• Los medicamentos se agrupan en familias, dependiendo del tipo de enfermedades a las que dicho medicamento se aplica.

• La farmacia tiene algunos clientes que realizan los pagos de sus pedidos a fin de cada mes (clientes con crédito). La farmacia quiere conocer las unidades de cada medicamento comprado (con o sin crédito) así como la fecha de compra. Además, es necesario tener los datos bancarios de los clientes con crédito, así como la fecha de pago de las compras que realizan.

Semana 8

Contenido:

- Definición y Formato para la creación de diccionario de datos.
- Ejercicios creando diccionario de Datos.
- Conceptos
- Recuperación
- Transacción
- Concurrencia – Problemas y soluciones
- Seguridad
- Manipulación de Datos
- Inserción de datos.
- Eliminación de registros
- Actualización de registros
- Conociendo el lenguaje SQL (Structured Query Language).
- Cláusulas SELECT, FROM, WHERE, ORDER BY.
- Ejercicios

DICCIONARIO DE DATOS - MANIPULACIÓN DE DATOS (DML)

➢ DICCIONARIO DE DATOS

✓ CONCEPTO

Un diccionario de datos es un conjunto de metadatos que contiene las características lógicas y puntuales de los datos que se van a utilizar en el sistema que se programa, incluyendo nombre, descripción, alias, contenido y organización.

Estos diccionarios se desarrollan durante el análisis de flujo de datos y ayuda a los analistas que participan en la determinación de los requerimientos del sistema, su contenido también se emplea durante el diseño del proyecto.

Identifica los procesos donde se emplean los datos y los sitios donde se necesita el acceso inmediato a la información, se desarrolla durante el análisis de flujo de datos y auxilia a los analistas que participan en la determinación de los requerimientos del sistema, su contenido también se emplea durante el diseño.

En un diccionario de datos se encuentra la lista de todos los elementos que forman parte del flujo de datos de todo el sistema. Los elementos más importantes son flujos de datos, almacenes de datos y procesos. El diccionario de datos guarda los detalles y descripción de todos estos elementos.

✓ FORMATO DE UN DICCIONARIO DE DATOS

Una definición de un dato se introduce mediante el símbolo "="; en este contexto el "=" se lee como "está definido por", o "está compuesto de", o "significa". Para definir un dato completamente, la definición debe incluir:

El significado del dato en el contexto de la aplicación. Esto se documenta en forma de comentario.
La composición del dato, si es que está compuesto de otros elementos significativos. Los valores que el dato puede tomar, si se trata de un dato elemental que ya no puede ser descompuesto.

Diccionario de datos (DD) Este elemento del enfoque de base de datos es el conjunto centralizado de atributos lógicos que especifican la identificación y caracterización de los datos que se manejan en la BD. La BD contiene el valor de los datos, el DD contiene meta datos, es decir los atributos lógicos de dichos datos.

✓ DATOS ELEMENTALES

Son aquellos para los cuales no hay una descomposición significativa. Por ejemplo, puede ser que no se requiera descomponer el nombre de una persona en primer-nombre, apellido-materno y apellido-paterno; esto depende del contexto del sistema que se esté modelando. Cuando se han identificado los datos elementales, deben ser introducidos en el DD y proveer una breve descripción que describa el significado del dato. En el caso de que el dato tenga un nombre significativo, se puede omitir la descripción, sin embargo; es importante especificar las unidades de medida que el dato puede tomar.

Ejemplo: Peso = * peso del paciente al ingresar al hospital *

unidad: kilo, rango:2-150 *
Altura = * unidad: cm, rango: 100-200 * Sexo = * valores : [F|M] *

✓ DATOS OPCIONALES

Un dato opcional es aquel que puede o no estar presente como componente de un dato compuesto. Ejemplo: Dirección = calle + número + (ciudad) + (país) + (código-postal)

Selección

Indica que un elemento consiste de exactamente una opción de un conjunto de alternativas.

Ejemplos:

Sexo = [Femenino | Masculino]

Tipo-de-cliente = [Gubernamental | Académico | Industria | Otros]

Iteración

Se usa para indicar ocurrencias repetidas de un componente en un elemento compuesto.

Ejemplo:

Orden-de compra = nombre-cliente + dirección-de-envío + {artículo}

Significa que una orden de compra siempre debe contener un nombre de cliente, una dirección de envío y cero o más ocurrencias de un artículo.

✓ EJERCICIOS

Se pueden especificar límites superiores e inferiores a las iteraciones.

Orden-de compra = nombre-cliente + dirección-de-envío + 1{artículo}10

Significa que una orden de compra siempre debe contener un nombre de cliente, una dirección de envío y de 1 a 10 artículos.

Ejemplos de iteraciones con límites:

a = 1{b}

a = {b}10

a = 1{b}10

a = {b}

EJEMPLO REGISTRO DE EMPLEADOS = {Registro del empleado}

REGISTRO DE TIEMPOS DEL EMPLEADO = {Registro de tiempos del empleado}

Registro del empleado = * Datos de cada empleado*
Número de empleado + Información personal + Información de pago + Información de pago actual + Información anual

Registro de tiempos del empleado = Número de empleado + Nombre del empleado + Horas trabajadas

Cheque de pago del empleado = Número de empleado + Nombre de empleado + Dirección + Cantidades del pago actual + 5

Produce el cheque de pago del empleado

REGISTRO DE TIEMPOS DEL EMPLEADO

Empleado

Cheque de pago del empleado

Registro del empleado

Registro de tiempos del empleado

REGISTRO DE EMPLEADOS

El DD provee información del DER. En general, las instancias del DER corresponden a los almacenes de datos de los DFD. EJEMPLO: CLIENTES = {cliente}

cliente = nombre-cliente + dirección + número-teléfono

compra = * asociación entre un cliente y uno o más artículos *

nombre-cliente + 1{id-artículo + cantidad-artículos}

ARTÍCULOS = {artículo}

artículo = id-artículo + descripción

En el ejemplo anterior, cliente es la definición de un tipo de objeto (entidad) y una instancia del almacén de datos CLIENTES. La llave de cliente es el atributo nombre-cliente, el cual diferencia una instancia de otra. El signo @ es usado para indicar los campos llave, o bien estos campos llave se subrayan.

