Universidad Tecnológica Nacional

Tecnicatura Universitaria en Programación

Plan 2024

Materia: Bases de Datos II

Profesor: Juan M. Arce

Desarrollo de un proyecto de Base de Datos utilizando SqlServer

La UTN definió el plan de estudios de la carrera “Tecnicatura Universitaria en Programación” en el año 2024.

El proyecto a desarrollar consiste en la creación e implementación de un modelo de datos que contemple al detalle el contenido del nuevo plan de la carrera, agregándole funcionalidad a fin de facilitar el acceso a la información del nuevo plan.

Tenemos entonces un grupo de áreas académicas que a su vez agrupan las materias cursar.

Estructuración por áreas

El plan de estudio estará organizado en tres áreas principales:

1. Ciencias Básicas
    - Matemática
    - Estadística

2. Disciplinas Tecnológicas
    - Programación I
    - Arquitectura y Sistemas Operativos
    - Programación II
    - Base de Datos I
    - Programación III
    - Base de datos II
    - Metodología de Sistemas I
    - Programación IV
    - Metodología de Sistemas II
    - Gestión de Desarrollo de Software
    - Introducción al Análisis de Datos

3. Disciplinas Complementarias
    - Inglés I
    - Inglés II
    - Organización Empresarial
    - Legislación

La estructura de cursado de la carrera es como muestra la siguiente imagen:

8. PLAN DE ESTUDIO

N°	ASIGNATURA	RÉGIMEN	HORAS	HORAS	CREDITOS
PRIMER CUATRIMESTRE					
1	Programación I	Cuatrimestral	8	128	13
2	Arquitectura y Sistemas Operativos	Cuatrimestral	4	64	5
3	Matemática	Cuatrimestral	4	64	5
4	Organización empresarial	Cuatrimestral	4	64	5
Total			**20**	**320**	**28**
SEGUNDO CUATRIMESTRE					
5	Programación II	Cuatrimestral	8	128	13
6	Probabilidad y Estadística	Cuatrimestral	4	64	5
7	Base de Datos I	Cuatrimestral	4	64	5
8	Inglés I	Cuatrimestral	4	64	5
Total			**20**	**320**	**28**
TERCER CUATRIMESTRE					
9	Programación III	Cuatrimestral	8	128	13
10	Base de Datos II	Cuatrimestral	4	64	5
11	Metodología de Sistemas I	Cuatrimestral	4	64	5
12	Inglés II	Cuatrimestral	4	64	5
Total			**20**	**320**	**28**
CUARTO CUATRIMESTRE					
13	Programación IV	Cuatrimestral	8	128	13
14	Metodología de Sistemas II	Cuatrimestral	4	64	5
15	Introducción al análisis de datos	Cuatrimestral	2	32	2
16	Legislación	Cuatrimestral	2	32	2
17	Gestión de desarrollo de software	Cuatrimestral	4	64	5
18	Trabajo Final Integrador	Cuatrimestral		95	9
Total			**20**	**415**	**36**
Duración de la carrera en h. y CRE				**1375**	**120**

El trabajo consiste en la identificación de entidades en base a la información disponible para su implementación en un modelo relacional utilizando el lenguaje SQL.

Entidades Identificadas:

- Institución
- Tipo de Carrera
- Modalidad de Estudio
- Carrera
- Régimen
- Área
- Plan de Estudio
- Cuatrimestre
- Nivel
- Asignatura
- Asignaturas del Plan
- Correlativas
- Persona
- Alumno
- Docente
- Tipo de Evaluación
- Evaluación

El proyecto arranca desde:

- la identificación de las entidades
- la creación de las tablas
- la carga incial de datos
- la creación de las Stored Procedures

Y luego la definición de objetivos de utilización del modelo generado para ser consumido por aplicaciones externas

Detalle de las entidades y sus tablas

Institución: Donde se dicta la carrera.

```
create table Institucion (
     id integer not null primary key identity(1,1),
     nombre  varchar(256) not null unique,
     domicilio varchar(256));
```

