GixSQL User Manual

Embedded SQL for GnuCOBOL — SuperBOL edition

The SuperBOL Team at Titagone-OCamlPro

November 12th, 2026

Generated on 2026-11-18 from commit superbol-manuals@a5e6f46

Table of contents

1 Preface 5
1.1 Who should read this manual 5
1.2 How this manual is organized 5
1.3 Conventions 6
2 Introduction 7
2.1 What is GixSQL? 7
2.2 Components 7
2.3 Development workflow 8
2.4 Relation with other precompilers 8
2.5 Feature summary 8
3 Installation 9
3.1 Requirements 9
3.2 Building from the git repository 10
3.3 Configuration options 10
3.4 Installed files 11
3.5 Building with Nix 11
3.6 Building on Windows 11
3.6.1 Visual Studio 11
3.6.2 MSYS2/MinGW 12
3.7 Checking the installation 12
4 Precompiler concepts 13
4.1 Embedded SQL in a COBOL program 13
4.2 The declare section 13
4.3 Including files 14
4.4 Precompiling, compiling and linking 14
4.5 A first program 14
4.6 Sample tables 15
5 Connections and transactions 17
5.1 Connecting to a database 17
5.2 Connection strings 17
5.2.1 The default driver 18
5.2.2 Connection options 18
5.3 Multiple connections 18
5.4 Disconnecting 19
5.5 Transactions 19

5.5.1 Autocommit 20
6 Host variables and datatypes 21
6.1 Host variables 21
6.2 Datatype conversion 21
6.3 Fixed-length character strings 22
6.4 Variable-length strings: VARYING 22
6.5 SQL TYPE IS 22
6.6 EXEC SQL VAR 23
6.7 Indicator variables 23
6.8 Host tables (arrays) 24
7 Embedded SQL statements 25
7.1 Executing SQL statements 25
7.2 Single-row queries: SELECT INTO 25
7.3 Cursors 26
7.3.1 Updatable cursors 27
7.3.2 Cursors on prepared statements 27
7.4 Transactions 27
7.5 Other statements 27
8 Error handling and diagnostics 28
8.1 The SQL Communication Area 28
8.2 SQLCODE and SQLSTATE 28
8.3 The “no record” condition 28
8.4 WHENEVER 29
8.5 Logging 29
9 Dynamic SQL 30
9.1 Method 1: EXECUTE IMMEDIATE 30
9.2 Method 2: PREPARE and EXECUTE 30
9.3 Method 3: prepared queries and cursors 30
9.4 Parameter placeholders 31
10 The gixpp precompiler 32
10.1 Synopsis 32
10.2 Options 32
10.2.1 Input and output 32
10.2.2 Source format 32
10.2.3 ESQL processing 33
10.2.4 Debugging 33
10.2.5 Information 34
10.3 Generated code 34
11 Runtime configuration 35

11.1 Environment variables 35
11.2 Finding the drivers 36
11.3 Logging 36
12 Migrating from Pro*COBOL 37
12.1 Recommended settings 37
12.2 Connecting 38
12.3 Running on PostgreSQL 38
12.4 Feature comparison 38
12.5 Precompiler options 40
13 Overview of the backends 41
13.1 Feature support 41
13.2 Choosing the placeholder style 41
14 PostgreSQL 43
14.1 Requirements 43
14.2 Connection string 43
14.3 Options 43
14.4 Cursors and transactions 43
14.5 Running Oracle programs 44
14.6 Testing 44
15 Oracle 45
15.1 Requirements 45
15.2 Connection string 45
15.3 Options and environment 45
15.4 Transactions 45
15.5 Data types 45
15.6 Testing 46
16 MySQL and MariaDB 47
16.1 Requirements 47
16.2 Connection string 47
16.3 Options 47
16.4 Placeholders 47
17 ODBC 48
17.1 Requirements 48
17.2 Connection string 48
17.3 Behavior 48
18 SQLite 49
18.1 Requirements 49
18.2 Connection string 49
18.3 Options 49

18.4 Behavior 49
19 Embedded SQL statements 50
20 Error codes 52

This manual is available:

- As an HTML website: in English or in French
- As a PDF File: in English or in French

Generated on 2026-11-18 from commit superbol-manuals@a5e6f46

1 Preface

This manual describes **GixSQL**, an embedded SQL (ESQL) precompiler and runtime library for GnuCOBOL. It covers the SuperBOL edition of GixSQL, maintained at https://github.com/superbol4gnucobol/gixsql-superbol, which continues the original work of Marco Ridoni (https://github.com/mridoni/gixsql).

1.1 Who should read this manual

This manual is for COBOL programmers who write programs that embed SQL statements (`EXEC SQL ... END-EXEC`), and for people who port programs written for other precompilers, in particular Oracle Pro*COBOL, to GnuCOBOL. Some knowledge of COBOL and SQL is assumed.

1.2 How this manual is organized

The organization follows that of most ESQL precompiler manuals, so that readers who know Pro*COBOL can find their way:

- **Getting started**: what GixSQL is, how to install it, and how to precompile, compile and run a first program.
- **Programming with embedded SQL**: connections and transactions, host variables and datatypes, the SQL statements, error handling and dynamic SQL.
- **Reference**: the options of the `gixpp` precompiler, the runtime configuration, and a guide to migrate Pro*COBOL programs.
- **Database backends**: the notes for each database driver.
- **Appendices**: a summary of the embedded SQL statements, and the error codes.

1.3 Conventions

- `gixpp` options are written as `--long-option` (with their short form `-x` when there is one).
- Environment variables are written in capitals: `GIXSQL_LOG_LEVEL`.
- COBOL examples use the fixed source format: the code starts at column 8.

2 Introduction

2.1 What is GixSQL?

GixSQL lets COBOL programs compiled with GnuCOBOL access relational databases through embedded SQL. A program contains SQL statements in `EXEC SQL ... END-EXEC` blocks:

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

These blocks are not COBOL: a **precompiler** translates them into calls to a **runtime library**, which sends the statements to the database and moves the data between the database and the COBOL fields.

2.2 Components

GixSQL consists of:

`gixpp`
: the precompiler. It reads a COBOL source with embedded SQL and writes a pure COBOL source, where each `EXEC SQL` block is commented and replaced by calls to the runtime library (see The gixpp precompiler).

`libgixsql`
: the runtime library, linked with the COBOL program. It manages the connections, cursors and host variables, and loads the database driver at run time.

