario un bloque try/cath

Una parte opcional de este paso es buscar información de la base de datos a través del método getMetaData de Connection. Este método devuelve un objeto DatabaseMetaData que tiene metodos para conocer el nombre y versión de la base de datos (getDatabaseProductName,getDatabaseProductVersion)o del controlador JDBC (getDriverName, getDriverVersion)

Creacion De Una Instrucción

Una vez que la conexión a una base de datos se establece, una conexión puede usarse para enviar sentencias SQL. Un objeto *Statement* se crea con el objeto *Connection* con su método create*Statement*, como se muestra en el código siguiente:

```
Connection con = DriverManager.getConnection(url, "mi_usuario", "mi_contraseña")
Statement stmt = con.createStatement();
```

La sentencia que se enviará a la base de datos se proporciona como argumento a uno de los métodos para ejecutar un objeto *Statement*:

```
ResultSet rs = stmt.executeQuery("select a, b, c from Tabla");
```

Un objeto Statement se utiliza para enviar consultas y comandos a la base de datos y se genera desde Connection como sigue:

```
Statement instruccion = conexion.createStatement();
```

Ejecucion De Una Consulta

La interface *Statement* provee tres métodos diferentes para ejecutar las sentencias SQL, *executeQuery*, *executeUpdate* y *execute*. Para determinar cual puede usarse depende de la complejidad y los resultados que arroje la sentencia.

El método *executeQuery* se diseño para sentencias que producen un conjunto único de resultados, tales como sentencias *SELECT* (palabra reservada o sentencia SQL para obtener registros de la base de datos).

El método *executeUpdate* se usa para sentencias INSERT, UPDATE O DELETE (palabras reservadas o sentencias SQL para insertar, actualizar o borrar registros de la base de datos) y también SQL DDL (Data Definition Languaje, lenguaje de definición de datos) sentencias como CREATE TABLE y DROP TABLE. El efecto de una sentencia INSERT, UPDATE O DELETE es una modificación de una o más columnas (campos) en cero o más renglones (registros o filas) en una tabla. El valor de regreso del *executeUpdate* es un entero que indica el número de filas afectadas. Para sentencias como CREATE TABLE y DROP TABLE, que no operan sobre filas, el valor de regreso de *executeUpdate* es siempre cero.

El método *execute* se usa para sentencias que producen un o más conjuntos de resultados, más de un conteo de actualización o una combinación de los dos. Generalmente éste es un aspecto avanzado que la mayoria de los programadores no necesitarán.

Todos los métodos para ejecutar sentencias cierran llamados anteriores a objetos *Statement* aún que tenga resultados pendientes. Siempre se necesita completar cualquier procedimiento del objeto ResulSet actual antes de ejecutar un objeto *Statement*.

Se debe de destacar que la interface *PreparedStatement*, que hereda de la interface *Statement*, tiene sus propias versiones de los métodos *executeQuery*, *executeUpdate* y *execute*. Los objetos *Statement* por si mismos no contienen una declaración SQL; por lo tanto, se debe de especificar como argumento del método *Statement.execute*. El objeto *PreparedStatement* no proporciona una sentencia SQL como un parámetro a estos métodos por que ellos ya contienen una sentencia precompilada. El objeto *CallableStatement* hereda las formas de estos métodos del objeto *PreparedStatement*. Si se usan parámetros en la sentencia con los métodos de los objetos *PreparedStatement* o *CallableStatement* ocasionan como resultado SQLException.

Los objetos *Statement* se cerrarán automáticamente por el recolector de basura Java. No obstante, se recomienda como una buena práctica del programador que ellos cierren los objetos cuando no se necesiten más. Esto libera los recursos del DBMS inmediatamente y ayudan a evitar problemas potenciales de memoria.

En el momento en que tenemos un objeto Statement, podrá utilizarlo para enviar consultas SQL con ayuda del método executeQuery, el cual regresa un objeto de tipo ResulSet

Para modificar la base de datos es necesario utilizar el método executeUpdate.

Procesamiento De Resultados

La mejor forma de manejar los resultados es procesarlos fila por fila utilizando el método next de ResultSet para moverse por una tabla una fila a la vez.

Dentro de una fila, ResultSet provee diversos métodos getXxx que toman como argumento el índice o nombre de una columna y devuelven el resultado en diversos tipos de Java. Por ejemplo, utilizar getInt si el valor es un integer, getString para un String, y así sucesivamente para los otros tipos de datos. Aunque si solo es necesario mostrar los datos es recomendable utilizar getString sin importar el tipo de datos de la columna real. Es importante recalcar que si se utilizar la forma que lleva el índice de la columna, es necesario tomar en cuenta que los índices empiezan en 1 de acuerdo con la convención SQL y no con 0 como en matrices, vectores, etc del lenguaje Java.

Cierre De La Conexion

En el momento que ya no se desean realizar operaciones con la base de datos, es necesario cerrar la conexión, pero se recomienda que sea en el momento que se esta seguro que no se necesita otra operación, por que la carga adicional de abrir una conexión es usualmente grande.

Para cerrar explícitamente la conexión únicamente se escribe la siguiente línea de código:

```
Conexión.close();
```

Resumen

El presente capítulo es una guía para realizar la conexión a una base de datos, para efectuar dicha conexión es necesario seguir siete pasos básicos:

Cargar el controlador JDBC: El controlador es el software que sabe como comunicarse con el servidor de la base de datos.

Definir el URL de la conexión: Una vez que se ha cargado el controlador JDBC se necesitará especificar la ubicación del servidor de base de datos.

Establecimiento de la conexión: Para hacer la conexión real a la red, pase el URL, el nombre del usuario de la base de datos y la contraseña al método getConnection de la clase DriverManager.

Creación de una Instrucción: Un objeto statement se utiliza para enviar consultas y comandos a la base de datos.

Ejecución de una consulta: Cuando se tiene un objeto statement se puede utilizar para enviar consultas SQL con ayuda del método executeQuery.

Procesamiento de Resultados: Con ayuda del metido next de ResultSet se puede mover por una tabla una fila a la vez.

Cierre de la conexión: Cuando ya no se requieren operaciones con la base de datos se debe cerrar la conexión.

CAPÍTULO VII

JAVA Y OTRAS TECNOLOGIAS

Introducción

En la actualidad no existe un lenguaje perfecto, es común para los programadores usar más de un lenguaje para un proyecto, ya que ciertos lenguajes pueden ser necesarios para interactuar con otras aplicaciones o librerías y otros son imprescindibles para una aplicación en particular.

La mayoría de los sistemas están escritos en mas de un lenguaje, pero la parte principal generalmente esta escrita en un lenguaje altamente estructurado como Java o C++ y las interfaces de usuario escritas en un lenguaje mas flexible como los de Script.

Este capítulo trata de lenguajes de programación que comparten características con Java con el objetivo de conocer que lenguajes pueden trabajar en conjunto con java o bien para ser utilizados en lugar de Java.

Para su mejor manejo, este capítulo es dividido en lenguajes de uso general, lenguajes seguros y lenguajes de Script.

7.1 Lenguajes De Uso General.

Existen varios lenguajes de uso general que comparten muchas de las características de Java para una aplicación general.

MODULA 3

Es un pariente de Modula 2, aunque no propiamente una extensión de modula 2, ya que aunque se le agregaron características, otras se eliminaron, resultando un lenguaje muy simple, característica que comparte con Java.

