**Proyecto Final Base De Datos Avanzada**

***Carmen María Sánchez Tolentino 2018-5862***

***Dawin Rafael Perez Batista 2018-6484***

***Nizar Sánchez 2018-6482***

**1- Elabora una síntesis que trate los temas de integridad, seguridad y disponibilidad de los datos.**

**Integridad**

El término integridad de datos se refiere la correctitud y completitud de la información en una base de datos. Cuando los contenidos se modifican con sentencias INSERT, DELETE o UPDATE, la integridad de los datos almacenados puede perderse de muchas maneras diferentes. Una de las funciones importantes de un DBMS relacional es **preservar** la integridad de sus datos almacenados en la mayor medida posible.

Las restricciones de integridad las podemos agrupar en **cuatro** grupos:

**a) Integridad del dominio:** La integridad de dominio es la validez de las restricciones que debe cumplir una determinada columna de la tabla.

-Datos Requeridos: establece que una columna tenga un valor no NULL. Se define efectuando la declaración de una columna es NOT NULL cuando la tabla que contiene las columnas se crea por primera vez, como parte de la sentencia CREATE TABLE.

-Chequeo de Validez: cuando se crea una tabla cada columna tiene un tipo de datos y el DBMS asegura que solamente los datos del tipo especificado sean ingresados en la tabla.

**b) Entidad:** Son reglas de integridad que aplican a la totalidad de los registros de la tabla, como un conjunto. Por ejemplo: Si validamos que el valor de un campo no se repita, estamos diciendo que solo un registro puede tener un determinado valor en ese campo. Como reglas de este nivel tenemos el PRIMARY KEY y el UNIQUE INDEX.

**c) Referencial:** asegura la integridad entre las llaves foráneas y primarias (relaciones padre/hijo). Existen cuatro actualizaciones de la base de datos que pueden corromper la integridad referencial:

-La inserción de una fila hijo se produce cuando no coincide la llave foránea con la llave primaria del padre.

-La actualización en la llave foránea de la fila hijo, donde se produce una actualización en la clave ajena de la fila hijo con una sentencia UPDATE y la misma no coincide con ninguna llave primaria.

-La supresión de una fila padre, con la que, si una fila padre -que tiene uno o más hijos- se suprime, las filas hijos quedarán huérfanas.

-La actualización de la clave primaria de una fila padre, donde si en una fila padre, que tiene uno o más hijos se actualiza su llave primaria, las filas hijos quedarán huérfanas.

**d) Del usuario:** Se refiere a restricciones programadas por el usuario utilizando herramientas como los “Triggers” y “Procedimiento almacenados”.

**Seguridad**

Un buen entorno de seguridad para Sql Server se fundamenta en la creación de conjunto de políticas y procedimientos orientados a garantizar la protección de los datos. Estos procedimientos deben trabajar los siguientes aspectos:

-La información es accedida solo por los usuarios autorizados.

-El almacenamiento de los datos es seguro.

-El tránsito de los datos es seguro.

-La data es recuperable.

**PRINCIPIOS DE SEGURIDAD**

Los principios de seguridad proporcionan la base sobre la cual se fundamentan todas las soluciones de seguridad. A continuación, una lista de algunos principios recomendados:

***-Comienza con un buen diseño.*** Debemos implementar sistemas que sean “Seguros por

defecto”. Sistemas que estén abiertos para recibir y aceptar cualquier cambio necesario en

característica o funcionalidades. Debemos trabajar sobre un diseño apropiado de la

infraestructura de red.

***-Confía, pero vigila.*** Aunque tengamos previstos ataques o amenazas conocidos debemos

protegernos contra los ataques que aún no conocemos. Tecnologías como Antivirus y Antispyware

puede ayudar.

***-Defensa a fondo (Defense-in-Depth***). Nunca debemos confiarnos en una sola solución de

seguridad. Por ejemplo, no debemos confiar solo en el Firewall para protegernos de ataques por

internet. No debemos confiar solo en el Active Directory para protegernos de ataques internos.

Por el contrario, debemos considerar una seguridad basada en niveles:

o Sensibilización de usuarios

o Seguridad en el cliente

o Seguridad en la red

o Seguridad en el servidor

o Seguridad perimetral

***- Menor privilegio***. Por defecto solo con lo indispensable.

o Sensibilización de usuarios

o Seguridad en el cliente

o Seguridad en la red

o Seguridad en el servidor

o Seguridad perimetral

**ASPECTOS DE SEGURIDAD EN LA ARQUITECTURA DE SQL SERVER**

Para conocer el modo de operar de la seguridad en SQL server se debe de conocer los siguientes aspectos:

• ***Autenticación***. Determinar la identidad del usuario.

