INSTITUCIÓN UNIVERSITARIA PASCUAL BRAVO | TECNOLOGÍA, INNOVACIÓN CULTURA Y VIDA.

PROGRAMACIÓN EN SQL

Elaborado por:

Oralia Cortés Grajales

PREÁMBULO

Así como en los lenguajes de propósito general utiliza sentencias para manejo de información, el SQL también utiliza dichas sentencias para manejo de información dentro de una base de datos.

Este documento contiene ejemplos de la programación en SQL, sobre:

1. Sentencias
2. Desencadenantes, disparadores o trigger
3. Vistas
4. Procedimientos almacenados
5. Funciones
6. Seguridad
7. Documentación

PROGRAMACIÓN EN SQL

1. SENTENCIAS

➢ Variables

```
/***mostrar un mensaje Hola Buen Día**/
declare @nombre varchar(50)-- declare declara una variable
                    -- @nombre es el identificador de la
                    -- variable de tipo varchar
set @nombre = 'Hola Buen Día' -- El signo = es un operador
                        -- www.devjoker.com es un literal
print @Nombre -- Imprime por pantalla el valor de @nombre.
          -- No diferencia mayúsculas ni minúsculas
```

➢ Go

```
/*******Un script está compuesto por uno o varios lotes. Un lote delimita el alcance de las variables y sentencias del script. Dentro de un mismo script se diferencian los diferentes lotes a través de la instrucción GO.*/

/* Este es el primer lote del script*/
SELECT * FROM medicamento
GO /* GO es el separador de lotes*/
/*Este es el segundo lote del script*/
SELECT getdate() as fecha_actual /* getdate() es una función integrada que devuelve la fecha */
```

➢ Asignación de variable con SET

```
/* devolver un único registro en variable****/

DECLARE @nombre VARCHAR(100)
-- La consulta debe devolver un único registro
SET @nombre =  (SELECT nom_emp
                FROM empleado
                WHERE cedula = 100)
print @nombre
```

➢ Asignación de variable con SET

```
/***asignacion con select*/
DECLARE @nombre VARCHAR(100),
        @telefono VARCHAR(100),
        @sala int
```

```
SELECT  @nombre=nom_emp,
        @telefono=telefono,
        @sala=salario
FROM empleado
WHERE cedula = 100

PRINT @nombre
PRINT @telefono
PRINT @sala
```

➢ **Condicionales**

```
DECLARE @Web varchar(100),
        @diminutivo varchar(3)

SET @diminutivo = 'si'

 IF  @diminutivo = 'no'
      BEGIN
         PRINT 'Buen Día'
      END
  ELSE
        BEGIN
         PRINT 'mal día'
        END
```

➢ **Try y catch**

```
/* Control de errores en el código de Transact-SQL similar a las características de
control de excepciones de los lenguajes de programación.*/

BEGIN TRY
    DECLARE @divisor int ,
            @dividendo int,
          @resultado int

    SET @dividendo = 100

    SET @divisor = 1
-- Esta línea provoca un error de división por 0

    SET @resultado = @dividendo/@divisor
    PRINT 'el resultado es ' + cast(@resultado AS varchar)
    END TRY
 BEGIN CATCH
    PRINT 'Hay error'
 END CATCH
```

➢ **Estructura condicional CASE**

```
/*La estructura condicional CASE permite evaluar una expresión y devolver
un valor u otro.*/

 DECLARE @Web varchar(100),
         @diminutivo varchar(5)
    SET @diminutivo = 'MAL'
    SET @Web = (CASE @diminutivo
                  WHEN 'BIEN' THEN 'www.ddddd.com'
                  WHEN 'MAL' THEN 'www.eeeeee.com'
                  ELSE 'www.ddddd.com'
                END)
    PRINT @Web

/*Otra sintaxis de CASE nos permite evaluar diferentes expresiones:****/

DECLARE @Web varchar(100),
         @diminutivo varchar(3)
    SET @diminutivo = 'BIEN'

    SET @Web = (CASE
                  WHEN @diminutivo = 'BIEN' THEN 'www.ddddd.com'
                  WHEN @diminutivo = 'MAL' THEN 'www.eeeee.com'
                  ELSE 'www.ddddd.com'
                END)
    PRINT @Web

    select * from empleado
```

