UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO

ENEP ACATLAN

POWERBUILDER UN 4GL ORIENTADO A OBJETOS

T E S I S

PARA OBTENER EL GRADO DE :

LICENCIADO EN MATEMÁTICAS
APLICADAS Y COMPUTACION

P R E S E N T A

CARLOS ALEJANDRO VILLARRUEL CORTES

Ciudad Universitaria, 1999.

PowerBuilder un 4GL Orientado a Objetos

Agradecimientos

A Dios

Sí enumerará todas las cosas que Dios me ha dado, implicaría realizar muchos trabajos como este, por cada día de mi vida. Por esto solo le agradezco todo lo que me ha dado y que también me ha quitado.

A mi mamá Chuy

Le doy las gracias por haberme dado la vida, por educarme, por ayudarme a ser lo que soy, por darme su cariño, amabilidad y comprensión durante mi vida y le pido a Dios que así siga siendo por mucho tiempo.

A mi familia

Les agradezco enormemente a mis hermanos, Miguel, José, Ello por todo lo que me han enseñado y por todo lo que me cuidaron. A mi hermana por su apoyo para la elaboración de este trabajo y por su ayuda en estos últimos años. A mi tíos, Lupe y Alfredo, que creyeron en mi y siempre me animaron a seguir estudiando. A mis primos Gabriel y Cesar por que cada vez que estoy con ellos me inyectan esa energía infantil que uno pierde cuando crece.

A mi novia Patty

Gracias por todo su apoyo, por lo que me haz enseñado, por lo mucho que me haz enseñado a crecer y principalmente por el amor que me das y que espero que así siga por siempre. Y recuerda que detrás de un gran hombre hay una gran mujer. Te amo

A todas las personas que han creído en mi, y me enseñaron a querer triunfar en la vida de la forma más correcta posible. A aquellas personas que tomaron un poco de su tiempo para educarme. A los sinodales que hicieron de este trabajo una gran aportación a la carrera. A las personas que me ayudaron con todos los trámites que implica este trabajo. A las personas que durante mi vida profesional me han enseñado a actuar como tal.

Dedicatoría

Este trabajo se lo dedico a varias personas que son y han sido base en mi vida.

A mi mamá Chuy,

A mi papá.

A mi novia Patty.

Especialmente se lo dedicó con todo mi amor y cariño a mi hermosa abuelita.

Índice

Introducción

En el área de la computación la palabra evolución no es un concepto ajeno, por el contrario es una palabra natural (todo lo que existe en la naturaleza sigue una regla muy sencilla, evoluciona o se extingue), la evolución se da en las dos áreas principales de la computación: el *hardware* y el *software*.
En el área del *hardware* la evolución se ha dado en diferente forma, en rapidez, capacidad de almacenamiento en discos u otros dispositivos, aumento de memoria, reducción en el tamaño de los equipos, nuevas formas de almacenamiento, etc.

La evolución del *software* se ha debido a los malos o nulos análisis sobre los sistemas que provocan que los tiempos de creación de sistemas sean bastante elevados, por lo que surge la necesidad de crear lenguajes de programación que permitan ahorrar tiempo en la fase de desarrollo. La siguiente frase muestra claramente la necesidad de reducir esos tiempos.

> **" ¿Qué hacer?" en vez de "¿Cómo Hacerlo?"**
> **"Como solucionar sus problemas sin entrar en más problemas"**

Es esta filosofía lo que da origen a los lenguajes de programación con metodología de programación orientada a objetos, los lenguajes visuales, los lenguajes de cuarta generación.
Powerbuilder es un lenguaje de cuarta generación con programación orientada a objetos que hace que esta se realice con mayor rapidez buscando tener un lenguaje natural (muy similar al del ser humano), permitiendo que usuarios no expertos en programación puedan desarrollar aplicaciones sin ningún problema.

Las necesidades con las que se han encontrado los usuarios de las computadoras, han llevado a la tecnología y a la industria del *software* ha tener que crear nuevas metodologías de desarrollo en las que se invierta menos tiempo en el desarrollo y más en el análisis.

Esta tesina tiene como objetivo mostrar los conceptos básicos para poder realizar una aplicación con Powerbuilder. Cuando inico este trabajo se encontraba en el mercado la versión 4.0 de Powerbuilder, con el paso del tiempo surgieron las versiones 5.0 y 6.0, por lo que se buscó enfocarlo para permitir el uso de cualquier versión.

Powerbuilder tiene sus bases de desarrollo en la metodología de programación orientada a objetos y en la de cliente servidor. Por esto que el capítulo 1 pretende exclusivamente definir conceptos relacionados con estas metodologías, dando así las bases que permitan el mejor entendimiento de este trabajo.

El capítulo 2 presenta los conceptos de base de datos y de SQL, que son básicos para el desarrollo de aplicaciones cliente servidor. Este capítulo habla acerca de la sintaxis requerida para la creación de una base de datos, de sus tablas, columnas, etc. Además de los conocimientos mínimos para permitir el uso de una base de datos eficiente y que refleje la información de la entidad, a la cual representa la base. Aunque Powerbuilder cuenta con una *Painter* que permite generar código SQL de manera gráfica, sin tener un dominio preciso de las sentencias SQL, se recomienda que se cuente con la información mínima necesaria para el mejor desempeño de las aplicaciones Powerbuilder. Por lo que en este capítulo se mencionará la información SQL sin entrar en el detalle de si pertenece al algebra relacional o al cálculo relacional, simplemente se concretará a mostrar las sentencias de SQL.

En el capítulo 3 se muestran los conceptos fundamentales para el desarrollo de aplicaciones Powerbuilder. Con este se busca dar a conocer las herramientas de desarrollo por las cuales a PowerBuilder se le considera un lenguaje de Cuarta Generación. Además de la manipulación de objetos, (ventanas, Datawindow, botones, menús), herramientas de base de datos, funciones, gráficos).

Esta tesina muestra a los alumnos de la carrera de matemáticas aplicadas y computación un nuevo lenjuaje de programación que se encuentra muy fuerte en el mercado, está enfocado para los alumnos de por lo menos el séptimo semestre de la carrera, pues los conocimientos con los cuales cuentan le permitará entender con mayor facilidad este trabajo.

Capítulo 1

4GL y OOP Fundamentos para PowerBuilder.

Objetivos

La información presentada en esta sección menciona una breve nota de la evolución de los lenguajes de programación, muestra algunos conceptos que permiten el mejor entendimiento de la OOP, del modelo cliente servidor y de ODBC, dejando así una base teórica que permita el mejor desempeño de una aplicación PowerBuilder.

1.1 Breve Introducción acerca de los Lenguajes de Programación

Primera Generación	⇒	Leng. Máquina	
Segunda Generación	⇒	Leng. Ensamblador	Niveles de los Lenguajes de programación
Tercera Generación	⇒	Leng. Alto Nivel	
Cuarta Generación	⇒	Leng. Declarativo	

Los "lenguajes ensambladores" y los "lenguajes máquina" son dependientes de la computadora. Cada tipo de máquina, tal como VAX de Digital, tiene su propio lenguaje ensamblador asociado, que es simplemente una representación simbólica del lenguaje máquina asociado, lo cual permite una programación menos tediosa, sin embargo, es necesario un conocimiento de la arquitectura mecánica para realizar una programación efectiva en cualquier de estos niveles de lenguajes.

Los lenguajes de "alto nivel", son los más conocidos y fueron los más utilizados hace algún tiempo. Aunque no son fundamentalmente declarativos, estos permiten que los algoritmos se expresen en un nivel y estilo de escritura fácilmente legible y comprensible por otros programadores. Además, los lenguajes al alto nivel tienen normalmente la característica de "Transportabilidad". Es decir están implementados sobre varias máquinas, de forma que en un programa puede ser fácilmente "transportado" de una máquina a otra sin una revisión substancial. En este sentido, se llaman "independientes de la máquina". Ejemplos de estos lenguajes de alto nivel son Pascal, APL y Fortran (para aplicaciones científicas), Cobol (para aplicaciones de procesamiento de datos), Snobol (para aplicaciones de procesamiento de Texto) Lisp y Prolog (para aplicaciones de Inteligencia Artificial), C y Ada (para aplicaciones de programación de Sistemas) y PL/1 (aplicaciones de propósito general).

Existen lenguajes de programación que se encuentran entre los lenguajes declarativos y los lenguajes de alto nivel. Para esta categoría no se maneja porque se cree que estos surgieron por el auge que tuvo Windows y que debido a ello se crearon los lenguajes visuales. No son lenguajes declarativos por todas las cosas que aun se les tiene que programar "Como hacer", tampoco caen en el rango de lenguajes de alto nivel por que existen muchas funciones y objetos que ya no tienen la necesidad de preocuparse por como hacerlas.

Los lenguajes de Cuarta Generación son los más parecidos al castellano o al inglés, en su potencia expresiva y en su funcionalidad y están en el nivel más alto con respecto a los otros. Son fundamentalmente lenguajes de órdenes, dominados por sentencias que expresan "lo que hay que hacer" en vez de "como hacerlo". Ejemplos de estos, son los lenguajes estadísticos como lo es SAS y SPSS y los lenguajes de búsqueda en base de datos, como Natural, y Powerbuilder. Estos lenguajes se desarrollaron con la idea de que los profesionales pudieran asimilar más rápidamente el lenguaje y usarlo en su trabajo, sin la necesidad de programadores expertos o prácticas de programación.

1.2 Programación Orientada a Objetos aplicada a Powerbuilder

La programación orientada a objetos (OOP) es una nueva forma de enfocar el trabajo de la programación. Los enfoques de la programación han cambiado drásticamente desde la invención de las computadoras para adaptarse a la creciente complejidad de los programas. Por ejemplo cuando se inventaron las computadoras, la programación se realizaba introduciendo, a través de la consola, las instrucciones máquina en binario. Esto funcionaba porque los programas solo tenían unos pocos cientos de instrucciones. Cuando crecieron los programas, se invento el lenguaje ensamblador, de modo que el programador pudiera tratar programas más largos y complejos usando una representación simbólica de las instrucciones máquina.

Los lenguajes de alto nivel se introdujeron para darle al programador más herramientas con las cuales gestionar esta complejidad. El primer lenguaje ampliamente utilizado fue Fortran. Aunque Fortran impresiona en un principio, no es un programa que anime a crear programas claros y fácilmente comprensibles.

En los sesenta nace la programación estructurada - método alentado y reforzado por lenguajes como C y Pascal. Al principio, con los lenguajes estructurados fue posible escribir programas moderadamente complejos de una forma bastante sencilla. Sin embargo, incluso usando programación estructurada, cuando los proyectos alcanzan un cierto tamaño, su complejidad se vuelve demasiado difícil para ser controlada por un programador. Para resolver este problema se desarrollo la OOP.

La OOP toma las mejores ideas de la programación estructurada y las combina con nuevos y poderosos conceptos que animan y alienta a una nueva visión de la tarea de la programación. La OOP permite descomponer fácilmente un problema en subgrupos de partes relacionadas. Así puede traducir estos subgrupos en unidades autocontenidas llamadas *objetos.*

La OOP incorpora elementos que facilitan la realización de interfaces gráficas de usuario. Powerbuilder tiene herramientas que facilitan la OOP denominados Painters.

Objetos y Atributos

Una aplicación Powerbuilder utiliza objetos tales como una ventana, un control, un gráfico, un dibujo, un icono, etc. Los objetos tienen atributos tales como el color, el tamaño, la forma, etc. Los objetos pueden incluir a su vez a otros objetos, por ejemplo, el objeto ventana de una captura de datos puede combinar una ventana principal, diversos objetos de control y un objeto menú. Algunos de los objetos que se pueden utilizar, tales como menús y ventanas, los da Powerbuilder; otros se tienen que crear.

Métodos

Una operación que requiere información sobre un objeto o que lo cambia de alguna manera se denomina método. En Powerbuilder los métodos se crean escribiendo los procedimientos. Por ejemplo, un procedimiento que abre una ventana o un procedimiento que la cierra son métodos de dicha ventana. En Powerbuilder los métodos son los procedimientos asociados con los diferentes objetos.

Eventos

Los procedimientos pueden incluir funciones. Una función es un procedimiento con nombre, que acepta argumentos de entrada y devuelve uno o más valores. Estas funciones pueden ser reutilizadas. Las funciones Powerbuilder son utilizadas en los métodos y pueden estar ligadas a un objeto determinado o ser globales a toda la aplicación.

En una aplicación Powerbuilder es el usuario principalmente quien determina la secuencia de eventos de la misma: abre y cierra ventanas, y provoca eventos, por ejemplo, haciendo clic sobre un botón de abrir o cerrar. La aplicación puede también por si misma provocar eventos. Por ejemplo un programa que sé esta ejecutando puede provocar el evento idle (después de un periodo predefinido de inactividad por parte del usuario).

Con Powerbuilder se puede realizar procedimientos que determinan la respuesta a un evento. Un objeto ventana puede contener un procedimiento que responda al evento open o un procedimiento que responda al evento close. Cuando una aplicación utiliza un objeto, están a su disposición los eventos y los correspondientes procedimientos del objeto.

Mensajes

Cuando una aplicación se ejecuta, los objetos se comunican entre sí por medio de mensajes. Supongamos un objeto ventana que contiene un objeto botón de finalización; cuando la aplicación se ejecuta y se presenta la pantalla, aparece el botón finalizar. Al hacer clic sobre dicho botón, se cierra la ventana que esta abierta. Al hacer el clic sobre el botón, se provoca el evento clicked del

botón finalizar. El procedimiento de este evento envía un mensaje de cierre a la ventana que contiene el botón. La ventana lo recibe provocando el evento close de la ventana, ejecutándose su correspondiente procedimiento.

Clases

Un lenguaje de programación tradicional contiene diferentes tipos de datos predefinidos, tales como integer, real, number, string, y con estos se pueden crear nuevas variables. Estas deben ser de uno de los tipos predefinidos.Para crear una nueva variable denominada i del tipo int: int i. En este ejemplo int es un tipo que esta predefinido por el lenguaje. Un lenguaje de OOP es similar tiene clases en lugar de tipo de datos. Una clase en la OOP es similar a un tipo de datos en la programación tradicional.

Una definición de clase es un conjunto de atributos y métodos combinados para definir un objeto. Se puede tener una clase denominada ventana, la definición incluye atributos y los métodos que operan sobre ella. Los atributos incluyen la posición x, la posición y, la altura, el ancho y el tipo de ventana. Los métodos pueden incluir procedimientos que controlan la operativa de la ventana cuando esta se abre o se cierra. Una clase es un marco para un objeto. En un lenguaje de OOP se crean clases. Una vez creadas estas, se crean sus correspondientes objetos.

Con un lenguaje de programación tradicional se crean nuevos objetos a partir de los tipos de datos predefinidos. En la OOP se crean nuevos objetos a partir de cualquier clase. Una librería de clases es un conjunto de clases incluida en una librería Powerbuilder.

Instancias

Cada objeto pertenece a una clase. Cuando un programa crea un objeto es instanciado el objeto. Una instancia es una manifestación de una clase. Por ejemplo, Coahuila es una instancia de la clase estado. Una ventana denominada w_entrada es una instancia de la clase ventana. Se podría construir un objeto ventana con el window Painters denominada w_principal y archivar el objeto en una librería. En una aplicación en ejecución, el procedimiento

```
open (w_principal)
```

abre al objeto ventana; es decir, instancia el objeto ventana. Esto es una llamada a una función Powerbuilder predefinida denominada open. Esta función trae el objeto ventana a memoria lista para su uso. Se debe instanciar un objeto antes de que pueda responder a eventos o mensajes. Powerbuilder tiene clases y funciones predefinidas. Una de estas clases es el objeto transaction, el cual se utiliza para comunicarse con una base de datos. Este conjunto de sentencias PowerScrip instancia un nuevo objeto transacción:

```
// Definición de un nuevo objeto del tipo "transaction"
        transaction newtrans
// Instancia de un objeto tipo newtrans
        newtrans = create transaction
// Referencia a alguno de los atributos del nuevo objeto
        newtrans.dbms = "Sybase"
        newtrans.logid = "dba"
```

La jerarquía de clases

Powerbuilder viene equipado con muchas clases, (ventanas, controles y menús); las clases están organizadas de manera jerárquica. Se puede ver la jerarquía de clases seleccionando el *browse* class *hierarchy* en el *library painter*.

Se pueden crear clases con los painter de objetos de Powerbuilder:

- √ clases de ventanas con el Windows painter
- √ clases de menús con el menú Painters
- √ clases de cualquier objeto con su correspondiente Painters.

Estas nuevas clases de ventanas expanden la jerarquía de clase:

Window
    w_principal
    w_entrada_datos

Herencia

La clase w_principal es de la clase ventana, y por la clase ventana es padre de la clase w_principal. A la clase ventana se le denomina también superclase de la ventana w_principal.

Con la OOP, una nueva clase hereda todas las características de su padre. Cualquier cambio realizado sobre una clase se propaga descendentemente a toda la jerarquía de clase. Un Painter de un objeto Powerbuilder permite crear un nuevo objeto que será miembro de una determinada clase, por ejemplo, se puede utilizar el *Window Painter* para crear una nueva ventana que sea miembro de la clase w_principal. Se puede utilizar un painter para crear una nueva clase. La nueva clase heredaría todos los atributos y métodos de su padre. Se puede incluir en la clase creada nuevos métodos y atributos, y utilizar los procedimientos tal y como se han heredado, ampliarlos o reescribirlos. Con Powerbuilder solo se puede crear nuevas clases a partir de las clases ventana menú y objeto de usuarios.
Crear un nuevo objeto a partir de la herencia de otro provoca que se cree una nueva clase. La selección en el correspondiente *menu painter*, ventana u objeto de usuario permite crear un nuevo objeto que hereda sus características de una clase existente, creándose a su vez una nueva clase.

Encapsulación y métodos

La encapsulación independiza y protege los datos. Un objeto encapsula sus datos y sus métodos. Por ejemplo el tamaño de una ventana es un atributo encapsulado del objeto ventana. Para conocer el tamaño de una ventana se debe enviar un mensaje a la ventana que pida información sobre su tamaño y solamente el objeto puede modificar sus datos encapsulados. Otros objetos pueden enviar mensajes a un determinado objeto solicitándole cambios, pero solo el propio objeto puede realizar los cambios.

Polimorfismo

Polimorfismo significa que objetos pertenecientes a clases diferentes pueden aceptar el mismo mensaje, por ejemplo se puede enviar el mensaje print a diferentes objetos. El mensaje provocará que cada uno de los objetos imprima algo. Supongamos un pequeño negocio que envía diferentes tipos de paquetes. Los paquetes pequeños van por correo, los paquetes medianos por medio de mensajero, los grandes por camión y los urgentes por avión. El mismo mensaje, enviar, sirve para cada uno de los objetos. El efecto sin embargo, es diferente.

En los lenguajes de programación tradicionales, las funciones que son diferentes deben tener nombres diferentes. Si se crea una función con el nombre de una de que ya existe, se provoca un error al compilar o al ligar el programa. En un lenguaje de programación orientada a objetos, funciones diferentes pueden tener el mismo nombre. Como ejemplo de programación imaginemos una aplicación con diversas ventanas correspondientes a clases diferentes. Cada una de las ventanas presenta diferentes datos a los usuarios. El mismo mensaje, print, puede dirigirse a cualquiera de las ventanas. Cada ventana puede utilizar una función que imprime los datos, independientemente de que cada ventana pertenezca a una clase diferente.

Sobrecarga de operadores.

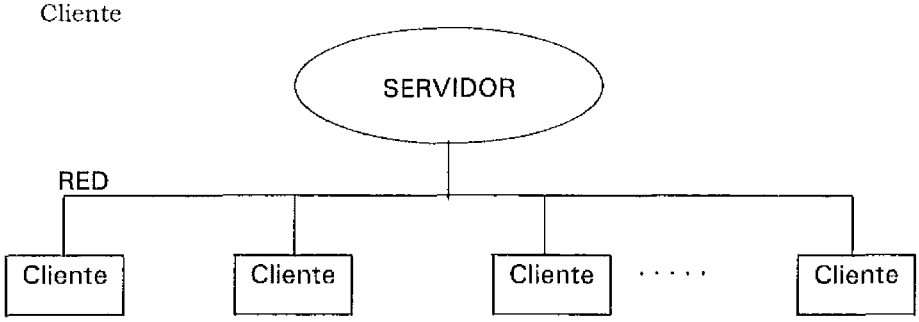
Normalmente un mensaje incluye uno o más argumentos. En Powerbuilder, funciones diferentes pueden tener nombres diferentes. Para que la función opere correctamente, los argumentos deben tener la capacidad de poder tomar tipos diferentes. Esta capacidad para usar la misma interfaz con propósitos diferentes es un ejemplo de sobrecarga de operadores.

1.3 Modelo Cliente - Servidor.

Con el aumento de la eficiencia, con las redes de computadoras más económicas y su costo de personal más bajo, está surgiendo un nuevo modelo de computación. Este es el modelo cliente - servidor.

Un modelo cliente - servidor tiene tres componentes:

√ Servidor
√ Red
√ Cliente



Servidor

El servidor es una computadora que almacena y maneja datos. Por ejemplo, en el servidor se puede ejecutar un sistema manejador de base de datos, es responsable de suministrar cualquier dato requerido por el usuario y obtener los datos en función de la petición realizada y enviarlos al cliente. Es así mismo responsable del control de la concurrencia, del mantenimiento de la integridad de la base de datos y de la administración de las transacciones. El servidor se responsabiliza la seguridad de los datos y del control de accesos. Finalmente, también suministra el espacio de almacenamiento necesario para los datos.
Tradicionalmente, una sola computadora era quién realizaba el procesamiento de datos. Todos los programas se ejecutaban en una única máquina, y todos los usuarios, hasta los más alejados, accedían a la computadora central desde una terminal o mediante algún tipo de emulación de terminal.

Red

Existen muchos tipos de redes, y también muchos fabricantes. Cada uno de ellos funciona con un Hardware y Software estándar. Una red conecta entre sí computadoras autónomas, transfiriendo datos entre ellos; ninguno controla a otro, ni existen relaciones maestro - esclavo.

Una red permite compartir programas, datos y periféricos, con independencia de su ubicación física. De este modo se mejora la operatividad del sistema. Ya que si una máquina no dispone de un recurso, otras máquinas pueden servirlo.

Cliente

Una máquina cliente solicita datos del servidor. La computadora cliente ejecuta un programa localmente, este programa, está preparado para solicitar adecuadamente los datos del servidor. Así mismo es responsable de la administración de la presentación de los datos, además de validar los datos introducidos por el usuario y generar las solicitudes de datos necesarias al servidor.
Un cliente envía una solicitud de datos a un servidor, este devuelve los datos seleccionados en respuesta a la solicitud. El cliente puede realizar cambios en los datos y devolverlos al servidor para incluirlos en la base de datos.
La computadora cliente puede realizar la mayor parte de la manipulación o procesamiento de los datos. Por ejemplo, si el usuario está introduciendo datos, todas las validaciones y correcciones tienen lugar en el cliente. Cuando los datos son correctos, se envían a la correspondiente base de datos del servidor. El cliente puede solicitar también cambios que se realizan directamente en los

datos almacenados en el servidor. Por ejemplo, el cliente puede enviar una solicitud al servidor para modificar o borrar todos los registros que están ligados a un determinado padre.

En el sistema de facturación de una compañía, la computadora central (un servidor de red) mantiene una base de datos con toda la información relativa al sistema Este servidor de base de datos está conectado a una red Varias computadoras personales y estaciones de trabajo están conectados a la misma red. Cada una de las estaciones de trabajo y computadoras personales son un cliente.

La máquina que contiene la base de datos actúa ahora como un servidor de base de datos, las máquinas cliente envían las solicitudes de datos a través de la red, el servidor obtiene los datos solicitados y los devuelve al cliente que los solicitó a través de la red. Una máquina cliente puede manejar todas las interacciones con el usuario, esto descargará la mayor parte del proceso desde el servidor al cliente.

Una aplicación Powerbuilder que se está ejecutando en una máquina cliente puede presentar un determinado formato de pantalla. El usuario puede introducir datos por medio de ese formato y el programa los utiliza para hacer una solicitud de datos al servidor. Cualquier dato utilizado por la aplicación cliente está almacenado en la base de datos del servidor. Si el empleado que introduce los pedidos requiere informaciones sobre ellos, dicha información proviene de la base de datos del servidor y va al correspondiente cliente a través de la red. Cualquier cambio realizado en las órdenes se reenvía de nuevo, utilizando la red, a la base de datos del servidor.

La aplicación cliente puede presentar los datos recuperados, por ejemplo, con un determinado formato de pantalla. También puede transformar los datos y presentarlos por pantalla. La aplicación podría utilizar los datos recuperados del servidor para realizar un informe, u obtener una información resumen y presentarla con un formato normal o utilizando gráficos.

En un entorno cliente servidor, los datos pueden, estar distribuidos en diferentes máquinas de una red. Estas, que actúan como servidores de base de datos, pueden conectarse a una red. Los datos pueden estar distribuidos en diferentes servidores de base de datos. La facturación, de las ventas y los envíos podrían estar en diferentes servidores, conteniendo cada uno de ellos la correspondiente porción de información de la empresa. Esta información puede encontrarse en un servidor de una base de datos, en un servidor y en una base de datos, o en diferentes servidores y diferentes base de datos.

PowerBuilder permite construir aplicaciones que accesen a los datos desde cualquier lugar de la empresa. Múltiples conexiones concurrentes a diversos sistemas de manejadores de base de datos pueden acceder a datos en cualquier máquina. Se puede desarrollar aplicaciones que accesen a datos remotos almacenados en mainframes, mini-computadoras o estaciones de trabajo, o accesar a datos almacenados en base de datos locales.

1.4 ODBC

Cada base de datos incluye su propio lenguaje o interfaz para la programación de aplicaciones. Al seleccionar una base de datos, se obliga a emplear la interfaz definida por el vendedor. Las interfaces propiedad de un vendedor dificultan el acceso a datos de otras bases de datos y la distribución por la red. Aunque existen facilidades para importar y exportar datos, es menos frecuente encontrar utilidades para mover o acceder a otros datos.

Microsoft ha definido un estándar para el acceso a datos almacenados en bases de datos relacionales, no relacionales e incluso formatos no de bases de datos para entornos Windows. El estándar se denomina Apertura a la Conectividad de la base de datos (ODBC) y forma parte de la Arquitectura de Servicios Abiertos en Windows (WOSA), de Microsoft.
El objetivo de ODBC es el acceso transparente desde un cliente a un servidor de datos. Con ODBC es posible acceder a datos almacenados en las principales bases de datos relacionales (como Oracle, DB2, Sybase, Informix, etc.), no relacionales como (Dbase) y archivos de hojas de cálculo, procesadores de texto, etc.

ODBC está basado en la Interfaz Nivel llamada del Grupo de Acceso SQL, CLI y define un único Interface de Programas de Aplicación (API), para el acceso, modificación y presentación simultánea

de tablas de una base de datos. Con ODBC, y con una única API, una aplicación puede tener acceso a diferentes fuentes de datos ubicadas en distintos lugares.

ODBC aísla la aplicación de la red, el manejador de la base de datos o aplicación origen de los datos. Los cambios o modificaciones en estos no afectan por tanto a la aplicación. Al estar disponibles para un elevado número de plataformas y aplicaciones, se facilita también la portabilidad de la aplicación. Tanto Microsoft como los principales vendedores independientes de software distribuyen controladores ODBC para las principales aplicaciones y plataformas.

Con ODBC se garantiza un nivel mínimo de funcionalidad para el acceso a cualquier base de datos soportada. Esta funcionalidad permite conexiones a cualquier base de datos, envío de órdenes a través del API de ODBC o transmisión de SQL empotrado. Con órdenes ODBC se pueden recuperar datos u obtener el resultado de su ejecución.

ODBC también normaliza los mensajes de error, la interfaz de conexión a una fuente de datos remota, los tipos de datos empleados y la sintaxis SQL, y además, independientemente de la base de datos.

Cuando una aplicación tiene que realizar una operación sobre una base de datos o ejecutar un conjunto de sentencias SQL, llama al API ODBC, quien convierte en SQL extendible por el servidor de datos.

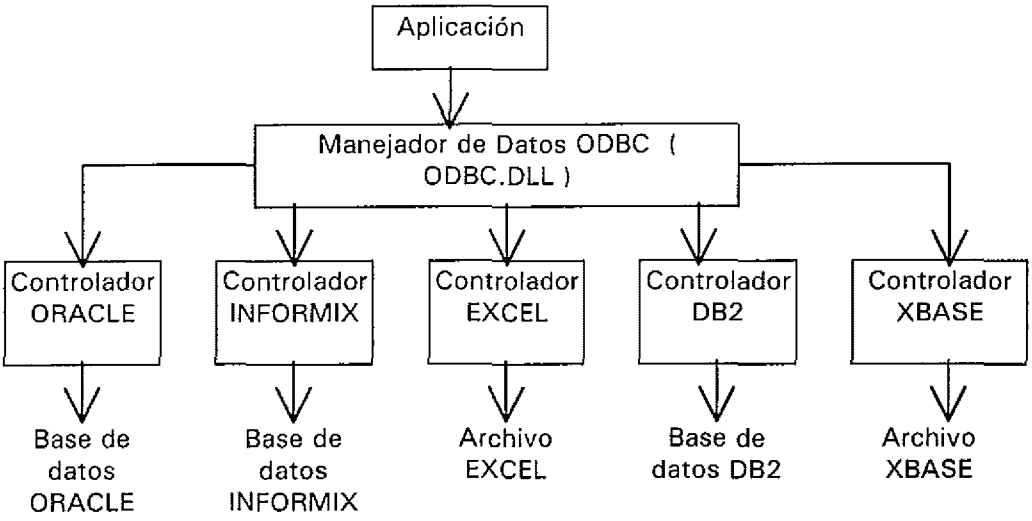
La interfaz ODBC soporta el manejo de cursores deslizables, consultas asíncronas y procesamiento múltiple de lecturas o actualizaciones. La gramática SQL incluye soporte completo de transacciones (sentencias *Rollback* y *commit*), así como de operadores para *outer join* y procedimientos almacenados; la gramática extendida permite el uso de tipos avanzados de datos, como fechas, horas, marcas de tiempo y objetos binarios.

Con ODBC una aplicación puede escoger cualquier fuente de datos, siempre que exista un controlador ODBC para la misma; es posible acceder simultáneamente a varias fuentes de datos. ODBC proporciona también todas las facilidades necesarias en sistemas transaccionales de procesamiento on line en misiones críticas o en sistemas de ayuda a la toma de decisiones.

El Manejador de Datos

El acceso transparente a los datos vía ODBC tiene lugar mediante el manejador de datos y los controladores específicos de cada base de datos. Una aplicación cliente llama al API ODBC, o lo que es lo mismo, al manejador de datos, quien se encarga de cualquier intercambio de información entre la aplicación y la fuente de datos.

El manejador de datos procesa la llamada, valida su sintaxis, la convierte a la sintaxis de la base de datos destino y carga y descarga los controladores ODBC conforme sean necesarios para ejecutar dicha llamada.
El manejador de datos ofrece servicios a las aplicaciones. En Windows, los APIs son DLL (*Dynamic Link Library*), biblioteca de enlaces dinámicos.



ODBC y Powerbuilder

Existe una gran cantidad de vendedores que distribuyen controladores ODBC. Cualquier aplicación que utilice ODBC, como Powerbuilder, necesita el controlador correspondiente para conectarse a la fuente de datos. Cuando Powerbuilder se le solicita una conexión vía ODBC, ha de localizar el correspondiente controlador ODBC, un archivo DLL para acceder a la base de datos.

Dado que Powerbuilder sabe como establecer una conexión vía ODBC a una base de datos y conexiones a otras fuentes de datos distintas sólo precisa corregir el controlador. Powerbuilder siempre interactúa según la interfaz ODBC, al igual que cada controlador para cada base de datos. Existen controladores ODBC para las principales bases de datos para sistemas de gestión de archivos como XBASE y para otras aplicaciones como hojas de cálculo.

La conexión ODBC, es totalmente transparente a la aplicación. Sólo se necesita especificar el origen de los datos, y si existe el correspondiente controlador ODBC, la conexión tiene lugar automáticamente.

Acceso a una base de datos

Una aplicación Powerbuilder en ejecución puede establecer una conexión con una base de datos. Para establecer esta conexión se debe escribir un procedimiento, el cual se asocia normalmente al evento abrir de la aplicación. También se puede conectar a una base de datos desde el entorno de desarrollo de Powerbuilder. La conexión ocurre cuando se crea una objeto DataWindow, como se muestra en un capítulo posterior.

Para conectar a una base de datos desde el entorno de desarrollo, primero se debe establecer las preferencias para esa conexión. Hay tres maneras de modificar estas preferencias. En primer lugar, se puede editar el archivo pb.ini, el archivo perfil de Powerbuilder, que no es una buena idea ya que requiere parar y arrancar Powerbuilder para que los cambios tengan efecto. En segundo lugar, se puede llamar a la ventana de perfiles de base de datos. Y en tercer lugar, se pueden cambiar las preferencias con el *Preference Painter.*

Después de arrancar el *Preference Painter* se proporciona una gran variedad de opciones para cambiar el entorno operativo de Powerbuilder. Haga clic sobre el icono de la base de datos para seleccionar las preferencias de conexión de este.
Los valores de las preferencias de la base de datos se utilizan al establecer una conexión a una base de datos. Estos valores se usan cada vez que se conecte a una base de datos con el *DataWindow Painter* o el *Database Painter.*

Las preferencias de la base de datos identifican la conexión por defecto a la base de datos dentro del entorno de desarrollo. Los valores de cada una de las preferencias varían según los sistemas de base de datos. Las preferencias para una conexión a una base de datos de Oracle se diferencian enormemente a las de una conexión de Sybase o DB2.

A continuación se incluye una lista de cada una de las preferencias que pueden establecer con el *Preference Painter*.

Variable	Descripción
Autocommit	FALSE para que tenga lugar un proceso normal de transacción recuperable en el momento del CONNECT. TRUE para inhabilitar el proceso normal de transacción recuperable. El valor por defecto es FALSE.
AutoQuote	Especifica si se añade automáticamente los símbolos de comillas simples a una cadena en la sintaxis de SQL que se genera con Powerbuilder. Cuando la opción es 1 (valor por defecto), las comillas se aplican automáticamente a una cadena. Cuando la opción es 0, no se aplican las comillas.
Columns	El número de columnas de tabla o vista que se muestran cuando se expande una tabla. Si el número de columnas de la tabla excede el número especificado, se muestra una barra de desplazamiento vertical para que pueda desplazarse y mostrar todas las columnas.
Database	El nombre de la base de datos por defecto.
DatabasePassword	La palabra clave de la base de datos que se utiliza.
DBMS	El nombre de su distribuidor por defecto del sistema de gestión de base de datos (Uno de los distribuidores especificado en Vendors).
DBParm	Dependiente de la base de datos.
ForeignkeyLineColor	El valor RGB del color que quiere utilizar para la línea entre los símbolos de llaves foráneas y la tabla. Por defecto es azul.
IndexKeyLineColor	El valor de RGB del color que quiere utilizar para la línea entre el símbolo de índice y la tabla. Por defecto es rojo.
PrimaryKeyLineColor	El valor de RGB del color que quiere utilizar para la línea entre el símbolo de la llave primaria y la tabla. Por defecto es verde.
Prompt	1 para solicitar información de la base de datos cuando se conecta a una base de datos, 0 (por defecto) para inhabilitar la solicitud. Esta opción se establece normalmente en la ventana Database Profile Setup.
Lock	El nivel de aislamiento. Dependiente de la base de datos.
LogId	Su identificador de conexión.
LogPassword	Su palabra clave.
NoCatalog	Acceso al catálogo. No para crear automáticamente las tablas del repositorio la primera vez que se conecte a la base de datos utilizando PowerBuilder. Yes para permitir sentencias DDL y DML. Si las tablas del repositorio existen, Powerbuilder no crea las tablas del repositorio. Si las tablas existen, Powerbuilder no referenciará las tablas. Se usarán siempre los valores por defecto. Por defecto es NO.
ReadOnly	Acceso a la base de datos. Si se pone esta variable a 0, Powerbuilder creará automáticamente las tablas del repositorio la primera vez que se conecte a la base de datos utilizando Powerbuilder. Cuando se pone esta variable a 1 y no existen las tablas del repositorio, Powerbuilder no las crea; se usan los valores por defecto. Si existen las tablas, Powerbuilder las utilizará pero no permitirá a los usuarios modificar su información.
ServerName	El nombre del Servidor.
ShowIndexKeys	0 para no mostrar las claves índice y 1 para mostrarlas.
ShowRefInt	Cuando esta opción es 0, no se presenta la integridad referencial en el Database Painter. Cuando la opción es 1, se presenta la integridad referencial.
StayConnect	Opción para arrancar la transacción solamente cuando se sale de Powerbuilder. Por defecto es 1, permanecer conectado; 0 es cerrar la transacción cuando se abandone le Painter.

