Manuel Utilisateur de GixSQL

SQL embarqué pour GnuCOBOL — édition SuperBOL

L'équipe SuperBOL de Titagone-OCamlPro

12 novembre 2026

Généré le 2026-11-18 à partir du commit superbol-manuals@a5e6f46

Table des matières

1 Préface 5
1.1 À qui s'adresse ce manuel 5
1.2 Organisation de ce manuel 5
1.3 Conventions 6
2 Introduction 7
2.1 Qu'est-ce que GixSQL ? 7
2.2 Composants 7
2.3 Processus de développement 8
2.4 Relation avec d'autres précompilateurs 8
2.5 Résumé des fonctionnalités 8
3 Installation 9
3.1 Prérequis 9
3.2 Compilation depuis le dépôt git 10
3.3 Options de configuration 10
3.4 Fichiers installés 11
3.5 Compilation avec Nix 11
3.6 Compilation sous Windows 12
3.6.1 Visual Studio 12
3.6.2 MSYS2/MinGW 12
3.7 Vérification de l'installation 12
4 Concepts du précompilateur 13
4.1 SQL embarqué dans un programme COBOL 13
4.2 La section de déclaration 13
4.3 Inclusion de fichiers 14
4.4 Précompilation, compilation et édition de liens 14
4.5 Un premier programme 14
4.6 Tables d'exemple 15
5 Connexions et transactions 17
5.1 Connexion à une base de données 17
5.2 Chaînes de connexion 17
5.2.1 Le pilote par défaut 18
5.2.2 Options de connexion 18
5.3 Connexions multiples 18
5.4 Déconnexion 19
5.5 Transactions 19

5.5.1 Autocommit 20
6 Variables hôtes et types de données 21
6.1 Variables hôtes 21
6.2 Conversion des types de données 21
6.3 Chaînes de caractères de longueur fixe 22
6.4 Chaînes de longueur variable : VARYING 22
6.5 SQL TYPE IS 23
6.6 EXEC SQL VAR 23
6.7 Variables indicatrices 23
6.8 Tables hôtes (tableaux) 24
7 Instructions SQL embarquées 25
7.1 Exécution d'instructions SQL 25
7.2 Requêtes mono-ligne : SELECT INTO 25
7.3 Curseurs 26
7.3.1 Curseurs modifiables 27
7.3.2 Curseurs sur des instructions préparées 27
7.4 Transactions 27
7.5 Autres instructions 28
8 Gestion des erreurs et diagnostics 29
8.1 La zone de communication SQL 29
8.2 SQLCODE et SQLSTATE 29
8.3 La condition « aucun enregistrement » 30
8.4 WHENEVER 30
8.5 Journalisation 31
9 SQL dynamique 32
9.1 Méthode 1 : EXECUTE IMMEDIATE 32
9.2 Méthode 2 : PREPARE et EXECUTE 32
9.3 Méthode 3 : requêtes préparées et curseurs 32
9.4 Marqueurs de paramètres 33
10 Le précompilateur gixpp 34
10.1 Synopsis 34
10.2 Options 34
10.2.1 Entrée et sortie 34
10.2.2 Format des sources 34
10.2.3 Traitement ESQL 35
10.2.4 Débogage 36
10.2.5 Information 36
10.3 Code généré 36
11 Configuration de l'exécution 37

11.1 Variables d'environnement 37
11.2 Trouver les pilotes 38
11.3 Journalisation 38
12 Migrer depuis Pro*COBOL 39
12.1 Réglages recommandés 39
12.2 Connexion 40
12.3 Exécution sur PostgreSQL 40
12.4 Comparaison des fonctionnalités 40
12.5 Options du précompilateur 42
13 Vue d'ensemble des backends 43
13.1 Fonctionnalités prises en charge 43
13.2 Choisir le style des marqueurs de paramètres 44
14 PostgreSQL 45
14.1 Prérequis 45
14.2 Chaîne de connexion 45
14.3 Options 45
14.4 Curseurs et transactions 46
14.5 Exécuter des programmes Oracle 46
14.6 Tests 46
15 Oracle 47
15.1 Prérequis 47
15.2 Chaîne de connexion 47
15.3 Options et environnement 47
15.4 Transactions 47
15.5 Types de données 47
15.6 Tests 48
16 MySQL et MariaDB 49
16.1 Prérequis 49
16.2 Chaîne de connexion 49
16.3 Options 49
16.4 Marqueurs de paramètres 49
17 ODBC 50
17.1 Prérequis 50
17.2 Chaîne de connexion 50
17.3 Comportement 50
18 SQLite 51
18.1 Prérequis 51
18.2 Chaîne de connexion 51
18.3 Options 51

18.4 Comportement 51
19 Récapitulatif des instructions SQL embarquées 52
20 Codes d'erreur 54

Ce manuel est disponible :

- En tant que site web HTML: en Français ou en Anglais
- En tant que fichier PDF: en Français ou en Anglais

Généré le 2026-11-18 à partir du commit superbol-manuals@a5e6f46

1 Préface

Ce manuel décrit **GixSQL**, un précompilateur SQL embarqué (ESQL) et une bibliothèque d'exécution pour GnuCOBOL. Il couvre l'édition SuperBOL de GixSQL, maintenue sur https://github.com/superbol4gnucobol/gixsql-superbol, qui poursuit le travail original de Marco Ridoni (https://github.com/mridoni/gixsql).

1.1 À qui s'adresse ce manuel

Ce manuel s'adresse aux programmeurs COBOL qui écrivent des programmes contenant des instructions SQL embarquées (`EXEC SQL ... END-EXEC`), et aux personnes qui portent vers GnuCOBOL des programmes écrits pour d'autres précompilateurs, en particulier Oracle Pro*COBOL. Une connaissance de COBOL et de SQL est supposée.

1.2 Organisation de ce manuel

L'organisation suit celle de la plupart des manuels de précompilateurs ESQL, afin que les lecteurs qui connaissent Pro*COBOL s'y retrouvent :

- **Prise en main** : ce qu'est GixSQL, comment l'installer, et comment précompiler, compiler et exécuter un premier programme.
- **Programmer en SQL embarqué** : connexions et transactions, variables hôtes et types de données, instructions SQL, gestion des erreurs et SQL dynamique.
- **Référence** : les options du précompilateur `gixpp`, la configuration de l'exécution, et un guide pour migrer des programmes Pro*COBOL.
- **Backends de bases de données** : les notes propres à chaque pilote de base de données.
- **Annexes** : un récapitulatif des instructions SQL embarquées, et les codes d'erreur.

1.3 Conventions

- Les options de `gixpp` sont écrites sous la forme `--option-longue` (avec leur forme courte `-x` lorsqu'elle existe).
- Les variables d'environnement sont écrites en majuscules : `GIXSQL_LOG_LEVEL`.
- Les exemples COBOL utilisent le format de source fixe : le code commence en colonne 8.

2 Introduction

2.1 Qu'est-ce que GixSQL ?

GixSQL permet aux programmes COBOL compilés avec GnuCOBOL d'accéder à des bases de données relationnelles grâce au SQL embarqué. Un programme contient des instructions SQL dans des blocs `EXEC SQL ... END-EXEC` :

```
           EXEC SQL
               SELECT COUNT(*) INTO :T1 FROM EMPTABLE
           END-EXEC.
```

Ces blocs ne sont pas du COBOL : un **précompilateur** les traduit en appels à une **bibliothèque d'exécution**, qui envoie les instructions à la base de données et transfère les données entre la base et les champs COBOL.

2.2 Composants

GixSQL est constitué de :

`gixpp`
: le précompilateur. Il lit un source COBOL contenant du SQL embarqué et écrit un source en COBOL pur, où chaque bloc `EXEC SQL` est mis en commentaire et remplacé par des appels à la bibliothèque d'exécution (voir Le précompilateur gixpp).

`libgixsql`
: la bibliothèque d'exécution, liée au programme COBOL. Elle gère les connexions, les curseurs et les variables hôtes, et charge le pilote de la base de données à l'exécution.

`libgixsql-<driver>`
: les pilotes de bases de données, chargés dynamiquement par `libgixsql` en fonction de la chaîne de connexion (voir Vue d'ensemble des backends) :

Pilote	Base de données	Bibliothèque cliente
`pgsql`	PostgreSQL	`libpq`
`oracle`	Oracle Database	ODPI-C (intégrée)
`mysql`	MySQL, MariaDB	`libmariadb`/`libmysqlclient`
`odbc`	toute base disposant d'un pilote ODBC	unixODBC
`sqlite`	SQLite	intégrée

copybook `SQLCA`
: la zone de communication SQL (*SQL Communication Area*), installée avec la bibliothèque d'exécution (`share/gixsql/copy`), et incluse dans les programmes avec `EXEC SQL INCLUDE SQLCA END-EXEC`.

2.3 Processus de développement



Figure 1. – Processus de développement

1. Écrire le programme avec du SQL embarqué.
2. Le précompiler avec `gixpp`.
3. Compiler et lier le résultat avec `cobc`, en ajoutant `libgixsql`.
4. L'exécuter, avec les bibliothèques des pilotes de bases de données dans le chemin de recherche des bibliothèques.

Concepts du précompilateur détaille chaque étape avec un exemple complet.

2.4 Relation avec d'autres précompilateurs

GixSQL est à l'origine un fork d'Open-COBOL-ESQL (OCESQL), en grande partie réécrit en C++. L'édition SuperBOL ajoute la compatibilité avec les programmes écrits pour Oracle Pro*COBOL (voir Migrer depuis Pro*COBOL).

