24

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES "ARAGON"

CONEXION DE JAVA A BASES DE DATOS RELACIONALES UTILIZANDO ODBC Y JDBC.

T E S I S
QUE PARA OBTENER EL TITULO DE:
INGENIERO EN COMPUTACION

P R E S E N T A :
PEDRO ISRAEL MONTAÑO GONZALEZ

ASESOR: ING. AMILCAR MONTERROSA ESCOBAR

MEXICO **2000**

Dedico este trabajo a Dios primeramente por haberme dado a mis padres y hermanos. Gracias a mis padres por mostrarme el camino recto y a mis hermanos por su confianza y amistad.

Pedro Israel Montaño González.

ÍNDICE

Introducción

En un periodo relativamente corto, el web se ha diversificado para proveer servicios que antes existían únicamente en los sueños. Una de las herramientas básicas que proveen de gran poder al web es el lenguaje Java. Java ha tenido gran popularidad debido a la incorporación de applets dentro de páginas HTML. Java a pesar de su fama propiciaba que se utilizara como un lenguaje para crear animaciones, sonidos en fin hacer de una página HTML algo más agradable. En sus inicios Java carecía de clases que permitieran enlazar una página HTML a una base de datos, y esto limitaba en gran manera el uso de Java para crear verdaderas aplicaciones cliente-servidor. JDBC surgió como una alternativa bastante buena para establecer este concepto. JDBC forma parte de las nuevas características de Java que aumentan su poder.

El presente trabajó demuestra las nuevas capacidades de Java para conectarse a bases de datos relacionales, tales como Oracle, Sybase, Informix, etc. o cualquier otra base de datos que cuente con drivers JDBC u ODBC. JDBC incorpora el esfuerzo de diversas compañías lidereadas por Sun MicroSystems con el fin de proponer un posible estándar de conexión entre Java y las bases de datos relacionales. JDBC es una nueva tecnología incorporada a Java que incluye un conjunto de clases que hacen posible el manejo de plataformas cliente-servidor en una intranet o bien en internet.

El objetivo que de este trabajo es crear una aplicación multicapas desarrollada puramente en Java. Mostrando de alguna manera la tecnología JDBC y a la vez las habilidades y ventajas que presenta Java sobre otros lenguajes; tales como la

serialización de objetos, la habilidad de ser multiplataforma y la capacidad de crear applets y aplicaciones.

El trabajo se divide en cuatro capítulos que tratan los conceptos fundamentales de bases de datos relacionales, un vistazo general al lenguaje Java, la relación entre Java y una base de datos. Además en el último capítulo se diseña una aplicación que hace uso de la tecnología JDBC, para vincular una página HTML a una base de datos. Este trabajo expone conceptos de sistemas cliente-servidor de varias capas, derivado de la red Internet que evoca este concepto. El esfuerzo realizado para elaborar este trabajo presenta de una forma muy sencilla, pero a la vez directa la capacidad y ventajas que presenta el lenguaje Java para desarrollar aplicaciones cliente-servidor reales.

Son muy diversos los usos que pueden darse utilizando esta tecnología, las aplicaciones van desde una simple consulta hasta aplicaciones complejas de comercio electrónico.

Conceptos globales de Bases de Datos

Actualmente el modelo de bases de datos utilizado con mayor frecuencia es el relacional. El modelo relacional de las bases de datos presenta la información como un conjunto de entidades. La entidad puede ser cualquier elemento que agrupe un conjunto de datos, como en el caso de los departamentos de una compañía o de sus empleados. Así dentro de la entidad de empleados de la compañía cada empleado es una unidad individual y no existen dos de la misma clase, de esta manera las entidades se modelan dentro de la base de datos como tablas. Una tabla es un grupo lógico de información similar. Las entidades pueden tener relaciones entre sí, las cuales se definen por medio de llaves (primarias y foráneas). No corresponde a este trabajo el efectuar un análisis exhaustivo sobre el modelo entidad-relación por lo que no se estudiarán los tipos de relaciones que existen entre las entidades. Otra característica de las entidades es que éstas poseen atributos, que se modelan en una tabla a manera de columnas; Retomando el ejemplo concerniente a la compañía, podría tener atributos tales como: nombre del empleado, puesto que ocupa, extensión telefónica, etc..

En una base de datos, los renglones de una tabla se asemejan a los registros de un archivo de datos plano. La misma semejanza se presenta en las columnas de cada tabla y en los campos del archivo de datos plano. Debido a esta semejanza se utilizarán los términos renglón y registro de manera indistinta al igual que los términos columna y campo.

Antes de intentar hacer una conexión a una base de datos hay que tener en cuenta dos puntos fundamentales: la manera en que ésta engloba la información y la

terminología básica de una base de datos relacional. Por este motivo, se pueden definir algunos de los términos utilizados más comúnmente:

- Renglones (Registros)

Un registro mantiene información de un elemento único dentro de la tabla. Generalmente lo que se busca es no tener registros repetidos.

- Columnas (Campos)

Cada columna dentro de cada tabla contiene una parte de la información global del registro.

- Tablas

Una tabla es un conjunto de renglones y columnas semejantes a una matriz. Una tabla representa una entidad y por ende comprende renglones que contengan información agrupada con una estructura común.

- Llaves

Una llave esta formada por una o varias columnas de la cada tabla. Una llave puede ser única o no única dependiendo si se desea o no la repetición de registros con esa misma llave. Una llave única puede asignarse como llave primaria lo cual identificaría a cada renglón de la base de datos, lo que como consecuencia no permitirá que existan renglones iguales. Existen también llaves foráneas que permiten que exista una liga o relación entre las tablas de la base de datos.

- Índices

Los índices son listas ordenas de ciertos campos de una tabla. Las dos causas que ameritan la creación de un índice para una determinada tabla son: el no permitir la repetición de columnas y el agilizar el proceso de búsqueda de registros dentro de la tabla. Los índices pueden estar formados por una única columna o bien por la unión de dos o más de ellas. Cada tabla puede tener varios índices con el fin de optimizar el tiempo de búsqueda. Es importante saber qué índices crear en cada tabla, ya que si no se usan de la manera adecuada incluso hasta podrían retrasar el tiempo de búsqueda. Si un índice se crea pero nunca o casi nunca se utiliza entonces creará retrasos cuando se genere una transacción, ya que al modificar algún renglón que afecte la(s) columna(s) del índice este se actualizará conjuntamente con el renglón modificado.

- Constraints (Restricciones)

Estas son restricciones aplicables a una tabla para establecer cierto grado de integridad de datos; como ejemplo de un constraint están las llaves primarias.

- Vistas

Las vistas constituyen una manera de ver los datos almacenados de una o más tablas. Mediante una vista el usuario es capaz de ver determinadas partes de la información global de una o mas tablas. Las vistas son muy utilizadas para ver información derivada o calculada de una o mas tablas. Las vistas simplifican el extraer información de una consulta muy utilizada, es decir, en lugar de efectuar esa consulta una y otra vez ésta se integra como una vista. Por otra parte aunque las vistas simplifican las consultas a la base de datos, en muchos casos no es recomendable hacer uso excesivo de estas debido a que retrasan la ejecución de consultas que se basen en ellas.

- Stored Procedures (Procedimientos almacenados)

Un procedimiento almacenado es una serie de comandos SQL previamente compilados[1] en el servidor de la base de datos. Estos procedimientos se compilan únicamente la primera vez que son ejecutados; la versión compilada de cada procedimiento se guarda en la base de datos y se puede ejecutar en futuras ocasiones con la diferencia de que al ya estar compilado se ejecutará con mayor rapidez. Cada stored procedure se ejecuta llamándolo desde una aplicación o por un usuario. Algunas de las ventajas obtenidas al crear rutinas y guardarlas como procedimientos almacenados son una mayor rapidez en la ejecución de los comandos SQL, mejor desempeño, ahorro de memoria y por supuesto mayor seguridad.

Los procedimientos almacenados agregan cierto grado de programación a la base de datos, ya que no solo incluyen comandos SQL sino también estructuras de control, variables, programación por bloques de código, etc. Es posible crear procedimientos almacenados que incluyan parámetros de entrada y salida, lo cual contribuye a una mayor flexibilidad por parte de la base de datos.

- Triggers (disparadores)

Un trigger es un caso especial de un procedimiento almacenado (stored procedure) ya que éstos se disparan por el surgimiento de un evento. Los eventos que ocasionan que un trigger se ejecute son el insertar, actualizar o eliminar registros de una tabla. En pocas palabras un trigger se ejecuta de manera automática. Al ocurrir un evento (es decir una acción que modifique alguna tabla) el trigger tiene la facultad de

[1] *El término compilar dentro del mundo de las bases de datos significa que el manejador de la basede datos determina el plan de ejecucion del procedimiento almacenado del que se trate.*

ejecutarse antes o después de que la acción que lo disparó se lleve a cabo. Por ejemplo si se insertaran registros nuevos, estos podrían ser validados por medio de un trigger con el fin de insertar registros que no violen la integridad de la base de datos. Uno de los muchos usos que se le dan a los triggers es el auditar modificaciones hechas a la información.

- Cursores

Un cursor es un área de trabajo empleada por la base de datos a la cual se puede accesar proporcionándole un nombre. Dicha área de trabajo contiene registros resultantes de haber ejecutado una cláusula *select*. Existen dos tipos de cursores, los implícitos y los explícitos. Los primeros son creados por la base de datos cada vez que se ejecuta una sentencia SQL; mientras que los explícitos son utilizados comúnmente por el usuario para almacenar más de un registro y acceder a cada uno de ellos uno a uno. Estos cursores explícitos , por lo regular, suelen utilizarse dentro de stored procedures o triggers y así establecer rutinas que procesen cada uno de los registros localizados en el cursor.

- Paquetes (Packages)

En algunas bases de datos, como en el caso de Oracle, existen objetos llamados *packages*. Los packages agrupan un conjunto de programas de la base de datos en una misma estructura y con una interfaz bien definida (encabezado del package). Es decir, que en lugar de crear procedimientos almacenados (stored procedures) e introducirlos como tales dentro de la base de datos, es posible agruparlos y almacenarlos a manera de paquetes. Una vez que se han establecido los paquetes la modularidad y simplicidad se acentúan y se contribuye a un mantenimiento más sencillo de los mismos.

Para poder obtener, insertar, modificar y eliminar información de una tabla se requiere de un método que permita la interacción con los datos, por lo que se considera el uso de SQL (Sequel)[2] como adecuado para esta función.

El lenguaje SQL

SQL es el lenguaje globalmente aceptado para el manejo de bases de datos relacionales. Haciendo referencia a los orígenes de las bases de datos relacionales, éstas encuentran sus inicios con el Dr. Codd en el año de 1970, quien publicó un documento enfocado al modelo relacional sobre los bancos de datos. Actualmente el modelo de Codd es aceptado como el modelo definitivo para los sistemas de bases de datos relacionales. Con respecto a SQL, éste fue implementado en un principio por IBM, pero fue Oracle, la compañía que introdujo la primer versión comercial de SQL.

SQL es un lenguaje muy poderoso dentro del mundo de las bases de datos. Al hacer uso de SQL los administradores de bases de datos, programadores y usuarios finales obtienen beneficios tales como los siguientes:

- SQL permite el manejo de grupos de registros, es decir que puede trabajar con un conjunto de registros en lugar de manipularlos uno a uno.

[2] *Las siglas SQL significan Structured Query Language o en español Lenguaje de consultas estructurado. SQL comúnmente se pronuncia como Sequel y no como SQL.*

- SQL es utilizado por todo tipo de usuarios sin importar la actividad que éstos desempeñen. Por lo tanto, constituye una forma común de tener acceso a la información.

- SQL provee una serie de comandos de gran utilidad para realizar distintas tareas como: la consulta, la actualización, la inserción y la eliminación de información, además de garantizar consistencia dentro de la base de datos.

- Una vez que se conoce no importa que manejador relacional de bases de datos se utilice, ya que SQL es un estándar dentro de las bases de datos relacionales y tiene soporte dentro de la mayoría de estas. Cabe mencionar que sí existen diferencias entre un RDBMS[3] y otro, pero la parte medular de SQL y sus estándares son los mismos (SQL ANSI[4]).

SQL puede variar en algunas de sus instrucciones según el RDBMS con el cual se esté trabajando, pero siempre se mantiene el estándar ANSI.

