Développer une application Oracle en C/C++ avec la librairie OCILIB

par Vincent Rogier

Date de publication : 07/01/2008

Dernière mise à jour : 20/04/2009

Comment programmer simplement et efficacement des applications Oracle performantes en C/C++.

I - Public concerné

II - Présentation

II-A - Révision

Liste des révisions du document :

- 07/01/2008 - Publication (version OCILIB 1.5.1)
- 13/02/2008 - Mise à jour (version OCILIB 2.0.0)
- 21/02/2008 - Mise à jour (paragraphe sur les performances + correction de la génération PDF)
- 07/03/2008 - Mise à jour (correction paragraphe III-E - options du linker sous linux/unix)
- 20/03/2008 - Mise à jour (correction paragraphe XIII - Fonctionnalités)
- 29/03/2008 - Mise à jour (correction coquilles code : OCI_Statment -> OCI_Statement + ajouts de commits)
- 01/04/2008 - Mise à jour (version OCILIB 2.3.0 : liste des fonctionnalités)
- 24/04/2008 - Mise à jour (version OCILIB 2.4.0 + diverses modifications du tutoriel)
- 21/07/2008 - Mise à jour (version OCILIB 2.5.0 + diverses modifications du tutoriel)
- 13/10/2008 - Mise à jour (version OCILIB 3.0.0 : mise à jour majeure)
- 28/01/2009 - Mise à jour (version OCILIB 3.1.0)
- 20/04/2009 - Mise à jour (version OCILIB 3.2.0)

II-B - Introduction

OCILIB est un driver Oracle open source et portable qui assure des accès performants et fiables aux bases de données Oracle.
La librairie OCILIB :

- offre une API riche et simple à utiliser
- tourne sur toutes les plateformes Oracle
- est écrite en pur ISO C avec un support natif Unicode en ISO C
- encapsule OCI (Oracle Call interface)
- est le wrapper OCI le plus complet disponible

II-C - Complexité d'OCI

Oracle Call Interface (OCI) est une API fournie par Oracle permettant aux développeurs de créer des applications en utilisant des appels C/C++ bas niveau afin d'accéder à des bases de données Oracle. OCI permet de contrôler tous les aspects de l'exécution d'ordres SQL tout en supportant les types de données, les conventions d'appel, la syntaxe et la sémantique des langages C, C++.
OCI est une API très puissante (c'est l'API la plus bas niveau fournie par Oracle) et utilisée par beaucoup d'applications et outils (à commencer par Sql*Plus, Sql*Loader et tous les produits Oracle) mais aussi très complexe et lourde à utiliser...
Par exemple, se connecter à Oracle requiert au minimum quasiment 100 lignes de code ! De plus, la plupart des fonctions d'OCI requiert souvent une liste d'arguments assez conséquente (souvent une dizaine de paramètres) et complexe (manipulation quasi exclusive de pointeurs, pointeurs de pointeurs, etc...)
Au final, une application OCI est souvent pénible à coder, relire et à maintenir et les "experts" en OCI ne sont pas aussi nombreux que les codeurs Java, .NET ou VB !

II-D - Pourquoi OCILIB ?

OCILIB encapsule OCI afin de fournir une interface beaucoup plus simple à coder, lire et surtout afin de permettre une réutilisabilité du code optimale.
OCILIB est gratuit (open source - license LGPL) et fournit près de **500** fonctions simples et son code source est indépendant de toute plateforme.
Les langages C et C++ sont parmi les plus performants et malgré cela, souvent lors du choix d'un langage pour coder une application Oracle, ils ne sont pas retenus, au profit de frameworks de type JAVA et .NET. Ils sont écartés bien souvent pour des raisons d'accès car les API C/C++ pour accéder à Oracle (fournies ou non par Oracle) sont complexes à mettre en oeuvre.
OCILIB permet de concilier les performances de C/C++ avec une mise en oeuvre très simple via une interface la plus pragmatique possible tout en gardant une richesse dans les fonctionnalités.
OCILIB est une des rares librairies C/C++ basées sur OCI apportant un support complet Unicode. Une application OCILIB peut être indifféremment compilée nativement en Ansi ou en Unicode.
Enfin, le code d'OCILIB est 100 % ISO C90 (et C99 pour l'Unicode) ce qui lui permet d'être extrêmement portable !
OCILIB ne nécessite pas de client Oracle pour développer. En effet, une application utilisant OCILIB peut lier les librairies Oracle à la compilation (liaison statique ou dynamique) ou charger dynamiquement ces librairies à l'exécution si l'OS cible supporte le chargement dynamique de modules.
OCILIB fonctionne sur toutes les plateformes supportées par Oracle (Microsoft Windows, Linux, Unixes, Mac, ...).
Actuellement, OCILIB est très utilisé dans :

- des applications bancaires
- des applications industrielles
- des frameworks génériques d'accès aux bases de données pour encapsuler l'accès à Oracle

III - Installation

III-A - Près requis

- Posséder un système Unix like ou Microsoft Windows supporté par Oracle
- Avoir un client Oracle classique ou Instant (non requis pour la compilation mais pour l'exécution)

III-B - Compatibilités

=> Les plateformes suivantes ont été validées (compilation, installation, construction et exécution d'un projet) :

- Microsoft Windows
- Linux
- Solaris
- HP/UX
- AIX
- Mac OS X
- OpenVMS

Les autres plateformes supportées par Oracle (comme z/OS par exemple) n'ont pas été officiellement "validées", mais au vu des plateformes déjà validées, cela ne devrait pas être trop problématique !
Remarque : Si vous validez OCILIB sur une plateforme non listée, merci de me contacter !

=> Les compilateurs suivants ont été validés :

- Microsoft Compilers (VC++, VS200X)
- GCC et MinGW
- IBM XLC
- CC propriétaires (HP, AIX, ...)

- LabView

=> Les versions d'Oracle suivantes (clients et serveurs) ont été validées :

- Oracle 8i
- Oracle 9i
- Oracle 10g
- Oracle 11g

III-C - Installation sous Unixes (Unix/Linux/Mac)

OCILIB utilise les outils GNU pour son déploiement et son installation.

- Décompresser l'archive courante (ocilib-x.y.z-gnu.tar.gz)
- $ cd ocilib-x.y.z
- $ ./configure
- $ ./make
- $ ./make install (généralement, il faut passer en root pour l'installation)

Vérifier la variable d'environnement des chemins d'accès des librairies dynamiques (LD_LIBRARY_PATH, LD_PATH, ..) afin que le répertoire des librairies Oracle s'y trouve ainsi que celui où est installé OCILIB (typiquement / usr/local/lib sous linux).
Typiquement, il suffit de rajouter à son fichier .profile :

- $ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib:/usr/local/lib

OCILIB supporte 6 options supplémentaires d'installation :

- --with-oracle-import=(linkage|runtime)
- --with-oracle-charset=(ansi|unicode|mixed)
- --with-oracle-home=(répertoire oracle)
- --with-oracle-headers-path=(chemin d'accès aux headers OCI)
- --with-oracle-lib-path=(chemin d'accès aux librairies d'Oracle)
- --with-custom-loader=(flag pour le linker indiquant le loader à utiliser pour le chargement dynamique à l'éxecution des librairies Oracle (par défaut "-ldl") si la plateforme utilise un loader particulier)

Les deux options 'headers-path' et 'lib-path' sont à utiliser pour les Instant Clients (packages différents pour les headers et les librairies).
Si le package contenant les librairies ne possède pas de lien symbolique sur la librairie partagée complètement qualifiée avec les numéros de version, il faut en créer un. Par exemple, sous Linux/Oracle 10g R1 :

- $ ln -s $ORALIBPATH/libclntsh.so.10.1 $ORALIBPATH/libclntsh.so

Remarque : Si le compilateur est GCC, par défaut les outils GNU (autoconf et automake) lui passent les flags '-g -O2'. Ce qui ajoute donc des infos pour le débogage dans les librairies générées :

- Jusqu'aux versions 2.x.x, cela pouvait augmenter la taille finale de la librairie de 50 % en moyenne (300 Ko sous Linux)
- Avec la version 3.0.0 et la scission du source princal en plusieurs source, l'augmentation peut être de 300 % ! (650 Ko sous Linux)
- Donc, il est possible de retirer ces informations en "stripant" les librairies finales libocilib.[ext].x.y.z (ext => 'so', 'a', ...) avec la commande strip. Ce qui sous linux fait passer de 650 Ko à 170Ko

III-D - Installation sous Microsoft Windows

Sous Microsoft Windows, des DLLs sont fournies pour les environnement 32bits et 64bits (x86) et peuvent être recompilées aisément.
Remarque : Avec la version 3.0.0, le fichier d'entête principal a été déplacé de 'ocilib\src' vers 'ocilib\include\'.