D'autres précompilateurs open source pour GnuCOBOL existent :

- OCESQL : https://github.com/opensourcecobol/Open-COBOL-ESQL (PostgreSQL) ;
- esqlOC : https://github.com/OCamlPro/gnucobol-contrib/tree/master/esql (ODBC).

2.5 Résumé des fonctionnalités

Les fonctionnalités de GixSQL sont comparées à celles de Pro*COBOL dans Comparaison des fonctionnalités, et le support de chaque backend est résumé dans Vue d'ensemble des backends.

3 Installation

GixSQL est distribué sous forme de code source. Ce chapitre explique comment le compiler et l'installer sous GNU/Linux. La compilation pour Windows (Visual Studio ou MSYS2/MinGW) est décrite dans Compilation sous Windows.

3.1 Prérequis

Pour compiler GixSQL, il faut :

- un compilateur C++17 (GCC 8 ou plus récent, ou Clang) ;
- les autotools (`autoconf`, `automake`, `libtool`) et `pkg-config`, pour compiler depuis le dépôt git ;
- `bison` 3.7 ou plus récent et `flex` ;
- le paquet de développement de spdlog (et de `fmt`, sa dépendance) ;
- pour chaque pilote de base de données souhaité, le paquet de développement de sa bibliothèque cliente :

Pilote	Paquet Debian/Ubuntu	Remarques
`pgsql`	`libpq-dev`	
`mysql`	`libmariadb-dev` (ou `libmysqlclient-dev`)	
`odbc`	`unixodbc-dev`	plus le pilote ODBC de la base de données
`oracle`	aucun pour la compilation	les bibliothèques clientes Oracle (Instant Client) sont nécessaires à l'exécution
`sqlite`	aucun	SQLite est intégré

Sous Debian ou Ubuntu :

```
$ sudo apt install build-essential autoconf automake libtool pkg-config \
    bison flex libspdlog-dev libfmt-dev \
    libpq-dev libmariadb-dev unixodbc-dev
```

Pour exécuter les programmes, il faut aussi GnuCOBOL (`cobc` et `libcob`).

3.2 Compilation depuis le dépôt git

```
$ git clone https://github.com/superbol4gnucobol/gixsql-superbol.git
$ cd gixsql-superbol
$ autoreconf -fi
$ mkdir _build && cd _build
$ ../configure --prefix=/opt/gixsql
$ make
$ sudo make install
```

Les fichiers générés par les autotools ne sont pas stockés dans le dépôt : `autoreconf -fi` est toujours nécessaire après un clone.

3.3 Options de configuration

Par défaut, `configure` compile le précompilateur et chaque pilote dont la bibliothèque cliente est trouvée. Les options suivantes modifient ce comportement :

Option	Effet
`--enable-gixpp` / `--disable-gixpp`	compiler le précompilateur `gixpp` (par défaut : oui)
`--disable-pgsql`	ne pas compiler le pilote PostgreSQL
`--disable-oracle`	ne pas compiler le pilote Oracle
`--disable-mysql`	ne pas compiler le pilote MySQL/MariaDB
`--disable-odbc`	ne pas compiler le pilote ODBC
`--disable-sqlite`	ne pas compiler le pilote SQLite
`--with-default-driver=DRIVER`	pilote utilisé lorsque la chaîne de connexion n'en nomme aucun : `none` (par défaut), `pgsql`, `oracle`, `mysql`, `odbc`, `sqlite` (voir Le pilote par défaut)

Lorsqu'un pilote est explicitement activé (`--enable-pgsql`) et que sa bibliothèque cliente est absente, `configure` échoue ; sinon, le pilote est ignoré silencieusement. Vérifiez le résumé affiché à la fin de `configure`.

Par exemple, pour compiler uniquement le précompilateur :

```
$ ../configure --disable-mysql --disable-odbc --disable-pgsql \
               --disable-oracle --disable-sqlite --enable-gixpp
```

Pour compiler la bibliothèque d'exécution avec PostgreSQL et Oracle seulement :

```
$ ../configure --prefix=/opt/gixsql --disable-mysql --disable-odbc --disable-sqlite
```

3.4 Fichiers installés

Avec `--prefix=/opt/gixsql`, `make install` installe :

Fichier	Description
`bin/gixpp`	le précompilateur
`lib/libgixsql.so`	la bibliothèque d'exécution, liée aux programmes
`lib/libgixsql-<driver>.so`	les pilotes de bases de données, chargés à l'exécution
`share/gixsql/copy/SQLCA.cpy`	le copybook SQLCA

> **ℹ À vérifier**
>
> Vérifier la liste exacte et les chemins des fichiers installés (`make install DESTDIR=...`).

À l'exécution, `libgixsql` charge le pilote `libgixsql-<driver>.so` avec `dlopen` : le répertoire `lib` doit se trouver dans le chemin de recherche des bibliothèques, par exemple :

```
$ export LD_LIBRARY_PATH=/opt/gixsql/lib:$LD_LIBRARY_PATH
```

3.5 Compilation avec Nix

Le dépôt contient un flake Nix :

```
$ nix build .#runtime     # the runtime, with the Oracle and PostgreSQL drivers
$ nix build .#gixpp       # the precompiler only
$ result/bin/gixpp --version
```

3.6 Compilation sous Windows

3.6.1 Visual Studio

Le fichier de solution `gixsql.sln` compile GixSQL avec Visual Studio 2019 ou plus récent. Les dépendances sont attendues depuis vcpkg :

```
> vcpkg install libpq:x64-windows libmariadb:x64-windows fmt:x64-windows-static-md
spdlog:x64-windows-static-md
```

3.6.2 MSYS2/MinGW

Le script `configure` fonctionne aussi sous MSYS2, avec les paquets :

```
$ pacman -S mingw-w64-x86_64-pkg-config autoconf make automake libtool bison flex \
   mingw-w64-x86_64-gcc mingw-w64-x86_64-postgresql mingw-w64-x86_64-libmariadbclient
\
    mingw-w64-x86_64-unixodbc mingw-w64-x86_64-spdlog
```

> ⚠ Avertissement
>
> Les compilations sous Windows ne sont pas testées par les mainteneurs de SuperBOL.

3.7 Vérification de l'installation

```
$ gixpp --version
gixpp - the ESQL preprocessor for Gix-IDE/GixSQL
Version: 1.0.21dev
libgixpp version: 1.0.21dev
```

La suite de tests du précompilateur utilise autofonce : depuis le répertoire des sources, après la compilation dans `_build`, lancez `autofonce run`.

4 Concepts du précompilateur

4.1 SQL embarqué dans un programme COBOL

Une instruction SQL embarquée commence par `EXEC SQL` et se termine par `END-EXEC`, suivi d'un point lorsque l'instruction termine une phrase COBOL :

```
           EXEC SQL
               UPDATE EMP SET SAL = SAL * 1.1 WHERE DEPTNO = :DEPT-NO
           END-EXEC.
```

À l'intérieur de l'instruction, les données COBOL, appelées **variables hôtes**, sont préfixées par deux-points (`:DEPT-NO`). Le précompilateur transmet leur adresse, leur type et leur taille à la bibliothèque d'exécution, qui convertit leur contenu depuis et vers les types de la base de données (voir Variables hôtes et types de données).

Les instructions SQL embarquées peuvent apparaître :

- dans la DATA DIVISION : `INCLUDE`, `BEGIN/END DECLARE SECTION`, `DECLARE ... CURSOR`, `VAR`, `DECLARE TABLE` et `WHENEVER`…

  > **i** À vérifier
  >
  > Vérifier où chaque instruction est acceptée par `gixpp`.

- dans la PROCEDURE DIVISION : toutes les instructions exécutables.

4.2 La section de déclaration

Pro*COBOL et la norme SQL exigent que les variables hôtes soient déclarées entre `BEGIN DECLARE SECTION` et `END DECLARE SECTION` :

```
           EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01  EMP-NAME    PIC X(20).
       01  SALARY      PIC S9(7)V99 COMP-3.
           EXEC SQL END DECLARE SECTION END-EXEC.
```

GixSQL accepte ces instructions, mais ne les exige pas : toute donnée de la WORKING-STORAGE, LOCAL-STORAGE ou LINKAGE SECTION peut être utilisée comme variable hôte.

4.3 Inclusion de fichiers

`EXEC SQL INCLUDE name END-EXEC` inclut un copybook, comme une instruction COBOL `COPY`, mais le fichier inclus est également traité par le précompilateur. Elle sert surtout pour la SQLCA :

```
           EXEC SQL INCLUDE SQLCA END-EXEC.
```

Les copybooks sont recherchés dans les répertoires indiqués avec `-I` / `--copypath`, avec les extensions indiquées avec `-E` / `--esql-copy-exts`. Les instructions COBOL `COPY` ne sont traitées par le précompilateur qu'avec `-p` / `--esql-preprocess-copy` (voir Le précompilateur gixpp).

4.4 Précompilation, compilation et édition de liens

La séquence complète pour construire un programme `prog.pco` est :

```
$ gixpp -e -S -I /opt/gixsql/share/gixsql/copy -I . \
        -i prog.pco -o prog.cob
$ cobc -x prog.cob -L /opt/gixsql/lib -lgixsql
```

où :

- `-e` active le traitement ESQL (il est obligatoire) ;
- `-S` génère des appels statiques à la bibliothèque d'exécution (recommandé) ;
- `-I` indique les répertoires des copybooks : celui de la SQLCA et ceux du programme ;
- `-i` et `-o` indiquent les fichiers d'entrée et de sortie.