TableDir	Opción para suprimir el mostrar la lista de tablas. Si TableDir es 1(Por defecto), Powerbuilder lista automáticamente las tablas de la base de datos cuando abre el Database Painter. Si tableDir es 0, Powerbuilder no muestra la lista de tablas cuando abre el Painter. Para abrir una tabla de la base de datos cuando tableDir es 0 haga clic en el icono Tables. Se muestra una caja de diálogo, seleccione el nombre de la tabla de la base de datos que quiere abrir y haga clic en Open.
TableSpace	Dependiente de la base de datos.
TerminatorCharacter	El carácter utilizado para terminar una sentencia SQL.
UserId	Su ID de usuario.
Vendedor	El nombre de su distribuidor de DBMS. Especificar cada DBMS al que tiene acceso; introduce primero el distribuidor por defecto y separar los distribuidores con comas.

Los valores que establezca con el *Preference Painter* residen en el archivo pb.ini. El archivo perfil está en el directorio en el que ha instalado Powerbuilder.
Los archivos perfil se dividen en varias secciones. Las preferencias que se cambien con el *Preference Painter* residirán en el área correspondiente del archivo perfil.

Los conceptos aquí presentados permitirán al lector tener un panorama general del entorno de una aplicación PowerBuilder. La creación de una aplicación óptima no solo implica un buen desarrollo por parte del programador, también implica un conocimiento general otros conceptos, como lo es ODBC, la estructura cliente servidor, etc.

Capítulo 2

PowerBuilder y SQL.

Objetivos
Ya que PowerBuilder es un lenguaje principalmente para aplicaciones cliente servidor, es necesario conocer los conceptos básicos de base de datos,esto con la finalidad de que el lector puede hacer uso de las sentencias SQL de una forma óptima.

2.1 Sistemas de Gestión de Base de Datos Relacional.

Los sistemas de Gestión de Base de Datos Relacionales (RDBMS) son el conjunto de programas que pueden soportar tanto a un mainframe, como a una PC. Estos sistemas incorporan toda la teoría de las bases de datos relacionales. La siguiente información se aplica a DB2 sobre un Mainframe de IBM, a Informix, Sybase u Oracle sobre miniordenadores, o a Access o Watcom sobre su PC.

Todos los sistemas de base de datos relacionales utilizan un lenguaje de consultas estructuradas (SQL).
RDBMS
A continuación se presentan algunos de los princiales conceptos relacionados con este tema:
Es el Software encargado de todas las transacciones con la base de datos. Una base de datos relacional es una o un conjunto de tablas que contienen datos. Un RDBMS automatiza la creación y manipulación de las tablas, suministrando facilidades para crear y eliminar tablas, permitiendo asimismo el almacenamiento, recuperación y cambio de los datos mantenidos en las tablas.

Un Sistema de Gestión de Bases de Datos incluye facilidades para:

- √ Creación de una base de datos.
- √ Eliminación de una base de datos existente.
- √ Agregado de tablas nuevas y vacías a una base de datos.
- √ Borrado de tablas existentes de una base de datos.
- √ Acceso a los datos de las tablas.
- √ Compartición de datos.
- √ Agragado de datos a las tablas.
- √ Borrado de datos de las tablas.
- √ Modificaciones a los datos de las tablas.
- √ Copias de datos de una tabla a otra.
- √ Movimiento de datos de una tabla a otra.

Utilizar un RDBMS es mucho más fácil, con las herramientas suministradas por el RDBMS, ya que es más fácil escribir una instrucción SQL.

Las bases de datos pueden compartir los datos. Los usuarios pueden acceder, a través de una red, a bases de datos ubicadas en un mainframe, minicomputadora, PC o estación de trabajo.

Un RDBMS tiene ventajas importantes, entre las que resaltan:

- √ Permite múltiples usuarios.
- √ Dirige el acceso concurrente a los datos compartidos.
- √ Coordina el compartir los procesos.
- √ Garantiza la seguridad de los datos.

Una única base de datos mantiene:

- √ Los cambios que se produzcan en los datos.
- √ Los cambios que se produzcan en los datos en una única localización.
- √ Los conjuntos de datos consistentes.

Los sistemas de gestión de bases de datos actuales permiten un almacenamiento y una recuperación fácil y eficiente, comparado con manejadores anteriores. Como los datos crecen, la velocidad de acceso, correspondientemente, llega a ser cada vez más importante, y la velocidad de acceso a los datos disminuye.

Cualquier RDBMS que se esté comercializando suministra un rápido acceso a los datos, incluso tratándose de grandes volúmenes, ya que utiliza mecanismos bastante sofisticados de recuperación de datos.
Una base de datos centralizada para un negocio:

- √ Mantiene todos los datos esenciales del negocio.
- √ Proporciona un modelo de todas las operaciones de la organización.
- √ Compone los datos para la totalidad de la organización, permitiendo a cada una de las partes actuar sobre la misma información.

Lenguajes de Datos

Un lenguaje de acceso a los datos (por ejemplo, SQL) o sublenguaje, accesa la base de datos, suministrando facilidades para:

- √ Definición de datos (DDL).
- √ Manipulación de Datos (DML).
- √ Control de datos (DCL).

Las instrucciones de definición de datos:

- √ Crean la base de datos.
- √ Crean las tablas de la base de datos.
- √ Especifican que datos pueden ser almacenados en las tablas.

Las instrucciones de manipulación de datos cambian los datos, mediante:

- √ Adición.
- √ Actualización.
- √ Modificación.
- √ Borrado.

Las instrucciones de control de datos:

- √ Gestiona el acceso a la base de datos.
- √ Posibilita la recuperación de los datos.
- √ Soporta el control de la concurrencia.

SQL proporciona un lenguaje estándar para realizar estas operaciones en la base de datos. SQL puede servir también como un sublenguaje ya que sus instrucciones pueden ser parte de lenguaje como C o Powerbuilder. Por ejemplo:

```
Select id_nombre From clientes
```

O bien, se puede incluir dentro de un programa en Powerbuilder

```
Select id_nombre
  into :ls_nombre
 From clientes
```

El control de accesos a la base de datos.

Un sistema de gestión de bases de datos relacionales necesita proteger los datos de accesos no autorizados. El RDBMS tiene:

√ Permisos de usuarios y palabras llave para controlar los accesos.
√ Órdenes para restringir el acceso a la base de datos o a grupos o personas individuales autorizadas.

El ciclo de vida de la Base de Datos.

Cualquier proyecto de base de datos incluye:

√ Una descripción simple del problema.
√ Un resumen de la solución.

Una construcción de la base de datos que consta de 4 fases:

1. - Planificación.
2. – Diseño.
3. – Implementación.
4. – Refinamiento.

2.2 Arquitectura de la Base de Datos Relacional.

Un sistema de Gestión de bases de datos relacionales consta, a nivel arquitectura, de tres vistas:

√ Vista conceptual.
√ Vista externa.
√ Vista interna.

Una única base de datos tiene:

√ Sólo una vista conceptual.
√ Varias vistas externas.
√ Solo una vista interna.

Vista Conceptual

La vista conceptual de la base de datos es la estructura lógica completa de la base de datos tal y como la ve la organización. Todos los usuarios de la base de datos comparten esta vista común, representa, en la base de datos, la forma en la que la empresa ve los datos, incluye todos los detalles sobre la estructura contenido de la base de datos, define los datos y sus relaciones lógicas, sobrepasa una vista particular de los usuarios sobre los datos. Así se evitan las restricciones del lenguaje o del sistema operativo. La vista conceptual se presenta como un esquema de nivel conceptual.

El esquema define cada una de las tablas y cada uno de los tipos de elementos de datos encontrados en la base de datos. Este esquema:

√ Crea las tablas.
√ Crea cada una de las columnas de las tablas.
√ Define los tipos de datos.
√ Define si los campos permiten nulos o no.

El modelo Entidad - Relación es una herramienta útil para el diseño de las bases de datos, y es útil

√ Cuando está diseñando la estructura de la base de datos.
√ Cuando está diseñando la vista conceptual de la base de datos.

Vista Externa

Las vistas externas son vistas individuales de la base de datos. Cada usuario puede tener diferentes vistas de la misma base de datos.

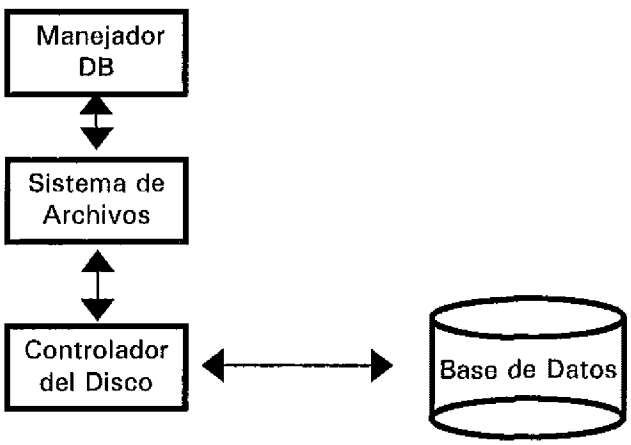
Ejemplo: Un empleado mantiene información sobre los clientes, añadiendo nuevos clientes, cambiando la información de estos o eliminando a tales. Este empleado conoce únicamente de la base de datos la parte referente a los clientes. Por otro lado, en otro departamento, los pedidos son mantenidos por otro empleado. Este empleado sabe de clientes y de pedidos; o bien se puede decir que tiene una vista diferente de la misma base de datos.

Puede existir varias vistas externas, pero sólo una única vista conceptual.
Una vista externa corresponde a una vista del usuario, posiblemente incorporando sólo parte del esquema.

Diferentes lenguajes de programación pueden también representar diferentes vistas externas.

Vista Interna

Solamente existe una vista interna. Cuenta con los siguientes componentes:
Componentes de la vista interna



La vista interna aísla al usuario de los métodos de almacenamiento y de las estructuras usadas por el sistema operativo. La vista interna gestiona las representaciones y el acceso a los datos. El usuario no necesita preocuparse de almacenar o recuperar un número o una cadena de caracteres, ya que el manejador lo realiza.

Manejador de Base de Datos

Una base de datos es un conjunto de datos manejados por un RDBMS. El manejador de la base de datos proporciona al usuario el acceso a esta base de datos.

El manejador de la base de datos:
√ Controla la interfaz entre el usuario y los trabajos internos de la computadora.
√ Supervisa las peticiones hechas en el nivel de la vista conceptual.
√ Acepta las peticiones de alto nivel.
√ Opera sobre tablas y filas de datos dentro de las tablas.
√ Traduce las peticiones para cualquier operación en las tablas y filas a peticiones para operaciones en ficheros y registros.

En los sistemas de gestión de bases de datos relacionales, las peticiones al manejador de la base de datos llegan como sentencias SQL.

En los entornos cliente - servidor el sistema *front-end* [1] hace uso para sentencias SQL y en el manejador de la base de datos *back-end*[2].

Por ejemplo, el manejador de la base de datos:

- √ Accesa a la tabla.
- √ Recupera datos de una tabla.
- √ Borra filas de una tabla.
- √ Crea una tabla nueva.
- √ Elimina tablas no deseadas.

El manejador de la base de datos traduce las peticiones de las tablas solicitadas a órdenes para el administrador de archivos.

El administrador de archivos:

- √ Recibe peticiones para operar en los archivos y procesa estas peticiones.
- √ Utiliza cualquier sistema operativo host, tal como DOS, OS/2, el de MAC o Unix. El RDBMS utiliza a menudo este administrador de archivos suministrado con el sistema operativo. En algunos casos, para mejorar la operativa, la compañía vendedora suministrará su propio sistema de archivos para reemplazar el sistema operativo nativo. Esto es más común en los entornos Unix más grandes.
- √ Traduce las peticiones de manipulación de archivos a órdenes que entienda el controlador de disco.
- √ El administrador de archivos gestiona grupos de archivos y da respuesta a las peticiones para realizar las operaciones en los archivos.
- √ Crea un nuevo archivo.
- √ Borra un determinado archivo.
- √ Cambia el nombre de un archivo.
- √ Mueve un archivo a una nueva localización.
- √ Recupera un registro de un determinado archivo.

2.3 Estructura de la base de datos relacional.

Un sistema de gestión de base de datos relacional almacena datos en las tablas. Estas tablas tienen filas de datos. Cada una de estas filas contiene múltiples elementos de datos.

Las partes de un sistema de gestión de base de datos relacional se parecen a las estructuras de datos usadas en los sistemas más antiguos. Las tablas son similares a los archivos, y las filas a los registros.

Los nombre formales de estas partes son:

- √ Una tabla es una relación.
- √ Una fila es una tupla.
- √ Un elemento de datos es un atributo.

Una base de datos relacional es un conjunto de tablas que contienen datos. Es relacional, ya que asocia datos de una tabla con datos de otras. Permite combinar datos de dos tablas para dar respuesta a una misma cuestión.

Valores Atómicos de los datos.

[1] Front-end. Término que se utiliza para definir los procesos que se ejecutan en el cliente.
[2] Back-end. Término que indica que los procesos se realizan en el servidor

En una base de datos relacional, los elementos de datos más pequeños disponibles son un valor único, como un número de pedido o un código postal. Estos valores de datos individuales son atómicos[3], ya que es la unidad de datos disponible más pequeña.

Un valor atómico de datos es la unidad de datos más pequeña que puede ser almacenada o recuperada de una base de datos relacional. Por ejemplo no hay forma de almacenar sólo la parte de un número de cliente o de recuperar únicamente una parte de este número.

Pero recorrer la base de datos con parte de un valor atómico sí es posible. Se pueden buscar todos aquellos nombres de clientes que comiencen con la letra A. Cada uno de los elementos que encaje con el criterio de búsqueda aparecerá completo, y no en parte. Su búsqueda por A obtendría sucesivamente todos los nombres de clientes que comienzan por A:

Dominios

Un dominio es un conjunto formado por todos los posibles valores atómicos de elementos de datos. Por ejemplo, si los números de clientes pueden tener valores entre 1 y 99999, el dominio de los números de clientes tiene 99999 valores - todos los números enteros desde 1 a 99999 -. Todos estos posibles números de clientes existen en el dominio de los números de clientes
No todos los valores del dominio aparecen necesariamente en la base de datos.

Tablas

Una tabla tiene una cabecera y un cuerpo, como se muestra:

CABECERA	Nombre_cliente	id_cliente	Teléfono
CUERPO	Patty Vega Nuñez	00001	7-51-77-01
	Alejandro Villarruel Cortés	00002	5-50-78-03
	Ma. De Jesús Cortés	00003	2-77-60-03
	Bernadino Perea	00004	7-33-74-30

Atributos
El encabezamiento de una relación consta de uno o más atributos. Los atributos dan nombre al tipo de datos que estarán contenidos en la tabla. Cada uno de los atributos da nombre al dominio que lo sustenta.

Nombre_cliente	id_cliente	Teléfono
Patty Vega Nuñez	00001	7-51-77-01
Alejandro Villarruel Cortés	00002	5-50-78-03
Ma. De Jesús Cortés	00003	2-77-60-03
Bernadino Perea	00004	7-33-74-30

Diferentes atributos pueden ser sinónimos refiriéndose al mismo dominio. Conviene no utilizar sinónimos en las aplicaciones de bases de datos. Cuando se usan datos en diferentes tablas correspondientes al mismo dominio, conviene utilizar el mismo atributo.

Atributos de diferentes dominios

Normalmente las tablas soportan diferentes tipos de datos, tales como nombres de clientes, identificador del cliente y número de teléfono. Cada uno de estos procede de un dominio diferente. Los atributos muestran los tipos de datos que la tabla soporta.

[3] Atómico que proviene del griego antiguo, indivisible.

Atributos del mismo dominio

En la misma tabla pueden existir múltiples atributos sobre el mismo dominio. Ejemplo: una tabla de pedidos podría tener un código de postal de embarque y un código de postal de factura.

El grado de una relación es el número de atributos encontrados en la cabecera.

Una tabla que tiene un único atributo se dice que es unaria o que tiene un grado uno. Una tabla que tiene dos atributos se dice que es binaria o que tiene grado dos. Una tabla que tiene 3 atributos se dice que es terciaria. Una tabla que tiene más de 3 atributos se denomina n-aria.

Tuplas. Una tabla contiene dos partes una cabecera y un cuerpo: la cabecera contiene atributos, el cuerpo de la tabla tiene tuplas o filas. Cada una de la tuplas está formada por parejas de atributos y valores de datos.

Cada pareja en una tupla contiene un atributo que corresponde a un atributo de la cabecera. Cada uno de los atributos en una tupla está asociado con un valor de datos. El valor asociado es un valor de datos perteneciente al dominio correspondiente, o bien un valor nulo indicando que todavía no se ha seleccionado algún valor.

No hay que olvidar que, para facilitar la presentación, los atributos aparecen en la cabecera, y no en cada una de las tuplas. Asimismo, los valores de datos se encuentran alineados bajo el atributo de cabecera. Aunque mostrar una relación de esta forma es más fácil, las filas constan de tuplas, es decir, de parejas atributos datos.

Cada una de las tuplas deben tener el mismo número de atributos que la cabecera. Todas las tuplas tiene el mismo número de atributos.

Las relaciones aparecen normalmente como tablas adecuadamente ordenadas. Un atributo encabeza cada columna. Una columna identifica en la tabla los mismos datos.

Una pareja atributo - dato puede aparecer en la fila en cualquier orden; las tuplas no es necesario que estén ordenadas.

Una tabla cambia a medida que pasa el tiempo, con filas que son añadidas o borradas. Debido a que las filas se añaden o se borran, el orden de las filas cambia. Una base de datos relacional no mantiene el orden de las filas de una tabla cada vez que la tabla cambia. No se puede mantener el orden de las filas durante estos cambios.

Valores de Datos Nulos

El valor nulo es un valor específico que indica que ningún dato determinado se encuentra presente en el dominio correspondiente. Esto es distinto a un valor vacío. Para dar respuesta a una consulta una tabla en la cual no se ha introducido ningún valor, es posible que aparezcan valores indeterminados. El valor nulo indica que ningún elemento del dominio se encuentra disponible.

El valor nulo produce resultados predecibles en comparaciones lógicas. Un valor no especificado producirá resultados desconocidos y no predecibles.
No hay ningún valor estándar que gobierne el uso de los valores nulos ni de la lógica de tres valores.

Cardinalidad.

La cardinalidad de una relación es el número de tuplas de la relación

Las parejas atributo - datos forman una fila en una tabla. Cada uno de los atributos sólo puede tener un valor de datos asociado. Una estructura de datos con múltiples elementos de datos asociados con un atributo no es una relación.

El cuerpo de la tabla contiene atributos. Los atributos se emparejan con los elementos de dato.

Llaves

Cada fila de la tabla es única. Esto es debido a que no se permite que existan filas duplicadas en una tabla.

Debido a que no hay filas duplicadas en las tablas, cualquier fila tiene una única identificación para permitir la selección.

Por ejemplo, en la tabla anterior que contiene la información de pedidos, el atributo num_pedido es una llave de esta relación. Es llave ya, que identifica unívocamente cada fila de la tabla.

Si una tabla tiene filas duplicadas, no hay forma de asegurar la existencia de una llave única. Esta es la razón por la que las relaciones no deberían tener filas duplicadas.

Uno o más atributos forman una llave. Un atributo es posible que identifique unívocamente cualquier fila en una relación, tal como un número de pedido. Si este atributo puede identificar unívocamente y seleccionar cualquier fila, este atributo puede servir como llave.

Llaves Compuestas

Algunas veces, un atributo no es suficiente para seleccionar una única fila en una tabla. Si es así, varios atributos deben combinarse para formar una llave. Una llave formada, por varios atributos es una llave compuesta.

Ejemplo: Un pedido puede incluir más de un tipo de artículo. Ya que un único pedido puede ir con más de un tipo de producto, el número de pedido no es único.

num_pedido	id_artículo
3	125
3	126
2	100

El mismo elemento de artículo puede aparecer en diferentes pedidos, así que el número de artículo no es único. El número de pedido no es único, por lo que no es llave en esta tabla. Una llave compuesta formada por el número de pedido y el número de artículo puede identificar y seleccionar unívocamente cualquier fila de la tabla.

Llaves Primarias.

En una tabla puede haber más de una llave. En este caso, cada llave por separado es una llave Primaria. Una relación debe tener al menos una llave Primaria. Puede existir más de una llave Primaria y estas pueden ser llaves individuales, compuestas o una combinación de ambas y deben ser las mínimas.

Las llaves proporcionan, en el modelo relacional, el único modo de direccionamiento. Solamente una llave selecciona una única fila de una tabla. La única forma de encontrar una tupla es conocer el nombre de su tabla y el valor de la llave primaria para esa fila.

Llaves Foráneas.

Las llaves permiten el enlace de la información de diferentes tablas. La llave forámea en una tabla puede localizar datos en otra tabla.

Ejemplo: Considera las dos tablas siguientes:
Artículos

Id_pedido	id_artículo
3	1024
3	1025
1	1026
1	1027

Pedidos

Id_pedido	id_cliente	fec_embarque
1	32	28/05/1997
2	17	19/04/1997
3	48	01/04/1997

La tabla pedidos muestra los pedidos de los clientes. La llave primaria es id_pedido. La tabla de artículos muestra los elementos de artículos incluidos en cualquier pedido. La llave primaria es la llave compuesta formada por id_pedido y id_artículo.

Cuando un cliente realiza un pedido, la tabla de artículos se incrementa en una o más filas para ese pedido. Cada uno de los elementos de artículos que aparecen en el pedido introduce una fila. La persona que introduce la fila para elementos de artículos añade en ella el número de pedido.

Esto simplifica el localizar todos los elementos de artículos para un único pedido. Recorriendo la tabla de artículo para un número de pedido se encuentran todos los elementos de artículo para ese pedido.

El número de pedido es la llave primaria para la tabla de pedidos. Es una llave foránea para la tabla de artículo. Una llave foránea es una llave primaria de una tabla que se utiliza para acceder a datos de una tabla diferente.

Las asociaciones entre llave primaria y llave foránea son el nexo de unión que mantiene las bases de datos relacionales.

A diferencia de los sistemas de bases de datos anteriores, en las bases de datos relacionales no existe ninguna asociación física entre las tablas. Las tablas se relacionan únicamente a través de las llaves primarias y llaves foráneas lo que permiten a los datos ser combinados desde múltiples tablas.

Nota: Una llave foránea no es necesariamente parte de la llave primaria de la relación. En el ejemplo anterior, la llave foránea id_pedido sucede que es parte de la llave primaria

Reglas de Integridad

El modelo relacional suministra dos reglas que ayudan a asegurar la integridad de los datos. Estas son la integridad de entidad y la integridad referencial. La regla de integridad de entidad establece que una llave primaria no puede tener valores nulos. Si una llave no es única, no puede ser llave primaria. Las llaves foráneas pueden contener valores nulos. No hay ningún registro en el modelo relacional que obligue a que una llave sea única.

La regla de integridad referencial considera la asociación entre las llaves primarias y la llave foránea. La integridad referencial establece que una llave foránea debe existir en una tabla que contenga la llave primaria.

Ejemplo: Aquí se presenta otra vez la pequeña base de datos de pedidos y artículos. La llave foránea en la tabla artículos es el número de pedido.

id_pedido	Artículo
3	1003
3	3233
3	4434
2	5545
1	4543
1	2454
1	1904

id_pedido	id_cliente	Fec_embarque
1	01	28/05/1997
2	02	19/04/1997
3	03	01/04/1997

El mantener la integridad referencial en esta base de datos requiere que en la tabla de pedidos haya un identificador para cada pedido encontrado en la tabla de artículos.

Si se borra la primera fila de la tabla de pedidos, se borraría el número de pedido. Se perdería la integridad referencial, ya que ahora habría elementos de artículos en la tabla de artículos para un pedido que no existe en la tabla de pedidos. Para mantener la integridad referencial se debería eliminar todos los elementos de artículo para el número de pedido que haya sido suprimida.

2.4 SQL

El SQL es el lenguaje estándar en la industria para la manipulación de bases de datos relacionales, ya que su facilidad de uso ha permitido que muchas herramientas lo utilicen como su lenguaje.

E.F. Codd, un investigador de IBM, describió el modelo relacional para los sistemas de gestión de Base de Datos en el artículo "A Relational Model Of Data For Large Shared Data Banks", Communications of the ACM (junio 1970). Este artículo guió los proyectos de otros muchos investigadores.

El proyecto SYSTEM/R de IBM comenzó en 1974, y produjo un sistema de base de datos relacionales en 1978.

El grupo de desarrollo de SYSTEM/R creó un lenguaje denominado Sequel (un acrónimo de Structured English Query Language).

IBM comercializó la primera base de datos relacional, SQL/DS, en 1982, IBM emitió otro sistema de gestión de bases de datos relacionales, DB2, 1985, DB2 también incluye SQL.

SQL es el lenguaje estándar para acceder a las bases de datos relacionales. El American National Institute (ANSI) aprobó formalmente SQL, en 1986 como ANSI X3.135.

Los grupos de desarrollo extremos a IBM también produjeron sistemas de gestión de bases de datos que se comercializaron. Un ejemplo, es el proyecto de Ingress de la Universidad de Berkley (California)

Definición de datos

Las sentencias de definición de datos crean la base de datos, crean las tablas en la base de datos y crean las columnas de las tablas.

El entorno de la base de datos.

SQL actúa sobre el entorno correspondiente a la base de datos que se trate. Esta base de datos contiene Tablas, y estas contienen datos. Dos tablas en la misma base de datos no pueden tener el mismo nombre.

Tipos de datos

No existe ningún estándar que determine los tipos de datos que pueden ser almacenados en una base de datos relacional. Diferentes sistemas de base de datos relacionales de diferentes fabricantes pueden almacenar diferentes tipos de datos. Los tipos de datos disponibles más comunes aparecen en la tabla siguiente:

Character (longitud)	Una cadena de uno o más caracteres.
Varchar(longitud)	Una cadena de caracteres de longitud variable
Numeric [(precision [,escala])]]	un número real, usualmente un sinónimo para un float

Dec [imal [(precision [,escala])])]	un número decimal
Number [(precision [,escala]))]	lo mismo que numeric
Integer	Entero
Smallint	un entero pequeño
Float precision	un número real
Double	un número real de doble precision
date [time]	Fecha con hora

Una longitud o precisión debe ser mayor que cero. Los valores por defecto para la longitud y escala dependen del fabricante. El valor por defecto de la precisión es dependiente de la implementación.

La precisión por defecto de los valores smallint e integer depende también de la implantación. Un integer siempre puede mantener un valor positivo mayor o negativo menor que un smallint. Un número double precision también es capaz de mantener valores positivos mayores y negativos más pequeños que un float.

SQL tiene una sintaxis bien específica y concreta. La sintaxis tiene varios elementos, incluyendo palabras reservadas, operadores e identificadores. Estos se combinan para hacer las sentencias de SQL. La mayoría son caracteres. Todos los elementos de SQL se derivan de un conjunto de caracteres que incluyen las letras minúsculas de la a la z y las letras mayúsculas de la A a la Z. También incluye a los dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, y un marcador de nuevas líneas y algunos caracteres especiales. Los caracteres especiales pueden ser:

% _ , ( ) <> . : = + - * /

y un carácter para el espacio en blanco. El conjunto de caracteres completos y la ordenación para el conjunto de caracteres dependen de la implementación.

Literales

Hay dos tipos de literales: cadenas y números. Una cadena es una serie de caracteres encerrados entre comillas

Tokens

Los tokens son las unidades sintácticas básicas del lenguaje SQL. Los tokens incluyen delimitadores y no delimitadores.

Un delimitador es una cadena de uno de los siguientes símbolos especiales :

, ( ) < > . : = + - * / <> <= >=

Los no delimitadores son literales numéricos, o bien un identificador, o bien una palabra reservada.

Cualquier número de separadores puede seguir un token. Un separador es un espacio, un marcador de nuevas líneas (intro) o un comentario. Un token separador o delimitador o delimitador debe seguir a cada uno de los token no delimitadores.

El usuario elige un identificador para algunos objetos en la base de datos, tales como un nombre de columna o un nombre de tabla. Un identificador es una serie de hasta 18 caracteres. El primer carácter de un identificador debe ser una letra

Palabras reservadas en Powerbuilder en SQL.

all	Delete	into	references
and	desc	is	rollback
any	double	language	section
asc	end	like	select
authorization	escape	max	setavg
access	exec	min	smallint
begin	exists	module	some
between	fetch	not	sql
by	float	null	sqlcode
character	foreig	of	sum
check	fortran	on	table
close	found	open	to
cobol	from	option	union
commit	go	or	unique
continue	goto	order	update
count	grant	pascal	user
create	group	pll	values
current	having	precision	view
cursor	In	primary	whenever
dec	Indicator	privliges	where
decimal	Insert	procedure	with
declare	Int	public	work
default	Integer	real	

Sintaxis SQL (Utilizando SYBASE SYSTEM 11)

Cada una de las sentencias de SQL tiene típicamente más de una posible variante. Una descripción de cualquier sentencia SQL.
Las sentencias SQL pueden clasificarse en diferentes conceptos:

- √ Manipulación de Objetos en la base de datos (Conserándo a objetos como tablas, indices, vistas triggers, stored procedure etc.).
- √ Consultas.
- √ Inserción de Filas.
- √ Borrado de Filas.
- √ Actualización de Filas.
- √ Seguridad.

Manipulación de Objetos en la base de datos

La sentencias Create Table permite crear una tabla nueva. La sentencia create Table también crea las columnas de la tabla. El estándar ANSI-86 para la creación de una tabla es:

```
CREATE TABLE  nombre_tabla
        ( nom_columna tipo de dato [not null,...]
          .....
          nom_columna tipo de dato);
```

la sentencia Create Table crea una tabla y, opcionalmente, las columnas para esa tabla.

Algunas implementaciones obligan a tener al menos una columna nombrada cuando se crea la tabla. Posteriormente la sentencia Alter Table podría crear columnas crear en la tabla, además de poner la Llave primaria de la tabla (PK).

Como parte de la integridad de la base de datos las llaves primaria deben encontrarse definidas como not null.

Eliminación de una tabla

No existe en ANSI para la sentencia de eliminación de una tabla de una base de datos. Una sintaxis común para la eliminación de una tabla es

```
DROP TABLE nombre_tabla ;
```

Cualquier información que hubiera en la tabla se perderá.

Creación de una tabla Temporal

La siguiente sentencia creará una tabla temporal en el SQL estándar:

```
CREATE TEMP  TABLE  nombre_tabla
        ( nom_columna tipo_dato Unique, . . .  )
```

y para SYBASE

```
CREATE  TABLE #nombre_tabla
        ( nom_columna tipo_dato Unique , . . . , )
```

Una tabla temporal puede funcionar para almacenar información durante la sesión en curso. Una tabla temporal desaparece cuando finalice esta sesión.

La palabra reservada temp no es parte del estándar SQL y no se encontrará en todas las implementaciones de bases de datos.

Agregado de columnas a una Tabla

La sentencia Alter Table añade columnas a una tabla que existe. Estas sentencias no forman parte del ANSI-86 del estándar de SQL.

```
ALTER TABLE nombre_tabla ADD COLUMN nom_columna tipo de dato Not null
        DEFAULT literal SYSTEM
```

Acceso a los Datos

Se pueden recuperar datos selectivamente de las tablas. Se pueden imprimir selectivamente algunas columnas y algunas de las filas. Se puede combinar, recuperar y presentar en la pantalla la información ubicada en múltiples tablas.

Consultas

La sentencia Select muestra en pantalla alguna o todas las filas de una tabla. El resultado de la selección es una nueva tabla conteniendo los elementos seleccionados. La siguiente figura muestra todas las filas para el número de pedido, siendo seleccionadas de una tabla que contiene tres pedidos.

Tabla Original

id_pedido	Id_artículo
3	1003
3	3233
3	4434
2	5545
1	4543
1	2454
1	1904

Los siguientes ejemplos comienzan con sentencias de selección simples y continúan con otras más complejas. Aquí está la sintaxis para una selección simple:

```
Select *
from nombre_tabla
```

El asterisco indica que todas las columnas de la tabla serán seleccionadas.

Una condición de búsqueda restringe la selección de los registros de una tabla.

```
SELECT *
FROM {nombre- de-tabla | nombre-de-vista}, ...
WHERE condición-de-búsqueda
```

La condición de búsqueda es de la forma:

```
[NOT] (predicado | condición de búsqueda) , ...
[ { [AND | OR] [NOT] (predicado | condición)
de búsqueda ) }. . .]
```

Descripciones de las subcondiciones aparecen más completamente en las siguientes secciones, pero aquí se presentan algunos ejemplos.:

```
Select * from clientes
  where cli-num = 3
  or cli-num > 4
```

Teclea la sentencia de selección en un sistema SQL interactivo hace que cada uno de los registros aparezca en la pantalla. Esta sentencia incluida en un programa host también recupera la misma filas. Los operadores relacionales en una condición de búsqueda pueden restringir una selección. Estos operadores demuestran a continuación:

Signo	Descripción	Ejemplo
=	Igual	Select * from pedidos where emb_tasa = .65
!=	Distinto	Select * from pedidos where emb_tasa != .65
<>	Distinto	Select * from pedidos where emb_tasa <> .65
<	Menor que	Select * from pedidos where emb_tasa < .65
<=	menor o igual que	Select * from pedidos where emb_tasa <= .65
>	mayor que	Select * from pedidos where emb_tasa > .65
>=	mayor o igual que	Select * from pedidos where emb_tasa >= .65

Las sentencias aritméticas pueden también restringir una operación.
Aquí se presentan los operadores aritméticos:

Signo	Descripción	Ejemplo
+	Adición	Select * from pedidos where emb_tasa = 2+.5
-	Resta	Select * from pedidos where emb_tasa = 2 - 6
*	Multiplicación	Select * from pedidos where emb_tasa = 3*.1
/	Menor División	Select * from pedidos where emb_tasa = 3/10
**	Exponenciación	Select * from pedidos where emb_tasa <= 5**2
Mod	Módulo	Select * from pedidos where emb_tasa = 2 mod 3

Orden de Precedencia

El orden de precedencia de los operadores aritméticos determina el orden de evaluación de una expresión aritmética, tal como se muestra en la siguiente figura. La multiplicación y la división aparecen a un nivel más alto en esta lista, así que ellas se realizan antes que la suma y la resta. La

resta la suma se realizan antes que la exponenciación. La exponenciación se realiza antes que el operador de módulo.

* /
\+ -
**
mod

Aunque este orden pueden ser cambiado por los paréntesis.

Operadores Booleanos.

Hay tres valores Booleanos en SQL - verdad, falso desconocido -. (Desconocido es lo mismo que nulo). Desconocido es el tercer valor lógico, como verdadero o falso. Esto es diferente a un valor no especificado o no determinado.

Por ejemplo, algunos lenguajes de programación pueden referirse a variables no inicializadas. Esta variable puede tener diferentes valores indeterminados en cada momento que se ejecute el programa sobre una computadora diferente. Una variable no inicializada puede contener un valor de diez en un momento y en el siguiente instante tener un valor de diez mil.

Esto es diferente a las variables Booleanas desconocidas. El valor desconocido es el mismo siempre que un programa se ejecuta o cuando un programa de ejecuta sobre diferentes computadoras.

Los dos operadores Booleanos and y or combinan dos valores para producir un valor de verdadero, falso o desconocido. Por ejemplo, la siguiente expresión produce un valor de verdadero:

verdad and verdad

Hay tres valores Booleanos - verdad, falso y desconocido -. El operador "not" invierte el valor de verdadero o falso. Not verdad es falso. Not falso es verdad.

El orden de la precedencia de los operadores Booleanos es:

and
or

Los paréntesis pueden cambiar este orden. Por ejemplo:

(not verdad) and verdad

La siguiente figura muestra como los valores Booleanos se combinan con el operador and para producir el valor nuevo de verdadero o falso.

AND	Verdad	falso	desconocido
Verdad	verdad	falso	desconocido
Falso	falso	falso	desconocido
Desconocido	desconocido	desconocido	desconocido

Por ejemplo

verdad and verdad

Produce un valor de verdadero. El valor Booleano verdad se combina con el segundo valor verdad a través del operador "and". Esto da como resultado un valor - Booleano de verdad.

La siguiente tabla muestra cómo dos valores lógicos se combinan para producir un tercer valor lógico.

OR	verdad	falso	desconocido
Verdad	verdad	verdad	desconocido
Falso	verdad	falso	desconocido
Desconocido	desconocido	desconocido	desconocido

Por ejemplo la expresión verdad or falso da como resultado un valor Booleano de verdad.

El operador not cambia el sentido de un valor. Not verdad tiene un valor de falso, y not tiene un valor de verdad.

Expresiones combinadas

Los operadores Booleanos pueden combinar operaciones aritméticas y relacionales. Aquí hay algunos ejemplos:

```
Select  *  from pedidos
where emb_tasa  !=  .65  and emb_tasa  .85
Select  *  from pedidos
where emb_tasa  is not null
Select  *  from pedidos
where emb_tasa is not  .45
```

Aquí hay un ejemplo de la tabla pedidos

Pedidos

Pedi-num	Num. Articulo
3	a1003
3	a3233
3	b4434
1	b5545
1	a4543
1	b2454
2	a1432

La siguiente consulta devuelve todos los pedidos donde el número de pedido es mayor que 1 pero menor que 6:

```
Select  *
  from pedidos
  where Pedi-num  >  1  and  Pedi-num  <  6
```

Cláusula Order By

Por definición, los registros en una relación en la tabla carecen de un orden predefinido.
La cláusula "order by" devuelve los registros de una selección de orden ascendente o descendente.

```
Select  *
  FROM {nombre-de-tabla  |nomdre-de-vista},  ...
  [WHERE condición-de-búsqueda]
  [ORDER BY {nombre-de-columna  [ASC  |  DESC]}...]
```

Aquí se presenta la tabla de pedidos:

Pedidos

Pedi_num	Num_ Articulo	Emb_cost
3	a1003	$1.53
3	a3233	$2.24
3	b4434	$3.33
2	a1432	null
1	a3332	$1.56
1	b2323	null
5	c4343	$1.85
6	a3232	null

Esta consulta devuelve seis registros de la tabla de pedidos donde el número del pedido es mayor que uno pero menor que cinco:

```
Select  *
  from pedidos
  where Pedi-num  >=1
  and Pedi-num  <  5
```

La siguiente tabla es el resultado de la selección.

