Objetos de una BD

Este apunte tiene como objetivo describir los distintos objetos existentes en un motor de base de datos, indicando ejemplos de diferentes motores de mercado.

A continuación se detallan los objetos a describir en este documento:

- **Tablas**
- **Tablas Temporales**
- **Tablas Anidadas (Ora)**
- **Tablas Organizadas por Índice (Ora)**
- **Clusters (Ora)**
- **Constraints**
- **Secuencias**
- **Sinónimos/NickNames**
- **DataBase Links (Ora)**
- **Directories (Ora)**
- **Views**
- **Snapshots /Summarized Tables /Materialized Views**
- **Indices**
- **Funciones propias del motor**
- **Funciones de usuario**
- **Stored Procedures**
- **Triggers**
- **Packages (Ora / DB2)**
- **Esquemas (Ora)**
- **DTS (SQLServer)**

Tablas

Es la unidad básica de almacenamiento de datos. Los datos están almacenados en filas y columnas. Son de existencia permanente y poseen un nombre identificatorio.
Cada columna tiene entre otros datos un nombre, un tipo de datos y un ancho (este puede estar predeterminado por el tipo de dato)

Ejemplos de Creación de Tablas

Motor Informix

```
CREATE TABLE ordenes (
        N_orden INTEGER,
        N_cliente INTEGER,
        F_orden DATE,
        C_estado SMALLINT,
        F_alta_audit  DATETIME YEAR TO SECOND,
        D_usuario      VARCHAR(20)  )
```

Motor DB2

```
CREATE TABLE ordenes (
        N_orden INTEGER,
        N_cliente INTEGER,
        F_orden DATE,
        C_estado SMALLINT,
        F_alta_audit  TIMESTAMP,
        D_usuario      VARCHAR(20)  )
```

Motor Oracle

```
CREATE TABLE ordenes
```

```
( N_orden NUMBER,
  N_cliente NUMBER,
  F_orden DATE,
  C_estado NUMBER,
  F_alta_audit DATE,
  D_usuario VARCHAR2(20) )
```

Motor SQL Server

```
CREATE TABLE ordenes (
        N_orden int NULL ,
        N_cliente int NULL ,
        F_orden datetime NULL ,
        C_estado smallint NULL ,
        F_alta_audit timestamp NULL ,
        D_usuario varchar (20) NULL )
```

Tablas temporales

Es posible crear tablas temporales que duran mientras se ejecute la aplicación o lo que el usuario desee dentro de la misma sesi.
No es posible alterar tablas temporarias. Si Dropearlas y crear los índices que necesite la aplicación. Las tablas temporales pueden ser creadas y usadas mientras dure la sesión. La tabla no grabará ningún log transaccional si así se configura. No se actualizan las tablas de catálogo.

Son muy usadas para cálculos de resultados intermedios en procesos y en algunos casos para dividir consultas complejas con muchas tablas (de cinco en adelante) en consultas más acotadas con menos tablas que generan tablas temporarias intermedias.

En algunos motores de BD las Tablas temporales generadas pueden ser Globales para las cuáles los datos en una tabla temporal serán automáticamente borrados en el caso de la terminación de la sesión. Cada sesión sólo puede ver y modificar sus propios datos. Sobre ellos, no se generan bloqueos de DML.

Ejemplos de Creación de Tablas Temporales

Motor Informix

Creación de una tabla temporal e inserción posterior de datos de la tabla ordenes. La tabla Ordenes_pendientes se borrará automáticamente cuando finalice la sesión.

```
CREATE TEMP TABLE ordenes_pendientes (
        N_orden INTEGER,
        N_cliente INTEGER,
        F_orden DATE,
        I_Total DECIMAL(15 , 2),
        C_estado SMALLINT,
        F_alta_audit  DATETIME YEAR TO SECOND,
        D_usuario      VARCHAR(20) )
        WITH NO LOG;

INSERT INTO ordenes_Pendientes
 (SELECT * FROM ordenes WHERE c_estado = 1)
```

Creación y carga de una tabla temporal desde un SELECT.

```
SELECT *
  FROM ordenes
 WHERE c_estado = 1
INTO TEMP ordenes_Pendientes
WITH NO LOG
```

Motor DB2

Creación de una tabla temporal e inserción posterior de datos de la tabla ordenes. La tabla Ordenes_pendientes se borrará automáticamente cuando finalice la sesión.

```
DECLARE GLOBAL TEMPORARY TABLE ordenes_pendientes
            LIKE ordenes
            ON COMMIT PRESERVE ROWS NOT LOGGED IN SESIONTEMP;

INSERT INTO ordenes_Pendientes
 SELECT * FROM ordenes WHERE c_estado = 1
```

Creación y carga de una tabla temporal desde un SELECT.

```
SELECT *
  FROM ordenes
 WHERE c_estado = 1
INTO TEMP ordenes_Pendientes
```

Motor Oracle

Creación de tabla temporal en base a la tabla ordenes. La tabla Ordenes_pendientes se borrará automáticamente cuando finalice la sesión.

```
CREATE GLOBAL TEMPORARY TABLE ordenes_pendientes
ON COMMIT PRESERVE ROWS
AS SELECT * FROM ordenes WHERE c_estado = 1;
```

Creación y carga de una tabla temporal desde un SELECT.

```
SELECT *
INTO ordenes_Pendientes
FROM ordenes
WHERE c_estado = 1
```

Motor SqlServer

Se pueden crear tablas temporales locales y globales. Las tablas temporales locales son visibles sólo en la sesión actual; las tablas temporales globales son visibles para todas las sesiones.
Coloque un prefijo de signo numérico simple (#table_name) en los nombres de las tablas temporales locales y un prefijo de un signo numérico doble (##table_name) en los nombres de las tablas temporales globales.

Creación de tabla temporal en base a la tabla ordenes. La tabla Ordenes_pendientes se borrará automáticamente cuando finalice la sesión.

```
CREATE TABLE #ordenes_pendientes (
        N_orden INTEGER,
        N_cliente INTEGER,
        F_orden DATE,
        I_Total DECIMAL(15 , 2),
        C_estado SMALLINT,
        F_alta_audit  TIMESTAMP,
        D_usuario  VARCHAR(20) )
        WITH NO LOG;

INSERT INTO #ordenes_Pendientes
 SELECT * FROM ordenes WHERE c_estado = 1
```

Creación y carga de una tabla temporal desde un SELECT.

```
SELECT *
INTO #ordenes_Pendientes
FROM ordenes
WHERE c_estado = 1
```

Tablas anidadas (Oracle)

Es posible crear una tabla con una columna cuyo tipo de dato sea otra tabla. De esta forma, las tablas pueden anidarse dentro de otras tablas como valores en una columna.

```
CREATE TYPE address_t AS OBJECT (
   street VARCHAR2(30),
   city VARCHAR2(20),
   state CHAR(2),
   zip CHAR(5) );
/
CREATE TYPE address_tab IS TABLE OF address_t;
/
CREATE TABLE customers (
   custid NUMBER,
   address address_tab )
NESTED TABLE address STORE AS customer_addresses;

INSERT INTO customers VALUES (1,
        address_tab(
          address_t('101 First', 'Redwood Shores', 'CA', '94065'),
          address_t('123 Maple', 'Mill Valley', 'CA', '90952')
        ) );
```

Tablas Organizadas por Índice (IOT - Index-Organized Tables) (Oracle)

Una IOT tiene una organización del almacenamiento que es una variante del Arbol-B. A diferencia de una tabla común, en la que los datos se guardan como un conjunto desordenado, en una IOT se guardan en una estructura de índice de Árbol-B, ordenado a la manera de una clave primaria. Sin embargo, además de almacenar los valores de las columnas de clave primaria de cada tabla, cada entrada en el índice almacena además los valores de las columnas no clave.

De esta forma, en vez de mantener dos estructuras separadas de almacenamiento, una para la tabla y una para el índice de Árbol-B, el sistema mantiene sólo un índice Árbol-B, porque además de almacenar el rowid de las filas, también se guardan las columnas no clave.

Motor Oracle

La siguiente sentencia crea la tabla ordenes, como una tabla organizada por índice, indicando que la columna N_cliente divide el área del índice, de las columnas no clave.

```
CREATE TABLE ordenes
(   N_orden NUMBER PRIMARY KEY,
    N_cliente NUMBER,
    F_orden DATE,
   C_estado NUMBER,
   F_alta_audit DATE,
   D_usuario VARCHAR2(20) )
ORGANIZATION INDEX
INCLUDING N_CLIENTE OVERFLOW
```

Clusters (Agrupamientos) (Oracle)

Un cluster es un grupo de tablas que comparten los mismos bloques de datos porque tienen columnas comunes compartidas y que a menudo se usan juntas. Cuando se agrupan tablas, el motor guarda físicamente todas las filas de cada una de ambas tablas en los mismos bloques de datos.
Esto brinda algunos beneficios:

- Reducción de E/S a disco para los joins de las tablas agrupadas
- Mejora en tiempos de acceso para los joins de las tablas agrupadas
- En un cluster, el valor de clave de cluster es el valor de las columnas clave del cluster para una fila en particular. Cada valor de clave del cluster se guarda sólo una vez en el cluster y en el índice del cluster, independientemente de cuántas filas de diferentes tablas contengan dicho valor. Por lo tanto, se requiere mens espacio para almacenar tablas relacionadas e índices de datos en un cluster.

Ejemplo de creación de Cluster

Motor Oracle

Creación de cluster con clave N_ORDEN.

```
CREATE CLUSTER ventas
(n_orden NUMBER)
SIZE 512
STORAGE (initial 100K next 50K);
```

Clave de cluster

La sentencia a continuación crea el índice de cluster sobre la clave de cluster de ventas.

```
CREATE INDEX idx_ventas ON CLUSTER ventas;
```

Luego de crear el índice de cluster, se pueden agregar tablas al índice. La siguiente sentencia crea tablas y las agrega al cluster:

```
CREATE TABLE ordenes
(   N_orden NUMBER PRIMARY KEY,
    N_cliente NUMBER,
```

```
   F_orden DATE,
   C_estado NUMBER,
   F_alta_audit DATE,
   D_usuario VARCHAR2(20) )
CLUSTER ventas (n_orden);

CREATE TABLE item_ordenes
(  N_orden NUMBER REFERENCES ordenes,
   N_item NUMBER,
   C_producto NUMBER,
   Q_cantidad NUMBER,
    I_precunit NUMBER(9,3),
   PRIMARY KEY (n_orden, n_item) )
CLUSTER ventas (n_orden);
```

Constraints (Restricciones)

Integridad de Entidades: La integridad de entidades es usada para asegurar que los datos pertenecientes a una misma tabla tienen una única manera de identificarse, es decir que cada fila de cada tabla tenga una primary key capaz de identificar unívocamente una fila y esa no puede ser nula.