4.5 Un premier programme

Le programme suivant se connecte à une base de données dont la chaîne de connexion se trouve dans la variable d'environnement `DATASRC`, compte les lignes d'une table et affiche le résultat.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. FIRST.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  DATASRC  PIC X(64).
       01  DBUSR    PIC X(64).
       01  DBPWD    PIC X(64).
       01  NB-ROWS  PIC 9(6) VALUE 0.
           EXEC SQL INCLUDE SQLCA END-EXEC.
       PROCEDURE DIVISION.
           ACCEPT DATASRC FROM ENVIRONMENT "DATASRC".
           ACCEPT DBUSR   FROM ENVIRONMENT "DATASRC_USR".
```

```
           ACCEPT DBPWD   FROM ENVIRONMENT "DATASRC_PWD".

           EXEC SQL
               CONNECT TO :DATASRC USER :DBUSR USING :DBPWD
           END-EXEC.
           IF SQLCODE NOT = 0
               DISPLAY "CONNECT failed: " SQLCODE " " SQLERRMC
               STOP RUN
           END-IF.

           EXEC SQL
               SELECT COUNT(*) INTO :NB-ROWS FROM EMP
           END-EXEC.
           DISPLAY "EMP contains " NB-ROWS " rows".

           EXEC SQL CONNECT RESET END-EXEC.
           STOP RUN.
```

Pour le construire et l'exécuter sur une base PostgreSQL :

```
$ gixpp -e -S -I /opt/gixsql/share/gixsql/copy -i first.pco -o first.cob
$ cobc -x first.cob -L /opt/gixsql/lib -lgixsql
$ export LD_LIBRARY_PATH=/opt/gixsql/lib
$ export DATASRC=pgsql://localhost:5432/demo
$ export DATASRC_USR=demo DATASRC_PWD=secret
$ ./first
EMP contains 000014 rows
```

> **i** À vérifier
>
> Exécuter cet exemple sur les bases de test et coller la sortie réelle.

4.6 Tables d'exemple

Les exemples de ce manuel utilisent les tables classiques `DEPT` et `EMP` :

```
CREATE TABLE DEPT (
  DEPTNO  NUMERIC(2) PRIMARY KEY,
  DNAME   VARCHAR(14),
  LOC     VARCHAR(13)
);
```

```sql
CREATE TABLE EMP (
  EMPNO    NUMERIC(4) PRIMARY KEY,
  ENAME    VARCHAR(10),
  JOB      VARCHAR(9),
  MGR      NUMERIC(4),
  HIREDATE DATE,
  SAL      NUMERIC(7,2),
  COMM     NUMERIC(7,2),
  DEPTNO   NUMERIC(2) REFERENCES DEPT
);
```

5 Connexions et transactions

5.1 Connexion à une base de données

GixSQL n'a pas d'étape de « bind » : un programme doit se connecter explicitement avec une instruction `CONNECT` avant d'exécuter toute autre instruction SQL. Les formes suivantes sont acceptées, où chaque opérande est soit une variable hôte (`:VAR`), soit une chaîne littérale :

```
EXEC SQL CONNECT TO :DATASRC END-EXEC.
EXEC SQL CONNECT TO :DATASRC USER :USR-PWD END-EXEC.
EXEC SQL CONNECT TO :DATASRC USER :USR USING :PWD END-EXEC.
EXEC SQL CONNECT :USR IDENTIFIED BY :PWD USING :DATASRC END-EXEC.
EXEC SQL CONNECT USING :DATASRC END-EXEC.
```

- `DATASRC` contient une **chaîne de connexion** (Chaînes de connexion).
- Le nom d'utilisateur et le mot de passe sont donnés soit séparément, soit dans une seule donnée sous la forme `user.password` (un point les sépare).
- Avec `CONNECT USING`, les identifiants doivent figurer dans la chaîne de connexion.
- La forme Pro*COBOL `CONNECT :USR-PWD`, où une seule donnée contient `user/password`, n'est pas supportée : utilisez `IDENTIFIED BY`.

Après le `CONNECT`, `SQLCODE` vaut 0 en cas de succès. En cas d'échec, il est négatif et `SQLERRMC` contient le message de la base de données.

5.2 Chaînes de connexion

Le format général d'une chaîne de connexion est :

```
<driver>://[<user>.<password>@]<host>[:<port>][/<database>]
[?<opt>=<val>[&<opt>=<val>]...]
```

où `<driver>` est l'un de `pgsql`, `oracle`, `mysql`, `odbc`, `sqlite` (voir Vue d'ensemble des backends). Exemples :

```
pgsql://localhost:5432/demo
pgsql://demo.secret@dbserver/demo?default_schema=app
oracle://dbserver:1521/FREEPDB1
mysql://localhost:3306/demo
odbc://MYDSN
sqlite:///var/data/demo.db
```

Deux autres formats sont acceptés, sans le nom du pilote :

- `[<user>.<password>@]<host>[:<port>][/<database>]`
- `<database>@<host>[:<port>]` (le format d'OCESQL)

5.2.1 Le pilote par défaut

Lorsque la chaîne de connexion ne nomme pas de pilote, GixSQL utilise le pilote par défaut, qui est :

1. le pilote choisi lors de la compilation de GixSQL (`configure --with-default-driver=...`), s'il y en a un ;
2. sinon, le contenu de la variable d'environnement `GIXSQL_DEFAULT_DRIVER`.

S'il n'y a pas de pilote par défaut, le `CONNECT` échoue.

5.2.2 Options de connexion

Les options suivent un `?` à la fin de la chaîne de connexion, séparées par `&`. Les options booléennes acceptent `on`/`off` ou `1`/`0`.

Option	Pilotes	Description
`autocommit`	tous	`on`, `off` ou `native` (Autocommit)
`client_encoding`	pgsql, mysql	encodage du client, par exemple `UTF8` (PostgreSQL) ou `utf8mb4` (MySQL)
`fixup_params`	tous	convertir les marqueurs de paramètres des instructions préparées (Marqueurs de paramètres)
`default_schema`	pgsql	le `search_path` (un ou plusieurs schémas, séparés par des virgules)
`native_cursors`	pgsql	utiliser les curseurs natifs de PostgreSQL (par défaut `on`)
`decode_binary`	pgsql	décoder les valeurs `bytea` (par défaut `on`)
`updatable_cursors`	mysql, sqlite	émuler les curseurs modifiables
`sslmode`, `sslcert`, …	pgsql	transmises à libpq (PostgreSQL)

Plusieurs options peuvent aussi être définies par des variables d'environnement (Configuration de l'exécution) ; l'option de la chaîne de connexion est prioritaire.

5.3 Connexions multiples

Un programme peut ouvrir plusieurs connexions, chacune avec un nom :

```
           EXEC SQL
               CONNECT TO :DATASRC-1 AS CONN1 USER :DBUSR-1
           END-EXEC.
           EXEC SQL
               CONNECT TO :DATASRC-2 AS CONN2 USER :DBUSR-2
           END-EXEC.
```

ou, avec la syntaxe Pro*COBOL :

```
           EXEC SQL
               CONNECT :USR IDENTIFIED BY :PWD AT CONN2 USING :DATASRC-2
           END-EXEC.
```

Une instruction est alors exécutée sur une connexion donnée avec la clause `AT` :

```
           EXEC SQL AT CONN1 DELETE FROM TAB1 END-EXEC.
           EXEC SQL AT CONN2 DELETE FROM TAB2 END-EXEC.
```

Les instructions sans `AT` utilisent la connexion par défaut, celle ouverte sans nom.

> ℹ À vérifier
>
> Quelle est la connexion par défaut lorsque toutes les connexions sont nommées ?

5.4 Déconnexion

```
           EXEC SQL CONNECT RESET END-EXEC.
           EXEC SQL CONNECT RESET CONN1 END-EXEC.
           EXEC SQL DISCONNECT ALL END-EXEC.
```

Les programmes Pro*COBOL se déconnectent avec `COMMIT WORK RELEASE` ou `ROLLBACK WORK RELEASE` : voir Migrer depuis Pro*COBOL.

5.5 Transactions

Une transaction est une suite d'instructions SQL qui sont validées (COMMIT) ou annulées (ROLLBACK) ensemble. Elle se termine par :

```
           EXEC SQL COMMIT WORK END-EXEC.
           EXEC SQL ROLLBACK WORK END-EXEC.
```

`SAVEPOINT name` et `ROLLBACK TO SAVEPOINT name` sont transmises telles quelles à la base de données.

5.5.1 Autocommit

La manière dont les transactions commencent dépend du **mode autocommit**, défini par l'option `autocommit` de la chaîne de connexion ou par la variable d'environnement `GIXSQL_AUTOCOMMIT` :

`native` (par défaut)
: la connexion utilise le comportement par défaut de la base de données : autocommit pour PostgreSQL, MySQL et ODBC, transactions explicites pour Oracle.

`on`
: chaque instruction réussie est validée.

`off`
: une transaction est démarrée lors de la connexion, puis une nouvelle après chaque `COMMIT` et chaque `ROLLBACK`.

> ! Important
>
> Sous PostgreSQL, les curseurs natifs (par défaut) n'existent qu'à l'intérieur d'une transaction : en mode autocommit, un `DECLARE CURSOR` échoue. Les programmes qui utilisent des curseurs doivent soit démarrer explicitement des transactions (`EXEC SQL START TRANSACTION END-EXEC`), soit s'exécuter avec `GIXSQL_AUTOCOMMIT=off`, qui est le comportement attendu par les programmes Pro*COBOL.

> ¡ À vérifier
>
> L'ancien README indique que la valeur par défaut est `off`, le code (`GIXSQL_AUTOCOMMIT_DEFAULT` dans `runtime/libgixsql/gixsql.h`) indique `native`. Vérifier le comportement réel de chaque pilote en mode `native`.

6 Variables hôtes et types de données

6.1 Variables hôtes

Une variable hôte est une donnée COBOL utilisée dans une instruction SQL, préfixée par deux-points. C'est une variable d'**entrée** lorsque sa valeur est envoyée à la base de données (clauses `WHERE`, `VALUES`, `SET`), et une variable de **sortie** lorsqu'elle reçoit une valeur de la base de données (clauses `INTO`).