Pedi-num	Num. Articulo	emb-cost
3	a1003	$1.53
3	a3233	$2.24
3	b4434	$3.33
2	a1432	Null
1	a3332	$1.56
1	b2323	Null

La siguiente sentencia Select devuelve los registros, en orden ascendente de número de artículo

```
Select * from pedidos where Pedi-num >=1 and Pedi-num < 5
  order by Num. Articulo
```

Esta sentencia de Select devuelve los mismos registros en orden descendente:

```
Select *
  from pedidos >= 1
  where Pedi-num < 5
  order by Num_Artículo DESC
```

Las operaciones para ordenar se pueden combinar. Por ejemplo, la siguiente consulta ordena los registros seleccionados en orden ascendente del número de pedido y orden descendente del coste del embarque.

```
Select *
  from pedidos >= 1
  where Pedi-num < 5
  order by Pedi-num, emb-cost DESC
```

Recuperación de registros distintos

Los contenidos de los registros de datos pueden determinar los resultados de una selección. Se pueden agrupar registros distintos basados en sus contenidos.

La palabra reservada *Distinct* elimina los registros duplicados de los resultados de una selección.

```
SELECT [Distinct] lista-de-selección
  FROM { nombre-de-tabla | nombre-de-vista}, ...
  [WHERE condición-de-búsquedas]
  [GROUP BY {nombre-de-columna} ]
  [ORDER BY {nombre-de-columna [ASC | DESC] }...]
```

A continuación se presenta otra tabla con los datos de pedidos

Pedi-num	Num-Artículo	emb-cost
3	a1003	$1.53
3	a3233	$2.24
3	b4434	$3.33
2	a1432	$1.56
1	a3332	$1.56
1	b2323	$1.85
5	c4343	$1.85
6	a3232	$2.20

La tabla del ejemplo mantiene ocho filas. La siguiente proyección selecciona todos los números de pedido de la tabla:

```
Select Pedi-num from pedidos
```

Si añade la palabra reservada Distinct, la consulta recuperará sólo cinco números de pedido, concretamente los cinco números de pedidos distintos.

Select Distinct Pedi-num from pedidos

Pedi-num
3
2
1
5
6

Existen cinco funciones agregadas SQL que operan sobre las filas recuperadas en una selección. El resultado es un único valor. Por ejemplo, la consulta.

Select avg (emb-tasa) from pedidos

Recupera la media de las tasas de embarque de la tabla pedidos. Sumando todos los valores encontrados en la columna de tasa de embarque y dividiendo por el número de registros encontrados en la tabla, se obtiene esta media.

Las cinco funciones de agregación de SQL son:

Max	devuelve el valor máximo encontrado entre las filas seleccionadas
Mín	devuelve el valor mínimo encontrado entre las filas seleccionadas
Sum	devuelve la suma de los valores de las filas seleccionadas
Count	devuelve el contador de las filas seleccionadas
Average	la media aritmética de las filas devueltas

La palabra reservada Distinct puede eliminar filas duplicadas de una consulta, y filas duplicadas antes de aplicar una de estas funciones. Por ejemplo, la consulta.

```
Select count(Distinct emb-tasa)
  from pedidos
```

Contabilizará cada una de las tasas de embarque distintas. Esto ocurre debido a que la consulta recupera primero cada una de las tasas y luego contabiliza cada una de las distintas tasas.

Group by

La cláusula *group by* conjunta datos seleccionados en subgrupos. Un dato se recupera de cada uno de los subgrupos.

```
SELECT  *
  FROM { nombre-de-tabla  |  nombre-de-vista},  ...
  [WHERE condición-de-búsqueda]
  [GROUP BY {nombre-de-columna} ]
  [ORDER BY {nombre-de-columna  [ASC  |  DESC]  }...]
```

Aquí se presenta una tabla de pedidos:

Pedi_num	Num_Artículos	Emb_cost
3	a1003	$1.53
3	a3233	$2.24
3	b4434	$3.33
2	a1432	$1.56
1	a3332	$1.56
1	b2323	$1.85
5	c4343	$1.85
6	a3232	$2.20

La siguiente consulta selecciona cada uno de los números de pedido de la tabla de pedidos. Esta consulta agrupa, posteriormente, los números de pedido.

```
Select  Pedi-num
from pedidos
Group by Pedi-num
```

La agrupación es extremadamente útil cuando se combinan con las funciones agregadas.
Sin una cláusula Group By, una función agregada opera sobre todos los registros recuperados en una selección. Una cláusula Group By puede dividir los registros seleccionas en grupos y permitir a las funciones agregadas operar sobre estos grupos.

Por ejemplo

```
Select Pedi-num, avg (emb-tasa)
  from pedidos
  Group by Pedi-num
```

Recupera la media de las tasas de embarque para cada uno de los grupos seleccionados de número de pedidos. Primero se realiza la agrupación de pedidos por número de pedido.

Cursores

Los cursores pueden funcionar solamente dentro de un programa escrito en un lenguaje host, por ejemplo, dentro de un programa C.

Se puede crear un cursor con una sentencia declare cursor.

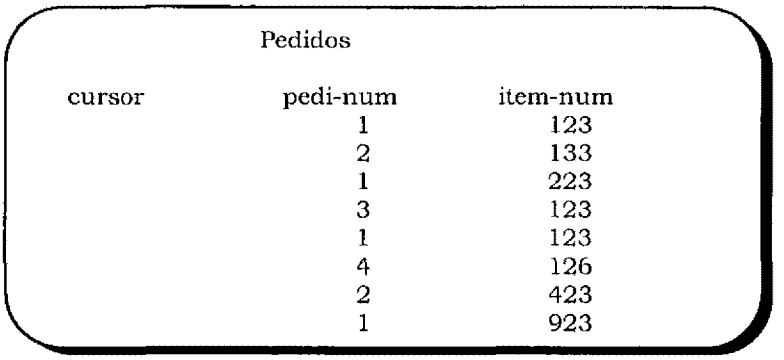
```
DECLARE nombre_de_cursor CURSOR FOR
Sentencia_select [ { UNION | UNION ALL }
  Sentencia_select...
[FOR UPDATE OF {nombre-de-columna},...] |
[ORDER BY {{nombre-de-columna | integer}
  [ASC  |  DESC ],...}]
```

La sentencia Declare cursor incluye una sentencia Select. Se debe declarar un cursor antes de usarla. Cuando el programa necesita un cursor, una sentencia open cursor abre el cursor. La sentencia open cursor ejecuta la sentencia Select declarada. La sentencia Select crea un hilo de unión a través de los datos seleccionados.

El programa host puede ir entonces por los elementos de los datos seleccionados. Esto quiere decir que el cursor puede estar leyendo de un elemento seleccionado a otro.

En algunos sistemas, el cursor puede moverse sólo hacia adelante. Otros sistemas proporcionan un mayor control. Algunos permiten a los cursores ir hacia atrás así como hacia adelante. Otros sistemas permiten a los cursos movimiento relativo o directo.

El hilo a través de los datos seleccionados es dinámico. Esto es a causa de los diferentes datos que pueden ser seleccionados en cada momento por el cursor que esté abierto. Aquí se presenta un ejemplo de cursor:



Pedidos

cursor	pedi-num	item-num
	1	123
	2	133
	1	223
	3	123
	1	123
	4	126
	2	423
	1	923

A continuación está sentencia que declara el cursor mostrado en la figura anterior:

```
declare c-pedi cursor for
  Select  * from pedidos
    where pedi-num = 1
```

Una sentencia declare cursor sólo puede funcionar dentro de un programa host. El programa host define el cursor con la sentencia declare cursor.

Conviene seguir estos pasos cuando se utilice un cursor dentro de un programa host. Primero, definir el cursor con la sentencia declare cursor. Después, cuando el programa host está preparado, usar el cursor la sentencia prepare cursor compila dinâmicamente el cursor.

El programa abre el cursor cuando se necesitan los datos, esta apertura selecciona un conjunto dinámico de filas de la base de datos.

El programa host puede entonces acceder a las filas una a una con el cursor abierto mediante la sentencia fetch. Por último, el cursor se cierra cuando no hay más datos necesarios. Aquí hay un ejemplo en un lenguaje host hipotético:

```
programa ejemplo
define record item-seleccionado
        pedi_num integer,
        part_num integer,
define sentencia-select char (512)
        let sentencia-select  =
        "Select   *   from pedidos where pedi-num  =  1"
declare c_pedi
        cursor from sentencia_select
prepare c_pedi
        open cursor c_pedi
        while SQLCODE <>  0
            fetch c-pedi into item-selccionado
            {sentencia para procesar los datos devueltos}
        end while
       close c_pedi
end program.
```

El uso de cursores varía de un sistema a otro, po esto se debe consultar con la documentación adjuntada con el sistema de gestión de base de datos y con el lenguaje host. Aquí se muestra una sintaxis típica para sentencias de manipulación de cursores.

La sentencia prepare compila un cursor declarado previamente:

```
PREPARE nombre-de-sentencia FROM
:variable_host
```

La sentencia open abre un cursor compilado

```
OPEN nombre-de-cursor
  [ USING :variable-host,... |
  USING DESCRIPTOR :variable_host]
```

El programa host recupera después elementos individuales seleccionados por el cursor con la sentencia fetch:

```
FETCH nombre_de_cursor
  USING DESCRIPTOR variable_host
```

Algunas implementaciones soportan el scroll de cursores. Con un scroll, el programa host puede moverse hacia delante y hacia atrás a través de los elementos seleccionados, mejor que desde el comienzo hasta el final exclusivamente. Por ejemplo, el primero, el último, el siguiente o el anterior elemento se puede seleccionar.

La sentencia close cursor cierra un cursor:

```
CLOSE nombre_de_cursor
```

Inserción de Filas

La sentencia insert incrementa los registros de una tabla

```
INSERT INTO {nombre-de-tabla  |
nombre-de-vista}
  [({ nombre-de-columna}, ...) ]
VALUES ({literal  |  NULL}, ...)  |
        sentencia-select
```

Una sentencia insert que tiene una cláusula values inserta una única fila. A continuación hay dos ejemplos:

```
insert into table a  value (1, 2, "A3", 6.2)
insert into pedidos (pedi_num, cli_num,  emb_fech )
values (2, 1, "12/14/93")
```

Una cláusula Select puede seleccionar registros de una tabla e insertar copias en la tabla destino. Por ejemplo:

```
insert into destino
Select  *
from origen
```

La cláusula Select elige todos los registros de la tabla denominada origen. Una copia de cada fila seleccionada aparece posteriormente en la tabla destino.
La cláusula Select puede proyectar columnas para la inserción en la tabla destino.

```
Insert into destino
Select a, b,  c
from origen
```

La cláusula Select puede enganchar información de más de una tabla.

```
Insert into destino
  Select  a.cl,  b.c2,  c.c3
from  a,  b,  c
```

Finalmente, se puede restringir la cláusula Select. A continuación hay un ejemplo:

```
insert into destino
Select a, b c
  from origen
  where cl < 100
```

Borrado de Filas

La sentencia delete borra filas de una tabla. Aquí hay sintaxis de la sentencia delete:

```
DELETE FROM  {nombre-de-tabla |
nombre-de-vista}
[WHERE condición-de-búsqueda]
```

Si se omite la cláusula where se borrarán todas las filas de la tabla. Este ejemplo borra todas las filas de la tabla pedidos:

```
delete from pedidos
```

Una cláusula where puede seleccionar las filas a borrar. Por ejemplo:

```
delete from pedidos
where cli-num  >  1
```

Actualización en las Filas

La sentencia update modifica la información de las filas de una tabla. Aquí está la sintaxis de la sentencia update:

```
UPDATE  {nombre_de_tabla    nombre_de_vista}
SET  {nombre_de_columna  =  expresión  |  NULL}
[WHERE condición_de_búsqueda]
```

Una sentencia update que omita la cláusula where cambiará todos los registros de una tabla, o bien a nulos, o bien a un valor dado.

```
Update pedidos set emb-tasa  =  null
update pedidos set emb-tasa  =  .6
```

Una cláusula where en una sentencia update cambia las filas si la cláusula where evalúa verdad. El ejemplo que sigue actualizará todas las filas donde emb_tasa sea .86 a .95.

```
Update pedidos
  set emb-tasa  =  .95
  where emb-tasa  =  .86
```

Seguridad

Vistas

Una vista crea una nueva tabla virtual ensamblando uno o más tablas existentes. Una vez que la vista se ha creado, se podrá accesar a ella como a una tabla.

Este mecanismo hace fácil la creación de diferentes formas de consulta de la base de datos. Una vista se puede crear para que presente al usuario justo la información que necesita.

Una vista también suministra protección de datos. Puede presentar sólo algunos de los datos de la tabla; oculta al usuario el resto de los datos de la tabla.

Con una vista, una parte seleccionada de una tabla existente aparece como una tabla separada con un único nombre. Una vista también puede traer columnas de varias tablas. Aquí está la sintaxis utilizada para crear una vista:

```
CREATE VIEW nombre_de_vista
[ ( { nombre_de_columna_de_vista },...) ]
```

A continuación hay un ejemplo que crea una vista llamada emb_info de la tabla existente pedidos:

```
create view emb_info
  (pedi_num, emb_fech)
```

Pedidos

pedi-num	cli-num	emb-fech
1	32	1/3
2	33	1/2
3	17	1/4
4	22	1/5
5	17	1/4

Aquí está la vista que resulta de la sentencia create view:

Emb_info

pedi-num	emb-fech
1	1/3
2	1/2
3	1/4
4	1/5
5	1/4

La vista funciona como cualquier otra tabla. Se puede recorrer para seleccionar y ver diferentes datos seleccionados, tal y como si fuera una tabla. Asimismo se pueden también eliminar vistas con DROP VIEW nombre_de_vista.

Cualquier sentencia Select puede crear una vista. Este ejemplo muestra cómo se puede hacer una vista de un JOIN:

```
create view backlog (
Select pedidos.pedi_num,
  cli_num,
  emb_fech,
  artículo_num
from pedidos,  artículo
where pedidos.pedi_num = artículo.pedi_num)
```

Ya que cualquier sentencia Select puede crear una vista. Una vista puede combinar selecciones, proyecciones y joins.

Las implementaciones que se venden pueden limitar la funcionalidad de las vistas. Por ejemplo, varios fabricantes no suministran facilidades para actualización a través de vistas que incluyan un JOIN. Se deberá consultar la documentación adjuntada con el sistema para más información acerca de las vistas.

Control de Acceso a los Datos

Un sistema de gestión de base de datos relacionados debe proporcionar facilidades para mantener la integridad de los datos. La sentencia de control de acceso a los datos proporciona esta integridad. Estas sentencias controlan el acceso a la base de datos o una parte de ella.

Las sentencias de control de acceso a los datos protegen la base de datos a través de la limitación de accesos. Ellas permiten o no el acceso a los usuarios.

Diferentes usuarios pueden obtener diferentes permiso. Además, las sentencias de control de acceso a los datos pueden evitar que los usuarios accedan simultáneamente a los mismos datos.

Transacciones

Una transacción es una unidad de trabajo que resulta de uno o más cambios a la base de datos. Se pueden agrupar varias inserciones o modificaciones en una única transacción.

También se pueden aceptar o descartar, al final de la transacción, todos los trabajos incluidos en una transacción. Las sentencias SQL son las que controlan el que se acepten o rechacen los trabajos realizados dentro de una transacción. El mecanismo de transacción asegura que o bien se descartan todos los cambios.

Considérese un sistema que tiene entradas de pedidos. La entrada de nuevo pedido es posible que cause cambios en varias tablas. Es posible que sea necesario descartar los cambios a todas las tablas si existe algún error durante el proceso de entrada de datos.

Una transacción ayuda asegurar que los datos introducidos en la base de datos sean correctos. La transacción comienza cuando entra un pedido. Al final del proceso de entrada del pedido, transacción pasa o falla. Si la entrada de los datos es correcta, la transacción pasa. Si la entrada del pedido ha fallado, la transacción falla, y entonces la base de datos no salva los cambios.

Logging y Commit

Las transacciones ayudan a mantener en una base de datos la integridad de los datos. Un log puede almacenar transacciones cuando ocurran. Si la base de datos se pierde por cualquier motivo, se pueden añadir transacciones que estén en el log a la copia de seguridad más reciente, con el fin de restaurar la base de datos al estado más reciente.

El fallo de hardware o el software puede dañar la integridad de los datos. Los fallos de la computadora a mitad de una transacción pueden crear datos corruptos. Un buen RDBMS suministra facilidades de recuperación que permiten la restauración de una base de datos después que hayan ocurrido fallos de hardware o de software.

Las sentencias commit y rollback controlan las transacciones. La sentencia commit aplica a una base de datos todo el trabajo que se lleva a cabo durante una transacción. La sentencia rollback rechaza la transacción.

Permisos

Los permisos controlan el acceso a la base de datos. Cada uno de los permisos concede a un usuario el privilegio de realizar alguna operación. Los permisos y el privilegio que se conceden son:

Permisos	Privilegio concedido
alter	cambia las estructuras de las tablas
delete	borra filas de una tabla
insert	añade filas a una tabla
Select	selecciona filas de una tabla
update	cambia filas en una tabla
All	todos los permisos
index	crear o eliminar índices

La sentencia Grant concede a un usuario uno o más permisos de los mostrados en la tabla anterior.

```
GRANT {ALL  |  ALL PRIVILEGES  |
(ALTER  |  DELETE  |  INDEX  |  INSET  |  SELECT  |
UPDATE [({nombre-de-columna}, ...)]}, ....}
  ON {nombre-de-nombre}, ...
  TO {id-usuario  |  PUBLIC }, ...
[WITH GRANT OPTION]
```

Esta sentencia GRANT ALL asigna todos los posibles permiso a John:

```
grant All on ventas-pedi to John
```

La palabra reservada PUBLIC concede permiso a todo el mundo para acceder a la base de datos.
La siguiente sentencia concede todos los permisos a cualquiera que use la tabla de clientes:

```
grant All on clientes to PUBLIC
```

La concesión de permisos individuales es por nombre. Para conceder tres permisos separados al usuario Sally, se escribe:

```
grant insert, Select, update
  on clientes to Sally
```

Usualmente, sólo el creador de una tabla o el administrador de la base de datos puede conceder permisos. El añadir la cláusula WITH grant OPTION a una sentencia grant puede cambiar ésta. La cláusula WITH grant OPTION no se aplica si un permiso tiene ya concedido el PUBLIC. Si a un permiso se le ha concedido ya que el PUBLIC, cada uno de los usuarios tiene ya ese permiso.

La sentencia REVOKE quita los permisos a los usuarios:

```
REVOKE {ALL | PRIVILEGES |
{ALTER | DELETE | INDEX | INSERT | SELECT |
UPDATE }, ...}
  ON {nombre-de-tabla}, ...
  FROM {id-usuario | PUBLIC}, ...
```

Aquí se encuentran dos ejemplos de la sentencia REVOKE:

```
REVOKE All on clientes from PUBLIC
REVOKE INDEX, insert, update
  on clientes from sam
```

Revocar un permiso que sea PUBLIC, borrará el permiso de todos los usuarios. La sentencia REVOKE puede también revocar permisos de usuarios individuales.

Si un usuario tiene asignado un permiso anteriormente, una sentencia REVOKE para el PUBLIC no revocará ese permiso asignado individualmente. La sentencia REVOKE no forma parte del estándar ANSI, variando su sintaxis de un sistema a otro.

Bloqueo de tablas

Cualquier usuario o transacción con permiso para acceder a una tabla puede bloquear temporalmente esta tabla. Mientras la tabla está bloqueada, solamente el usuario o la transacción que la bloquean pueden acceder a esta tabla. Este mecanismo de bloqueo suministra el control de concurrencia para la base de datos. El control de concurrencia evita que transacciones separadas o usuarios entren en colisión unos con otros.

Consideremos dos empleados que introducen pedidos y son responsables de actualizar tales pedidos. El primer empleado accesa al pedido. Este empleado ha encontrado que uno de los elementos de los pedidos devueltos, se encuentra ahora en artículo. Un cliente llama al segundo empleado y le pregunta por el estado del pedido. El segundo empleado accede al pedido y ve dos elementos sobre el pedido devuelto. Ambos empleados tienen ahora una copia del mismo pedido, la cual están mirando. Ellos pueden acceder a la misma tabla en el mismo momento. El primer empleado actualiza la base de datos indicando que el elemento del pedido devuelto está ahora disponible. La base de datos muestra ahora el elemento como disponible. El primer empleado borra entonces el elemento del pedido devuelto, y se actualiza la base de datos mediante el borrado del elemento solicitado del pedido.

Los dos empleados han colisionado cuando accedieron a la tabla. Si el primer empleado hubiera bloqueado la tabla antes de hacer cualquier cambio, el segundo empleado habría tenido el acceso denegado para esa tabla durante la actualización. El segundo empleado habría estado inhabilitado para acceder a la tabla hasta que el pedido actualizado indicara el elemento en artículo.

Este es un ejemplo de control de concurrencia. El control de concurrencia aplicado con la sentencia de lock habría evitado a los usuarios colisionar uno con otro. La ejecución de una sentencia commit o una sentencia rollback desbloquea automáticamente una tabla. El final de un programa o transacción también elimina cualquier bloqueo.

La sentencia lock table evita que otros usuarios o transacciones accedan a una tabla bloqueada. Dos usuarios diferentes no pueden actualizar concurrentemente una tabla bloqueada.

La sentencia lock table no forma parte del estándar ANSI. La sintaxis para SYBASE es la siguiente y la más común.

```
LOCK TABLE {nombre-de-tabla | nombre-de-vista}
IN { SHARE | EXCLUSIVE}MODE
```

El usuario puede bloquear una tabla de dos maneras diferentes. Otro usuario puede leer pero no actualizar una tabla bloqueada en modo compartido, o bien otro usuario ni puede leer ni puede escribir en una tabla bloqueada en modo exclusivo.

Una tabla bloqueada puede disminuir dramáticamente el rendimiento de una base datos multiusuarios. Una vez que el usuario bloquea una tabla, todos los demás usuarios tienen que esperar hasta que se elimine el bloqueo. Es posible que el usuario bloquee una tabla y se vaya luego a comer. Todos los demás usuarios interesados en esta tabla deberán esperar el regreso de este usuario y la eliminación de bloqueo para que ellos puedan acceder a la tabla. El final de una transacción o el final de un programa libera todos los bloqueos.

Bloqueo de Filas

Una sentencia Select con una cláusula for update bloquea todas las filas seleccionadas. Esto evita que otros usuarios cambien estas filas hasta que se haya completado la actualización.

```
SELECT *
  FROM {nombre_de_tabla | nombre_de_vista}, ...
  [WHERE condición_de_búsqueda]
  FOR UPDATE OF {nombre_de_columna}, ...
```

Algunas bases de datos suministran mecanismos de bloqueo sofisticados que bloquean al nivel de fila. Algunos pueden bloquear todos los datos seleccionados con un cursor mientras permiten a otros usuarios acceder a cualquier otro dato no seleccionado por el cursor.

Aquí hay un ejemplo de una sentencia Select que bloqueará filas hasta que se haya completado la siguiente sentencia update:

```
Select *
  where cli-num = 151
  for update OF cli-num
update pedidos
  set cli-num =152
  where cli-num = 151
```

Todas las filas seleccionadas por la sentencia Select están bloqueadas hasta que se haya completado la sentencia update.

Rendimiento e Indexación

El diseño de una base de datos debería comenzar siempre con una representación lógica exacta de los datos apropiada para la base de datos. A menudo, la mayoría de las estructuras lógicas para los datos no proporcionan un rendimiento adecuado.

El diseño lógico de la base de datos da como resultado un esquema que presenta la estructura lógica de los datos. El diseño físico representa la forma en que los datos se almacenan físicamente. El diseño lógico cambia a menudo cuando se crea el modelo físico, con el fin de mejorar el rendimiento. Esta sección describe algunas de las técnicas usadas para optimizar el rendimiento de una base de datos.

Supongamos que tenemos el número de teléfono de alguien y una guía. Encontrar el nombre de la persona en la guía con ese número de teléfono es difícil ya que los listados están por nombre y no

por número. Para encontrar el nombre, se debería recorrer la guía secuencialmente hasta encontrar el número.

Es más fácil encontrar la persona por el nombre ya que los listados están ordenados por orden. Encontrar el nombre si se tiene el número de teléfono es difícil, aunque la información este ahí.

Por definición y en la práctica, las filas de una relación no están en ningún orden particular. Como en el ejemplo anterior, encontrar una fila en particular requería una búsqueda Secuencial en la tabla.

Se puede utilizar un índice para accesar más rápidamente a las filas de la tabla. Un índice en una base de datos es como un índice en un libro.

Un índice de libros una lista de información llave sobre varios aspectos mantenidos en un orden determinado. El índice apunta a la información de varias materias.

Hay que tener en cuenta que el índice de materias facilita el encontrar la información sobre el servicio o un producto en particular en las páginas amarillas.

Tomemos como ejemplo las entradas de un índice para automóviles en las páginas amarillas, incluyendo aquellos para reparación, alquiler, carrocería, etc. Lo que viene a continuación de una entrada en el índice de cualquier de automóvil es un número de página. El número de página apunta a la sección, o secciones, que contienen el dato indexado. Recorrer el índice evita recorrer el libro entero.
Un índice de una base de datos referencia de datos en la base de datos. Encontrar la entrada del índice aumenta la rapidez de acceso a los datos indexados.

La siguiente tabla tiene dos entradas, cli_num y part_num.

Esta podría ser una representación de todos los clientes que tienen pedida una parte determinada.

Pedi-num	part-num
1	20
2	21
3	30
4	22
5	20
6	22
7	19
8	20

Encontrar cualquier cosa en esta tabla, por número de cliente es fácil, ya que los clientes están en orden ascendente. Si la tabla no está en orden, como ocurre realmente con las tablas, encontrar un número de clientes es más difícil.

Aquí está la misma información en una tabla nueva donde los números de clientes no están en un orden en particular. Esta tabla tiene también un número de fila que da el direccionamiento de cualquier fila de datos.

Pedi-num	cli-num	part-num
1	1	20
2	4	21
3	8	30
4	2	22
5	5	20
6	7	22
7	6	19
8	3	20

Encontrar un número de clientes en particular en la tabla anterior es más difícil.

Un índice hace más fácil la localización de cualquier número de cliente en la tabla desordenada.

Aquí se presenta un índice para los números de cliente en esta tabla.

Pedi-num	localiza
1	1
2	4
3	8
4	2
5	5
6	7
7	6
8	8

Este índice facilita de nuevo el encontar cualquier fila por número de cliente. Para encontrar la fila para el número de cliente cuatro, se recorre la columna cli_num en el índice. Una vez que se encuentra el número de cliente deseado, índice proporciona la ubicación física de la tabla. La localización por el índice permite el acceso directo a la fila deseada.

El índice puede también ordenar los datos mantenidos en una tabla. En vez de ir moviendo cada una de las filas, el índice las ordena. La selección de cada uno de los elementos indexados en orden selecciona los datos de la tabla en orden.

Un índice puede ser usado para aumentar la velocidad de acceso a cualquier columna de una tabla. Encontrar cualquier cosa en la tabla ordenada original por número de parte es difícil, ya que los números de parte carecen de cualquier orden en particular. Un índice facilita y aumenta la rapidez en encontrar la información mediante el número de parte. La siguiente tabla indexa las partes que aparecen en la tabla ordenada originalmente.

part_no	cli_num
19	7
20	1,5,8
21	2
22	4,6
30	3

Encontrar cualquier cliente que utilice una parte en concreto es ahora fácil. El número de parte se encuentra en la primera columna de índice. Cada uno de los clientes que usa la parte aparece en la segunda columna.

Por ejemplo, la parte número 20 puede ser encontrada en las localizaciones 1, 5 y 8.

Encontrar todos los valores de datos de 20 es fácil a través de la localización de una entrada en el índice, en vez de cada una de las entradas de las tablas de datos. Con el índice, todas las entradas para un elemento de datos particular están disponibles sin tener que recorrer la tabla entera.

Nótese que esta indexación no es parte del modelo relacional. La localización es una dirección que es interna a los RDBMS.

La asignación del espacio al índice reduce el tiempo necesario para recorrer los datos. El tiempo de compilación de índice una sola vez reduce la cantidad de tiempo necesaria para el recorrido posterior.

La ventaja de los índices es que aumentan la velocidad de acceso a los datos en una tabla. Su desventaja es que la adición de datos es lenta, ya que el índice debe ser actualizado con los datos añadidos.

La sentencia

```
CREATE [UNIQUE] INDEX nombre-de-índice
        on nombre-de-tabla
    ( [ nombre-de-columna  { ASC  DESC} ]...)
```

crea un índice.
La sentencia siguiente elimina el índice.

```
DROP INDEX nombre-de índice
```

Cuando debe usar un índice

√ Puede utilizarse una prueba para determinar cuando conviene añadir índices a una base de datos: si una aplicación se ejecuta lentamente cuando recorre los datos, se puede pensar en crear un índice.

√ Un índice apropiado aumentará la velocidad de hacer joins o de recorrer la tabla. Si frecuentemente recorre una tabla sobre una columna, construya un índice para esa columna. Si tiene dos tablas a las que frecuentemente realiza JOIN sobre la misma columna, construye un índice para cada una de las tablas sobre esa columna.

√ Si su aplicación debe recorrer frecuentemente la tabla de número de cliente para los números de cliente, construya un índice para los números de cliente.

√ No debería construir un índice sobre cualquier tabla que contenga nada más que cientos de registros. El índice no ayuda; de hecho, es posible que disminuya el acceso a la base de datos.

√ El índice para un elemento de datos puede ocupar tanto espacio como los datos que están siendo indexados. Esto varía de un sistema a otro y depende enormemente del esquema de índices elegido por el implementador de SGBD.

√ Consulte la documentación suministrada con el sistema de base de datos para encontrar más información sobre el indexamiento. El esquema de índices varía de un sistema a otro y así su sobrecarga.

Llaves basadas en múltiples columnas

SQL permite un índice de múltiples columnas - un índice de múltiples columnas o índice compuesto-.

El índice de múltiples columnas puede exceder las llaves Primarias. En muchos casos, el acceso a la base de datos se mejora con índices de múltiples columnas.

Esta sentencia crea un índice sobre la tabla partes, basado sobre los dos campos peso y costo:

```
create INDEX ix
on partes  (peso, costo)
```

El índice creado ayudaría a encontrar todas las partes de un peso especificado dentro de un rango de pesos. Este índice haría más rápido el encontrar todas las partes que pesen más de una libra y cuyo costo sea menor de 100 dólares.

Un índice puede evitar entradas de datos duplicados en una tabla. Si el operador UNIQUE aparece en la sentencia create INDEX, la tabla rechaza la entrada de valores duplicados. La sentencia

```
CREATE UNIQUE INDEX c_idx
  on clientes (cli-num)
```

evitará que el mismo cliente entre más de una vez en la tabla de clientes.

2.5 Diseño lógico de la base de datos

Las anomalías en el borrado ocurren cuando el borrado en una única fila de una tabla causa la pérdida de más información. Una base de datos con anomalías en la actualización requiere que varios registros sean cambiados como nueva entrada información a la base de datos.

La redundancia de datos se minimiza en una tabla cuando los datos que están relacionados con la llave contienen solamente datos que están relacionados con la llave primaria de la tabla. Por ejemplo, aquí se presentan una tabla de clientes y una tabla de pedidos.

Clientes

Cli_num	Teléfono	Cli_nomb
2	234-3423	Chuy Cortes
1	343-2341	Samuel Portillo
3	2323-3343	Juan Carlos Rendon
6	343-4443	Patty Vega
4	955-5675	Berna Perea
5	333-4343	Delia Villarruel
7	434-4434	Fernando Otero

Pedidos

Cli-num	pedi-num	peso	costo
1	4	11.2	45.54
4	5	25.3	65.55
4	33	14.5	55.34
7	54	35.8	126.34

La llave primaria en la tabla de clientes es el número de cliente. Todas las otras filas en la tabla contienen datos sobre cliente. El nombre de la compañía es el nombre del cliente. El número de teléfono en el número de teléfono del cliente. Cada uno de los atributos no llave estan relacionados con la llave primaria.

La llave primaria en la tabla de pedidos es el número de pedido. Las otras columnas contienen datos sobre esa llave. El número de cliente muestra que el cliente tiene pedido. Existe un peso para el pedido y costo para ese pedido. Esto es información de las llaves primaria; es decir, es información del pedido.
Aquí se presentan las dos tablas combinadas en una única tabla.

cli-num	teléfono	cli-nomb	cli-num	cli-num	cli-num
2	234-3423	Johnson Supply Co.			
1	343-2341	Redmond Feed	4	11	45.54
3	232-3343	Sasquatch Pet Supply			
6	343-4443	Laredo Feed			
4	955-5675	Autrey Barn Supply	5	25.30	65.55
4	955-5675	Autrey Barn Supply	33	14.5	55.34
5	333-4343	Cats and Dogs			
7	434-4434	Cats and Dogs	54	35.8	126.34

Esta tabla es mucho más difícil de cambiar y de actualizar ya que contiene información redundante. Por ejemplo, el número de teléfono para el número de compañía cuatro tiene ocurrencias en dos filas de la tabla. Si el número de teléfono para esa compañía cambia, varios registros en la tabla necesitan ser cambiados. Con las tablas separadas mostradas anteriormente, sólo un registro tiene que cambiar en la tabla de clientes cuando el número de teléfono cambia.

La llave primaria para esta tabla combinada es ahora el número de pedido. Mucha de la información en la tabla no se relaciona con la llave primaria. Por ejemplo, el número de teléfono es sobre la compañía, no sobre el pedido. El nombre de la compañía es información de la compañía, no del pedido.

Esta única tabla es más difícil de recorrer o modificar; no está tan normalizada como las anteriores.

Las siguientes secciones ayudan a formalizar ciertas nociones de sentido común dirigidas a que las columnas no llave en la relación deberían solamente mantener información relacionada con la llave primaria.

El secreto para construir bases de datos normalizadas está en diseñar bases de datos donde las tablas son simples y las columnas no llave sólo contienen información relacionada con la columna llave.

Normalización de una tabla.

Una relación normalizada no contiene grupos de datos repetidos. La normalización cambia una estructura de datos no normalizada en una tabla. Una relación normalizada es más simple que una estructura de datos no normalizada.

Hacer cambios en una estructura normalizada es más fácil que hacer cambios en una estructura de datos no normalizada.

Una base de datos relacional es un grupo de relaciones normalizadas de varios grados. El conjunto de tupla en las relaciones cambia con el tiempo. La normalización de las relaciones es necesaria en una base de datos. Las relaciones están vacías cuando son creadas. El número de tuplas añadidas posteriormente determina el grado de cada una de las relaciones.

El grado de una relación, igual que el conjunto de tuplas, cambia con el tiempo. Las tuplas no cambian en el modelo relacional. En vez de eso, una nueva tupla reemplaza a la tupla antigua. En la práctica la mayoría de los sistemas de base de datos relacional permiten cambiar tuplas.

La redundancia de datos no es deseable, ya que impone mayor dificultad a la base de datos para actualizar o cambiar. Una base de datos con redundancia de datos condiciona la aparición de problemas de actualización o de consulta. Por ejemplo, en una tabla de la base de datos se mantienen tanto los pedidos como las direcciones de los clientes. Si se cambia la dirección de un cliente, significaría cambiarla en cada uno de los pedidos de la tabla. Mover las direcciones de los clientes a una tabla separada hace que sea más fácil actualizar la dirección del suministrador cuando estos cambian. Sólo se tendría que cambiar una dirección de un registro, en vez de cambiarla en cada uno de los pedidos.

Cada una de las tablas en una base de datos debería contener solamente datos acerca de una única entidad. Por ejemplo, una tabla de pedidos debería Contener sólo columnas que contengan datos sobre los pedidos. Una tabla de clientes debería contener información sólo de los clientes.

El proceso de asegurar la singularidad de los datos (así como el correspondiente proceso de eliminar la redundancia de datos) se llama normalización.

Para diseñar una base de datos normalizada, primero se necesita comprender la estructura de una relación, tal y como se describe en el capítulo anterior. En suma, se debe llegar a familiarizarse con los tres tipos de dependencias de datos - funcional, transitiva y multivaluada -.

Las formas normales son guías para construir una base de datos bien estructurada. Estas formas normales básicas son:

- √ Primera forma normal.
- √ Segunda forma normal.
- √ Tercera forma normal.
- √ Forma normal de Boyce-Codd.
- √ Cuarta forma normal.