Tipo de Carrera: Ingeniería, Tecnicatura, Cursos de Posgrado, etc.

```
create table TipoCarrera(
     id integer not null primary key identity(1,1),
     nombre varchar(128)not null unique);
```

Modalidad de Carrera: Presencial, A Distancia, Mixta.

```
create table ModalidadCarrera(
     id integer not null primary key identity(1,1),
     nombre varchar(128)not null unique);
```

Carrera: A cursar

```
create table Carrera (
     id integer not null primary key identity(1,1),
     institucion_id integer not null,
     tipocarrera_id integer not null,
     modalidadcarrera_id integer not null,
     nombre  varchar(256) not null unique,
     duracion integer not null,
     foreign key (institucion_id) references  Institucion(id),
     foreign key (tipocarrera_id) references  TipoCarrera(id),
     foreign key (modalidadcarrera_id) references
ModalidadCarrera(id));
```

Régimen: Anual, Cuatrimestral

```
create table Regimen(
     id integer not null primary key identity(1,1),
     nombre varchar(128)not null unique);
```

Área: Área académica

```
create table Area(
     id integer not null primary key identity(1,1),
     nombre varchar(128)not null unique);
```

Plan de Estudio: De la carrera

```
create table PlanEstudio (
     id integer not null primary key identity(1,1),
     nombre  varchar(256) not null unique,
     carrera_id integer not null,
     fecha_inicio datetime not null,
     fecha_fin datetime,
     foreign key (carrera_id) references Carrera(id))
```

Cuatrimeste: Primero, Segundo

```
create table Cuatrimestre (
     id integer not null primary key identity(1,1),
     numero integer not null check (numero >= 0),
     descripcion varchar(128) not null);
```

Nivel: Primer año, Segundo año

```
create table Nivel (
     id integer not null primary key identity(1,1),
     numero integer not null check (numero >= 0),
     descripcion varchar(128) not null);
```

Asignatura: Materias de la carrera a cursar

```
create table Asignatura(
     id integer not null primary key identity(1,1),
     area_id integer not null,
     regimen_id integer not null,
     nombre varchar(256) not null,
     horassemana integer not null,
     horascuatrimestre integer not null,
     creditos integer not null,
     foreign key (area_id) references Area(id),
     foreign key (regimen_id) references Regimen(id));
```

Asignaturas del Plan: Materias del plan

```
create table AsignaturasPlan(
     id integer not null primary key identity(1,1),
     plan_id integer not null,
     asignatura_id integer,
     nivel_id integer not null,
     cuatrimestre_id integer not null,
     Foreign Key(plan_id) references PlanEstudio(id),
     Foreign Key(asignatura_id) references Asignatura(id),
     Foreign Key(nivel_id) references Nivel(id),
     Foreign Key(cuatrimestre_id) references Cuatrimestre(id));
```

Correlativas del Plan: Correlatividades del plan

```
create table Correlativas(
     id integer not null primary key identity(1,1),
     asignatura_id integer not null,
     correlativa_id integer not null
          check (asignatura_id != correlativa_id),
     foreign key (asignatura_id) references Asignatura(id),
     foreign key (correlativa_id) references Asignatura(id));

     create unique index uix_correlativas on
     Correlativas(asignatura_id, correlativa_id);
```

Personas: Datos de Personas

```
create table Persona (
     id integer not null primary key identity(1,1),
     apellido varchar(64) not null,
     nombre varchar(64) not null,
     dni integer not null unique);
```

Alumnos: Datos de Alumnos

```
create table Alumno(
     id integer not null primary key identity(1,1),
     persona_id integer not null,
     planestudio_id integer not null,
     legajo integer not null,
     foreign key (persona_id) references Persona(id),
     foreign key (planestudio_id) references PlanEstudio(id));
```

Tipos de Evaluación: Final, Parcial, Trabajo Práctico, Monografía

```
create table TipoEvaluacion(
     id integer not null primary key identity(1,1),
     nombre varchar(128) not null unique);
```

