



Universidad Autónoma de Santo Domingo

Facultad de Ciencias

Estructura de Datos

100386331

Matrícula

Manuel García Aybar

Nombre

Mini-DBMS de Sistema de Transporte

Proyecto

23 de diciembre de 2017

Fecha

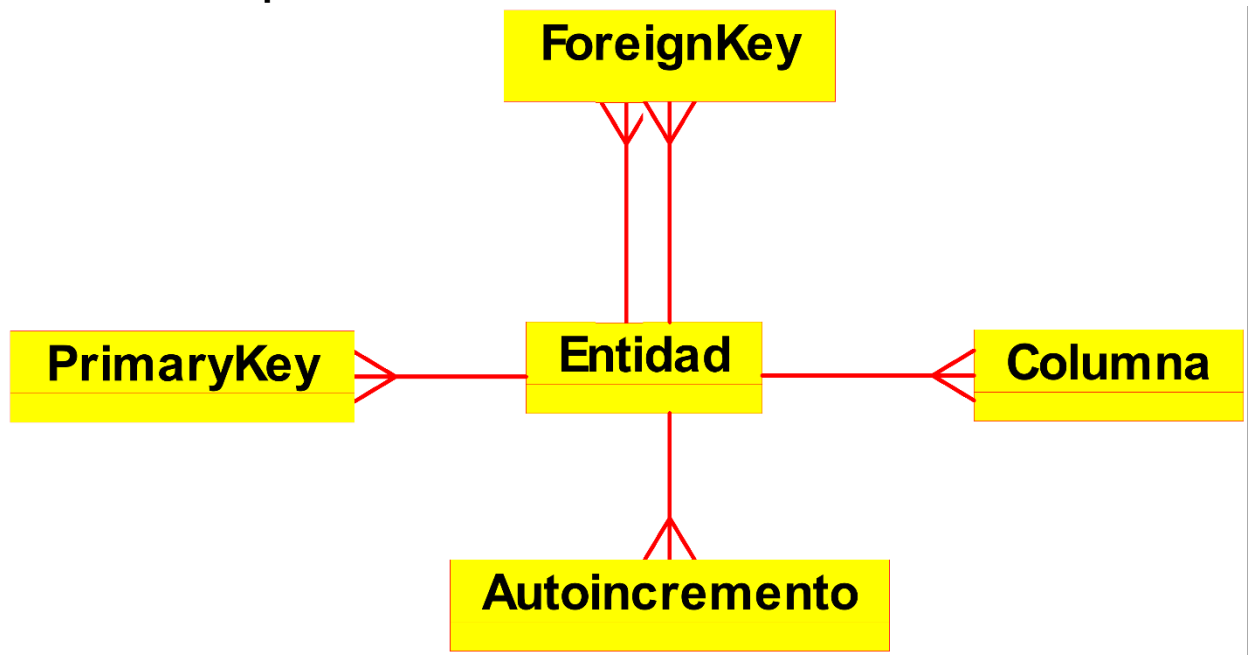
Contenido

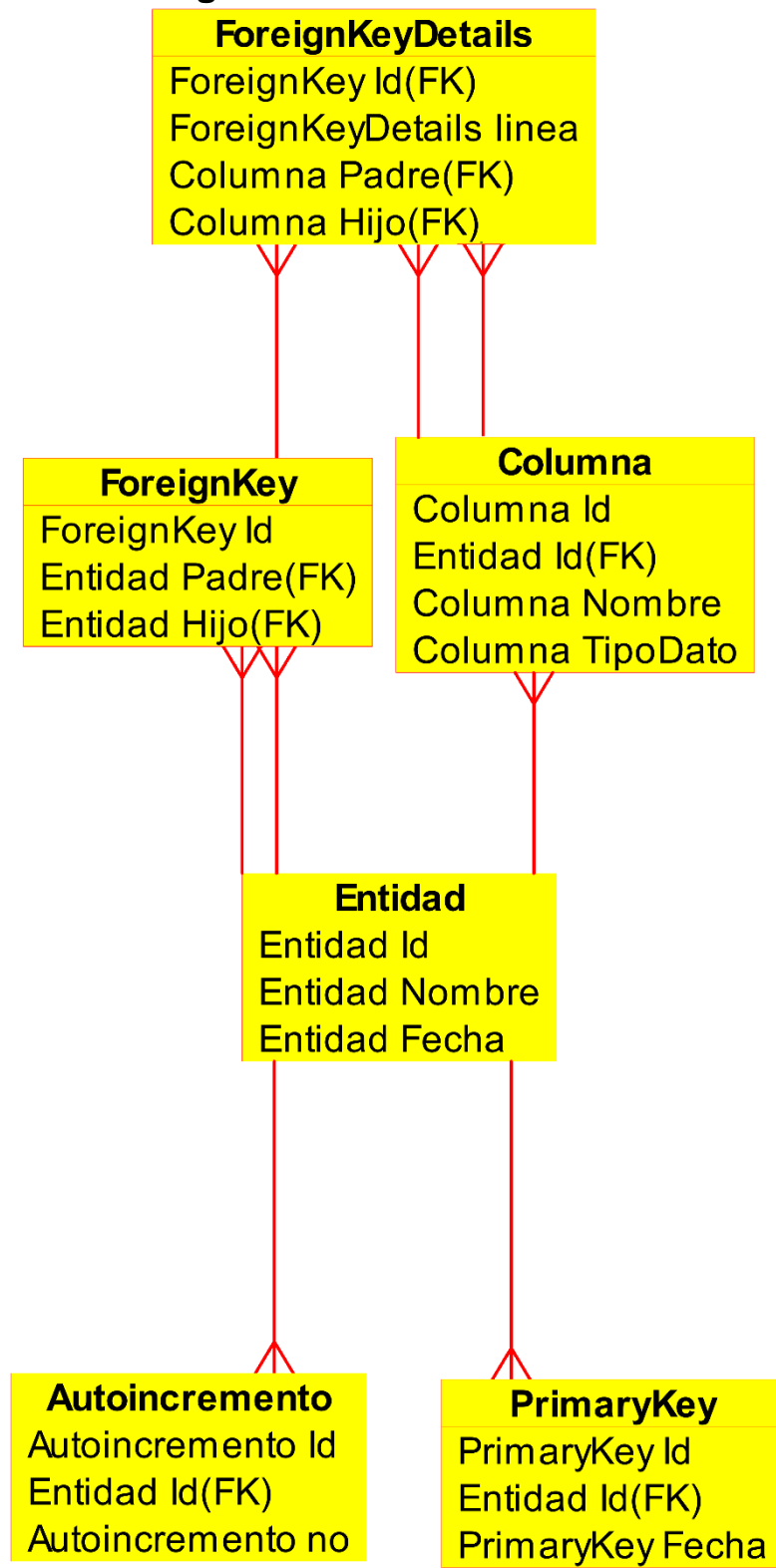
Diagrama E-R de Modelado de Datos	1
Modelo Conceptual	1
Modelo Lógico	2
Modelo Físico	3
Diagrama E-R de Transporte	4
Modelo Conceptual	4
Modelo Lógico	4
Modelo Físico	5
IOInterface.....	6
Header	6
PrimaryKey	7
Header	7
CPP	7
ForeignKey	9
Header	9
CPP	10
Datatype	12
Header	12
CPP	13
Columna	14
Header	14
CPP	15
Autoincremento.....	19
Header	19
CPP	19
Entidad	20
Header	20

