Fujitsu Enterprise Postgres

Embedded SQL in C and COBOL

Many companies still run legacy systems that they find to be too large, complex, or vital to be rewritten using newer languages, especially since they have stood the test of time. With Fujitsu Enterprise Postgres you can also avoid the risk and expense of porting older code by reusing your C and COBOL programs with little to no modification.

About Fujitsu Enterprise Postgres

Founded on PostgreSQL, the world's most advanced open source relational database system, Fujitsu Enterprise Postgres extends base PostgreSQL functionality with a number of enhanced enterprise features.

Why retain legacy systems?

Companies tend to invest in their ICT systems to ensure that they are up to date with current technologies, so they can take advantage of the latest advances in performance, usability, and security.

But the reality is also that companies may find the need to retain legacy systems because they do not see enough justification to invest in their overhaul. Several factors may contribute to this decision, such as the fact that while not as up-to-date with current technologies, they have stood the test of time, and are still running and performing the job they have been created for. Other factors are the risk involved in running the new solution rewritten for another language or technology, and the financial investment required to do that.

C and COBOL still have a place in your organization

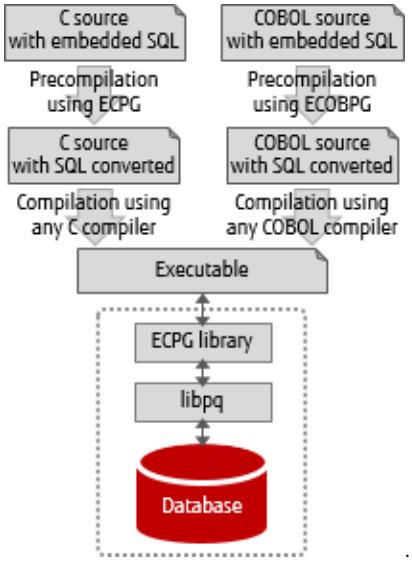
These languages have existed for several decades now, and proved their versatility and robustness with millions of lines of code written for all types of applications across multiple industries for all types of organizations, from enterprises to small businesses.

Accessing PostgreSQL using C

Embedded SQL in C programs is precompiled by a library provided by the PostgreSQL community, and replaced with special functions calls using C language, so the result can be processed with any C compiler.

Accessing PostgreSQL using COBOL

Similarly, embedded SQL in COBOL programs is precompiled by ECOBPG, a library provided by Fujitsu, so that its output can be processed by any COBOL compiler.



Commands

All SQL commands can be executed from C and COBOL programs. The table below lists the commands available only via embedded SQL.

shaping tomorrow with you

FUJITSU

Command	Description	Synopsis
ALLOCATE DESCRIPTOR	Allocate an SQL descriptor area	ALLOCATE DESCRIPTOR *descId*
CONNECT	Establish a database connection	CONNECT TO *connTarget* *1 [AS *connName*] [USER *connUser*]
		CONNECT { *connUsername* | TO DEFAULT }
		DATABASE *connTarget* *1
DEALLOCATE DESCRIPTOR	Deallocate an SQL descriptor area	DEALLOCATE DESCRIPTOR *descId*
DECLARE	Define a cursor	DECLARE *curName* [BINARY] [INSENSITIVE] [[NO] SCROLL] CURSOR [{ WITH | WITHOUT } HOLD] FOR { *preprdStmt* | *query* }
DESCRIBE	Obtain information about a prepared statement or result set	DESCRIBE [OUTPUT] *prepdStmt* {USING|INTO} SQL*2 DESCRIPTOR *descId*
		DESCRIBE [OUTPUT] *prepdStmt* INTO *sqlDaName**3
DISCONNECT	Close a database connection	DISCONNECT [*connName* | CURRENT | DEFAULT | ALL]
EXECUTE IMMEDIATE	Prepare and execute a statement	EXECUTE IMMEDIATE *stmt*
GET DESCRIPTOR	Get information from an SQL descriptor area	GET DESCRIPTOR *descId* :*hostVar* = *descHdrItem**4, ...
		GET DESCRIPTOR *descId* VALUE *colNum* :*hostVar* = *descItem**5, ...
OPEN	Open a dynamic cursor	OPEN *curName* [USING { *val*, ... | SQL DESCRIPTOR *descId* }]
PREPARE	Prepare a statement for execution	PREPARE *varPrepdStmt* FROM *sqlCmd*
SET AUTOCOMMIT	Set the autocommit behavior of the current session	SET AUTOCOMMIT { = | TO } { ON | OFF }
SET CONNECTION	Select a database connection	SET CONNECTION [TO | =] *connName*
SET DESCRIPTOR	Set information in SQL descriptor area	SET DESCRIPTOR *descId* { *descHdrItem**4=*val* , ... | VALUE *descItemNum descItem**5=*val*, ... }
TYPE	Define a new data type	TYPE *typeName* IS *ctype*
VAR	Define a variable	VAR *varName* IS *ctype*
WHENEVER	Specify the action when SQL causes a condition to be raised	WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } *action*