PRIMARY KEY CONSTRAINT: Puede estar compuesta por una o más columnas, y deberá representar unívocamente a cada fila de la tabla. No debe permitir valores nulos.

Integridad Referencial: La integridad referencial es usada para asegurar la coherencia entre los datos. Ej.: Si se ingresa un comprobante de un cliente, este debe existir en la tabla de clientes.

FOREIGN KEY CONSTRAINT: Puede estar compuesta por una o más columnas, y estará referenciando a la PRIMARY KEY de otra tabla.

Los constraints referenciales permiten a los usuarios especificar claves primarias y foráneas para asegurar una relación PADRE-HIJO (MAESTRO-DETALLE).

Reglas que son aseguradas con el uso de constraints referenciales:

1. Si un usuario BORRA (DELETE) una PRIMARY KEY y dicha clave está correspondida por FOREIGNS KEY en otras tablas, el DETELE fallará. Esta regla podría pasarse mediante la aplicación de cláusulas de ON DELETE CASCADE (Borrado en Cascada).
2. Si un usuario MODIFICA (UPDATE) una PRIMARY KEY y la clave original está correspondida por FOREIGNS KEY en otras tablas, el UPDATE fallará
3. No hay restricciones asociadas al borrado de FOREIGNS KEYS.
4. Si un usuario MODIFICA (UPDATE) una FOREIGN KEY y no hay una PRIMARY KEY en la tabla de referencia que corresponda a esa nueva clave NO NULA, el UPDATE fallará.
5. Todos los valores en una PRIMARY KEY deben ser únicos. Al tratar de insertar una clave duplicada en una PRIMARY KEY dará un error.
6. Cuando un usuario INSERTA (INSERT) una fila en una tabla hijo, si todas las FOREIGN KEYS son NO NULAS, todas deberán estar correspondidas por una PRIMARY KEY, en caso contrario, el INSERT fallará.

TIPOS DE CONSTRAINTS REFERENCIALES

CICLIC REFERENTIAL CONTSTRAINT.

Asegura una relación de PADRE-HIJO entre tablas. Es el más común.

Ej. CLIENTE → FACTURAS

PK Clientes.c_cliente

FK Facturas.c_cliente que referencia a la PK de la tabla Clientes.

SELF REFERENCING CONSTRAINT.

Asegura una relación de PADRE-HIJO entre la misma tabla.

Ej. EMPLEADOS → EMPLEADOS

PK Empleados.c_empleado

FK Empleados.c_jefe que referencia a la PK de la tabla Empleados, este campo además admite NULOS.

Esto significa que para todo empleado que el campo jefe sea distinto de NULL, dicho código debe existir como empleado.

MULTIPLE PATH CONSTRAINT.

Se refiere a una PRIMARY KEY que tiene múltiples FOREIGN KEYS. Este caso también es muy común.

Ej. CLIENTES → FACTURAS
CLIENTES → RECLAMOS

PK Clientes.c_cliente

FK Facturas.c_cliente que referencia a la PK de la tabla Clientes.

FK Reclamos.c_cliente que referencia a la PK de la tabla Clientes.

Integridad Semántica:

La integridad semántica es la que nos asegura que los datos que vamos a almacenar tengan una apropiada configuración. (*)

DATA TYPE: Este define el tipo de valor que se puede almacenar en una columna. (Ver apuntes ORA, IFX, DB2 Data Types)

DEFAULT CONSTRAINT: Es el valor insertado en una columna cuando al insertar un registro ningún valor fue especificado para dicha columna. El valor default por default es el NULL.

- Se aplica a columnas no listadas en una sentencia INSERT.
- El valor por default puede ser un valor literal o una función SQL (USER, TODAY, etc.)
- Aplicado sólo durante un INSERT (NO UPDATE).

CHECK CONSTRAINT: Especifica condiciones para la inserción o modificación en una columna. Cada fila insertada en una tabla debe cumplir con dichas condiciones. Actua tanto en el INSERT, como en el UPDATE.

- Es una expresión que devuelve un valor booleano de TRUE o FALSE.
- Son aplicados para cada fila que es INSERTADA o MODIFICADA.
- Todas las columnas a las que referencia deben ser de la misma tabla (la corriente).
- No puede contener subconsultas, secuencias, funciones (de fecha, usuario) ni pseudocolumnas.
-
- Todas las filas existentes en una tabla deben pasar un nuevo constraint creado para dicha tabla. En el caso de que alguna de las filas no cumpla, no se podrá crear dicho constraint.

UNIQUE CONSTRAINT: Especifica sobre una o más columnas que la inserción o actualización de una fila contiene un valor único en esa columna o conjunto de columnas.

NOT NULL CONSTRAINT: Asegura que una columna contenga un valor durante una operación de INSERT o UPDATE. Se considera el NULL como la ausencia de valor.

Ejemplos de Restricciones Primary y Foreign Keys

Motor Informix

Creación de una tabla con Primary Key y otra con una Primary key compuesta y con una Foreign Keys.

```
CREATE TABLE ordenes (
        N_orden INTEGER PRIMARY KEY,
        N_cliente INTEGER,
        F_orden DATE,
        C_estado SMALLINT,
        F_alta_audit  DATETIME YEAR TO SECOND,
        D_usuario      VARCHAR(20)
                        );

CREATE TABLE items_ordenes (
        N_orden         INTEGER REFERENCES ordenes,
        N_item          SMALLINT,
        C_producto      INTEGER,
        Q_cantidad      INTEGER,
        I_precunit      DECIMAL ( 9,3),
        PRIMARY KEY (n_orden, n_item) );
```

Creación de una tabla con Primary Key y una Foreign Key consigo misma.

```
CREATE TABLE empleados (
        N_empleado    INTEGER PRIMARY KEY,
        D_Apellido    VARCHAR(60,20),
        D_nombres     VARCHAR(60,20),
        N_cuil        DECIMAL(11,0),
        N_jefe        INTEGER,
        FOREIGN KEY (n_jefe) REFERENCES empleados (n_empleado) );
```

El motor no permitirá ingresar un empleado cuyo nro. de jefe no exista como número de empleado. Lo que si permitirá es ingresar un nro. de jefe NULO.

Motor DB2

Creación de una tabla con Primary Key y otra con una Primary key compuesta y con una Foreign Keys.

```
CREATE TABLE ordenes (
        N_orden INTEGER  NOT NULL PRIMARY KEY ,
        N_cliente INTEGER,
        F_orden DATE,
        C_estado SMALLINT,
        F_alta_audit  TIMESTAMP,
        D_usuario      VARCHAR(20)
);

CREATE TABLE items_ordenes (
        N_orden         INTEGER NOT NULL REFERENCES ordenes,
        N_item          SMALLINT NOT NULL,
        C_producto      INTEGER,
        Q_cantidad      INTEGER,
        I_precunit      DECIMAL ( 9,3),
        PRIMARY KEY (n_orden, n_item)
);
```

Creación de una tabla con Primary Key y una Foreign Key consigo misma.

```
CREATE TABLE empleados (
        N_empleado      INTEGER NOT NULL,
        D_Apellido      VARCHAR(60,20),
        D_nombres       VARCHAR(60,20),
        N_cuil          DECIMAL(11,0),
        N_jefe          INTEGER,
        PRIMARY KEY (n_empleado),
        FOREIGN KEY (n_jefe) REFERENCES empleados (n_empleado)
);
```

El motor no permitirá ingresar un empleado cuyo nro. de jefe no exista como número de empleado. Lo que si permitirá es ingresar un nro. de jefe NULO.

(En DB2 Los atributos pertenecientes a una clave primaria o que tengan un constraint de UNIQUE deben tener obligatoriamente el constraint de NOT NULL en forma explicita.)

Motor ORACLE

Creación de una tabla con Primary Key y otra con una Primary key compuesta y con una Foreign key.

```
CREATE TABLE ordenes
(       N_orden NUMBER PRIMARY KEY,
        N_cliente NUMBER,
        F_orden DATE,
        C_estado NUMBER,
        F_alta_audit DATE,
        D_usuario VARCHAR2(20) );

CREATE TABLE items_ordenes
(       N_orden NUMBER REFERENCES ordenes,
```

```
N_item NUMBER,
C_producto NUMBER,
Q_cantidad NUMBER,
I_precunit NUMBER(9,3),
PRIMARY KEY (n_orden, n_item) );
```

Creación de una tabla con Primary Key y una Foreign Key consigo misma.

```
CREATE TABLE empleados
(       N_empleado NUMBER PRIMARY KEY,
        D_Apellido VARCHAR2(60),
        D_nombres VARCHAR2(60),
        N_cuil NUMBER (11),
        N_jefe NUMBER,
        FOREIGN KEY (n_jefe) REFERENCES empleados (n_empleado)
);
```

El motor no permitirá ingresar un empleado cuyo nro. de jefe no exista como número de empleado. Lo que si permitirá es ingresar un nro. de jefe NULO.

Motor SQL Server

Creación de una tabla con Primary Key y otra con una Primary key compuesta y con una Foreign key.

```
CREATE TABLE ordenes (
        N_orden int PRIMARY KEY ,
        N_cliente int NULL ,
        F_orden datetime NULL ,
        C_estado smallint NULL ,
        F_alta_audit datetime NULL ,
        D_usuario varchar (20) NULL )

CREATE TABLE items_ordenes (
        N_orden int REFERENCES ordenes ,
        N_item int NOT NULL ,
        C_producto int NULL ,
        Q_cantidad int NULL ,
        I_precunit numeric(9, 3) NULL,
        PRIMARY KEY (N_orden, N_item)
)
```

Creación de una tabla con Primary Key y una Foreign Key consigo misma.

```
CREATE TABLE empleados (
        N_empleado int PRIMARY KEY ,
        D_Apellido varchar (60) NULL ,
        D_nombres varchar (60) NULL ,
        N_cuil numeric(11, 0) NULL ,
        N_jefe int NULL REFERENCES empleados
)
```

El motor no permitirá ingresar un empleado cuyo nro. de jefe no exista como número de empleado. Lo que si permitirá es ingresar un nro. de jefe NULO.