`libgixsql-<driver>`
: the database drivers, loaded dynamically by `libgixsql` according to the connection string (see Overview of the backends):

Driver	Database	Client library
`pgsql`	PostgreSQL	`libpq`
`oracle`	Oracle Database	ODPI-C (built in)
`mysql`	MySQL, MariaDB	`libmariadb`/`libmysqlclient`
`odbc`	any database with an ODBC driver	unixODBC
`sqlite`	SQLite	built in

`SQLCA` copybook
: the SQL Communication Area, installed with the runtime (`share/gixsql/copy`), and included in programs with `EXEC SQL INCLUDE SQLCA END-EXEC`.

2.3 Development workflow

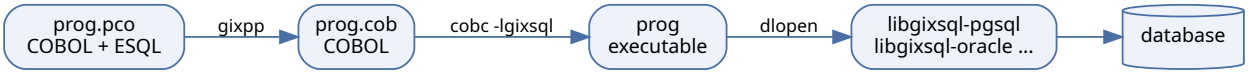


Figure 1: Development workflow

1. Write the program with embedded SQL.
2. Precompile it with `gixpp`.
3. Compile and link the result with `cobc`, adding `libgixsql`.
4. Run it, with the database driver libraries in the library path.

Precompiler concepts details each step with a complete example.

2.4 Relation with other precompilers

GixSQL started as a fork of Open-COBOL-ESQL (OCESQL), and was largely rewritten in C++. The SuperBOL edition adds compatibility with programs written for Oracle Pro*COBOL (see Migrating from Pro*COBOL).

Other open-source precompilers for GnuCOBOL are:

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

2.5 Feature summary

The features of GixSQL are compared with those of Pro*COBOL in Feature comparison, and the support of each backend is summarized in Overview of the backends.

3 Installation

GixSQL is distributed as source code. This chapter explains how to build and install it on GNU/Linux. Building for Windows (Visual Studio or MSYS2/MinGW) is described in Building on Windows.

3.1 Requirements

To build GixSQL, you need:

- a C++17 compiler (GCC 8 or later, or Clang);
- the autotools (`autoconf`, `automake`, `libtool`) and `pkg-config`, when building from the git repository;
- `bison` 3.7 or later and `flex`;
- the development package of spdlog (and of `fmt`, its dependency);
- for each database driver you want, the development package of its client library:

Driver	Debian/Ubuntu package	Notes
`pgsql`	`libpq-dev`	
`mysql`	`libmariadb-dev` (or `libmysqlclient-dev`)	
`odbc`	`unixodbc-dev`	plus the ODBC driver of the database
`oracle`	none to build	the Oracle client libraries (Instant Client) are needed at run time
`sqlite`	none	SQLite is built in

On Debian or Ubuntu:

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

To run the programs, you also need GnuCOBOL (`cobc` and `libcob`).

3.2 Building from the git repository

```
$ 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
```

The generated autotools files are not stored in the repository: `autoreconf -fi` is always needed after a clone.

3.3 Configuration options

By default, `configure` builds the precompiler and every driver whose client library is found. The following options change this behavior:

Option	Effect
`--enable-gixpp` / `--disable-gixpp`	build the precompiler `gixpp` (default: yes)
`--disable-pgsql`	do not build the PostgreSQL driver
`--disable-oracle`	do not build the Oracle driver
`--disable-mysql`	do not build the MySQL/MariaDB driver
`--disable-odbc`	do not build the ODBC driver
`--disable-sqlite`	do not build the SQLite driver
`--with-default-driver=DRIVER`	driver used when the connection string does not name one: `none` (default), `pgsql`, `oracle`, `mysql`, `odbc`, `sqlite` (see The default driver)

When a driver is explicitly enabled (`--enable-pgsql`) and its client library is missing, `configure` fails; otherwise the driver is silently skipped. Check the summary printed at the end of `configure`.

For example, to build only the precompiler:

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

To build the runtime with PostgreSQL and Oracle only:

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

3.4 Installed files

With `--prefix=/opt/gixsql`, `make install` installs:

File	Description
`bin/gixpp`	the precompiler
`lib/libgixsql.so`	the runtime library, linked with the programs
`lib/libgixsql-<driver>.so`	the database drivers, loaded at run time
`share/gixsql/copy/SQLCA.cpy`	the SQLCA copybook

> **i To be verified**
>
> Check the exact list and paths of installed files (`make install DESTDIR=...`).

At run time, `libgixsql` loads the driver `libgixsql-<driver>.so` with `dlopen`: the `lib` directory must be in the library search path, for example:

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

3.5 Building with Nix

The repository contains a Nix flake:

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

3.6 Building on Windows

3.6.1 Visual Studio

The solution file `gixsql.sln` builds GixSQL with Visual Studio 2019 or later. The dependencies are expected from vcpkg:

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

3.6.2 MSYS2/MinGW

The `configure` script also works under MSYS2, with the packages:

```
$ 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
```

> ⚠ Warning
>
> The Windows builds are not tested by the SuperBOL maintainers.

3.7 Checking the installation

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

The test suite of the precompiler uses autofonce: from the source directory, after building in `_build`, run `autofonce run`.

4 Precompiler concepts

4.1 Embedded SQL in a COBOL program

An embedded SQL statement starts with `EXEC SQL` and ends with `END-EXEC`, followed by a period when the statement ends a COBOL sentence:

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

Inside the statement, COBOL data items, called **host variables**, are prefixed by a colon (`:DEPT-NO`). The precompiler passes their address, type and size to the runtime, which converts their contents to and from the database types (see Host variables and datatypes).

Embedded SQL statements can appear:

- in the DATA DIVISION: `INCLUDE`, `BEGIN/END DECLARE SECTION`, `DECLARE ... CURSOR`, `VAR`, `DECLARE TABLE` and `WHENEVER`...

  > **i** To be verified
  >
  > Check where each statement is accepted by `gixpp`.

- in the PROCEDURE DIVISION: all the executable statements.

4.2 The declare section

Pro*COBOL and the SQL standard require the host variables to be declared between `BEGIN DECLARE SECTION` and `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 accepts these statements, but does not require them: any data item of the WORKING-STORAGE, LOCAL-STORAGE or LINKAGE SECTION can be used as a host variable.

4.3 Including files

`EXEC SQL INCLUDE name END-EXEC` includes a copybook, like a COBOL `COPY` statement, but the included file is also processed by the precompiler. It is mostly used for the SQLCA:

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

The copybooks are searched in the directories given with `-I` / `--copypath`, with the extensions given with `-E` / `--esql-copy-exts`. COBOL `COPY` statements are processed by the precompiler only with `-p` / `--esql-preprocess-copy` (see The gixpp precompiler).