➢ **El bucle WHILE**

```
/*Se repite mientras expresión se evalúe como verdadero**/

DECLARE @contador int
    SET @contador = 0
    WHILE (@contador < 100)
    BEGIN
          SET @contador = @contador + 1

          PRINT 'Iteracion del bucle ' + cast(@contador AS varchar)
    END
```

```
/*    Podemos pasar a la siguiente iteración del bucle utilizando CONTINUE.*/
DECLARE @contador int
   SET @contador = 0
   WHILE (@contador < 100)
   BEGIN
         SET @contador = @contador + 1
         IF (@contador % 2 = 0)
            CONTINUE
         PRINT 'Iteracion del bucle ' + cast(@contador AS varchar)
   END

/*El bucle se dejará de repetir con la instrucción BREAK.**/
   DECLARE @contador int
   SET @contador = 0
   WHILE (1 = 1)
   BEGIN
         SET @contador = @contador + 1
         IF (@contador % 50 = 0)
            BREAK
         PRINT 'Iteracion del bucle ' + cast(@contador AS varchar)
   END
```

2. DESCENCADENANTES, DISPARADORES O TRIGGER

➢ **Operaciones con los trigger**

CREATE TRIGGER (Transact-SQL): Crea el trigger
DISABLE TRIGGER (Transact-SQL): Deshabilita el trigger
ALTER TRIGGER (Transact-SQL): Habilita el trigger
ENABLE TRIGGER (Transact-SQL): Habilita el trigger
DROP TRIGGER (Transact-SQL): Borra el trigger

Los trigger son: Trigger DML y Trigger DDL.

A continuación se describen cada uno de ellos.

➢ **Trigger DML**

Se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML).

Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.

Sintaxis

```
CREATE TRIGGER nom_trigger
ON tabla que inserta
FOR operacion INSERT, delete, update
as update tabla que actualiza
set campo=actualizacion
from tablas  /*las tablas involucradas*/
where /*condicion*/
```

Ejemplo: Estudiante/ pago

Se tiene la siguiente base de datos:

Estudiante(#cedula, nom_e, deuda)

Pago(#num_pago, fecha, valor, cedula)

/*Crear un trigger para que cada vez que se realice un pago se decremente la deuda del estudiante.*/

```
CREATE TRIGGER matri
ON pago
FOR INSERT
as update estudiante
set estudiante.deuda=estudiante.deuda-inserted.valor
from estudiante inner join inserted
on estudiante.cedula=pago.cedula
```

Ejemplo trigger DML en la base de datos brigadas

/* Crear un trigger que permita controlar el salario del empleado cuando participa en una brigada si se le da el 5% de la cantidad utilizada por el valor del medicamento*/

```
create trigger Tr_Contr_Salario1
on  bri_med
for insert
as update empleado
set
empleado.salario=empleado.salario+(bri_med.canti_utilizada*medicamento.valor)*0.05)
from  empleado inner join participa on empleado.cedula=participa.cedula
inner join brigada
on brigada.cod_bri=participa.cod_bri
inner join bri_med
on brigada.cod_bri=inserted.cod_bri
 inner join medicamento on medicamento.cod_med=bri_med.cod_med
```

➢ **Trigger DDL**

Se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Los eventos DDL corresponden principalmente a operaciones CREATE,ALTER y DROP de Transact-SQL y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

/*Ejemplo trigger DDL que no permite modificar ni crear tablas en la base de datos brigadas*/

```
create trigger TR_Brigadas_medicamentos
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
BEGIN
RAISERROR ('No está permitido borrar ni modificar tablas !' , 16, 1)
ROLLBACK TRANSACTION
END
```

3. VISTAS[1]

Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre. Sin embargo, a menos que esté indexada, una vista no existe como conjunto de valores de datos almacenados en una base de datos. Las filas y las columnas de datos proceden de tablas a las que se hace referencia en la consulta que define la vista y se producen de forma dinámica cuando se hace referencia a la vista.

Una vista actúa como filtro de las tablas subyacentes a las que se hace referencia en ella.

No existe ninguna restricción a la hora de consultar vistas.