Una herramienta flexible para el diseño de programas grandes y complejos es el lenguaje Modula 3 pues presenta facilidades de programación a nivel modular, estructurado, orientado a objetos y concurrente

Modula 3 es un lenguaje imperativo de propósito general que maneja excepciones, programación orientada a objetos, compilación separada, programación concurrente y programación para Internet. Modula 3 se diseño para programar aplicaciones grandes, pero también es conveniente para programar sistemas de bajo nivel. Modula 3 utiliza módulos similares a los paquetes en Java.

EIFFEL

Eiffel constituye un amplio entorno de desarrollo de software (Eiffel de ISE) basado en un método que cubre todo el ciclo de vida del software, no sólo la implementación, sino también el análisis, el diseño y el mantenimiento. El entorno se basa en el lenguaje Eiffel, aplicándose minuciosamente los principios de la tecnología de objetos como Java e implementándose los conceptos de Design by Contract con el fin de crear aplicaciones de gran confiabilidad, con capacidad de extensión y reutilizables. El lenguaje Eiffel de ISE está especialmente orientado a sistemas complejos y de gran tamaño y se utiliza en las principales organizaciones financieras y de defensa, así como en otras

industrias para desarrollos de especial importancia. Las universidades de todo el mundo (como la Universidad de Monash) también utilizan Eiffel en la enseñanza de programación e ingeniería de software para todos los niveles.

COMMON LISP y CLOS

Common LISP es un dialecto normal del lenguaje LISP, se ha usado principalmente por la comunidad de la inteligencia artificial. LISP tiene una reputación fuerte como un vehículo bueno para desarrollo de aplicaciones grandes y complejas.

LISP siempre ha sido un lenguaje permisivo, con variables dinámicas, los programadores pueden preparar la implementación de una función para pueda cambiar mientras el sistema está corriendo.

El Common Lisp Object System (CLOS) es una extensión de LISP para el uso de objetos construida en Common LISP. CLOS soporta la herencia múltiple, es dinámico y flexible. Los objetos tienen slots que son el equivalente de instancias o variables de la clase.

Common Lisp soporta las excepciones, y tiene una gran variedad de tipos de datos útiles ya construidos, proporcionados en bibliotecas. La facilidad con que pueden representarse las estructuras de datos sumamente complicadas en el LISP siempre ha sido uno de las grandes fuerzas del lenguaje, junto con su apoyo excelente para la matemática de alto nivel, además posee otras bibliotecas de funciones útiles.

Dylan

Dylan es un nuevo lenguaje desarrollado recientemente en Apple, se pensaba que Dylan era un lenguaje dinámico como LISP, pero con más uniformidad y mando. (El nombre Dylan viene de lenguaje

dinámico) tiene muchas similitudes a CLOS, aunque la sintaxis es más convencional. Dylan es orientado objetos como Eiffel y Smalltalk.

Dylan fue diseñado para ser más fácil de compilar que CLOS y diseñado para ser más simple, sin sacrificar las características dinámicas de LISP que los diseñadores de Dylan consideran importantes para construir sistemas grandes y complejos.

Smalltalk

Como C++, Smalltalk es un descendiente directo de Simula, el lenguaje orientado a objetos original. Sin embargo, se diseñaron Smalltalk y C++ con metas muy diferentes. Smalltalk se diseñó por investigadores en el cetro de Investigaciones Xerox en Palo Alto en paralelo con su exploración de ideas básicas y mecanismos de programación orientada a objetos. Smalltalk es un lenguaje orientado a objetos de los más puros: Todos los funcionamientos en el lenguaje, incluso la aritmética básica, asignación inconstante, y estructuras del mando, son cumplidos por métodos que ejecutan en el contexto de objetos. Incluso se representan bloques del código, como los cuerpos del método, como los objetos. El poderío de esta pureza parece extremo a programadores más acostumbrado al modelo de la programación procesal tradicional, pero era una de las maneras que los investigadores se obligaron explorar los límites de sus nuevas ideas. El lenguaje resultante es sumamente poderoso y muy agradable de.

Como Java, Smalltalk más que un lenguaje, es una plataforma para programar. Smalltalk siempre se ha definido por lo que se refiere a una maquina virtual, y tiene una biblioteca grande de clases para las ventanas, y la gráficos-mayoría de las mismas cosas que la biblioteca de Java proporciona. Como muchos de los dialectos del LISP, Smalltalk incorpora también un ambiente de desarrollo

La sintaxis de Smalltalk y la completamente pura base orientado a objetos son únicas entre los lenguajes de programación mayores, pero en muchos otro respeta Smalltalk es similar a LISP. Aunque se teclean los valores, una variable dada puede almacenar un valor de cualquier tipo. Es interpretado y dinámico, para que cualquier parte del sistema pueda modificarse o reemplazarse mientras el systema esta activo.

Aunque Smalltalk no tiene particularmente un perfil alto, normalmente se usa en varias comunidades, más notablemente la industria financiera.

7.2 Lenguajes De Script

La estructura, encapsulación y tipo de verificación de los lenguajes de propósito general ayudan a los desarrolladores a manejar la complejidad y localizar errores en programas complicados y largos, pero en pequeños proyectos es mas conveniente utilizar Lenguajes de Script.

Los lenguajes de Script son permisivos e interpretados, son fáciles de aprender y se pueden escribir muy rápido. Los prototipos experimentales se pueden construir y modificar fácilmente.

Con frecuencia los lenguajes de Script son incrustados en programas.

JAVASCRIPT

Netscape en un esfuerzo por extender la funcionalidad de su navegador (browser), desarrolló un lenguaje de programación que se puede colocar dentro de archivos HTML. Originalmente fue llamado LiveScript, pero después fue renombrado a JavaScript con la idea de capitalizar la fama de Java, lenguaje desarrollado por Sun Microsystems.

JavaScript fue diseñado para ser un lenguaje de elaboración de scripts que pudieran incrustarse en archivos HTML. No es compilado, sino que, en vez de ello, es interpretado por el navegador. A diferencia de Java, que primero es convertido a código de byte fácil de interpretar, JavaScript es leído por el navegador como código fuente. Esto facilita el aprendizaje de JavaScript mediante ejemplos, debido a que se puede ver la manera en que otros usan JavaScript en sus páginas.

Originalmente se llamaba LiveScript, es un lenguaje desarrollado por Netscape con una sintaxis básicamente basada en Java aunque con la limitación de que no existe interface entre Java y JavaScript.

Java Script provee diferentes tipos de variables y procedimientos, tiene objetos de ordenamiento pero no es un lenguaje propiamente de objetos ya que realmente son arreglos asociativos.

Java Script Provee una librería de funciones útiles como Math, Netscape Navigator tiene objetos que permiten a los scripts controlar el visualizador de alguna manera y trabajar con elementos de formas y otras partes de documentos Html.

TCL

El nombre TCL viene de Tool Command Languaje y esto refleja su meta original; ser un común y reusable lenguaje de script para aplicaciones, Tcl es implementado como una libreria de C que puede ser ligada a una aplicación. Existe una interface Java disponible.

Tcl es extremadamente simple y debido a su diseño para trabajar con aplicaciones brinda funcionalidad tal como variables, estructuras simples de datos, manipulación de cadenas y con estructuras de control incluyendo manejo de errores. El lenguaje provee también facilidades de I/O incluyendo soporte de red.

PERL

PERL (Practical Extraction and Reporting Languaje) a diferencia de Tcl, Perl no fue diseñado como lenguaje de script, pero su poder y las facilidades de manipulación de texto han hecho el favorito para aplicaciones web.

