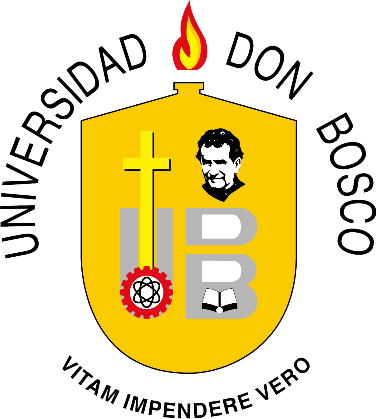
**UNIVERSIDAD**

**DON BOSCO**



**FACULTAD DE INGENIERÍA**

# Desafio2

**Asignatura:**

**DSM**

**Grupo de laboratorio:**

**G01L**

**Docente:**

**Mario Alvarado**

**Presentado por:**

**Joel Alexander Flores Hernández**

**LINK DEL VIDEO DEMOSTRACTIVO:  
https://youtu.be/DgrW3SmnLZs**

<https://github.com/Joelfloreshz/DSM>

class HelperDB(context: Context): SQLiteOpenHelper(context, "CarsMotors.db", null, 1) {  
  
 override fun onCreate(db: SQLiteDatabase?) {  
 db?.execSQL(  
 "CREATE TABLE marcas (" +  
 "idmarcas INTEGER PRIMARY KEY," +  
 "nombre VARCHAR(45))"  
 )  
  
 db?.execSQL(  
 "CREATE TABLE colores (" +  
 "idcolores INTEGER PRIMARY KEY," +  
 "descripcion VARCHAR(45))"  
 )  
  
 db?.execSQL(  
 "CREATE TABLE tipo\_automovil (" +  
 "idtipoautomovil INTEGER PRIMARY KEY," +  
 "descripcion VARCHAR(45))"  
 )  
  
 db?.execSQL(  
 "CREATE TABLE automovil (" +  
 "idautomovil INTEGER PRIMARY KEY," +  
 "modelo VARCHAR(45)," +  
 "numero\_vin VARCHAR(45)," +  
 "numero\_chasis VARCHAR(45)," +  
 "numero\_motor VARCHAR(45)," +  
 "numero\_asientos INTEGER," +  
 "anio YEAR," +  
 "capacidad\_asientos INTEGER," +  
 "precio DECIMAL(10,2)," +  
 "uri\_img VARCHAR(45)," +  
 "descripcion VARCHAR(45)," +  
 "idmarcas INTEGER," +  
 "idtipoautomovil INTEGER," +  
 "idcolores INTEGER," +  
 "FOREIGN KEY (idmarcas) REFERENCES marcas(idmarcas)," +  
 "FOREIGN KEY (idtipoautomovil) REFERENCES tipo\_automovil(idtipoautomovil)," +  
 "FOREIGN KEY (idcolores) REFERENCES colores(idcolores))"  
 )  
  
 db?.execSQL(  
 "CREATE TABLE favoritos\_automovil (" +  
 "idfavoritosautomovil INTEGER PRIMARY KEY," +  
 "idusuario INTEGER," +  
 "idfavoritoautomovil INTEGER," +  
 "fecha\_agregado TIMESTAMP," +  
 "FOREIGN KEY (idusuario) REFERENCES usuario(idusuario)," +  
 "FOREIGN KEY (idfavoritoautomovil) REFERENCES automovil(idautomovil))"  
 )  
  
 db?.execSQL(  
 "CREATE TABLE usuario (" +  
 "idusuario INTEGER PRIMARY KEY," +  
 "nombres VARCHAR(45)," +  
 "apellidos VARCHAR(45)," +  
 "email VARCHAR(45)," +  
 "user VARCHAR(45)," +  
 "password VARCHAR(45)," +  
 "tipo VARCHAR(45)," +  
 "UNIQUE(user))"  
 )  
 }  
  
 override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {  
 db?.execSQL("DROP TABLE IF EXISTS marcas")  
 db?.execSQL("DROP TABLE IF EXISTS colores")  
 db?.execSQL("DROP TABLE IF EXISTS tipo\_automovil")  
 db?.execSQL("DROP TABLE IF EXISTS automovil")  
 db?.execSQL("DROP TABLE IF EXISTS favoritos\_automovil")  
 db?.execSQL("DROP TABLE IF EXISTS usuario")  
 onCreate(db)  
 }  
  
 fun insertUsuario(usuario: Usuario): Long {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("nombres", usuario.nombres)  
 put("apellidos", usuario.apellidos)  
 put("email", usuario.correoElectronico)  
 put("user", usuario.usuario)  
 put("password", usuario.contrasena)  
 put("tipo", usuario.tipo)  
 **}** val id = db.insert("usuario", null, values)  
 db.close()  
 return id  
 }  
  
