Gnome Data Access Library

Libgda

Xabier Rodríguez Calvar

<xrcalvar@igalia.com>

Que é Libgda...

- É unha biblioteca que abstrae o acceso a fontes de datos.
  - BD relacionais.
  - Ficheiros XML.
  - Directorios LDAP.
- A abstracción conséguese mediante o uso de plugins sinxelamente extensíbeis.
- Centrarémonos nas *BD relacionais* accedidas a través de *SQL*.
- É software libre pertencente á plataforma GNOME.

Coñecementos previos necesarios

- Linguaxe de programación *C*.
  - Tamén hai *bindings* para as principais linguaxes, como *C++*...
- Linguaxe de acceso a bases de datos *SQL*.
  - O acceso será estándar a non ser que usemos extensións propietarias de *SQL*.
- Pode obterse en `http://www.gnome-db.org`.
  - Hai paquetes debian en *testing* e *unstable*.

Comezando e compilando

- Ha de incluirse o ficheiro de cabeceiras `libgda/libgda.h`.

```
#include <libgda/libgda.h>
```

- Para compilar ha de facerse usando `pkg-config` do seguinte xeito:

```
gcc -c full_example.c ‘pkg-config --cflags libgda‘
gcc -o full_example ‘pkg-config --libs libgda‘ full_example.o
```

- No comezo do programa chamamos a `gda_init()` para inicializa-la librería pasándolle o nome da aplicación e a versión, xunto cos argumentos do programa:

```
gda_init ("TestGDA", "0.1", argc, argv);
```

Data Sources

- Os *data sources* son os medios polos cales se identifican as conexións ás bases de datos. Almacénase os datos relativos ó fornecedores (*driver*), servidor, etc.
- Para xestionalos pódese facer de dous xeitos:
  - Usando a aplicación `gnome-database-properties`.
  - Usando a aplicación `Mergeant`.
    - Permite o acceso ás táboas de xeito gráfico.
  - Coas funcións da API:

```
gda_config_save_data_source ("calvaris", "PostgreSQL",
     "DATABASE=calvaris",
     "exemplo", NULL, NULL);
```

Data Sources (e 2)

- A función `gda_config_save_data_source()` recibe cinco parámetros:
  - Nome do data source.
  - Fornecedor do xestor da base de datos.
  - String da conexión, que indica a base de datos, o servidor, e outros parámetros dependentes do xestor. Exemplo:
    - `"DATABASE=macro$oft,HOST=un.host.com`
  - Comentario.
  - Nome de usuario na base de datos.
  - Clave na base de datos.

Chamando a funcións

- Á hora de chamar a funcións que encapsulen código da biblioteca pódese facer de dous xeitos:
  - Chamar á función directamente.
  - Usa-la función `gda_main_run()` chamando ó final desta a `gda_main_quit()`:

```
gda_main_run ((GdaInitFunc) do_stuff, (gpointer) NULL);
```

- Se queremos suscribirnos a eventos é mester usa-la segunda forma.

Conectando

- Para conectar será mester facer dúas cousas:
  - Obter un cliente (*pool* de conexións) (`gda_client_new()`).
  - Obter unha conexión (`gda_client_open_connection()`). Recibe como parámetros: o cliente, data source, nome de usuario, clave, opcións da conexión (xeralmente `GDA_CONNECTION_OPTIONS_READ_ONLY`).
- Ó final pódense pechar tódalas conexións do pool con `gda_client_close_all_connections()`.

Conectando (e 2)

```
void
do_stuff ()
{
  GdaClient *client;
  GdaConnection *connection;

  client = gda_client_new ();

  connection = gda_client_open_connection (client, "calvaris",
        NULL, NULL, GDA_CONNECTION_OPTIONS_READ_ONLY);

  /* executing_some_queries (connection); */

  gda_client_close_all_connections (client);
  g_object_unref (G_OBJECT (client));
  gda_main_quit ();
}
```

Executar comandos

- O primeiro que hai que facer é crea-lo comando con `gda_command_new()` cos seguintes parámetros:
  - Texto do comando, normalmente en SQL.
  - Tipo do comando, normalmente `GDA_COMMAND_TYPE_SQL`.
  - Opcións. Como só lanzamos unha sentencia *SQL* por comando usaremos `GDA_COMMAND_OPTION_STOP_ON_ERRORS`.
- Ó rematar para ceibar usamos `gda_command_free()`.

