FIEC

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

PROGRAMACIÓN APLICADA A LA AUTOMATIZACIÓN

M.Sc. Alexander Prieto León

Conferencia XIV

Unidad IV Comunicación y manejo de datos en automatización.

4.2 Bases de Datos.

OBJETIVOS

Objetivos:

- Elaborar aplicaciones que incluyan comunicación con otros dispositivos y manejo de Base de Datos.

BIBLIOGRAFÍA

Bibliografía:

- Zhi Eng, Lee; Rischpater, Ray. Application Development with Qt Creator: Build cross-platform applications and GUIs using Qt 5 and C++, 3rd Edition. (2020). Packt Publishing. ISBN-10: 1789951755, ISBN-13: 978-1789951752.
- http://www.qtrac.eu/C++-GUI-Programming-with-Qt-4-1st-ed.zip
- Qt 5.14.2 Reference Documentation. Qt Creator Help.
- https://doc.qt.io/qtcreator/index.html
- https://doc.qt.io/qt-5/qtdesigner-manual.html

- **En la clase anterior:**

INTRODUCCIÓN

- **¿Qué es una base de datos?**

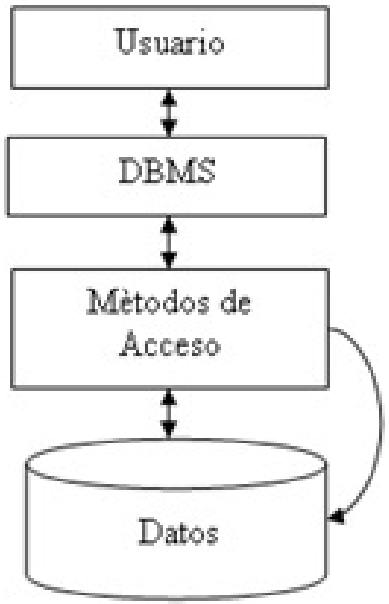
¿Qué es una base de datos?

- Una **base de datos** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta.

- Una **base de datos** es una colección organizada de datos , generalmente almacenados y accesibles electrónicamente desde un sistema informático.

DBMS

- Hay programas denominados sistemas gestores de bases de datos, abreviado SGBD (del inglés Database Management System o DBMS), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada.



¿Dónde se almacenan los datos?

En ficheros en el disco duro de la computadora, generalmente, servidores.

DBMS

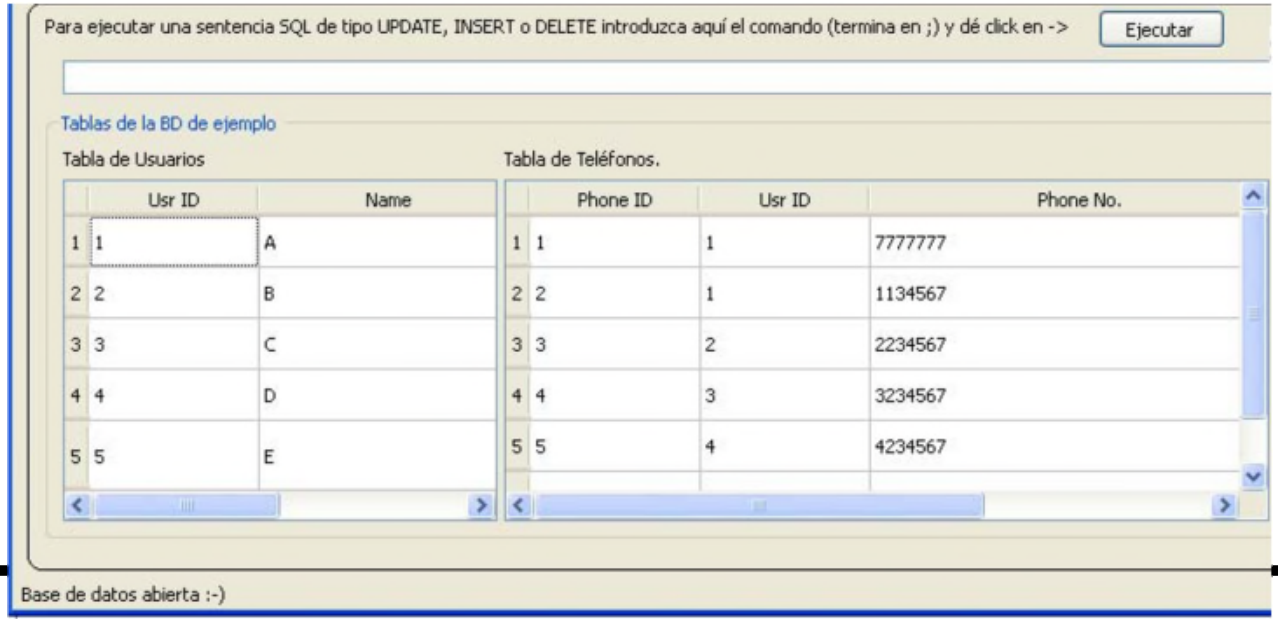
- Los DBMS existentes proporcionan varias funciones que permiten la gestión de una base de datos y sus datos, que se pueden clasificar en cuatro grupos funcionales principales:
  - **Definición de datos**: creación, modificación y eliminación de definiciones que definen la organización de los datos.
  - **Actualización**: inserción, modificación y eliminación de los datos reales.
  - **Recuperación**: proporcionar información en una forma directamente utilizable o para su posterior procesamiento por otras aplicaciones. Los datos recuperados pueden estar disponibles en una forma básicamente igual a la que se almacenan en la base de datos o en una nueva forma obtenida alterando o combinando datos existentes de la base de datos.
  - **Administración**:…