Perl fue diseñado por Larry Wall y se baso en lenguajes UNIX como Bourne Shell, sed, awk y C. La sintaxis es un poco compleja pero no es difícil de aprender, es posible escribir código confuso en Perl pero con un poco de disciplina se pueden escribir programas fáciles de leer.

PHYTON

Es un lenguaje para Scripts diseñado por Guido Van Rossum. y a diferencia de Tcl que fue diseñado especificamente como un lenguaje de script para aplicaciones y de Perl que fue diseñado para tareas de manejo de texto, Pitón fue diseñado como lenguaje de propósito general para desarollar programas y prototipos.

Phytón es un lenguaje permisivo, tiene estructuras de control, procedimientos, estructuras de datos como arreglos asociativos, caracteristicas de lenguaje orientado a objetos y posee una gran cantidad de librerias de clase.

Al igual que Perl y Tcl puede ser incrustado en una aplicación.

VBSCRIPT

El Visual Basic Script (en adelante VBScript) es un lenguaje de script, directamente derivado de Visual Basic. Los lenguajes de script son versiones recortadas de otros lenguajes. Estas versiones se usan para su integración en páginas web. Un código escrito en un lenguaje de script se incorpora directamente dentro de un código HTML y se ejecuta interpretado, no compilado. Dos son los lenguajes de script mas importantes: el VBScript (derivado de Visual Basic) y el JavaScript (derivado de Java).

Decimos que los lenguajes de script se ejecutan interpretados, no compilados. Esto significa que un código escrito en un lenguaje de script no sufre ninguna transformación previa a su ejecución. Cada linea de código es traducida a lenguaje máquina justo antes de su ejecución. Después es ejecutada y la traducción no se conserva en ningún sistema de almacenamiento (como discos, cintas, etc). Si es necesaria otra ejecución, el intérprete se verá abocado a realizar una nueva traducción de cada línea de código.

SCHEME

Scheme es un dialecto de Lisp, que es un lenguaje declarativo del tipo funcional. Un lenguaje declarativo es uno que –al contrario de uno imperativo como Pascal o C- se supone que tiene un mayor nivel de abstracción, más parecido a la forma de pensar de los humanos.

Los lenguajes declarativos, y especialmente los modernos, han sido desarrollados principalmente con el fin de ser usados en el proceso de representar el conocimiento humano y reproducir sus capacidades deductivas y cognitivas, de acuerdo al desafío emprendido en el campo de la inteligencia artificial. Además, estos lenguajes son especialmente útiles para representar problemas de modelación y simulación en las distintas áreas de la ingeniería.

Resumen

Frecuentemente la elaboración de un sistema se realiza en mas de un lenguaje, es común que la parte principal se elabore en un lenguaje altamente estructurado y otras se elaboren en otros lenguajes.

Actualmente utilizamos o un lenguaje de uso general como: Modula 3, Eiffel, Common Lisp , CLOS, Dylan y Smalltalk o lenguajes de Script como: Java Script, TCL, PERL, Phyton, VBScript y Scheme cada uno con características especificas que se adecuan a ciertas necesidades, por lo que es importante conocer sus características para que el programador pueda saber que herramienta utilizar en determinado momento.

Ninguno de los idiomas existentes son perfectos, así que habrá mas. Java debe convivir y competir con los lenguajes que existen actualmente en el área de sistemas. La llave del éxito de un programador es saber que lenguaje utilizar para el problema que se este enfrentando.

CAPÍTULO VIII

FUTURO DE JAVA

Introducción

En este capítulo se realiza un interesante análisis acerca del futuro de Java, sus potencialidades y su protagonismo en el cambio del escenario para los próximos años.

Se piensa que la tecnología más prometedora del año 2002 y para los años futuros será Java. Con una rapidez sin precedente, Java ha pasado de un concepto revolucionario a una plataforma para aplicaciones de corporación estimulando la industria de la Tecnología de Informática.

Ya se encuentra funcionando en 70 millones de computadoras, mientras miles más de programas Java están siendo desarrollados. Desde que salió a la luz este lenguaje de programación en mayo de 1995, el apoyo industrial para Java ha sido sin precedente.

Este Capítulo se divide en dos partes: la primera que es Java y las empresas que trata de cómo se espera que avance Java en el mundo empresarial y de consultoría y la segunda en la que se habla de la incursión de java en dispositivos electrónicos.

8.1 Java En Las Empresas

Java puede ser adaptada poco a poco a un costo más bajo que otras tecnologías. Debido a su independencia de plataforma, Java parte de una infraestructura establecida, influenciando los sistemas existentes tales como las computadoras centrales legendarias y las Pcs. En lugar de exigir que las compañías abandonen sus inversiones pasadas, Java las ayuda a maximizar la ganancia en ellas.

Un creciente número de herramientas y aplicaciones de desarrollo de soporte de Java han sido colocadas en organizaciones, aliviando a las organizaciones de muchas de las complicaciones que típicamente causan demoras en la implementación de tecnologías de vanguardia.. Muchas de las otras compañías están utilizando Java para mejorar la eficacia. Casi toda la industria del software ha licenciado a Java, o la está evaluando.

Las compañías en telecomunicaciones, transporte y banca están planeando combinar el alcance de Java con el crecimiento de Internet. Particularmente para aplicaciones donde la PC es muy complicada y costosa de operar, los dispositivos de Java ofrecen una nueva opción poderosa. Después que los clientes inconsistentes de Java se expandan dentro de la empresa, la nueva gama serán dispositivos de Computación Java, tales como teléfonos celulares, beepers, cajas de conversión para acceso de aparatos de televisión al Internet y otros dispositivos.

La corporación Corel ha anunciado tener planes para desarrollar un organizador de tamaño de la palma de la mano basado en Java con funciones para Internet que podría aparecer a fines de invierno o primavera (de los EU.). Muchas empresas están modernizando sus aplicaciones empresariales utilizando Java, tales como Oracle.

La infraestructura tecnológica necesaria para la nueva generación de servicios en Internet, refleja la actual asincronía entre la velocidad con que se desarrolla tecnología y la que llevan los negocios en Internet. Un gran número de compañías con gran experiencia han proporcionado la infrestructura necesaria a una nueva generación de empresas puntocom. Fabricantes como Oracle, SAP, Vigente o Citrix; consultoras como Accenture y proveedores de equipo como Cisco siguen siendo importantes para el desarrollo de Internet.

8.2 Java Y La Tecnología

Java es una fuerza poderosa que está arrasando a la industria tecnológica. Cumple con algunas de las tendencias más importantes de computación: la unión de diversas redes; la migración del poder de procesamiento de la computadora central a la del escritorio a la de la red; el balance de las necesidades de administradores de sistema en relación a aquellos de los usuarios individuales; reduciendo el costo de administración del cliente y el costo total de propiedad. Java se desarrolla y ayuda a unificar la infraestructura existente como redes de cliente/proveedor, plataformas de computación diversas, sistemas de legado, intrared y la Internet. Todo eso explica la razón por la cual muchos lideres de corporaciones ahora ven a Java como la nueva fase en la estrategia de computación empresarial.

La telefonía inalámbrica y la tecnología para los nuevos servidores en internet han sido los asuntos de mayor interés en las Java Expo.

Ingenieros de Sun Microsystems y de una filial Telefónica mostraron desde la descarga de un videojuego a una terminal móvil hasta el compartir una agenda a través de distintas vías (terminales GSM y GRPS).

La tecnología necesaria, que coordina Jon Bostrom, (responsable del proyecto i-Mode en Sun), se desarrolla en parte de España y estará disponible a finales de año denominada JavaME.