Evaluación:

```
create table Evaluacion(
     id integer not null primary key identity(1,1),
     tipoevaluacion_id integer not null,
     materiaplan_id integer not null,
     fecha datetime not null,
     aprobada_sn char(1) not null check (aprobada_sn in ('S','N')),
     calificacion float);
```

Docente:

```
create table Docente(
     id integer not null primary key identity(1,1),
     persona_id integer not null,
     materiaplan_id integer not null,
     legajo integer not null,
     foreign key (persona_id) references Persona(id),
     foreign key (materiaplan_id) references AsignaturasPlan(id));
```

Detalle de las Stored Procedures

Por cada tabla generaremos 3 Stored Procedures básicas para las funciones:

- **Insert**
- **Update**
- **Delete**

Para la generación disponemos de una serie de Stored Procedures que facilitan el proceso:

- **DropGens**
- **GenDrops**
- **GenDelete**
- **GenInsert**
- **GenUpdate**
- **GenAll**
- **GetErrorInfo**

DropGens: Borra las Stored Procedures genéricas. La idea es comenzar con una Base de Datos limpia.

```
IF OBJECT_ID('geninsert', 'P') IS NOT NULL
BEGIN
DROP PROCEDURE geninsert;
END;

IF OBJECT_ID('gendelete', 'P') IS NOT NULL
BEGIN
DROP PROCEDURE gendelete;
END;

IF OBJECT_ID('genupdate', 'P') IS NOT NULL
BEGIN
DROP PROCEDURE genupdate;
END;

IF OBJECT_ID('GetErrorInfo', 'P') IS NOT NULL
BEGIN
DROP PROCEDURE GetErrorInfo;
END;
```

GenDrops: Para cada tabla, genera los comandos para la eliminación de Stored Procedures asociadas. La salida a pantalla debe guardarse en un archivo llamado DropAll.sql

```
Begin
      declare ctab cursor for SELECT name from sys.tables
      declare @tab varchar(64)
      open ctab
      fetch ctab into @tab
      while @@fetch_status = 0
      Begin
            PRINT ''
            PRINT 'IF OBJECT_ID(' + CHAR(39) + 'I_' + @tab + CHAR(39) +
',' + CHAR(39) + 'P' + CHAR(39) + ') IS NOT NULL'
            PRINT '    DROP PROCEDURE I_' + @tab + ';'
            PRINT 'GO'
            PRINT 'IF OBJECT_ID(' + CHAR(39) + 'U_' + @tab + CHAR(39) +
',' + CHAR(39) + 'P' + CHAR(39) + ') IS NOT NULL'
            PRINT '    DROP PROCEDURE U_' + @tab + ';'
            PRINT 'GO'
            PRINT 'IF OBJECT_ID(' + CHAR(39) + 'D_' + @tab + CHAR(39) +
',' + CHAR(39) + 'P' + CHAR(39) + ') IS NOT NULL'
            PRINT '    DROP PROCEDURE D_' + @tab + ';'
            PRINT 'GO'
            fetch ctab into @tab
      END
      close ctab
      deallocate ctab
End;
```

Como ejemplo, la salida a pantalla comienza así:

```
IF OBJECT_ID('I_Institucion','P') IS NOT NULL
    DROP PROCEDURE I_Institucion;
GO
IF OBJECT_ID('U_Institucion','P') IS NOT NULL
    DROP PROCEDURE U_Institucion;
GO
IF OBJECT_ID('D_Institucion','P') IS NOT NULL
    DROP PROCEDURE D_Institucion;
GO

IF OBJECT_ID('I_TipoCarrera','P') IS NOT NULL
    DROP PROCEDURE I_TipoCarrera;
GO
IF OBJECT_ID('U_TipoCarrera','P') IS NOT NULL
    DROP PROCEDURE U_TipoCarrera;
GO
IF OBJECT_ID('D_TipoCarrera','P') IS NOT NULL
    DROP PROCEDURE D_TipoCarrera;
GO
….
```

Nota; El uso de **GO** es obligatorio cuando usamos la linea de comandos para conectarnos a la base de datos.
Cuando usamos SQL Management Studio, el uso de GO es configurable y se reemplaza por punto y coma.

