Proyecto Final

Bases de Datos 2

Tema:

Integridad en Informix

Indice

Pág

Introducción

Nuestro presente trabajo investigativo tiene como finalidad desarrollar y aplicar los diferentes funcionamientos con que trabaja el Sistema Gestor de Base de Datos Informix con el propósito de verificar si cuenta con los procedimientos necesarios de Integridad para darle la estabilidad y consistencia adecuada a los datos almacenados en nuestra base de datos.

Durante la investigación se intenta desarrollar uno de los aspectos más importantes y necesarios como lo es la Integridad, de esto se pretende obtener las características más relevantes para un adecuado trabajo en lo que a base de datos se refiere.

La importancia de la Integridad se basa principalmente en asegurar la consistencia permanente de los datos. Un mal manejo de la Integridad puede terminar en la redundancia de la información que puede ser de vital importancia y traer grandes perdidas económicas.

Los puntos más destacados son la declaración de las Llaves Primarias y Llaves Foráneas de las distintas entidades, las cuales son indispensables para crear las relaciones entre las tablas.

Objetivos

Objetivo General:

- Manipular el Sistema Gestor de Bases de Datos Informix con el fin de evaluar la Integridad que éste le puede proporcionar a los Datos.

Objetivos Específicos:

- Definir los diferentes conceptos que conforman los procedimientos de Integridad.

- Investigar a fondo la estructura del Informix con respecto a Integridad.

- Probar el Sistema Administrador de Bases de Datos del Informix con la finalidad de poder implementar un modelo de datos para ejemplificar la utilización del Informix.

Alcances

- Comprensión sobre como Informix trabaja a nivel de Sistema Gestor de Base de Datos.

- Desarrollo y Manipulación de las diferentes reglas de Integridad que maneja Informix.

- Aplicación exitosa de un modelo de datos cuyo propósito fue el de comprobar la Integridad de los datos que se iban almacenando.

Limitaciones

- El Informix en Costa Rica no se consigue por ningún lado, ya que el costo es muy elevado, y ya está demasiado obsoleto.

- La adquisición del Informix tiene que ser por Internet, puesto que tomando en cuenta el tamaño de los archivos, la lentitud de la Internet en Costa Rica, y el costo elevado del acceso telefónico, nos fue demasiado difícil.

- El acceso a los demos del Informix en su página oficial www.informix.com es muy complicado porque además de que los links a veces no funcionan, hay que navegar por el sitio en ciertas horas específicas. Además, los mismos links no traen su descripción, por lo que al bajar un archivo tuvimos que adivinar que era cual archivo. Sin olvidar que casi todos los demos que encontramos son para Linux, exceptuando uno que otro para Windows 2000 Server.

- La instalación del Informix en Linux es demasiado tediosa, ya que se tiene que instalar a puro comando, sin dejar de lado que hay que saber mucho sobre Linux para poder crear los grupos, los usuarios, etc.

- La información del Informix en Internet actualmente es muy poca, y en Costa Rica la cantidad de libros sobre éste es demasiado limitada (por no decir inexistente).

- Por último, las personas que sepan sobre cómo instalar y manipular el Informix en Costa Rica es demasiado escasa.

Historia de Informix

Informix es uno de los más importantes líderes en tecnología de base de datos, cuyo objetivo es proveer a las más grandes corporaciones alrededor del mundo con herramientas de vanguardia y alto desempeño que les permitan obtener la mayor eficiencia y productividad en el manejo de información corporativa que les confieran importantes ventajas competitivas.

Desde 1980, año de su creación, la empresa ha sido un gran exponente en el área de bases de datos relacionales, la arquitectura OLP y lenguajes de cuarta generación, así como en el lanzamiento al mercado de tecnología de bases de datos relacionales con orientación a objetos, tanto para Unix como para Windows NT.

Actualmente, las soluciones de bases de datos de Informix, que se encuentran soportadas por los más importantes proveedores de hardware, desarrolladores de software e integradores de soluciones, incluyen

sistemas de alto desempeño para ambientes corporativos muy diversos, desde pequeños grupos de trabajos hasta las más complejas aplicaciones de procesamiento paralelo, entre ellas destacan:

- Aplicaciones DatawareHouse.
- Manejo Dinámico de Contenidos Web.
- Servidores de Bases de Datos.
- Herramientas para el Desarrollo de las Aplicaciones en Areas de Procesamiento de Transacciones en Línea.
- Sistemas Empresariales de Computación Distribuida.

Reseña Histórica:

En 1980, Informix Corporation fue fundada por Roger Sippl y Laura King.

En 1981, lanzaron al mercado su primer producto el C−ISAM, bajo el nombre de Marathon.

En 1982, lanzaron la colección de productos pre−SQL, los cuales fueron:

- Informer Query Language.
- Perform Form Manager.
- Ace Report Writer.
- All−Reaching.

En 1984, lanzaron la segunda generación de productos SQL para Informix. Los primeros lanzamientos generales comenzaron con Informix−SQL e Informix−ESQL/C versión 1.10 y la versión C−ISAM 2.10.

En 1986, la compañía lanzó seguidamente el Informix−4GL.

En 1988, Informix cambió su nombre a Informix Software Inc. Lanzando este mismo año los siguientes productos:

- Informix−4GL/RDS.
- Informix−4GL/ID.

Además, lanzaron WingZ para Macintosh, el cual fue el comienzo de las hojas de cálculo gráficas con la velocidad, flexibilidad y capacidades gráficas de alta calidad.

Para 1990, se introduce al mercado el Informix−OnLine. Y en 1993, el Informix−OnLine Dynamic Server.

En 1994, lanzaron la nueva era de lenguajes Informix, el GUI OOP.

En 1995, Informix compró el Ilustra, un objeto de base de datos relacionales, para la integración dentro del Informix−OnLine DSA.

En 1996, Informix lanzó el Universal Server, que es la combinación de el Informix−OnLine Dynamic Server y el Ilustra.

El 24 de abril de 1999, IBM e Informix anunciaron que han hecho un acuerdo definitivo a través del cual IBM adquirirá el negocio de base de datos de Informix, en una transacción por $1000 millones, la cual aún está sujeta a la aprobación final de los accionistas de Informix y revisión de las autoridades gubernamentales de Estados Unidos.

Productos de Informix

Algunos de los productos que Informix vende se pueden dividir en tres categorías:

Paquetes para Conectividad:

Estos son los softwares que proporcionan la conectividad entre su computadora y un sistema informático grande, para así tener toda la porción de los datos satisfactoriamente.

I−Net:

Este es el software que usted necesita en la máquina cliente para comunicarse con el servidor de la Base de Datos en una máquina servidor.

I−Star:

El software de trabajo en red que usted necesita sobre su máquina servidor. Proporciona capacidades para bases de datos distribuidas incluyendo uniones multisite y actualizaciones multisite con transparente recuperación de commit de dos fases.

Herramientas:

Estos son productos que dejan construir una interfaz de usuario para los datos sostenidos en la base de datos.

- Informix−4GL, Lenguaje Principal de Programación de Informix.
- Informix−4GL/GX, Corredor X−Windows GUI para Informix−4GL.
- Informix−4GL para Windows.
- Informix−DBA.
- Informix−SQL.
- Informix−Data Director para Java.
- Informix−ESQL para C, COBOL, FORTRAN y Ada.
- ViewPoint, Smartware, WingZ, NewEra.
- Formularios Informix−4GL.
- Menúes Informix−4GL.
- Herramientas HyperScript.

Motores de Bases de Datos:

Estos son los productos de la base de datos cruda capaces de entender los ordenes del SQL. Usted necesita éstos para agregar modificaciones y eliminar los archivos en una base de datos, pero ellos no le permitirán hacerlo en cualquier terminal I/O.

C−ISAM:

- Rutinas en C para crear y usar archivos secuenciales indexados.
- Primer producto de Informix.

Artefactos Estándar:

- Bajos requerimientos de administración.
- Conveniente para bases de datos de pequeñas a medianas.
- Corre sobre los Archivos del Sistema Cooked.

- Levanta hasta 8 columnas (120 bytes por índice).

Artefactos en Línea:

- Relativamente altos Requerimientos de Administración.
- Corre sobre los Archivos del Sistema RAW.
- Tiene la habilidad de hacer un respaldo mientras la base de datos está todavía corriendo.