*1: *connTarget* for C is [*dbName*][@*host*][:*port*], tcp:postgresql://*host*[:*port*]/[*dbName*][?*options*], unix:postgresql://*host*[:*port*]/[*dbName*][?*options*], for COBOL it is *dbName*@*host*:*port*, tcp:postgresql://*host*:*port*/*dbName*[?*options*], unix:postgresql://*host*[:*port*][/*dbName*][?*options*]
*2: The 'SQL' keyword is optional in C *3: Statement supported in C only
*4: *descHdrItem* identifies the header information to retrieve/set *(only COUNT is supported at the moment)* *5: *descItem* identifies the descriptor item to retrieve/set

Tasks

The table below shows how to use embedded SQL to perform the most common tasks when working with a database.

Category	Task	Synopsis*1
Manage connections	Connect to server	EXEC SQL CONNECT TO *connTarget* [AS *connName*] [USER *user*];
	Choose connection	EXEC SQL { SET CONNECTION *connName* | AT *connName sqlCmd*; };
	Close connection	EXEC SQL DISCONNECT [*connName* | DEFAULT | CURRENT | ALL];
Run SQL	Execute SQL	EXEC SQL *cmd* ;
	Declare cursors	EXEC SQL DECLARE *curName* CURSOR FOR *sqlCmd* ;
		EXEC SQL DECLARE *curName* CURSOR FOR *varPrepdStmt* ;
	Use cursors	EXEC SQL OPEN *curName* [USING { *val1*, ... | SQL DESCRIPTOR *descId* }]; EXEC SQL FETCH *curName* INTO :*hostVar1*, ... ; ... EXEC SQL CLOSE *curName* ; EXEC SQL COMMIT;
	Manage transactions	EXEC SQL COMMIT [PREPARED *txId*]; EXEC SQL ROLLBACK [PREPARED *txId*]; EXEC SQL SET AUTOCOMMIT TO { ON | OFF } ;
	Declare prepared statements	EXEC SQL PREPARE *varPrepdStmt* FROM *prepdStmt* ;
	Execute prepared statements	EXEC SQL EXECUTE *varPrepdStmt* INTO :*hostVar1*, ...USING *val* ;
		EXEC SQL EXECUTE *prepdStmt* USING SQL DESCRIPTOR *descIdIn* INTO SQL DESCRIPTOR *descIdOut* ;
	Deallocate prepared statements	EXEC SQL DEALLOCATE PREPARE *varPrepdStmt* ;

*1: SQL statements are terminated with semicolon in C, or with END-EXEC. in COBOL

Category	Task	Synopsis
Dynamic SQL	Statements without a result set	EXEC SQL EXECUTE IMMEDIATE :*varPrepdStmt*;
	Statement with a result set	EXEC SQL EXECUTE *varPrepdStmt* INTO :*var1*, ... [USING *val1*, ...];
Use host variables	Declare host variables	EXEC SQL BEGIN DECLARE SECTION; *hostVarDeclaration* EXEC SQL END DECLARE SECTION;
		EXEC SQL *dataType* *varName* = *val* ;*1
	Retrieve query result into host variables	EXEC SQL SELECT *col1*, ... INTO :*hostVar1*, ... FROM *tbl*;
		EXEC SQL FETCH NEXT FROM *curName* INTO :*hostVar1*, ... ;
	Indicators	EXEC SQL SELECT *val* INTO :*hostVar* :*valInd**2 FROM *test1* END-EXEC.
Use SQL descriptor areas *3	Allocate descriptor area	EXEC SQL ALLOCATE DESCRIPTOR *descId* ;
	Retrieve data into descriptor area	EXEC SQL FETCH NEXT FROM *curName* INTO SQL DESCRIPTOR *descId*;
		EXEC SQL FETCH *numOfRows* FROM *curName* INTO SQL DESCRIPTOR *descId*;
	Obtain field data from descriptor area	EXEC SQL GET DESCRIPTOR *descId* :*hostVar* = COUNT;
	Obtain field metadata from descriptor area	EXEC SQL GET DESCRIPTOR *descId* VALUE *colNum* :*hostVar* = *field* ;
	Deallocate descriptor area	EXEC SQL DEALLOCATE DESCRIPTOR *descId* ;