4.4 Precompiling, compiling and linking

The full sequence to build a program `prog.pco` is:

```
$ 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
```

where:

- `-e` enables the ESQL processing (it is mandatory);
- `-S` generates static calls to the runtime (recommended);
- `-I` gives the copybook directories: the one of the SQLCA and the ones of the program;
- `-i` and `-o` give the input and output files.

4.5 A first program

The following program connects to a database whose connection string is in the environment variable `DATASRC`, counts the rows of a table, and displays the result.

```
       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.
```

Build and run it against a PostgreSQL database:

```
$ 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** To be verified
>
> Run this example against the test databases and paste the actual output.

4.6 Sample tables

The examples of this manual use the classic `DEPT` and `EMP` tables:

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

5.1 Connecting to a database

GixSQL has no "bind" step: a program must connect explicitly with a `CONNECT` statement before executing any other SQL statement. The following forms are accepted, where each operand is either a host variable (`:VAR`) or a string literal:

```
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` contains a **connection string** (Connection strings).
- The user name and the password are given either separately, or in a single item as `user.password` (a dot separates them).
- With `CONNECT USING`, the credentials must be in the connection string.
- The Pro*COBOL form `CONNECT :USR-PWD`, where a single item contains `user/password`, is not supported: use `IDENTIFIED BY`.

After the `CONNECT`, `SQLCODE` is 0 on success. On failure, it is negative and `SQLERRMC` contains the message of the database.

5.2 Connection strings

The general format of a connection string is:

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

where `<driver>` is one of `pgsql`, `oracle`, `mysql`, `odbc`, `sqlite` (see Overview of the backends). Examples:

```
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
```

Two other formats are accepted, without the driver name:

- `[<user>.<password>@]<host>[:<port>][/<database>]`
- `<database>@<host>[:<port>]` (the format of OCESQL)

5.2.1 The default driver

When the connection string does not name a driver, GixSQL uses the default driver, which is:

1. the driver chosen when GixSQL was built (`configure --with-default-driver=...`), if any;
2. otherwise, the contents of the environment variable `GIXSQL_DEFAULT_DRIVER`.

If there is no default driver, the `CONNECT` fails.

5.2.2 Connection options

Options follow a `?` at the end of the connection string, separated by `&`. Boolean options accept `on`/`off` or `1`/`0`.

Option	Drivers	Description
`autocommit`	all	`on`, `off` or `native` (Autocommit)
`client_encoding`	pgsql, mysql	encoding of the client, for example `UTF8` (PostgreSQL) or `utf8mb4` (MySQL)
`fixup_params`	all	convert the parameter placeholders of prepared statements (Parameter placeholders)
`default_schema`	pgsql	the `search_path` (one or more schemas, separated by commas)
`native_cursors`	pgsql	use native PostgreSQL cursors (default `on`)
`decode_binary`	pgsql	decode `bytea` values (default `on`)
`updatable_cursors`	mysql, sqlite	emulate updatable cursors
`sslmode`, `sslcert`, ...	pgsql	passed to libpq (PostgreSQL)

Several options can also be set by environment variables (Runtime configuration); the option of the connection string has priority.

5.3 Multiple connections

A program can open several connections, each one with a name:

```
           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.
```

or, with the Pro*COBOL syntax:

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

A statement is then executed on a given connection with the `AT` clause:

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

Statements without `AT` use the default connection, the one opened without a name.

> **i** To be verified
>
> Which connection is the default one when all the connections are named?

5.4 Disconnecting

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

Pro*COBOL programs disconnect with `COMMIT WORK RELEASE` or `ROLLBACK WORK RELEASE`: see Migrating from Pro*COBOL.

5.5 Transactions

A transaction is a sequence of SQL statements that are committed or rolled back together. It ends with:

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

`SAVEPOINT name` and `ROLLBACK TO SAVEPOINT name` are passed unchanged to the database.

5.5.1 Autocommit

The way transactions start depends on the **autocommit mode**, set by the `autocommit` option of the connection string or by the environment variable `GIXSQL_AUTOCOMMIT`:

`native` (default)
: the connection uses the default behavior of the database: autocommit for PostgreSQL, MySQL and ODBC, explicit transactions for Oracle.

`on`
: each successful statement is committed.

`off`
: a transaction is started when connecting, and a new one after each `COMMIT` and `ROLLBACK`.

> ! Important
>
> On PostgreSQL, native cursors (the default) only exist inside a transaction: with autocommit, a `DECLARE CURSOR` fails. Programs that use cursors must either start transactions explicitly (`EXEC SQL START TRANSACTION END-EXEC`) or run with `GIXSQL_AUTOCOMMIT=off`, which is the behavior expected by Pro*COBOL programs.

> i To be verified
>
> The old README states that the default is `off`, the code (`GIXSQL_AUTOCOMMIT_DEFAULT` in `runtime/libgixsql/gixsql.h`) says `native`. Check the actual behavior of each driver in `native` mode.

6 Host variables and datatypes

6.1 Host variables

A host variable is a COBOL data item used in an SQL statement, prefixed by a colon. It is an **input** variable when its value is sent to the database (`WHERE`, `VALUES`, `SET` clauses), and an **output** variable when it receives a value from the database (`INTO` clauses).

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

Host variable names are case-insensitive and may be qualified by their group, as in COBOL (`:EMPNO OF EMP-REC`).

> ℹ To be verified
>
> Qualification (`OF`/`IN`) and the use of a group item as a list of host variables (`INTO :EMP-REC`).

6.2 Datatype conversion

The runtime converts the COBOL data to and from the database types, according to the `PICTURE` and `USAGE` of the host variable:

COBOL declaration	Sent to the database as
`PIC X(n)`	character string (see Fixed-length character strings)
`PIC 9(n)`, `PIC S9(n)V9(m)` (DISPLAY)	number
`PIC S9(n)V9(m) COMP-3`	number
`PIC S9(n) COMP`, `COMP-5`, `BINARY`	integer
`COMP-1`, `COMP-2`	floating point
`PIC N(n)`	national string

> ℹ To be completed
>
> Complete this table from `CobolVarType` (`runtime/libgixsql/SqlVar.h`) and document the conversion of dates.