Sintaxis:

CREATE VIEW <nombre_vista>
AS
(<sentencia_select>);

Operaciones:

- Create view
- Alter view
- Drop view

[1] Tomado de Vistas en SQL Server 2008 (Agosto 12 de 2015). Recuperado de http://documents.mx/documents/vistas-en-sql-server-2008pdf.html#

Ejemplo

```
/*Crear una vista lineal que muestre los datos de los empleados y un aumento del 10% sobre su bonificacion y su salario si este esta entre 600000 y 2000000 y han participado en mas de 2 brigadas*/

create view v_empleadoCondicion
as
select empleado.*, salario+(salario*0.1) as
salarioAunmento,bonificacion+(bonificacion*0.1) as bonificacionAumento
from empleado inner join participa
on empleado.cedula=participa.cedula
where salario between 600000 and 2000000
group by
participa.cedula,empleado.cedula,empleado.bonificacion,empleado.nom_emp,empleado.s
alario,empleado.telefono
having COUNT(*)>2
```

Para ver la vista

```
Select * from v_empleadoCondicion
```

Las principales razones por las que podemos crear vistas son:

Seguridad: nos pueden interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.

Comodidad: como hemos dicho el modelo relacional no es el más cómodo para visualizar los datos, lo que nos puede llevar a tener que escribir complejas sentencias SQL, tener una vista nos simplifica esta tarea.

Las vistas no tienen una copia física de los datos, son consultas a los datos que hay en las tablas, por lo que si actualizamos los datos de una vista, estamos actualizando realmente la tabla, y si actualizamos la tabla estos cambios serán visibles desde la vista.

***Nota**:* No siempre podremos actualizar los datos de una vista, dependerá de la complejidad de la misma (dependerá de si el conjunto de resultados tiene acceso a la clave principal de la tabla o no).

Tipos de vistas:

Se pueden crear varios tipos de vistas, cada uno de los cuales tienen ventajas en ciertas situaciones. El tipo de vista que se ha de crear depende completamente de para qué se quiera usar la vista. Se pueden crear vistas de cualquiera de las siguientes maneras:

- **Subconjunto de columnas de una tabla**: Una vista puede consistir en una o más columnas de una tabla. Probablemente es el tipo de vista más común y se puede utilizar para simplificar los datos o para seguridad.
- **Subconjunto de filas de una tabla:** Una vista puede contener todas las filas que se deseen. Este tipo de vista también es útil para la seguridad.

- **Unión de dos o más tablas:** Se puede crear una vista usando una operación de unión. Las operaciones de unión complejas se pueden simplificar cuando se usa una vista.
- **Información de agregación:** Se puede crear una vista que contenga datos de agregación. Este tipo de vista también se usa para simplificar operaciones complejas.

Las vistas también pueden ser usadas para consolidar datos divididos. Se pueden dividir los datos de una tabla muy larga en varias tablas de menor tamaño para una mejor administración y, posteriormente, se pueden usar las vistas para unir estas tablas en una gran tabla virtual que facilite el acceso a la misma.

Restricciones de las vistas:

SQL Server asigna una serie de restricciones en la creación y el uso de las vistas. Estas restricciones incluyen las siguientes:

Limitación de columnas: Una vista puede hacer referencia hasta a 1024 columnas de una tabla. Si se necesita hacer referencia a un número mayor de columnas, se tendrán que usar otros métodos.

Limitación de la base de datos: Solamente se puede crear una vista de una tabla en la base de datos a la que el creador de la vista está accediendo.

Limitación de seguridad: El creador de la vista debe tener acceso a todas las columnas a las que se haga referencia en la vista.

Reglas de integridad de los datos: Cualquier actualización, modificación, entre otras, que se haga en la vista, no puede romper las reglas de integridad de los datos. Por ejemplo, si la tabla subyacente de la vista no admite los valores nulos, la vista tampoco los admitirá.

Limitación de niveles en las vistas anidadas: Las vistas se pueden crear sobre otras vistas (en otras palabras, se puede crear una vista que acceda a otras vistas). Las vistas se pueden anidar hasta 32 niveles.

Limitación de la orden SELECT: La orden SELECT de una vista no puede contener una cláusula ORDER BY ni la palabra clave INTO.

Limitación de agrupación: No puede tener agrupaciones

4. PROCEDIMIENTOS ALMACENADOS[2]

Un procedimiento almacenado (store procedure - SP) es una secuencia ordenada de instrucciones T-SQL, que pueden recibir y proporcionar parámetros provistos por el usuario

[2] Recuperado de: SQL Procedimientos almacenados paso a paso. (Octubre 31 de 2008). Recuperado de https://mspnor.wordpress.com/author/norber/page/18/

y se pueden guardar en el servidor con un nombre, para posteriormente se invocados y ejecutados.