```
EXEC SQL
    SELECT ENAME, SAL INTO :EMP-NAME, :SALARY
      FROM EMP WHERE EMPNO = :EMP-NUMBER
END-EXEC.
```

Les noms des variables hôtes ne sont pas sensibles à la casse et peuvent être qualifiés par leur groupe, comme en COBOL (`:EMPNO OF EMP-REC`).

> ℹ À vérifier
>
> Qualification (`OF`/`IN`) et utilisation d'une donnée de groupe comme liste de variables hôtes (`INTO :EMP-REC`).

6.2 Conversion des types de données

La bibliothèque d'exécution convertit les données COBOL depuis et vers les types de la base de données, selon la `PICTURE` et l'`USAGE` de la variable hôte :

Déclaration COBOL	Envoyée à la base de données comme
`PIC X(n)`	chaîne de caractères (voir Chaînes de caractères de longueur fixe)
`PIC 9(n)`, `PIC S9(n)V9(m)` (DISPLAY)	nombre
`PIC S9(n)V9(m) COMP-3`	nombre
`PIC S9(n) COMP`, `COMP-5`, `BINARY`	entier
`COMP-1`, `COMP-2`	virgule flottante
`PIC N(n)`	chaîne nationale

> **i** À compléter
>
> Compléter ce tableau à partir de `CobolVarType` (`runtime/libgixsql/SqlVar.h`) et documenter la conversion des dates.

6.3 Chaînes de caractères de longueur fixe

L'option du précompilateur `-P` / `--picx-as` choisit comment les données `PIC X` sont envoyées à la base de données :

`char` (par défaut), `charf`
: comme des chaînes de longueur fixe, espaces de fin compris ;

`varchar`
: comme des chaînes de longueur variable, espaces de fin supprimés.

Cette option ne s'applique pas aux données `VARYING`, dont la longueur est toujours donnée par leur champ de longueur.

6.4 Chaînes de longueur variable : VARYING

Une donnée `VARYING` possède une partie longueur et une partie données :

```
       01  ENAME  PIC X(20) VARYING.
```

Le précompilateur la remplace par un groupe de deux données de niveau 49 :

```
       01  ENAME.
           49 ENAME-LEN  PIC 9(8) COMP-5.
           49 ENAME-ARR  PIC X(20).
```

En sortie, la bibliothèque d'exécution affecte à `ENAME-LEN` la longueur de la valeur ; en entrée, seuls les `ENAME-LEN` premiers caractères sont envoyés.

- Les suffixes `-LEN` et `-ARR` peuvent être modifiés avec `-Y` / `--varying` (par exemple `--varying=LEN,TEXT`).
- Le champ de longueur fait 4 octets par défaut. Pro*COBOL utilise une longueur sur 2 octets (`PIC S9(4) COMP`) : utilisez `-N 2` / `--varying-length-size 2`. Le code généré indique à la bibliothèque d'exécution la taille des champs de longueur.

Les données de groupe contenant des données de niveau 49 écrites à la main sont également reconnues comme des chaînes de longueur variable, comme dans Pro*COBOL. Leur champ de longueur doit avoir la taille donnée par `-N`.

> **i** À vérifier
>
> Détection des groupes de niveau 49 écrits à la main, et `PIC X(n) VARYING` par rapport à `USAGE VARRAW`.

6.5 SQL TYPE IS

La clause `SQL TYPE IS` déclare une variable hôte avec un type SQL :

```
       01  BUFFER  SQL TYPE IS VARBINARY(100).
       01  STMT    SQL TYPE IS VARCHAR(500).
```

`VARCHAR` et `VARBINARY` génèrent un groupe `VARYING` comme ci-dessus.

6.6 EXEC SQL VAR

L'instruction `VAR` donne un type SQL à une donnée existante (*datatype equivalencing*, dans la terminologie de Pro*COBOL) :

```
       01  VARD  PIC X(120).
       01  NUM3  COMP-2.
       ...
           EXEC SQL VAR VARD IS VARCHAR(120) END-EXEC.
           EXEC SQL VAR NUM3 IS FLOAT END-EXEC.
```

Les types supportés sont `CHAR`, `CHARF`, `VARCHAR`, `VARCHAR2` (identique à `VARCHAR`), `VARBINARY`, `INTEGER`, `FLOAT`, `DECIMAL` et `LONG RAW`.

6.7 Variables indicatrices

Une variable indicatrice est associée à une variable hôte pour gérer les valeurs `NULL`. Elle doit être déclarée `PIC S9(4) COMP`, et s'écrit juste après la variable hôte, éventuellement précédée de `INDICATOR` :

```
       01  COMM      PIC S9(5)V99 COMP-3.
       01  COMM-IND  PIC S9(4) COMP.
       ...
           EXEC SQL
               SELECT COMM INTO :COMM:COMM-IND
                 FROM EMP WHERE EMPNO = :EMP-NUMBER
           END-EXEC.
           IF COMM-IND = -1
```

```
            DISPLAY "No commission"
        END-IF.
```

Valeur	En sortie (`SELECT`, `FETCH`)	En entrée (`INSERT`, `UPDATE`)
`-1`	la colonne est `NULL`	`NULL` est envoyé, la variable hôte est ignorée
`0`	la valeur a été stockée	la variable hôte est envoyée

La troncature (valeurs positives de l'indicateur) n'est pas signalée.

> **i** À vérifier
>
> Mot-clé `INDICATOR`, tables d'indicateurs pour les données de groupe.

6.8 Tables hôtes (tableaux)

GixSQL ne supporte pas les tables hôtes : une variable hôte ne peut pas être une table (`OCCURS`) permettant de lire ou d'insérer plusieurs lignes en une seule instruction. Utilisez plutôt un curseur et une boucle. Voir Comparaison des fonctionnalités.

7 Instructions SQL embarquées

7.1 Exécution d'instructions SQL

La plupart des instructions SQL sont transmises telles quelles à la base de données, après remplacement des variables hôtes par des marqueurs de paramètres : `INSERT`, `UPDATE`, `DELETE`, instructions DDL (`CREATE`, `DROP`…), `SAVEPOINT`, etc. Le précompilateur ne vérifie pas la syntaxe SQL : les erreurs sont signalées par la base de données à l'exécution.

```
EXEC SQL
    INSERT INTO EMP (EMPNO, ENAME, SAL, DEPTNO)
    VALUES (:EMP-NUMBER, :EMP-NAME, :SALARY, :DEPT-NO)
END-EXEC.

EXEC SQL
    UPDATE EMP SET SAL = :SALARY WHERE EMPNO = :EMP-NUMBER
END-EXEC.

EXEC SQL
    DELETE FROM EMP WHERE DEPTNO = :DEPT-NO
END-EXEC.
```

Après un `INSERT`, un `UPDATE` ou un `DELETE`, le champ `SQLERRD(3)` de la SQLCA contient le nombre de lignes traitées.

> ℹ À vérifier
>
> `SQLERRD(3)` pour chaque pilote.

7.2 Requêtes mono-ligne : SELECT INTO

Un `SELECT` qui renvoie au plus une ligne stocke directement son résultat dans des variables hôtes :

```
EXEC SQL
    SELECT ENAME, SAL, COMM
      INTO :EMP-NAME, :SALARY, :COMM:COMM-IND
      FROM EMP
```

```
             WHERE EMPNO = :EMP-NUMBER
        END-EXEC.
```

Si aucune ligne n'est trouvée, `SQLCODE` contient le code « aucun enregistrement » (100 par défaut, voir La condition « aucun enregistrement »). Si plusieurs lignes sont trouvées, une erreur est renvoyée.

7.3 Curseurs

Une requête qui renvoie plusieurs lignes est traitée avec un curseur :

1. déclarer le curseur (`DECLARE`), avec sa requête ;
2. l'ouvrir (`OPEN`) : la requête est exécutée ;
3. lire les lignes une par une (`FETCH`), jusqu'à la condition « aucun enregistrement » ;
4. le fermer (`CLOSE`).

```
        EXEC SQL
            DECLARE EMPCUR CURSOR FOR
              SELECT ENAME, SAL FROM EMP
               WHERE DEPTNO = :DEPT-NO
               ORDER BY ENAME
        END-EXEC.

