FUJITSU

FUJITSU Enterprise Postgres 9.4

Application Development Guide

Windows/Linux

J2UL-2038-01ENZ0(00)
July 2015

Preface

Purpose of this document

This is a guide for the developers of Enterprise Postgres applications.

Intended readers

This document is intended for developers of applications that use Enterprise Postgres. Of the interfaces provided by Enterprise Postgres, this guide describes the PostgreSQL extended interface.

Readers of this document are also assumed to have general knowledge of:

- PostgreSQL
- SQL
- Linux

- PostgreSQL
- SQL
- Windows

Structure of this document

This document is structured as follows:

Chapter 1 Overview of the Application Development Function

Provides an overview of Enterprise Postgres application development.

Chapter 2 JDBC Driver

Explains how to use JDBC drivers.

Chapter 3 ODBC Driver

Explains how to use ODBC drivers.

Chapter 4 .NET Data Provider

Explains how to use .NET Data Provider.

Chapter 5 C Library (libpq)

Explains how to use C applications.

Chapter 6 Embedded SQL in C

Explains how to use embedded SQL in C.

Chapter 7 Embedded SQL in COBOL

Explains how to use embedded SQL in COBOL.

Chapter 8 SQL References

Explains the SQL statements which were extended in Enterprise Postgres development.

Chapter 9 Compatibility with Oracle Databases

Explains features that are compatible with Oracle databases.

Chapter 10 Java Applications using PL/extJava

Explains the development of Java applications using PL/extJava.

Chapter 11 Application Connection Switch Feature

Explains the application connection switch feature.

Chapter 12 Performance Tuning

Explains how to tune application performance.

Appendix A Precautions when Developing Applications

Provides some points to note about application development.

Appendix B Conversion Procedures Required due to Differences from Oracle Database

Explains how to convert from an Oracle database to Enterprise Postgres, within the scope noted in "Compatibility with Oracle Databases" from the following perspectives.

Appendix C Tables Used by the Features Compatible with Oracle Databases

Explains the tables used by the features compatible with Oracle databases.

Appendix D ECOBPG - Embedded SQL in COBOL

Explains application development using embedded SQL in COBOL.

Appendix E Quantitative Limits

This appendix explains limitations.

Appendix F Reference

Provides a reference for each interface.

Export restrictions

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Issue date and version


```
First  edition: July 2015
```


Copyright

Copyright 2015 FUJITSU LIMITED

Contents

Chapter 1 Overview of the Application Development Function 1
1.1 Support for National Characters 2
1.1.1 Literal 2
1.1.2 Data Type 3
1.1.3 Functions and Operator 3
1.2 Integration with Visual Studio 3
1.2.1 Relationship between .NET Framework and Enterprise Postgres 4
1.2.2 Automatic Application Generation 5
1.3 Compatibility with Oracle Database 6
1.4 Application Connection Switch Feature 6
1.4.1 Integration with Database Multiplexing 7
1.5 Notes on Application Compatibility 7
1.5.1 Checking Execution Results 7
1.5.2 Referencing System Catalogs 7
1.5.3 Using Functions 8

Chapter 2 JDBC Driver 9
2.1 Development Environment 9
2.1.1 Combining with JDK or JRE 9
2.2 Setup 9
2.2.1 Environment Settings 9
2.2.2 Message Language and Encoding System Used by Applications Settings 10
2.2.3 Settings for Encrypting Communication Data 10
2.3 Connecting to the Database 11
2.3.1 Using the DriverManager Class 11
2.3.2 Using the PGConnectionPoolDataSource Class 12
2.3.3 Using the PGXADataSource Class 13
2.4 Application Development 14
2.4.1 Relationship between the Application Data Types and Database Data Types 14
2.4.2 Statement Caching Feature 16
2.4.3 Creating Applications during Cluster Operations 17
2.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions 17
2.4.3.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode 17

Chapter 3 ODBC Driver 19
3.1 Development Environment 19
3.2 Setup 19
3.2.1 Registering ODBC Drivers 19
3.2.2 Registering ODBC Data Sources(for Windows) 21
3.2.2.1 Registering using GUI 21
3.2.2.2 Registering using commands 25
3.2.3 Registering ODBC Data Sources(for Linux) 28
3.2.4 Message Language and Encoding System Used by Applications Settings 30
3.3 Connecting to the Database 32
3.4 Application Development 32
3.4.1 Compiling Applications (for Windows) 32
3.4.2 Compiling Applications (for Linux) 32
3.4.3 Creating Applications during Cluster Operations 33
3.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions 33
3.4.3.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode 34

Chapter 4 .NET Data Provider 35
4.1 Development Environment 35
4.2 Setup 35
4.2.1 Setting Up .NET Data Provider 35

4.2.2 Setting Up the Visual Studio Integration Add-On....36
4.2.3 Message Language and Encoding System Used by Applications Settings....36
4.3 Connecting to the Database....37
4.3.1 Using NpgsqlConnection....37
4.3.2 Using NpgsqlConnectionStringBuilder....37
4.3.3 Using the ProviderFactory Class....38
4.3.4 Connection String....38
4.4 Application Development....41
4.4.1 Data Types....41
4.4.2 Relationship between Application Data Types and Database Data Types....43
4.4.3 Creating Applications during Cluster Operations....44
4.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions....44
4.4.3.2 Notes when Creating Applications during Cluster Operations in Shared Disk Mode....45
4.4.4 Notes....45

Chapter 5 C Library (libpq)....47
5.1 Development Environment....47
5.2 Setup....47
5.2.1 Environment Settings....47
5.2.2 Message Language and Encoding System Used by Applications Settings....48
5.2.3 Settings for Encrypting Communication Data....49
5.3 Connecting with the Database....49
5.4 Application Development....50
5.4.1 Compiling Applications....50
5.4.2 Creating Applications during Cluster Operations....51
5.4.2.1 Errors when an Application Connection Switch Occurs and Corresponding Actions....51
5.4.2.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode....51

Chapter 6 Embedded SQL in C....53
6.1 Development Environment....53
6.2 Setup....53
6.2.1 Environment Settings....53
6.2.2 Message Language and Encoding System Used by Applications Settings....53
6.2.3 Settings for Encrypting Communication Data....53
6.3 Connecting with the Database....54
6.4 Application Development....56
6.4.1 Support for National Character Data Types....56
6.4.2 Compiling Applications....56
6.4.3 Bulk INSERT....58
6.4.4 DECLARE STATEMENT....62
6.4.5 Creating Applications during Cluster Operations....63
6.4.5.1 Errors when an Application Connection Switch Occurs and Corresponding Actions....63
6.4.5.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode....64
6.4.6 Notes....64

Chapter 7 Embedded SQL in COBOL....65
7.1 Development Environment....65
7.2 Setup....65
7.2.1 Environment Settings....65
7.2.2 Message Language and Encoding System Used by Applications Settings....65
7.2.3 Settings for Encrypting Communication Data....65
7.3 Connecting with the Database....65
7.4 Application Development....67
7.4.1 Compiling Applications....67
7.4.2 Bulk INSERT....68
7.4.3 DECLARE STATEMENT....72

Chapter 8 SQL References....73

8.1 Expanded Trigger Definition Feature ... 73
8.1.1 CREATE TRIGGER ... 73
8.1.2 How to Define Triggers in pgAdmin ... 74

Chapter 9 Compatibility with Oracle Databases ... 75
9.1 Overview ... 75
9.2 Precautions when Using the Features Compatible with Oracle Databases ... 75
9.2.1 Notes on SUBSTR ... 75
9.2.2 Notes when Integrating with the Interface for Application Development ... 76
9.3 Queries ... 76
9.3.1 Outer Join Operator (+) ... 76
9.3.2 DUAL Table ... 79
9.4 SQL Function Reference ... 79
9.4.1 DECODE ... 79
9.4.2 SUBSTR ... 81
9.4.3 NVL ... 83
9.5 Package Reference ... 83
9.5.1 DBMS_OUTPUT ... 84
9.5.1.1 Description ... 84
9.5.1.2 Example ... 87
9.5.2 UTL_FILE ... 88
9.5.2.1 Registering and Deleting Directories ... 89
9.5.2.2 Description ... 90
9.5.2.3 Example ... 94
9.5.3 DBMS_SQL ... 95
9.5.3.1 Description ... 97
9.5.3.2 Example ... 101

Chapter 10 Java Applications using PL/extJava ... 104
10.1 Development Procedure ... 104
10.2 Points to Consider when Developing Java Applications ... 104
10.3 Connecting to the Database ... 105
10.4 Data type ... 105
10.4.1 Relationship between the Java Function Data Types and Java Application Data Types ... 106
10.4.2 Relationship between the Java Application Data Types and Database Data Types ... 107

Chapter 11 Application Connection Switch Feature ... 108
11.1 Connection Information for the Application Connection Switch Feature ... 108
11.2 Using the Application Connection Switch Feature ... 109
11.2.1 Using the JDBC Driver ... 109
11.2.2 Using the ODBC Driver ... 110
11.2.3 Using a .NET Data Provider ... 112
11.2.4 Using a Connection Service File ... 113
11.2.5 Using the C Library (libpq) ... 114
11.2.6 Using Embedded SQL ... 116
11.2.7 Using the psql Command ... 117

Chapter 12 Performance Tuning ... 119
12.1 Enhanced Query Plan Stability ... 119
12.1.1 Optimizer Hints ... 119
12.1.2 Locked Statistics ... 121

Appendix A Precautions when Developing Applications ... 125
A.1 Precautions when Using Functions and Operators ... 125
A.1.1 General rules of Functions and Operators ... 125
A.1.2 Errors when Developing Applications that Use Functions and/or Operators ... 125
A.2 Notes when Using Temporary Tables ... 126
A.3 Implicit Data Type Conversions ... 126

A.3.1 Function Argument ... 128
A.3.2 Operators ... 128
A.3.3 Storing Values ... 129
A.4 Notes on Using Index ... 129
A.4.1 Hush Index ... 129
A.4.2 SP-GiST Index ... 129
A.5 Notes on Using the psql Commands ... 130
A.5.1 Entering Multibyte Characters ... 130
A.6 Notes on Using Multibyte Characters in Definition Names ... 130

Appendix B Conversion Procedures Required due to Differences from Oracle Database ... 132
B.1 Outer Join Operator (Perform Outer Join) ... 132
B.1.1 Comparing with the ^= Comparison Operator ... 132
B.2 DECODE (Compare Values and Return Corresponding Results) ... 133
B.2.1 Comparing Numeric Data of Character String Types and Numeric Characters ... 133
B.2.2 Obtaining Comparison Result from more than 50 Conditional Expressions ... 133
B.2.3 Obtaining Comparison Result from Values with Different Data Types ... 134
B.3 SUBSTR (Extract a String of the Specified Length from Another String) ... 135
B.3.1 Specifying a Value Expression with a Data Type Different from the One that can be Specified for Function Arguments ... 135
B.3.2 Extracting a String with the Specified Format from a Datetime Type Value ... 136
B.3.3 Concatenating a String Value with a NULL value ... 136
B.4 NVL (Replace NULL) ... 137
B.4.1 Obtaining Result from Arguments with Different Data Types ... 137
B.4.2 Operating on Datetime/Numeric, Including Adding Number of Days to a Particular Day ... 138
B.4.3 Calculating INTERVAL Values, Including Adding Periods to a Date ... 138
B.5 DBMS_OUTPUT (Output Messages) ... 139
B.5.1 Outputting Messages Such As Process Progress Status ... 139
B.5.2 Receiving a Return Value from a Procedure (PL/SQL) Block (For GET_LINES) ... 141
B.5.3 Receiving a Return Value from a Procedure (PL/SQL) Block (For GET_LINE) ... 143
B.6 UTL_FILE (Perform File Operation) ... 144
B.6.1 Registering a Directory to Load and Write Text Files ... 144
B.6.2 Checking File Information ... 145
B.6.3 Copying Files ... 149
B.6.4 Moving/Renaming Files ... 150
B.7 DBMS_SQL (Execute Dynamic SQL) ... 151
B.7.1 Searching Using a Cursor ... 151

Appendix C Tables Used by the Features Compatible with Oracle Databases ... 157
C.1 UTL_FILE.UTL_FILE_DIR ... 157

Appendix D ECOBPG - Embedded SQL in COBOL ... 158
D.1 Precautions when Using Functions and Operators ... 158
D.2 Managing Database Connections ... 158
D.2.1 Connecting to the Database Server ... 159
D.2.2 Choosing a Connection ... 160
D.2.3 Closing a Connection ... 161
D.3 Running SQL Commands ... 161
D.3.1 Executing SQL Statements ... 161
D.3.2 Using Cursors ... 162
D.3.3 Managing Transactions ... 162
D.3.4 Prepared Statements ... 162
D.4 Using Host Variables ... 163
D.4.1 Overview ... 163
D.4.2 Declare Sections ... 163
D.4.3 Retrieving Query Results ... 164
D.4.4 Type Mapping ... 165
D.4.5 Handling Nonprimitive SQL Data Types ... 169
D.4.6 Indicators ... 173

D.5 Dynamic SQL ... 173
D.5.1 Executing Statements without a Result Set ... 173
D.5.2 Executing a Statement with Input Parameters ... 174
D.5.3 Executing a Statement with a Result Set ... 174
D.6 Using Descriptor Areas ... 175
D.6.1 Named SQL Descriptor Areas ... 175
D.7 Error Handling ... 177
D.7.1 Setting Callbacks ... 177
D.7.2 sqlca ... 178
D.7.3 SQLSTATE vs. SQLCODE ... 179
D.8 Preprocessor Directives ... 182
D.8.1 Including Files ... 182
D.8.2 The define and undef Directives ... 183
D.8.3 ifdef, ifndef, else, elif, and endif Directives ... 183
D.9 Processing Embedded SQL Programs ... 184
D.10 Large Objects ... 184
D.11 Embedded SQL Commands ... 184
D.11.1 ALLOCATE DESCRIPTOR ... 185
D.11.2 CONNECT ... 186
D.11.3 DEALLOCATE DESCRIPTOR ... 188
D.11.4 DECLARE ... 188
D.11.5 DESCRIBE ... 189
D.11.6 DISCONNECT ... 190
D.11.7 EXECUTE IMMEDIATE ... 191
D.11.8 GET DESCRIPTOR ... 192
D.11.9 OPEN ... 193
D.11.10 PREPARE ... 194
D.11.11 SET AUTOCOMMIT ... 195
D.11.12 SET CONNECTION ... 195
D.11.13 SET DESCRIPTOR ... 196
D.11.14 TYPE ... 197
D.11.15 VAR ... 198
D.11.16 WHENEVER ... 199
D.12 PostgreSQL Client Applications ... 200
D.12.1 ecobpg ... 200

Appendix E Quantitative Limits ... 202

Appendix F Reference ... 207
F.1 JDBC Driver ... 207
F.1.1 Java Programming Language API ... 207
F.1.2 PostgreSQL Fixed API ... 227
F.2 ODBC Driver ... 237
F.2.1 List of Supported APIs ... 237
F.3 .NET Data Provider ... 240
F.4 C Library (libpq) ... 257
F.5 Embedded SQL in C ... 257

Index ... 258

Chapter 1 Overview of the Application Development Function

The interface for application development provided by Enterprise Postgres is perfectly compatible with PostgreSQL.

Along with the PostgreSQL interface, Enterprise Postgres also provides the following extended interfaces:

- Support for National Characters

  In order to secure portability from mainframes and databases of other companies, Enterprise Postgres provides data types that support national characters. The national characters are usable from the client application languages.

  Refer to "1.1 Support for National Characters" for details.

- Integration with Visual Studio

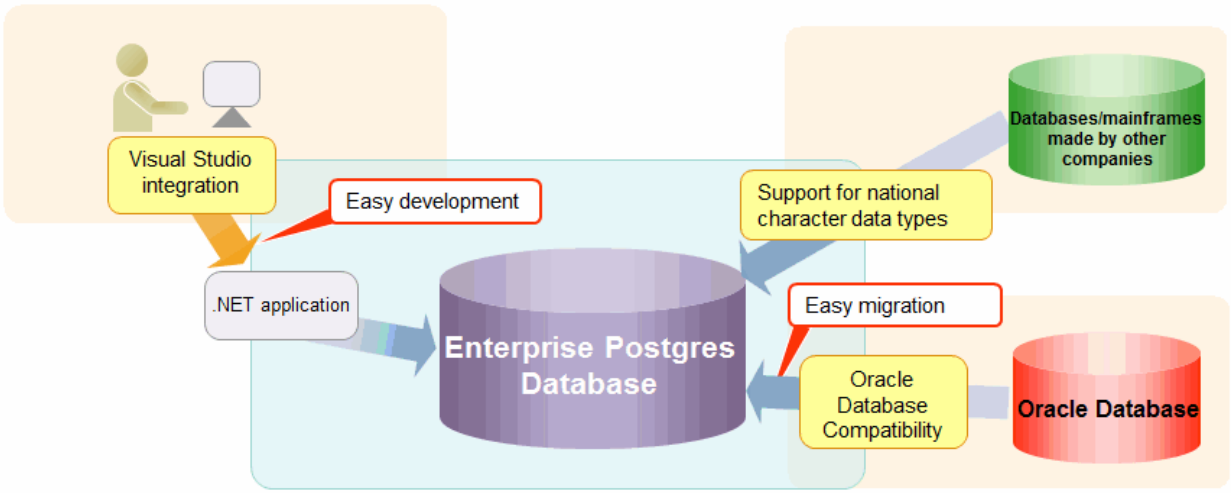
  By integrating with Visual Studio, applications can be created using a standard framework for the building of a database server.

  Refer to "1.2 Integration with Visual Studio" for details.

- Compatibility with Oracle Databases

  Compatibility with Oracle databases is offered. Use of the compatible features means that the revisions to existing applications can be isolated, and migration to open interfaces is made simpler.

  Refer to "1.3 Compatibility with Oracle Database" for details.



- Java applications using PL/extJava

  Using PL/extJava, it is possible to reduce network traffic and improve the job processing time by performing business logic, implemented via stored functions, on the database server instead of on the client.

  Refer to "Chapter 10 Java Applications using PL/extJava" for details.

- Application connection switch feature

  The application connection switch feature is provided to enable automatic connection to the target server when there are multiple servers with redundant configurations.

  Refer to "1.4 Application Connection Switch Feature" for details.

- Performance tuning

  The following features are provided to control SQL statement query plans:

  - Optimizer hints
  - Locked statistics

  Refer to "12.1 Enhanced Query Plan Stability" for details.

1.1 Support for National Characters

NCHAR type is provided as the data type to deal with national characters.

The NCHAR type can be used with Enterprise Postgres pgAdmin.

Point

- NCHAR can only be used when the character set of the database is UTF-8.
- NCHAR can be used in the places where CHAR can be used (function arguments, etc.).
- For applications handling NCHAR type data in the database, the data format is the same as CHAR type. Therefore, applications handling data in NCHAR type columns can also be used to handle data stored in CHAR type columns.

Note

Note the following in order to cast NCHAR type data as CHAR type.

- When comparing NCHAR type data where the length differs, ASCII spaces are used to fill in the length of the shorter NCHAR type data so that it can be processed as CHAR type data.
- Depending on the character set, the data size may increase by between 1.5 and 2 times.

1.1.1 Literal

Syntax

```
{ N | n }'[national character [ ...]]'
```

General rules

National character string literals consist of an 'N' or 'n', and the national character is enclosed in single quotation marks ('). Example: N'ABCDEF'

The data type is national character string type.

1.1.2 Data Type

Syntax

```
{ NATIONAL CHARCTER | NATIONAL CHAR | NCHAR } [ VARYING ][(length) ]
```

The data type of the NCHAR type column is as follows:

Data type specification format	Explanation
NATIONAL CHARACTER(n) NATIONAL CHAR(n) NCHAR(n)	National character string with a fixed length of n characters This will be the same as (1) if (n) is omitted. *n* is a whole number larger than 0.
NATIONAL CHARACTER VARYING(n) NATIONAL CHAR VARYING(n) NCHAR VARYING(n)	National character string with a variable length with a maximum of n characters Any length of national character string can be accepted when this is omitted. *n* is a whole number larger than 0.

General rules

NCHAR is the national character string type data type. The length is the number of characters.

The length of the national character string type is as follows:

- When VARYING is not specified, the length of national character strings is fixed and will be the specified length.
- When VARYING is specified, the length of national character strings will be variable.
  In this case, the lower limit will be 0 and the upper limit will be the value specified for length.
- NATIONAL CHARACTER, NATIONAL CHAR, and NCHAR each have the same meaning.

When the national character string to be stored is shorter than the declared upper limit, the NCHAR value is filled with spaces, whereas NCHAR VARYING is stored as is.

The upper limit for character storage is approximately 1GB.

1.1.3 Functions and Operator

Comparison operator

When a NCHAR type or NCHAR VARYING type is specified in a comparison operator, comparison is only possible between NCHAR types or NCHAR VARYING types.

String functions and operators

All of the string functions and operators that can be specified by a CHAR type can also be specified by a NCHAR type. The behavior of these string functions and operators is also the same as with CHAR type.

Pattern matching (LIKE, SIMILAR TO regular expression, POSIX regular expression)

The patterns specified when pattern matching with NCHAR types and NCHAR VARYING types specify the percent sign (%) and the underline (_).

The underline (_) means a match with one national character. The percent sign (%) means a match with any number of national characters 0 or over.

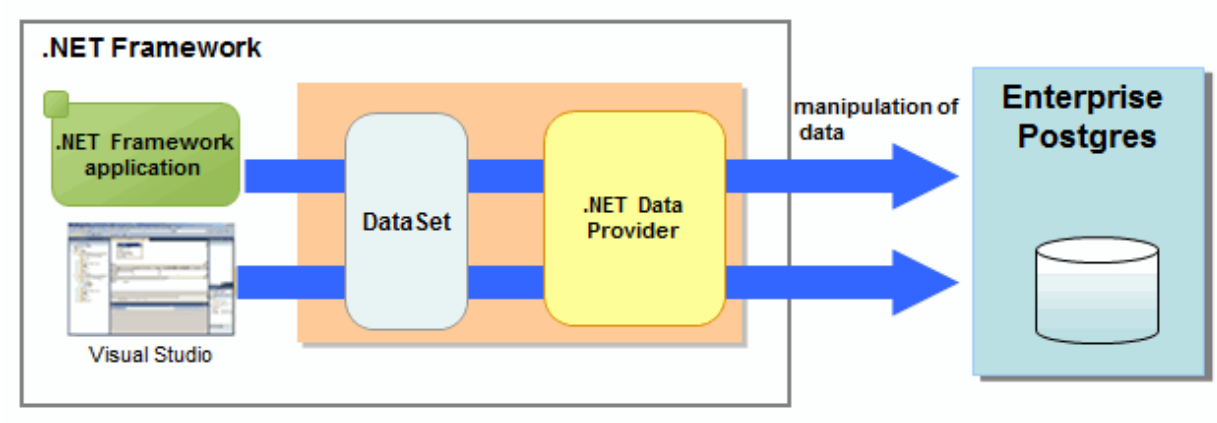
1.2 Integration with Visual Studio

When developing an application to access database server resources, you can create applications and build database server environments integrated with Microsoft Visual Studio.

Refer to "Chapter 4 .NET Data Provider" for information on integration with Visual Studio.

1.2.1 Relationship between .NET Framework and Enterprise Postgres

Enterprise Postgres provides .NET Data Provider, which is an interface for ADO.NET of .NET Framework. This enables you to select Enterprise Postgres as the connection destination database of ADO.NET and use the intuitive and efficient application development features of Visual Studio.



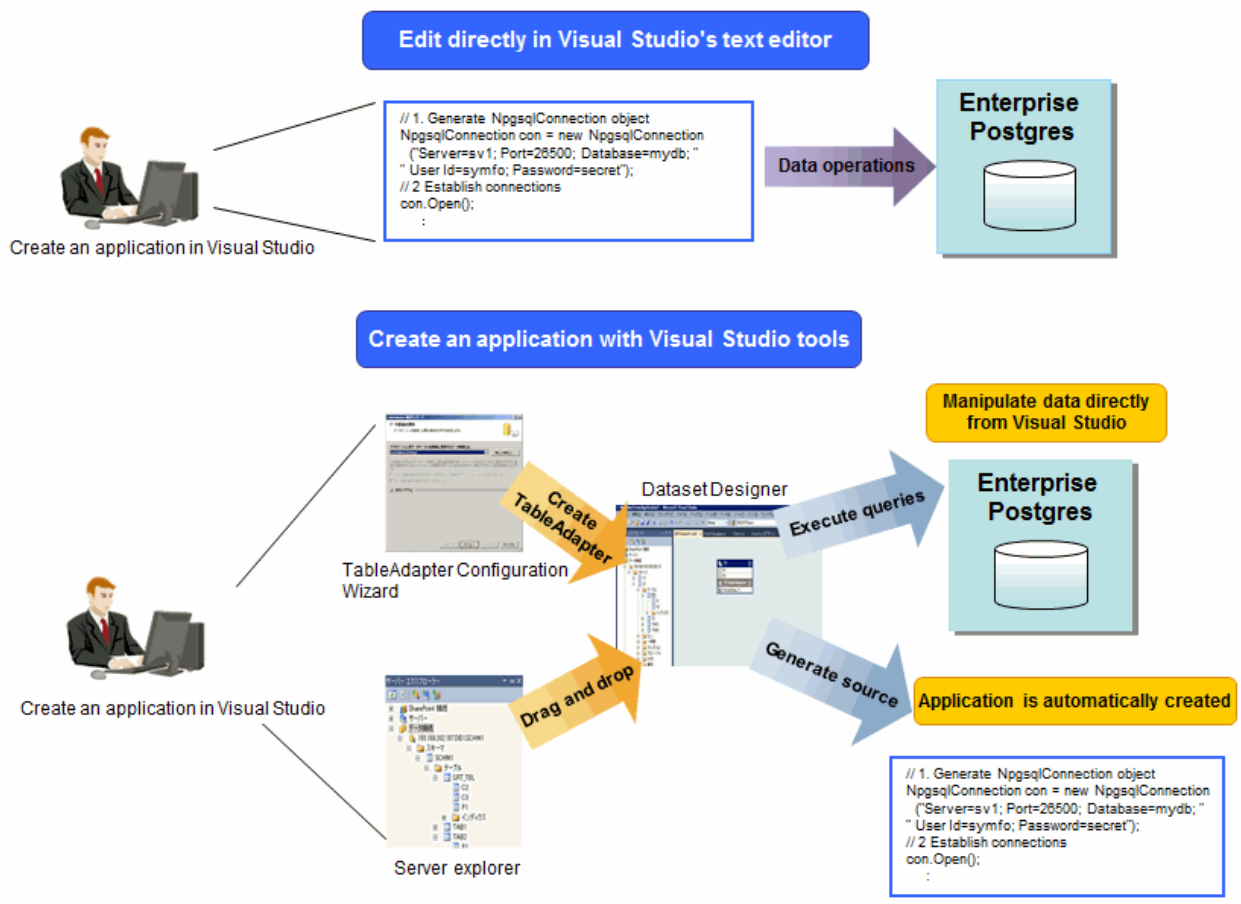
The following provides an overview of application development integrated with Visual Studio.

Edit directory in Visual Studio's text editor

By using a component specified in Visual Studio, applications to access database resources can be created manually.

Create an application with Visual Studio tools

By using basic drag-and-drop operations in the tools provided in Visual Studio, programs to access database resources can be generated automatically.

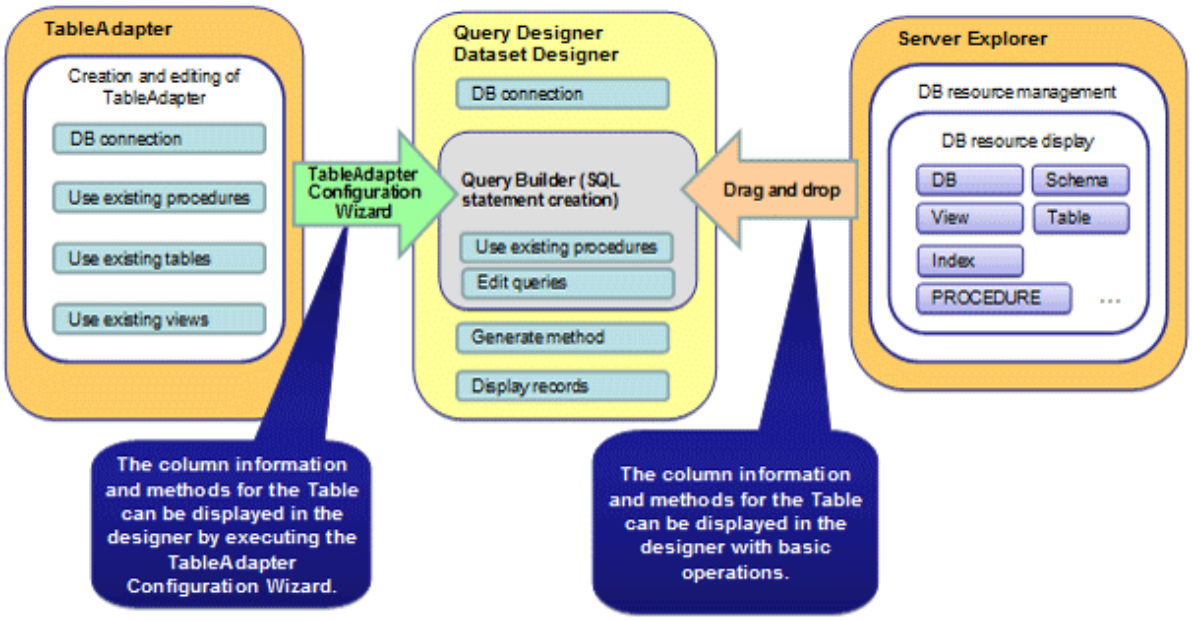


1.2.2 Automatic Application Generation

The Visual Studio tools used to automatically generate applications include TableAdapter and Server Explorer, which enable the following:

- Data manipulation of database resources with TableAdapter
- Management of database resources with Server Explorer

Whether you use TableAdapter or the Server Explorer, programs can be created with basic operations like drag and drop with the resources and tools that comprise Visual Studio.



The following features are available with TableAdapter and Server Explorer:

- Manipulation of database resources with TableAdapter
  - Generating queries using existing tables/views
  - Generating methods using existing tables/views
  - Generating queries using procedures
  - Generating methods using procedures
- Management of database resources with Server Explorer
  - Listing of database resources
  - Generating queries using existing tables/views
  - Generating methods using existing tables/views
  - Generating queries using procedures
  - Generating methods using procedures

1.3 Compatibility with Oracle Database

The following features have been extended in order to enhance compatibility with Oracle databases:

- Query (external join operator (+), DUAL table)
- Function (DECODE, SUBSTR, NVL)
- Built-in package (DBMS_OUTPUT, UTL_FILE, DBMS_SQL)

Refer to "Chapter 9 Compatibility with Oracle Databases" for information on the features compatible with Oracle databases.

1.4 Application Connection Switch Feature

The application connection switch feature enables automatic connection to the target server when there are multiple servers with redundant configurations.

Refer to " Chapter 11 Application Connection Switch Feature " for information on the application connection switch feature.

1.4.1 Integration with Database Multiplexing

The application connection switch feature is provided to enable automatic connection to the appropriate server when there are multiple servers with redundant configurations.

See

Refer to "Database Multiplexing Mode" in the Cluster Operation Guide for information on database multiplexing.

1.5 Notes on Application Compatibility

Enterprise Postgres upgrades contain feature improvements and enhancements that may affect the applications.

Accordingly, note the points below when developing applications, to ensure compatibility after upgrade.

- Checking execution results
- Referencing system catalogs
- Using functions

1.5.1 Checking Execution Results

Refer to SQLSTATE output in messages to check the SQL statements used in applications and the execution results of commands used during development.

See

Refer to Messages for information on the message content and number.

Refer to "PostgreSQL Error Codes" under "Appendixes" in the PostgreSQL Documentation for information on SQLSTATE.

1.5.2 Referencing System Catalogs

System catalogs can be used to obtain information about the Enterprise Postgres system and database objects.

However, system catalogs may change when the Enterprise Postgres version is upgraded. Also, there are many system catalogs that return information that is inherent to Enterprise Postgres.

Accordingly, reference the information schema defined in standard SQL (information_schema) wherever possible. Note also that queries specifying "*" in the selection list must be avoided to prevent columns being added.

See

Refer to "The Information Schema" under "Client Interfaces" in the PostgreSQL Documentation for details.

The system catalog must be referenced to obtain information not found in the information schema. Instead of directly referencing the system catalog in the application, define a view for that purpose. Note, however, that when defining the view, the column name must be clearly specified after the view name.

An example of defining and using a view is shown below.

Example

```
CREATE VIEW my_tablespace_view(spcname) AS SELECT spcname FROM pg_tablespace;
SELECT * FROM my_tablespace_view V1, pg_tables T1 WHERE V1.spcname = T1.tablespace;
```

If changes are made to a system catalog, the user will be able to take action by simply making changes to the view, without the need to make changes to the application.

The following shows an example of taking action by redefining a view as if no changes were made.

The pg_tablespace system catalog is redefined in response to the column name being changed from spcname to spacename.

Example

```
DROP VIEW my_tablespace_view;
CREATE VIEW my_tablespace_view(spcname) AS SELECT spacename FROM pg_tablespace;
```

1.5.3 Using Functions

The default functions provided with Enterprise Postgres enable a variety of operations and manipulations to be performed, and information to be obtained, using SQL statements.

However, it is possible that internal Enterprise Postgres functions, such as those relating to statistical information or for obtaining system-related information, may change as Enterprise Postgres versions are upgraded.

Accordingly, when using these functions, define them as new functions and then use the newly-defined functions in the applications.

An example of defining and using a function is shown below.

Example

```
CREATE FUNCTION my_func(relid regclass) RETURNS bigint LANGUAGE SQL AS 'SELECT
pg_relation_size(relid)';
SELECT my_func(2619);
```

If changes are made to a function, the user will be able to take action by simply redefining the function, without the need to make changes to the application.

The following shows an example of taking action by redefining a function as if no changes were made.

The pg_relation_size function is redefined after arguments are added.

Example

```
DROP FUNCTION my_func(regclass);
CREATE FUNCTION my_func(relid regclass) RETURNS bigint LANGUAGE SQL AS 'SELECT
pg_relation_size(relid,$$main$$)';
```

Chapter 2 JDBC Driver

This section describes how to use JDBC drivers.

2.1 Development Environment

This section describes application development using JDBC drivers and the runtime environment.

2.1.1 Combining with JDK or JRE

Refer to Installation and Setup Guide for Client for information on combining with JDK or JRE where JDBC drivers can operate.

2.2 Setup

This section describes the environment settings required to use JDBC drivers and how to encrypt communication data.

2.2.1 Environment Settings

Configuration of the CLASSPATH environment variable is required as part of the runtime environment for JDBC drivers.

The name of the JDBC driver file is as follows:

- If using JDK 6 or JRE 6

```
postgresql-jdbc4.jar
```

- If using JDK 7, JRE 7, JDK 8, or JRE 8

```
postgresql-jdbc41.jar
```

The examples below show how to set the CLASSPATH environment variable if JDK 6 or JRE 6 is used.

If JDK 7, JRE 7, JDK 8, or JRE 8 is used, only the name of the JDBC driver file will be different. The method for configuring the CLASSPATH environment variable is the same.

L

- Linux (32-bit)
  - Setting example (TC shell)

```
setenv CLASSPATH /opt/fsepclient32/jdbc/lib/postgresql-jdbc4.jar:${CLASSPATH}
```

  - Setting example (bash)

```
CLASSPATH=/opt/fsepclient32/jdbc/lib/postgresql-jdbc4.jar:$CLASSPATH;export
CLASSPATH
```

- Linux (64-bit)
  - Setting example (TC shell)

```
setenv CLASSPATH /opt/fsepclient64/jdbc/lib/postgresql-jdbc4.jar:${CLASSPATH}
```

  - Setting example (bash)

```
CLASSPATH=/opt/fsepclient64/jdbc/lib/postgresql-jdbc4.jar:$CLASSPATH;export
CLASSPATH
```

- Windows (32-bit)
  - Setting example

```
set CLASSPATH=C:\Program Files\Fujitsu\fsepclient32\JDBC\lib\postgresql-jdbc4.jar;
%CLASSPATH%
```

- Windows (64-bit)
  - Setting example (when Enterprise Postgres Client 32-bit is installed)

```
set CLASSPATH=C:\Program Files (x86)\Fujitsu\fsepclient32\JDBC\lib\postgresql-
jdbc4.jar;%CLASSPATH%
```

  - Setting example (when Enterprise Postgres Client 64-bit is installed)

```
set CLASSPATH=C:\Program Files\Fujitsu\fsepclient64\JDBC\lib\postgresql-jdbc4.jar;
%CLASSPATH%
```

2.2.2 Message Language and Encoding System Used by Applications Settings

If the JDBC driver is used, it will automatically set the encoding system on the client to UTF-8, so there is no need to configure this.

See

Refer to "Automatic Character Set Conversion Between Server and Client" in "Server Administration" in the PostgreSQL Documentation for information on encoding systems.

Language settings

You must match the language settings for the application runtime environment with the message locale settings of the database server.

Set language in the "user.language" system property.

Example

Example of running a Java command with system property specified

```
java -Duser.language=en TestClass1
```

2.2.3 Settings for Encrypting Communication Data

When using the communication data encryption feature to connect to the database server, set as follows:

Settings for encrypting communication data for connection to the server

This section describes how to create applications for encrypting communication data.

Set the property of the SSL parameter to "true" to encrypt. The default for the SSL parameter is "false".

Example

- Setting example 1

```
String url = "jdbc:postgresql://sv1/test";
Properties props = new Properties();
```

```
props.setProperty("user","fsepuser");
props.setProperty("password","secret");
props.setProperty("ssl","true");
Connection conn = DriverManager.getConnection(url, props);
```

- Setting example 2

```
String url = "jdbc:postgresql://sv1/test?user=fsepuser&password=secret&ssl=true";
Connection conn = DriverManager.getConnection(url);
```

To prevent spoofing of the database server, you need to use the keytool command included with Java to import the CA certificate to the Java keystore.

Refer to JDK documentation and the Oracle website for details.

Note

There is no need to set the ssl parameter if the connection string of the DriverManager class is specified, or if the sslmode parameter is specified in the data source, such as when the application connection switch feature is used. If the ssl parameter is set, the value in the sslmode parameter will be enabled.

See

Refer to "Secure TCP/IP Connections with SSL" in "Server Administration" in the PostgreSQL Documentation for information on encrypting communication data.

2.3 Connecting to the Database

This section explains how to connect to a database.

- Using the DriverManager Class
- Using the PGConnectionPoolDataSource Class
- Using the PGXADataSource Class

Note

Do not specify "V2" for the "*protocolVersion*" of the connection string.

2.3.1 Using the DriverManager Class

To connect to the database using the DriverManager class, first load the JDBC driver, then specify the connection string as a URI in the API of the DriverManager class.

Load the JDBC driver

Specify org.postgresql.Driver.

Connection string

URI connection is performed as follows:

```
jdbc:postgresql://host:port/database?
user=user&password=password1&loginTimeout=loginTimeout&socketTimeout=socketTimeout
```

Argument	Description
host	Specify the host name for the connection destination.
port	Specify the port number for the database server. The default is "27500".
database	Specify the database name.
user	Specify the username that will connect with the database. If this is omitted, the username logged into the operating system that is executing the application will be used.
password	Specify a password when authentication is required.
loginTimeout	Specify the timeout for connections (in units of seconds). Specify a value between 0 and 9223372036854775. There is no limit set if you set 0 or an invalid value. An error occurs when a connection cannot be established within the specified time.
socketTimeout	Specify the timeout for communication with the server (in units of seconds). Specify a value between 0 and 2147483647. There is no limit set if you set 0 or an invalid value. An error occurs when data is not received from the server within the specified time.
maxStatements	Specify the number of statements to be cached. Specify an integer value between 0 and 2147483647. The default is 0. The statement caching feature will be disabled if 0 is specified. An error will not occur if an invalid value is specified, however, the behavior will be as if 0 has been specified. Refer to "2.4.2 Statement Caching Feature" for information on this feature.

Example

Code examples for applications

```
import java.sql.*;
...
Class.forName("org.postgresql.Driver");
String url = "jdbc:postgresql://sv1:27500/mydb?
user=myuser&password=myuser01&loginTimeout=20&socketTimeout=20&maxStatements=20";
Connection con = DriverManager.getConnection(url);
```

2.3.2 Using the PGConnectionPoolDataSource Class

To connect to databases using data sources, specify the connection information in the properties of the data source.