Las guías presentadas en este capítulo ayudarán a construir bases de datos que permiten actualizaciones y consultas adecuadas.

Formas Normales

Una relación por definición, debe tener una llave primaria. Las formas normales codifican (no es un juego de palabra) la noción de sentido común de que todas las otras columnas en la tabla deberían contener sólo información directamente relacionada con la llave primaria.

Ciertas reglas ayudan a verificar que las tablas tienen esta propiedad. Estas reglas están expresadas como las formas normales descritas más adelante. Las formas normales ayudan a establecer tablas bien organizadas.

Una tabla que está en una de estas formas normales se llama normalizada.La normalización cambia una tabla NO normaliza en una tabla normalizada. Hay métodos simples, descritos posteriormente, para cambiar una tabla que no está normalizada en otra que sí lo está. La normalización es reversible. Una relación se puede convertir a una forma normal mayor. Hay métodos simples, descritos posteriormente, para cambiar una tabla que no está normalizada en otra que sí lo esta. Esto es debido a que no hay pérdida de información durante la normalización. La transformación se produce sin pérdida de información.

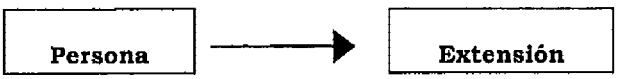
Por ejemplo observa la siguiente tabla con dos columnas, nombre y extensión. Hay sólo una llave Primaria, nombre. La estructura de los datos es simple. Cada individuo utiliza un determinado teléfono, pudiendo compartir distintos usuarios el mismo teléfono.

nombre	extensión
Paul	10
Susan	10
George	14
Sabrina	16
Bill	18
Karen	20

Un simple diagrama muestra la relación entre la extensión y el nombre. El diagrama muestra que el número de la extensión depende del nombre de la persona y que cada una posee una extensión.

Es tabla de ejemplo está normalizada ya que la información en la columna no llave extensión está directamente relacionada con la información de la columna llave primaria nombre. La columna extensión contienen información que directamente está relacionada con la llave nombre.

La extensión depende de la persona.



La extensión es información que está directamente relacionada con el individuo.

Añadiendo una tercera columna se hace que la tabla no esté tan normalizada. En este ejemplo, sólo ciertos teléfonos están autorizados para llamadas de larga distancia:

nombre	Extensión	larga-distancia
Paul	10	sí
Susan	10	sí
George	14	no
Sabrina	16	no
Bill	18	sí
Karen	20	sí

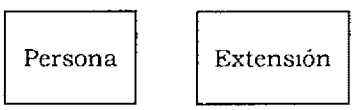
En la primera tabla, el número de extensión corresponde a la persona. La columna extensión está referida a la llave primaria nombre. La extensión es un número usado por el individuo.

Sólo hay una llave Primaria en esta tabla, nombre. La estructura de los datos es simple. Cada uno de los individuos puede usar un determinado teléfono, pudiendo los usuarios compartir un teléfono. Algunas extensiones tienen autorización para llamadas de larga distancia.

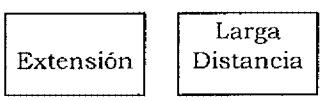
La larga distancia no está referida al usuario. La larga distancia está referida al teléfono. El teléfono está autorizado para largas distancias, no así el usuario.

Este otro diagrama indica de forma más clara la asociación entre estos tres elementos:

Extensión depende de persona



Larga distancia depende de extensión



Es una tabla normalizada, todas las columnas no llave de la tabla deberían contener información sobre la llave primaria. Es este ejemplo, la columna Larga-distancia no está referida a la llave primaria nombre. Esto significa que la tabla no está normalizada. Las reglas de normalización muestran que esta tabla descomponerse en dos tablas separadas.

Usuarios

Nombre	Extensión
Paul	10
Susan	10
George	14
Sabrina	16
Bill	18
Karen	20

Autorizaciones

Extensión	larga-distancia
10	sí
10	sí
14	no
16	no
18	sí
20	sí

La descomposición de la tabla original en dos tablas produce tablas en una forma más normalizada. Con dos tablas, cada uno de los atributos no llave en una tabla sólo contiene datos referentes a la llave primaria de esa tabla.

Con estas dos tablas hay menos redundancia en datos. Esto hace que las dos tablas, y a su vez la base de datos, sean más fáciles de actualizar. Si cambia la autorización de larga distancia para un número, sólo será necesario cambiar un número de registro en la tabla de autorizaciones. En la

tabla previa, múltiples registros necesitan cambiarse cada vez que una autorización se altera. Cada una de las dos asociaciones mostradas en el último diagrama de dependencia se aplica ahora a una tabla separada.

En este ejemplo, una tabla compleja se convirtió en dos tablas simples.

Esto es la normalización. Las reglas de normalización muestran que la tabla se puede organizar de una forma mejor, para minimizar la redundancia de datos. Las reglas de normalización también muestran como cambiar la tabla para crear un esquema más simple.

La normalización da como resultado una base de datos que refleja la naturaleza de los datos, tiene simplicidad y contiene una pequeña redundancia de datos. Pasos concretos, descritos posteriormente, pueden normalizar siempre una base de datos no normalizada. La eliminación de los grupos repetitivos de datos puede cambiar una tabla no normalizada a una equivalente normalizada.

Cada una de las formas normales hace que una base de datos esté más o menos organizada. Cada una de las formas normales debe ser realizada antes de que la siguiente pueda ser aplicada. Una serie de pasos pueden aplicar niveles sucesivos de normalización a una base de datos no normalizada.

Formas Normales Como Guías

Hay que considerar que las formas normales son un conjunto de guías para construir un esquema. No hay nada en un esquema, ni en SQL, ni en el modelo relacional, ni en un RDBMS que imponga la normalización (se puede usar índices únicos para ayudar a mantener la normalización). Las reglas de normalización se desarrollaron como añadidos al modelo relacional. Construir una base de datos que no este normalizada es fácil.

Cuando no normalizar

Un diseño lógico de la base de datos debería siempre comenzar con un intento para cumplir con las formas normales que se describen más adelante. Una vez que tal diseño se ha realizado, puede ser desnormalizado para las posibles restricciones de manejo físico u operacional. Por ejemplo, es posible encontrar que una base de datos completamente normalizada sea más lenta o que añadirá más sobrecarga en la construcción de programas o informes.

Usualmente, una base de datos con la que esté trabajando debería estar al menos en tercera forma normal (la tercera forma normal se describe más adelante). En la práctica, muchas bases de datos bien diseñadas no alcanzan este objetivo.

LA normalización de una base de datos no es siempre apropiado. Por ejemplo, la siguiente tabla contiene la información de la dirección de los autores.

Nombre	Ciudad	calle	estado	c.p.
Paul Smith	San Francisco	1 Main St	CA	094102

Esta tabla tiene cinco columnas, nombre, calle, ciudad, estado y c.p. En cualquier registro donde la ciudad sea San Francisco, el estado será California. Esta puede tener varios nombres de ciudad duplicados.

Convirtiendo esta tabla en dos, una tabla para cod_postal y otra para nombre, se eliminan los nombres de ciudad duplicados.

Nombre

Nombre	Calle	CodPostal
Paul Mahler	1800 Market St ·257	94102

Cod_postal

CodPostal	estado	Ciudad
94102	CA	San Francisco

Cuando se usen los datos en esta base de datos, codPostal, ciudad, y estado casi siempre irán juntos. Por ejemplo, si este dato es usado para imprimir una lista de correos, todos estos campos aparecerán siempre juntos. Imprimir toda la información de ambas tablas requeriría unir las dos tablas juntas en cada momento en que los datos combinados sean necesarios.

Tampoco los códigos postales cambian a menudo. Separar el Codpostal, ciudad y estado en tablas separadas elimina todos los duplicados de nombres de ciudad y códigos postales encontrados en la tabla combinada. Ya que el código postal para la ciudad no es probable que vaya a cambiar, no hay una ventaja operacional en eliminar los datos duplicados.

Muy de vez en cuando será necesaria cambiar el codigo postal para una ciudad. La tabla combinada raramente cambiará aunque una ciudad tenga un nuevo estado. Los nombres de ciudad duplicados cambiarán probablemente, La información duplicada no cambia a menudo, así que la actualización no es tanto problema.

Reducir la redundancia de datos para mejorar la velocidad de actualización cuando no hay actualización no tiene sentido. En teoría, romper la tabla en dos más simples reduciría la redundancia de datos y mejoraría el rendimiento. En estas circunstancias, se introduciría únicamente una sobrecarga adicional.

Este ejemplo muestra que las formas normales son sólo una ayuda para el diseñador de la base de datos. Las formas normales siguieron guías para el diseño lógico de la base de datos, pero no siempre es apropiado seguir estas guías, debido a que existen base de datos a las que no se les pueden aplicar la normalización, ya que complicarían la manipulación de la información almacenada.

Primera Forma Normal

Esta primera forma normal (1FN) establece que una tabla no puede contener grupos repetitivos. Esto quiere decir que ningún grupo repetitivo de datos estará permitido en una tabla. Cada uno de los registros debe tener el mismo número de atributos. Cada una de las filas debe tener el mismo número de columnas. Por ejemplo, considerar la siguiente tabla de datos sobre álbunes de discos compactos que tiene varios grupos repetitivos:

no es una relación

Intérprete	Nombre de álbum
Talking Heads	Speaking in Toungues,
	Little Creatures,
	Stop Making Sense

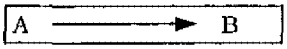
Cambiar una tabla con grupos repetitivos a una tabla sin ellos es simple. Se reemplazan los grupos repetitivos por filas completas. Por ejemplo, hacer para cada uno de los grupos repetitivos de la tabla anterior filas separadas que hagan que la tabla está en IFN (primera forma normal):

Intérprete	Nombre de álbum
Talking Heads	Speaking in Toungues,
Talking Heads	Little Creatures,
Talking Heads	Stop Making Sense

Dependencia Funcional

La dependencia funcional muestra las asociaciones lógicas de los datos.

La dependencia funcional significa que los valores para dos columnas están lógicamente asociados. Para dos columnas A y B, el valor de B será siempre el mismo para un valor particular de A. El valor de A determina el valor de B. Si B es funcionalmente dependiente de A, entonces A es el determinante de B. La siguiente figura muestra que " Determina funcionalmente B":



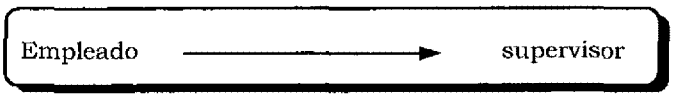
Esto puede también referirse a la relación que contiene. En este ejemplo, la dependencia funcional se lee "R1 punto A determina funcionalmente R2 punto B".

Por ejemplo, considerar una organización donde cada uno de los individuos sólo puede trabajar para un único jefe. Es importante para la estructura lógica del esquema que un individuo esté asociado únicamente con un jefe. El diseño lógico del esquema debe mostrar de alguna forma que el jefe depende del empleado.
Mirar la siguiente tabla con dos columnas, empleado y supervisor:

empleado	supervisor
Paul	Sam
Susan	Sam
George	Sam
Sabrina	Judy
Bill	Judy
Karen	Judy

En este ejemplo, el valor de los datos en la columna llamada supervisor depende del valor de los datos de la columna empleado. Selecciona cualquier valor para empleado únicamente determina un valor para supervisor. Por ejemplo, si el valor de empleado es Paul, entonces el valor de supervisor es siempre Sam.
Empleado es el determinante de supervisor.

Un simple diagrama de dependencias muestra esta dependencia funcional.



Si una persona puede trabajar para dos supervisores, esta dependencia funcional se pierde. Por ejemplo, Paul puede trabajar para dos diferentes personas, Samy y Judy.

La tabla podría ser como ésta:

empleado	supervisor
Paul	Sam
Susan	Sam
Paul	Judy
George	Sam
Sally	Judy
Harry	Judy
Jerry	Judy

Todavía existe una relación ya que existe una llave Primaria. La llave Primaria está ahora compuesta de las columnas empleado y supervisor. Está todavía en primera forma normal, ya que no existen grupos repetitivos, ni filas duplicadas. Sigue siendo una tabla.

El supervisor no es una dependencia funcional sobre los empleados. Para un único valor de la columna empleado (Paul) hay dos posibles valores de la columna supervisor (Sam y Judy).

Nótese que la dependencia funcional es parte de la estructura de los datos, no es parte del modelo relacional. Si Paul puede trabajar sólo para un supervisor, los datos en esta en esta tabla están mal, ya que es inconsistente con la estructura lógica de la empresa.

La tabla no sería un modelo exacto del mundo real. Nada en los datos la hace inconsistente con el modelo relacional. Los datos, a pesar de todo, encajan en la tabla pero ésta no modela el mundo real.

Dependencia Funcional y Columnas Compuestas

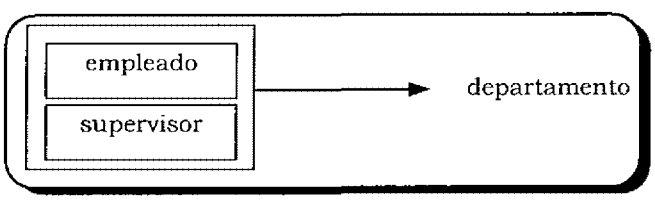
Una columna puede ser funcionalmente dependiente de dos o más columnas tomadas juntas como una llave compuesta. Por ejemplo, la siguiente tabla:

Supervisor	Empleado	departamento
Judy	Paul	a
Judy	Sam	a
Judy	Joe	a
Bill	Paul	b
Bill	Susan	b
Bill	Samantha	b

Aquí el departamento no es funcionalmente dependiente del nombre del supervisor, ya que Paul está trabajando ahora para dos supervisores en dos dependencias diferentes.

Poner dos columnas juntas, esto es, dos columnas compuestas, logra la dependencia funcional. El departamento es funcionalmente dependiente de las columnas compuestas {empleado, supervisor}.

Otro diagrama de dependencias funcionales muestra esto en la siguiente figura. El departamento es funcionalmente dependiente del supervisor y del empleado tomados juntos. La estructura de los datos especifica que cualquier persona puede trabajar para cualquier supervisor en un departamento dado. Esto es, el departamento depende de los empleados y del supervisor.



Llaves y Dependencia Funcional

Cada una de las otras columnas de una tabla dependen funcionalmente de una llave Primaria.

Una llave Primaria es un determinante para todas las columnas que no sean llaves Primarias. La razón es que seleccionar una llave Primaria individual siempre selecciona los mismos valores para las columnas no llave.

Conviene notar que nada en la definición de dependencias funcional dice que un determinante sea llave primaria. Cualquier columna puede depender funcionalmente de otra columna que no sea llave Primaria. Por ejemplo, aquí hay otra tabla.

Número	x	y
1	a	1
2	a	1
3	a	1

En esta tabla, la columna x no es, obviamente, una llave Primaria. Todavía y depende funcionalmente de x ya que y debe tener al valor de 1 siempre que x tenga el valor de a.

Dependencia Funcional Completa

Cualquier columna depende funcionalmente de forma completa de una columna compuesta cuando es funcionalmente dependiente de la columna compuesta pero no depende funcionalmente de cualquier parte de la columna compuesta tomada separadamente. Aquí hay ejemplo:

Supervisor	empleado	departamento
Judy	Paul	a
Judy	Sam	b
Judy	Joe	c
Bill	Paul	d
Bill	Susan	e
Bill	Samantha	f

En este ejemplo, la columna denominada departamento depende funcionalmente de la columna compuesta {supervisor, empleado}. En este conjunto de datos, cada uno de los empleados trabaja en un único departamento. Ya que el departamento también depende funcionalmente de la columna empleado, entonces no tiene dependencia funcional completa de las columnas compuestas {supervisor, empleado}.

Aquí hay otro ejemplo:

apellido	nombre	departamento
Mahler	Paul	1
Mahler	Susan	2
Mahler	Joe	3
Mahler	Kathy	4
Mahler	George	5
Jones	Paul	2

Aquí la columna departamento depende funcionalmente de la columna compuesta {apellido, nombre}. La columna departamento no depende funcionalmente sólo de la columna apellido o de la columna nombre. Ya que la columna departamento depende funcionalmente de las columnas compuestas, es funcionalmente dependiente de forma completa.

Las dependencias muestran la estructura de los datos en la base de datos. El hecho de que el departamento depende funcionalmente de forma completa de la columnas compuestas apellido y nombre significa que cada uno de los individuos es de un único departamento. (Esto también significa que no hay dos personas en la organización con el mismo nombre.) Ya que es importante para la estructura de los datos que un individuo esté en un único departamento, esta estructura debe ser representada de alguna forma en el esquema de la base de datos. La estructura de los datos en una base de datos puede ser especificada mediante el establecimiento de dependencias funcionales.

Dependencia Transcitiva

Hay una interdependencia entre estado y almacén. El estado depende transitivamente del número de embarque. El estado depende del número del embarque a través de su relación con almacén.
Esta proyección normaliza la tabla ejemplo en una tabla en 3FN.

estado

Almacén	estado
CA	abierto
NV	cerrado

almacén

embarque	almacén
s1	CA
s2	NV
s3	CA
s4	NV
s5	NV
s6	NV

Una proyección diferente también normaliza el ejemplo en dos tablas en 3FN. Esta segunda proyección descompone la relación entorno a la dependencia transitiva.

En el primer ejemplo, cada una de las proyecciones es independiente. Actualizar una tabla no requiere actualizar la otra (excepto donde cambie la ciudad). El estado para un almacén puede cambiar, independientemente del almacén para un embarque.

En el segundo ejemplo, las proyecciones no son independientes. El cambio del estado para un almacén requiere actualizar ambas tablas en el segundo ejemplo:

En el primer ejemplo, la unicidad de las llaves persiste, pudiéndose reunir las dos tablas a la vez en una relación en 2FN. Incluso después de varias actualizaciones, se pueden unir las dos tablas separadas en una tabla válida en 2FN.

En el segundo ejemplo, una actualización a una relación puede hacer imposible volver a juntar las dos relaciones en una tabla en 2FN con un inner join. A continuación hay un ejemplo.

almacén

embarque	Almacén
s1	CA
s2	NV
s3	CA
s4	NV
s5	NV
s6	NV

estado

embarque	Estado
s1	Abierto
s2	Cerrado
s3	Abierto
s4	Cerrado
s5	Cerrado
s6	Cerrado

Borrando el primer registro de la tabla estado se hace imposible volver a cambiar todos los registros de ambas tablas en una única tabla con un inner join. No habría ningún registro para un embarque en los resultados del join.

```
Select almacén.embarque, estado, almacén
  from almacén, estado
  where almacén. Embarque =
  estado.embarque
```

Aquí se presenta la tabla que resulta del join.

Embarque	estado	almacén
S2	cerrado	NV
S3	abierto	CA
S4	cerrado	NV
s5	Cerrado	NV

Debido a que la dependencia funcional del estado sobre el almacén no ocurre dentro de una única tabla, existe una restricción interrelacional entre las dos relaciones. Esto hace que sea fácil ver que es preferible hacer una descomposición donde las proyecciones sean independientes a una donde no lo sean.

Las tablas son independientes solamente si las dependencias funcionales de la tabla original pueden ser lógicamente deducidas de las dependencias funcionales de las tablas compuestas. También, los atributos que son comunes a las dos tablas deben ser una llave Primaria para al menos una de las tablas resultado.

Las tablas en la primeras proyección son independientes. El atributo común embarque es la llave primaria para la tabla almacén. Todas las dependencias funcionales en la tabla original aparecen en las dos proyecciones o pueden ser deducidas de ellas.
En el segundo ejemplo, las proyecciones no son independientes. La dependencia funcional del estado sobre el almacén no puede ser deducida de las dependencias funcionales en las tablas resultado.

Segunda Forma Normal

Una tabla que esté en segunda forma normal (2FN) debe estar, primero, en primera forma normal y todas las columnas no llave deben tener dependencia funcional completa de la llave primaria.

El ejemplo siguiente muestra una tabla usada anteriormente con dos columnas, nombre y extensión.

empleado	extensión
Paul	10
Susan	10
George	14
Sabrina	16
Bill	18
Karen	20

Esta tabla está en primera forma normal; no hay grupos repetitivos. Esta tabla está también en segunda forma normal, ya que la columna no llave extensión depende de la columna llave nombre.

El siguiente ejemplo añade otra columna a esta tabla que también depende de la llave primaria una columna de la hora a la que comienzan a trabajar los individuos. Esta columna se llama hora que indica si el individuo empieza a trabajar pronto o tarde.

empleado	extensión	hora
Paul	10	sí
Susan	10	no
George	14	sí
Sabrina	16	sí
Bill	18	sí
Karen	20	sí

Esta nueva columna contiene información relacionada con la llave primaria. Un individuo está autorizado para comenzar a trabajar pronto o tarde. Esto significa que añadir la nueva columna a la tabla no ha cambiado el que esté en 2FN.

Aquí está la tabla con la columna larga-distancia añadida otra vez.

empleado	extensión	larga-distancia
Paul	10	sí
Susan	10	sí
George	14	no
Sabrina	16	no
Bill	18	sí
Karen	20	sí

Esta tabla no está en 2FN. No está en 2FN ya que la columna larga distancia no está relacionada con la extensión. La columna larga distancia no depende funcionalmente de forma completa de la columna nombre.

La proyección de la tabla original en 1FN en dos tablas elimina cualquier dependencia funcional no completa. Esta proyección cambia la tabla 1FN en dos tablas en 2FN. La proyección crea dos tablas en 2FN, donde cada una de las dos tablas tiene una llave primaria y columnas relacionadas con la llave primaria. Esto normaliza la tabla en 1FN. Cada una de las nuevas tablas tiene dependencias funcionales más simples. El resultado aparece en las dos tablas siguientes:

teléfono

empleado	extensión
Paul	10
Susan	10
George	14
Sabrina	16
Bill	18
Karen	20

autorizaciones

extensión	larga-distancia
10	sí
14	no
16	no
18	sí
20	sí

El diagrama de dependencias funcionales predice el cambio de una tabla en 1FN en dos en 2FN.

Cada una de las dos dependencias se mueve a su propia tabla. La tabla en 1FN tiene dos dependencias funcionales diferentes. Actuando sobre la tabla en 1FN, se crea una nueva tabla para cada una de las dependencias.

Nótese que esta descomposición se realiza sin pérdida de información. Las dos tablas que resultan pueden fácilmente recombinarse en una única tabla.

Terecera Forma Normal

En una tabla en 2FN, todas las columnas no llave deben tener dependencias funcional completa con respecto a la llave primaria. Una tabla debe estar en 2FN antes de estar 3FN.

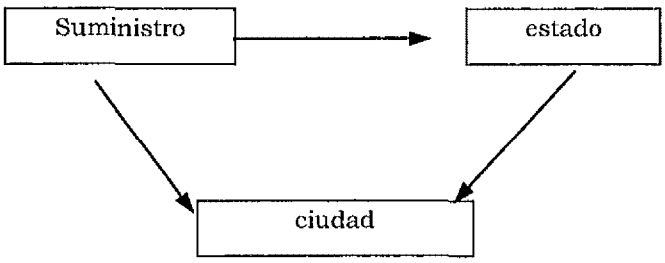
En una tabla en 3FN, todos los elementos no llave deben ser mutuamente independientes. Las columnas no llave son todas mutuamente independientes si ninguna de ellas es funcionalmente dependiente de otra. Para que una tabla esta en 3FN, ninguna de las columnas no llave deben depender funcionalmente de cualquiera de las otras columnas no llave.

Aquí hay una tabla con tres columnas, suministrador, estado y ciudad.

Suministrador	estado	ciudad
s1	10	Detroit
s2	20	Boston
s3	20	Boston
s4	30	Dallas
s5	30	Dallas
s6	10	Detroit

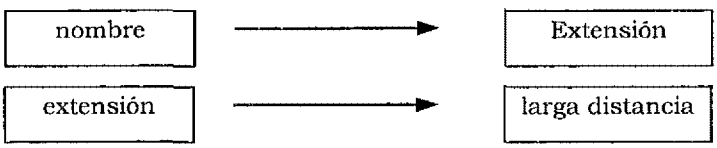
En este conjunto de datos, cada uno de los suministradores corresponde a una ciudad determinada. Cada una de las ciudades tienen un estado específico. El estado depende de la ciudad. La ciudad depende del suministrador.

Debido a que la ciudad depende del suministrador y el estado depende de la ciudad, el estado también depende del suministrador. Si se conoce el suministrador, se puede determinar el estado, pero sólo a través de la ciudad. El estado sólo depende del suministrador transitivamente a través de la ciudad.



Esta clase de dependencias transitiva complica la actualización de la tabla. Si el estado cambia para una ciudad, cada uno de los registros para esa ciudad deben ser cambiados.

La inserción de un nuevo registro puede ser difícil. Añadir un registro para notar que una ciudad determinada tiene un estado particular es imposible hasta que haya suministrado a la ciudad. Por ejemplo, no se puede añadir la información de que San Francisco tiene de estado 70 hasta que haya un suministrador para San Francisco.



Borrar la única fila para una ciudad en particular hace perder dos elementos de información: (1) la información de que un suministrador es de una determinada ciudad, y (2) la información de que una ciudad tiene un estado determinado.

Al proyectar la tabla en 2FN en dos tablas separadas en 3FN resuelve este problema.

La proyección separa cada una de las dependencias funcionales en su propia tabla. Las dos tablas que resultan son estas:

suministrador	ciudad
s1	Detroit
s2	Boston
s3	Boston
s4	Dallas
s5	Dallas
s6	Detroit

estado	ciudad
10	Detroit
20	Boston
30	Dallas

Cada una de estas dos tablas está en 3FN. Cada una de las tablas está en 1FN ya que no hay grupos repetitivos. Cada una de las tablas está en 2FN ya que las columnas no llave son

funcionalmente dependientes de la llave primaria. Cada una de ellas está en 3FN ya que ninguna columna no llave es funcionalmente dependiente de otra columna no llave. Está en tercera forma normal ya que cada una de las columnas es mutuamente independiente.
En general, se puede manipular las tablas más fácilmente con dependencias simples. Éstas son más fáciles de actualizar. Este es el beneficio de las formas normales. Las formas normales ayudan a asegurar que las tablas, y un esquema, son simples estructuralmente.

Elección de una Descomposición

No se pueden descomponer algunas tablas en componentes independientes. Estas tablas se llaman atómicas. Se pueden descomponer algunas tablas, pero no hay por qué hacerlo necesariamente.

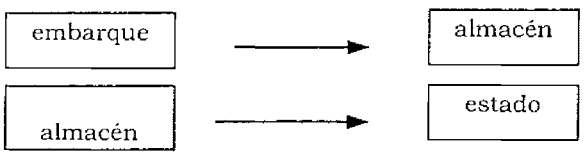
A menudo hay más de una posible descomposición de una tabla. Examinemos la siguiente tabla. En este ejemplo, los embarques son hechos desde un almacén y cada uno de los almacenes tiene un estado de abierto o cerrado.

embarque	estado	almacén
s1	abierto	CA
s2	cerrado	NV
s3	abierto	CA
s4	cerrado	VN
s5	cerrado	VN
s6	cerrado	VN

La entrada bajo el estado depende de la entrada bajo el almacén. Cada una de las entradas bajo almacén y cada uno de los almacenes tienen un estado de abierto o cerrado.

La entrada bajo almacén depende del número bajo embarque. Cada elemento de embarque navega desde un único lugar bajo almacén.

El estado depende del número de embarque. Cada uno de los embarques tienen un estado asociado de abierto o cerrado. Nótese que esto probablemente no es parte de la estructura lógica de los datos. Es razonable para el almacén que tenga un estado abierto o cerrado. No es razonable para un embarque que tenga un estado de abierto o cerrado. El siguiente diagrama muestra las dependencias de esta tabla.



FNBC (Forma Norma de Boyce Codd)

Una tabla está en Forma Normal de Boyce-Codd (FNBC) cuando todas las dependencias funcionales son sobre las llaves Primarias. Este es un requisito más fuerte que el de la 3FN, donde las columnas no llave tienen que ser mutuamente independientes funcionalmente. En una tabla en FNBC no hay dependencias funcionales sobre las columnas que no sean llaves Primarias. Una relación está en FNBC cuando todo determinante de una relación es una llave Primaria.

La FNBC se aplica para la normalización de una tabla que esté en 3FN y tenga las siguientes propiedades:

múltiples llaves Primarias
llaves Primarias compuestas
columnas compartidas entre las llaves compuestas

Aquí hay un ejemplo. La siguiente tabla contiene datos sobre los estudiantes y los profesores.

estudiante	asignatura	profesor
Smith	Lectura	Frank
Smith	Escritura	Sally
Jones	Lectura	Frank
Jones	Escritura	Jane

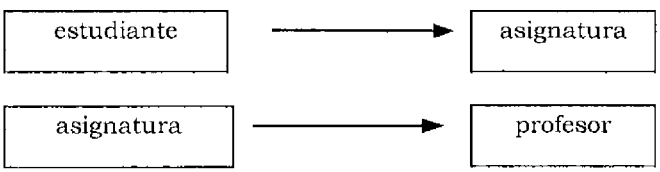
Existen tres restricciones lógicas sobre esta tabla:

Un estudiante recibe una asignatura de un único profesor.
Cada uno de los profesores enseña sólo una signatura.
Una asignatura puede ser enseñada por más de un profesor.

Hay dos llaves Primarias para esta tabla. Cada una es llave compuesta:

Llaves Primarias
{estudiante, asignatura}
{estudiante, profesor}

Las dependencias funcionales aparecen en el siguiente diagrama de dependencias. La llave compuesta (estudiante, asignatura) es un determinante para profesor. El atributo profesor es un determinante para asignatura. La llave compuesta (estudiante, asignatura) es un determinante para profesor.



Si la llave primaria es (estudiante, asignatura), la relación está en 1FN ya que no hay grupos repetitivos. La relación está en 2FN ya que el profesor depende funcionalmente de forma completa de la llave primaria (estudiante, profesor). Hay únicamente un atributo no llave, profesor. La relación está en 3FN ya que los atributos no llave son independientes.

Si la llave primaria es (estudiante, profesor), la relación está 1FN ya que no hay grupos repetitivos. La relación está en 2FN ya que la asignatura depende funcionalmente de forma completa de la llave primaria (estudiante, profesor). La relación está en 3FN ya que sólo hay un atributo no llave asignatura, así que todos los atributos no llave son mutuamente independientes.

La tabla no está en FNBC ya que hay tres determinantes, (estudiante, asignatura), (estudiante, profesor) y profesor, pero profesor no es una llave Primaria.

Proyectando, se normaliza esta tabla a dos en FNBC.

profesor	asignatura
Frank	lectura
Sally	escritura
Jane	escritura

estudiante	profesor
Smith	Frank
Smith	Sally
Jones	Frank
Jones	Jane

Cuarta forma normal

Algunas tablas no pueden ser normalizadas sobre la base de las dependencias funcionales. La cuarta forma normal se apoya sobre dependencias multivaluadas.

A continuación se presenta un ejemplo donde un profesor puede enseñar en más de un curso. En un curso se puede utilizar más de un libro de texto. Un curso debe ser realizado con todos los libros de texto. Un curso debe ser realizado con todos los libros de texto requeridos.

profesor	curso	libro
Frank	lectura	Catcher in the Rye
Frank	lectura	Justine
Frank	lectura	Red Badge OF Courage
Sally	lectura	Catcher in the Rye
Sally	lectura	Justine
Sally	lectura	Red Badge OF Courage
Frank	escritura	Cry OF the Wolf
Frank	escritura	The Man in the Iron Mask

En esta tabla existe sólo una llave Primaria (profesor, cursor, libro). No hay dependencias funcionales ya que todas las columnas forman parte de la llave Primaria.

La tabla está en FNBC. Incluso así, es claramente difícil de actualizar, Si otro profesor, Jim, comienza a enseñar lectura, tres registros se deberán añadir a la tabla, uno por cada uno de los libros de texto. Estos tres registros son:

Jim lectura	Catcher in the Rye
Jim lectura	Justine
Jim lectura	Red Badge OF Courage

Un curso no tiene un único profesor. El profesor no es dependiente del curso. Sin embargo, un conjunto bien definido de profesores enseña cada uno de los cursos. El mismo conjunto de profe4sores enseña siempre un único curso. Seleccionado un curso, siempre resulta el mismo conjunto de profesores que enseñan el curso. En el ejemplo, si el curso de la lectura, los profesores son siempre Frank y Sally. Esta tabla es difícil de actualizar debido a las dependencias multivaluadas. El profesor es multidependiente del curso. El texto es multidependiente del curso.

A continuación se presenta la tabla después de añadir el nuevo profesor.

Profesor	curso	libro
Frank	lectura	Catcher in the Rye
Frank	lectura	Justine
Frank	lectura	Red Badge OF Courage
Sally	lectura	Catcher in the Rye
Sally	lectura	Justine
Sally	lectura	Red Badge OF Courage
Frank	escritura	Cry OF the Wolf
Frank	escritura	The Man in the Iron Mask
Frank	escritura	The Scarlet Letter
Jim	lectura	Catcher in the Rye
Jim	lectura	Justine
Jim	lectura	Red Badge OF Courage

Imaginemos una relación con tres atributos, x, y, z. El atributo z sólo puede ser dependencia multivaluada de x si el conjunto de valores seleccionados para un valor dado del par de atributos

(x,y) depende del valor x y no depende del valor de y. Esta definición requiere que una relación deba tener al menos tres atributos para ser dependencia multivaluada.

Por ejemplo, selcccionando el curso lectura se obtiene el conjunto de profesores Frank, Sally y Jim. Seleccionados los valores de (lectura, Justine) para el par de atributos curso y libro de texto también se obtiene el conjunto de profesores Frank, Sally y Jim. Este ejemplo muestra que el conjunto de profesores seleccionados depende sólo del curso, no de una combinación de curso y libro.

Una proyección en dos tablas normaliza la tabla única.

Profesor
curso profesor
texto
curso texto

Separar las dependencias multivaluadas en dos tablas hace el esquema más simple. Después de la proyección, la base de datos es más fácil de actualizar. Después de la proyección, la base de datos es más fácil de actualizar. Después de la proyección, no hay dependencias multivaluadas. Una tabla está en 4FN solamente cuando no existen dependencias multivaluadas.

Otras Formas Normales

La teoría de las dependencias es la teoría de la normalización y aspectos relacionados con la normalización. Existen otras formas normales. Por ejemplo, existe una quinta forma normal (5FN) y una (3,3 FN)

Estas formas normales avanzadas salen del alcance de este trabajo. Las formas normales presentadas en las secciones previas ilustran problemas que solamente se pueden confrontar en las aplicaciones de bases de datos reales.

Es poco probable que se contrasten los problemas ilustrados por las formas normales más altas en una base de datos actual. La aplicación de las formas normales descritas anteriormente, particularmente a través de las formas normales descritas anteriormente, particularmente a través de la FNBC, debería crear una base de datos diseñada correctamente.

Redundancia de Datos y Actualización

La normalización está referida al significado de los datos en la base de datos. Las formas normales son guías para ayudar al diseñador de la base de datos a diseñar la estructura lógica de los datos para una empresa en el esquema de la base de datos.

Es mejor tener tablas menos complejas y esquemas menos complejos. Un esquema será simple cuando todas las columnas no llave en cualquier tabla contengan información que sea sólo referente a la llave primaria de la tabla.

Conocer la teoría de la base de datos, permitirá crear aplicaciones bastante óptimas; por el contrario sin la información de este capítulo, cuando se realizan consultas mal diseñadas los tiempos de respuesta pueden ser muy altos, así mismo cuando se crean base de datos mal diseñadas la actualización a las tablas puede llegar a ser muy lenta.

Capítulo 3

Herramientas de PowerBuilder

Objetivos
La base para desarrollar aplicaciones con PowerBuilder son los objetos y controles, por esto se buscaen este capítulo mostrar los diferentes objetos ycontroles e identificar el uso que se les puede dar a cada de estos objetos y controles.

Powerbuilder es un producto que inicialmente fue creado por PowerSoft, este lenguaje introdujo nuevas ideas de desarrollo enfocadas a la reutilización de código. Además de agregar ideas de diseño simples para permitir a cualquier persona generar sus aplicaciones sin tener que ser experto en desarrollo de aplicaciones. Actualmente PowerSoft pertenece a Sybase, con lo cual ha mejorado el tiempo de acceso a datos, de aplicaciones Powerbuilder hacia los motores de base de datos de Sybase.