Con seis años de vida, el Java de Sun ha captado la mayoría de los servidores de ahora busca su implantación en todo tipo de dispositivos clientes, en especial la televisión interactiva y la telefonía móvil. Pero se encuentra con la frontal oposición de Microsoft con su estrategia .NET.

La estrategia .NET intenta sustituir el papel de Java. La arquitectura .NET es otro intento de Microsoft para crear un plataforma propietaria.

Las interfaces de Java son propuestas por los actores de cada sector de la industria. Y Sun propone a la industria su arquitectura Sun One, basada en la unión de Java con XML. Sun One es una tecnología que no ata a un único proveedor y plataforma porque las aplicaciones funcionan en un mainframe, un sistema Unix o Windows. Si una empresa se inclina por .NET se ata a una tecnología, una plataforma y a un único proveedor.

El mercado se va llenando cada vez más con pequeños dispositivos que cada vez realizan funciones más sofisticadas. Sun ha anunciado una nueva tecnología abierta, concebida específicamente con las últimas tecnologías inalámbricas en mente.

La tecnología de Sun se pondrá al día con los rápidos avances que están siendo conseguidos por los desarrolladores de dispositivos inalámbricos por lo que próximamente conoceremos teléfonos, beepers y otros dispositivos que adoptarán la plataforma Java.

Con java se puede reproducir sonido directamente desde el navegador, se pueden visitar home pages con animaciones, se puede enseñar al navegador a manejar nuevos formatos de archivos, e incluso, cuando se pueda transmitir video por las lineas telefonicas, el navegador ya estara preparado para mostrar esas imágenes.Smile

Resumen

La adopción del lenguaje Java en las empresas resulta ser mas económico debido a la independencia de plataforma, se puede trabajar con Java con una infraestructura ya establecida evitando también las demoras en la implementación.

Muchas empresas están modernizando sus aplicaciones empresariales utilizando Java y compañías de telecomunicaciones, transporte y banca están planeando combinar el alcance de Java con el crecimiento de internet.

Con seis años de vida, el Java de Sun ha captado la mayoría de los servidores de ahora busca su implantación en todo tipo de dispositivos clientes, en especial la televisión interactiva y la telefonía móvil.

El mercado se va llenando cada vez más con pequeños dispositivos que cada vez realizan funciones más sofisticadas.

CAPÍTULO IX

PROYECTO

Introducción

El proyecto que se desarrolló, tiene como proposito dar a conocer, con una aplicación muy común, el uso de JDBC desde una aplicación Cliente / Servidor.

El proyecto en su contexto general es una aplicación que permite ejecutar sentencias SQL (permitida por el DBMS) en cualquier base de datos que se requiera.

9.1 Descripción

De inicio se planteara el problema a solucionar.

Se requiere en todo proyecto o desarrollo de sistema, cuente con una aplicación que permita ejecutar sentencias SQL hacia la base de datos que almacenará la información propia del sistema.

Esto nos da como resultado, tener que usar la herramienta que incluye el manejador de bases de datos al ser instalado. Pero, nos encontramos con el problema de que, la herramienta no es flexible, fácil de usar o no presenta la información de una manera agradable a la vista. En ocasiones se puede buscar en Internet alguna herramienta libre o de prueba, que nos permita cubrir, ésta necesidad, hasta cierto punto. Pero, esta herramienta sigue siendo limitada a dicha base de datos o plataforma (sistema operativo).

Otro punto importante es: el equipo de trabajo de un proyecto, que se acaba de familiarizar con alguna herramienta similar para interactuar con la base de datos, al cambiar de base de datos necesitará, lo más seguro, tener que aprender a usar otra para otra base de datos. Esto pasa muy a menudo en empresas que desarrollan varios tipos de proyectos, sobre varias bases de datos.

Por tal motivo, queremos mostrarles un uso genérico que se le puede dar a JDBC y explotando la bondad de Java, que como el lema de SUN nos dice, "desarrolla y ejecuta en donde sea".

El proyecto fue desarrollado 100% Java y es independiente a los controladores (Drivers), ya sea Java u ODBC, que se instalan y configuran independientemente; con esto nos la posibilidad de interactuar con una gama más amplia de base de datos, casi en cualquier plataforma.

9.2 Antecedentes

Hemos participado en varios desarrollos de sistemas o tomado cursos, con las bases de datos más comunes como: Informix, DB2, Sybase, SQL Server, Oracle, MySQL, Access, Dbase y Clipper.

Al trabajar con cada una de esas bases de datos requerimos de tener que aprender una nueva herramienta propia del manejador de base de datos, para poder hacer consultas parciales o temporales, en el momento del desarrollo. En ocasiones las herramientas que hemos utilizado, han sido en modo consola o modo de comandos, no muy flexibles, como el caso de DB2, donde se usa un emulador (en modo caracter) para poderse conectar a Mainfraims, ejecutar utilerías para realizar consultas y lo peor, es que la salida de datos se limita a aproximadamente 80 caracteres por línea y si el registro es más largo que esa cantidad, invade renglones abajo y la información se hace ilegible.

9.3 Arquitectura

Flujo de comunicación

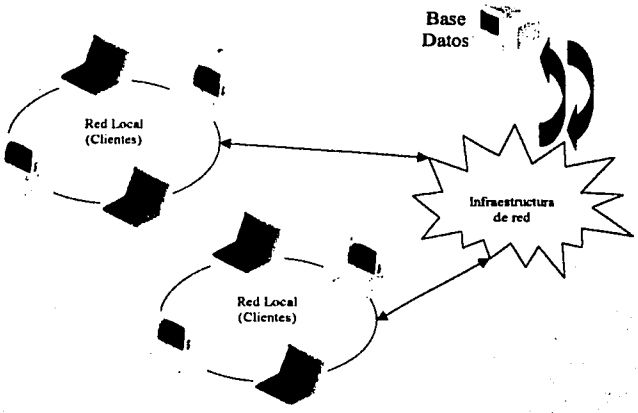


Figura 9.1

9.4 Requerimientos

Para ejecutar el proyecto se requiere instalar, en la computadora en donde se ejecutará, el JRE (ambiente de ejecución de Java) ó el JDK (si se quisieran hacer cambios en el código), se recomienda que sea de la versión 1.2.x o superior. Aún que el corazón del proyecto que se conecta a la base de datos, puede ejecutarse en la versión 1.1.6 o superior.

Por otro lado instalar la base de datos con que se desee trabajar, además de contar con los datos necesarios para conectarse a ella. Además de contar con los controladores apropiados para poderse conectar ya sean JDBC u ODBC.

En el CD se tiene la base de datos MySQL, así como su controlador JDBC

El proyecto se ha probado hasta ahora en Oracle y MySQL, con controladores JDBC puros. También se probó con Access utilizando OBDC.

9.5 Estructura Del Proyecto

Antes de continuar, explicaremos la finalidad de cada archivo, que se desarrollaron para éste proyecto:

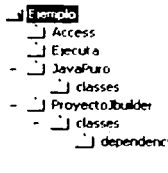


Figura 9.2

Ejemplo: Contiene el código fuente y codigo compilado del proyecto.
Access: Base de datos Access y archivo para registrar el DSN de esa base.
Ejecuta: Archivos compilados necesarios para la ejecución, es independiente al resto de los directorios. Versión de distribución.
JavaPuro: Unicamente archivos java y class dentro de *classes*. Archivos 100% java puro.
classes: Contiene el codigo compilado del proyecto, cuando se compilan los archivos Java que están dentro de JavaPuro se remplazan todos los existentes en éste directorio. Los archivos class actuales, también se deberan de actualizar en el directorio Ejecuta, para tener actualizada, la versión de distribución.
ProyectoJbuilder: muy similar a JavaPuro, pero este contiene un proyecto de JBuilder 4.0, el resultado es el mismo, sólo que se tiene usaría una herramienta gráfica para su modificación. Recomendable si se tiene JBuilder.