        EXEC SQL OPEN EMPCUR END-EXEC.
        PERFORM UNTIL SQLCODE NOT = 0
            EXEC SQL
                FETCH EMPCUR INTO :EMP-NAME, :SALARY
            END-EXEC
            IF SQLCODE = 0
                DISPLAY EMP-NAME " " SALARY
            END-IF
        END-PERFORM.
        EXEC SQL CLOSE EMPCUR END-EXEC.
```

Les variables hôtes d'entrée de la requête (`:DEPT-NO`) sont lues à l'ouverture du curseur.

- `DECLARE CURSOR` peut apparaître dans la DATA DIVISION ou dans la PROCEDURE DIVISION. Ce n'est pas une instruction exécutable : elle ne modifie pas `SQLCODE`.
- `FETCH` n'accepte pas la clause `AT` : le curseur est utilisé sur la connexion où il a été déclaré.
- Un curseur déclaré `WITH HOLD` reste ouvert après un `COMMIT` :

```
        EXEC SQL
            DECLARE C1 CURSOR WITH HOLD FOR SELECT ...
        END-EXEC.
```

> ! Important
>
> Avec les curseurs natifs de PostgreSQL, les curseurs doivent être utilisés à l'intérieur d'une transaction : voir Autocommit.

7.3.1 Curseurs modifiables

La ligne courante d'un curseur déclaré `FOR UPDATE` peut être modifiée ou supprimée avec `WHERE CURRENT OF` :

```
        EXEC SQL
            DECLARE C2 CURSOR FOR
              SELECT ENAME, SAL FROM EMP FOR UPDATE
        END-EXEC.
        ...
        EXEC SQL
            UPDATE EMP SET SAL = SAL * 1.1 WHERE CURRENT OF C2
        END-EXEC.
```

Leur support dépend de la base de données :

- Oracle et PostgreSQL (avec les curseurs natifs) les supportent directement.
- MySQL et SQLite les émulent, avec l'option `updatable_cursors` ; la table doit avoir une clé unique.
- Avec ODBC, cela dépend du pilote ODBC.

7.3.2 Curseurs sur des instructions préparées

Un curseur peut aussi être déclaré sur une instruction préparée (voir SQL dynamique) :

```
        EXEC SQL DECLARE C3 CURSOR FOR STMT1 END-EXEC.
```

7.4 Transactions

Voir Connexions et transactions.

7.5 Autres instructions

> ℹ À compléter
>
> `EXEC SQL IGNORE`, `CALL` / procédures stockées, `LOCK TABLE`, `SET TRANSACTION`, `DECLARE TABLE`.

8 Gestion des erreurs et diagnostics

8.1 La zone de communication SQL

La SQLCA (*SQL Communication Area*) est une structure de données mise à jour par chaque instruction SQL exécutable. Elle est incluse dans la WORKING-STORAGE SECTION avec :

```
           EXEC SQL INCLUDE SQLCA END-EXEC.
```

Ses principaux champs sont :

Champ	Description
`SQLCODE`	0 en cas de succès, une valeur positive pour les avertissements et « aucun enregistrement », une valeur négative pour les erreurs
`SQLSTATE`	le code SQLSTATE à 5 caractères de la norme SQL
`SQLERRML`, `SQLERRMC`	longueur et texte du message d'erreur
`SQLERRD(3)`	nombre de lignes traitées
`SQLWARN0` …	indicateurs d'avertissement

> **i** À compléter
>
> Recopier la structure réelle de `copy/SQLCA.cpy`.

8.2 SQLCODE et SQLSTATE

Lorsque la base de données signale une erreur, GixSQL affecte à `SQLCODE` le code d'erreur de la base de données s'il existe (par exemple `-942` pour l'erreur Oracle `ORA-00942`) et à `SQLSTATE` le SQLSTATE correspondant. Lorsque l'erreur provient de GixSQL lui-même, `SQLCODE` est l'un des codes négatifs listés dans Codes d'erreur.

> **i** À vérifier
>
> Correspondance de `SQLCODE` pour chaque pilote (PostgreSQL n'a pas de codes numériques).

8.3 La condition « aucun enregistrement »

Lorsqu'un `SELECT INTO` ne trouve aucune ligne, ou qu'un `FETCH` atteint la fin d'un curseur, `SQLSTATE` vaut `02000` et `SQLCODE` vaut 100. Oracle utilise 1403 au lieu de 100 ; pour obtenir cette valeur :

- précompilez avec `--no-rec-code 1403`, afin que `WHENEVER NOT FOUND` teste cette valeur ;
- exécutez avec `GIXSQL_NOREC_CODE=1403`, afin que la bibliothèque d'exécution l'utilise.

Les deux réglages doivent être cohérents.

8.4 WHENEVER

L'instruction `WHENEVER` demande au précompilateur de générer un test après chacune des instructions SQL suivantes, jusqu'au prochain `WHENEVER` pour la même condition :

```
           EXEC SQL WHENEVER SQLERROR GO TO SQL-ERROR END-EXEC.
           EXEC SQL WHENEVER NOT FOUND GO TO END-OF-FETCH END-EXEC.
           EXEC SQL WHENEVER SQLWARNING CONTINUE END-EXEC.
```

Conditions :

`SQLERROR`
: `SQLCODE` est négatif ;

`NOT FOUND`
: la condition « aucun enregistrement » ;

`SQLWARNING`
: un avertissement est positionné.

Actions :

`CONTINUE`
: pas de test (par défaut) ;

`GO TO label`, **`GOTO label`**
: saut au paragraphe ou à la section `label` ;

`PERFORM label`
: exécution (PERFORM) du paragraphe ou de la section `label`.

`WHENEVER` est une déclaration : elle s'applique aux instructions qui la suivent dans le source, et non dans l'ordre d'exécution.

8.5 Journalisation

Lorsqu'une erreur n'est pas claire d'après la SQLCA, le journal de la bibliothèque d'exécution donne le détail de chaque appel, y compris les instructions SQL envoyées à la base de données (voir Journalisation).

9 SQL dynamique

Les instructions SQL dynamiques sont construites à l'exécution, dans une variable hôte. GixSQL supporte les trois premières « méthodes » de Pro*COBOL ; la quatrième (descripteurs, `DESCRIBE`) n'est pas supportée.

9.1 Méthode 1 : EXECUTE IMMEDIATE

Pour une instruction sans variables hôtes et qui ne renvoie aucune ligne :

```
           MOVE "DELETE FROM EMP WHERE DEPTNO = 40" TO SQL-STMT.
           EXEC SQL EXECUTE IMMEDIATE :SQL-STMT END-EXEC.

           EXEC SQL
               EXECUTE IMMEDIATE 'UPDATE TAB1 SET FLD1 = FLD1 + 100'
           END-EXEC.
```

9.2 Méthode 2 : PREPARE et EXECUTE

Pour une instruction exécutée plusieurs fois, avec des paramètres d'entrée :

```
       01  DYNSTMT  SQL TYPE IS VARCHAR(200).
       ...
           MOVE "INSERT INTO TAB1 (F1, F2) VALUES (?, ?)" TO DYNSTMT-ARR.
           MOVE FUNCTION LENGTH(FUNCTION TRIM(DYNSTMT-ARR))
             TO DYNSTMT-LEN.
           EXEC SQL PREPARE STMT1 FROM :DYNSTMT END-EXEC.

           MOVE 1 TO T1. MOVE 2 TO T2.
           EXEC SQL EXECUTE STMT1 USING :T1, :T2 END-EXEC.
```

`EXECUTE ... INTO :a, :b USING ...` renvoie les valeurs d'une requête mono-ligne.

9.3 Méthode 3 : requêtes préparées et curseurs

Pour une requête renvoyant plusieurs lignes :

```
           EXEC SQL PREPARE STMT2 FROM :DYNSTMT END-EXEC.
           EXEC SQL DECLARE C1 CURSOR FOR STMT2 END-EXEC.
           EXEC SQL OPEN C1 USING :DEPT-NO END-EXEC.
           EXEC SQL FETCH C1 INTO :EMP-NAME END-EXEC.
```

```
           ...
           EXEC SQL CLOSE C1 END-EXEC.
```

> **i** À vérifier
>
> `OPEN ... USING` sur un curseur déclaré sur une instruction préparée.

9.4 Marqueurs de paramètres

Chaque base de données a sa propre syntaxe pour les paramètres :

- PostgreSQL : `$1`, `$2`…
- Oracle : `:name` ou `:1`, `:2`…
- ODBC : `?`
- SQLite : n'importe laquelle de ces syntaxes.

Pour les instructions statiques, le précompilateur génère les marqueurs choisis avec `-z` / `--param-style` : `d` pour `$1` (par défaut), `a` pour `?`, `c` pour `:1`. Le pilote Oracle accepte aussi les marqueurs `$n`.

Pour les instructions préparées, le texte est écrit par le programme. GixSQL peut convertir les marqueurs d'une instruction préparée en ceux de la base de données, avec l'option de connexion `fixup_params=on` ou la variable d'environnement `GIXSQL_FIXUP_PARAMS=on`.

10 Le précompilateur gixpp

10.1 Synopsis