CPP	21
Main	27
Screenshots	32

Diagrama E-R de Modelado de Datos

Modelo Conceptual



Modelo Lógico

Modelo Físico

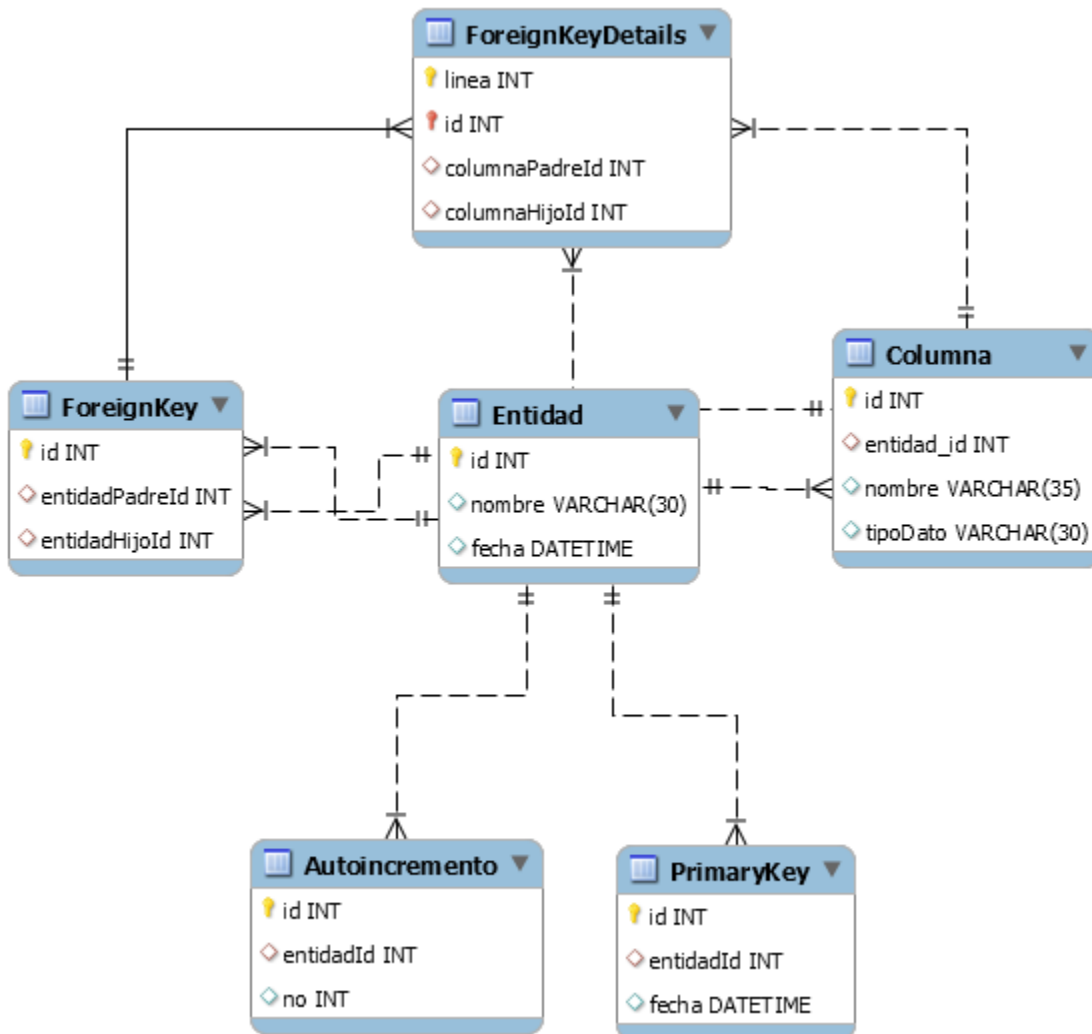
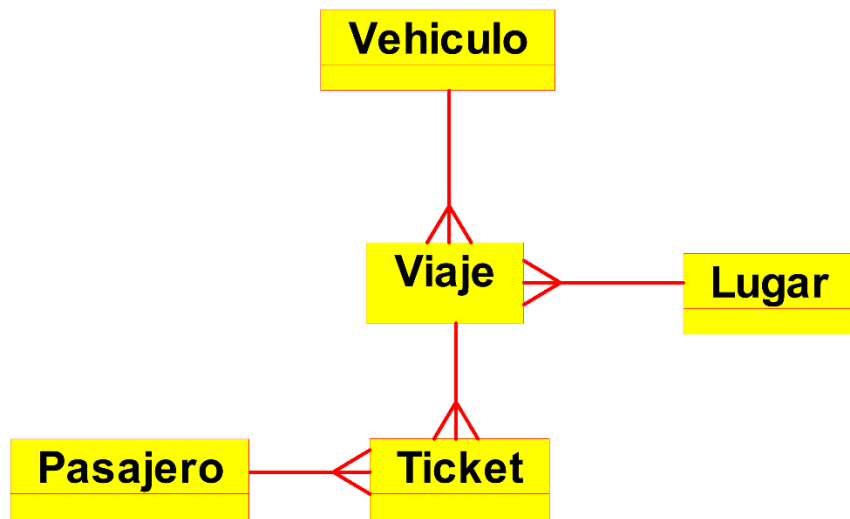
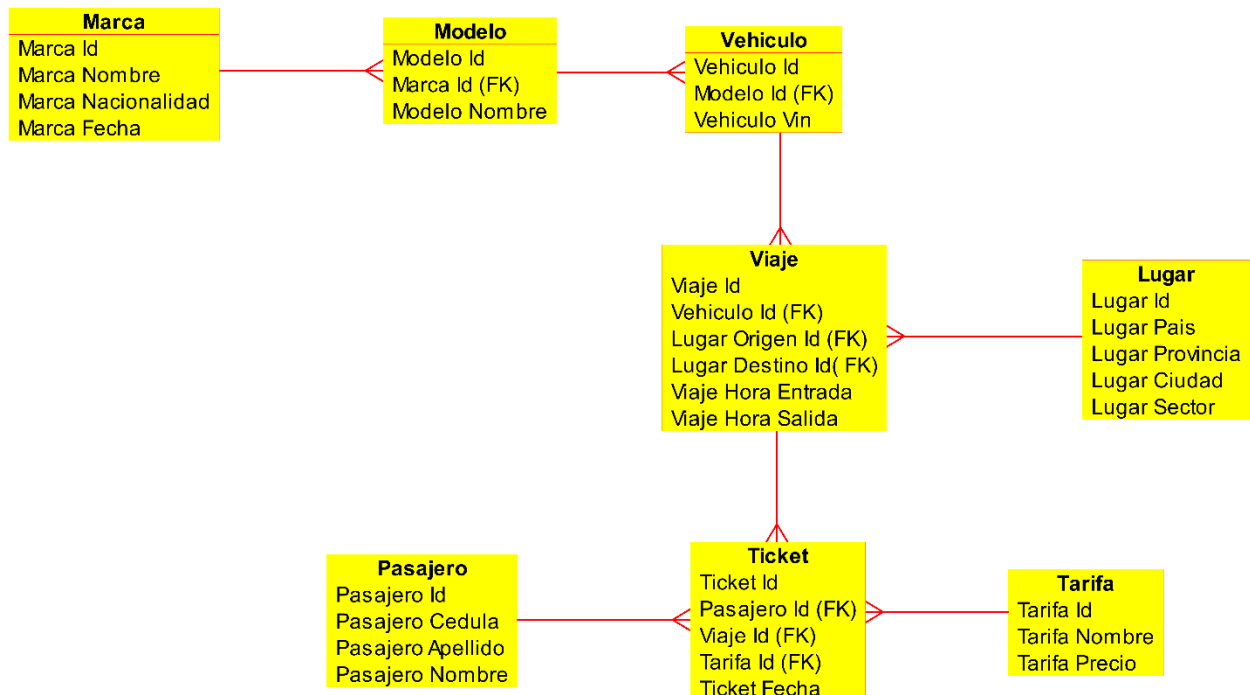