- Tiene la habilidad de Bases de Datos Distribuidas (Informix−STAR).
- Levanta hasta 16 columnas (255 bytes por índice).

Introducción a la Integridad

Cuando alguna persona comienza a investigar Integridad siempre se pone a pensar preguntas claves para comprender con mayor facilidad este tópico. Entre las más importantes destacan: ¿Cómo guardar usuarios de acceso a items o datos específicos y cómo proteger el dato de daño o pérdida? ¿Qué ayuda se asegura que el dato sea correcto y exacto?. ¿Puede verificar una base de datos automáticamente que un artículo está en acción antes de una orden se asigne a ese item?

Todas estas preguntas tratan de la Integridad de los datos, o si los datos se guardaron en la base de datos es completa y legítima. Los CONSTRAINTS de la Integridad de los datos ayudan guardan el datos parece y completo. La funcionalidad de la Integridad de los CONSTRAINTS se construye sobre los servidores de bases de datos de Informix para controlar estas situaciones.

Los CONSTRAINTS con respecto a la Integridad de los datos pueden prevenir que estos lleguen a ser incorrectos pero no pueden garantizarlo con exactitud. Hay a pesar de eso suficiente espacio para el error humano. Los CONSTRAINTS pueden prevenir que se introduzca un dato fuera de los valores permitidos (por ejemplo, para el género de una persona, M o F), pero no pueden prevenir que una persona accidentalmente ingrese F para el caso de masculino, o viceversa.

Con respecto a las reglas del negocio, la Integridad de los CONSTRAINTS ayuda a guardar los datos en Línea con los valores como lo relatan el mundo real. Considerando la seguridad se puede prevenir acciones para usuarios específicos durante la ejecución de las funciones INSERT, UPDATE, DELETE y SELECT, los CONSTRAINTS observan los datos durante el INSERT, UPDATE O DELETE y no mira el usuario.

Los tres tipos de CONSTRAINTS de la Integridad que pueden ser dados obligatoriamente por todos los servidores de bases de datos de Informix son:

- CONSTRAINTS Semánticos: Estos tratan de cómo una columna maneja datos específicos ingresados y el valor que se guarda en esa columna.

- CONSTRAINTS de la Entidad: Son los que dan fuerza a filas dentro de una tabla manteniendo las llaves propias de la Llave Primaria.

- CONSTRAINTS de Integridad Referencial: Son aquellos que tratan de cómo las filas dentro de una tabla corresponden con otra fila dentro de otra tabla.

Debe usar una combinación de estos tres tipos de Integridad al servidor y aplicaciones del cliente con el fin de asegurarse la Integridad de los datos.

Integridad Semántica

La Integridad semántica es la más básica de todos los CONSTRAINTS. También conoce sobre el dominio del CONSTRAINT, la semántica trabaja directamente con los datos ingresados y su valor inicial. Los tres tipos de CONSTRAINTS semánticos usados por los servidores de bases de datos Informix son el tipo de dato, el valor por defecto y el CHECK del CONSTRAINT.

Tipos de Datos:

El tipo de dato se le asigna a la columna cuando se crea la tabla originalmente o en la alteración que se de más tarde. Por ejemplo una columna saldo puede declararse como INTEGER, SMALL INTEGER o FLOAT, ya que la columna es numérica. Si durante la ejecución de un INSERT o UPDATE se trata de ingresar un carácter inválido, Informix no permitirá que se ingrese.

Tipos de Datos en Informix

Tipo de Dato	Valor de los Datos
BYTE	Blob datos
CHAR o CHARACTER	Cordón del tamaño determinado
DATE	Configurable fecha esquemas
DATETIME	Configurable fecha y esquemas del tiempo
DEC**,** DECIMAL**,** o NUMERIC	Configuración de los números a una precisión específica
FLOAT o DOUBLE PRECISION	Números prefijaron a doble−precisión
INT or INTEGER	Números enteros de− 2,147,483,647 a 2,147,483,647
INTERVAL	Configurable cronómetro esquema del palmo
MONEY	Configurable esquema del divisado
NCHAR	Modo Mixto (cartas, números y símbolos), tamaño determinado
NVARCHAR	Modo Mixto (cartas, números y símbolos), tamaño variable
REAL o SMALLFLOAT	Solo−precisión cuenta
SERIAL	Datos Tasa
SMALLINT	Blob datos
TEXT	Cordón del tamaño determinado
VARCHAR o CHARACTERVARYING	Configurable fecha esquemas
VARYING	

Los tipos de Datos se asigna cuando se crea la tabla. La sintaxis SQL estándar CRATE TABLE es usada para asignar los tipos de datos desde Informix.

CREATE TABLE cliente

(

id_cliente NUMBER (5),

cliente VARCHAR (20),

calle VARCHAR (30,20),

ciudad VARCHAR (20),

last_update DATE,

saldo INTEGER (5,2),

total_orders INT

);

Para cambiar un tipo de dato, el SQL usa el ALTER TABLE. Durante un ALTER, Informix copia los datos fuera de la tabla, cambia los tipos de datos, intenta convertir el valor del dato, y devuelve los datos a la tabla. Se asegura que los tipos de datos no va causar un error, tal como cuando se cambia un INTEGER a CHAR. El tipo de dato que se modifica, debe ser capaz de manejar la base de datos restaurada, tal como cambia un SMALLFLOAT a un FLOAT o un CHAR a un VARCHAR.

Para alterar las columnas de una tabla existente, usa el MODIFY dentro de la declaración ALTER. Puede cambiar más de una columna dentro de paréntesis, separado por una coma:

ALTER TABLE customer MODIFY city VARCHAR(20,10);

ALTER TABLE customer MODIFY

(

city VARCHAR (20,10),

total_orders SMALLINT

);

Puede usar también la declaración ALTER para agregar columnas nuevas a una tabla que ya existe. En lugar de usar MODIFY, usa el ADD. Puede usar la sentencia BEFORE para especificar donde se ubica la columna nueva en la fila o registro. Si no se usa la sentencia BEFORE agrega la columna al final de la fila o registro.

ALTER TABLE customer ADD

phone CHAR (10) BEFORE last_update;

ALTER TABLE customer ADD

(

area_code CHAR (3),

line CHAR (7)

);

Valores por Defecto:

Los valores por defecto son otro tipo de CONSTRAINT Semántico. Si no se provee de ningún dato para una

columna específica durante el INSERT, se utiliza un valor predeterminado al instante. El dato usado por defecto puede ser una constante definida por un literal como 1 o Y. Se puede usar funciones por defecto para tipos de datos especiales, como la fecha actual para un campo fecha.

Tipos de Datos: Literales Estándar

Tipo de Dato	Literal	Ejemplo
INT, SMALLINT, DEC, MONEY	INTEGER	325, 185
FLOAT, SMALLFLOAT, DEC, MONEY	Decimal	3.25, 0.185
FLOAT, SMALLFLOAT, CHAR, NCHAR, NVCHAR, VARCHAR, DATE	Caracter	"C", "2−8−81"
INTERVAL	Interval	(2 11) DAY TO DAY
DATETIME	Date and time	96−04−19 11:30

Tipos de Datos: Funciones Estándares

Tipo de Dato	Función	Propósito
CHAR, NCHAR, NVARCHAR, VARCHAR	DBSERVERNAME o SITENAME	Provee el nombre del Servidor de la Base de Datos.
CHAR, VARCHAR	USER	Provee el user ID.
DATE	TODAY	Provee el formato para la fecha.
DATETIME	CURRENT	Provee la fecha y hora actual del sistema.

Use la sentencia SQL DEFAULT dentro de la sentencia CREATE TABLE y ALTER TABLE de la tabla, para asignarle un valor por defecto dentro de Informix. La sentencia ALTER usa el MODIFY, como se muestra en la sección de tipos de datos, para especificar un valor por defecto para una columna existente.

```
CREATE TABLE customer

(

customer_name CHAR (20) NOT NULL,

customer_id SERIAL,

street VARCHAR (30,20),

city CHAR (20),

state CHAR (10) DEFAULT "Puntarenas",

last_update DATE DEFAULT TODAY,

balance MONEY(5,2) DEFAULT 0,

total_orders INT DEFAULT 0

);
```

ALTER TABLE customers MODIFY

(

city DEFAULT "Puntarenas Centro",

);

INSERT INTO customer (customer_name) VALUES ("Shu Ching");

La inserción de un registro en la tabla customer sería de la siguiente forma (asumiendo la fecha del sistema):

Tabla Customer

Columna	Valor
customer_name	Shu Ching
customer_id	1
street	NULL
city	Puntarenas Centro
state	Puntarenas
last_update	06−04−01
balance	0.00
total_orders	0

Cuando no se especifica un valor por defecto y no se ingresa ningún valor, se pone un valor NULO en la columna. Usando la sentencia Not Null se obliga a que un valor debe ser ingresado en la columna.