Ejemplos de Restricciones NOT NULL, UNIQUE, DEFAULT, CHECK

Motor Informix

Creación de una tabla con una restricción de NOT NULL y otra de DEFAULT.

```
CREATE TABLE ordenes (
        N_orden INTEGER NOT NULL,
        N_cliente INTEGER,
        F_orden DATE,
        C_estado SMALLINT DEFAULT 1,
        F_alta_audit  DATETIME YEAR TO SECOND,
        D_usuario      VARCHAR(20)  );
```

El motor de BD no permitirá ingresar un registro a la tabla ordenes con un nro. de orden NULA y en sólo en el caso de un Insert si campo c_estado viene NULO le asignará por default el valor 1.

Creación de una tabla con una restricción de UNIQUE y otra de CHECK

```
CREATE TABLE empleados (
        N_empleado    INTEGER,
        D_Apellido    VARCHAR(60,20),
        D_nombres     VARCHAR(60,20),
        N_cuil        DECIMAL(11,0) UNIQUE,
        F_nacimiento  DATE,
        F_ingreso     DATE,
        N_jefe        INTEGER,
        CHECK (F_nacimiento < F_ingreso)
                              );
```

Al ingresar un registro el sistema va a validar que no haya en la tabla de empleados otro empleado con el mismo nro. de cuil, y tanto para la inserción como para las modificaciones futuras se va a asegurar que para cada registro de la tabla siempre se cumpla el valor del campo f_ingreso sea mayor el valor del campo f_nacimiento.

Motor DB2

Creación de una tabla con una restricción de NOT NULL y otra de DEFAULT.

```
CREATE TABLE ordenes (
        N_orden INTEGER NOT NULL,
        N_cliente INTEGER,
        F_orden DATE,
        C_estado SMALLINT WITH DEFAULT 1,
        F_alta_audit  TIMESTAMP,
        D_usuario      VARCHAR(20)  );
```

El motor de BD no permitirá ingresar un registro a la tabla ordenes con un nro. de orden NULA y en sólo en el caso de un Insert si campo c_estado viene NULO le asignará por default el valor 1.

Creación de una tabla con una restricción de UNIQUE y otra de CHECK

```
CREATE TABLE empleados (
        N_empleado      INTEGER,
        D_Apellido      VARCHAR(60),
        D_nombres       VARCHAR(60),
        N_cuil          DECIMAL(11,0) NOT NULL UNIQUE,
        F_nacimiento    DATE,
        F_ingreso       DATE,
        N_jefe          INTEGER,
        CHECK (F_nacimiento < F_ingreso)
                        );
```

Al ingresar un registro el sistema va a validar que no haya en la tabla de empleados otro empleado con el mismo nro. de cuil, y tanto para la inserción como para las modificaciones futuras se va a asegurar que para cada registro de la tabla siempre se cumpla el valor del campo f_ingreso sea mayor el valor del campo f_nacimiento.

Motor Oracle

Creación de una tabla con una restricción de NOT NULL y otra de DEFAULT.

```
CREATE TABLE ordenes
(
        N_orden NUMBER NOT NULL,
        N_cliente NUMBER,
        F_orden DATE,
       C_estado NUMBER DEFAULT 1,
       F_alta_audit DATE,
       D_usuario VARCHAR2(20)
);
```

El motor de BD no permitirá ingresar un registro a la tabla ordenes con un nro. de orden NULA y sólo en el caso de un Insert si campo c_estado viene NULO le asignará por default el valor 1.

Creación de una tabla con una restricción de UNIQUE y otra de CHECK

```
CREATE TABLE empleados
(
        N_empleado NUMBER,
        D_Apellido VARCHAR2(60),
        D_nombres VARCHAR2(60),
        N_cuil NUMBER (11) UNIQUE,
        F_nacimiento DATE,
        F_ingreso DATE,
        N_jefe NUMBER,
        CHECK (F_nacimiento < F_ingreso)
);
```

Al ingresar un registro el sistema va a validar que no haya en la tabla de empleados otro empleado con el mismo nro. de cuil, y tanto para la inserción como para las modificaciones futuras se va a asegurar que para cada registro de la tabla siempre se cumpla el valor del campo f_ingreso sea mayor el valor del campo f_nacimiento.

Motor SQL Server

Creación de una tabla con una restricción de NOT NULL y otra de DEFAULT.

```
CREATE TABLE ordenes (
        N_orden int NOT NULL ,
        N_cliente int NULL ,
        F_orden datetime NULL ,
        C_estado smallint NULL DEFAULT 1,
        F_alta_audit datetime NULL ,
        D_usuario varchar (20) NULL )
```

El motor de BD no permitirá ingresar un registro a la tabla ordenes con un nro. de orden NULA y en sólo en el caso de un Insert si campo c_estado viene NULO le asignará por default el valor 1.

Creación de una tabla con una restricción de UNIQUE y otra de CHECK

```
CREATE TABLE empleados (
        N_empleado int IDENTITY (1, 1) NOT NULL ,
        D_Apellido varchar (60) NULL ,
        D_nombres varchar (60) NULL ,
        N_cuil numeric(11, 0) NULL UNIQUE,
        F_nacimiento datetime NULL ,
        F_ingreso datetime NULL ,
        N_jefe int NULL,
        CHECK  (F_nacimiento < F_ingreso)
)
```

Al ingresar un registro el sistema va a validar que no haya en la tabla de empleados otro empleado con el mismo nro. de cuil, y tanto para la inserción como para las modificaciones futuras se va a asegurar que para cada registro de la tabla siempre se cumpla el valor del campo f_ingreso sea mayor el valor del campo f_nacimiento.

Secuencias

Los generadores de secuencias proveen una serie de números secuenciales, especialmente usados en entornos multiusuarios para generar una números secuenciales y únicos sin el overhead de I/O a disco o el lockeo transaccional.

Una secuencia debe tener un nombre, debe ser ascendente o descendente, debe tener definido el intervalo entre números, tiene definidos métodos para obtener el próximo número ó el actual (entre otros).

Ejemplos de Secuencias

Motor Informix

El motor Informix no posee secuencias, en su defecto tiene un tipo de datos SERIAL que permite realizar lo mismo. Al insertar una fila en dicha tabla y asignarle un valor cero, el motor va a buscar el próximo nro. del más alto existente en la tabla.

```
CREATE TABLE ordenes (
        N_orden SERIAL,
        N_cliente INTEGER,
        F_orden DATE,
        I_Total  DECIMAL(15 , 2),
        C_estado SMALLINT,
        F_alta_audit  DATETIME YEAR TO SECOND,
        D_usuario  VARCHAR(20)
                        );

INSERT INTO ordenes
     VALUES (0, 17, '15/05/2006', 1, CURRENT, USER)
```

Las funciones CURRENT y USER son funciones propias de Informix y recuperan el año, mes, día, hora, minutos, segundos y fracción, y el usuario que tiene abierto la sesión, respectivamente.

Motor DB2

El motor DB2 posee secuencias y posee un atributo para popular una columna de una tabla con una secuencia automática.

En el ejemplo se observa la creación de una secuencia denominada sqc_orden_nro, la inserción de una fila en la tabla ordenes con el proximo valor de la secuencia y la inserción en la tabla ordenes_itens con el último valor de orden generado en la secuencia.

```
CREATE SEQUENCE sqc_orden_nro AS INTEGER;

INSERT INTO ordenes VALUES
                (NEXTVAL FOR  sqc_orden_nro, 17, '05/15/2006', 1, CURRENT
                 TIMESTAMP, USER)

INSERT INTO items_ordenes VALUES
                (PREVVAL FOR sqc_orden_nro,  1, 176,  20, 10.57 );
```

Las funciones CURRENT TIMESTAMP y USER son funciones propias de DB2 y recuperan el año, mes, día, hora, minutos, segundos y fracción, y el usuario que tiene abierto la sesión, respectivamente.

En este otro ejemplo vemos el uso del atributo IDENTITY.

```
CREATE TABLE ordenes (
        N_orden INT  GENERATED ALWAYS AS IDENTITY ,
        N_cliente INTEGER,
        F_orden DATE,
        I_Total DECIMAL(15 , 2),
        C_estado SMALLINT,
        F_alta_audit  DATETIME YEAR TO SECOND,
        D_usuario      VARCHAR(20) );

INSERT INTO ordenes
(n_cliente, f_orden, I_total, c_estado,F_alta_audit, D_usuario)
VALUES   (17, '05/15/2006', 100, 1,CURRENT TIMESTAMP, USER);
```

Motor Oracle

El motor DB2 posee secuencias. En el ejemplo se observa la creación de una secuencia denominada sqc_orden_nro, la inserción de una fila en la tabla ordenes con el proximo valor de la secuencia y la inserción en la tabla ordenes_itens con el último valor de orden generado en la secuencia.

```
CREATE SEQUENCE sqc_orden_nro
INCREMENT BY 1
START WITH 10
MAXVALUE 9999
NOCYCLE
NOCACHE;

INSERT INTO ordenes VALUES
            (sqc_orden_nro.NEXTVAL, 17, to_date('15/05/2006', 'dd/mm/yyyy'), 1,
             SYSDATE, USER);

INSERT INTO items_ordenes VALUES
            (sqc_orden_nro.CURRVAL, 1, 176, 20, 10.57);
```

Las funciones SYSDATE y USER son funciones de Oracle que recuperan la fecha y hora y usuario actuales respectivamente.

Motor SQL Server

El motor SQL Server no posee secuencias, en su defecto se define un campo como IDENTITY que permite realizar lo mismo. Al insertar una fila en dicha tabla, el motor va a buscar el próximo nro. del más alto existente en la tabla.

```
CREATE TABLE ordenes (
N_orden int IDENTITY (1, 1) NOT NULL ,
N_cliente int NULL ,
F_orden datetime NULL ,
I_Total decimal(15 , 2),
C_estado smallint NULL ,
F_alta_audit datetime NULL ,
D_usuario varchar (20) NULL )

INSERT INTO ordenes
(N_cliente, F_orden, I_total, C_estado, F_alta_audit, D_usuario)
VALUES (17, '15/05/2006', 100, 1, GETDATE(), USER)
```

Las funciones GETDATE() y USER son funciones propias de sql Server y recuperan el año, mes, día, hora, minutos, segundos y fracción, y el usuario que tiene abierto la sesión, respectivamente. El valor para el campo identity no debe completarse.