Diagrama E-R de Transporte

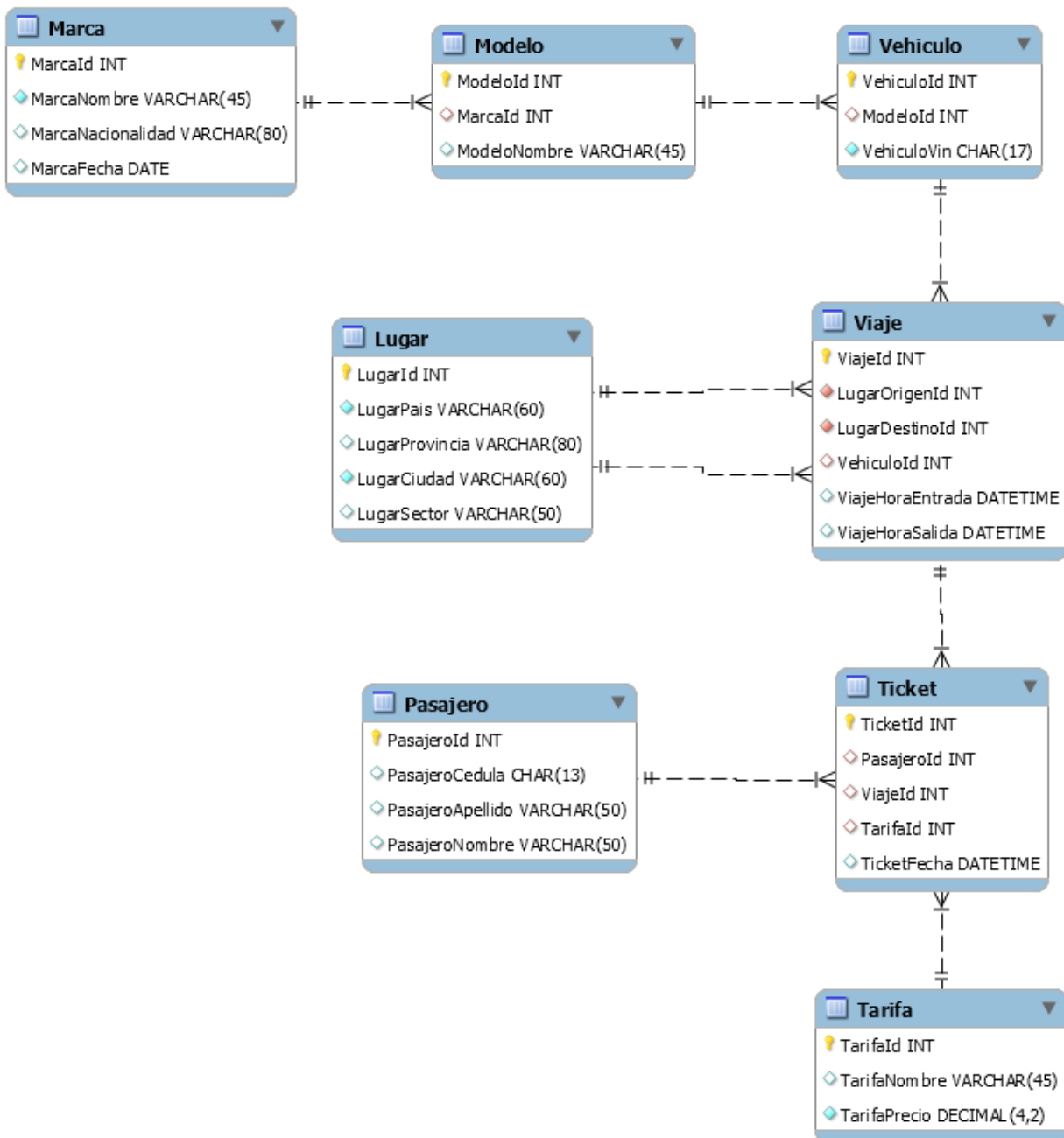
Modelo Conceptual



Modelo Lógico



Modelo Físico



IOInterface

Header

```
#pragma once
#include <string>
class IOInterface {
public:

    IOInterface() {
        filename_ = "Default.txt";
    }

    IOInterface(const std::string& filename) {
        filename_ = filename;
    }

    virtual ~IOInterface() {

    }

    std::string getFilename() const {
        return filename_;
    }
    virtual void read() = 0;
    virtual void write() = 0;
    virtual std::string toString() = 0;

private:
    std::string filename_;

};
```

PrimaryKey

Header

```
#pragma once
#include "IOInterface.h"
#include "Columna.h"
#include <string>
#include <set>
namespace dbms {
class PrimaryKey : IOInterface
{
private:
    int id_;
    static int contador_;
    int entidadId_;
    std::string fecha_;
public:
    PrimaryKey(){}
    PrimaryKey(int entidadId);
    virtual ~PrimaryKey();
    int getId();
    int getEntidadId();
    virtual void read() override;
    virtual void write() override;
    virtual std::string toString() override;
    static std::string mostrarCampos();
};
}
```

CPP

```
#include "stdafx.h"
#include "PrimaryKey.h"
#include <chrono>
#include <ctime>
namespace dbms {
    using std::chrono::system_clock;

    int PrimaryKey::contador_ = 0;

    PrimaryKey::PrimaryKey(int entidadId) : IOInterface("PrimaryKey.txt")
    {
        id_ = ++contador_;
        entidadId_ = entidadId;
        //-----
        time_t fechaActual = system_clock::to_time_t(system_clock::now());
        char temp[28];
        ctime_s(temp, 28, &fechaActual);
        fecha_ = temp;
        //-----
    }
}
```

```
    PrimaryKey::~~PrimaryKey()
    {
    }

    int PrimaryKey::getId()
    {
        return id_;
    }

    int PrimaryKey::getEntidadId()
    {
        return entidadId_;
    }

    void PrimaryKey::read()
    {
    }

    void PrimaryKey::write()
    {
    }

    std::string PrimaryKey::toString()
    {
        std::string mensaje = std::to_string(id_) + "\t\t"
            + std::to_string(entidadId_) + "\t\t"
            + fecha_ + "\n";
        return mensaje;
    }

    std::string PrimaryKey::mostrarCampos()
    {
        std::string mensaje = "id \t\t entidad_id \t\t fecha\n";
        return mensaje;
    }

}
```

ForeignKey

Header

```
#pragma once
#include "IOInterface.h"
#include <string>
#include <memory>
namespace dbms {

class ForeignKey : IOInterface
{
private:
    int id_;
    int entidadPadreId_;
    int entidadHijoId_;
    // Detalles
    class ForeignKeyDetails : IOInterface
    {
    private:
        int id_;
        int columnaPadreId_;
        int columnaHijoId_;
    public:
        ForeignKeyDetails(int id, int columnaPadreId, int columnaHijoId);
        virtual void read() override;
        virtual void write() override;
        virtual std::string toString() override;
        int getColumnaPadreId();
        int getColumnaHijoId();

    };
    std::unique_ptr<ForeignKeyDetails> detalles_;

public:
    ForeignKey(ForeignKey&& obj);
    ForeignKey(const ForeignKey& obj); // Copy constructor
    ForeignKey& operator=(const ForeignKey& obj);
    ForeignKey(int id, int entidadPadreId, int entidadHijoId, int columnaPadreId, int
columnaHijoId);
    virtual ~ForeignKey();
    virtual void read() override;
    virtual void write() override;
    virtual std::string toString() override;
    int getId();
    int getEntidadPadreId();
    int getEntidadHijoId();

};

}
```