6.3 Fixed-length character strings

The precompiler option `-P` / `--picx-as` chooses how `PIC X` items are sent to the database:

`char` (default), `charf`
: as fixed-length strings, trailing spaces included;

`varchar`
: as variable-length strings, trailing spaces removed.

This option does not apply to `VARYING` items, whose length is always given by their length field.

6.4 Variable-length strings: VARYING

A `VARYING` item has a length and a data part:

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

The precompiler replaces it with a group of two level-49 items:

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

On output, the runtime sets `ENAME-LEN` to the length of the value; on input, only the first `ENAME-LEN` characters are sent.

- The suffixes `-LEN` and `-ARR` can be changed with `-Y` / `--varying` (for example `--varying=LEN,TEXT`).
- The length field is 4 bytes by default. Pro*COBOL uses a 2-byte length (`PIC S9(4) COMP`): use `-N 2` / `--varying-length-size 2`. The generated code tells the runtime the size of the length fields.

Group items with level-49 items written by hand are also recognized as variable-length strings, as in Pro*COBOL. Their length field must have the size given by `-N`.

> **i** To be verified
>
> Detection of hand-written level-49 groups, and `PIC X(n) VARYING` vs `USAGE VARRAW`.

6.5 SQL TYPE IS

The `SQL TYPE IS` clause declares a host variable with an SQL type:

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

`VARCHAR` and `VARBINARY` generate a `VARYING` group as above.

6.6 EXEC SQL VAR

The `VAR` statement gives an SQL type to an existing data item (datatype equivalencing, in Pro*COBOL terms):

```
       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.
```

The supported types are `CHAR`, `CHARF`, `VARCHAR`, `VARCHAR2` (same as `VARCHAR`), `VARBINARY`, `INTEGER`, `FLOAT`, `DECIMAL` and `LONG RAW`.

6.7 Indicator variables

An indicator variable is associated with a host variable to handle `NULL` values. It must be declared `PIC S9(4) COMP`, and is written just after the host variable, optionally preceded by `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.
```

Value	On output (`SELECT`, `FETCH`)	On input (`INSERT`, `UPDATE`)
`-1`	the column is `NULL`	`NULL` is sent, the host variable is ignored
`0`	the value was stored	the host variable is sent

Truncation (positive indicator values) is not reported.

> **i To be verified**
>
> `INDICATOR` keyword, indicator tables for group items.

6.8 Host tables (arrays)

GixSQL does not support host tables: a host variable cannot be a table (`OCCURS`) to fetch or insert several rows in a single statement. Use a cursor and a loop instead. See Feature comparison.

7 Embedded SQL statements

7.1 Executing SQL statements

Most SQL statements are passed to the database as they are, after the host variables are replaced by parameter placeholders: `INSERT`, `UPDATE`, `DELETE`, DDL statements (`CREATE`, `DROP`...), `SAVEPOINT`, and so on. The precompiler does not check the SQL syntax: errors are reported by the database at run time.

```
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.
```

After an `INSERT`, `UPDATE` or `DELETE`, `SQLERRD(3)` of the SQLCA contains the number of rows processed.

> ℹ To be verified
>
> `SQLERRD(3)` for each driver.

7.2 Single-row queries: SELECT INTO

A `SELECT` that returns at most one row stores its result directly in host variables:

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

If no row is found, `SQLCODE` is the "no record" code (100 by default, see The "no record" condition). If several rows are found, an error is returned.

7.3 Cursors

A query that returns several rows is processed with a cursor:

1. `DECLARE` the cursor, with its query;
2. `OPEN` it: the query is executed;
3. `FETCH` the rows one by one, until the "no record" condition;
4. `CLOSE` it.

```
           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.
```

The input host variables of the query (`:DEPT-NO`) are read when the cursor is opened.

- `DECLARE CURSOR` can appear in the DATA DIVISION or in the PROCEDURE DIVISION. It is not an executable statement: it does not set `SQLCODE`.
- `FETCH` does not accept the `AT` clause: the cursor is used on the connection where it was declared.
- A cursor declared `WITH HOLD` stays open after a `COMMIT`:

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

> ! Important
>
> With PostgreSQL native cursors, cursors must be used inside a transaction: see Auto-commit.

7.3.1 Updatable cursors

The current row of a cursor declared `FOR UPDATE` can be modified or deleted with `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.
```

The support depends on the database:

- Oracle and PostgreSQL (with native cursors) support them directly.
- MySQL and SQLite emulate them, with the `updatable_cursors` option; the table must have a unique key.
- ODBC depends on the ODBC driver.

7.3.2 Cursors on prepared statements

A cursor can also be declared on a prepared statement (see Dynamic SQL):

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

7.4 Transactions

See Connections and transactions.

7.5 Other statements

> i To be completed
>
> `EXEC SQL IGNORE`, `CALL` / stored procedures, `LOCK TABLE`, `SET TRANSACTION`, `DECLARE TABLE`.

8 Error handling and diagnostics

8.1 The SQL Communication Area

The SQLCA is a data structure updated by each executable SQL statement. It is included in the WORKING-STORAGE SECTION with:

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

Its main fields are:

Field	Description
`SQLCODE`	0 on success, a positive value for warnings and "no record", a negative value for errors
`SQLSTATE`	the 5-character SQLSTATE code of the SQL standard
`SQLERRML`, `SQLERRMC`	length and text of the error message
`SQLERRD(3)`	number of rows processed
`SQLWARN0` …	warning flags

> **i To be completed**
>
> Copy the actual layout of `copy/SQLCA.cpy`.

8.2 SQLCODE and SQLSTATE

When the database reports an error, GixSQL sets `SQLCODE` to the error code of the database when there is one (for example `-942` for Oracle's `ORA-00942`) and `SQLSTATE` to the corresponding SQLSTATE. When the error comes from GixSQL itself, `SQLCODE` is one of the negative codes listed in Error codes.

> **i To be verified**
>
> `SQLCODE` mapping for each driver (PostgreSQL has no numeric codes).

8.3 The "no record" condition

When a `SELECT INTO` finds no row, or a `FETCH` reaches the end of a cursor, `SQLSTATE` is `02000` and `SQLCODE` is 100. Oracle uses 1403 instead of 100; to get this value:

- precompile with `--no-rec-code 1403`, so that `WHENEVER NOT FOUND` tests this value;
- run with `GIXSQL_NOREC_CODE=1403`, so that the runtime sets it.

Both settings must be consistent.