- Décompresser l'archive courante (ocilib-x.y.z-windows.zip)
- Copier ocilib\include\ocilib.h dans un répertoire inclus dans la liste de l'option "fichier d'entêtes" du compilateur
- Copier ocilib\lib32|64\ocilib[x].lib dans un répertoire inclus dans la liste de l'option "fichier bibliothèques" de l'éditeur de lien
- Copier ocilib\lib32|64\ocilib[x].dll dans un répertoire inclus dans la variable d'environnement PATH

[x] représente la version compilée d'OCILIB : "a" pour Ansi, "w" pour Unicode et "m" pour mixed mode

III-E - Configuration de projets

Pour utiliser OCILIB dans un projet, il faut inclure le header "ocilib.h".
Certaines options doivent être définies avant l'inclusion du header ocilib.h

=> Mode de liaison Oracle :

- OCI_IMPORT_LINKAGE : pour lier les librairies Oracle (statiques ou partagées) à la compilation **(option par défaut sous Unix like)**
- OCI_IMPORT_RUNTIME : pour charger dynamiquement les librairies Oracle à l'exécution **(option par défault dans les projets MS Windows fournis)**

=> Charsets utilisés :

- OCI_CHARSET_ANSI : toute les chaînes de caractères sont en Ansi **(option par défaut)**
- OCI_CHARSET_UNICODE : toute les chaînes de caractères sont en Unicode **(versions de Oracle >= 9i)**
- OCI_CHARSET_MIXED : Ordres SQL, métadatas en Ansi et données utilisateurs fournies et récupérées des requêtes en Unicode

=> Convention d'appel **(MS Windows uniquement)** :

- OCI_API : non défini **(option par défaut)**
- OCI_API : __sdtcall pour utiliser les dll précompilées

Sous Windows, pour utiliser OCILIB en ANSI, en utilisant les Dlls fournies, il suffit de créer un nouveau projet et :

- Inclure "ocilib.h"
- Définir OCI_API=__sdtcall dans les options du préprocesseur du projet
- Si toutes les variantes ocilib[x].lib sont disponibles pour le linker, il faut alors préciser quel version en insérant **#pragma comment(lib, "ocilib[x].lib")** dans un des fichiers du projet

Remarque : Avec la version 3.2.0, des librairies statiques sont aussi fournies pour les portages Windows de GCC : libociliba.a, libocilibm.a et libocilibw.a
Sous Linux, pour utiliser OCILIB en ANSI et avec une liaison des librairies partagées d'Oracle à la compilation, il faut :

=> Supposons que $USER_LIBS pointe vers le répertoire parent ou est installé OCILIB (par défaut : usr/local sous Linux)
=> Inclure "ocilib.h"
=> Ajouter au makefile pour le compilo :

- -I$USER_LIBS/include pour le header ocilib.h
- -DOCI_IMPORT_LINKAGE -DOCI_CHARSET_ANSI pour configurer ocilib

=> Ajouter au makefile pour le linker :

- -L/$ORACLE_HOME/lib -lclntsh pour les librairies Oracle
- -L$USER_LIBS/lib -locilib pour la librairie OCILIB

IV - Se Connecter à Oracle

IV-A - Initialiser OCILIB

Avant tout chose, OCILIB doit être initialisée. Pour cela il faut, avant tout appel à une fonction de la librairie, appeler OCI_Initilalize(). Cette fonction initialise la librairie et prend en paramètres :

- [Optionnel] Une pointeur sur une fonction de gestion des erreurs
- [Optionnel] Un répertoire ou se trouve les librairies Oracle (si runtime loading et plusieurs clients Oracle installés)
- [Optionnel] Le mode d'initialisation

Les valeurs possibles pour le mode d'initialisation sont :

- OCI_ENV_DEFAULT : mode par défaut
- OCI_ENV_THREADED : activation du support multithread
- OCI_ENV_CONTEXT : activation de la gestion contextuelle des erreurs par thread

Tout tentative de création d'un objet OCILIB avant l'initialisation génèrera une erreur qui pourra être récupérée par OCI_GetLastError().

Exemple

```
#include "ocilib.h"

int main()
{
    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    /* ... application code here ... */

    OCI_Cleanup();


    return EXIT_SUCCESS;
}
```

En fin d'application, un appel à OCI_Cleanup() est nécessaire pour :

- Désallouer les objets non explicitement libérés
- Décharger les librairies Oracle dans le cas d'un runtime loading

IV-B - Connexions

Se connecter à une base de données Oracle s'effectue par la fonction OCI_ConnectionCreate() qui prend en paramètres :

- Le nom du service Oracle (alias Oracle, SID, ... en fonction des paramètres du client Oracle).
- Le nom du user Oracle sur lequel se connecter
- Le mot de passe du user Oracle
- Le mode de session (OCI_SESSION_DEFAULT, OCI_SESSION_SYSDBA, OCI_SESSION_SYSOPER)

Les valeurs possibles pour le mode de session sont :

- OCI_SESSION_DEFAULT mode de session par défaut
- OCI_SESSION_SYSDBA : mode de session sysdba Oracle
- OCI_SESSION_SYSOPER : mode de session sysoper Oracle

Cette fonction de charge de se connecter au serveur, créer une session et d'établir une transaction. En cas de succès, elle renvoie un handle de connexion et l'application peut de suite exécuter des ordres SQL.

Exemple

```
#include "ocilib.h"

int main(void)
{
  OCI_Connection *cn;

  if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
    return EXIT_FAILURE;

  cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);

  if (cn != NULL)
  {
    printf(OCI_GetVersionServer(cn));

    printf("Server major    version : %i\n", OCI_GetServerMajorVersion(cn));
    printf("Server minor    version : %i\n", OCI_GetServerMinorVersion(cn));
    printf("Server revision version : %i\n", OCI_GetServerRevisionVersion(cn));

    printf("Connection      version : %i\n", OCI_GetVersionConnection(cn));

    OCI_ConnectionFree(cn);
  }

  OCI_Cleanup();

  return EXIT_SUCCESS;
}
```

Pour fermer la connexion au serveur, il suffit d'appeler OCI_ConnectionFree()

IV-C - Gestion des erreurs

OCILIB propose une gestion avancée des erreurs basée sur deux mécanismes distincts mais complémentaires :

- Erreurs générées par OCI
- Erreurs OCILIB internes (allocations mémoire, type-checking, etc....)
- Erreurs applicatives

Les mécanismes proposés par OCILIB sont :

- Gestion globale par callback
- Gestion contextuelle par thread

IV-C-1 - Gestion globale par callback

Lorsqu'une exception est déclenchée, OCILIB appelle la fonction de gestion des erreur fournie par l'application.
Le prototype de la fonction à fournir est le suivant :

```
typedef void (*POCI_ERROR) (OCI_Error *err);
```

Un handle sur un object OCI_Error est alors fourni à la fonction. La librairie fournit des fonctions pour accéder aux propriétés de l'erreur générée.