CPP

```
#include "stdafx.h"
#include "ForeignKey.h"
#include <fstream>
namespace dbms {

    ForeignKey::ForeignKey(ForeignKey && obj)
    {
        id_ = std::move(obj.id_);
        entidadPadreId_ = std::move(obj.entidadPadreId_);
        entidadHijoId_ = std::move(obj.entidadHijoId_);
        detalles_ = std::move(obj.detalles_);
    }
    ForeignKey::ForeignKey(const ForeignKey & obj)
    {
        id_ = obj.id_;
        entidadPadreId_ = obj.entidadPadreId_;
        entidadHijoId_ = obj.entidadHijoId_;
        detalles_ = std::unique_ptr<ForeignKeyDetails>(&(*obj.detalles_));
    }
    ForeignKey & ForeignKey::operator=(const ForeignKey & obj)
    {
        id_ = obj.id_;
        entidadPadreId_ = obj.entidadPadreId_;
        entidadHijoId_ = obj.entidadHijoId_;
        detalles_ = std::unique_ptr<ForeignKeyDetails>(&(*obj.detalles_));
        return *this;
    }

    ForeignKey::ForeignKey(int id, int entidadPadreId, int entidadHijoId, int
columnaPadreId, int columnaHijoId)
        :IOInterface("ForeignKey.txt")
    {
        id_ = id;
        entidadPadreId_ = entidadPadreId;
        entidadHijoId_ = entidadHijoId;
        detalles_ = std::unique_ptr<ForeignKeyDetails>(new ForeignKeyDetails(id,
columnaPadreId, columnaHijoId));
    }

    ForeignKey::~~ForeignKey()
    {
    }

    void ForeignKey::read()
    {
    }

    void ForeignKey::write()
    {
    }
}
```

```
ForeignKey::ForeignKeyDetails::ForeignKeyDetails(int id, int columnaPadreId, int
columnaHijoId)
    : IOInterface("ForeignKeyDetails.txt")
{
    id_ = id;
    columnaPadreId_ = columnaPadreId;
    columnaHijoId_ = columnaHijoId;
}

void ForeignKey::ForeignKeyDetails::read()
{
}

void ForeignKey::ForeignKeyDetails::write()
{
}

std::string ForeignKey::ForeignKeyDetails::toString()
{
    std::string mensaje = std::to_string(id_) + "\t\t"
        + std::to_string(columnaPadreId_) + "\t\t"
        + std::to_string(columnaHijoId_) + "\n";
    return std::string();
}

int ForeignKey::getId()
{
    return id_;
}

int ForeignKey::getEntidadPadreId()
{
    return entidadPadreId_;
}

int ForeignKey::getEntidadHijoId()
{
    return entidadHijoId_;
}

std::string ForeignKey::toString()
{
    std::string mensaje = std::to_string(id_) + "\t\t"
        + std::to_string(entidadPadreId_) + "\t\t"
        + std::to_string(entidadHijoId_) + "\n";
    return mensaje;
}

int ForeignKey::ForeignKeyDetails::getColumnaPadreId()
{
    return columnaPadreId_;
}

int ForeignKey::ForeignKeyDetails::getColumnaHijoId()
{
    return columnaHijoId_;
}
```

```
}
```

Datatype

Header

```
#pragma once
#include <string>
#include <memory>
#include <map>
namespace dbms {
    enum class Type {
        SMALLINT = 1, INT, BIGINT, NUMERIC, CHAR, VARCHAR
    };

    class Datatype
    {
    private:
        std::string nombre_;
        short length_;
        short precision_;
        Type type_;

        Datatype(const Type& type, const std::string& nombre);

    public:
        Datatype();
        static const Datatype SMALLINT;
        static const Datatype BIGINT;
        static const Datatype INT;
        static const Datatype NUMERIC;
        static const Datatype VARCHAR;
        static const Datatype CHAR;

        static int getSize() { return 6; }

        std::string toString() const;
        Type getType() const;
        short getLength() const;
        short getPrecision() const;
        void setLength(short length);
        void setPrecision(short precision);

        friend class Columna;
        bool operator==(const Datatype& obj);
    };

    // Para transformar enum en su correspondiente tipo de dato.
    const auto tipos = std::unique_ptr<std::map<Type, Datatype>>(<
        new std::map<Type, Datatype>{
            { Type::SMALLINT, Datatype::SMALLINT }, { Type::INT, Datatype::INT
}, { Type::BIGINT, Datatype::BIGINT },
            { Type::NUMERIC, Datatype::NUMERIC }, { Type::CHAR, Datatype::CHAR
}, { Type::VARCHAR, Datatype::VARCHAR }
        }
    );}
```

C++

```
#include "Datatype.h"
namespace dbms {

    // Definicion de constantes

    const Datatype Datatype::SMALLINT = Datatype(Type::SMALLINT, "SMALLINT" );
    const Datatype Datatype::INT = Datatype(Type::INT, "INT");
    const Datatype Datatype::BIGINT = Datatype(Type::BIGINT, "BIGINT");
    const Datatype Datatype::NUMERIC = Datatype(Type::NUMERIC, "NUMERIC");
    const Datatype Datatype::CHAR = Datatype(Type::CHAR, "CHAR");
    const Datatype Datatype::VARCHAR = Datatype(Type::VARCHAR, "VARCHAR");

    Datatype::Datatype(const Type& type, const std::string& nombre) : Datatype()
    {
        type_ = type;
        nombre_ = nombre;
    }

    Datatype::Datatype() {
        length_ = 1;
        precision_ = 0;
    }

    std::string Datatype::toString() const
    {
        return nombre_;
    }
    Type Datatype::getType() const
    {
        return type_;
    }
    short Datatype::getLength() const
    {
        return length_;
    }
    short Datatype::getPrecision() const
    {
        return precision_;
    }
    void Datatype::setLength(short length)
    {
        length_ = length;
    }
    void Datatype::setPrecision(short precision)
    {
        precision_ = precision;
    }

    bool Datatype::operator==(const Datatype& obj)
    {
        return obj.type_ == obj.type_;
    }
}
```


Columna

Header

```
#pragma once
#include "IOInterface.h"
#include "Datatype.h"
#include <string>
#include <map>
#include <list>
#include <memory>
namespace dbms {

    class Columna : IOInterface
    {
    private:
        int id_;
        int entidadId_; // FK
        std::string nombre_;
        Datatype tipoDato_;

        Columna(const int entidadId, const int id, const std::string& nombre, const
Datatype& tipoDato);

    public:
        Columna(){}
        Columna(const Columna& columna);
        Columna(const std::string& entidadNombre, const int id, const std::string&
nombre, const Datatype& tipoDato);
        Columna(const std::string& entidadNombre, const int entidadId, const int
id,
                const std::string& nombre, const Datatype& tipoDato);
        virtual ~Columna();
        int getId();
        int getEntidadId(); // FK
        std::string getNombre();
        Datatype getTipoDato();
        virtual void read() override;
        virtual void write() override;
        std::list<Columna> readAll();
        std::string toString() override;
        std::string mostrarCampos();
    };
}
```

CPP

```

#include "stdafx.h"
#include "Columna.h"
#include <fstream>
#include <algorithm>
namespace dbms {

    using namespace std;

    // Copy constructor
    Columna::Columna(const Columna & columna)
    {
        id_ = columna.id_;
        entidadId_ = columna.entidadId_;
        nombre_ = columna.nombre_;
        tipoDato_ = columna.tipoDato_;
    }

    Columna::Columna(const int entidadId, const int id, const std::string & nombre,
const Datatype & tipoDato)
    {
        id_ = id;
        entidadId_ = entidadId;
        nombre_ = nombre;
        tipoDato_ = tipoDato;
    }

    Columna::Columna(const std::string& entidadNombre, const int id, const
std::string& nombre, const Datatype& tipoDato)
        : IOInterface(entidadNombre + "/Columna.txt")
    {
        id_ = id;
        nombre_ = nombre;
        tipoDato_ = tipoDato;
        entidadId_ = 0;
    }

    Columna::Columna(const std::string& entidadNombre, const int entidadId, const int
id,
        const std::string& nombre, const Datatype& tipoDato) :
IOInterface(entidadNombre + "/Columna.txt")
    {
        id_ = id;
        entidadId_ = entidadId;
        nombre_ = nombre;
        tipoDato_ = tipoDato;
    }

    Columna::~~Columna()
    {
    }
}