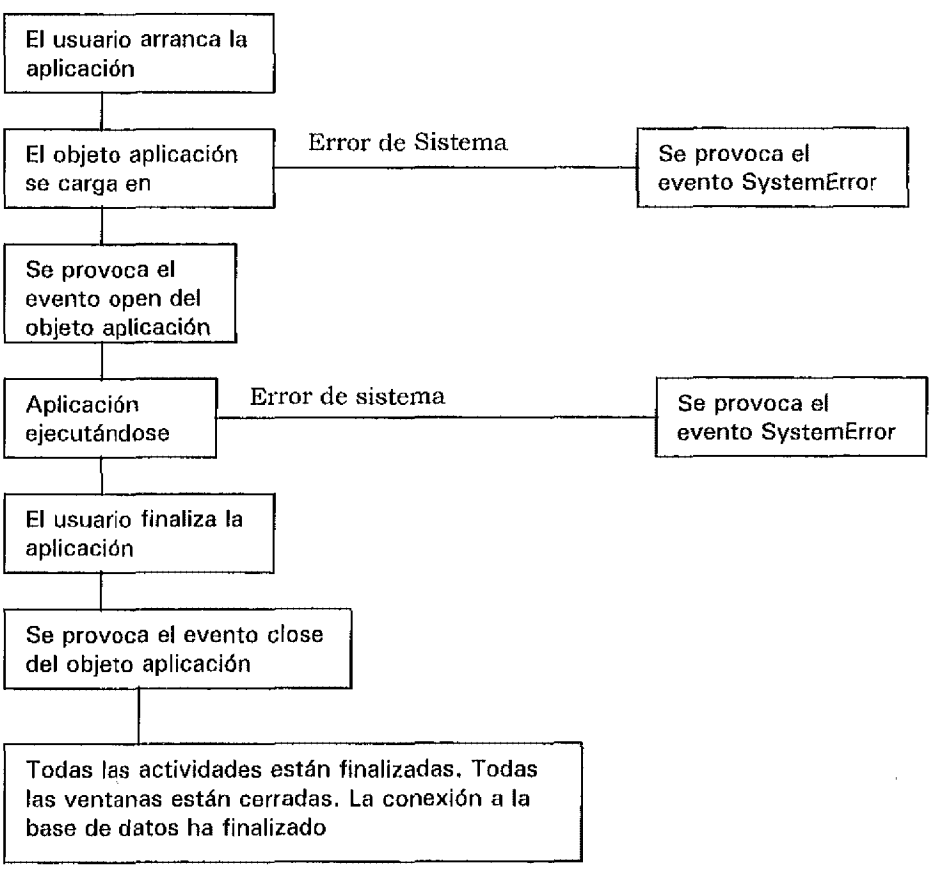
Los requerimientos mínimos con los cuales Powerbuilder puede trabajar son (aunque estos pueden variar según la versión de Powerbuilder):

√ Procesador 486
√ Memoria RAM 16
√ Windows 3.x aunque puede correr sobre cualquier plataforma, excepto DOS

Powerbuilder tiene un conjunto de Painters. Cada painter es una herramienta con un propósito particular. Cada icono situado en la PowerBar permite el acceso a un diferente painter.

3.1 Aplication Painter

El marco general para cualquier objeto que se construya con Powerbuilder es la aplicación. Una aplicación PowerBuilder es un conjunto de objetos, y cuando se están utilizando en una aplicación en ejecución, tienen una funcionalidad. Todos se encuentran en una o en varias librerías PowerBuilder.



El objeto aplicación contiene el punto de comienzo de la aplicación, cuando esta se ejecuta, el objeto aplicación se carga en memoria y empieza a ejecutarse.

El diagrama de la página anterior muestra la secuencia de eventos que tienen lugar cuando se ejecuta una aplicación.

Una aplicación no es un objeto visual. La única parte de la aplicación que se ve alguna vez es el icono asociado con la aplicación.

Cada objeto aplicación tiene los siguientes atributos asociados:
- √ El nombre de la aplicación.
- √ El icono de la aplicación.
- √ Ruta de acceso de la librería.
- √ El tipo de letra de texto por defecto.
- √ Variables globales.
- √ Funciones externas globales.

Ventanas y controles

La interfaz entre un usuario y una aplicación Powerbuilder es una o más ventanas. El usuario interactúa con la aplicación en ejecución a través de la ventana activa. Las ventanas pueden mostrar información o responder a las acciones del teclado o del ratón. El usuario puede introducir información en una ventana. Durante el uso de una aplicación, las ventanas se abren y cierran como respuesta a una orden del usuario.

Estilo

Una ventana, como cualquier otro objeto Powerbuilder, tiene atributos y eventos. Los atributos de la ventana controlan su apariencia y comportamiento. Por ejemplo, una ventana tiene atributos que controlan su tamaño y forma, o posiblemente su capacidad de ser redimensionada o ser visible. Algunos de los atributos de una ventana son visuales, como por ejemplo, el tamaño de la ventana, la forma de la ventana o la presencia de su barra de título.

Los atributos de una ventana, tomados en conjunto, determinan el estilo de la ventana, quien a su vez determina su apariencia y comportamiento.

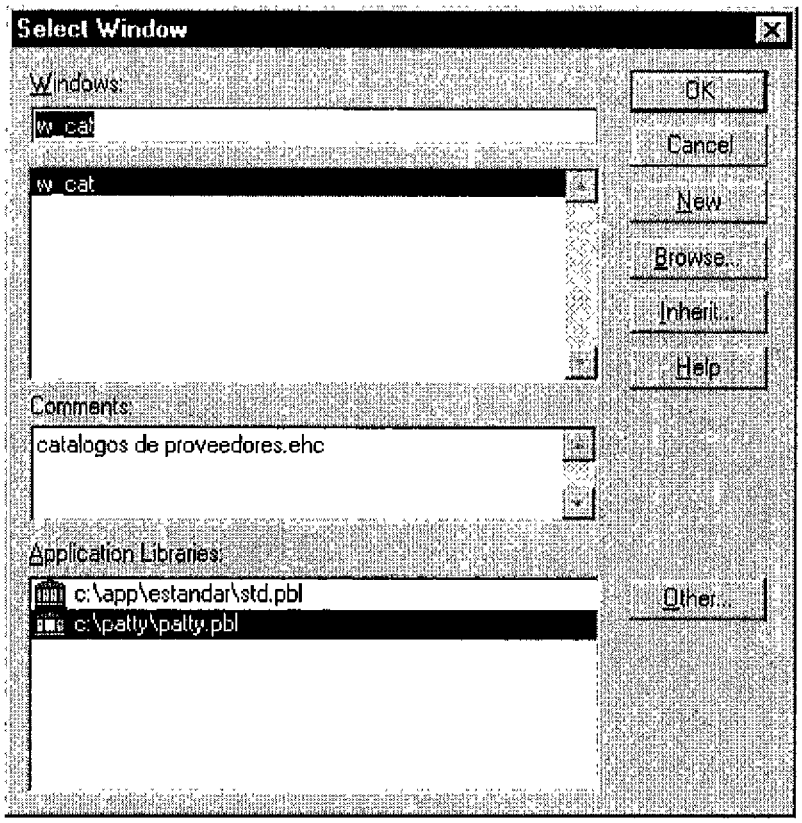
Controles

Cualquier objeto que aparezca en una ventana es un control. Los controles son objetos situados dentro de una ventana. Los controles muestran datos, aceptan datos o validan datos. Los controles responden a una acción del usuario, como por ejemplo, un clic de ratón.

Los controles, como cualquier otro objeto, tienen sus propios atributos y eventos. Algunos controles son objetos estándar Windows, otros son particulares de Powerbuilder. Además, puede crear sus propios controles personalizados.

3.2 Painter Window

Utilice el Window Painter para crear nuevas ventanas o modificar las existentes. Para crear una nueva ventana arranque el Window Painter, construya una nueva ventana y archive el objeto ventana en una librería. Para arrancar el Painter Window, utilice el icono de ventana de Powerbuilder o la Powerbar.



Una vez seleccionada la aplicación, se puede utilizar el Window Painter para crear la primera ventana de la aplicación. Para crear una nueva ventana, haga clic en el botón New de esta figura.

Esta es una ventana vacía, que creará con este painter, será el prototipo de su aplicación. Se puede utilizar la ventana prototipo para crear otras ventanas mediante la herencia.

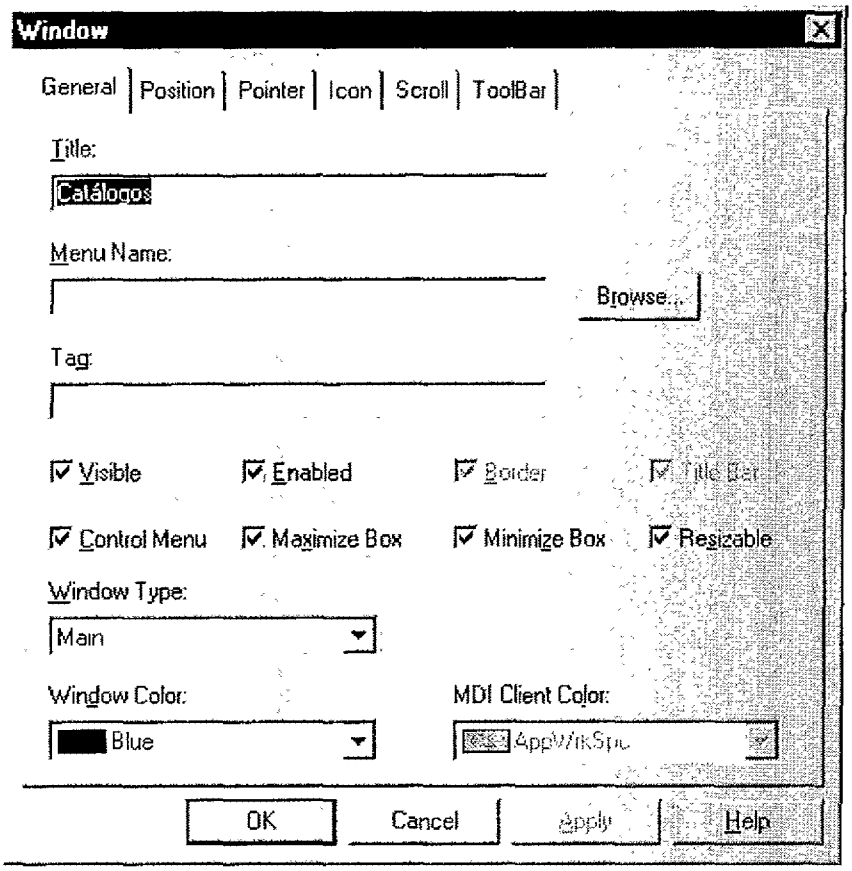
Para archivar este objeto ventana en su librería Powerbuilder, seleccione la opción Save del menú File, y por ejemplo archive con w_proto. Observe que hay también un espacio para adjuntar un comentario con el nuevo objeto ventana. Observe también que aparece la ruta de acceso a la librería.

En este ejemplo, se ha creado una clase nueva que desciende del objeto Powerbuilder window. En la continuación del ejemplo, se añadirán controles al objeto w_proto.

Esta ventana de w_proto puede no ser utilizada directamente en la aplicación. En vez de ello, se puede crear una segunda ventana, w_principal que hereda los atributos del objeto w_proto. Esta ventana w_principal, es la ventana principal de la aplicación que se está construyendo.

Se ha creado una nueva base w_proto y se ha archivado el objeto w_proto en la librería de la aplicación. Ahora se modifica la ventana incremento su tamaño. Sitúe el cursor en la esquina inferior de la ventana. Cuando el cursor cambie a una flecha con doble punta, haga clic y mantenga pulsado el botón izquierdo del ratón. Mientras que presione el botón del ratón, agrande la ventana. Hay que destacar que se puede arrastrar la ventana para hacerla más grande que el espacio de trabajo:

A continuación, sitúe el cursor en algún lugar de la ventana que acaba de agrandar y haga doble clic. Esto provoca que se muestre la ventana Window Style (ventana de estilo). Haga clic en el botón Cancel para cerrar esta ventana.



Como forma alternativa de mostrar esta ventana, sitúe el cursor en la ventana que se está construyendo haga clic y mantenga pulsado el botón derecho del ratón. Aparece una ventana de respuestas con una lista de atributos. Una marca de comprobación indica los atributos seleccionados. Para seleccionar o deseleccionar el atributo de estilo, sitúe el cursor sobre el nombre Style y libere el botón derecho del ratón.

Para evitar una selección, libere el botón fuera de la ventana de respuesta.

Teclas de acceso rápido

Las teclas de acceso rápido del Window Painter incluyen:

Acción	Teclas de acceso rápido
Borrar	Del
Cambiar a	Ctrl+Esc o Alt+Esc
Cerrar	Ctrl+F4
Copiar	Ctrl+C
Cortar	Ctrl+X
Cursiva	Ctrl+I
Depurar	Ctrl+D
Deshacer	Ctrl+Z
Devolver el foco al control	Ctrl+O
Duplicar	Ctrl+T
Editar texto	Ctrl+E
Editor de archivos	Shift+F6
Ejecutar	Ctrl+R
Ejecutar ventana	Ctrl+W
Justificación de texto a la derecha	Ctrl+G
Justificar texto a la izquierda	Ctrl+L
Negrita	Ctrl+B
Pegar	Ctrl+V
PowerPanel	Ctrl+P
Presentación preliminar	Ctrl+Shift+W
Próxima ventana	Ctrl+F6
Procedimiento	Ctrl+S
Seleccionar todo	Shift+A
Subrayado	Ctrl+U
Tabulación hacia atrás	Shift+Tab
Texto centrado	Ctrl+N
Tipo de fuente	Ctrl+F

Algunas de las anteriores pueden variar dependiendo de la versión.

Tipos de ventana

Observe que seis tipos de ventana son estilos de ventana. Estos son:

main (principal)
child (hija)
popup (emergente)
response (de respuesta)
estructura MDI (MDI Frame - Interfaz Múltiple de Documento)
estructura MDI con microayuda (MDI Frame with Microhelp).

Ventanas principales

Una ventana main (principal) es la primera ventana que aparece en una aplicación. Es una ventana aislada, y opera independientemente de otras ventanas de aplicación.

La ventana w_proto es realmente una ventana principal. Esta es la elección adecuada para w_proto. Haga doble clic en la ventana para mostrar las selecciones de estilo. Observe que el tipo de ventana seleccionado es Main.

Ventanas hijas

Una ventana child (hija) muestra los estados de los elementos de la ventana padre. Una ventana hija es secundaria respecto al padre, solamente se le permite existir dentro del marco de la padre. Por lo tanto, la hija se cierra automáticamente cuando se cierra la padre. Una hija no es nunca una ventana activa. Cualquier usuario que intente mover la hija fuera del marco de la ventana padre

recortará la hija. Por último, al minimizar la hija, ésta se reduce a un icono. El icono aparece dentro de la ventana padre.

Ventanas emergentes

Las ventanas popup (emergentes) normalmente, muestra información, como por ejemplo, ayuda en línea. Una ventana emergente siempre se muestra en una ventana padre. La emergente no puede estar detrás de la ventana padre. Al contrario de una ventana hija, se puede mostrar fuera del marco de la ventana padre. Al minimizar la ventana emergente, se muestra el icono fuera de la ventana padre cerca de la parte inferior de la pantalla. Al minimizar la ventana padre, también se minimiza la ventana emergente.

Ventanas de respuesta

Una ventana response (de respuesta) muestra un mensaje de error o aviso. Una ventana de respuesta puede requerir del usuario información crítica. Una ventana de respuesta a menudo solicita al usuario información necesaria antes de poder continuar con el trabajo.

Una ventana de respuesta es una aplicación modal; el usuario no puede minimizar otra ventana. Puesto la ventana es modal, el usuario no puede seleccionar otra ventana de la aplicación de la ejecución mientras se muestra la ventana de las respuestas. El usuario puede utilizar ventanas de otras aplicaciones en ejecución mientras se presenta una ventana de respuestas. La ventana de respuestas permanece activa hasta que el usuario responda, es decir, permanece visible hasta que el usuario realiza una actividad especifica por la ventana de respuestas.

Ventanas MDI

Las descripciones de la ventana MDI (Interfaz Múltiple de Documentos -Múltiple Document Interface) se incluyen en la sección dedicada a aplicaciones MDI.

Selección de un icono

Una ventana minimizada aparece como un icono. Se puede seleccionar cualquier icono que aparezca cuando se minimiza una ventana.

En el Window Painter, haga clic en el botón derecho del ratón: aparece la ventana de estilo, que es una ventana de respuesta.

Sitúe el cursor sobre el icono llamado Icono y libere el botón derecho del ratón. Aparece la ventana de selección de iconos.

NOTA: Se puede utilizar una técnica parecida para cambiar el icono de la aplicación.

Minimice la ventana de Application Painter. Use el icono del Application Painter del Powerbar o el PowerPanel. Cuando se abra la Application Painter, utilice la selección de icono de la barra de painter o elija la selección del Application Icono del menú Edit. Aparece la misma ventana de selección de iconos. Utilice esta ventana para cambiar el icono de la aplicación.

Selección de un puntero

Se puede cambiar el puntero de la ventana del menú de estilos emergentes. Elija la opción Pointer selection del menú. Se puede utilizar esta opción para cambiar el puntero a cualquier puntero mostrado en la lista artículo Pointers.

Tamaño y posición

Active las cajas de verificación Center Horizontally y Center Vertically para tener la ventana abierta en el centro de la pantalla. Puede establecer el tamaño con el que se abrirá la ventana. También puede arrastrar la ventana ejemplo "tirando" de su esquina. Y puede seleccionar el estado de apertura de la ventura normal, maximizada o minimizada. Haga clic en el botón OK para archivar sus cambios.

Unidades de Powerbuilder

Las unidades Powerbuilder (PBU) miden objetos. Una PBU es 1/32 del tamaño de ancho de la fuente del sistema para medidas horizontales y 1/64 de la altura de la fuente del sistema para medidas verticales. Medir objetos en PBU hace más fácil la construcción de aplicaciones de tamaño parecido sobre las pantallas de diferentes tamaños o proporciones de aspecto.

3.3 Controles

A una ventana se le pueden añadir diferentes controles Windows estándares, controles Powerbuilder especializados, o definidos por el usuario. A continuación se listan los objetos y controles que se puede añadir una ventana.

Nombre	Descripción
CheckBox(caja de verificación)	Una caja cuadrada pequeña que se usa para seleccionar una opción.
CommandButton(botón de órdenes)	Un botón de texto; cuando se hace clic sobre él provoca una acción.
DataWindow(ventana de datos)	Un control de datawindow es un lugar donde se pone un objeto datawindow.
DropDownListBox (caja de lista desplegable)	Usa una caja de texto y una flecha para permitir al usuario mostar las opciones disponibles.
EditMask (máscara de edición)	Determina el formato y la validación de una línea de edición.
Graph (gráfico)	Un gráfico derivado de datos de programa.
GroupBox (caja de grupo)	Un rectángulo con un título que se usa para cambiar un grupo de botones relacionados.
HscrollBar (barra de Desplazamiento horizontal)	Una barra horizontal que se usa para desplazar la información que no cabe en una ventana.
Line (línea)	Una línea recta, continua o disminuye.
List Box (caja de lista)	Muestra una lista de elementos para una selección de usuarios.
Menultem (elemento de menú)	Un elemento en un menú.
MDI Client (cliente MDI)	El área dentro de un marco MDI; el nombre de este marco es MDI.
MiltiLine Edit (edición de Múltiples líneas)	Una caja de texto que permite múltiples líneas de entrada de datos.
Oval (óvalo)	Un óvalo o figura circular.
Picture (figura)	Un objeto gráfico, como por ejemplo, una mapa de bits.
Picture Button (botón con figura)	Un botón que muestra dentro de él un objeto gráfico, como por ejemplo, un mapa de bits.
RadioButton (botón de opción, de dial de radio)	Un botón; se utiliza normalmente en grupos para seleccionar una opción de entre varias.
Rectangle (rectángulo)	Un rectángulo.
RoungRectangle (rectángulo redondo)	Un rectángulo con las esquinas redondeadas.
SingleLineEdit (edición de una línea)	Una caja de texto para traducir una sola línea de texto.
StaticTex (texto estático)	Una cadena de texto; el usuario no puede cambiar un elemento de texto estático.
UserObject (objeto de usurio)	Un objeto definido por el que desarrolla la aplicación, que hereda sus atributos y eventos de un objeto estándar Powerbuilder.
VScrollBar (barra de desplazamiento vertical)	Una barra vertical para desplasar la información que no cabe completamente en la pantalla.
OLE 2.0	Objeto que soporta la versión 2.0 de Microsoft OLE (Enlace e Incrustación de Objetos -Object Linking and Embedding).

La tabla siguiente sugiere usos para los diferentes controles y objetos de la tabla anterior.

Función	Objeto o control
Inicia una acción.	CommandButton Picturebutton
Existencial. Establece un estado de existencia.	CheckBox RadioButton GroupBox
Muestra datos.	DataWindowControl Graph Control
Introduce datos.	DataWindowControl
Cabeceras y etiquetas.	SingleLineEdit EditMask MultiLineEdit StaticText
Lista de elementos.	ListBox DropDownListBox
Desplazamiento.	HorizaontalScrollBar VerticalScrollBar
Gráficos.	Picture, Rectangle, Rounded Rentangle, Oval, Line
Control personalidad.	UserObject Control

Situando un objeto en una ventana

En Window Painter se puede seleccionar un objeto y situarlo sobre una ventana. También se puede seleccionar un nombre de control del menú Controls para situar un control en una ventana.

Añada un control de Picture a la ventana w_proto. Despliegue el menú Controls y libere el botón del ratón cuando el cursor esté sobre el elemento de menú Controls y libere el botón del ratón cuando el cursor esté sobre el elemento de menú Picture. Tambien puede hacer clic en el icono figura de la barra de iconos.

Después de seleccionar el tipo de objeto figura del menú Controls o la barra de iconos, mueva de nuevo el cursor sobre la ventana que está diseñado. Observa que la forma de cursor ha cambiado a una cruz cuadrada. Mueva el cursor a donde quiera que aparezca la figura y haga clic en el botón izquierdo del ratón. Aparece una caja. Esta es la caja en la que pronto situará una figura.

Mueva el cursor a una esquina de la caja. El cursor se convierte en una flecha. Haga clic y mantenga pulsado el botón izquierdo del ratón, y arrastre el cursor. Esto agrandará el control Picture.

Ahora que ha dimensionado su control Picture, puede incluir el mapa de bits asociado a él. Sitúe el cursor en la mitad del control Picture y haga doble clic en el botón izquierdo del ratón.

Utilice el menú Directorios para seleccionar el directorio que contenga el archivo.bmp. Para seleccionar este archivo, mueva el cursor al nombre de la ventana de Nombre de archivo y haga clic en el botón izquierdo del ratón. Se destaca el nombre del archivo y se copia al área File Name. Haga clic en el botón Aceptar para seleccionar el mapa de bits.

Observe que el último carácter del nombre, número, se destaca. Powerbuilder ha asignado un nombre por defecto al objeto de la figura. El prefijo p indica que el objeto es una figura. El número de la figura indica que es el primer objeto figura situado en esta ventana. El nombre será ahora p_logo. El mapa de bits queda asociado al control Picture.

Añadir un botón con la figura

Del menú Controls, seleccione la opción PictureButton. Mueva el cursor a la ventana w_proto. Haga clic una vez bajo el logotipo. Aparece un botón con figura en la posición del cursor.

Si el botón no esta centrado bajo el logotipo, puede moverlo. Sitúe el cursor sobre el PictureButton. Haga clic y mantenga pulsado el botón izquierdo del ratón. Arrastre el botón a la posición deseada y libere el botón izquierdo del ratón.

Observe que la figura del botón tiene la etiqueta none. Para hacer desaparecer el none y poner la figura en el botón, mueva el cursor al botón y haga doble clic en el botón izquierdo del ratón.

Utilice la ventana de dirección si es necesario para encontrar el directorio que contenga el archivo exit.bmp haga clic en el nombre de archivo exit.bmp; esto situará su nombre en la caja File Name (nombre de archivo). Cambie el nombre PictureButton de pb_1 a pb_exit. El PictureButton tendrá ahora el exit.bmp en lugar de texto none. Si todavía aparace la palabra none, quiere decir que olvidó borrarla de la caja de diálogo.

Ahora que ha situado un botón de salida en la ventana w_proto, necesita escribir un procedimiento para el botón. Sitúe el cursor sobre el PictureButton para salir, y haga clic en el botón derecho de su ratón.

Seleccione la opción script. También podría haber hecho doble con el botón izquierdo sobre el PictureButton y haber elegido la opción script en la ventana que aparece. Se arranca el Script Painter. Introduzca el siguiente procedimiento para el botón Exit.

```
//   Solicita al usuario confirmar la salida y sale de la aplicación
If   MessageBox('ExitApplication','Exit?',Question!,YesNo!) = 1 then
  DISCONNECT;
  close (Parent)
End If
```

Este procedimiento muestra una caja de mensajes. El botón presentado para seleccionar salir de la aplicación. Si el usuario elige salir de aplicación, el procedimiento se le llama a la función cerrar. La llamada a la función close (Parent) provoca la ejecución de la función cerrar del objeto padre del botón. En este caso, el padre del objeto PictureButton es la ventana que muestra el PictureButton.

Compile el procedimiento tecleando Ctrl-L o seleccionado Compile Script del menú compilar del Window Painter. Cuando se haya copilado el procedimiento con éxito. Ahora, si que quiere puede ver la forma preliminar de la ventana prototipo. Seleccione la opción Preview del menú Design, o teclee Ctrl-Shift-w. Aparecerá la ventana como se ve una aplicación en ejecución.

Puede hacer clic en el PainterButton Exit. A pesar del clic del botón, la ventana no se cierra ya que este es el modo de presentación preliminar. Para cerrarla, haga doble clic en la caja de control de la esquina superior izquierda o despliegue el menú y seleccione close, o teclee Crtl-F4. Al cerrar el Window Painter un mensaje le solicita archivar los cambios que haya hecho a la ventana w_proto.

Creación de la ventana principal

Ahora que ha creado la ventana prototipo para esta aplicación, puede utilizarla para crear la ventana principal de la aplicación. Arranque de nuevo el window Painter. Aparecerá la Select Window. Esta vez haga clic en el botón inherit. **(Consultar el tema de herencia en el capítulo 1)**

Seleccione la ventana w_proto haciendo clic en su nombre. Aparece el Window Painter mostrando la nueva ventana sin título. Esta ventana ha heredado los atributos y controles de la ventana w_proto. Comiense archivando esta ventana con el nombre w_principal. Puede hacerlo con la opción, ésta no se ejecutará correctamente. Debe volver a la Application Painter y editar el procedimiento para el evento open de la aplicación.

La última línea del procedimiento es un comentario. La última sentencia abre la ventana principal. Sin esta sentencia, la aplicación se ejecuta, a continuación se para, y parece como si nada hubiera pasado. Borre las dos barras para hacer que sea una sentencia en vez de un comentario. Compile el procedimiento (CTRL+L).

Ahora puede ejecutar la aplicación sea escrito. Seleccione Run del menú File, use el application.ico o teclee Ctrl-r. Habrá un pequeño intervalo de espera hasta que arranque la aplicación.

La aplicación se ejecuta y se muestra la ventana principal.
Haga clic en el botón Exit para detener la ejecución de la aplicación.

Debe darse cuenta de que ha tenido que hacer un poco de desarrollo para llegar hasta aquí. Observe qué poco código de procedimientos fue necesario para conectar su aplicación a la base de datos y para añadir funcionalidad al PictureButton para salir. La construcción de la aplicación de la ventana y el añadir controles fue facil comparado a cualquier otro sistema o lenguaje que pudiera haber actualizado.

NOTA: Puede haberse dado cuenta de que el abrir la ventana de su aplicación, ésta no está centrada en la pantalla. Si vuelve al Window Painter y hace clic con el botón derecho de la ventana, obtiene una lista de opciones sobre estilos. Seleccione Position de la lista de estilos. Márquelas cajas de position vertically y position horizontally. La próxima vez que se abra, la ventana aparecerá centrada en la pantalla.

Estilo de los Controles

Los objetos Powerbuilder tienen asociado un estilo determinado por sus atributos. El estilo determina la apariencia y comportamiento del objeto. A continuación se presenta los atributos de un PictureButton:

Opción	Tipo de datos Estado inicial o valor
Attribute (atributo)	Boolean
BringTo Top (traer a primer plano)	Boolean
Cancel (cancelar)	Boolean
Default (por defecto)	String
DisabledName	Boolean
DragAuto (arrastrar automáticamente)	String
DragIcon (icono de arrastrar)	Boolean
Enabled (habilitado)	Boolean
FaceName	String
FontCharSet (conjunto de fuentes de carácter)	FontChartSet (enumerado)
FontFamily (familia de fuentes)	FontFamily (enumerado)
FontPitch (tamaño de fuentes)	FontPitch (enumerado)
Height (altura)	Integer
HTextAlign (alineación horizontal del texto)	HtextAlign (enumerado)
Italic (cursiva)	Boolean
OriginalSize (tamaño original)	Boolean
PictureNeme (nombre de la figura)	String
Pounter (puntero)	String
TabOrder (orden de tabulación)	Integer
Tag (etiqueta)	String
Text (texto)	String
TextSize (tamaño del texto)	Integer
Underline (subrayado)	Boolean
Visible	Boolean
VTextAlign (alineación vertical de texto)	VTextAling (enumerado)
Weight (peso)	Integer
Wigth (ancho)	Integer
X	Integer
Y	Integer

Para acceder a una lista completa de los eventos y atributos de los controles, vea los manuales de Powerbuilder sobre objetos y controles. También puede encontrar esta información en la ayuda en línea.

Foco

El Foco de una ventana determina dónde sucederá la siguiente acción.

Para situar el foco, mueva el cursor a un control y haga clic. Para cambiar el foco, presione la tecla Tab (de tabulación) o la combinación de teclas Shift-Tab. Cuando tiene el foco un CommandButton, puede mostrar un rectángulo de foco. Si tiene un foco en SigleLineEdit, puede presentar un cursor en forma de I-Beam (símbolo en forma de una pequeña barra vertical).

Un control pierde el foco cuando un usuario cambia el foco a otro control. Los controles tienen un evento GetFocus y otro LoseFocus. Se pueden escribir procedimientos para estos eventos para controlar lo que sucede cuando un control adquiere o pierde el foco.

Orden de Tabulación

La tecla de tabulación cambia el foco en una ventana de un control a otro. El Window Painter puede establecer el orden de selección de los controles. Se pueden introducir nuevos números para cambiar el orden de tabulación. Cambiando a cero el orden de tabulación de un elemento se asegura que el elemento no será seleccionable.
Un valor cero indica que el control no puede adquirir el foco. Es decir, no se puede seleccionar.

No cambie los ordenes de tabulación de los objetos de su ventana w_principal. Para cambiar la orden de tabulación de los controles, ponga el cursor sobre el orden de tabulación y haga clic sobre él. Teclee un orden de tabulación nuevo.
Vaya al menú Design y haga clic de nuevo en la opción Tab Order. Esto archiva un orden de tabulación nuevo, incorporando todos los cambios.

Imprimiendo una ventana

Seleccione print del menú file para imprimir la información sobre la ventana actual. El informe impreso se dirige a la impresora activa seleccionada en Windows. La información impresa depende de las preferencias que se hayan establecido en el Preferencia Painter.

3.4 DataWindows

La DataWindows (ventanas de datos) presentan, manipulan, actualizan e imprimen informes de datos. DataWindow automatiza la interfaz entre la base de datos fuente, como por ejemplo, un sistema de gestación de base de datos, un fichero plano u otra aplicación en ejecución. Una DataWindow también automatiza la interfaz entre una aplicación en ejecución y el usuario.

La Data Windows proporciona muchas facilidades útiles para la presentación de datos de una forma efectiva y agradable. Se pueden presentar datos de una gran variedad de formatos, incluyendo presentaciones tabulares, formato libre, etiquetas y gráficos y "crosstab". Se puede mejorar cada uno de estos formatos de presentación. Aun objeto DataWindows se le puede añadir gráficos como por ejemplo líneas, círculos o cajas. Se puede definir el formato de informes impresos con cabeceras, pies o información acumulada. Se puede reorganizar los elementos de datos o reordenarlos en nuevos órdenes. Cualquier estilo de presentación se puede aplicar en cualquier fuente de datos.

Uso de DataWindows

Una DataWindow es un objeto que gestiona la conexión a una fuente de datos. El DataWindow Painter crea un objeto DataWindow.

Un control DataWindows es un control Powerbuilder que se sitúa en una ventana. Se puede utilizar el Window Painter para crear un control DataWindow.

En el Window Painter se asocia el objeto DataWindow al control DataWindow; por lo tanto, el objeto DataWindow se puede reutilizar al poderse asociar a otros controles de otras ventanas de una aplicación.

El objeto DataWindow gestiona el acceso a la fuente de datos, mientras que el control Data Window gestiona la presentación del objeto Data Window.

Fuente de Datos

Un objeto DataWindow puede presentar datos de diferentes fuentes.
Estas fuentes incluyen:

- √ Una base de datos relacional.
- √ Una entrada de usuario.
- √ Un archivo de texto.
- √ Un manejador de archivos planos, por ejemplo, archivos DBF.
- √ Un enlace DDE a otra aplicación.

Cuando se crea un objeto DataWindow con el DataWindow Painter, se selecciona la de fuente de datos. También se selecciona la parte de los datos seleccionados de la fuente de datos. Por ejemplo, se puede seleccionar información más de una tabla de una base de datos relacional, o se puede seleccionar solamente algunas columnas de esas tablas.

Selección de datos

Hay diversas formas posibles de seleccionar datos de una fuente de datos:

- √ Quick Select (selección rápida).
- √ SQL Select (selección mediante SQL).
- √ Query (consulta).
- √ External (procedimiento externo).
- √ Database stored procedure (procedimiento almacenado de la base de datos).

Se usa la opción Quick Select para recuperar datos de una sola tabla; sólo es necesario elegir las columnas, los criterios de selección y el tipo de ordenamiento de los datos recuperados. No se utiliza la opción de la selección rápida si son necesarias columnas calculadas, operaciones de agrupación u otras operaciones complejas de recuperación. Se usa la opción SQL Select para tener más control sobre la recuperación de datos. Esta opción permite unión de tablas y otras opciones de recuperación más complejas.

Se usa la opción Database Stored Procedure si las opciones de recuperación de datos residen en la base de datos como procedimientos almacenados. **Nota**: No todos los SGBD soportan los procedimientos almacenados.

La opción External se aplica cuando la fuente de datos no es un sistema de gestación de base de datos relacional.

Estilo de Presentación

El estilo de presentación de un objeto DataWindows puede ser uno de los siguientes:

- √ Crosstab.
- √ Freeform (formato libre).
- √ Graph (gráfico).
- √ Grid (malla).
- √ Group (agrupado).
- √ Label (etiqueta).
- √ N-UP (encolumnado).
- √ Tabular (tabulado).

Estilo Tabular

El estilo tabular presenta los datos en columnas a lo ancho de la página. Cada columna tiene como cabecera. Se muestra tantas filas como quepan en la pantalla. Las barras de desplazamiento pueden mover todos los datos del área de presentación.

Estilo libre

El estilo libre (freeform) lista las columnas a lo largo de la página con una etiqueta para cada columna. El estilo libre es útil para ventanas de entrada de datos. Las herramientas proporcionadas con el DataWindows Painter pueden mover cualquier objeto mostrado en la ventana de datos de formato libre. También pueden cambiar los atributos de texto o la forma de los objetos. Otras herramientas facilitan el realiniamiento de los objetos o el añadir varios objetos gráficos, incluyendo líneas, círculos o figuras. Esto permite la presentación de los datos de un formato más agradable.

Estilo Malla

El estilo malla (grid) muestra los datos en una presentación fila-y-columna. Las líneas de la malla separan cada una de las filas y las columnas. Cada elemento de datos debe caber dentro de uno de los elementos de la malla.

El DataWindow Painter no puede mover columnas y las filas como hace la presentación en formato libre. La selección de los datos de la fuente de datos fija esta presentación.

El usuario puede reordenar los elementos de la pantalla en tiempo de ejecución, al contrario que con otros estilos de presentación. El usuario puede redimensionar y reordenar las columnas con el ratón mientras la aplicación se está ejecutando.

Estilo de Etiqueta

El estilo de etiqueta (label), como las etiquetas del correo. Soportan varios formatos comunes de etiquetas.

Estilo Columnado

El estilo columnado (N-UP) se parece el estilo de etiqueta. Presenta dos o más filas de datos, una al lado de la otra. Esto permite presentar a lo ancho de la página varias filas de información que conforma la base de datos.

Estilo Agrupado

El estilo agrupado (group) proporciona un objeto DataWindow tabular que tiene predefinadas ciertas propiedades del grupo.

Estilo Gráfico y "Crosstab·"

Ambos estilos, gráficos (graph) y crosstab, permite presentación de datos en formato gráfico. Compuesto

El tipo compuesto (composite) es nuevo en la versión 4.0 y permite tener DataWindow dentro de otras DataWindows.

Buffer Primario

Un control DataWindow permite al usuario recuperar, mostrar, modificar y actualizar información. Este control presenta datos del usuario para estas operaciones. Un buffer en memoria mantiene datos para el objeto DataWindows; este es buffer primario. El objeto DataWindow gestiona la conexión a la base de datos.

Control de Edición

Un objeto DataWindow se divide en un conjunto de celdas. Cada celda contiene un solo elemento de datos recuperado de la base datos. El elemento tiene un tipo definido, como por ejemplo, entero, real o cadena de caracteres.

Cada control DataWindow tiene un solo control de edición. El control de edición es un campo que solapa con una celda. El contenido del control de edición es el texto del control de edición.

De la misma forma que el usuario cambia el foco de una celda o otra para introducir datos, el control de edición se mueve al elemento con el foco. Los datos introducidos por el usuario permanecen en el control de edición hasta que puedan ser válidos. Después de que los datos están validados, se transfieren al buffer.

