

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 organisation

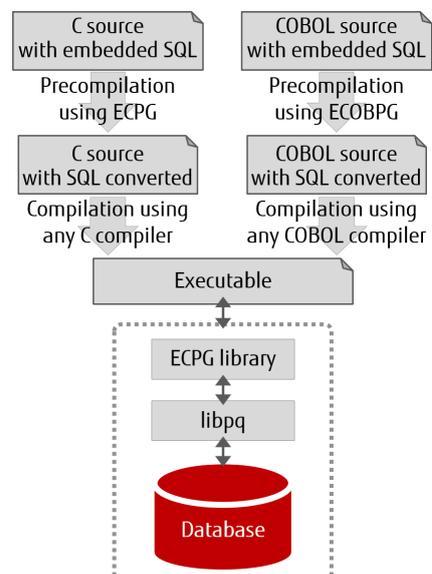
These languages have existed for several decades now, and proved their versatility and robustness with millions of lines of code written for all types of applications across multiple industries for all types of organisations, from enterprises to small businesses.

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.

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

*1: *connTarget* for C is [*dbName*][@*host*][:*port*], tcp:postgres://*host*[:*port*]/[*dbName*][?*options*], unix:postgres://*host*[:*port*]/[*dbName*][?*options*], for COBOL it is *dbName*@*host*:*port*, tcp:postgres://*host*:*port*/*dbName*[?*options*], unix:postgres://*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 <i>connTarget</i> [AS <i>connName</i>] [USER <i>user</i>];
	Choose connection	EXEC SQL { SET CONNECTION <i>connName</i> AT <i>connName</i> <i>sqlCmd</i> };
	Close connection	EXEC SQL DISCONNECT [<i>connName</i> DEFAULT CURRENT ALL];
Run SQL	Execute SQL	EXEC SQL <i>cmd</i> ;
	Declare cursors	EXEC SQL DECLARE <i>curName</i> CURSOR FOR <i>sqlCmd</i> ; EXEC SQL DECLARE <i>curName</i> CURSOR FOR <i>varPrepdStmt</i> ;
	Use cursors	EXEC SQL OPEN <i>curName</i> [USING { <i>val1</i> , ... SQL DESCRIPTOR <i>descld</i> }]; EXEC SQL FETCH <i>curName</i> INTO : <i>hostVar1</i> , ... ; ... EXEC SQL CLOSE <i>curName</i> ; EXEC SQL COMMIT;
	Manage transactions	EXEC SQL COMMIT [PREPARED <i>txId</i>]; EXEC SQL ROLLBACK [PREPARED <i>txId</i>]; EXEC SQL SET AUTOCOMMIT TO { ON OFF };
Declare prepared statements	EXEC SQL PREPARE <i>varPrepdStmt</i> FROM <i>prepdStmt</i> ;	
Execute prepared statements	EXEC SQL EXECUTE <i>varPrepdStmt</i> INTO : <i>hostVar1</i> , ... USING <i>val</i> ; EXEC SQL EXECUTE <i>prepdStmt</i> USING SQL DESCRIPTOR <i>descldIn</i> INTO SQL DESCRIPTOR <i>descldOut</i> ;	
Deallocate prepared statements	EXEC SQL DEALLOCATE PREPARE <i>varPrepdStmt</i> ;	

*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 : <i>varPrepdStmt</i> ;
	Statement with a result set	EXEC SQL EXECUTE <i>varPrepdStmt</i> INTO : <i>var1</i> , ... [USING <i>val1</i> , ...];
Use host variables	Declare host variables	EXEC SQL BEGIN DECLARE SECTION; <i>hostVarDeclaration</i> EXEC SQL END DECLARE SECTION; EXEC SQL <i>dataType varName = val</i> ; *1
	Retrieve query result into host variables	EXEC SQL SELECT <i>col1</i> , ... INTO : <i>hostVar1</i> , ... FROM <i>tbl</i> ; EXEC SQL FETCH NEXT FROM <i>curName</i> INTO : <i>hostVar1</i> , ...;
	Indicators	EXEC SQL SELECT <i>val</i> INTO : <i>hostVar</i> : <i>valInd</i> *2 FROM <i>test1</i> END-EXEC.
Use SQL descriptor areas *3	Allocate descriptor area	EXEC SQL ALLOCATE DESCRIPTOR <i>descId</i> ;
	Retrieve data into descriptor area	EXEC SQL FETCH NEXT FROM <i>curName</i> INTO SQL DESCRIPTOR <i>descId</i> ; EXEC SQL FETCH <i>numOfRows</i> FROM <i>curName</i> INTO SQL DESCRIPTOR <i>descId</i> ;
	Obtain field data from descriptor area	EXEC SQL GET DESCRIPTOR <i>descId</i> : <i>hostVar</i> = COUNT;
	Obtain field metadata from descriptor area	EXEC SQL GET DESCRIPTOR <i>descId</i> VALUE <i>colNum</i> : <i>hostVar</i> = <i>field</i> ;
	Deallocate descriptor area	EXEC SQL DEALLOCATE DESCRIPTOR <i>descId</i> ;