Ahora veremos los archivos Java, internamente tienen sus comentarios, explicando el código:

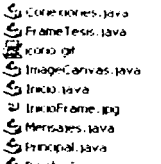


Figura 9.3

Conexiones: Archivo llamado el corazón, éste interactua con la base de datos, dejando en arreglos los datos devueltos por la base, todos los demás són mera presentación. Éste es el que usa JDBC. Abre y cierra la conexión, cada que se manda una sentencia SQL.
FrameTesis: Archivo que contiene, la interface de interacción con el usuario y en donde se presentarán los datos devueltos por la base de datos. Este llamará a *Conexiones*.
icono: Pequeña imagen, que aparece en la esquina superior derecha de todas las pantallas.
ImageCanvas: Archivo que permite cargas las imágenes.
Inicio: Archivo con el llamado de la pantalla de inicio, carga la imagen *InicioFrame*. Este llamará a *Principal*.
InicioFrame: Archivo con la imagen de inicio y de acerca de.
Mensajes: Archivo con el manejo de varios tipos de envio de mensajes, ya sea a modo consola o atraves de una ventana gráfica.
Principal: Archivo con la pantalla principal en donde el usuario introduce sus datos necesarios para conectarse a la base de datos. Este llamará a *FrameTesis*.
PruebaConexio: Archivo con hace el llamado de Conexiones, utilizado como prueba sin gráficos, para corroborar que Conexiones trabaje correctamente. Archivo no necesario dentro la versión de distribución.

9.6 Ejecución Del Proyecto

Una vez que se conocen todos los archivos que comprenden el proyecto, veremos el modo de ejecución.

Se debe de contar con el JDK o JRE instalados y funcionando correctamente, adicionalmente deberán de existir el o los controladores necesarios para la conexión a la base de datos en la misma computadora en donde se tiene el proyecto y se ejecutará. Por otra parte, se deberá de tener la base de datos instalada y en ejecución, para que nos permita interactuar con ella. La instalación de todas éstas herramientas, queda fuera del alcance de este documento.

Se recomienda, para que no se edite el archivo de inicialización para asignar nuevos valores a la variable CLASSPATH, cuando se requiere utilizar otro controlador, se escriba como parámetros adicionales del comando *java* al ejecutar el proyecto. Para utilizar un controlador ODBC, no es necesario modificar el CLASSPATH o usar el parámetro de *java*.

java -classpath ruta_controlador+archivo_zip_ó_jar;. Inicio

donde

java	Comando o utileria que interpretará el bytecode.
-classpath	Parámetro de la utileria, para indicarle una ruta adicional en donde se encontrarán clases requeridas por el proyecto, en este caso los controladores.
ruta_controlador+ archivo_zip_ó_jar	Ruta en donde están los controladores, en ocasiones las clases, las distribuyen en archivos ZIP o JAR, por lo que cuando se escriba el directorio se deberá de cerciorar, si se tienen alguno de esos archivos compactados, de ser así, se deberá de escribir el nombre completo más su extensión, como parte del classpath especificado ahora en el interprete.
;	En ambiente Windows, se usa punto y coma como separador de las rutas especificadas al classpath, en UNIX es es dos puntos. Y por último agregamos un punto para indicar que en la ruta actual, también se deberán de buscar clases que se llegarán a requerir, en el caso de este proyecto, es necesario para que la clase Inicio, encuentre todas las demás propias al proyecto.
Inicio	Clase principal que se deberá llamar del proyecto.

Ejemplos:

Para el caso de Oracle, en el caso de JDK 1.2 o superior:

```
java -classpath D:\OraHome1\jdbc\lib\classes12.zip;. Inicio
```

Para el caso de MySQL:

```
java -classpath .\mm.mysql.jdbc-1.2c\mysql_comp.jar;.\mm.mysql.jdbc-1.2c\mysql_uncomp.jar;.\mm.mysql.jdbc-1.2c;.
Inicio
```

Si los controladores ya están registrados en la variable de ambiente CLASSPATH, no será necesario adicionarlo como parámetro de java. En éste caso sería:

```
java Inicio
```

Por supuesto que para éste caso, primero se debe de cambiar al directorio en donde estén las clases que componen al proyecto.

Una vez que se haya ejecutado, aparecerá la siguiente imagen:

Figura 9.4

Con lo que sabremos que se ha cargado el proyecto favorablemente, ahora su funcionalidad dependerá de que los controladores se hayan instalado, configurado y registrado en el classpath correctamente.

9.7 Conexión

Después de que se presenta la pantalla de inicio (Figura 9.4), al dar click en ella, aparecerá la siguiente pantalla:

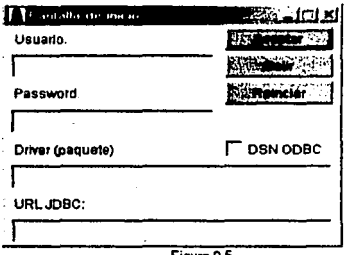


Figura 9.5

En la cual, se deberán de proporcionar los datos necesarios para poderce conectar a la base de datos. Usuario, Contraseña, Driver y URL.

Si se deseara utilizar ODBC, se deberá marcar la casilla *DSN ODBC*, con lo que hará que la pantalla se ajuste y pedirá menos datos, presentandose como sigue:

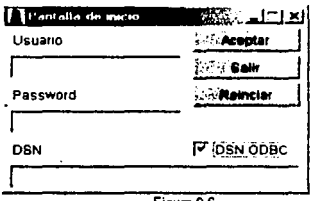


Figura 9.6

En la cual, sólo se deberán de proporcionar los datos: Usuario, Contraseña y DSN. Aún que en la pantalla mostrada en la Figura 9.5, se pueden proporcionar los datos ODBC, como si fuera un controlador JDBC.

Aún que no se proporcionen datos del usuario para la conexión, se puede entrar a la pantalla de trabajo. Esto es sólo con fines de conocer todo el proyecto sin necesitar la base de datos. Otro punto importante es que sólo cuando se ejecuta una sentencia SQL se abre y cierra la conexión a la base de datos, esto es para evitar las conexiones abiertas inecesarias, administración de las licencias de conexión y lo más importante es evitar que el servidor tenga conexiones abiertas que no atenderá.

A continuación se muestrán, los posibles datos, que pueden ser utilizados como ejemplo para conectarse a las bases de datos en que se probo el proyecto.

Oracle

Usuario	Contraseña	Driver	URL
scott	tiger	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@localhost:1521:ifs
system	manager	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@localhost:1521:ifs

donde:

oracle.jdbc.driver.OracleDriver	Siempre es el mismo, para la version Oracle 8i
jdbc:oracle:thin:@	Siempre es el mismo, para la version Oracle 8i
localhost	Nombre o IP donde está corriendo el servidor de Oracle.
1521	Puerto al que esta atendiendo el servidor Oracle, este valor casi siempre es el mismo.
Ifs	Nombre del DSN, registrado en el cliente de Oracle. Vea documentación de Oracle, para mayor detalle.