Modelos de base de datos

- Los informáticos pueden clasificar los sistemas de gestión de bases de datos de acuerdo con los modelos de base de datos que admiten.
  - Las bases de datos relacionales se hicieron dominantes en la década de 1980. Estos modelan datos como filas y columnas en una serie de tablas , y la gran mayoría usa SQL para escribir y consultar datos.
  - En la década de 2000, las bases de datos no relacionales se hicieron populares, conocidas como NoSQL porque utilizan diferentes lenguajes de consulta (más usado XML)

Bases de datos relacionales

- Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas".
- Esto es pensando en cada relación como si fuese una tabla que está compuesta por *registros* (las filas de una tabla), que representarían las tuplas, y *campos* (las columnas de una tabla).



SQL

- El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, *Structured Query Language* o *Lenguaje Estructurado de Consultas*, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

- Muy utilizado por los sistemas SCADA en automatización y en la amplia mayoría de los sistemas informáticos.

SQL

- Presentación: Lenguaje SQL de Pedro Pablo Alarcón

SQL en Qt

- Cuando no se utiliza un DBMS para realizar las operaciones y consultas sobre una BD y visualizar los resultados, se necesita un mecanismo que nos "conecte" con esa base de datos.

- En el caso de Qt, existe un manipulador (driver) que nos "conecta" con esa base de datos y se encarga de la "comunicación" con la misma. Se puede considerar un DBMS interno para ser usado mediante programación

SQL en Qt

- Qt tiene incorporados varios **Drivers** para poder manipular diferentes bases de datos. La siguiente figura refleja los drivers de Qt así como el string que los identifica:

Driver	Database
QDB2	IBM DB2 version 7.1 and later
QIBASE	Borland InterBase
QMYSQL	MySQL
QOCI	Oracle (Oracle Call Interface)
QODBC	ODBC (includes Microsoft SQL Server)
QPSQL	PostgreSQL versions 6.x and 7.x
QSQLITE	SQLite version 3 and later
QSQLITE2	SQLite version 2
QTDS	Sybase Adaptive Server

SQL en Qt

- El complemento Qt SQLite es muy adecuado para el almacenamiento local.
- SQLite es un sistema de administración de bases de datos relacionales contenido en una pequeña biblioteca C (~ 350 KiB).
- A diferencia de otros sistemas de administración de bases de datos, SQLite no es un proceso separado al que se accede desde la aplicación cliente, sino una parte integral de ella.
- SQLite opera en un solo archivo, que debe establecerse como el nombre de la base de datos al abrir una conexión. Si el archivo no existe, SQLite intentará crearlo.

SQL en Qt

Layer	Purpose	Example class
•Driver layer	•Low-level communication between database and the SQL API layer	•QSqlDriver, QSqlDriverCreator
•SQL API layer	•Provide access to databases	•QSqlDatabase, QSqlQuery
•User Interface layer	•Link data from a database to data-aware widgets	•QSqlQueryModel (readonly), QSqlTableModel (read/write), QSqlRelationalTableModel (read/write with foreign-key support)

SQL en Qt

QSql	Contains miscellaneous identifiers used throughout the Qt SQL library
QSqlDatabase	Represents a connection to a database
QSqlDriver	Abstract base class for accessing specific SQL databases
QSqlDriverCreator	Template class that provides a SQL driver factory for a specific driver type
QSqlDriverCreatorBase	The base class for SQL driver factories
QSqlError	SQL database error information
QSqlField	Manipulates the fields in SQL database tables and views

SQL en Qt

QSqlIndex	Functions to manipulate and describe database indexes
QSqlQuery	Means of executing and manipulating SQL statements
QSqlQueryModel	Read-only data model for SQL result sets
QSqlRecord	Encapsulates a database record
QSqlRelationalTableModel	Editable data model for a single database table, with foreign key support
QSqlResult	Abstract interface for accessing data from specific SQL databases
QSqlTableModel	Editable data model for a single database table

SQL API layer: QSqlDatabase, QSqlQuery

- Problema: se desea crear una aplicación que permita introducir registros de usuarios a un BD de tal forma que estos contengan el nombre del usuario (n_usr) y su año de nacimiento (birthyear).