✓ RAZONES PARA LA UTILIZACIÓN DE LOS DICCIONARIOS DE DATOS:

- Para manejar los detalles en sistemas muy grandes, ya que tienen enormes cantidades de datos, aun en los sistemas más chicos hay gran cantidad de datos. Los sistemas al sufrir cambios continuos, es muy difícil manejar todos los detalles. Por eso se registra la información, ya sea sobre hoja de papel o usando procesadores de texto. Los analistas mas organizados usan el diccionario de datos automatizados diseñados específicamente para el análisis y diseño de software.

- Para asignarle un solo significado a cada uno de los elementos y actividades del sistema. Los diccionarios de datos proporcionan asistencia para asegurar significados comunes para los elementos y actividades del sistema y registrando detalles adicionales relacionados con el flujo de datos en el sistema, de tal manera que todo pueda localizarse con rapidez.

- Para documentar las características del sistema, incluyendo partes o componentes así como los aspectos que los distinguen. También es necesario saber bajo que circunstancias se lleva a cabo cada proceso y con qué frecuencia ocurren. Produciendo una comprensión más completa. Una vez que las características están articuladas y registradas, todos los participantes en el proyecto tendrán una fuente común de información con respecto al sistema.

- Para facilitar el análisis de los detalles con la finalidad de evaluar las características y determinar donde efectuar cambios en el sistema. Determina si son necesarias nuevas características o si están en orden los cambios de cualquier tipo. Se abordan las características:

✓ CONCEPTOS BASICOS

RECUPERACION

La recuperación significa que, si se da algún error en los datos, hay un bug de programa ó de hardware, el DBA (Administrador de base de datos) puede traer de vuelta la base de datos al tiempo y estado en que se encontraba en estado consistente antes de que el daño se causara. Las actividades de recuperación incluyen el hacer respaldos de la base de datos y almacenar

esos respaldos de manera que se minimice el riesgo de daño o pérdida de los mismos, tales como hacer diversas copias en medios de almacenamiento removibles y almacenarlos fuera del área en antelación a un desastre anticipado. La recuperación es una de las tareas más importantes de los DBA's.

La recuperación, frecuentemente denominada "recuperación de desastres", tiene dos formas primarias. La primera son los respaldos y después las pruebas de recuperación.

La recuperación de las bases de datos consiste en información y estampas de tiempo junto con bitácoras los cuales se cambian de manera tal que sean consistentes en un momento y fecha en particular. Es posible hacer respaldos de la base de datos que no incluyan las estampas de tiempo y las bitácoras, la diferencia reside en que el DBA debe sacar de línea la base de datos en caso de llevar a cabo una recuperación.

Las pruebas de recuperación consisten en la restauración de los datos, después se aplican las bitácoras a esos datos para restaurar la base de datos y llevarla a un estado consistente en un tiempo y momento determinados. Alternativamente se puede restaurar una base de datos que se encuentra fuera de línea sustituyendo con una copia de la base de datos.

Si el DBA (o el administrador) intentan implementar un plan de recuperación de bases de datos sin pruebas de recuperación, no existe la certeza de que los respaldos sean del todo válidos. En la práctica, los respaldos de la mayoría de los RDBMSs son raramente válidos si no se hacen pruebas exhaustivas que aseguren que no ha habido errores humanos o bugs que pudieran haber corrompido los respaldos.

TRANSACCION

Una transacción es una unidad lógica de procesamiento de la base de datos que incluye una o más operaciones de acceso a la base de datos, que pueden ser de inserción eliminación, modificación o recuperación. Las transacciones pueden delimitarse con sentencias explícitas begin transaction y end transaction.

Un SGBD se dice transaccional, si es capaz de mantener la integridad de los datos, haciendo que estas transacciones no puedan finalizar en un estado intermedio. Cuando por alguna causa el sistema debe cancelar la transacción, empieza a deshacer las órdenes ejecutadas hasta dejar la base de datos en su estado inicial (llamado punto de integridad), como si la orden de la transacción nunca se hubiese realizado.

Para esto, el lenguaje de consulta de datos SQL (Structured Query Language), provee los mecanismos para especificar que un conjunto de acciones deben constituir una transacción.

BEGIN TRAN: Especifica que va a empezar una transacción.
COMMIT TRAN: Le indica al motor que puede considerar la transacción completada con éxito.
ROLLBACK TRAN: Indica que se ha alcanzado un fallo y que debe restablecer la base al punto de integridad.

En un sistema ideal, las transacciones deberían garantizar todas las propiedades ACID; en la práctica, a veces alguna de estas propiedades se simplifica o debilita con vistas a obtener un mejor rendimiento.

PROPIEDADES

Toda transacción debe cumplir cuatro propiedades ACID:

- Atomicidad (Atomicity): es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias.

- Consistencia (Consistency): es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto, se ejecutan aquellas operaciones que no van a romper la reglas y directrices de integridad de la base de datos.

- Aislamiento (Isolation): es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información nunca generará ningún tipo de error.

- Permanencia (Durability): es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.

La atomicidad frente a fallos se suele implementar con mecanismos de journaling, y la protección frente a accesos concurrentes mediante bloqueos en las estructuras afectadas. La serialibilidad viene garantizada por la atomicidad. La permanencia se suele implementar forzando a los periféricos encargados de almacenar los cambios a confirmar la completa y definitiva transmisión de los datos al medio (generalmente, el disco).

La forma algorítmica que suelen tener las transacciones es la siguiente:

```
iniciar transacción (lista de recursos a bloquear)
ejecución de las operaciones individuales.
if (todo_ok){
 aplicar_cambios
}
else{
 cancelar_cambios
}
```

En cualquier momento, el programa podría decidir que es necesario hacer fallar la transacción, con lo que el sistema deberá revertir todos los cambios hechos por las operaciones ya hechas. En el lenguaje SQL se denomina COMMIT a aplicar cambios y ROLLBACK a cancelar cambios.