  
 @SuppressLint("Range")  
 fun getUsuario(username: String, password: String): Usuario? {  
 val db = *readableDatabase* val columns =  
 *arrayOf*("idusuario", "nombres", "apellidos", "email", "user", "password", "tipo")  
 val selection = "user = ? AND password = ?"  
 val selectionArgs = *arrayOf*(username, password)  
 val cursor: Cursor =  
 db.query("usuario", columns, selection, selectionArgs, null, null, null)  
 var usuario: Usuario? = null  
 if (cursor.moveToFirst()) {  
 val idusuario = cursor.getInt(cursor.getColumnIndex("idusuario"))  
 val nombres = cursor.getString(cursor.getColumnIndex("nombres"))  
 val apellidos = cursor.getString(cursor.getColumnIndex("apellidos"))  
 val email = cursor.getString(cursor.getColumnIndex("email"))  
 val user = cursor.getString(cursor.getColumnIndex("user"))  
 val tipo = cursor.getString(cursor.getColumnIndex("tipo"))  
 usuario = Usuario(idusuario, nombres, apellidos, email, user, password, tipo)  
 }  
 cursor.close()  
 db.close()  
 return usuario  
 }  
 fun updateUsuario(usuario: Usuario): Int {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("nombres", usuario.nombres)  
 put("apellidos", usuario.apellidos)  
 put("email", usuario.correoElectronico)  
 put("user", usuario.usuario)  
 put("password", usuario.contrasena)  
 put("tipo", usuario.tipo)  
 **}** val selection = "idusuario = ?"  
 val selectionArgs = *arrayOf*(usuario.id.toString())  
 val count = db.update("usuario", values, selection, selectionArgs)  
 db.close()  
 return count  
 }  
  
 fun deleteUsuario(id: Int): Int {  
 val db = *writableDatabase* val selection = "idusuario = ?"  
 val selectionArgs = *arrayOf*(id.toString())  
 val count = db.delete("usuario", selection, selectionArgs)  
 db.close()  
 return count  
 }  
 @SuppressLint("Range")  
 fun buscarUsuarios(query: String): List<Usuario> {  
 val usuarios = *mutableListOf*<Usuario>()  
 val db = *readableDatabase* val columns =  
 *arrayOf*("idusuario", "nombres", "apellidos", "email", "user", "password", "tipo")  
 val selection = "nombres LIKE ? OR apellidos LIKE ? OR email LIKE ?"  
 val selectionArgs = *arrayOf*("%$query%", "%$query%", "%$query%")  
 val cursor: Cursor = db.query("usuario", columns, selection, selectionArgs, null, null, null)  
 while (cursor.moveToNext()) {  
 val idusuario = cursor.getInt(cursor.getColumnIndex("idusuario"))  
 val nombres = cursor.getString(cursor.getColumnIndex("nombres"))  
 val apellidos = cursor.getString(cursor.getColumnIndex("apellidos"))  
 val email = cursor.getString(cursor.getColumnIndex("email"))  
 val user = cursor.getString(cursor.getColumnIndex("user"))  
 val password = cursor.getString(cursor.getColumnIndex("password"))  
 val tipo = cursor.getString(cursor.getColumnIndex("tipo"))  
 val usuario = Usuario(idusuario, nombres, apellidos, email, user, password, tipo)  
 usuarios.add(usuario)  
 }  
 cursor.close()  
 db.close()  
 return usuarios  
 }  
  
  
  
 // Métodos para la tabla "marcas"  
  
 fun insertMarca(nombre: String): Long {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("nombre", nombre)  
 **}** val id = db.insert("marcas", null, values)  
 db.close()  
 return id  
 }  
  
 @SuppressLint("Range")  
 fun getMarca(idMarca: Int): Marca? {  
 val db = *readableDatabase* val columns = *arrayOf*("nombre", "pais")  
 val selection = "idmarca = ?"  
 val selectionArgs = *arrayOf*(idMarca.toString())  
 val cursor = db.query("marca", columns, selection, selectionArgs, null, null, null)  
 var marca: Marca? = null  
 if (cursor.moveToFirst()) {  
 val nombre = cursor.getString(cursor.getColumnIndex("nombre"))  
 val pais = cursor.getString(cursor.getColumnIndex("pais"))  
 marca = Marca(idMarca, nombre, pais)  
 }  
 cursor.close()  
 db.close()  
 return marca  
 }  
  
 fun updateMarca(marca: Marca): Int {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("nombre", marca.nombre)  
 **}** val selection = "idmarcas = ?"  
 val selectionArgs = *arrayOf*(marca.id.toString())  
 val count = db.update("marcas", values, selection, selectionArgs)  
 db.close()  
 return count  
 }  
  
 fun deleteMarca(id: Int): Int {  
 val db = *writableDatabase* val selection = "idmarcas = ?"  
 val selectionArgs = *arrayOf*(id.toString())  
 val count = db.delete("marcas", selection, selectionArgs)  
 db.close()  
 return count  
 }  
  