CHECK CONSTRAINT:

Cuando los datos están dentro de un rango específico de valores (un subconjunto de un tipos de datos específico, tal como un entero entre 5 y 10 o un carácter igual a M o F) puede alcanzar la Integridad de los datos por rango de valores permitidos por el CONSTRAINT CHECK. El CONSTRAINT CHECK provee el tipo de datos específicos permitido dentro del rango de valores definido por el CHECK. Dentro de Informix el CONSTRAINT CHECK está disponible tanto a nivel de columna o tabla. Cualquier fila que se inserte o modifique debe ser evaluada por el CONSTRAINT CHECK antes de que se inserte en la tabla.

Column−Level CHECK CONSTRAINT:

Para utilizar un CONSTRAINT CHECK en una columna, se debe utilizar dentro de la función CREATE TABLE o ALTER TABLE con la sentencia MODIFY. El CHECK debe ser seguido por la condición. La condición no puede contener subqueries o funciones. El ejemplo siguiente usa CHECK:

CREATE TABLE customer

(

customer_name CHAR(20) NOT NULL,

customer_id SERIAL,

```
street VARCHAR(30,20),

city CHAR(20),

state CHAR(10) DEFAULT "Puntarenas"

CHECK (state IN ("Puntarenas","Limon","Guanacaste")),

last_update DATE DEFAULT TODAY,

balance MONEY(5,2) DEFAULT 0

CHECK (balance BETWEEN 0 and 999),

total_orders INT DEFAULT 0

CHECK (total_orders >= 0)

);
```

Cualquier valor insertado o actualizado dentro de estas columnas debe encontrar el criterio en CHECK. Usar un valor fuera del criterio, tal como una codificación del Estado de "Papá" causaría un error

Cuando se altera una tabla para agregar o cambiar un CONSTRAINT tipo CHECK, todos los datos actuales de la tabla debe pasar la condición nueva. Si los valores de los datos no se encuentran dentro del nuevo CONSTRAINT, este causaría un error en el ALTER.

```
ALTER TABLE customer MODIFY

total_orders CHECK (total_orders >= 1);
```

TABLE−Level CHECK CONSTRAINT:

Para agregar un registro completo dentro de la tabla este debe primero pasar por el CHECK. Con el CHECK a nivel de columna un INSERT completo para agregar un nuevo registro falla cuando falla el CHECK a nivel de columna. Con inserciones individuales por cada columna, sólo falla la inserción a causa del CHECK a nivel de columna no es suficiente; las otras columnas son pobladas por las otras inserciones.

Los CONSTRAINTS a nivel de tabla para un registro completo permiten revisar en todo momento si ha entrado un nuevo registro.

Los CHECKs a nivel de tabla permiten accesar todas las columnas de in registro, considerando que el CHECK a nivel de columna permite accesar solo la columna actual. Crear una tabla con CHECK a nivel de tabla, usa el comando CHECK en una línea independiente dentro del CREATE TABLE. Debido a que el CHECK es dueño de la línea, Informix sabe que no se asocia con una columna específica.

El ejemplo siguiente usa esta técnica:

```
CREATE TABLE customer

(
```

```
customer_name CHAR(20) NOT NULL,

customer_id SERIAL,

street VARCHAR(30,20),

city CHAR(20),

state CHAR(10) DEFAULT "Puntarenas"

CHECK (state IN ("Puntarenas","Limon","Guanacaste")),

last_update DATE DEFAULT TODAY,

cur_balance MONEY(5,2) DEFAULT 0

CHECK (cur_balance BETWEEN 0 and 999),

prev_balance MONEY(5,2) DEFAULT 0

CHECK (prev_balance BETWEEN 0 and 999),

last_payment MONEY(5,2) DEFAULT 0

CHECK (last_payment BETWEEN 0 and 999),

total_orders INT DEFAULT 0

CHECK (total_orders >= 0),

CHECK (prev_balance − last_payment = cur_balance)

);
```

En cualquier momento se puede insertar o cambiar un registro de la tabla customer, por ejemplo es importante verificar que el prev_balance menos el last_payment debe ser igual que al current_balance. Para implementar este CHECK la sintaxis correcta sería:

```
ALTER TABLE customer ADD CONSTRAINT

CHECK (prev_balance − last_payment = cur_balance);
```

Integridad de la Entidad

Una entidad es como un nombre en el idioma inglés; es una persona, lugar, o cosa. Una entidad es usualmente la columna principal usada para referenciar las otras columnas en ese registro. Debido a que esta columna es importante para encontrar el registro, se considera una Llave Primaria. El ser Llave Primaria, obliga a que los datos en esa columna sean únicos.

En el modelo de datos relacional el requerimiento para tener una Llave Primaria es que esta identifique cada registro de la tabla referenciando la Integridad de la entidad por el CONSTRAINT. Informix tiene en desarrollo un proceso que verifica que cada registro de la tabla tiene una Llave Primaria única.

Informix actualmente tiene dos maneras de asegurar que la columna que identifica un registro es única. La primera manera es usando la sentencia UNIQUE en el CREATE o ALTER TABLE.

Por ejemplo una tabla contiene información acerca de los consejeros de una universidad, utiliza el numero de asegurado para identificar a cada uno.

CREATE TABLE advisors

(

ssn CHAR (9) UNIQUE,

name CHAR (20)

);

Declarar la columna única después de creada la tabla, se utiliza la sentencia ALTER TABLE con el comando MODIFY. Recuerde que si se altera una columna para que esta sea única y en la tabla existen datos duplicados, esta sentencia va a fallar:

ALTER TABLE advisors MODIFY ssn UNIQUE;

La otra forma de asegurarse de que la entidad principal es única es especificándola como PRIMARY KEY. Las Llaves Primarias (PRIMARY KEYS) son usadas principalmente para forzar la Integridad referencial de una tabla, ya que estas obligan a ser únicas pues este es el requerimiento básico de toda PRIMARY KEY.

Para especificar un campo como Llave Primaria use la sintaxis PRIMARY KEY en el CREATE o ALTER TABLE de la siguiente forma:

CREATE TABLE advisors

(

ssn CHAR (9),

name CHAR (20),

PRIMARY KEY (ssn)

);

La sintaxis en el caso del ALTER TABLE es la siguiente:

ALTER TABLE advisors ADD CONSTRAINT PRIMARY KEY (name);

Cuando se requiere una Llave Primaria compuesta la sentencia UNIQUE no funciona. Se debe utilizar la sentencia PRIMARY KEY para combinar varios campos y así convertir esta combinación de campos únicos. Por ejemplo en una universidad se imparte el mismo curso en diferentes días y a diferentes horas por lo que la Llave Primaria sería el curso, el día y la hora para que este realmente sea único.

El ejemplo siguiente usa un compuesto importante:

```
CREATE TABLE classes

(

course_number INT (5),

daytaught CHAR,

timetaught DATETIME (HOUR),

teacher CHAR (9),

PRIMARY KEY (course_number, daytaught, timetaught)

);
```

Si por ejemplo el curso se imparte a la misma hora pero diferentes días la Llave Primaria debería contener únicamente el campo course_number y daytaugh. Para modificar la Llave Primaria se usaría el comando ALTER TABLE, con la sentencia ADD CONSTRAINT. La Llave Primaria existente automáticamente sería dropeada.

```
ALTER TABLE classes ADD CONSTRAINT PRIMARY KEY (course_number,daytaught);
```

En estos tres ejemplos de declaración de campos únicos, Llave Primaria y Llave Primaria compuesta, cualquiera INSERT o UPDATE que se haga a una entidad que ya exista fallará. No solo este tipo de CONSTRAINT provee a la base de datos un esquema de los requerimientos del modelo relaciona sino que provee un a forma rápida de indexación.