Las transacciones suelen verse implementadas en sistemas de bases de datos y, más recientemente, se han visto incorporadas a como gestiona un sistema operativo la interacción con un sistema de archivos (como varias características de las bases de datos, debido a que son muy similares arquitectónicamente).

CONCURRENCIA

En sistemas multiusuario, es necesario un mecanismo para controlar la concurrencia. Se pueden producir inconsistencias importantes derivadas del acceso concurrente (3 problemas).

Las transacciones de los usuarios se podrían ejecutar de manera concurrente y podrían acceder y actualizar los mismos elementos de la BD.

Problemas y Soluciones

Problemas:

- Actualización perdida
- Actualización temporal (lectura sucia)

- Resumen incorrecto

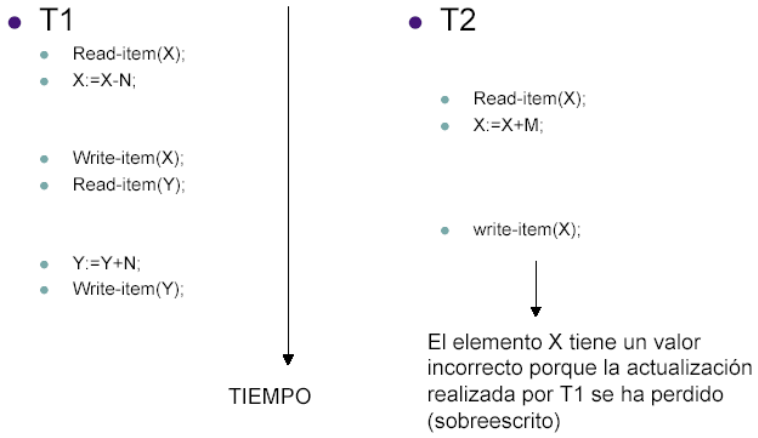
Ejemplo

Sistema de BD de reservas en una línea área.

T1: transfiere N reservas de un vuelo, cuyo número de asientos reservados está almacenado en el elemento de la BD llamado X, a otro vuelo, cuyo número de asientos reservados está almacenado en el elemento de la BD llamado Y.

ACTUALIZACIÓN PERDIDA

- Esto ocurre cuando las transacciones que tienen acceso a los mismos elementos de la BD tienen sus operaciones intercaladas de modo que hacen incorrecto el valor de algún elemento.
- T1 y T2 se introducen al mismo tiempo y sus operaciones se intercalan.



- El valor final del elemento X es incorrecto, porque T2 lee el valor de X ANTES de que T1 lo modifique en la BD, con lo que se pierde el valor actualizado que resulta de T1.

- Si X=80 al principio, N=5 (T1 transfiere 5 reservas de asientos del vuelo que corresponde a X al vuelo que corresponde a Y) y M=4 (T2 reserva 4 asientos en X), el resultado final debería ser X=79, pero es X=84, porque la actualización de T1 que eliminó 5 asientos de X se ha perdido.

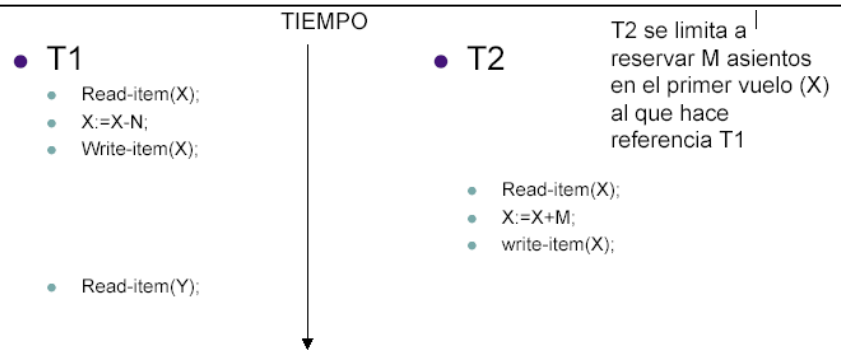
ACTUALIZACIÓN TEMPORAL

- Esto ocurre cuando una transacción actualiza un elemento de la BD y luego la transacción falla por alguna razón. Otra transacción tiene acceso al elemento actualizado antes de que se restaure a su valor original.

- T1 actualiza el elemento X y después falla antes de completarse, así que el sistema debe cambiar X otra vez a su valor original.

- Antes de que pueda hacerlo, la transacción T2 lee el valor “temporal” de X, que no se grabará permanentemente en la BD debido al fallo de T1.

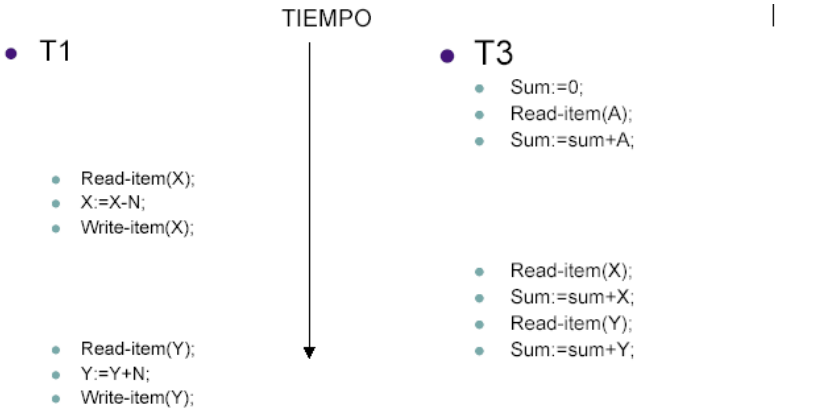
- El valor que T2 lee de X se llama dato sucio, porque fue creado por una transacción que no se ha completado ni confirmado todavía.