Exemple d'une fonction de gestion des erreurs qui se contente d'afficher à l'écran une erreur SQL :

```
#include "ocilib.h"

void err_handler(OCI_Error *err)
{
  printf("code  : ORA-%05i\n"
         "msg   : %s\n"
         "sql   : %s\n",
         OCI_ErrorGetOCICode(err),
         OCI_ErrorGetString(err),
         OCI_GetSql(OCI_ErrorGetStatement(err)));
}

int main()
{
  OCI_Connection *cn;

  if (!OCI_Initialize(err_handler, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

  cn = OCI_ConnectionCreate("wrong_db", "wrong_usr", "wrong_pwd",
                            OCI_SESSION_DEFAULT);

  /* ... application code here ... */

  OCI_Cleanup();

  return EXIT_SUCCESS;
}
```

IV-C-2 - Gestion contexuelle par thread

La gestion contextuelle par thread est activée par le flag OCI_ENV_CONTEXT passé à OCI_Initialize().
Ce mode permet de récupérer à tout moment la dernière erreur générée au sein du thread courant (qui peut être le thread principal de l'application).

Exemple d'une fonction de gestion des erreurs qui se contente d'afficher à l'écran une erreur SQL :

```
#include "ocilib.h"

int main()
{
  OCI_Connection *cn;

  if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT | OCI_ENV_CONTEXT))
    return EXIT_FAILURE;

  cn = OCI_ConnectionCreate("wrong_db", "wrong_usr", "wrong_pwd", OCI_SESSION_DEFAULT);

  if (cn == NULL)
  {
     OCI_Error *err = OCI_GetLastError();
```

Exemple d'une fonction de gestion des erreurs qui se contente d'afficher à l'écran une erreur SQL :

```
        printf("errcode %d, errmsg %s", OCI_ErrorGetOCICode(err),  OCI_ErrorGetString(err));
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

IV-D - Transactions

OCILIB supporte les mécanismes de transactions (locales et globales) proposées par Oracle via OCI.
Par défaut, une fois que l'application s'est connectée à Oracle, une transaction locale est créée.
Toute modification de données (insert, update, delete) n'est pas visible des autres sessions tant que l'on a pas explicitement validé les modifications par un appel à OCI_Commit().
Si l'application ne veut pas valider ses modifications et donc annuler les modifications de données effectuées depuis la dernière validation (ou par défaut depuis la connexion au serveur), elle doit utiliser OCI_Rollback().
Il y des cas ou les simples transactions locales par défaut ne suffisent pas :

- gérer une connexion en lecture seule par exemple
- gérer une transaction globale dans un environnement distribué
- etc,..

Dans ces cas là, OCILIB propose une gestion explicite des transactions où il est possible de créer une transaction et de l'associer à un handle de connexion.

IV-E - Types de donnée supportés

OCILIB supporte tous les types de données fournis par Oracle
Liste des types supportés :

- Tous les types scalaires (strings, numériques, flottants, ..) : CHAR/NCHAR, VARCHAR2/NVARCHAR2, NUMBER, FLOAT, ...
- Types binaires : RAW, LONG RAW, VARRAW, ..
- Larges Objects (Lobs et Files) : BLOB, CLOB, NCLOB, BFILE, CFILE
- Types LONGs : LONG, VAR LONG, LONG RAW, ...
- Date, Timestamps et Intervals : DATE, TIMESTAMP (tous), INTERVAL (tous)
- PL/SQL types : Ref cursors, PL/SQL Tables (tableaux) et PL/SQL Nested Tables
- Objects (Named Types) : types utilisateurs et types systèmes
- Références objects : Type REF
- SQL Collections : VARRAYs and Nested Tables
- ROWIDs et UROWIDs

Tous ces types supportés peuvent être liés à des statements ou être récupérés d'un select.

TYPE Oracle	Type OCILIB
Strings : CHAR/NCHAR, VARCHAR2/NVARCHAR2, ...	dtext * (char * or wchar_t * selon le build)
Numbers sans précision (entiers) : INT, NUMBER, ...	shorts, ints, big ints (signed/unsigned)
Flottants : FLOAT, REAL, NUMBER(X,Y), ...	double
RAW	void *
LONG, LONG RAW, VARRAW, ..	OCI_Long
BLOB / CLOB / NCLOB	OCI_Lob
DATE	OCI_Date
TIMESTAMP, TIMESTAMP_TZ, TIMESTAMP_LTZ	OCI_Timestamp
INTERVAL, INTERVAL_YM, INTERVAL_DS	OCI_Interval
PL/SQL Ref Cursors	OCI_Statement
PL/SQL Tables	Tableaux C de types scalaires ou objets OCILIB
PL/SQL Nested Tables	OCI_Statement
Objects (User Types)	OCI_Object
Références Objects (REF)	OCI_Ref
Collections (Varrays et Nested Tables)	OCI_Coll et OCI_Elem
ROWID et UROWID	dtext * (char * or wchar_t * selon le build)

V - Exécuter des ordres SQL

V-A - SQL Statements

Afin d'exécuter des ordres SQL sur la base de données, il faut créer un object SQL statement via la fonction OCI_CreateStatement().
Un fois cet objet crée, il est possible d'exécuter des requêtes par la fonction OCI_ExecuteStmt().

Exemple

```
OCI_Connection *cn;
OCI_Statement *st;

/* ... */

st = OCI_StatementCreate(cn);

OCI_ExecuteStmt(st, "delete from my_table where code is null");

printf("%d row deleted", OCI_GetAffectedRows(st));

OCI_Commit(cn);

OCI_StatementFree(st);

/* ... */
```

OCI_ExecuteStmt() a donc préparé et exécuté l'ordre SQL et OCI_GetAffectedRows() a récupéré le nombre de lignes supprimées dans la table.
Une fois que le statement n'est plus utile, un appel à OCI_StatementFree() libère toutes les ressources associées (resultset, ...).
Une object OCI_Statement peut être réutilisé pour exécuter autant d'ordres SQL que nécessaire.

V-B - Lier des variables

Dans l'exemple précédent, la requête était simple et ne nécessitait aucune variable d'entrée. Souvent, il est nécessaire de fournir à une requête des valeurs d'entrées non connues à l'avance. De plus, il est utile de pouvoir exécuter plusieurs fois un même ordre SQL avec des valeurs différentes sans pour autant avoir à faire repréparer le SQL par Oracle afin d'optimiser les performances.
Il est donc possible de lier des variables du programme en fournissant leur adresse au statement.
Dans ce cas, il faut :

- Préparer le SQL par OCI_Prepare()
- Lier les variables avec les fonctions OCI_bindXXX() ou XXX est le type de donnée
- Exécuter le SQL par OCI_Execute()

Exemple

```
OCI_Connection *cn;
OCI_Statement *st;
int code;

/* ... */

st = OCI_StatementCreate(cn);

OCI_Prepare(st, "delete from test_fetch where code = :code");
OCI_BindInt(st, ":code", &code);

code = 3;
OCI_Execute(st);
printf("%d row deleted"; OCI_GetAffectedRows(st));

code = 56;
OCI_Execute(st);
printf("%d row deleted"; OCI_GetAffectedRows(st));

OCI_Commit(cn);

/* ... */
```

Pour faire une insertion de masse, on peut procéder de la manière suivante :

Exemple

```
OCI_Connection *cn;
OCI_Statement *st;
int code;
char name[30];
char value[20];

/* ... */

st = OCI_StatementCreate(cn);

OCI_Prepare(st, "insert into my_table values(:code, :name, :value)");
OCI_BindInt(st, ":code", &code);
OCI_BindString(st, ":name", name, 30);
OCI_BindString(st, ":value", value, 20);

for (code = 1; code < 1000; code++);
{
    sprintf(name, "name %i", code);
    sprintf(value, "value %i", code);

    OCI_Execute(st);
```

Exemple

```
    }

    OCI_Commit(cn);

