Implementar bases de datos relacionales y sus componentes según los requerimientos de la clientela.

Objetivos

- Data Definition Language (DDL)

  CREATE / ALTER / DROP •
- Data Manipulation Language (DML)

  INSERT / UPDATE / DELETE
- Query Language (QL)

  SELECT
- Data Control Language (DCL)

  GRANT / REVOKE

Objetos de base de datos

- ESQUEMA
- DATABASE
- TABLA
- TRIGGER
- VIEW
- PROCEDURE
- INDEX

Esquemas

- La mayoría de los RDBMSs soportan múltiples esquemas, que son una forma de agrupar lógicamente los objetos
- Un ERP podría tener esquemas GL (contabilidad general), AP (cuentas a pagar), AR (cuentas a cobrar), HR (recursos humanos), etc
- Normalmente hay esquemas Information_schema, SYS, SYSCAT, etc para los objetos del sistema, como el catálogo
- Los RDBMSs tienen objetos de base y objetos de esquema

DATABASE

- Una **base** de **datos** es una colección de información organizada de forma que un programa de ordenador pueda seleccionar rápidamente los fragmentos de **datos** que necesite.
- Las **bases** de **datos** tradicionales se organizan por campos, registros y archivos.

- **SQL** (por sus siglas en inglés **Structured Query Language**; en español **lenguaje de consulta estructurada**) es un lenguaje específico del dominio utilizado en programación, diseñado para administrar sistemas de gestión de bases de datos relacionales.

TABLA

- Las tablas son objetos de base de datos que contienen todos sus datos. En las tablas, los datos se organizan con arreglo a un formato de filas y columnas, similar al de una hoja de cálculo. Cada fila representa un registro único y cada columna un campo dentro del registro.

TRIGGERS

- Los **Triggers** o Disparadores son objetos que se asocian con tablas y se almacenan en la **base de datos**. Su nombre se deriva por el comportamiento que presentan en su funcionamiento, ya que se ejecutan cuando sucede algún evento sobre las tablas a las que se encuentra asociado.

VIEW

- Una **vista** es una consulta que se presenta como una tabla (virtual) a partir de un conjunto de tablas en una base de datos relacional.

PROCEDURE

- Un **procedimiento almacenado** (***stored procedure*** en inglés) es un programa (o procedimiento) almacenado físicamente en una base de datos. Su implementación varía de un gestor de bases de datos a otro. **La ventaja de un procedimiento almacenado** es que al ser ejecutado, en respuesta a una petición de usuario, es ejecutado directamente en el motor de bases de datos, el cual usualmente corre en un servidor separado. Como tal, posee acceso directo a los datos que necesita manipular y sólo necesita enviar sus resultados de regreso al usuario, deshaciéndose de la sobrecarga resultante de comunicar grandes cantidades de datos salientes y entrantes.

INDEX

- El **índice** de una **base de datos** es una estructura de**datos** que mejora la velocidad de las operaciones, por medio de identificador único de cada fila de una tabla, permitiendo un rápido acceso a los registros de una tabla en una **base de datos**.

TIPOS DE DATOS

Tabla 1. Tipos de datos en Microsoft SQL Server

Tipos de Datos en Microsoft SQL Server		
Alfanuméricos	**Numéricos**	**Fecha**
char	tinyint	smalldatetime
varchar	smallint	datetime
binary	int	timestamp
varbinary	bigint	date
text	float	time
image	money	datetime2
nchar	numeric	datetimeoffset
nvarchar	real	
ntext	decimal	
	bit	
	smallmoney	

TABLE 3-1 Exact Numeric Data Types

DATA TYPE	STORAGE SIZE	POSSIBLE VALUES	COMMENTS
tinyint	1 byte	0 to 255	Equal to the *byte* data type in most programming languages, cannot store negative values
smallint	2 bytes	–32768 to 32767	A signed 16-bit integer
int	4 bytes	–2,147,483,648 to 2,147,483,647	A signed 32-bit integer
bigint	8 bytes	–2E63 to 2E63 – 1	A signed 64-bit integer
decimal (precision, scale)	5 to 17 bytes depending on precision	–10E38 + 1 to 10E38 – 1	A decimal number containing up to 38 digits
numeric (precision, scale)	Functionally equivalent to the decimal data type		

TABLE 3-3 Approximate Numeric Data Types

DATA TYPE	STORAGE SIZE	POSSIBLE VALUES
float (n <= 24)	4 bytes	–3.40E38 to –1.18E-38, 0 and 1.18E-38 to 3.40E38
float (24 > n <= 53)	8 bytes	–1.79E308 to –2.23E-308, 0 and 2.23E-308 to 1.79E308
real	Functionally equivalent to *float(24)*	