8.4 WHENEVER

The `WHENEVER` statement tells the precompiler to generate a test after each following SQL statement, up to the next `WHENEVER` for the same 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` is negative;

`NOT FOUND`
: the “no record” condition;

`SQLWARNING`
: a warning is set.

Actions:

`CONTINUE`
: no test (default);

`GO TO label`, **`GOTO label`**
: jump to the paragraph or section `label`;

`PERFORM label`
: perform the paragraph or section `label`.

`WHENEVER` is a declaration: it applies to the statements that follow it in the source, not in the execution order.

8.5 Logging

When an error is not clear from the SQLCA, the runtime log gives the details of each call, including the SQL statements sent to the database (see Logging).

9 Dynamic SQL

Dynamic SQL statements are built at run time, in a host variable. GixSQL supports the first three “methods” of Pro*COBOL; the fourth one (descriptors, `DESCRIBE`) is not supported.

9.1 Method 1: EXECUTE IMMEDIATE

For a statement without host variables and that returns no rows:

```
           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 Method 2: PREPARE and EXECUTE

For a statement executed several times, with input parameters:

```
       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 ...` returns the values of a single-row query.

9.3 Method 3: prepared queries and cursors

For a query returning several rows:

```
           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 To be verified
>
> `OPEN ... USING` on a cursor declared on a prepared statement.

9.4 Parameter placeholders

Each database has its own syntax for parameters:

- PostgreSQL: `$1`, `$2`...
- Oracle: `:name` or `:1`, `:2`...
- ODBC: `?`
- SQLite: any of these.

For static statements, the precompiler generates the placeholders chosen with `-z` / `--param-style`: `d` for `$1` (default), `a` for `?`, `c` for `:1`. The Oracle driver also accepts `$n` placeholders.

For prepared statements, the text is written by the program. GixSQL can convert the placeholders of a prepared statement to those of the database, with the connection option `fixup_params=on` or the environment variable `GIXSQL_FIXUP_PARAMS=on`.

10 The gixpp precompiler

10.1 Synopsis

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

`gixpp` reads a COBOL source with embedded SQL and writes a COBOL source where the `EXEC SQL` blocks are commented out and replaced with calls to `libgixsql`.

10.2 Options

10.2.1 Input and output

`-i FILE`, `--infile FILE`
: the input file.

`-o FILE`, `--outfile FILE`
: the output file.

`-I DIRS`, `--copypath DIRS`
: the directories where copybooks are searched (for `EXEC SQL INCLUDE` and, with `-p`, `COPY`). The option can be repeated.

`-E EXTS`, `--esql-copy-exts EXTS`
: the extensions of the copybooks, separated by commas, for example `-E ".,cpy,CPY"`.

`-c`, `--consolidate`
: write a single source file, with the copybooks expanded.

`-k`, `--keep`
: keep the temporary files.

`-s FILE`, `--symfile FILE`
: write a symbol file.

`-m`, `--map`
: write a map file (correspondence between the input and output lines).

10.2.2 Source format

`--code-end-col N`
: the last column of the code area in fixed format, from 72 to 512 (default 72). The text after this column is ignored, with a warning. Use `--code-end-col 512` for sources written for Pro*COBOL with `format=terminal ireclen=512`.

> **i** To be completed
>
> Free format support.

10.2.3 ESQL processing

`-e`, `--esql`
: process the embedded SQL statements. This option is required.

`-p`, `--esql-preprocess-copy`
: process the embedded SQL statements in all the copybooks included with `COPY`, not only those included with `EXEC SQL INCLUDE`.

`-S`, `--esql-static-calls`
: generate static calls to the runtime (`CALL STATIC`). Recommended.

`-z STYLE`, `--param-style STYLE`
: the style of the parameter placeholders in the generated SQL statements: `d` for `$1` (default), `a` for `?`, `c` for `:1`. See Parameter placeholders.

`-P MODE`, `--picx-as MODE`
: how `PIC X` host variables are sent: `char` (default), `charf` or `varchar` (see Fixed-length character strings).

`-Y LEN,ARR`, `--varying LEN,ARR`
: the suffixes of the length and data items of the `VARYING` groups (default `LEN,ARR`).

`-N SIZE`, `--varying-length-size SIZE`
: the size in bytes of the length items of the `VARYING` groups: `4` (default) or `2` (Pro*COBOL).

`--no-rec-code CODE`
: the `SQLCODE` of the “no record” condition, tested by `WHENEVER NOT FOUND` (default 100; 1403 for Oracle). See The “no record” condition.

`-C`, `--cobol85`
: generate COBOL 85 code.

> **i** To be verified
>
> What `--cobol85` changes in the generated code.

10.2.4 Debugging

`-g`, `--debug-info`
: generate debug information.

`-v`, `--verbose`
verbose output.

`-d`, `--verbose-debug`
debug output.

`-D`, `--parser-scanner-debug`
trace the parser and the scanner.

10.2.5 Information

`-h`, `--help`
display the options.

`-V`, `--version`
display the version.

10.3 Generated code

> ℹ To be completed
>
> Show the code generated for a `SELECT INTO`: the commented statement, the `CALL "GIXSQLExec..."` sequence, and the generated `SQ0001` data items holding the SQL text.

11 Runtime configuration

The runtime library `libgixsql` is configured by environment variables. Some of them can also be set per connection, with an option of the connection string (Connection options), which has priority.

11.1 Environment variables

Variable	Default	Description
`GIXSQL_DEFAULT_DRIVER`		driver used when the connection string names none (The default driver)
`GIXSQL_AUTOCOMMIT`	`native`	`on`, `off` or `native` (Autocommit)
`GIXSQL_CLIENT_ENCODING`		client encoding (PostgreSQL, MySQL)
`GIXSQL_FIXUP_PARAMS`	`off`	convert the placeholders of prepared statements (Parameter placeholders)
`GIXSQL_NOREC_CODE`	`100`	`SQLCODE` of the "no record" condition (The "no record" condition)
`GIXSQL_VARYING_LEN_SZ_SHORT`	`0`	`1` if the `VARYING` length fields are 2 bytes; not needed for programs precompiled with `-N 2`
`GIXSQL_LOG_LEVEL`	`error`	see Logging
`GIXSQL_LOG_FILE`	`gixsql.log`	see Logging
`GIXSQL_LOG_TRUNCATE`	`off`	see Logging
`GIXSQL_LOG_ROTATE`	`off`	see Logging
`GIXSQL_LOG_ROTATE_MAX_SIZE`	`10M`	see Logging
`GIXSQL_LOG_ROTATE_MAX_FILES`	`5`	see Logging
`GIXSQL_LOG_ROTATE_ON_OPEN`	`on`	see Logging

Boolean variables accept `on`/`off` and `1`/`0`.

> ⚠ Warning
>
> `GIXSQL_USRPWD_SEP` is read by the runtime but has no effect in the current version: the separator between the user name and the password is always a dot.

The drivers also use the variables of their client libraries, for example `PGHOST` or `PGSSLMODE` for PostgreSQL, `NLS_LANG` and `TNS_ADMIN` for Oracle, `ODBCINI` for unixODBC.

11.2 Finding the drivers

`libgixsql` loads the driver `libgixsql-<driver>.so` (`libgixsql-<driver>.dll` on Windows) with the system dynamic loader. The directory of the drivers must be in `LD_LIBRARY_PATH` (`PATH` on Windows), or in the system library path.

11.3 Logging

The runtime writes a log, with spdlog.

`GIXSQL_LOG_LEVEL`
: `off`, `critical`, `error` (default), `warn`, `info`, `debug` or `trace`. The `trace` level logs every call, with the values of the host variables (including passwords and other sensitive data), and slows down the programs.

`GIXSQL_LOG_FILE`
: the log file (default `gixsql.log`, in the current directory).

`GIXSQL_LOG_TRUNCATE`
: `on` to truncate the log file when the program starts.

`GIXSQL_LOG_ROTATE`
: `on` to rotate the log files.

`GIXSQL_LOG_ROTATE_MAX_SIZE`
: the maximal size of a log file, with an optional suffix `B`, `K`, `M` (default) or `G` (default `10M`).

`GIXSQL_LOG_ROTATE_MAX_FILES`
: the number of files kept (default 5).

`GIXSQL_LOG_ROTATE_ON_OPEN`
: `on` (default) to rotate when the program starts.

For example, to trace a program:

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

12 Migrating from Pro*COBOL

This chapter explains how to precompile and run with GixSQL programs written for Oracle Pro*COBOL, either on an Oracle database or on PostgreSQL.

12.1 Recommended settings

Setting	Where	Why
`-N 2`	gixpp	`VARYING` items with a 2-byte length (`PIC S9(4) COMP`), as Pro*COBOL
`--no-rec-code 1403`	gixpp	`SQLCODE` of the "no record" condition, as Oracle, for `WHENEVER NOT FOUND`
`GIXSQL_NOREC_CODE=1403`	environment	the same, at run time
`GIXSQL_AUTOCOMMIT=off`	environment	transactions are explicit with Pro*COBOL; on PostgreSQL, cursors need a transaction
`--code-end-col 512`	gixpp	sources precompiled by `procob` with `format=terminal ireclen=512`, with code after column 72

A typical build is:

```
$ 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 Connecting

Pro*COBOL programs connect with `CONNECT :USR IDENTIFIED BY :PWD`, optionally `USING :DB-STRING` with a SQL*Net alias. With GixSQL, the `USING` operand is a GixSQL connection string (Connection strings), not an Oracle alias.

> ℹ To be verified
>
> Behavior of `CONNECT ... IDENTIFIED BY` without `USING` (mode 4 of the grammar is "unsupported").

`COMMIT WORK RELEASE` and `ROLLBACK WORK RELEASE` end the transaction and close the connection.

12.3 Running on PostgreSQL

Programs written for Oracle can run on PostgreSQL, provided their SQL is compatible. The orafce extension provides many Oracle functions (`NVL`, `DECODE`, `TO_DATE`...). Its `oracle` schema must come before `pg_catalog` in the search path, so that `oracle.to_date` is used instead of the PostgreSQL `to_date`:

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

> ℹ To be completed
>
> Common SQL differences: `ROWNUM`, outer joins `(+)`, `SYSDATE`, empty strings and `NULL`, `DUAL`, sequences (`seq.NEXTVAL`).

12.4 Feature comparison

The table follows the chapters of the *Pro*COBOL Programmer's Guide*.

Pro*COBOL feature	GixSQL
Declare section	accepted, not required
`EXEC SQL INCLUDE`	supported
Conditional precompilation (`EXEC ORACLE IFDEF`)	not supported

Pro*COBOL feature	GixSQL
`CONNECT ... IDENTIFIED BY`, `AT`	supported
`CONNECT :USR-PWD` (`user/password` in one item)	not supported
`COMMIT`, `ROLLBACK`, `SAVEPOINT`, `RELEASE`	supported
`SET TRANSACTION`, `LOCK TABLE`	passed to the database
Host variables, indicator variables	supported
`VARCHAR` / `VARYING` variables (2-byte length)	supported with `-N 2`
Datatype equivalencing (`EXEC SQL VAR`)	partial: see Host variables and datatypes
`EXEC SQL TYPE`	not supported
Universal `ROWID`	fetched as character strings (Oracle)
`SELECT INTO`, `INSERT`, `UPDATE`, `DELETE`	supported
Cursors, `WHERE CURRENT OF`	supported
Scrollable cursors (`FETCH PRIOR`, `ABSOLUTE`...)	not supported
Embedded PL/SQL blocks (`EXEC SQL EXECUTE BEGIN ... END; END-EXEC`)	not supported
Host tables (arrays)	not supported
SQLCA	supported
ORACA	not supported
`WHENEVER` (`SQLERROR`, `SQLWARNING`, `NOT FOUND`)	supported
`WHENEVER ... DO`, `STOP`	not supported
Dynamic SQL methods 1, 2, 3	supported
Dynamic SQL method 4 (SQLDA), ANSI dynamic SQL (descriptors)	not supported
Runtime contexts, multithreading	not supported
LOB statements	not supported
Precompiler options	see the correspondence below

i To be verified

Each “not supported” line above, in particular `SET TRANSACTION`, `EXEC SQL TYPE`, scrollable cursors, `WHENEVER ... DO`, and the generic passthrough of unknown statements.

12.5 Precompiler options

`procob` option	`gixpp` equivalent
`INAME`, `ONAME`	`-i`, `-o`
`INCLUDE`	`-I`
`FORMAT=TERMINAL`, `IRECLEN`	`--code-end-col`
`MODE=ANSI` (`SQLCODE` 100 on “no record”)	default
`MODE=ORACLE` (`SQLCODE` 1403)	`--no-rec-code 1403`
`PICX=VARCHAR2`	`-P varchar`
`PICX=CHARF`	`-P charf` (default)
`VARCHAR=YES`	level-49 groups are always recognized
`SQLCHECK`, `USERID`	no equivalent: no semantic checking
`RELEASE_CURSOR`, `HOLD_CURSOR`, `MAXOPENCURSORS`	no equivalent

> **i To be verified**
>
> Check `MODE` and `PICX` against the Pro*COBOL documentation.