```
gixpp -e [options] -i <input> -o <output>
```

`gixpp` lit un source COBOL contenant du SQL embarqué et écrit un source COBOL dans lequel les blocs `EXEC SQL` sont mis en commentaire et remplacés par des appels à `libgixsql`.

10.2 Options

10.2.1 Entrée et sortie

`-i FILE`, `--infile FILE`
: le fichier d'entrée.

`-o FILE`, `--outfile FILE`
: le fichier de sortie.

`-I DIRS`, `--copypath DIRS`
: les répertoires dans lesquels les copybooks sont recherchés (pour `EXEC SQL INCLUDE` et, avec `-p`, `COPY`). L'option peut être répétée.

`-E EXTS`, `--esql-copy-exts EXTS`
: les extensions des copybooks, séparées par des virgules, par exemple `-E ".,cpy,CPY"`.

`-c`, `--consolidate`
: écrire un unique fichier source, avec les copybooks développés.

`-k`, `--keep`
: conserver les fichiers temporaires.

`-s FILE`, `--symfile FILE`
: écrire un fichier de symboles.

`-m`, `--map`
: écrire un fichier de correspondance (entre les lignes d'entrée et de sortie).

10.2.2 Format des sources

`--code-end-col N`
: la dernière colonne de la zone de code en format fixe, de 72 à 512 (72 par défaut). Le texte situé après cette colonne est ignoré, avec un avertissement. Utilisez `--code-end-col 512` pour les sources écrits pour Pro*COBOL avec `format=terminal ireclen=512`.

> **À compléter**
>
> Prise en charge du format libre.

10.2.3 Traitement ESQL

`-e`, `--esql`
: traiter les instructions SQL embarquées. Cette option est obligatoire.

`-p`, `--esql-preprocess-copy`
: traiter les instructions SQL embarquées dans tous les copybooks inclus avec `COPY`, et pas seulement ceux inclus avec `EXEC SQL INCLUDE`.

`-S`, `--esql-static-calls`
: générer des appels statiques à la bibliothèque d'exécution (`CALL STATIC`). Recommandé.

`-z STYLE`, `--param-style STYLE`
: le style des marqueurs de paramètres dans les instructions SQL générées : `d` pour `$1` (par défaut), `a` pour `?`, `c` pour `:1`. Voir Marqueurs de paramètres.

`-P MODE`, `--picx-as MODE`
: la manière dont les variables hôtes `PIC X` sont envoyées : `char` (par défaut), `charf` ou `varchar` (voir Chaînes de caractères de longueur fixe).

`-Y LEN,ARR`, `--varying LEN,ARR`
: les suffixes des éléments de longueur et de données des groupes `VARYING` (`LEN,ARR` par défaut).

`-N SIZE`, `--varying-length-size SIZE`
: la taille en octets des éléments de longueur des groupes `VARYING` : `4` (par défaut) ou `2` (Pro*COBOL).

`--no-rec-code CODE`
: le `SQLCODE` de la condition « aucun enregistrement », testé par `WHENEVER NOT FOUND` (100 par défaut ; 1403 pour Oracle). Voir La condition « aucun enregistrement ».

`-C`, `--cobol85`
: générer du code COBOL 85.

> **À vérifier**
>
> Ce que `--cobol85` change dans le code généré.

10.2.4 Débogage

`-g`, `--debug-info`
: générer des informations de débogage.

`-v`, `--verbose`
: sortie détaillée.

`-d`, `--verbose-debug`
: sortie de débogage.

`-D`, `--parser-scanner-debug`
: tracer l'analyseur syntaxique et l'analyseur lexical.

10.2.5 Information

`-h`, `--help`
: afficher les options.

`-V`, `--version`
: afficher la version.

10.3 Code généré

> **i** À compléter
>
> Montrer le code généré pour un `SELECT INTO` : l'instruction mise en commentaire, la séquence `CALL "GIXSQLExec..."`, et les données `SQ0001` générées contenant le texte SQL.

11 Configuration de l'exécution

La bibliothèque d'exécution `libgixsql` est configurée par des variables d'environnement. Certaines d'entre elles peuvent aussi être définies par connexion, avec une option de la chaîne de connexion (Options de connexion), qui est alors prioritaire.

11.1 Variables d'environnement

Variable	Défaut	Description
`GIXSQL_DEFAULT_DRIVER`		pilote utilisé lorsque la chaîne de connexion n'en nomme aucun (Le pilote par défaut)
`GIXSQL_AUTOCOMMIT`	`native`	`on`, `off` ou `native` (Autocommit)
`GIXSQL_CLIENT_ENCODING`		encodage du client (PostgreSQL, MySQL)
`GIXSQL_FIXUP_PARAMS`	`off`	convertir les marqueurs de paramètres des instructions préparées (Marqueurs de paramètres)
`GIXSQL_NOREC_CODE`	`100`	`SQLCODE` de la condition « aucun enregistrement » (La condition « aucun enregistrement »)
`GIXSQL_VARYING_LEN_SZ_SHORT`	`0`	`1` si les champs de longueur `VARYING` font 2 octets ; inutile pour les programmes précompilés avec `-N 2`
`GIXSQL_LOG_LEVEL`	`error`	voir Journalisation
`GIXSQL_LOG_FILE`	`gixsql.log`	voir Journalisation
`GIXSQL_LOG_TRUNCATE`	`off`	voir Journalisation
`GIXSQL_LOG_ROTATE`	`off`	voir Journalisation
`GIXSQL_LOG_ROTATE_MAX_SIZE`	`10M`	voir Journalisation
`GIXSQL_LOG_ROTATE_MAX_FILES`	`5`	voir Journalisation
`GIXSQL_LOG_ROTATE_ON_OPEN`	`on`	voir Journalisation

Les variables booléennes acceptent `on`/`off` et `1`/`0`.

⚠ Avertissement

`GIXSQL_USRPWD_SEP` est lue par la bibliothèque d'exécution mais n'a aucun effet dans la version actuelle : le séparateur entre le nom d'utilisateur et le mot de passe est toujours un point.

Les pilotes utilisent aussi les variables de leurs bibliothèques clientes, par exemple `PGHOST` ou `PGSSLMODE` pour PostgreSQL, `NLS_LANG` et `TNS_ADMIN` pour Oracle, `ODBCINI` pour unixODBC.

11.2 Trouver les pilotes

`libgixsql` charge le pilote `libgixsql-<driver>.so` (`libgixsql-<driver>.dll` sous Windows) avec le chargeur dynamique du système. Le répertoire des pilotes doit se trouver dans `LD_LIBRARY_PATH` (`PATH` sous Windows), ou dans le chemin des bibliothèques du système.

11.3 Journalisation

La bibliothèque d'exécution écrit un journal, avec spdlog.

`GIXSQL_LOG_LEVEL`
: `off`, `critical`, `error` (par défaut), `warn`, `info`, `debug` ou `trace`. Le niveau `trace` journalise chaque appel, avec les valeurs des variables hôtes (y compris les mots de passe et autres données sensibles), et ralentit les programmes.

`GIXSQL_LOG_FILE`
: le fichier journal (`gixsql.log` par défaut, dans le répertoire courant).

`GIXSQL_LOG_TRUNCATE`
: `on` pour tronquer le fichier journal au démarrage du programme.

`GIXSQL_LOG_ROTATE`
: `on` pour effectuer la rotation des fichiers journaux.

`GIXSQL_LOG_ROTATE_MAX_SIZE`
: la taille maximale d'un fichier journal, avec un suffixe facultatif `B`, `K`, `M` (par défaut) ou `G` (`10M` par défaut).

`GIXSQL_LOG_ROTATE_MAX_FILES`
: le nombre de fichiers conservés (5 par défaut).

`GIXSQL_LOG_ROTATE_ON_OPEN`
: `on` (par défaut) pour effectuer la rotation au démarrage du programme.

Par exemple, pour tracer un programme :

```
$ GIXSQL_LOG_LEVEL=trace GIXSQL_LOG_FILE=/tmp/prog.log ./prog
```

12 Migrer depuis Pro*COBOL

Ce chapitre explique comment précompiler et exécuter avec GixSQL des programmes écrits pour Oracle Pro*COBOL, sur une base de données Oracle ou sur PostgreSQL.

12.1 Réglages recommandés

Réglage	Où	Pourquoi
`-N 2`	gixpp	éléments `VARYING` avec une longueur sur 2 octets (`PIC S9(4) COMP`), comme Pro*COBOL
`--no-rec-code 1403`	gixpp	`SQLCODE` de la condition « aucun enregistrement », comme Oracle, pour `WHENEVER NOT FOUND`
`GIXSQL_NOREC_CODE=1403`	environnement	la même chose, à l'exécution
`GIXSQL_AUTOCOMMIT=off`	environnement	les transactions sont explicites avec Pro*COBOL ; sur PostgreSQL, les curseurs nécessitent une transaction
`--code-end-col 512`	gixpp	sources précompilés par `procob` avec `format=terminal ireclen=512`, avec du code après la colonne 72

Une compilation typique est :

```
$ gixpp -e -S -N 2 --no-rec-code 1403 --code-end-col 512 \
        -I /opt/gixsql/share/gixsql/copy -I copy -E ".,cpy,CPY" \
        -i prog.pco -o prog.cob
$ cobc -x prog.cob -L /opt/gixsql/lib -lgixsql
$ export GIXSQL_NOREC_CODE=1403 GIXSQL_AUTOCOMMIT=off
$ export DATASRC=oracle://dbserver:1521/ORCLPDB1
$ ./prog
```

12.2 Connexion

Les programmes Pro*COBOL se connectent avec `CONNECT :USR IDENTIFIED BY :PWD`, éventuellement `USING :DB-STRING` avec un alias SQL*Net. Avec GixSQL, l'opérande de `USING` est une chaîne de connexion GixSQL (Chaînes de connexion), et non un alias Oracle.

> **i** À vérifier
>
> Comportement de `CONNECT ... IDENTIFIED BY` sans `USING` (le mode 4 de la grammaire est « unsupported »).

`COMMIT WORK RELEASE` et `ROLLBACK WORK RELEASE` terminent la transaction et ferment la connexion.

12.3 Exécution sur PostgreSQL

Les programmes écrits pour Oracle peuvent s'exécuter sur PostgreSQL, à condition que leur SQL soit compatible. L'extension orafce fournit de nombreuses fonctions Oracle (`NVL`, `DECODE`, `TO_DATE`…). Son schéma `oracle` doit précéder `pg_catalog` dans le chemin de recherche, afin que `oracle.to_date` soit utilisée à la place de la fonction `to_date` de PostgreSQL :