Sinónimos (NickName / Synonym)

Un sinónimo es un alias definido sobre una tabla, vista ó snapshot. Dependiendo el motor de BD puede ser también definido sobre un procedure, una secuencia, función o package.
Los sinónimos se usan a menudo por seguridad o por conveniencia (por ej. en entornos distribuidos).
Permiten enmascarar el nombre y dueño de un determinado objeto.
Proveen de una ubicación transparente para objetos remotos en una BD distribuida.
Simplifican las sentencias SQL para usuarios de la BD.

Ejemplos de Sinónimos

Motor Informix

Creación de un sinónimo con el nombre ordenes_cordoba en la BD_comercial en Buenos Aires que apunta a la tabla ordenes de la BD bd_comercial del Servidor en Córdoba.

```
DATABASE bd_comercial@server_bsas;
CREATE SYNONYM  ordenes_cordoba FOR bd_comercial@server_cordoba:ordenes
```

Motor DB2

Creación de un nickname en Bases de Datos Federadas. Un Sistema Federado de base de datos es un tipo de especial de sistemas de base de datos distribuidas.
Se crea un nickname con el nombre ordenes_cordoba en la BD_comercial en Buenos Aires que apunta a la tabla ordenes de la BD bd_comercial del Servidor en Córdoba.
Previamente se deberá crear una definición de un server en la que se debe especificar el usuario y password que puede contectarse a la fuente remota y el nombre del server al que se hará referencia.

El nickname puede ser creado para acceder localmente a una tabla que se encuentra en un sitio remoto.

```
CREATE SERVER server_cordoba
    TYPE DB2/UDB VERSION '8.1'
    WRAPPER "DRDA"
    AUTHID "PEPE"
    PASSWORD "*******"
   OPTIONS ( DBNAME 'bd_comercial' );

CREATE USER MAPPING FOR USER
    SERVER server_cordoba
    OPTIONS ( REMOTE_AUTHID 'PEPE'
                REMOTE _PASSWORD '**********');

CREATE NICKNAME  ordenes_cordoba FOR .server_cordoba.pepe.ordenes;
```

Motor Oracle

Creación de un sinónimo público con el nombre ordenes_cordoba en la bd_bsas, que apunta a la tabla ordenes de la bd_cordoba, propiedad del usuario admin.

Previamente se deberá crear un enlace de Base de Datos (DBLink) que permita ver la bd_cordoba desde la bd_bsas.

CREATE PUBLIC SYNONYM ordenes_cordoba FOR admin.ordenes@bd_cordoba;

Database Links (Enlaces de Base de Datos) (Oracle)

Un database link es un puntero que define una ruta de comunicaci´n unidirecciónal desde un servidor de base de datos hasta otro. Para acceder al enlace, se debe estar conectado a la base de datos local que contiene la entrada en el diccionario de datos que define dicho puntero.

Hay dos tipos de enlace según la forma en que ocurre la conexión a la base remota:

- Enlace de usuario conectado: el usuario debe tener una cuenta en la base remota con el mismo nombre de usuario de la base local.
- Enlace de usuario fijo: el usuario se conecta usando el nombre de usuari y password referenciados en el enlace.

Ejemplos de Creación de un Database Link

Motor Oracle

Creación de un database link público, llamado “remota”, que hace referencia a la base de datos especificada por el nombre de servicio “ventas”:

```
CREATE PUBLIC DATABASE LINK remota
USING ’ventas’;
```

Uso del database link en una actualización de la tabla remota ordenes para el cliente 200.

```
UPDATE ordenes@remota
SET c_estado = 0
WHERE n_cliente = 200;
```

Creación de un database link privado con usuario fijo admin/admin, a la base de datos especificada por el nombre de servicio “ventas”.

```
CREATE DATABASE LINK remota
CONNECT TO admin IDENTIFIED BY admin
USING ‘ventas’;
```

Directories (Directorios) (Oracle)

Un directorio especifica un alias para un directorio en el file system del servidor, donde se ubican archivos binario externos (BFILES) y datos de tablas externas.
Se pueden usar nombres de directorios al referirse a ellos desde el código, en vez de hardcodear el nombre de ruta de sistema operativo, suministrando por lo tanto una mayor flexibilidad en la administración de archivos.
Los directorios no son propiedad de ningún esquema individual.

Motor Oracle

CREATE DIRECTORY admin AS ’oracle/admin’;

Views (Vistas)

Una view es también llamada una tabla virtual o ventana dinámica.
Una view es un conjunto de columnas, ya sea reales o virtuales, de una misma tabla o no, con algún filtro determinado o no.
De esta forma, es una presentación adaptada de los datos contenidos en una o más tablas, o en otras vistas. Una vista toma la salida resultante de una consulta y la trata como una tabla.
Se pueden usar vistas en la mayoría de las situaciones en las que se pueden usar tablas.

A diferencia de una tabla, una vista no aloca espacio de almacenamiento, ni contiene datos almacenados, sino que está definida por una consulta que extrae u obtiene datos desde las tablas a las que la vista hace referencia.

Las vistas se pueden utilizar para:

- Suministrar un nivel adicional de seguridad restringiendo el acceso a un conjunto predeterminado de filas o columnas de una tabla.
- Ocultar la complejidad de los datos.
- Simplificar sentencias al usuario.
- Presentar los datos desde una perspectiva diferente a la de la tabla base.
- Aislar a las aplicaciones de los cambios en la tabla base.

RESTRICCIONES:

1. No se pueden crear índices en las Views
2. Una view depende de las tablas a las que se haga referencia en ella, si se elimina una tabla todas las views que dependen de ella se borraran o se pasará a estado INVALIDO, dependiendo del motor. Lo mismo para el caso de borrar una view de la cual depende otra view.
3. Algunas views tienen restringido los: Inserts, Deletes, Updates.
    - Aquellas que tengan joins
    - Una función agregada
    - Tener en cuenta ciertas restricciones para el caso de Actualizaciones:
        - Si en la tabla existieran campos que no permiten nulos y en la view no aparecen, los inserts fallarían
        - Si en la view no aparece la primary key los inserts podrían fallar
4. Se puede borrar filas desde una view que tenga una columna virtual.
5. Con la opcion WITH CHECK OPTION, se puede actualizar siempre y cuando el checkeo de la opción en el where sea verdadero.
6. Al crear la view el usuario debe tener permiso de select sobre las columnas de las tablas involucradas.
7. No es posible adicionar a una View las cláusulas de: ORDER BY y UNION.

Ejemplos de Vistas

Motor Informix / DB2 (sentencias similares)

Creación de Vista a partir de un Select de dos tablas.

```
CREATE VIEW V_Ordenes_Pendientes
                (Orden_nro, Fecha_orden, Cliente, Importe_total)
AS
SELECT n_orden, f_orden,  d_apellido || ", " || d_nombre ,  i_total
  FROM ordenes, clientes
 WHERE ordenes.n_cliente = clientes.n_cliente
     AND c_estado = 1;
```

En la vista se concatenaron el apellido y el nombre del cliente en un solo campo, al realizar la siguiente consulta:

```
SELECT * FROM v_ordenes_pendientes
```

Obtendríamos por ejemplo los siguientes resultados:

Orden_nro	Fecha_orden	Cliente	Importe_total
122	15/05/2006	Ledesma, Mario	1234.56
123	15/05/2006	Contempomi, Felipe	2345,78
....			

Motor Oracle (sentencia idéntica pero con comillas simples)

Creación de Vista a partir de un Select de dos tablas.

```
CREATE VIEW V_Ordenes_Pendientes
                (Orden_nro, Fecha_orden, Cliente, Importe_total)
AS
SELECT n_orden, f_orden,  d_apellido || ', ' || d_nombre ,  i_total
  FROM ordenes, clientes
 WHERE ordenes.n_cliente = clientes.n_cliente
     AND c_estado = 1;
```

En la vista se concatenaron el apellido y el nombre del cliente en un solo campo, al realizar la siguiente consulta:

```
SELECT * FROM v_ordenes_pendientes
```

Obtendríamos por ejemplo los siguientes resultados:

Orden_nro	Fecha_orden	Cliente	Importe_total
122	15/05/2006	Ledesma, Mario	1234.56
123	15/05/2006	Contempomi, Felipe	2345,78
....			

Motor SQL Server

Creación de Vista a partir de un Select de dos tablas.

```
CREATE VIEW V_Ordenes_Pendientes

AS
SELECT ordenes.N_orden AS Orden_nro, ordenes.F_orden AS Fecha_orden,
```

```
clientes.d_apellido + ',' + clientes.d_nombre AS Cliente,
ordenes.i_total AS Importe_total
FROM ordenes INNER JOIN
clientes ON ordenes.N_cliente = clientes.n_cliente
WHERE (ordenes.C_estado = 1)
```

```
SELECT * FROM v_ordenes_pendientes
```

Obtendríamos por ejemplo los siguientes resultados:

Orden_nro	Fecha_orden	Cliente	Importe_total
122	15/05/2006	Ledesma, Mario	1234.56
123	15/05/2006	Contempomi, Felipe	2345,78
....			

Indices

Los índices son estructuras opcionales asociadas a una tabla. La función de los índices es la de permitir un acceso más rápido a los datos de una tabla, se pueden crear distintos tipos de índices sobre uno o más campos.
Los índices son lógica y físicamente independientes de los datos en la tabla asociada. Se puede crear o borrar un índice en cualquier momento sin afectar a las tablas base o a otros índices.

TIPOS DE INDICES

- Btree Index
  - Estructura de índice estándar.
- Btree Cluster Index
  - La tabla se ordena igual que el índice.
- Reverse Key Index (Oracle)
  - Invierte los bytes de la clave a indexar. Esto sirve para los índices cuyas claves son una serie constante con por ej. Crecimiento ascendente. para que las inserciones se distribuyan por todas las hojas del árbol de índice.

- Bitmap Index (Oracle)
  - Son utilizados para pocas claves con muchas repeticiones
  - Cada bit en el Bitmap corresponde a una fila en particular.
  - Si el bit esta en on significa que la fila con el correspondiente rowid tiene el valor de la clave.

CARACTERÍSTICAS DIFERENCIADORAS PARA LOS ÍNDICES

1. Unique Índice de clave única.

2. Cluster
   - Este tipo de índice provoca al momento de su creación que físicamente los datos de la tabla sean ordenados por el mismo. (Informix / SQLServer / DB2)

   Variante en ORACLE: un Cluster es un objeto de la base que contiene los datos de una o más tablas, con una o más columnas en común. Oracle almacena juntas todas las filas (de todas las tablas) que comparten la misma clave de cluster. Un índice cluster, crea un índice en el cluster que se especifica.