*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

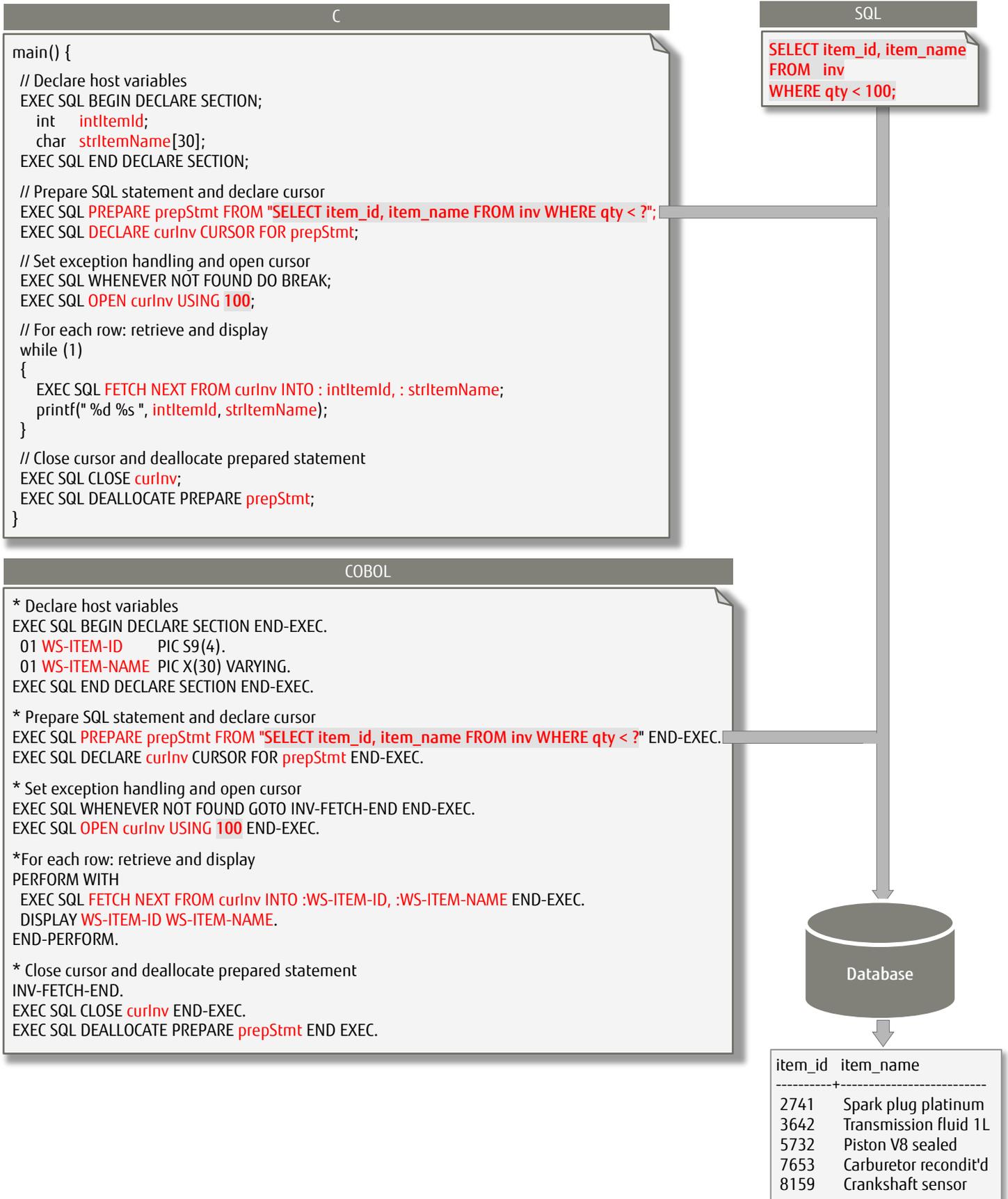
Data mapping - C/COBOL ↔ PostgreSQL

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 long int	PIC S9([10-18]) { BINARY COMP COMP-5 }
decimal	decimal *1	PIC S9(m)V9(n) PACKED-DECIMAL
numeric	numeric *1	PIC 9(m)V9(n) DISPLAY *3 PIC S9(m)V9(n) DISPLAY PIC S9(m)V9(n) DISPLAY SIGN { LEADING TRAILING } [SEPARATE]
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 long int	PIC S9([10-18]) { BINARY COMP COMP-5 }
oid	unsigned int	PIC 9(9) { BINARY COMP COMP-5 }
character(n)	char[n+1]	PIC X(n)
varchar(n)	VARCHAR[n+1] *2	PIC X(n) VARYING
text		
name	char[NAMEDATALEN]	PIC X(NAMEDATALEN)
timestamp	timestamp *1	PIC X(n)
interval	interval *1	PIC X(n) VARYING
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 ecgplib.h *3: If no USAGE is specified, host variable is regarded as DISPLAY

*4: Type definition 'PIC X(1)' is added during precompilation

Example

**Contact**

Fujitsu Australia Software Technology Pty Ltd
 Address: 14 Rodborough Rd
 Frenchs Forest NSW 2086
 Australia
 Phone: +61 2 9452 9191
 Email: postgresql@fast.au.fujitsu.com
 Website: www.postgresql.fastware.com

Copyright 2018 FUJITSU AUSTRALIA SOFTWARE TECHNOLOGY. Fujitsu, the Fujitsu logo and Fujitsu brand names are trademarks or registered trademarks of Fujitsu Limited in Japan and other countries. Other company, product and service names may be trademarks or registered trademarks of their respective owners. All rights reserved. No part of this document may be reproduced, stored or transmitted in any form without prior written permission of Fujitsu Australia Software Technology. Fujitsu Australia Software Technology endeavours to ensure the information in this document is correct and fairly stated, but does not accept liability for any errors or omissions.