Method description

Argument	Description
setServerName	Specify the host name for the connection destination.
setPortNumber	Specify the port number for the database server. The default is "27500".
setDatabaseName	Specify the database name.
setUser	Specify the username of the database.

Argument	Description
	By default, the name used will be that of the user on the operating system that is executing the application.
setPassword	Specify a password for server authentication.
setLoginTimeout	Specify the timeout for connections (in units of seconds). Specify a value between 0 and 9223372036854775. There is no limit set if you set 0 or an invalid value. An error occurs when a connection cannot be established within the specified time.
setSocketTimeout	Specify the timeout for communication with the server (in units of seconds). Specify a value between 0 and 2147483647. There is no limit set if you set 0 or an invalid value. An error occurs when data is not received from the server within the specified time.
setMaxStatements	Specify the number of statements to be cached. Specify an integer value between 0 and 2147483647. The default is 0. The statement caching feature will be disabled if 0 is specified. An error will not occur if an invalid value is specified, however, the behavior will be as if 0 has been specified. Refer to "2.4.2 Statement Caching Feature" for information on this feature.

Example

Code examples for applications

```
import java.sql.*;
import org.postgresql.ds.PGConnectionPoolDataSource;
...
PGConnectionPoolDataSource source = new PGConnectionPoolDataSource();
source.setServerName("sv1");
source.setPortNumber(27500);
source.setDatabaseName("mydb");
source.setUser("myuser");
source.setPassword("myuser01");
source.setLoginTimeout(20);
source.setSocketTimeout(20);
source.setMaxStatements(20);
...
Connection con = source.getConnection();
```

2.3.3 Using the PGXADataSource Class

To connect to databases using data sources, specify the connection information in the properties of the data source.

Method description

Argument	Description
setServerName	Specify the host name for the connection destination.
setPortNumber	Specify the port number for the database server. The default is "27500".
setDatabaseName	Specify the database name.
setUser	Specify the username that will connect with the database.

Argument	Description
	If this is omitted, the name used will be that of the user on the operating system that is executing the application.
setPassword	Specify a password when authentication by a password is required.
setLoginTimeout	Specify the timeout for connections. The units are seconds. Specify a value between 0 and 9223372036854775. There is no limit set if you set 0 or an invalid value. An error occurs when a connection cannot be established within the specified time.
setSocketTimeout	Specify the timeout for communication with the server. The units are seconds. Specify a value between 0 and 2147483647. There is no limit set if you set 0 or an invalid value. An error occurs when data is not received from the server within the specified time.
setMaxStatements	Specify the number of statements to be cached. Specify an integer value between 0 and 2147483647. The default is 0. The statement caching feature will be disabled if 0 is specified. An error will not occur if an invalid value is specified, however, the behavior will be as if 0 has been specified. Refer to "2.4.2 Statement Caching Feature" for information on this feature.

Example

Code examples for applications

```
import java.sql.*;
import org.postgresql.xa.PGXADataSource;
...
PGXADataSource source = new PGXADataSource();
source.setServerName("sv1");
source.setPortNumber(27500);
source.setDatabaseName("mydb");
source.setUser("myuser");
source.setPassword("myuser01");
source.setLoginTimeout(20);
source.setSocketTimeout(20);
source.setMaxStatements(20);
...
Connection con = source.getConnection();
```

2.4 Application Development

This section describes the data types required when developing applications that will be connected with Enterprise Postgres.

2.4.1 Relationship between the Application Data Types and Database Data Types

The following table shows the correspondence between data types in applications and data types in databases.

Data type on the server	Java data type	Data types prescribed by java.sql.Types
character	String	java.sql.Types.CHAR

Data type on the server	Java data type	Data types prescribed by java.sql.Types
national character	String	java.sql.Types.NCHAR
character varying	String	java.sql.Types.VARCHAR
national character varying	String	java.sql.Types.NVARCHAR
text	String	java.sql.Types.VARCHAR
bytea	byte[]	java.sql.Types.BINARY
smallint	short	java.sql.Types.SMALLINT
integer	int	java.sql.Types.INTEGER
bigint	long	java.sql.Types.BIGINT
smallserial	short	java.sql.Types.SMALLINT
serial	int	java.sql.Types.INTEGER
bigserial	long	java.sql.Types.BIGINT
real	float	java.sql.Types.REAL
double precision	double	java.sql.Types.DOUBLE
numeric	java.math.BigDecimal	java.sql.Types.NUMERIC
decimal	java.math.BigDecimal	java.sql.Types.DECIMAL
money	String	java.sql.Types.OTHER
date	java.sql.Date	java.sql.Types.DATE
time with time zone	java.sql.Time	java.sql.Types.TIME
time without time zone	java.sql.Time	java.sql.Types.TIME
timestamp without time zone	java.sql.Timestamp	java.sql.Types.TIMESTAMP
timestamp with time zone	java.sql.Timestamp	java.sql.Types.TIMESTAMP
interval	org.postgresql.util.PGInterval	java.sql.Types.OTHER
boolean	boolean	java.sql.Types.BIT
bit	boolean	java.sql.Types.BIT
bit varying	org.postgresql.util.Pgobject	java.sql.Types.OTHER
oid	long	java.sql.Types.BIGINT
xml	java.sql.SQLXML	java.sql.Types.SQLXML
array	java.sql.Array	java.sql.Types.ARRAY
uuid	java.util.UUID	java.sql.Types.OTHER
point	org.postgresql.geometric.Pgpoint	java.sql.Types.OTHER
box	org.postgresql.geometric.Pgbox	java.sql.Types.OTHER
lseg	org.postgresql.geometric.Pglseg	java.sql.Types.OTHER
path	org.postgresql.geometric.Pgpath	java.sql.Types.OTHER
polygon	org.postgresql.geometric.PGpolygon	java.sql.Types.OTHER
circle	org.postgresql.geometric.PGcircle	java.sql.Types.OTHER
json	org.postgresql.util.PGobject	java.sql.Types.OTHER
Network address type (inet,cidr,macaddr)	org.postgresql.util.PGobject	java.sql.Types.OTHER

Data type on the server	Java data type	Data types prescribed by java.sql.Types
Types related to text searches (svector, tsquery)	org.postgresql.util.PGobject	java.sql.Types.OTHER
Enumerated type	org.postgresql.util.PGobject	java.sql.Types.OTHER
Composite type	org.postgresql.util.PGobject	java.sql.Types.OTHER
Range type	org.postgresql.util.PGobject	java.sql.Types.OTHER

All data types allow the use of the getString() method to acquire string values.

2.4.2 Statement Caching Feature

The statement caching feature caches SQL statements for each individual connection. This means that when an SQL statement with an identical string is next executed, the analysis and creation of the statement can be skipped. This improves performance in cases such as when an SQL statement with an identical string is executed within a loop or method that is executed repeatedly. Furthermore, the statement caching feature can be combined with the connection pooling feature to further enhance performance.

The mechanism used by the statement caching feature is described below:

- When determining if an SQL statement contains an identical string, the check is case-sensitive.
- Only SQL statements prepared using the prepareStatement method or the prepareCall method are targets for statement caching.
- When the close method of the PreparedStatement class or the CallableStatement class is executed, the specified SQL statement is registered in the cache. When the prepareStatement method or the prepareCall method is executed, the feature checks if the specified SQL statement exists in the cache. If it exists, the analysis and creation processes for the statement can be skipped.
- Specify the upper limit of the cache using the setMaxStatements method of the PGConnectionPoolDataSource class or the PGXADataSource class. The statement will not be cached unless a valid value other than 0 is specified. If the upper limit of the cache is reached, the least recently used cache will be discarded (this is known as a Least Recently Used (LRU) system).
- If "false" is specified in the setPoolable(boolean) method of the PreparedStatement class, the SQL statement will not be registered in the cache, even if the close method of the prepared SQL statement is executed.

Cache registration controls

You can configure whether to cache SQL statements using the setPoolable(boolean) method of the PreparedStatement class when the statement caching feature is enabled.

Values that can be configured are shown below:

false

SQL statements will not be cached, even when the statement caching feature is enabled.

true

SQL statements will be cached if the statement caching feature is enabled.

Number of caches

Specify the number of SQL statements with different strings per connection.

Note

- The specified values may become unsuitable when changes are made to the application, so periodically execute pg_stat_statement to check and update to suitable values if required.

- Cached SQL statement information will consume a certain amount of memory, so consideration must be given to the upper limit for memory consumption. A guide to estimating the memory consumed by the statement caching feature is shown below:

  - Client

```
connsPerAppProc x numOfCaches x 2 KB (*1)
```

  - Server

```
maxConnsPerAppProc x numOfCaches x 10 KB (*1)
```

    *1: Approximate memory consumption per SQL statement

2.4.3 Creating Applications during Cluster Operations

This section explains points to consider when creating applications during cluster operations.

Refer to the Cluster Operation Guide for information on cluster operations.

2.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs during cluster operations, explicitly close the connection and then reestablish the connection or reexecute the application.

The table below shows errors that may occur during a switch, and the corresponding action to take.

State		Error information (*2)	Action
Node failure (server failure) or Enterprise Postgres system failure (*1)	Failure occurs during access	57P01 08006	After the switch is complete, reestablish the connection, or reexecute the application.
	Accessed during node/system failure	08001	
Switch to the standby node (standby server) (*1)	Switched during access	57P01 08006	
	Accessed during switch	08001	

*1: Terms enclosed in () are terms used in database multiplexing mode.

*2: Return value of the getSQLState method of SQLException.

2.4.3.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode

In the application, specify the same IP address as the one that was used when creating Gls or takeover network resources. This will enable the application to be executed if a state transition occurs, simply by reexecuting the application without modification.

See

Refer to "Creating Gls or Takeover Network Resources" in the Cluster Operation Guide for details.

Chapter 3 ODBC Driver

This section describes application development using ODBC drivers.

3.1 Development Environment

Applications using ODBC drivers can be developed using ODBC interface compatible applications, such as Access, Excel, and Visual Basic.

Refer to the manuals for the programming languages corresponding to the ODBC interface for information about the environment for development.

Enterprise Postgres supports ODBC 3.5.

3.2 Setup

You need to set up PsqlODBC, which is an ODBC driver, in order to use applications that use ODBC drivers with Enterprise Postgres. PsqlODBC is included in the Enterprise Postgres client package.

The following describes how to register the ODBC drivers and the ODBC data source.

3.2.1 Registering ODBC Drivers

When using the ODBC driver on a Linux platform, register the ODBC driver using the following procedure:

1. Installing the ODBC driver manager (unixODBC)

   Information

   - Enterprise Postgres supports unixODBC Version 2.3 or later.

     You can download unixODBC from the following site:

     http://www.unixodbc.org/

   - To execute unixODBC, you must first install libtool 2.2.6 or later.

     You can download libtool from the following website:

     http://www.gnu.org/software/libtool/

   [Note]

   - ODBC driver operation is supported.
   - unixODBC operation is not supported.

2. Registering ODBC drivers

   Edit the ODBC driver manager (unixODBC) odbcinst.ini file.

   Information

   [location of the odbcinst.ini file]

   ```
   unixOdbcInstallDir/etc/odbcinst.ini
   ```

   Set the following content:

Definition name	Description	Setting value

[Driver name]	ODBC driver name	Set the name of the ODBC driver.
		
		Select the two strings below that correspond to the application type. Concatenate the strings with no spaces, enclose in "[ ]", and then specify this as the driver name.
		
		Note
		
		The placeholders shown below are enclosed in angle brackets '<>' to avoid confusion with literal text. Do not include the angle brackets in the string.
		
		- Application architecture
		  - For data sources used by 32-bit applications
		  "EnterprisePostgres *<enterprisePostgresClientVersAndLvl>*"
		  - For data sources used by 64-bit applications
		  "EnterprisePostgres *<enterprisePostgresClientVersAndLvl>*x64"
		- Encoding system used by the application
		  - In Unicode (only UTF-8 can be used)
		  "unicode"
		  - Other than Unicode
		  "ansi"
		
		Example: In a 32-bit application, where the encoding system used by the application is Unicode:
		
		"[EnterprisePostgres *<enterprisePostgresClientVersAndLvl>*unicode]"

Description	Description of the ODBC driver	Specify a supplementary description for the current data source. Any description may be set.

Driver	Path of the ODBC driver (32-bit)	Set the path of the ODBC driver (32-bit).
		
		- If the encoding system is Unicode:
		  enterprisePostgresClientInstallDir/odbc/lib/psqlodbcw.so
		- If the encoding system is other than Unicode:
		  enterprisePostgresClientInstallDir/odbc/lib/psqlodbca.so

Driver64	Path of the ODBC driver (64-bit)	Set the path of the ODBC driver (64-bit). Setting is not required when you are using a 32-bit operating system. - If the encoding system is Unicode: *enterprisePostgresClientInstallDir*/odbc/lib/psqlodbcw.so

Definition name	Description	Setting value
		- If the encoding system is other than Unicode: *enterprisePostgresClientInstallDir*/odbc/lib/psqlodbca.so
FileUsage	Use of the data source file	Specify 1.
Threading	Level of atomicity secured for connection pooling	Specify 2.

Example

- Setting example when using 32-bit ODBC driver on 32-bit Linux

```
[EnterprisePostgres9.4unicode]
Description = Enterprise Postgres 9.4 unicode driver
Driver      = /opt/fsepclient32/odbc/lib/psqlodbcw.so
FileUsage   = 1
Threading   = 2
```

- Setting example when using 64-bit ODBC driver on 64-bit Linux

```
[EnterprisePostgres9.4x64unicode]
Description = Enterprise Postgres 9.4 x64 unicode driver
Driver64    = /opt/fsepclient64/odbc/lib/psqlodbcw.so
FileUsage   = 1
Threading   = 2
```

- Setting example when using both 32-bit and 64-bit ODBC drivers on 64-bit Linux

```
[EnterprisePostgres9.4unicode]
Description = Enterprise Postgres 9.4 unicode driver
Driver      = /opt/fsepclient32/odbc/lib/psqlodbcw.so
FileUsage   = 1
Threading   = 2

[EnterprisePostgres9.4x64unicode]
Description = Enterprise Postgres 9.4 x64 unicode driver
Driver64    = /opt/fsepclient64/odbc/lib/psqlodbcw.so
FileUsage   = 1
Threading   = 2
```

3.2.2 Registering ODBC Data Sources(for Windows)

This section describes how to register ODBC data sources.

There are the following two ways to register ODBC data sources on Windows.

3.2.2.1 Registering using GUI

This section describes how to start the [ODBC Data Source Administrator] and register ODBC data sources.

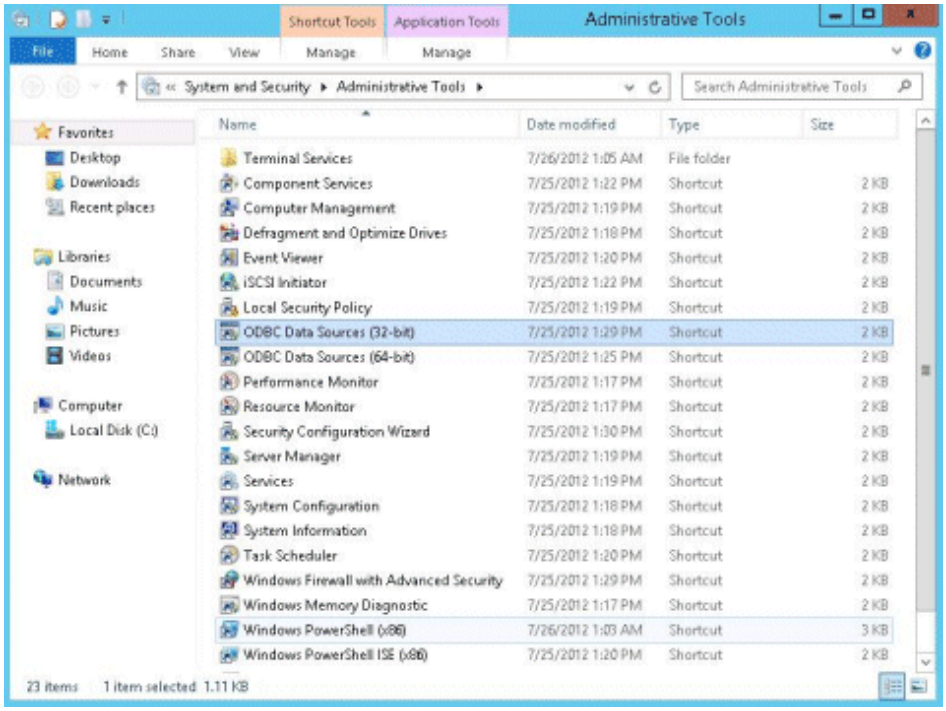
Use the following procedure to register ODBC data sources:

1. Start the [ODBC Data Source Administrator].

   Select [Start] >> [Control Panel] >> [Administrative Tools] >> [ODBC Data Source Administrator].

Example

This is an example of starting [ODBC Data Sources (32-bit)] from [Administrative Tools] in Windows Server 2012.

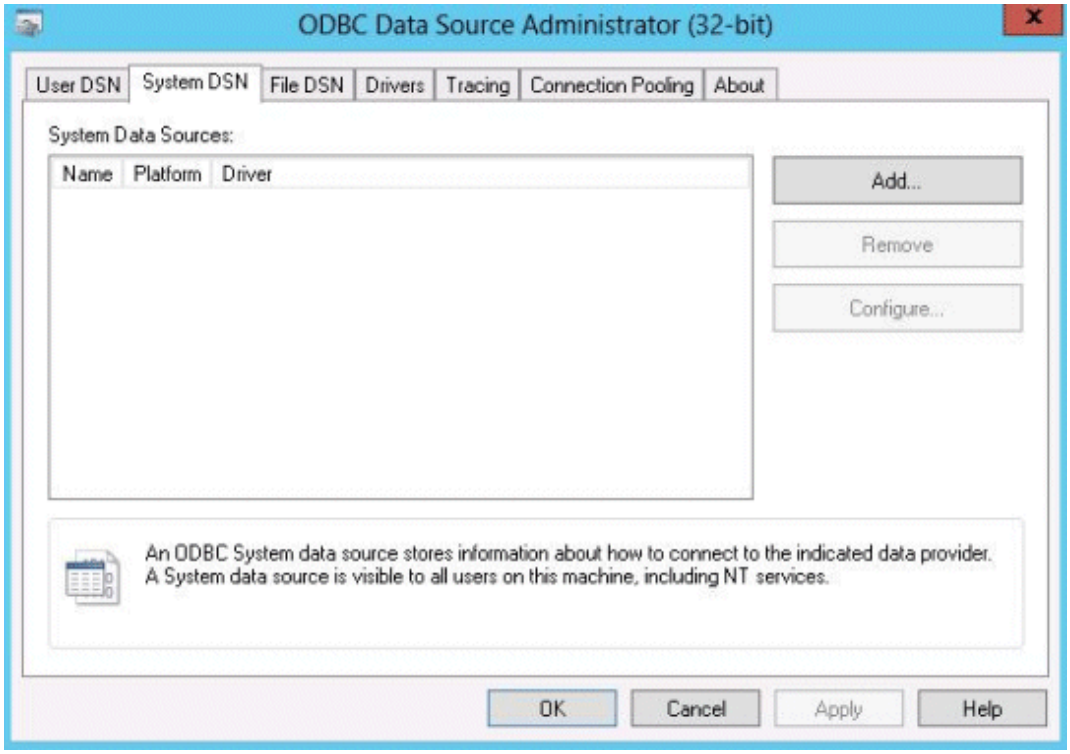


Note

To register data sources for 32-bit applications in Windows(R) for 64-bit, execute the ODBC administrator (odbcad32.exe) for 32-bit, as shown below.

```
%SYSTEMDRIVE%\WINDOWS\SysWOW64\odbcad32.exe
```

2. When only the current user is to use the ODBC data source, select [User DSN]. When all users using the same computer are to use the ODBC data source, select [System DSN].

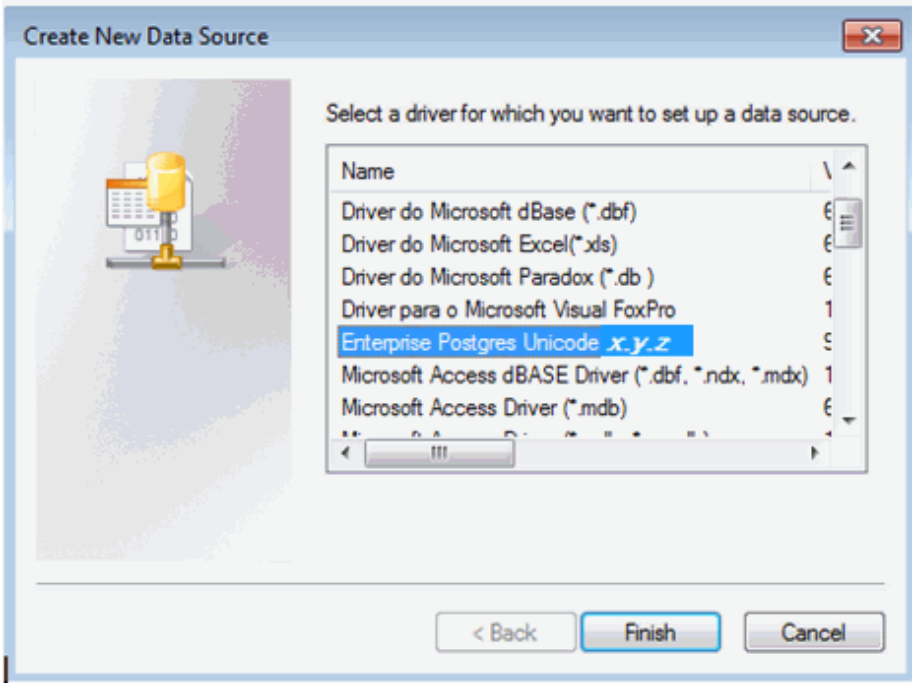


3. Click [Add].

4. Select one of the following drivers from the list of available ODBC drivers displayed in [Create New Data Source], and then click [Finish]. The notation "*x.y.z*" indicates the version and level of the Enterprise Postgres client feature.

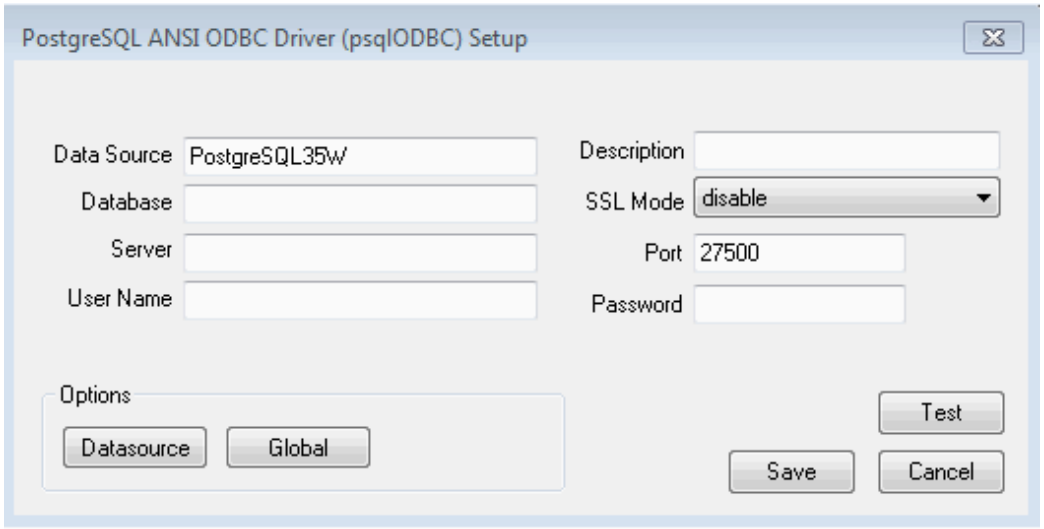
   - Enterprise Postgres Unicode *x.y z*
     Select this driver if using Unicode as the application encoding system.

- Enterprise Postgres ANSI *x.y.z*
  Select this driver if using other than Unicode as the application encoding system.



*1: The notation "*x.y.z*" indicates the version and level of the Enterprise Postgres client feature.

5. The [PostgreSQL ANSI ODBC Driver (psqlODBC) Setup] window is displayed. Enter or select the required items, then click [Save].



Set the following content:

Definition name	Setting value
Data Source	Specify the data source name to be registered in the ODBC driver manager. The application will select the name specified here and connect with the Enterprise Postgres database. This parameter cannot be omitted. Specify the following characters up to 32 bytes.

Definition name	Setting value
	- National characters - Alphanumerics - "_", "<", ">", "+", "`", "|", "~", "'" " "&" , """, "#" , "$" , "%" , "-" , "^" , ":" , "/" , "."
Description	Specify a supplementary description for the current data source. Specify characters up to 255 bytes. - National characters - Alphanumerics
Database	Specify the database name to be connected.
SSLMode	Specify to encrypt communications. The default is "disable". The setting values for SSLMode are as follows: - disable: Connect without SSL - allow: Connect without SSL, and if it fails, connect using SSL - prefer: Connect using SSL, and if it fails, connect without SSL - require: Connect always using SSL - verify-ca: Connect using SSL, and use a certificate issued by a trusted CA (*1) - verify-full: Connect using SSL, and use a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
Server	Specify the host name of the database server to connect to, using up to 18 bytes. This parameter cannot be omitted.
Port	Specify the port number to be used for remote access. The default value is "27500".
Username(*2)	Specify the user that will access the database.
Password(*2)	Specify the password for the user that will access the database.

*1: If specifying either "verify-ca" or "verify-full", use the system environment variable PGSSLROOTCERT of your operating system to specify the CA certificate file as shown below.

Example:

```
Variable name: PGSSLROOTCERT
Variable value: cACertificateFile
```

*2: In consideration of security, specify the Username and the Password by the application.

3.2.2.2 Registering using commands

This section describes how to use commands to register ODBC data sources.

Use the following tools from Microsoft to register ODBC data sources.

- ODBCConf.exe
- Add-OdbcDsn (can be used with Windows(R) 8 or later, or Windows Server(R) 2012 or later.)

Refer to the Microsoft Developer Network (MSDN) Library for information on how to use these tools.

When using ODBCConf.exe

ODBCConf.exe is a tool supported on all Windows platforms.

Specification format

```
ODBCConf.exe /A { dataSourceType "odbcDriverName" "optionName=value[|
optionName=value...]"} [/Lv fileName]
```

Refer to the Microsoft MSDN library for information on the format and parameters.

Description

Set the following content:

Definition name	Setting value
Data source type	Specify the data source type. - "CONFIGSYSDSN": A system data source is created. This requires user admin rights. The data source can be used by all users of the same computer. - "CONFIGDSN": A user data source is created. The data source can be used by the current user only. Note When CONFIGSYSDSN is specified as the data source type, it is necessary to execute the command in the command prompt in administrator mode.
ODBC driver name	Specify an ODBC driver name that has already been registered on the system. Specify one of the following. Note The placeholders shown below are enclosed in angle brackets '<>' to avoid confusion with literal text. Do not include the angle brackets in the string. - "Enterprise Postgres Unicode *<enterprisePostgresClientVersAndLvl>*" Specify this driver name if using Unicode as the application encoding system. - "Enterprise Postgres ANSI *<enterprisePostgresClientVersAndLvl>*" Specify this driver name if using other than Unicode as the application encoding system.
Option name	The following items must be set: - "DSN": Specify the data source name. - "Servername": Specify the host name for the database server. - "Port": Specify the port number for connection to the database - "Database": Specify the database name. Specify the following values as required: - "UID": User ID - "Password": Password - "SSLMode": Specify to encrypt communications. The default is "disable". Refer to the SSLMode explanation in the table under step 5 of "3.2.2.1 Registering using GUI" for information on how to configure SSLMode.

Definition name	Setting value
File Name	You can output process information to a file when creating a data source. This operand can be omitted.

Example

```
ODBCConf.exe /A {CONFIGSYSDSN "Enterprise Postgres Unicode 9.4" "DSN=odbcconf1|
Servername=sv1|Port=27500|Database=db01|SSLMode=verify-ca"} /Lv log.txt
```

Note

In consideration of security, specify the UID and the Password by the application.

When using Add-OdbcDsn

Add-OdbcDsn is used in the PowerShell command interface.

Specification format

```
Add-OdbcDsn dataSourceName -DriverName "odbcDriverName" -DsnType dataSourceType -
Platform oSArchitecture -SetPropertyValue @("optionName=value" [,"optionName=value"...])
```

Refer to the Microsoft MSDN library for information on the format and parameters.

Description

Set the following content:

Definition name	Setting value
Data source name	Specify any name for the data source name.
ODBC driver name	Specify an ODBC driver name that has already been registered on the system. Specify one of the following. **Note** The placeholders shown below are enclosed in angle brackets '<>' to avoid confusion with literal text. Do not include the angle brackets in the string. - "Enterprise Postgres Unicode *<enterprisePostgresClientVersAndLvl>*" Specify this driver name if using Unicode as the application encoding system. - "Enterprise Postgres ANSI *<enterprisePostgresClientVersAndLvl>*" Specify this driver name if using other than Unicode as the application encoding system.
Data source type	Specify the data source type. - "System": A system data source is created. Requires user admin rights. The data source can be used by all users of the same computer. - "User": A user data source is created. The data source can be used by the current user only.

Definition name	Setting value
	Note When System is specified as the data source type, it is necessary to execute the command in the administrator mode of the command prompt.
OS architecture	Specify the OS architecture of the system. - "32-bit": 32-bit system - "64-bit": 64-bit system
Option name	The following items must be set: - "Servername": Specify the host name for the database server. - "Port": Specify the port number for connection to the database - "Database": Specify the database name. Specify the following values as required: - "SSLMode": Specify to encrypt communications. The default is "disable". Refer to the SSLMode explanation in the table under step 5 of "3.2.2.1 Registering using GUI" for information on how to configure SSLMode. **Note** When using Add-OdbcDsn, the strings "UID" and "Password" cannot be set as option names. These can only be used when using ODBCConf.exe.

Example

```
Add-OdbcDsn odbcps1 -DriverName "Enterprise Postgres Unicode 9.4" -DsnType System -
Platform 32-bit -SetPropertyValue @("Servername=sv1", "Port=27500", "Database=db01",
"SSLMode=verify-ca")
```

L

3.2.3 Registering ODBC Data Sources(for Linux)

This section describes how to register ODBC data sources on Linux.

1. Registering data sources

   Edit the odbc.ini definition file for the data source.

   Information

   Edit the file in the installation directory for the ODBC driver manager (unixODBC)

   ```
   unixOdbcInstallDir/etc/odbc.ini
   ```

   Or

   Create a new file in the HOME directory

   ```
   ~/.odbc.ini
   ```

Point

If *unixOdbcInstallDir* is edited, these will be used as the shared settings for all users that log into the system. If created in the HOME directory (~/), the settings are used only by the single user.

Set the following content:

Definition name	Setting value
[Data source name]	Set the name for the ODBC data source.
Description	Set a description for the ODBC data source. Any description may be set.

Driver	Set the following as the name of the ODBC driver. Do not change this value.
	Select the two strings below that correspond to the application type. Concatenate the strings with no spaces and then specify this as the driver name.
	Note
	The placeholders shown below are enclosed in angle brackets '<>' to avoid confusion with literal text. Do not include the angle brackets in the string.
	- Application architecture
	  - For data sources used by 32-bit applications
	  "EnterprisePostgres<*enterprisePostgresClientVersAndLvl*>"
	  - For data sources used by 64-bit applications
	  "EnterprisePostgres<*enterprisePostgresClientVersAndLvl*>x64"
	- Encoding system used by the application
	  - In Unicode (only UTF-8 can be used)
	  "unicode"
	  - Other than Unicode
	  "ansi"
	Example: In a 32-bit application, where the encoding system used by the application is Unicode:
	"EnterprisePostgres<*enterprisePostgresClientVersAndLvl*>unicode"

Database	Specify the database name to be connected.
Servername	Specify the host name for the database server.
Username	Specify the user ID that will connect with the database.
Password	Specify the password for the user that will connect to the database.
Port	Specify the port number for the database server. The default is "27500".
SSLMode	The setting values for SSLMode are as follows: - disable: Connect without SSL - allow: Connect without SSL, and if it fails, connect using SSL - prefer: Connect using SSL, and if it fails, connect without SSL - require: Connect always using SSL

Definition name	Setting value
	- verify-ca: Connect using SSL, and use a certificate issued by a trusted CA (*1) - verify-full: Connect using SSL, and use a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
ReadOnly	Specify whether to set the database as read-only. - 1: Set read-only - 0: Do not set read-only

*1: If specifying either "verify-ca" or "verify-full", use the environment variable PGSSLROOTCERT to specify the CA certificate file as shown below.

Example

```
export PGSSLROOTCERT=cACertificateFileStorageDir/root.crt
```

Example:

Linux 32-bit

```
[MyDataSource]
Description    = EnterprisePostgres
Driver         = EnterprisePostgres9.4ansi
Database       = db01
Servername     = sv1
Port           = 27500
ReadOnly       = 0
```

Note

In consideration of security, specify the UserName and the Password by the application.

2. Environment variable settings

   To execute applications that use ODBC drivers, all of the following settings must be configured in the LD_LIBRARY_PATH environment variable:

   - *enterprisePostgresClientInstallDir*/lib
   - *unixOdbcInstallDir(*1)*/lib
   - *libtoolInstallDir(*1)*/lib

   *1: If the installation directory is not specified when unixODBC and libtool are installed, they will be installed in /usr/local.

3.2.4 Message Language and Encoding System Used by Applications Settings

This section explains the language settings for the application runtime environment and the encoding settings for the application.

Language settings

You must match the language settings for the application runtime environment with the message locale settings of the database server.

L

- Linux

  The language settings are made with the LANG environment variable or with the setlocale function in the source code.

- LANG environment variable settings

Apart from LANG, other environment variables for specifying language are LC_ALL and LC_MESSAGES. When multiple of these environment variables are set, the order of priority will be 1. LC_ALL, 2. LC_MESSAGES, and 3. LANG.

Example

Example of specifying "en_US.UTF-8" with the LANG environment variable (Bash)

```
> LANG=en_US.UTF-8; export LANG
```

- Settings for setlocale

Set the language of the application using the setlocale function in the source code.

Example

Example of specifying "en_US.UTF-8" with the setlocale function

```
setlocale(LC_ALL,"en_US.UTF-8");
```

Information

Refer to the documentation for the operating system for information on using the setlocale function.

W - Windows

Follows the locale of the OS.

Encoding System Settings

Ensure that the encoding system that is embedded in the application and passed to the database, and the encoding system setting of the runtime environment, are the same. The encoding system cannot be converted correctly on the database server.

Use one of the following methods to set the encoding system for the application:

- Set the PGCLIENTENCODING environment variable in the runtime environment.
- Set the client_encoding keyword in the connection string.
- Use the PQsetClientEncoding function.

See

Refer to "Supported Character Sets" in "Server Administration" in the PostgreSQL Documentation for information on the strings that represent the encoding system that can be set.

For example, when using "Unicode" and "8 bit", set the string "UTF8".

Example

Setting the "PGCLIENTENCODING" environment variable

L An example of setting when the encoding of the client is "UTF8" (Bash)

```
> PGCLIENTENCODING=UTF8; export PGCLIENTENCODING
```

An example of setting when the encoding of the client is "UTF8"

```
> set PGCLIENTENCODING=UTF8
```

Note

Text may be garbled when outputting results to the command prompt. Review the font settings for the command prompt if this occurs.

3.3 Connecting to the Database

Refer to the manual for the programming language corresponding to the ODBC interface, i.e. Access, Excel, or Visual Basic, for example.

3.4 Application Development

This section describes how to develop applications using ODBC drivers.

3.4.1 Compiling Applications (for Windows)

Refer to the manual for the programming language corresponding to the ODBC interface, i.e. Access, Excel, or Visual Basic, for example.

Note

The cl command expects input to be a program that uses one of the following code pages, so convert the program to these code pages and then compile and link it (refer to the Microsoft documentation for details).

- ANSI console code pages (example: Shift-JIS for Japanese)
- UTF-16 little-endian with or without BOM (Byte Order Mark)
- UTF-16 big-endian with or without BOM
- UTF-8 with BOM

The cl command converts strings in a program to an ANSI console code page before generating a module, so the data sent to and received from the database server becomes an ANSI console code page. Therefore, set the coding system corresponding to the ANSI console code page as the coding system of the client.

Refer to "Character Set Support" in "Server Administration" in the PostgreSQL Documentation for information on how to set the client encoding system.
(Example: To use environment variables in Japanese, set SJIS in PGCLIENTENCODING.)

3.4.2 Compiling Applications (for Linux)

Specify the following options when compiling applications.

Table 3.1 Include file and library path

Architecture	Option	How to specify the option
32-bit	Path of the include file	-I *unixOdbc32bitIncludeFileDir*
	Path of the library	-L *unixOdbc32bitLibraryDir*
64-bit	Path of the include file	-I *unixOdbc64bitIncludeFileDir*
	Path of the library	-L *unixOdbc64bitLibraryDir*

Table 3.2 ODBC library

Type of library	Library name
Dynamic library	libodbc.so

Note

Specify -m64 when creating a 64-bit application. Specify -m32 when creating a 32-bit application.

Example

The following are examples of compiling ODBC applications:

- Linux 64-bit

```
gcc -m64 -I/usr/local/include(*1) -L/usr/local/lib(*1) -lodbc testproc.c -o testproc
```

- Linux 32-bit

```
gcc -m32 -I/usr/local/include(*1) -L/usr/local/lib(*1) -lodbc testproc.c -o testproc
```

*1: This is an example of building and installing from the source without specifying an installation directory for unixODBC. If you wish to specify a location, set the installation directory.

3.4.3 Creating Applications during Cluster Operations

This section explains points to consider when creating applications during cluster operations.

See

Refer to the Cluster Operation Guide for information on cluster operations.

3.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs during cluster operations, explicitly close the connection and then reestablish the connection or reexecute the application.

The table below shows errors that may occur during a switch, and the corresponding action to take.

State		Error information (*2)	Action
Node failure (server failure) or Enterprise Postgres system failure (*1)	Failure occurs during access	57P01 08S01	After the switch is complete, reestablish the connection, or reexecute the application.

State		Error information (*2)	Action
	Accessed during node/ system failure	08001	
Switch to the standby node (standby server) (*1)	Switched during access	57P01 08S01	
	Accessed during switch	08001	

*1: Terms enclosed in () are terms used in database multiplexing mode.

*2: Return value of SQLSTATE.

3.4.3.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode

In the application, specify the same IP address as the one that was used when creating Gls or takeover network resources. This will enable the application to be executed if a state transition occurs, simply by reexecuting the application without modification.

See

Refer to "Creating Gls or Takeover Network Resources" in the Cluster Operation Guide for details.

Chapter 4 .NET Data Provider

This chapter describes how to configure for the purpose of creating .NET applications with Visual Studio.

4.1 Development Environment

.NET Data Provider can operate in the following environments:

.NET Framework environment for the development and running of applications	.NET Framework 4.5 .NET Framework 4 .NET Framework 3.5 SP1 or later	
Integrated development environment for applications running in a .NET Framework environment	Visual Studio 2013 Visual Studio 2012 Visual Studio 2010 Visual Studio 2008	
Combinations when TableAdapter is used	Visual Studio 2013	.NET Framework 4.5
	Visual Studio 2012	
	Visual Studio 2010	.NET Framework 4
	Visual Studio 2008	.NET Framework 3.5 SP1 or later
Available development languages	C# Visual Basic .NET	

4.2 Setup

This section explains how to setup .NET Data Provider.

4.2.1 Setting Up .NET Data Provider

You need to make the .NET Data Provider available for use to create applications with Visual Studio.

Add the reference to Fujitsu Npgsql .NET Data Provider for each Visual Studio project using the procedure below. The following describes the setup procedure in Visual Studio 2010:

1. In a Windows application, select [Add Reference] from the [Project] menu.
   In a Web application, select [Add Reference] from the [Web Site] menu.
2. Select "Fujitsu Npgsql .NET Data Provider" in the [Component Name] column of the [.NET] tab, and then click [OK].

There are two versions of "Fujitsu Npgsql .NET Data Provider". Decide the version of "Fujitsu Npgsql .NET Data Provider" to add according to the version of .NET Framework you will use. The notation "*x.y.z*" indicates the version and level of the Enterprise Postgres client feature.

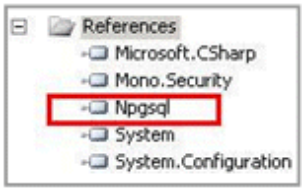
Version of .NET Data Provider	Description
2.*x.y.z*	Specify when the version of .NET Framework is 3.5.SP1 or later.
4.*x.y.z*	Specify when the version of .NET Framework is 4.0 or later.