 // Métodos para la tabla "colores"  
  
 fun insertColor(descripcion: String): Long {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("descripcion", descripcion)  
 **}** val id = db.insert("colores", null, values)  
 db.close()  
 return id  
 }  
  
 @SuppressLint("Range")  
 fun getColor(idColor: Int): Color? {  
 val db = *readableDatabase* val columns = *arrayOf*("nombre", "codigo\_hexadecimal")  
 val selection = "idcolor = ?"  
 val selectionArgs = *arrayOf*(idColor.toString())  
 val cursor = db.query("color", columns, selection, selectionArgs, null, null, null)  
 var color: Color? = null  
 if (cursor.moveToFirst()) {  
 val nombre = cursor.getString(cursor.getColumnIndex("nombre"))  
 val codigoHexadecimal = cursor.getString(cursor.getColumnIndex("codigo\_hexadecimal"))  
 color = Color(idColor, nombre, codigoHexadecimal)  
 }  
 cursor.close()  
 db.close()  
 return color  
 }  
  
 fun updateColor(color: Color): Int {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("descripcion", color.descripcion)  
 **}** val selection = "idcolores = ?"  
 val selectionArgs = *arrayOf*(color.id.toString())  
 val count = db.update("colores", values, selection, selectionArgs)  
 db.close()  
 return count  
 }  
  
 fun deleteColor(id: Int): Int {  
 val db = *writableDatabase* val selection = "idcolores = ?"  
 val selectionArgs = *arrayOf*(id.toString())  
 val count = db.delete("colores", selection, selectionArgs)  
 db.close()  
 return count  
 }  
  
 // Métodos para la tabla "tipo\_automovil"  
  
 fun insertTipoAutomovil(descripcion: String): Long {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("descripcion", descripcion)  
 **}** val id = db.insert("tipo\_automovil", null, values)  
 db.close()  
 return id  
 }  
  
 @SuppressLint("Range")  
 fun getTipoAutomovil(idTipoAutomovil: Int): TipoAutomovil? {  
 val db = *readableDatabase* val columns = *arrayOf*("nombre", "descripcion")  
 val selection = "idtipoautomovil = ?"  
 val selectionArgs = *arrayOf*(idTipoAutomovil.toString())  
 val cursor = db.query("tipoautomovil", columns, selection, selectionArgs, null, null, null)  
 var tipoAutomovil: TipoAutomovil? = null  
 if (cursor.moveToFirst()) {  
 val nombre = cursor.getString(cursor.getColumnIndex("nombre"))  
 val descripcion = cursor.getString(cursor.getColumnIndex("descripcion"))  
 tipoAutomovil = TipoAutomovil(idTipoAutomovil, nombre, descripcion)  
 }  
 cursor.close()  
 db.close()  
 return tipoAutomovil  
 }  
  
 fun updateTipoAutomovil(tipoAutomovil: TipoAutomovil): Int {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("descripcion", tipoAutomovil.descripcion)  
 **}** val selection = "idtipoautomovil = ?"  
 val selectionArgs = *arrayOf*(tipoAutomovil.id.toString())  
 val count = db.update("tipo\_automovil", values, selection, selectionArgs)  
 db.close()  
 return count  
 }  
  
 fun deleteTipoAutomovil(id: Int): Int {  
 val db = *writableDatabase* val selection = "idtipoautomovil = ?"  
 val selectionArgs = *arrayOf*(id.toString())  
 val count = db.delete("tipo\_automovil", selection, selectionArgs)  
 db.close()  
 return count  
 }  
  
  
 // Métodos para la tabla "automovil"  
  
 // Método Create: Agregar un nuevo registro a la tabla "automovil".  
 fun insertAutomovil(automovil: Automovil): Long {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("modelo", automovil.modelo)  
 put("numero\_vin", automovil.numeroVIN)  
 put("numero\_chasis", automovil.numeroChasis)  
 put("numero\_motor", automovil.numeroMotor)  
 put("numero\_asientos", automovil.numeroAsientos)  
 put("anio", automovil.anio)  
 put("capacidad\_asientos", automovil.capacidadAsientos)  
 put("precio", automovil.precio)  
 put("uri\_img", automovil.uriImg)  
 put("descripcion", automovil.descripcion)  
 put("idmarcas", automovil.marca.id)  
 put("idtipoautomovil", automovil.tipoAutomovil.id)  
 put("idcolores", automovil.color.id)  
 **}** val id = db.insert("automovil", null, values)  
 db.close()  
 return id  
 }  
  