Procesamiento de Entradas

Cuando el usuario cambia el dato de una celda, y cambia entonces el foco de esa celda, Powerbuilder ejecuta los siguientes pasos para procesar los datos introducidos:

1. El texto de control de edición se convierte al tipo de datos de la celda en el buffer primario. Por ejemplo, si el usuario introduce el texto 123 en el control de edición, y la celda es de tipo entero, el texto 123 se convierte al valor entero ciento veintitres.
2. Si la converción del texto al tipo de datos subyacente tiene éxito, Powerbuilder intentará validar el dato. Las validaciones pueden realizarse en la base de datos o incluirse directamente en una DataWindow.
3. Una validación correcta provoca el evento ItemChanged del control DataWindow.
4. Después del evento ItemChanged, se provoca un evento ItemFocusChanged.

Si hay un error por ejemplo, si los datos no son válidos o la conversión del texto no tiene éxito, se provoca un evento ItemError. Si es necesario un procediento adicional después de la validación del texto introducido, se deberá introducir un procedimiento para el evento ItemChanged.

Validación de datos

La validación de datos empieza cuando el usuario ha modificado los contenidos del control de edición y sucede cualquiera de los siguientes eventos:

se pulsa la tecla INTRO
se cambia el foco a una celda diferente del control DataWindow
un procedimiento ejecuta la función AcceptText()

Cuando el usuario ha introducido algo en el control de edición, los contenidos de control de edición pueden ser diferentes de los contenidos de la celda en el buffer. El dato introducido por el usuario se mantiene en un control de edición hasta que es validado. El dato de la celda del buffer no se cambia hasta que el dato en el control de edición recibe la validación. Una vez validado, el dato del control de edición se copia en el buffer primario. Entonces, los contenidos del control de edición y los de la celda del buffer primario son los mismos.

Secuencia de Validación

Cuatro pruebas validan el dato mantenido en la máscara de edición. Estas pruebas se llevan acabo en un orden. Cuando los contenidos del control edición han pasado todas las pruebas, cambia la celda en el buffer de la DataWindow.

1. ¿Cambió algo?
2. ¿Coincide el tipo de datos del objeto introducido con el tipo de datos en la celda?
3. ¿Los datos encuentran las reglas de validación en el objeto DataWindow o éstas se mantienen en la base de datos?
4. ¿Se ha producido el evento ItemChanged?

Accediendo al texto de un elemento

Hay disponibles variables funcionales para acceder al texto actual en el control de edición o a un dato especifico del buffer primario.

Gettext()-devuelve el texto del control de edición
SetText() -sitúa texto en el control de edición
GetItemDate(), GetItemNumber(), GetItemDateTime(),
GetItemDecimal(), GetItemString(), GetItemTime()
-cada una devuelve el dato de una celda del buffer primario
SetItem () -sitúa un elemento en una celda del buffer primario.

Otros Buffers

Un objeto DataWindow tiene tres buffers de datos. El primer buffer, como se vio anteriormente, es el buffer de datos primario. Éste contiene información recuperada en la base de datos, o información que una vez obtenida de la base de datos ha sido modificada por el usuario.

Existen otros dos buffers. Es posible filtrar datos de los seleccionados de la base de datos. Sólo se muestran los datos que pasan por el filtro. Es decir, solamente los datos que pasan por el filtro se copian al buffer primario. Los datos que no pasan el filtro permanecen en el buffer de filtro. Los datos borrados del buffer primario permanecen en el buffer de borrado. Por jemplo, si el usuario borra una fila de la DataWindow, esa fila se copia al buffer de borrado.

Actualización de la Base de Datos

Es necesario descartar que los cambios realizados en el buffer primario de la DataWindow no se proponga automáticamente a la base de datos. Para actualizar los cambios del buffer y hacer los cambios en la base de datos, se le debe llamar a la función Update() desde un PowerScript.

Otras operaciones como borrar, añadir u ordenar las filas de buffer no cambian tampoco la base de datos. Así borrar las filas de buffer primario sólo las transfiere al buffer borrado. Para que los resultados de cualquiera de estas operaciones se reflejen en la base de datos, se debe llamar a la función Update().

Manipulación de los Contenidos del Control de Edición

El control de edición del buffer primario es parecido al control MultiLineEdit de las ventanas. Para cambiar el contenido del control de edición hay varias funciones disponibles.

CanUndo()
Copy()
LineCount()
Position()
Scroll()
SelectedLine()
SelectedText()
TextLine()
Clear()
Cut()
Paste()
ReplaceText()
SelectedLength()
SelectedStart()
SelectText()
Undo()

Funciones

Con Powerbuilder se proporcionan varias funciones muy útiles para manipular controles DataWindow. Algunas de las funciones muy útiles para manipular controles DataWindow:

Función	Propósito
AcceptText()	arranca el procesado del texto del control de edición
dwShareData()	permite que dos DataWindow compartan los mismos datos
GetRow()	devuelve el número de la fila actual albuffer primario
DeleteRow()	mueve la fila actual al buffer de borrado
Filter()	restringe la presentación de filas que aquéllas que pasan la especificación del filtro
InsertRow()	inserta una nueva fila en el buffer primario
Reset()	limpia el buffer primario
Retrive()	recupera las filas especificadas por la selección de la base de datos
ScrollToRow()	desplaza los datos hasta mostrar la fila especializada
SelectRow()	destaca la fila seleccionada
Update()	Actualiza cualquier cambio en la base de datos

3.5 Construcción de una DataWindow

Añade una ventana de datos a su aplicación, es un proceso de tres pasos. Primero se construye un objeto DataWindow con el DataWindow Painter. A continuación se añade un control DataWindow a una ventana de la aplicación. Por último, se conecta el objeto DataWindow al control DataWindow. Este tema describe como construir un objeto DataWindow.

Los pasos para la creación de un objeto DataWindow son:

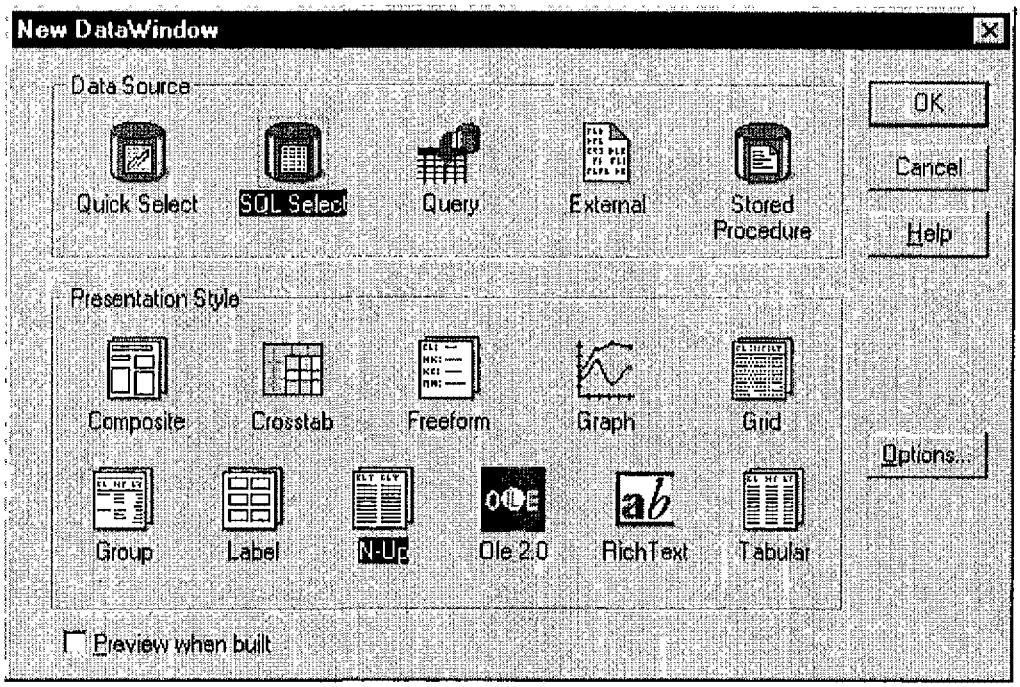
- √ Elegir una fuente de datos
- √ Elegir un estilo de presentación
- √ Elegir las opciones de diseño
- √ Seleccionar tablas
- √ Seleccionar las columnas de las tablas
- √ Seleccionar los argumentos de recuperación (opcional)
- √ Especificar las opciones de actualización
- √ Archivar el objeto de DataWindow

Desde el PowerBar o el PowerPanel, arranque el DataWindow Painter.
Habrá pequeño intervalo de espera mientras que Powerbuilder conecta a la base de datos. Durante esta pausa debería ver el mensaje Connecting to database (conectando a la base de datos) en la esquina inferior izquierda de la pantalla.

Cuando el DataWindow Painter se conecta con la base de datos, aparece la ventana Select DataWindow:

Podría editar un objeto DataWindow haciendo clic sobre su nombre para seleccionarlo y haciendo clic a continuación sobre el botón OK. Como en otras ventanas de selección, puede moverse por los directorios buscando objetos con el árbol de archivos mostrado en la caja directories.

Haga clic en el botón New, y aparece la ventana New Data Window:



Con esta ventana seleccione en la fuente de datos, el estilo de presentación y los atributos del objeto DataWindow.
Al seleccionar la caja Preview When Built hace que aparezca inmediatamente una presentación preliminar del DataWindow Painter. Posteriormente, se puede usar el icono de presentación preliminar del DataWindow Painter para tener una presentación preliminar del objeto DataWindow.

La fuente de datos y el estilo de la DataWindow

La fuente de datos y el estilo de objeto DataWindow seleccionados aparecen en gris en la ventana de selección. Si no están seleccionados, haga clic SQL Select en la sección superior y en Tabular en la sección inferior.

Botón de Opciones

Para cambiar los valores por defecto utilizados por algunos de los atributos de un objeto DataWindow, haga clic en el botón Options. Se puede modificar cualquiera de las opciones listadas para el objeto DataWindow y hacer clic en el botón OK. Para cerrar la ventana sin cambiar ninguna de las opciones, haga clic en botón Cancel. Help arranca la ayuda en línea y proporciona una descripción de cada uno de los elementos listados en la ventana.

Seleccionar tablas y columnas

Si la lista de tablas que aparecen es diferente de la lista de tablas de su base de datos, es que se ha conectado a una base de datos equivocada. Si aparece un mensaje de error en vez de la ventana Select Tables, entonces las preferencias del sistema no se han establecido de forma correcta, o la base de datos de no estaba instalada adecudamente.

Se puede cambiar el orden en el que aparecen las columnas en el DataWindow arrastrando un nombre en la barra superior. Para cambiar la posición de una columna, haga clic, y mantenga pulsado el botón del ratón, en un nombre de la barra superior. Arrastre nombre a la nueva posición deseada. El nombre permanecera en la nueva posición.

Selección, Ordenación, Agrupación

La sentencia de selección asociada con el objeto DataWindow puede restringir o agrupar los datos devueltos por la sentencia selección. Se puede también restringir o agrupar datos con la aplicación después de que los devuelvan la sentencia de selección que restringe, agrupa u ordena datos en el servidor de base de datos.

La sentencia de selección asociada con un objeto DataWindow puede tener lo siguiente:

- √ una cláusula where para restringir los datos seleccionados
- √ una cláusula order by para ordenar los datos devueltos por una selección
- √ una cláusula group by para agrupar los datos seleccionados
- √ una cláusula having para restringir la cláusula "group by"

Se puede utilizar este conjunto de herramientas de SQL para construir cualquier cláusula where en la selección de un objeto DataWindow.

Cambio de pantalla

La primera sección de la pantalla de la DataWindow es la barra de cabecera. En ésta muestra la información que aparece en la parte superior de cada pantalla o página impresa. En esta información se pueden incluir cabeceras u otra información, como por ejemplo, una fecha.

La segunda área es la barra de detalle. Ésta muestra los datos recuperados en la fuente de datos.

La tercera área es una barra de resumen. Esta barra muestra información de resumen que aparece después de todos los datos o al final de una impresión. Esta área puede añadir información como un total contador al objeto DataWindow.

La última área footer (pie) añade la información en la parte inferior de cada pantalla o página. Esta área normalmente añade un número de página o contador de página a la pantalla.

Se puede cambiar la posición de cualquier columna o cabecera de una columna haciendo clic, manteniendo pulsado en el botón del ratón y arrastrando el elemento.

Si se puede cambiar el tamaño de cualquier campo haciendo clic en el campo para seleccionarlo. A continuación se usa el cursor para redimensionar el campo.

Presentación Preliminar de la DataWindow

Vea la aparencia preliminar del objeto DataWindow tecleando Ctrl-W o seleccionando Preview del menú Design. Después de un corto período de espera el DataWindow Painter conecta con la base de datos y aparece la apariencia preliminar de la ventana de datos.

Los datos mostrados en la presentación preliminar provienen de la base de datos de demostración. El objeto DataWindow conecta a la base de datos, como ocurrirá posteriormente en la aplicación en ejecución. Observe que el número total de las filas recuperadas aparece en la parte inferior de la ventana.

De izquierda a derecha, el icono de la DataWindow cierra la presentación preliminar y le devuelve al DataWindow Painter. El segundo icono reselecciona las filas de la base de datos. Cada vez que se hace un clic sobre este icono, se ejecuta la sentencia de selección asociada al objeto DataWindow. El diseño del objeto DataWindow determina la selección.

El tercer icono actualiza la base de datos con los cambios realizados en las filas seleccionadas. Ahora no puede cambiar ninguna fila pero pronto podrá.

El cuarto icono inserta una fila en la pantalla. Si hace clic en este icono, suena un pitido pero no se inserta ninguna fila. La razón es que la DataWindow no está todavía configurada para su actualización.

El quinto icono borra la fila actual de la DataWindow. Ya que no ha habilitado a la DataWindow para que sea actualizado, no se destacará ninguna fila como fila actual.

El resto de los iconos desplazan en la pantalla de datos dentro de la DataWindow. Esto iconos mueven respectivamente la pantalla a la primera fila de datos a la ventana previa de datos, a la siguiente ventana y a la última ventana.

Características de Actualización

Las características de actualización de un objeto DataWindow están determinadas inicialmente cuando se crea el objeto DataWindow. Esta sección muestra cómo cambiar las características de actualización para un objeto DataWindow.

Selección de una sola tabla

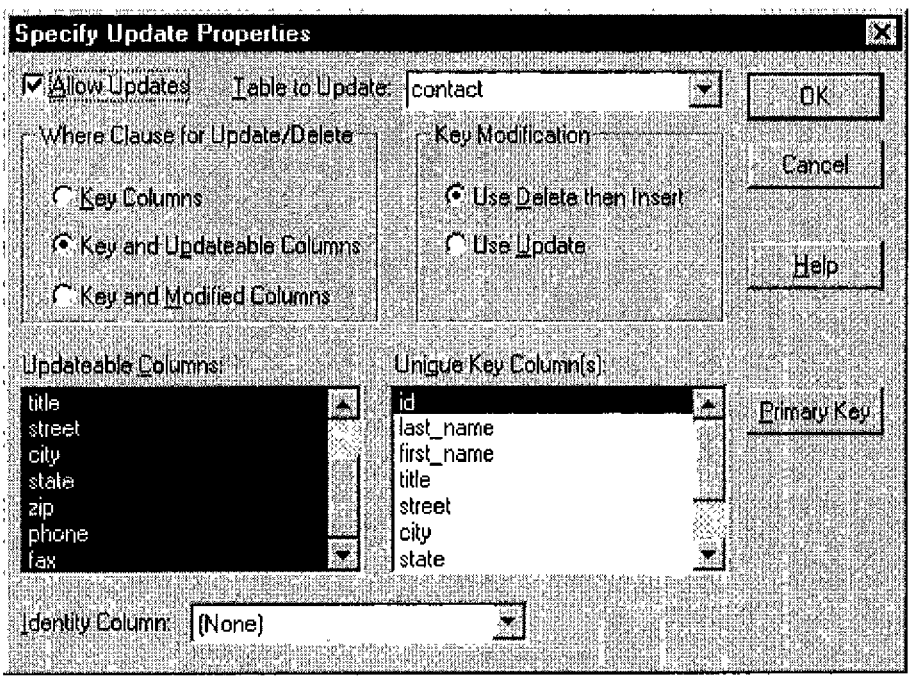
Si todas las columnas seleccionadas provienen de una sola tabla y se selecciona la columna llave de la tabla, todas las columnas seleccionadas son actualizables. El objeto DataWindow asigna un orden de tabulación para cada columna. Esto permite al usuario seleccionar cualquier columna para su actualización. El objeto anterior no proporciona actualización, ya que los datos provienen de varias tablas.

Join

Si la selección de datos de una DataWindow incluye datos de más de una tablas (es decir, se usa un join para recuperar los datos), características de actualización no permiten la actualización. La DataWindow que ha construido no permite todavía la actualización. La razón es que la sentencia de selección del objeto DataWindow incluye un join.

Para gestionar la actualización de varias tablas con un objeto DataWindow, se deben escribir procedimientos para el control de DataWindow que seleccione una tabla cada vez y actualice cada tabla individualmente. El curso de DataWindows Avanzadas enseña las técnicas para hacer esta actualización. El ámbito de este texto introductorio no incluye estas técnicas.

En el DataWindow Painter, seleccione Update del menú Rows.



La selección Allow Updates en la esquina superior izquierda se muestra como no seleccionada, ya que la selección del objeto DataWindow une varias tablas. Haga clic en caja de verificación Allow Updates. Esto permite que el objeto DataWindow pueda actualizar. Los campos que parecerían en gris aparecerán ahora todo color.

Seleccione la tabla a actualizar. A continuación, seleccione las columnas de la lista Update Columns:
Después, haga clic en el botón Primary Key. Observe que después de un corto período de espera esto habilitará el botón OK y este no volverá aparecer en gris. Haciendo clic en el botón Primary Key se cancela cualquier cambio que de haya hecho en la caja Unique Key Columns. Esto también destaca y selecciona las llaves primarias por defecto. Ahora se puede seleccionar el botón OK.

Esto ha habilitado la sentencia de selección del objeto DataWindow para actualizar la tabla. Una vez cambiado el orden de tabulación, se puede usar la presentación preliminar de la DataWindow para actualizar la base de datos.

¿Insertar o actualizar?

El objeto DataWindow crea una sentencia SQL sobre modificación de datos para un objeto DataWindow y ante una petición para confirmar las modificaciones en la base de datos. La selección de Key Modification mostrada en la ventana de actualización determinada en que tipo de sentencia SQL se genera cuando cambia una columna llave. Una columna llave especifica en la caja Key Columns. Estas opciones son:

INSERT y DELETE
UPDATE

La selección insert and delete (inserta y borra) borra las filas originales de la base de datos y a continuación inserta nuevas filas en la base de datos. La segunda opción actualiza las filas existentes. Utilece la opción de actualización para modificar una sola fila; es más rápido.

Si la actualización cambia varias filas, un borrado e inserción siempre funciona, mientras que una serie de actualizaciones puede que no. Imagine que una actualización cambie el valor de una llave en la primera de dos filas. Entonces la actualización intenta cambiar el valor de la llave en la segunda fila al valor original de la llave primaria fila. En algunos sistemas de gestión de base de datos, una secuencia de actualización como ésta falla.

Cláusula where para actualizaciones y borrado

Cuando los datos de un objeto DataWindow han cambiado a una presentación confirma las modificaciones en la base de datos, el objeto de DataWindow crea una sentencia SQL. La cláusula where generada por el objeto DataWindow determina las actualizaciones o borrados. La cláusula where seleccionada incluye una opción de las siguientes:

Sólo columnas llave
Todas las columnas llave y todas las actualizables
Todas las columnas llave y todas las modificables

Cuando varios usuarios pueden acceder simultáneamente a una misma tabla, se debe seleccionar una estrategia de actualización. Si se permite que la aplicación actualice la base de datos bajo cualquier circunstancia, los usuarios pueden entrar en conflicto unos con otros, resultando un mal control de concurrencia.
Se puede controlar la estrategia de actualizacións se especifica que columnas se incluyen en la cláusula where en la sentencia de actualización o borrado generada por el objeto DataWindow:

```
Delete from tabla
Where col1 = valor1
  and col2 = valor2
```

Si se selecciona la opción Key Columns, la cláusula where incluye sólo columnas llave. Las columnas llave son las columnas seleccionadas en la caja Key Columns de la ventana Update Characteristics. Esto provoca una comparación de los valores de los elementos llave tal como se

recuperararon originalmente con las columnas llaves de la base de datos. Si corresponden, la actulización tiene éxito.

Si selecciona Key and update Columns, la cláusula where incluye ambas. Se comparan los valores llave originales y las columnas actualizables recuperadas originalmente para las columnas llave y las modificadas con los valores de la base de datos. Para actualización funciona, los valores deben corresponderse con los encontrados en la base de datos. Si algún otro usuario ha cambiado valores en la base de datos, la actualización falla.

Un objeto DataWindow actualiza una tabla de empleados. Hay tres columnas llave y las modificadas. Se compran los valores recuperados originalmente para las columnas llave y las modificadas con los valores de la base de datos. Si algún valor ha cambiado desde la recuperación de las filas, la actualización falla.

Un objeto DataWindow actualiza una tabla de empleados. Hay tres columnas en la tabla -EmpId. Todas las columnas de la tabla son actualizables. El usuario recupera la fila del empleado número 1001. El usuario cambia el salario de este empleado de 5,000,000 pesos a 5,500,00 pesos. He aquí el resultado para cada una de las tres opciones de actualización.

(1) Con la opción Key Columns:

```
update Empleado
set    Salario = 5500000
where EmpId = 1001
```

En este ejemplo, la actualización funciona incluso en el caso de que otro usuario haya combinado la fila de la base de datos. Cualquier actualización del salario hacer perder el cambio
(2) Con la opción Key and Modifies columns:

```
update Empleado
set    Salario = 5500000
where EmpId = 1001 and Salario = 5000000
```

Aquí la actulización falla si otro usuario ha cambiado el salario del empleado.
(3) Con la opción Key and Update columns:

```
update Empleado
set   Salario    = 5500000
where EmpId  = 1001 and Salario = 5000000
  and Nombre = valor_original
```

En este ejemplo, la sentencia falla si otro usuario ha cambiado cualquiera de las columnas actualizables de la tabla, ya que el registro viene de la base de datos.

Establece el orden de tabulación

Aunque las características de la actualización de la tabla han cambiado para permitir la actualización de la tabla, el objeto DataWindow todavía no puede actualizar. La razón es que el orden de tabulación de cada uno de los objetos mostrados en la banda de detalle es cero. El orden de tabulación de cada objeto es cero porque el objeto DataWindow se origina a partir de un join, y un join por defecto no permite actulizaciones. Seleccione Tab Order en el menú Design.
Haga clic en el elemento del menú Tab Order del Design. Desaparece el orden de tabulación y la pantalla vuelve al DataWindow painter.

Teclee Crtl+w o seleccione Preview de menú Design para tener una presentación preliminar del objeto DataWindow:

Observe que se puede cambiar la información de las órdenes de venta. La tecla Tab mueve el control de edición de un campo a otro.

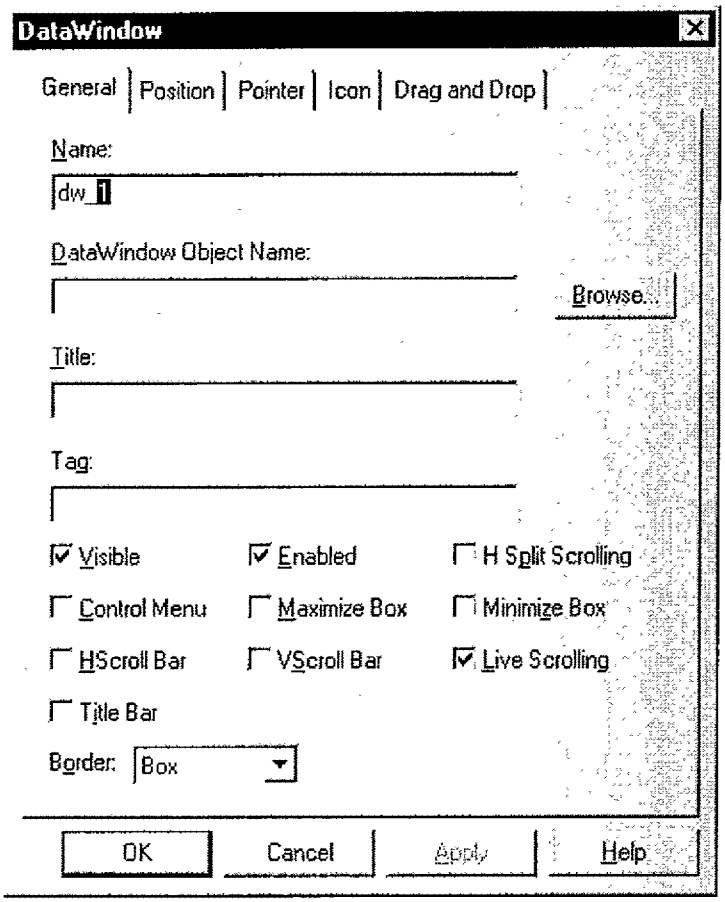
Todos los iconos están ahora habilitados. Se pueden introducir nuevos datos. Se pueden actualizar la base de datos para reflejar los cambios realizados. Se pueden añadir nuevas filas o borrar filas existentes.

3.6 Control de Data Window

El procedimiento asociado al evento open de la aplicación establece una conexión entre las tablas y columnas seleccionadas en el objeto DataWindow y la base de datos. Este procedimiento abre la base de datos, al tiempo que el objeto DataWindow se conecta a las tablas y columnas de las que es preciso recuperar información.

El Control Data Window

Seleccione la opción DataWindow del menú Controls, o haga clic sobre el icono DataWindow de la barra de iconos del Window Painter:



Entonces cambiará el aspecto del cursor al de una cruz cuadrada. Situando el cursor en la esquina superior izquierda de la posición en donde se desea situar en el control DataWindow y haciendo clic, aparece el correspondiente control DataWindow.

Para agrandar el control, se mueve el cursor hasta la esquina inferior derecha del control. El aspecto del cursor cambia a una doble fecha. Haga clic manteniendo pulsado el botón del ratón y desplace el cursor hacia abajo a y la derecha.

Para asociar un objeto DataWindow, se da doble clic dentro del área que ocupa el control, apareciendo entonces el cuadro de diálogo Select DataWindow (Selección del DataWindow); en él selecciona el DataWindow.

Haga clic en el campo Name (Nombre) para cambiar el nombre del control a dw_nombre. Haga clic en la casilla de verificación Title Bar y teclee algún título en el campo Title.

Haciendo clic con el botón derecho sobre la opción Style, aparece el submenú asociado. Seleccione la opción Hscroll Bar para añadir una barra de desplazamiento horizontal, y luego repita la operación para añadir la barra de desplazamiento vertical (opción Vscroll Bar).

A continuación se muestra el aspecto del control de DataWindow con los atributos seleccionados. La barra de desplazamiento vertical no aparece hasta que se ejecute la aplicación y sea necesaria para mostrar la información adicional, que por las dimensiones físicas del control, no puede presentarse.

Cuando las dimensiones del control no permiten presentar la totalidad de las filas y columnas recuperadas por el objeto DataWindow, las barras de desplazamiento permiten al usuario navegar por ellas.
No existen diferencias por seleccionar las barras de desplazamiento siguiendo el procedimiento descrito, hacerlo desde la ventana de estilo, descrita con anterioridad.

La tecla rápida Crtl-may-w ejecuta la ventana de edición. Si no se han salvado las últimas modificaciones realizadas, aparece la ventana solicitando confirmación para su archivo.

Entonces aparece el procedimiento asociado al evento clicked. El nombre del evento cuyo procedimiento se encuentra editando aparece en el título de la ventana del Script Painter.
Desplegando la lista nombrada como Select Event, se observa que efectivamente es asociado un procedimiento al evento clicked, pero sin embargo dicho procedimiento no aparece en la ventana del Script Painter. Ello es debido a que el procedimiento se definió para el botón del objeto ascendente. Despliegue el menú asociado al elemento Compile.

Con Extend Ancestor Script (Añadir al procedimiento del ascendente), se añaden las sentencias introducidas en el Script Painter a las que fueron escritas en el objeto ascendente. Con Override Ancestor Script (Sobreescribir el procedimiento ascendente), se ignora cualquier procedimiento asociado en el objeto ascendente, por lo que sólo se ejecutará el procedimiento asociado en el control heredero. La última posibilidad, Display Ancestor Script (Mostrar el procedimiento ascendente), permite visulizar el procedimiento asociado al control ascendente.

Más Controles

Tanto en la presentación preliminar de la ventana como al ejecutar la aplicación, no aparecerían datos en el control DataWindow, ya que no se había indicado en ningún momento el procedimiento para ejecutar tal información desde de la base de datos y presentarla en el correspondiente control.

Las sentencias siguientes permiten indicar ¿qué? objeto de transacción utilizará, además de traer los registros de la base de datos:

```
dw_pedidos.SetTransObject (SQLCA)
dw_pedidos.Retrive()
```

En el Script Painter se dispone de la tecla rápida Ctrl-L, o bien de la opción Script del menú Compile, para compilar el procedimiento en edición.

La primera sentencia asocia el objeto transacción de información al objeto DataWindow por medio de la función Retrive(). El objeto recupera los datos solicitados de la base de datos en sus buffers internos.

Al volver al Aplication painter, inicie el Script Painter para editar el procedimiento asociado al evento Open del aplication. Asegúrese de que la última línea, la contiene la sentencia Open(w_principal) no se encuentra comentada (comienza con dos barras incluidas); si así fuese, habrá que suprimir las barras incluidas.

Para ejecutar la aplicación puede hacer clic en el correspondiente icono de la PowerBar o del PowerPanel, o seleccionar la opción Run del menú File, o teclear ctrl-R.

Tras una breve pausa, comienza la ejecución de la aplicación. Haciendo clic en botón Recuperar, y tras unos instantes, aparecen datos en el control DataWindow y también en barra de desplazamiento vertical. Con ayuda de la barra de desplazamiento horizontal.

Este es el procedimiento con el cual se podrá actualizar una DataWindow.

```
dw_pedidos.Update ()
```

Tratamiento de Errores

Los procedimientos asociados a los botones Recuperar y Actualizar no procesan posibles errores. ¿Qué sucedería si se produce un error?

La función Retrieve devuelve el número de registros que ha recuperado la base de datos, o -1 en caso de error. Devuelve cero cuando no ha conseguido recuperar registros en la base de datos, pero esto no es en sí una condición de error.
Utilizando este valor devuelto puede mejorarse el procedimiento asociado al botón de Actualizar:

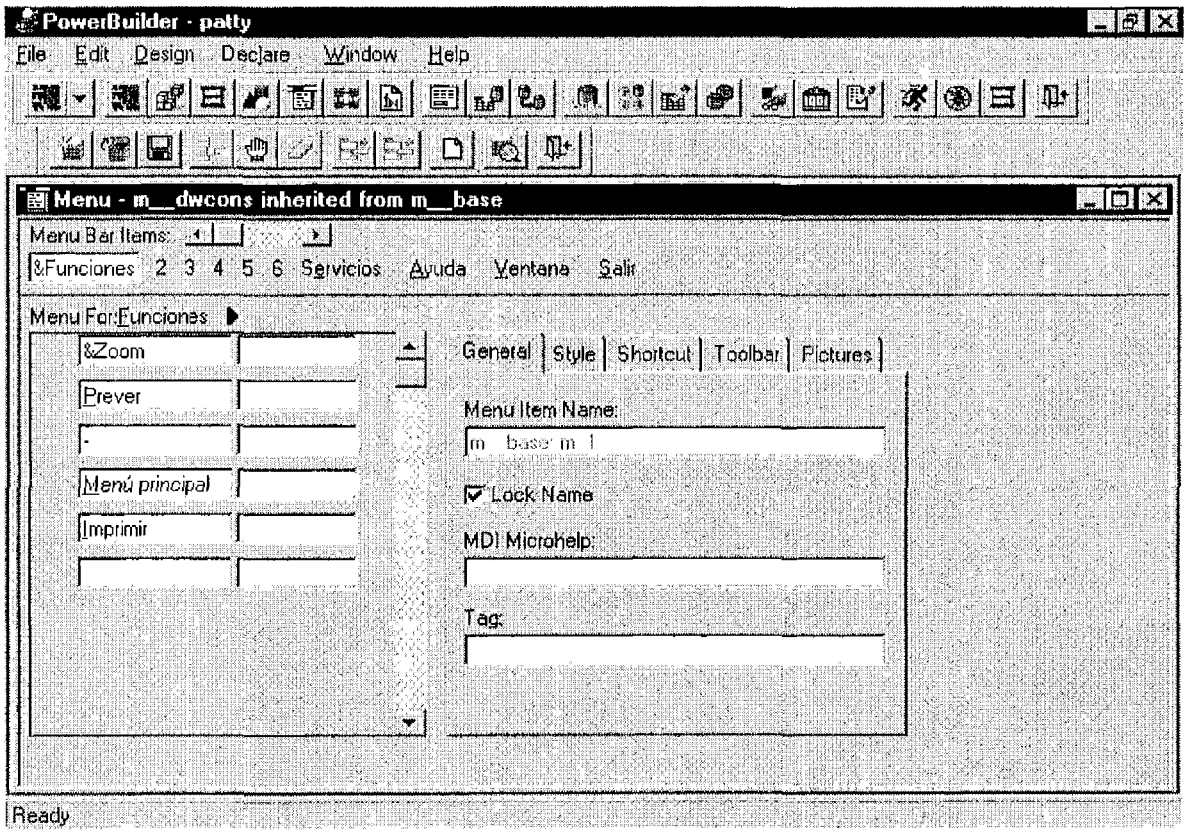
```
If dw_pedidos. Update () > 0 Then
  COMMIT;
Else
  ROLLBACK;
End if
```

Este procedimiento finaliza la transacción asociada a una actualización sin errores (Commit). De producirse algún error durante la actualización, deshace las operaciones realizadas hasta el error (Rollback).

Impresión de Resultados

Añada un nuevo botón, cb_imprimir, con el texto imprimir. Asocie el siguiente procedimiento al evento clicked:

```
int IdTrabajo
IdTrabajo=PrintOpen ("Informe de Pedidos")
PrintDataWindow(IdTrabajo, dw_pedidos)
PrintClose (IdTrabajo)
```



3.7 Menu Painters

Añadirá un menú a la aplicación. La operatoria de menú es bastante familiar; cualquier aplicación Window, como por ejemplo Powerbuilder, dispone alguno.

Para *añadir un menú, primero es preciso crearlo y después asociarlo a la ventana en que va a* aparecer. Esta asociación se realiza desde Window Painter.

Partes del Menú

Un menú es un objeto visual de Powerbuilder. Un menú permite al usuario seleccionar opciones o acciones de una lista. Una barra de menú es una alternativa para usar estas opciones como si fuesen botones.

Al finalizar el trabajo con el Menu Painter, el menú se guarda en una librería de la aplicación Powerbuilder. Un menú también tiene atributos, los cuales rigen una presentación o permiten el uso de teclas de acceso rápido o acelerados.

Se conoce como barra de menú la parte de la ventana en la que se muestra el menú; la barra del menú contiene una lista de órdenes o acciones, a las que se le llama también elementos del menú. Todo elemento de un menú realiza alguna acción; una acción puede ser mostrar una nueva lista de opciones o elementos, es decir, un submenú.

Por ejemplo, en el Menu Painter, cada uno de los elementos que aparecen en la barra de menú son opciones, es decir, al hacer clic sobre cualquiera de ellos, aparece una lista con nuevas opciones u órdenes, como sucede por ejemplo para la opción Design. Los elementos de una barra de menú pueden ser indistintamente opciones u órdenes.

Los elementos del menú del Menu Painter son File, Edit, Design, Declare, Window y Help. Al hacer clic en cualquiera de ellos, se despliega una lista con nuevos comandos u opciones.

En la opción File es posible escoger entre New, Open, Inherit, Close, Save, Save As, Run, Debug, PowerPanel, Print, Printer Setup y Exit. Cada uno de estos elementos es una orden. Al hacer clic en uno cualquiera de estos elementos se ejecuta una acción, en nuestro caso especifica al diseñar el Menu Painter. Los elementos de este nivel también pueden presentarse otra lista de opciones o ejecutar una orden.

Todos los elementos de una opción deberán tener alguna opción. Por ejemplo, todos los elementos de la opción File son operaciones sobre archivos aplicables a un objeto Menú.

Presentación de un Menú

Existen varias formas de presentar un menú. Los menús desplegables (dropdown) son los que presentan las opciones de la barra de un menú.

Al seleccionar uno de los elementos de un menú desplegable puede aparecer un menú en cascada. Los menús en cascadas muestran posibles selecciones posteriores, todas ellas relacionadas con el elemento a partir del cual se mostraron.

Un menú emergente (popup) aparece bajo petición del usuario, normalmente haciendo clic, cerca de la posición actual del cursor. Ejemplos son los menús emergentes de DataWindow Painter.

Opciones de un Menú

Existen varias opciones que ayudan en la compresión y uso de un menú. Por jemplo, en el Menu Painter:

Las líneas de separación agrupan aquellas órdenes del menú lógicamente relacionadas.
Una marca de verificación en la parta izquierda del elemento indica la activación de ese elemento.
Las teclas rápidas y aceleradas permiten un fácil acceso a órdenes del menú desde el teclado. Una tecla rápida ejecuta un elemento del menú con una sola combinación del teclado. Por ejemplo, Crtl-F4 cierra el Menu Painter.