Data Storage Type	BINARY is used to store the image files (BMP, TIFF, GIF, or JPEG format files) and binary data. **BINARY(N)** When **N** is not specified in a data definition or variable declaration statement, the default length is 1. When **N** is not specified with the CAST function, the default length is 30.	**VARBINARY(N)** When **N** is not specified in a data definition or variable declaration statement, the default length is 1. When **N** is not specified with the CAST function, the default length is 30. VARBINARY(MAX) data type is used to store images/pdf/word etc files and any data.
Declaring		

DDL

Data Definition Language

DDL

- CREATE [OBJETO] NAME
- DROP [OBJETO] NAME
- ALTER [OBJETO] NAME

DATABASE

- CREATE DATABASE tbClase7
- DROP DATABASE tbClase7
- ALTER DATABASE tbClase7 MODIFY NAME = tbClase07 ;

TABLA: Crear

- CREATE TABLE *table_name* (
  column1 datatype,
  column2 datatype,
  column3 datatype,
  ....
  );

- CREATE TABLE Persons (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
  );

TABLA: Eliminar

- DROP TABLE *table_name*;
- DROP TABLE Shippers;
- TRUNCATE TABLE *table_name*;
- TRUNCATE TABLE *tbPersona*;

Tarea: Editar

- ALTER TABLE *table_name*
  ADD *column_name datatype*;
- ALTER TABLE Customers
  ADD Email varchar(255);

- ALTER TABLE *table_name*
  DROP COLUMN *column_name*;
- ALTER TABLE Customers
  DROP COLUMN Email;

- ALTER TABLE *table_name*
  ALTER COLUMN *column_name datatype*;

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

- ALTER TABLE Persons
  ADD DateOfBirth date;

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Modificar y eliminar una columna

```
ALTER TABLE Persons
ALTER COLUMN DateOfBirth year;

ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

CONSTRAINTS

- Para asegurar la integridad de los datos almacenados en nuestras tablas, podemos crear restricciones, algunos los hemos utilizado sin querer o simplemente desconocemos que lo que hicimos fue una restricción, por ejemplo una llave primaria. Estas restricciones las podemos implementar al momento de crear nuestras tablas o de modificarlas, también es necesario señalar que dichas restricciones son objetos propios de la base de datos y por lo tanto requieren de un nombre único compuesto del nombre del esquema al que pertenece y el nombre que lo identifica, un ejemplo sería **nombreEsquema.nombreRestriccion.**

TIPOS

- PRIMARY KEY
- UNIQUE
- FOREIGN KEY
- CHECK
- DEFAULT

Primary Key

- Es la más común de todas debido a que cada una de nuestras tablas debe ser completamente relacional y para lograr esto siempre debe existir una llave primaria dentro de cada tabla que identifique cada fila como única.

```
1 CREATE TABLE nombreEsquema.nombreTabla
2 (
3  nombreColumna1 INT     NOT NULL,
4  nombreColumna2 VARCHAR(100)  NOT NULL,
5  nombreColumna3 NVARCHAR(200) NOT NULL,
6  CONSTRAINT PK_nombreRestriccion PRIMARY KEY( nombreColumna1 )
7 );
```

Modificando una tabla:

```
1 ALTER TABLE nombreEsquema.nombreTabla
2 ADD CONSTRAINT PK_nombreRestriccion PRIMARY KEY( nombreColumna1 );
```

UNIQUE

- Este tipo de restricción es muy parecida a PRIMARY KEY, las diferencias son las siguientes:
  - También genera un índice automáticamente pero es de tipo de NON CLUSTERED.
  - La tabla puede tener más de una restricción de tipo UNIQUE.
  - Si puede aceptar NULL, pero solo una fila puede contenerlo ya que como su nombre lo indica, es de tipo UNIQUE o único.

```
1  CREATE TABLE nombreEsquema.nombreTabla
2  (
3   nombreColumna1 INT     NULL,
4   nombreColumna2 VARCHAR(100)  NOT NULL,
5   nombreColumna3 NVARCHAR(200) NOT NULL,
6   CONSTRAINT UQ_nombreRestriccion UNIQUE( nombreColumna1 ),
7   CONSTRAINT UQ_nombreRestriccion2 UNIQUE( nombreColumna2 ),
8   CONSTRAINT UQ_nombreRestriccion3 UNIQUE( nombreColumna1,nombreColumna2 )
9  );
```

Para consultar las restricciones UNIQUE se puede utilizar:

```
1  SELECT *
2  FROM sys.key_constraints
3  WHERE type = 'UQ';
```

FOREIGN KEY

- Se forma de una columna o la combinación de varias columnas de una tabla que sirve como enlace hacia otra tabla donde en esta última, dicho enlace son la o las columnas que forman la PRIMARY KEY. En la primera tabla donde creamos la llave foránea es posible que existan valores duplicados de la/las columnas que conforman la llave primaria de la segunda tabla, además las columnas involucradas en la llave foránea deben tener el mismo tipo de datos que la llave primaria de la segunda tabla.