    /* ... */
```

Néanmoins, Oracle supporte les "bulks operations" (cf. chapitre VI) qui permettent de manipuler des tableaux de variables et donnent des performances incomparables.

V-C - Récupérer le résultat d'un select

Récupérer le résultat d'une requête est très simple. Il suffit une fois le statement exécuté via OCI_Execute() ou OCI_ExecuteStmt() d'appeler la fonction OCI_GetResultSet() qui retourne un handle sur un objet OCI_Resultset. Ce resultset comprend 2 choses :

- La description des colonnes retournées par la requête
- Les données (lignes de résultats)

Afin d'accéder aux valeurs de chaque colonne du resultset, il faut:

- Parcourir le resultset par un appel à OCI_FetchNext() qui retourne TRUE tant qu'il y a des lignes de résultat
- Appeler une des fonctions OCI_GetXXX() en précisant l'index de la colonne (ou son nom) et où XXX est le type : Int, String, Date, ...

Exemple

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement  *st;
    OCI_Resultset  *rs;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn  = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st  = OCI_StatementCreate(cn);

    OCI_ExecuteStmt(st, "select * from products");

    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
        printf("code: %i, name %s\n", OCI_GetInt(rs, 1)  , OCI_GetString(rs, 2));

    printf("\n%d row(s) fetched\n", OCI_GetRowCount(rs));

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

OCI_GetRowCount() donne le nombre de lignes du resultset déjà parcourues.
Le même exemple avec un accès aux colonnes par leur nom :

Exemple

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement  *st;
    OCI_Resultset  *rs;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn  = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st  = OCI_StatementCreate(cn);

    OCI_ExecuteStmt(st, "select * from products");

    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
        printf("code: %i, name %s\n", OCI_GetInt2(rs, "CODE")  , OCI_GetString2(rs, "NAME"));

    printf("\n%d row(s) fetched\n", OCI_GetRowCount(rs));

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

Un objet statement peut être réutilisé autant de fois que voulu. De plus, il est possible d'en créer plusieurs et de les imbriquer.

Exemple

```
    OCI_Connection *cn;
    OCI_Statement *st1, *st2;
    OCI_Resultset *rs;

    /* ... */

    st1 = OCI_StatementCreate(cn);
    st2 = OCI_StatementCreate(cn);

    int code;

    OCI_ExecuteStmt(st1, "select code from my_table");
    OCI_Prepare(st2, "delete from my_table2 where code_ref = :code");
    OCI_Bind(st2, ":code", &code);

    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
    {
        code = OCI_GetInt(rs, 1);

        OCI_Execute(st2);
    }

    /* ... */
```

Il est possible de récupérer les informations relatives (metadatas) à chaque colonne du resulset (colonnes sélectionnées par la requête). Pour cela :

- Un appel à OCI_GetColumnCount() permet de savoir le nombre de colonnes contenues dans le resultset
- Un appel à OCI_GetColumn() en précisant l'index (position dans le select) permet de récupérer un handle sur un objet OCI_Column
- Toute les propriétés de la colonne sont accessibles par une série de fonctions du type OCI_GetColumnXXX() où XXX est la propriété

Exemple

```
#include "ocilib.h"

int main()
{
  OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Resultset *rs;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    OCI_ExecuteStmt(st, "select * from my_table");

    rs = OCI_GetResultset(st);
    nb = OCI_GetColumnCount(rs);

    for(i = 1; i <= nb; i++)
    {
        col = OCI_GetColumn(rs, i);

        printf("%-20s%-10s%-8i%-8i%-8i%-s\n",
                OCI_GetColumnName(col),
                OCI_GetColumnSQLType(col),
                OCI_GetColumnSize(col),
                OCI_GetColumnPrecision(col),
                OCI_GetColumnScale(col),
                OCI_GetColumnNullable(col) == TRUE ? "Y" : "N");
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

V-D - Curseurs bidirectionnels

OCILIB supporte les "scrollables cursors" introduits avec Oracle 9i. Ces curseurs (resultsets au sens OCILIB) peuvent être parcourus :

- Séquentiellement dans les deux sens (en arrière et en avant)
- Il est possible de directement se placer sur le premier ou dernier enregistrement
- Il est possible de directement se placer sur un enregistrement donné

ExempleL

```
 #include "ocilib.h"

int main()
```

ExempleL

```
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Resultset *rs;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
       return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);

    st = OCI_StatementCreate(cn);

    OCI_SetFetchMode(st, OCI_SFM_SCROLLABLE);

    OCI_ExecuteStmt(st, "select int_col1, str_col from my_table");

    rs = OCI_GetResultset(st);

    /* get resultset row count */
    OCI_FetchLast(rs);
    printf("resultset contains %i rows\n", OCI_GetRowCount(rs));

    /* go to row 1 */
    OCI_FetchFirst(rs);
    printf("%i - %s\n", OCI_GetInt(rs, 1), OCI_GetString(rs, 2));

    /* enumerate from row 2 to row X */
    while (OCI_FetchNext(rs))
        printf("%i - %s\n", OCI_GetInt(rs, 1), OCI_GetString(rs, 2));

    /* enumerate from row X back to row 1 */
    while (OCI_FetchPrev(rs))
        printf("%i - %s\n", OCI_GetInt(rs, 1), OCI_GetString(rs, 2));

    /* print the 30th row */
    OCI_FetchSeek(rs, OCI_SFD_ABSOLUTE,  30);
    printf("%i - %s\n", OCI_GetCurrentRow(rs), OCI_GetInt(rs, 1), OCI_GetString(rs, 2));

    /* fetch next 30 rows */
    while ((OCI_GetCurrentRow(rs) < 60) << OCI_FetchNext(rs))
        printf("%0i - %s\n", OCI_GetInt(rs, 1), OCI_GetString(rs, 2));

    /* move back to the 45th row */
    OCI_FetchSeek(rs, OCI_SFD_RELATIVE,  -15);
    printf("%i - %s\n", OCI_GetInt(rs, 1), OCI_GetString(rs, 2));

   OCI_Cleanup();

   return EXIT_SUCCESS;
}
```

V-E - PL/SQL blocks

Des variables peuvent être liées en entrée comme en sortie.
Pour exécuter du PL/SQL, il suffit d'englober le code PL/SQL par un "begin" en début et un "end;" en fin de block.

Exemple d'un bloc PL/SQL

```
    OCI_Connection *cn;
    OCI_Statement *st;

    /* ... */

    st = OCI_StatementCreate(cn);
```

Exemple d'un bloc PL/SQL

```
OCI_Prepare(st, "begin :n := trunc(sysdate+1)-trunc(sysdate-1); end;");
OCI_BindInt(st, ":n", &n);
OCI_Execute(st);

printf("Result : %i\n", n);

/* ... */
```

Exemple de récupération d'un cursor PL/SQL et fetch de son résultat dans le programme C :

Exemple

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement  *st1; *st2;
    OCI_Resultset *rs;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", pwd", OCI_SESSION_DEFAULT);

    st1 = OCI_StatementCreate(cn);
    st2 = OCI_StatementCreate(cn);

    OCI_Prepare(st1, "begin open :c for select * from my_table; end;");
    OCI_BindStatement(st2, ":c", st2);
    OCI_Execute(st1);

    rs = OCI_GetResultset(st2);

    while (OCI_FetchNext(rs))
    {
        printf("value %s\n", OCI_GetString(rs, 1));
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

V-F - Support de la clause SQL RETURNING

Oracle propose une fonctionnalité très intéressante qui est la clause SQL "returning into".
L'utilisation de cette clause dans un DML (insert/update/delete) permet de combiner 2 requêtes en 1, réduisant ainsi le nombre d'allers-retours entre le client et le serveur.
L'ajout de cette clause permet de sélectionner les lignes affectées par le DML au sein de la même requête. Ce qui est pratique, notamment dans le cas de delete car les valeurs supprimées sont ainsi récupérables !
OCILIB implémente cette fonctionnalité en créant un objet OCI_Resultset à partir des colonnes sélectionnées dans la clause RETURNING. Ce resultset est ensuite manipulable comme un resultset classique.
L'Array Interface est également compatible avec cette clause. Dans ce cas, chaque entrée du tableau peut affecter différentes lignes. OCILIB crée alors un resultset pour chaque entrée du tableau.
Par exemple, si le tableau utilisé pour lier des variables au statement contient 100 éléments, l'application commence par récupérer le premier resulset par un appel à OCI_GetResulset(). Une fois ce resultset parcouru, le resultset suivant est récupérable par un appel à OCI_GetNextResultset().
OCILIB ne pré-parse jamais les requêtes SQL (pour des raisons de performances) et donc ne peut déterminer les champs (ainsi que leur type) utilisés dans la clause RETURNING.