Algunos elementos tienen una de las letras del texto que los nombra subrayada. Por ejemplo, el elemento File tiene la F subrayada, y el elemento Open la O. Los caracteres subrayados son acelaradores; permiten navegar rápidamente por el menú. Por ejemplo, Crtl-F despliega el menú File; Alt-N selecciona el elemento New.

Cuando el texto de un elemento finaliza con tres puntos suspensivos indica que al seleccionarlo aparecerá un cuadro de diálogo. Por ejemplo, al seleccionar Save As aparece una ventana en la que se solicita un nuevo nombre para un menú.

Los elementos cuyo texto aparecen degradados no se encuentran disponibles; Cuando puedan seleccionarse aparecerán en color normal.

Eventos

Un elemento de menú dispone de dos eventos, selected y clicked.

Al mover el cursor hasta un elemento dado, se selecciona y dispara el evento selected. Alt-F despliega el menú File, y al navegar a lo largo del menú con las teclas del cursor, se va resaltando y volviendo a su aspecto normal cada elemento según vamos avanzando. El usuario puede seleccionar también un elemento, haciendo clic sobre él y manteniendo pulsado el botón izquierdo del ratón.

El evento clicked se dispara al hacer clic sobre el elemento o al introducir la tecla rápida o acelerador para el mismo. Haciendo clic sobre el elemento File, se despliega el menú asociado. Manteniendo pulsado el botón izquierdo y moviendo el cursor, se selecciona el elemento bajo el mismo. Al liberar el botón izquierdo, se completa el clic, activándose el elemento.

Procedimientos

El procedimiento asociado a cada evento de un elemento define su comportamiento. El procedimiento asociado al evento clicked puede emplearse para abrir una ventana o realizar cualquier otra acción.

Relativo a los controles
Una ventana puede contener controles como botones de órdenes o botones con figuras. Un menú se asocia a una ventana. A menudo se necesita referenciar a un control dado, por ejemplo, desde un menú.

Para referirse a un control de la ventana desde un procedimiento, se necesita indicar el nombre completo para el mismo; es decir, el nombre de la ventana seguido del nombre del control. La sintaxis es la siguiente:

ventana.control.atributo = valor

Uno de los atributos de un botón es enabled, el cual habilita o inhabilita el botón. Por ejemplo, para inhabiltar el botón cb_insertar de la ventana w_principal, la sentencia es:

w_principal.cb_insertar.enabled = FALSE

ParentWindow

Un procedimiento asociado a un evento de un elemento de menú puede referirse a la ventana asociada con la palabra reservada ParentWindow. ParentWindow y Parent, dependiendo del contexto, pueden coincidir o denotar distintos objetos.

La siguiente sentencia en un procedimiento asociado a un elemento de un menú, cierra la ventana asociada al mismo:

Close (ParentWindow)

Sólo se puede utilizar ParentWindow para referirse a los atributos de la ventana. No puede emplearse para referenciar a sus controles. Así, por ejemplo, la siguiente sentencia ES INVÁLIDA:

ParentWindow.cb_insertar = FALSE

This

Un elemento de menú puede tener asociado un procedimiento para cada uno de sus eventos posibles. La palabra reservada This indica el elemento al que se asocia el procedimiento.

Uso de Menu Painter

Haciendo clic en el botón New aparece el Menu Painter.

Los menús se encuentran dentro de los objetos Powerbuilder heredables. Se podrían crear nuevos menús heredando de los que ya existen en la aplicación.

De entre todos los de Powerbuilder, el Menu Painter es uno de los painters más sencillos y con menos opciones.

Definición de los elementos del menú

Lo primero será nombrar los elementos de la barra del menú. Posicione el cursor en el primer campo del área nombrada como Menu Bar Items en la ventana del Menu Painter e introduzca el texto Archivo.

El carácter indica que el siguiente carácter, en nuestro caso A, será un acelerador. El acelerador no tiene que ser el primero de los caracteres, sino cualquiera.

Haga clic en el campo Menu Bar Items, justo a la derecha del elemento creado. Aparece entonces en nuevo campo de edición para un segundo elemento del menú.

El nombre de este elemento será Ayuda. Para que el acelerador fuera un carácter y en lugar de la A, el texto debería ser Ayuda. Obsérvese (campo Menu Item Name) que automáticamente se asigna un nombre a cada elemento a partir del texto introducido. Posicionando el cursor en el elemento Archivo y haciendo clic, aparece bajo él una lista nombrada como Menu For Archivo.

Al posicionar el cursor en el primer campo de esta lista aparece una mano. El campo seleccionado es el primero de los que compondrán el menú Archivo. Teclee Recuperar y, despúes, con ayuda del tabulador, continúe introduciendo los siguientes elementos: Guardar e Imprimir. Para el cuarto elemento, introduzca un simple guión (-); así aparecerá una línea de separación en su lugar. El último elemento será Salir.

Asegúrese de que el cursor se encuentra en el último elemento del menú, Salir, y despliegue la lista Shortcut Key (tecla de acceso rápido). Desplazándose hasta la opción F10 y seleccionándola, se asigna esta tecla rápida para finalización de la aplicación.

Se pueden usar también combinaciones de teclas como Alt, Ctrl o Mayúsculas (seleccionando la correspondiente casilla de verificación a la derecha del menú desplegado).

Cada elemento de menú tiene un nombre o identificador asignado automáticamente. Podemos comprobar que al elemento introducido, Salir, se asignó m_salir. Es posible cambiar este nombre por otro más significativo, si así se desea.

Cada elemento tiene también el atributo enabled. Cuando se le asigna el valor FALSE, el elemento se inhabilita, por lo que el usuario no puede ejecutar la orden asociada. Desde una aplicación en ejecución se puede ir habilitando e inhabilitando cualquier elemento. Análogamente, el atributo visible determina si se mostrará el elemento al presentar el menú.

El otro atributo es checked, el cual añade una marca de verificación a la izquierda del elemento, confimando su activación. Todo esto es muy habitual en las aplicaciones Windows.

Finalmente, la visualisación del menú es posible mediante la tecla rápida Crtl-w o con la opción Preview del menú Design (ahora, y al igual que Powerbuilder, ya sabe asignar teclas rápidas a sus aplicaciones).

Al hacer clic en Archivo se despliega el menú. Haciendo clic en Imprimir se selecciona la opción, pero no sucede nada, ya que esto es sólo una vista previa del menú sin ejecutar aplicación.

F10 selecciona el elemento Salir. ESC elimina la selección. Para volver al Menu Painter haga doble clic en el menú del control de la ventana o teclee F4.

Asignación de un menú a una ventana

Minimice el Menu Painter y abra el Window Painter con la ventana w_principal. Seleccionando la opción Window Style (Estilo de la Ventana) del menú Design, o haciendo doble clic en el área de la ventana y fuera de cualquier control.

Al hacer clic en la casilla de verificación Menu, automáticamente aparece m_principal. En este cuadro de diálogo asigna un título a la ventana. Haga click en el botón OK para cerrar el cuadro de diálogo y asociar el menú m_principal a la ventana w_principal.

Asignación de procedimientos

Maximice la ventana del Menu Painter y haga clic en el elemento Recuperar. Haciendo clic en el icono Script, seleccionando la opción Script del menú Edit o usando la tecla rápida Crtl-S aparece el Script Painter para editar el procedimiento asociado al evento clicked del elemento.
El procedimiento asociado al evento clicked del elemento Recuperar es el siguiente:

```
//m_principal.m_recuperar Recupera registros de la base de datos
w_principal.dw_pedidos.SetTransObject (SQLCA)
```

```
w_principal.dw_pedidos.Retrive ()
```

Ahora, al seleccionar la opción recuperar, este procedimiento recuperará registros de la base de datos en el DataWindow.

Análogamente, el procedimiento asociado al evento clicked del elemento Guardar sería:

```
w_principal.dw_pedidos.Update ()
```

Para el elemento Imprimir:

```
int IdTrabajo
IdTrabajo = PrintOpen( "Informe Pedidos" )
PrintDataWindow (IdTrabajo, & w_principal.dw_pedidos)
PrintClose (IdTrabajo)
```

Y para el elemento Salir:

```
Close (ParentWindow)
```

Finalmente se añade un nuevo elemento antes de Imprimir, denominado "Seleccionar Impresora" (use la opción Insert del menú Edit), para acceder al cuadro de diálgo Printer Setup estándar de Windows. El procedimiento asociado es el siguiente:

```
//m_seleccionarimpresora-Evento clicked
If PrintSetUp ( ) = -1 Then
  MessageBox (&
        "Selección de impresora", &
        "No es posible la selección")
End If
```

Funciones Definidas por el Usuario

Una función es un procedimiento que tiene un nombre y que realiza una tarea específica. Cualquier sentencia válida en un procedimiento lo es también dentro de una función. La llamada a la función se realiza en PowerScript con su nombre, ejecutándose entonces las sentencias que contenga. También se pueden pasar parámetros a una función.
Al finalizar la ejecución de las sentencias de la función es posible devolver un valor, frecuentemente un número o cadena de caracteres. Normalmente, una función devuelve el valor 1 si no se produjo ningún error en la ejecución, y -1 en caso contrario. Una subrutina es una función que no devuelve ningún valor.

El ámbito de validez de una función determina su accesibilidad dentro de la validación. El ámbito de validez de una función se determina en base a tres niveles:

√ Público (public).
√ Privado (private).
√ Protegido (protected).

Las funciones con un nivel de acceso público son globales. Pueden llamarse desde cualquier parte de la aplicación. Aquellas funciones de propósito general y con grandes posibilidades de reutilización deben tener un nivel de acceso público.

Ventanas, menús u objetos de usuarios pueden tener asociadas funciones. Estas funciones son al nivel de objeto y tienen un nivel de acceso privado o protegido.

Las funciones de un objeto con nivel de acceso privado solamente pueden llamarse desde cualquiera de los procedimientos del mismo. Las funciones de un objeto con nivel acceso protegido pueden llamarse desde cualquiera de los procedimientos de dicho objeto y todos sus descendientes.

Powerbuilder soporta el polimorfismo, por lo que objetos diferentes pueden tener funciones diferentes con el mismo nombre. No existe ambigüedad, ya que las funciones de un objeto se llaman haciendo referencias al objeto que las contiene.

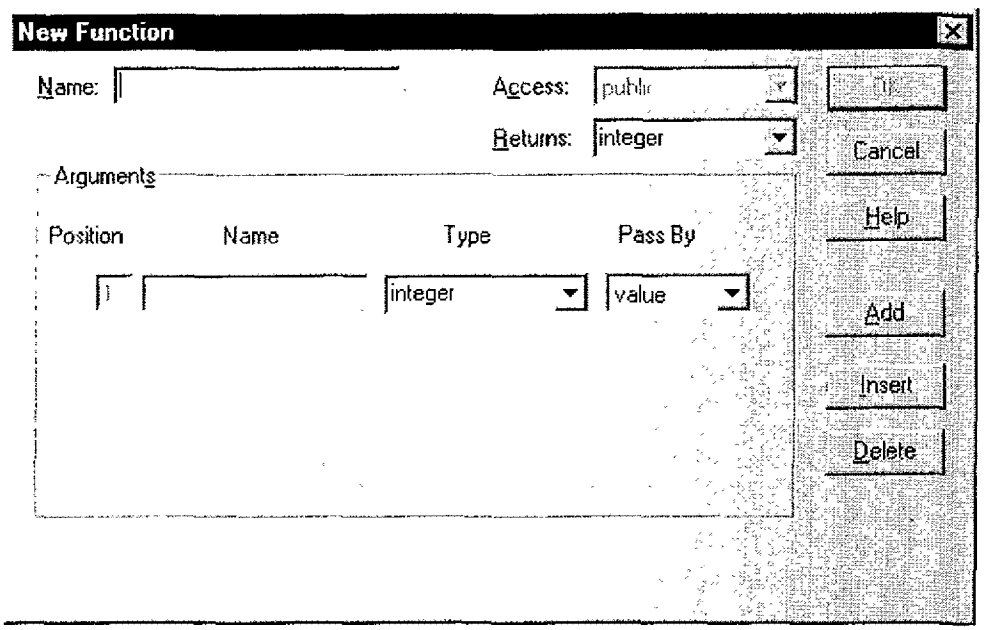
Por ejemplo, supongamos una aplicación con varios objetos entre ellos, ventanas w_entrada_de_datos y w_ventanas. Ambas pueden tener funciones con igual nombre, distintas definiciones y realizando diferentes tareas.

Se pueden llegar al Function Painter desde la PowerBar, el PowerPanel, o desde el menú Declare de cualquiera de los siguientes Painters: Window, Menu on User-Object. También se puede llegar desde el menú Declare del Script Painter, invocado a su vez desde cualquiera de los Painters anteriores.

Si se inicia el Function Painter desde PowerBar, el PowerPanel, sólo se pueden crear funciones globales, es decir, con un nivel de acceso público. Durante la construcción de un procedimiento en el Script Painter, invocado desde cualquiera de los siguientes Painters: Window, Menu O user-Object, se puede especificar el nivel acceso de la función: público privado o protegido. En estos Painters se pueden crear indistintamente funciones globales o al nivel de objeto.

3.8 Function Painter

Una función global tiene un nivel de acceso público. El Function Painter se activa haciendo clic en el icono correspondiente del PowerPanel o la PowerBar.



De existir funciones, aparecen como elementos de la lista de la caja superior. Al hacer clic sobre una de ellas, su nombre se repetiría en la caja de Function; para editarla, bastaría entonces con hacer clic sobre el botón OK. El botón Other permite la búsqueda de funciones en otras librerías.

Al tratarse de una función global, el nivel de acceso queda fijo en público, no siendo posible su modificación. Haga clic en la caja Name y escriba el nombre de la función, por ejemplo, f_tarifa. Sólo se permiten minúsculas.

A continuación se indica el tipo de valor que devuelve la función, como por ejemplo, si es una cadena de caracteres o un número entero. El tipo por defecto es número entero. El menú desplegable Returns contiene todos los tipos de datos que pueden devolver una función, y en especial, (None), para aquellas funciones que no devuelven ningún valor. La función ejemplo devuelve un número entero.

Finalmente, hay que especificar los argumentos. Haciendo clic en la caja Name se puede introducir el nombre del primer argumento, por ejemplo, "peso"; aunque en esta caja se permite teclear caracteres en mayúsculas, se convertirán a minúsculas al guardar la función. El tipo del argumento se escoge en el menú desplegable Type; en el ejemplo, el argumento es de tipo entero.

El argumento especificado, es un parámetro formal, es decir, toma valor durante la ejecución de la aplicación cuando se llama a la función.

Cuando se llama a la función durante la ejecución de un procedimiento, se emplea un parámetro actual. Un ejemplo de llamada a la función podría ser:

```
If f_tarifa (Mi Peso) > 0 Then
  // Tratamiento
Else
  //Tratamiento de error
End if
```

En el ejemplo, se llama a la función con el parámetro actual MiPeso; el valor del parámetro actual se pasa a la función mediante un parámetro formal.
Finalmente hay que editar cómo se pasarán los argumentos a la función. Los argumentos pueden pasarse por refencia o por valor.

Cuando se sepa un argumento por valor, la función recibe una copia de él, por lo que si durante su ejecución se modifica su valor, el valor del parámetro actual no cambia.
Cuando el argumento se pasa por referencia, la función recibe un identificador del mismo de tal forma que las modificaciones del parámetro formal se reflejan de igual manera en el parámetro actual. Al devolver en control al procedimiento que llamó a la función, el valor del parámetro actual será el último que tuvo el parámetro formal.
Los botones Add e Insert permiten añadir nuevos argumentos, mientras que el botón Delete permite eliminarlos.

Ahora habría que introducir el procedimiento asociado a la función. A continuación se incluye un ejemplo; en él se define la variable Tarifa Postal, la cual al procedimiento y se emplea para contener el valor a devolver.

```
// Calcula tarifa a partir del peso
Real TarifaPostal
Choose case Peso
  case is < 16
    TarifaPostal = Peso * 0.30
  case 16 to 48
    TarifaPostal = 4.50
  case else
    TarifaPostal = -1.0
End choose
Return TarifaPostal
```

No es preciso definir la varible Peso, ya que se trata de un parámetro formal, y por tanto, ya ha sido definido. Tanto el Function Painter como el procedimiento asociado a la función tiene acceso a la declaración, por lo que ésta es conocida.

El menú despegable Past Argument contiene la lista de argumentos de la función. Al seleccionar uno de sus elementos/argumentos, se inserta su nombre justo en la posición actual del cursor.

Para compilar el procedimiento asociado a la función, se dispone de la opción Script del menú Compile o la tecla de acceso rápido Ctrl-L. Si se detectan errores no es posible almacenar la función, pues automáticamente se invoca antes de proceder al almacenamiento.

Un ejemplo

A continuación se muestra un ejemplo de función y su uso. En el evento close de la ventana w_financieros se hace la siguiente llamada a la función f_cerrar_ventanas:

```
...
f_cerrar_ventana (w_financiacion, "financiacion")
...
```

De los argumentos, w_financiacion es un objeto de ventana, mientras que el segundo, "financiacion", es una cadena.

La función tiene dos parámetros formales, aw_seleccionada del tipo window, y ac_nombre, que es una cadena.

El código del procedimiento es el siguiente:

```
String ls_key
// Cambia cursor de la ventana
SetPointer (HourGlass!)
If aw_seleccionada.WindowState = Normal!Then
  // Guarda posisiones  x - y
  ls_key =ac_nombre + " X "
  SetProfileString("carcost.ini","aplication", &
   ls_key, String (aw_seleccionada, X, "!"####"))
 ls_key =ac_nombre + " Y "
  SetProfileString("carcost.ini","aplication", &
   ls_key, String (aw_seleccionada.Y, "!"####"))

//Guarda Dimensiones
ls_key =ac_nombre + " W "
  SetProfileString("carcost.ini","aplication", &
   ls_key, String (aw_seleccionada.W, "!"####"))
ls_key =ac_nombre + " H "
  SetProfileString("carcost.ini","aplication", &
   ls_key, String (aw_seleccionada.H, "!"####"))
End if

SetPointer (Beam!)
Return 0
```

La función se llama con dos parámetros: el primero es el objeto ventana, w_financiacion, y el segundo es una cadena contenido el nombre "financiero".

En la primera sentencia del procedimiento se define una variable local del tipo cadena, ls_key. En la siguiente instrucción se llama a la función de Powerbuilder SetPointer, la cual cambia el icono asociado al cursor; en nuestro caso se cambia el cursor al reloj de arena, transmitiendo así al usuario una sensación de espera.

La sentencia if comprueba si el estado de la ventana es normal o no, como por ejemplo, minimizado. Si el estado de la ventana es normal, se almacenan su posición y dimensiones en un archivo de inicialización de la aplicación.

Para ver la sintaxis de la función de Powerbuilder SetProfileString, puede hacerse el uso de la ayuda Esta función almacena información en la zona del archivo que indique sus argumentos.

Funciones a nivel objeto

Los niveles de acceso de las funciones pueden ser públicos, privados o protegidos. Las funciones a nivel objeto pueden ser privadas o protegidas. Para la creación de este tipo de funciones se seguirán los mismos pasos que los indicados para una función global. Al activar Function Painter desde cualquiera de los siguientes Painters: Window, Menu o UserObject, se puede especificar el nivel acceso. Niveles de acceso privados o protegidos harán que la función se asocie al objeto desde el cual se activó el Function Painter.

Modificaciones de Funciones

Una función puede modificarse fácilmente desde Function Painter. Si se desea cambiar la declaración, la opción Function Declaration del menú Edit vuelve a presentar la ventana de declaración de la función.

Llamada a una Función

Para llamar a una función definida por el usuario, basta con su nombre y la relación de argumentos que necesita, encerrados en paréntesis:

```
integer total
total = f_pago (12, 8.5)
```

En el ejemplo se le llama a la función f_pago y se le pasan dos argumentos: una cantidad entera y un número real.

El número de parámetros actuales tiene que coincidir con el número de parámetros formales indicados en la declaración de la función. Además, según el orden de llamada y definición, el tipo de cada parámetro actual ha de coincidir con el parámetro formal. Es decir, si se definió el primer argumento de tipo real, al llamar a la función, el primero de los argumentos de la llamada tiene que ser también de tipo real.

Las funciones a nivel objeto tienen que incluir el nombre del objeto al que están asociadas, conforme con la siguiente sintaxis:

```
objeto.Funcion (argumentos)
```

Un ejemplo de llamada a una función al nivel de objeto sería:

```
w_entrada.f_leerNombres("t")
```

3.9 Database Painter

En esta sección se aplica toda la teoría mostrada en el capítulo 2 acerca de Base de datos.

Cualquier manejador de base de datos incluye, además del motor de base de datos, una serie de herramientas complementarias, como por ejemplo, utilidades de SQL para la creación y manipulación de tablas. Algunas de estas funciones pueden realizarse desde el Database Painter, como por ejemplo:

- √ Crea nuevas tablas o borrar alguna de las existentes.
- √ Añade o elimina columnas de una tabla.
- √ Modifica columnas de una tabla.
- √ Define y elimina llaves primarias o foráneas.
- √ Crea o borrar índices.
- √ Crea nuevas vistas o borrar alguna de las existencias.

El acceso a estas facilidades puede verse limitado por los permisos que el administrador de la base de datos haya asignado al usuario que las está utilizando, o por las propias posibilidades del Database Painter.

Atributos Adicionales de una Columna

Con Powerbuilder es posible crear y mantener formatos de presentación y validación, máscaras de edición y valores iniciales, y por lo tanto, utilizables en las aplicaciones Powerbuilder. Estos

atributos se conocen como adicionales y se aplican a columnas. Afectan a la presentación en la pantalla de contenido de las columnas.

Por ejemplo, una máscara de edición determina que los caracteres que pueden utilizar el usuario. Los formatos de validación comprueban la validez de los valores introducidos.

Esta información se almacenan en tablas de la base de datos, las cuales Powerbuilder automáticamente crea y gestiona. En ningún momento el manejador de la base de datos mantiene o utiliza estos atributos.

Utilidad para la Manipulación de Datos.

El Database Painter contiene una utilidad para introducir, actualizar y borrar información de la base de datos. La interfaz de esta herramienta es simple y fácil.

Utilidad para Administración de la Base de Datos

El Database Painter también incluye una utilidad para la administración de la base de datos. Con esta herramienta es posible cambiar permiso de acceso, así como construir y ejecutar sentencias SQL.

Empleo del Database Painter

El Database Painter se inicia haciendo clic sobre un icono, bien en el PowerPanel o en el PowerBar. El Database Painter intenta la conexión a la última base de datos usada. Si es posible la conexión, aparece entonces en la ventana Select Tables En esta ventana aparece una lista con cada una de las tablas y vistas definidas para la base de datos.

Selección y Desplazamiento de Tablas

La ventana Select Tables permite seleccionar cualquiera de las tablas o vistas que muestra. Haciendo clic en el botón New se crea una nueva tabla.

La ventana también contiene una casilla de verificación, Show system tables (Presentar las tablas del sistema), que cuando se encuentra activada incluye las tablas del sistema en la lista de tablas y vistas mostradas. Activándola podrán verse las tablas en las que Powerbuilder almacenan los atributos adicionales a las columnas.

Tras hacer clic, el nombre de la tabla queda resaltado. Seleccionadas las tablas, haga clic en el botón Open. El Database painter muestra entonces las tablas de seleccionadas. Haciendo clic en la barra de título de la tabla, y manteniendo pulsado el botón del ratón, es posible arrastarla hasta la posición que desee; entonces basta con soltar el botón del ratón. En la pantalla se presenta cada una de las tablas, con sus columnas y la descripción de las mismas. También se presenta las llaves primarias o foráneas, y sus posibles relaciones.

Presentación de Información Sobre las Llaves

Haciendo doble clic en el icono de la llave aparece la información sobre la misma. Haga doble clic en el icono de la llave primaria (con un carácter "P" de color verde) de la tabla.

Presentación y Mantenimiento de una Tabla

Haciendo doble clic dentro del área que ocupa la definición de una tabla o haciendo clic en el botón derecho, aparece un menú emergente en el que encontramos la opción Definition. La definición de una tabla incluye características al nivel de tabla y a nivel columna.

Características a nivel tabla

las características a nivel tabla que presenta la ventana Alter Table son las siguientes:

√ Tipo de letra de los datos, cabeceras y etiquetas.

√ Comentarios sobre la tabla.
√ Llave primaria y foránea.

Tipos de letra

haciendo clic en el botón Font ("tipo de letra"), aparece el cuadro de diálogo Font, en el que es posible asignar diferentes tipos de letra, tamaños y estilos separadamente para:

√ Datos almacenados en la tabla.
√ Cabeceras de las columnas.
√ Etiquetas para las columnas.

Comentarios

Haciendo clic en el botón Comment ("Comentario"), aparece el cuadro de diálogo Comments; desde aquí se puede introducir o modificar la descripción de una tabla. Este comentario se almacena en una tabla adicional definida por Powerbuilder.

Llave Primaria

Haga clic en el botón Primary Key (Llave Primaria) para mostrar el cuadro de diálogo "Primary key Definition".

Para cambiar la llave primaria, se haría clic sobre los nombres de las columnas que la forman; con este orden aparecen en el cuadro superior, Key Columns. Al hacer clic en una columna que ya forma parte de la llave primaria, se suprime de ésta. Haga clic en el botón Cancel para cerrar la ventana Primary Key Definition.

Es preciso recordar que a este cuadro de diálogo, Primary Key Definition, también se puede llegar haciendo doble clic en el icono Primary Key del Database Painter.

Llaves Foráneas

Haciendo clic en el botón Foreign Key (Llave Foránea), aparece el cuadro de diálogo Foreign Key Selection.

Aunque es posible tener más de una llave foránea. Haciendo clic en el botón New aparece el cuadro de diálogo Foreign key Definition, para la definición de llaves foráneas:

Para definir una nueva llave foránea, primero se asignaría un nombre y después habría que escoger la tabla foránea. Entonces se seleccionaría las columnas de la tabla que formarían la llave foránea.

Haga clic en botón Cancel para cerrar este cuadro de diálogo y volver a la ventana Foreign key Selection. Haga clic en el botón Edit para mostrar la definición de la llave foránea.

Características a nivel columna

Desde el Database Painter, y concretamente desde la ventana Alter table ("Modificación de una tabla"), es posible especificar muchas de las características a nivel columna de una tabla. El motor del manejador de la base de datos empleado condiciona las posibilidades disponibles, pero generalmente entre éstas se incluyen el nombre y tipo de los datos, de la columna, así como si ésta permite valores nulos.

Atributos adicionales de una columna

Powerbuilder, y no el manejador de la base de datos, añade y mantiene los atributos adicionales de una columna; entre estos atributos se incluyen las cabeceras de la columna y valores iniciales. Cualquier aplicación Powerbuilder con acceso a la base de datos puede accesar a los atributos adicionales.

A continuación se incluye una tabla con los atributos adicionales que Powerbuilder es capaz de gestionar.

Atributo adicional de la columna	Descripción
Formato de presentación	Define el formato de presentación de los datos.
Estilo de edición	Define el formato de los datos que un usuario intenta almacenar en una columna.
Regla de validación	Fórmula o criterio que decide la validez de los datos Tecleados.
Cabecera	Texto por defecto a emplear como cabecera de la columna.
Comentario	Comentario o descripción de la columna seleccionada.
Alineación	Alineación por un defecto, a la izquierda, derecha o centrada, para los datos de una columna.
Ancho, alto	Dimensiones en pulgadas por defecto de la columna, al incluirla en un obejto DataWindow.
Valor inicial	Valor inicial para la columna.
Etiqueta	Etiqueta por defecto para la columna. Suelen aparecer a la izquierda de la columna.

Desde la ventana Alter Table pueden modificarse los valores de cualquiera de los anteriores atributos. Al hacer clic sobre el botón Alter, Powerbuilder construye las sentencias SQL necesarias para modificar cualquiera tabla o los atributos adicionales de la columna.

Con los atributos adicionales se dispone de un potente conjunto de valores por defecto mientras se construye la aplicación. Esto facilita la unificación del aspecto de los objetos DataWindow y también facilita su creación. No es preciso especificar estas características cada vez que se construye un objeto DataWindow; una vez definidos los atributos, pueden utilizarse para cada objeto DataWindow, así como compartirlos entre los objetos de la aplicación, e incluso entre distintas aplicaciones.

Manipulación de la base de datos

Con la utilidad Data Manipulation (Mantenimiento de datos) es posible, para una base de datos:

- √ Recupera los datos.
- √ Presenta los datos.
- √ Inserta los datos.
- √ Modifica los datos.
- √ Borra los datos.

La manipulación de los datos de una tabla se realiza cómoda y rápidamente con esta herramienta. Esto también puede hacerse seleccionando el elemento Data Manipulation del menú Objects. A continuación se escogerá el estilo de presentación, apareciendo entonces la ventana de mantenimiento de datos (Data Manipulation) con la información de la tabla.

Con los siguientes botones se controla y manipula la información recuperada de la base de datos:

Excepto First y Last, el resto de los iconos tienen un elemento de menú asociado. De izquierda a derecha, las funciones de los elementos son:

Control	Función
Retrieve	Recuperar. Recupera información de la Base de datos, mostrandola en la pantalla.
Update db	Actualizar la base de datos. Cualquier cambio o modificación realizada sobre los datos mostrados en pantalla se actualiza la base de datos.
insert row	Inserta una fila "en blanco". Inserta una nueva fila sobre la actual. Esto no afecta a la base de datos.
delete row	Suprime una fila. Elimana la fila Actual. Esto no afecta a la base de Datos.
First	Principio. Desplazamiento hasta la Primera página de información.
Prior	Anterior. Desplazamiento a la anterior Página de información.
Next	Siguiente. Desplazamiento a la siguiente Página de información.
Last	Final. Desplazamiento a la última Página de información.

Además existen dos opciones en el menú File, una para importar datos desde un archivo, y otra para exportar la información presentada a un archivo externo.

Intente desplazarse a través de la información de la tabla recuperada, insertando y suprimiendo filas.
Incorporación de nuevos elementos a la base de datos

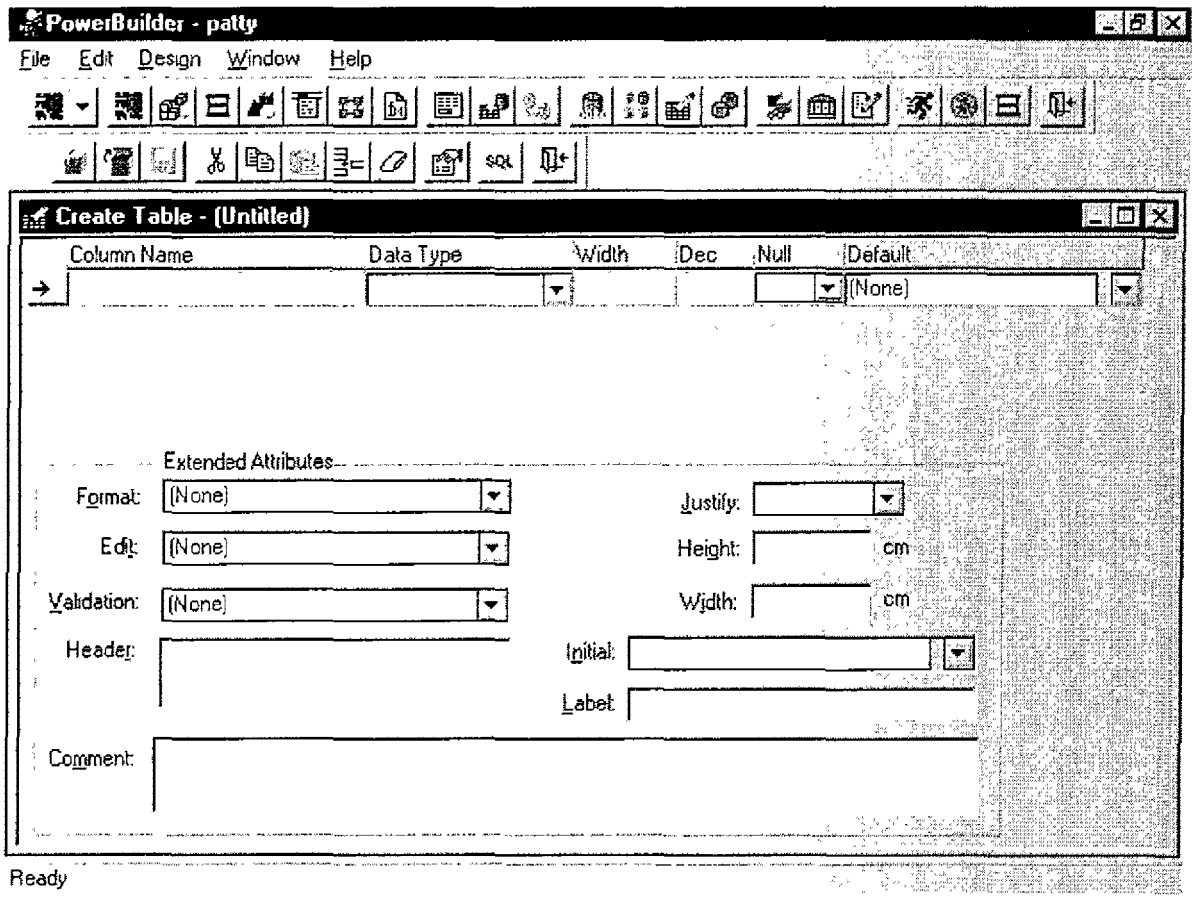
Desde el Database Painter es posible añadir a la base de datos (Ver SQL en Capítulo II):

- √ Nuevas tablas.
- √ Índices.
- √ Formatos de presentación.
- √ Estilo de edición.
- √ Reglas de validación.

Creación de una nueva tabla

Las tablas se crean desde la ventana Create Table, a la cual se accede de tres formas posibles:

- √ haciendo clic en el botón New de la ventana Select Tables
- √ haciendo clic en el botón New Table del toolbar
- √ seleccionando la opción New del menú Objects, y entonces la opción Table del menú en cascada aparece



En el primer lugar se indica el nombre de la nueva tabla. Introducido el nombre, se especifica el tipo de letra (botón Font) y una descripción o comentario acerca de la tabla (botón Comment).

Ahora se podrían definir los atributos adicionales de cada columna en la forma en que se detalló con anterioridad o hacerlo más tarde.

Definidas todas las columnas que forman la tabla, se selecciona una llave primaria para la misma. De no seleccionar ninguna, al intentar cerrar la ventana aparecerá un mensaje recordando esta situación.

Creación de nuevos índices

Los índices pueden mejorar la eficiencia de las aplicaciones. Los índices se crean desde la ventana Create Index, a la cual se llega de las dos siguientes maneras:

- √ Haciendo clic en el botón key del toolbar.
- √ Seleccionando la opción New del menú Objects, y entonces la opción Index del menú en cascada que aparece.
- √ Indica el nombre del índice.
- √ Indica si se trata de un índice simple o compuesto.
- √ Escoge entre ordenación ascendente o descendiente.
- √ Selecciona las columnas del índice.
- √ hacer clic en el botón OK

No todas las bases de datos aceptan llaves ascendentes. Al hacer clic en el botón OK se construyen las sentencias SQL necesarias para definir y añadir el índice a la base de datos.

Borrado de Índices, Tablas, Vistas o Llaves

También se puede utilizar el Database Painter para borrar índices, llaves, vistas o tablas. En el Database Painter, y seleccionando el elemento a borrar, por ejemplo, la tabla, se haría clic en el botón Drop del toolbar o en la opción Drop del menú Objects. A continuación aparece una ventana de respuestas solicitando la confirmación para el borrado; haciendo clic en el botón OK se borrará el elemento seleccionado.

Empleo de Atributos Adicionales

En el Database Painter se pueden definir, modificar o borrar formatos de presentación, reglas de validación o estilos de edición, los cuales permanecen en un repositorio accesible por cualquier aplicación que pueda conectarse a la base de datos. También se puede utilizar el Database Painter para asignar cualquiera de estos atributos a una columna en particular. En las siguientes secciones se describirá brevemente la forma de actualización. La documentación de Powerbuilder contiene información más detallada al respecto.

Aunque los atributos adicionales se almacenan en tablas de la base de datos, no forman parte del sistema de gestación de la base de datos. Únicamente los emplea Powerbuilder.

También se pueden definir atributos adicionales en el DataWindow Painter, pero entonces sólo podrán ser utilizados por la ventana en custión y sus descendientes. Utilizando el DataWindow Painter es posible:

√ Aceptar el formato de presentación por defecto asignado a la columna en el Database Painter.
√ Rechazar este formato y escoger otro de entre los del repositorio.
√ Crear un formato, sin nombre ni identificación, para utilizarlo únicamente en la columna o DataWindow en cuestión.

Al crear el objeto DataWindow, se incluyen en él los correspodientes atributos adicionales, no existiendo un enlace entre los atributos y el repositorio. Las modificaciones de los atributos del repositorio **no se actualizan automáticamente** en el objeto DataWindow. Es preciso volver a crear el DataWindow. PowerSoft distribuye un sincronizador de atributos adicionales con la versión Enterprise de Powerbuilder. También se puede obtener esta utilidad en el Tablero de Mensajes de PowerSoft. Esta utilidad es gran ayuda cuando se precisa sincronizar un objeto DataWindow con las modificaciones realizadas en los atributos adicionales almacenados en el repositorio.