Integridad Referencial

La regla de Integridad usada para ligar una Llave Primaria con una Llave Foránea se llama Integridad Referencial. Este tipo de relación entre una Llave Primaria y una Llave Foránea es comúnmente llamada relación Padre−Hijo, donde el hijo es la Llave Foránea y el padre la Llave Primaria.

Relaciones Padre−Hijo:

La llave primaria en una columna es la única que representa la fila entera en una tabla. Por ejemplo un número del seguro social es una llave única, el cual significa que cada uno tiene uno diferente. Se requiere una Llave Primaria en el modelo de bases de datos relacional para cada tabla de la base de datos. Para el usuario de la base de datos la Llave Primaria llega a ser un índice. Con este índice es mucho más fácil hacer una búsqueda de un registro, sobre la tabla que sin tener el índice, sin un índice la búsqueda sería secuencial y sobre tablas muy grandes esta podría tomar mucho tiempo.

Cuando ninguna columna por si sola llega a identificar un registro de la tabla se puede hacer una combinación de campos con el fin de formar la Llave Primaria.

Una Llave Foránea es una columna, la cual usualmente esta relaciona con la Llave Primaria de otra tabla. Esta relación puede estar compuesta de múltiples campos siempre que la Llave Primaria que le esté referenciando sea compuesta.

La relación entre una Llave Primaria y una Llave Foránea es considerada una unión. Por ejemplo una tabla contiene los datos de los estudiantes en curso en particular. Esta tabla tiene su propia Llave Primaria

compuesta por los campos: semestres, curso, día y seguro social del estudiante; la tabla también contiene información extra sobre la nota final del estudiante pero esta información no es parte de la Llave Primaria.

Interacción entre Llaves Primarias y Llaves Foráneas:

La regla de la relación Padre−Hijo es que todo padre debe ser único y que todo hijo debe tener un padre. Informix tiene la capacidad de dar fuerza a esta regla y todas las situaciones que se levantan mientras trata de mantenerlo.

Las siguientes son algunas de estas situaciones:

- INSERTS: Para insertar un hijo debe de existir una Llave Primaria. No se puede crear una Llave Primaria si ésta realmente existe.

- UPDATES: Cuando se va a cambiar una Llave Foránea no se puede separar de la Llave Primaria existente, pero ésta puede cambiar a una Llave Primaria diferente. Si se cambia la llave ésta no se puede separar de sus hijos (Llaves Foráneas). Los hijos primero deben moverse a una Llave Primaria diferente.

- DELETES: Antes de eliminar una Llave Primaria se debe tomar en cuenta que se deben de eliminar primero sus hijos o también relacionarlos con otra Llave Primaria.

Aunque Informix obliga la relación Padre−Hijo, esta debe ser configurada por un tipo específico de relación Padre−Hijo.

Privilegios en el Nivel de Tablas y Columnas

Cuando un usuario tiene acceso a una base de datos, el DBA puede limitar el acceso a tablas específicas y columnas dentro de las tablas. El creador de la tabla o cualquier nivel de usuario Resource o nivel DBA puede crear tablas. Ese dueño o cualquier DBA puede conceder privilegios en el nivel de tablas a otros usuarios por esa tabla.

La sintaxis de la instrucción para otorgar privilegios a los usuarios es la siguiente:

GRANT privilegio−DB TO PUBLIC | usuario1, usuario2,...;

La sintaxis de la instrucción para retirar privilegios a los usuarios es la siguiente:

REVOKE privilegios−tabla ON nombre−tabla FROM PUBLIC | lista−usuarios;

Un total de ocho palabras proveen privilegios diferentes en el nivel de tablas, las cuales se definirán a continuación:

INSERT:

Conceder privilegios de inserción deja que usuarios agreguen datos nuevos a la tabla. Revocar ese privilegio detiene a usuarios de agregar datos a la tabla.

GRANT INSERT ON customer_table TO *user1*;

REVOKE INSERT ON customer_table FROM PUBLIC;

DELETE:

Conceder privilegios de borrado deja que usuarios quiten datos de una tabla. Revocar ese privilegio detiene usuarios de quitar datos de la tabla.

GRANT DELETE ON customer_table TO *user1*;

REVOKE DELETE ON customer_table FROM PUBLIC;

SELECT:

Los privilegios de seleccionar pueden conceder a los niveles de la tabla o niveles de columna específicas. Usuarios pueden tener la habilidad de escoger una fila entera en la tabla o sólo campos específicos. En el primer ejemplo, user1 puede mirar algunas columnas o cualquiera fila del customer_table. La concesión del segundo sólo permite escoger PUBLICO el customer_id, y columnas del customer_table. Se puede revocar privilegios en la misma manera.

GRANT SELECT ON customer_table TO *user1*;

GRANT SELECT (customer_id, balance) ON customer_table TO PUBLIC;

REVOKE SELECT ON customer_table FROM *user3*;

REVOKE SELECT (customer_id, balance) ON customer_table FROM *user4*;

UPDATE:

Puede conceder privilegios de update en el nivel de tablas o nivel de columnas específicas. Usuarios pueden tener la habilidad de cambiar una fila entera en la tabla o sólo campos específicos. En el primer ejemplo user1 puede actualizar cualquier columna o cualquier fila del customer_table. La concesión del segundo permite de manera PUBLICO actualizar sólo el customer_id y columnas customer_table. Se puede revocar privilegios en la misma manera.

GRANT UPDATE ON customer_table TO *user1*;

GRANT UPDATE (customer_id, balance) ON customer_table TO PUBLIC;

REVOKE UPDATE ON customer_table FROM *user3*;

REVOKE UPDATE (customer_id, balance) ON customer_table FROM *user4*;

REFERENCES:

Puede concederles la habilidad a usuarios forzar referential constraint en la fila entera o columnas específicas de una tabla. El usuario debe ser un recurso del nivel de la base de datos ante los privilegios de referencias. El Referenciar constraints hacen que se ejecuten tareas tales como el borrar de forma de cascada o cualquiera otra tarea como el de relacionar columnas con otras columnas.

GRANT REFERENCES ON customer_table TO *user1*;

GRANT REFERENCES (customer_id, balance) ON customer_table TO PUBLIC;

REVOKE REFERENCES ON customer_table FROM *user3*;

REVOKE REFERENCES (customer_id, balance) ON customer_table FROM *user4*;

INDEX:

El privilegio de indexar concede la habilidad a los usuarios de crear y borrar índices relacionados a una tabla. Los usuarios deben tener el privilegio de Resource en combinación con el privilegio de indexar. Los usuarios con el privilegio de connect no puede crear un índice, aun cuando tienen el privilegio de indexar. No hay privilegios de niveles de columnas porque los índices se construyen índices en todas las filas de la tabla.

GRANT INDEX ON customer_table TO *user1*;

REVOKE INDEX ON customer_table FROM *user3*;

ALTER:

El privilegio de alterar permite que usuarios cambien el esquema de las columnas dentro de la tabla. Usuarios con el privilegio de alterar puede agregar, borrar, y cambiar columnas y los tipos de datos de la columna. Solo usuarios con conocimiento del sistema de la base de datos y como protegerlo deben tener este privilegio. Este privilegio es casi como el nivel más alto como el del DBA. Alterar aplicaciones sólo en el nivel de la tabla.

GRANT ALTER ON customer_table TO *user1*;

REVOKE ALTER ON customer_table FROM *user3*;

ALL:

La palabra todo (all) provee todos los privilegios de tablas y columnas a usuarios. Usar la palabra todo (all) concede o revoca privilegios de cualquier tabla que el usuario puede tener.

GRANT ALL ON customer_table TO *user1*;

REVOKE ALL ON customer_table FROM *user2*;

Combinaciones:

Puede conceder o revocar combinaciones diferentes privilegios de tablas y columnas en un comando. Para colocar privilegios en cualquier secuencia, debe ser separado por una coma, después de la concesión o revoque de la palabra.

GRANT INSERT, DELETE, UPDATE ON customer_table TO PUBLIC;

GRANT SELECT, UPDATE (customer_id, balance) ON customer_table TO *user2*;

REVOKE INDEX, ALTER ON customer_table FROM *user1*;

Puede combinar también privilegios de niveles de tablas y columnas en una declaración. Los privilegios de niveles de columnas usan al especificación de columnas, y los privilegios de niveles de tablas usan la especificación de tablas.