3. Duplicado Permite múltiples entradas para una misma clave.

4. Compuesto La clave se compone de varias columnas.
   - Las principales funciones de un índice compuesto son:

- Facilitar múltiples joins entre columnas
- Incrementar la unicidad del valor de los índices

Ejemplo de índice compuesto:

customer_nun, lname, fname

Este índice se usará en o para los siguientes casos:

- Joins sobre customer_num, o customer_num y lname o customer_num, lname y fname, es decir, sentencias donde la cláusula WHERE haga referencia a todo o a una porción inicial del índice (las columnas más selectivas o más accedidas, irán al principio del índice).
- Filtros sobre customer_num, o customer_num y lname o customer_num, lname y fname.
- ORDERS BY sobre o customer_num y lname o customer_num, lname y fname.
- Joins sobre customer_num y Filtros sobre lname y fname.
- Joins sobre customer_num y lname y Filtros sobre fname.

5. Function based Index (Oracle): Calcula el valor de la función o la expresión, y lo guarda en el índice.
   Puede crearse tanto como un índice B-Tree como Bitmap.

BENEFICIOS DE LA UTILIZACIÓN DE INDICES:

1. Se le provee al sistema mejor perfomance al equipo ya que no debe hacer lecturas secuenciales sino accede a través de los índices, sólo en los casos que las columnas del Select no formen parte del índice.
2. Mejor perfomance en el ordenamiento de filas
3. Asegura únicos valores para las filas almacenadas
4. Cuando las columnas que intervienen en un JOIN tienen índices se le da mejor perfomance si el sistema logra recuperar los datos a través de ellas

COSTO DE LA UTILIZACIÓN DE INDICES

1. El primer costo asociado es el espacio que ocupa en disco, que en algunos casos suele ser mayor al que ocupan los datos.
2. El segundo costo es el de procesamiento, hay que tener en cuenta que cada vez que una fila es insertada o modificada o borrada, el índice debe estar bloqueado, con lo cual el sistema deberá recorrer y actualizar los distintos índices.

GUIA RESUMEN CUANDO DEBERIAMOS INDEXAR

- Indexar columnas que intervienen en Joins
- Indexar las columnas donde se realizan filtros
- Indexar columnas que son frecuentemente usadas en orders by
- Evitar duplicación de índices
  - ◊ Sobre todo en columnas con pocos valores diferentes Ej: Sexo, Estado Civil, Etc.
- Limitar la cantidad de índices en tablas que son actualizadas frecuentemente
  - ◊ Porque sobre estas tablas se estarán ejecutando Selects extras
- Verificar que el tamaño de índice debería ser pequeño comparado con la fila
  - ◊ Tratar sobre todo en crear índices sobre columnas cuya longitud de atributo sea pequeña

- ◊ No crear índices sobre tablas con poca cantidad de filas, no olvidar que siempre se recupera de a páginas. De esta manera evitaríamos que el sistema lea el árbol de índices
- Tratar de usar índices compuestos para incrementar los valores únicos
  - ◊ Tener en cuenta que si una o más columnas intervienen en un índice compuesto el optimizador podría decidir acceder a través de ese índice aunque sea solo para la búsqueda de los datos de una columna, esto se denomina "*partial key search"*
- Usando cluster index se agiliza la recuperación de filas
  - ◊ Uno de los principales objetivos de la Optimización de bases de datos es reducir la entrada/salida de disco. Reorganizando aquellas tablas que lo necesiten, se obtendría como resultado que las filas serían almacenadas en bloques contiguos, con lo cual facilitaría el acceso y reduciría la cantidad de accesos ya que recuperaría en menos páginas los mismos datos.

CONSTRUCCIÓN DE ÍNDICES EN PARALELO

Los motores de BD usan en general métodos de construcción de índices en paralelo.

El arbol B+ es construido por 2 o más procesos paralelos. Para esto el motor realiza una muestra de la filas a Indexar (aproximadamente 1000) y luego decide como separar en grupos. Luego scanea las filas y las ordena usando el mecanismo de sort en paralelo.

Las claves ordenadas son colocadas en los grupos apropiados para luego ir armarndo en paralelo un subárbol por cada grupo. Al finalizar los subárboles se unen en un único Arbol B+.

Ejemplos de Creación Indices

Motor Informix

- Informix utiliza una estructura de Arbol B+
- Máxima cantidad de campos para la clave de un índice compuesto son 16 y el tamaño de clave máximo permitido es 255 bytes.

Creación de Índice único y simple.

```
CREATE UNIQUE INDEX ix1_ordenes ON ordenes (n_orden);
```

Creación de Indice duplicado y compuesto.

```
CREATE INDEX ix2_ordenes ON ordenes (n_cliente, f_orden);
```

Informix permite con el mismo índice realizar consultas ascendentes o descendentes indistintamente, sin requerir de ninguna cláusula adicional.

Creación de Indice cluster.

```
CREATE CLUSTER INDEX ix3_ordenes ON ordenes (f_orden);
```

El motor ordena los datos de la tabla igual que el índice. Sólo puede haber un índice cluster por tabla.

Manejo del Load Factor

FILLFACTOR – Porcentaje de cada página del índice a ser llenado sólo en el momento de su creación.

Por ej. Si el FILLFACTOR=80%, en la creación del índice se ocupará hasta el 80% de cada nodo.

CREATE INDEX ix4_ordenes ON ordenes (n_cliente) FILLFACTOR 80;

Motor DB2

- DB2 utiliza una estructura de Arbol B+
- Máxima cantidad de campos para la clave de un índice compuesto son 16 y el tamaño de clave máximo permitido es 1024 bytes.

Creación de Índice único y simple, con campos de consulta adicionales incluidos.

```
CREATE UNIQUE INDEX ix1_ordenes
    ON ordenes (n_orden)
    INCLUDE (f_orden, c_cliente);
```

Los campos f_orden y c_cliente no conforman la clave del índice, pero se guardan dentro para acelerar las consultas, obteniendo dicha infomración del índice, sin necesidad de buscarla en la tabla.

Creación de Indice duplicado y compuesto.

```
CREATE INDEX ix2_ordenes
ON ordenes (n_cliente, f_orden)
ALLOW REVERSE SCANS;
```

DB2 requiere de la clausula “ALLOW REVERSE SCANS” para poder usar el índice en consultas descendentes. Esta cláusula convierte al índice en un índice bi-direccional.

```
SELECT * FROM ordenes ORDER BY n_cliente DESC
```

Creación de Indice cluster.

DB2 no posee índices cluster pero permite reorganizar físicamente una tabla en función a un determinado índice.

CREATE INDEX ix3_ordenes ON ordenes (f_orden);

REORG TABLE ordenes INDEX ix3_ordenes;

Manejo del Load Factor

PCTFREE – Porcentaje de cada página del índice a ser dejado como espacio libre en su creación.

Por ej. Si el PCTFREE=20%, en la creación del índice se ocupará hasta el 80% de cada nodo.

CREATE INDEX ix4_ordenes ON ordenes (n_cliente) PCTFREE 20;

Motor ORACLE

- Oracle utiliza una estructura de Arbol B+.
- Máxima cantidad de campos para la clave de un índice compuesto: 32.
- Tamaño máximo de clave permitido: aproximadamente la mitad del espacio disponible en el bloque de datos.

Creación de un índice único y simple:

CREATE UNIQUE INDEX ix1_ordenes ON ordenes (n_orden);

Creación de Indice duplicado y compuesto.

CREATE INDEX ix2_ordenes ON ordenes (n_cliente, f_orden);

Creación de Indice cluster.

Oracle no posee índices cluster o clustered, pero posee el objeto IOT que tiene una estructura interna similar.

Ver Objeto Tabla Organizada por índices (IOT - Index-Organized Tables)

Creación de índice basado en función:

CREATE INDEX upper_ix ON ordenes (UPPER(d_usuario));

Creación de un índice bitmap:

CREATE BITMAP INDEX estado_bm_ix ON ordenes(c_estado)

Manejo del Load Factor

PCTFREE – Porcentaje de cada página del índice a ser dejado como espacio libre en su creación.
Por ej. Si el PCTFREE=20, en la creación del índice se ocupará hasta el 80% de cada nodo.

CREATE INDEX ix4_ordenes ON ordenes (n_cliente) PCTFREE 20;

Motor SQL Server

- Sql Server utiliza una estructura de Arbol B+.
- Máxima cantidad de campos para la clave de un índice compuesto: 16.

Creación de un índice único y simple:

CREATE UNIQUE INDEX ix1_ordenes ON ordenes (n_orden);

Creación de Indice duplicado y compuesto.

CREATE INDEX ix2_ordenes ON ordenes (n_cliente, f_orden);

Creación de Indice cluster.

```
CREATE  CLUSTERED  INDEX ix3_ordenes ON ordenes(N_orden) ON
[PRIMARY];
```

Manejo del Load Factor

FILLFACTOR– Porcentaje de cada página del índice a ser dejado como espacio libre en su creación.
Por ej. Si el FILLFACTOR=20, en la creación del índice se ocupará hasta el 80% de cada nodo.

```
CREATE  UNIQUE  INDEX ix1_ordenes ON ordenes(N_orden)
WITH  FILLFACTOR = 20;
```

Snapshots / Summary Table / Materialized Views

Los snapshots, tambièn llamados vistas materializadas o tablas sumarizadas, son objetos del esquema de una BD que pueden ser usados para sumarizar, precomputar, distribuir o replicar datos. Se utilizan sobre todo en DataWarehouse, sistemas para soporte de toma de decisión, y para computación móvil y/o distribuida.
Consumen espacio de almacenamiento en disco.
Deben ser recalculadas o refrescadas cuando los datos de las tablas master cambian. Pueden ser refrescadas en forma manual, o a intervalos de tiempo definidos dependiendo el motor de BD.

Motor DB2

Creación de una tabla de consulta sumarizada (Summary Table o Materialized Query Table) basada en una consulta de varias tablas. Esta tabla deberá ser refrescada periódicamente para que está actualizada antes de consultarla.

```
CREATE SUMMARY TABLE ST_total_ordenes_clientes
   AS (SELECT A.C_cliente, B.D_apellido, B.D_nombre, SUM(A.I_total) AS
Total_ordenes
         FROM ordenes A, clientes B
        WHERE A.C_cliente = B.C_cliente
        GROUP BY A.C_cliente, B.D_apellido, B.D_nombre )
   DATA INITIALLY DEFERRED
   REFRESH DEFERRED;
```

La cláusula **DATA INITIALY DEFERRED** causa que en la creación de la tabla sumarizada no se realice el INSERT de los datos, eso se realizará en el momento que se ejecute la siguiente instrucción:

```
REFRESH TABLE ST_total_ordenes_clientes
```

La instrucción REFRESH TABLE realiza una actualización de la tabla sumarizada.