Ventajas de usar SP

- **Compilación:** La primera vez que se invoca un SP, el motor lo compila y a partir de ahí, se sigue usando la versión compilada del mismo, hasta que se modifique o se reinicie el servicio de SQL. Esto significa que se tendrá un mejor rendimiento que las consultas directas que usan cadenas con las instrucciones T-SQL, que se compilan cada vez que se invocan.
- **Automatización:** si tenemos un conjunto de instrucciones T-SQL, las cuales queremos ejecutar de manera ordenada, un SP es la mejor manera de hacerlo.
- **Administración:** cuando realizamos aplicaciones con un gran número de líneas de código, y queremos hacer cambios, solo implica modificar un SP y no toda la aplicación, lo que significa solo cambiamos los SP en el servidor y no tenemos que actualizar la aplicación en todos los equipos cliente**.**
- **Seguridad:** una parte importante es que a los usuarios de nuestra aplicación, solo les proporcionamos los permisos para ejecutar los procedimientos almacenados y no el acceso a todos los objetos de la base.
- **Programabilidad:** Los SP admiten el uso de variables y estructuras de control como IF, Bucles, Case, entre otros, además del manejo de transacción y permite controlar excepciones. Y cuando trabajamos con SP CLR podemos hacer uso de cualquier lenguaje .NET como lo son C# y VB.NET.
- **Tráfico de Red**: Pueden reducir el tráfico de la red, debido a que se trabaja sobre el motor (en el servidor).

Sintaxis

```
Create procedure nom_procedimiento
@A varchar(50),
@B varchar(50),
as
        Select
        From
        Where
```

Ejemplo en la base de datos brigadas

/*Crear un procedimiento que muestre los datos de los medicamentos con nombre A o B entrados por el usuario y que han participado en más de X brigadas.*/

```
Create procedure Sp_MostrarMedicamentos
@A varchar(50),
@B varchar(50),
@XBrigada int
```

```
as
Select
medicamento.cod_med,medicamento.nom_med,medicamento.forma_uso,medicamento.c
antidad,medicamento.valor, COUNT(*) as BRIGADAS
from medicamento inner join bri_med
on medicamento.cod_med = bri_med.cod_med
Where medicamento.nom_med= @A or medicamento.nom_med=@B
group by
bri_med.cod_med,medicamento.cod_med,medicamento.nom_med,medicamento.forma_u
so,medicamento.cantidad,medicamento.valor
having COUNT(*)>@XBrigada
```

5. FUNCIONES

- **Funciones lineales**

Las funciones lineales son las funciones que devuelven un conjunto de resultados correspondientes a la ejecución de una sentencia SELECT.

La sintaxis para una función de tabla en línea es la siguiente:

```
CREATE FUNCTION Function_NameFunctionName
(
-- Lista de parámetros
<@param1, sysname, @p1> <Data_Type_For_Param1, , int>,...
)
RETURNS  tipo TABLE
AS
RETURN valores
(
-- Sentencia Transact SQL
)
```

Ejemplo en la base de datos brigadas

/*crear una función que muestre los datos de los empleados con un salario mayor que un valor dado por el usuario y que han participado en una cantidad mayor que un valor dado por el usuario*/

```
create function f_empleado(
@val int,
@can int)
returns table
as
return(
select *
from empleado
where salario>@val
```

```
group by
participa.cedula,empleado.cedula,empleado.nom_emp,empleado.telefono,empleado.salari
o,empleado.bonificacion
Having count(*) >@can)
```

- **Funciones escalares**

Son muy similares a procedimientos almacenados con parámetros de salida, pero estas pueden ser utilizadas en consultas de selección y en la cláusula where de las mismas.

Las funciones no pueden ejecutar sentencias INSERT o UPDATE.

Ejemplo en la base de datos brigadas

```
/*Crear una función que muestre el máximo valor de los medicamentos*/
create function maximovalor()
returns int
as
begin
declare @maxi int
select @maxi=max(valor)
from medicamento
return (@maxi)
end
```

- **Funciones ya definidas en el SQL**

Como todos los lenguajes de programación el SQL ya tiene incorporadas muchas funciones para el manejo de los datos.

6. SEGURIDAD

Logins, están asociados a un servidor.