```
1 CREATE TABLE nombreEsquema.nombreTabla
2 (
3  nombreColumna1 INT     NULL,
4  nombreColumna2 VARCHAR(100)  NOT NULL,
5  nombreColumna3 NVARCHAR(200) NOT NULL,
6  CONSTRAINT FK_nombreRestriccion FOREIGN KEY (nombreColumna1) REFERENCES
   nombreEsquema.otraTabla (nombreColumna1)
7 );
```

Modificando la tabla:

```
1 ALTER TABLE nombreEsquema.nombreTabla
2 ADD CONSTRAINT FK_nombreRestriccion FOREIGN KEY(nombreColumna1)
3 REFERENCES nombreEsquema.otraTabla (nombreColumna1)
```

FOREING KEY

Algunos requerimientos para la restricción FOREIGN KEY:

- Los valores ingresados en la o las columnas de la llave foránea, deben existir en la tabla a la que se hace referencia en la o las columnas de la llave primaria.
- Solo se pueden hacer referencia a llaves primaria de tablas que se encuentren dentro de la misma base de datos.
- Puede hacer referencia a otra columnas de la misma tabla.
- Solo puede hacer referencia a columnas de restricciones PRIMARY KEY o UNIQUE.
- No se puede utilizar en tablas temporales.

```
1 SELECT *
2 FROM sys.foreign_keys
3 WHERE name = 'nombreEsquema.nombreTabla';
```

CHECK

- Con este tipo de restricción, se especifica que los valores ingresados en la columna deben cumplir la regla o formula especificada. Por ejemplo:

```
CREATE TABLE nombreEsquema.nombreTabla
(
 nombreColumna1 INT    NULL,
 nombreColumna2 VARCHAR(100)  NOT NULL,
 nombreColumna3 NVARCHAR(200) NOT NULL,
 --VALORES POSITIVOS
 CONSTRAINT CH_nombreRestriccion CHECK (nombreColumna1>=0),
 -- SOLO VALORES IGUALES A 10 20 30 40
 CONSTRAINT CH_nombreRestriccion2 CHECK (nombreColumna1 IN
(10,20,30,40)),
 --VALORES CONTENIDOS EN UN RANGO
 CONSTRAINT CH_nombreRestriccion3 CHECK (nombreColumna1>=1 AND
nombreColumna1 <=30)
);
```

Modificando una tabla:

```
ALTER TABLE nombreEsquema.nombreTabla
ADD CONSTRAINT CH_nombreRestriccion CHECK (nombreColumna1>=0);
GO

ALTER TABLE nombreEsquema.nombreTabla
ADD CONSTRAINT CH_nombreRestriccion2 CHECK (nombreColumna1 IN
(10,20,30,40));
GO

ALTER TABLE nombreEsquema.nombreTabla
ADD CONSTRAINT CH_nombreRestriccion3 CHECK (nombreColumna1>=1 AND
nombreColumna1 <=30);
GO
```

CHECK

Algunos requerimientos son:

- Una columna puede tener cualquier número de restricciones CHECK.
- La condición de búsqueda debe evaluarse como una expresión booleana y no puede hacer referencia a otra tabla.
- No se pueden definir restricciones CHECK en columnas de tipo text, ntext o image

DEFAULT

- Se puede decir que no es una restricción, ya que solo se ingresa un valor en caso de que ninguno otro sea especificado. Si una columna permite NULL y el valor a insertar no se especifica, se puede sustituir con un valor predeterminado.

```
1 CREATE TABLE nombreTabla
2 (
3  nombreColumna1 INT     NULL CONSTRAINT DF_nombreRestriccion DEFAULT(0),
4  nombreColumna2 VARCHAR(100)  NOT NULL,
5  nombreColumna3 NVARCHAR(200) NOT NULL,
6 );
```

Para obtener una lista de las restricciones DEFAULT:

```
1 SELECT *
2 FROM sys.default_constraints
3 WHERE parent_object_id = OBJECT_ID('nombreEsquema.nombreTabla');
```