En la mayoría de los sistemas que emplean SQL estos dividen el lenguaje de comandos de definición de datos (DDL[5]) y el lenguaje de comandos de manipulación de datos (DML[6]). El lenguaje de comandos de definición de datos permite que el usuario

[3] *RDBMS Relational Database Manager System. Manejador de sistema de base de datos relacional.*

[4] *ANSI es la asociacion que controla estándares dentro de varias plataformas en este caso regula un estandar para SQL.*

[5] *DDL o Data Definition Language*

[6] *DML o Data Management Language*

ejecute acciones como crear, modificar y eliminar objetos de la base de datos, otorgar y quitar privilegios, analizar información de un índice o tabla, así como también agregar comentarios al diccionario de datos. Por su parte el lenguaje de comandos de manipulación de datos sirve para consultar y manipular información de las tablas. Además de los lenguajes de definición de datos y de manipulación de datos algunos manejadores de bases de datos definen un tercer conjunto de comandos conocidos como comandos de control de transacciones. Dichos comandos sirven para controlar todos los cambios hechos a la base de datos mediante el uso del lenguaje de manipulación de datos.

Comandos del lenguaje de definición de datos. (DDL)

Algunos de los comandos del lenguaje de definición de datos requieren de un uso exclusivo de los objetos de la base de datos con los que van a operar. En este tipo de comandos se incluyen aquellos que crean, modifican o eliminan objetos de la base de datos. Para ilustrar esto último podemos mencionar que en caso de que se quisiera eliminar alguna de las tablas de la base de datos se utilizara el comando DROP TABLE, pero si algún otro usuario tiene pendiente una transacción en ese momento con esa tabla entonces el comando no se ejecutaría con éxito.

Comandos del lenguaje de manipulación de datos (DML).

Debido a que SQL permite manipular la información mediante grupos de registros es que es posible utilizar instrucciones que además de ser capaces de leer y/u obtener

uno a uno cada registro que el usuario solicite, sean capaces de insertar, actualizar, eliminar y obtener un conjunto o grupo de registros conocidos como Resultsets[7].

Las instrucciones con mayor uso en SQL son las que se requieren para insertar, actualizar, eliminar y obtener información de las tablas de una base de datos. Existe una diferencia radical que separa éstas instrucciones en dos grupos; el primero constituido por instrucciones que alteran la información entre ellas se encuentran INSERT, UPDATE y DELETE. El segundo grupo emplea la instrucción SELECT para obtener información de la base de datos. Por ser éstas las instrucciones más empleadas en SQL a continuación presento una explicación mas detallada de cada una de ellas.

- Comando SELECT

El comando SELECT es sin duda el que mayor uso recibe para obtener información de objetos de la base de datos tales como tablas y vistas. Este comando involucra tres cláusulas principales: SELECT, FROM y WHERE. La cláusula SELECT se utiliza seguida de los nombres de las columnas de la o las tablas a partir de las cuales se obtendrá información. La cláusula FROM seguida del o de los nombres de tablas (o vistas) indica cuales son las tablas (o vistas) de los que se extraerán datos; de esta manera finalmente la cláusula WHERE define las condiciones de búsqueda deseadas. Además existen otras cláusulas de las que se tratarán más adelante. La sintaxis básica de un comando *Select* es la siguiente:

[7] *Un Resultset es semejante a una matriz formada por filas y columnas con informacion obtenida de la base de datos.*

SELECT (ALL / DISTINCT) *lista_de_columnas*

FROM *lista_de_tablas o vistas*

WHERE *restricciones*

(GROUP BY)

(HAVING)

(ORDER BY)

Un ejemplo sencillo de una sentencia que incluya un comando *Select* puede ser el que se detalla a continuación (ejemplo 1). Es importante saber que el símbolo "*" (asterisco) le indica a la base de datos que incluya todas las columnas de la tabla que se encuentre en la cláusula *from*. Para este caso específico se da la instrucción de que regrese todos los renglones y todas las columnas de la tabla Contratos.

```
Select * from Contratos (ejemplo 1)
```

```
Select Numero_Contrato, Numero_Subcontrato, Descripcion
from Contratos
where Numero_Subcontrato = 2  (ejemplo 2)
```

En caso de desear únicamente obtener información de ciertas columnas, éstas se declaran inmediatamente después de la cláusula *Select* como en el ejemplo 2. En este ejemplo aparte de indicar qué columnas deberán presentarse, la cláusula *where* permite restringir el resultado obtenido solamente a los renglones cuyo Numero_Subcontrato sea dos.

Dentro de la cláusula *where* no sólo es posible utilizar el operador de igualdad (=), sino también aquellos operadores que la mayoría de los RDBMS aceptan y que se pueden resumir de la siguiente forma:

Operadores de comparación	= > < >= <= <>
Operadores de rangos	between
Operadores para trabajar con lista	in
Operadores para buscar cadenas	like
Operadores para verificar datos nulos	is null, is not null
Operadores para combinar varios criterios de búsqueda	and, or
Operadores de negación	not

La cláusula *where* también se utiliza para obtener un resultado que combine columnas de una o más tablas, como se ilustra a continuación:

```
Select Contratos.Numero_Subcontrato, Negocio.Descripcion
from Contratos, Negocio
where Contratos.Numero_contrato = Negocio.Numero_contrato
```

En este ejemplo se obtiene un resultado combinado de las tablas Contratos y Negocio, en el cual se incluyen el numero_subcontrato proveniente de la tabla de contratos y la descripción proveniente de la tabla de Negocio. Con la condición de que para cada numero_contrato que exista en la tabla Contratos exista ese mismo Numero_contrato en la tabla de Negocio. Este uso de la cláusula *where* se conoce como un *join*.

Cláusula *Distinct*

La cláusula *distinct* se antepone a la lista de columnas que se desean extraer a partir de la base de datos, y si llegará a existir repetición de renglones en el resultado devuelto por la sentencia *select*, únicamente se regresará un renglón por cada grupo de renglones que tuvieran los mismos datos. Es decir, que la cláusula *distinct* regresará renglones únicos aunque en realidad pueden o no estar repetidos. La siguiente sentencia muestra un uso simple de la cláusula *distinct*, que consiste en extraer un número de subcontrato único de la tabla Contratos sin importar cuantos registros tengan el mismo numero de subcontrato.

```
Select distinct Numero_Subcontrato
From Contratos
```

Cláusula *Order by*

La cláusula *order by* se aplica cuando se desea regresar de manera ordenada, de acuerdo a alguna(s) columna(s) en particular, aquellos registros devueltos por una sentencia *select*. Para activar la cláusula *order by* hay que incluirla dentro de la sentencia del comando *select* e inmediatamente después indicar cuales son las columnas que se tomarán como base para llevar a cabo la ordenación de los renglones, por ejemplo si de la tabla de Contratos se tuvieran que extraer los contratos ordenados de acuerdo al número de contrato la sentencia *select* sería:

```
Select *
From Contratos
order by Numero_Contrato
```

Cláusulas *Group by* y *Having by*

Las cláusulas *group by* y *having* permiten organizar los renglones devueltos por una sentencia *select* de manera que se pueda resumir la información devuelta. *Group by* resumirá las columnas indicadas, mientras que la cláusula *Having by* actuará de manera similar a la cláusula *where*, pero de manera grupal o resumida. Una aplicación de la cláusula *group by* podría ser el mostrar subtotales de los registros de la tabla Contratos según el campo cantidad y agrupando esos subtotales por el numero_contrato; entonces el query que se requiere es el siguiente:

```
Select Numero_Contrato, sum(cantidad)
From Contratos
Group by Numero_Contrato
```

Ahora bien si de los subtotales obtenidos por la sentencia anterior se requiriera mostrar únicamente aquellos que tuvieran una suma mayor a 10 se aplicaría la cláusula *Having*, por lo que quedaría:

```
Select Numero_Contrato, sum(cantidad) Cant
From Contratos
Group by Numero_Contrato
```

```
Having by Cant > 10
```

El comando *select* también permite la anidación de queries (un query es una sentencia SQL). La creación de queries anidados son llamados subqueries y son de gran utilidad para desarrollar una gran cantidad de sentencias sql. Un subquery no sólo puede aplicarse a una sentencia con un comando *select* sino también a una sentencia con un comando Insert, Update o Delete. El siguiente ejemplo muestra como se utiliza un subquery para extraer el numero_contrato y la descripción de todos los contratos siempre y cuando la cantidad que tengan sea menor a la cantidad que tiene el numero_contrato uno.

```
Select Numero_Contrato, descripcion
        from Contratos where Cantidad < (Select Cantidad from Contratos
        where Numero_Contrato = 1)
```

En el ejemplo anterior primero se evalúa el subquery (el query interno que pregunta por la cantidad del numero_contrato 1), y una vez que se regresa su resultado el motor de la base de datos evalúa el otro query.

El operador UNION

Existen ocasiones en las que es necesario extraer datos de dos o más tablas, pero no se requiere crear un join entre estas tablas; sino más bien extraer columnas semejantes de cada una las tablas. Por ejemplo si además de la tabla de Contratos hubiera una tabla llamada Contratos_Anteriores y fuera necesario obtener los contratos

de ambas tablas, entonces se procedería a crear una sentencia *select* para cada tabla y unir los resultados arrojados por cada una de las tablas. Esto se logra a través del operador *union*. Para ilustrar el funcionamiento del operador *union* se presenta el siguiente ejemplo:

```
Select Numero_Contrato, Descripcion
from Contratos
Union
Select Numero_Contrato, Descripcion
from Contratos_Anteriores
```

Este ejemplo arroja todos los contratos que se encuentren en las dos tablas. El operador *union* procede a eliminar renglones duplicados, pero si se necesitará desplegar absolutamente todos los renglones regresados entonces se agrega el operador *all*. Para poder usar el operador *union* las columnas regresadas por cada sentencia *select* involucrada deben ser similares.

Hay que aclarar que el operador *union* no es lo mismo, ni siquiera semejante a lo que se hace con un join; ya que aunque ambos involucren datos de dos o más tablas la forma de mostrarlos no es la misma. Un join extrae datos de una y otra tabla relacionando columnas que funcionan como ligas entre las tablas. El operador *union* no utiliza elementos que funcionan como ligas entre las tablas que muestra, sino más bien presenta las columnas de las tablas involucradas en el mismo orden y que sean del mismo tipo. En resumen se puede decir que un join hace que el resultado obtenido crezca de manera horizontal y el operador union aumenta el resultado obtenido pero de manera vertical.

Comando INSERT

INSERT es el comando que permite agregar renglones a una tabla. Éste agrupa la cláusula *INSERT INTO* y la cláusula *VALUES*. La primera hace referencia a la tabla o vista en la cual se insertará el nuevo renglón. Además puede contener una lista de las columnas en las que debe insertarse información. La segunda cláusula (*values*) define la información que se agregará. La sintaxis es la siguiente:

INSERT INTO *nombre_de_la_tabla o vista lista_de_columnas*
VALUES *lista_de_valores*

Ejemplo:

Este ejemplo agrega una nueva columna en la tabla Contratos con los siguientes valores: Numero_Contrato = 15, Numero_Subcontrato = 2 y Descripcion = Fideicomiso, Cantidad = 10.

Insert into Contratos (Numero_Contrato, Numero_Subcontrato, Descripcion, Cantidad) values (15, 2, 'Fideicomiso',10)

El siguiente ejemplo añade el mismo registro pero la diferencia es que no se indica el nombre de las columnas en donde debe agregarse cada valor por lo que se entiende que se agregarán todas las columnas para este renglón.

```
Insert into Contratos values (15, 2, 'Fideicomiso',10)
```

El comando INSERT puede funcionar no sólo con la cláusula *Values* sino que puede insertar los renglones y columnas resultantes de haber efectuado un comando SELECT. Es importante que el resultado regresado por el comando SELECT sea compatible con la tabla en la cual se desea agregar nuevos registros. La compatibilidad entre el resultado regresado por el comando SELECT y la tabla en la que se insertarán los nuevos registros está dada por el número de columnas, el orden y tipo de datos que cada una de ellas manejen. En el siguiente ejemplo se insertarán en la tabla Contratos todos los registros provenientes de la tabla Negocio.

```
Insert into Contratos
        Select Numero_Contrato, Numero_Subcontrato, Descripcion,
        Cantidad
        From Negocio
```

Una diferencia notoria entre el usar las cláusulas *Insert - Values* y la cláusula *Insert Select* es la cantidad de renglones que pueden insertarse en una tabla. La cláusula *Insert - Values* únicamente puede insertar un renglón mientras que la cláusula *Insert Select* puede insertar ninguno, uno o más de un renglón, esto dependerá del número de renglones que el comando *Select* regrese.