Information

When .NET Data Provider setup is complete, the following names will be displayed in [References] in Visual Studio Solution Explorer.

- Npgsql

Example



4.2.2 Setting Up the Visual Studio Integration Add-On

TableAdapter is available for use when you use Visual Studio integration. Visual Studio integration is provided as a Visual Studio .NET add-on ("Npgsql Development Tools for .NET").

A user with administrator privileges can register Npgsql Development Tools for .NET as an add-on by executing pgx_ndtregister.exe as administrator.

Stored location of pgx_ndtregister.exe

The pgx_ndtregister.exe command is stored in the following location:

```
enterprisePostgresClientInstallDir\DOTNET\BIN\
```

Using pgx_ndtregister.exe

```
> pgx_ndtregister.exe [-x86| -x64]
```

Options

-x86

Use if registering a 32-bit add-on.

-x64

Use if registering a 64-bit add-on.

If no option is specified, a 32-bit add-on is registered.

Note

Execute this command after installing Visual Studio .NET.

4.2.3 Message Language and Encoding System Used by Applications Settings

If the .NET Data Provider is used, it will automatically set the encoding system on the client to UTF-8, so there is no need to configure this.

See

Refer to "Automatic Character Set Conversion Between Server and Client" in "Server Administration" in the PostgreSQL Documentation for information on encoding systems.

Language settings

You must match the language settings for the application runtime environment with the message locale settings of the database server.

Set language using the "System.Globalization.CultureInfo.CreateSpecificCulture" method.

Example

Code example for changing the locale in a C# application

```
System.Threading.Thread.CurrentThread.CurrentUICulture =
    System.Globalization.CultureInfo.CreateSpecificCulture("en");
```

4.3 Connecting to the Database

This section explains how to connect to a database.

- Using NpgsqlConnection
- Using NpgsqlConnectionStringBuilder
- Using the ProviderFactory Class

4.3.1 Using NpgsqlConnection

Connect to the database by specifying the connection string.

Example

Code examples for applications

```
using Npgsql;

NpgsqlConnection conn = new NpgsqlConnection("Server=sv1;Port=27500;Database=mydb; User
Id=myuser;Password=myuser01; Timeout=20;CommandTimeout=20;");
```

Refer to "4.3.4 Connection String" for information on database connection strings.

4.3.2 Using NpgsqlConnectionStringBuilder

Generate connection strings by specifying the connection information in the properties of the NpgsqlConnectionStringBuilder object.

Example

Code examples for applications

```
using Npgsql;

NpgsqlConnectionStringBuilder sb = new NpgsqlConnectionStringBuilder();
sb.Host = "sv1";
sb.Port = 27500;
sb.Database = "mydb";
sb.UserName = "myuser";
sb.Password = "myuser01";
sb.Timeout = 20;
```

```
sb.CommandTimeout = 20;
NpgsqlConnection conn = new NpgsqlConnection(sb.ConnectionString);
```

Refer to "4.3.4 Connection String" for information on database connection strings.

4.3.3 Using the ProviderFactory Class

Obtain the NpgsqlConnection object from the provider factory.

Example

Code examples for applications

```
using System.Data.Common;

DbProviderFactory factory = DbProviderFactories.GetFactory("Fujitsu.Npgsql");
DbConnection conn = factory.CreateConnection();
conn.ConnectionString = "Server=sv1;Port=27500;Database=mydb; User
Id=myuser;Password=myuser01; Timeout=20;CommandTimeout=20;";
```

Refer to "4.3.4 Connection String" for information on database connection strings.

4.3.4 Connection String

Specify the following connection information to connect to the database.

```
Server=127.0.0.1;Port=27500;Database=mydb;User Id=myuser;Password=myuser01;...;
      (1)           (2)           (3)          (4)            (5)      (6)
```

(1) Specify the host name or IP address of the server to be connected. This must be specified.

(2) Specify the port number for the database server. The default is "27500".

(3) Specify the database name to be connected.

(4) Specify the user ID that will connect with the database.

(5) Specify the password for the user that will connect to the database.

(6) Refer to the following for information on how to specify other connection information.

The following shows the keywords that are available to specify in the connection string in .NET Data Provider (Npgsql):

Note that some settings require care if using an Oracle database-compatible feature (refer to "9.2.2 Notes when Integrating with the Interface for Application Development" for details).

Keyword	Default value	Description
Server, Host (*1)	None	Specify the host name or IP address of the server to be connected. Specify up to 63 bytes when specifying a host name. A host name or IP address must be specified.
Port	27500	Specify the port number for the database server.
User Id, User, UID, UserName,	None	Specify the user ID that will connect with the database.

Keyword	Default value	Description
User Name, UserId (*1)		
Password, Pwd, Psw (*1)	None	Specify the password for the user ID that will connect to the database.
Database, DB (*1)	User name	Specify the database name to be connected.
SearchPath	(*2)	Specify the default schema name of the SQL statements used in the application. *2: If omitted, the value set in the search_path parameter in postgresql.conf on the server will be used.
Timeout	15	Specify the timeout for connections. Specify a value between 0 and 1024 (in seconds). There is no limit if you specify 0. The default is 15 seconds. An error occurs when a connection cannot be established within the specified time.
ConnectionLifeTime	15	For specifying the retention time of the connection, starting from the time of connection to the server and including the period the connection is to be retained in the connection pool. Specify in the range between 0 and 2147483647 (in seconds). If "0" is specified, there will be no limit on the retention time.
Pooling	True	Specify whether to use connection pooling or not. - Connection pooling is used if you specify True. Connection pooling is not used if you specify False.
MaxPoolSize	20	Maximum size of a connection pool. If the request exceeds this limit, it will wait until the another connection closes and the pool is available. Specify in the range between 0 and 1024. Connection pooling is not used if you specify 0.
MinPoolSize	1	Minimum size of a connection pool. When you specify MinPoolSize, NpgsqlConnection will pre-allocate connections with the specified number of servers. Specify in the range from 0 to the value specified at MaxPoolSize. When you specify 0, NpgsqlConnection will not pre-allocate connections with the specified number of servers.
SSL	False	Specify whether to encrypt the communication route between the client and server with the SSL protocol. - Encryption is performed when True is specified. - Encryption is not performed when False is specified.
Sslmode	Disable	Specify one of the following values for the SSL connection control mode: - Prefer: SSL is used for connection wherever possible. - Require: An exception is thrown when SSL connection is not possible. - Allow: Not yet supported. Connection is performed without SSL. - Disable: SSL connection is not performed.

Keyword	Default value	Description
Enlist	True	Specify whether to have connections participate in transactions with the transaction scope declared: - Connections will participate in transactions when True is specified. - Connections will not participate in transactions when False is specified.
CommandTimeout	20	Specify the timeout for communication with the server. Specify a value between 0 and 2147483647 (in seconds). There is no limit set if you set 0 or a negative value. The default is 20 seconds. An error occurs when data is not received from the server within the specified time.
Compatible	(*3)	Any effects from operational changes when performing future Npgsql version upgrades can be kept to a minimum by specifying the version of Npgsql that Fujitsu Npgsql .NET Data Provider is based on. Refer to the Installation and Setup Guide for Client for the version of Npgsql that Fujitsu Npgsql .NET Data Provider is based on. *3: If omitted, the version of Npgsql that Fujitsu Npgsql .NET Data Provider is based on will be set.
IntegratedSecurity	False	Set this when using Windows Integrated Security.
Protocol	3	Specify the protocol version to use. You must specify 3.
SyncNotification	False	Specify whether Npgsql will use synchronized notification.
Use Extended Types, UseExtendedTypes (*1)	False	This option determines how Nqgsql data and time types are used by DataAdapter. Data and time types include the "System.DateTime" type of .NET and "NpgsqlTimeStamp" ("NpgsqlTimeStamp" has functionality and range that exceeds "System.DateTime"). Both can be used, whatever value is set. If the value set is "True", however, it is possible to pass a specific Npgsql type for the relevant field of DataAdapter. If the value set is "False", on the other hand, DataAdapter requests "System.DateTime".
PreloadReader, Preload Reader (*1)	False	If True is specified, the entire result set will be loaded to DataReader before ExecuteReader() returns. Performance may decline when "True" is set (especially noticeable when the recordset is large). According to ADO.NET documentation, other operations are not possible while IDbConnection is being used, as it is used to get IDataReader. Npgsql enforces this rule. While NpgsqlDataReader is open, most operations except for the NpgsqlDataReader on the NpgsqlConnection that is trying to get the NpgsqlConnection will return an InvalidOperationException. (If NpgsqlDataReader is called at the end of the resultset(s), then Npgsql relaxes the rules and allows use of the connection even if it is not closed. This is because the connection is not being used by any resources at this point.)

*1: Only those keywords listed below can be used when using NpgsqlConnectionStringBuilder.

- Host

- UserName
- Password
- Database
- UseExtendedTypes
- PreloadReader

4.4 Application Development

This section explains the range of support provided with Visual Studio integration.

4.4.1 Data Types

A variety of data types can be used with Enterprise Postgres.

Data types belonging to base data types are supported whether you automatically generate applications using tools in Visual Studio (Query Builder in TableAdapter and Server Explorer), or create applications yourself (with DataProvider).

Table 4.1 List of supported data types

Data Types	Supported	
	Operation in the Visual Studio integration window	Fujitsu Npgsql .NET Data Provider
character	Y	Y
character varying	Y	Y
national character	Y	Y
national character varying	Y	Y
text	Y	Y
bytea	N	Y
smallint	Y	Y
integer	Y	Y
bigint	Conditional (*1)	Y
smallserial	Y	Y
serial	Y	Y
bigserial	Conditional (*1)	Y
real	Y	Y
double precision	Y	Y
numeric	Y	Y
decimal	Y	Y
money	N	N
date	Y	Y
time with time zone	Conditional (*2)	Conditional (*2,*3)
time without time zone	Conditional (*2)	Conditional (*2)
timestamp without time zone	Y	Y
timestamp with time zone	Y	Y

Data Types	Supported	
	Operation in the Visual Studio integration window	Fujitsu Npgsql .NET Data Provider
interval	Conditional (*4)	Y
boolean	Y	Y
bit	Conditional (*5)	Conditional (*5)
bit varying	N	N
uuid	Conditional (*4)	Y
inet	Conditional (*4, *6)	Conditional (*6)
macaddr	N	Y
cidr	N	N
Geometric data type (point,lseg,box,path,polygon,circle)	N	Y
array	N	Y
oid	N	N
xml	Y	Y
json	N	N
Types related to text searches(tsvector,tsquery)	N	N
Enumerated type	N	N
Composite type	N	N
Range type	N	N

Y: Supported

N: Not supported

*1: When used as a dynamic parameter, only data values in the int32 range can be substituted.

*2: As shown below, "time with time zone" and "time without time zone" values display the date portion as additional information. However, the actual data comprises the time data only, so with the exception of this displayed format, there are no other resulting issues.

Example:

Composition of table (t1)

col1 (time with time zone)	col2 (time without time zone)
10:21:30 +08:00	10:21:30
23:34:03 +08:00	23:34:03
17:23:54 +08:00	17:23:54

"time with time zone" values display the execution date in the date portion, while "time without time zone" values display "0001/01/01" in the date portion.

```
SELECT *
  FROM t1;
 col1                        |    col2
-----------------------------+---------------------
 2014/09/03 10:21:30 +08:00 | 0001/01/01 10:21:30
```

```
2014/09/03 23:34:03 +08:00 | 0001/01/01 23:34:03
2014/09/03 17:23:54 +08:00 | 0001/01/01 17:23:54
```

*3: When used as a dynamic parameter, it is not possible to substitute parameter values in DataGridView automatically generated by DDEX.

The example here shows c3 being made the "time with time zone" data type.

```
DataRow dr = ds.Tables[0].Rows[ds.Tables[0].Rows.Count - 1];
dr["c3"] = new DateTimeOffset(2000, 1, 1, 0, 0, 0, new TimeSpan(9, 0, 0));
```

*4: Only lengths of 2 or longer are supported.

*5: Only single hosts are supported.

*6: When updating this data type, set the values as shown below:

4.4.2 Relationship between Application Data Types and Database Data Types

The data types available for SQL data types are as follows:

List of data types belonging to base data types

Data types belonging to these base data types are supported whether you automatically generate applications using tools in Visual Studio (Query Builder in TableAdapter and Server Explorer), or create applications yourself (with Data Provider).

SQL data types	Npgsql	
	Data type	.NET CLS data type
character	NpgsqlDbType.Char	System.String
character varying	NpgsqlDbType.Varchar	System.String
national character	NpgsqlDbType.NChar	System.String
national character varying	NpgsqlDbType.NVarchar	System.String
text	NpgsqlDbType.Text	System.String
bytea	NpgsqlDbType.Bytea	System.Byte[]
smallint	NpgsqlDbType.Smallint	System.Int16
integer	NpgsqlDbType.Integer	System.Int32
bigint	NpgsqlDbType.Bigint	System.Int64
smallserial	NpgsqlDbType.Smallint	System.Int16
serial	NpgsqlDbType.Integer	System.Int32
bigserial	NpgsqlDbType.Bigint	System.Int64
real	NpgsqlDbType.Real	System.Single
double precision	NpgsqlDbType.Double	System.Double
numeric	NpgsqlDbType.Numeric	System.Decimal
decimal	NpgsqlDbType.Numeric	System.Decimal
date	NpgsqlDbType.Date	System.DateTime
		NpgsqlTypes.NpgsqlDate
time with time zone	NpgsqlDbType.TimeTZ	System.DateTimeOffset
		NpgsqlTypes.NpgsqlTimeTZ
time without time zone	NpgsqlDbType.Time	System.DateTime

SQL data types	Npgsql	
	Data type	.NET CLS data type
		NpgsqlTypes.NpgsqlTime
timestamp without time zone	NpgsqlDbType.Timestamp	System.DateTime
		NpgsqlTypes.NpgsqlTimeStamp
timestamp with time zone	NpgsqlDbType.TimestampTZ	System.DateTime
		NpgsqlTypes.NpgsqlTimeStampTZ
interval	NpgsqlDbType.Interval	System.String/System.TimeSpan(*1)
boolean	NpgsqlDbType.Boolean	System.Boolean
bit	NpgsqlDbType.Bit	System.Boolean
		NpgsqlTypes.BitString
uuid	NpgsqlDbType.Uuid	System.Guid
inet	NpgsqlDbType.Inet	NpgsqlTypes.NpgsqlInet
		System.Net.IPAddress
macaddr	NpgsqlDbType.MacAddr	NpgsqlTypes.NpgsqlMacAddress
		System.Net.NetworkInformation.PhysicalAddress
box	NpgsqlDbType.Box	NpgsqlTypes.NpgsqlBox
circle	NpgsqlDbType.Circle	NpgsqlTypes.NpgsqlCircle
lseg	NpgsqlDbType.LSeg	NpgsqlTypes.NpgsqlLSeg
path	NpgsqlDbType.Path	NpgsqlTypes.NpgsqlPath
point	NpgsqlDbType.Point	NpgsqlTypes.NpgsqlPoint
polygon	NpgsqlDbType.Polygon	NpgsqlTypes.NpgsqlPolygon
array	NpgsqlDbType.Array	System.Array

*1: The .NET CLS data types that can be used vary depending on the SQL interval type.
If using interval year/interval month/interval year to month, use System.String.
For other data types, use System.TimeSpan.

4.4.3 Creating Applications during Cluster Operations

This section explains points to consider when creating applications during cluster operations.

Refer to the Cluster Operation Guide for information on cluster operations.

4.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs during cluster operations, explicitly close the connection and then reestablish the connection or reexecute the application.

The table below shows errors that may occur during a switch, and the corresponding action to take.

State		Error information	Action
Node failure (server failure) or	Failure occurs during access	57P01 (*2) Empty string (*2)	After the switch is complete, reestablish the

State		Error information	Action
Enterprise Postgres system failure (*1)		NullReferenceException is generated	connection, or reexecute the application.
	Accessed during node/system failure	Empty string (*2)	
Switch to the standby node (standby server) (*1)	Switched during access	57P01 (*2) Empty string (*2) NullReferenceException is generated	
	Accessed during switch	Empty string (*2)	

*1: Terms enclosed in () are terms used in database multiplexing mode.

*2: This is the return value of the NpgsqlException attribute Code.

4.4.3.2 Notes when Creating Applications during Cluster Operations in Shared Disk Mode

In the application, specify the same IP address as the one that was used when creating Gls or takeover network resources. This will enable the application to be executed if a state transition occurs, simply by reexecuting the application without modification.

See

Refer to "Creating Gls or Takeover Network Resources" in the Cluster Operation Guide for details.

4.4.4 Notes

Notes on TableAdapter

- If [SELECT which returns a single value] is selected when adding a query to a TableAdapter, it will not be possible to execute the SQL statement displayed on the window - therefore, correct the SQL statement.
- If there is more than one Function with the same name, it will not be possible to create a TableAdapter using that Function.
- When defining Function, ensure that the OUT parameter is the last one defined.
  If the OUT parameter is not the last one defined, it will not be possible to enter any parameter values after the OUT parameter for the TableAdapter DataPreview.
  - Example incorrect definition: func(out p1 integer, inout p2 integer)
  - Example correct definition: func(inout p2 integer, out p1 integer)

Notes on the Query Builder

- Prefix named parameters with "@".
- Uppercase object names cannot be used, even when enclosed in double quotation marks.
  To use uppercase object names enclosed in double quotation marks, include them in SQL statements and enter these in the [Generate the SQL statements] window rather than in the Query Builder.
- SQL statements that specify date or time data literals cannot be used. To use these, enter the SQL statements directly in the [Generate the SQL statements] window rather than in the Query Builder.

- SQL statements cannot be correctly generated if the SQL statement specified in Filter matches any of the conditions below:
  - It uses PostgreSQL intrinsic operators such as << or ::.
  - It uses functions with keywords such as AS, FROM, IN, OVER.

    Example: extract(field from timestamp), RANK( ) OVER
  - It uses functions with the same names as those prescribed in SQL conventions, but that require different arguments.

Notes on Server Explorer

- The temporary table is not displayed.
- The database object function (FUNCTION) is displayed as a procedure.

Notes on metadata

- The CommandBehavior.KeyInfo argument must be specified if executing ExecuteReader before obtaining metadata using GetSchemaTable.

Example

```
NpgsqlDataReader  ndr=cmd.ExecuteReader(CommandBehavior.KeyInfo);
DataTable dt = dr.GetSchemaTable();
```

Notes on automatically generating update-type SQL statements

- If the SQL statement includes a query (which cannot be updated) that matches any of the conditions below, an update-type SQL statement will be generated (note that it may not be possible to execute this SQL statement in some cases):
  - It includes derived tables
  - It includes the same column name as the select list

  Update-type SQL statements will be automatically generated in the following cases:
  - If update statements are obtained using NpgsqlCommandBuilder
  - If data is updated using NpgsqlDataAdapter
  - If data is updated using TableAdapter

Notes on distributed transactions

- Applications using transaction scope can use distributed transactions by linking with Microsoft Distributed Transaction Coordinator (MSDTC). In this case, note the following:
  - Ensure that the value of max_prepared_transactions is greater than max_connection, so that "PREPARE TRANSACTION" can be issued for each transaction that simultaneously connects to the database server.
  - If each transaction in the transaction scope accesses the same resource using different connections, the database server will perceive it as requests from different applications, and a deadlock may occur. By configuring a timeout value for the transaction scope beforehand, the deadlock can be broken.

Chapter 5 C Library (libpq)

This chapter describes how to use C libraries.

5.1 Development Environment

Install the Enterprise Postgres Client package for the architecture to be developed and executed.

See

Refer to Installation and Setup Guide for Client for information on the C compiler required for C application development.

5.2 Setup

This section describes the environment settings required to use C libraries and how to encrypt data for communication.

5.2.1 Environment Settings

To execute an application that uses libpq, set the environment variable as shown below.

Linux

- Required for compile/link
  - LD_LIBRARY_PATH
    enterprisePostgresClientInstallDir/lib
- Required for execution of the application
  - PGLOCALEDIR
    enterprisePostgresClientInstallDir/share/locale

Example

When the 32-bit version client package is installed.

```
> LD_LIBRARY_PATH=/opt/fsepclient32/lib:$LD_LIBRARY_PATH;export LD_LIBRARY_PATH
> PGLOCALEDIR=/opt/fsepclient32/share/locale;export PGLOCALEDIR
```

Windows (R)

- Required for compile/link
  - LIB
    enterprisePostgresClientInstallDir\lib
  - INCLUDE
    enterprisePostgresClientInstallDir\include
- Required for execution of the application
  - PATH
    enterprisePostgresClientInstallDir\lib
  - PGLOCALEDIR
    enterprisePostgresClientInstallDir\share\locale

Example

When the 32-bit version client package is installed on a 64-bit operating system.

```
> SET PATH=%ProgramFiles(x86)%\Fujitsu\fsepclient32\lib;%PATH%
> SET LIB=%ProgramFiles(x86)%\Fujitsu\fsepclient32\lib;%LIB%
> SET INCLUDE=%ProgramFiles(x86)%\Fujitsu\fsepclient32\include;%INCLUDE%
> SET PGLOCALEDIR=%ProgramFiles(x86)%\Fujitsu\fsepclient32\share\locale
```

5.2.2 Message Language and Encoding System Used by Applications Settings

This section explains the language settings for the application runtime environment and the encoding settings for the application.

Language settings

You must match the language settings for the application runtime environment with the message locale settings of the database server.

L

- Linux

The language settings are made with the LANG environment variable or with the setlocale function in the source code.

- LANG environment variable settings

Apart from LANG, other environment variables for specifying language are LC_ALL and LC_MESSAGES. When multiple of these environment variables are set, the order of priority will be 1. LC_ALL, 2. LC_MESSAGES, and 3. LANG.

Example

Example of specifying "en_US.UTF-8" with the LANG environment variable (Bash)

```
> LANG=en_US.UTF-8; export LANG
```

- Settings for setlocale

Set the language of the application using the setlocale function in the source code.

Example

Example of specifying "en_US.UTF-8" with the setlocale function

```
setlocale(LC_ALL,"en_US.UTF-8");
```

Information

Refer to the documentation for the operating system for information on using the setlocale function.

W

- Windows(R)

Follows the locale of the OS.

Encoding System Settings

Ensure that the encoding system that is embedded in the application and passed to the database, and the encoding system setting of the runtime environment, are the same. The encoding system cannot be converted correctly on the database server.

Use one of the following methods to set the encoding system for the application:

- Set the PGCLIENTENCODING environment variable in the runtime environment.
- Set the client_encoding keyword in the connection string.
- Use the PQsetClientEncoding function.

See

Refer to "Supported Character Sets" in "Server Administration" in the PostgreSQL Documentation for information on the strings that represent the encoding system that can be set.

For example, when using "Unicode" and "8 bit", set the string "UTF8".

Note

Text may be garbled when outputting results to the command prompt. Review the font settings for the command prompt if this occurs.

5.2.3 Settings for Encrypting Communication Data

Set in one of the following ways when performing remote access using communication data encryption:

When setting from outside with environment variables

Specify "require", "verify-ca", or "verify-full" in the PGSSLMODE environment variable.

In addition, the parameters for the PGSSLROOTCERT and PGSSLCRL environment variables need to be set to prevent spoofing of the database server.

See

Refer to "Environment Variables" in "Client Interfaces" in the PostgreSQL Documentation for information on environment variables.

When specifying in the connection URI

Specify "require", "verify-ca", or "verify-full" in the "sslmode" parameter of the connection URI.

In addition, the parameters for the sslcert, sslkey, sslrootcert, and sslcrl need to be set to prevent spoofing of the database server.

See

Refer to "Secure TCP/IP Connections with SSL" in "Server Administration" in the PostgreSQL Documentation for information on encrypting communication data.

5.3 Connecting with the Database

Point

Use the connection service file to specify the connection destination. In the connection service file, a name (service name) is defined as a set, comprising information such as connection destination information and various types of tuning information set for connections. By using the service name defined in the connection service file when connecting to databases, it is no longer necessary to modify applications when the connection information changes.

Refer to "Client Interfaces", "The Connection Service File" in the PostgreSQL Documentation for details.

See

Refer to "Database Connection Control Functions" in "Client Interfaces" in the PostgreSQL Documentation.

In addition, refer to "6.3 Connecting with the Database" in "Embedded SQL in C " for information on connection string.

5.4 Application Development

See

Refer to "libpq - C Library" in "Client Interfaces" in the PostgreSQL Documentation for information on developing applications.

However, if you are using the C library, there are the following differences to the PostgreSQL C library (libpq).

5.4.1 Compiling Applications

Specify the following options when compiling applications:

L - Linux

L Table 5.1 Include file and library path

Option	How to specify the option
Path of the include file	-I/*enterprisePostgresClientInstallDir*/include
Path of the library	-L/*enterprisePostgresClientInstallDir*/lib

L Table 5.2 C Library (libpq library)

Type of library	Library name
Dynamic library	libpq.so
Static library	libpq.a

W - Windows(R)

If the include file and the library path have been set in the environment variable, there is no need to specify the options shown below for the compile.

W Table 5.3 Include file and library path

Option	How to specify the option
Path of the include file	*enterprisePostgresClientInstallDir*\include
Path of the library	*enterprisePostgresClientInstallDir*\lib

Table 5.4 C Library (libpq library)

Type of library	Library name
Library for links	libpq.lib
Dynamic library	libpq.dll

5.4.2 Creating Applications during Cluster Operations

This section explains points to consider when creating applications during cluster operations.

See

Refer to the Cluster Operation Guide for information on cluster operations.

5.4.2.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs during cluster operations, explicitly close the connection and then reestablish the connection or reexecute the application.

The table below shows errors that may occur during a switch, and the corresponding action to take.

State		Error information (*2)	Action
Node failure (server failure) or Enterprise Postgres system failure (*1)	Failure occurs during access	PGRES_FATAL_ERROR(*2) 57P01(*3) NULL(*3)	After the switch is complete, reestablish the connection, or reexecute the application.
	Accessed during node/ system failure	CONNECTION_BAD(*4)	
Switch to the standby node (standby server) (*1)	Switched during access	PGRES_FATAL_ERROR(*2) 57P01(*3) NULL(*3)	
	Accessed during switch	CONNECTION_BAD(*4)	

*1: Terms enclosed in () are terms used in database multiplexing mode.

*2: Return value of PQresultStatus().

*3: Return value of PQresultErrorField() PG_DIAG_SQLSTATE.

*4: Return value of PQstatus().

5.4.2.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode

In the application, specify the same IP address as the one that was used when creating Gls or takeover network resources. This will enable the application to be executed if a state transition occurs, simply by reexecuting the application without modification.

See

Refer to "Creating Gls or Takeover Network Resources" in the Cluster Operation Guide for details.

Chapter 6 Embedded SQL in C

This chapter describes application development using embedded SQL in C.

6.1 Development Environment

Install the Enterprise Postgres Client package for the architecture to be developed and executed.

See

Refer to Installation and Setup Guide for Client for information on the C compiler required for C application development.

Note

C++ is not supported. Create a library by implementing embedded SQL in C, and call it from C++.

6.2 Setup

6.2.1 Environment Settings

When using embedded SQL in C, the same environment settings as when using the C library (libpq) are required.

Refer to "5.2.1 Environment Settings" in "C Library (libpq)" for information on the environment settings for the library for C.

Additionally, set the following path for the precompiler ecpg:

L

Linux

```
enterprisePostgresClientInstallDir/bin
```

W

Windows(R)

```
enterprisePostgresClientInstallDir\bin
```

6.2.2 Message Language and Encoding System Used by Applications Settings

The message language and the encoding System Settings Used by Applications settings are the same as when using the library for C.

However, in embedded SQL, the PQsetClientEncoding function cannot be used in the encoding system settings. In embedded SQL, use the SET command to specify the encoding system in client_encoding.

Refer to "5.2.2 Message Language and Encoding System Used by Applications Settings" in "C Library (libpq)" for information on the settings for the library for C.

6.2.3 Settings for Encrypting Communication Data

When encrypting the communication data, the same environment settings as when using the C library (libpq) are required.

Refer to "5.2.3 Settings for Encrypting Communication Data" in "C Library (libpq)" for information on the environment settings for the C library.

6.3 Connecting with the Database

Point

- It is recommended to use a connection service file to specify connection destinations. In the connection service file, a name (service name) is defined as a set, comprising information such as connection destination information and various types of tuning information set for connections. By using the service name defined in the connection service file when connecting to databases, it is no longer necessary to modify applications when the connection information changes. Refer to "The Connection Service File" in "Client Interfaces" in the PostgreSQL Documentation for information.
- If using a connection service file, perform either of the procedures below:
  - Set the service name as a string literal or host variable, as follows:

    tcp:postgresql://?service=my_service
  - Set the service name in the environment variable PGSERVICE, and use CONNECT TO DEFAULT

Use the CONNECT statement shown below to create a connection to the database server.

Format

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

target

Write in one of the following formats:

- dbname@host:port
- tcp:postgresql://host:port/dbname[?options]
- L unix:postgresql://host[:port][/dbname][?options]
  (Definition method when using the UNIX domain socket)
- SQL string literal containing one of the above formats
- Reference to a character variable containing one of the above formats
- DEFAULT

user-name

Write in one of the following formats:

- username
- username/password
- username IDENTIFIED BY password
- username USING password

Description of the arguments

Argument	Description
dbname	Specify the database name.
host	Specify the host name for the connection destination.
port	Specify the port number for the database server. The default is "27500".
connection-name	Specify connection names to identify connections when multiple connections are to be processed within a single program.

Argument	Description
username	Specify the user that will connect with the database. If this is omitted, the name used will be that of the user on the operating system that is executing the application.
password	Specify a password when authentication is required.
options	Specify the following parameter when specifying a time for timeout. Connect parameters with & when specifying more than one. The following shows the values specified for each parameter. - connect_timeout Specify the timeout for connections. Specify a value between 0 and 2147483647 (in seconds). There is no limit set if you set 0 or an invalid value. If "1" is specified, the behavior will be the same as when "2" was specified. An error occurs when a connection cannot be established within the specified time. - keepalives This enables keepalive. Keepalive is disabled if 0 is specified. Keepalive is enabling when any other value is specified. The default is keepalive enabled. Keepalive causes an error to occur when it is determined that the connection with the database is disabled. - keepalives_idle Specify the time until the system starts sending keepalive messages when communication with the database is not being performed. - Linux Specify a value between 1 and 32767 (in seconds). The default value of the system is used if this is not specified. - Windows Specify a value between 1 and 2147483647 (in seconds). 7200 will be set as default if a value outside this range is specified or if nothing is specified. - keepalives_interval Specify the interval between resends when there is no response to keepalive messages. - Linux Specify a value between 1 and 32767 (in seconds). The default value of the system is used if this is not specified. - Windows Specify a value between 1 and 2147483647 (in seconds). 1 will be set as default if a value outside this range is specified or if nothing is specified. - keepalives_count Specify the number of resends for keepalive messages. - Linux Specify a value between 1 and 127. The default value of the system is used if this is not specified. - Windows The system default value is used irrespective of what is specified for this parameter.

Code examples for applications

```
EXEC SQL CONNECT TO tcp:postgresql://sv1:27500/mydb?
connect_timeout=20&keepalives_idle=20&keepalives_interval=5&keepalives_count=2&keepalives=
1 USER myuser/myuser01;
```

6.4 Application Development

Refer to "ECPG - Embedded SQL in C" in "Client Interfaces" in the PostgreSQL Documentation for information on developing applications.

However, when using embedded SQL in C, there are the following differences to the embedded SQL (ECPG) in PostgreSQL C.

6.4.1 Support for National Character Data Types

This section describes how to use the national character data types using the SQL embedded C preprocessor.

The following explains the C language variable types corresponding to the NCHAR type:

Specify the number of characters specified for the NCHAR type multiple by 4, plus 1 for the length of the host variable.

Data Type	Host variable type
NATIONAL CHARACTER(n)	NCHAR variable name [nx4+1]
NATIONAL CHARACTER VARYING(n)	NVARCHAR variable name [nx4+1]

See

Refer to "Handling Character Strings" in "Client Interfaces" in the PostgreSQL documentation for information on using character string types.

6.4.2 Compiling Applications

Append the extension "pgc" to the name of the source file for the embedded SQL in C.

When the pgc file is precompiled using the ecpg command, C source files will be created, so use the C compiler for the compile.

Precompiling example

```
ecpg testproc.pgc
```

If an optimizer hint block comment is specified for the SQL statement, specify the following option in the ecpg command:

--enable-hint

Enables the optimizer hint block comment (hereafter, referred to as the "hint clause"). If this option is not specified, the hint clause will be removed as a result of the ecpg precompile and be disabled.

The SQL statements that can be specified in the hint clause are SELECT, INSERT, UPDATE, and DELETE.

The locations in which the hint clause can be specified are immediately after one of the SELECT, INSERT, UPDATE, DELETE, or WITH keywords. A syntax error will occur if any other location is specified.

Example of specifying the hint clause

```
EXEC SQL SELECT /*+ IndexScan(prod ix01) */ name_id INTO :name_id FROM prod WHERE id
= 1;
```

Refer to "12.1.1 Optimizer Hints" for information on optimizer hints.

Note

Take the following points into account when using embedded SQL source files:

- Multibyte codes expressed in SJIS or UTF-16 cannot be included in statements or host variable declarations specified in EXEC SQL.
- Do not use UTF-8 with a byte order mark (BOM), because an error may occur during compilation if the BOM character is incorrectly recognized as the source code.
- Multibyte characters cannot be used in host variable names.
- It is not possible to use a TYPE name that contains multibyte characters, even though it can be defined.

Specify the following options when compiling a C application output with precompiling.

L

Linux

L

Table 6.1 Include file and library path

Option	How to specify the option
Path of the include file	-I/*enterprisePostgresClientInstallDir*/include
Path of the library	-L/*enterprisePostgresClientInstallDir*/lib

L

Table 6.2 C Library

Type of library	Library name	Note
Dynamic library	libecpg.so	
	libpgtypes.so	When using the pgtypes library
Static library	libecpg.a	
	libpgtypes.a	When using the pgtypes library

W

Windows(R)

If the include file and the library path have been set in the environment variable, there is no need to specify the options shown below for the compile.

W

Table 6.3 Include file and library path

Type of option	How to specify the option
Path of the include file	*enterprisePostgresClientInstallDir\include*
Path of the library	*enterprisePostgresClientInstallDir\lib*

W

Table 6.4 C Library

Type of library	Library name	Note
Library for links	libecpg.lib	
	libpgtypes.lib	When using the pgtypes library
Dynamic library	libecpg.dll	
	libpgtypes.dll	When using the pgtypes library

Note

- The libecpg library in Windows is created by "release" and "multithreaded" options. When using the ECPGdebug function included in this library, compile using the "release" and "multithreaded" flags in all programs that use this library. When you do this, use the "dynamic" flag if you are using libecpg.dll, and use the "static" flag if you are using libecpg.lib.

  Refer to "Library Functions" in "Client Interfaces" in the PostgreSQL Documentation for information on the ECPGdebug function.

- The cl command expects input to be a program that uses one of the following code pages, so convert the program to these code pages and then compile and link it (refer to the Microsoft documentation for details).

  - ANSI console code pages (example: Shift-JIS for Japanese)
  - UTF-16 little-endian with or without BOM (Byte Order Mark)
  - UTF-16 big-endian with or without BOM
  - UTF-8 with BOM

  The cl command converts strings in a program to an ANSI console code page before generating a module, so the data sent to and received from the database server becomes an ANSI console code page. Therefore, set the coding system corresponding to the ANSI console code page as the coding system of the client.

  Refer to "Character Set Support" in "Server Administration" in the PostgreSQL Documentation for information on how to set the client encoding system.
  (Example: To use environment variables in Japanese, set SJIS in PGCLIENTENCODING.)

6.4.3 Bulk INSERT

Bulk INSERT can be used to input multiple rows of data into the table using a single ECPG statement that uses the newly introduced 'FOR' clause.

This functionality allows the user to make use of the data stored in host array variables, resulting in 'C' client programs that are simpler and easier to maintain.

Synopsis

The syntax of the bulk INSERT statement is given below:

```
EXEC SQL [ AT connection ] [ FOR {number_of_rows|ARRAY_SIZE}]
    INSERT INTO table_name [ ( column_name [, ...] ) ]
    {  VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...]
      INTO output_host_var  [ [ INDICATOR ] indicator_var  ]  [, ...]]
```

When the above bulk INSERT command is used, ECPG inserts '*number_of_rows*' number of rows into the table, using the data that is stored in the '*expression*'.

FOR Clause

The 'FOR' clause indicates to ECPG that the given INSERT statement is a bulk insert statement. The 'FOR' clause currently only support INSERT statements. When a 'FOR' clause specified, ECPG executes the INSERT statement for 'number_of_rows' number of times, iterating through the host array variables.

The 'FOR' clause can iterate through all the array elements or can be limited to process only a fixed number of array elements. The value for 'number_of_rows' should be greater than zero.

The 'FOR' clause can accept an integer host variable or an integer literal as its parameter. It can also accept the constant '*ARRAY_SIZE*' in which case all the elements in the host array variable are inserted into the table.

Given below are examples of acceptable bulk INSERT statements.

```
int  number_of_rows = 10;
int id[25];
char name[25][10];

EXEC SQL FOR :number_of_rows    /* will process 10 rows */
INSERT INTO prod (name, id) VALUES (:name, :id);

EXEC SQL FOR ARRAY_SIZE          /* will process 25 rows */
INSERT INTO prod (name, id) VALUES (:name, :id);
```

When 'FOR' clause is specified, the '*expression*' can be a host array variable, host variable, constant number or string.

The value given for the 'number_of_rows' should be greater than zero. Value which is less than or equal to zero will result in a run-time error.

When 'FOR ARRAY_SIZE' is specified, the values clause should consist of at least one host array variable. 'SELECT' queries cannot be used for input values when using 'FOR ARRAY_SIZE'.

When the value given for the 'number_of_rows' is greater than one, the specified 'SELECT' query should returns only one row. More than one returned row will result in an error.

Values Clause

The VALUES clause includes the input data that is to be inserted into the table. When working with FOR clause, the values in the '*expression*' can be host array variables, host variable, constant number, string or pointers.

For more detailed usage of the INSERT statement, please refer to the ECOBPG section of the PostgreSQL documentation.

Error Messages

Given below are the error messages that are output when bulk INSERT functionality is not used correctly.

Invalid value for number_of_rows

ECPG error

invalid statement name "FOR value should be positive integer"

Cause

The value given for number_of_rows is less than or equal to 0.

Solution

Specify a value that is more than or equal to 1 for number_of_rows.

Invalid input for ARRAY_SIZE

ECPG error

invalid statement name "Host array variable is needed when using FOR ARRAY_SIZE"

Cause

A host array is not specified in the values clause when using the ARRAY_SIZE keyword.

Solution

At least one host array variable should be included in the values clause

Too many rows from SELECT... INTO

ECPG error

SELECT...INTO returns too many rows

Cause

The number of rows returned by the 'SELECT ... INTO' query in the INSERT statement is more than one.

Solution

When the value of 'number_of_rows' is more than one, the maximum number of rows that can be returned by the 'SELECT ... INTO' query in the INSERT statement is one.

Limitations

The limitations when using bulk INSERT are given below.

- Array of structures should not be used as an input in the 'VALUES' clause. Attempted use will result in junk data being inserted into the table.
- Array of pointers should not be used as an input in the 'VALUES' clause. Attempted use will result in junk data being inserted into the table.
- ECPG supports the use of 'WITH' clause in single INSERT statements. 'WITH' clause cannot be used in bulk INSERT statements.
- ECPG does not calculate the size of the pointer variable. So when a pointer variable is used that includes multiple elements, the 'number_of_rows' should be less than or equal to the number of elements in the pointer. Otherwise, junk data will be inserted into the table.

Samples

Given below are some sample usages of the bulk INSERT functionality.

