ANTES DE LA CLASE UNA REFLEXIÓN: EL PERRO FIEL.

Una pareja de jóvenes tenía varios años de casados y no podían tener hijos. Para no sentirse solos, compraron un cachorro pastor alemán y lo amaron como si fuera su propio hijo. El cachorro creció hasta convertirse en un grande y hermoso pastor alemán.

El perro salvó, en más de una ocasión, a la pareja de ser atacada por ladrones. Siempre fue muy fiel, quería y defendía a sus dueños contra cualquier peligro.

Luego de siete años de tener al perro, la pareja logro tener el hijo tan ansiado. La pareja estaba muy contenta con su hijo y disminuyeron las atenciones que tenían con el perro. Éste se sintió relegado y comenzó a sentir celos del bebe y no era el perro cariñoso y fiel que tuvieron durante siete años.

Un día la pareja dejó al bebé plácidamente durmiendo en la cuna y fueron a la terraza a preparar una carne asada. Cuál fue su sorpresa cuando se dirigían al cuarto del bebé, ven al perro en el pasillo con la boca ensangrentada y moviéndoles la cola.

El dueño del perro pensó lo peor, saco un arma que llevaba y en el acto mató al perro. Corre al cuarto del bebé y encuentra totalmente degollada a una gran serpiente. El dueño comienza a llorar y exclama: ¡He matado a mi perro fiel!

¿Cuántas veces no hemos juzgado injustamente a las personas? Lo que es peor, las juzgamos y condenamos sin investigar a que se debe su comportamiento, cuáles son sus pensamientos y sentimientos. Muchas veces las cosas no son tan malas como parecen, sino todo lo contrario.

La próxima vez que nos sintamos tentados a juzgar y condenar a alguien recordemos la historia del perro fiel, así aprenderemos a no levantar falsos contra una persona hasta el punto de dañar su imagen ante los demás.

USO DE CONSTRAINTS (Restricciones).

Por lo general, en el ambiente académico se le enseña al Estudiante que debe validar los Datos que ingresan a las Tablas utilizando sus programas, los cuales son escritos en Lenguajes como Java, Visual Basic, Visual Fox u otros. Por ello, en muchos casos, el individuo que aprende SQL, suele mantener el Paradigma de que no puede validar entradas de Datos en este último lenguaje.

Con regularidad, la entrada de Datos directa por SQL presenta algunas anomalías como las siguientes:

- Se aceptan valores Nulos en Filas (Campos) que deben ser obligatorios. Por ejemplo, una empresa puede fijar como política que cada vez que se ingrese un cliente debería llevar obligatoriamente su Límite de Crédito, pero, el DBA puede cometer el error de confiar en que los programadores validarán que el Campo Nombre en el Formulario de entrada (Programa en Java o Visual Basic) no se dejará en blanco. Pero, que pasa si uno de los mismos Programadores debe ingresar un Cliente por SQL a la Base de Datos para resolver un problema puntual y no coloca el Límite de Crédito???. Para que esto suceda, el programador puede ejecutar una instrucción como la que sigue:

```
SQL> insert into clientes(codigo, nombre, estatus)
  2  values ('0075','BANCOR','A');

1 row created.
```

En este caso, se ha creado un cliente sin Límite de Crédito, lo cual contradice la política interna de la empresa de ejemplo.

Para evitar que esto vuelva a suceder, se puede utilizar un **NOT NULL CONSTRAINT.**

Como ejemplo se va a crear nuevamente la Tabla de clientes pero se le va a colocar el nombre clientesv (La v para representar que está validada, pero no es obligatoria, solo como ejemplo).

```
SQL> create table clientesv(
  2  codigo varchar2(4),
  3  nombre varchar2(20),
  4  limcre number(10,2) NOT NULL,
  5  estatus varchar2(1));

Table created.
```

Luego si se intenta ingresar un Cliente en esta tabla sin el Límite de Crédito:

```
SQL> insert into clientesv(codigo, nombre, estatus)
  2  values ('0071','ACME SISTEMAS','A');
insert into clientesv(codigo, nombre, estatus)
*
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SCOTT"."CLIENTESV"."LIMCRE")
```

Se emitirá un mensaje de ERROR porque no se pueden ingresar valores NULOS en la columna (Campo) LIMCRE.

- También puede suceder que **en SQL se acepten claves repetidas**. Para evitar esta situación, se utiliza el **PRIMARY KEY CONTSTRAINT,** el cual debe ser especificado cuando se realiza la creación de la Base de Datos.

Para ejemplificar esta situación, se eliminará la Tabla clientesv y se volverá a crear, pero ahora con un PRIMARY KEY CONSTRAINT.

```
SQL> drop table clientesv;

Table dropped.
```

```
SQL> create table clientesv(
  2  codigo varchar2(4) NOT NULL,
  3  nombre varchar2(20) NOT NULL,
  4  limcre number(10,2) NOT NULL,
  5  numfax varchar2(15),
  6  estatus varchar2(1) NOT NULL,
  7  CONSTRAINT clientesv_pk PRIMARY KEY (CODIGO));

Table created.
```

Hay que resaltar varias cosas en este punto:

- Se le colocó NOT NULL a todas las columnas (Campos) obligatorias. La columna numfax (Número de Fax) no es obligatoria, porque puede suceder que un cliente no tenga FAX.
- Clientesv_pk es el nombre del CONSTRAINT y puede ser como el diseñador quiera. Se pueda llamar restriccion01 por ejemplo, o se puede llamar UCLA01. En este caso se le agregaron las letras pk para

ejemplificar que es un PRIMARY KEY pero estas letras no son obligatorias.