GenDelete: Por cada tabla, la Stored Procedure para borrar una fila de la tabla asociada. Parámetro de entrada: ID (Primary Key de la tabla)

```
create procedure gendelete
@tab varchar(128)
AS
BEGIN
    PRINT ''
    PRINT 'CREATE PROCEDURE D_' + @tab
    PRINT '@ID INTEGER'
    PRINT 'AS'
    PRINT 'BEGIN'
    PRINT '  BEGIN TRY'
    PRINT '    DELETE FROM ' + @tab + ' WHERE ID = @ID'
    PRINT '  END TRY'
    PRINT '  BEGIN CATCH'
    PRINT '     EXECUTE GetErrorInfo'
    PRINT '  END CATCH'
    PRINT 'END;'
END;
```

Luego de la ejecución, podemos verificar en la Base de Datos que esta nueva Stored Procedure exista.

GenInsert: Por cada tabla, la Stored Procedure recibe los parámetros necesarios y como parámetro de salida devuelve el ID del registro generado.

```
create procedure geninsert
@tab varchar(128)
AS
BEGIN
    DECLARE @SQLTEXT VARCHAR(1024)
    DECLARE @VALUES VARCHAR(1024)
    DECLARE @WHERE VARCHAR(1024)
    DECLARE C_COL CURSOR FOR
    SELECT
        UPPER(c.name),
        CASE y.name
            WHEN 'VARCHAR' then 'VARCHAR(' + TRIM(STR(c.MAX_LENGTH)) + ')'
            WHEN 'CHAR' then 'CHAR(' + TRIM(STR(c.MAX_LENGTH)) + ')'
            ELSE UPPER(y.name)
        END,
        i.is_primary_key
    FROM sys.tables t
        INNER JOIN sys.columns c
              ON c.object_id = t.object_id
        INNER JOIN sys.types y
            ON y.user_type_id = c.user_type_id
        LEFT JOIN sys.index_columns ic
             on ic.object_id = t.object_id
             and ic.column_id = c.column_id
        LEFT JOIN sys.indexes i
             ON i.object_id = t.object_id
             and i.index_id = ic.index_id
    WHERE t.name = @tab

    DECLARE @COLNAME VARCHAR(128)
    DECLARE @COLTYPE VARCHAR(128)
```

```
    DECLARE @ISPK INT
    DECLARE @CNT INTEGER


--    PRINT 'IF OBJECT_ID(' + CHAR(39) + 'I_' + @tab + CHAR(39) + ',' + CHAR(39) + 'P' +
CHAR(39) + ') IS NOT NULL'
--    PRINT '    DROP PROCEDURE I_' + @tab + ';'
    PRINT ''
    PRINT 'CREATE PROCEDURE I_' + @tab
    OPEN C_COL
    FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    SET @CNT = 0
    WHILE @@FETCH_STATUS = 0
    BEGIN
        BEGIN
            IF @ISPK = 1
            BEGIN
                IF @CNT = 0
                    BEGIN
                        PRINT '    @' + @COLNAME + ' ' + @COLTYPE + ' OUTPUT'
                        SET @CNT = 1
                    END
                ELSE
                        PRINT '   ,@' + @COLNAME + ' ' + @COLTYPE + ' OUTPUT'
            END
            ELSE
            BEGIN
                IF @CNT = 0
                    BEGIN
                        PRINT '    @' + @COLNAME + ' ' + @COLTYPE
                        SET @CNT = 1
                    END
                ELSE
                        PRINT '   ,@' + @COLNAME + ' ' + @COLTYPE
            END
        END
        FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    END
    CLOSE C_COL

    PRINT 'AS'
    PRINT 'BEGIN'
    PRINT '  BEGIN TRY'
    PRINT '    INSERT INTO ' + @tab
    SET @CNT = 0
    OPEN C_COL
    FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @ISPK = 1
            BEGIN
                SET @WHERE = '    WHERE ' + UPPER(@COLNAME) + ' = @' + @COLNAME
            END
        ELSE
            IF @CNT = 0
                BEGIN
                    SET @SQLTEXT = '    (' + UPPER(@COLNAME)
                    SET @VALUES =  '    (@' + UPPER(@COLNAME)
                    SET @CNT = 1
                END
            ELSE
                BEGIN
                    SET @SQLTEXT = @SQLTEXT + ' ,' + UPPER(@COLNAME)
                    SET @VALUES =  @VALUES + ' ,@' + UPPER(@COLNAME)
                END
        FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    END
    SET @SQLTEXT = @SQLTEXT + ')'
    SET @VALUES = @VALUES + ')'
    PRINT @SQLTEXT
```

```
    PRINT '    VALUES'
    PRINT @VALUES
    CLOSE C_COL
    DEALLOCATE C_COL
    PRINT '     SET @ID = @@IDENTITY'
    PRINT '  END TRY'
    PRINT '  BEGIN CATCH'
    PRINT '     EXECUTE GetErrorInfo'
    PRINT '  END CATCH'
    PRINT 'END;'
END;
```