Motor Oracle

Creación de una vista materializada basada en una consulta de varias tablas. Esta tabla deberá ser refrescada periódicamente para que esté actualizada

```
CREATE MATERIALIZED VIEW ST_total_ordenes_clientes
BUILD DEFERRED
REFRESH COMPLETE
AS SELECT A.C_cliente, B.D_apellido, B.D_nombre, SUM(A.I_total) AS Total_ordenes
FROM ordenes A, clientes B
WHERE A.C_cliente = B.C_cliente
```

GROUP BY A.C_cliente, B.D_apellido, B.D_nombre ;

La cláusula **BUILD DEFERRED** causa que en la creación de la vista materializada no se realice el INSERT de los datos, que se realizará en el momento que se ejecute el próximo refresco. El valor por defecto es **BUILD IMMEDIATE**, que sí inserta los datos.
La cláusula **REFRESH COMPLETE** indica que el método de refresco será completo, que ejecuta nuevamente la subconsulta. De otra forma, la opción **FAST** le indica actualizar de acuerdo a los cambios que han ocurrido en la tabla master.

Funciones Propias de Motor (Built in Functions)

Estos objetos son funciones ya desarrolladas con el Motor de BD, las cuales pueden clasificarse de la siguiente manera:

Funciones Agregadas

Se aplican a un conjunto de valores derivados de una expresión, actúan específicamente en agrupamientos.

SUM, COUNT, AVG, MAX, MIN, etc.

Funciones Escalares

Se aplican a un dato específico de cada fila de una consulta, o en una comparación en la sección WHERE.

Funciones Algebraicas and Trigonometricas
Funciones Estadisticas
Funciones de Fecha
Funciones de Strings
Otras

Funciones de Tablas (algunos motores)

Retornan el equivalente a una tabla y pueden ser usadas solamente en la sección FROM de una sentencia.
ORA: Se denominan “Vista en línea (inline view)”

Funciones de Usuario

Una función es un objeto de la base de datos que puede recibir uno o más parámetros de input y devolver sólo un parámetro de output.

Motor Informix

No posee funciones, pero permite ejecutar los Stored Procedures como si fuesen funciones, dentro de la sentencia Select.

```
CREATE PROCEDURE nombre_dpto (p_c_empleado INT)
        RETURNING VARCHAR(30);

        DEFINE v_error varchar(70);
        DEFINE v_nombre_dpto varchar (30);


        SELECT d_dpto
            INTO v_nombre_dpto
          FROM departamentos d, empleados e
        WHERE d.c_dpto = e.c_dpto
             AND e.c_empleado = p_c_empleado ;


        IF v_nombre_dpto IS NULL THEN
                SET v_error = 'Error: empleado ' || p_c_empleado || ' no se ha encontrado';

                RAISE EXCEPTION -736, 0,  v_error;
        END IF;

        RETURN v_nombre_dpto

END PROCEDURE;
```

Invocación de stored procedure en un Select en la sección WHERE

```
SELECT *
  FROM departamentos d
WHERE d.d_dpto = nombre_dpto(6010);
```

Invocación de stored procedure en un Select en la sección Lista de Columnas

```
SELECT e.c_empleado, e.d_nombre, e.d_apellido, nombre_dpto(e.c_dpto)
depto_nombre
FROM empleados e
WHERE e.c_empleado = 6010;
```

Motor DB2

Creación de la función nombre_dpto

```
CREATE FUNCTION nombre_dpto (p_c_empleado INTEGER) )
RETURNS VARCHAR(30)
SPECIFIC  departamentos
BEGIN ATOMIC
        DECLARE v_nombre_dpto VARCHAR(30);
        DECLARE v_error VARCHAR(70);
```

```
        SET v_nombre_dpto = (
                        SELECT d_dpto FROM departamentos d, empleados e
                        WHERE d.c_dpto = e.c_dpto
                            AND e.c_empleado = p_c_empleado );


        IF v_nombre_dpto IS NULL THEN
           SET v_error = 'Error: empleado ' || p_c_empleado || ' no se ha encontrado';

           SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT=v_error;
        END IF;
        RETURN v_nombre_dpto
END
```

Invocación de función en un Select en la sección WHERE

```
        SELECT *
          FROM departamentos d
        WHERE d.d_dpto = nombre_dpto(6010);
```

Invocación de función en un Select en la sección Lista de Columnas

```
        SELECT e.c_empleado, e.d_nombre, e.d_apellido, nombre_dpto(e.c_dpto)
        depto_nombre
        FROM empleados e
        WHERE e.c_empleado = 6010;
```

Motor Oracle

```
CREATE FUNCTION nombre_dpto (p_c_empleado NUMBER)
RETURN departamentos.d_depto%TYPE;
DECLARE
        v_nombre_dpto           departamentos.d_depto%TYPE;
        v_error             VARCHAR2(70);
BEGIN
        SELECT d_dpto INTO v_nombre_dpto
        FROM departamentos d, empleados e
        WHERE d.c_dpto = e.c_dpto
        AND e.c_empleado = p_c_empleado;

RETURN v_nombre_dpto;

EXCEPTION
        WHEN NO_DATA_FOUND THEN
                v_error := 'Error: empleado ' || p_c_empleado || ' no se ha
        encontrado';
                raise_application_error(-20101, v_error);

END nombre_dpto;
```

Invocación de función en un Select en la sección WHERE

```
        SELECT *
        FROM departamentos d
        WHERE d.d_dpto = nombre_dpto(6010);
```

Invocación de función en un Select en la sección Lista de Columnas

```
SELECT e.c_empleado, e.d_nombre, e.d_apellido, nombre_dpto(e.c_dpto)
depto_nombre
FROM empleados e
WHERE e.c_empleado = 6010;
```

Motor SQL Server

```
CREATE FUNCTION nombre_dpto
  (@p_c_empleado INT)
RETURNS varchar(30)
AS

BEGIN

RETURN (SELECT d_dpto FROM departamentos d, empleados e
WHERE d.c_dpto = e.c_dpto
AND e.c_empleado = p_c_empleado)

END
```

Invocación de función en un Select en la sección WHERE

```
SELECT *
FROM departamentos d
WHERE d.d_dpto = nombre_dpto(6010);
```

Invocación de función en un Select en la sección Lista de Columnas

```
SELECT e.c_empleado, e.d_nombre, e.d_apellido,
        nombre_dpto(e.c_dpto) depto_nombre
FROM empleados e
WHERE e.c_empleado = 6010;
```

Stored Procedures

QUE ES UN STORED PROCEDURE?

Es un procedimiento almacenado como un objeto en la Base de Datos.

Características:

1. Incluyen sentencias de SQL y sentencias de lenguaje propias. Lenguaje SPL (Informix), PL/SQL (Oracle), TRANSAC/SQL (SQL Server).
2. Son almacenados en la base de Datos
   - Solo las sentencias del lenguaje pr son permitidas además de las de SQL
   - Algunos motores permiten además Stored Procedures en JAVA.
3. Se guarda la sentencia SQL ya parseada y optimizada
   - Antes de ser almacenada en la base de datos las sentencias SQL son parseadas y optimizadas. Cuando el stored procedure es ejecutado puede que no sea necesario su optimización, en caso contrario se optimiza la sentencia antes de ejecutarse.

VENTAJAS DE LOS STORED PROCEDURES:

- Pueden reducir la complejidad en la programación. Creando SP con las funciones + usadas.
- Pueden ganar perfomance en algunos casos
- Otorgan un nivel de seguridad extra
- Pueden definirse ciertas reglas de negocio independientemente de las aplicaciones.
- Diferentes aplicaciones acceden al mismo código ya compilado y optimizado.
- En un ambiente cliente servidor, no sería necesario tener distribuido el código de la aplicación
- En proyectos donde el código puede ser ejecutados desde diferentes interfaces, Ud. mantiene un solo tipo de código.
- Menor tráfico en el PIPE / SOCKET, no en la cantidad de bytes que viajan sino en los ciclos que debo ejecutar una instrucción.

Ej.:

Si este proceso se ejecuta desde una workstation en red, por la red viajarán 1000 instrucciones Update/Delete, x cada una el motor tendrá que Parsearla y Optimizarla. El costo es muy alto.

```
FOR

        SELECT ....

        IF status
                UPDATE
                ELSE
                DELETE
        END IF
END FOR
```

Sería mejor hacer desde la aplicación cliente servidor , ejecutar un stored procedure,

EXECUTE PROCEDURE PEPE(var,var) 1 sola vez

en este caso por la red viajará sólo 1 instrucción (execute procedure...), el motor no parseará las instrucciones porque ya están parseadas dentro del Procedure, y las optimizará en caso de que sea necesario.

Ejemplos de Creación y ejecución de Stored Procedures

Motor Informix

Creación de un Stored Procedure que inserta una orden de pedido a partir de los datos existentes en varias tablas termporales manejando una transacción y creación de un procedure de grabación de log de errores.

```
CREATE PROCEDURE sp_inserta_orden (V_n_orden INT)
RETURNING SMALLINT;

        DEFINE sql_err int;
        DEFINE isam_err int;
        DEFINE error_info char(70);

        ON EXCEPTION SET sql_err, isam_err, error_info
           ROLLBACK WORK;
           CALL error_log (sql_err, isam_err, error_info,
                           V_n_orden, "sp_inserta_orden");
           RAISE EXCEPTION sql_err, isam_err, error_info;
        END EXCEPTION;

        BEGIN WORK;
                INSERT INTO ordenes
                        SELECT * FROM ordenes_tmp
                        WHERE n_orden = V_n_orden;

                INSERT INTO items_ordenes
                        SELECT * FROM items_tmp
                        WHERE n_orden = V_n _orden;

        COMMIT WORK;
        RETURN 0;
END PROCEDURE;

CREATE PROCEDURE sp_error_log (sql_err int,
                               isam_err int,
                          error_info char(70),
                          nro_orden INT,
                          proc_name char(18) )

        INSERT INTO error_logs
        VALUES (proc_name, v_nro_orden, sql_err, isam_err,
                    error_info, USER, CURRENT);
END PROCEDURE;
```

Si la alguna de las instrucciones de la transacción falla, tomará el control la sección del ON EXCEPTION y en dicho caso realiza un ROLLBACK de la transacción y ejecuta el procedure sp_error_log y devuelve el código y la descripción del error a la aplicación finalizando luego el stored procedure.
En el caso que la transacción se realice en forma correcta retornará un cero a la aplicación.