GRANT INSERT, DELETE, SELECT, UPDATE (customer_id, balance)

ON customer_table TO *user2*;

REVOKE INDEX, SELECT, ALTER (customer_id, balance) ON customer_table FROM *user3*;

Otras Cláusulas:

Puede usar dos de otras palabras en conjunto con el comando GRANT. El primero es el con la palabra GRANT OPTION. Cuando combine el comando GRANT, el usar la recepción de privilegios puede también conceder los mismos privilegios a otro usuarios.

En el ejemplo siguiente user1 no sólo tiene privilegios de inserción, borrado, selección y actualización en el customer_table, pero él o ella pueden también conceder algunos o todos esos privilegios de otros

GRANT INSERT, DELETE, SELECT, UPDATE ON customer_table TO *user1*

WITH GRANT OPTION;

Si user1 tiene uno o todos de los privilegios de revoque, todos los usuarios que user1 concedió privilegios tendrán también los mismos privilegios que revocó.

La otra palabra usada como concesión es la palabra AS. La palabra AS permite que se ejecute una concesión como si otro usuario ejecuta la concesión. El establecer la situación descrita arriba previamente; si el conceder es revocar, todos los usuarios concedidos por ese usuario se revoca también.

Continuar con el presente ejemplo, user1 se da los privilegios de inserción, borrado, selección, y actualización en el customer_table y el derecho conceder estos privilegios. Un DBA, el dueño de la tabla , o el usuario que le concedió los privilegios a user1 podría conceder entonces como user1 a otros usuarios:

GRANT INSERT, DELETE, SELECT, UPDATE

ON customer_table TO *user2*, *user3*, *user4*, *user5* AS *user1*;

Ahora user1 a través de user5 tiene algunos privilegios. También el revocar los privilegios sobre todos los cinco usuarios, solamente revoca el user1:

REVOKE ALL ON customer_table FROM *user1*;

Restricciones de Integridad

En una restricción normal es posible distinguir los siguientes componentes:

- La Operación de Actualización: Cuya ejecución ha de dar lugar a la comprobación del cumplimiento de la restricción.
- La Condición: La cual debe cumplirse, y es en general, una proposición lógica que se define sobre uno o varios elementos del esquema, que puede tomar uno de los valores de verdad (verdadero o falso).
- La Acción: Aquella que debe llevarse a cabo dependiendo del resultado de evaluar la condición.

Procedimientos Almacenados:

Los procedimientos almacenados se consideran separados de las entidades de la base de datos, y porque están separados, usuarios deben tener los privilegios apropiados para crear, editar y procesar los programas. Estos procedimientos pueden tener acceso a áreas específicas de la base de datos que los usuarios no son capacez de ver. De cualquier modo que estos mismos usuarios pueden tener la habilidad de correr un procedimiento almacenado, ejecutan funciones específicas en áreas restringidas. Por eso, los procedimientos almacenados habilitan las áreas restringidas de usuarios, pero no permite que tengan acceso lleno de correr.

Por ejemplo, una tabla contiene el sueldo de todos los empleados e información del pago extraordinaria. Un procedimiento almacenado es ejecutado cuando un usuario inserta información acerca de una venta que ganó o comisión. El procedimiento almacenado verifica si es una comisión válida, y entonces agrega esa cantidad al sueldo de la persona apropiada. El usuario no tiene acceso a la tabla que contiene la información del sueldo, y si él o ella intentan agregar la comisión a la tabla sin usar el procedimiento almacenado, o ejecuta cualquiera otra actividad en la tabla fallaría.

Restricciones de Disparo (Triggers):

Los Disparadores son instrumentos de las bases de datos activas que permiten definir reglas distintas de las restricciones; en realidad, las restricciones, como ya hemos indicado, no son otra cosa que un tipo especial de reglas de las bases de datos activas en las que el evento que las activa es una actualización.

En la Restricción de Disparo, se formula una condición de forma declarativa, mediante una proposición lógica; el cumplimiento de las misma dispara una acción especificada de forma procedimental, esto quiere decir que que al contrario de lo que pasa en otros tipos de restricciones, la acción se desencadena ante un resultado de verdadero en la condición. Si no se especificara la condición, ya que es opcional, se considera el resultado como verdadero y la acción se dispara siempre que se ejecute la operación.

Para combinar disparadores con procedimientos almacenados, se puede construír una biblioteca de procesos que maneje la seguridad de los datos y la auditoría.

Restricciones de Verificación:

Son las cláusulas CHECK de la mayoría de lenguajes. La expresión lógica mediante la cual se formula la condición está definida sobre uno a varios atributos de un mismo elemento. Este tipo de restricción se declara al tiempo que se define el elemento del esquema al cual afecta. Se les puede dar un nombre, pero al no tener existencia por sí mismas sino dentro del elemento al que afectan, el nombre no es obligatorio.

Restricciones de Aserción:

Son análogas a las Restricciones de Verificación, aunque se diferencian de ellas en que pueden estar referidas a más de un elemento del esquema (varias tablas), ya que tienen existencia por sí mismas; por tal motivo, es obligatorio ponerles un nombre.

Vistas

Una Vista se puede definir de distintas maneras, entre las conceptualizaciones más específicas y acertadas están:

- Tabla Virtual que no tiene existencia física como una tabla base aunque es percibida y tratada como si así fuera. Constituye una ventana sobre los datos de una o varias tablas, y posee una definición análoga al de una tabla, y está almacenada en el diccionario de datos pero no tiene un archivo físico que soporte los datos.

- Representación Lógica de columnas físicas de una o múltiples tablas. Una vista parece y actúa como una tabla, pero verdaderamente no es una tabla física que reside en disco.

- Gran manera de presentar información específica a usuarios específicos, sin poner los datos de un tabla entera en el abrir o guardar versiones múltiples de datos por grupos diferentes de usuarios.

Las vistas proveen restricciones de columnas seguras de usuarios. Una vista puede mostrar y dejar acceso a esas pocas columnas. Este método hace las tablas parece menos aplastante a usuarios quienes no entienden totalmente todas las columnas de los datos.

Algunos datos contenidos en las tablas podrían ser sensibles a usuarios específicos por razones legales o morales. Puede permitir usar vistas para que usuarios accedan a algunos datos contenidos en tablas sensibles mientras se restringe acceso a otras columnas. Por ejemplo una tabla de empleados puede contener información de la dirección de cada empleado, número de teléfono, quien notificar en caso de una emergencia, y sueldo. Obviamente, los empleados no deben tener acceso a otros sueldos de empleados, pero el contacto de la información podría ser importante.

Las vistas pueden representar derivados valores de datos. Pero usando las columnas de una o más tablas el lugar del dato en la vista puede derivar valores para usar agregados, sustracciones o algunas otras funciones matemáticas, en SQL se puede representar en una vista con una columna virtual. Por ejemplo, se puede crear una exhibición de la vista de los artículos, cantidad, y valor de ordenes por la Pizza de Shu al cliente, donde el valor es igual a la cantidad multiplicado contra el precio.

Creación de una Vista:

Al crear una vista, el usuario intenta el crear más de las que tiene, selecciona por lo menos privilegios en todas las tablas con columnas que se representan en la vista. Las dos partes a la declaración de la creación de la vista son vista apellida y selección de la columna.

El nombre de una vista debe ser un nombre único de máximo a los 18 caracteres. Este nombre es usado para acceder a la vista después de que ha sido creada:

CREATE VIEW *view_name* AS

Para asignarle columnas a una vista, se usa el estándar de declaración SQL SELECT. Las cláusulas ORDER BY y UNION no se permiten en el SELECT.

Los valores del tipo de dato se heredan automáticamente de las columnas originales a las columnas de la vista nuevas. Se pueden heredar los nombres de las columnas también a la vista a menos que se especifique como algo diferente. Cuando se crea una columna virtual, debe especificarse el nombre de la columna nueva. Cuando el nombre de una columna se especifica, la declaración del CREATE VIEW requiere todas los nombre de columnas especificadas, indiferente de si los nombres son diferentes:

CREATE VIEW *view_name* (*column list*) AS

SELECT *columns* FROM *TABLES;*

El primer ejemplo establecido arriba de una vista estándar por las direcciones de todo empleados quien se emplean corrientemente jornada completa. No son renombradas las columnas, así ninguna de las columna en la lista es requerida.