La T1 falla y debe restaurar X a su antiguo valor; mientras tanto T2 ha leído el valor temporal incorrecto de X

RESUMEN INCORRECTO

Si una transacción está calculando una función agregada de resumen sobre varios registros mientras otras transacciones están actualizando algunos de ellos, puede ser que la función agregada calcule algunos valores antes de que se actualicen y otros después de actualizarse



T3 lee X después de retarle N y lee Y antes de sumarsele N, así que el resultado es un resumen incorrecto (discrepancia de N)

SEGURIDAD

Seguridad significa la capacidad de los usuarios para acceder y cambiar los datos de acuerdo a las políticas del negocio, así como, las decisiones de los encargados. Al igual que otros metadatos, una DBMS relacional maneja la seguridad en forma de tablas. Estas tablas son las "llaves del reino" por lo cual se deben proteger de posibles intrusos

➢ LENGUAJE DE DEFINICION DE DATOS (DML)

Un lenguaje de manipulación de datos (Data Manipulation Language, o DML en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

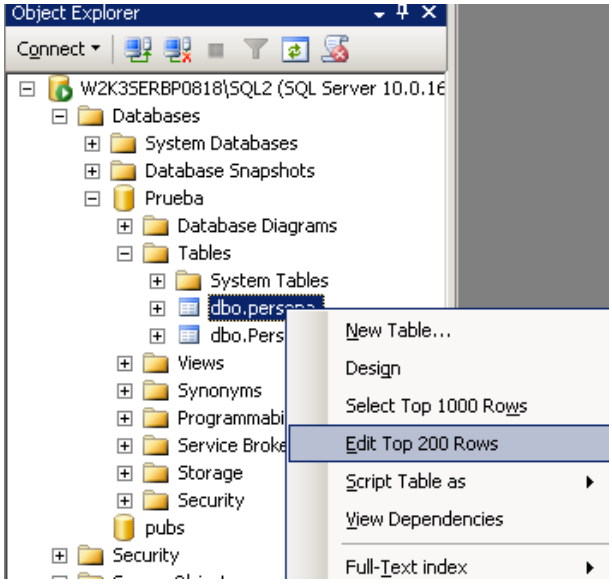
El lenguaje de manipulación de datos más popular hoy en día es SQL, usado para recuperar y manipular datos en una base de datos relacional. Otros ejemplos de DML son los usados por bases de datos IMS/DL1, CODASYL u otras.

Tenemos 4 sentencias DML's:

- **Insert into:** Sentencia que permite insertar registros de datos a las tablas de lavase de datos. Está muy ligado a la estructura de la tabla.
- **Delete:** Permite eliminar registros de datos de las tablas.
- **Update:** Sentencia que permite hacer modificaciones a los datos de las tablas.
- **Select:** La sentencia más poderosa del SQL, permite hacer consultas y recuperación de registros de datos de las tablas.

✓ INSERCION DE REGISTROS DE DATOS

Para insertar un registro de datos, hacemos clic derecho sobre la tabla en el Explorador de Objetos, seleccionamos Editar las primeras 200 filas (Edit top 200 Rows)…

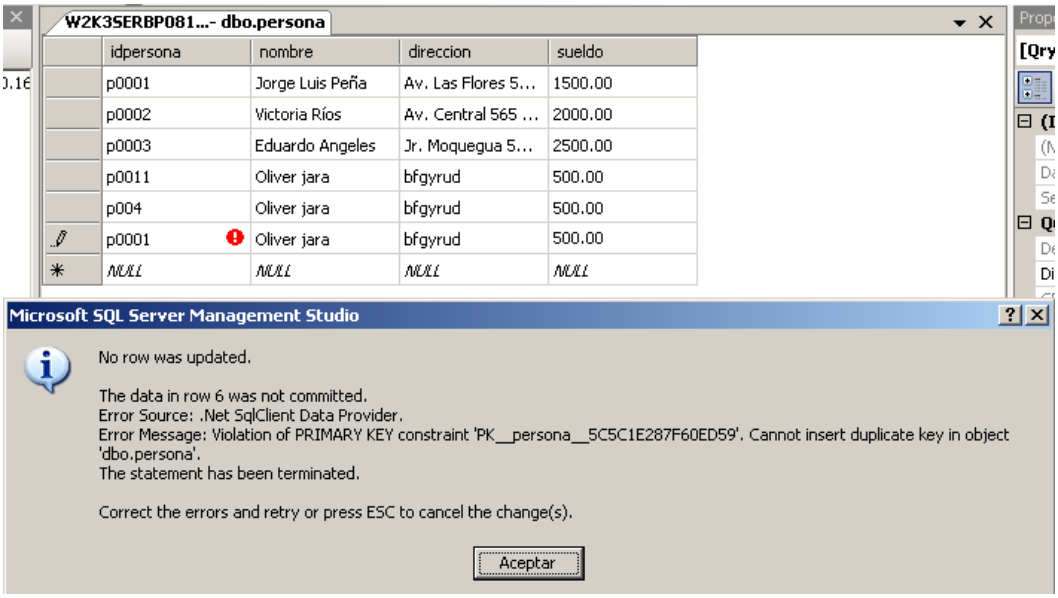


Luego de ello procedemos a insertar los registros uno a uno…

W2K3SERBP081...- dbo.persona

idpersona	nombre	direccion	sueldo
p0001	Jorge Luis Peña	Av. Las Flores 5...	1500.00
p0002	Victoria Ríos	Av. Central 565 ...	2000.00
p0003	Eduardo Angeles	Jr. Moquegua 5...	2500.00
p0011	Oliver jara	bfgyrud	500.00
p004	Oliver jara	bfgyrud	500.00
p05	Oliver jara	bfgyrud	500.00
NULL	NULL	NULL	NULL

Si insertamos algún dato que no corresponde a la definición de la tabla, o intentamos insertar una clave duplicada, nos aparece una ventana de mensaje de error…



✓ SENTENCIA DML INSERT INTO

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica

INSERT INTO "tabla" ("columna1", ["columna2,... "]) VALUES ("valor1", ["valor2,..."])

Las cantidades de columnas y valores deben ser las mismas. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

Ejemplo

INSERT INTO agenda_telefonica (nombre, numero) VALUES ('Roberto Jeldrez', '4886850');

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

INSERT INTO "tabla" VALUES ("valor1", ["valor2,..."])