- El Campo que representa la Clave primaria se coloca entre paréntesis.

Para probar como funciona este CONSTRAINT, se va a incluir el cliente ACME Sistemas.

```
SQL> insert into clientesv(codigo,nombre,limcre,estatus)
  2  values ('0051','ACME Sistemas',1500,'A');

1 row created.
```

Luego al intentar ingresar ese cliente de nuevo:

```
SQL>  insert into clientesv(codigo,nombre,limcre,estatus)
  2   values ('0051','ACME Sistemas',1500,'A');
 insert into clientesv(codigo,nombre,limcre,estatus)
*
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.CLIENTESV_PK) violated
```

Emite un Mensaje de ERROR porque se ha intentado violar la Restricción denominada CLIENTESV_PK.

- Otro hecho inadecuado que puede ocurrir en la Base de datos, consiste en **Registrar un mismo cliente varias veces con diferentes códigos**. Para visualizar este caso, se incluirá el cliente ACME Sistemas en la Tabla Clientesv pero ahora con otro código.

```
SQL> insert into clientesv(codigo,nombre,limcre,estatus)
  2  values ('0071','ACME Sistemas',1500,'A');

1 row created.
```

En este caso la validación de Clave Primaria no influyó porque se incluyó nuevamente el cliente, pero con otro código, como se nota seguidamente:

```
SQL> select * from clientesv;

CODI NOMBRE                    LIMCRE NUMFAX          E
---- -------------------- ---------- --------------- -
0051 ACME Sistemas              1500                  A
0071 ACME Sistemas              1500                  A
```

Esta situación también se puede validar, al no permitir que ciertos campos como el nombre se repitan. Para ello, se puede utilizar un UNIQUE Key CONSTRAINT.

Para ejemplificarlo, es necesario eliminar nuevamente la tabla Clientesv y volverla a crear con UNIQUE Key CONSTRAINT.

```
SQL> create table clientesv(
  2  codigo varchar2(4) NOT NULL,
  3  nombre varchar2(20) NOT NULL,
  4  limcre number(10,2) NOT NULL,
  5  numfax varchar2(15),
  6  estatus varchar2(1) NOT NULL,
  7  CONSTRAINT clientesv_pk PRIMARY KEY (CODigo),
  8  CONSTRAINT clientesv_uk UNIQUE (nombre));

Table created.
```

Una vez creada la tabla se procede de nuevo a Incluir el Cliente ACME Sistemas.

```
SQL>  insert into clientesv
  2   values ('0051','ACME Sistemas',1500,'0251-2555555','A');

1 row created.
```

Seguidamente, se tratará de incluir el mismo cliente, pero con otro código:

```
SQL> insert into clientesv
  2  values ('0071','ACME Sistemas',1500,'0251-2555555','A');
insert into clientesv
*
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.CLIENTESV_UK) violated
```

HABLANDO DE INTEGRIDAD REFERENCIAL.

Antes de hablar del próximo CONSTRAINT, se debe examinar el concepto de Integridad Referencial.

Según este concepto, cuando se vaya a eliminar un Cliente (Por ejemplo) es necesaria validar que el mismo no se encuentre referenciado en otras tablas. Un caso práctico, pudiera estar referenciado por un pequeño Sistema de facturación. Mientras a un cliente no se le hayan elaborado Facturas puede ser eliminado, caso contrario no se puede porque el código de Cliente se encuentra refernciado como una Clave Foránea.

Para ejemplificar esta situación, se va a crear una Tabla denominada Facturasv y se le colocará un FOREIGN KEY CONSTRAINT .

```
SQL> create table facturasv(
  2  numero varchar2(6) NOT NULL,
  3  codcli varchar2(4) NOT NULL,
  4  monto number(10,2) NOT NULL,
  5  estatus varchar2(1) NOT NULL,
  6  CONSTRAINT facturas_fk FOREIGN KEY (codcli)
  7  REFERENCES clientesv (codigo));

Table created.
```

Al crear la Tabla facturasv, se le dijo a ORACLE que el Campo codcli representa una clave externa (Foránea) de la Tabla Clientes y que no se puede Eliminar un Cliente sin que chequear la tabla Facturasv. Los campos que enlazan esta relación son: Codcli en la tabla facturasv y Codigo en la Tabla Clientesv.

Al revisar el contenido de la Tabla Clientesv, se encuentra lo siguiente:

```
SQL> select * from clientesv;

CODI NOMBRE                    LIMCRE NUMFAX          E
---- -------------------- ---------- --------------- -
0051 ACME Sistemas              1500 0251-2555555    A
```

En ese orden de ideas vamos a Incluir en la Tabla Facturasv un Registro del Cliente 0051.

```
SQL> insert into facturasv
  2  values ('001526','0051',20000,'A');

1 row created.
```

Ahora para probar la Integridad Referencial, se va a eliminar Físicamente el Cliente 0051.

```
SQL> delete from clientesv
  2  where codigo='0051'
  3  and estatus='A';
delete from clientesv
*
ERROR at line 1:
ORA-02292: integrity constraint (SCOTT.FACTURAS_FK) violated - child record
found
```

ORACLE va a indicar que se encontró un Registro hijo (Se refiere a la Factura No. 001256), por ende no borrará el Registro de Clientes.

OJO PELAO: El FOREIGN KEY CONSTRAINT funciona solo para eliminaciones Físicas, si se cambia el estatus de un Cliente, la operación se realizará sin problemas. En el caso de la Eliminación Lógica, la Integridad Referencial la realizará el propio programador, sin descargar esa función en el Manejador de Base de Datos.