Luego de la ejecución, podemos verificar en la Base de Datos que esta nueva Stored Procedure exista.

Como ejemplo, con la Stored Procedure **I_Persona**:

```
declare @Id int
EXEC I_Persona @Id OUT, 'Acosta', 'Fausto', 44867259
EXEC I_Persona @Id OUT, 'Alberoni', 'Azul', 47139322
EXEC I_Persona @Id OUT, 'Albitre', 'Maite', 44380799
EXEC I_Persona @Id OUT, 'Aragon', 'Joaquin', 45641180
EXEC I_Persona @Id OUT, 'Araya', 'Ariana', 42136261
EXEC I_Persona @Id OUT, 'Aroza', 'Aitor Manuel', 46093128
…
```

El orden de los parámetros, en caso de no ponerle el nombre correspondiente, debe respetar como están declarados en la Stored Procedure.
En caso de lo ser así, deberíamos hacer lo siguiente:

```
declare @Id int
EXEC I_Persona @apellido='Acosta', @nombre='Fausto', @dni=44867259, @Id=@Id OUT
...
```

En este caso, al poner el @Id al final, debemos decirle a la Stored Procedure que es cada cosa y podemos variar el orden.

GenUpdate: Por cada tabla, la Stored Procedure recibe los parámetros necesarios y actualiza el registro cuyo ID se envía.

```
create procedure genupdate
@tab varchar(128)
AS
BEGIN
    DECLARE @SQLTEXT VARCHAR(1024)
    DECLARE @WHERE VARCHAR(1024)
    DECLARE C_COL CURSOR FOR
    SELECT
        UPPER(c.name),
        CASE y.name
            WHEN 'VARCHAR' then 'VARCHAR(' + TRIM(STR(c.MAX_LENGTH)) + ')'
            WHEN 'CHAR' then 'CHAR(' + TRIM(STR(c.MAX_LENGTH)) + ')'
            ELSE UPPER(y.name)
```

```
        END,
        i.is_primary_key
    FROM sys.tables t
        INNER JOIN sys.columns c
              ON c.object_id = t.object_id
        INNER JOIN sys.types y
            ON y.user_type_id = c.user_type_id
        LEFT JOIN sys.index_columns ic
            on ic.object_id = t.object_id
            and ic.column_id = c.column_id
        LEFT JOIN sys.indexes i
            ON i.object_id = t.object_id
            and i.index_id = ic.index_id
    WHERE t.name = @tab

    DECLARE @COLNAME VARCHAR(128)
    DECLARE @COLTYPE VARCHAR(128)
    DECLARE @ISPK INT
    DECLARE @CNT INTEGER



    PRINT ''
    PRINT 'CREATE PROCEDURE U_' + @tab
    OPEN C_COL
    FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    SET @CNT = 0
    WHILE @@FETCH_STATUS = 0
    BEGIN
        BEGIN
            IF @CNT = 0
                BEGIN
                    PRINT '    @' + @COLNAME + ' ' + @COLTYPE
                    SET @CNT = 1
                END
            ELSE
                    PRINT '   ,@' + @COLNAME + ' ' + @COLTYPE
        END
        FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    END
    CLOSE C_COL

    PRINT 'AS'
    PRINT 'BEGIN'
    PRINT '  BEGIN TRY'
    PRINT '    UPDATE ' + @tab + ' SET'
    SET @CNT = 0
    OPEN C_COL
    FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @ISPK = 1