Ejemplo:

Asumiendo que 'nombre' y 'numero' son las únicas columnas de la tabla 'agenda_telefonica'

INSERT INTO agenda_telefonica VALUES ('Roberto Jeldrez', '4886850');

Formas avanzadas

Inserciones en múltiples filas

Una característica de SQL (desde SQL-92) es el uso de constructores de filas para insertar múltiples filas a la vez, con una sola sentencia SQL:

INSERT INTO "tabla" ("columna1", ["columna2,... "]) VALUES ("valor1a", ["valor1b,..."]), ("value2a", ["value2b,..."]),….

Ejemplo

Asumiendo ese 'nombre' y 'numero' son las únicas columnas en la tabla 'agenda_telefonica'):

INSERT INTO agenda_telefonica VALUES ('Roberto Fernández', '4886850'), ('Alejandro Sosa', '4556550')

Que podía haber sido realizado por las sentencias

INSERT INTO agenda_telefonica VALUES ('Roberto Fernández', '4886850');

INSERT INTO agenda_telefonica VALUES ('Alejandro Sosa', '4556550');

Notar que las sentencias separadas pueden tener semántica diferente (especialmente con respecto a los triggers), y puede tener diferente rendimiento que la sentencia de inserción múltiple.

- ✓ **EJERCICIO**

Insertar registros a la siguiente tabla:

```
CREATE TABLE PERSONA(
        CODPER  CHAR(6),
        NOMBREPER  VARCHAR(40),
        SEXO CHAR(1),
        NRORUC CHAR(11),
        FECHA_INSC  DATETIME DEFAULT GETDATE()
        CONSTRAINT PK_CLIE PRIMARY KEY(CODPER),
        CONSTRAINT CK_CLIE CHECK(SEXO IN('M','F'))
)
```

Insertando registros…

INSERT INTO CLIENTE VALUES (‘PE0001’,’Arturo Perez’,’M’,’41254687841’,’12/20/2010’)

Con esto nos aparecerá un mensaje que dice: 1 fila insertada…

Pero vemos que el campo Fecha_insc tiene un constraint default, entonces podemos obviar este campo ya que tendrá este valor de todas maneras…

INSERT INTO CLIENTE (codper, nombreper, sexo, nroruc) VALUES (‘PE0002’,’ARTURO PEREZ’,’M’)

Veamos otro ejemplo con una tabla llamada Distrito:

```
CREATE TABLE DISTRITO
(
        CODIS CHAR(5) PRIMARY KEY,
        DESCRIPCION VARCHAR(50)
)
```

Insertando registros…

```
INSERT INTO DISTRITO VALUES('D0001','LA MOLINA')
INSERT INTO DISTRITO VALUES('D0002','LA VICTORIA')
INSERT INTO DISTRITO VALUES('D0003','LINCE')
INSERT INTO DISTRITO VALUES('D0004','SURQUILLO')
INSERT INTO DISTRITO VALUES('D0005','SAN ISIDRO')
```

Tener en cuenta:

- El orden de los datos debe ser el mismo orden que tienen los campos de las tablas, de lo contrario SQL Server nos mostrará un mensaje que indica que se está respetando la estructura definida de la tabla.

- Los valores numéricos para aquellos campos definidos con un tipo de dato numérico como Decimal o Int, se colocan sin comilla. Sólo llevarán comillas aquellos datos de texto y fecha

- No se ingresan aquellos datos de campos marcados con la restricción Identity.

Veamos un ejemplo de un registro mal insertado:

Utilizando nuevamente la tabla Persona:

INSERT INTO CLIENTE VALUES (‘PE0001’,’Arturo Perez’,’M’,’12/20/2010’)

Estamos cometiendo un grave error, ya que no estamos respetando la estructura de lat abla, porque no estamos indicando un valor para el campo RUCCLIE, que va después del campo SEXO, SQL Server no permitirá el ingreso de este registro.

Además estamos reingresando una clave ya repetido en el campo CODPER, según la integridad referencial de la tabla, este campo es una llave primaria por lo que no puede existir dos códigos repetidos.

Debemos ingresar este siguiente registro entonces:

INSERT INTO CLIENTE VALUES ('PE0003','Arturo Perez','M','65302104710','12/20/2010')

Debemos insertar los registros que correspondan a los campos, tomando en cuenta tipo de dato, restricción y reglas de negocio para tener siempre los datos correctos, las cuales nos entregarán siempre la información correcta requerida.

✓ ELIMINACION DE REGISTROS

Utilizamos la Sentencia SQL Delete, que permite eliminar uno o más registros de una tabla, esto lo haremos cuando ya no necesitemos dicho registro.

Forma Básica

DELETE FROM TABLA WHERE CONDICION

Descripción de las cláusulas

- FROM: Palabra clave opcional que se puede utilizar entre la palabra clave DELETE y el destino (tabla, vista o rowset)

- FROM <tabla>: Especifica una cláusula FROM adicional. Esta extensión de Transact-SQL para DELETE permite especificar datos y eliminar las filas correspondientes de la tabla en la primera cláusula FROM. Se puede utilizar esta extensión, que especifica una combinación, en lugar de una subconsulta en la cláusula WHERE para identificar las filas que se van a quitar.

- WHERE: Especifica las condiciones utilizadas para limitar el número de filas que se van a eliminar. Si no se proporciona una cláusula WHERE, DELETE quita todas las filas de la tabla.

Ejemplo:

Tenemos la siguiente tabla:

CODPER	NOMBRE	DIRECCION
PE0001	ALAN GARCIA	AV.ELSOL 345
PE0002	ERICK TORRES	JR. LAMPA 456
PE0003	JUAN VARGAS	AV. LOS ROBLES 564

Si queremos eliminar a la persona Juan Vargas, ejecutaremos la siguiente sentencia:

DELETE FROM Persona WHERE codper='PE0001'

Si no indicamos una cláusula Where, se eliminarán todos los registros sin excepción, la tabla quedaría así:

CODPER	NOMBRE	DIRECCION
PE0001	ALAN GARCIA	AV.ELSOL 345
PE0002	ERICK TORRES	JR. LAMPA 456

Tener en cuenta la integridad referencial de datos, que indica que no se podrá eliminar un registro cuya clave está relacionada con algún otro registro de una tabla relacionada, siempre y cuando se realice una eliminación en cascada, como hemos visto en sesiones anteriores.