Users, están asociados a una base de datos

Ejemplo

1. Crear un usuario de Windows llamado María
2. Abrir el MS Management Studio
3. Buscamos en el Explorador de Objetos, la Carpeta Security, Logins. Aquí vamos a definir un nuevo Login, el cual tendrá acceso al Servidor, pero no a una base específica.
4. Nuevo Login. Search. Escribimos el nombre de nuestro usuario “Maria”. Comprobamos. Finalmente Aceptar

5. Buscamos nuestra Base de Datos “Demo”. Folder Security. Folder Users. Nuevo User.en la base de datos
6. Aquí vamos a definir el nombre del User, que puede ser o no el mismo del Login. Escribimos Maria y buscamos el Login equivalente. Click en OK
7. Iniciemos sesión de Windows con el Login Maria.
8. Verifiquemos que no tenemos acceso a otras bases más que a “Demo”, adicional no podemos visualizar ninguna Tabla, solo vistas, pero al ejecutarla nos regresaran 0 Registros.
9. Regresemos a la sesión de Administrador y asignemos Roles al User Maria. Vamos a la Base de Datos “Demo”. Users y buscamos a “Maria”. Asignaremos db_datareader y db_datawriter.
10. Nuevamente regresamos a la sesión de Windows “Maria”. Y verificamos los cambios. Podemos ver las Tablas. Y podemos hacer consulta de búsqueda e inserción de datos.
11. En la sesión Administrador. Buscamos el user “Maria” y vamos a la sección Securables. Los Securables o “Asegurables” nos van a dar la opción de asignar permisos al usuario a determinados objetos y determinadas acciones.
12. En securables seleccionamos Search, “All Objects of the types”, “Tables”. Por ejemplo: Para tabla Productos definimos que no tendrá permisos para borrar.
13. Regresamos a la sesión “Maria” y verificamos esta acción.

Roles

Cuando hablamos de Roles en SQL Server, tenemos 2 Categorías:

- Server-**Level Role:** a nivel de servidor
- **DataBase-Level** Role: a nivel de base de datos

En la siguiente tabla[3] se pueden observar las funciones a nivel del servidor

Función de nivel de servidor	**Descripción**
sysadmin	Pueden realizar cualquier actividad en el servidor.
serveradmin	Pueden cambiar las opciones de configuración en el servidor y cerrar el servidor.
securityadmin	Administran los inicios de sesión y sus propiedades. Administran los permisos de servidor GRANT, DENY y REVOKE. También administran los permisos de base de datos GRANT, DENY y REVOKE. Asimismo, pueden restablecer las contraseñas para los inicios de sesión de SQL Server.
processadmin	Pueden finalizar los procesos que se ejecutan en una instancia de SQL Server.

[3] Tomado de TechNet (s.f.). Roles de Nivel de Base de Datos. Recuperado de: https://technet.microsoft.com/es-es/library/ms189121(v=sql.105).aspx

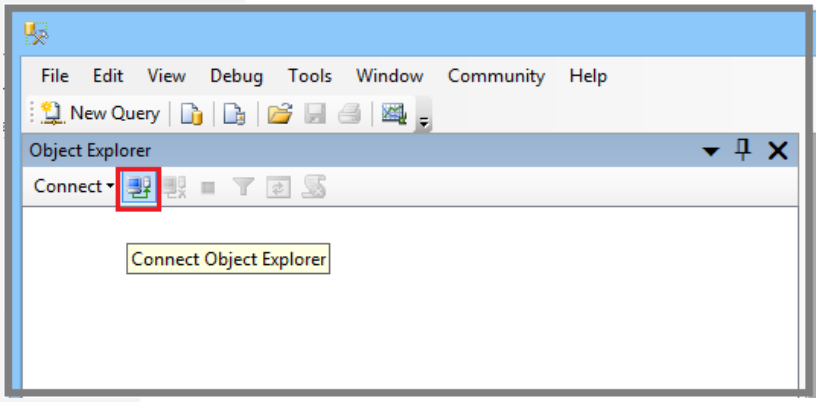
setupadmin	Pueden agregar y quitar los servidores vinculados.
bulkadmin	Pueden ejecutar la instrucción BULK INSERT.
diskadmin	Se utiliza para administrar archivos de disco.
dbcreator	Pueden crear, modificar, quitar y restaurar cualquier base de datos.
public	Cada inicio de sesión de SQL Server pertenece a la función pública de servidor. Cuando a una entidad de seguridad de servidor no se le han concedido ni denegado permisos específicos para un objeto protegible, el usuario hereda los permisos concedidos a la función pública para ese elemento. Solo asigne los permisos públicos en cualquier objeto cuando desee que el objeto esté disponible para todos los usuarios.