Ejecución del Stored Procedure desde SQL de forma Online

EXECUTE PROCEDURE sp_inserta_odenes (1234)

Ejecutará el procedure insertando la orden Nro. 1234. En el ejemplo se asume que las ordenes a pasar como parámetro existen y fueron validadas por otro programa.

Motor SQL Server

Creación de un Stored Procedure que inserta una orden de pedido a partir de los datos existentes en varias tablas termporales manejando una transacción y creación de un procedure de grabación de log de errores.

```
CREATE PROCEDURE dbo.sp_inserta_orden
@v_n_orden INT

AS

SET NOCOUNT ON

DECLARE
        @nError int,
        @error_info char(30)

        SET @nError = 0

        BEGIN TRANSACTION

        INSERT INTO ordenes
        SELECT * FROM ordenes_tmp
        WHERE n_orden = @v_n_orden

        INSERT INTO item_ordenes
        SELECT * FROM items_tmp
        WHERE n_orden = @v_n_orden

        set @nError = @@error
        IF( @nError <> 0 )
        BEGIN
                ROLLBACK
                SELECT @nError AS nError, @error_info AS Descripcion
                exec sp_error_log  @nError, @error_info, @v_n_orden, 'sp_inserta_orden'
        END
        ELSE
        BEGIN
                COMMIT
                SELECT 0 AS nError ,'Se insertó bien' AS Descripcion
        END
GO


CREATE PROCEDURE dbo.sp_error_log
@sql_err int,
@error_info char(70),
@v_nro_orden INT,
@proc_name char(18)

AS
```

```
SET NOCOUNT ON

INSERT INTO error_logs
VALUES (@proc_name, @v_nro_orden, @sql_err,
@error_info, USER, getdate())
GO
```

Ejecución del Stored Procedure desde SQL de forma Online

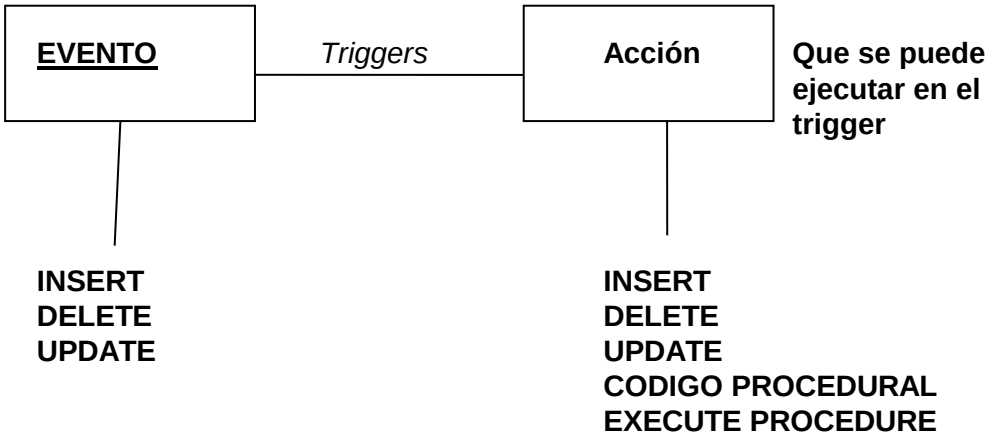
EXEC sp_inserta_oden 1234

Ejecutará el procedure insertando la orden Nro. 1234. En el ejemplo se asume que las ordenes a pasar como parámetro existen y fueron validadas por otro programa.

Triggers

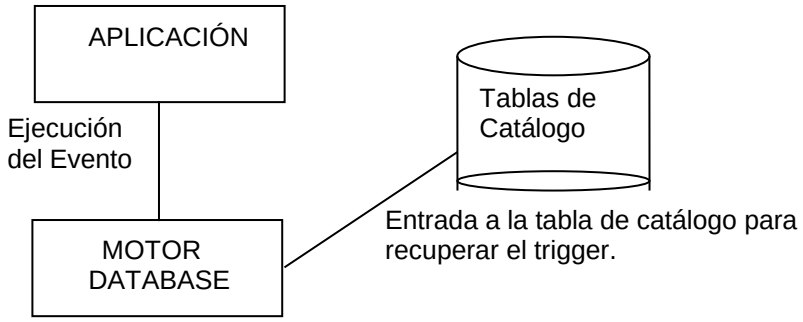
QUE ES UN TRIGGER?

Es un mecanismo que ejecuta una sentencia de SQL automáticamente cuando cierto evento ocurre.



1. Cuando el evento ocurre sobre una tabla, se dispara la acción.
2. Esta ejecución es independiente de la aplicación que la invoca
3. Solo en el evento UPDATE puedo hacer referencia a una columna .

Como un trigger es ejecutado



La acción del trigger es recuperada, optimizado y ejecutado.

PORQUE USAR TRIGGERS?

- Se pueden aplicar las reglas de negocio
  Por Ej.: Si el inventario de una columna pasa x valor, entonces insertar un pedido en la tabla de compras
- Valores de columnas derivadas
  En algunos casos es necesario que ante tal acción se deriven datos en base a otros.
- Replicación automática de tablas

- Logs. De Auditoría
- Delete en Cascada
- Autorización de Seguridad
  Ej. permitir solo a ciertos usuarios a emitir facturas de + de 1000

Eventos:

Instrucciones DML sobre Tablas o Views

INSERT ON tab_name
DELETE FROM tab_name
UPDATE tab_name
UPDATE of col_name ON tab_name

SELECT tab_name (Informix)
SELECT of col_name ON tab_name (Informix)

Instrucciones DDL sobre Base de Datos (Oracle/Sql Server)

CREATE
ALTER
DROP

Operaciones de Base de Datos (Oracle/Sql Server)

SERVERERROR
LOGON
LOGOFF
STARTUP
SHUTDOWN

En algunos motores se permiten múltiples triggers sobre una tabla, pero sólo 1 por tipo. Para el caso de UPDATE las columnas deben ser mutuamente exclusivas.
La tabla especificada por el trigger debe estar en modo local, en general los motores no aceptan tablas en servers remotos.

Acciones de triggers

Pueden ser sentencias SQL, 1 o varias, o Stored Procedures. En caso de Oracle se puede ejecutar directamente código PL/SQL.

Se pueden ejecutar en distintos momentos:

- BEFORE (execute procedure xyz()) → Se ejecuta antes de que el evento de trigger ocurra (Oracle / Informix)
- AFTER (execute procedure xyz()) → Se ejecuta despues de que el evento de trigger ocurra
- FOR EACH ROW (execute procedure xyz()) → Se ejecuta para cada una de las filas del evento. (existen diferencias de uso entre motores) (Oracle / Informix)
- INSTEAD OF (Sql Server / Oracle/Informix) → Se ejecuta en lugar del evento de trigger recibido. Reemplaza el evento que disparó el trigger por la acción del trigger.

  En Oracle/Informix sólo pueden ser utilizados en Triggers sobre VIEWS.
  En SQLServer pueden ser utilizados tanto en Triggers sobre TABLAS como sobre VIEWS.

Transaccionabilidad entre Evento y Acción

- Para BD sin logs transaccional No ocurre el Rollback.
- En base de datos con log, el evento y la acción del trigger se les hace un roll back automático si alguno falla.

Ejemplos de Creación de Triggers

Motor Informix

Tablas/Vistas del Diccionario de Datos asociadas

- SYSTRIGGERS
- SYSTRIGBODY

Ejemplos

Creación de un trigger de Update con acciones ANTES, DURANTE y DESPUES del evento INSERT. En este ejemplo el trigger ejecuta Stored Procedures.

```
CREATE TRIGGER test1 UPDATE ON ordenes
BEFORE(EXECUTE PROCEDURE sp_check_permisos())
FOR EACH ROW (EXECUTE PROCEDURE log_chg())
AFTER (EXECUTE PROCEDURE log_total());
```

Ante la ejecución de la siguiente instrucción:

```
UPDATE ORDENES SET C_ESTADO = 2
WHERE n_cliente = 106 and c_estado = 1;
```

Suponiendo que hayan sólo 2 filas de la tabla Ordenes que cumplan con el WHERE del UPDATE, el trigger ejecutará las acciones de la siguiente forma:

check_permission Acción ejecutada por el trigger antes del primer UPDATE
UPDATE de la primera fila
log_chg Acción ejecutada por el trigger por cada fila modificada
UPDATE de la segunda fila
log_chg Acción ejecutada por el trigger por cada fila modificada
log_total Acción ejecutada por el trigger luego de finalizar el UPDATE

Creación de un trigger de Insert con acciones múltiples que se ejecutan por cada fila que se inserte durante el evento de INSERT. Borra en tabla temporal la cabecera de la orden y graba una fila de auditoría.

```
CREATE TRIGGER test1 INSERT ON ordenes
```

```
REFERENCING NEW AS nuevo
FOR EACH ROW(DELETE FROM ordenes_tmp
WHERE n_orden = nuevo.n_orden,
INSERT INTO audit_ordenes VALUES
(nuevo.n_orden, USER, CURRENT) );
```

Creación de un trigger de DELETE para borrado en cascada de los items de la orden de pedido antes de borrar la orden de pedido.

```
CREATE TRIGGER del_ordenes DELETE ON ordenes
REFERENCING OLD AS oden_del
BEFORE ( DELETE FROM items_ordenes
WHERE n_orden = orden_del.n_orden  );
```

Creación de un trigger de UPDATE que modifica el campo i_total de la tabla Ordenes ante un UPDATE del campo q_ en la tabla Items_ordenes.

```
CREATE TRIGGER upd_items UPDATE OF Q_cantidad ON items
REFERENCING OLD AS vie NEW AS nuev
FOR EACH ROW    (
                UPDATE ordenes SET i_total=
                        (i_total – (viej.q_cantidad * viej.i_precunit)
                            + (nuev.q_cantidad * nuev.i_precunit)
                        )
                WHERE n_orden = nuev.n_orden
                );
```