Il faut donc que l'application indique explicitement les colonnes de la clause RETURNING par les fonctions OCI_RegisterXXX() ou XXX est le type de données. OCILIB construit alors les resultsets sur la base des colonnes ainsi déclarées.

Exemple de sélection des objets d'une table :

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement  *st;
    OCI_Resultset  *rs;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_CreateStatement(cn);

    OCI_Prepare(st, "update products set code = code+10 returning code into :i");
    OCI_RegisterInt(st, ":i");
    OCI_Execute(st);

    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
        printf("%i\n", OCI_GetInt(rs, 1));

    printf("count : %i\n", OCI_GetRowCount(rs));

    OCI_Commit(cn);

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

V-G - Contrôler les statements

OCILIB propose une série de fonctions permettant de personnaliser les comportements d'un statement.
Par exemple, il est possible de spécifier :

- Le nombre de lignes pré-fetchées par le client Oracle afin de diminuer le nombre de roundtrips avec le serveur
- Le nombre de lignes fetchées de manière interne par OCILIB afin de réduire les appels OCI
- Le mode de liaison des variables : par position ou par nom
- Le format de date par défaut
- Etc, ..

VI - Bulk Operations

Les "bulk operations" sont une fonctionnalité très puissante supportée par Oracle qui permet d'effectuer des mises à jour massives (insert/update/update) de tables dans des temps records.
Dans un schéma classique, si l'on souhaite insérer 1000 lignes dans une table par exemple, il est possible de :

- préparer la requête
- lier les variables
- boucler de 1 à 1000 pour exécuter le SQL en ayant à chaque tour de boucle mis à jour les valeurs.

Ce schéma va générer 1000 appels d'exécution de la requête et donc 1000 allers / retours avec le serveur pour :

- Transmettre les valeurs
- Exécuter de nouveau le SQL

Tous ces appels et allers/retours réseau sont couteux en termes de performances ! Si l'on souhaite insérer 1 millions de lignes, cela peut être long !
Oracle propose donc une solution simple qui permet d'obtenir des performances optimales : OCI Interface Array

VI-A - Oracle Array interface

Dans l'exemple précédent, le trafic réseau généré par un nombre important d'exécution d'une même requête est la source principale de la lenteur finale du traitement.
Oracle propose de minimiser ce trafic en lui fournissant des tableaux de variables. Au lieu de fournir 1 variable et envoyer 1000 fois sa valeur au serveur, on fournit un tableau de 1000 valeurs qui est envoyé en une seule fois.
Ce système permet de réduire le temps de traitement par des facteurs de 3 à 5 chiffres au minimum sur des gros volumes.
OCILIB supporte ce mécanisme et donne la possibilité de lier des tableaux de n'importe quel type supporté par OCILIB (excepté OCI_Long et OCI_Statement)

VI-B - Exemple

Exemple pour insérer d'un seul coup 1000 lignes dans une table :

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;

    int tab_int[1000];
    char tab_str[1000][21];

    int i;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    /* ... create connection and statement ... */

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    /* binding */

    OCI_Prepare(st, "insert into products values(:i, :s)");
    OCI_BindArraySetSize(st, 1000);
    OCI_BindArrayOfInts(st, ":i", (int*) tab_int, 0);
    OCI_BindArrayOfStrings(st, ":s", (char*) tab_str, 20, 0);

    /* filling arrays */

    for(i = 0; i < 1000 ; i++)
    {
        tab_int[i] = i+1;
        sprintf(tab_str[i],"Name %d",i+1);
    }

    /* execute */

    OCI_Execute(st);
```

Exemple pour insérer d'un seul coup 1000 lignes dans une table :

```
    printf("row processed : %d\n", OCI_GetAffectedRows(st));

    OCI_Commit(cn);

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

VI - Utiliser les Larges Objects

Oracle 8 a introduit les LOBs (Larges Objects) afin de pouvoir stocker et manipuler des objets de grosse taille (> plusieurs Go) afin de répondre aux besoins de stockage de gros fichiers (multimédia, vidéo, ...) et de données brutes au sein de la DB.
Il existe fondamentalement 2 types de large objects :

- Les BLOB/CLOBS : objets stockés sous forme binaire (BLOBS) / textuelle (CLOBS)
- Les BFILES : objets non stockés au sein de la DB. Il s'agit de pointeurs sur des fichiers accessibles par le serveur (sur le même système de fichier ou non) et qui sont manipulables par l'application cliente

OCILIB supporte intégralement ces types de données.
De plus, OCILIB supporte également l'ancienne implémentation de LOBs : les types LONG, RAW et LONG RAW. Ces types, toujours maintenus par Oracle, sont les "ancêtres" des LOBs et représentaient dans les versions antérieures (jusqu'à Oracle 7) l'unique moyen de stocker des gros volumes (mais plus limités) dans la DB.
Les types LONGs sont toujours actuellement très employés dans les bases et applications Oracle.
C'est pourquoi OCILIB propose une API très proche de celles des LOBs (alors que la sous couche OCI de gestion des LONGs est très différente de celle des LOBs).

VI-A - Internal Objects (LOBs)

Les BLOB/CLOB sont stockés dans la base de données. Leur utilisation diffère de celles des autres types de données. En effet, une simple requête SQL ne suffit pas à lire ou écrire dans un champ de plusieurs centaines de Mo.
OCILIB propose tout une série de fonctions permettant de manipuler très simplement ces objets (OCI_Lob *).

Par exemple, pour insérer le contenu d'un fichier dans un champ de type BLOB d'une table :

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Lob *lob;
    FILE *f;
    unsigned char buffer[1024];
    int size;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    f  = fopen("My file", "rb");

    if (f != NULL)
    {
        fseek (f , 0 , SEEK_END);
        size = ftell(f);
        rewind (f);
```

Par exemple, pour insérer le contenu d'un fichier dans un champ de type BLOB d'une table :

```
        printf("\nTotal bytes to write : %d\n", size);

        lob = OCI_LobCreate(cn, OCI_BLOB);

        OCI_Prepare(st, "insert into my_blob_table(code, content) values (1, :data)");
        OCI_BindLob(st, ":data", lob);
        OCI_Execute(st);

        /* write data into table by chunks of 1024 bytes */
        while ((n = fread(buffer, 1, sizeof(buffer), f)))
        {
            OCI_LobWrite(lg, buffer, n);
        }

        printf("\nTotal bytes written : %s\n", OCI_LobGetLenght(lob));
        fclose(f);

        OCI_Commit(cn);

        OCI_LobFree(lob);
    }

 OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

Par exemple, pour lire le contenu d'un fichier texte ANSI stocké dans un champ de type CLOB d'une table :

Exemple

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Lob *lob1, *lob2;

    int code, n;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    OCI_ExecuteStmt(st, "select code, content from test_lob for update");

    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
    {
        code = OCI_GetInt(rs, 1);
        lob1 = OCI_GetLob(rs, 2);
        lob2 = OCI_LobCreate(cn, OCI_CLOB);

        n = OCI_LobWrite(lob1, "Today, ", 7);
        OCI_LobSeek(lob1, n, OCI_SEEK_SET);

        n = OCI_LobWrite(lob2, "I'm going to the cinema !", 25);

        OCI_LobAppendLob(lob1, lob2);
        OCI_LobSeek(lob1, 0, OCI_SEEK_SET);

        n = OCI_LobRead(lob1, temp, 100);
```

Exemple

```
        temp[n] = 0;

        printf("code: %i, action : %s\n", code, temp);

        OCI_LobFree(lob2);
    }

    printf("\n%d row(s) fetched\n", OCI_GetRowCount(rs));

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

VI-B - External Objects (FILEs)

Les objets de type FILE (OCI_File *) sont des références à des fichiers externes à la DB qui peuvent être ouverts pour lecture par le serveur et dont le contenu peut être récupéré par l'application cliente.
Les objets FILEs sont similaires aux LOBs et donc leur API est similaire sauf que les FILEs sont en lecture seule.