➢ ACTUALIZACION DE REGISTROS

Utilizamos la sentencia SQL Update, que permite modificar los datos de los registros de las tablas, ideal cuando se desea actualizar un dato importante o corregir un dato erróneo.

Forma Básica.

UPDATE TABLA SET CAMPO='NUEVOVALOR' WHERE CONDICION

Descripción de las cláusulas:

- UPDATE: Aquí indicamos el nombre de la tabla sobre la cual se realizará alguna modificación de dato.

- SET: Cláusula que indica el campo sobre el cual se hará el cambio de valor, colocaremos el nuevo valor que contendrá el campo, aquí no es importante el valor que tenía hasta ese momento.

- WHERE: Especifica las condiciones utilizadas para limitar el número de filas que se van a actualizar. Si no se proporciona una cláusula WHERE, DELETE actualiza todas las filas de la tabla.

Ejemplo:

Tenemos la siguiente tabla:

CODPER	NOMBRE	DIRECCION
PE0001	ALAN GARCIA	AV.ELSOL 345
PE0002	ERICK TORRES	JR. LAMPA 456
PE0003	JUAN VARGAS	AV. LOS ROBLES 564

Para modificar la dirección de la persona Erick Torres, debemos ejecutar la siguiente instrucción:

UPDATE Persona SET Dirección = 'Av. Las Torres 453 Cercado' WHERE codper='pe0002'

En la cláusula Where debemos indicar algún dato de algunos de los campos que permita identificar al registro que debe actualizar, si indicamos mal esta condición, modificaremos registros que no deseábamos cambiar. La tabla queda así:

CODPER	NOMBRE	DIRECCION
PE0001	ALAN GARCIA	AV.ELSOL 345
PE0002	ERICK TORRES	AV. LAS TORRES 453 CERCADO'
PE0003	JUAN VARGAS	AV. LOS ROBLES 564

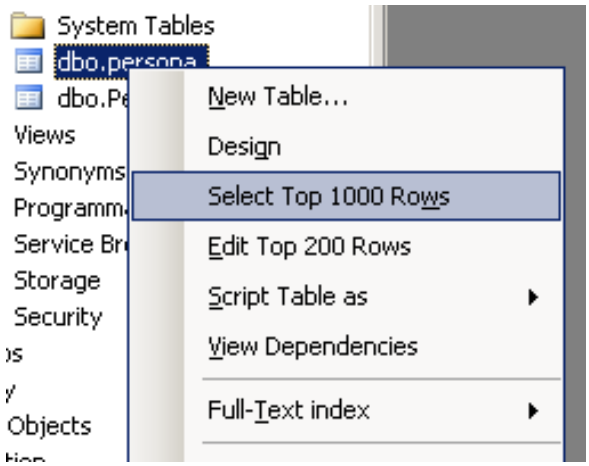
Tener en cuenta la integridad referencial de datos, que indica que se puede actualizar un dato de la tabla principal y de los registros de las tablas relacionadas, siempre y cuando se realice una actualización en cascada, como hemos visto en sesiones anteriores.

➢ CONSULTA DE DATOS

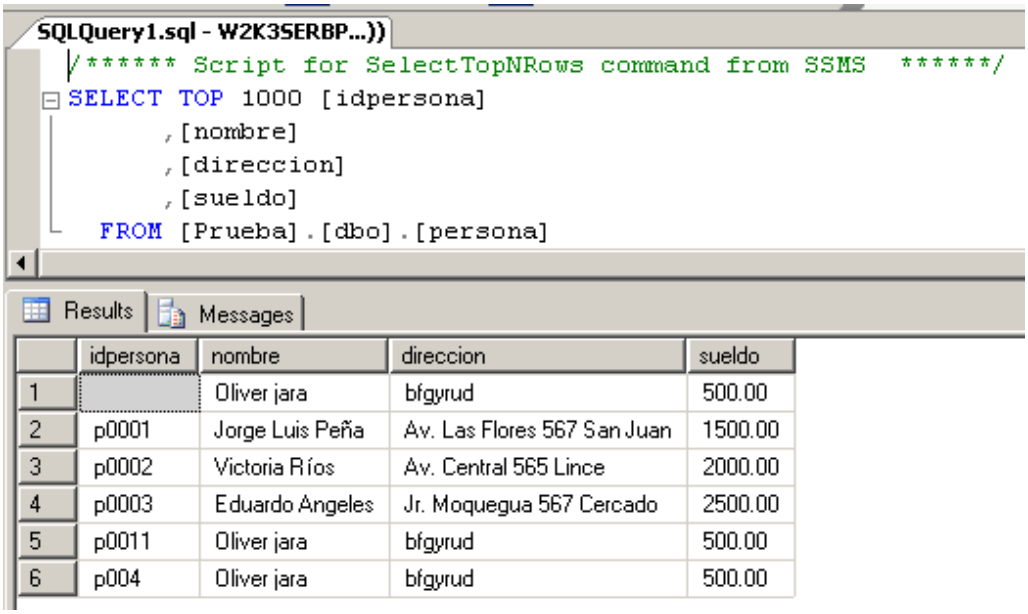
El proceso más importante que podemos llevar a cabo en una base de datos es la consulta de los datos. De nada serviría una base de datos si no pudiéramos consultarla. Es además la operación que efectuaremos con mayor frecuencia.

Para consultar la información SQL pone a nuestra disposición la sentencia SELECT.

Para consultar los registros de una tabla, podemos hacer clic derecho y seleccionamos Seleccionar las primeras 1000 filas (Select top 1000 Rows)…



Veremos la consulta con el script generado….



	idpersona	nombre	direccion	sueldo
1		Oliver jara	bfgyrud	500.00
2	p0001	Jorge Luis Peña	Av. Las Flores 567 San Juan	1500.00
3	p0002	Victoria Ríos	Av. Central 565 Lince	2000.00
4	p0003	Eduardo Angeles	Jr. Moquegua 567 Cercado	2500.00
5	p0011	Oliver jara	bfgyrud	500.00
6	p004	Oliver jara	bfgyrud	500.00

Pero veamos ahora cómo crear una consulta por medio de la sentencia DML Select…

LA SENTENCIA SELECT

La sentencia SELECT nos permite consultar los datos almacenados en una tabla de la base de datos.