```

```
int Columna::getId()
{
    return id_;
}

int Columna::getEntidadId()
{
    return entidadId_;
}

std::string Columna::getNombre()
{
    return nombre_;
}

Datatype Columna::getTipoDato()
{
    return tipoDato_;
}

std::string Columna::toString()
{
    std::string tipoDato = tipoDato_.toString();

    // IF ES NUMERIC O CHAR
    switch (tipoDato_.getType()) {
    case Type::CHAR:
    case Type::VARCHAR:
    case Type::NUMERIC:
        tipoDato.append("(" + std::to_string(tipoDato_.getLength()) );

        // Si es NUMERIC ponle precision.
        tipoDato.append( (tipoDato_.getType() == Type::NUMERIC)? ", "
            + std::to_string(tipoDato_.getPrecision()) : "");

        tipoDato.append(")");
        break;
    }

    std::string registro = std::to_string(entidadId_) + "\t|\t"
        + std::to_string(id_) + "\t|\t"
        + nombre_ + "\t|\t"
        + tipoDato + '\n';

    return registro;
}

void Columna::read()
{
    // En proceso
}
```

```
void Columna::write()
{
    std::ofstream archivo;
    archivo.open(getFilename(), std::ios::out | std::ios::app);

    if (!archivo.fail()) {
        archivo << toString();
    }
    else {
        throw std::ios_base::failure("El archivo " + getFilename() + " no
pudo ser escrito correctamente.\n");
    }

    archivo.close();
}

std::list<Columna> Columna::readAll()
{
    std::ifstream archivo;
    archivo.open(getFilename());
    list<Columna> columnas;

    // Debo ignorar los campos;
    if (!archivo.fail()) {
        char pipe;
        bool sonCampos = true;

        // Columna Metadata;
        int entidadId;
        int id;
        std::string nombre;
        std::string tipoDato;
        Datatype datatype;

        while (!archivo.eof()) {

            if (sonCampos) {
                for (int i = 0; i < 3; ++i)
                {
                    archivo >> tipoDato;
                    archivo >> pipe;
                }
                archivo >> tipoDato;
                sonCampos = false;
                continue;
            }
            archivo >> entidadId;
            archivo >> pipe;
            archivo >> id;
            archivo >> pipe;
            archivo >> nombre;
            archivo >> pipe;

            archivo >> tipoDato;

            std::transform(tipoDato.begin(), tipoDato.end(),
tipoDato.begin(), ::toupper);
```

```

        // INTENTAR AVERIGUAR SI TIENE PARENTESIS;
        size_t open = tipoDato.find_first_of('(');
        bool isFound = open >= 0;
        if (!isFound)
        {
            datatype.nombre_ = tipoDato;
        }
        else
        {
            datatype.nombre_ = tipoDato.substr(0, open - 1);

            bool esNumeric = (datatype.toString() ==
Datatype::NUMERIC.toString());

            if (esNumeric)
            {
                // Busca longitud del NUMERIC.
                datatype.type_ = Type::NUMERIC;
                size_t coma =
datatype.nombre_.find_first_of(',', open);
                datatype.length_ =
stoi(datatype.nombre_.substr(open + 1, (coma - open) - 1));

                // Busca precision del NUMERIC.
                size_t close =
datatype.toString().find_first_of(')', coma);
                datatype.precision_ =
stoi(datatype.nombre_.substr(coma + 1, (close - coma) - 1));
            }
            else {
                bool esVarchar = datatype.toString() ==
Datatype::VARCHAR.toString();

                datatype.type_ = (esVarchar) ? Type::VARCHAR :
Type::CHAR;

                size_t close =
datatype.toString().find_first_of(')', open);
                datatype.length_ =
stoi(datatype.nombre_.substr(open + 1, (close - open) - 1));
            }

            Columna columna = Columna(id, entidadId, nombre, datatype);
            columnas.push_back(columna);
        }
    }
    else {
        string mensaje = "El archivo " + getFilename() + "no pudo ser leído
correctamente.\n";
        throw std::ios_base::failure(mensaje);
    }

    return columnas;
}

```

```
        std::string Columna::mostrarCampos()
        {
            std::string mensaje = "entidad_id \t|\t colId \t|\t colNombre \t|\t
colTipo\r\n";
            return mensaje;
        }
    }
```

Autoincremento

Header

```
#pragma once
namespace dbms {

    class Autoincremento
    {
        int incremento_;
        int entidadId_;
    public:
        Autoincremento(int entidadId);
        virtual ~Autoincremento();
        int getIncremento();
        void setIncremento(int incremento);
    };

}
```

CPP

```
#include "stdafx.h"
#include "Autoincremento.h"

namespace dbms {

    Autoincremento::Autoincremento(int entidadId)
    {
        entidadId_ = entidadId;
    }

    Autoincremento::~~Autoincremento()
    {
    }

    int Autoincremento::getIncremento()
    {
        return incremento_;
    }

    void Autoincremento::setIncremento(int incremento)
    {
        incremento_ = incremento;
    }

}
```

Entidad

Header

```
#pragma once
#include "IOInterface.h"
#include "Columna.h"
#include "PrimaryKey.h"
#include "ForeignKey.h"
#include <string>
#include <vector>
#include <map>
#include <memory>
namespace dbms {
class Entidad : IOInterface
{
private:
    int contadorColumna_;
    int id_;
    int contadorRegistro_;
    std::string nombre_;
    std::string fecha_;
    PrimaryKey primaryKey_;
    std::vector<Columna> columnas_;
    std::vector<ForeignKey> foreignKeys_;
    std::map<int, std::vector<std::string>> registros_; // Registros
    std::string mensaje_;
    bool esPrimeraColumna = true;

    // Persistencia automatizada.
    virtual void read() override;

    bool isValidData(std::vector<std::string> data);

    virtual void write() override; // Agregar entidades

    void writeColumnsFields(Columna& columna);

public:
    Entidad(const Entidad& obj);
    Entidad(const int id, const std::string& nombre);
    Entidad& operator=(const Entidad& obj);
    virtual ~Entidad();

    void agregarColumna(std::string nombre, Datatype tipo);
    std::vector<Columna> getColumnas();
    bool buscar(int pk);
    bool buscar(std::string texto);
    void alta(std::vector<std::string> data); // Enviar a Tabla.txt
    void baja(int id);
    void sort(std::vector<std::string> data);
    virtual std::vector<Entidad> readAll();
    virtual std::string toString() override;

    int getId();
```

```

        std::string getNombre();
        std::string getFecha();
        std::map<int, std::vector<std::string>> getData();

};

}