MySQL

Usuario	Contraseña	Driver	URL
root		org.gjt.mm.mysql.Driver	jdbc:mysql://localhost/dashboard

donde:

donde:

org.gjt.mm.mysql.Driver	Siempre es el mismo, para la version 3.2x
jdbc:mysql://	Siempre es el mismo, para la version 3.2x
localhost	Nombre o IP donde está corriendo el servidor de MySQL.
dashboard	Nombre de la base de datos

Access

Usuario	Contraseña	Driver	URL
admin		sun.jdbc.odbc.JdbcOdbcDriver	jdbc:odbc:enep

donde:

sun.jdbc.odbc.JdbcOdbcDriver	Siempre es el mismo, no importa que controlador de ODBC se utilice.
jdbc:odbc:	Siempre es el mismo, no importa que controlador de ODBC se utilice.
enep	Nombre del DSN que se haya registrado en el Panel de Control / Fuentes ODBC (ambiente Windows)

Access como ODBC

Usuario	Contraseña	DSN
admin		Enep

donde:

enep	Nombre del DSN que se haya registrado en el Panel de Control / Fuentes ODBC (ambiente Windows)

9.8 Conociendo El Ambiente De Trabajo

Después de introducir o no los datos en la figura 9.5 ó 9.6, se pasará de inmediato a la siguiente pantalla:

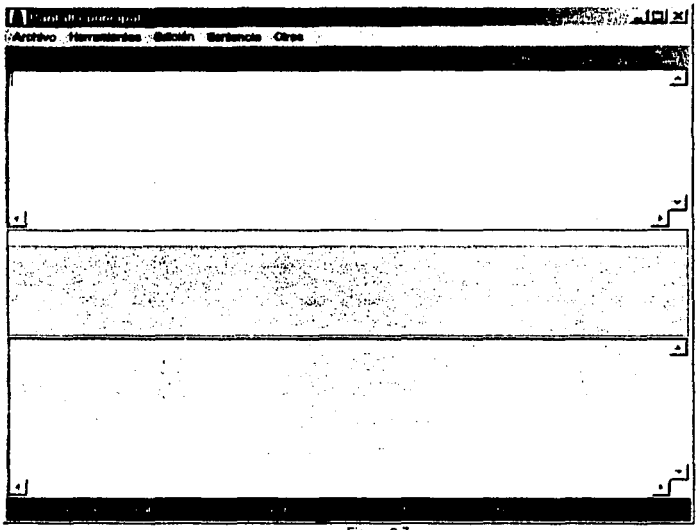


Figura 9.7

En esta se identifican varios elementos importantes:

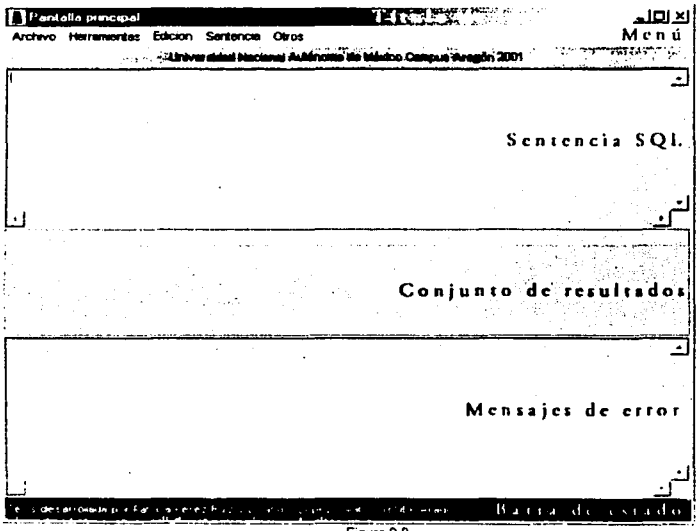


Figura 9.8

donde

Titulo de la pantalla	Titulo identificador de la pantalla actual.
Menú	Elementos disponibles, con el resto de la funcionalidad del proyecto.
Encabezado	Encabezado de la universidad.
Sentencia SQL	Caja de texto en donde se escriben las sentencias SQL a ejecutar.
Conjunto de resultados	Tabla en donde se desplegarán los datos devueltos por la base de datos al ejecutar la sentencia.
Mensajes de error	Caja de textos (deshabilitada) en la que se mostrarán todos los errores que llegarán a ocurrir dentro del proyecto.
Barra de estado	Barra en donde se despegarán algunos mensajes informativos del proyecto.

9.9 Conociendo El Menú

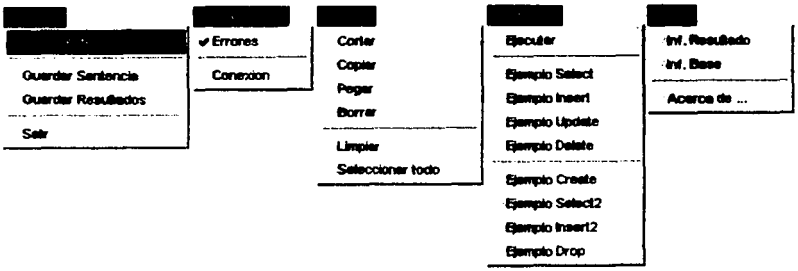


Figura 9.9

donde:

Archivo — Se divide en:
Abrir Sentencia: Módulo con el cual se puede leer un archivo de texto plano, con extensión SQL, que contenga sentencias almacenadas en disco.
Guardar Sentencia: Módulo con el cual se puede escribir un archivo de texto plano, con extensión SQL, que contendrá sentencias previamente utilizadas. Se almacenará el contenido de la caja de texto *Sentencia SQL*, vista anteriormente.
Guardar Resultados: Módulo con el cual se puede escribir un archivo de texto plano, con extensión RPT, que contendrá la sentencia previamente utilizada, más los datos devueltos por la base de datos al ejecutar dicha sentencia. Se almacenará el contenido de la caja de texto *Sentencia SQL* y la tabla *Conjunto de resultados*, vistos anteriormente.
Salir: Módulo con el cual se puede salir del sistema.

Herramentas — Se divide en:
Errores: Opción que se puede marcar o no, si se quiere o no mostrar la caja de texto Mensajes de error, de la figura 9.8
Conexión: Muestra la pantalla mostrada en la figura 9.4 ó 9.5, por si se requiriera conectarse a otra base de datos.

Edición — Todas estas opciones, trabajan con la caja de texto Sentencia SQL, mostrada en la figura 9.8. Se divide en:
Cortar: Opción que copia a memoria y borra los caracteres seleccionados.
Copiar: Opción que copia a memoria los caracteres seleccionados.
Pegar: Opción que pega de memoria caracteres en la posición seleccionada.
Borrar: Opción que borra los caracteres seleccionados.
Limpiar: Opción que limpia toda la caja de texto.
Seleccionar todo: Opción que selecciona todo el contenido de la caja.

Sentencia — Se divide en:
Ejecutar: Opción que ejecuta la sentencia introducida en la caja de texto Sentencia SQL, mostrando el contenido dentro de la tabla Conjunto de resultados, si se llegarán a presentar algúntipo de aviso de la base de datos, se mostrará dentro de la caja de texto Mensajes de error.
Ejemplo Select: Pone una sentencia Select, dentro de la caja de texto Sentencia SQL, como ejemplo, para consultar la base de datos, se ideo, para la base de datos Access distribuida en el CD.
Ejemplo Insert: Pone una sentencia Insert, dentro de la caja de texto Sentencia SQL, como ejemplo, para insertar un registro en la base de datos, se ideo, para la base de datos Access distribuida en el CD.
Ejemplo Update: Pone una sentencia Update, dentro de la caja de texto Sentencia SQL, como ejemplo, para actualizar un registro de la base de datos, se ideo, para la base de datos Access distribuida en el CD.
Ejemplo Delete: Pone una sentencia Delete, dentro de la caja de texto Sentencia SQL, como ejemplo, para un borrar un registro de la base de datos, se ideo, para la base de datos Access distribuida en el CD.
Ejemplo Create: Pone una sentencia Create Table, dentro de la caja de texto Sentencia SQL, como ejemplo, para crear una tabla dentro de la base de datos, se puede ejecutar en la mayoría de bases de datos.