Executar comandos (e 2)

- Á hora de executar comandos teremos dous xeitos de facelo:
  - `gda_connection_execute_non_query()` que devolve o número de tuplas afectadas pola sentencia.
  - `gda_connection_execute_single_command()` que devolverá os datos en forma de *data model*.
- Ámbolos dous métodos necesitan como parámetros:
  - A conexión (`GdaConnection`).
  - O comando (`GdaCommand`).
  - Outros parámetros, para *prepared statements* (se non desexamos usalos pasamos `NULL`).

Executar comandos (e 3)

- Exemplo:

```
gint
execute_sql_non_query (GdaConnection *connection,
                       const gchar * buffer)
{
  GdaCommand *command;
  gint number;
  command = gda_command_new (buffer,
                             GDA_COMMAND_TYPE_SQL,
                             GDA_COMMAND_OPTION_STOP_ON_ERRORS);
  number  = gda_connection_execute_non_query (connection,
                                              command, NULL);
  gda_command_free (command);
  return (number);
}
```

Executar comandos (e 4)

- Os *prepared stetements* usanse para parametrizar consultas.
  - Coa mesma estructura.
  - Distintos datos.
- Vantaxes:
  - Só as escribimos unha vez, podendo centralizalas.
  - O xestor debería planificalas unha soa vez.
- Non están implementadas aínda.

Executar comandos (e 5)

- Sintaxe *SQL*:
  - Como *SQL* normal.
  - Os parámetros especifícanse antepoñendo **:**
- Exemplo:

```
INSERT INTO accounts (acc_name, acc_owner, acc_balance)
        VALUES (:name, :owner, :balance)
```

Executar comandos (e 6)

- Para crear un parámetro usamos `gda_parameter_new()`, pasando:
  - Nome
  - Un `GdaValue`.
    - Para crear un obxecto desta clase usámo-las funcións do tipo `gda_value_new_`*`tipo_desexado`*`()`.

Executar comandos (e 7)

- Para crear unha lista de parámetros usamos `gda_parameter_list_new()`.
- Para introduci-lo parámetro na lista usamos `gda_parameter_list_add_parameter()`, pasando
  - Lista
  - Parámetro
- Para ceibar estes obxectos temos dúas funcións:
  - Lista de parámetros: `gda_parameter_list_free()`
  - `GdaValue`'s: `gda_value_free()`

Executar comandos (e 8)

```
void
play_with_parameters ()
{
  GdaParameterList *list;
  GdaParameter *parameter;
  GdaValue *value;

  list = gda_parameter_list_new ();

  value = gda_value_new_integer (10);
  parameter = gda_parameter_new ("p1", value);
  gda_parameter_list_add_parameter (list, parameter);
  gda_value_free (value);
```

Executar comandos (e 9)

```
  value = gda_value_new_integer (2);
  parameter = gda_parameter_new ("p2", value);
  gda_parameter_list_add_parameter (list, parameter);
  gda_value_free (value);

  /* create command and execute query */

  gda_parameter_list_free (list);
}
```

Extraendo datos

- Os datos devólvense en forma de `GdaDataModel`, que haberá que ceibar con `g_object_unref()`.
- Hai dous xeitos de acceder:
  - A nivel de táboa.
  - A nivel de fila.
- Explicarémo-lo nivel de fila.

Extraendo datos (e 2)

- Exemplo:

```
void
show_table2 (GdaDataModel * dm)
{
  gint row_id;
  gint column_id;
  GdaValue *value;
  GdaRow *row;
  gchar *string;

  for (column_id = 0; column_id <
          gda_data_model_get_n_columns (dm);
       column_id++)
    g_print("%s\t",gda_data_model_get_column_title (dm,
                                                   column_id));
  g_print("\n");
```

Extraendo datos (e 3)

```
    for (row_id = 0; row_id <
              gda_data_model_get_n_rows (dm); row_id++)
      {
        row = (GdaRow *) gda_data_model_get_row (dm, row_id);
        for (column_id = 0; column_id <
                  gda_data_model_get_n_columns (dm);
              column_id++)
          {
            value = gda_row_get_value (row, column_id);
            string=gda_value_stringify (value);
            g_print ("%s\t", string);
            g_free(string);
          }
        g_print ("\n");
      }
}
```

Extraendo datos (e 4)

- O primeiro bucle imprime os nomes de columna:
  - `gda_data_model_get_n_columns()` devolve o número de columnas.
  - `gda_data_model_get_column_title()` devolve o nome de columna.
- O bucle exterior fai o traballo para tódalas filas:
  - `gda_data_model_get_n_rows()` devolve o número de filas.
  - `gda_data_model_get_row()` devolve fila. Non se ha de ceibar porque devolve `const`.