```

C++

```

#include "stdafx.h"
#include "Entidad.h"
#include <chrono>
#include <ctime>
#include <fstream>
#include <filesystem> // C++ 17+
namespace dbms {

    using std::string;
    using std::vector;
    using std::chrono::system_clock;

    // Copy constructor
    Entidad::Entidad(const Entidad & obj) : IOInterface("Systable.txt")
    {
        contadorColumna_ = obj.contadorColumna_;
        id_ = obj.id_;
        contadorRegistro_ = obj.contadorRegistro_;
        nombre_ = obj.nombre_;
        fecha_ = obj.fecha_;
        primaryKey_ = obj.primaryKey_;
        columnas_ = obj.columnas_;
        foreignKeys_ = obj.foreignKeys_;
        registros_ = obj.registros_; // Registros
        mensaje_ = obj.mensaje_;
        esPrimeraColumna = obj.esPrimeraColumna;
    }

    Entidad::Entidad(const int id, const std::string& nombre) :
    IOInterface("Systable.txt")
    {
        contadorRegistro_ = 0;
        contadorColumna_ = 0;
        id_ = id;
        primaryKey_ = PrimaryKey(id);
        nombre_ = nombre;
        esPrimeraColumna = true;
        columnas_ = vector<Columna>();
        foreignKeys_ = vector<ForeignKey>();
        registros_ = std::map<int, std::vector<std::string>>();
        mensaje_ = "";
        //-----
        time_t fechaActual = system_clock::to_time_t(system_clock::now());
        char temp[27];
        ctime_s(temp, sizeof(temp), &fechaActual);
        fecha_ = temp;
    }
}

```



```
        //-----
    }

    // Copy operator.
    Entidad & Entidad::operator=(const Entidad & obj)
    {
        contadorColumna_ = obj.contadorColumna_;
        id_ = obj.id_;
        contadorRegistro_ = obj.contadorRegistro_;
        nombre_ = obj.nombre_;
        fecha_ = obj.fecha_;
        primaryKey_ = obj.primaryKey_;
        columnas_ = obj.columnas_;
        foreignKeys_ = obj.foreignKeys_;
        registros_ = obj.registros_; // Registros
        mensaje_ = obj.mensaje_;
        esPrimeraColumna = obj.esPrimeraColumna;

        return *this;
    }

    Entidad::~Entidad()
    {
    }

    void Entidad::agregarColumna(std::string nombre, Datatype tipo)
    {
        Columna columna = Columna(nombre_, (contadorColumna_ + 1), nombre, tipo);
        if (esPrimeraColumna)
        {
            // Write metadata a systable.txt write();
            write(); // Registra la entidad en el catalogo del sistema.
            writeColumnsFields(columna);
            esPrimeraColumna = false;
        }
        columna.write();
        columnas_.push_back(columna);
        ++contadorColumna_;
    }

    std::vector<Columna> Entidad::getColumnas()
    {
        Columna col = Columna(nombre_, 0, nombre_, Datatype::CHAR);

        auto columnasEncontradas = col.readAll();
        std::vector<Columna> columnas{
            std::make_move_iterator(std::begin(columnasEncontradas)),
            std::make_move_iterator(std::end(columnasEncontradas)) };
        columnas_ = columnas;
        return columnas_;
    }

    bool Entidad::buscar(int pk)
    {
        return false;
    }
```

```
bool Entidad::buscar(std::string texto)
{
    return false;
}

bool Entidad::isDataValid(std::vector<std::string> data)
{
    bool isValid = columnas_.size() == data.size();
    // isValid &= !buscar(data[primaryKey_.getId()]); // En proceso
    for (auto columna : columnas_)
    {
        string valor = data[columna.getId() - 1];
        switch (columna.getTipoDato().getType()) {
        case Type::SMALLINT:
        {
            short entero = std::stoi(valor);
        }
        break;
        case Type::INT:
        {
            int entero = std::stoi(valor);
        }
        break;
        case Type::BIGINT:
        {
            long entero = std::stol(valor);
        }
        break;
        case Type::NUMERIC:
        {
            double numeric = std::stod(valor);
            break;
        }
        }
    }

    return isValid;
}

// Beta.
void Entidad::writeColumnsFields(Columna& columna)
{
    std::ofstream archivo;
    archivo.open(nombre_ + "/Columna.txt", std::ios::out);

    if (!archivo.fail()) {
        archivo << columna.mostrarCampos();
    }
    else {
        throw std::ios_base::failure("Los campos no pudieron ser escritas
correctamente.\n");
    }

    archivo.close();
}
```

```
void Entidad::alta(std::vector<std::string> data)
{
    if (isDataValid(data)) {
        registros_[++contadorRegistro_] = data; // Guarda en el map
        std::ofstream archivo;
        archivo.open(nombre_ + "/Data.txt", std::ios::out | std::ios::app);

        if (!archivo.fail()) {
            for (int i = 0; i < data.size() - 1; ++i) {
                archivo << data[i] << "\t\t";
            }
            archivo << data[data.size() - 1] << "\n";
        }
        archivo.close();
    }
}

void Entidad::baja(int id)
{
    //registros_.erase(registros_.begin() + id);
    --contadorRegistro_;
}

void Entidad::sort(std::vector<string> data)
{
}

std::string Entidad::toString()
{
    return "En proceso";
}

void Entidad::read()
{
}

// Cada vez que agregue una entidad, crear carpeta
void Entidad::write()
{
    std::ofstream archivo;
    archivo.open(getFilename(), std::ios::out | std::ios::app);

    if (!std::experimental::filesystem::exists(nombre_)) {
        std::experimental::filesystem::create_directory(nombre_); // Crea
una carpeta con el nombre de la entidad
    }
    if (!archivo.fail()) {
        archivo << id_ << "\t\t" << nombre_ << "\t\t" << fecha_;
    }
    else {
        throw std::ios_base::failure("El archivo " + getFilename() + " no
pudo ser escrito correctamente.\n");
    }

    archivo.close();
}
```

```
// Dame todas las tablas disponibles
std::vector<Entidad> Entidad::readAll()
{
    std::ifstream archivo;
    archivo.open(getFilename());

    // Debo ignorar los campos;
    std::vector<Entidad> vectorEntidades; // = std::vector<Entidad>();
    if (!archivo.fail()) {

        std::list<Entidad> entidades;

        bool sonCampos = true;
        while (!archivo.eof()) {
            char pipe;
            if (sonCampos) {
                std::string campos = "";
                std::getline(archivo, campos, '\n');
                sonCampos = false;
            }
            else {
                // Entidad Metadata;
                int id = 0;
                std::string nombre = "";
                std::string fecha = "";

                if (archivo >> id) {
                    archivo >> pipe;
                    archivo >> nombre;
                    archivo >> pipe;
                    archivo.ignore();
                    std::getline(archivo, fecha);

                    Entidad entidad = Entidad(id, nombre);
                    entidad.fecha_ = fecha;
                    entidades.push_back(entidad);
                }
            }
        }

        std::vector<Entidad> tempVectorEntidades{
            std::make_move_iterator(std::begin(entidades)),
            std::make_move_iterator(std::end(entidades))
        };

        tempVectorEntidades.shrink_to_fit();
        vectorEntidades = tempVectorEntidades;
    }
    else {
        string mensaje = "El archivo " + getFilename() + "no pudo ser leído
correctamente.\n";
        throw std::ios_base::failure(mensaje);
    }

    archivo.close();
}
```

```
        return vectorEntidades;
    }

    int Entidad::getId()
    {
        return id_;
    }

    std::string Entidad::getNombre()
    {
        return nombre_;
    }

    std::string Entidad::getFecha()
    {
        return fecha_;
    }

    // En proceso.
    std::map<int, std::vector<string> > Entidad::getData()
    {
        std::ifstream archivo;
        archivo.open(nombre_ + "/Data.txt");
        if (!archivo.fail()) {

        }
        return registros_;
    }
}
```