CREATE VIEW empl_address AS

SELECT name, street, city, zip

FROM employee_info

WHERE current = Y AND work_status = F;

La próxima creación establecida de una vista por la clasificación de todos los clientes y actividad del pago. Se crea una columna virtual también. Debe listar todas las columnas porque la columna virtual requiere un nombre. Esta vista también junta dos tablas para recuperar la información:

CREATE VIEW customer_bal (cust_id, last_order, last_payment, current_balance) AS

SELECT cust_id, total_order, total_payment, total_order, total_payment

FROM order_TABLE, payment_TABLE

WHERE order_TABLE.cust_id = payment_TABLE.cust_id;

Puede usar otras vistas de columnas para construir otras vistas. El próximo creación de ejemplo determina el total del balance pendiente a pagar a la compañía de la vista del customer_view. Puede usar también agregaciones de comando en SQL para crear columnas virtuales. Agregue comando incluyendo: SUM, MIN, MAX, AVG, y COUNT. El comando SUM es usado para agregar todos los balances juntamente.

CREATE VIEW total_balance (balance) AS

SELECT SUM(current_balance)

FROM customer_view;

El próximo ejemplo establecido arriba crea una vista en todas las columnas de los datos relacionados a las filas específicas. Todas las ventas hechas por el vendedor 12, Christian, se lista en la vista siguiente:

CREATE VIEW shu_sales AS

SELECT *

FROM sales

WHERE sales_person = 12;

La Creación de Vistas se puede resumir de la siguiente manera:

CREATE VIEW Nombre−Lista [(Lista−de−Columnas)]

AS

SELECT Lista−de−Selección

[WITH CHECK OPCION]

- La Lista de Columnas: es opcional, pero se hace obligatoria en el caso de presencia de ambigüedad de columnas. En caso contrario la vista está compuesta de las columnas seleccionadas en la cláusula

SELECT.

- La sentencia SELECT no puede contener la cláusula ORDER BY ni el operador UNION.
- La opción WITH CHECK OPTION permite mantener la Integridad referencial de los datos.

Accesando y Manipulando Vistas:

El creado de una vista es considerado el de una vista; los dueños usuarios y el nivel del DBA pueden conceder y revocan acceso a la vista a otros usuarios. Pueden restringir acceso de una tabla entera, pero usuarios obtienen acceso a los datos de la tabla por una vista. Este fuerza a los usuarios a usar la vista para acceder los datos.

La restricción de usuarios normales de accesar la tabla del empleado entera, pero todavía permite que accedan sus direcciones, usando el ejemplo siguiente:

REVOKE ALL ON employee_info;

CREATE VIEW empl_address AS

SELECT name, street, city, zip

FROM employee_info;

GRANT SELECT, UPDATE ON empl_address TO PUBLIC;

Trabajar con una vista es sólo como accesar a una tabla. Use el nombre de la vista en lugar del nombre de la tabla en todos los comandos de SQL. Algunas restricciones relacionadas a vistas no se hallan como tablas individuales. Primero, se pueden crear ningunos índices en una vista. Algunas relaciones de tablas indexadas usan el acceso de índices datos por una vista. Si una tabla o vista es borrada , algunas vistas de estas usan la tabla o columna para borrarlas también.

Las vistas contenidas junta o agregadas pueden ser accesadas sólo con declaraciones SELECT porque una junta o agregado de vista toma a los mismos datos diferentes de lugares diferentes y hechuras que se parece a está todo de un lugar. Informix no puede determinar como un cambio de datos en una vista inverso relata a las tablas originales. Casi la misma situación aplica a las columnas virtuales; porque se derivan las columnas virtuales de fuentes de los datos múltiples, está hace imposible insertar o actualizar el valor en una vista. Es posible borrar la fila de una vista que contiene una columna virtual porque Informix puede rastrear inverso a la columna original y llaves.

CHECK OPTION:

Como se mencionó en la discusión previa en creación de vistas, se puede crear una vista con información relacionado a un dato específico o fila. Como en el ejemplo del shu_sales, una vista puede contener un subconjunto de una tabla o datos de tablas.

El ejemplo del shu_sales muestra la codificación siguiente:

CREATE VIEW shu_sales (sales_person, customer, sub_length, price) AS

SELECT *

FROM sales

WHERE sales_person = 12;

Si esta vista es disponible por Christian, puede querer insertar sus ventas nuevas directamente por esta vista en lugar de usar la tabla de las ventas entera. Si Christian vende subscripciones del periódico y le hace una subscripción del mes a Luis Segura a ¢500.25, se pone en la tabla de las ventas por la vista del shu_sales esta información:

INSERT INTO shu_sales

VALUES (12, "Luis Segura", 1, 500.25);

Si Christian hace una equivocación y usa el número del sales_person erróneo, se da el crédito a su venta a algún otro:

INSERT INTO shu_sales

VALUES (11, "Luis Segura", 1, 500.25);

Aunque se use shu_sales, la inserción por sales_person 11, Fabian, tiene éxito inverso a la tabla de las ventas. Christian puede verificar su vista:

SELECT * FROM shu_sales;

La entrada por 11 no se muestra arriba porque se limita a sales_person 12. Fabian puede ver la entrada de sus vista propias:

SELECT * FROM fabians_sales;

Usuarios con acceso directo a la tabla de las ventas pueden ver también la entrada:

SELECT * FROM sales;

Para prevenir este problema, use la palabra WITH CHECK OPTION cuando se crea la vista. El WITH CHECK OPTION permite inserciones, actualizaciones, y borrados ocurridos sólo cuando se selecciona la vista creada.

CREATE VIEW shu_sales (sales_person, customer, sub_length, price) AS

SELECT *

FROM sales

WHERE sales_person = 12

WITH CHECK OPTION;

Cuando Christian trata de insertar sus ventas con el número malo del sales_person, recibe un mensaje del error.

Eliminación de una Vista:

Un dueño de la vista o un DBA puede borrar una vista existente. Cuando se borra una vista, no se pierden los datos, las columnas, o las tablas; únicamente se va sólo la vista de esos datos. Los datos todavía quedan residentes en las tablas subyacentes y columnas de la base de datos. En cambio, si las tablas reales y columnas son borradas, cualquier vista usada por estas tablas y columnas son borradas automáticamente. En la vista shu_sales, si Christian no tuviera ventas en la tabla de las ventas, la vista continuará existiendo, pero contiene cero filas.

El borrado de una vista usa el comando DROP VIEW:

DROP VIEW *view_name*;

El siguiente ejemplo utiliza DROP VIEW**:**

DROP VIEW shu_sales;

Modificación de una Vista:

No se puede usar un ALTER para cambiar el esquema de una vista. Se necesita cambiar un esquema de la vista, además borrar y recrear la vista del nuevo esquema. Verificar las esquemas de la vista presentes, se usa el sysviews y sysdepends de las tablas del system. El sysviews de la actual tabla system contiene actualmente la declaración original CREATE VIEW. Para ver todas las vistas corrientemente en el servidor de la base de datos, use el siguiente SELECT:

SELECT * FROM sysviews;

El sysdepends de la tabla system contiene información de cada vista y las tablas o de otras vistas que proveen los datos hechos en las vistas originales.

Ver todo las dependencias cada vista contenida, se usa el siguiente SELECT:

SELECT * FROM sysdepends;

Cuando se borra un vista, esta información no reside en el sysviews o sysdepends de tablas. Es una idea buena para preservar una copia de los procedimiento preguntados como un backup de todo las vistas usadas como una referencia cuando se crea o recrea una vista.

Descripción del Modelo

El siguiente sistema de base de datos, trata de una Compañía de Excursiones Turísticas en todo el territorio costarricense.

La compañía ofrece las excursiones por una determinada cantidad de días (en esta compañía son de dos noches, tres días), tanto para turistas nacionales como para turistas extranjeros.

La compañía contrata varios choferes y guías muy capacitados para que se encarguen de las excursiones, y así poder atender satisfactoriamente a los excursionistas.