• ***Autorización***. Determinar a qué objetos o funciones tiene acceso.

• ***Integridad***. Asegurar que la información no se modifique en el transito (Firma digital).

• ***Privacidad***. Asegurar que la información solo puede ser leída por el destinatario ( Cifrado).

**Disponibilidad de los datos**

**Alta Disponibilidad (High Availability)**

Se refiere al protocolo seguido para garantizar la continuidad del sistema. Que ese tiempo de inactividad (Downtime) sea el mínimo permitido. El tiempo de inactividad puede ser **planificado**: se refiere a interrupciones lógicas y por lo regular programadas, cosas aplicar actualizaciones, cambio de alguna configuración que requiera reinicio del sistema, etc. y **no** **planificado**: se refiere a interrupciones caudadas por imprevistos como son los errores de Software o Hardware, etc.

Todo sistema de tener un acuerdo de servicio (SLA) que defina el tiempo que el sistema debe estar en línea. Sistema de bajo criticidad puede ofrecer una disponibilidad de 8x5 (8 días por 5 horas). En sistemas críticos el nivel de servicio puede ser de 24/7 los 365 días del año (24\*365). La siguiente formula nos permite calcular la disponibilidad del sistema:

Disponibilidad = ((A-B) /A) \* 100%

Donde: A = Horas comprometidas. Eje: 24\*365 = 8,760 horas/año

B = Downtime durante las horas compromietidas. Eje: 15 horas por fallas de discos, 9 por mantenimientos = 24.

Entonces: D = ((8760-24) /8760 \* 100 = 99.726%

**Niveles de Alta Disponibilidad**

**Convencional**: 90% de disponibilidad. La continuidad e integridad no son un tema crítico. Servidor tradicional con técnicas de recuperación y copias de seguridad.

**Media (High Reliable):** 95% de disponibilidad. Se pueden tolerar algunas interrupciones, pero la integridad es un tema crítico. Se utilizan esquema de protección transaccionales.

**Alta disponibilidad:** 99% de disponibilidad. Se soportan interrupciones muy breves. Se utilizan esquema de protección transaccional y recuperación automática. Resistente a fallas. 99.9% de disponibilidad. El sistema debe trabajar sin interrupciones en horario laboral. Se utilizan esquemas de “mirroring” y “Clustering”.

**Tolerancia a fallas:** 99.99% de disponibilidad. El procesamiento es continuo y la recuperación ante fallas es transparente al usuario. Se utilizan esquemas de duplicidad y redundancia.

**Tolerancia a desastres:** 99.999% (los cinco 9s). Disponibilidad en todo momento. Recuperación transparente, incluso ante desastre naturales o humanos. Se utilizan todos los esquemas anteriores, con duplicidad extra.

**Técnicas de Alta Disponibilidad**

Se refiere a los procedimientos o prácticas que permiten que un sistema no deje de estar en funcionamiento y accesible en ningún momento o que ese tiempo de inactividad sea de un porcentaje de tiempo realmente mínimo. También nos referimos a estas técnicas como “soluciones de alta disponibilidad”. Algunas técnicas que ayudan en el proceso de proveer alta disponibilidad en las bases de datos son las siguientes:

-**Replicación**: permite la copia automática de data perteneciente a objetos seleccionados de la base de datos hacia otros servidores locales o remotos.

**-Espejos (Mirroring):** permite tener una copia exacta de nuestra base de datos en otro servidor.

**-Clustering**: es una agrupación de sistemas o componentes (Hardware o Software) que trabajan de forma combinada, como si fuera uno solo, con el objetivo de ofrecer rendimiento, escalabilidad, pero también alta disponibilidad.

**2. Selecciona una empresa, negocio o proyecto al que podamos diseñar la base de datos.**

Requerimientos del sistema para un Rent-a-car

Se necesita guardar la información básica del cliente como lo son el nombre completo, correo electrónico, No. Identificación, código postal, país, contraseña, etc. Un usuario solo puede pertenecer a un cliente o a un empleado. También se necesita almacenar los datos del empleado como lo son el sueldo, y su posición. Un empleado solo puede trabajar en una sucursal mientras que en una sucursal pueden trabajar varios empleados, de igual manera que una sucursal solo puede tener un supervisor.