Main

```
#include "stdafx.h"
#include <iostream>
#include <limits> // Valida entrada por teclado.
#include <cctype> // toupper();
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include "Entidad.h"

using dbms::Entidad;
using std::cout;
using std::cin;
using std::endl;
using std::string;
using dbms::Type;
using dbms::Datatype;

const std::string SYSTABLE = "Systable.txt";

void clear_screen()
{
#ifdef _WIN64 || _WIN32
    std::system("cls");
#else
    std::system("clear");
#endif
}

enum class MenuTabla {
    EXIT, CREATE, MOSTRAR, DROP, INSERT, SELECT
};
MenuTabla mostrarMenu();

template <class T>
void validarNumero(T& numero, string mensaje);

const std::string CATALOGO_CAMPOS = "entidad_id \t|\t entidad_nombre \t|\t
entidad_fecha_creada \r\n";
void crearCatalogo();
void mostrarEntidades(std::vector<Entidad> entidades);

void crearEntidad(int& contador);
void verData(std::vector<Entidad> entidades);
void buscarData(std::vector<Entidad> entidades);
void alta(std::vector<Entidad> entidades);

int main()
{
    crearCatalogo();
    std::vector<Entidad> entidades = std::vector<Entidad>();

    bool haTerminado = false;
    while (!haTerminado) {
        try {

            entidades = Entidad(1, "a").readAll();
```

```

        int lastId = entidades.size();
        MenuTabla opcion = mostrarMenu();
        switch (opcion) {
            case MenuTabla::CREATE:
                crearEntidad(lastId);
                break;
            case MenuTabla::MOSTRAR:
                mostrarEntidades(entidades);
                break;
            case MenuTabla::INSERT:
                alta(entidades);
                break;
            case MenuTabla::DROP:
                // Inserte funcion aqui.
                break;
            case MenuTabla::SELECT:
                verData(entidades);
                break;
            default:
                haTerminado = true;
        }
    }
    catch (std::exception& e) {
        cout << e.what();
    }
    //-----
    cout << "\n\n";
    cout << "Presione ENTER para continuar...";
    cin.ignore();
    cin.get();
    cout << "\n\n";
    clear_screen();
}

}

MenuTabla mostrarMenu() {
    unsigned short opcion;
    const unsigned short ULTIMA = 3;
    bool esValida = false;
    string menu = "\t\tMenu de entidad: \n";
    menu.append("1. Crear una entidad.\n");
    menu.append("2. Mostrar entidades.\n");
    menu.append("3. Eliminar una entidad.\n");
    menu.append("4. Agregar data a una entidad.\n");
    menu.append("5. Ver data a una entidad.\n");
    menu.append("0. Salir.\n\n");

    while (!esValida) {
        validarNumero(opcion, menu);
        esValida = (opcion == static_cast<unsigned short>(MenuTabla::EXIT)) ||
(opcion <= ULTIMA);
    }

    return static_cast<MenuTabla>(opcion);
}

```

```

template <class T>
void validarNumero(T& numero, string mensaje) {
    bool esValido = false;
    while (!esValido) {
        cout << mensaje;
        if (cin >> numero) {
            esValido = true;
        }
        else {
            cin.clear();
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        cout << "\n\n";
    }
}

void crearCatalogo() {
    if (!std::experimental::filesystem::exists(SYSTABLE)) {
        std::ofstream archivo;
        archivo.open(SYSTABLE, std::ios::out);
        if (!archivo.fail()) {
            archivo << CATALOGO_CAMPOS;
        }

        archivo.close();
    }
}

void mostrarEntidades(std::vector<Entidad> entidades)
{
    cout << CATALOGO_CAMPOS;
    for (auto entidad : entidades)
    {
        cout << entidad.getId() << "\t\t" << entidad.getNombre() << "\t\t" <<
entidad.getFecha() << "\n";
    }
    cout << "\n";
}

void crearEntidad(int& contador) {
    string nombre;
    int tipoDato;
    cout << "Nombre de la entidad: ";
    cin >> nombre;

    auto entidad = std::unique_ptr<Entidad>(new Entidad(contador + 1, nombre));

    bool haTerminado = false;
    while (!haTerminado)
    {
        string columnaNombre, mensaje;
        cout << "Nombre de la columna: ";
        cin >> columnaNombre;

        mensaje.append("\nSeleccione tipo de dato\n");
        Datatype datatype;
    }
}

```



```

        for (int i = static_cast<int>(dbms::Type::SMALLINT); i <=
Datatype::getSize(); ++i) {
            datatype = dbms::tipos->operator[](static_cast<Type>(i));
            mensaje.append(std::to_string(i) + ". " + datatype.toString() +
"\n");
        }
        validarNumero(tipoDato, mensaje);

        datatype = dbms::tipos->operator[](static_cast<Type>(tipoDato));
        switch (datatype.getType())
        {
            case Type::CHAR:
            case Type::VARCHAR:
            case Type::NUMERIC:
            {
                std::string mensaje = "Ingrese longitud del " + datatype.toString()
+ ": ";

                short length = 0;
                validarNumero(length, mensaje);
                datatype.setLength(length);

                if (datatype.getType() == Type::NUMERIC) {
                    short precision = 0;
                    mensaje.clear();
                    mensaje.append("Ingrese precision del NUMERIC: ");
                    validarNumero(precision, mensaje);
                    datatype.setPrecision(precision);
                }
            }
            default:
                break;
        }

        entidad->agregarColumna(columnaNombre, datatype);

        char opcion;
        cout << "Desea agregar otra columna? [S/N]\n";
        cin >> opcion;
        opcion = std::toupper(opcion);
        haTerminado = (opcion == 'N');
    }
}

void verData(std::vector<Entidad> entidades) {
    mostrarEntidades(entidades);

    int id;
    cout << "Ingrese id de la entidad a consultar: ";
    cin >> id;

    Entidad entidad = entidades[id - 1];

    auto registros = entidad.getData();

    cout << "Numero de registros: " << registros.size() << "\n\n";
    for (int i = 1; i < registros.size(); i++) {
        cout << registros.find(i)->first;
    }
}

```

```

    }
}

void buscarData(std::vector<Entidad> entidades) {
    mostrarEntidades(entidades);

    int id;
    cout << "Ingrese id de la entidad a consultar: ";
    cin >> id;

    Entidad entidad = entidades[id - 1];

    cout << "Ingrese id del registro a buscar: ";
    cin >> id;

    auto map = entidad.getData();
    auto registro = map.find(id);

    if (registro != map.end()) {
        cout << registro->first << "\n";
    }
    else
    {
        cout << "El registro no pudo ser encontrado.\n\n";
    }
}

void alta(std::vector<Entidad> entidades) {
    mostrarEntidades(entidades);
    int id;
    cout << "Ingrese id de la entidad: ";
    cin >> id;

    Entidad entidad = entidades[id - 1];

    bool haTerminado = false;
    while (!haTerminado)
    {
        std::vector<string> data;
        for (auto columna : entidad.getColumns())
        {
            string dato;
            cout << columna.getNombre() << ": ";
            cin >> dato;
            data.push_back(dato);
        }

        entidad.alta(data);

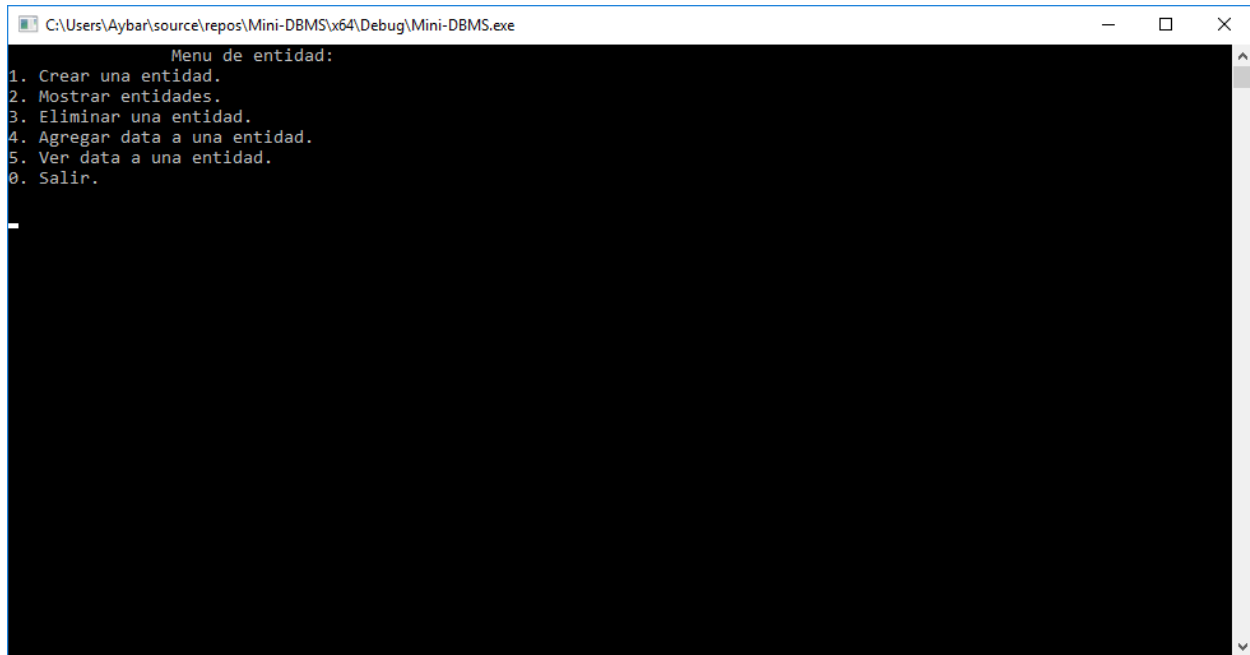
        cout << "\n\n";
        char opcion;
        cout << "Desea insertar otro registro? [S/N] ";
        cin >> opcion;
        opcion = toupper(opcion);

        haTerminado = (opcion == 'S');
    }
}