El formato de la sentencia select es:

```
SELECT [ALL | DISTINCT ]
          <nombre_campo> [{,<nombre_campo>}]
FROM <nombre_tabla>|<nombre_vista>
      [{,<nombre_tabla>|<nombre_vista>}]
[WHERE <condicion> [{ AND|OR <condicion>}]]
[GROUP BY <nombre_campo> [{,<nombre_campo >}]]
[HAVING <condicion>[{ AND|OR <condicion>}]]
[ORDER BY <nombre_campo>|<indice_campo> [ASC | DESC]
            [{,<nombre_campo>|<indice_campo> [ASC | DESC ]}]]
```

Veamos por partes que quiere decir cada una de las partes que conforman la sentencia.

SELECT. Palabra clave que indica que la sentencia de SQL que queremos ejecutar es de selección.

ALL. Indica que queremos seleccionar todos los valores. Es el valor por defecto y no suele especificarse casi nunca.

DISTINCT. Indica que queremos seleccionar sólo los valores distintos.

FROM. Indica la tabla (o tablas) desde la que queremos recuperar los datos. En el caso de que exista más de una tabla se denomina a la consulta "consulta combinada" o "join". En las consultas combinadas es necesario aplicar una condición de combinación a través de una cláusula WHERE.

WHERE. Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Admiten los operadores lógicos AND y OR.

GROUP BY. Especifica la agrupación que se da a los datos. Se usa siempre en combinación con funciones agregadas.

HAVING. Especifica una condición que debe cumplirse para los datos. Especifica una condición que debe cumplirse para que los datos sean devueltos por la consulta. Su funcionamiento es similar al de WHERE pero aplicado al conjunto de resultados devueltos por la consulta. Debe aplicarse siempre junto a GROUP BY y la condición debe estar referida a los campos contenidos en ella.

ORDER BY: Presenta el resultado ordenado por las columnas indicadas. El orden puede expresarse con ASC (orden ascendente) y DESC (orden descendente). El valor predeterminado es ASC.

Para formular una consulta a la tabla tCoches (creada en el capítulo de tablas) y recuperar los campos matricula, marca, modelo, color, numero_kilometros, num_plazas debemos ejecutar la siguiente consulta. Los datos serán devueltos ordenados por marca y por modelo en orden ascendente, de menor a mayor.

```
SELECT matricula, marca, modelo, color, numero_kilometros, num_plazas FROM tCoches
ORDER BY marca,modelo
```

La palabra clave FROM indica que los datos serán recuperados de la tabla tCoches. Podríamos haber especificado más de una tabla, pero esto se verá en el apartado de consultas combinadas.

También podríamos haber simplificado la consulta a través del uso del comodín de campos, el asterisco "*".

```
SELECT * FROM tCoches ORDER BY marca,modelo
```

El uso del asterisco indica que queremos que la consulta devuelva todos los campos que existen en la tabla.

LA CLÁUSULA WHERE

La cláusula WHERE es la instrucción que nos permite filtrar el resultado de una sentencia SELECT. Habitualmente no deseamos obtener toda la información existente en la tabla, sino que queremos obtener sólo la información que nos resulte util es ese momento. La cláusula WHERE filtra los datos antes de ser devueltos por la consulta.

En nuestro ejemplo, si queremos consultar un coche en concreto debemos agregar una cláusula WHERE. Esta cláusula especifica una o varias condiciones que deben cumplirse para que la sentencia SELECT devuelva los datos. Por ejemplo, para que la consulta devuelva sólo los datos del coche con maricula M-1525-ZA debemos ejecutar la siguiente sentencia:

```
SELECT matricula, marca, modelo, color, numero_kilometros, num_plazas FROM tCoches
WHERE matricula = 'M-1525-ZA'
```

Cuando en una cláusula Where queremos incluir un tipo texto, debemos incluir el valor entre comillas simples.

Además, podemos utilizar tantas condiciones como queramos, utilizando los operadores lógicos AND y OR . El siguiente ejemplo muestra una consulta que devolverá los coches cuyas matriculas sean M-1525-ZA o bien M-2566-AA.

```
SELECT matricula, marca, modelo, color, numero_kilometros, num_plazas FROM tCoches
WHERE matricula = 'M-1525-ZA' OR matricula = 'M-2566-AA'
```

Además una condición WHERE puede ser negada a través del operador lógico NOT. La siguiente consulta devolverá todos los datos de la tabla tCohes menos el que tenga matrícula M-1525-ZA.

```
SELECT matricula, marca, modelo, color, numero_kilometros, num_plazas FROM tCoches
WHERE NOT matricula = 'M-1525-ZA'
```

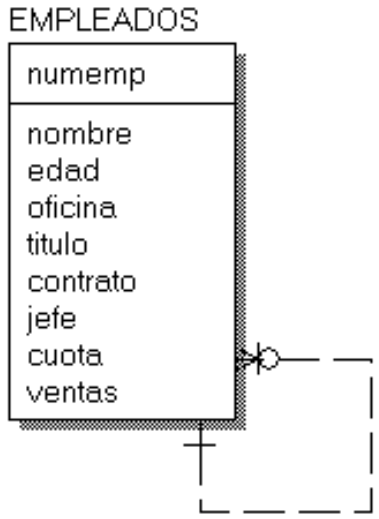
Forma básica.

La forma más básica del Select es la siguiente.

```
SELECT [Campos][*] FROM Tabla WHERE CONDICION
```

LABORATORIO # 8 – EJERCICIOS

Tenemos la siguiente tabla en nuestro modelo, contiene una relación recursiva, que indica que existen empleados que dependen de otros, el jefe.