Basic Bulk INSERT

```
int in_f1[4] = {1,2,3,4};
...
EXEC SQL FOR 3 INSERT INTO target (f1) VALUES (:in_f1);
```

The number of rows to insert indicated by the FOR clause is 3, so the data in the first 3 elements of the host array variable are inserted into the table. The contents of the target table will be:

```
 f1
----
  1
  2
  3
(3 rows)
```

Also a host integer variable can be used to indicate the number of rows that will be inserted in FOR clause, which will produce the same result as above:

```
int num = 3;
int in_f1[4] = {1,2,3,4};
...
EXEC SQL FOR :num INSERT INTO target (f1) VALUES (:in_f1);
```

Inserting constant values

Constant values can also be bulk INSERTed into the table as follows:

```
EXEC SQL FOR 3 INSERT INTO target (f1,f2) VALUES (DEFAULT,'hello');
```

Assuming the 'DEFAULT' value for the 'f1' column is '0', the contents of the target table will be:

```
f1 |  f2
---+-------
 0 | hello
 0 | hello
 0 | hello
(3 rows)
```

Using ARRAY_SIZE

'FOR ARRAY_SIZE' can be used to insert the entire contents of a host array variable, without explicitly specifying the size, into the table.

```
    int in_f1[4] = {1,2,3,4};
...
EXEC SQL FOR ARRAY_SIZE INSERT INTO target (f1) VALUES (:in_f1);
```

In the above example, four rows are inserted into the table.

Note

If there are multiple host array variables specified as input values, then the number of rows inserted is same as the smallest array size. The example given below demonstrates this usage.

```
int in_f1[4] = {1,2,3,4};
char in_f3[3][10] = {"one", "two", "three"};
...
EXEC SQL FOR ARRAY_SIZE INSERT INTO target (f1,f3) VALUES (:in_f1,:in_f3);
```

In the above example, the array sizes are 3 and 4. Given that the smallest array size is 3, only three rows are inserted into the table. The table contents are given below.

```
 f1 |  f3
----+-------
  1 | one
  2 | two
  3 | three
(3 rows)
```

Using Pointers as Input

Pointers that contain multiple elements can be used in bulk INSERT.

```
int *in_pf1 = NULL;
in_pf1 = (int*)malloc(4*sizeof(int));
in_pf1[0]=1;
in_pf1[1]=2;
in_pf1[2]=3;
in_pf1[3]=4;
...
EXEC SQL FOR 4 INSERT INTO target (f1) values (:in_pf1);
```

The above example will insert four rows into the target table.

Using SELECT query

When using bulk INSERT, the input values can be got from the results of a SELECT statement. For ex.,

```
EXEC SQL FOR 4 INSERT INTO target(f1) SELECT age FROM source WHERE name LIKE 'foo';
```

Assuming that the 'SELECT' query returns one row, the same row will be inserted into the target table four times.

If the 'SELECT' query returns more than one row, the INSERT statement will throw an error.

```
EXEC SQL FOR 1 INSERT INTO target(f1) SELECT age FROM source;
```

In the above example, all the rows returned by the 'SELECT' statement will be inserted into the table. In this context '1' has the meaning of 'returned row equivalent'.

Using RETURNING clause

Bulk INSERT supports the same RETURNING clause syntax as normal INSERT. An example is given below.

```
int out_f1[4];
int in_f1[4] = {1,2,3,4};
...
EXEC SQL FOR 3 INSERT INTO target (f1) VALUES (:in_f1) RETURNING f1 INTO :out_f1;
```

After the execution of the above INSERT statement, the 'out_f1' array will have 3 elements with the values of '1','2' and '3'.

6.4.4 DECLARE STATEMENT

This section describes the DECLARE STATEMENT statement.

Synopsis

```
EXEC SQL [ AT connection_name] DECLARE statement_name STATEMENT
```

Description

DECLARE STATEMENT declares SQL statement identifier. SQL statement identifier is associated with connection. DELARE CURSOR with a SQL statement identifier can be written before PREPARE.

Parameters

connection_name

A database connection name established by the CONNECT command.

If AT clause is omitted, a SQL statement identifier is associated with the DEFAULT connection.

statement_name

An identifier for SQL statement identifier which is SQL identifier or host variable.

Examples

```
EXEC SQL CONNECT TO postgres AS con1
EXEC SQL AT con1 DECLARE sql_stmt STATEMENT
EXEC SQL DECLARE cursor_name CURSOR FOR sql_stmt
EXEC SQL PREPARE sql_stmt FROM :dyn_string
EXEC SQL OPEN cursor_name
```

```
EXEC SQL FETCH cursor_name INTO :column1
EXEC SQL CLOSE cursor_name
```

Note

- An SQL statement with a SQL statement identifier must use a same connection as the connection that the SQL statement identifier is associated with.
- An SQL statement without a SQL statement identifier must not use AT clause.

6.4.5 Creating Applications during Cluster Operations

This section explains points to consider when creating applications during cluster operations.

See

Refer to the Cluster Operation Guide for information on cluster operations.

6.4.5.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs during cluster operations, explicitly close the connection and then reestablish the connection or reexecute the application.

The table below shows errors that may occur during a switch, and the corresponding action to take.

State		Error information (*2)	Action
Node failure (server failure) or Enterprise Postgres system failure (*1)	Failure occurs during access	57P01 57P02 YE000 26000 40001	After the switch is complete, reestablish the connection, or reexecute the application.
	Accessed during node/system failure	08001	
Switch to the standby node (standby server) (*1)	Switched during access	57P01 57P02 YE000 26000 40001	
	Accessed during switch	08001	

*1: Terms enclosed in () are terms used in database multiplexing mode.

*2: Return value of SQLSTATE.

6.4.5.2 Notes when Creating Applications during Failover Operations in Shared Disk Mode

In the application, specify the same IP address as the one that was used when creating Gls or takeover network resources. This will enable the application to be executed if a state transition occurs, simply by reexecuting the application without modification.

See

Refer to "Creating Gls or Takeover Network Resources" in the Cluster Operation Guide for details.

6.4.6 Notes

Notes on creating multithreaded applications

In embedded SQL in C, DISCONNECT ALL disconnects all connections within a process, and therefore it is not thread-safe in all operations that use connections. Do not use it in multithreaded applications.

Chapter 7 Embedded SQL in COBOL

This chapter describes application development using embedded SQL in COBOL.

7.1 Development Environment

Install the Enterprise Postgres Client package for the architecture to be developed and executed.

See

Refer to Installation and Setup Guide for Client for information on the COBOL compiler required for COBOL application development.

7.2 Setup

7.2.1 Environment Settings

When using embedded SQL in COBOL, the same environment settings as when using the C library (libpq) are required.

Refer to "5.2.1 Environment Settings" in "C Library (libpq)" for information on the environment settings for the library for C.

Additionally, set the following path for the precompiler ecobpg:

L Linux

```
enterprisePostgresClientInstallDir/bin
```

W Windows(R)

```
enterprisePostgresClientInstallDir\bin
```

7.2.2 Message Language and Encoding System Used by Applications Settings

The message language and the encoding System Settings Used by Applications settings are the same as when using the library for C.

However, in embedded SQL, the PQsetClientEncoding function cannot be used in the encoding system settings. In embedded SQL, use the SET command to specify the encoding system in client_encoding.

Refer to "5.2.2 Message Language and Encoding System Used by Applications Settings" in "C Library (libpq)" for information on the settings for the library for C.

7.2.3 Settings for Encrypting Communication Data

When encrypting the communication data, the same environment settings as when using the C library (libpq) are required.

Refer to "5.2.3 Settings for Encrypting Communication Data" in "C Library (libpq)" for information on the environment settings for the C library.

7.3 Connecting with the Database

Use the CONNECT statement shown below to create a connection to the database server.

Format

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name]END-EXEC.
```

target

Write in one of the following formats:

- dbname@host:port
- tcp:postgresql://host:port/dbname[?options]
- unix:postgresql://host[:port][/dbname][?options]
  (Definition method when using the UNIX domain socket)
- SQL string literal containing one of the above formats
- Reference to a character variable containing one of the above formats
- DEFAULT

user-name

Write in one of the following formats:

- username
- username/password
- username IDENTIFIED BY password
- username USING password

Description of the arguments

Argument	Description
dbname	Specify the database name.
host	Specify the host name for the connection destination.
port	Specify the port number for the database server. The default is "27500".
connection-name	Specify connection names to identify connections when multiple connections are to be processed within a single program.
username	Specify the user that will connect with the database. If this is omitted, the name used will be that of the user on the operating system that is executing the application.
password	Specify a password when authentication is required.
options	Specify the following parameter when specifying a time for timeout. Connect parameters with & when specifying more than one. The following shows the values specified for each parameter. - connect_timeout Specify the timeout for connections. Specify a value between 0 and 2147483647 (in seconds). There is no limit set if you set 0 or an invalid value. If "1" is specified, the behavior will be the same as when "2" was specified. An error occurs when a connection cannot be established within the specified time. - keepalives This enables keepalive.

Argument	Description
	Keepalive is disabled if 0 is specified. Keepalive is enabling when any other value is specified. The default is keepalive enabled. Keepalive causes an error to occur when it is determined that the connection with the database is disabled. - keepalives_idle Specify the time until the system starts sending keepalive messages when communication with the database is not being performed. - Linux Specify a value between 1 and 32767 (in seconds). The default value of the system is used if this is not specified. - Windows Specify a value between 1 and 2147483647 (in seconds). 7200 will be set as default if a value outside this range is specified or if nothing is specified. - keepalives_interval Specify the interval between resends when there is no response to keepalive messages. - Linux Specify a value between 1 and 32767 (in seconds). The default value of the system is used if this is not specified. - Windows Specify a value between 1 and 2147483647 (in seconds). 1 will be set as default if a value outside this range is specified or if nothing is specified. - keepalives_count Specify the number of resends for keepalive messages. - Linux Specify a value between 1 and 127. The default value of the system is used if this is not specified. - Windows The system default value is used irrespective of what is specified for this parameter.

Code examples for applications

```
EXEC SQL CONNECT TO tcp:postgresql://sv1:27500/mydb?
connect_timeout=20&keepalives_idle=20&keepalives_interval=5&keepalives_count=2&keepalives=
1 USER myuser/myuser01 END-EXEC.
```

7.4 Application Development

Refer to "Appendix D ECOBPG - Embedded SQL in COBOL" for information on developing applications.

7.4.1 Compiling Applications

Append the extension "pco" to the name of the source file for the embedded SQL in C.

When the pco file is precompiled using the ecobpg command, COBOL source files will be created, so use the COBOL compiler for the compile.

Precompiling example

```
ecobpg testproc.pco
```

Specify the following options when compiling a COBOL application output with precompiling.

L Linux

L Table 7.1 Include file and library path

Option	How to specify the option
Path of the include file	-I/*enterprisePostgresClientInstallDir*/include
Path of the library	-L/*enterprisePostgresClientInstallDir*/lib

L Table 7.2 COBOL Library

Type of library	Library name
Dynamic library	libecpg.so
Static library	libecpg.a

W Windows(R)

If the include file and the library path have been set in the environment variable, there is no need to specify the options shown below for the compile.

W Table 7.3 Include file and library path

Option	How to specify the option
Path of the include file	*enterprisePostgresClientInstallDir\include*
Path of the library	*enterprisePostgresClientInstallDir\lib*

W Table 7.4 COBOL Library

Type of library	Library name
Library for links	libecpg.lib
Dynamic library	libecpg.dll

7.4.2 Bulk INSERT

Bulk INSERT can be used to input multiple rows of data into the table using a single ECOBPG statement that uses the newly introduced 'FOR' clause.

This functionality allows the user to make use of the data stored in host array variables, resulting in 'COBOL' client programs that are simpler and easier to maintain.

In sample codes, declarations of section and division, line numbers and/or indents may be omitted.

Synopsis

The syntax of the bulk INSERT statement is given below:

```
EXEC SQL [ AT connection ] [ FOR {number_of_rows|ARRAY_SIZE}]
    INSERT INTO table_name [ ( column_name [, ...] ) ]
    {  VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...]
      INTO output_host_var  [ [ INDICATOR ] indicator_var  ]  [, ...]] END-EXEC
```

When the above bulk INSERT command is used, ECOBPG inserts '*number_of_rows*' number of rows into the table, using the data that is stored in the '*expression*'.

FOR Clause

The 'FOR' clause indicates to ECOBPG that the given INSERT statement is a bulk insert statement. The 'FOR' clause currently only support INSERT statements. When a 'FOR' clause specified, ECOBPG executes the INSERT statement for 'number_of_rows' number of times, iterating through the host array variables.

The 'FOR' clause can iterate through all the array elements or can be limited to process only a fixed number of array elements. The value for 'number_of_rows' should be greater than zero.

The 'FOR' clause can accept an integer host variable(COMP or COMP-5) or an integer literal as its parameter. It can also accept the constant '*ARRAY_SIZE*' in which case all the elements in the host array variable are inserted into the table.

Given below are examples of acceptable bulk INSERT statements.

```
01 NUMBER-OF-ROWS PIC S9(9) COMP VALUE 10.
01 GROUP-ITEM.
    05 ID1 PIC S9(9) OCCURS 25.
    05 NAME PIC X(10) OCCURS 25.

* will process 10 rows
EXEC SQL FOR :NUMBER-OF-ROWS
INSERT INTO prod (name, id) VALUES (:NAME, :ID1) END-EXEC

* will process 25 rows
EXEC SQL FOR ARRAY_SIZE
INSERT INTO prod (name, id) VALUES (:NAME, :ID1) END-EXEC
```

When 'FOR' clause is specified, the '*expression*' can be a host array variable, host variable, constant number or string.

The value given for the 'number_of_rows' should be greater than zero. Value which is less than or equal to zero will result in a run-time error.

When 'FOR ARRAY_SIZE' is specified, the values clause should consist of at least one host array variable. 'SELECT' queries cannot be used for input values when using 'FOR ARRAY_SIZE'.

When the value given for the 'number_of_rows' is greater than one, the specified 'SELECT' query should returns only one row. More than one returned row will result in an error.

Values Clause

The VALUES clause includes the input data that is to be inserted into the table. When working with FOR clause, the values in the '*expression'* can be host array variables, host variable, constant number, string or pointers.

For more detailed usage of the INSERT statement, please refer to the ECOBPG section of the PostgreSQL documentation.

Error Messages

Given below are the error messages that are output when bulk INSERT functionality is not used correctly.

Invalid value for number_of_rows

ECOBPG error

invalid statement name "FOR value should be positive integer"

Cause

The value given for number_of_rows is less than or equal to 0.

Solution

Specify a value that is more than or equal to 1 for number_of_rows.

Invalid input for ARRAY_SIZE

ECOBPG error

invalid statement name "Host array variable is needed when using FOR ARRAY_SIZE"

Cause

A host array is not specified in the values clause when using the ARRAY_SIZE keyword.

Solution

At least one host array variable should be included in the values clause

Too many rows from SELECT... INTO

ECOBPG error

SELECT...INTO returns too many rows

Cause

The number of rows returned by the 'SELECT ... INTO' query in the INSERT statement is more than one.

Solution

When the value of 'number_of_rows' is more than one, the maximum number of rows that can be returned by the 'SELECT ... INTO' query in the INSERT statement is one.

Limitations

The limitations when using bulk INSERT are given below.

- Array of structures should not be used as an input in the 'VALUES' clause. Attempted use will result in junk data being inserted into the table.
- Array of pointers should not be used as an input in the 'VALUES' clause. Attempted use will result in junk data being inserted into the table.
- ECOBPG supports the use of 'WITH' clause in single INSERT statements. 'WITH' clause cannot be used in bulk INSERT statements.

Samples

Given below are some sample usages of the bulk INSERT functionality.

Basic Bulk INSERT

```
01 GROUP-ITEM.
    05 IN-F1 PIC S9(9) OCCURS 4.
MOVE 1 TO IN-F1(1)
MOVE 2 TO IN-F1(2)
MOVE 3 TO IN-F1(3)
MOVE 4 TO IN-F1(4)
...
EXEC SQL FOR 3 INSERT INTO target (f1) VALUES (:IN-F1) END-EXEC
```

The number of rows to insert indicated by the FOR clause is 3, so the data in the first 3 elements of the host array variable are inserted into the table. The contents of the target table will be:

```
f1
----
  1
  2
```

```
  3
(3 rows)
```

Also a host integer variable can be used to indicate the number of rows that will be inserted in FOR clause, which will produce the same result as above:

```
01 NUM PIC S9(9) COMP VALUE 3.
01 GROUP-ITEM.
    05 IN-F1 PIC S9(9) OCCURS 4.
MOVE 1 TO IN-F1(1)
MOVE 2 TO IN-F1(2)
MOVE 3 TO IN-F1(3)
MOVE 4 TO IN-F1(4)
...
EXEC SQL FOR :NUM INSERT INTO target (f1) VALUES (:IN-F1) END-EXEC
```

Inserting constant values

Constant values can also be bulk INSERTed into the table as follows:

```
EXEC SQL FOR 3 INSERT INTO target (f1,f2) VALUES (DEFAULT,'hello') END-EXEC
```

Assuming the 'DEFAULT' value for the 'f1' column is '0', the contents of the target table will be:

```
f1 |  f2
---+-------
 0 | hello
 0 | hello
 0 | hello
(3 rows)
```

Using ARRAY_SIZE

'FOR ARRAY_SIZE' can be used to insert the entire contents of a host array variable, without explicitly specifying the size, into the table.

```
01 GROUP-ITEM.
    05 IN-F1 PIC S9(9) OCCURS 4.
MOVE 1 TO IN-F1(1)
MOVE 2 TO IN-F1(2)
MOVE 3 TO IN-F1(3)
MOVE 4 TO IN-F1(4)
...
EXEC SQL FOR ARRAY_SIZE INSERT INTO target (f1) VALUES (:IN-F1) END-EXEC
```

If there are multiple host array variables specified as input values, then the number of rows inserted is same as the smallest array size. The example given below demonstrates this usage.

```
01 GROUP-ITEM.
    05 IN-F1 PIC S9(9) OCCURS 4.
    05 IN-F3 PIC X(10) OCCURS 3.
MOVE 1 TO IN-F1(1)
MOVE 2 TO IN-F1(2)
MOVE 3 TO IN-F1(3)
MOVE 4 TO IN-F1(4)

MOVE "one" TO IN-F3(1)
```

```
MOVE "two" TO IN-F3(2)
MOVE "three" TO IN-F3(3)
...
EXEC SQL FOR ARRAY_SIZE INSERT INTO target (f1,f3) VALUES (:IN-F1,:IN-F3) END-EXEC
```

In the above example, the array sizes are 3 and 4. Given that the smallest array size is 3, only three rows are inserted into the table. The table contents are given below.

```
 f1 |  f3
----+------
  1 | one
  2 | two
  3 | three
(3 rows)
```

Using SELECT query

When using bulk INSERT, the input values can be got from the results of a SELECT statement. For ex.,

```
EXEC SQL FOR 4 INSERT INTO target(f1) SELECT age FROM source WHERE name LIKE 'foo' END-
EXEC
```

Assuming that the 'SELECT' query returns one row, the same row will be inserted into the target table four times.

If the 'SELECT' query returns more than one row, the INSERT statement will throw an error.

If the 'number_of_rows' is '1' and the 'SELECT' query returns more than one row, then all the rows are inserted into the table.

```
EXEC SQL FOR 1 INSERT INTO target(f1) SELECT age FROM source END-EXEC
```

In the above example, all the rows returned by the 'SELECT' statement will be inserted into the table. In this context '1' has the meaning of 'returned row equivalent'.

Using RETURNING clause

Bulk INSERT supports the same RETURNING clause syntax as normal INSERT. An example is given below.

```
01 GROUP-ITEM.
    05 IN-F1 PIC S9(9) OCCURS 4.
    05 OUT-F1 PIC S9(9) OCCURS 4.
MOVE 1 TO IN-F1(1)
MOVE 2 TO IN-F1(2)
MOVE 3 TO IN-F1(3)
MOVE 4 TO IN-F1(4)
...
EXEC SQL FOR 3 INSERT INTO target (f1) VALUES (:IN-F1) RETURNING f1 INTO :OUT-F1 END-EXEC
```

After the execution of the above INSERT statement, the 'out_f1' array will have 3 elements with the values of '1','2' and '3'.

7.4.3 DECLARE STATEMENT

Refer to "6.4.4 DECLARE STATEMENT" in "Embedded SQL in C".

Chapter 8 SQL References

This chapter explains the SQL statement features expanded by Enterprise Postgres.

8.1 Expanded Trigger Definition Feature

This section explains the expanded trigger definition feature.

8.1.1 CREATE TRIGGER

In addition to features of PostgreSQL, triggers can be created with OR REPLACE option and DO option.

Synopsis

```
CREATE [ OR REPLACE ][ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event
[ OR ... ] }
    ON table_name
    [ FROM referenced_table_name ]
    { NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    { EXECUTE PROCEDURE function_name ( arguments ) | DO [ LANGUAGE lang_name ] code }
```

Description

Refer to the PostgreSQL Documentation for information about CREATE TRIGGER. This section describes OR REPLACE option and DO option.

A trigger which is created with OR REPLACE option and DO option will be associated with the specified table or view and will execute the specified code by the specified procedural language of DO (unnamed code block) when certain events occur.

Parameters

OR REPLACE

If the specified trigger is not defined in the table, it defines a new trigger.
If the specified trigger is already defined in the table, the named trigger replaces existing trigger.

code

When the certain events occur, it executes the code in a specified procedural language. The unnamed code block does not require a prior definition like a function. Syntax is same as procedural language.

lang_name

The name of the language that the function is implemented in. Can be SQL, C, internal, or the name of a user-defined procedural language. The default is 'plpgsql'.

plpgsql is supported in CREATE TRIGGER.

Note

- A normal trigger cannot be replaced by a constraint trigger.
- A constraint trigger cannot be replaced by a normal trigger.
- A trigger defined with DO option cannot be replaced by a trigger defined with EXECUTE PROCEDURE option.
- A trigger defined with EXECUTE PROCEDURE option cannot be replaced by a trigger defined with DO option.

Examples

It executes the code block that is specified by DO before the table is updated.
(Example that LANGUAGE is plpgsql)

```
CREATE TRIGGER check_update
    BEFORE UPDATE ON accounts
    FOR EACH ROW
    DO $$BEGIN RETURN NEW; END;$$ ;
```

Information

When a trigger created with DO option, a new function is created internally. The name of function is "schema name"."on table name"_"trigger name"_TRIGPROC(serial number).

8.1.2 How to Define Triggers in pgAdmin

The expanded features of the trigger definition can also be used in pgAdmin.

See

Refer to "pgAdmin Help" for information on how to define triggers using pgAdmin.

Chapter 9 Compatibility with Oracle Databases

This chapter describes the environment settings and functionality offered for features that are compatible with Oracle databases.

9.1 Overview

Features compatible with Oracle databases are provided. These features enable you to easily migrate to Enterprise Postgres and reduce the costs of reconfiguring applications.

The following compatible features are provided:

Table 9.1 Features compatible with Oracle databases

Category		Features compatible with Oracle databases	
		Item	Overview
SQL	Queries	Outer join operator (+)	Operator for outer joining
		DUAL table	Tables provided by the system
	Functions	DECODE	Compares and converts values
		SUBSTR	Extracting part of a character string
		NVL	NULL value conversion
Package		DBMS_OUTPUT	Message sending
		UTL_FILE	File operation
		DBMS_SQL	Dynamic SQL execution

9.2 Precautions when Using the Features Compatible with Oracle Databases

Features compatible with Oracle databases are defined as user-defined functions in the "public" schema created by default when database clusters are created, so they can be available for all users without the need for special settings.

For this reason, ensure that "public" (without the double quotation marks) is included in the list of schema search paths specified in the search_path parameter.

9.2.1 Notes on SUBSTR

SUBSTR is implemented on Enterprise Postgres and Oracle databases using different external specifications.

For this reason, when using SUBSTR, define which specification is to be prioritized. The default setting executes with the Enterprise Postgres specifications prioritized.

When using the SUBSTR function compatible with Oracle databases, set "oracle" and "pg_catalog" in the "search_path" parameter of postgresql.conf. You must specify "oracle" in front of "pg_catalog" when doing this.

```
search_path = '"$user", public, oracle, pg_catalog'
```

Information

- The search_path feature specifies the priority of the schema search path. The SUBSTR function in Oracle database is defined in the oracle schema.
- Refer to "Statement Behavior" in "Server Administration" in the PostgreSQL Documentation for information on search_path.

9.2.2 Notes when Integrating with the Interface for Application Development

The SQL noted in "Table 9.1 Features compatible with Oracle databases" can be used in the interface for application development. However, outer join operators cannot be used when integrated with Visual Studio.

When integrated with Visual Studio or using the features compatible with Oracle databases from Fujitsu Npgsql .NET Data Provider, select one of the actions below for the SearchPath parameter, which is one of the pieces of information needed to connect to databases specified for individual connections.

- Do not specify the SearchPath parameter itself, or
- Specify both "public" and the schema name in the SQL statement.

Note that both "public" and the schema name in the SQL statement must be specified as the SearchPath parameter before "oracle" and "pg_catalog" when using the Oracle database-compatible feature SUBSTR.

9.3 Queries

The following queries are supported:

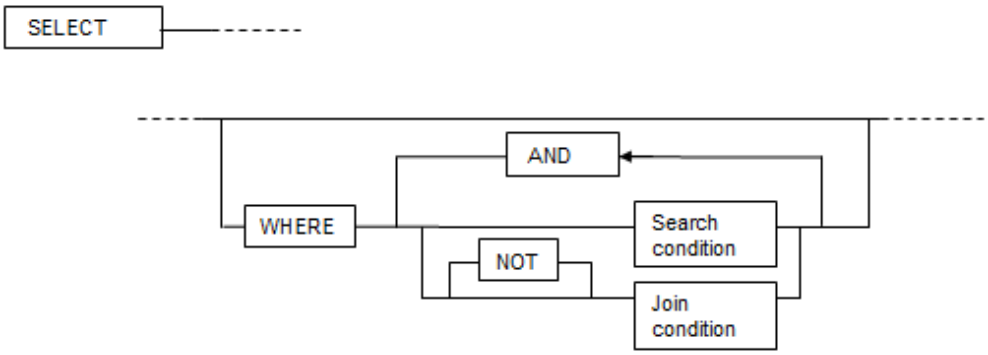
- Outer Join Operator (+)
- DUAL Table

9.3.1 Outer Join Operator (+)

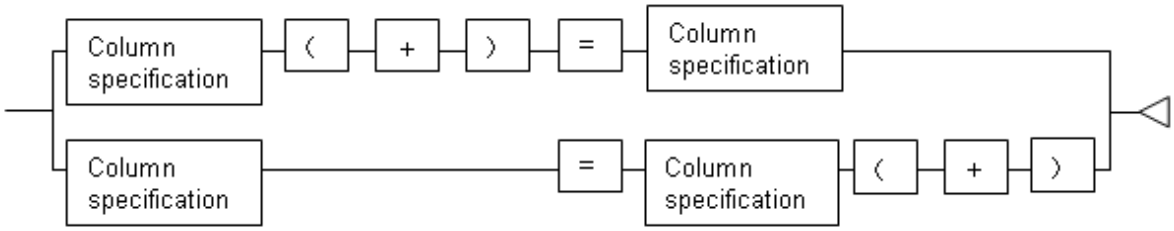
In the WHERE clause conditional expression, by adding the plus sign (+), which is the outer join operator, to the column of the table you want to add as a table join, it is possible to achieve an outer join that is the same as a joined table (OUTER JOIN).

Specification format

SELECT statement



Join condition



Note

Here we are dealing only with the WHERE clause of the SELECT statement. Refer to "SQL Commands" in "Reference" in the PostgreSQL Documentation for information on the overall specification format of the SELECT statement.

General rules

WHERE clause

- The WHERE clause specifies search condition or join conditions for the tables that are derived.
- Search conditions are any expressions that return BOOLEAN types as the results of evaluation. Any lines that do not meet these conditions are excluded form the output. When the values of the actual lines are assigned to variables and if the expression returns "true", those lines are considered to have met the conditions.
- Join conditions are comparison conditions that specify outer join operators. Join conditions in a WHERE clause return a table that includes all the lines that meet the join conditions, including lines that do not meet all the join conditions.
- Join conditions are prioritized over search conditions. For this reason, all lines returned by the join conditions are subject to the search conditions.
- The following rules and restrictions apply to queries that use outer join operators. We therefore recommend the use of FROM clause joined tables (OUTER JOIN) rather than outer join operators:
  - Outer join operators can only be specified in the WHERE clause.
  - Outer join operators can only be specified for base tables or views.
  - To perform outer joins using multiple join conditions, it is necessary to specify outer join operators for all join conditions.
  - When combining join conditions with constants, specify outer join operators in the corresponding column specification. When not specified, they will be treated as search conditions.
  - The results column of the outer join of table t1 is not returned if table t1 is joined with table t2 by specifying an outer join operator in the column of t1, then table t1 is joined with table t3 by using search conditions.
  - It is not possible to specify columns in the same table as the left/right column specification of a join condition.
  - It is not possible to specify an expression other than a column specification for outer join operators, but they may be specified for the columns that compose the expression.

    There are the following limitations on the functionality of outer join operators when compared with joined tables (OUTER JOIN). To use functionality that is not available with outer join operators, use joined tables (OUTER JOIN).

Table 9.2 Range of functionality with outer join operators

Functionality available with joined tables (OUTER JOIN)	Outer join operator
Outer joins of two tables	Y
Outer joins of three or more tables	Y (*1)
Used together with joined tables within the same query	N
Use of the OR logical operator to a join condition	N
Use of an IN predicate to a join condition	N
Use of subqueries to a join condition	N

Y: Available
N: Not available

*1: The outer joins by outer join operators can return outer join results only for one other table. For this reason, to combine outer joins of table t1 and table t2 or table t2 and table t3, it is not possible to specify outer join operators simultaneously for table t2.

Example

Table configuration

t1

col1	col2	col3
1001	AAAAA	1000
1002	BBBBB	2000
1003	CCCCC	3000

t2

col1	col2
1001	aaaaa
1002	bbbbb
1004	ddddd

Example 1:

In the following example, all records in table t2 that include ones that do not exist in table t1 are returned:

```
SELECT *
  FROM t1, t2
  WHERE t1.col1(+) = t2.col1;
 col1 |    col2    | col3 | col1 |    col2
------+------------+------+------+------------
 1001 | AAAAA      | 1000 | 1001 | aaaaa
 1002 | BBBBB      | 2000 | 1002 | bbbbb
      |            |      | 1004 | ddddd
(3 rows)
```

This is the same syntax as the joined table (OUTER JOIN) of the FROM clause shown next.

```
SELECT *
  FROM t1 RIGHT OUTER JOIN t2
       ON t1.col1 = t2.col1;
```

Example 2:

In the following example, the results are filtered to records above 2000 in t1.col3 by search conditions, and the records are those in table t2 that include ones that do not exist in table t1. After filtering with the join conditions, there is further filtering with the search conditions, so there will only be one record returned.

```
SELECT *
  FROM t1, t2
  WHERE t1.col1(+) = t2.col1
    AND t1.col3 >= 2000;
 col1 |    col2    | col3 | col1 |    col2
------+------------+------+------+------------
 1002 | BBBBB      | 2000 | 1002 | bbbbb
(1 row)
```

This is the same syntax as the joined table (OUTER JOIN) of the FROM clause shown next.

```
SELECT *
  FROM t1 RIGHT OUTER JOIN t2
       ON t1.col1 = t2.col1
  WHERE t1.col3 >= 2000;
```

9.3.2 DUAL Table

DUAL table is a virtual table provided by the system. Use when executing SQL where access to a base table is not required, such as when performing tests to get result expressions such as functions and operators.

Example

The following example shows acquiring the current date from the system:

```
SELECT  CURRENT_DATE  "date" FROM DUAL;
    date
------------
 2013-05-14
(1 row)
```

9.4 SQL Function Reference

The following SQL functions are supported:

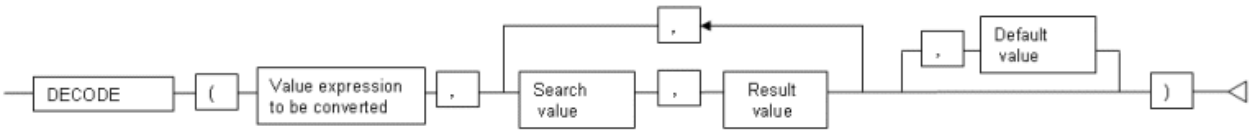
- DECODE
- SUBSTR
- NVL

9.4.1 DECODE

Features

Compares values and converts to other values.

Specification format



General rules

- DECODE compares values of the value expression to be converted and the search values one by one, and if the values of the value expression to be converted and the search values match, a corresponding result value is returned. If none of the value expressions to be converted and search values match, the default value is returned if a default value has been specified. A NULL value is returned if a default value has not been specified.
- If the same search value is specified more than once, then the result value returned is the one listed for the first occurrence of the search value.
- The following data types can be used in result values and in the default value:
  - CHAR
  - VARCHAR

  - NCHAR
  - NCHAR VARYING
  - TEXT
  - INTEGER
  - BIGINT
  - NUMERIC
  - DATE
  - TIME WITHOUT TIME ZONE
  - TIMESTAMP WITHOUT TIME ZONE
  - TIMESTAMP WITH TIME ZONE
- The same data type must be specified for all value expressions to be converted and the search values. However, note that different data types may also be specified if a literal is specified in the search value, and the value expressions that will be converted contain data types that can be converted. Refer to "Table A.1 Data type combinations that contain literals and can be converted implicitly" in "A.3 Implicit Data Type Conversions" for information on data types that can be specified when a literal is specified in the search value.
- If the result values and default value are all literals, the data types for these values will be as shown below:
  - If all values are character string literals, all will become character string types.
  - If there is one or more numeric literal, all will become numeric types.
  - If there is one or more literal cast to the datetime/time types, all will become datetime/time types.
- If the result values and default value contain a mixture of literals and non-literals, the literals will be converted to the data types of the non-literals. Refer to "Table A.1 Data type combinations that contain literals and can be converted implicitly" in "A.3 Implicit Data Type Conversions" for information on data types that can be converted.
- The same data type must be specified for all result values and for the default value. However, different data types can be specified if the data type of any of the result values or default value can be converted - these data types are listed below:

Table 9.3 Data type combinations that can be converted by DECODE (summary)

		Other result values or default value		
		Numeric type	Character string type	Date/time type
Result value (any)	Numeric type	Y	N	N
	Character string type	N	Y	N
	Date/time type	N	N	S (*1)

Y: Can be converted
S: Some data types can be converted
N: Cannot be converted

*1: The data types that can be converted for date/time types are listed below:

Table 9.4 Result value and default value date/time data types that can be converted by DECODE

		Other result values or default value			
		DATE	**TIME** WITHOUT TIME ZONE	**TIMESTAMP** WITHOUT TIME ZONE	**TIMESTAMP WITH TIME ZONE**
Result value (any)	DATE	Y	N	Y	Y
	TIME WITHOUT TIME ZONE	N	Y	N	N
	TIMESTAMP WITHOUT TIME ZONE	Y	N	Y	Y
	TIMESTAMP WITH TIME ZONE	Y	N	Y	Y

Y: Can be converted
N: Cannot be converted

- The data type of the return value will be the data type within the result or default value that is longest and has the highest precision.

Example

In the following example, the value of col3 in table t1 is compared and converted to a different value. If the col3 value matches search value 1, the result value returned is "one". If the col3 value doe not match any of search values 1, 2, or 3, the default value "other number" is returned.

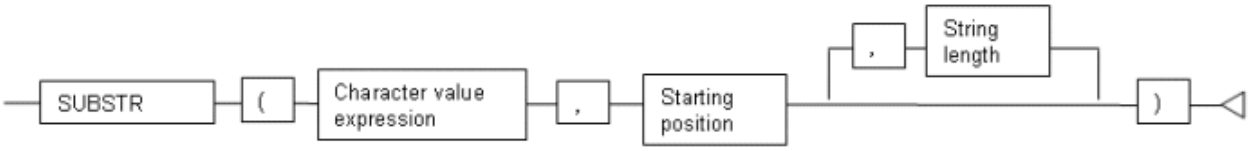
```
SELECT col1,
       DECODE(col3, 1, 'one',
                    2, 'two',
                    3, 'three',
                    'other number') "num-word"
       FROM t1;
col1  | num-word
------+----------
 1001 | one
 1002 | two
 1003 | three
(3 rows)
```

9.4.2 SUBSTR

Features

Extracts part of a character string.

Specification format



General rules

- SUBSTR returns the number of characters specified in the third argument (starting from the position specified in the second argument) from the string specified in the first argument.
- When the starting position is positive, the starting position will be from the beginning of the character value expression.

- When the starting position is 0, it will be that same as if 1 is specified in the starting position.
- When the starting position is negative, the starting position will be from the end of the character value expression.
- When string length is not specified, all characters until the end of the character value expression are returned. NULL is returned when the string length is less than 1.
- Specify SMALLINT or INTEGER as the data type for the start position and string length. Refer to "Table A.1 Data type combinations that contain literals and can be converted implicitly" in "A.3 Implicit Data Type Conversions" for information on data types that can be specified when a literal is specified.
- The data type of the return value is TEXT.

Note

- There are two types of SUBSTR. One that behaves as described above, and one that behaves the same as SUBSTRING. The search_path needs to be modified for it to behave the same as the specification described above.
- It is recommended to set search_path in postgresql.conf. In this case, it will be effective for each instance. Refer to "9.2.1 Notes on SUBSTR" for information on how to configure postgresql.conf.
- The configuration of search_path can be done at the user level or at the database level. Setting examples are shown below.
  - Example of setting at the user level

    This can be set by executing an SQL command. "user1" will be used as the username in this example.

    ```
    ALTER USER user1 SET search_path = "$user",public,oracle,pg_catalog;
    ```

  - Example of setting at the database level

    This can be set by executing an SQL command. "db1" will be used as the database name in this example.

    ```
    ALTER DATABASE db1 SET search_path = "$user",public,oracle,pg_catalog;
    ```

    You must specify "oracle" in front of "pg_catalog".
- If the change has not been implemented, SUBSTR is the same as SUBSTRING.

See

Refer to "SQL Commands" under "Reference" in the PostgreSQL Documentation for information on ALTER USER and ALTER DATABASE.

Information

The general rules for SUBSTRING are as follows:

- The starting position will be from the beginning of the character value expression, whether the starting position is positive, 0, or negative.
- When string length is not specified, all characters until the end of the character value expression are returned.
- An empty string is returned if the returned string is 0 or less or if the specified string length is less than 1.

See

Refer to "String Functions and Operators" under "The SQL Language" in the PostgreSQL Documentation for information on SUBSTRING.

Example

In the following example, part of the string "ABCDEFG" is extracted:

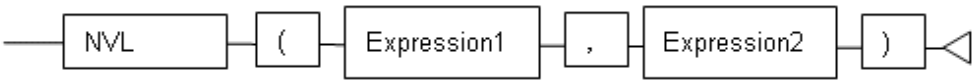
```
SELECT SUBSTR('ABCDEFG',3,4) "Substring" FROM DUAL;

 Substring
-----------
 CDEF
(1 row)

SELECT SUBSTR('ABCDEFG',-5,4) "Substring" FROM DUAL;

 Substring
-----------
 CDEF
(1 row)
```

9.4.3 NVL

Features

Converts NULL values.

Specification format



General rules

- NVL converts NULL values. When expression 1 is NULL, expression 2 is returned. When expression 1 is not NULL, expression 1 is returned.
- Specify the same data types for expression 1 and expression 2. However, if a constant is specified in expression 2, and the data type can also be converted by expression 1, different data types can be specified. When this happens, the conversion by expression 2 is done to suit the data type in expression 1, so the value of expression 2 returned when expression 1 is a NULL value will be the value converted in the data type of expression 1.
- Refer to "Table A.1 Data type combinations that contain literals and can be converted implicitly" in "A.3 Implicit Data Type Conversions" for information on data types that can be converted for literals.

Example

In the following example, "IS NULL" is displayed if the value of col1 in table t1 is a NULL value.