Cada cliente puede tener varias reservaciones y cada reservación puede tener varios complementos extras. Un vehículo puede pertenecer a varias reservaciones. Un vehículo puede estar en varias sucursales almacenando la fecha en la cual llegó y en la fecha en la que salió de la sucursal. El mismo tipo de vehículo puede estar también en varias sucursales.

Para poder reservar un vehículo se necesita la dirección de recogida, dirección de entrega, fecha y hora de recogida y fecha y hora de entrega, dependiendo de esto se consultarán los vehículos que estén disponibles en la sucursal de recogida y que también estén disponibles dentro de ese intervalo de tiempo. Luego pasara a coger que protecciones y/o equipamientos extras que desea tener.

Luego se generará un reporte con los detalles del vehículo, protecciones y equipamientos.

Finalmente se generará un baucher con el cual debe ir a recoger el vehículo.

A close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='TRADITIONAL,ALLOW\_INVALID\_DATES';

CREATE SCHEMA IF NOT EXISTS `Rent-a-car` DEFAULT CHARACTER SET utf8 ;

USE `Rent-a-car` ;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Persona` (

  `noIdentificacion` CHAR(13) NOT NULL,

  `direccion` VARCHAR(100) NULL,

  `nombre` VARCHAR(45) NULL,

  `apellido` VARCHAR(45) NULL,

  PRIMARY KEY (`noIdentificacion`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Usuario` (

  `correoElectronico` VARCHAR(100) NOT NULL,

  `contrasena` VARCHAR(45) NULL,

  `noIdentificacionPersona` CHAR(13) NOT NULL,

  PRIMARY KEY (`correoElectronico`),

  INDEX `fk\_Usuario\_Persona1\_idx` (`noIdentificacionPersona` ASC),

  CONSTRAINT `fk\_Usuario\_Persona1`

    FOREIGN KEY (`noIdentificacionPersona`)

    REFERENCES `Rent-a-car`.`Persona` (`noIdentificacion`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`TelefonoPersona` (

  `numero` VARCHAR(15) NOT NULL,

  `noIDentificacionPersona` CHAR(13) NOT NULL,

  PRIMARY KEY (`numero`, `noIDentificacionPersona`),

  INDEX `fk\_TelefonoPersona\_Persona\_idx` (`noIDentificacionPersona` ASC),

  CONSTRAINT `fk\_TelefonoPersona\_Persona`

    FOREIGN KEY (`noIDentificacionPersona`)

    REFERENCES `Rent-a-car`.`Persona` (`noIdentificacion`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Cliente` (

  `licenciaDeConducir` INT NULL,

  `idCliente` INT NOT NULL AUTO\_INCREMENT,

  `codigo\_postal` VARCHAR(10) NULL,

  `Pais` VARCHAR(20) NULL,

  `noIdentificacionPersona` CHAR(13) NOT NULL,

  PRIMARY KEY (`idCliente`),

  INDEX `fk\_Cliente\_Persona1\_idx` (`noIdentificacionPersona` ASC),

  CONSTRAINT `fk\_Cliente\_Persona1`

    FOREIGN KEY (`noIdentificacionPersona`)

    REFERENCES `Rent-a-car`.`Persona` (`noIdentificacion`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`TipoVehiculo` (

  `idTipoVehiculo` INT NOT NULL AUTO\_INCREMENT,

  `foto` VARCHAR(100) NULL,

  `descripcion` VARCHAR(100) NULL,

  `precioActual` DECIMAL(10,4) NULL,

  `tipoTransmision` VARCHAR(10) NULL,

  `tipoCombustible` VARCHAR(10) NULL,

  `noPuertas` CHAR(1) NULL,

  PRIMARY KEY (`idTipoVehiculo`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Empleado` (

  `idEmpleado` INT NOT NULL AUTO\_INCREMENT,

  `sueldo` DECIMAL(10,4) NULL,

  `posicion` VARCHAR(45) NULL,

  `noIdentificacionPersona` CHAR(13) NOT NULL,

  `idSucursal` INT NOT NULL,

  PRIMARY KEY (`idEmpleado`),

  INDEX `fk\_Empleado\_Persona1\_idx` (`noIdentificacionPersona` ASC),

  INDEX `fk\_Empleado\_Sucursal1\_idx` (`idSucursal` ASC),

  CONSTRAINT `fk\_Empleado\_Persona1`

    FOREIGN KEY (`noIdentificacionPersona`)