En la siguiente tabla[4] se pueden observar las funciones a nivel de la base de datos

Función de nivel de base de datos	**Descripción**
db_owner	Pueden realizar todas las actividades de configuración y mantenimiento en la base de datos y también pueden quitar la base de datos.
db_securityadmin	Pueden modificar la pertenencia a funciones y administrar permisos. Si se agregan entidades de seguridad a esta función, podría habilitarse un aumento de privilegios no deseado.
db_accessadmin	Pueden agregar o quitar el acceso a la base de datos para inicios de sesión de Windows, grupos de Windows e inicios de sesión de SQL Server.
db_backupoperator	Pueden crear copias de seguridad de la base de datos.
db_ddladmin	Pueden ejecutar cualquier comando del lenguaje de definición de datos (DDL) en una base de datos.
db_datawriter	Pueden agregar, eliminar o cambiar datos en todas las tablas de usuario.
db_datareader	Pueden leer todos los datos de todas las tablas de usuario.
db_denydatawriter	No pueden agregar, modificar ni eliminar datos de tablas de usuario de una base de datos.
db_denydatareader	No pueden leer datos de las tablas de usuario dentro de una base de datos.

[4] Tomado de TechNet (s.f.). Roles de Nivel de Base de Datos. Recuperado de: https://technet.microsoft.com/es-es/library/ms189121(v=sql.105).aspx

Para verificar que el usuario está funcionando correctamente, oprimimos el icono de conectar como se muestra en la siguiente figura.




```
/* Ejemplo utilizando comandos EN SQLSERVER 2008*/

   TEMA: LOGINS & USERS
*/

USE MASTER
GO

-- Creamos un login sql

CREATE LOGIN L1 WITH PASSWORD = 'SIMPLE'
GO


-- Mostramos que somos admin

SELECT SUSER_SNAME()

-- Ahora nos ponemos como l1

EXECUTE AS LOGIN= 'L1'
GO
SELECT SUSER_SNAME()

-- Intentamos entrar a la base de datos DB_Delitos1 dando como resultado un error.

USE DB_Delitos1
GO

REVERT -- REVERTIMOS

-- Ahora creamos un usuario en DB_Delitos1 que represente al login L1

USE DB_Delitos1
GO
CREATE USER L1
GO
```

```
-- Ahora nos ponemos como L1

EXECUTE AS LOGIN= 'L1'
GO
SELECT SUSER_SNAME()

-- Ahora si podemos entrar a la base DB_Delitos1

USE DB_Delitos1
GO

REVERT -- REVERTIMOS

-- Borramos el usuario l1 en DB_Delitos1

USE DB_Delitos1
GO

DROP USER L1
/*****/
```

- **Dar permiso**

```
USE DB_AT
GRANT alter ON OBJECT::DBO.TBL_deduccion TO JULIO
GO

EXECUTE AS LOGIN= 'julio'
GO
SELECT SUSER_SNAME()

alter table TBL_deduccion add desde float
select * from TBL_deduccion
```

- Denegar permiso

```
USE DB_AT
DENY insert ON OBJECT::DBO.TBL_deduccion
 TO JULIO
GO
```

- **Revocar permiso**

```
USE DB_AT
REVOKE alter ON OBJECT::dbo.TBL_deduccion FROM julio
GO
```

7. DOCUMENTACIÓN

Documentación de la base de datos

Se debe documentar la base de datos, tanto para los usuarios como para los programadores, ofreciendo manuales de usuario según resulte conveniente e incluyendo descripciones de los códigos o campos en el diseño de la base de datos para orientar a los programadores. En esta debe utilizar:

- Los nombres para todos los objetos incluidos en la base de datos (como tablas, reglas de integridad, consultas, procedimientos, triggers funciones etc.)
- Índices e identificadores únicos para acelerar las búsquedas y permitir ligar tablas de datos.