SQL API layer: QSqlDatabase, QSqlQuery

```
#include <QCoreApplication>
#include <QtSql/QSqlDatabase>
#include <QtSql/QSqlQuery>
#include <QtSql/QSqlError>
#include <QDebug>
#include <iostream>
#include <QString>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
```

SQL API layer: QSqlDatabase, QSqlQuery

```
if (!QSqlDatabase::isDriverAvailable("QSQLITE"))
{
    cout<<"No hay driver QSQLITE!";
    return 0;
}
else cout<<"Esta el driver disponible.";

QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName("C:/Users/alexa/Documents/BD_NoVisual1/bd_ejemp.db");
if (!db.open())
{
    qDebug()<<db.lastError().text();
    return 0;
}
cout<<"Conectado...\n";
```

SQL API layer: QSqlDatabase, QSqlQuery

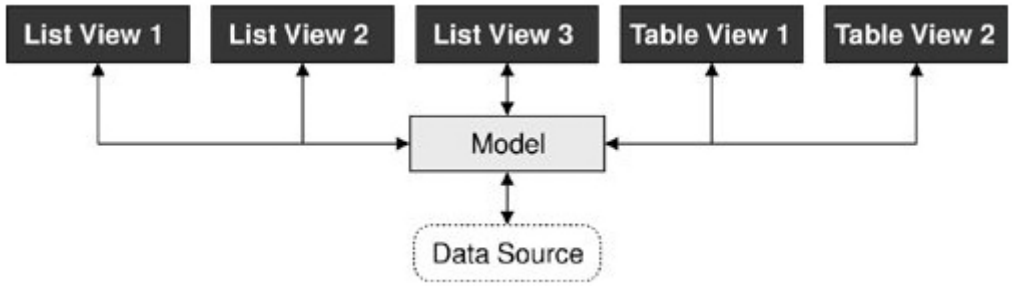
```
QSqlQuery query;
if (!query.exec("CREATE TABLE IF NOT EXISTS usr "
                "(id_usr INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "
                "n_usr VARCHAR(255) NOT NULL,"
                "birthyear INTEGER NOT NULL)"))
    qDebug()<<query.lastError().text();

query.exec("INSERT INTO usr(n_usr,birthyear) VALUES ('Jose',2001)");
```

SQL API layer: QSqlDatabase, QSqlQuery

```
// ejecutando la consulta
QString SqlStmt = "SELECT * FROM usr;";

query.exec(SqlStmt); // sacando los valores obtenidos de la consulta

while (query.next())
{
    int id=query.value("id_usr").toInt();
    QString nombre = query.value("n_usr").toString();
    int anno=query.value("birthyear").toInt();
    cout <<"ID:"<<id
         << " Nombre: "<<nombre.toStdString()
         <<" Year: "<<anno<<endl;
}

db.close();
return a.exec();
}
```

Arquitectura Modelo-Vista

- La arquitectura **Modelo/Vista** permite separar la manera en que los datos son almacenados de la manera en que estos son presentados al usuario. Esta separación permite mostrar en diferentes vistas los mismos datos así como crear nuevos tipos de vistas sin que se afecten las estructuras de datos utilizadas.



- **Modelo.** El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

- **Vista.** Maneja la visualización de la información.

QSqlQueryModel y QSqlTableModel (Arquitectura Modelo-Vista)

- QSqlQueryModel es una interfaz de alto nivel para ejecutar declaraciones SQL y recorrer el conjunto de resultados. Está construido sobre QSqlQuery de nivel inferior y se puede utilizar para proporcionar datos para ver clases como QTableView. Por ejemplo:

```
QSqlQueryModel *model = new QSqlQueryModel;
model->setQuery("SELECT name, salary FROM employee");
model->setHeaderData(0, Qt::Horizontal, "Name");
model->setHeaderData(1, Qt::Horizontal, "Salary");

QTableView *view = new QTableView;
view->setModel(model);
view->show();
```

QSqlQueryModel y QSqlTableModel (Arquitectura Modelo-Vista)

- Si se utilizara de forma similar la clase **QSqlTableModel** entonces los cambios realizados manualmente sobre la **TableView** se transmitarán directamente al modelo, o sea, modificará los datos en la BD.

- Ver 2do ejemplo en capítulo 13 del libro: C++ GUI programming with Qt 4.

CONCLUSIONES

- De esta manera
- En el laboratorio…