Formatos de presentación

La principal ventaja de disponer de un repositorio es que, a menudo, el formato de presentación en una propiedad es en sí del dato a presentar. Por ejemplo, se puede crear un formato de presentación para cantidades de dinero, con el nombre de la moneda y separadores de miles y decimales. Se pueden definir un estilo para presentar la fecha con las iniciales del mes, en lugar de un número.

Estos estilos se encuentran disponibles durante la creación de un DataWindow, y se aplican al ejecutar la aplicación.

Obsérvese que el formato de presentación no tiene lugar durante la introducción de los datos. Cuando el foco llega a un campo de edición, el formato desaparece. Si además se desea que el formato se aplica también durante la introducción de datos, habría que utilizar una Máscara de Edición, que se describirá más tarde.

Acceso al Painter Display Format

En el Database painter, y teniendo en pantalla la tabla para cuyas columnas se quiere crear el formato, haga clic con el botón derecho en la columna en cuestión. Aparece entonces un menú emergente en el que se seleccionará la opción Display.

Se puede seleccionar uno de los estilos predefinidos, o crear uno nuevo. Para crear un nuevo formato de presentación, haga clic en el botón New.

En esta ventana se asigna un nombre de estilo y se especifica la máscara de presentación. Para probar la máscara, introduzca un valor en el campo Test Value y haga clic en el botón Test. El resultado de la aplicación del estilo aparece en el campo Result.

Definición de Formatos de Presentación

Un formato de presentación contiene una máscara, cuyos caracteres tienen un significado especial. Los caracteres de una máscara pueden representar:

- √ números
- √ cadenas de caracteres
- √ fechas
- √ horas

Por ejemplo, en una máscara de presentación de cadenas, cada carácter @ representa un carácter cualquiera a mostrar. Otros caracteres se asumen literalmente, y se muestran así. Por ejemplo, la máscara

(@@) @@@ @@ @@

presentaría un número teléfono, como por ejemplo 916555112, de la forma,

(91) 655 51 12

Una máscara se compone de secciones, separadas por puntos y comas. El significado de cada sección depende del tipo de dato a presentar.

Por ejemplo, en máscaras numéricas, la segunda sección se aplica a los números negativos. Así la siguiente máscara sólo presenta cerrados entre parentesis los números negativos:

$ ## , ## 0 , ($ ## , ## 0)

Al menos ha de haber una sección. El resto son opcionales.

Se pueden combinar varios formatos en única máscara. Espacios en blanco separan elementos de distintos tipos. Por ejemplo, la siguiente máscara es útil para mostrar la fecha y hora:

dd-mm-yyyy h:mm

La Guía del Usuario (User's Guide) de Powerbuilder contiene el capítulo, "Displaying and Validating Data" (Presentación y Validación de Datos), con más información sobre los formatos de máscaras de presentación. También el botón de ayuda, Help, de la ventana Display Format proporciona información adicional al respecto. Por ejemplo, descripciones de los tipos diferentes de máscaras (fecha, cadenas, etc.) pueden obtenerse rápidamente desde la ayuda de Powerbuilder.

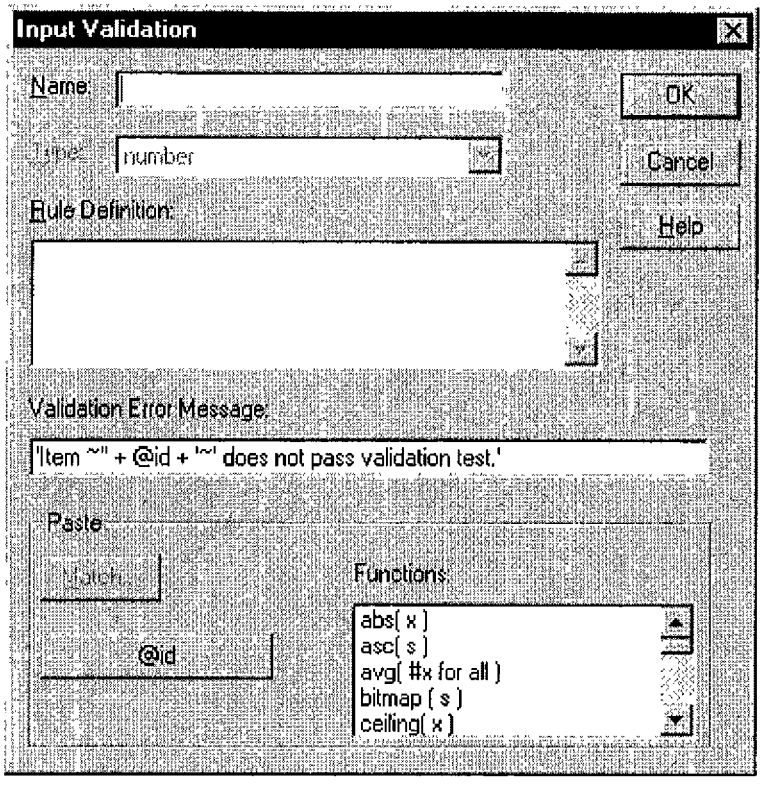
Haciendo clic en el momento "display formats", aparecen los temas asociados.

La tabla pbcatfmt (ha de estar activada en la casilla de verificación "Show system tables") es el repositorio de los formatos de presentación. Consulte la tabla para ver el formato de la información almacenada.

Reglas de Validación

Las reglas de validación comprueban la validez de los datos introducidos por el usuario. Por ejemplo, la siguiente asegura que se ha introducido un valor mayor que cero:

@ col > 0



Al igual que los formatos de presentación, las reglas de validación se almacenan en el repositorio y se incluyen en los objetos DataWindow cuando se crean; tampoco existe ningún tipo de enlace entre el objeto DataWindow y el repositorio.

Las reglas de validación se incorporan automáticamente al objeto DataWindow en el momento de su creación. Al igual que ocurría con los dos formatos de presentación, es posible rechazar las reglas del repositorio por otras nuevas.

Powerbuilder construye y manteniene las validaciones. El sistema de gestión de la base de datos queda la margen de todas estas operaciones. La ayuda on-line de Powerbuilder contiene más información al respecto.

La evalución de una regla de validación proporciona un valor lógico, CIERTO O FALSO. En el primer caso se aceptan datos tecleados; en el segundo caso de dispara el evento ItemError.

Acceso al Painter Validation Format

En el Database Painter, y teniendo en pantalla la tabla para cuyas columnas se quiere crear una regla de validación, haga clic en el botón derecho en la columna. Aparece entonces un menú emergente en el que se seleccionará la opción Validation.

En este cuadro de diálogo se puede seleccionar una regla existente o crear una nueva. Para crear una nueva regla de validación, se hace clic en el botón New. Aparece entonces la ventana Input Validation.

Haciendo clic en el botón Help, se obtiene información de ayuda para el diseño de la regla de validación. Algunos ejemplos de reglas de validación pueden consultarse en la tabla del sistema pbcstvld. Esta tabla es el repositorio de las reglas de validación.

Estilos de Edición

Los estilo de edición aplican tanto a la presentación como a la introducción de los datos. Supongamos una columna que guarda los valores O, D o V para indicar el estado de un empleado: Ocupado, trabajando; Disponible, asignable; de Vacaciones. Además se quiere que la elección se haga mediante botones de opción.

Ocupado
Disponible
Vacaciones

Existen los siguientes estilos de información:

√ Campo de definición, en el que el usuario puede introducir un valor (es por defecto).
√ Lista despegable.
√ Casilla de verificación.
√ Botones de opción.
√ Máscara de edición especificando caracteres permitidos.
√ Lista despegable con DataWindow asociada.

Mantemiento de los atributos adicionales

En el menú Objects del database Painter, al seleccionar cualquiera de las siguientes opciones: Edit Style Maintenance (Mantenimiento de Estilos en Edición), Display Format Maintenance (Mantenimiento de Formatos de Presentación) o Valition Maintenance (Mantenimiento de Reglas de Validación), aparece un cuadro de diálogo desde el que es posible crear uno nuevo, modificar uno existente o borrarlo.

3.10 Componentes de una Aplicación

Aplicaciones MDI

Gran parte de aplicaciones Windows ofrece una interfaz denominada MDI (Multiple Document Interface, Interfaz con Múltiples Documentos). Una aplicación MDI tiene una ventana principal, MDI frame window, que hace el marco dentro del cual se pueden abrir otras ventanas diferentes denominadas hojas, sheets. Powerbuilder es un excelente ejemplo de aplicación MDI.

La figura muestra a Powerbuilder con dos painters abiertos. La ventana MDI frame es la ventana principal de Powerbuilder (con barra de título, la barra de menú y barra de iconos). Además aparecen dos sheets ("hojas"), una parte Library Painter y otra para la Aplication painter.

Las principales aplicaciones Windows, como procesadores de texto u hojas de cálculo, son aplicables en MDI.

La interfaz MDI es la más apropiada para aplicaciones en las que se realizan habitualmente tareas similares. Por ejemplo, una hoja de cálculo puede manipular varias hojas a la vez, o un procesador de texto puede editar varios documentos simultáneamente. En estos ejemplo, todas las tareas son similares. En otros casos, las tareas pueden no ser tan parecidas, pero aún puede ser preferible esta interfaz. Por ejemplo una aplicación en la que se desea mantener simultáneamente tres ventanas abiertas para el mantenimiento de clientes, alta de peticiones y control de inventario.

En cualquier caso, el usuario encuentra una interfaz única e integrada, lo cual hace que la aplicación sea más fácil de usar. Aplicaciones con varias ventanas y continuas conmutaciones a cada una de ellas, han de diseñarse como MDI.

Componentes de la Interfaz MDI

La ventana principal de la aplicación MDI consta de varios componentes: la estructura MDI, una barra de menú, el área cliente y, opcionalmente, el área de estado o MicroAyuda (MicroHelp).

Estructura MDI

La estructura MDI es la ventana principal de la aplicación; por tanto, incluye el área cliente. Powerbuilder admite dos tipos de estructura, estándar y personalizada.

En cualquier estructura MDI se dispone de las siguientes teclas de acceso rápido.

Tecla	Acción
Ctrl-F4	Cierra la hoja activa, activando la hoja anterior activa.
Ctrl-F6	Activa la siguiente hoja.

Estructura Estándar

Una estructura estándar contiene, opcionalmente, una barra de menú y un área de estado o MicroAyuda. Las hojas abiertas se muestran solamente en el área cliente.

Las hojas pueden heredar el menú de la ventana principal, o tener el suyo propio. La barra de menú de una aplicación MDI sólo se muestra en la ventana principal, nunca en la hoja.

Aunque no es preciso que una aplicación MDI disponga del menú es aconsejable tener al menos una para la aplicación, ya que así es más fácil el manejo cuando no existan hojas abiertas.

Al abrir una nueva hoja, se ajusta el tamaño del área cliente, para que pueda presentar en su totalidad.

Estructura Personalizada

Al igual que la estructura estándar, puede contener, opcionalmente, una barra de menú o ún área del estado o MicroAyuda; pero, sin embargo, el área cliente puede contener objetos, como por ejemplo, botones y texto estático.

Es obligación de la aplicación ajustar las dimensiones del área cliente para que las hojas se muestren en su totalidad; de no ser así, las hojas podrían presentarse parcialmente.

Área cliente

Una aplicacón MDI puede tener varias hojas abiertas en el área cliente. Al abrir la ventana principal MDI, automáticamente se crea el control MDI_1, el cual identifica el área cliente de la estructura MDI. En estructuras estándar, Powerbuilder automáticamente ajusta el tamaño de la estructura MDI_1. Para las estructuras personalizadas, la aplicación debe incluir un procedimiento al respecto (que ajuste las dimensiones de la ventana a un tamaño adecuado) en el Resize.

Desde el PowerScript Painter, inicie el Browse Objets y haga doble clic en el nombre de una ventana MDI para obtener una lista de sus controles. Uno de estos controles ha de ser el área cliente MDI_1. Haga clic en cualquiera de los botones de opción Attibutes o Functions para obtener atributos y funciones asociadas.

Hojas de una Interfaz MDI

Una hoja MDI es una ventana que se muestra en el área cliente de una estructura MDI. Cualquier tipo de ventana puede ser una hoja MDI, excepto si es una ventana principal MDI. Para abrir una hoja se emplea la función OpenSheet.

Una hoja permite a un usuario realizar tareas específicas; puede abrir varias hojas a la vez y conmutar entre ellas. Por ejemplo, Powerbuilder cada Painter es una hoja; el usuario puede abrir el Application Painter, y además el Library Painter, y conmutar entre ellos.

Activación

Sólo puede estar activa una ventana por aplicación, la que tiene la barra de título resaltada. Cualquier entrada desde el teclado o ratón se envía a la ventana activa.

Al abrir una aplicación MDI, su ventana principal pasa a ser la ventana activa hasta que el usuario conmuta a otra aplicación o abre una hoja MDI (entonces, tanto la ventana principal como la hoja entan activas), si existen varias hojas abiertas, sólo la ventana principal y una hoja estaran activas.

MicroAyuda

Como el tamaño de la barra de menú es limitado, sólo se puede incluir un número limitado de palabras. Por ejemplo, un procesador de texto puede tener los siguientes elementos en su barra de menú: Archivo, Edición, Ver, Insertar, Formato, Herramientas, Tabla, Ventana, Ayuda. Cuando existen tantas opciones, sólo se puede incluir una palabra. Para ayudar al usuario, las aplicaciones MDI son capaces de presentar información adicional al elemento del menú en el área de estado o MicroAyuda.

Existen dos opciones al crear la ventana principal MDI, con MicroAyuda (MicroHelp) o sin ella.

Desde el Menu Painter se asocia a cada elemento del menú el texto que debe aparecer en el área de MicroAyuda de la aplicación MDI cuando el usuario lo seleccione. Por ejemplo, haciendo clic y manteniendo pulsado el botón sobre el icono Application Painter, aparece el siguiente texto en el área de MicroAyuda: "Run application painter" (iniciar el Application painter).

Menús

Los menús se crean en el Menu Painter y se asocian a una ventana desde el Window Painter. Esto no cambia las aplicaciones MDI.

La ventana principal de una aplicación MDI puede tener asociado un menú. Entonces todas las hojas heredan el menú. Esto puede ser interesante en aplicaciones como procesadores de texto, que aunque puedan tener varios documentos abiertos a la vez, los comandos empleados para su manipulación son siempre los mismos.

En cambio, en otras aplicaciones cada hoja tiene un propósito diferente. En estos casos, se pueden asociar un menú distinto a cada hoja. El menú de la hoja de muestra en la barra de menú de la ventana principal, no en la propia hoja.

El menú que se muestra es el menú actual. Si se abre una hoja sin menú, se toma el actual, es decir, el de la última hoja activa.

Construcción de Aplicaciones MDI

No existen grandes diferencias al crear una aplicación MDI, respecto de otras no MDI.
Activación

Una aplicación se construye creando un objeto aplicación y una librería. A continuación se creará la ventana principal MDI; cuando el Window Painter muestre la nueva ventana, haga doble clic en la ventana o seleccione la opción Window Style (Estilo de la Ventana) del menú Design (Diseño).

Seleccione el tipo de la ventana, MDI Frame o MDI with MicroHelp (Estructura MDI sin o con MicroAyuda). Es preferible usar MicroAyuda, a menos que exista una razón justificada para lo contrario.

Cómo abrir una estructura MDI

Creada la ventana principal MDI, regrese al Application Painter y edite el procedimiento asociado en el evento open de la aplicación.

A continuación se incluye un procedimiento ejemplo, en el que se obtiene un fichero de inicialización la información necesaria para la conexión a la base de datos. Al no indicarse la ubicación del fichero de inicialización, el fichero debe encontrarse en alguno de los directorios que contenga la variable PATH (variable de DOS), o en el actual directorio.

La aplicación intenta la conexión a la base de datos; si es posible, abre la ventana principal MDI. Obsérvese que no existen diferencias al abrir una ventana MDI o no MDI.

```
//Procedimiento ejemplo para el evento open
// de una aplicación MDI
sqlca.DBMS = &
ProfileString("demo.ini", "sqlca", "dbms" )
sqlca.database = &
ProfileString("demo.ini", "sqlca", "database" )
sqlca.userid = &
ProfileString("demo.ini", "sqlca", "userid" )
sqlca.dbpass= &
ProfileString("demo.ini", "sqlca", "dbpass" )
sqlca.logid = &
ProfileString("demo.ini", "sqlca", "logid" )
sqlca.logpass = &
ProfileString("demo.ini", "sqlca", "logpass" )
sqlca.servername = &
ProfileString("demo.ini", "sqlca", "servername" )
sqlca.dbparm = &
ProfileString("demo.ini", "sqlca", "dbparm" )
connect;
if sqlca.sqlcode <> 0 then
  MessageBox (" Conexión a la base de dato" , &
        sqlca.sqlerrtext)
halt close
return
end if
Open (w_principal)
```

Creación de un Menú

A continuación se crea un menú para la aplicación. Han de agruparse elementos lógicamente seleccionados, y es recomendable seguir convenidos de Windows. Obsérvese las opciones, Archivo, Ventana y Ayuda, tan comunes de las aplicaciones Windows.

Entre las opciones del menú archivo se encuentra Guardar. Este elemento tiene asociado el mensaje "Guarda la información introducida", como texto de MicroAyuda. También se encuentra como elemento Imprimir, Seleccionar Impresora y salir. Estas opciones son habituales en las aplicaciones Windows.

El procedimiento asociado a la opción Seleccionar Impresora podria ser:

```
// m:seleccionar impresora-Evento clicked
If PrintSetUp ( ) = -1 Then
MessageBox (&
        "Seleccionando de Impresora" , &
        "No es posible la selección")
End if
```

Y el asociado a la opción Salir

```
//Pide al usuario confirmación de
```

```
//la salida y finaliza la aplicación

If MenssageBox('Salida de la aplicación', &
'¿Seguro que desea Salir?', Question! , &
YesNO!) = 1 Then
        Close (ParentWindow)
End if
```

Este procedimiento solicita al usuario la conformidad para abandonar la aplicación. Respuestas afirmativas cerrarán la ventana principal de la aplicación, finalizando la ejecución de la aplicación.

Las opciones del elemento Ventana son: Mosaico, Capa, Cascada y Organizar Iconos, también habituales a las aplicaciones Windows.

Al evento clicked del elemento Ventana: Mosaico se le podría asociar el siguiente procedimiento:

```
//Procedimiento para el evento clicked
//de m_cascada
ParentWindow.ArrangeSheets(Tile!)
```

Para los restantes elementos, el procedimiento sería identico; la única diferencia sería el argumento (un tipo enumerado) a la función ArrangeSheets:

- √ Layer!, para el elemento Capa
- √ Cascade!, para el elemento Cascada
- √ Icons!, para el elemento organizar Iconos

Creación de una hoja

A continuación se creará una hoja MDI, para ello, desde el Window Painter, se creará una nueva ventana MDI, cuyo tipo será Child.

Aunque esta ventana se emplee como hoja MDI, en realidad se trata de un objeto ventana normal, al que se pueden añadir controles, como por ejemplo, un DataWindow.

Eventos definidos por el usuario

Cualquier usuario puede definir sus propios eventos. Al seleccionar el elemento User Events del menú Declare Window Painter.

Para crear un nuevo evento, se indica su nombre al final de lista, por ejemplo actualizar_dw. Al introducir el nombre, se activa la lista de todos los eventos Windows, aunque no todos esten disponibles para los objetos Powerbuilder. Además de los eventos Windows, se incluye un conjunto de eventos de usuario cuyos identificadores comienzan por pbm_custom para el uso que éste desee.

Tras guardar los cambios y cerrar la ventana Events, al iniciar Script Painter y desplegar la lista Select Event se puede comprobar que existen los nuevos eventos (curiosamente son los últimos de la lista).

Un ejemplo del procedimiento asociado al evento actualizar_dw podría ser:

```
f_actualizar_dw (dw_inf_financiera, "fncr")

Y para imprimir_dw
        Integer ImpId
        ImpId=PrintOpen("Información Financiera")
        PrintDataWindow(ImpId, dw_inf_financiera)
        PrintClose(ImpID)
```

```
Window HojaActiva
HojaActiva=w_principal. GetActiveSheet ( )
If IsValid(hojaActiva) Then

HojaActiva.TriggerEvent ("actualizar_dw")
End if
```

al seleccionar este elemento, se busca cuál es la hoja activa de estos momentos, activándose el evento actualizar _dw en dicha hoja.

La posibilidad de que el usuario puede definir sus propios eventos y activarlos cuando desee, es muy útil, simplificando muchas tareas. Análogamente, para opción Print del menú.

```
Window HojaActiva
HojaActiva=w_principal.GetActiveSheet ( )
If IsValid(HojaActiva) Then
 HojaActiva.PostEvent ("Imprimir_dw")
End If
```

En este procedimiento se ha llamado a la función PostEvent, en lugar de a TriggerEvent. La diferencia entre ambas es que mientras la última (triggerEvent) dispara inmeditamente el vento, la primera (PostEvent) lo añade a la cola de eventos del objeto, activandose cuando Windows llegue a él.

Cómo abrir una hoja

A continuación se incluye el procedimiento para abrir una hoja desde una opción menú.

```
OpenSheet(w_inf_financiera, &
  "w_inf_fianciera", w_principal, 4, &
  Cascaded!)
```

Información sobre la función OpenSheet y sus argumentos pueden obtenerse de la ayuda on-linc de Powerbuilder. La función OpenSheet abre una nueva hoja, de forma parecida a la función Open para las ventanas.

El evento open de la ventana w_inf_financiera podría contener el siguiente procedimiento, el cual se ejecutaría al abrir la hoja:

```
SetPointer(HourGlass!)
dw_inf_financiera.SetTransObject(SQLCA)
dw_inf_fiananciera Retrive ( )
m_principal.m_entradatos.m_inffinanciera.check ( )

//inicializa dimensiones
w_inf_fianaciera.width= &
  profileInt ("demo.ini", "aplicacion",
"InfFinancieraW", 2000)
w_inf.financiera.height= &
   ProfileInt ("demo.ini", "aplicacion",
"InfFinancieraH", 1100)
//Inicializa posición
w_inf.financiera.X= &
   ProfileInt ("demo.ini", "aplicacion",
"InfFinancieraX", 100)
w_inf.financiera.Y= &
   ProfileInt ("demo.ini", "aplicacion",
"InfFinancieraY", 100)
SetPointer (beam!)
```

Lo primero que hace el procedimiento es cambiar el icono del cursor a un reloj de arena, así el usuario que está ejecutando alguna acción que puede suponer una cierta espera.

A continuación, inicializa el objeto transacción y recupera la información de la base de datos en los buffers del DataWindow.

En la siguiente línea se añade una marca de verificación del elemento menú, con la que se tiene indicación de que la hoja ya se ha abierto.

Las siguientes sentencias dimensionales la altura y anchura de la hoja con los mismos valores de la última vez que se útilizo la ventana (estos valores se guardan en el fichero de inicialización de la aplicación).

Como ya es posible trabajar con la hoja, se restura el icono del cursor ("Y").

Finalmente al evento close de la hoja se podría asociar el siguiente procedimiento:

```
f_cerrar_hoja(w_inf_financiera, "fncr")
m_principal.m_entradatos.m_inffinanciera.unchecked()
```

El procedimiento cierra la hoja y quita la marca de verificación al elemento del menú.

Toolbars

Con toolbar se facilita el manejo de una aplicación, al tener que hacer clic en el correspondiete icono, en lugar de seleccionar un elemento del menú. Las toolbars de Powerbuilder son un ejemplo de este modo de operación.

Es muy fácil añadir una toolbar. Al definir un nuevo elemento de un menú, se le puede asociar un icono mediante el botón Change:

Cualquier gráfico en formato estándar Windows por ejemplo, bmp se puede asociar en forma de icono a un elemento del menú.

Al asociar un menú a una ventana en el Window Painter, es asociado también al correspondiente Toolbar formada por los iconos que se hayan asignado a los elementos del menú en cuestión.

Como ejemplo de manejo y posibilidades de las toolbars en aplicaciones MDI, utilice en menú Options del Window Painter para mostrar u ocultar dos diferente toolbars, una con iconos de colores, y otra con iconos de opciones de texto.

Definición de la aplicación

Los programas Windows son archivos ejecutables, y por tanto, con extensión Exe. Así por ejemplo, pb040.exe. Estas librerias son archivos con extensión Dll (de "Dynamic Link library", Biblioteca de Enlaces Dinámicos). Durante la ejecución de un aplicación Powerbuilder, se puede llamar además a otro tipo de librerías, archivos con extensión Pbd (de "Powerbuilder Dynamic Library", Librería Dinámica de Powerbuilder), especificas del propio Powerbuilder.

Para distribuir una aplicación se tiene que construir un archivo ejecutable, Exe, a partir de la información contenida en las librerías de la aplicación, archivos Pbl. Entonces el usuario ejecuta el archivo resultante Exe dentro del entorno Windows. En terminología de orientación a objetos crear un ejecutable para la aplicación Powerbuilder es "definir la aplicación".

El archivo Exe contiene una rutina de carga de arranque inicial ("bootstrap") que Windows puede ejecutar, empleándola también para interpretar la aplicación. La librería también contiene la versión compilada de todos los objetos referenciados por la aplicación.

Ejecución de la Aplicación

La ejecución de una aplicación comienza con la rutina de carga y arranque inicial contenida en el archivo Exe. Esta rutina hace el uso del "Powerbuilder Database Development and Deployment Kit" (DDDK), kit de distribución y desarrolllo de la base de datos de Powerbuilder: El DDDK contiene una serie de programas que permiten a Windows acceder a las versiones compiladas de los objetos contenidos en el archivo ejecutable.

Al distribuir una aplicación Powerbuilder, además del archivo ejecutable, el usuario necesita el DDDK. No es preciso todo el entorno de desarrollo de Powerbuilder, sino sólo el DDDK. También se necesitan, en el caso de existir, las librerías que contienen la aplicación.

Redefinición de la Aplicación

La definición de la aplicación y creación del ejecutable final tiene lugar desde el Application Painter.

Por defecto, el nombre del ejecutable es el de la aplicación. Aunque se pueda cambiar este nombre, deberá tener por extesión Exe.

En esta ventana se podrían especificar qué archivos de recursos o librerías son necesarias para crear el ejecutable. Haciendo clic en el botón OK y tras una breve pausa, Powerbuilder construye el archivo Exe.

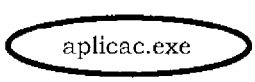
Ahora es posible ejecutar desde Windows el archivo.exe. Para distribuir la aplicación habría que copiar disquetes el ejecutable Exe y el DDDK de Powerbuilder.

Una aplicación Powerbuilder se compone de:
√ Ventanas
√ DataWindows
√ Controles
√ Objetos de usuarios
√ Iconos
√ Mapa de bits

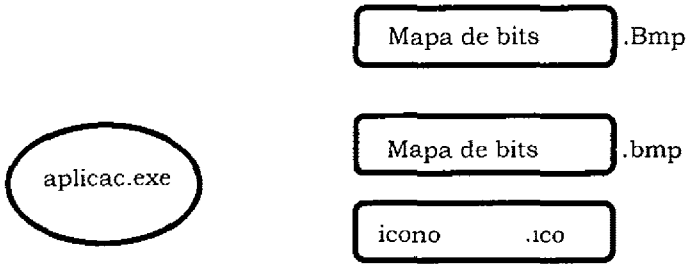
Muchas de estos objetos residen en librerías, archivos con extensión .pbl, creadas junto con la aplicación. No todos estos objetos se almacenan en librerías; por ejemplo, un mapa de bits se guarda un archivo DOS con la extención .bmp, o un icono en un archivo DOS con extensión .ico.

Existen cuatro formas de definir la aplicación, y por lo tanto, de distribuirla.

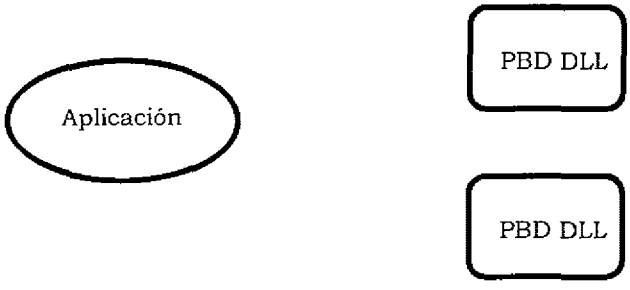
(1) Incluir todos los objetos, incluso los mapas de bits o iconos, en un único archivo ejecutable.

(2) 

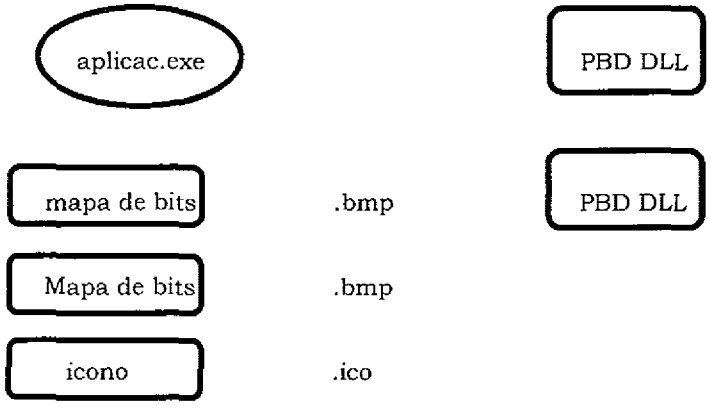
Incluir la mayor parte de los objetos en el archivo ejecutable u guardar en archivos Dos los mapas bits o iconos. Con ello se reduce el tamaño ejecutable, ya que los mapas bits son objetos gráficos y por tanto, posiblemente, de gran tamaño. Así se podría reducir drásticamente el tamaño del archivo .exe.



(3) Utlizar las Powerbuilder Dynamic Libraries (Librerías Dinámica de Powerbuilder), PBD. Una PBD puede contener alguno de los objetos de la aplicación. Así, parte de los objetos de Powerbuilder, junto a mapas de bits e iconos, pueden incluirse en el archivo .exe y el resto en una PBD.



Finalmente, se puede emplear una combinación de archivos PBD con archivos externos DOS para almacenar iconos y mapas de bits.



Opciones de Distribución

Si durante la construcción de la aplicación se han utilizado los mecanismos de herencia, existen dos posibilidades. Incluir el objeto ascendente, y todos los herederos del mismo, en el archivo .exe, o repartidos entre varios archivos .pbd.

En pocas palabras, no es posible separar el objeto ascendiente de sus descendientes entre el archivo .exe y los archivos pbd. O todos se incluyen en el archivo .exe, o todos se incluyen en archivos pbd.

Librerías Dinámica de Powerbuilder

Una librería dinámica es un archivo DOS que contiene objetos Powerbuilder compilados, y con extensión pbd. Durante una ejecución de una aplicación Powerbuilder accede a los objetos de la librería, los cuales se van cargando en memoria conforme los va necesitando la aplicación (de ahí el nombre de librería dinámica).

Se deberían usar varios archivos.pbd para reducir el tamaño de los archivo.exe, de tal forma que éstos no alcanzen tamaños superiores a 1.2Mb 1.5Mb; si así sucediese, han de emplarse archivos.pbd. Las librerías también deben ser pequeñas, conteniendo entre cincuenta y sesenta objetos y no superando los 80 Kb. De superar este tamaño, habría que dividir la aplicación en tantas librerías como fuese preciso.

Las librerías dinámicas permiten a varias aplicaciones compartir los mismos objetos. Objetos reutilizables por otras aplicaciones deberían organizarse en librerías separadas que hicieran más fácil de compartirlos.

La distribución de los componentes de la aplicación se ve también favorecida por el uso de librerías dinámicas. La distribución de nuevas versiones de la aplicación, o componentes modificados como consecuencia de la correción de errores detectados, sólo involucran a los módulos afectados, y no a toda la aplicación. Bastaría con eviar de nuevo la librería implicada, y no toda la aplicación.

Componentes de la Aplicación

Además de las librerías dinámicas, ciertos tipos de archivos pueden distribuirse con la ayuda de archivos de recursos. En Powerbuilder, un archivo de recursos es un archivo DOS con extensión pbr, conteniendo:

√ Mapas de bits (archivos rle o bmp)
√ Cursores (archivos cur)
√ Iconos (archivos ico)
√ Objetos DataWindow

Los archivos pbr son útiles en aquellos procedimientos que dinámicamente asignan recursos a atributos o controles de un objeto. Por ejemplo, una aplicación puede dinámicamente asignar un objeto DataWindow a un control:

```
dw_control.DataObject = "dw_obj_nombre"
```

O pueden asignar dinámicamente un mapa de bits a un botón con figura:

```
cb_figura.PictureName = "archivo.bmp"
```

O pueden asignar dinámicamente un icono a un DataWindow:

```
dw_control.icon = "nombre.ico"
```

Los archivos de recursos no son necesarios para definir la aplicación, aunque pueden reducir el número de archivos a distribuir; en lugar de distribuir el ejecutable exe y una serie de archivos pbr y exe. En cualquier caso, de no existir el archivo pbr, todos los objetos asignados dinámicamente han de existir en una librería dinámica.

Los archivos pbr se crean en cualquier editor de texto, como por ejemplo, el Bloc de Notas de Window. Los nombres de los objetos se incluyen de la siguiente forma:

```
emp3.bmp
chart1.bmp
```

```
chart2.bmp
dist1.pbl (d_orders)
chart3.bmp
dw2.ico
```

Sólo debe haber un objeto por línea y un único archivo pbr por aplicación.

Para incluir un objeto DataWindow, primero se indica la librería que lo contiene, y después el nombre del objeto cerrado entre paréntesis, como muestra la cuarta línea del ejemplo.

Si los objetos no se encuentran en el directorio actual, hay que indicar todo el cambio de acceso a ellos, incluida la unidad que contiene. Entonces, la referencia al objeto en la aplicación ha de contener también todo el camino o vía de acceso hasta él, de forma idéntica a como se indicó el archivo.pbr.

Como recomendación final, al definir la aplicación desde Application Painter, asegúrese de que la aplicación a distribuir es la actual, que se incluyen en todas las librerías dinámicas y que se ha creado previamente el archivo de recursos.

Conclusiones.

Powerbuilder es un lenguaje de 4ta. Generaciòn que simplifica el desarrollo de aplicaciones cliente-servidor. Con el lenguaje se puede hacer uso de herramientas muy poderosas y sencillas que agilizan el proceso de desarrollo de un sistema.

La herencia un concepto de Programación Orientada a Objetos, (explicada con anterioridad), crea un concepto en Powerbuilder muy util para desarrollar aplicaciones en un corto plazo este concepto es el de Estándares.

Los estandares son aplicaciones creadas sobre Powerbuilder con la cuales podemos heredar los objetos creados en las aplicaciones estandar y reutilizar la programación de los objetos para evitar volver a realizar este proceso. Otra ventaja importante al momento de desarrollar varias aplicaciones, es la de que todas las aplicaciones tienen la misma funcionalidad, lo que permite que los usuarios al momento de aprender a usar una aplicacion ya pueden utilizar otra aplicación diferente sin ningún problema.

El tiempo de desarrollo de aplicaciones Powerbuilder bajo el esquema de incluir estandares redunce en una 15 o 20% el tiempo de desarrollo, debido a sus objetos como los DataWindows y a su manejo de la herencia.

Un concepto que Powerbuilder creo que tiene una gran sencillez de uso además de tener rápidez al diseño fue el de las DataWindows, que realizan una conexión directa con la base con diferentes formatos y presentaciones que le dan un nuevo enfoque a la creación de ventanas y que simplifica el desarrollo de aplicaciones, preocupándose más por él ¿ Qué mejora dará? en vez de ¿ Cómo se hará ?.

Así como Powerbuilder cuenta con grandes ventajas obviamente también tiene sus desventajas, su lado obscuro. Uno de los principales problemas el excesivo consumo de recursos que requiere tanto en disco como en memoria RAM.

PowerBuilder es una herramienta recomendable para el desarrollo de aplicaciones que requieren ser desarrolladas en un corto periodo.

La siguiente tabla es una pequeña comparación entre tres lenguajes de programación.

Herramienta	Tiempo Desarrollo	Simplicidad en su Uso	Mantenimiento Aplicaciones	Acceso Datos	Costo
PowerBuilder 5.0	**E**	**E**	**E**	**E**	**E**
Visual Basic 4.0	**R**	**B**	**B**	**B**	**B**
Delphi 2.0	**B**	**E**	**B**	**E**	**R**

Excelente = E
Bueno = B
Regular = R

Bibliografía

"Lenguajes de Programación"
Tucker A.
Prentice Hall
Segunda Edición
1988

"Powerbuilder Desarrollo de Aplicaciones cliente Servidor"
Paul Mahler
Prentice Hall
Primera Edición
1996

"Turbo C/C++"
Herbert Schildt
McGraw-Hill
Primera Edición
1991

"Fundamentos de las Estructuras de datos Relacionales"
Rubén Abad, Miguel Angel Careaga
Ed. Grupo Noriega
1993

"Organización de la base de datos"
James Martin
Prentice Hall
1975

"Relational Theory Concepts and Application"
Kenmore S.Byath Waite
Academy Press
1991