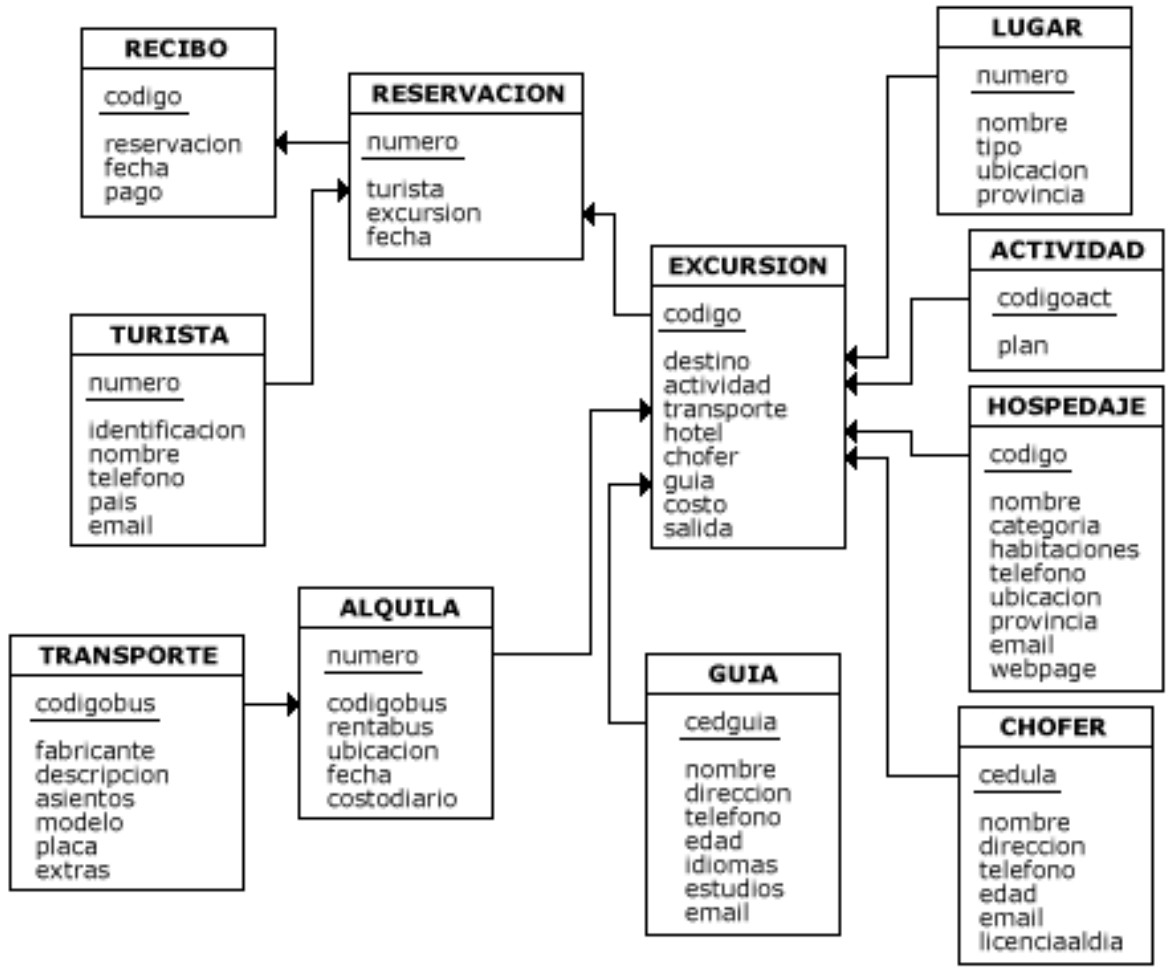
Las actividades en las excursiones no tienen un tiempo límite, sino que se efectúan cuando el guía lo considere debido.

El transporte se alquila dependiendo a los días de las excursiones. Un bus no será alquilado si no hay ninguna excursión a la vista.

Los turistas podrán reservar sus campos en las excursiones, pero se les entregará su respectivo recibo de comprobación hasta que hayan pagado el monto correspondiente.

Modelo Físico

La siguiente imagen representa la estructura con sus respectivas relaciones del sistema de la Compañía de Excursiones Turísticas Nacionales.



Diccionario de Datos

Tabla CHOFER: Es la tabla donde se almacena cierta información sobre los choferes que conducirán el transporte de las excursiones.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Cedula	Caracter	15	Número de Cédula del Chofer	Sí, Llave Primaria	No
Nombre	Caracter	35	Nombre del Chofer	No	No
Direccion	Caracter	80	Dirección del Chofer	No	No
Telefono	Caracter	14	Número Telefónico del Chofer	No	No
Edad	Numérico	2	Edad del Chofer	No	Sí
Email	Caracter	50	Correo Electrónico del Chofer	No	Sí
Licenciaaldia	Caracter	1	Si el Chofer tiene la Licencia al día	No	No

Tabla HOSPEDAJE: Tabla que contiene información sobre los hoteles en donde se hospedarán los excursionistas.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Codigo	Caracter	5	Código del Hotel	Sí, Llave Primaria	No
Nombre	Caracter	35	Nombre del Hotel	No	No
Categoria	Numérico	1	Categoría del Hotel	No	No
Habitaciones	Numérico	4	Cantidad de Habitaciones en el Hotel	No	Sí
Telefono	Caracter	14	Número Telefónico del Hotel	No	No
Ubicacion	Caracter	50	Ubicación del Hotel	No	No
Provincia	Caracter	10	Provincia donde está ubicado el Hotel	No	No
Email	Caracter	50	Correo Electrónico del Hotel	No	Sí
Webpage	Caracter	40	Página en Internet del Hotel	No	Sí

Tabla LUGAR: En la tabla LUGAR se guardan los datos de los lugares a donde van a ir las excursiones.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Numero	Numérico	3	Número del Lugar	Sí, Llave Primaria	No
Nombre	Caracter	50	Nombre del Lugar	No	No
Tipo	Caracter	6	Tipo de Lugar	No	No
Ubicación	Caracter	50	Ubicación del Lugar	No	No
Provincia	Caracter	10	Provincia donde está ubicado el Lugar	No	No

Tabla GUIA: Aquí se almacenan los datos sobre los guías de las excursiones.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Cedguia	Caracter	15	Número de Cédula del Guía	Sí, Llave Primaria	No
Nombre	Caracter	35	Nombre del Guía	No	No
Direccion	Caracter	80	Dirección del Guía	No	No
Telefono	Caracter	14	Número Telefónico del Guía	No	No
Edad	Numérico	2	Edad del Guía	No	Sí
Idiomas	Caracter	40	Idiomas que habla el Guía	No	No
Estudios	Caracter	80	Instituciones donde obtuvo sus Títulos	No	Sí
Email	Caracter	50	Correo Electrónico del Guía	No	Sí

Tabla ACTIVIDAD: En esta tabla se guarda la información de las actividades que se van a efectuar durante las excursiones.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Codigoact	Caracter	5	Código de la Actividad	Sí, Llave Primaria	No
Plan	Caracter	80	Todas las Actividades de la Excursión	No	No

Tabla TRANSPORTE: Aquí se almacena la información de los buses.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Codigobus	Caracter	5	Código del Bus	Sí, Llave Primaria	No

Fabricante	Caracter	55	Marca del Fabricante	No	Sí
Descripcion	Caracter	60	Descripción del Bus	No	No
Asientos	Numérico	3	Cantidad de Asientos que tiene el Bus	No	No
Modelo	Numérico	4	Año de Fabricación del Bus	No	Sí
Placa	Caracter	14	Número de Placa del Bus	No	No
Extras	Caracter	70	Lujos Adicionales	No	No

Tabla ALQUILA: En esta tabla se guardan los datos sobre los alquileres de los buses.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Numero	Numérico	5	Número de Alquiler	Sí, Llave Primaria	No
Codigobus	Caracter	5	Código del Bus Alquilado	Sí, Llave Foránea	No
Rentabus	Caracter	35	Nombre del Renta Bus	No	No
Ubicación	Caracter	50	Ubicación del Renta Bus	No	No
Fecha	Fecha	9	Fecha en que se efectuó el Alquiler	No	No
Costodiario	Numérico	6	Costo Diario del Alquiler	No	No