La tabla en SQL Server…

numemp	nombre	edad	oficina	titulo	contrato	jefe	cuota	ventas
101	Antonio Viguer	45	12	representante	20/10/86	104	300.000 Pts	305.000 Pts
102	Alvaro Jaumes	48	21	representante	10/12/86	108	350.000 Pts	474.000 Pts
103	Juan Rovira	29	12	representante	01/03/87	104	275.000 Pts	286.000 Pts
104	José González	33	12	dir ventas	19/05/87	106	200.000 Pts	143.000 Pts
105	Vicente Pantalla	37	13	representante	12/02/88	104	350.000 Pts	368.000 Pts
106	Luis Antonio	52	11	dir general	14/06/88		275.000 Pts	299.000 Pts
107	Jorge Gutiérrez	49	22	representante	14/11/88	108	300.000 Pts	186.000 Pts
108	Ana Bustamante	62	21	dir ventas	12/10/89	106	350.000 Pts	361.000 Pts
109	María Sunta	31	11	representante	12/10/99	106	300.000 Pts	392.000 Pts
110	Juan Victor	41		representante	13/01/90	104		76.000 Pts

Diccionario de datos de la tabla Empleado:

Tabla empleados con los siguientes campos:

- numemp: número del empleado
- nombre : nombre y apellidos del empleado
- edad : edad del empleado
- oficina : número de la oficina donde trabaja el empleado, p.ej. Antonio Viguer trabaja en la oficina 12 de Alicante
- titulo : el cargo que desempeña el empleado
- contrato : fecha en que se contrató al empleado
- jefe: número de su jefe inmediato, p.ej. El jefe de Antonio Viguer es José González. Observar que Luis Antonio no tiene jefe, es el director general.
- cuota : cuota del empleado, sería el importe mínimo de ventas que debe alcanzar el empleado en el año
- ventas : importe de ventas realizadas durante este año

Ejecutemos algunas sentencias:

1. Insertar un nuevo registro de datos.

INSERT INTO Empleados VALUES (‘111’, ’Oliver Jara’, ’30’, ‘11’, ’dir general’, ’25/01/2010’, ’ ‘, 300.00,85.00)

2. Modificar la edad del empleado Luis Antonio quien tiene 40 años pero aparece con 52 años.

UPDATE Empleados SET edad=’40’ WHERE nombre = ‘Luis Antonio’

3. Eliminar al empleado Juan Víctor.

DELETE FROM Empleados WHERE nombre = ‘Juan Víctor’

4. Mostrar en una consulta a aquellos empleados que tienen como jefe al señor José Gonzales cuyo código es el 104.

SELECT * FROM Empleados WHERE numemp = ‘104’

Podemos hacerlo también así:

SELECT * FROM Empleados WHERE nombre = ‘José Gonzales’

5. Mostrar en una consulta a todos aquellos Empleados que tengan el título de representante o directores de ventas.

SELECT * FROM Empleados WHERE titulo = ‘representante’ OR titulo =’dir ventas’

Debemos colocar los datos exactamente igual a como se encuentran en los registros de la tabla.

6. Asignar una nueva oficina a todos aquellos que tienen la número 12 actualmente, su nueva oficina será la número 22.

UPDATE Empleados SET oficina = ‘22’ WHERE oficina =’12’

7. Mostrar en una consulta el nombre y número de todos los empleados y el de sus respectivos jefes.

SELECT numemp, nombre, jefe FROM Empleados

8. Mostrar en una consulta a todos los empleados en un listado ordenado ascendentemente según el número de oficina.

SELECT * FROM Empleados ORDER BY oficina

EJERCICIO DE INSERCION DE REGISTROS

Ejecutar las siguientes tablas e inserción de registros…

```
CREATE TABLE CLIENTES(
IDCLIENTE INT IDENTITY NOT NULL,
NOMCLIENTE VARCHAR(18)NOT NULL,
APECLIENTE VARCHAR(25)NOT NULL,
DIRCLIENTE VARCHAR(50)
PRIMARY KEY (IDCLIENTE)
)

CREATE TABLE EMPLEADO(
IDEMPLEADO INT IDENTITY NOT NULL,
NOMEMPLEADO VARCHAR(20) NOT NULL,
APEEMPLEADO VARCHAR(25)NOT NULL,
FECINGRESO SMALLDATETIME NOT NULL,
SUELDO DECIMAL(10,2) NULL CHECK (SUELDO > 0)
PRIMARY KEY(IDEMPLEADO)
)

CREATE TABLE PEDIDO(
IDPEDIDO VARCHAR(7) NOT NULL,
IDCLIENTE INT NOT NULL,
IDEMPLEADO INT NOT NULL,
FECPEDIDO SMALLDATETIME NOT NULL,
TOTAL_PEDIDO DECIMAL(8,2) CHECK (TOTALPEDIDO < 2000)
FOREIGN KEY (IDCLIENTE) REFERENCES CLIENTES,
FOREIGN KEY (IDEMPLEADO) REFERENCES EMPLEADO
)
```

Registros a insertar…

```
Insert Clientes (NomCliente,ApeCliente,DirCliente) Values('ROSA','SOLIS','BARRANCO')
Insert Clientes (NomCliente,ApeCliente,DirCliente) Values('CARLOS','MORI','LIMA')
Insert Clientes (NomCliente,ApeCliente,DirCliente) Values('JUAN','ROJAS','ATE')
Insert Clientes (NomCliente,ApeCliente,DirCliente) Values('DANIELA','CASTRO','COMAS')
Insert Clientes (NomCliente,ApeCliente,DirCliente) Values('LOURDES','VILLENA','LIMA')

Insert Empleado (NomEmpleado,ApeEmpleado,FecIngreso)
Values('CARLOS','TORRES','17/04/2000')
Insert Empleado (NomEmpleado,ApeEmpleado,FecIngreso)
Values('SOFIA','CONTRERAS','18/03/2002')
Insert Empleado (NomEmpleado,ApeEmpleado,FecIngreso)
Values('RAUL','FLORES','05/06/2001')

Insert Pedido  values('0000001',1,1,'04/11/2008',100)
Insert Pedido  values('0000002',2,1,'05/08/2008',200)
Insert Pedido  values('0000003',1,2,'14/06/2008',500)
```

Comprobamos los datos insertados…

```
SELECT * FROM CLIENTES
SELECT * FROM EMPLEADO
SELECT * FROM PEDIDO
```