```
SELECT col2, NVL(col1,'IS NULL') "nvl" FROM t1;
 col2 |   nvl
------+---------
 aaa  | IS NULL
(1 row)
```

9.5 Package Reference

A "package" is a group of features, brought together by schemas, that have a single functionality, and are used by calling from PL/pgSQL.

The following packages are supported:

- DBMS_OUTPUT
- UTL_FILE
- DBMS_SQL

To call the different functionalities from PL/pgSQL, use the PERFORM statement or SELECT statement, using the package name to qualify the name of the functionality. Refer to the explanations for each of the package functionalities for information on the format for calling.

9.5.1 DBMS_OUTPUT

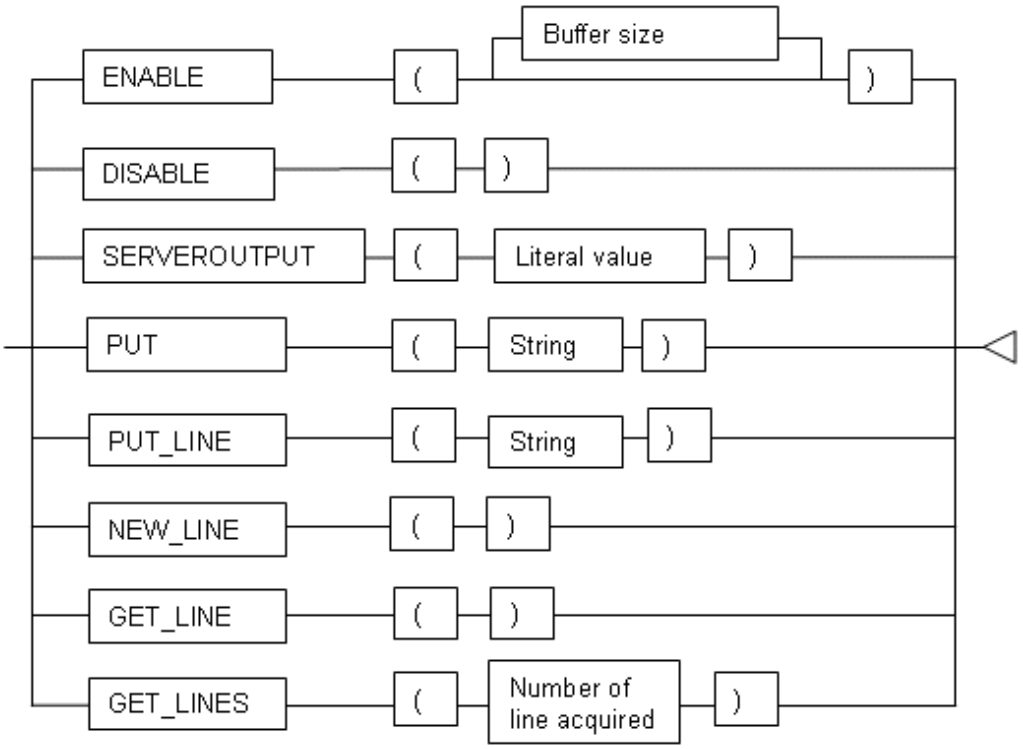
Overview

Sends messages to clients such as psql from PL/pgSQL.

Features

Features	Description
ENABLE	Features of this package are enabled.
DISABLE	Features of this package are disabled.
SERVEROUTPUT	Controls whether messages are sent.
PUT	Messages are sent.
PUT_LINE	Messages are sent with a newline appended.
NEW_LINE	Newlines are sent as messages.
GET_LINE	1 line is read from the buffer.
GET_LINES	Multiple lines are read from the buffer.

Specification format



9.5.1.1 Description

This section explains the procedures available in the DBMS_OUTPUT package.

ENABLE

- ENABLE enables the use of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES.
- With multiple executions of ENABLE, the value specified last is the buffer size (in bytes). Specify in the range between 2000 and 1000000 if a buffer size is to be specified.
- The default value of the buffer size is 20000. If NULL is specified as the buffer size, 1000000 will be used.
- If ENABLE has not been executed, PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES are ignored even if they are executed.

Example

```
PERFORM DBMS_OUTPUT.ENABLE(20000);
```

DISABLE

- DISABLE disables the use of PUT, PUT_LINE, NEW_LINE, GET_LINE, and GET_LINES.
- Remaining buffer information is discarded.

Example

```
PERFORM DBMS_OUTPUT.DISABLE();
```

SERVEROUTPUT

- SERVEROUTPUT controls whether messages are sent.
- The logical value specifies whether to send messages.
- If the "true" logical value is specified, then when PUT, PUT_LINE, or NEW_LINE is executed, the message is sent to a client such as psql and not stored in the buffer.
- If the "false" logical value is specified, then when PUT, PUT_LINE, or NEW_LINE is executed, the message is stored in the buffer and not sent to a client such as psql.

See

Refer to "Boolean Type" in "The SQL Language" in the PostgreSQL Documentation for information on logical values.

Example

```
PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
```

PUT

- PUT configures the message that is sent.
- The string is the message that is sent.
- When "True" is specified in the logical value at SERVEROUTPUT, the messages are sent to clients such as psql.

- When "False" is specified in the logical value at SERVEROUTPUT, the messages are retained in the buffer.
- PUT does not append a newline, so to append a newline, execute NEW_LINE.
- If a string longer than the buffer size specified in ENABLE is sent, an error occurs.

Example

```
PERFORM DBMS_OUTPUT.PUT('abc');
```

PUT_LINE

- PUT_LINE appends a newline to messages for sending.
- The string is the message that is sent.
- When "True" is specified in the logical value at SERVEROUTPUT, the messages are sent to clients such as psql.
- When "False" is specified in the logical value at SERVEROUTPUT, the messages are retained in the buffer.
- PUT_LINE appends a newline to the end of messages.
- If a string longer than the buffer size specified in ENABLE is sent, an error occurs.

Example

```
PERFORM DBMS_OUTPUT.PUT_LINE('abc');
```

NEW_LINE

- NEW_LINE sets a newline in the message that is sent.
- When "True" is specified in the logical value at SERVEROUTPUT, the messages are sent to clients such as psql.
- When "False" is specified in the logical value at SERVEROUTPUT, the messages are retained in the buffer.

Example

```
PERFORM DBMS_OUTPUT.NEW_LINE();
```

GET_LINE

- GET_LINE extracts one line messages stored in the buffer.
- The information extracted is acquired as the values of the line and status columns with the SELECT statement.
- line is a TEXT type. A one line message extracted from the buffer is stored.
- status is an INTEGER type. The extracted result will be stored. 0 is stored when successful. 1 is stored when it fails.
- If GET_LINE or GET_LINES is executed, then PUT, PUT_LINE, or NEW_LINE is executed with message still not extracted from the buffer, the messages not extracted from the buffer will be discarded.

Example

```
DECLARE
    buff1   VARCHAR(20);
    stts1   INTEGER;
BEGIN
    SELECT line,status INTO buff1,stts1 FROM DBMS_OUTPUT.GET_LINE();
```

GET_LINES

- GET_LINES extracts multiple line messages stored in the buffer.
- The information extracted is acquired as the values of the lines and numlines columns with the SELECT statement.
- lines is a TEXT type. It stores the lines acquired from the buffer.
- numlines is an INTEGER type. It stores the number of lines acquired from the buffer.
- The number of lines acquired is an INTEGER type. It is the maximum number of lines extracted from the buffer.
- If GET_LINE or GET_LINES is executed, then PUT, PUT_LINE, or NEW_LINE is executed with message still not extracted from the buffer, the messages not extracted from the buffer will be discarded.

Example

```
DECLARE
    buff    VARCHAR(20)[10];
    stts    INTEGER := 10;
BEGIN
    SELECT lines, numlines INTO buff,stts FROM DBMS_OUTPUT.GET_LINES(stts);
```

9.5.1.2 Example

A usage example of DBMS_OUTPUT is shown below.

```
CREATE FUNCTION dbms_output_exe() RETURNS VOID AS $$
DECLARE
    buff1    VARCHAR(20);
    buff2    VARCHAR(20);
    stts1    INTEGER;
    stts2    INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT.DISABLE();
    PERFORM DBMS_OUTPUT.ENABLE();
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(FALSE);
    PERFORM DBMS_OUTPUT.PUT('DBMS_OUTPUT TEST 1');
    PERFORM DBMS_OUTPUT.NEW_LINE();
    PERFORM DBMS_OUTPUT.PUT_LINE('DBMS_OUTPUT TEST 2');
    SELECT line,status INTO buff1,stts1 FROM DBMS_OUTPUT.GET_LINE();
    SELECT line,status INTO buff2,stts2 FROM DBMS_OUTPUT.GET_LINE();
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    PERFORM DBMS_OUTPUT.PUT_LINE(buff1);
    PERFORM DBMS_OUTPUT.PUT_LINE(buff2);
END;
$$ LANGUAGE plpgsql;
SELECT dbms_output_exe();
DROP FUNCTION dbms_output_exe();
```

9.5.2 UTL_FILE

Overview

Text files can be written and read using PL/pgSQL.

To perform these file operations, the directory for the operation target must be registered in the UTL_FILE.UTL_FILE_DIR table beforehand. Use the INSERT statement as the database administrator or a user who has INSERT privileges to register the directory. Also, if the directory is no longer necessary, delete it from the same table. Refer to "9.5.2.1 Registering and Deleting Directories" for information on the how to register and delete the directory.

Refer to "C.1 UTL_FILE.UTL_FILE_DIR" for information on the UTL_FILE.UTL_FILE_DIR table.

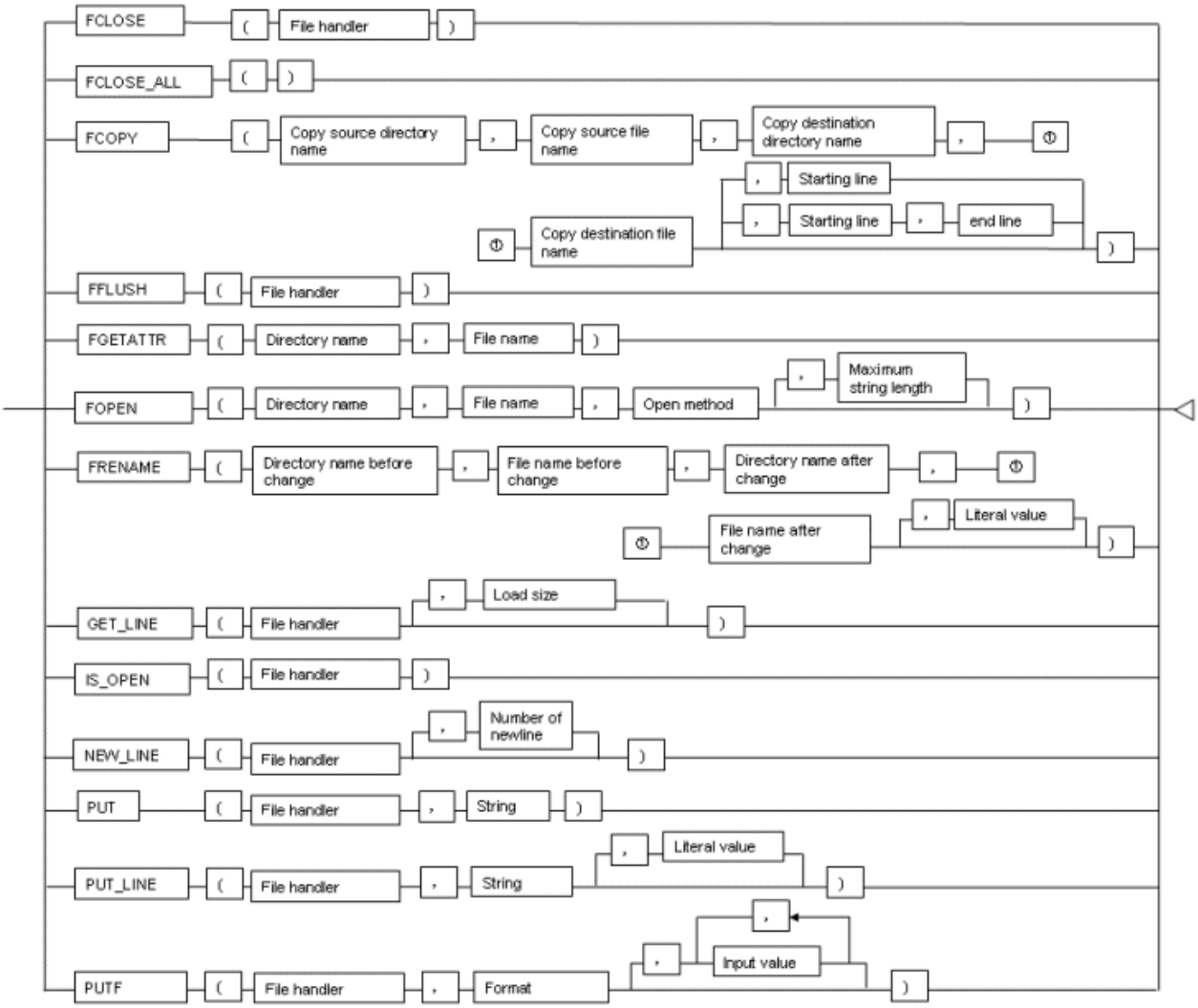
Declare the file handler explained hereafter as follows in PL/pgSQL:

```
DECLARE
f UTL_FILE.FILE_TYPE;
```

Features

Feature	Description
FCLOSE	Closes the specified file.
FCLOSE_ALL	Closes all files open in a session.
FCOPY	Copies a file.
FFLUSH	Flushes the buffer.
FGETATTR	Acquires the file existence and the size information.
FOPEN	Opens a file.
FRENAME	Changes the name of the file.
GET_LINE	Reads one line from a text file.
IS_OPEN	Checks if there are open files.
NEW_LINE	Writes a newline.
PUT	Writes a character string.
PUT_LINE	Adds a string and a newline and writes them.
PUTF	Writes a formatted character string.

Specification format



9.5.2.1 Registering and Deleting Directories

The examples in this sections are for Linux. In Windows(R), use a forward slash (/) as the separator for the directory.

Windows(R) example: 'c:/ fsep'

Registering the directory

1. Check if the directory is already registered (if it is, then step 2 is not necessary).

```
SELECT * FROM UTL_FILE.UTL_FILE_DIR WHERE dir='/home/fsep';
```

2. Register the directory.

```
INSERT INTO UTL_FILE.UTL_FILE_DIR VALUES('/home/fsep');
```

Deleting the directory

```
DELETE FROM UTL_FILE.UTL_FILE_DIR WHERE dir='/home/fsep';
```

9.5.2.2 Description

This section explains the procedures available in the UTL_FILE package.

FCLOSE

- FCLOSE closes a file that is open.
- Specify an open file handler as the file handler.
- The value returned is a NULL value.

Example

```
f := UTL_FILE.FCLOSE(f);
```

FCLOSE_ALL

- FCLOSE_ALL closes all files open in a session.
- Files closed with FCLOSE_ALL can no longer be read or written.

Example

```
PERFORM UTL_FILE.FCLOSE_ALL();
```

FCOPY

- FCOPY copies from a copy source file to a copy destination file.
- Specify the directory where the copy source file exists as the copy source directory name.
- Specify the copy source file name as the file name of the copy source.
- Specify the directory where the copy destination file is stored as the copy destination directory name.
- Specify the copy destination file name as the file name of the copy destination.

- Specify a slash (/) or a backslash (\) as a separator in the copy source directory name and the copy destination directory name.
- Specify the starting line number of the file to be copied as the beginning line. If specifying, specify a value greater than 0. The starting line number is 1 if this is not specified.
- Specify the end line number of the file to be copied as the end line. The last line number of the file is used if this is not specified.

Example

Linux

```
PERFORM UTL_FILE.FCOPY('/home/fsep', 'regress_fsep.txt', '/home/fsep',
'regress_fsep2.txt');
```

Windows(R)

```
PERFORM UTL_FILE.FCOPY('c:/fsep', 'regress_fsep.txt', 'c:/fsep', 'regress_fsep2.txt');
```

FFLUSH

- FFLUSH forcibly writes the buffer data to a file.
- Specify an open file handler as the file handler.

Example

```
PERFORM UTL_FILE.FFLUSH(f);
```

FGETATTR

- FGETATTR extracts the file existence, file size, and information about the block size of the file.
- Specify the directory where the relevant file exists as the directory name.
- Specify a slash (/) or a backslash (\) as a separator in the directory name.
- Specify the relevant file name as the file name.
- The information extracted is acquired as the values of the fexists, file_length, and blocksize columns with the SELECT statement.
- fexists is a BOOLEAN type. If the file exists, the logical value is "true". If the file does not exist, the logical value is "false".
- file_length is an INTEGER type. The length of the file is set in bytes. If the file does not exist, the value is NULL.
- blocksize is an INTEGER type. The block size of the file is set in bytes. If the file does not exist, the value is NULL.

Example

Linux

```
SELECT fexists, file_length, blocksize INTO file_flag, file_chack_length, size FROM
UTL_FILE.FGETATTR('/home/fsep', 'regress_fsep.txt');
```

Windows(R)

```
SELECT fexists, file_length, blocksize INTO file_flag, file_chack_length, size FROM
UTL_FILE.FGETATTR('c:/fsep', 'regress_fsep.txt');
```

FOPEN

- FOPEN opens a file.
- Specify the directory where the relevant file exists as the directory name.
- Specify a slash (/) or a backslash (\) as a separator in the directory name.
- Specify the relevant file name as the file name.
- Specify the mode for opening the file at open method. Specify the following:
  r: Read
  w: Write
  a: Add

- Specify the maximum string length (in bytes) that can be processed with one operation. If omitted, the default is "1024". Specify in the range from 1 to 32767.
- Up to 50 files per session can be open at the same time.

Example

Linux

```
f := UTL_FILE.FOPEN('/home/fsep','regress_fsep.txt','r',1024);
```

Windows(R)

```
f := UTL_FILE.FOPEN('c:/fsep','regress_fsep.txt','r',1024);
```

FRENAME

- FRENAME changes the name of an existing file.
- The directory name before change is the directory where the file was before change.
- The file name before change is the file name before change.
- The directory name after change is the directory where the file was created after change.
- The file name after change is the file name after change.
- Specify a slash (/) or a backslash (\) as a separator in the directory name.
- Specify whether to overwrite a file with changes, if one exists in the directory with the logical value. If the "true" logical value is specified, files with changes are overwritten even if they exist. If the "false" logical value is specified, an error occurs if files with changes exist. If this is omitted, the logical value for "False" will be set.

See

Refer to "Boolean Type" in "The SQL Language" in the PostgreSQL Documentation for information on logical values.

Example

Linux

```
PERFORM UTL_FILE.FRENAME('/home/fsep', 'regress_fsep.txt', '/home/fsep',
 'regress_fsep2.txt', TRUE);
```

Windows(R)

```
PERFORM UTL_FILE.FRENAME('c:/fsep', 'regress_fsep.txt', 'c:/fsep',
 'regress_fsep2.txt', TRUE);
```

GET_LINE

- GET_LINE reads one line from a file.
- Specify an open file handler as the file handler. Specify the file handler returned by FOPEN using the r (read) mode.

- The load size is the number of bytes to be loaded from the file. If not specified, it will be the maximum string length of FOPEN.
- The return value is the buffer that receives the line loaded from the file.
- Newline characters are not loaded to the buffer.
- An empty string is returned if a blank line is loaded.
- Specify the maximum size(in bytes) of the data to be loaded at load size. Specify in the range from 1 to 32767. If a load size is not specified, then the maximum string length specified at FOPEN is set when a maximum string length is specified at FOPEN. 1024 is set if a maximum string length is not set at FOPEN.
- If the line length is greater than the load size, the load size is loaded and the remaining is loaded upon the next call.
- The NO_DATA_FOUND will be raised when trying to read past the last line.

Example

```
buff := UTL_FILE.GET_LINE(f);
```

IS_OPEN

- IS_OPEN checks if there are open files.
- Specify the file handler to be verified as the file handler.
- The return value is a BOOLEAN type. A "true" logical value represents the open state and the "false" logical value represents the closed state.

See

Refer to "Boolean Type" under "The SQL Language" in the PostgreSQL Documentation for information on logical values.

Example

```
IF UTL_FILE.IS_OPEN(f) THEN
  PERFORM UTL_FILE.FCLOSE(f);
END IF;
```

NEW_LINE

- NEW_LINE writes one or more newlines.
- Specify an open file handler as the file handler.
- The number of newlines is the number of newlines to be written to the file. If omitted, "1" is set.

Example

```
PERFORM UTL_FILE.NEW_LINE(f, 2);
```

PUT

- PUT writes a string to a file.
- Specify an open file handler as the file handler. Specify the file handler that was opened with w (write) or a (append) with the FOPEN opening method.
- Specify the character string to be written to the file for the character string.
- The maximum size(in bytes) of the string is the maximum string length specified at FOPEN.
- The return value is a TEXT type and is the buffer that receives the line loaded from the file.

Example

```
PERFORM UTL_FILE.PUT(f, 'ABC');
```

PUT_LINE

- PUT_LINE adds a newline to a string and writes it.
- Specify an open file handler as the file handler. Specify the file handler that was opened with w (write) or a (append) with the FOPEN opening method.
- The logical value specifies whether to forcibly write to the file. If the "true" logical value is specified, file writing is forced. If the "false" logical value is specified, file writing is asynchronous. If this is omitted, the logical value for "False" will be set.
- The maximum size of the string (in bytes) is the maximum string length value specified at FOPEN.

Example

```
PERFORM UTL_FILE.PUT_LINE(f, 'ABC', TRUE);
```

PUTF

- PUTF writes a string that uses formatting.
- Specify an open file handler as the file handler. Specify the file handler that was opened with w (write) or a (append) with the FOPEN opening method.
- Format is a string that includes the formatting characters \n and %s.
- The \n in the format is code for a newline.
- Specify the same number of input values as there are %s in the format. Up to a maximum of five input values can be specified. The %s in the format are replaced with the corresponding input characters. If an input value corresponding to %s is not specified, it is replaced with an empty string.

Example

```
PERFORM UTL_FILE.PUTF(f, '[1=%s, 2=%s, 3=%s, 4=%s, 5=%s]\n', '1', '2', '3', '4', '5');
```

9.5.2.3 Example

The procedure when using UTL_FILE, and a usage example, are shown below.

1. Preparation

   Before starting a new job that uses UTL_FILE, register the directory in the UTL_FILE.UTL_FILE_DIR table.

   Refer to "9.5.2.1 Registering and Deleting Directories" for information on how to register the directory.

2. Performing a job

   Perform a job that uses UTL_FILE. The example is shown below.

```
CREATE OR REPLACE FUNCTION gen_file(mydir TEXT, infile TEXT, outfile TEXT, copyfile
TEXT) RETURNS void AS $$
DECLARE
  v1 VARCHAR(32767);
  inf UTL_FILE.FILE_TYPE;
  otf UTL_FILE.FILE_TYPE;
BEGIN
  inf := UTL_FILE.FOPEN(mydir, infile,'r',256);
  otf := UTL_FILE.FOPEN(mydir, outfile,'w');
  v1 := UTL_FILE.GET_LINE(inf,256);
  PERFORM UTL_FILE.PUT_LINE(otf,v1,TRUE);
  v1 := UTL_FILE.GET_LINE(inf,256);
  PERFORM UTL_FILE.PUTF(otf,'%s\n',v1);
  v1 := UTL_FILE.GET_LINE(inf, 256);
  PERFORM UTL_FILE.PUT(otf,v1);
  PERFORM UTL_FILE.NEW_LINE(otf);
  PERFORM UTL_FILE.FFLUSH(otf);

  inf := UTL_FILE.FCLOSE(inf);
  otf := UTL_FILE.FCLOSE(otf);

  PERFORM UTL_FILE.FCOPY(mydir, outfile, mydir, copyfile, 2, 3);
  PERFORM UTL_FILE.FRENAME(mydir, outfile, mydir, 'rename.txt');

END;
$$ LANGUAGE plpgsql;

/* Linux */
SELECT gen_file('/home/fsep', 'input.txt', 'output.txt', 'copyfile.txt');

/* Windows(R) */
SELECT gen_file('c:/fsep', 'input.txt', 'output.txt', 'copyfile.txt');
```

3. Post-processing

   If you remove a job that uses UTL_FILE, delete the directory information from the UTL_FILE.UTL_FILE_DIR table. Ensure that the directory information is not being used by another job before deleting it.

   Refer to "9.5.2.1 Registering and Deleting Directories" for information on how to delete the directory.

9.5.3 DBMS_SQL

Overview

Dynamic SQL can be executed from PL/pgSQL.

Features

Feature	Description
BIND_VARIABLE	Sets values in the host variable within the SQL statement.
CLOSE_CURSOR	Closes the cursor.
COLUMN_VALUE	Acquires the value of the column in the select list extracted with FETCH_ROWS.

Feature	Description
DEFINE_COLUMN	Defines the column from which values are extracted and the storage destination.
EXECUTE	Executes SQL statements.
FETCH_ROWS	Positions the specified cursor at the next line and extracts values from the line.
OPEN_CURSOR	Opens a new cursor.
PARSE	Parses SQL statements.

Note

- In DBMS_SQL, the data types supported in dynamic SQL are limited, and therefore the user must be careful - they are:
  - INTEGER
  - DECIMAL
  - NUMERIC
  - REAL
  - DOUBLE PRECISION
  - CHAR(*1)
  - VARCHAR(*1)
  - NCHAR(*1)
  - NCHAR VARYING(*1)
  - TEXT
  - DATE
  - TIMESTAMP WITHOUT TIME ZONE
  - TIMESTAMP WITH TIME ZONE
  - INTERVAL(*2)
  - SMALLINT
  - BIGINT

  ***1:**
  The host variables with CHAR, VARCHAR, NCHAR, and NCHAR VARYING data types are treated as TEXT, to match the string function arguments and return values. Refer to "String Functions and Operators" under "The SQL Language" in the PostgreSQL Documentation for information on string functions.
  When specifying the arguments of the features compatible with Oracle databases NVL and/or DECODE, use CAST to convert the data types of the host variables to ensure that data types between arguments are the same.

  ***2:**
  When using COLUMN_VALUE to obtain an INTERVAL type value specified in the select list, use an INTERVAL type variable with no interval qualifier or with a range that matches that of the variable in the select list. If an interval qualifier variable with a narrow range is specified, then the value within the interval qualifier range will be obtained, but an error that the values outside the range have been truncated will not occur.

Example

This example illustrates where a value expression that returns an INTERVAL value is set in the select list and the result is received with COLUMN_VALUE. Note that the SQL statement operation result returns a value within the INTERVAL DAY TO SECOND range.

[Bad example]

Values of MINUTE, and those after MINUTE, are truncated, because the variable(v_interval) is INTERVAL DAY TO HOUR.

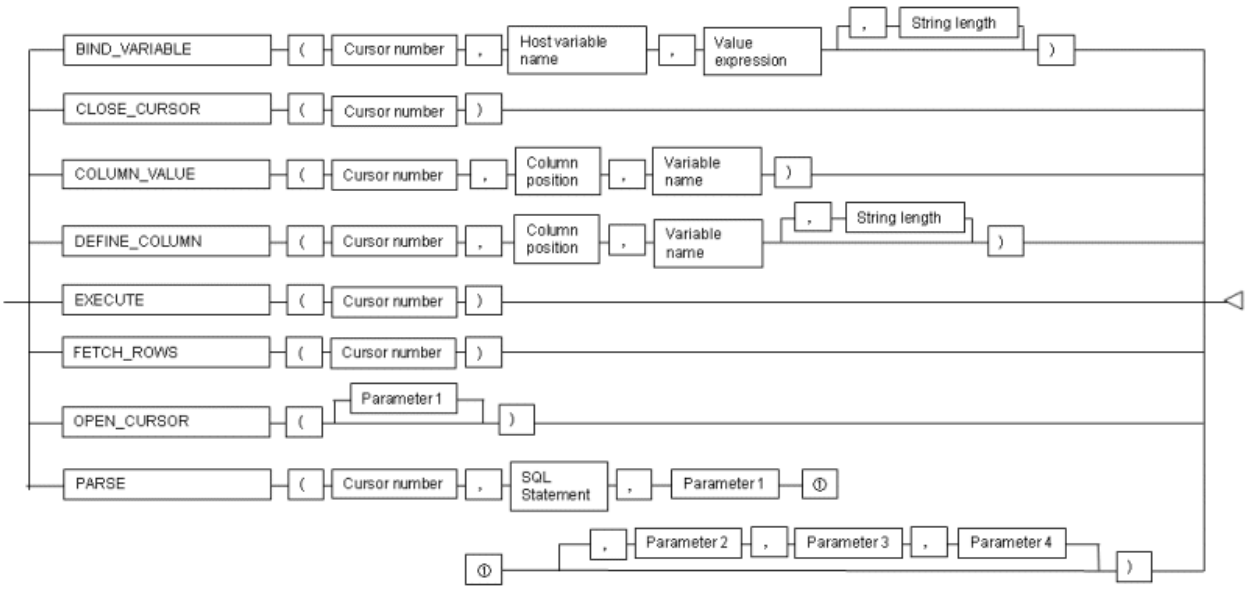
```
    v_interval     INTERVAL DAY TO HOUR;
...
    PERFORM DBMS_SQL.PARSE(cursor, 'SELECT CURRENT_TIMESTAMP - ''2010-01-01'' FROM
DUAL', 1);
...
    SELECT value INTO v_interval FROM DBMS_SQL.COLUMN_VALUE(cursor, 1, v_interval);
    result:1324 days 09:00:00
```

[Good example]

By ensuring that the variable(v_interval) is INTERVAL, the values are received correctly.

```
    v_interval     INTERVAL;
...
    PERFORM DBMS_SQL.PARSE(cursor, 'SELECT CURRENT_TIMESTAMP - ''2010-01-01'' FROM
DUAL', 1);
...
    SELECT value INTO v_interval FROM DBMS_SQL.COLUMN_VALUE(cursor, 1, v_interval);
    result:1324 days 09:04:37.530623
```

Specification format



9.5.3.1 Description

This section explains the procedures available in the DBMS_SQL package.

BIND_VARIABLE

- BIND_VARIABLE sets values in the host variable within the SQL statement.
- The cursor number is the cursor number to be processed.
- Specify the name of the host variable within the SQL statement using a string for the host variable name.

- The value expression is the value set in the host variable. The data type of the host variable is the same as that of the value expression - it is implicitly converted in accordance with its position within the SQL statement. Refer to "A.3 Implicit Data Type Conversions" for information on implicit conversions.
- If the value expression is a character string type, the string length is the number of characters. If the string length is not specified, the size is the total length of the string.
- It is necessary to place a colon at the beginning of the host variable in SQL statements to identify the host variable. The colon does not have to be added to the host variable names specified at BIND_VARIABLE. The following shows examples of host variable names specified with SQL statements and host variable names specified with BIND_VARIABLE:

```
PERFORM DBMS_SQL.PARSE(cursor, 'SELECT emp_name FROM emp WHERE sal > :x', 1);
```

In this example, BIND_VARIABLE will be as follows:

```
PERFORM DBMS_SQL.BIND_VARIABLE(cursor, ':x', 3500);
```

Or,

```
PERFORM DBMS_SQL.BIND_VARIABLE(cursor, 'x', 3500);
```

- The length of the host variable name can be up to 30 bytes (excluding colons).
- If the data type of the set value is string, specify the effective size of the column value as the fourth argument.

Example

If the data type of the value to be set is not a string:

```
PERFORM DBMS_SQL.BIND_VARIABLE(cursor, ':NO', 1);
```

If the data type of the value to be set is a string:

```
PERFORM DBMS_SQL.BIND_VARIABLE(cursor, ':NAME', h_memid, 5);
```

CLOSE_CURSOR

- CLOSE_CURSOR closes the cursor.
- The cursor number is the cursor number to be processed.
- The value returned is a NULL value.

Example

```
cursor := DBMS_SQL.CLOSE_CURSOR(cursor);
```

COLUMN_VALUE

- COLUMN_VALUE acquires the value of the column in the select list extracted with FETCH_ROWS.
- The cursor number is the cursor number to be processed.
- The column position is the position of the column of the select list in the SELECT statement. The position of the first column is 1.
- Specify the variable name for the storage location as the variable name.

- The information extracted is acquired as the values of the value, column_error, and actual_length columns with the SELECT statement.
- value returns the value of the column specified at column position. The data type of the variable name must match the data type of the column. If the data type of the column in the SELECT statement specified in PARSE is not compatible with DBMS_SQL, use CAST to convert to a compatible data type.
- column_error is a NUMERIC type. If the column value could not be set correctly in "value", a value other than 0 will be returned:
  22001: The extracted string has been truncated
  22002: The extracted value contains a NULL value
- actual_length is an INTEGER type. If the extracted value is a character string type, the number of characters will be returned (if the value was truncated, the number of characters prior to the truncation will be returned), otherwise, the number of bytes will be returned.

Example

When getting the value of the column, the error code, and the actual length of the column value:

```
SELECT value, column_error, actual_length INTO v_memid, v_col_err, v_act_len FROM
DBMS_SQL.COLUMN_VALUE(cursor, 1, v_memid);
```

When just getting the value of the column:

```
SELECT value INTO v_memid FROM DBMS_SQL.COLUMN_VALUE(cursor, 1, v_memid);
```

DEFINE_COLUMN

- DEFINE_COLUMN defines the column from which values are extracted and the storage destination.
- The cursor number is the cursor number to be processed.
- Specify the relative position of the select list for the column position. The position of the first column is 1.
- The variable name defines the storage location. The data type in the storage destination should be match with the data type of the column from which the value is to be extracted. If the data type of the column in the SELECT statement specified in PARSE is not compatible with DBMS_SQL, use CAST to convert to a compatible data type.
- String length is the maximum string numbers of character string type column values.
- If the data type of the column value is string, specify the effective size of the column value as the fourth argument.

Example

When the data type of the column value is not a string:

```
PERFORM DBMS_SQL.DEFINE_COLUMN(cursor, 1, v_memid);
```

When the data type of the column value is a string:

```
PERFORM DBMS_SQL.DEFINE_COLUMN(cursor, 1, v_memid, 10);
```

EXECUTE

- EXECUTE executes SQL statements.
- The cursor number is the cursor number to be processed.

- The return value is an INTEGER type, is valid only with INSERT statement, UPDATE statement, and DELETE statement, and is the number of lines processed. Anything else is invalid.

Example

```
ret := DBMS_SQL.EXECUTE(cursor);
```

FETCH_ROWS

- FETCH_ROWS positions at the next line and extracts values from the line.
- The cursor number is the cursor number to be processed.
- The return value is an INTEGER type and is the number of lines extracted. 0 is returned if all are extracted.
- The extracted information is acquired with COLUMN_VALUE.

Example

```
LOOP
    IF DBMS_SQL.FETCH_ROWS(cursor) = 0 THEN
        EXIT;
    END IF;

...

END LOOP;
```

OPEN_CURSOR

- OPEN_CURSOR opens a new cursor.
- Parameter 1 is a parameter used for compatibility with Oracle databases only, and are ignored by Enterprise Postgres. An INTEGER type can be specified, but it will be ignored. Although specifying a value here has no meanings, if you are specifying a value anyway, specify 1. If migrating from an Oracle database, the specified value does not need to be changed.
- Close unnecessary cursors by executing CLOSE_CURSOR.
- The return value is an INTEGER type and is the cursor number.

Example

```
cursor := DBMS_SQL.OPEN_CURSOR();
```

PARSE

- PARSE analyzes dynamic SQL statements.
- The cursor number is the cursor number to be processed.
- The SQL statement is the SQL statement to be parsed.
- Parameters 1, 2, 3, and 4 are parameters used for compatibility with Oracle databases only, and are ignored by Enterprise Postgres. Although specifying a value here has no meanings, if you are specifying a value anyway, specify the following:

- Parameter 1 is an INTEGER type. Specify 1.
- Parameters 2 and 3 are TEXT types. If specifying, specify NULL.
- Parameter 4 is a BOOLEAN type. If specifying, specify TRUE.

If migrating from an Oracle database, the specified values for parameters 2, 3, and 4 do not need to be changed.

- Add a colon to the beginning of host variables in SQL statements.
- The DDL statement is executed when PARSE is issued. EXECUTE is not required for the DDL statement.
- If PARSE is called again for opened cursors, the content in the data regions within the cursors are reset, the SQL statement is parsed anew.

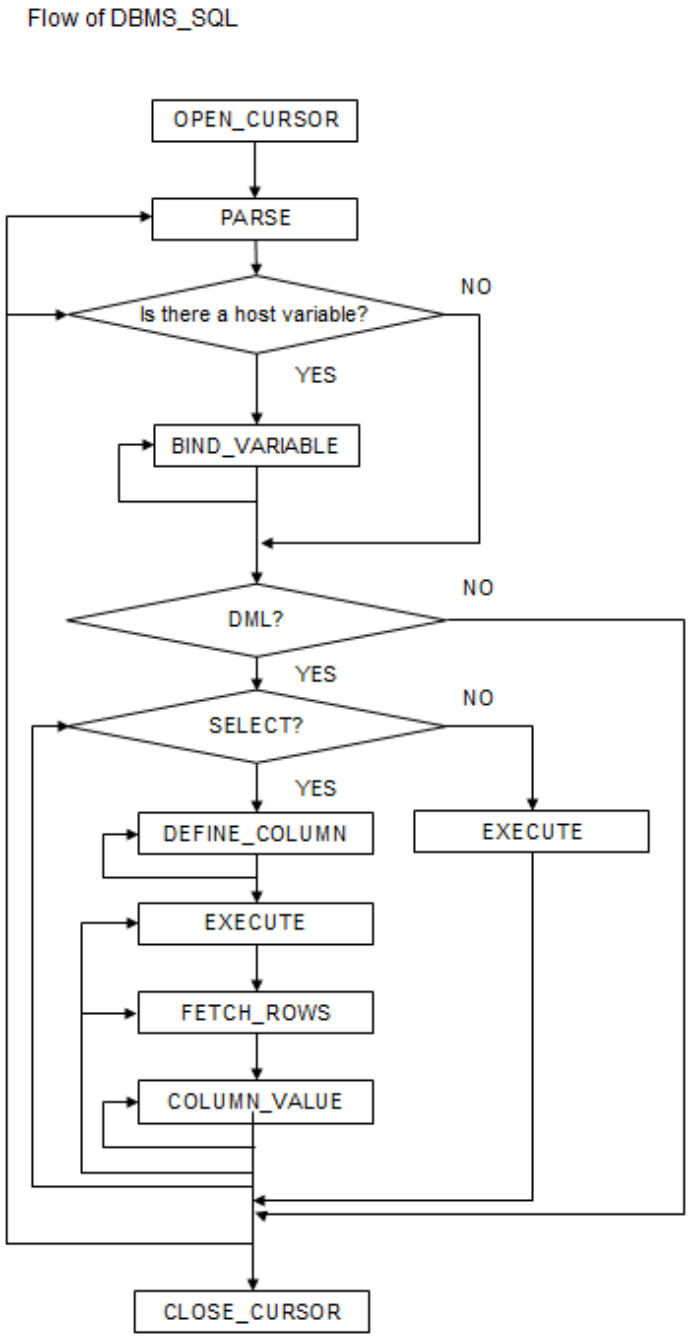
Example

```
PERFORM DBMS_SQL.PARSE(cursor, 'SELECT memid, memnm FROM member WHERE memid = :NO', 1);
```

9.5.3.2 Example

This section explains the flow of DBMS_SQL and the example.

Flow of DBMS_SQL



Example

```
CREATE FUNCTION smp_00()
RETURNS INTEGER
AS $$
DECLARE
    str_sql     VARCHAR(255);
    cursor      INTEGER;
    h_smpid     INTEGER;
    v_smpid     INTEGER;
    v_smpnm     VARCHAR(20);
    v_smpage    INTEGER;
    errcd       INTEGER;
```

```
    length      INTEGER;
    ret         INTEGER;
BEGIN
    str_sql     := 'SELECT smpid, smpnm, smpage FROM smp_tbl WHERE smpid < :H_SMPID ORDER
BY smpid';
    h_smpid     := 3;
    v_smpid     := 0;
    v_smpnm     := '';
    v_smpage    := 0;

    cursor := DBMS_SQL.OPEN_CURSOR();

    PERFORM DBMS_SQL.PARSE(cursor, str_sql, 1);

    PERFORM DBMS_SQL.BIND_VARIABLE(cursor, ':H_SMPID', h_smpid);

    PERFORM DBMS_SQL.DEFINE_COLUMN(cursor, 1, v_smpid);
    PERFORM DBMS_SQL.DEFINE_COLUMN(cursor, 2, v_smpnm, 10);
    PERFORM DBMS_SQL.DEFINE_COLUMN(cursor, 3, v_smpage);

    ret := DBMS_SQL.EXECUTE(cursor);
    loop
        if DBMS_SQL.FETCH_ROWS(cursor) = 0 then
            EXIT;
        end if;

        SELECT value,column_error,actual_length INTO v_smpid,errcd,length FROM
DBMS_SQL.COLUMN_VALUE(cursor, 1, v_smpid);
        RAISE NOTICE '------------------------------------------------------';
        RAISE NOTICE '------------------------------------------------------';
        RAISE NOTICE 'smpid       = %', v_smpid;
        RAISE NOTICE 'errcd       = %', errcd;
        RAISE NOTICE 'length      = %', length;

        SELECT value,column_error,actual_length INTO v_smpnm,errcd,length FROM
DBMS_SQL.COLUMN_VALUE(cursor, 2, v_smpnm);
        RAISE NOTICE '------------------------------------------------------';
        RAISE NOTICE 'smpnm       = %', v_smpnm;
        RAISE NOTICE 'errcd       = %', errcd;
        RAISE NOTICE 'length      = %', length;

        select value,column_error,actual_length INTO v_smpage,errcd,length FROM
DBMS_SQL.COLUMN_VALUE(cursor, 3, v_smpage);
        RAISE NOTICE '------------------------------------------------------';
        RAISE NOTICE 'smpage      = %', v_smpage;
        RAISE NOTICE 'errcd       = %', errcd;
        RAISE NOTICE 'length      = %', length;
        RAISE NOTICE '';
    end loop;

    cursor := DBMS_SQL.CLOSE_CURSOR(cursor);
    RETURN 0;
END;
$$ LANGUAGE plpgsql;
```

Chapter 10 Java Applications using PL/extJava

This chapter explains how to develop Java applications using PL/extJava.

See

Refer to "Setting up and Operating PL/extJava" in the Operation Guide for details.

Point

In this chapter, PL/extJava stored functions that are registered in the database are referred to as "Java functions". Applications that operate on the Java VM are referred to as "Java applications".

10.1 Development Procedure

Perform the following procedure to develop the Java application that runs in the PL/extJava environment:

1. Develop the Java application

   Use the same approach as the one generally used to develop Java applications.
   There is no need to implement a special interface in order to use the application in PL/extJava.
   The connection from the Java application to the database uses the JDBC connection pool. Refer to "10.3 Connecting to the Database" for details.

2. Compile the Java application

   Compile the Java application that was created, and package (jar) the class files.

3. Deploy the jar file to the PL/extJava environment

   Refer to "Registering Java Functions" in the Operation Guide for details.

10.2 Points to Consider when Developing Java Applications

This section contains points that should be considered regarding Java applications. These are as follows:

- Use JDK7 to develop the Java function.
- Implement methods that are to be called as Java functions as static.
- Do not use variable arguments as method arguments.
- Only a single row can be returned as the return value.
- In Java applications, the auto commit mode for transactions is 'off' by default. You can change it using the setAutoCommit method, as necessary.
- Standard output and standard error output
  If a Java application outputs to the standard output or to the standard error output, then it will be redirected to the server log of the container.
- File output
  In Java applications, the directories below are used as the current directory.
  If files are output using an absolute path, ensure that the output directory is secure.

  L - **Linux**

```
domainRoot(*1)/nodes/localhost-domain1/[serverInstanceName]/current
```

Java application are started by an instance administrator user, which must have file access privileges.

- **Windows(R)**

```
domainRoot(*1)\nodes\localhost-domain1\serverInstanceName\current
```

*1: *domainRoot* is the directory that stores the domain resources (refer to "PL/extJava Configuration" in the Operation Guide for details).

- The following system administrator functions will wait until the execution of the Java application completes:
  - pg_terminate_backend
  - pg_cancel_backend
- Java applications run as transactions that are not dependent on the caller. For this reason, if the caller transaction is rolled back, the Java application transaction will not be rolled back.
  Additionally, if an error occurs in the Java application, it will be output to the server log of the container (refer to "PL/extJava Log Information" in the Operation Guide for details).

10.3 Connecting to the Database

The Java application uses the JDBC connection pool to connect to the database.

However, when using an integrated development environment such as Interstage Studio for testing during the Java application development stage, the JDBC driver is called directly in order to connect to the database.

Select the connection methods shown below according to the phase.

Development/testing phase

Call the JDBC directly to connect to the database.

- Set CLASSPATH.

  The CLASSPATH setting will be required to check the behavior by calling the JDBC driver directly.

  The name of the JDBC driver file that can be used in the Java function is as follows:

```
postgresql-jdbc41.jar
```

  Refer to "2.2.1 Environment Settings" for information on how to set CLASSPATH.

- The database server URL is called directly.

  Refer to "2.3 Connecting to the Database" for information on how to use the JDBC driver.

Operation phase

The JDBC connection pool of PL/extJava is used to connect to the database.

Use DriverManager.getConnection() to obtain the connection.
"jdbc:default:connection" must be specified as the DriverManager.getConnection() argument.