            BEGIN
                SET @WHERE = '    WHERE ' + UPPER(@COLNAME) + ' = ' + @COLNAME
            END
        ELSE
            IF @CNT = 0
                BEGIN
                    PRINT '    ' + @COLNAME + ' = ISNULL(@' + @COLNAME + ', ' +
@COLNAME + ')'
                    SET @CNT = 1
                END
            ELSE
```

```
                    PRINT '   ,' + @COLNAME + ' = ISNULL(@' + @COLNAME + ', ' +
@COLNAME + ')'

        FETCH C_COL INTO @COLNAME, @COLTYPE, @ISPK
    END
    PRINT @WHERE
    CLOSE C_COL
    DEALLOCATE C_COL
    PRINT '  END TRY'
    PRINT '  BEGIN CATCH'
    PRINT '     EXECUTE GetErrorInfo'
    PRINT '  END CATCH'
    PRINT 'END;'
END;
```

Ejemplo de ejecución de U_Persona:

```
declare @id int
exec I_Persona @id out, 'Arce', 'Juan', 13798792
select @id
select * from Persona where id = 123

id |apellido|nombre|dni     |
---+--------+------+--------+
123|Arce    |Juan  |13798792|


exec U_Persona @id=123, @nombre='Juan María'

select * from Persona where id = 123
id |apellido|nombre    |dni     |
---+--------+----------+--------+
123|Arce    |Juan María|13798792|
```

GetErrorInfo: Cubre la captura de errores. Se usa en los bloques Begin Catch/End Catch

```
CREATE PROCEDURE GetErrorInfo_sp
AS
BEGIN
SELECT
    ERROR_NUMBER() AS ErrorNumber
    ,ERROR_SEVERITY() AS ErrorSeverity
    ,ERROR_STATE() AS ErrorState
    ,ERROR_PROCEDURE() AS ErrorProcedure
    ,ERROR_LINE() AS ErrorLine
    ,ERROR_MESSAGE() AS ErrorMessage;
END;
```

SQLCMD

SQLCMD es la consola de SqlServer para acceder a la base de datos.
Facilita la ejecución de scripts para la creación de objetos, en nuestro caso, las Stored Procedures.

```
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i GetErrorInfo.sql
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i GenInsert.sql
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i GenUpdate.sql
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i GenDelete.sql
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i GenDrops.sql  > sqlcmddropall.sql
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i GenAll.sql  > sqlcmdcreateall.sql
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i sqlcmddropall.sql
sqlcmd -U sa -P Soler225 -S localhost -d Plan2024 -C -i sqlcmdcreateall.sql
```

Stored La lista anterior es la automoatizaciń de la creación de las Stored Procedures que generan a su vez, otras Stored Procedures.

- Sqlcmd es el comando
- -U El usuario
- -P La password
- -S El server
- -d La Base de Datos
- -C Para que no use la encriptación en la conexión
- -i Script de Entrada

Opcional: -o archivo_de_salida (A elección)
Hay que respetar mayúsculas y minúsculas.

El símbolo > hace que lo que saldria por pantalla vaya a parar al archivo que se nombra.