Comando UPDATE

UPDATE modifica la información que ya existe en renglones de la tabla a la que hace referencia. El comando *update* utiliza las cláusulas *SET* y *WHERE*. En donde el uso de la cláusula *where* es opcional. La cláusula *set* indica qué columnas son las que se actualizarán, y la cláusula *where* indica que renglones de la tabla se actualizarán. En caso de no indicar una cláusula *where*, se actualizarán todos los renglones de la tabla definida por el comando *update*. La sintaxis es la que se muestra a continuación:

UPDATE *nombre_de_la_tabla o vista*
SET *lista_de_columnas, lista_de_variables o datos*
WHERE *restricciones*

Ejemplo:

Aquí se modificará el valor de la columna Descripción por Nuevo Fideicomiso, siempre y cuando el valor que contenga sea Fideicomiso.

```
Update Contratos
Set Descripcion = 'Nuevo Fideicomiso'
Where Descripcion = 'Fideicomiso'
```

De manera análoga al comando *Insert*, el comando *update* puede llevar a cabo una transacción en base al resultado obtenido por un comando *Select*. El siguiente ejemplo ilustra esta situación.

```
Update Contratos
Set Cantidad = (Select sum(Cantidad)
From Totales)
Where Numero_Contrato = 15
```

En este ejemplo se actualiza el renglón con el número de contrato 15 en la columna cantidad con la suma del campo cantidad de la tabla Totales.

Comando DELETE

El comando *DELETE* permite la eliminación de uno o más renglones de una tabla. Las cláusulas que intervienen son *DELETE* y *WHERE*. La sintaxis del comando es:

```
DELETE nombre_de_la_tabla
WHERE restricciones
```

En el siguiente ejemplo se eliminarán todos los renglones de la tabla Contratos siempre y cuando la descripción sea Fideicomiso.

```
Delete Contratos
Where Descripcion = 'Fideicomiso'
```

En caso de no incluir la cláusula *where* con sus respectivas restricciones después del comando *delete* se eliminan todos los renglones de la tabla. De manera similar a como se introduce una cláusula *select* en un comando *insert* o *update*, se hace con un

comando *delete*. Es decir que se puede utilizar una cláusula *select* después del comando *delete* para eliminar registros en base a los resultados regresados por este comando *select*. Así tenemos el siguiente ejemplo:

```
Delete Contratos
where Numero_Contrato in (Select Numero_Contrato From Negocio)
```

En el ejemplo anterior se eliminaran todos los renglones de la tabla Contratos siempre que el Numero_contrato de estos se encuentre en la tabla Negocio.

Funciones de SQL

El lenguaje SQL cuenta con funciones para manejo de cadenas, números, etc., pero muchas de ellas varían de acuerdo al RDBMS que se esté utilizando. Una función de SQL es parecida a un operador ya que manipula ciertos datos y regresa un resultado, pero difiere de un operador en el formato que presenta. El formato que presentan las funciones les permite trabajar con ninguno o con varios argumentos. Las funciones SQL pueden ser de dos tipos: las que trabajan sobre un renglón o dato único y las que trabajan sobre un grupo de registros. Por ejemplo la función AVG obtiene el promedio de un grupo de renglones por lo que se dice que trabaja sobre un grupo de registros.

Además de tener funciones implícitas dentro de la base de datos, existen manejadores de base de datos que permiten al programador desarrollar sus propias funciones.

Comandos de control de transacciones

Una transacción de base de datos es una unidad lógica de trabajo formada por una o más sentencias SQL que son ejecutadas por el mismo usuario. Un ejemplo típico de una transacción es el traspasar fondos monetarios de una cuenta bancaria a otra. Para llevar a cabo esta transacción es necesario eliminar los fondos monetarios de una cuenta y agregar estos fondos monetarios a la otra cuenta. Es posible que se ejecute la primer sentencia de SQL que consiste en eliminar fondos de la primer cuenta, pero al querer agregarlos en la segunda cuenta no sea posible por una u otra circunstancia. Esto sería totalmente incorrecto, ya que se eliminaron fondos monetarios pero nunca se agregaron a la segunda cuenta. De ahí el objetivo de agrupar las dos sentencias SQL como una sola transacción. Al agrupar las sentencias SQL como transacciones se adquieren ventajas tales como:

- Si una sentencia SQL falla es posible revertir todo el proceso.
- Permite conservar la integridad de la base de datos sin dejar la información de manera inestable o errónea.

Los comandos más comunes que manejan las transacciones dentro de la base de datos son el comando "commit" y el comando "rollback". El propósito que persigue el comando commit es el hacer permanentes las modificaciones hechas a la base de datos desde que se comenzó la transacción. Por su parte el comando rollback desactiva los cambios hechos a la base de datos. Para comprender como funcionan un comando commit y un comando rollback se debe conocer la manera en que trabaja la base de datos con las transacciones. Cuando se ejecuta una transacción en la base de datos,

como puede ser una actualización a la información de una tabla, esta transacción se almacena en un *buffer* previamente definido por la base de datos y esta actualización no se almacena de manera permanente, sino hasta que se utiliza el comando commit. Una vez que se ha introducido el comando commit ya no es posible deshacer la o las transacciones hechas hasta ese punto. Por el contrario si la transacción todavía no se hace permanente y se encuentra todavía en el *buffer* es posible deshacer la o las transacciones con el comando rollback; lo cual regresaría la base de datos a su estado inicial antes de haber iniciado las transacciones.

Las transacciones únicamente funcionan con comandos del lenguaje de manipulación de datos. Es decir, mediante comandos como *SELECT, UPDATE,* etc.. En las aplicaciones de negocios actuales resulta muy práctico efectuar transacciones, ya que, evitan muchas de las inconsistencias en la base de datos.

Fundamentos básicos del lenguaje Java

Características generales de Java

El objetivo de este capítulo no es explicar o ser un tutorial del lenguaje de programación Java, sino únicamente mencionar los conceptos fundamentales de Java para aplicarlos posteriormente a las conexiones de este lenguaje a una base de datos relacional.

Antes de comenzar a identificar las interfaces de Java con bases de datos relacionales, se dedicará este capítulo a explicar de forma muy general lo que es Java. Con el surgimiento de Internet y de páginas Web cada vez fue más necesario crear lenguajes de programación que se mantuvieran a la par con esta tecnología. Primeramente se creó HTML, pero debido a sus muy restringidos límites para crear animaciones y conexiones se diseñaron y crearon nuevos lenguajes. Es ahí donde surge Java.

Java es un lenguaje de programación que a diferencia de cualquier otro involucra nuevas características que lo convierten en un lenguaje muy robusto. Java no es únicamente un lenguaje que se utilice para programar sobre el Web, sino que fue construido para poder realizar las mismas tareas y el mismo tipo de aplicaciones que se pueden efectuar con otro lenguaje pero que además permite la creación de programas que pueden ser incrustados en páginas Web. Java fue desarrollado por Sun Microsystems y fue diseñado como un lenguaje orientado a objetos desde sus raíces. Al

examinar un fragmento de código escrito en Java es posible determinar un gran número de semejanzas entre Java y el lenguaje C++, esto se debe a que Java tomó como base el lenguaje C++. Incluso a pesar de heredar características de C++, existen puntos determinantes en Java que lo distinguen de C++.

La máquina virtual de Java.

Ésta constituye la piedra clave de los cimientos del lenguaje Java. Gracias a la máquina virtual de Java es que el lenguaje del mismo nombre posee elementos significativos que lo hacen un fuerte candidato para desarrollar aplicaciones preferentemente en Java y no en otro lenguaje de programación, tales características son las siguientes:

- La portabilidad que tiene Java a través de diferentes plataformas es quizá el punto más importante a considerar cuando se desarrollan aplicaciones multiplataforma. La independencia de plataforma de Java consiste en la posibilidad de trasladar un programa escrito en Java que trabaje en una plataforma Windows a una plataforma UNIX. Java mantiene una independencia de plataforma tanto a nivel código fuente como binario. Se dice que el código binario generado por el compilador Java es independiente de plataforma, ya que el compilador Java genera sus archivos en bytecodes[8]. Ahora bien, como los bytecodes no son específicos para una máquina en particular para poder ejecutar un programa Java se llama al intérprete Java, el cual lee los bytecodes

[8] *Bytecode. Los bytecodes de Java son una serie de instrucciones similares al código máquina, pero con la diferencia que no son específicas para un cierto procesador.*

y ejecuta el código. En el siguiente diagrama (figura 2.1) se ilustra la manera en que el compilador y el intérprete Java colaboran para dar a Java esa capacidad multiplataforma:

Figura 2.1

- Java fue diseñado como un lenguaje orientado a objetos, lo cual permite una cierta facilidad para modularizar programas. El enfoque de programación orientada a objetos permite eliminar en Java la necesidad de variables globales. Es decir, que se deben relacionar unos objetos con otros y no con variables que sean visibles para todos los módulos de un programa.
- Otra ventaja que tiene Java para aquellos programadores que utilicen C++ es la familiaridad que encontrarán entre ambos lenguajes ya que; Java deriva gran parte del lenguaje C++, pero elimina muchos de los conceptos que hacen compleja la programación en C++. Java no necesita de conceptos tales como apuntadores, ni de tener cuidado al hacer uso de la memoria debido a que Java la maneja de forma automática. Los apuntadores suelen ser complicados debido a que manipulan directamente la memoria de la computadora.
- Otra gran ventaja que tiene Java sobre otros lenguajes de programación es la posibilidad de crear applets. Un applet es un programa escrito con el fin de ejecutarse sobre una página del Web dentro del código HTML[9] de la página.

[9] *HTML. HTML por sus siglas en inglés abrevia Hypertext Markup Management Language.*

Para que un applet pueda ejecutarse se necesita un visualizador[10] habilitado para Java. Las diferencias que existen entre una aplicación Java y un applet son: una aplicación Java se ejecuta utilizando el intérprete Java por ejemplo desde la línea de comandos mientras que un applet aprovecha las ventajas del visualizador (browser) dentro del cual se ejecute. También al ejecutarse un applet dentro de un visualizador, éste limita al applet a ejecutar ciertas acciones lo que resulta en un aumento en la seguridad. Por esto, el crear un applet es una espada de dos filos, ya que aprovecha los recursos del visualizador a la vez que la seguridad del visualizador restringe muchas de las acciones que una simple aplicación Java es capaz de llevar a cabo.

- Independencia de lenguajes propietarios. Debido a que las especificaciones del lenguaje Java abarcan aspectos de un desarrollo abierto, éstas lo hacen un lenguaje libre de restricciones de sentencias únicas para una arquitectura en particular.

De manera resumida las principales características de Java son:

- Ofrece Independencia de plataforma
- Trabaja con una arquitectura distribuida
- Es dinámico
- Utiliza intérprete y compilador

[10] *Un visualizador es una aplicación creada para poder navegar sobre paginas HTML, utilizado para navegar en Internet o en una intranet. Algunos de los más comunes y que están habilitados para Java son Netscape Navigator, Internet Explorer y HotJava de Sun Microsystems.*

- Permite multiprocesos
- Ofrece enfoque para redes
- Es orientado a objetos
- Es portable
- Es robusto
- Es seguro

Terminología empleada en la Programación orientada a objetos por Java.

Ya que Java se basa en la metodología de "Programación orientada a objetos", entonces antes de codificar un programa Java se deben conocer los conceptos fundamentales de la programación orientada a objetos empleados por Java. Los términos más comunes empleados por Java son los siguientes:

- Clase: una clase es una plantilla o una definición de un objeto Java. Es decir que una clase define como serán cada una de las características de un objeto.
- Instancia: una instancia de clase es un objeto; así, si una clase es una manera abstracta de un objeto, la instancia es un elemento generado a partir de su definición de clase.
- Método: un método es una función o procedimiento que se encuentra definida para cada clase, por ejemplo una clase llamada base_de_datos podría tener un método llamado conexión.

- Atributos: al igual que un método una clase puede tener atributos. Los atributos de una clase son las características que distinguen una clase de otra. Cada vez que se requiere definir un atributo de una clase se incorpora una nueva variable. De hecho son variables globales del objeto.
- Herencia: La herencia dentro de la programación orientada a objetos es un punto muy relevante. La herencia le proporciona a una nueva clase el acceso a las variables y métodos de la clase de la cual está heredando.
- Interfaz: una interfaz es un conjunto de especificaciones que pueden ser implementadas por alguna clase, es decir que extiende el funcionamiento de dicha clase. Las interfaces proporcionan métodos adicionales a una clase además de los que haya heredado de una superclase. En Java las interfaces son importantes, ya que no es posible la herencia múltiple como en C++.
- Paquete: Los paquetes en Java se forman por una colección de clases y/o interfaces. Para poder hacer uso de cualquier paquete (a excepción del paquete default Java.lang), este debe importarse a la clase en donde se haga referencia.