Exemple de lecture d'un champ BILE d'un table Oracle :

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_File *file;
    char buffer[256];
    int n;

    if (!OCI_Initialize(err_handler, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    file = OCI_FileCreate(cn, OCI_CFILE);

    OCI_FileSetName(file, "MYDIR", "MyfileName");

    /* check if faile exists */

    if (OCI_FileExists(file))
    {
         printf("file size : %d\n", OCI_FileGetSize(file));
         printf("file dir  : %s\n", OCI_FileGetDirectory(file));
         printf("file name : %s\n", OCI_FileGetName(file));
    }

    /* bind for inserting into table */

    OCI_Prepare(st, "insert into my_bfile_table(code, value) values (1, :bfile)");
    OCI_BindFile(st, ":bfile", file);
    OCI_Execute(st);
    OCI_Commit(cn);

    /* free local file object */

    OCI_FileFree(file),

    /* fetch bfile data from table */

    OCI_ExecuteStmt(st, "select code, value from my_bfile_table");
```

Exemple de lecture d'un champ BILE d'un table Oracle :

```
    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
    {
        file = OCI_GetFile(rs, 2);

        OCI_FileOpen(file);

        printf("file size  %d\n", OCI_FileGetSize(file));
        printf("file dir   %s\n", OCI_FileGetDirectory(file));
        printf("file name  %s\n", OCI_FileGetName(file));

        while (n = OCI_FileRead(file, buffer, sizeof(buffer)-1))
        {
            buffer[n] = 0;
            printf(buffer);
        }

        OCI_FileClose(file);
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

VI-C - Long Objects

OCILIB implémente les "anciens" LONGs (OCI_Long *) de manière similaire aux LOBs et FILEs. Les APIs de manipulation des champs LONGs sont donc très proches de celles des Larges Objects.

Exemples pour stocker un fichier binaire en base et ensuite récupérer le contenu :

```
 #include "ocilib.h"

#define FILENAME "data.lst"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Long *lg;
    FILE *f;
    char buffer[2048];
    int n;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

   /* INSERT LONGS  ------------------------------------------ */

     /* open the app file */

    f = fopen(FILENAME, "rb");

    if (f)
    {
        fseek (f , 0 , SEEK_END);
        n = ftell(f);
        rewind (f);

        printf("\n%d bytes to write\n", n);

        lg = OCI_LongCreate(st, OCI_BLONG);
```

Exemples pour stocker un fichier binaire en base et ensuite récupérer le contenu :

```
        OCI_Prepare(st,  "insert into test_long_raw(code, content) values (1, :data)");
        OCI_BindLong(st, ":data", lg, n);
        OCI_Execute(st);

        /* write data into table by chunks of 2048 bytes */
        while ((n = fread(buffer, 1, sizeof(buffer), f)))
        {
            OCI_LongWrite(lg, buffer, n);
        }

        printf("\n%d bytes written\n", OCI_LongGetSize(lg));
        fclose(f);

        OCI_LongFree(lg);
        OCI_Commit(cn);
    }

    /* FETCHING LONGS  ---------------------------------------- */


    OCI_ExecuteStmt(st, "select content from test_long_raw where code = 1");
    OCI_SetLongMaxSize(st, 1000000);

    rs = OCI_GetResultset(st);

    /* read data by chunks of 2048 bytes*/
    while (OCI_FetchNext(rs))
    {
        lg = OCI_GetLong(rs, 1);

        while ((n = OCI_LongRead(lg, buffer, sizeof(buffer)))) {}

        printf("\n%d bytes read\n", OCI_LongGetSize(lg));
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

VIII - Manipuler les Named Types

OCILIB supporte les "Named types", c'est à dire des types de données nommés crées par l'utilisateur ou fournis par Oracle. Un "TYPE" oracle peut vu comme l'équivalent d'une structure C. Il s'agit d'une agrégation de champs de types scalaires ou agrégés.

Exemple de création d'un type de données :

```
create type t_produit as object
(
    code       number,
    libelle    varchar2(30),
    prix       number(5,3),
    reference  varchar2(100)
);

create table ventes_produits
(
  product    t_produit,
  nombre     int,
  date_vente date
);
```

OCILIB permet de :

- créer des instances d'objet
- de lier des objets à requêtes SQL
- sélectionner les objets d'une table
- manipuler les attributs d'un objet

VIII-A - Instances autonomes

Afin de créer une instance autonome, il faut récupérer le descripteur du type concerné. Cela se réalise par un appel à OCI_TypeInfoGet().

Exemple de création d'une instance d'un objet pour une insertion dans une table :

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Resultset *rs;
    OCI_Object *prod;
    OCI_Date *date;
    int qte;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    prod = OCI_ObjectCreate(cn, OCI_TypeInfoGet(cn, "t_product", OCI_TIF_TYPE));
    date = OCI_DateCreate(cn);

    qte = 356;
    OCI_DateSysDate(date);

    OCI_ObjectSetInt(prod, "CODE", 1);
    OCI_ObjectSetString(prod, "LIBELLE", "USB KEY 2go");
    OCI_ObjectSetDouble(prod, "PRIX", 12.99);
    OCI_ObjectSetString(prod, "REFERENCE", "A56547WSAA");

    OCI_Prepare(st, "insert into ventes_produits values(:p, :n, :d)");

    OCI_BindObject(st, ":p", prod);
    OCI_BindInt(st, ":n", &qte);
    OCI_BindDate(st, ":d", date);

    OCI_Execute(st);

    printf("\n%d row(s) inserted\n", OCI_GetAffectedRows(st));

    OCI_Commit(cn);

    OCI_ObjectFree(obj);

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

VIII-B - Sélection d'objets

Les objets peuvent être sélectionnés de la même façon que les types de donnée SQL.

Exemple de sélection des objets d'une table :

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Resultset *rs;
    OCI_Object *prod;
    OCI_Date *date;
    int qte;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    OCI_ExecuteStmt(st, "select * from ventes_produits");

    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
    {
        prod = OCI_GetObject(rs, 1);
        qte  = OCI_GeInt(rs, 2);
        date = OCI_GetDate(rs, 3);

        printf("Produit : %s, Ventes : %i\n", OCI_ObjectGetString(prod, "LIBELLE"), qte);
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

IX - Chargement de données en mode Direct Path

IX-A - Oracle Direct Path Loading

Oracle propose une API permettant de charger des données directement dans les tables sans passer par le moteur SQL.
Cette fonctionnalité, disponible dans l'API OCI, est implémentée dans l'outils SQL*Loader (mode direct). Les données sont généralement issues de fichiers textes.
Ce mode de chargement est extremement rapide car Les données sont directement converties au format de stockage dans les tables.
OCILIB implémente cette fonctionnalité à l'exception des types de données suivants : objets, refs et string functions.
Pour plus de détails, se référer à la documentation d'OCILIB.

IX-B - Exemple

Exemple pour charger des données dans une table :

```
#include "ocilib.h"