```
java.sql.Connection conn = DriverManager.getConnection("jdbc:default:connection");
```

10.4 Data type

This section explains the data type.

10.4.1 Relationship between the Java Function Data Types and Java Application Data Types

This section explains the relationship between the Java function data type and the Java application data type.

Use the mapping shown in the table below to implement the relationship between the Java function to be registered using CREATE FUNCTION and the arguments and return values of the Java application. If the method names match but the Java application data types do not match, the Java function cannot be executed and returns an error.

Table 10.1 Java function and Java application data type conversion rules

Java function data type (CREATE FUNCTION argument, return value type)	Java application data type (method argument, return value type)
void	void
character	java.lang.String
national character	java.lang.String
character varying	java.lang.String
national character varying	java.lang.String
text	java.lang.String
bytea	byte[]
smallint	short
integer	int
bigint	long
smallserial	short
serial	int
bigserial	long
real	float
double precision	double
date	java.sql.Date
time with time zone	java.sql.Time
time without time zone	java.sql.Time
timestamp without time zone	java.sql.Timestamp
timestamp with time zone	java.sql.Timestamp
boolean	boolean

Note

In the Java function, the following data types cannot be used:

- numeric
- decimal
- money
- interval
- bit
- bit varying

- xml
- array
- uuid
- point
- box
- lseg
- path
- polygon
- circle
- json
- Network address type (inet, cidr, macaddr)
- Types related to text searches (tsvector, tsquery)
- Enumerated type
- Composite type
- Range type
- Object identifier data type
- Pseudo-data types (except void)

10.4.2 Relationship between the Java Application Data Types and Database Data Types

When developing a Java application that uses JDBC driver, refer to "2.4.1 Relationship between the Application Data Types and Database Data Types" for information on this relationship.

Chapter 11 Application Connection Switch Feature

The application connection switch feature enables automatic connection to the target server when there are multiple servers with redundant configurations.

When using this feature, specify the primary server and secondary server as the connected servers in the application connection information.

If an application connection switch occurs, explicitly close the connection and then reestablish the connection or reexecute the application. Refer to "Errors when an Application Connection Switch Occurs and Corresponding Actions" of the relevant client interface for information on how to confirm the switch.

11.1 Connection Information for the Application Connection Switch Feature

To use the application connection switch feature, set the information shown below when connecting the database.

IP address or host name

Specify the IP address or host name that will be used to configure the database multiplexing system.

Port number

A port number used by each database server to listen for connections from applications.
In each client interface, multiple port numbers can be specified, however in the format shown below, for example, if only one port number is specified, it will be assumed that host1: 27500 (the default value) and host2:port2 were specified.
host1,host2:port2
Omit all port numbers, or specify only one per server.

Target server

From the specified connection destination server information, specify the selection sequence of the servers to which the application will connect. The values specified for the target server have the meanings shown below. If a value is omitted, primary will be used.

primary

The primary server is selected as the connection target from the specified "IP addresses or host names". Specify this to perform tasks that can be performed only on the primary server, such as applications in line with updates, or management tasks such as REINDEX and VACUUM.

prefer_standby

The standby server is selected preferentially as the connection target from the specified "IP addresses or host names". If there is no standby server, the application will connect to the primary server.

This value can be specified when the Mirroring Controller option is used.

any (this value can be used only when the JDBC driver is used)

This method is not recommended in database multiplexing systems. This is because, although the connection destination server is selected in the specified sequence from the specified "IP addresses or host names", if the server that was successfully connected to first is the standby server, the update will always fail.

This method is not published in the manuals for the PostgreSQL JDBC driver, but has a feature that performs the same type of connection, so this is provided for applications that depend strongly on that feature.

SSL server certificate Common Name (CN)

To perform SSL authentication by creating the same server certificate for each server in a multiplexing system, specify the SSL server certificate Common Name (CN) in this parameter. Accordingly, SSL authentication using the CN can be performed without having to consider the names of the multiple servers contained in the multiplexing system.

11.2 Using the Application Connection Switch Feature

This section explains how to set the connection destination server using the application connection switch feature.

Of the parameters used as connection information for each client interface, only the parameters specific to the application connection switch feature are explained here. Refer to "Setup" and "Connecting to the Database" for information on the other parameters of each client interface.

11.2.1 Using the JDBC Driver

Set the following information in the connection string of the DriverManager class, or in the data source.

Table 11.1 Information to be set

Argument	Explanation
host1 host2	Specify the IP address or host name.
port1 port2	Specify the port number for the connection. The port number can be omitted. If omitted, the default is 27500.
database_name	Specify the database name.
targetServer	Specify the selection sequence of the servers to which the application will connect. Refer to "Target server" for details.
sslmode	Specify this to encrypt communications. By default, this is disabled. The setting values for sslmode are as follows: disable: Connect without SSL require: Connect always with SSL verify-ca: Connect with SSL, using a certificate issued by a trusted CA (*1) verify-full: Connect with SSL, using a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
sslservercertcn	This parameter is enabled only to perform SSL authentication (sslmode=verify-full). Specify the server certificate CN. If this is omitted, the value will be null, and the server certificate CN will be authenticated using the host name specified in host.

*1: If specifying either "verify-ca" or "verify-full", the CA certificate file can be specified using connection string sslrootcert.

When using Driver Manager

Specify the following URL in the API of the DriverManager class:

```
jdbc:postgresql://host1[:port1],host2[:port2]/database_name[?targetServer={primary |
prefer_standby | any}][&sslmode=verify-
full&sslrootcert=cACertificateFile&sslservercertcn=targetServerCertificateCN]
```

- If the target server is omitted, the default value "primary" is used.
- When using IPV6, specify the host in the "[*host*]" (with square brackets) format.

  [Example]

```
jdbc:postgresql://[2001:Db8::1234]:27500,192.168.1.1:27500/database_name
```

When using the data source

Specify the properties of the data source in the following format:

```
source.setServerName("host1[:port1],host2[:port2]");
source.setTargetServer("primary");
source.setSslmode("verify-full");
source.setSslrootcert("cACertificateFile");
source.setSslservercertcn("targetServerCertificateCN");
```

- If the port number is omitted, the value specified in the portNumber property will be used. Also, if the portNumber property is omitted, the default is 27500.
- If the target server is omitted, the value will be primary.
- When using IPV6, specify the host in the "[*host*]" (with square brackets) format.

  [Example]

```
source.setServerName("[2001:Db8::1234]:27500,192.168.1.1:27500");
```

Note

If using the connection parameter loginTimeout, the value will be applied for the time taken attempting to connect to all of the specified hosts.

11.2.2 Using the ODBC Driver

Set the following information in the connection string or data source.

Table 11.2 Information to be set

Parameter	Explanation
Servername	Specify IP address 1 and IP address 2, or the host name, using a comma as the delimiter. Based on ODBC rules, it is recommended to enclose the whole string containing comma delimiters with {}. Format: {host1,host2}
Port	Specify the connection destination port numbers, using a comma as the delimiter. Based on ODBC rules, it is recommended to enclose the whole string containing comma delimiters with {}. Format: {port1,port2} Specify the port number corresponding to the IP address or host specified for the nth Servername as the nth Port. The port number can be omitted. If omitted, the default is 27500. If n server names are specified, the (n+1)th and later ports will be ignored.
TargetServer	Specify the selection sequence of the servers to which the application will connect. Refer to "Target server" for details.
SSLMode	Specify this to encrypt communications. By default, this is disabled. The setting values for SSLMode are as follows: disable: Connect without SSL allow: Connect without SSL, and if it fails, connect with SSL prefer: Connect with SSL, and if it fails, connect without SSL require: Connect always with SSL verify-ca: Connect with SSL, using a certificate issued by a trusted CA (*1)

Parameter	Explanation
	verify-full: Connect with SSL, using a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
SSLServerCertCN	This parameter is enabled only to perform SSL authentication (SSLMode=verify-full). Specify the server certificate CN. If this is omitted, the value will be null, and the server certificate CN will be authenticated using the host name specified in Servername.

*1: If specifying either "verify-ca" or "verify-full", use the system environment variable PGSSLROOTCERT of your operating system to specify the CA certificate file as shown below.

Example)
Variable name: PGSSLROOTCERT
Variable value: *cACertificateFile*

When specifying a connection string

Specify the following connection string:

```
...;Servername={host1,host2};Port={port1,port2};[TargetServer={primary |
prefer_standby }];[ SSLMode=verify-full;SSLServerCertCN=targetServerCertificateCN]...
```

- When using IPV6, specify the host in the "*host*" format.

  [Example]

```
Servername={2001:Db8::1234,192.168.1.1};Port={27500,27500};
```

When using the data source

Specify the properties of the data source in the following format:

```
Servername={host1,host2}
Port={port1,port2}
TargetServer={primary | prefer_standby }
SSLMode=verify-full
SSLServerCertCN=targetServerCertificateCN
```

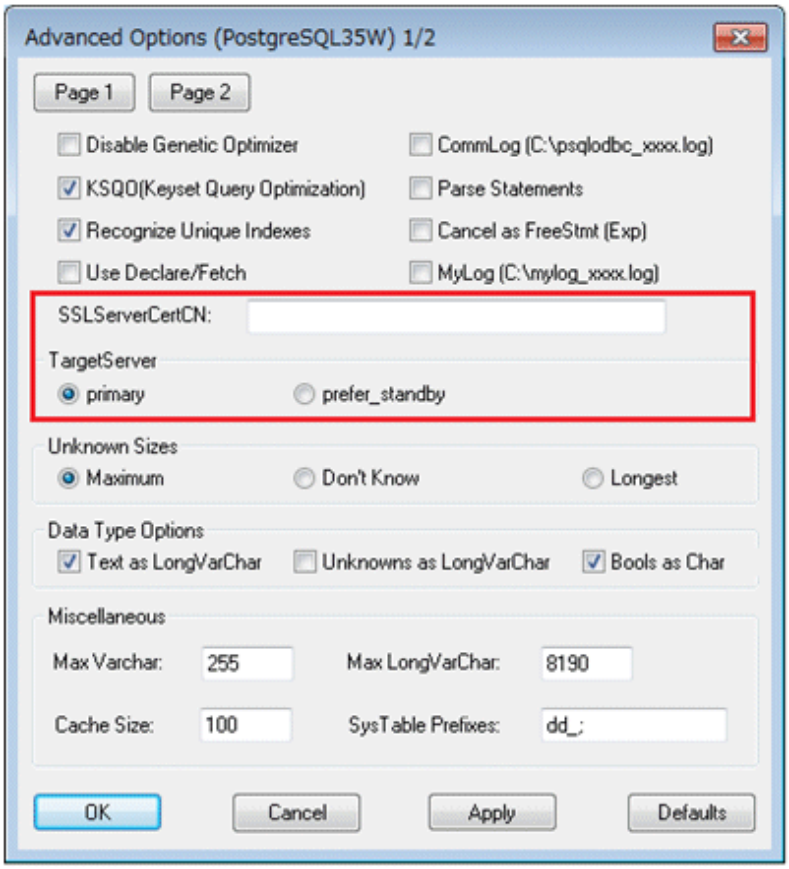
- When using IPV6, specify the host in the "*host*" format.

  [Example]

```
Servername={2001:Db8::1234,192.168.1.1}
```

Registering the data source using the ODBC Data Source Administrator

Using the ODBC Data Source Administrator, specify the items within the red border below:



Note

If using the connection parameter login_timeout, this value is applied for connections to each of the specified hosts. If both multiplexed database servers have failed, the connection will time out when a time equal to double the login_timeout value elapses.

11.2.3 Using a .NET Data Provider

Set the following information in the connection string of NpgsqlConnection, or in the data source.

Table 11.3 Information to be set

Argument	Explanation
host1 host2	Specify the IP address or host name.
port1 port2	Specify the port number for the connection.
targetServer	Specify the selection sequence of the servers to which the application will connect. Refer to "Target server" for details.

When specifying a connection string

Specify the following connection string:

```
host1[:port1],host2[:port2];[targetServer={primary | prefer_standby }];
```

- If the port number is omitted, the value specified for the Port keyword of the connection string will be used. Refer to "4.3.4 Connection String" for information on the Port keyword.
- If the target server is omitted, the value will be primary.
- When using IPV6, specify the host in the "[*host*]" (with square brackets) format.

  [Example]

```
host=[2001:Db8::1234]:27500,192.168.1.1:27500;
```

When specifying the NpgsqlConnectionStringBuilder property, or adding a connection in TableAdapter

Specify the Host property of the data source in the following format:

```
host1[:port1],host2[:port2]
```

- If the port number is omitted, the value specified in the Port property will be used. Also, if the Port property is omitted, the default is 27500.

Specify the TargetServer property of the data source in the following format:

```
TargetServer.primary | TargetServer.prefer_standby
```

- If the target server is omitted, the value will be primary.

Note

If using the connection parameter Timeout, this value is applied for connections to each of the specified hosts. If both multiplexed database servers have failed, the connection will time out when a time equal to double the Timeout value elapses.

11.2.4 Using a Connection Service File

Set the connection parameters as follows.

Table 11.4 Information to be set

Parameter	Explanation
host	Specify the host names, using a comma as the delimiter.
hostaddr	Specify IP address 1 and IP address 2, using a comma as the delimiter.
port	Specify the connection destination port numbers, using a comma as the delimiter. Specify the port number for the server specified for the nth host or hostaddr as the nth port. The port number can be omitted. If omitted, the default is 27500. If n server names are specified, the (n+1)th and later ports will be ignored.
target_server	Specify the selection sequence of the servers to which the application will connect. Refer to "Target server" for details.
sslmode	Specify this to encrypt communications. By default, this is disabled. The setting values for sslmode are as follows: disable: Connect without SSL allow: Connect without SSL, and if it fails, connect with SSL prefer: Connect with SSL, and if it fails, connect without SSL

Parameter	Explanation
	require: Connect always with SSL verify-ca: Connect with SSL, using a certificate issued by a trusted CA (*1) verify-full: Connect with SSL, using a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
sslservercertcn	This parameter is enabled only to perform SSL authentication (sslmode=verify-full). Specify the server certificate CN. If this is omitted, the value will be null, and the server certificate CN will be authenticated using the host name specified in host.

*1: If specifying either "verify-ca" or "verify-full", use the system environment variable PGSSLROOTCERT (connection parameter sslrootcert) of your operating system to specify the CA certificate file as shown below.

Example)
Variable name: PGSSLROOTCERT
Variable value: *cACertificateFile*

Note

If using the connection parameter connect_timeout, this value is applied for connections to each of the specified hosts. If both multiplexed database servers have failed, the connection will time out when a time equal to double the connect_timeout value elapses.

Point

If using the C Library, embedded SQL or psql commands (including other client commands that specify connection destinations), it is recommended to use a connection service file to specify connection destinations.

In the connection service file, a name (service name) is defined as a set, comprising information such as connection destination information and various types of tuning information set for connections. By using the service name defined in the connection service file when connecting to databases, it is no longer necessary to modify applications when the connection information changes.

11.2.5 Using the C Library (libpq)

It is recommended that you use a connection service file. Refer to "11.2.4 Using a Connection Service File" for details.

If a connection service file will not be used, set the following information for the database connection control functions (PQconnectdbParams, PQconnectdb, and so on) or environment variables.

Table 11.5 Information to be set

Parameter (environment variable name)	Explanation
host(PGHOST)	Specify the host names, using a comma as the delimiter.
hostaddr(PGHOSTADDR)	Specify IP address 1 and IP address 2, using a comma as the delimiter.
port(PGPORT)	Specify the connection destination port numbers, using a comma as the delimiter. Specify the port number for the server specified for the nth host or hostaddr as the nth port. The port number can be omitted. If omitted, the default is 27500. If n server names are specified, the (n+1)th and later ports will be ignored.

Parameter (environment variable name)	Explanation
target_server(PGXTARGETSERVER)	Specify the selection sequence of the servers to which the application will connect. Refer to "Target server" for details.
sslmode(PGSSLMODE)	Specify this to encrypt communications. By default, this is disabled. The setting values for sslmode are as follows: disable: Connect without SSL allow: Connect without SSL, and if it fails, connect with SSL prefer: Connect with SSL, and if it fails, connect without SSL require: Connect always with SSL verify-ca: Connect with SSL, using a certificate issued by a trusted CA (*1) verify-full: Connect with SSL, using a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
sslservercertcn(PGXSSLSERVERCERTCN)	This parameter is enabled only to perform SSL authentication (sslmode=verify-full). Specify the server certificate CN. If this is omitted, the value will be null, and the server certificate CN will be authenticated using the host name specified in host.

*1: If specifying either "verify-ca" or "verify-full", use the system environment variable PGSSLROOTCERT (connection parameter sslrootcert) of your operating system to specify the CA certificate file as shown below.

Example)
Variable name: PGSSLROOTCERT
Variable value: *cACertificateFile*

When using URI

```
postgresql://host1[:port1],host2[:port2][,...]/database_name
[?target_server={primary | prefer_standby}]
```

- When using IPV6, specify the host in the "[*host*]" (with square brackets) format.

  [Example]

```
postgresql://postgres@[2001:Db8::1234]:27500,192.168.1.1:27500/database_name
```

When using key-value

```
host=host1[,host2] port=port1[,port2] user=user1 password=pwd1 dbname=mydb
[target_server={primary | prefer_standby}]
```

- When using IPV6, specify the host in the "*host*" format.

  [Example]

```
host=2001:Db8::1234,192.168.1.1 port=27500,27500
```

Note

If using the connection parameter connect_timeout, this value is applied for connections to each of the specified hosts. If both multiplexed database servers have failed, the connection will time out when a time equal to double the connect_timeout value elapses.

Information

If using a password file (.pgpass), describe the entries matching each server.

- Example 1:

```
host1:port1:dbname:user:password
host2:port2:dbname:user:password
```

- Example 2:

```
*:port:dbname:user:password
```

11.2.6 Using Embedded SQL

It is recommended that you use a connection service file. Refer to "11.2.4 Using a Connection Service File" for details.

Point

If using a connection service file, either of the following methods is available:

- Set the service name as a string literal or host variable, as follows:

  tcp:postgresql://?service=my_service

- Set the service name in the environment variable PGSERVICE, and use CONNECT TO DEFAULT

If a connection service file will not be used, use a literal or variable to specify the connection destination server information for target in the SQL statement below:

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name];
```

Method used

```
dbname@host1,host2[:[port1][,port2]]
tcp:postgresql://host1,host2[:[port1][,port2]] [/dbname] [?target_server={primary |
prefer_standby}][&sslmode=verify-full&sslservercertcn=targetServerCertificateCN]
```

- The above format cannot be specified directly without using a literal or variable.

Table 11.6 Information to be set

Argument	Explanation
host1 host2	Specify the IP address or host name. IPv6 format addresses cannot be specified.
port1 port2	Specify the connection destination port numbers, using a comma as the delimiter. The port number can be omitted. If omitted, the default is 27500.
dbname	Specify the database name.

Argument	Explanation
target_server	Specify the selection sequence of the servers to which the application will connect. Refer to "Target server" for details.
sslmode	Specify this to encrypt communications. By default, this is disabled. The setting values for sslmode are as follows: disable: Connect without SSL allow: Connect without SSL, and if it fails, connect with SSL prefer: Connect with SSL, and if it fails, connect without SSL require: Connect always with SSL verify-ca: Connect with SSL, using a certificate issued by a trusted CA (*1) verify-full: Connect with SSL, using a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
sslservercertcn	This parameter is enabled only to perform SSL authentication (sslmode=verify-full). Specify the server certificate CN. If this is omitted, the value will be null, and the server certificate CN will be authenticated using the host name specified in host.

*1: If specifying either "verify-ca" or "verify-full", use the system environment variable PGSSLROOTCERT (connection parameter sslrootcert) of your operating system to specify the CA certificate file as shown below.

Example)
Variable name: PGSSLROOTCERT
Variable value: *cACertificateFile*

Point

Environment variables can also be used. Refer to "11.2.5 Using the C Library (libpq)" for information on environment variables.

Note

If using the connection parameter connect_timeout, this value is applied for connections to each of the specified hosts. If both multiplexed database servers have failed, the connection will time out when a time equal to double the connect_timeout value elapses.

11.2.7 Using the psql Command

It is recommended that you use a connection service file. Refer to "11.2.4 Using a Connection Service File" for details.

If a connection service file will not be used, specify the following information in the psql command option/environment variable.

Table 11.7 Information to be set

Option (environment variable)	Explanation
-h/--host(PGHOST/ PGHOSTADDR)	Specify IP address 1 and IP address 2, or the host name, using a comma as the delimiter. This can also be specified for the environment variable PGHOST or PGHOSTADDR.

Option (environment variable)	Explanation
-p/--port(PGPORT)	Specify the connection destination port numbers, using a comma as the delimiter. This can also be specified for the environment variable PGPORT. Specify the port number corresponding to the IP address specified for the nth -h option as the nth -p option. The port number can be omitted. If omitted, the default is 27500. If n -h options are specified, the (n+1)th and later -p options will be ignored.
(PGXTARGETSERVER)	Specify the selection sequence of the servers to which the application will connect. Refer to "Target server" for details.
(PGSSLMODE)	Specify this to encrypt communications. By default, this is disabled. The setting values for PGSSLMODE are as follows: disable: Connect without SSL allow: Connect without SSL, and if it fails, connect with SSL prefer: Connect with SSL, and if it fails, connect without SSL require: Connect always with SSL verify-ca: Connect with SSL, using a certificate issued by a trusted CA (*1) verify-full: Connect with SSL, using a certificate issued by a trusted CA to verify if the server host name matches the certificate (*1)
(PGXSSLSERVERCERTCN)	This environment variable is enabled only to perform SSL authentication (PGSSLMODE=verify-full). Specify the server certificate CN. If this is omitted, the value will be null, and the server certificate CN will be authenticated using the host name specified in host.

*1: If specifying either "verify-ca" or "verify-full", use the system environment variable PGSSLROOTCERT (connection parameter sslrootcert) of your operating system to specify the CA certificate file as shown below.

Example)
Variable name: PGSSLROOTCERT
Variable value: *cACertificateFile*

Note

If using the connection parameter connect_timeout, this value is applied for connections to each of the specified hosts. If both multiplexed database servers have failed, the connection will time out when a time equal to double the connect_timeout value elapses.

Information

Use the same method as for psql commands to specify connection destination server information for other client commands used to specify connection destinations.

Chapter 12 Performance Tuning

This chapter explains how to tune application performance.

12.1 Enhanced Query Plan Stability

By stabilizing the SQL statement query plan so that it does not change, deterioration of the application performance is suppressed.

12.1.1 Optimizer Hints

This section explains the basic feature content of the optimizer hint (pg_hint_plan).

Refer to the open-source software webpage for information on pg_hint_plan.

In Enterprise Postgres, the optimizer hints can be specified in all application interfaces.

Description

You can specify a query plan in each SQL statement.

List of Features

The main query plans that can be specified using this feature are as follows:

- Query methods
- Join methods
- Join sequences

Query methods

Specify which method to use to query the specified table.

The main features are as follows:

- SeqScan (*tableName*)
- BitMapScan (*tableName* [*indexName* ...])
- IndexScan (*tableName* [*indexName* ...])
- IndexOnlyScan (*tableName* [*indexName* ...])

Note

- If the specified index does not exist, or is not related to the search condition column specified in the WHERE clause, for example, SeqScan will be used.
- Even if IndexOnlyScan is specified, IndexScan may be used if it is necessary to access the table because a row was updated, for example.
- If multiple query methods were specified for the same table, the method specified last will be used.

Join methods

Specify the join method.

The main features are as follows:

- NestLoop (*tableName tableName* [*tableName* ...])

- MergeJoin (*tableName tableName* [*tableName* ...])
- HashJoin (*tableName tableName* [*tableName* ...])

Note

- These cannot be specified for view tables and subqueries.
- If multiple methods were specified for the same table combination, the method specified last will be used.

Join sequences

The tables will be joined in the specified table sequence.

Specify the information using the following method:

- Leading ((*table table*))

  The method used to specify [*table*] is as follows:

  table = *tableName* or (*table table*)

Note

If multiple sequences were specified for the same table combination, the sequence specified last will be used.

Usage method

The use of this feature is explained below.

Preparation

The following preparation is required to use this feature.

1. Run CREATE EXTENSION for the database that uses this feature.
   The target database is described as "postgres" here.
   Use the psql command to connect to the "postgres" database.

   Example

   ```
   Postgres=# CREATE EXTENSION pg_hint_plan;
   CREATE EXTENSION
   ```

   Information

   Hereafter, also perform this preparatory task for the "template1" database when creating a new database, so that this feature can be used by default.

2. Set the postgresql.conf file parameters.
   Add "pg_hint_plan" to the "shared_preload_libraries" parameter.
3. Restart Enterprise Postgres.

Method used to define this feature

Define this feature by specifying the format (block comment) " /*+ ... */".

- To specify hint clauses in each SELECT statement, for example when there are multiple SELECT statements in the SQL statement, define all hint clauses in the first block comment.

Specifying hint clauses for the emp table and the dept table

```
WITH /*+ IndexScan(emp emp_age_index) IndexScan(dept dept_deptno_index) */ age30
AS (SELECT * FROM emp WHERE age BETWEEN 30 AND 39)
SELECT * FROM age30, dept WHERE age30.deptno = dept.deptno;
```

- To specify separate hint clauses for the same object in the SQL statement, define aliases in each object, and then specify hint clauses for those aliases.

Specifying separate hint clauses for the emp table

```
WITH /*+ SeqScan(ta) IndexScan(tb) */ over100
AS (SELECT empno FROM emp ta WHERE salary > 1000000)
SELECT * FROM emp tb, over100 WHERE tb.empno = over100.empno AND tb.age < 30
```

- When using embedded SQL in C, the locations in which the hint clause block comment is specified are restricted. Refer to "6.4.2 Compiling Applications" for details.

Usage notes

- If a hint clause was specified in multiple block comments in the SQL statement, the hint clause specified in the second block comment and thereafter will be ignored.
- If characters other than those listed below appear before the hint clause in the SQL statement, they will be invalid even for hint clause block comments.
  - Space, tab, line feed
  - Letter (uppercase and lowercase), number
  - Underscore, comma
  - Brackets ()

12.1.2 Locked Statistics

This section explains the basic feature content for locked statistics (pg_dbms_stats).

Refer to the open-source software webpage for information on pg_dbms_stats.

Description

Locks the statistics.

By using this feature to lock the statistics for performance obtained in job load testing in an environment that simulates a production environment, performance degradation caused by changes to the query plan after go-live can be suppressed.

Additionally, by using the export and import features, statistics that were checked in the test environment can also be reproduced in the production environment.

List of Features

The main features that can be specified using this feature are as follows.

[Features]

Feature	Details	Description
Lock/unlock of the statistics	Lock	Locks the statistics.
	Unlock	Unlocks the statistics.
Backup/restore of the statistics	Backup	Backs up the current statistics.
	Restore	Restores the statistics to the point when they were backed up, and then locks them.
	Purge	Deletes backups that are no longer necessary.
Backup/restore using external files	Export	Exports the statistics (binary format).
	Import	Imports the statistics, and then locks them.

[Target object]

Target resource	Range of feature
Database	In the database
Schema	In the schema
Table	In the table
Column	ID column

Usage method

The use of this feature is explained below.

Preparation

The following preparation is required to use this feature.

1. Run CREATE EXTENSION for the database that will use this feature.
   The target database is described as "postgres" here.
   Use the psql command to connect to the "postgres" database.

   Example

```
Postgres=# CREATE EXTENSION pg_dbms_stats;
CREATE EXTENSION
```

   Information

   Hereafter, also perform this preparatory task for the "template1" database, so that this feature can be used by default when creating a new database.

2. Set the postgresql.conf file parameter.
   Add "pg_dbms_stats" to the "shared_preload_libraries" parameter.

3. Restart Enterprise Postgres.

Method used to specify this feature

Specify this feature as an SQL function.

The methods used to specify the main features are shown in the table below.

Feature	Object	Function specified
Lock	Database	dbms_stats.lock_database_stats()
	Schema	dbms_stats.lock_schema_stats('*schemaName*')
	Table	dbms_stats.lock_schema_stats('*schemaName.tableName*')
Unlock	Database	dbms_stats.unlock_database_stats()
	Schema	dbms_stats.unlock_schema_stats('*schemaName*')
	Table	dbms_stats.unlock_schema_stats('*schemaName.tableName*')
Import	Database	dbms_stats.import_database_stats('*fullPathOfExportedFile*')
Backup	Database	dbms_stats.backup_database_stats('*commentUsedForIdentification*')
Restore	Database	[Format 1] dbms_stats.restore_database_stats('*timestamp*') [Timestamp] Specify in the same format as the time column of the backup_history table. Backups earlier than the specified time will be restored. [Format 2] dbms_stats.restore_stats(*backupId*) [Backup ID] Specify a value in the id column of the backup_history table. The specified backup will be restored.
Purge	Backup	dbms_stats.purge_stats(*backupId,flagUsedForDeletion*) [Backup ID] Specify a value in the id column of the backup_history table. [Flag used for deletion] true: The target backup is forcibly deleted. false: The target backup is deleted only when there are also backups for the entire database.

Remark 1: The export feature is executed using the COPY statement, not the SQL function.

Example

Example 1: Locking the statistics of the entire database

```
userdb=# SELECT dbms_stats.lock_database_stats();
  lock_database_stats
-----------------------
  tbl1
  tbl1_pkey
```

Note that the locked information can be referenced as follows:

```
userdb=# select relname  from dbms_stats._relation_stats_locked;
  relname
-----------------------
  tbl1
  tbl1_pkey
```

Example 2: Unlocking the statistics of the entire database

```
userdb=# SELECT dbms_stats.unlock_database_stats();
  unlock_database_stats
-----------------------
  tbl1
  tbl1_pkey
```

Example 3: Backing up the statistics of the entire database

```
userdb=# SELECT dbms_stats.backup_database_stats('backup1');
  backup_database_stats
-----------------------
                      1
```

Note that the backed up statistics can be referenced as follows:

```
userdb=# select id,comment,time,unit from dbms_stats.backup_history;
  id | comment  |             time              | unit
 ----+----------+-------------------------------+------
   1 | backup1  | 2014-03-04 11:08:40.315948+09 | d
```

The ID:1 backup "backup1" is obtained for each database at "2014-03-04 11:08:40.315948+09".
[Meaning of unit] d: database s: schema t: table c: column

Example 4: Exporting the statistics of the entire database

```
$ psql -d userdb -f export.sql
BEGIN
COMMIT
```

export.sql is the file in which the COPY statement is defined.
Refer to "export_effective_stats-9.2.sql_sample" for information on the content of the COPY statement.

"export_effective_stats-9.2.sql_sample" is stored as follows:
enterprise PostgresInstallDir/share/extension/pg_dbms_stats

Example 5: Importing the statistics of the entire database

```
$ psql -d userdb -c  "SELECT dbms_stats.import_database_stats ('$PWD/
export_stats.dmp')"
 import_database_stats
-----------------------

(1 row)
```

Usage notes

- You must run the ANALYZE command once for the target tables of this feature. If the ANALYZE command is not run, the statistics cannot be locked.
  Refer to "SQL Commands" in "Reference" in the PostgreSQL Documentation for information on the ANALYZE command.
- To use this feature to delete an object that has locked the statistics, use the unlock feature to delete the object lock information first.
- This feature does not specify the statistics value directly. It reproduces the status that has actually occurred. For this reason, if the text format is specified in the COPY statement when the export occurs, restore will not be possible. Always use the binary format when performing the export.

Appendix A Precautions when Developing Applications

This appendix describes precautions when developing applications with Enterprise Postgres.

A.1 Precautions when Using Functions and Operators

This section describes notes for using functions and operators.

A.1.1 General rules of Functions and Operators

This section describes general rules for using functions and operators Ensure the general rules are followed when using functions and operators to develop applications.

General rules

- Specify the stated numbers for arguments when specifying numbers for arguments in functions.
- Specify the stated data types when specifying data types for functions. If you use a data type other than the stated data types, use CAST to explicitly convert the data type.
- Specify data types that can be compared when specifying data types for operators. If you use a data type that cannot be compared, use CAST to explicitly convert the data type.

See

Refer to "Functions and Operators" under "The SQL Language" in the PostgreSQL Documentation for information on the functions and operators available with Enterprise Postgres.

A.1.2 Errors when Developing Applications that Use Functions and/or Operators

This section provides examples of problems that may occur when developing applications that use functions and/or operators, and describes how to deal with them.

The error "Function ***** does not exist" occurs when executing SQL

The following error will occur when executing an SQL statement that does not abide by the general rules for functions:

```
ERROR:  Function ****** does not exist
```

Note: "*****" denotes the function for which the error occurred, and the data type of its arguments.

The cause of the error will be one of the following:

- The specified function does not exist.
- The wrong number of arguments or wrong argument data type was specified

Corrective action

Check the following points and correct any errors:

- Check if there are any errors in the specified function name, number of arguments, or argument data type, and revise accordingly.
- Check the argument data type of the function displayed in the message. If an unintended data type is displayed, use a function such as CAST to convert it.

The error "Operator does not exist" occurs when executing SQL

The following error will occur when executing an SQL statement that specifies a data type in the operator that cannot be compared:

```
ERROR:  Operator does not exist: *****
```

Note: "*****" denotes the operator for which the error occurred, and the data type of the specified value.

Corrective action

Ensure the data type of the expressions specified on the left and right sides of the operator can be compared. If required, revise to ensure these data types can be compared by using a function such as CAST to explicitly convert them.

A.2 Notes when Using Temporary Tables

In standard SQL, a temporary table can be defined in advance to enable an empty temporary table to be created automatically when the application connects to the database. However, in Enterprise Postgres, a temporary table must be created when the application connects to the database by explicitly using the CREATE TABLE statement.

If the same temporary table is repeatedly created and deleted during the same session, the system table might expand, and memory usage might increase. To prevent this, specify the CREATE TABLE statement to ensure the temporary table is reused.

For example, in cases where a temporary table would be created and deleted for repeatedly executed transactions, specify the CREATE TABLE statement as shown below:

- Specify "IF NOT EXISTS" to create a temporary table only if none exists when the transaction starts.
- Specify "ON COMMIT DELETE ROWS" to ensure all rows are deleted when the transaction ends.

See

Refer to "SQL Commands" under "Reference" in the PostgreSQL Documentation for information on the CREATE TABLE statement.

Examples of SQL using a temporary table are shown below:

Example of bad use (creating and deleting a temporary table)

```
BEGIN;
CREATE TEMPORARY TABLE mytable(col1 CHAR(4), col2 INTEGER) ON COMMIT DROP;
     (mytable processes)

COMMIT;
```

Example of good use (reusing a temporary table)

```
BEGIN;
CREATE TEMPORARY TABLE IF NOT EXISTS mytable(col1 CHAR(4), col2 INTEGER) ON COMMIT
DELETE ROWS;
     (mytable processes)

COMMIT;
```

A.3 Implicit Data Type Conversions

An implicit data type conversion refers to a data type conversion performed automatically by Enterprise Postgres, without the need to explicitly specify the data type to convert to.

The combination of possible data type conversions differs, depending on whether the expression in the conversion source is a literal.

For non-literals, data types can only be converted to other types within the same range.

For literals, character string literal types can be converted to the target data type. Numeric literals are implicitly converted to specific numeric types. These implicitly converted numeric literals can then have their types converted to match the conversion target data type within the numeric type range. For bit character string literals, only the bit column data type can be specified. The following shows the range of type conversions for literals.

Table A.1 Data type combinations that contain literals and can be converted implicitly

Conversion target		Conversion source		
		Character literal (*1)	Numeric literal(*2)	Bit character string literal
Numeric type	SMALLINT	Y	N	N
	INTEGER	Y	Y (*3)	N
	BIGINT	Y	Y (*4)	N
	DECIMAL	Y	Y (*5)	N
	NUMERIC	Y	Y (*5)	N
	REAL	Y	N	N
	DOUBLE PRECISION	Y	N	N
	SMALLSERIAL	Y	N	N
	SERIAL	Y	Y (*3)	N
	BIGSERIAL	Y	Y (*4)	N
Currency type	MONEY	Y	N	N
Character type	CHAR	Y	N	N
	VARCHAR	Y	N	N
	NCHAR	Y	N	N
	NCHAR VARYING	Y	N	N
	TEXT	Y	N	N
Binary data type	BYTEA	Y	N	N
Date/time type	TIMESTAMP WITHOUT TIME ZONE	Y	N	N
	TIMESTAMP WITH TIME ZONE	Y	N	N
	DATE	Y	N	N
	TIME WITHOUT TIME ZONE	Y	N	N
	TIME WITH TIME ZONE	Y	N	N
	INTERVAL	Y	N	N
Boolean type	BOOLEAN	Y	N	N
Geometric type	POINT	Y	N	N
	LSEG	Y	N	N
	BOX	Y	N	N

Conversion target		Conversion source		
		Character literal (*1)	Numeric literal(*2)	Bit character string literal
	PATH	Y	N	N
	POLYGON	Y	N	N
	CIRCLE	Y	N	N
Network address type	CIDR	Y	N	N
	INET	Y	N	N
	MACADDR	Y	N	N
Bit string type	BIT	Y	N	Y
	BIT VARYING	Y	N	Y
Text search type	TSVECTOR	Y	N	N
	TSQUERY	Y	N	N
UUID type	UUID	Y	N	N
XML type	XML	Y	N	N
JSON type	JSON	Y	N	N

Y: Can be converted
N: Cannot be converted

*1: Only strings that can be converted to the data type of the conversion target can be specified (such as "1" if the conversion target is a numeric type)

*2: "Y" indicates specific numeric types that are converted first.