Jerarquía de clases en Java

Como ya se mencionó el lenguaje Java se basa en la programación orientada a objetos, de ahí que involucre el diseño e implantación de clases para crear nuevos tipos de objetos. Cuando se define una clase en Java interviene la definición de métodos y atributos, aunque algunas de sus características pueden ser heredadas de alguna otra clase. Por ejemplo: se podría crear una clase denominada *animales* la cual abarcaría características comunes a cualquier animal existente sin importar su hábitat, o elementos

fisiológicos que distinguen a un animal de otro. Posteriormente se podría crear una subclase llamada *animales_terrestres.* El término subclase implica una clase que hereda variables y métodos de otra clase, en el ejemplo anterior la clase *animales_terrestres* heredaría las variables y métodos que se hayan definido en la clase *animales.* La clase *animales_terrestres* no sólo es capaz de heredar variables y métodos de una superclase, sino que además puede definir sus propios métodos y variables. De esta manera, la clase *animales_terrestres* contiene todos los elementos de la clase animales y elementos propios de su clase. La herencia en Java es un asunto mucho menos complejo a la herencia que existe en C++. En Java la herencia es sencilla, es decir, que cada subclase tiene una única superclase por lo que hereda características de una sola clase. Esto representa una condición de características ambivalente, ya que; aunque la comprensión de la jerarquía de clases resulta ser muy sencilla podría decirse que se restringe la posibilidad de heredar métodos y variables de más de una clase. Java soluciona este aparente problema con el uso de interfaces. Las interfaces le proporcionarán a la clase cierta funcionalidad que no sería capaz si se utilizara únicamente la herencia sencilla.

Cuando se crea una subclase, ésta invoca la palabra reservada "extends" para indicar quien es su clase padre (o superclase) y la palabra reservada "implements" para indicar que interfaces ocupará. Ya que una subclase hereda métodos de su clase padre estos pueden ser implementados por alguna instancia creada a partir de la subclase. Pero existen circunstancias en las cuales es necesario que alguno de los métodos de la superclase sea sustituido por otro método de la subclase; a lo que se conoce como sobreponer métodos. Una de las consecuencias que trae consigo la jerarquización de clases es el sobreponer métodos, ya que si se declara un método en una subclase que sea igual a otro en su clase padre entonces ¿ a qué método escuchará un objeto creado a partir de la subclase? La respuesta es muy simple porque dicho objeto siempre ejecutará

el método más próximo a la clase en la cual fue creado con una jerarquización ascendente. Un ejemplo de jerarquización de clases es el siguiente diagrama:

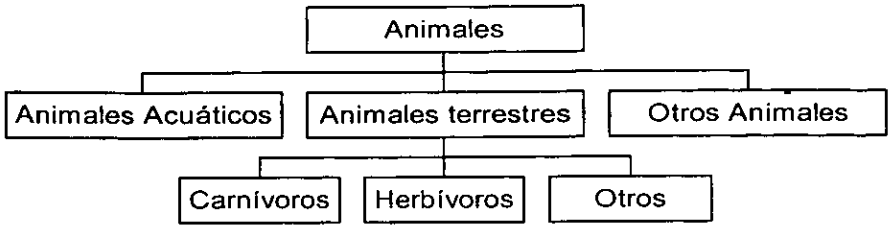

Figura 2.2

En la figura 2.2 el cuadro superior identificado como animales es la superclase de las clases animales acuáticos, animales terrestres y otros animales. En este mismo diagrama es posible observar como cada una de las subclases tiene una única clase padre, que representa lo que se conoce como herencia sencilla. Por ejemplo la clase *carnívoros* tiene como clase padre a la clase *animales terrestres*. En el momento en que se crea una instancia de la clase *carnívoros*, esta instancia hereda todas las características de la clase padre *animales terrestres* y ésta a su vez hereda todas las características de la clase *Animales*.

Principales Estructuras del lenguaje Java.

El compilador e intérprete de Java creados por Sun Microsystems carecen de un ambiente de desarrollo. Para crear un programa Java se hace en un editor común de texto. Es decir que se edita el programa como un archivo de texto plano. Este archivo se compila (generando así un archivo class) y posteriormente se ejecuta mediante el

intérprete Java. Cada archivo que ya este compilado se denomina como unidades de compilación. Las unidades de compilación de Java se distinguen por contener cuatro elementos que forman la estructura de un programa Java, siendo estos: declaración de paquetes, declaración de sentencias Import, declaraciones de clases e interfaces.

Declaración de paquetes:

Por default el API de Java incluye los paquetes básicos de Java y se asume que todos los objetos que se implementen se desarrollan a partir de este paquete. Ésta es la razón por la que Java asume que el código compilado se encuentra en el directorio actual. Ahora bien, para indicarle a Java dónde se encuentran los paquetes que se desean emplear se hace uso de la declaración de paquetes. Esto se implementa con la palabra reservada *package* seguida del nombre del paquete.

Sentencias Import

La sentencia Import de Java se asemeja mucho a sentencias que realizan de cierta forma la misma operación que include en el lenguaje C. Import se usa para indicarle al compilador Java cuando se involucran clases definidas en paquetes distintos al paquete Java.lang. Debido a que el paquete Java.lang incluye gran parte del código base de Java, este paquete es el default para Java y no se tiene que indicar al compilador que se buscarán clases en él. Las sentencias Import deben ser las primeras dentro de cada programa Java, ya que aparecen inmediatamente después de haber declarado el

paquete como tal. Como mencioné en el primer enunciado de este párrafo Import se asemeja a sentencias de otros lenguajes como por ejemplo: #include en C.

Import permite al programador crear una mejor modularización del desarrollo que esté realizando. Para ejemplificar esto es posible crear un paquete Java por cada módulo del sistema. Se declara una sentencia Import escribiendo la palabra Import seguida del nombre del paquete que se desea incluir en el programa; como sucede en el caso de hacer uso de las clases que definen el ambiente gráfico de Java en donde se indicaría: Import Java.awt.*;

Declaración de clases

La declaración de clases es una parte fundamental de Java. Java es un lenguaje orientado a objetos por lo que para crear cada objeto, éste posee características propias de una clase. En otras palabras la clase define el comportamiento de un objeto, de hecho la clase es un molde o plantilla de lo que será cada objeto creado a partir de ella. Todas las clases de Java derivan de otra, a excepción de la clase Object. La declaración de una clase en Java se hace de la siguiente forma:

```
class nombre_de_la_clase {
        atributos y/o métodos de la clase
}
```

Declaración de Interfaces

Las interfaces son por naturaleza clases abstractas. Una interface plantea una serie de métodos y variables sin tener que especificar la forma en como estos últimos se implementan. Cada interface puede ser añadida a una o más clases, incluso una única clase puede implementar más de una interface, lo que convierte a las interfaces en un sustituto de la herencia múltiple provista por otros lenguajes. La diferencia principal entre una interface y una clase radica en el hecho de que una interface sólo define la declaración de los métodos, más no implementa cada uno de ellos como lo hace una clase. La manera de declarar una interface es la siguiente:

```
Interface nombre_de_la_interface {
        Atributos y/o métodos que define la interface.
}
```

Ya que se conocen las definiciones tanto de una clase como de una interface es hora de conocer la manera de indicarle a una clase que además de habilitar sus métodos y atributos también ha de habilitar una interface tal y como se muestra a continuación:

```
Class nombre_de_la_clase implements nombre_de_la_interface {
        Atributos y/o métodos de la clase
}
```

Enseguida se muestra un ejemplo de cómo se declararía una clase sencilla que defina características (atributos) y métodos de carro.

```
Class carro extends vehículos {
        String Marca;
        String Num_cilindros;
        String Transmisión;
        Int encendido = 0;
}
```

En la definición anterior de la clase *carro* únicamente incluí variables dentro de la clase, es decir atributos. Siendo estos la Marca, Num_cilindros, Transmisión y encendido. La clase todavía no cuenta con métodos propios únicamente con aquellos que haya heredado de la superclase vehículos. Una acción muy sencilla que puede hacer el carro es encender su motor y para esto es posible escribir un método que lo haga. El método resultante sería el siguiente.

```
Class carro extends vehículos {
        String Marca;
        String Num_cilindros;
        String Transmisión;
        Int encendido = 0;
        Void Enciende(){
                If (encendido == 0)
                        encendido = 1;
        }
}
```

Applets

Anteriormente ya se mencionó que una de la ventajas principales que tiene Java sobre otros lenguajes es que posee la capacidad de crear applets. Los applets son programas creados en Java que para poder ejecutarse necesitan un visualizador que funcione como contenedor. Los applets funcionan como objetos incrustados dentro de las páginas Web creadas con html. Es decir que un applet se encuentra dentro de una página html y ésta a su vez se lee mediante el uso del visualizador. El visualizador proporciona ciertas ventajas a los applets, como son: una ventana, un ambiente gráfico y la interfaz de usuario. Alguien podría decir que una aplicación Java también es capaz de crear esta estructura y estaría en lo correcto; solo que una aplicación Java no requiere forzosamente de crear esta interfaz y para un applet esto ya esta creado por el visualizador.

Hasta este punto, parece que un applet fuera más poderoso que una aplicación Java, pero los applets también tienen sus restricciones. Las aparentes ventajas de un applet como son el poder ejecutarse desde cualquier máquina cliente lo limitan en cuanto a cuestiones de seguridad. La seguridad diseñada sobre los applets incluye la siguientes limitaciones:

- Un applet no puede leer ni escribir en el sistema de archivos de la máquina cliente sobre la cual se este ejecutando.
- Algunos visualizadores no permiten que el applet sea capaz de comunicarse con otro servidor además de aquel en el cual reside.

- Un applet no es capaz de ejecutar algún programa en la máquina cliente sobre la que se este ejecutando.
- Un applet no puede cargar programas nativos de la plataforma de la máquina cliente, como por ejemplo bibliotecas de intercambio dinámico.

Un ejemplo muy sencillo de un applet sería el siguiente:

```
Import Java.awt.*;

public class ejemplo extends Java.applet.Applet {
        Font letra = new Font("TimesRoman", Font.BOLD,10);
        Public void paint(Graphics letrero){
                letrero.setFont(letra);
                letretro.setColor(Color.black);
                letrero.drawString("Este es un ejemplo"), 10,60);
        }
}
```

Ahora bien, para que este applet pueda ser ejecutado habrá que incluir una llamada a este desde una página html, lo cual se realiza de la siguiente manera:

```
<HTML>
<HEAD>
<TITLE>Ejemplo</TITLE>
</HEAD>
<H1>Ejemplo</H1>
```

```
<BODY>
<HR ALIGN=CENTER WIDTH=75%>
<APPLET CODE="ejemplo.class" width=200 height=200>
</APPLET>
</BODY>
</HTML>
```

Es decir que dentro del código html se incluye una etiqueta llamada applet la cual hace referencia al applet ya compilado, en este caso a ejemplo.class. Cuando se carga esta página html dentro de un visualizador[11] habilitado para Java se ejecuta el applet ejemplo desplegando la leyenda "Este es un ejemplo".

Conexiones JDBC y ODBC

Una vez establecidos los conceptos básicos de lo que es una base de datos relacional y el lenguaje Java, se procede a explicar como es que existe una relación entre ambas tecnologías. Java en una primera instancia no aplicaba dentro de su campo de alcance conexiones con bases de datos, pero a medida que éste se fue desarrollando y mejorando incluye una manera sencilla que establece lazos entre un programa Java y la información contenida en una base de datos.

[11] *Es importante que el visualizador este habilitado para Java, ya que de otra forma no podrá ejecutarse el applet aunque si se cargará la página html.*

En una gran variedad de lenguajes de programación se implementó ODBC[12]. ODBC es un estándar de conexión a bases de datos relacionales. ODBC no es más que un traductor entre el lenguaje de programación y la base de datos, por medio de éste es posible realizar transacciones en la base de datos desde un programa desarrollado en un lenguaje de alto nivel. Existen diversos manejadores (drivers) de ODBC para una base de datos, ya que estos pueden haber sido hechos por la misma compañía que desarrolló la base de datos, el lenguaje de programación o hasta por terceros. Por ejemplo si se deseara establecer una conexión vía ODBC de Microsoft Visual Basic con Oracle Server existen drivers proporcionados por Oracle, Microsoft o por terceros como es Intersolv. Una de las características más importantes de los controladores ODBC es la amplia flexibilidad que proporcionan en cuanto a las bases de datos que pueden manejar además de no ser costosos. Por otro lado, los controladores ODBC no son tan rápidos como un controlador nativo. Un controlador nativo es aquel que fue diseñado específicamente para una base de datos en particular, de ahí que resulte ser mucho más eficaz pero mucho más costoso que un controlador ODBC.