/*COMANDOS DE SQL PARA LA DOCUMENTACIÓN DE UNA BASE DE DATOS*/

```
/* Consulta para obtener las tablas de una base de datos*/

SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES

/*Obtener todas las columnas*/

SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS

/*Obtener todas las columnas y tablas relacionadas puedes utilizar*/

SELECT TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS

/**********/
/**Muestra  el catálogo, tablas, columnas, columnas por defecto*/

SELECT TABLE_CATALOG, TABLE_NAME, COLUMN_NAME, COLUMN_DEFAULT
FROM Biblioteca1.INFORMATION_SCHEMA.COLUMNS

/**consulta para SQL Server 2008 que muestra: tabla, Columna, Tipo de dato, Tamaño del dato***/

SELECT so.name AS Tabla,sc.name AS Columna,st.name AS Tipo,
sc.max_length AS Tamaño
FROM sys.objects so INNER JOIN sys.columns sc
ON so.object_id = sc.object_id INNER JOIN sys.types st
ON st.system_type_id = sc.system_type_id
AND st.name != 'sysname'
WHERE so.type = 'U'
ORDER BY so.name, sc.name
/****consultar las tablas del sistema*/
select *
from sysobjects
WHERE NAME LIKE 'TBL_%'
```

Puede cambiar la condición según el objeto que busque

También por tipo de objeto

```
SELECT sys.objects.name
FROM sys.objects
WHERE sys.objects.type =  'PK'
```

Tipo de objeto:

C = Restricción CHECK
D = DEFAULT (restricción o independiente)
F = Restricción FOREIGN KEY
FN = Función escalar de SQL
P = Procedimiento almacenado de SQL
PK = Restricción PRIMARY KEY
TA = Desencadenador DML del ensamblado (CLR)
TF = Función con valores de tabla SQL
TR = Desencadenador DML de SQL
U = Tabla (definida por el usuario)
UQ = Restricción UNIQUE
V = Vista

Herramientas para documentar bases de datos: Entre otras están.

- Data Dictionary Creator
- OpenAlfa, ApexSQL
- DBDESC.

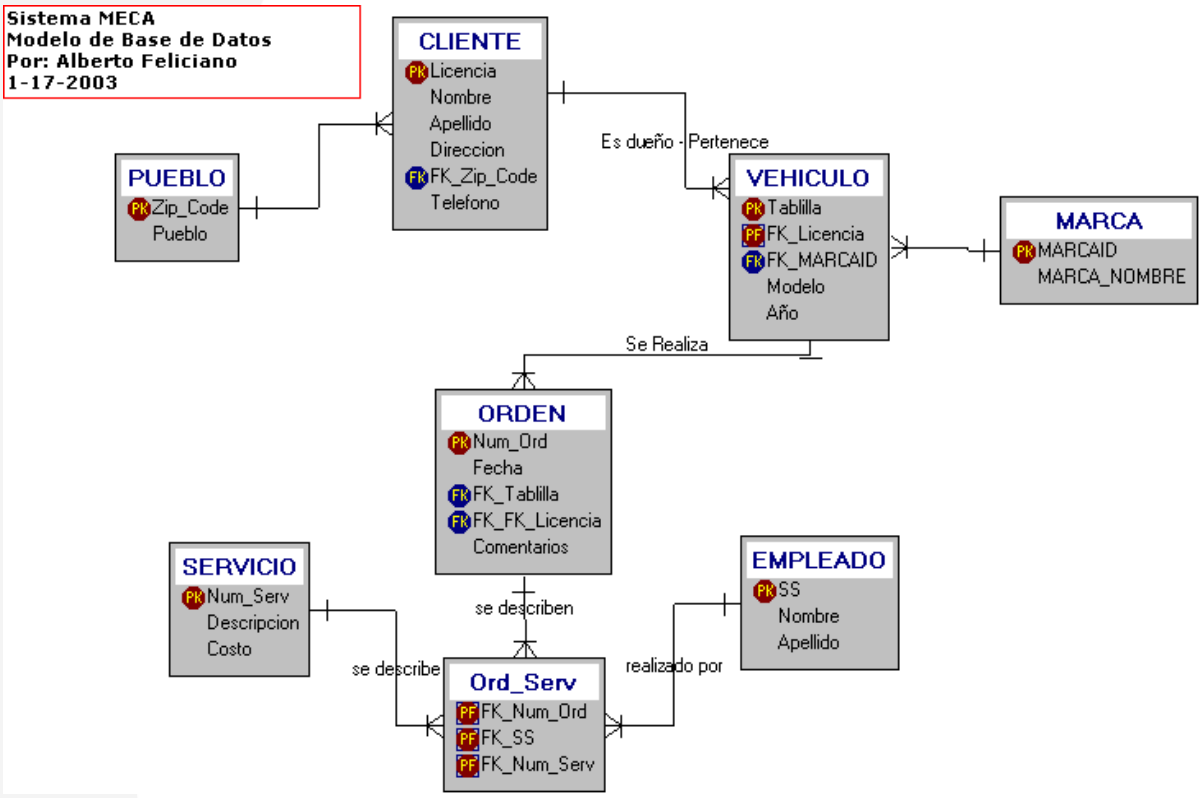
Ejemplo de documentación:

Sistema propuesto[5]

Descripción general: Establecer un sistema de base de datos que lo llamaremos MECA. Este sistema tendrá la capacidad para manejar de una forma eficiente y controlar las órdenes de reparaciones y servicios de mecánica para el taller de Orlando Concepción. Se espera que con este sistema se reduzcan mucho gasto por operación y se controle y se aumente de una manera positiva los ingresos que el taller de mecánica tiene.

[5] Tomado de Feliciano, A. (Julio 7 de 2015). Análisis y Estudio de Requerimientos Taller Mecánico. Recuperado de http://documents.mx/documents/analisis-y-estudio-de-requerimientos-taller-mecanico.html#

Modelo entidad relación de la situación propuesto



Diccionario de Datos

Projectname :MECA
Projectauthor :Alberto Feliciano
Projectdescription :Sistema para un taller de mecánica
Date report
4/5/2003

Title of report List of attributes

Objectname	Attributename	Datatype	Length	Dec.	Domain	Default
CLIENTE	Licencia	VARCHAR2	7	0		
CLIENTE	Nombre	VARCHAR2	10	0		
CLIENTE	Apellido	VARCHAR2	12	0		
CLIENTE	Direccion	VARCHAR2	50	0		
CLIENTE	FK_Zip_Code	VARCHAR2	10	0		
CLIENTE	Telefono	VARCHAR2	12	0		
VEHICULO	Tablilla	VARCHAR2	6	0		
VEHICULO	FK_Licencia	VARCHAR2	7	0		
VEHICULO	FK_MARCAID	INTEGER	0	0		

VEHICULO	Modelo	VARCHAR2	25	0
VEHICULO	Año	VARCHAR2	4	0
ORDEN	Num_Ord	AUTONUMBER	0	0
ORDEN	Fecha	DATE	0	0
ORDEN	FK_Tablilla	VARCHAR2	6	0
ORDEN	FK_FK_Licencia	VARCHAR2	7	0
ORDEN	Comentarios	VARCHAR2	100	0
SERVICIO	Num_Serv	AUTONUMBER	0	0
SERVICIO	Descripcion	VARCHAR2	50	0
SERVICIO	Costo	MONEY	8	2
EMPLEADO	SS	VARCHAR2	11	0
EMPLEADO	Nombre	VARCHAR2	20	0
EMPLEADO	Apellido	VARCHAR2	35	0
PUEBLO	Zip_Code	VARCHAR2	10	0
PUEBLO	Pueblo	VARCHAR2	15	0
MARCA	MARCAID	AUTONUMBER	0	0
MARCA	MARCA_NOMBRE	VARCHAR2	75	0
Ord_Serv	FK_Num_Ord	INTEGER	0	0
Ord_Serv	FK_SS	VARCHAR2	11	0
Ord_Serv	FK_Num_Serv	INTEGER	0	0

Restricciones en el Sistema Propuesto:

- Las cuentas son estrictamente de contado.
- El cliente tiene que traer las piezas declaradas en el estimado o evaluación.
- Luego de reparado el auto, el cliente tiene dos días laborables para recoger el auto
- Se le cobran 5 dólares por hora por cada día adicional que el cliente deje el auto en el taller después del aviso de la reparación.

Diferencias entre el sistema actual y el sistema propuesto:

- El primer sistema actual es completamente manual, mientras que el sistema propuesto está automatizado.
- En el sistema propuesto se minimiza la duplicación de datos.
- Los datos están más seguros en el sistema propuesto.
- El sistema propuesto será más ágil en el proceso de registrar al cliente.
- Los reportes a la administración son fácilmente generados.
- Se conoce mejor la cantidad de clientes y vehículos que se atienden y se reparan.
- Los costos operacionales se reducen en el sistema propuesto.
- En el sistema propuesto, los datos vitales están a tiempo para la toma de decisiones.
- En el sistema propuesto, los datos estarán más seguros ya que el acceso a estos estará restringido según los privilegios que se le dé al usuario.
- Es más difícil perder un expediente en el sistema propuesto.

INSTITUCIÓN UNIVERSITARIA
PASCUAL BRAVO

Unidad de Educación Virtual