*3: Integers that can be expressed as INTEGER types can be specified

*4: Integers that cannot be expressed as INTEGER types, but can be expressed as BIGINT types, can be specified

*5: Integers that cannot be expressed as INTEGER or BIGINT types, but that can be expressed as NUMERIC types, or numeric literals that contain a decimal point or the exponent symbol (e), can be specified

Implicit data type conversions can be used when comparing or storing data.

The conversion rules differ, depending on the reason for converting. Purpose-specific explanations are provided below.

A.3.1 Function Argument

Value expressions specified in a function argument will be converted to the data type of that function argument.

Refer to "Functions and Operators" under "The SQL Language" in the PostgreSQL Documentation for information on data types that can be specified in function arguments.

A.3.2 Operators

Comparison operators, BETWEEN, IN

Combinations of data types that can be compared using comparison operators, BETWEEN, or IN are shown below.

Table A.2 Combinations of comparable data type

Left side	Right side		
	Numeric type	Character string type	Date/time type
Numeric type	Y	N	N
Character type	N	Y	N
Date/time type	N	N	Y

Y: Can be compared
N: Cannot be compared

When strings with different lengths are compared, the shorter one is padded with spaces to make the lengths match.

When numeric values with different precisions are compared, data will be converted to the type with the higher precision.

Set operation and CASE also follow the same rules.

Other operators

Value expressions specified in operators will be converted to data types that are valid for that operator.

See

Refer to "Functions and Operators" under "The SQL Language" in the PostgreSQL Documentation for information on data types that can be specified in operators.

A.3.3 Storing Values

Value expressions specified in the VALUES clause of the INSERT statement or the SET clause of the UPDATE statement will be converted to the data type of the column in which they will be stored.

A.4 Notes on Using Index

This section explains the notes on using the following indexes:

- Hush index
- SP-GiST index

A.4.1 Hush Index

Update operations for the hush index are not recorded in the WAL.

Therefore, if the database server fails, the hush index needs to be rebuilt with REINDEX to recover the database server.

Also, after obtaining the first backup, queries that use the hush index would return incorrect results, since index updates are not reflected in streaming replication and file-based replication. For these reasons, use of the hush index is not recommended.

A.4.2 SP-GiST Index

If more than 2 concurrent updates are performed on a table in which the SP-GiST index is defined, applications may stop responding. When this occur, all system processes including the Check Pointer process will also be in the state of no response. For these reasons, use of the SP-GiST index is not recommended.

A.5 Notes on Using the psql Commands

This section explains the notes on using the psql commands.

A.5.1 Entering Multibyte Characters

Multibyte characters cannot be entered from the psql command prompt.

Multibyte characters can be entered from a file by specifying the -f option.

If using this option, the target file encoding system must be explicitly specified as the client encoding system.

See

Refer to "Character Set Support" in "Server Administration" in the PostgreSQL Documentation for information on how to configure the client encoding system.

A.6 Notes on Using Multibyte Characters in Definition Names

Do not use multibyte characters in database names or user names if using a Windows database server.

Multibyte characters must not be used in database space names or user names on non-Windows database servers, because certain conditions may apply or it may not be possible to connect to some clients.

Related notes and constraints are described below.

1) Configuring the client encoding system

The client encoding system must be configured when the names are created.

See

Refer to "Character Set Support" in "Server Administration" in the PostgreSQL Documentation for information on how to configure the client encoding system.

2) Encoding system of names used for connection

Ensure that the encoding system of names used for connection is the same as that of the database that was connected when these names were created.

The reasons for this are as follows:

- Storage system for names in Enterprise Postgres

  The system catalog saves encoded names by using the encoding system of the database at the time the names were created.

- Encoding conversion policy when connected

  When connected, names sent from the client are matched with names in the system catalog without performing encoding conversion.

Accordingly, if the database that was connected when the names were defined uses the EUC_JP encoding system, but the database name is specified using UTF-8 encoding, then the database will be considered to be non-existent.

3) Connection constraints

The table below shows the connection constraints for each client type, based on the following assumptions:

- The conditions described in 1) and 2) above are satisfied.

- The database name and user names use the same encoding system.

Client type	Client operating system	
	Windows	Linux
JDBC driver	Cannot be connected	Cannot be connected
ODBC driver	Cannot be connected	No connection constraints
.NET Data Provider	Can only connect when the encoding system used for definitions is UTF-8	-
SQLEmbedded SQL in C	Can only connect when the connection service file (pg_service.conf) is used	No connection constraints
psql command	Can only connect when the connection service file (pg_service.conf) is used	No connection constraints

Appendix B Conversion Procedures Required due to Differences from Oracle Database

This appendix explains how to convert from an Oracle database to Enterprise Postgres, within the scope noted in "Chapter 9 Compatibility with Oracle Databases" from the following perspectives:

- Feature differences
- Specification differences

This document assumes that the version of the Oracle database to be converted is 7-10.2g.

B.1 Outer Join Operator (Perform Outer Join)

Features

In the WHERE clause conditional expression, by adding the plus sign (+), which is the outer join operator, to the column of the table you want to add as a table join, it is possible to achieve an outer join that is the same as a joined table (OUTER JOIN).

B.1.1 Comparing with the ^= Comparison Operator

Oracle database

```
SELECT *
  FROM t1, t2
  WHERE t1.col1(+) ^= t2.col1;
```

* col1 is assumed to be CHAR(4) type

Enterprise Postgres

```
SELECT *
  FROM t1, t2
  WHERE t1.col1(+) != t2.col1;
```

* col1 is assumed to be CHAR(4) type

Feature differences

Oracle database

The ^= comparison operator can be specified.

Enterprise Postgres

The ^= comparison operator cannot be specified.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "^=" is used.
2. Ensure that the keyword, "(+)", is either on the right or left-hand side.
3. Change "^=" to " !=".

B.2 DECODE (Compare Values and Return Corresponding Results)

Features

DECODE compares values of the conversion target value expression and the search values one by one, and if the values of the conversion target value expression and the search values match, a corresponding result value is returned.

B.2.1 Comparing Numeric Data of Character String Types and Numeric Characters

Oracle database

```
SELECT DECODE( col1,
               1000, 'ITEM-A',
               2000, 'ITEM-B',
               'ITEM-C')
  FROM t1;
```

* col1 is assumed to be CHAR(4) type

Enterprise Postgres

```
SELECT DECODE( CAST(col1 AS INTEGER),
               1000, 'ITEM-A',
               2000, 'ITEM-B',
               'ITEM-C')
  FROM t1;
```

* col1 is assumed to be CHAR(4) type

Feature differences

Oracle database

When the value expression is a string and the search value is a numeric, the string value will be converted to the data type of the comparison target numeric, so that they can be compared.

Enterprise Postgres

If the conversion target value expression is a string value, then no search value can be specified with numbers.

Conversion procedure

Since the data type that can be specified for the conversion target value expression is unknown, use CAST to explicitly convert the conversion target value expression (col1 in the example) to a numeric (INTEGER type in the example).

B.2.2 Obtaining Comparison Result from more than 50 Conditional Expressions

Oracle database

```
SELECT DECODE(col1,
               1,'A',
               2,'B',
               ...
               78,'BZ',
               NULL,'UNKNOWN',
               'OTHER')
   FROM t1;
```

* col1 is assumed to be INTEGER type

Enterprise Postgres

```
SELECT CASE
           WHEN col1 = 1 THEN 'A'
           WHEN col1 = 2 THEN 'B'
           ...
           WHEN col1 = 78 THEN 'BZ'
           WHEN col1 IS NULL THEN 'UNKNOWN'
           ELSE 'OTHER'
         END
    FROM t1;
```

* col1 is assumed to be INTEGER type

Feature differences

Oracle database

Search value with a maximum of 127 items (up to 255 arguments in total) can be specified.

Enterprise Postgres

Search value with a maximum of 49 items (up to 100 arguments in total) only can be specified.

Conversion procedure

Convert to the CASE expression using the following procedure:

1. Specify the DECODE conversion target value expression (col1 in the first argument, in the example) and the search value (1 in the second argument, in the example) for the CASE expression search condition. Specify the DECODE result value ('A' in the third argument, in the example) for the CASE expression THEN (WHEN col1 = 1 THEN 'A', in the example). Note that if the search value is NULL, specify "IS NULL" for the search condition for the CASE expression.
2. If the DECODE default value ('OTHER' in the last argument, in the example) is specified, specify the default value for the CASE expression ELSE (ELSE 'OTHER', in the example).

B.2.3 Obtaining Comparison Result from Values with Different Data Types

Oracle database

```
SELECT DECODE( col1,
               '1000', 'A',
               '2000', 1,
               'OTHER')
FROM t1;
```

* col1 is assumed to be CHAR(4) type

Enterprise Postgres

```
SELECT DECODE( col1,
              '1000', 'A',
              '2000', '1',
              'OTHER')
FROM t1;
```

* col1 is assumed to be CHAR(4) type

Feature differences

Oracle database

The data types of all result values are converted to the data type of the first result value.

Enterprise Postgres

Results in an error.

Conversion procedure

Convert using the following procedure:

1. Check the literal data type for the first result value specified.
2. Change the literals specified for each result value to the literal data type checked in the step 1.

B.3 SUBSTR (Extract a String of the Specified Length from Another String)

Features

SUBSTR returns the number of characters specified in the third argument (starting from the position specified in the second argument) from the string specified in the first argument.

Refer to "9.2.1 Notes on SUBSTR" for details on precautions when using SUBSTR.

B.3.1 Specifying a Value Expression with a Data Type Different from the One that can be Specified for Function Arguments

Oracle database

```
SELECT SUBSTR( col1,
               1,
               col2)
  FROM DUAL;
```

* col1 and col2 are assumed to be CHAR type

Enterprise Postgres

```
CREATE CAST (CHAR AS INTEGER) WITH INOUT AS IMPLICIT;

SELECT SUBSTR( col1,
               1,
               col2)
  FROM DUAL;
# No changes to SELECT statement;
```

* col1 and col2 are assumed to be CHAR type

Feature differences

Oracle database

If the type can be converted to a data type that can be specified for function arguments, conversion is performed implicitly.

Enterprise Postgres

If the data types are different from each other, or if loss of significance occurs, implicit conversion is not performed.

Conversion procedure

Since the data type of the string length is clear, first execute the following CREATE CAST only once so that the CHAR type value (col2 in the example) specified for the string length is implicitly converted to INTEGER type.

```
CREATE CAST (CHAR AS INTEGER) WITH INOUT AS IMPLICIT;
```

B.3.2 Extracting a String with the Specified Format from a Datetime Type Value

Oracle database

```
SELECT SUBSTR( CURRENT_TIMESTAMP,
               1,
               8)
  FROM DUAL;
```

Enterprise Postgres

```
SELECT SUBSTR( TO_CHAR(CURRENT_TIMESTAMP,
               'DD-MON-YY HH.MI.SS.US PM')
                1,
                8)
  FROM DUAL;
```

Feature differences

Oracle database

A datetime value such as CURRENT_TIMESTAMP can be specified for character value expressions.

Enterprise Postgres

A datetime value such as CURRENT_TIMESTAMP cannot be specified for character value expressions.

Conversion procedure

First, specify TO_CHAR for the SUBSTR character value expression.
Specify datetime type (CURRENT_TIMESTAMP, in the example) in firstArg of TO_CHAR, and specify the format template pattern ('DD-MON-YY HH.MI.SS.US PM', in the example) for secondArg to match with the result of SUBSTR before conversion.

TO_CHAR specification format: TO_CHAR(*firstArg, secondArg*)

Information

Refer to "Data Type Formatting Functions" in the PostgreSQL Documentation for information on format template patterns that can be specified for TO_CHAR in Enterprise Postgres.

B.3.3 Concatenating a String Value with a NULL value

Oracle database

```
SELECT SUBSTR( col1 || col2,
               2,
               5)
  FROM t1;
```

* col1 and col2 are assumed to be character string type, and col2 may contain NULL

Enterprise Postgres

```
SELECT SUBSTR( col1 || NVL(col2, '')
               2,
               5)
  FROM t1;
```

* col1 and col2 are assumed to be character string type, and col2 may contain NULL

Feature differences

Oracle database

NULL is handled as an empty string, and strings are joined.

Enterprise Postgres

NULL is not handled as an empty string, and the result of joining the strings becomes NULL.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "||" is used.
2. Check if any of the value expressions can contain NULL - if they can, then execute step 3.
3. Modify to NVL(*valExpr,*'').

B.4 NVL (Replace NULL)

Features

NVL converts NULL values.

B.4.1 Obtaining Result from Arguments with Different Data Types

Oracle database

```
SELECT NVL( col1,
            col2)
  FROM t1;
```

* col1 is assumed to be VARCHAR(100) type, and col2 is assumed to be CHAR(100) type

Enterprise Postgres

```
SELECT NVL( col1,
            CAST(col2 AS VARCHAR(100)))
  FROM t1;
```

* col1 is assumed to be VARCHAR(100) type, and col2 is assumed to be CHAR(100) type

Feature differences

Oracle database

Value expressions with different data types can be specified. If the first argument is a string value, then VARCHAR2 is returned, and if it is a numeric, then a numeric type with greater range is returned.

Enterprise Postgres

Value expressions with different data types cannot be specified.

Conversion procedure

Since the data types that can be specified for expressions 1 and 2 are unknown, use the following steps to convert:

1. Check the data types specified for the expressions 1 and 2.
2. Using the data type that is to be received as a result, explicitly convert the other argument with CAST.

B.4.2 Operating on Datetime/Numeric, Including Adding Number of Days to a Particular Day

Oracle database

```
SELECT NVL( col1 + 10, CURRENT_DATE)
  FROM t1;
```

* col1 is assumed to be TIMESTAMP WITHOUT TIME ZONE type or TIMESTAMP WITH TIME ZONE type

Enterprise Postgres

```
SELECT NVL( CAST(col1 AS DATE) + 10, CURRENT_DATE)
  FROM t1;
```

* col1 is assumed to be TIMESTAMP WITHOUT TIME ZONE type or TIMESTAMP WITH TIME ZONE type

Feature differences

Oracle database

Numbers can be operated (added to or subtracted from) with either TIMESTAMP WITHOUT TIME ZONE type or TIMESTAMP WITH TIME ZONE type. Operation result will be DATE type.

Enterprise Postgres

Numbers cannot be operated (added to or subtracted from) with neither TIMESTAMP WITHOUT TIME ZONE type nor TIMESTAMP WITH TIME ZONE type. However, numbers can be operated (added to or subtracted from) with DATE type.

Conversion procedure

Convert using the following procedure:

1. Search locations where the keyword "+" or "-" is used in addition or subtraction, and check if these operations are between numbers and TIMESTAMP WITHOUT TIME ZONE type or TIMESTAMP WITH TIME ZONE type.
2. If they are, use CAST to explicitly convert TIMESTAMP WITHOUT TIME ZONE type or TIMESTAMP WITH TIME ZONE type to DATE type.

B.4.3 Calculating INTERVAL Values, Including Adding Periods to a Date

Oracle database

```
SELECT NVL( CURRENT_DATE + (col1 * 1.5), col2)
  FROM t1;
```

* col1 and col2 are assumed to be INTERVAL YEAR TO MONTH types

Enterprise Postgres

```
SELECT NVL( CURRENT_DATE +
        CAST(col1 * 1.5 AS
```

```
          INTERVAL YEAR TO MONTH), col2)
  FROM t1;
```

* col1 and col2 are assumed to be INTERVAL YEAR TO MONTH types

Feature differences

Oracle database

INTERVAL YEAR TO MONTH type multiplication and division result in INTERVAL YEAR TO MONTH type and any fraction (number of days) will be truncated.

Enterprise Postgres

INTERVAL YEAR TO MONTH type multiplication and division result in INTERVAL type and fractions (number of days) will not be truncated.

Conversion procedure

Convert using the following procedure:

1. Search locations where the keywords "*" or "/" are used in multiplication or division, and check if the specified value is INTERVAL YEAR TO MONTH type.
2. If the value is INTERVAL YEAR TO MONTH type, use CAST to explicitly convert the operation result to INTERVAL YEAR TO MONTH type.

B.5 DBMS_OUTPUT (Output Messages)

Features

DBMS_OUTPUT sends messages to clients such as psql from PL/pgSQL.

B.5.1 Outputting Messages Such As Process Progress Status

Oracle database

```
set serveroutput on;...(1)

DECLARE
  v_col1        CHAR(20);
  v_col2        INTEGER;
  CURSOR c1 IS
    SELECT col1, col2 FROM t1;
BEGIN
  DBMS_OUTPUT.PUT_LINE('-- BATCH_001 Start --');
  OPEN c1;
  DBMS_OUTPUT.PUT_LINE('-- LOOP Start --');
  LOOP
    FETCH c1 INTO v_col1, v_col2;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT('.');
  END LOOP;
  DBMS_OUTPUT.NEW_LINE; ...(2)
  DBMS_OUTPUT.PUT_LINE('-- LOOP End --');
  CLOSE c1;

  DBMS_OUTPUT.PUT_LINE('-- BATCH_001 End --');


EXCEPTION
  WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('-- SQL Error --');
     DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
```

```
END;
/
```

Enterprise Postgres

```
DO $$
DECLARE
    v_col1      CHAR(20);
    v_col2      INTEGER;
    c1 CURSOR FOR
        SELECT col1, col2 FROM t1;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE); ...(1)
    PERFORM DBMS_OUTPUT.ENABLE(NULL); ...(1)

    PERFORM DBMS_OUTPUT.PUT_LINE('-- BATCH_001 Start --');

    OPEN c1;
    PERFORM DBMS_OUTPUT.PUT_LINE('-- LOOP Start --');
    LOOP
        FETCH c1 INTO v_col1, v_col2;
        EXIT WHEN FOUND = false;
        PERFORM DBMS_OUTPUT.PUT('.');
    END LOOP;
    PERFORM DBMS_OUTPUT.NEW_LINE(); ...(2)

    PERFORM DBMS_OUTPUT.PUT_LINE('-- LOOP End --');
    CLOSE c1;

    PERFORM DBMS_OUTPUT.PUT_LINE('-- BATCH_001 End --');

EXCEPTION
    WHEN OTHERS THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('-- SQL Error --');
        PERFORM DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
END;
$$
;
```

(1) SERVEROUTPUT/ENABLE

Specification differences

Oracle database

Use SET statement and specify SERVEROUTPUT ON.

Enterprise Postgres

Specify DBMS_SQL.SERVEROUTPUT(TRUE).

Conversion procedure

Convert using the following procedure:

1. Check if a SET SERVEROUTPUT statement is specified before the PL/SQL block of a stored procedure.
2. If a SET SERVEROUTPUT statement is specified, specify DBMS_SQL.SERVEROUTPUT straight after BEGIN of PL/pgSQL. If ON is specified to have messages output to a window, then specify TRUE. If OFF is specified, then specify FALSE.
3. Specify DBMS_SQL.ENABLE only if SET SERVEROUTPUT is ON. The values to be specified for the argument are as follows:
   - If SIZE is specified for the SET SERVEROUTPUT statement, specify this size for the argument.

- If SIZE is not specified for the SET SERVEROUTPUT statement, then specify 2000 for Oracle10.1g or earlier, NULL for Oracle10.2g or later.

If DBMS_SQL.ENABLE is specified for the PL/SQL block of the stored procedure, specify the same value as that argument.

(2) NEW_LINE

Specification differences

Oracle database

If there is no argument for *packageName.featureName*, parenthesis can be omitted.

Enterprise Postgres

Even if there is no argument for *packageName.featureName*, parenthesis cannot be omitted.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "DBMS_OUTPUT.NEW_LINE" is used in the stored procedure.
2. If there is no parenthesis after *packageName.featureName*, add the parenthesis.

B.5.2 Receiving a Return Value from a Procedure (PL/SQL) Block (For GET_LINES)

Oracle database

```
set serveroutput off;

DECLARE
    v_num        INTEGER;
BEGIN

    DBMS_OUTPUT.DISABLE; ...(3)
    DBMS_OUTPUT.ENABLE(20000); ...(4)
    DBMS_OUTPUT.PUT_LINE('-- ITEM CHECK --');


    SELECT count(*) INTO v_num FROM t1;

    IF v_num = 0 THEN
        DBMS_OUTPUT.PUT_LINE('-- NO ITEM --');

    ELSE
        DBMS_OUTPUT.PUT_LINE('-- IN ITEM(' || v_num || ') --');
    END IF;
END;
/

set serveroutput on;

DECLARE
    v_buffs      DBMSOUTPUT_LINESARRAY; ...(5)
    v_num        INTEGER := 10;
BEGIN

    DBMS_OUTPUT.GET_LINES(v_buffs, v_num); ...(5)

    FOR i IN 1..v_num LOOP
        DBMS_OUTPUT.PUT_LINE('LOG : ' || v_buffs(i)); ...(5)
```

```
    END LOOP;
END;
/
```

Enterprise Postgres

```
DO $$
DECLARE
    v_num        INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(FALSE);
    PERFORM DBMS_OUTPUT.DISABLE(); ...(3)
    PERFORM DBMS_OUTPUT.ENABLE(20000); ...(4)
PERFORM DBMS_OUTPUT.PUT_LINE('-- ITEM CHECK --');

    SELECT count(*) INTO v_num FROM t1;

    IF v_num = 0 THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('-- NO ITEM --');
    ELSE
        PERFORM DBMS_OUTPUT.PUT_LINE('-- IN ITEM(' || v_num || ') --');
    END IF;
END;
$$
;

DO $$
DECLARE
    v_buffs      VARCHAR[]; ...(5)
    v_num        INTEGER := 10;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    SELECT lines, numlines INTO v_buffs, v_num FROM DBMS_OUTPUT.GET_LINES(v_num); ...(5)

    FOR i IN 1..v_num LOOP
        PERFORM DBMS_OUTPUT.PUT_LINE('LOG : ' || v_buffs[i]); ...(5)
    END LOOP;
END;
$$
;
```

(3) DISABLE

Same as the NEW_LINE in the DBMS_OUTPUT package. Refer to NEW_LINE for information on specification differences and conversion procedures associated with specification differences.

(4) ENABLE

Same as NEW_LINE in the DBMS_OUTPUT package. Refer to NEW_LINE for information on specification differences and conversion procedures associated with specification differences.

(5) GET_LINES

Specification format for Oracle database

DBMS_OUTPUT.GET_LINES(*firstArg, secondArg*)

Specification differences

Oracle database

Obtained values are received with variables specified for arguments.

Enterprise Postgres

Since obtained values are the search results for DBMS_OUTPUT.GET_LINES, they are received with variables specified for the INTO clause of the SELECT statement.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "DBMS_OUTPUT.GET_LINES" is used in the stored procedure.
2. Change the data type (DBMSOUTPUT_LINESARRAY in the example) of the variable (v_buffs in the example) specified as *firstArg* of DBMS_OUTPUT.GET_LINES into a VARCHAR type array (VARCHAR[] in the example).
3. Replace the DBMS_OUTPUT.GET_LINES location called with a SELECT INTO statement.
   - Use the literal "lines, numlines" in the select list.
   - Specify *firstArg* (v_buffs in the example) and *secondArg* (v_num in the example) configured in DBMS_OUTPUT.GET_LINES, in the INTO clause.
   - Use DBMS_OUTPUT.GET_LINES in the FROM clause. Specify only *secondArg* (v_num in the example) before modification.
4. Identify the location that references *firstArg* (v_buffs in the example), and change it to the PL/pgSQL array reference format (v_buffs[i] in the example).

B.5.3 Receiving a Return Value from a Procedure (PL/SQL) Block (For GET_LINE)

Oracle database

```
set serveroutput on;

DECLARE
    v_buff1       VARCHAR2(100);
    v_buff2       VARCHAR2(1000);
    v_num        INTEGER;
BEGIN

    v_buff2 := '';
    LOOP
        DBMS_OUTPUT.GET_LINE(v_buff1, v_num); ...(6)
        EXIT WHEN v_num = 1;
        v_buff2 := v_buff2 || v_buff1;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE(v_buff2);
END;
/
```

* Only the process to obtain a value is stated

Enterprise Postgres

```
DO $$
DECLARE
    v_buff1       VARCHAR(100);
    v_buff2       VARCHAR(1000);
    v_num        INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    v_buff2 := '';
    LOOP
```

```
        SELECT line, status INTO v_buff1, v_num FROM DBMS_OUTPUT.GET_LINE(); ...(6)
        EXIT WHEN v_num = 1;
        v_buff2 := v_buff2 || v_buff1;
    END LOOP;

    PERFORM DBMS_OUTPUT.PUT_LINE(v_buff2);
END;
$$
;
```

* Only the process to obtain a value is stated

(6) GET_LINE

Specification format for Oracle database

DBMS_OUTPUT.GET_LINE(*firstArg, secondArg*)

Specification differences

Oracle database

Obtained values are received with variables specified for arguments.

Enterprise Postgres

Since obtained values are the search results for DBMS_OUTPUT.GET_LINES, they are received with variables specified for the INTO clause of the SELECT statement.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "DBMS_OUTPUT.GET_LINE" is used in the stored procedure.
2. Replace the DBMS_OUTPUT.GET_LINE location called with a SELECT INTO statement.
   - Use the literal "line, status" in the select list.
   - Specify *firstArg* (v_buff1 in the example) and *secondArg* (v_num in the example) configured in DBMS_OUTPUT.GET_LINE, in the INTO clause.
   - Use DBMS_OUTPUT.GET_LINE in the FROM clause. Although arguments are not specified, parenthesis must be specified.

B.6 UTL_FILE (Perform File Operation)

Features

UTL_FILE reads and writes text files from PL/pgSQL.

B.6.1 Registering a Directory to Load and Write Text Files

Oracle database

```
[Oracle9i or earlier]
Configure the following with initialization parameter
   UTL_FILE_DIR='/home/fsep' ...(1)

[Oracle9.2i or later]
   Configure the following with CREATE DIRECTORY statement
   CREATE DIRECTORY DIR AS '/home/fsep'; ...(1)
```

Enterprise Postgres

```
INSERT INTO UTL_FILE.UTL_FILE_DIR(dir)
       VALUES('/home/fsep'); ...(1)
```

(1) UTL_FILE_DIR/CREATE DIRECTORY

Feature differences

Oracle database

Configure the directory to be operated, using the CREATE DIRECTORY statement or the initialization parameter UTL_FILE_DIR.

Enterprise Postgres

The directory to be operated cannot be configured using the CREATE DIRECTORY statement or the initialization parameter UTL_FILE_DIR.

Conversion procedure

Configure the target directory information in the UTL_FILE.UTL_FILE_DIR table using the INSERT statement. Note that this conversion procedure should be performed only once before executing the PL/pgSQL function.

- When using the initialization parameter UTL_FILE_DIR:

  1. Check the initialization parameter UTL_FILE_DIR value ('/home/fsep' in the example).
  2. Using the INSERT statement, specify and execute the directory name checked in step 1.
     - Specify UTL_FILE.UTL_FILE_DIR(dir) for the INTO clause.
     - Using the character string literal ('/home/fsep' in the example), specify the target directory name for the VALUES clause.
     - If multiple directories are specified, execute the INSERT statement for each directory.

- When using the CREATE DIRECTORY statement:

  1. Check the directory name ('/home/fsep' in the example) registered with the CREATE DIRECTORY statement. To check, log in SQL*Plus as a user with DBA privileges, and execute "show ALL_DIRECTORIES;".
  2. Using the INSERT statement, specify and execute the directory name checked in step 1. Same steps are used to specify the INSERT statement as when using the initialization parameter UTL_FILE_DIR.

B.6.2 Checking File Information

Oracle database

```
CREATE PROCEDURE read_file(fname VARCHAR2) AS


    v_file      UTL_FILE.FILE_TYPE;
    v_exists    BOOLEAN;
    v_length    NUMBER;
    v_bsize     INTEGER;
    v_rbuff     VARCHAR2(1024);
BEGIN


    UTL_FILE.FGETATTR('DIR', fname, v_exists, v_length, v_bsize); ...(2)


    IF v_exists <> true THEN
        DBMS_OUTPUT.PUT_LINE('-- FILE NOT FOUND --');
        RETURN;
    END IF;
```

```
    DBMS_OUTPUT.PUT_LINE('-- FILE DATA --');

    v_file := UTL_FILE.FOPEN('DIR', fname, 'r', 1024); ...(3)
    FOR i IN 1..3 LOOP
        UTL_FILE.GET_LINE(v_file, v_rbuff, 1024); ...(4)
        DBMS_OUTPUT.PUT_LINE(v_rbuff);
    END LOOP;
    DBMS_OUTPUT.PUT_LINE('... more');
    DBMS_OUTPUT.PUT_LINE('-- READ END --');

    UTL_FILE.FCLOSE(v_file); ...(5)
    RETURN;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('-- FILE END --');

        UTL_FILE.FCLOSE(v_file);
        RETURN;
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('-- SQL Error --');

        DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
        UTL_FILE.FCLOSE_ALL; ...(6)
        RETURN;
END;
/

set serveroutput on

call read_file('file01.txt');
```

Enterprise Postgres

```
CREATE FUNCTION read_file(fname VARCHAR) RETURNS void AS $$
DECLARE
    v_file      UTL_FILE.FILE_TYPE;
    v_exists    BOOLEAN;
    v_length    NUMERIC;
    v_bsize     INTEGER;
    v_rbuff     VARCHAR(1024);
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);

    SELECT fexists, file_length, blocksize
       INTO v_exists, v_length, v_bsize
       FROM UTL_FILE.FGETATTR('/home/fsep', fname); ...(2)
    IF v_exists <> true THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('-- FILE NOT FOUND --');
        RETURN;
    END IF;

    PERFORM DBMS_OUTPUT.PUT_LINE('-- FILE DATA --');
    v_file := UTL_FILE.FOPEN('/home/fsep', fname, 'w', 1024); ...(3)
    FOR i IN 1..3 LOOP
        v_rbuff := UTL_FILE.GET_LINE(v_file, 1024); ...(4)
        PERFORM DBMS_OUTPUT.PUT_LINE(v_rbuff);
    END LOOP;
    PERFORM DBMS_OUTPUT.PUT_LINE('... more');
    PERFORM DBMS_OUTPUT.PUT_LINE('-- READ END --');

    v_file := UTL_FILE.FCLOSE(v_file); ...(5)
```

```
    RETURN;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('-- FILE END --');
        v_file := UTL_FILE.FCLOSE(v_file);
        RETURN;
    WHEN OTHERS THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('-- SQL Error --');
        PERFORM DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
        PERFORM UTL_FILE.FCLOSE_ALL(); ...(6)
        RETURN;
END;
$$
LANGUAGE plpgsql;

SELECT read_file('file01.txt');
```

(2) FGETATTR

Specification format for Oracle database

UTL_FILE.FGETATTR(*firstArg, secondArg*, *thirdArg*, *fourthArg*, *fifthArg*)

Feature differences

Oracle database

If using a CREATE DIRECTORY statement (Oracle9.2i or later), specify a directory object name for the directory name.

Enterprise Postgres

A directory object name cannot be specified for the directory name.

Specification differences

Oracle database

Obtained values are received with variables specified for arguments.

Enterprise Postgres

Since obtained values are the search results for UTL_FILE.FGETATTR, they are received with variables specified for the INTO clause of the SELECT statement.

Conversion procedure

Convert using the following procedure. Refer to UTL_FILE_DIR/CREATE DIRECTORY for information on how to check if the directory object name corresponds to the actual directory name.

1. Locate the places where the keyword "UTL_FILE.FOPEN" is used in the stored procedure.
2. Check the actual directory name ('/home/fsep' in the example) that corresponds to the directory object name ('DIR' in the example).
3. Replace the directory object name ('DIR' in the example) in *firstArg* with the actual directory name ('/home/fsep' in the example) verified in step 2.
4. Replace the UTL_FILE.FGETATTR location called with a SELECT INTO statement.
   - Use the literal "fexists, file_length, blocksize" in the select list.
   - Specify *thirdArg*, *fourthArg*, and *fifthArg* (v_exists, v_length, v_bsize, in the example) specified for UTL_FILE.FGETATTR to the INTO clause in the same order as that of the arguments.
   - Use UTL_FILE.FGETATTR in the FROM clause. Specify only the actual directory name for *firstArg* ('/home/fsep' in the example) and *secondArg* (fname in the example) before modification for the arguments.

(3) FOPEN

Specification format for Oracle

UTL_FILE.FOPEN(*firstArg, secondArg*, *thirdArg*, *fourthArg*, *fifthArg*)

Feature differences

Oracle database

If using a CREATE DIRECTORY statement (Oracle9.2i or later), specify a directory object name for the directory name.

Enterprise Postgres

A directory object name cannot be specified for the directory name.

Conversion procedure

Convert using the following procedure. Refer to UTL_FILE_DIR/CREATE DIRECTORY for information on how to check if the directory object name corresponds to the actual directory name.

1. Locate the places where the keyword "UTL_FILE.FOPEN" is used in the stored procedure.
2. Check the actual directory name ('/home/fsep' in the example) that corresponds to the directory object name ('DIR' in the example).
3. Replace the directory object name ('DIR' in the example) in *firstArg* with the actual directory name ('/home/fsep' in the example) checked in step 1.

(4) GET_LINE

Specification format for Oracle database

UTL_FILE.GET_LINE(*firstArg, secondArg*, *thirdArg*, *fourthArg*)

Specification differences

Oracle database

Obtained values are received with variables specified for arguments.

Enterprise Postgres

Since obtained values are the returned value of UTL_FILE.GET_LINE, they are received with variables specified for substitution statement.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "UTL_FILE.GET_LINE" is used in the stored procedure.
2. Replace the UTL_FILE.GET_LINE location called with a value assignment (:=).
   - On the left-hand side, specify *secondArg* (v_rbuff in the example) specified for UTL_FILE.GET_LINE.
   - Use UTL_FILE.GET_LINE in the right-hand side. Specify only *firstArg* (v_file in the example) and *thirdArg* (1024 in the example) before modification.

(5) FCLOSE

Specification format for Oracle database

UTL_FILE.FCLOSE(*firstArg*)

Specification differences

Oracle database

After closing, the file handler specified for the argument becomes NULL.

Enterprise Postgres

After closing, set the file handler to NULL by assigning the return value of UTL_FILE.FCLOSE to it.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "UTL_FILE.FCLOSE" is used in the stored procedure.
2. Replace the UTL_FILE.FCLOSE location called with a value assignment (:=) so that the file handler (v_file in the example) becomes NULL.
   - On the left-hand side, specify the argument (v_file in the example) specified for UTL_FILE.FCLOSE.
   - Use UTL_FILE.FCLOSE in the right-hand side. For the argument, specify the same value (v_file in the example) as before modification.

(6) FCLOSE_ALL

Same as NEW_LINE in the DBMS_OUTPUT package. Refer to NEW_LINE in the DBMS_OUTPUT for information on specification differences and conversion procedures associated with specification differences.

B.6.3 Copying Files

Oracle database

```
CREATE PROCEDURE copy_file(fromname VARCHAR2, toname VARCHAR2) AS
BEGIN


    UTL_FILE.FCOPY('DIR1', fromname, 'DIR2', toname, 1, NULL); ...(7)

    RETURN;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('-- SQL Error --');

        DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
        RETURN;
END;
/

set serveroutput on

call copy_file('file01.txt','file01_bk.txt');
```

Enterprise Postgres

```
CREATE FUNCTION copy_file(fromname VARCHAR, toname VARCHAR) RETURNS void AS $$
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);

    PERFORM UTL_FILE.FCOPY('/home/fsep', fromname, '/home/backup', toname, 1, NULL); ...(7)
    RETURN;

EXCEPTION
    WHEN OTHERS THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('-- SQL Error --');
        PERFORM DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
        RETURN;
END;
```

```
$$
LANGUAGE plpgsql;


SELECT copy_file('file01.txt','file01_bk.txt');
```

(7) FCOPY

Specification format for Oracle database

UTL_FILE.FCOPY(*firstArg, secondArg*, *thirdArg*, *fourthArg, fifthArg, sixthArg*)

Feature differences

Oracle database

If using a CREATE DIRECTORY statement (Oracle9.2i or later), specify a directory object name for the directory name.

Enterprise Postgres

A directory object name cannot be specified for the directory name.

Conversion procedure

Convert using the following procedure. Refer to UTL_FILE_DIR/CREATE DIRECTORY for information on how to check if the directory object name corresponds to the actual directory name.

1. Locate the places where the keyword "UTL_FILE.FCOPY" is used in the stored procedure.
2. Check the actual directory names ('/home/fsep' and '/home/backup', in the example) that correspond to the directory object names ('DIR1' and 'DIR2', in the example) of *firstArg* and *thirdArg* argument.
3. Replace the directory object name ('DIR1' and 'DIR2', in the example) with the actual directory names ('/home/fsep' in the example) checked in step 1.

B.6.4 Moving/Renaming Files

Oracle database

```
CREATE PROCEDURE move_file(fromname VARCHAR2, toname VARCHAR2) AS
BEGIN


    UTL_FILE.FRENAME('DIR1', fromname, 'DIR2', toname, FALSE); ...(8)
    RETURN;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('-- SQL Error --');

        DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
        RETURN;
END;
/

set serveroutput on

call move_file('file01.txt','file02.txt');
```

Enterprise Postgres

```
CREATE FUNCTION move_file(fromname VARCHAR, toname VARCHAR) RETURNS void AS $$
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
```

```
    PERFORM UTL_FILE.FRENAME('/home/fsep', fromname, '/home/backup', toname, FALSE); ...(8)
    RETURN;

EXCEPTION
    WHEN OTHERS THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('-- SQL Error --');
        PERFORM DBMS_OUTPUT.PUT_LINE('ERROR : ' || SQLERRM );
        RETURN;
END;
$$
LANGUAGE plpgsql;


SELECT move_file('file01.txt','file02.txt');
```

(8) FRENAME

Same as FCOPY for the UTL_FILE package. Refer to FCOPY in the UTL_FILE package for information on specification differences and conversion procedures associated with specification differences.

B.7 DBMS_SQL (Execute Dynamic SQL)

Features

For DBMS_SQL, dynamic SQL can be executed from PL/pgSQL.