En Java existen controladores similares a ODBC pero para uso exclusivo de Java llamados JDBC[13]. JDBC pretende crear un estándar de conexión para bases de datos relacionales entre Java y el mundo exterior. JDBC son controladores nativos para Java, por lo que en la mayoría de las situaciones en las que no se requiera flexibilidad de conexión a las bases de datos, éste sugiere una mejor alternativa que ODBC. No por esto se descarta la conexión vía ODBC. Además para poder establecer una conexión con un controlador ODBC se debe hacer un puente entre un controlador JDBC y otro ODBC.

---

[12] *ODBC significa Open Database Connectivity*

[13] *JDBC significa Java Database Connectivity*

Conexión vía ODBC y JDBC

Una vez que se conoce tanto la manera en que operan Java y una base de datos relacional, es conveniente comprender la forma en que éstas interactúan para crear aplicaciones de manera conjunta. Las formas más convencionales de establecer una conexión entre Java y una base de datos relacional es mediante el uso de controladores JDBC y controladores ODBC. El uso de cualquiera de estos controladores o de un puente JDBC/ODBC depende del diseño de la aplicación. Este modelo de conectividad entre Java y una base de datos contribuye a reformar la arquitectura que tradicionalmente existe entre cliente/servidor de dos capas a tres capas.

Es importante definir a ODBC, como un estándar de conexión entre una aplicación y una base de datos. ODBC presenta las siguientes características:

ODBC es una interfaz que permite trazar un puente entre una aplicación basada en Windows y una base de datos. ODBC permite que los desarrolladores de aplicaciones Windows puedan obtener información de más de una fuente de información cambiando el controlador de ODBC. Gran cantidad de ambientes y herramientas de desarrollo, así como la mayoría de las bases de datos relacionales utilizan ODBC.

Aunque los controladores ODBC no son muy rápidos, sustituyen esta aparente deficiencia con la flexibilidad que presentan en cuanto a las bases de datos que pueden manejar.

Por otra parte existen los controladores de tipo JDBC, estos no requieren de ODBC por lo que son llamados controladores nativos. Un controlador nativo es mucho más rápido y consecuentemente más eficiente que un controlador ODBC. Generalmente se propone el uso de controladores nativos para necesidades de bases de datos a gran escala; lo que normalmente sugiere que la base de datos se encuentre corriendo sobre una plataforma UNIX o bien un mainframe. La desventaja que presentan un controlador nativo JDBC frente a su contraparte ODBC es el costo.

JDBC día con día está convirtiéndose en un estándar para conectar Java con bases de datos. La relación entre JDBC y ODBC es estrecha debido a que pretenden cumplir la misma función. En Java para establecer una conexión con un controlador ODBC es necesario utilizar un puente JDBC/ODBC, que a su vez utiliza un controlador JDBC como el proporcionado por JavaSoft llamado JDBC-ODBC bridge, dicho controlador JDBC invoca al controlador ODBC que se encuentre instalado en la máquina cliente. De esta manera la responsabilidad de establecer un vínculo con el servidor de la base de datos recae en ODBC. Esto se ilustra en el siguiente diagrama (figura 3.1):

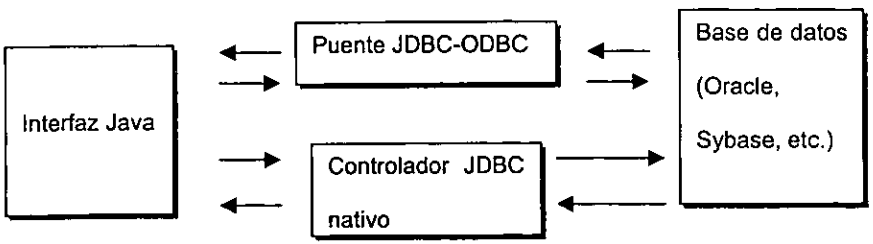

Figura 3.1

En la figura anterior se puede observar que ODBC depende en gran manera de JDBC cuando se requiere establecer un vínculo entre Java y una fuente de información. Sin embargo, hay que recordar que JDBC trabaja únicamente con Java, por lo que este caso no aplica en otras aplicaciones que utilicen ODBC.

JDBC no requiere de ODBC para conectarse a una base de datos, ya que estos son controladores nativos. Además JDBC fue creado explícitamente para Java por lo que resulta mucho más natural el uso de JDBC, debido a que este se implementa como un paquete interno de Java.

Un puente JDBC/ODBC presenta la desventaja de no ser compatible con algunos de los visualizadores de páginas web más comunes, el problema entonces se transforma de ser un asunto de compatibilidad a un asunto de seguridad. Java fue diseñado con base en la seguridad que presentarían los applets a quien los ejecutará en una máquina cliente. Dado que un controlador ODBC no fue escrito en Java se pierde el concepto de seguridad de Java y algunos visualizadores denotarían un error al intentar correr un applet que pretenda realizar una llamada a un controlador ODBC. Esta seguridad impuesta por el visualizador afectaría únicamente applets de Java y nunca a una aplicación Java como tal. Sun Microsystems provee de un visualizador llamado HotJava el cual tiene la facultad de eliminar las restricciones de seguridad para que pueda ejecutarse un applet. Pero como en la mayoría de las situaciones lo que se pretende es utilizar un applet que se ejecute sobre alguno de los visualizadores más reconocidos, en este caso sería más factible hacer uso de un controlador jdbc y dejar de un lado el puente JDBC/ODBC. Otra alternativa sería diseñar una aplicación sobre la base de un modelo de tres capas como el que se describe en los siguientes párrafos.

Modelos de dos y tres capas

El API[14] de JDBC tiene soporte para las arquitecturas de dos y tres capas dentro del modelo cliente-servidor. En los modelos de dos capas, un applet de Java o bien una aplicación Java interactúa directamente con la base de datos. Para esto es necesario contar con un driver JDBC que establezca la comunicación entre la interfaz y la base de datos. Los comandos SQL son enviados directamente a la base de datos y ésta contesta a la aplicación. Esta configuración se conoce como cliente/servidor, componiéndose de la máquina que cuenta con la aplicación o front-end (máquina cliente) y del servidor o back-end.

En el modelo de tres capas, la aplicación que soporta la interfaz del usuario envía los comandos a una capa intermedia conocida como capa de servicios, la que a su vez envía las sentencias SQL al servidor que soporta la base de datos. Este modelo tiene la ventaja de restringir los accesos a la información, ya que actúa como un filtro de seguridad. Además al hacer uso de una capa intermedia la capa que soporta la interfaz del usuario maneja un API de mayor nivel sin tener que efectuar llamadas de bajo nivel a la base de datos.

Actualmente la mayoría de las capas intermedias se escriben en lenguajes como C o C++, los que ofrecen un desempeño de buena calidad. Sin embargo, con la llegada de nuevos compiladores que traducen los bytecodes de Java en código nativo diseñado

[14] *API. Application Programmer's Interface. API es la interface que expone las rutinas de programación de una aplicación.*

específicamente para cada plataforma es una realidad el programar esta capa intermedia en Java.

Las tres capas que conforman este modelo son el front-end, la capa de servicios y el back-end. El front-end es la aplicación que interactúa directamente con el usuario y mediante esta se solicitan los servicios de la capa intermedia (capa de servicios). La capa de servicios contiene las reglas y lógica del negocio y actúa como un puente y filtro entre la aplicación cliente y la base de datos (back-end.). Aplicar este modelo de tres capas resulta muy eficiente a la hora de hacer mantenimiento, debido a que la lógica del negocio esta encapsulada en la capa de servicios únicamente hay que actualizar esta. Las máquinas cliente no se modificarán y podrán actuar de la misma forma en como lo venían realizando.

El esquema de tres capas en un principio resulta más abstracto y más difícil de analizar que una arquitectura de dos capas; pero su ventaja radica en la manera en que encapsula las reglas del negocio en la capa de servicios. El front-end o GUI solicita al servidor de servicios aquella información o procedimientos que necesita efectuar, mientras que por su parte, la capa de servicios es la que tiene contacto directo con el servidor de la base de datos. Un ejemplo de estas tres capas en Java pudiera ser una aplicación que funcionará a través de una intranet o internet. De esta forma el GUI estaría creado sobre un applet de Java. El applet se ejecutaría sobre una página HTML que estaría disponible sobre un visualizador de páginas Web habilitado para Java. Esta interfaz solicitaría objetos a la capa de servicios. La capa de servicios estaría conformada por un servidor de objetos Java los cuales se enviarían a las máquinas cliente como un flujo de objetos. El servidor de objetos sería quien solicitará a la base de datos todas las peticiones hechas por las máquinas cliente y contestará a ellas enviando objetos. Un

ejemplo de este tipo podría ser implementado con un controlador JDBC o hasta con un puente ODBC-JDBC.

En el ejemplo anterior es posible hacer uso de un controlador ODBC-JDBC debido a que el diseño se desarrolla sobre un esquema de tres capas. Por el contrario si se quisiera hacer uso de un puente ODBC-JDBC y a la vez de una arquitectura de dos capas sería algo fastidioso tener que instalar en cada una de las máquinas cliente el controlador odbc correspondiente. En el modelo de tres capas el controlador ODBC se instalaría únicamente en el servidor que contiene la capa de servicios, es decir en el servidor de objetos. Las máquinas cliente no necesitan ni siquiera contar con el controlador ODBC, ya que estas solicitarían objetos a la capa de servicios y nunca efectuarían una requisición o una consulta directa al servidor de la base de datos.

Ahora bien, si lo que se pretende utilizar es un controlador JDBC puro sería factible usar un modelo de dos capas. Claro esta que un modelo de dos capas sería más fácil de desarrollar en una primera etapa, pero cuando se requieran cambios o entrar en alguna fase de mantenimiento del proyecto la complejidad se vería afectada.

A continuación presento un diagrama sencillo que ilustra una aplicación diseñada sobre un esquema de tres capas.

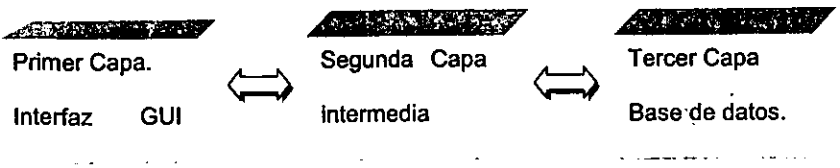

Figura 3.2

La conexión de Java

Java establece una conexión a una base de datos mediante un controlador que actúa como un puente entre Java y la base de datos. En Java existe una clase denominada "DriverManager" que determina qué controladores están activos y en base a estos determina el apropiado para establecer la conexión a la base de datos.

La clase "DriverManager" mantiene una lista de las clases "Driver" (controlador) que han sido previamente registradas a través del método DriverManager.registerDriver. Los controladores JDBC deben ser creados de manera que cuando el programador Java intente cargar una clase Driver ésta se registre automáticamente. Normalmente se carga un controlador JDBC llamando al método Class.forName, como por ejemplo:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

La línea anterior carga el controlador odbc.JdbcOdbcDriver y lo registra. Para abrir una conexión de Java con una base de datos se utiliza el objeto connection. Este objeto incluye las sentencias SQL que se envían a través del vínculo. Una aplicación es capaz de tener varias conexiones con una misma base de datos o con diferentes bases de datos. La forma de conectarse consiste en llamar al método DriverManager. GetConnection. Este método consiste en buscar un controlador JDBC que intente conectarse a una base de datos especificada como dirección URL[15]. Esto se ejemplifica

[15] *URL Uniform Resource Locator*

de la siguiente manera: Supóngase la base de datos NOMINA, la cual maneja un esquema con el usuario GVARGA01 y el password NOIMPORTA.

```
String direccion_url = "jdbc:odbc:NOMINA";
String usuario = "GVARGA01"
String password = "NOIMPORTA"
Connection conecta = DriverManager.getConnection(direccion_url, usuario, password);
```

En el fragmento de código anterior se utiliza un URL para identificar la base de datos requerida y establecer una conexión con ella. La forma como se compone un URL utilizado por JDBC determina cómo se conectará y a dónde se conectará. Este URL se divide en tres partes separadas entre ellas por dos puntos (:) . La primera parte es jdbc que indica que se utilizará JDBC para establecer la conexión. Enseguida se incluye el subprotocolo de conexión como por ejemplo ODBC y finalmente la cadena que identifica la base de datos. En el caso anterior esta cadena fue NOMINA.