#define SIZE_ARRAY  1000
#define SIZE_COL1   20
#define SIZE_COL2   30
#define SIZE_COL3   8
#define NUM_COLS    3
```

Exemple pour charger des données dans une table :

```
int main(void)
{
    OCI_Connection *cn;
    OCI_DirPath    *dp;
    OCI_TypeInfo   *tbl;

    dtext val1[SIZE_COL1+1];
    dtext val2[SIZE_COL2+1];
    dtext val3[SIZE_COL3+1];

    int i = 0, nb_rows = SIZE_ARRAY;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
       return EXIT_FAILURE;

    cn  = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    tbl = OCI_TypeInfoGet(cn, "test_directpath", OCI_TIF_TABLE);
    dp  = OCI_DirPathCreate(tbl, NULL, NUM_COLS, nb_rows);

    /* optional attributes to set */

    OCI_DirPathSetBufferSize(dp, 64000);
    OCI_DirPathSetNoLog(dp, TRUE);
    OCI_DirPathSetParallel(dp, TRUE);

    /* describe the target table */

    OCI_DirPathSetColumn(dp, 1, "VAL_INT",  SIZE_COL1, NULL);
    OCI_DirPathSetColumn(dp, 2, "VAL_STR",  SIZE_COL2, NULL);
    OCI_DirPathSetColumn(dp, 3, "VAL_DATE", SIZE_COL3, "YYYYMMDD");

    /* prepare the load */

    OCI_DirPathPrepare(dp);

    nb_rows = OCI_DirPathGetMaxRows(dp);

    for (i = 1; i <= nb_rows; i++)
    {
        /* fill test values */

        sprint_dt(val1, SIZE_COL1+1, "%04d", i);
        sprint_dt(val2, SIZE_COL2+1, "value %05d", i);
        sprint_dt(val3, SIZE_COL3+1, "%04d%02d%02d", (i%23)+1 + 2000,
                                                     (i%11)+1,
                                                     (i%23)+1);

        OCI_DirPathSetEntry(dp, i, 1, val1, (unsigned int) strlen(val1), TRUE);
        OCI_DirPathSetEntry(dp, i, 2, val2, (unsigned int) strlen(val2), TRUE);
        OCI_DirPathSetEntry(dp, i, 3, val3, (unsigned int) strlen(val3), TRUE);
    }

    /* load data to the server */

    OCI_DirPathConvert(dp);
    OCI_DirPathLoad(dp);

    /* commits changes */

    OCI_DirPathFinish(dp);

    printf("%04d row(s) processed\n", OCI_DirPathGetAffectedRows(dp));
    printf("%04d row(s) loaded\n",  OCI_DirPathGetRowCount(dp));

    /* free direct path object */

    OCI_DirPathFree(dp);

    OCI_Cleanup();

   return EXIT_SUCCESS;
```

Exemple pour charger des données dans une table :

```
}
```

X - Manipulation des dates et timestamps

IX-A - Dates

OCILIB permet de manipuler les dates Oracle (date/heure) avec facilité.
Exemple où une date est récupérée d'une table, puis affichée, incrémentée de 5 jours et 2 mois et enfin réaffichée :

Exemple

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Date *dt;
    char str[100];
    int n;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    OCI_ExecuteStmt(st, "select mon_champs_date from my_table");
    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
    {
        dt = OCI_GetDate(rs, 1);

        OCI_DateToText(dt, "DD/MM/YYYY HH24:MI:SS", sizeof(str)-1, str);
        printf("Date is %s\n", str);

        OCI_DateAddDays(dt, 5);
        OCI_DateAddMonths(dt, 2);

        OCI_DateToText(dt, "DD/MM/YYYY HH24:MI:SS", sizeof(str)-1, str);
        printf("Date + 5 days and 2 months is %s\n", str);
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

IX-B - Timestamps

Les timestamps sont des extensions au type DATE permettant :

- de gérer avec précision les dates grâce à la gestion des fractions de secondes
- de gérer les fuseaux horaires

Exemple où la date/heure courante est récupérée du serveur et affichée avec les fractions de secondes :

Exemple

Exemple

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement *st;
    OCI_Timestamp *tm;
    char str[100];
    int n;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    OCI_ExecuteStmt(st, "SELECT CURRENT_TIMESTAMP FROM dual");
    rs = OCI_GetResultset(st);

    OCI_FetchNext(rs);
    tm = OCI_GetTimestamp(rs, 1);

    OCI_TimestampToText(tm, "DD/MM/YYYY HH24:MI:SS:FF3\n", 100, str, 3);
    printf(str);

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

IX-C - Intervals

Les intervals sont des périodes de temps exprimées :

- soit en années et mois
- soit en jours, heures, minutes et secondes

Ils permettent d'effectuer des opérations sur les timestamps.
Exemple où la date/heure courante locale est récupérée dans la variable tm, puis incrémentée de 1 jour, 1 heure, 1 minute et 1 seconde puis affichée avec les fractions de secondes :

Exemple

```
#include "ocilib.h"

#define SIZE_STR 260

int main()
{
    OCI_Timestamp *tm;
    OCI_Interval *itv;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    tm  = OCI_TimestampCreate(NULL, OCI_TIMESTAMP);
    itv = OCI_IntervalCreate(NULL, OCI_INTERVAL_DS);

    OCI_TimestampSysTimeStamp(tm);
    OCI_TimestampToText(tm, "DD/MM/YYYY HH24:MI:SS:FF3", SIZE_STR, str, 3);
    printf("%s\n", str);

    OCI_IntervalSetDaySecond(itv, 1,1,1,1,0);
    OCI_IntervalToText(itv, 10, 10, SIZE_STR, str);
```

Exemple

```
    printf("%s\n", str);

    OCI_TimestampIntervalAdd(tm, itv);
    OCI_TimestampToText(tm, "DD/MM/YYYY HH24:MI:SS:FF3", SIZE_STR, str, 3);
    printf("%s\n", str);

    OCI_TimestampFree(tm);
    OCI_IntervalFree(itv);

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

XI - Pour aller plus loin

X-A - Support d'Unicode

OCILIB supporte totalement Unicode sur toute les plateformes à travers le type de donnée C wchar_t (UTF16/UTF32 selon les plateformes).
OCILIB, pour les builds Unicode, initialise OCI en mode UTF16, qui est implémenté par Oracle par un encodage à taille fixe sur 2 bytes.
Donc, sur les systèmes implémentant wchar_t sur 2 bytes (comme MS Windows), les chaines de caractères et buffers utilisateurs sont directement passés à Oracle.
Sur les autres systèmes (quasiment tous les unix et linux), wchar_t est implémenté sur 4 bytes et UTF32 est usuellement utilisé comme encodage.
Dans ce cas, OCILIB :

- utilise des buffers temporaires pour passer à OCI les chaînes de caractères de meta-données (requêtes, attributs, ...)
- alloue ses buffers sur la taille de wchar_t et exécute une expansion de UTF16 à UTF32 pour les buffers de données

L'expansion de données est faite "inplace", ce qui a l'avantage de ne pas nécessiter de buffer supplémentaire. Cela permet d'optimiser au mieux la gestion de la taille variable du type wchar_t.