```
pgsql://dbserver/db?default_schema=myschema,oracle,pg_catalog,public
```

> **i** À compléter
>
> Différences SQL courantes : `ROWNUM`, jointures externes `(+)`, `SYSDATE`, chaînes vides et `NULL`, `DUAL`, séquences (`seq.NEXTVAL`).

12.4 Comparaison des fonctionnalités

Le tableau suit les chapitres du *Pro*COBOL Programmer's Guide*.

Fonctionnalité Pro*COBOL	GixSQL
Section de déclaration	acceptée, non obligatoire
`EXEC SQL INCLUDE`	pris en charge
Précompilation conditionnelle (`EXEC ORACLE IFDEF`)	non prise en charge
`CONNECT ... IDENTIFIED BY`, `AT`	pris en charge
`CONNECT :USR-PWD` (`user/password` dans un seul élément)	non pris en charge
`COMMIT`, `ROLLBACK`, `SAVEPOINT`, `RELEASE`	pris en charge
`SET TRANSACTION`, `LOCK TABLE`	transmis à la base de données
Variables hôtes, variables indicatrices	prises en charge
Variables `VARCHAR` / `VARYING` (longueur sur 2 octets)	prises en charge avec `-N 2`
Équivalence de types de données (`EXEC SQL VAR`)	partielle : voir Variables hôtes et types de données
`EXEC SQL TYPE`	non pris en charge
`ROWID` universel	lu sous forme de chaînes de caractères (Oracle)
`SELECT INTO`, `INSERT`, `UPDATE`, `DELETE`	pris en charge
Curseurs, `WHERE CURRENT OF`	pris en charge
Curseurs défilants (`FETCH PRIOR`, `ABSOLUTE`…)	non pris en charge
Blocs PL/SQL embarqués (`EXEC SQL EXECUTE BEGIN ... END; END-EXEC`)	non pris en charge
Tables hôtes (tableaux)	non prises en charge
SQLCA	prise en charge
ORACA	non prise en charge
`WHENEVER` (`SQLERROR`, `SQLWARNING`, `NOT FOUND`)	pris en charge
`WHENEVER ... DO`, `STOP`	non pris en charge
SQL dynamique, méthodes 1, 2, 3	pris en charge
SQL dynamique méthode 4 (SQLDA), SQL dynamique ANSI (descripteurs)	non pris en charge
Contextes d’exécution, multithreading	non pris en charge

Fonctionnalité Pro*COBOL	GixSQL
Instructions LOB	non prises en charge
Options du précompilateur	voir la correspondance ci-dessous

> **i** À vérifier
>
> Chaque ligne « non pris en charge » ci-dessus, en particulier `SET TRANSACTION`, `EXEC SQL TYPE`, les curseurs défilants, `WHENEVER ... DO`, et la transmission générique des instructions inconnues.

12.5 Options du précompilateur

Option de `procob`	Équivalent `gixpp`
`INAME`, `ONAME`	`-i`, `-o`
`INCLUDE`	`-I`
`FORMAT=TERMINAL`, `IRECLEN`	`--code-end-col`
`MODE=ANSI` (`SQLCODE` 100 sur « aucun enregistrement »)	par défaut
`MODE=ORACLE` (`SQLCODE` 1403)	`--no-rec-code 1403`
`PICX=VARCHAR2`	`-P varchar`
`PICX=CHARF`	`-P charf` (par défaut)
`VARCHAR=YES`	les groupes de niveau 49 sont toujours reconnus
`SQLCHECK`, `USERID`	pas d'équivalent : aucune vérification sémantique
`RELEASE_CURSOR`, `HOLD_CURSOR`, `MAXOPENCURSORS`	pas d'équivalent

> **i** À vérifier
>
> Vérifier `MODE` et `PICX` par rapport à la documentation de Pro*COBOL.

13 Vue d'ensemble des backends

Un **backend** (ou pilote) est une bibliothèque `libgixsql-<driver>` qui implémente l'accès à un type de base de données. `libgixsql` la charge lorsqu'un programme se connecte, en fonction du préfixe de la chaîne de connexion. Un programme peut utiliser plusieurs backends en même temps, avec plusieurs connexions.

Pilote	Base de données	Chaîne de connexion	Chapitre
`pgsql`	PostgreSQL	`pgsql://host[:port]/database`	PostgreSQL
`oracle`	Oracle Database	`oracle://host[:port]/service`	Oracle
`mysql`	MySQL, MariaDB	`mysql://host[:port]/database`	MySQL et MariaDB
`odbc`	toute source de données ODBC	`odbc://dsn`	ODBC
`sqlite`	SQLite	`sqlite://path`	SQLite

13.1 Fonctionnalités prises en charge

Fonctionnalité	pgsql	oracle	mysql	odbc	sqlite
Autocommit `native`	autocommit	transactions	autocommit	pilote	?
Curseurs natifs	oui (par défaut)	oui	émulés	pilote	émulés
Curseurs modifiables	curseurs natifs	oui	émulés	pilote	émulés
`client_encoding`	oui	non (`NLS_LANG`)	oui	?	non
Marqueur de paramètre par défaut	`$1`	`:1` (accepte `$1`)	`?`	`?`	tous

> **i** À vérifier
>
> Chaque cellule de ce tableau, et les cellules marquées « ? ».

13.2 Choisir le style des marqueurs de paramètres

Les instructions SQL du programme sont générées avec le style de marqueurs de paramètres choisi lors de la précompilation (`-z`, Marqueurs de paramètres). Il doit être accepté par le backend utilisé à l'exécution :

- `-z d` (par défaut) : PostgreSQL, Oracle, SQLite ;
- `-z a` : ODBC, MySQL, SQLite ;
- `-z c` : Oracle, SQLite.

14 PostgreSQL

14.1 Prérequis

- Compilation : les fichiers de développement de `libpq` (`libpq-dev`), trouvés avec `pkg-config`.
- Exécution : `libpq`.

14.2 Chaîne de connexion

```
pgsql://[user.password@]host[:port]/database[?options]
```

Exemple :

```
pgsql://localhost:5432/demo?default_schema=app&autocommit=off
```

Le nom d'utilisateur et le mot de passe sont généralement donnés dans l'instruction `CONNECT`. Les variables d'environnement de libpq (`PGHOST`, `PGPORT`, `PGPASSWORD`, `PGSSLMODE`…) et le fichier `~/.pgpass` sont pris en compte par libpq pour ce que la chaîne de connexion ne précise pas.

14.3 Options

`default_schema`
: les schémas du `search_path`, séparés par des virgules.

`client_encoding`
: l'encodage du client, par exemple `UTF8` ou `LATIN1`.

`native_cursors`
: `on` (par défaut) pour utiliser les curseurs de PostgreSQL (`DECLARE ... CURSOR`, `FETCH`) ; `off` pour exécuter la requête à l'ouverture du curseur et conserver toutes les lignes en mémoire.

`decode_binary`
: `on` (par défaut) pour décoder les valeurs `bytea` ; `off` pour les recevoir dans leur encodage textuel (plus volumineux).

Options de libpq
: `hostaddr`, `connect_timeout`, `application_name`, `keepalives`, `keepalives_idle`, `keepalives_interval`, `keepalives_count`, `sslmode`, `requiressl`, `sslcert`, `sslkey`, `sslrootcert`, `sslcrl`, `krbsrvname`, `gsslib`, `service` sont transmises à libpq. Voir la documentation de libpq.

14.4 Curseurs et transactions

Les curseurs natifs n'existent qu'à l'intérieur d'une transaction. Avec le mode autocommit par défaut, chaque instruction constitue sa propre transaction, et `OPEN` échoue. Il faut donc :

- démarrer une transaction avant d'ouvrir le curseur (`EXEC SQL START TRANSACTION END-EXEC`), ou
- se connecter avec `autocommit=off` (ou `GIXSQL_AUTOCOMMIT=off`), ou
- utiliser des curseurs émulés (`native_cursors=off`).

14.5 Exécuter des programmes Oracle

Voir Migrer depuis Pro*COBOL pour l'extension orafce et le chemin de recherche.

14.6 Tests

> **i** À compléter
>
> Comment mettre en place une base de données de test (image Docker, orafce).

15 Oracle

15.1 Prérequis

- Compilation : rien ; le pilote inclut ODPI-C.
- Exécution : les bibliothèques clientes d'Oracle (`libclntsh.so`), par exemple celles de Oracle Instant Client. ODPI-C les charge à l'exécution : leur répertoire doit se trouver dans `LD_LIBRARY_PATH` (ou définissez `ORACLE_HOME`).

15.2 Chaîne de connexion

```
oracle://[user.password@]host[:port]/service
```

Le pilote se connecte avec la syntaxe Easy Connect `//host:port/service`. Seuls les noms de service sont pris en charge, ni les SID ni les alias de `tnsnames.ora`.

Exemple, avec le conteneur Oracle Free :

```
oracle://localhost:1521/FREEPDB1
```

15.3 Options et environnement

- `client_encoding` est ignorée : définissez `NLS_LANG` avant d'exécuter le programme.
- Les variables `DPI_DEBUG_LEVEL` et `DPI_DEBUG_PREFIX` activent les traces d'ODPI-C.

15.4 Transactions

Oracle travaille toujours dans une transaction : avec le mode autocommit `native` par défaut, un programme doit valider ses modifications (`COMMIT`), comme avec Pro*COBOL.

15.5 Types de données

- Les colonnes `ROWID` sont lues sous forme de chaînes de caractères.
- Les paramètres peuvent s'écrire `:1`, `:name`, ou `$1` (converti par le pilote).

> **À compléter**
>
> Conversions `DATE`/`TIMESTAMP`, précision de `NUMBER`, `CLOB`/`BLOB`.

15.6 Tests

> **i** À compléter
>
> Comment exécuter le conteneur Oracle Free (`gvenzl/oracle-free`) et créer un utilisateur de test.

16 MySQL et MariaDB

16.1 Prérequis

- Compilation : les fichiers de développement de `libmysqlclient` ou de `libmariadb`, trouvés avec `pkg-config` (`mysqlclient` est essayé en premier, puis `libmariadb`).
- Exécution : la bibliothèque cliente correspondante.

16.2 Chaîne de connexion