Extraendo datos (e 5)

- O bucle interior fai o traballo para cada fila:
  - `gda_row_get_value()` devolve un `GdaValue` que NON ha de liberarse ó final.
- O para extrae-lo valor hai diversas funcións, como por exemplo:
  - `gda_value_stringify()` (ha de liberarse despois con `g_free()`).
  - `gda_value_is_null()`
  - `gda_value_copy()`
  - `gda_value_get_integer()`
  - `gda_value_get_double()`
  - ...

Transaccións

- Os pasos que hai que levar a cabo son:
  - Crea-la transacción (`gda_transaction_new()`).
  - Cambia-lo nivel de aíllamento da transacción (`gda_transaction_set_isolation_level()`):
    - `GDA_TRANSACTION_ISOLATION_READ_UNCOMMITTED`
    - `GDA_TRANSACTION_ISOLATION_READ_COMMITTED`
    - `GDA_TRANSACTION_ISOLATION_REPEATABLE_READ`
    - `GDA_TRANSACTION_ISOLATION_SERIALIZABLE`

Transaccións (e 2)

- Os pasos que hai que levar a cabo son:
  - Enlaza-la transacción á conexión (`gda_connection_begin_transaction()`).
  - Para cada comando:
    - Crea-lo comando (`gda_command_new()`).
    - Enlaza-lo comando á transacción (`gda_command_set_transaction()`)
    - Executa-lo comando (`gda_connection_execute_single_command()` ou `gda_connection_execute_non_query()`)
    - Libera-lo comando (`gda_command_free()`)

Transaccións (e 3)

- Ó final:
  - Facer *commit* ou *rollback* da transacción (`gda_connection_commit_transaction()` ou `gda_connection_rollback_transaction()`)
  - Libera-la transacción (`g_object_unref()`).

Transaccións (e 4)

- Exemplo:

```
transaction_one=gda_transaction_new("accounts1");
gda_transaction_set_isolation_level(transaction_one,
                GDA_TRANSACTION_ISOLATION_SERIALIZABLE);
gda_connection_begin_transaction(connection,transaction_one);

command=gda_command_new (
                          "UPDATE accounts SET balance=balance+50"
                          "WHERE account_code=456",
                          GDA_COMMAND_TYPE_SQL,
                          GDA_COMMAND_OPTION_STOP_ON_ERRORS);
gda_command_set_transaction(command,transaction_one);
gda_connection_execute_non_query(connection,command,NULL);
gda_command_free(command);
```

Transaccións (e 4)

```
command=gda_command_new (
                          "UPDATE accounts SET balance=balance-50"
                          "WHERE account_code=12",
                          GDA_COMMAND_TYPE_SQL,
                          GDA_COMMAND_OPTION_STOP_ON_ERRORS);
gda_command_set_transaction(command,transaction_one);
gda_connection_execute_non_query(connection,command,NULL);
gda_command_free(command);

gda_connection_commit_transaction(connection,transaction_one);
g_object_unref(transaction_one);
```

Erros

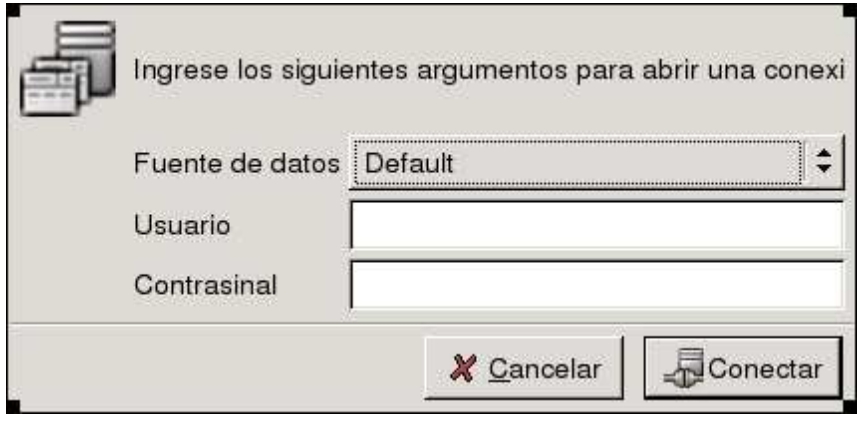
- Despois de executar un comando pódense ve-los erros usando a función `gda_connection_get_errors()` sobre unha conexión, do seguinte xeito:

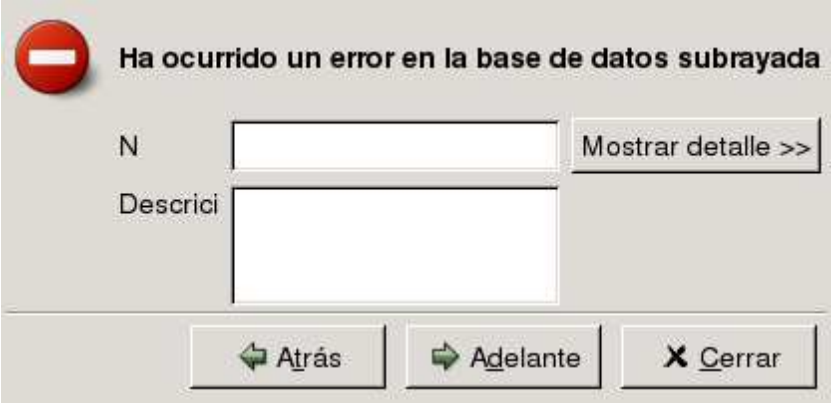
```
list = (GList *) gda_connection_get_errors (connection);

 for (node = g_list_first (list); node != NULL;
      node = g_list_next (node))
   {
     error = (GdaError *) node->data;
     g_print ("Error no: %d\t", gda_error_get_number (error));
     g_print ("desc: %s\t", gda_error_get_description (error));
     g_print ("source: %s\t", gda_error_get_source (error));
     g_print ("sqlstate: %s\n", gda_error_get_sqlstate (error));
   }
```

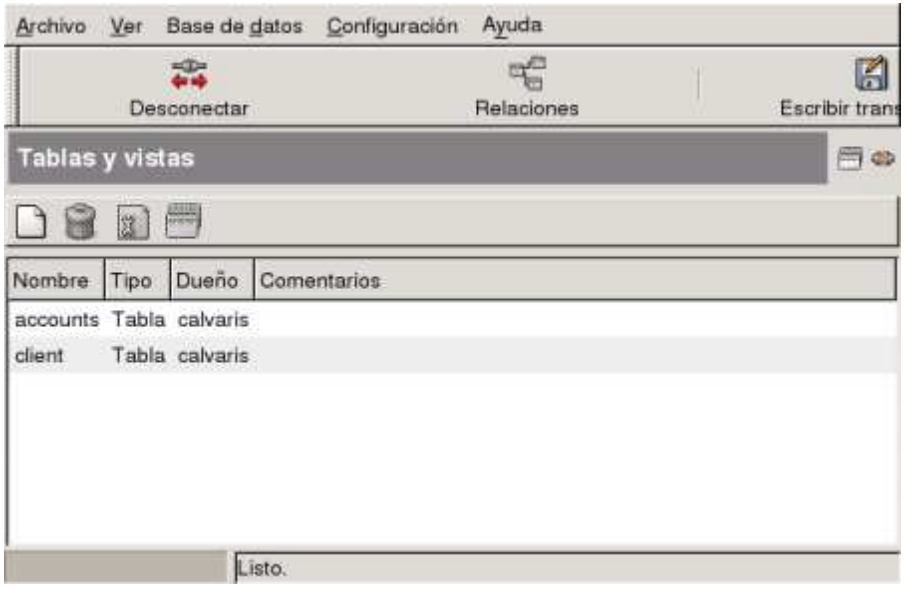
libgnomedb

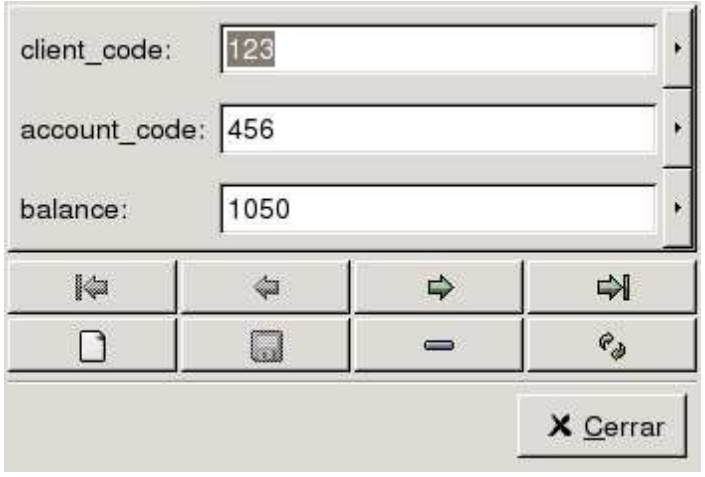
- Librería de *widgets* baseada en *gtk* para manexar bases de datos.





- Aplicación para manexo de BD.





Fin

- Manual de libgda2:
  `http://www.gnome-db.org/docs/libgda/index.html`
- Web do proxecto *Gnome-db*:
  `http://www.gnome-db.org`
- GPUL: `http://www.gpul.org`