    REFERENCES `Rent-a-car`.`Persona` (`noIdentificacion`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Empleado\_Sucursal1`

    FOREIGN KEY (`idSucursal`)

    REFERENCES `Rent-a-car`.`Sucursal` (`idSucursal`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Sucursal` (

  `idSucursal` INT NOT NULL AUTO\_INCREMENT,

  `direccion` VARCHAR(100) NULL,

  `idEmpleadoSupervisor` INT NOT NULL,

  PRIMARY KEY (`idSucursal`),

  INDEX `fk\_Sucursal\_Empleado1\_idx` (`idEmpleadoSupervisor` ASC),

  CONSTRAINT `fk\_Sucursal\_Empleado1`

    FOREIGN KEY (`idEmpleadoSupervisor`)

    REFERENCES `Rent-a-car`.`Empleado` (`idEmpleado`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Vehiculo` (

  `idVehiculo` INT NOT NULL AUTO\_INCREMENT,

  `placa` CHAR(7) NOT NULL,

  `year` INT NULL,

  `color` VARCHAR(20) NULL,

  `modelo` VARCHAR(20) NULL,

  `marca` VARCHAR(20) NULL,

  `noChasis` CHAR(17) NOT NULL,

  `idTipoVehiculo` INT NOT NULL,

  `idSucursal` INT NOT NULL,

  PRIMARY KEY (`idVehiculo`),

  UNIQUE INDEX `no\_chasis\_UNIQUE` (`noChasis` ASC),

  UNIQUE INDEX `placa\_UNIQUE` (`placa` ASC),

  INDEX `fk\_Vehiculo\_TipoVehiculo1\_idx` (`idTipoVehiculo` ASC),

  INDEX `fk\_Vehiculo\_Sucursal1\_idx` (`idSucursal` ASC),

  CONSTRAINT `fk\_Vehiculo\_TipoVehiculo1`

    FOREIGN KEY (`idTipoVehiculo`)

    REFERENCES `Rent-a-car`.`TipoVehiculo` (`idTipoVehiculo`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Vehiculo\_Sucursal1`

    FOREIGN KEY (`idSucursal`)

    REFERENCES `Rent-a-car`.`Sucursal` (`idSucursal`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Reservacion` (

  `idReservacion` INT NOT NULL AUTO\_INCREMENT,

  `fechaInicio` DATETIME NULL,

  `fechaFin` DATETIME NULL,

  `idCliente` INT NOT NULL,

  `precioVehiculo` DECIMAL(10,4) NULL,

  `idVehiculo` INT NOT NULL,

  PRIMARY KEY (`idReservacion`, `idVehiculo`),

  INDEX `fk\_Reservacion\_Cliente1\_idx` (`idCliente` ASC),

  INDEX `fk\_Reservacion\_Vehiculo1\_idx` (`idVehiculo` ASC),

  CONSTRAINT `fk\_Reservacion\_Cliente1`

    FOREIGN KEY (`idCliente`)

    REFERENCES `Rent-a-car`.`Cliente` (`idCliente`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Reservacion\_Vehiculo1`

    FOREIGN KEY (`idVehiculo`)

    REFERENCES `Rent-a-car`.`Vehiculo` (`idVehiculo`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`Extra` (

  `idExtra` INT NOT NULL AUTO\_INCREMENT,

  `precioActual` DECIMAL(10,4) NULL,

  `descripcion` VARCHAR(45) NULL,

  PRIMARY KEY (`idExtra`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`ResevacionHasExtra` (

  `Reservacion\_idReservacion` INT NOT NULL,

  `idExtra` INT NOT NULL,

  `precioExtra` DECIMAL(10,4) NULL,

  PRIMARY KEY (`Reservacion\_idReservacion`, `idExtra`),

  INDEX `fk\_Reservacion\_has\_Extra\_Extra1\_idx` (`idExtra` ASC),

  INDEX `fk\_Reservacion\_has\_Extra\_Reservacion1\_idx` (`Reservacion\_idReservacion` ASC),

  CONSTRAINT `fk\_Reservacion\_has\_Extra\_Reservacion1`

    FOREIGN KEY (`Reservacion\_idReservacion`)

    REFERENCES `Rent-a-car`.`Reservacion` (`idReservacion`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Reservacion\_has\_Extra\_Extra1`

    FOREIGN KEY (`idExtra`)

    REFERENCES `Rent-a-car`.`Extra` (`idExtra`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `Rent-a-car`.`SucursalHasTIpoVehiculo` (