 // Método Read: Leer un registro específico de la tabla "automovil".  
 @SuppressLint("Range")  
 fun getAutomovil(): List<Automovil> {  
 val db = *readableDatabase* val columns = *arrayOf*(  
 "idautomovil",  
 "modelo",  
 "numero\_vin",  
 "numero\_chasis",  
 "numero\_motor",  
 "numero\_asientos",  
 "anio",  
 "capacidad\_asientos",  
 "precio",  
 "uri\_img",  
 "descripcion",  
 "idmarcas",  
 "idtipoautomovil",  
 "idcolores"  
 )  
 val cursor = db.query("automovil", columns, null, null, null, null, null)  
 val automoviles = *mutableListOf*<Automovil>()  
 while (cursor.moveToNext()) {  
 val id = cursor.getInt(cursor.getColumnIndex("idautomovil"))  
 val modelo = cursor.getString(cursor.getColumnIndex("modelo"))  
 val numeroVIN = cursor.getString(cursor.getColumnIndex("numero\_vin"))  
 val numeroChasis = cursor.getString(cursor.getColumnIndex("numero\_chasis"))  
 val numeroMotor = cursor.getString(cursor.getColumnIndex("numero\_motor"))  
 val numeroAsientos = cursor.getInt(cursor.getColumnIndex("numero\_asientos"))  
 val anio = cursor.getInt(cursor.getColumnIndex("anio"))  
 val capacidadAsientos = cursor.getInt(cursor.getColumnIndex("capacidad\_asientos"))  
 val precio = cursor.getDouble(cursor.getColumnIndex("precio"))  
 val uriImg = cursor.getString(cursor.getColumnIndex("uri\_img"))  
 val descripcion = cursor.getString(cursor.getColumnIndex("descripcion"))  
 val idMarca = cursor.getInt(cursor.getColumnIndex("idmarcas"))  
 val idTipoAutomovil = cursor.getInt(cursor.getColumnIndex("idtipoautomovil"))  
 val idColor = cursor.getInt(cursor.getColumnIndex("idcolores"))  
 val marca = getMarca(idMarca) ?: throw IllegalStateException("La marca no existe")  
 val tipoAutomovil = getTipoAutomovil(idTipoAutomovil)  
 ?: throw IllegalStateException("El tipo de automóvil no existe")  
 val color = getColor(idColor) ?: throw IllegalStateException("El color no existe")  
 val automovil = Automovil(  
 id,  
 modelo,  
 numeroVIN,  
 numeroChasis,  
 numeroMotor,  
 numeroAsientos,  
 anio,  
 capacidadAsientos,  
 precio,  
 uriImg,  
 descripcion,  
 marca,  
 tipoAutomovil,  
 color  
 )  
 automoviles.add(automovil)  
 }  
 cursor.close()  
 db.close()  
 return automoviles  
 }  
  
 // Método Update: Actualizar un registro existente en la tabla "automovil".  
 fun updateAutomovil(automovil: Automovil): Int {  
 val db = *writableDatabase* val values = ContentValues().*apply* **{** put("modelo", automovil.modelo)  
 put("numero\_vin", automovil.numeroVIN)  
 put("numero\_chasis", automovil.numeroChasis)  
 put("numero\_motor", automovil.numeroMotor)  
 put("numero\_asientos", automovil.numeroAsientos)  
 put("anio", automovil.anio)  
 put("capacidad\_asientos", automovil.capacidadAsientos)  
 put("precio", automovil.precio)  
 put("uri\_img", automovil.uriImg)  
 put("descripcion", automovil.descripcion)  
 put("idmarcas", automovil.marca.id)  
 put("idtipoautomovil", automovil.tipoAutomovil.id)  
 put("idcolores", automovil.color.id)  
 **}** val selection = "idautomovil = ?"  
 val selectionArgs = *arrayOf*(automovil.id.toString())  
 val rowsUpdated = db.update("automovil", values, selection, selectionArgs)  
 db.close()  
 return rowsUpdated  
 }  
  
 // Método Delete: Eliminar un registro existente en la tabla "automovil".  
 fun deleteAutomovil(id: Automovil): Int {  
 val db = *writableDatabase* val selection = "idautomovil = ?"  
 val selectionArgs = *arrayOf*(id.toString())  
 val rowsDeleted = db.delete("automovil", selection, selectionArgs)  
 db.close()  
 return rowsDeleted  
 }  
  
}