Ahora que se logró establecer contacto con la base de datos hay que hacer que esta conexión funcione para interactuar con la información de la base de datos. Es decir mandar sentencias SQL a la base de datos y hacer que ésta proporcione una respuesta. Una gran ventaja de JDBC es que no restringe en lo absoluto el uso de las sentencias SQL que pueden ser enviadas a la base de datos; por lo tanto, las sentencias SQL pueden ser específicas de la base de datos que se esté utilizando.

JDBC propone tres clases y tres métodos en la interfaz Connection para enviar comandos SQL a una base de datos. Estas tres interfaces crean a su vez tres instancias de clase:

1. Statement. Este objeto se utiliza para enviar sentencias SQL muy sencillas y tienen la facultad de regresar registros.
2. PreparedStatement. Este objeto se usa para sentencias SQL que incluyan uno o más parámetros de entrada.
3. CallableStatement. Los objetos de este tipo se ocupan cuando se requiere ejecutar procedimientos de la base de datos. Este tipo de objetos además de manejar parémetros de entrada (IN) abarcan parámetros de salida (OUT) y parámetros INOUT.

Los objetos Statement contribuyen para enviar una sentencia SQL a a la base de datos utilizando el método "createStatement" de la instancia Connection. A su vez la instancia del tipo statement aporta el método executeQuery con la finalidad de que éste envíe la sentencia SQL a la base de datos y asigne su resultado a un objeto del tipo Resultset[16]. Además del método executeQuery un objeto Statement cuenta con los métodos executeUpdate y execute. Estos métodos se diferencian entre sí según el resultado que se desee obtener de la base de datos.

- Método executeQuery: Este método se ocupa para obtener Resultsets derivados de haber enviado una simple sentencia "Select".

[16] *ResultSet. Un Resultset es una matriz bidimensional que contiene los registros que forman el resultado devuelto por la base de datos después de haber enviado una sentencia SQL de tipo Select a la base de datos.*

- Método executeUpdate: La función de executeUpdate es transferir sentencias que no regresan un Resultset, pero que sí modifican la base de datos como por ejemplo sentencias Insert, Delete y Update. Mientras que el método executeQuery regresa un Resultset como resultado de la operación, el método executeUpdate regresa el número de columnas afectadas por la sentencia enviada a la base de datos. Este método se ocupa también en la ejecución de comandos referentes al lenguaje de definición de datos de la base de datos.
- Método execute: Este método no goza de gran popularidad debido a que se utiliza cuando ciertas sentencias SQL llegan a regresar más de un ResultSet, a efectuar más de una actualización, o bien a hacer una combinación de estas. Esto no sucede con frecuencia pero en ocasiones puede haber ciertos procedimientos de una base de datos que ejecuten alguna de estas combinaciones.

El método executeQuery regresa un objeto de tipo ResultSet que permite tener acceso a cada uno de sus elementos. Un Resultset maneja un apuntador al registro activo, es decir, a alguno de sus renglones. A través de dicho apuntador se tiene acceso a los campos que forman el registro, lo que significa que podemos accesar a sus columnas. El método next del objeto Resultset se encarga de mover el apuntador al siguiente renglón. Métodos tales como getString, getFloat, getInt, etc. proporcionan acceso al contenido de cada uno de los campos del registro activo. Por ejemplo, en el caso que se quisiera obtener el valor del campo "nombre" de un resultset cuya primer columna fuera el nombre el procedimiento es el siguiente:

```
String lstrNombre = rs.getString("Nombre");
```

o bien;

```
String lstrNombre = rs.getString(1);
```

En la primera situación se obtiene el valor del campo teniendo el conocimiento de cómo se llama la columna, mientras que en la segunda línea se obtiene el valor de la columna sabiendo qué es la primer columna del Resultset. En ambos casos Java tiene la capacidad de obtener valores de la base de datos y transformarlos en objetos que Java puede manipular. Para que Java pueda llevar a cabo la transformación de valores de la base de datos en objetos se utilizan los métodos get... (getFloat, getInt, getString, etc.), pero existen ciertos métodos get... que son más adecuados que otros para obtener un determinado tipo de valor. Por ejemplo; el método getInt es mucho más adecuado que el método getString para accesar un valor de tipo Integer, lo que no implica que getString sea incapaz de transformar el valor Integer en un objeto Java.

Enseguida se muestra un ejemplo de cómo funciona el objeto Statement. En este mismo ejemplo infiero que ya existe un objeto de conexión denominado Conex.

```
Java.sql.Statement Sentencia = Conex.createStatement();
ResultSet Resultado = Sentencia.executeQuery("Select Nombre from Agenda"
Where Clave = '007');
Resultado.next();
String lstrNombre = Resultado.getString(1);
```

En este ejemplo se envía la sentencia SQL para obtener el campo nombre de la tabla Agenda para aquel registro que tenga la clave '007'. Un detalle muy importante es

que cuando la base de datos regresa el Resultset Java no posiciona el apuntador al primer renglón del ResultSet. Por esto es que se ejecuta el método next del objeto ResultSet. El contenido del campo se almacena como una instancia de un objeto tipo String, y Java lo manipula como cualquier otro objeto.

Derivado del objeto Statement, se encuentra el objeto PreparedStatement frecuentemente usado para ejecutar sentencias SQL que lleven a cabo alguna modificación en la base de datos. PreparedStatement tiene la faculta de manejar parámetros de entrada (IN) que lo convierten en un objeto un poco más flexible. De forma análoga a como se obtienen valores de las columnas de un ResultSet se indican los parámetros de entrada que puede llevar un objeto PreparedStatement. Es decir que, en lugar de ocupar los métodos get... existen los métodos set... Un ejemplo muy sencillo para crear un objeto PreparedStatement y transferir a la base de datos una rutina que actualice la tabla Agenda es el siguiente:

Java.sql.Statement Sentencia = Conex.prepareStatement("Update Agenda set Nombre = 'Israel' Where Clave = '0074'");

Lo mismo ocurre en este caso pero con la única diferencia de transferir el valor de nombre y de clave como parámetros de la sentencia SQL de manera que:

```
PreparedStatement Actualiza = Conex.prepareStatement ("Update Agenda set Nombre = ? Where Clave = ?");
    Actualiza.setString(1,'Israel');
    Actualiza.setString(2,'0074');
    Int Total_Actualizado = Actualiza.executeUpdate();
```

Los parámetros se indican con el signo "?" y se transfieren con los métodos set..., en este caso particular con setString. Es notable que los parámetros deben indicarse antes de llamar al método executeUpdate. Obviamente en el ejemplo anterior el paso de parámetros a la sentencia SQL no tiene ningún sentido, a menos que estos parámetros cambiarán y se ejecutará una y otra vez la sentencia SQL de actualización.

Además de los objetos Statement y PreparedStatement existe un tercer objeto llamado CallableStatement. Este objeto permite la ejecución de procedimientos almacenados dentro de la base de datos. La llamada que se hace con este objeto a un Stored Procedure regresa una especie de parámetro de salida. Además se pueden incluir los parámetros de entrada y salida comunes al procedimiento. El objeto CallableStatement deriva atributos tanto del objeto Statement como del objeto PreparedStatement, con estas características heredadas se manejan sentencias SQL y parámetros de entrada (IN); pero como en ocasiones se requiere de parámetros de salida (OUT) estos deben registrarse antes de hacer la llamada al stored procedure.

Un objeto CallableStatement se crea a partir del método prepareCall del objeto connection y la llamada al Stored Procedure se realiza por medio de una secuencia en donde se debe incluir la palabra "call" seguida del nombre del procedimiento y de sus parámetros. Las líneas siguientes muestran como se hace la llamada a un procedimiento almacenado en una base de datos.

```
CallableStatement Llamada = Conex.prepareCall ("{call Actualiza_Agenda}")
```

En este ejemplo se presupone que existe un objeto de conexión llamado Conex, y que la base de datos enlazada a este objeto contiene un Stored Procedure llamado Actualiza_Agenda. De manera similar la forma en que se agregan parámetros de entrada a un objeto PreparedStatement se hace con parámetros de entrada a un objeto CallableStatement. Un parámetro de salida se indica con el método registerOutParameter del objeto CallableStatement señalando la posición ordinal del parámetro y el tipo de dato que regresará. El siguiente ejemplo asume que existe un objeto Connection llamado Conex y un Stored Procedure llamado Obtiene_Nombre que se encarga de regresar el nombre de un empleado teniendo como fuente su clave de empleado.

```
CallableStatement Llamada = Conex.prepareCall ("{call Obtiene_Nombre(?, ?)}");
Llamada.setString (1,'001'); // agrego la clave 001
Llamada.registerOutParameter(2, Java.sql.Types.Varchar);
Llamada.executeQuery(); // ejecuto el Stored Procedure
String Nombre = Llamada.getString(2);  //almaceno el nombre en la variable Nombre.
```

Manejo de transacciones

Este término es el más importante en la computación de empresas (Enterprise computing). Las Transacciones son un conjunto de eventos de negocios que se llevan a cabo para efectuar un trabajo. La transacción debe de ser considerada como una unidad indivisible y es indispensable llevar un control estricto desde su inicio hasta su etapa final. En caso de que cualquier parte dentro de la transacción fracase toda la operación

fracasará, por lo que se necesita una decisión unánime para decir que la transacción fue un éxito.

Una transacción consiste en un conjunto de sentencias SQL que después de ejecutarse son almacenadas permanentemente (a través de un commit) o en caso contrario, permite deshacer los cambios que éstas hicieron en la base de datos (a través de un rollback). En JDBC cuando se hace una conexión es por default de tipo auto-commit[17], es decir que después de ejecutar cada sentencia los cambios son permanentes. Si el modo auto-commit es deshabilitado entonces la transacción no será completada sino hasta que se ejecute el comando commit o el comando rollback.

En muchos casos es deseable no ejecutar ciertas sentencias hasta que una sentencia previa haya sido ejecutada con éxito. Es debido a esto que se agruparían todas estas sentencias como una transacción y en caso de que alguna de ellas fracase entonces se puede mandar todo el proceso hacia atrás. A continuación se presentarán propiedades llamadas ACID que deben de ser cumplidas por las transacciones.

El término ACID significa (Atomicidad "Atomicity", Consitencia "Consistency", Aislamiento "Isolation" y Durabilidad "Durability") a continuación se explicará cada una de ellas:

[17] *Auto-commit. Implica que después de ejecutar una sentencia SQL los cambios a la base de datos se hacen permanentes de forma automática*

Atomicidad: Una transacción es una unidad indivisible. Esta propiedad indica que no puede ejecutarse media transacción. Los dos estados que puede tener una transacción son o falla o se ejecuta con éxito.

Consistencia - Significa que la transacción debe de dejar el sistema en condiciones estables. En caso de que la transacción falle el sistema deberá ser reinicializado a su estado original.

Separación (Aislamiento) – La separación nos habla de la concurrencia en un sistema. Los cambios no serán visibles hasta que se lleve a cabo un commit. El comportamiento en un sistema multi-usuario debe de ser semejante al de un sistema monolítico.

Durabilidad –Los cambios serán permanentes después de que se efectúe un commit.

Una manera sencilla de ilustrar cómo trabaja un conjunto de sentencias para formar una transacción es el siguiente: en un sistema de inventarios cada vez que se efectúa una salida de productos del inventario el sistema debe contabilizar la operación. Es entonces cuando se tienen que completar dos operaciones conjuntamente. La transacción se conforma por las operaciones de salida del almacén y contabilización de la misma. Ambas operaciones no se pueden llevar a cabo de manera independiente. Cada una de las operaciones se ejecuta mediante una sentencia SQL, pero se agrupan como una única transacción; es decir, que cuando se declare la salida de algún producto del almacén, este tendrá que ser contabilizado, y si ambas operaciones fueron exitosas los cambios hechos a la base de datos serán considerados como permanentes.

Otro ejemplo es un traspaso electrónico de fondos. Esta es una transacción que agrupará dos operaciones: la primera consiste en sacar los fondos de una cuenta bancaria y la segunda en ingresar dichos fondos en otra cuenta bancaria. Para que dicho traspaso de fondos sea exitoso se deben haber realizado tanto la operación de salida de fondos como la de entrada de fondos, y mientras que las dos operaciones no sean efectuadas no se procederá a hacer los cambios permanentes en la base de datos.