Ejemplo Select2: Pone una sentencia Select, dentro de la caja de texto Sentencia SQL, para consultar la tabla creada con la opción del menú anterior.
Ejemplo Insert2: Pone una sentencia Insert, dentro de la caja de texto Sentencia SQL, para insertar en la tabla creada con la opción del menú **Ejemplo Create**.
Ejemplo Drop: Pone una sentencia Drop Table, dentro de la caja de texto Sentencia SQL, para borrar la tabla creada con la opción del menú **Ejemplo Create**.

Otros — Se divide en:
Inf. Resultado: Opción que muestra dentro de la tabla Conjunto de resultados, información de cada campo, incluido dentro del conjunto de resultados. Mostrando los valores: #, Nombre del Campo, Longitud, Tipo de Dato, Nombre de la Tabla, Autoincrementa, MayMin, Catálogo , Etiqueta, NomTipoDato, Precisión, Escala, EsModena y EsNulo.
Inf. Base: Opción que muestra dentro de la tabla Conjunto de resultados, información de la base de datos consultada. Mostrando los valores: allProceduresAreCallable, allTablesAreSelectable, bestRowSession, columnNullable, dataDefinitionCausesTransactionCommit, getCatalogTerm, getDatabaseProductName, getDatabaseProductVersion, getDefaultTransactionIsolation, getDriverMajorVersion, getDriverMinorVersion, getDriverName","getDriverVersion, getMaxBinaryLiteralLength, getMaxColumnNameLength, getMaxColumnsInGroupBy, getMaxColumnsInOrderBy, getMaxColumnsInSelect, getMaxColumnsInTable, getMaxConnections, getMaxProcedureNameLength, getMaxTableNameLength, getMaxTablesInSelect y getNumericFunctions.
Acerca de: Muestra la pantalla mostrada en la figura 9.4

Resumen

En este capítulo se mestró el uso de JDBC para una aplicación de escritorio, explicando desde la pantalla de inicio (que permite la conexión a la base de datos) hasta la instalación. Así como la estructura del proyecto, para su optimo funcionamiento.

Esta aplicación cubrió los puntos necesarios de JDBC: la conexión, ejecución de sentencias SQL y la lectura del resultado de dicha sentencia.

Dicha aplicación, como se mostró al inicio del capítulo se puede utilizar en diversas plataformas y accesar a varias bases de datos, gracias a Java.

CONCLUSIÓN GENERAL

Como se ha discutido en este documento, JDBC desde que fue liberada por SUN, se ha vuelto el corazón del ambiente de desarrollo Java, para aplicaciones que requieren comunicación con bases de datos.

Esta API, es y será una de las más usadas en el desarrollo en aplicaciones y destaca notablemente toda el potencial que tiene Java ante otros lenguajes.

APÉNDICE I

TIPOS DE DATOS

Tipos De Datos JDBC Y Su Relación Con Tipos Java

JDBC	Java
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	Boolean
TINYINT	Byte
SMALLINT	Short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Tipos Java Y Su Relación Con Tipos Jdbc

Java	JDBC
String	VARCHAR o LONGVARCHAR
java.math.BigDecimal	NUMERIC
Boolean	BIT
Byte	TINYINT
Short	SMALLINT
Int	INTEGER
Long	BIGINT
Float	REAL
Double	DOUBLE
byte[]	VARBINARY o LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Tipos De Datos JDBC Y Su Relación Con Objetos Java

JDBC	Java
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	Boolean
TINYINT	Integer
SMALLINT	Integer
INTEGER	Integer
BIGINT	Long
REAL	Float

FLOAT	Double
DOUBLE	Double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Objetos Java Y Su Relación Con Tipos De Datos JDBC

Java	JDBC
String	VARCHAR o LONGVARCHAR
java.math.BigDecimal	NUMERIC
Boolean	BIT
Integer	INTEGER
Long	BIGINT
Float	REAL
Double	DOUBLE
byte[]	VARBINARY o LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

Objetos Java Y Su Relación Con Tipos De Datos JDBC, Convertidos Por setObject

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
String	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
java.math.BigDecimal	X	X	X	X	X	X	X	X	X	X	X	X	X						
Boolean	X	X	X	X	X	X	X	X	X	X	X	X	X						
Integer	X	X	X	X	X	X	X	X	X	X	X	X	X						
Long	X	X	X	X	X	X	X	X	X	X	X	X	X						
Float	X	X	X	X	X	X	X	X	X	X	X	X	X						
Double	X	X	X	X	X	X	X	X	X	X	X	X	X						
byte[]														X	X	X			
java.sql.Date											X	X	X				X		X
java.sql.Time											X	X	X					X	
java.sql.Timestamp											X	X	X				X	X	X

Tipos De Datos JDBC Obtenidos Por Los Metodos ResultSet.getXXX

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
GetByte	*	X	X	X	X	X	X	X	X	X	X	X	X						
GetShort	X	*	X	X	X	X	X	X	X	X	X	X	X						
GetInt	X	X	*	X	X	X	X	X	X	X	X	X	X						
GetLong	X	X	X	*	X	X	X	X	X	X	X	X	X						
GetFloat	X	X	X	X	*	X	X	X	X	X	X	X	X						
GetDouble	X	X	X	X	X	*	*	X	X	X	X	X	X						
getBigDecimal	X	X	X	X	X	X	X	*	*	X	X	X	X						
GetBoolean	X	X	X	X	X	X	X	X	X	*	X	X	X						
GetString	X	X	X	X	X	X	X	X	X	X	*	*	X	X	X	X	X	X	X
GetBytes														*	*	X			
GetDate											X	X	X				*		X
GetTime											X	X	X					*	X
getTimestamp											X	X	X				X		*
getAsciiStream											X	X	*	X	X	X			
getUnicodeStream											X	X	*	X	X	X			
getBinaryStream														X	X	*			
GetObject	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

En la tabla con X se marcan los métodos que pueden traer el tipo de dato JDBC y con * se marca la recomendación más apropiada.

APÉNDICE II

USO GENERAL DE URLs

En pocas palabras que un URL es el nombre que se le da a un hosts y que através de un protocolo se uso común (como ftp, http y otros) se puede hacer uso de algún servicio existente en ese hosts. Por ejemplo:

ftp://javasoft.com/docs/JDK-1_apidocs.zip
por medio del protocolo *ftp* se esta solicitando el servicio de transferencia de archivos del hosts *javasoft.com*
http://java.sun.com/products/jdk/CurrentRelease
por medio del protocolo *http* se esta solicitando el servicio de lectura de una página de Web del hosts *java.sun.com*
file:/home/haroldw/docs/books/tutorial/summary.html
por medio del protocolo *file* se esta solicitando el servicio de lectura o transferencia de una página de Web del directorio */home/haroldw/docs/books/tutorial/*

URLs de JDBC

Las URLs de JDBC proveen una manera de identificar una base datos para que manejador apropiados los reconozca y establezca una conexión con ella. Los programadores de manejadores son quienes realmente determinan que URL JDBC identificará a su manejador. Los usuarios no necesitan preocuparse sobre como y que forma el URL JDBC. El papel de JDBC es recomendar algunas convenciones para los escritores de manejadores para seguir una estructura de URLs JDBC.