```
mysql://[user.password@]host[:port]/database[?options]
```

Exemple :

```
mysql://localhost:3306/demo?client_encoding=utf8mb4
```

16.3 Options

`client_encoding`
: le jeu de caractères de la connexion, par exemple `utf8mb4`.

`autocommit`
: transmise à la bibliothèque cliente (`mysql_autocommit`).

`updatable_cursors`
: émuler les curseurs modifiables ; la table doit avoir une clé unique.

> **ℹ À vérifier**
>
> `updatable_cursors` est-elle lue par le pilote MySQL (elle l'est par le pilote SQLite) ?

16.4 Marqueurs de paramètres

MySQL n'accepte que les marqueurs de paramètres `?` : précompilez avec `-z a`, ou connectez-vous avec `fixup_params=on`.

> **ℹ À vérifier**
>
> Si le pilote MySQL convertit lui-même les marqueurs de paramètres `$n`.

17 ODBC

17.1 Prérequis

- Compilation : les fichiers de développement de unixODBC (`unixodbc-dev`), trouvés avec `pkg-config`.
- Exécution : unixODBC, et le pilote ODBC de la base de données (psqlODBC, MySQL Connector/ODBC, pilote IBM Db2…), configuré dans `odbcinst.ini`.

17.2 Chaîne de connexion

```
odbc://dsn
```

`dsn` est le nom d'une source de données définie dans `odbc.ini` (ou `~/.odbc.ini`) ; le pilote se connecte avec `SQLConnect`, en utilisant le nom d'utilisateur et le mot de passe de l'instruction `CONNECT`.

Exemple de fichier `~/.odbc.ini` :

```
[demo]
Driver     = PostgreSQL Unicode
Servername = localhost
Port       = 5432
Database   = demo
```

17.3 Comportement

La plupart des fonctionnalités dépendent du pilote ODBC : autocommit, curseurs modifiables, conversions de types de données.

- ODBC n'accepte que les marqueurs de paramètres `?` : précompilez avec `-z a`.
- Avec psqlODBC, réglez l'option « rollback on error » sur `nop` (`Protocol = 7.4-0`), afin que ce soit GixSQL, et non le pilote, qui gère les erreurs.

> **i** À vérifier
>
> Le réglage de psqlODBC pour « rollback on error ».

18 SQLite

18.1 Prérequis

Aucun : SQLite est intégré au pilote.

18.2 Chaîne de connexion

La chaîne de connexion donne le chemin du fichier de base de données :

```
sqlite:///home/user/demo.db
sqlite://c:/Users/user/demo.db
```

Le fichier est créé s'il n'existe pas. Le nom d'utilisateur et le mot de passe sont ignorés.

18.3 Options

`updatable_cursors`
: émuler les curseurs modifiables (`WHERE CURRENT OF`) ; la table doit avoir une clé unique.

18.4 Comportement

- SQLite accepte les marqueurs de paramètres `$1`, `?` et `:1`.
- Les types de SQLite sont dynamiques : la conversion est guidée par le type de la variable hôte.

> **i** À vérifier
>
> Comportement de l'autocommit et des curseurs du pilote SQLite.

19 Récapitulatif des instructions SQL embarquées

Cette annexe récapitule les instructions reconnues par `gixpp`. Chaque instruction est écrite entre `EXEC SQL` et `END-EXEC`. Les instructions marquées `[AT conn]` acceptent la clause `AT` pour choisir une connexion.

Instruction	Description	Voir
`BEGIN DECLARE SECTION`, `END DECLARE SECTION`	délimiter les variables hôtes (facultatif)	Concepts du précompilateur
`INCLUDE name`	inclure un copybook (`SQLCA`…)	Concepts du précompilateur
`VAR item IS type`	donner un type SQL à une variable hôte	Variables hôtes et types de données
`DECLARE name CURSOR [WITH HOLD] FOR select`	déclarer un curseur	Instructions SQL embarquées
`DECLARE name CURSOR [WITH HOLD] FOR stmt`	déclarer un curseur sur une instruction préparée	SQL dynamique
`DECLARE name TABLE (...)`	déclarer une table (ignoré)	
`CONNECT TO src [AS conn] [USER usr [USING pwd]]`	se connecter	Connexions et transactions
`CONNECT usr IDENTIFIED BY pwd [AT conn] USING src`	se connecter	Connexions et transactions
`CONNECT USING src`	se connecter	Connexions et transactions
`CONNECT RESET [conn]`, `DISCONNECT [conn | ALL]`	se déconnecter	Connexions et transactions
`[AT conn] COMMIT [WORK] [RELEASE]`	valider (COMMIT)	Connexions et transactions
`[AT conn] ROLLBACK [WORK] [RELEASE]`	annuler (ROLLBACK)	Connexions et transactions
`[AT conn] ROLLBACK [WORK] TO SAVEPOINT name`	annuler jusqu’à un point de sauvegarde	Connexions et transactions

Instruction	Description	Voir
`[AT conn] SELECT ... INTO ...`	requête ne renvoyant qu'une ligne	Instructions SQL embarquées
`[AT conn] INSERT, UPDATE, DELETE`	manipulation de données	Instructions SQL embarquées
`UPDATE, DELETE ... WHERE CURRENT OF cursor`	modification/suppression positionnée	Instructions SQL embarquées
`[AT conn] OPEN cursor [USING ...]`	ouvrir un curseur	Instructions SQL embarquées
`FETCH cursor INTO ...`	lire une ligne	Instructions SQL embarquées
`CLOSE cursor`	fermer un curseur	Instructions SQL embarquées
`[AT conn] PREPARE stmt FROM src`	préparer une instruction	SQL dynamique
`[AT conn] EXECUTE stmt [INTO ...] [USING ...]`	exécuter une instruction préparée	SQL dynamique
`[AT conn] EXECUTE IMMEDIATE src`	exécuter une instruction dynamique	SQL dynamique
`WHENEVER condition action`	gestion des erreurs	Gestion des erreurs et diagnostics
`IGNORE ...`	ignoré par le précompilateur	
autres instructions	transmises telles quelles à la base de données	Instructions SQL embarquées

i À vérifier

Chaque ligne de ce tableau par rapport à `libgixpp/gix_esql_parser.yy`, en particulier `DISCONNECT`, `DECLARE TABLE`, `IGNORE`, et les instructions qui acceptent `AT`.

20 Codes d’erreur

Lorsqu’une opération échoue pour une raison interne à GixSQL (contrôle de cohérence, fonctionnalité non prise en charge, problème de pilote…), `SQLCODE` reçoit l’un des codes négatifs suivants. Le journal donne plus de détails (Journalisation).

Nom	`SQLCODE`	Description
`DBERR_NO_ERROR`	0	pas d’erreur
`DBERR_CONNECTION_FAILED`	−100	la connexion à la base de données a échoué
`DBERR_BEGIN_TX_FAILED`	−101	une transaction n’a pas pu être démarrée
`DBERR_END_TX_FAILED`	−102	une transaction n’a pas pu être terminée
`DBERR_CONN_NOT_FOUND`	−103	connexion introuvable
`DBERR_CONN_RESET_FAILED`	−104	la connexion n’a pas pu être fermée
`DBERR_EMPTY_QUERY`	−105	requête vide
`DBERR_SQL_ERROR`	−106	erreur SQL ou de pilote générique
`DBERR_TOO_MANY_ARGUMENTS`	−107	trop d’arguments
`DBERR_TOO_FEW_ARGUMENTS`	−108	pas assez d’arguments
`DBERR_NO_PARAMETERS`	−109	des paramètres étaient attendus mais n’ont pas été fournis
`DBERR_CURSOR_EXISTS`	−110	le curseur existe déjà
`DBERR_NO_SUCH_CURSOR`	−111	curseur inexistant
`DBERR_CLOSE_CURSOR_FAILED`	−112	le curseur n’a pas pu être fermé
`DBERR_DISCONNECT_FAILED`	−113	impossible de se déconnecter
`DBERR_OUT_OF_MEMORY`	−114	mémoire insuffisante
`DBERR_DECLARE_CURSOR_FAILED`	−115	la déclaration du curseur a échoué
`DBERR_OPEN_CURSOR_FAILED`	−116	le curseur n’a pas pu être ouvert
`DBERR_FETCH_ROW_FAILED`	−117	impossible de lire une ligne
`DBERR_INVALID_COLUMN_DATA`	−118	données de colonne invalides
`DBERR_CURSOR_CLOSED`	−119	le curseur est fermé

Nom	SQLCODE	Description
DBERR_MOVE_TO_FIRST_FAILED	−120	impossible de se placer sur la première ligne d'un ensemble de résultats
DBERR_FIELD_COUNT_MISMATCH	−121	le nombre de colonnes ne correspond pas à la requête
DBERR_NO_DATA	−122	aucune ligne alors que des lignes étaient attendues
DBERR_TOO_MUCH_DATA	−123	plus de lignes que prévu
DBERR_PREPARE_FAILED	−124	l'instruction n'a pas pu être préparée
DBERR_CONN_INIT_ERROR	−201	erreur d'initialisation de la connexion
DBERR_CONN_INVALID_DBTYPE	−202	type de base de données invalide
DBERR_NUM_OUT_OF_RANGE	−410	valeur numérique hors limites
DBERR_INTERNAL_ERR	−8880	erreur interne
DBERR_NOT_IMPL	−9990	fonctionnalité non implémentée