Como se mencionó JDBC ofrece métodos para ejecutar procedimientos almacenados en una base de datos. Estos procedimientos pueden efectuar el manejo de transacciones por sí solos. Existen ocasiones en las cuales las sentencias SQL se envían al servidor desde cualquier máquina cliente y el manejo de las transacciones se controla desde el cliente. JDBC emplea métodos de la interfaz Connection para indicarle a la base de datos cuando realizar un "commit" o un "rollback". Son tres los métodos de la interfaz Connection:

setAutoCommit()

Este método recibe como parámetro un valor booleano indicando si se activa o desactiva el modo automático del comando commit. Por ejemplo la siguiente línea deshabilita el modo automático de confirmación de transacciones, por lo tanto el manejo de transacciones se hace de manera manual.

```
Coneccion1.setAutoCommit(false)[18]
```

[18] *En este ejemplo se considera que ya se cuenta con un objeto del tipo connection llamado Coneccion1.*

commit()

El método commit confirma los cambios hechos a la base de datos. Con este método los cambios realizados se hacen permanentes. La sentencia funciona de la siguiente forma:

Coneccion1.commit()

rollback()

El último de estos métodos, llamado rollback(), anula los cambios hechos a la base de datos desde el último commit que se haya hecho. La sentencia es como sigue:

Coneccion1.commit()

Al igual que el método commit(), el método rollback() debe utilizarse cuando el manejo de transacciones se realiza de forma manual.

Aplicación Prototipo para enlazar Java y una base de datos relacional usando JDBC.

Hasta este momento se conoce lo que es una base de datos relacional, así como la manera en que se interrelacionan cada uno de sus elementos. Una base de datos relacional es muy poderosa, pero para exponer a los usuarios la información que les interesa es necesario crear aplicaciones que hagan uso de ella. Para efectos de esta tesis se trata el lenguaje Java como aquella herramienta con la cual se construirán aplicaciones que den uso a la base de datos relacional. Este capítulo explica como es que una aplicación creada con Java se comunica con la base de datos.

La aplicación presupone que existe una base de datos para un sistema fiduciario el cual requiere mostrar en una intranet o bien en internet los saldos actualizados de cada uno de los clientes del banco. Lo que se pretende es realizar una consulta de saldos para cada cliente. Cada uno de los clientes tiene un número de contrato único. La aplicación cliente se pretende insertar dentro de una página HTML. Al insertar la aplicación cliente dentro de una página HTML permite su ejecución desde cualquier navegador habilitado para Java.

La aplicación estará formada por tres capas. La primera de ellas es la base de datos como tal. La segunda es la capa de servicios y la tercera la aplicación cliente. Tanto la aplicación cliente como la capa de servicios se construyen en Java y la base de datos se encuentra sobre Oracle. El siguiente diagrama (figura 4.1) muestra la forma de comunicación entre las capas.

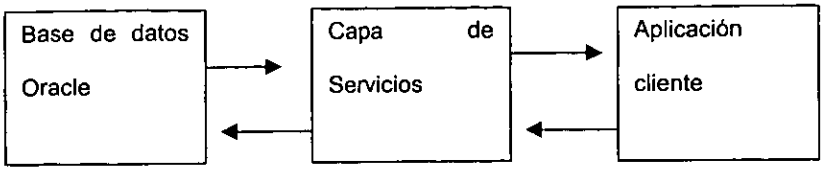

Figura 4.1

Descripción de cada una de las capas:

- La capa de la base de datos se encuentra montada sobre Oracle[19] y proporciona toda la información de los saldos de los clientes. Toda esta información proviene de otro sistema que generá día a día el saldo de cada uno de los clientes.
- La capa intermedia, es decir la capa de servicios se encarga de consultar los saldos de los clientes y para esto hace uso de un controlador JDBC y de sentencias SQL que manda a la base de datos. Además la capa de servicios se encarga de construir objetos que contengan los resultados regresados por la base de datos. Como ya había mencionado además de usar un controlador JDBC es posible establecer un puente JDBC/ODBC que será el utilizado para esta aplicación. EL uso del puente JDBC/ODBC se debe a la facilidad que existe para encontrar en el mercado un controlador ODBC para Oracle. Además como esta es la capa que se conectará directamente a la base de datos la instalación de un controlador ODBC únicamente se llevará a cabo una sola vez.
- La tercera y última capa puede ser una aplicación Java tradicional o bien un applet, pero como en este caso se desea que el cliente se encuentre insertado en una página HTML entonces se construirá un applet de Java. El enlace entre la capa de servicios y el applet se realiza a través de sockets.

[19] *Oracle es un motor de base de datos. Algunos otros son Sybase, SQL Server, Informix, etc..*

Es importante hacer notar las características de Java que hacen posible el enlace entre cada una de las capas del sistema. El enlace entre la base de datos y la capa de servicios ocupa un puente JDBC/ODBC para esto hace uso de un controlador JDBC de Javasoft y un controlador ODBC de Oracle. Mientras tanto la comunicación entre la aplicación cliente y la capa de servicios utiliza flujos de objetos y sockets. EL flujo de objetos utiliza la serialización de Java para llevar a cabo su trabajo. La serialización de objetos en Java funciona de una forma muy parecida a como lo hace un flujo de datos. Los flujos de objetos se pueden escribir a un archivo como lo hace un flujo de datos. En este caso específico se ocupa el flujo de objetos pero escribiéndolos y obteniéndolos de puertos, de ahí el uso de sockets.

Otra característica importante de Java que proporcionará elementos para construir la capa de servicios es la multitarea. Java puede manejar hilos[20]. Para este caso específico se construye un servicio único que consiste en hacer una consulta de cada cliente sobre su saldo actual. Aún así la capa intermedia contendrá esta consulta dentro de un hilo, esto con el fin de que cuando sean requeridos otros servicios se construyan cada uno de ellos dentro de un hilo. Cada servicio conforma una tarea que se mantiene ejecutando mientras que la capa de servicios esté corriendo.

Las ventajas que presenta esta arquitectura del sistema son las siguientes:

- Las conexiones a la base de datos se realizan a través de la capa de servicios lo que reditúa en un mayor grado de seguridad.

[20] *Los hilos en Java permiten el manejo de múltiples procesos en una aplicación Java.*

- En situaciones de requerir modificar consultas a la base de datos, los cambios se hacen sobre la capa de servicios y no en cada aplicación cliente.
- El tiempo ocupado en hacer instalaciones se reduce significativamente, ya que la capa de servicios es única y el applet insertado en la página HTML no necesita más que instalarse en el servidor web o de intranet.
- Como la aplicación esta desarrollada en Java y esta insertada sobre un página HTML se puede ejecutar sobre cualquier plataforma que tenga un navegador habilitado para Java.
- El manejo de transacciones se hace dentro de cada servicio por lo que reduce el riesgo de quitar integridad a la base de datos.
- Las consultas y por lo tanto la construcción de objetos se lleva a cabo en la capa de servicios, es decir en un servidor propiamente establecido para ello y se evita que el cliente ejecute procesos por sí solo lo que resulta en un mejor tiempo de respuesta mejorando así el desempeño general del sistema.
- Cuando se requiera deshabilitar alguno de los servicios simplemente se puede eliminar de la capa de servicios sin caer en la necesidad de eliminarlo de cada máquina cliente.

Desventajas que presenta esta arquitectura:

- El desarrollo inicial de una aplicación de tres capas es más complejo que el desarrollo de dos capas.

Desarrollo de la aplicación

Los elementos que se requieren para desarrollar la aplicación son:

- La base de datos implantada en Oracle
- La capa de servicios desarrollada en Java
- La capa de cliente desarrollada como un applet de Java
- El controlador ODBC para Oracle
- La página HTML y por supuesto algún navegador que permita cargar la página
- Además todo el código se implantará dentro del personal web server de Microsoft.

Dentro de la base de datos existe una tabla llamada Patrimonio_Negocio, la cual se encuentra definida de la siguiente manera:

Nombre de Columna	Tipo
NO_CONTRATO	**NUMBER(10)**
NO_SUBCONTRATO	**NUMBER(10)**
CVE_MONEDA	**CHAR(3)**
SALDO_EFECTIVO	**CHAR(17)**
SALDO_INMUEBLES	**CHAR(17)**
SALDO_INVERSIONES	**CHAR(17)**
SALDO_MUEBLES	**CHAR(17)**
SALDO_TOTAL	**CHAR(17)**

Esta tabla muestra el saldo de cada cliente clasificado según el producto o productos de su cuenta, los productos son los saldos de efectivo, inversiones, etc.. La llave primaria para esta tabla esta definida por la columna: no_contrato. La aplicación requiere consultar el último saldo total del patrimonio de un contrato determinado; es decir, que se requiere el saldo total actual. El saldo total actual se obtiene con una sentencia SQL muy sencilla como la siguiente:

```
SELECT saldo_total
FROM patrimonio_negocio
WHERE no_contrato = 1
```

En la sentencia anterior se presupone que se requiere el saldo total para el contrato número 1, subcontrato número 0 y clave de moneda 'MN'. Claro que para que esto sea en verdad útil los parámetros anteriores deben ser modificables a solicitud del usuario. Por lo tanto se construirá sobre la base de una consulta dinámica. Estos parámetros se obtienen a partir de la capa cliente y enviados a la capa de servicios como un flujo de objetos. La capa de servicios ensambla la sentencia SQL dinámica con los parámetros recibidos y envía la sentencia a la base de datos a través del puente JDBC/ODBC. Utilizando este mismo enlace la capa de servicios recibe la respuesta del servidor y procede a construir un objeto que contenga la información solicitada por la capa cliente. Por eso es que la capa de servicios para este caso en específico puede referirse como servidor de objetos. Esta capa regresa al cliente un objeto y el cliente se encarga de mostrarlo al usuario.

Capa de Servicios

La capa más importante y medular de la aplicación es la capa de servicios, ya que es en realidad la que efectúa la consulta y ensambla el objeto solicitado por el usuario. A continuación se muestra el código necesario para implementar el servicio de consulta de saldos:

```
import Java.awt.*;
import Java.io.*;
import Java.net.*;
import Java.net.URL;
import Java.sql.*;

public class MainServer{
        public MainServer(){
                Servicio1 S1Thread = new Servicio1(8888,8889);
                Thread s1 = new Thread(S1Thread);
                s1.start();
        }

        public static void main(String args[]) {
                MainServer app = new MainServer();
        }
}
```

```
//clase que implementa el servicio uno que consiste en obtener el saldo del cliente
class Servicio1 implements Runnable {
        int id1 = 0;
        int id2 = 0;
        public Servicio1 (int port1, int port2){
                id1 = port1;
                id2 = port2;
        }

        public void run() {
                try {
                        ServerSocket serverIN = new ServerSocket(id1);
                        ServerSocket serverOUT = new ServerSocket(id2);
                        while (true) {
                                Socket s = serverIN.accept();
                                //objeto de entrada
                                InputStream istream = s.getInputStream();
                                ObjectInputStream in = new ObjectInputStream(istream);
                                //leemos un numero (el numero de contrato)
                                Integer Contrato = new Integer(0);//inicializo el objeto.
                                Contrato = (Integer)in.readObject();
                                s.close();
                                System.out.println(Contrato.toString());
```

```
                    //objeto de salida , creamos el objeto
                    Salida salida1 = new Salida(Contrato);

                    Socket s2 = serverOUT.accept();
                    //escribo el objeto.
                    OutputStream ostream = s2.getOutputStream();
                ObjectOutputStream out = new ObjectOutputStream(ostream);
                    out.writeObject(salida1);
                    out.flush();
                    ostream.close();
                    s2.close();
                }
            }
            catch (Exception e) {System.out.println(e.getMessage());}
        }
}

//clase que hace la consulta y crea el objeto de salida.
class Salida extends Frame{
        Integer lint_NoContrato = new Integer(0);

        public Salida(Integer NoContrato){
            lint_NoContrato = NoContrato;
            List list = new List();
            list.addItem (Consulta(lint_NoContrato));
```

```
        this.add("Center",list);
        this.setSize(300,100);
        this.show();
    }

    public String Consulta(Integer NoContrato){
        String url   = "jdbc:odbc:produccion"; //jdbc-odbc bridge.
        String query = "SELECT saldo_total FROM patrimonio_negocio WHERE
        no_contrato =" + NoContrato.toString();
        String regresa = "";

        try {
            Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection (url, "serfdes1",
            "produc");

            Statement stmt = con.createStatement ();
            ResultSet rs = stmt.executeQuery (query);
            rs.next();
            regresa = rs.getString(1);
            rs.close();
            stmt.close();
            con.close();
        }

        catch (Exception e) {e.printStackTrace ();}
```

```
            return regresa;
        }
}
```

Antes de describir el funcionamiento del programa se deben mencionar las librerías Java que se utlizan y el porque de ellas; dichas librerías se agrupan en tres bloques: la librerías awt que proporcionan todo el manejo de ambiente de ventanas, las librerías io, net y net.URL que proporcionan los sockets y sus conexiones, y finalmente la librería SQL que maneja la conexión con la base de datos y las sentencias enviadas a esta.