*1: For C only *2 :*valInd* will be negative if retrieved value is null, positive if it is truncated, or 0 otherwise *3: SQLDA is not supported by ECOBPG

Data mapping - C/COBOL ↔ PostgreSQL

PostgreSQL data type	C host variable data type	COBOL host variable data type
smallint	short	PIC S9([1-4]) { BINARY | COMP | COMP-5 }
integer	int	PIC S9([5-9]) { BINARY | COMP | COMP-5 }
bigint	long int	PIC S9([10-18]) { BINARY | COMP | COMP-5 }
decimal	decimal *1	PIC S9(m)V9(n) PACKED-DECIMAL PIC 9(m)V9(n) DISPLAY *3 PIC S9(m)V9(n) DISPLAY PIC S9(m)V9(n) DISPLAY SIGN { LEADING | TRAILING } [SEPARATE]
numeric	numeric *1	
real	float	COMP-1
double precision	double	COMP-2
smallserial	short	PIC S9([1-4]) { BINARY | COMP | COMP-5 }
serial	int	PIC S9([1-9]) { BINARY | COMP | COMP-5 }
bigserial	long int	PIC S9([10-18]) { BINARY | COMP | COMP-5 }
oid	unsigned int	PIC 9(9) { BINARY | COMP | COMP-5 }
character(n) varchar(n) text	char[n+1] VARCHAR[n+1] *2	PIC X(n) PIC X(n) VARYING
name	char[NAMEDATALEN]	PIC X(NAMEDATALEN)
timestamp	timestamp *1	PIC X(n) PIC X(n) VARYING
interval	interval *1	
date	date *1	
boolean	bool *2	BOOL*4
bytea	char *	PIC X(n) PIC X(n) VARYING

*1: Accessed via pgtypes libraries *2: Declared in ecpglib.h *3: If no USAGE is specified, host variable is regarded as DISPLAY
*4: Type definition 'PIC X(1)' is added during precompilation

Example

C

```
main() {
 // Declare host variables
 EXEC SQL BEGIN DECLARE SECTION;
    int     intItemId;
    char    strItemName[30];
 EXEC SQL END DECLARE SECTION;

 // Prepare SQL statement and declare cursor
 EXEC SQL PREPARE prepStmt FROM "SELECT item_id, item_name FROM inv WHERE qty < ?";
 EXEC SQL DECLARE curInv CURSOR FOR prepStmt;

 // Set exception handling and open cursor
 EXEC SQL WHENEVER NOT FOUND DO BREAK;
 EXEC SQL OPEN curInv USING 100;

 // For each row: retrieve and display
 while (1)
 {
    EXEC SQL FETCH NEXT FROM curInv INTO : intItemId, : strItemName;
    printf(" %d %s ", intItemId, strItemName);
 }

 // Close cursor and deallocate prepared statement
 EXEC SQL CLOSE curInv;
 EXEC SQL DEALLOCATE PREPARE prepStmt;
}
```

SQL

```
SELECT item_id, item_name
FROM   inv
WHERE qty < 100;
```

COBOL

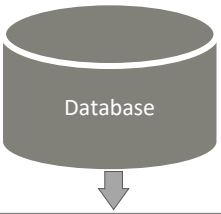
```
* Declare host variables
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01 WS-ITEM-ID        PIC S9(4).
 01 WS-ITEM-NAME  PIC X(30) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

* Prepare SQL statement and declare cursor
EXEC SQL PREPARE prepStmt FROM "SELECT item_id, item_name FROM inv WHERE qty < ?" END-EXEC.
EXEC SQL DECLARE curInv CURSOR FOR prepStmt END-EXEC.

* Set exception handling and open cursor
EXEC SQL WHENEVER NOT FOUND GOTO INV-FETCH-END END-EXEC.
EXEC SQL OPEN curInv USING 100 END-EXEC.

*For each row: retrieve and display
PERFORM WITH
 EXEC SQL FETCH NEXT FROM curInv INTO :WS-ITEM-ID, :WS-ITEM-NAME END-EXEC.
 DISPLAY WS-ITEM-ID WS-ITEM-NAME.
END-PERFORM.

* Close cursor and deallocate prepared statement
INV-FETCH-END.
EXEC SQL CLOSE curInv END-EXEC.
EXEC SQL DEALLOCATE PREPARE prepStmt END EXEC.
```



item_id	item_name
2741	Spark plug platinum
3642	Transmission fluid 1L
5732	Piston V8 sealed
7653	Carburetor recondit'd
8159	Crankshaft sensor

Contact
Fujitsu Limited
Email: enterprisepostgresql@fujitsu.com
Website: fast.fujitsu.com

2025-03-26 WW EN