X-B - Décrire les tables d'un schéma

OCILIB permet de décrire les tables d'un schéma. Par exemple :

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_TypeInfo *tbl;
    int n;

    if (!OCI_Initialize(NULL, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn  = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    tbl = OCI_TypeInfoGet(cn, "MyTable", OCI_TIF_TABLE);

    if (tbl)
    {
        printf ("Column Name Type Length Prec. Scale Null ?\n");
        printf ("--------------------------- ------------------------------\n");
```

OCILIB permet de décrire les tables d'un schéma. Par exemple :

```
        n = OCI_TypeInfoGetColumnCount(tbl);

        for(i = 1; i <= n; i++)
        {
            col = OCI_TypeInfoGetColumntbl, i);

            printf("%-20s%-10s%-8i%-8i%-8i%-s\n",
                    OCI_ColumnGetName(col),
                    OCI_ColumnGetSQLType(col),
                    OCI_ColumnGetSize(col),
                    OCI_ColumnGetPrecision(col),
                    OCI_ColumnGetScale(col),
                    OCI_ColumnGetNullable(col) == TRUE ? "Y" : "N");
            }
        }
    }

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

X-C - Fonctions "'All In One"

OCILIB propose des fonctions "all in one" qui ont une liste d'arguments variables.
Elles permettent de réduire le nombre de lignes de code au minimum possible !
Par exemple, OCILIB permet d'écrire en 1 seule ligne de code les actions suivantes :

- Préparation du SQL
- Liaison d'un variable
- Exécution de la requête
- Fetch du résultat (1 ligne)

Ce qui équivaut au code suivant :

- OCI_Prepare(st, "select article from test_fetch where code = :i");
- OCI_Bind(st, ":code", &code);
- OCI_Execute(st);
- OCI_GetResultset(st);
- OCI_FetchNext(rs);
- OCI_GetString(rs, 1);

Qui peut être remplacé par le code suivant :

- OCI_ImmediateFmt(cn, "select article from test_fetch where code = %i", code, OCI_ARG_TEXT, name);

D'autres exemples :

Exemples

```
#include "ocilib.h"

int main()
{
    OCI_Connection *cn;
    OCI_Statement  *st;
    OCI_Resultset  *rs;

    int code = 1;
```

Exemples

```
    char name[50];

    if (!OCI_Initialize(err_handler, NULL, OCI_ENV_DEFAULT))
        return EXIT_FAILURE;

    cn = OCI_ConnectionCreate("db", "usr", "pwd", OCI_SESSION_DEFAULT);
    st = OCI_StatementCreate(cn);

    /* sql format with params ------------------------------------------- */

    OCI_ExecuteStmtFmt(st, "select article from test_fetch where code = %i", code);

    rs = OCI_GetResultset(st);

    while (OCI_FetchNext(rs))
        printf("article : %s\n", OCI_GetString(rs, 1));

    /* sql immediate (parse, exec, one fetch) ------------------------------ */

    OCI_Immediate(cn, "select code, article from test_fetch where code = 1",
                  OCI_ARG_INT, &code, OCI_ARG_TEXT, name);

    printf("article : %s - code %i\n", name, code);

    /* sql immediate (parse, exec, one fetch) with params ------------------ */


    OCI_ImmediateFmt(cn, "select article from test_fetch where code = %i",
                         code, OCI_ARG_TEXT, name);

    printf("article : %s\n", name);

    OCI_Cleanup();

    return EXIT_SUCCESS;
}
```

X-D - Documentation

- **Consulter la documentation en ligne**
- La documentation est également incluse dans les distributions de OCILIB

XII - Optimiser les performances

=> Quelques conseils pour optimiser au maximum les performances :

- Eviter les fonction OCI_xxxFmt() car, bien que très pratiques, elles requièrent un parsing de l'ordre SQL et un formatage des variables qui peuvent être coûteux si les requêtes sont exécutées un grand nombre de fois
- Si un ordre SQL doit être répété plusieurs fois, il est préférable de le préparer une seule fois avec OCI_Prepare(), lier les variables avec OCI_Bindxxx() et ensuite l'exécuter autant de fois que voulu avec OCI_Exectute()
- Si possible, utiliser les tableaux pour lier les variables
- Si les valeurs affectées par un DML doivent être ensuite resélectionnées, utiliser la clause "Returning into"
- Ajuster le nombre de lignes par fetch avec OCI_SetFetchSize()
- Ajuster le nombre de lignes préfetchées par le serveur avec OCI_SetPrefetchSize()
- Préférer les Lobs aux LONG et LONG RAW
- Pour récupérer les valeurs d'un resultet, utiliser l'accès par index au lieu de l'accès par nom de colonnes afin d'éviter l'utilisation de maps index/nom

XIII - Téléchargement

- **Disponible sur le site de OCILIB sur developpez.com**
- **Disponible sur le site de OCILIB anglophone (Sourceforge.net)**

XIV - Fonctionnalités (version 3.2.0)

Voici une liste non exhaustive des principales fonctionnalités à la date de la dernière révision du tutoriel :

=> Librairie :

- Pur code ISO C (C90 et C99 pour le support Unicode)
- Portable sur toutes les plateformes supportées par Oracle
- Compatible avec Oracle 8i, 9i, 10g et 11g
- API simple et compacte
- Support Unicode complet
- Supporte le chargement dynamique des librairies Oracle (donc Oracle n'est pas requis pour compiler)
- Consommation mémoire réduite
- Gestion contextuelle des erreurs par thread

=> Type de données :

- Support de tous les types de données Oracle
- Tous les types scalaires (strings, numériques, flottants, ..) : CHAR/NCHAR, VARCHAR2/NVARCHAR2, NUMBER, FLOAT, ...
- Types binaires : RAW, LONG RAW, VARRAW, ..
- Larges Objects (Lobs et Files) : BLOB, CLOB, NCLOB, BFILE, CFILE
- Type LONG (piecewise operations) : LONG, VAR LONG, ...
- Date, Timestamps et Intervals : DATE, TIMESTAMP (tous), INTERVAL (tous)
- PL/SQL types : Ref cursors, Tables (tableaux) et Nested Tables
- Objects (Named Types) : types utilisateurs et types systèmes
- Références objects : Type REF
- SQL Collections : VARRAYs and Nested Tables
- ROWIDs et UROWIDs

=> Fonctionnalités :

- Support de toute l'API relationnelle fournie par OCI
- Support quasi complet de l'API objet fournie par OCI
- Près de 500 fonctions simples
- Liaisons de variables
- Bulk operations (array interface) : insert/update/delete avec liaison de tableaux de variables pour accélérer les performances
- Statements SQL réutilisables
- Support des curseurs bidirectionnels
- Pools de connexions
- Transactions globales
- Direct Path loading
- Support de la clause SQL-Oracle "Returning into"
- Gestion des erreurs
- Gestion des traces applicatives (+ DB au sein des vues systèmes)
- Description de schémas et d'objets
- Accès aux colonnes d'un select par position ou par nom
- Gestion simple des lectures / écritures dans les champs LONG / LOBS

- Support PL/SQL : blocs, ref cursors, tables, nested tables
- Collections et nested tables
- Support des LOBs > 4Go
- API portable de tables de hachage, threads et mutexes

XV - Conclusion

OCILIB permet de développer simplement et efficacement une application Oracle portable en C/C++.
L'interface de cette librairie a été conçue pour :

- Etre la plus simple possible
- Proposer la richesse d'OCI
- Réduire le nombre de lignes de code et d'appels de fonctions nécessaires
- Proposer des fonctions avec le moins de paramètres possible
- Respecter le principe d'encapsulation afin de pérenniser la librairie (c'est pourquoi aucune structure de donnée n'est publique par exemple)

OCILIB implémente actuellement la majorité des fonctionnnatilités OCI.
C'est une librairie open source et chacun peut s'il le souhaite faire évoluer la librairie pour ses propres besoins et en faire profiter les autres.
Ce tutoriel a survolé dans les grandes lignes la librairie OCILIB.
La documentation HTML apporte plus de détails et couvre des fonctionnalités non évoquées dans ce tutoriel.
La roadmap actuelle est la suivante :

- version 3.3.0 : Tuning des Lobs + gestion du cache de statement et Support XA
- version 3.4.0 : Support des Queues/Subscriptions

Contacter l'auteur : **Vincent Rogier (developpez.com)**, **vince_del_paris (sourceforge.net)**

XVI - Remerciements

Corrections initiales

- **RideKick**

Mise en forme

- **Franck.H** (pour ses barres magiques de niveaux de difficulté)