13 Overview of the backends

A **backend** (or driver) is a library `libgixsql-<driver>` that implements the access to one kind of database. `libgixsql` loads it when a program connects, according to the prefix of the connection string. A program can use several backends at the same time, with several connections.

Driver	Database	Connection string	Chapter
`pgsql`	PostgreSQL	`pgsql://host[:port]/database`	PostgreSQL
`oracle`	Oracle Database	`oracle://host[:port]/service`	Oracle
`mysql`	MySQL, MariaDB	`mysql://host[:port]/database`	MySQL and MariaDB
`odbc`	any ODBC data source	`odbc://dsn`	ODBC
`sqlite`	SQLite	`sqlite://path`	SQLite

13.1 Feature support

Feature	pgsql	oracle	mysql	odbc	sqlite
Autocommit `native`	autocommit	transactions	autocommit	driver	?
Native cursors	yes (default)	yes	emulated	driver	emulated
Updatable cursors	native cursors	yes	emulated	driver	emulated
`client_encoding`	yes	no (`NLS_LANG`)	yes	?	no
Default placeholder	`$1`	`:1` (accepts `$1`)	`?`	`?`	any

> **i** To be verified
>
> Every cell of this table, and the cells marked “?”.

13.2 Choosing the placeholder style

The SQL statements of the program are generated with the placeholder style chosen at precompile time (`-z`, Parameter placeholders). It must be accepted by the backend used at run time:

- `-z d` (default): PostgreSQL, Oracle, SQLite;

- `-z a`: ODBC, MySQL, SQLite;
- `-z c`: Oracle, SQLite.

14 PostgreSQL

14.1 Requirements

- Build: `libpq` development files (`libpq-dev`), found with `pkg-config`.
- Run: `libpq`.

14.2 Connection string

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

Example:

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

The user name and password are usually given in the `CONNECT` statement. The libpq environment variables (`PGHOST`, `PGPORT`, `PGPASSWORD`, `PGSSLMODE`...) and the `~/.pgpass` file are taken into account by libpq for what the connection string does not give.

14.3 Options

`default_schema`
: the schemas of the `search_path`, separated by commas.

`client_encoding`
: the client encoding, for example `UTF8` or `LATIN1`.

`native_cursors`
: `on` (default) to use PostgreSQL cursors (`DECLARE ... CURSOR`, `FETCH`); `off` to execute the query when the cursor is opened and keep all the rows in memory.

`decode_binary`
: `on` (default) to decode `bytea` values; `off` to receive them in their text encoding (larger).

libpq options
: `hostaddr`, `connect_timeout`, `application_name`, `keepalives`, `keepalives_idle`, `keepalives_interval`, `keepalives_count`, `sslmode`, `requiressl`, `sslcert`, `sslkey`, `sslrootcert`, `sslcrl`, `krbsrvname`, `gsslib`, `service` are passed to libpq. See the libpq documentation.

14.4 Cursors and transactions

Native cursors only exist inside a transaction. With the default autocommit mode, every statement is its own transaction, and `OPEN` fails. Either:

- start a transaction before opening the cursor (`EXEC SQL START TRANSACTION END-EXEC`), or
- connect with `autocommit=off` (or `GIXSQL_AUTOCOMMIT=off`), or
- use emulated cursors (`native_cursors=off`).

14.5 Running Oracle programs

See Migrating from Pro*COBOL for the orafce extension and the search path.

14.6 Testing

> **i** To be completed
>
> How to set up a test database (Docker image, orafce).

15 Oracle

15.1 Requirements

- Build: nothing; the driver includes ODPI-C.
- Run: the Oracle client libraries (`libclntsh.so`), for example from the Oracle Instant Client. ODPI-C loads them at run time: their directory must be in `LD_LIBRARY_PATH` (or set `ORACLE_HOME`).

15.2 Connection string

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

The driver connects with the Easy Connect syntax `//host:port/service`. Only service names are supported, not SIDs nor `tnsnames.ora` aliases.

Example, with the Oracle Free container:

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

15.3 Options and environment

- `client_encoding` is ignored: set `NLS_LANG` before running the program.
- The variables `DPI_DEBUG_LEVEL` and `DPI_DEBUG_PREFIX` enable the traces of ODPI-C.

15.4 Transactions

Oracle always works in a transaction: with the default `native` autocommit mode, a program must `COMMIT` its changes, as with Pro*COBOL.

15.5 Data types

- `ROWID` columns are fetched as character strings.
- Parameters can be written `:1`, `:name`, or `$1` (converted by the driver).

> **i** To be completed
>
> `DATE`/`TIMESTAMP` conversions, `NUMBER` precision, `CLOB`/`BLOB`.

15.6 Testing

> ℹ To be completed
>
> How to run the Oracle Free container (`gvenzl/oracle-free`) and create a test user.

16 MySQL and MariaDB

16.1 Requirements

- Build: `libmysqlclient` or `libmariadb` development files, found with `pkg-config` (`mysqlclient` is tried first, then `libmariadb`).
- Run: the corresponding client library.

16.2 Connection string

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

Example:

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

16.3 Options

`client_encoding`
: the character set of the connection, for example `utf8mb4`.

`autocommit`
: passed to the client library (`mysql_autocommit`).

`updatable_cursors`
: emulate updatable cursors; the table must have a unique key.

> **ℹ To be verified**
>
> Is `updatable_cursors` read by the MySQL driver (it is in the SQLite driver)?

16.4 Placeholders

MySQL only accepts `?` placeholders: precompile with `-z a`, or connect with `fixup_params=on`.

> **ℹ To be verified**
>
> Whether the MySQL driver converts `$n` placeholders itself.

17 ODBC

17.1 Requirements

- Build: the unixODBC development files (`unixodbc-dev`), found with `pkg-config`.
- Run: unixODBC, and the ODBC driver of the database (psqlODBC, MySQL Connector/ODBC, IBM Db2 driver...), configured in `odbcinst.ini`.