Desde que las URLs JDBC se usan con diversos tipos de manejadores, las convenciones son muy flexibles, Primero, ellas permiten a diferentes manejadores usar diversos esquemas para llamar bases de datos. El subprotocolo ODBC, por ejemplo, puede contener en valores de atributos en la URL (pero no son requeridos).

El segundo lugar, las URLs JDBC permiten a los programadores en manejadores codificado toda la información necesaria de conexión dentro de ellas. Esto hace posible que, por ejemplo, para un *applet* que requirió comunicarse a una base de datos determinada, puede abrir la conexión a dicha base sin requerir que el usuario haga ninguna tarea de administración del sistema.

En tercer lugar, las URLs JDBC permiten un nivel de indirección. Esto significa que JDBC puede referirse al nombre lógico de la base datos o anfitrión que se traduce dinámicamente al nombre real por un sistema nombrador de red. Esto permite a los administradores de sistemas evitar especificar anfitriones articulares como parte del nombre JDBC. Hay varios y diferentes sistemas nombradores de red (tal como DSN, NIS y DCE) y no hay restricción respecto del cual pueda usarse.

De sintaxis estándar para las URLs JDBC es cómo sigue, tiene tres partes, que son separados por dos puntos:

jdbc:<subprotocolo>:<subnombre>

Las tres partes de una URL JDBC son descritas a continuación:

1.- El protocolo JDBC. El protocolo en una URL JDBC es siempre 'jdbc'.
2.- <subprotocolo>. Es el nombre del manejador o nombre del mecanismo de conexión a una base de datos, que puede ser usado por uno o más manejadores. Un ejemplo típico de un nombre que subprotocolo es 'odbc', que se ha reservado para URLs especifican el estilo de nombres ODBC de una fuente de datos (DSN, data source name). Por ejemplo, para acceaar una base de datos mediante un puente JDBC-ODBC. uno podria usar una URL tal como el siguiente:
jdbc:odbc:fred
En éste ejemplo, el subprotocolo es "odbc" y el subnombre "fred" es una fuente local de datos que ODBC
Si uno quiero usar un sistema nombrador de red (para que el nombre de la base de que datos en la URL JDBC no tenga que ser un nombre actual), el sistema nombrador puede ser el subprotocolo. Por ejemplo, uno podria tener un URL parecido a.
jdbc:nombramiento-dce:conteo-ejecutarle
En este ejemplo, el URL especifica que el DCE local como sistema nombrador de red deberia encontrar la base de datos nombraba "conteo-ejecutarle" dentro de un nombre más especifico que puede usarse para usarse para conectarse a la base de datos verdadera.
3.- <subnombre> - una manera para identificar la base de datos. El <subnombre> puede vanar, dependiendo del subprotocolo y puede tener un subsubnombre con cualquier sintaxis interna propia del creador del manejador. La intención de un subnombre es dar información suficiente para ubicar la base de datos. En el ejemplo anterior "fred" es suficiente por que el ODBC provee el resto del información. Sin embargo, una base que datos sobre un servidor remoto requieren más información. Si la base de datos esta para ser consultado sobre Internet, por ejemplo, la dirección en la red deberá incluirse en la URL JDBC como partes del subnombre y deberá ser antecedida por el URL estándar que nombre la convención, el subnombre puede ser
//computadora:puerto/subsubnombre
Por ejemplo, si suponemos que "dbnet" es un protocolo red conectarse a un anfitrión sobre Internet, una URL JDBC podria ser como
jdbc:dbnet://wombat:356/fred

El Subprotocolo ODBC

El subprotocolo ODBC es un caso especial. Se ha reservado para URLs que especifica un estilo ODBC para nombres de fuente de datos y tienen la característica especial de permitir cualquier número

de valores de atributos a ser especificados después del subnombre (el nombre de la fuente de datos). La sintaxis completa para el subprotocolo ODBC es:

```
jdbc:odbc:<nombre-fuente-datos>[;<nombre-datos>=<nombre-datos>]
```

Todos estos son nombres correctos de URLs:

```
jdbc:odbc:qeor7
jdbc:odbc:wombat
jdbc:odbc:wombat;CacheSize=20;ExtensionCase=LOWER
jdbc:odbc:qeora;UID=kgh;PWD=fooey
```

Registro de Subprotocolos

Un desarrollador de manejadores puede reservar un nombre para ser usado como el subprotocolo en una URL JDBC. Cuando la clase *DriverManager* incluye este nombre de su lista de manejadores registrados, el nombre del manejadores se reserva para realizar una conexión a la base de datos identificada. Por ejemplo, 'odbc' se reserva para el puente JDBC-ODBC. Sin empresa quisiera reservar su subprotocolo, por ejemplo, Miracle Corporation, puede registrar "miracle" como su subprotocolo de su manejador JDBC que conecta a su DBMS Milagro para que ninguna empresa use ese nombre.

JavaSoft actúa como un registro de información para nombres de subprotocolos JDBC. Para registrar un nombre de subprotocolo, puede enviar un e-mail a:

```
jdbc@wombat.eng.sun.com
```

APÉNDICE III

CONTENIDO DEL CD

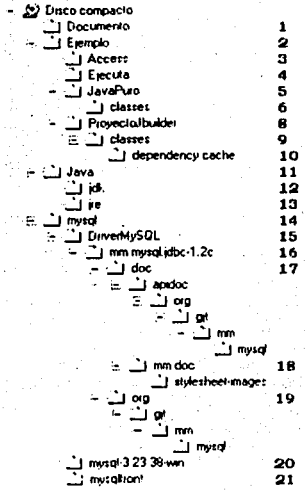


Figura 5.1

donde:

1	Contiene, el documento de tesis, más sus archivos que lo conforman.
2 – 10	Explicados en la figura 9.2
11	Contiene utilerías Java
12	Ambiente de desarrollo Java 1.2 y 1.3 para Windows y Linux, además de documentos propia del JDK
13	Ambiente de ejecución Java 1.3 para Windows
14	Contiene utilerías de la base de datos libre (freeware) para probar el proyecto. Importante: contiene el archivo start_mysql.bat, que fue creado por nosotros donde se muestra el comando para iniciar la base, si se tuviera problemas con ella
15	Controlador 100% JDBC, que nos permite conectar MySQL con el proyecto
16	Directorio base de donde se buscará que el paquete del controlador exista
17	Documentación del controlador
18	Documentación adicional
19	Directorio principal de la estructura del paquete del controlador MySQL
20	Contiene la versión 3.23 de MySQL para windows
21	Utilería, similar a éste proyecto, pero que interactua con MySQL (freeware)

BIBLIOGRAFIA

- Ashton Hobbs, **Aprendiendo programación para Bases de Datos con JDBC**, Prentice Hall.
- Alan Williamson, **Java Database Proggramming**, Prentice Hall.
- Martyl lall, **Servlets y JavaServer Pages**, Prenticel lall.
- Paul S. Wang, **Java con Programación orientada a objetos**, International Thomson Editores.
- Agustin Froute, **Java 2** Segunda Edición
- Laurence Vanhelsuwe, **La biblia de Java**, Anaya Multimedia.
- William Weinman, **El libro del CGI**, Prentice Hall.
- Michael M. Gorman, **Database Management Systems**,QED.