  `idSucursal` INT NOT NULL,

  `idTipoVehiculo` INT NOT NULL,

  PRIMARY KEY (`idSucursal`, `idTipoVehiculo`),

  INDEX `fk\_Sucursal\_has\_TipoVehiculo\_TipoVehiculo1\_idx` (`idTipoVehiculo` ASC),

  INDEX `fk\_Sucursal\_has\_TipoVehiculo\_Sucursal1\_idx` (`idSucursal` ASC),

  CONSTRAINT `fk\_Sucursal\_has\_TipoVehiculo\_Sucursal1`

    FOREIGN KEY (`idSucursal`)

    REFERENCES `Rent-a-car`.`Sucursal` (`idSucursal`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION,

  CONSTRAINT `fk\_Sucursal\_has\_TipoVehiculo\_TipoVehiculo1`

    FOREIGN KEY (`idTipoVehiculo`)

    REFERENCES `Rent-a-car`.`TipoVehiculo` (`idTipoVehiculo`)

    ON DELETE NO ACTION

    ON UPDATE NO ACTION)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

**3. DDL/DML**

**• Crea la base de datos con las restricciones de integridad requeridas.**

create database BancoXY

go

use BancoXY

go

create table Pais(

Id\_pais int primary key identity(1,1),

Nombre varchar(20) not null

)

go

create table Cliente (

Id\_cliente int primary key identity(1,1),

Nombre varchar(50) not null,

IdPais int not null foreign key references Pais(Id\_pais)

)

go

create table Cuenta(

Id\_cuenta int primary key identity(1,1),

Id\_cliente int not null foreign key references Cliente(Id\_cliente),--Asumiendo que Nombre hace referencia a la tabla cliente se

--reemplazo por el campo Id\_cliente la cual hace referencia a dicha tabla

Tipo int not null check (Tipo in(1,2))--Donde 1 es cuenta de Ahorro y 2 es cuenta de Prestamo

)

go

Create table Ahorro(

Id\_cuenta int primary key foreign key references Cuenta(Id\_cuenta),

Monto\_apertura decimal(12,1) not null,

Balance decimal (12,1) not null

)

go

create table Prestamo(

Id\_cuenta int primary key foreign key references Cuenta(Id\_cuenta),

CantidadCuotas int not null,

Monto decimal(12,1) not null,

Saldo decimal(12,1) not null

)

go

create table PagoPrestamo(

Id\_pago int primary key identity(1,1),

Id\_cuenta int foreign key references Prestamo(Id\_cuenta),

Monto decimal(12,1) not null,

Fecha datetime default getdate() not null

)

**• Crea una función que partiendo del tipo de la cuenta me retorne la descripción.**

create function descripcion\_cuenta(@tipo int)

returns varchar(50)

begin

if @tipo=1

begin

return 'Descripcion de cuenta de ahorro'

end

else if @tipo=2

begin

return 'Descripcion de cuenta de prestamo'

end

return 'El tipo de cuenta especificado no se ha encontrado en la base de datos'

end

select dbo.descripcion\_cuenta(1)

**A screenshot of a social media post

Description automatically generated**

**• Crea un procedimiento almacenado que dependiendo del tipo de cuenta retorne el balance o**

**saldo.**

create procedure PA\_retorna\_balance\_saldo

@id\_cuenta int

as

begin

declare @tipo int

select @tipo=tipo from Cuenta where Cuenta.Id\_cuenta=@id\_cuenta

if @tipo=1

begin

Select \* from Ahorro where Id\_cuenta=@id\_cuenta

end

else

begin

Select \* from Prestamo where Id\_cuenta=@id\_cuenta

end

end

exec PA\_retorna\_balance\_saldo 1

**A screenshot of a social media post

Description automatically generated**

**• Crea una consulta que genere un listado de los clientes con cada préstamo que ha tomado y el**

**total pagado.**

select Cliente.Nombre,Count(Prestamo.Id\_cuenta) as Prestamos,sum(PagoPrestamo.Monto) as 'Total Pagado' from Prestamo

inner join Cuenta on Cuenta.Id\_cuenta=Prestamo.Id\_cuenta

inner join Cliente on Cuenta.Id\_cliente=Cliente.Id\_cliente

inner join PagoPrestamo on PagoPrestamo.Id\_cuenta=Prestamo.Id\_cuenta

group by Cliente.Nombre

**A screenshot of a social media post

Description automatically generated**