17.2 Connection string

```
odbc://dsn
```

`dsn` is the name of a data source defined in `odbc.ini` (or `~/.odbc.ini`); the driver connects with `SQLConnect`, using the user name and password of the `CONNECT` statement.

Example `~/.odbc.ini`:

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

17.3 Behavior

Most features depend on the ODBC driver: autocommit, updatable cursors, data type conversions.

- ODBC only accepts `?` placeholders: precompile with `-z a`.
- With psqlODBC, set the “rollback on error” option to `nop` (`Protocol = 7.4-0`), so that GixSQL, not the driver, handles the errors.

> **i** To be verified
>
> The psqlODBC setting for “rollback on error”.

18 SQLite

18.1 Requirements

None: SQLite is built in the driver.

18.2 Connection string

The connection string gives the path of the database file:

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

The file is created if it does not exist. The user name and password are ignored.

18.3 Options

`updatable_cursors`
: emulate updatable cursors (`WHERE CURRENT OF`); the table must have a unique key.

18.4 Behavior

- SQLite accepts the `$1`, `?` and `:1` placeholders.
- SQLite types are dynamic: the conversion is driven by the type of the host variable.

> **i** To be verified
>
> Autocommit and cursor behavior of the SQLite driver.

19 Embedded SQL statements

This appendix summarizes the statements recognized by `gixpp`. Each statement is written between `EXEC SQL` and `END-EXEC`. Statements marked with `[AT conn]` accept the `AT` clause to choose a connection.

Statement	Description	See
`BEGIN DECLARE SECTION`, `END DECLARE SECTION`	delimit the host variables (optional)	Precompiler concepts
`INCLUDE name`	include a copybook (`SQLCA`…)	Precompiler concepts
`VAR item IS type`	give an SQL type to a host variable	Host variables and datatypes
`DECLARE name CURSOR [WITH HOLD] FOR select`	declare a cursor	Embedded SQL statements
`DECLARE name CURSOR [WITH HOLD] FOR stmt`	declare a cursor on a prepared statement	Dynamic SQL
`DECLARE name TABLE (...)`	declare a table (ignored)	
`CONNECT TO src [AS conn] [USER usr [USING pwd]]`	connect	Connections and transactions
`CONNECT usr IDENTIFIED BY pwd [AT conn] USING src`	connect	Connections and transactions
`CONNECT USING src`	connect	Connections and transactions
`CONNECT RESET [conn]`, `DISCONNECT [conn | ALL]`	disconnect	Connections and transactions
`[AT conn] COMMIT [WORK] [RELEASE]`	commit	Connections and transactions
`[AT conn] ROLLBACK [WORK] [RELEASE]`	roll back	Connections and transactions
`[AT conn] ROLLBACK [WORK] TO SAVEPOINT name`	roll back to a savepoint	Connections and transactions
`[AT conn] SELECT ... INTO ...`	single-row query	Embedded SQL statements

Statement	Description	See
`[AT conn] INSERT, UPDATE, DELETE`	data manipulation	Embedded SQL statements
`UPDATE, DELETE ... WHERE CURRENT OF cursor`	positioned update/delete	Embedded SQL statements
`[AT conn] OPEN cursor [USING ...]`	open a cursor	Embedded SQL statements
`FETCH cursor INTO ...`	fetch a row	Embedded SQL statements
`CLOSE cursor`	close a cursor	Embedded SQL statements
`[AT conn] PREPARE stmt FROM src`	prepare a statement	Dynamic SQL
`[AT conn] EXECUTE stmt [INTO ...] [USING ...]`	execute a prepared statement	Dynamic SQL
`[AT conn] EXECUTE IMMEDIATE src`	execute a dynamic statement	Dynamic SQL
`WHENEVER condition action`	error handling	Error handling and diagnostics
`IGNORE ...`	ignored by the precompiler	
other statements	passed unchanged to the database	Embedded SQL statements

i To be verified

Every line of this table against `libgixpp/gix_esql_parser.yy`, in particular `DISCONNECT`, `DECLARE TABLE`, `IGNORE`, and the statements that accept `AT`.

20 Error codes

When an operation fails for a reason internal to GixSQL (consistency check, unsupported feature, driver problem...), `SQLCODE` is set to one of the following negative codes. The log gives more details (Logging).

Name	SQLCODE	Description
DBERR_NO_ERROR	0	no error
DBERR_CONNECTION_FAILED	−100	the connection to the database failed
DBERR_BEGIN_TX_FAILED	−101	a transaction could not be started
DBERR_END_TX_FAILED	−102	a transaction could not be ended
DBERR_CONN_NOT_FOUND	−103	connection not found
DBERR_CONN_RESET_FAILED	−104	the connection could not be closed
DBERR_EMPTY_QUERY	−105	empty query
DBERR_SQL_ERROR	−106	generic SQL or driver error
DBERR_TOO_MANY_ARGUMENTS	−107	too many arguments
DBERR_TOO_FEW_ARGUMENTS	−108	too few arguments
DBERR_NO_PARAMETERS	−109	parameters were expected but not supplied
DBERR_CURSOR_EXISTS	−110	the cursor already exists
DBERR_NO_SUCH_CURSOR	−111	no such cursor
DBERR_CLOSE_CURSOR_FAILED	−112	the cursor could not be closed
DBERR_DISCONNECT_FAILED	−113	could not disconnect
DBERR_OUT_OF_MEMORY	−114	out of memory
DBERR_DECLARE_CURSOR_FAILED	−115	the cursor declaration failed
DBERR_OPEN_CURSOR_FAILED	−116	the cursor could not be opened
DBERR_FETCH_ROW_FAILED	−117	could not fetch a row
DBERR_INVALID_COLUMN_DATA	−118	invalid column data
DBERR_CURSOR_CLOSED	−119	the cursor is closed
DBERR_MOVE_TO_FIRST_FAILED	−120	cannot move to the first row of a result set

Name	SQLCODE	Description
DBERR_FIELD_COUNT_MISMATCH	−121	the number of columns does not match the query
DBERR_NO_DATA	−122	no rows when rows were expected
DBERR_TOO_MUCH_DATA	−123	more rows than expected
DBERR_PREPARE_FAILED	−124	the statement could not be prepared
DBERR_CONN_INIT_ERROR	−201	connection initialization error
DBERR_CONN_INVALID_DBTYPE	−202	invalid database type
DBERR_NUM_OUT_OF_RANGE	−410	numeric value out of range
DBERR_INTERNAL_ERR	−8880	internal error
DBERR_NOT_IMPL	−9990	feature not implemented