Motor Oracle

Tablas/Vistas del Diccionario de Datos asociadas

- USER_TRIGGERS
- ALL_TRIGGERS
- DBA_TRIGGERS

Ejemplos

Creación de trigger de Update con acciones ANTES, DURANTE y DESPUES del evento INSERT. En este ejemplo los triggers ejecutan Stored Procedures.

```
CREATE TRIGGER test1
BEFORE UPDATE ON ordenes
BEGIN
        sp_check_permisos();
END;
```

```
CREATE TRIGGER test2
AFTER UPDATE ON ordenes
FOR EACH ROW
BEGIN
        log_chg();
END;

CREATE TRIGGER test3
AFTER UPDATE ON ordenes
BEGIN
        log_total();
END;
```

Ante la ejecución de la siguiente instrucción:

```
UPDATE ORDENES SET C_ESTADO = 2
WHERE n_cliente = 106 and c_estado = 1;
```

Suponiendo que hayan sólo 2 filas de la tabla Ordenes que cumplan con el WHERE del UPDATE, los trigger ejecutarán las acciones de la siguiente forma:

check_permission Acción ejecutada por el trigger antes del primer UPDATE
UPDATE de la primera fila
log_chg Acción ejecutada por el trigger por cada fila modificada
UPDATE de la segunda fila
log_chg Acción ejecutada por el trigger por cada fila modificada

log_total Acción ejecutada por el trigger luego de finalizar el UPDATE

Creación de un trigger de Insert con acciones múltiples que se ejecutan por cada fila que se inserte durante el evento de INSERT. Borra en tabla temporal la cabecera de la orden y graba una fila de auditoría.

```
CREATE TRIGGER test1
AFTER INSERT ON ordenes
REFERENCING NEW AS nuevo
FOR EACH ROW
BEGIN
    DELETE FROM ordenes_tmp
    WHERE n_orden = :nuevo.n_orden;

    INSERT INTO audit_ordenes VALUES
    (:nuevo.n_orden, USER, CURRENT);
END;
```

Creación de un trigger de DELETE para borrado en cascada de los items de la orden de pedido antes de borrar la orden de pedido.

```
CREATE TRIGGER del_ordenes
BEFORE DELETE ON ordenes
REFERENCING OLD AS oden_del
FOR EACH ROW
BEGIN
   DELETE FROM items_ordenes
        WHERE n_orden = :orden_del.n_orden;
END;
```

Creación de un trigger de UPDATE que modifica el campo i_total de la tabla Ordenes ante un UPDATE del campo q_ en la tabla Items_ordenes.

```
CREATE TRIGGER upd_items
AFTER UPDATE OF Q_cantidad ON items
REFERENCING OLD AS vie NEW AS nuev
FOR EACH ROW
BEGIN
                    UPDATE ordenes SET i_total=
                     (i_total – (:viej.q_cantidad * :viej.i_precunit)
                              + (:nuev.q_cantidad * :nuev.i_precunit)
                     )
                    WHERE n_orden = nuev.n_orden;
END;
```

Motor Sql Server

Tablas/Vistas del Diccionario de Datos asociadas

- sys.triggers
- sys.trigger_events
- sys.sql_modules (triggers, vistas, storedProcedures)
- sys.all_objects (tablas y todos los objetos)

Ejemplos

Creación de trigger de Update con acciones ANTES, DURANTE y DESPUES del evento INSERT.
En este ejemplo los triggers ejecutan Stored Procedures.

```
CREATE TRIGGER test1
ON state
INSTEAD OF UPDATE
AS
BEGIN
  EXEC check_permisos;
  DECLARE TrigUpdCursor CURSOR FOR
  SELECT n_cliente,c_estado,n_orden,f_orden FROM inserted
  DECLARE @n_cliente int, @c_estado smallint, @n_orden int,
          @f_orden datetime

 OPEN TrigUpdCursor;

 FETCH NEXT FROM TrigUpdCursor
       INTO @n_cliente, @c_estado, @n_orden, @f_orden
 WHILE @@FETCH_STATUS = 0
 BEGIN

   UPDATE ordenes SET n_cliente=@n_cliente,
                      c_estado = @c_estado,
                      n_orden = @n_orden,
                      f_orden = @f_orden
   WHERE n_orden = @n_orden;

   EXEC log_chg;

   FETCH NEXT FROM TrigUpdCursor
         INTO @n_cliente, @c_estado, @n_orden, @f_orden
 END;


 CLOSE TrigUpdCursor;
 DEALLOCATE TrigUpdCursor;

 EXEC log_total;
END;
GO
```

Ante la ejecución de la siguiente instrucción:

UPDATE ORDENES SET C_ESTADO = 2
WHERE n_cliente = 106 and c_estado = 1;

Suponiendo que hayan sólo 2 filas de la tabla Ordenes que cumplan con el WHERE del UPDATE, los trigger ejecutarán las acciones de la siguiente forma:

check_permisos Acción ejecutada por el trigger antes del primer UPDATE

UPDATE de la primera fila
log_chg Acción ejecutada por el trigger por cada fila modificada
UPDATE de la segunda fila
log_chg Acción ejecutada por el trigger por cada fila modificada

log_total Acción ejecutada por el trigger luego de finalizar el UPDATE

Creación de un trigger de Insert con acciones múltiples que se ejecutan por cada fila que se inserte durante el evento de INSERT. Borra en tabla temporal la cabecera de la orden y graba una fila de auditoría.

```
CREATE TRIGGER test2
ON ordenes
INSTEAD OF INSERT
AS
BEGIN

  DECLARE TrigInsCur CURSOR FOR
  SELECT * FROM inserted
  DECLARE @n_cliente, @c_estado, @n_orden, @f_orden

 OPEN TrigInsCur;

 FETCH NEXT FROM TrigInsCur
         INTO @n_cliente, @c_estado, @n_orden, @f_orden

 WHILE @@FETCH_STATUS = 0
 BEGIN

   INSERT INTO ordentes  (n_cliente, c_estado, n_orden, f_orden)
    VALUES (@n_cliente, @c_estado, @n_orden, @f_orden);

    DELETE FROM ordenes_tmp
    WHERE n_orden = @n_orden;
```

```
    INSERT INTO audit_ordenes
    VALUES
    (@n_orden, USER_NAME(), GETDATE());

   FETCH NEXT FROM TrigInsCur
            INTO @n_cliente, @c_estado, @n_orden, @f_orden

  END;

  CLOSE TrigInsCur;
  DEALLOCATE TrigInsCur;
END;
GO
```

Creación de un trigger de DELETE para borrado en cascada de los items de la orden de pedido antes de borrar la orden de pedido.

```
CREATE TRIGGER del_ordenes
ON ordenes
INSTEAD OF DELETE
AS
BEGIN

  DECLARE TrigDelCur CURSOR FOR
  SELECT * FROM deleted
  DECLARE @n_cliente, @c_estado, @n_orden, @f_orden

 OPEN TrigDelCur;

 FETCH NEXT FROM TrigDelCur
         INTO @n_cliente, @c_estado, @n_orden, @f_orden

 WHILE @@FETCH_STATUS = 0
 BEGIN

    DELETE FROM items_ordenes
      WHERE n_orden = @n_orden;

    DELETE FROM ordenes
      WHERE n_orden = @n_orden;

   FETCH NEXT FROM TrigDelCur
            INTO @n_cliente, @c_estado, @n_orden, @f_orden

 END;

 CLOSE TrigDelCur;
```

```
  DEALLOCATE TrigDelCur;
END;
GO
```

Creación de un trigger de UPDATE que modifica el campo i_total de la tabla Ordenes ante un UPDATE del campo q_ en la tabla Items_ordenes.

```
CREATE TRIGGER upd_items
ON items
AFTER UPDATE
AS
BEGIN
DECLARE @q_cant_del int, @i_prec_del  dec(8,2), @n_orden int,
            @q_cant_ins int, @i_prec_ins  dec(8,2);

SELECT q_cantidad, i_precunit INTO @q_cant_del , @i_prec_del
FROM deleted;
SELECT n_orden, q_cantidad, i_precunit INTO @n_orden, @q_cant_ins,
@i_prec_ins FROM inserted;

IF UPDATE (q_cantidad) OR UPDATE(i_precunit)
BEGIN
            UPDATE ordenes  SET i_total=
             (i_total - (@q_cant_del * @i_prec_del)
                      + (@q_cant_ins * @i_prec_ins))
                       WHERE n_orden = @n_orden;
END
GO
```

Packages (Oracle / DB2)

Un Package (paquete) es un objeto que agrupa tipos y subprogramas relacionads lógicamente. Habitualmente tienen dos partes: una especificación y un cuerpo. La especificación es la interfaz para las aplicaciones: declara los tipos, variables, constantes, excepciones, cursores y subprogramas disponibles para su uso. El cuerpo define en forma completa los cursores y subprogramas, y por lo tanto implementa la especificación.

Los paquetes suministran varias ventajas: modularidad, facilidad en el diseño de la aplicación, ocultamiento de información y mejora en el rendimiento.

Esquema (Oracle)

Es el conjunto de objetos de los que es dueño un usuario, y tiene el mismo nombre que el usuario. Cada usuario posee un solo esquema. El esquema lo crea Oracle al crearse el usuario.

DTS (SqlServer) (Deprecado)

Los Servicios de transformación de datos (DTS) solo existen en el motor SQL Server y proporcionan un conjunto de herramientas que permiten extraer, transformar y consolidar datos de distintos orígenes (Tablas de base de datos, excel, Archivos de Texto, Access, etc) en uno o varios destinos

compatibles con la conectividad DTS. Para crear soluciones de transferencia de datos personalizadas se deben generar gráficamente los paquetes.

- Importación y exportación de datos.

DTS puede importar datos de un archivo de texto o de un origen de datos OLE DB (por ejemplo, una tabla de Microsoft Access 2000) en SQL Server. De forma alternativa, puede exportar datos desde SQL Server a un destino de datos OLE DB (por ejemplo, una hoja de cálculo Microsoft Excel 2000). DTS también permite la carga de datos de alta velocidad desde archivos de texto a tablas de SQL Server.

- Transformación de datos.

El Diseñador DTS incluye la tarea Transformar datos, que permite seleccionar datos de una conexión de origen de datos, asignar las columnas de datos a un conjunto de transformaciones y enviar los datos transformados a una conexión de destino. El Diseñador DTS también contiene una tarea de consulta controlada por datos que permite asignar datos a consultas parametrizadas.

- Copia de objetos de base de datos.

Con DTS, puede transferir, además de los datos, índices, vistas, inicios de sesión, procedimientos almacenados, desencadenadores, reglas, valores predeterminados, restricciones y tipos de datos definidos por el usuario. Además, puede generar secuencias de comandos para copiar los objetos de base de datos.