```

```
}  
}
```

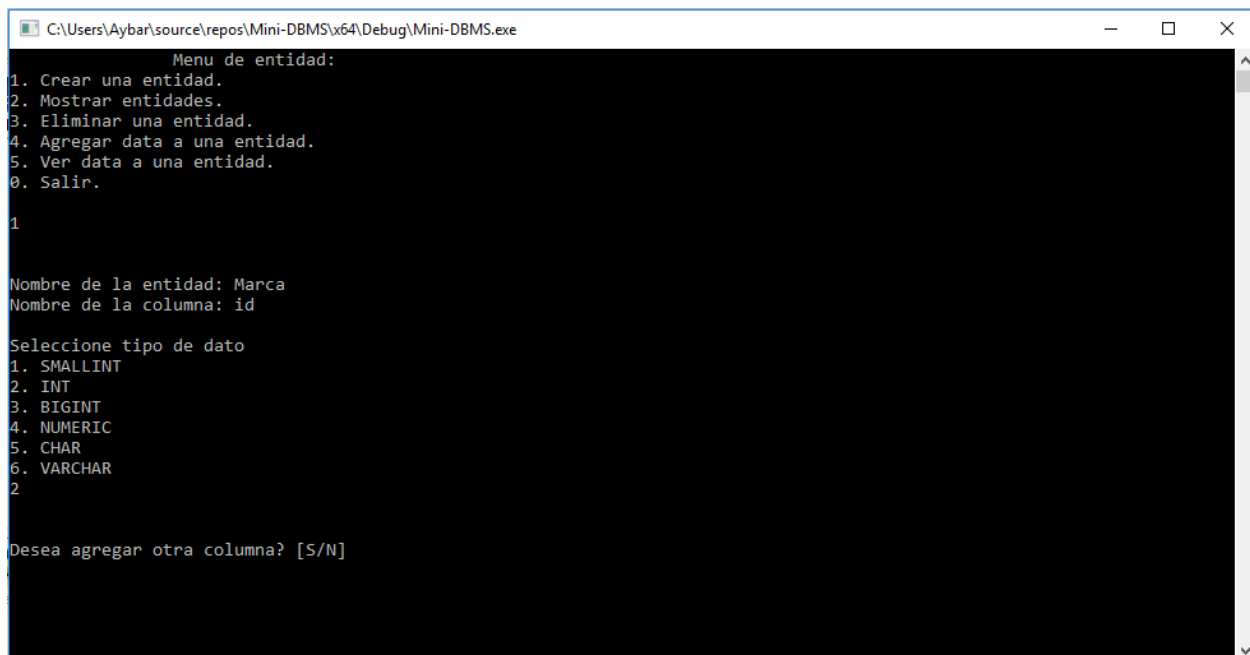
Screenshots



A screenshot of a Windows command prompt window titled "C:\Users\Aybar\source\repos\Mini-DBMS\x64\Debug\Mini-DBMS.exe". The window displays a menu titled "Menu de entidad:" with the following options:

- 1. Crear una entidad.
- 2. Mostrar entidades.
- 3. Eliminar una entidad.
- 4. Agregar data a una entidad.
- 5. Ver data a una entidad.
- 0. Salir.

The cursor is positioned at the end of the first option.



A screenshot of the same Mini-DBMS application window, now showing the data entry process. The menu is still visible at the top. Below it, the user has entered '1' to select the first option. The application prompts for the entity name and column name:

Nombre de la entidad: Marca
Nombre de la columna: id

Next, it prompts for the data type:

Seleccione tipo de dato

- 1. SMALLINT
- 2. INT
- 3. BIGINT
- 4. NUMERIC
- 5. CHAR
- 6. VARCHAR

The user has entered '2' to select INT. Finally, it asks if the user wants to add another column:

Desea agregar otra columna? [S/N]

Columna.txt - Notepad

File Edit Format View Help

entidad_id	colId	colNombre	colTipo
0	1	id	INT
0	2	nombre	VARCHAR(45)
0	3	nacionalidad	VARCHAR(60)

Name	Date modified	Type	Size
.vs	12/14/2017 11:05 ...	File folder	
Debug	12/18/2017 1:11 PM	File folder	
Marca	12/22/2017 6:21 PM	File folder	
src	12/21/2017 1:51 PM	File folder	
x64	12/14/2017 9:25 PM	File folder	
LICENSE	12/14/2017 11:04 ...	File	2 KB
README.md	12/14/2017 11:04 ...	Markdown Source...	1 KB
Mini-DBMS.sln	12/14/2017 5:01 PM	Microsoft Visual S...	2 KB
Mini-DBMS.vcxproj.user	12/16/2017 10:47 ...	Per-User Project O...	1 KB
.gitattributes	12/14/2017 11:07 ...	Text Document	3 KB
.gitignore	12/14/2017 11:04 ...	Text Document	5 KB
Systable.txt	12/22/2017 6:21 PM	Text Document	1 KB
Mini-DBMS.vcxproj	12/21/2017 5:14 PM	VC++ Project	9 KB
Mini-DBMS.vcxproj.filters	12/21/2017 2:31 PM	VC++ Project Filte...	3 KB

metadatos de la entidad

4 SYS TABLES

```
C:\Users\Aybar\source\repos\Mini-DBMS\x64\Debug\Mini-DBMS.exe
Menu de entidad:
1. Crear una entidad.
2. Mostrar entidades.
3. Eliminar una entidad.
4. Agregar data a una entidad.
5. Ver data a una entidad.
0. Salir.
2

entidad_id | entidad_nombre | entidad_fecha_creada
1 | Marca | Fri Dec 22 18:21:10 2017
2 | Pasajero | Fri Dec 22 20:29:28 2017
3 | Modelo | Fri Dec 22 20:30:25 2017
4 | Vehiculo | Fri Dec 22 20:30:47 2017
5 | Lugar | Fri Dec 22 20:31:28 2017
6 | Ticket | Fri Dec 22 20:32:50 2017
7 | Viaje | Fri Dec 22 20:33:09 2017

Presione ENTER para continuar...
```