Tabla TURISTA: La tabla TURISTA es la encargada de almacenar la información de todos los turistas que se inscriban en las excursiones.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Numero	Numérico	5	Número de Turista	Sí, Llave Primaria	No
Identificacion	Caracter	15	Identificación del Turista	No	No
Nombre	Caracter	35	Nombre del Turista	No	No
Telefono	Caracter	14	Número Telefónico del Turista	No	Sí
Pais	Caracter	25	País de Origen del Turista	No	No
Email	Caracter	50	Correo Electrónico del Turista	No	Sí

Tabla RESERVACION: En la tabla RESERVACION se almacenará la información de las reservaciones de las excursiones.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Numero	Numérico	9	Número de Reservación	Sí, Llave Primaria	No
Turista	Numérico	5	Número de Turista	Sí, Llave Foránea	No
Excursion	Caracter	5	Código de la Excursión	Sí, Llave Foránea	No
Fecha	Fecha	9	Fecha de la Reservación	No	No

Tabla EXCURSION: En esta tabla se almacenará toda la información acerca de las excursiones de la Compañía.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Codigo	Caracter	5	Código de la Excursión	Sí, Llave Primaria	No
Destino	Numérico	3	Número del Destino de la Excursión	Sí, Llave Foránea	No
Actividad	Carácter	5	Código de la Actividad en la Excursión	Sí, Llave Foránea	No
Transporte	Numérico	5	Número de Alquiler del Bus	Sí, Llave Foránea	No
Hotel	Caracter	5	Código del Hotel	Sí, Llave Foránea	No

Chofer	Caracter	15	Número de Cédula del Chofer	Sí, Llave Foránea	No
Guia	Caracter	15	Número de Cédula del Guía	Sí, Llave Foránea	No
Costo	Numérico	6	Costo de la Excursión	No	No
Salida	Caracter	80	Dirección de Salida de la Excursión	No	No

Tabla RECIBO: En esta tabla se guardarán los datos de los recibos o comprobantes que se le entregarán a los turistas cuando ya hallan cancelado el monto correspondiente a la excursión reservada.

Atributo	Tipo	Tamaño	Descripción	Unicidad	Nulo
Codigo	Caracter	5	Código del Recibo Efectuado	Sí, Llave Primaria	No
Reservacion	Numérico	9	Número de la Reservación	Sí, Llave Foránea	No
Fecha	Fecha	9	Fecha en que se Imprimió el Recibo	No	No
Pago	Numérico	6	Monto del Pago realizado	No	No

Scripts

Tablas:

```
CREATE TABLE TRANSPORTE

(

codigobus varchar2(5) CONSTRAINT buspk_codigobus PRIMARY KEY,

fabricante varchar2(55),

descripcion varchar2(60) CONSTRAINT busnn_descripcion NOT NULL,

asientos number(3) CONSTRAINT busnn_asientos NOT NULL,

modelo number(4),

placa varchar2(14) CONSTRAINT busnn_placa NOT NULL,

extras varchar2(70) CONSTRAINT busnn_extras NOT NULL

);

CREATE TABLE HOSPEDAJE

(

codigo varchar2(5) CONSTRAINT hospk_codigo PRIMARY KEY,

nombre varchar2(35) CONSTRAINT hosnn_nombre NOT NULL,

categoria number(1) CONSTRAINT hosnn_categoria CHECK (categoria between 0 and 6),

habitaciones number(4),
```

```
telefono varchar2(14) CONSTRAINT hosnn_telefono NOT NULL,
ubicacion varchar2(50) CONSTRAINT hosnn_ubicacion NOT NULL,
provincia varchar2(10) CONSTRAINT hosnn_provincia NOT NULL,
email varchar2(50),
webpage varchar2(40)
);
CREATE TABLE GUIA
(
cedguia varchar2(15) CONSTRAINT guiapk_cedguia PRIMARY KEY,
nombre varchar2(35) CONSTRAINT guiann_nombre NOT NULL,
direccion varchar2(80) CONSTRAINT guiann_direccion NOT NULL,
telefono varchar2(14) CONSTRAINT guiann_telefono NOT NULL,
edad number(2),
idiomas varchar2(40) CONSTRAINT guiann_idiomas NOT NULL,
estudios varchar2(80),
email varchar2(50)
);
CREATE TABLE CHOFER
(
cedula varchar2(15) CONSTRAINT chopk_cedula PRIMARY KEY,
nombre varchar2(35) CONSTRAINT chonn_nombre NOT NULL,
direccion varchar2(80) CONSTRAINT chonn_direccion NOT NULL,
telefono varchar2(14) CONSTRAINT chonn_telefono NOT NULL,
edad number(2),
email varchar2(50),
licenciaaldia char(1) CONSTRAINT chock_licenciaaldia CHECK (licenciaaldia IN (`S', `N'))
```

```
);
CREATE TABLE ACTIVIDAD
(
codigoact varchar2(5) CONSTRAINT actpk_codigoact PRIMARY KEY,
plan varchar2(80) CONSTRAINT actnn_plan NOT NULL
);
CREATE TABLE LUGAR
(
numero number(3) CONSTRAINT lugarpk_numero PRIMARY KEY,
nombre varchar2(50) CONSTRAINT lugarnn_nombre NOT NULL,
tipo varchar2(6)
CONSTRAINT lugarnn_tipo CHECK (tipo IN (`PLAYA', `VOLCAN', `BOSQUE', `RIO')),
ubicacion varchar2(50) CONSTRAINT lugarnn_ubicacion NOT NULL,
provincia varchar2(10) CONSTRAINT lugarnn_provincia NOT NULL
);
CREATE TABLE TURISTA
(
numero number(5) CONSTRAINT turpk_numero PRIMARY KEY,
identificacion varchar2(15) CONSTRAINT turuk_identificacion UNIQUE (identificacion),
nombre varchar2(35) CONSTRAINT turnn_nombre NOT NULL,
telefono varchar2(14),
pais varchar2(25) CONSTRAINT turnn_pais NOT NULL,
email varchar2(50)
);
CREATE TABLE EXCURSION
(
```

```
codigo varchar2(5) CONSTRAINT excpk_codigo PRIMARY KEY,
destino number(3) CONSTRAINT excfk_destino REFERENCES lugar (numero),
actividad varchar2(5) CONSTRAINT excfk_actividad REFERENCES actividad (codigoact),
transporte number(5) CONSTRAINT excfk_alquila REFERENCES alquila (numero),
hotel varchar2(5) CONSTRAINT excfk_hotel REFERENCES hospedaje (codigo),
chofer varchar2(15) CONSTRAINT excfk_chofer REFERENCES chofer (cedula),
guia varchar2(15) CONSTRAINT excfk_guia REFERENCES guia (cedguia),
costo number(6) CONSTRAINT excnn_costo NOT NULL,
salida varchar2(80) CONSTRAINT excnn_salida NOT NULL
);
CREATE TABLE RESERVACION
(
numero number(9) CONSTRAINT respk_numero PRIMARY KEY,
turista number(5) CONSTRAINT resfk_turista REFERENCES turista (numero),
excursion varchar2(5) CONSTRAINT resfk_excursion REFERENCES excursion (codigo),
fecha date CONSTRAINT resnn_fecha NOT NULL
);
CREATE TABLE RECIBO
(
codigo varchar2(5) CONSTRAINT recpk_codigo PRIMARY KEY,
reservacion number(9) CONSTRAINT recfk_reserva REFERENCES reservacion (numero),
fecha date CONSTRAINT recnn_fecha NOT NULL,
pago number(6) CONSTRAINT recnn_pago NOT NULL
);
CREATE TABLE ALQUILA
(
```

numero number(5) CONSTRAINT alqpk_numero PRIMARY KEY,

codigobus varchar2(5) CONSTRAINT alqfk_codbus REFERENCES transporte (codigobus),

rentabus varchar2(35) CONSTRAINT alqnn_rentabus NOT NULL,

ubicacion varchar2(50) CONSTRAINT alqnn_ubicacion NOT NULL,

fecha date CONSTRAINT alqnn_fecha NOT NULL,

costodiario number(6) CONSTRAINT alqnn_costodiario NOT NULL

);

Bibliografía

Libros:

Sistemas de Bases de Datos

Carlos González Alvarado

Editorial Tecnológica de Costa Rica

Impreso en Costa Rica, 2000

Direcciones en Internet:

http://www.ibm.com/products/ar/

http://www.informix.com