El programa esta formado por tres clases MainServer, Servicio1 y Salida. La clase MainServer es la principal y se encarga de levantar cada uno de los hilos que contienen a los servicios, además pasa como parámetros los puertos que utilizarán los sockets para levantar el servicio1 . Es en esta clase en la que se tendrían que levantar cada uno de los servicios que sean requeridos según el desarrollo de la aplicación.

Enseguida está la clase Servicio1 que realiza dos funciones fundamentales, la primera consiste en escuchar a través de un socket[21] el llamado que efectuará el applet de Java. En este puerto recibirá el número de contrato para el cual se requiere efectuar la consulta. Posteriormente hace un llamado a la clase Salida. Con los resultados

[21] *Los sockets en Java son similiares a los sockets estándares en UNIX, en los cuales para establecer una conexión desde un cliente se indica la dirección del servidor y el número de puerto al cual desea conectarse. Una vez establecida la conexión de sockets, en Java es posible establecer un flujo de datos u objetos.*

obtenidos de la clase Salida escribe el objeto resultante en un segundo puerto con el fin de que el applet de Java lea el contenido de este y lo despliegue al usuario.

La tercera y última clase de la aplicación es la llamada clase Salida. Esta clase se encarga de armar la sentencia SQL que enviará a la base de datos a través del puente JDBC/ODBC. Una vez obtenida la respuesta crea un objeto de tipo Frame el cual regresa a la clase Servicio1. Este objeto creado por la clase Salida es el que se envía a la aplicación cliente usando el flujo de objetos de Java.

Este programa se ejecuta como una aplicación Java común y no necesita de un navegador, ya que no es un applet.

Capa cliente

La capa cliente esta formada tanto por la aplicación cliente, el cual es un applet de Java como por la página HTML que hace el llamado al applet. El código del applet es el siguiente:

```
import Java.awt.*;
import Java.io.*;
import Java.net.*;
import Java.applet.*;
public class Serial3 extends Applet {
        private Frame f;
```

```
public void init() {
        try {
        int NoContrato = 102;
        Integer Contrato = new Integer(NoContrato);

        Socket s = new Socket("205.239.204.224",8888);
        Socket s2 = new Socket("205.239.204.224",8889);
        //mando el numero de contrato que me interesa
        OutputStream ostream = s.getOutputStream();
        ObjectOutputStream out = new ObjectOutputStream(ostream);
        out.writeObject(Contrato);
        out.flush();
        ostream.close();
        s.close();

        InputStream istream = s2.getInputStream();
        ObjectInputStream in = new ObjectInputStream(istream);
        f = (Frame)in.readObject();
        istream.close();
        s2.close();
        }
        catch (Exception e) {}
}

 public void start() {
```

```
            f.setLocation(0,0);
            f.setVisible(true);
            f.requestFocus();
        }

        public void stop() {
        f.setVisible(false);
        }

        public void destroy() {
        f.setVisible(false);
        f.dispose();
        }
}
```

Este programa deriva de la clase Applet de Java lo que le permite comportarse como tal. Las funciones que realiza son dos. La primera consiste en enviar el número de contrato para el cual nos interesa obtener su saldo. En este caso se escogio arbitrariamente el número de contrato 102. Para enviar este número de contrato se ocupa como referencia la dirección IP[22] de la máquina sobre la cual esta corriendo la capa de servicios. Además se indica el puerto al cual se enviará el número de contrato. Es entonces que entra en operación la capa de servicios y al finalizar envía un objeto al applet. El applet lee el objeto y lo muestra al usuario. El applet utiliza dos sockets para

[22] *La dirección IP de una máquina la identifica como única dentro de una red configurada con el protocolo TCP-IP.*

establecer la comunicación con la capa de servicios, el primero para enviar objetos, es decir el número de contrato y el segundo para recibir objetos; en este caso una ventana que contiene el saldo del contrato solicitado.

En este applet es posible observar que la responsabilidad de efectuar la consulta de saldos recae en la capa de servicios. El applet jamás se conecta con la base de datos y su trabajo consiste única y exclusivamente en solicitar objetos, recibirlos y mostrar su contenido al usuario final. Al ejecutar este applet se mostrará una pequeña ventana que contendrá el saldo actual del contrato número 102.

Ahora bien para que un applet se pueda ejecutar necesita de un contenedor. Alguien debe llamarlo, es decir una página HTML. El código HTML para llamar a un applet como este puede ser algo tan sencillo como lo siguiuente:

```
<title>Consulta Saldos</title>
<h1>Consulta Saldos</h1>
<hr>
<applet code="Serial3.class" width=300 height=100>
</applet>
<hr>
<a href="Serial3.Java"> </a>
<br>
<a href="example1.html"> </a>
```

La arquitectura de tres capas utilizada para desarrollar esta aplicación permite la conexión de múltiples clientes que pueden consultar sus saldos y enlzarse a ellos a través

de una capa intermedia. De manera que si se requiere eliminar un servicio, este se da de baja en la capa de servicios sin tener que modificar las aplicaciones cliente.

La manera general de operar de la aplicación es la siguiente. Primero el usuario a través de una página web ejecuta el applet de Java el cual envía el número de contrato 102 a la capa de servicios. Ver la siguiente figura (4.2):

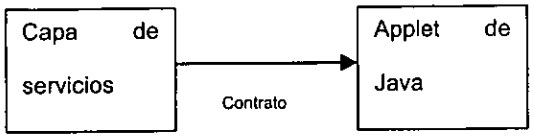

Figura 4.2

Una vez que la capa de servicios recibe el número de contrato 102 consulta el saldo del mismo en la base de datos de Oracle (utilizando JDBC). Se regresa este saldo a la capa de servicios.

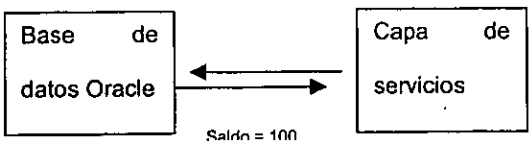

Figura 4.3

La capa de servicios procede a armar el objeto que enviará al applet de Java. Es importante saber que entre la capa de servicios y el applet la comunicación utiliza dos sockets uno para enviar objetos y otro para recibirlos. Ver la siguiente figura (4.4).

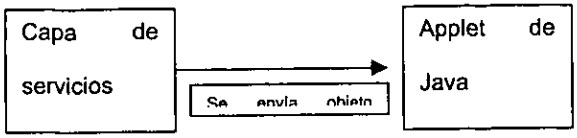

Figura 4.4

Finalmente el applet de Java recibe el objeto Salida y lo muestra al usuario. De esta forma se lleva a cabo la consulta de saldos para el contrato 102. Esta aplicación es muy sencilla ya que inclusive el numero de contrato es predeterminado; pero a pesar de ello, la manera en como esta construida permite hacerla crecer sin mayores complicaciones.

Conclusiones

Es un hecho que realizar este trabajo requirió de investigación pero no solo de forma teórica sino práctica también. De hecho el enfoque principal que era llegar a construir una aplicación multicapas con Java se alcanzó con éxito. Claro está que para llegar a esta meta primero aprendí como es Java y como interactúa con una base de datos. Esto por el lado del lenguaje Java, pero también se requirió de conocimientos sobre bases de datos relacionales y por consiguiente de SQL.

Este trabajo nunca pretendió ser un manual de Java o de SQL o una combinación de ambos; sino más bien conseguir que tanto Java como una base de datos relacional conformen un núcleo para implementar nuevas aplicaciones utilizando estas tecnologías. Un sistema o una aplicación es un rompecabezas, ya que para que éste funcione como tal requiere de muchas partes individuales que en su conjunto funcionan como un solo ente. Aquí se explicó como es que estas partes se interrelacionan entre sí para construir una aplicación de tipo cliente servidor multicapas. Se demostró que la tecnología proporcionada por Sun Microsystems, es decir JDBC, y la tecnología ODBC (creada por Microsoft) pueden unirse para proporcionar el acceso que Java necesita para llevar a cabo un sin fin de tareas con una base de datos relacional.

La aplicación que se construyó en el último capítulo del presente trabajo; aunque es muy sencilla proporciona las bases necesarias para construir una aplicación a gran escala utilizando Java, alguna base de datos relacional como puede ser Oracle, JDBC, ODBC. Además como la aplicación se construye con Java esta es multiplataforma y también pueden accesarla clientes que se conecten mediante Internet.

Esta investigación no pretende quedarse como tal, sino que es práctica y tiene un sin fin de usos. La Red Internet tan ampliamente divulgada en los últimos años se beneficia en gran manera de Java y JDBC; ya que aunque existen distintas tecnologías muy pocas prometen llegar tan lejos como Java.

Algunas situaciones en las cuales Java y JDBC muestran su alcance son Internet, esquemas cliente-servidor y por supuesto la capacidad de operar en plataformas distintas. En la actualidad existen muchísimos sistemas que hacen uso de un esquema cliente-servidor, en muchos casos es necesario migrar a Internet y éste es el campo de acción para JDBC. Esto significa que la tecnología JDBC no solo implica el desarrollar aplicaciones nuevas, sino que además permite utilizar las bases de datos existentes en el mercado e implementar nuevas interfaces gráficas que puedan operar en Internet, o bien en una Intranet. En otras situaciones se requiere de una aplicación que interactúe con bases de datos existentes. Esta aplicación debe correr en plataformas tan variadas como puede ser un sistema Windows u otro UNIX. Java tiene una naturaleza multiplataforma, por lo que aunque la aplicación opere en sistemas operativos distintos el código fuente es el mismo. JDBC proporciona el acceso a las base de datos sin importar sobre que plataforma se éste trabajando.

Espero que mucha gente puede verse beneficiada con la lectura de este documento y que sirva como fuente para construir nuevas aplicaciones e interfaces utilizando tecnología relativamente nueva.

Bibliografía

- Afergan, M. et al. (1997). Java. En *Programación en Web* , 401-660. México: Prentice Hall Hispanoamericana S.A.
- Arnow, D.M. y Weiss G. (1999). *Introduction to Programming using Java.* Addison-Wesley Pub. Co.
- Baum, D. (1998). *Three Tiers for Client/Server*, Editorial WILEY.
- Becerril, F. (1998). *Java a su alcance*, México: McGraw Hill.
- Flanagan D. (1997). *Java in a Nutshell*, Sebastopol: O'Reilly and Associates.
- Groff, J.R. y Weinburg P.N. (1998). *Guía de SQL*, México: McGraw Hill.
- Hamilton G. et al. (1997). *JDBC Database Access with Java*. Addison-Wesley Pub. Co.
- Hobbs, A. (1998). *Aprendiendo programación para bases de datos con JDBC*. México: Prentice Hall Hispanoamericana S.A.
- Hoth D. et al. (1999). SQL, 223-260, y ODBC 1377-1432. En *Oracle Development Unleashed,* Indianapolis: SAMS.
- Lemay, L. y Perkins, C.L. (1996). *Aprendiendo Java en 21 días*. México: Prentice Hall Hispanoamericana S.A.
- Liu, J.y Narayanan, S.N. (1999). *Entreprise Java Developers Guide*. McGraw Hill.
- Muñoz C. (1998). *Como Elaborar y Asesorar una Investigación de Tesis,* México: Prentice Hall Hispanoamericana, S.A.
- Orfali, R. et al. (1998). *Cliente/Servidor Guía de Supervivencia*, México: McGraw Hill.
- Pappas, C.H. y Murray, W.H. (1993). Clases. En *Manual de Borland C++*, 363-408. Madrid: Osborne McGraw Hill.

- Reese. G. (1997). *Database Programming with JDBC and Java*, Sebastopol: O'Reilly and Associates.
- Urman, S. (1998). *Oracle8 Programación PL/SQL*, Madrid: Osborne McGraw Hill.
- White, B. et al. (1997). Getting Started with the Java Development Kit, 15-26., y Object Serialization, 199-230. En *Using Java Beans* Indianapolis: QUE.

Referencias en Internet.

- developer.Java.sun.com
- ftp.iftech.com/DevJournal/pdf/9909_taylor_jdbc.pdf
- web2.Java.sun.com/docs/ (Java whitepapers)
- www.Java.sun.com
- www.Javasoft.com
- www-personal.umich.edu/~hmasing/gradschool/si544/JDBC/index.html
- www.dfki.uni-kl.de/km/Java/Java/tutorial/tutorial/jdbc/basics/
- www.stars.com/Authoring/DB/Intro/jdbc.html