B.7.1 Searching Using a Cursor

Oracle database

```
CREATE PROCEDURE search_test(h_where CLOB) AS


    str_sql     CLOB;
    v_cnt       INTEGER;
    v_array     DBMS_SQL.VARCHAR2A;
    v_cur       INTEGER;
    v_smpid     INTEGER;
    v_smpnm     VARCHAR2(20);
    v_addbuff   VARCHAR2(20);
    v_smpage    INTEGER;
    errcd       INTEGER;
    length      INTEGER;
    ret         INTEGER;
BEGIN

   str_sql    := 'SELECT smpid, smpnm FROM smp_tbl WHERE ' || h_where || ' ORDER BY smpid';
    v_smpid     := 0;
    v_smpnm     := '';
    v_smpage    := 0;

    v_cur := DBMS_SQL.OPEN_CURSOR; ...(1)

    v_cnt :=
      CEIL(DBMS_LOB.GETLENGTH(str_sql)/1000);
    FOR idx IN 1 .. v_cnt LOOP
        v_array(idx) :=
            DBMS_LOB.SUBSTR(str_sql,
                            1000,
                            (idx-1)*1000+1);
```

```
    END LOOP;
    DBMS_SQL.PARSE(v_cur, v_array, 1, v_cnt, FALSE, DBMS_SQL.NATIVE); ...(2)

    DBMS_SQL.DEFINE_COLUMN(v_cur, 1, v_smpid);

    DBMS_SQL.DEFINE_COLUMN(v_cur, 2, v_smpnm, 10);


    ret := DBMS_SQL.EXECUTE(v_cur);
    LOOP
        v_addbuff := '';

        IF DBMS_SQL.FETCH_ROWS(v_cur) = 0 THEN
            EXIT;
        END IF;

        DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------------');
        DBMS_SQL.COLUMN_VALUE(v_cur, 1, v_smpid, errcd, length); ...(3)




        IF errcd = 1405 THEN ...(3)

          DBMS_OUTPUT.PUT_LINE('smpid       = (NULL)');
        ELSE
          DBMS_OUTPUT.PUT_LINE('smpid       = ' || v_smpid);
        END IF;

        DBMS_SQL.COLUMN_VALUE(v_cur, 2, v_smpnm, errcd, length);

        IF errcd = 1406 THEN
          v_addbuff := '... [len=' || length || ']';
        END IF;
        IF errcd = 1405 THEN
          DBMS_OUTPUT.PUT_LINE('v_smpnm     = (NULL)');
        ELSE
          DBMS_OUTPUT.PUT_LINE('v_smpnm     = ' || v_smpnm || v_addbuff );
        END IF;

DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------------');

        DBMS_OUTPUT.NEW_LINE;
    END LOOP;

    DBMS_SQL.CLOSE_CURSOR(v_cur); ...(4)

    RETURN;
END;
/

Set serveroutput on

call search_test('smpid < 100');
```

Enterprise Postgres

```
CREATE FUNCTION search_test(h_where text) RETURNS void AS $$
DECLARE
    str_sql     text;


    v_cur       INTEGER;
```

```
    v_smpid     INTEGER;
    v_smpnm     VARCHAR(20);
    v_addbuff   VARCHAR(20);
    v_smpage    INTEGER;
    errcd       INTEGER;
    length      INTEGER;
    ret         INTEGER;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    str_sql     := 'SELECT smpid, smpnm FROM smp_tbl WHERE ' || h_where || ' ORDER BY smpid';
    v_smpid     := 0;
    v_smpnm     := '';
    v_smpage    := 0;

    v_cur := DBMS_SQL.OPEN_CURSOR(); ...(1)

    PERFORM DBMS_SQL.PARSE(v_cur, str_sql, 1); ...(2)
    PERFORM DBMS_SQL.DEFINE_COLUMN(v_cur, 1, v_smpid);
    PERFORM DBMS_SQL.DEFINE_COLUMN(v_cur, 2, v_smpnm, 10);

    ret := DBMS_SQL.EXECUTE(v_cur);
    LOOP
        v_addbuff := '';

        IF DBMS_SQL.FETCH_ROWS(v_cur) = 0 THEN
            EXIT;
        END IF;

        PERFORM
DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------------');
        SELECT value,column_error,actual_length
          INTO v_smpid, errcd, length
          FROM DBMS_SQL.COLUMN_VALUE(v_cur,
                                     1,
                                     v_smpid); ...(3)
        IF errcd = 22002 THEN ...(3)
          PERFORM DBMS_OUTPUT.PUT_LINE('smpid       = (NULL)');
        ELSE
          PERFORM DBMS_OUTPUT.PUT_LINE('smpid       = ' || v_smpid);
        END IF;

        SELECT value,column_error,actual_length INTO v_smpnm, errcd, length FROM
DBMS_SQL.COLUMN_VALUE(v_cur, 2, v_smpnm);
        IF errcd = 22001 THEN
          v_addbuff := '... [len=' || length || ']';
        END IF;
        IF errcd = 22002 THEN
          PERFORM DBMS_OUTPUT.PUT_LINE('v_smpnm     = (NULL)');
        ELSE
          PERFORM DBMS_OUTPUT.PUT_LINE('v_smpnm     = ' || v_smpnm || v_addbuff );
        END IF;

        PERFORM
DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------------');
        PERFORM DBMS_OUTPUT.NEW_LINE();
    END LOOP;

    v_cur := DBMS_SQL.CLOSE_CURSOR(v_cur); ...(4)
    RETURN;
END;
$$
LANGUAGE plpgsql;
```

```
SELECT search_test('smpid < 100');
```

(1) OPEN_CURSOR

Same as NEW_LINE in the DBMS_OUTPUT package. Refer to NEW_LINE in the DBMS_OUTPUT package for information on specification differences and conversion procedures associated with specification differences.

(2) PARSE

Specification format for Oracle database

DBMS_SQL.PARSE(*firstArg, secondArg*, *thirdArg*, *fourthArg*, *fifthArg*)

Feature differences

Oracle database

SQL statements can be specified with string table types (VARCHAR2A type, VARCHAR2S type). Specify this for *secondArg*.

DBMS_SQL.NATIVE, DBMS_SQL.V6, DBMS_SQL.V7 can be specified for processing SQL statements.

Enterprise Postgres

SQL statements cannot be specified with string table types.

DBMS_SQL.NATIVE, DBMS_SQL.V6, DBMS_SQL.V7 cannot be specified for processing SQL statements.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "DBMS_SQL.PARSE" is used in the stored procedure.
2. Check the data type of the SQL statement specified for *secondArg* (v_array in the example).
   - If the data type is either DBMS_SQL.VARCHAR2A type or DBMS_SQL.VARCHAR2S type, then it is a table type specification. Execute step 3 and continue the conversion process.
   - If the data type is neither DBMS_SQL.VARCHAR2A type nor DBMS_SQL.VARCHAR2S type, then it is a string specification. Execute step 7 and continue the conversion process.
3. Check the SQL statement (str_sql in the example) before it was divided into DBMS_SQL.VARCHAR2A type and DBMS_SQL.VARCHAR2S type.
4. Delete the sequence of the processes (processes near FOR idx in the example) where SQL is divided into DBMS_SQL.VARCHAR2A type and DBMS_SQL.VARCHAR2S type.
5. Replace *secondArg* with the SQL statement (str_sql in the example) before it is divided, that was checked in step 2.
6. Delete *thirdArg*, *fourthArg*, and *fifthArg* (v_cnt, FALSE, DBMS_SQL.NATIVE, in the example).
7. If DBMS_SQL.NATIVE, DBMS_SQL.V6, and DBMS_SQL.V7 are specified, then replace *thirdArg* with a numeric literal 1.
   - If either DBMS_SQL.VARCHAR2A type or DBMS_SQL.VARCHAR2S type is used, then *sixthArg* becomes relevant.
   - If neither DBMS_SQL.VARCHAR2A type nor DBMS_SQL.VARCHAR2S type is used, then *thirdArg* becomes relevant.

(3) COLUMN_VALUE

Specification format for Oracle database

DBMS_SQL.COLUMN_VALUE(*firstArg, secondArg*, *thirdArg*, *fourthArg*, *fifthArg*)

Feature differences

Oracle database

The following error codes are returned for column_error.

- 1406: fetched column value was truncated
- 1405: fetched column value is NULL

Enterprise Postgres

The following error codes are returned for column_error.

- 22001: string_data_right_truncation
- 22002: null_value_no_indicator_parameter

Specification differences

Oracle database

Obtained values are received with variables specified for arguments.

Enterprise Postgres

Since obtained values are the search results for DBMS_SQL.COLUMN_VALUE, they are received with variables specified for the INTO clause of the SELECT statement.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "DBMS_SQL.COLUMN_VALUE" is used in the stored procedure.
2. Replace the DBMS_SQL.COLUMN_VALUE location called with a SELECT INTO statement.
   - Check the number of arguments (v_smpid, errcd, and length in the example) specified after *secondArg* (1 in the example) of DBMS_SQL.COLUMN_VALUE.
   - Specify "value", "column_error", and "actual_length" in the select list, according to the number of arguments checked in the previous step (for example, if only *thirdArg* is specified, then specify "value" only.)
   - Specify *thirdArg*, *fourthArg*, and *fifthArg* (v_smpid, errcd, length in the example) configured for DBMS_SQL.COLUMN_VALUE, for the INTO clause.
   - Use DBMS_SQL.COLUMN_VALUE in the FROM clause. Specify *firstArg*, *secondArg*, and *thirdArg* (v_cur, 1, v_smpid, in the example) before modification.
3. If the *fourthArg* (column_error value in the example) is used, then check the location of the target variable (errcd in the example).
4. If a decision process is performed in the location checked, then modify the values used in the decision process as below:
   - 1406 to 22001
   - 1405 to 22002

(4) CLOSE_CURSOR

Specification format for Oracle database

DBMS_SQL.CLOSE_CURSOR(*firstArg*)

Specification differences

Oracle database

After closing, the cursor specified in *firstArg* becomes NULL.

Enterprise Postgres

After closing, set the cursor to NULL by assigning the return value of DBMS_SQL.CLOSE_CURSOR to it.

Conversion procedure

Convert using the following procedure:

1. Locate the places where the keyword "DBMS_SQL.CLOSE_CURSOR" is used in the stored procedure.
2. Set the cursor to NULL by assigning (:=) the return value of DBMS_SQL.CLOSE_CURSOR to it.
   - On the left-hand side, specify the argument (v_cur in the example) specified for DBMS_SQL.CLOSE_CURSOR.
   - Use DBMS_SQL.CLOSE_CURSOR in the right-hand side. For the argument, specify the same value (v_cur in the example) as before modification.

Appendix C Tables Used by the Features Compatible with Oracle Databases

This chapter describes the tables used by the features compatible with Oracle databases.

C.1 UTL_FILE.UTL_FILE_DIR

Register the directory handled by the UTL_FILE package in the UTL_FILE.UTL_FILE_DIR table.

Name	Type	Description
dir	text	Name of the directory handled by the UTL_FILE package

Appendix D ECOBPG - Embedded SQL in COBOL

This appendix describes application development using embedded SQL in COBOL.

D.1 Precautions when Using Functions and Operators

An embedded SQL program consists of code written in an ordinary programming language, in this case COBOL, mixed with SQL commands in specially marked sections. To build the program, the source code (*.pco) is first passed through the embedded SQL preprocessor, which converts it to an ordinary COBOL program (*.cob), and afterwards it can be processed by a COBOL compiler. (For details about the compiling and linking see " D.9 Processing Embedded SQL Programs"). Converted ECOBPG applications call functions in the libpq library through the embedded SQL library (ecpglib), and communicate with the PostgreSQL server using the normal frontend-backend protocol.

Embedded SQL has advantages over other methods for handling SQL commands from COOBL code. First, it takes care of the tedious passing of information to and from variables in your C program. Second, the SQL code in the program is checked at build time for syntactical correctness. Third, embedded SQL in COBOL is specified in the SQL standard and supported by many other SQL database systems. The PostgreSQL implementation is designed to match this standard as much as possible, and it is usually possible to port embedded SQL programs written for other SQL databases to PostgreSQL with relative ease.

As already stated, programs written for the embedded SQL interface are normal COBOL programs with special code inserted to perform database-related actions. This special code always has the form:

```
EXEC SQL ... END-EXEC
```

These statements syntactically take the place of a COBOL statement. Depending on the particular statement, they can appear at the data division or at the procedure division. Actual executable SQLs need to be placed at the procedure division, and host variable declarations need to be placed at data division. However, the precompiler does not validate their placements. Embedded SQL statements follow the case-sensitivity rules of normal SQL code, and not those of COBOL.

The precompiler introduces fixed syntax for embedded SQL in COBOL. In each line, 1st column to 6th column constitute line number area, and 7th column do indicator area. Embedded SQL programs also should be placed on the area B(12-72 column).

Note that sample codes in this document ommitt indents for each area.

ECOBPG processes and outputs programs compliance for fixed syntax. However, there are a few restrictions for using ecobps as follows.

- Ecobpg does not validate the limitation of number of characters other than embedded SQLs. 73 and later columns are deleted in the precompiled source.

ECOBPG accepts generally possible COBOL statement. However, there are a few restrictions for using ecobps as follows.

- In declaring host variable section, you can't use debug line.
- Outside of declaring host variable section, you can use debug line, but you can't contain any SQL in debug lines.
- In declaring host variable section, you can't use commas or semicolons as separator. Use space instead.
- EXEC SQL VAR command, it can be used in ECPG, is not available in ECOBPG. Use REDEFINE clause of COBOL insetead.

The following sections explain all the embedded SQL statements.

D.2 Managing Database Connections

This section describes how to open, close, and switch database connections.

D.2.1 Connecting to the Database Server

One connects to a database using the following statement:

```
EXEC SQL CONNECT TO target [AS connection-name] [USER user-name] END-EXEC.
```

The target can be specified in the following ways:

- dbname[@hostname][:port]
- tcp:postgresql://hostname[:port][/dbname][?options]
- unix:postgresql://hostname[:port][/dbname][?options]
- an SQL string literal containing one of the above forms
- a reference to a character variable containing one of the above forms (see examples)
- DEFAULT

If you specify the connection target literally (that is, not through a variable reference) and you don't quote the value, then the case-insensitivity rules of normal SQL are applied. In that case you can also double-quote the individual parameters separately as needed. In practice, it is probably less error-prone to use a (single-quoted) string literal or a variable reference. The connection target DEFAULT initiates a connection to the default database under the default user name. No separate user name or connection name can be specified in that case.

There are also different ways to specify the user name:

- username
- username/password
- username IDENTIFIED BY password
- username USING password

As above, the parameters username and password can be an SQL identifier, an SQL string literal, or a reference to a character variable.

The connection-name is used to handle multiple connections in one program. It can be omitted if a program uses only one connection. The most recently opened connection becomes the current connection, which is used by default when an SQL statement is to be executed (see later in this chapter).

Here are some examples of CONNECT statements:

```
EXEC SQL CONNECT TO mydb@sql.mydomain.com END-EXEC.
```

```
EXEC SQL CONNECT TO tcp:postgresql://sql.mydomain.com/mydb AS myconnection USER john END-
EXEC.
```

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 TARGET PIC X(25).
01 USER PIC X(5).
EXEC SQL END DECLARE SECTION END-EXEC.
 ...
MOVE "mydb@sql.mydomain.com" TO TARGET.
MOVE "john" TO USER.
EXEC SQL CONNECT TO :TARGET USER :USER END-EXEC.
```

The last form makes use of the variant referred to above as character variable reference. For this purpose, only fixed-length string(no VARYING) variable can be used. Trailing spaces are ignored. You will see in later sections how COBOL variables can be used in SQL statements when you prefix them with a colon.

Be advised that the format of the connection target is not specified in the SQL standard. So if you want to develop portable applications, you might want to use something based on the last example above to encapsulate the connection target string somewhere.

D.2.2 Choosing a Connection

SQL statements in embedded SQL programs are by default executed on the current connection, that is, the most recently opened one. If an application needs to manage multiple connections, then there are two ways to handle this.

The first option is to explicitly choose a connection for each SQL statement, for example:

```
EXEC SQL AT connection-name SELECT ... END-EXEC.
```

This option is particularly suitable if the application needs to use several connections in mixed order.

If your application uses multiple threads of execution, they cannot share a connection concurrently. You must either explicitly control access to the connection (using mutexes) or use a connection for each thread. If each thread uses its own connection, you will need to use the AT clause to specify which connection the thread will use.

The second option is to execute a statement to switch the current connection. That statement is:

```
EXEC SQL SET CONNECTION connection-name END-EXEC.
```

This option is particularly convenient if many statements are to be executed on the same connection. It is not thread-aware.

Here is an example program managing multiple database connections:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 DBNAME PIC X(7).
EXEC SQL END DECLARE SECTION END-EXEC.

    EXEC SQL CONNECT TO testdb1 AS con1 USER testuser END-EXEC.
    EXEC SQL CONNECT TO testdb2 AS con2 USER testuser END-EXEC.
    EXEC SQL CONNECT TO testdb3 AS con3 USER testuser END-EXEC.

*   This query would be executed in the last opened database "testdb3".
    EXEC SQL SELECT current_database() INTO :DBNAME END-EXEC.
    DISPLAY "current=" DBNAME " (should be testdb3)".

*   Using "AT" to run a query in "testdb2"
    EXEC SQL AT con2 SELECT current_database() INTO :DBNAME END-EXEC.
    DISPLAY "current=" DBNAME " (should be testdb2)".

*   Switch the current connection to "testdb1".
    EXEC SQL SET CONNECTION con1 END-EXEC.

    EXEC SQL SELECT current_database() INTO :DBNAME END-EXEC.
    DISPLAY "current=" DBNAME " (should be testdb1)".

    EXEC SQL DISCONNECT ALL END-EXEC.
```

This example would produce this output:

```
current=testdb3 (should be testdb3)
current=testdb2 (should be testdb2)
current=testdb1 (should be testdb1)
```

D.2.3 Closing a Connection

To close a connection, use the following statement:

```
EXEC SQL DISCONNECT [connection] END-EXEC.
```

The connection can be specified in the following ways:

- connection-name
- DEFAULT
- CURRENT
- ALL

If no connection name is specified, the current connection is closed.

It is good style that an application always explicitly disconnect from every connection it opened.

D.3 Running SQL Commands

Any SQL command can be run from within an embedded SQL application. Below are some examples of how to do that.

D.3.1 Executing SQL Statements

Creating a table:

```
EXEC SQL CREATE TABLE foo (number integer, ascii char(16)) END-EXEC.
EXEC SQL CREATE UNIQUE INDEX num1 ON foo(number) END-EXEC.
EXEC SQL COMMIT END-EXEC.
```

Inserting rows:

```
EXEC SQL INSERT INTO foo (number, ascii) VALUES (9999, 'doodad') END-EXEC.
EXEC SQL COMMIT END-EXEC.
```

Deleting rows:

```
EXEC SQL DELETE FROM foo WHERE number = 9999 END-EXEC.
EXEC SQL COMMIT END-EXEC.
```

Updates:

```
EXEC SQL UPDATE foo
    SET ascii = 'foobar'
    WHERE number = 9999 END-EXEC.
EXEC SQL COMMIT END-EXEC.
```

SELECT statements that return a single result row can also be executed using EXEC SQL directly. To handle result sets with multiple rows, an application has to use a cursor; see "D.3.2 Using Cursors" below. (As a special case, an application can fetch multiple rows at once into an array host variable; see "Arrays".)

Single-row select:

```
EXEC SQL SELECT foo INTO :FooBar FROM table1 WHERE ascii = 'doodad' END-EXEC.
```

Also, a configuration parameter can be retrieved with the SHOW command:

```
EXEC SQL SHOW search_path INTO :var END-EXEC.
```

The tokens of the form :something are *host variables*, that is, they refer to variables in the COBOL program. They are explained in "D.4 Using Host Variables".

D.3.2 Using Cursors

To retrieve a result set holding multiple rows, an application has to declare a cursor and fetch each row from the cursor. The steps to use a cursor are the following: declare a cursor, open it, fetch a row from the cursor, repeat, and finally close it.

Select using cursors:

```
EXEC SQL DECLARE foo_bar CURSOR FOR
    SELECT number, ascii FROM foo
    ORDER BY ascii END-EXEC.
EXEC SQL OPEN foo_bar END-EXEC.
EXEC SQL FETCH foo_bar INTO :FooBar, :DooDad END-EXEC.
...
EXEC SQL CLOSE foo_bar END-EXEC.
EXEC SQL COMMIT END-EXEC.
```

For more details about declaration of the cursor, see "D.11.4 DECLARE", and refer to "SQL Commands" in "Reference" in the PostgreSQL Documentation for information on FETCH command.

Note: The ECOBPG DECLARE command does not actually cause a statement to be sent to the PostgreSQL backend. The cursor is opened in the backend (using the backend's DECLARE command) at the point when the OPEN command is executed.

D.3.3 Managing Transactions

In the default mode, statements are committed only when EXEC SQL COMMIT is issued. The embedded SQL interface also supports autocommit of transactions (similar to libpq behavior) via the -t command-line option to ecobpg or via the EXEC SQL SET AUTOCOMMIT TO ON statement. In autocommit mode, each command is automatically committed unless it is inside an explicit transaction block. This mode can be explicitly turned off using EXEC SQL SET AUTOCOMMIT TO OFF.

See

Refer to "ecpg" in "PostgreSQL Client Applications" in the PostgreSQL Documentation for information on -t command-line option to ecobpg.

The following transaction management commands are available:

EXEC SQL COMMIT END-EXEC

Commit an in-progress transaction.

EXEC SQL ROLLBACK END-EXEC

Roll back an in-progress transaction.

EXEC SQL SET AUTOCOMMIT TO ON END-EXEC

Enable autocommit mode.

EXEC SQL SET AUTOCOMMIT TO OFF END-EXEC

Disable autocommit mode. This is the default.

D.3.4 Prepared Statements

When the values to be passed to an SQL statement are not known at compile time, or the same statement is going to be used many times, then prepared statements can be useful.

The statement is prepared using the command PREPARE. For the values that are not known yet, use the placeholder "?":

```
EXEC SQL PREPARE stmt1 FROM "SELECT oid, datname FROM pg_database WHERE oid = ?" END-EXEC.
```

If a statement returns a single row, the application can call EXECUTE after PREPARE to execute the statement, supplying the actual values for the placeholders with a USING clause:

```
EXEC SQL EXECUTE stmt1 INTO :dboid, :dbname USING 1 END-EXEC.
```

If a statement returns multiple rows, the application can use a cursor declared based on the prepared statement. To bind input parameters, the cursor must be opened with a USING clause:

```
EXEC SQL PREPARE stmt1 FROM "SELECT oid,datname FROM pg_database WHERE oid > ?" END-EXEC.
EXEC SQL DECLARE foo_bar CURSOR FOR stmt1 END-EXEC.

* when end of result set reached, break out of while loop
EXEC SQL WHENEVER NOT FOUND GOTO FETCH-END END-EXEC.

EXEC SQL OPEN foo_bar USING 100 END-EXEC.
...
PERFORM NO LIMIT
    EXEC SQL FETCH NEXT FROM foo_bar INTO :dboid, :dbname END-EXEC
END-PERFORM.

FETCH-END.
EXEC SQL CLOSE foo_bar END-EXEC.
```

When you don't need the prepared statement anymore, you should deallocate it:

```
EXEC SQL DEALLOCATE PREPARE name END-EXEC.
```

For more details about PREPARE, see "D.11.10 PREPARE". Also see "D.5 Dynamic SQL"for more details about using placeholders and input parameters.

D.4 Using Host Variables

In "D.3 Running SQL Commands" you saw how you can execute SQL statements from an embedded SQL program. Some of those statements only used fixed values and did not provide a way to insert user-supplied values into statements or have the program process the values returned by the query. Those kinds of statements are not really useful in real applications. This section explains in detail how you can pass data between your COBOL program and the embedded SQL statements using a simple mechanism called host variables. In an embedded SQL program we consider the SQL statements to be guests in the COBOL program code which is the host language. Therefore the variables of the COBOL program are called host variables.

Another way to exchange values between PostgreSQL backends and ECOBPG applications is the use of SQL descriptors, described in "D.6 Using Descriptor Areas".

D.4.1 Overview

Passing data between the COBOL program and the SQL statements is particularly simple in embedded SQL. Instead of having the program paste the data into the statement, which entails various complications, such as properly quoting the value, you can simply write the name of a COBOL variable into the SQL statement, prefixed by a colon. For example:

```
EXEC SQL INSERT INTO sometable VALUES (:v1, 'foo', :v2) END-EXEC.
```

This statements refers to two COBOL variables named v1 and v2 and also uses a regular SQL string literal, to illustrate that you are not restricted to use one kind of data or the other.

This style of inserting COBOL variables in SQL statements works anywhere a value expression is expected in an SQL statement.

D.4.2 Declare Sections

To pass data from the program to the database, for example as parameters in a query, or to pass data from the database back to the program, the COBOL variables that are intended to contain this data need to be declared in specially marked sections, so the embedded SQL preprocessor is made aware of them.

This section starts with:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
```

and ends with:

```
EXEC SQL END DECLARE SECTION END-EXEC.
```

Between those lines, there must be normal COBOL variable declarations, such as:

```
01 INTX PIC S9(9) COMP VALUE 4.
01 FOO PIC X(15).
01 BAR PIC X(15).
```

As you can see, you can optionally assign an initial value to the variable. The variable's scope is determined by the location of its declaring section within the program.

You can have as many declare sections in a program as you like.

The declarations are also echoed to the output file as normal COBOL variables, so there's no need to declare them again. Variables that are not intended to be used in SQL commands can be declared normally outside these special sections.

The definition of a group item also must be listed inside a DECLARE section. Otherwise the preprocessor cannot handle these types since it does not know the definition.

D.4.3 Retrieving Query Results

Now you should be able to pass data generated by your program into an SQL command. But how do you retrieve the results of a query? For that purpose, embedded SQL provides special variants of the usual commands SELECT and FETCH. These commands have a special INTO clause that specifies which host variables the retrieved values are to be stored in. SELECT is used for a query that returns only single row, and FETCH is used for a query that returns multiple rows, using a cursor.

Here is an example:

```
*
* assume this table:
* CREATE TABLE test (a int, b varchar(50));
*

 EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01 V1 PIC S9(9).
 01 V2 PIC X(50) VARYING.
 EXEC SQL END DECLARE SECTION END-EXEC.

 ...

 EXEC SQL SELECT a, b INTO :V1, :V2 FROM test END-EXEC.
```

So the INTO clause appears between the select list and the FROM clause. The number of elements in the select list and the list after INTO (also called the target list) must be equal.

Here is an example using the command FETCH:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 V1 PIC S9(9).
01 V2 PIC X(50) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

 ...

EXEC SQL DECLARE foo CURSOR FOR SELECT a, b FROM test END-EXEC.

 ...

PERFORM WITH
```

```
    ...
    EXEC SQL FETCH NEXT FROM foo INTO :V1, :V2 END-EXEC
    ...
END-PERFORM.
```

Here the INTO clause appears after all the normal clauses.

D.4.4 Type Mapping

When ECOBPG applications exchange values between the PostgreSQL server and the COBOL application, such as when retrieving query results from the server or executing SQL statements with input parameters, the values need to be converted between PostgreSQL data types and host language variable types (COBOL language data types, concretely). One of the main points of ECOBPG is that it takes care of this automatically in most cases.

In this respect, there are two kinds of data types: Some simple PostgreSQL data types, such as integer and text, can be read and written by the application directly. Other PostgreSQL data types, such as timestamp and date can only be accessed through character strings. special library functions does not exist in ecobpg. (pgtypes, exists in ECPG, for COBOL is not implemented yet)

"Table D.1 Mapping Between PostgreSQL Data Types and COBOL Variable Types" shows which PostgreSQL data types correspond to which COBOL data types. When you wish to send or receive a value of a given PostgreSQL data type, you should declare a COBOL variable of the corresponding COBOL data type in the declare section.

Table D.1 Mapping Between PostgreSQL Data Types and COBOL Variable Types

PostgreSQL data type	COBOL Host variable type
smallint	PIC S9([1-4]) {BINARY|COMP|COMP-5}
integer	PIC S9([5-9]) {BINARY|COMP|COMP-5}
bigint	PIC S9([10-18]) {BINARY|COMP|COMP-5}
decimal	PIC S9(m)V9(n) PACKED-DECIMAL PIC 9(m)V9(n) DISPLAY (*1) PIC S9(m)V9(n) DISPLAY PIC S9(m)V9(n) DISPLAY SIGN TRAILING [SEPARATE] PIC S9(m)V9(n) DISPLAY SIGN LEADING [SEPARATE
numeric	(same with decimal)
real	COMP-1
double precision	COMP-2
small serial	PIC S9([1-4]) {BINARY|COMP|COMP-5}
serial	PIC S9([1-9]) {BINARY|COMP|COMP-5}
bigserial	PIC S9([10-18]) {BINARY|COMP|COMP-5}
oid	PIC 9(9) {BINARY|COMP|COMP-5}
character(n), varchar(n), text	PIC X(n), PIC X(n) VARYING
name	PIC X(NAMEDATALEN)
boolean	BOOL(*2)
other types(e.g. timestamp)	PIC X(n), PIC X(n) VARYING

*1: If no USAGE is specified, host variable is regarded as DISPLAY.
*2: Type definition is added automatically on pre-compiling.
Body of BOOL is PIC X(1). '1' for true and '0' for false.

You can use some pattern of digits for integer(see table), but if database sends big number with more digits than specified, behavior is undefined .

PostgreSQL data type	COBOL Host variable type
VALUE clause can't be used with VARYING. (Can be used with other types) REDEFINE clause can be used, but it won't be validated on pre-compilation (Your COBOL compiler will do	

Handling Character Strings

To handle SQL character string data types, such as varchar and text, there is a possible way to declare the host variables.

The way is using the PIC X(n) VARYING type (we call it VARCHAR type from now on), which is a special type provided by ECOBPG. The definition on type VARCHAR is converted into a group item consists of named variables. A declaration like:

```
01 VAR PIC X(180) VARYING.
```

is converted into:

```
01 VAR.
49 LEN PIC S9(4) COMP-5.
49 ARR PIC X(180).
```

if --varchar-with-named-member option is used, it is converted into:

```
01 VAR.
49 VAR-LEN PIC S9(4) COMP-5.
49 VAR-ARR PIC X(180).
```

You can use level 1 to 48 for VARCHAR. Don't use level 49 variable right after VARCHAR variable. To use a VARCHAR host variable as an input for SQL statement, LEN must be set the length of the string included in ARR.

To use a VARCHAR host variable as an ouput of SQL statement, the variable must be declared in a sufficient length. if the length is insufficient, it can cause a buffer overrun.

PIC X(n) and VARCHAR host variables can also hold values of other SQL types, which will be stored in their string forms.

Accessing Special Data Types

ECOBPG doesn't have special support for date, timestamp, and interval types.
(ECPG has pgtypes, but ECOBPG doesn't.)
You can use PIC X(n) or VARCHAR for DB I/O with these types. See "Data Types" section in PostgreSQL's document.

Host Variables with Nonprimitive Types

As a host variable you can also use arrays, typedefs, and group items.

Arrays

To create and use array variables, OCCURENCE syntax is provided by COBOL.

The typical use case is to retrieve multiple rows from a query result without using a cursor. Without an array, to process a query result consisting of multiple rows, it is required to use a cursor and the FETCH command. But with array host variables, multiple rows can be received at once. The length of the array has to be defined to be able to accommodate all rows, otherwise a buffer overrun will likely occur.

Following example scans the pg_database system table and shows all OIDs and names of the available databases:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 GROUP-ITEM.
    05 DBID PIC S9(9) COMP OCCURS 8.
    05 DBNAME PIC X(16) OCCURS 8.
01 I PIC S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC.


    EXEC SQL CONNECT TO testdb END-EXEC.
```

```
*   Retrieve multiple rows into arrays at once.
    EXEC SQL SELECT oid,datname INTO :DBID, :DBNAME FROM pg_database END-EXEC.

    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 8
        DISPLAY "oid=" DBID(I) ", dbname=" DBNAME(I)
    END-PERFORM.

    EXEC SQL COMMIT END-EXEC.
    EXEC SQL DISCONNECT ALL END-EXEC.
```

You can use member of array as simple host variable by specifying subscript of array. For specifying subscript, use C-style "[1]", not COBOL-style "(1)". But subscript starts with 1, according to COBOL syntax.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 GROUP-ITEM.
    05 DBID PIC S9(9) COMP OCCURS 8.
EXEC SQL END DECLARE SECTION END-EXEC.

    EXEC SQL CONNECT TO testdb END-EXEC.

    EXEC SQL SELECT oid INTO :DBID[1] FROM pg_database WHERE oid=1 END-EXEC.

        DISPLAY "oid=" DBID(1)

    EXEC SQL COMMIT END-EXEC.
    EXEC SQL DISCONNECT ALL END-EXEC.
```

Group Item

A group item whose subordinate item names match the column names of a query result, can be used to retrieve multiple columns at once. The group item enables handling multiple column values in a single host variable.

The following example retrieves OIDs, names, and sizes of the available databases from the pg_database system table and using the pg_database_size() function. In this example, a group item variable dbinfo_t with members whose names match each column in the SELECT result is used to retrieve one result row without putting multiple host variables in the FETCH statement.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 DBINFO-T TYPEDEF.
       02 OID PIC S9(9) COMP.
       02 DATNAME PIC X(65).
       02 DBSIZE PIC S9(18) COMP.

    01 DBVAL TYPE DBINFO-T.
EXEC SQL END DECLARE SECTION END-EXEC.


    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size
FROM pg_database END-EXEC.
    EXEC SQL OPEN cur1 END-EXEC.

*   when end of result set reached, break out of loop
    EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

    PERFORM NO LIMIT
*        Fetch multiple columns into one structure.
        EXEC SQL FETCH FROM cur1 INTO :DBVAL END-EXEC

*       Print members of the structure.
        DISPLAY "oid=" OID ", datname=" DATNAME ", size=" DBSIZE
    END-PERFORM.
```

```
    END-FETCH.
    EXEC SQL CLOSE cur1 END-EXEC.
```

group item host variables "absorb" as many columns as the group item as subordinate items. Additional columns can be assigned to other host variables. For example, the above program could also be restructured like this, with the size variable outside the group item:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 DBINFO-T TYPEDEF.
       02 OID PIC S9(9) COMP.
       02 DATNAME PIC X(65).

    01 DBVAL TYPE DBINFO-T.
    01 DBSIZE PIC S9(18) COMP.
EXEC SQL END DECLARE SECTION END-EXEC.


    EXEC SQL DECLARE cur1 CURSOR FOR SELECT oid, datname, pg_database_size(oid) AS size
FROM pg_database END-EXEC.
    EXEC SQL OPEN cur1 END-EXEC.

*   when end of result set reached, break out of loop
    EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

    PERFORM NO LIMIT
*       Fetch multiple columns into one structure.
        EXEC SQL FETCH FROM cur1 INTO :DBVAL, :DBSIZE END-EXEC

*       Print members of the structure.
        DISPLAY "oid=" OID ", datname=" DATNAME ", size=" DBSIZE
    END-PERFORM

    FETCH-END.
    EXEC SQL CLOSE cur1 END-EXEC.
```

You can use only non-nested group items for host variable of SQL statement. Declaration of nested group items are OK, but you must specify non-nested part of group items for SQL. (VARCHAR, is translated to group item on pre-compilation, is not considered as offense of this rule.) When using inner item of group item in SQL, use C-struct like period separated syntax(not COBOL's A OF B). Here is example.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 NESTED-GROUP.
  02 CHILD1.
    03 A PIC X(10).
    03 B PIC S9(9) COMP.
  02 CHILD2.
    03 A PIC X(10).
    03 B PIC S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC.

* This SQL is valid. CHILD1 has no nested group items.
EXEC SQL SELECT * INTO :NESTED-GROUP.CHILD1 FROM TABLE1 END-EXEC.
```

For specifying basic item of group items, full specifucation is not needed if the specification is enough for identifing the item. This is from COBOL syntax.For more detail, see resources of COBOL syntax.

TYPEDEF

Use the typedef keyword to map new types to already existing types.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 MYCHARTYPE TYPEDEF PIC X(40).
    01 SERIAL-T TYPEDEF PIC S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC.
```

Note that you could also use:

```
EXEC SQL TYPE SERIAL-T IS PIC S9(9) COMP-5. END-EXEC.
```

This declaration does not need to be part of a declare section.

D.4.5 Handling Nonprimitive SQL Data Types

This section contains information on how to handle nonscalar and user-defined SQL-level data types in ECOBPG applications. Note that this is distinct from the handling of host variables of nonprimitive types, described in the previous section.

Arrays

SQL-level arrays are not directly supported in ECOBPG. It is not possible to simply map an SQL array into a COBOL array host variable. This will result in undefined behavior. Some workarounds exist, however.

If a query accesses elements of an array separately, then this avoids the use of arrays in ECOBPG. Then, a host variable with a type that can be mapped to the element type should be used. For example, if a column type is array of integer, a host variable of type PIC S9(9) COMP can be used. Also if the element type is varchar or text, a host variable of type VARCHAR can be used.

Here is an example. Assume the following table:

```
CREATE TABLE t3 (
    ii integer[]
);

testdb=> SELECT * FROM t3;
     ii
-------------
 {1,2,3,4,5}
(1 row)
```

The following example program retrieves the 4th element of the array and stores it into a host variable of type PIC S9(9) COMP-5:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 II PIC S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC.


EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[4] FROM t3 END-EXEC.
EXEC SQL OPEN cur1 END-EXEC.


EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.


PERFORM NO LIMIT
    EXEC SQL FETCH FROM cur1 INTO :II  END-EXEC
    DISPLAY "ii=" II
END-PERFORM.
```

```
END-FETCH.
EXEC SQL CLOSE cur1 END-EXEC.
```

To map multiple array elements to the multiple elements in an array type host variables each element of array column and each element of the host variable array have to be managed separately, for example:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 GROUP-ITEM.
    05 II_A PIC S9(9) COMP OCCURS 8.
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii[1], ii[2], ii[3], ii[4] FROM t3 END-EXEC.
EXEC SQL OPEN cur1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

PERFORM NO LIMIT
    EXEC SQL FETCH FROM cur1 INTO :II_A[1], :II_A[2], :II_A[3], :II_A[4] END-EXEC
    ...
END-PERFORM.
```

Note again that.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 GROUP-ITEM.
    05 II_A PIC S9(9) COMP OCCURS 8.
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL DECLARE cur1 CURSOR FOR SELECT ii FROM t3 END-EXEC.
EXEC SQL OPEN cur1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

PERFORM NO LIMIT
*   WRONG
    EXEC SQL FETCH FROM cur1 INTO :II_A END-EXEC
    ...
END-PERFORM.
```

would not work correctly in this case, because you cannot map an array type column to an array host variable directly.

Another workaround is to store arrays in their external string representation in host variables of type VARCHAR. For more details about this representation.

Refer to "Arrays" in "Tutorial" in the PostgreSQL Documentation for information more details about this representation.

Note that this means that the array cannot be accessed naturally as an array in the host program (without further processing that parses the text representation).

Composite Types

Composite types are not directly supported in ECOBPG, but an easy workaround is possible. The available workarounds are similar to the ones described for arrays above: Either access each attribute separately or use the external string representation.

For the following examples, assume the following type and table:

```
CREATE TYPE comp_t AS (intval integer, textval varchar(32));
CREATE TABLE t4 (compval comp_t);
INSERT INTO t4 VALUES ( (256, 'PostgreSQL') );
```

The most obvious solution is to access each attribute separately. The following program retrieves data from the example table by selecting each attribute of the type comp_t separately:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 INTVAL PIC S9(9) COMP.
01 TEXTVAL PIC X(33) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

* Put each element of the composite type column in the SELECT list.
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4 END-
EXEC.
EXEC SQL OPEN cur1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

PERFORM NO LIMIT
*   Fetch each element of the composite type column into host variables.
    EXEC SQL FETCH FROM cur1 INTO :INTVAL, :TEXTVAL END-EXEC

    DISPLAY "intval=" INTVAL ", textval=" ARR OF TEXTVAL
END-PERFORM.

END-FETCH.
EXEC SQL CLOSE cur1 END-EXEC.
```

To enhance this example, the host variables to store values in the FETCH command can be gathered into one group item. For more details about the host variable in the group item form, see "Group Item". To switch to the group item, the example can be modified as below. The two host variables, intval and textval, become subordinate items of the comp_t group item, and the group item is specified on the FETCH command.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 COMP-T TYPEDEF.
    02 INTVAL PIC S9(9) COMP.
    02 TEXTVAL PIC X(33) VARYING.

01 COMPVAL TYPE COMP-T.
EXEC SQL END DECLARE SECTION END-EXEC.

* Put each element of the composite type column in the SELECT list.
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).intval, (compval).textval FROM t4 END-
EXEC.
EXEC SQL OPEN cur1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

PERFORM NO LIMIT
*   Put all values in the SELECT list into one structure.
    EXEC SQL FETCH FROM cur1 INTO :COMPVAL END-EXEC

    DISPLAY "intval=" INTVAL ", textval=" ARR OF TEXTVAL
END-PERFORM.

END-FETCH.
EXEC SQL CLOSE cur1 END-EXEC.
```

Although a group item is used in the FETCH command, the attribute names in the SELECT clause are specified one by one. This can be enhanced by using a * to ask for all attributes of the composite type value.

```
...
EXEC SQL DECLARE cur1 CURSOR FOR SELECT (compval).* FROM t4 END-EXEC.
```

```
EXEC SQL OPEN cur1 END-EXEC.
EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

PERFORM NO LIMIT
*   Put all values in the SELECT list into one structure.
    EXEC SQL FETCH FROM cur1 INTO :COMPVAL END-EXEC

    DISPLAY "intval=" INTVAL ", textval=" ARR OF TEXTVAL
END-PERFORM.
```

This way, composite types can be mapped into structures almost seamlessly, even though ECOBPG does not understand the composite type itself.

Finally, it is also possible to store composite type values in their external string representation in host variables of type VARCHAR. But that way, it is not easily possible to access the fields of the value from the host program.

User-defined Base Types

New user-defined base types are not directly supported by ECOBPG. You can use the external string representation and host variables of type VARCHAR, and this solution is indeed appropriate and sufficient for many types.

Here is an example using the data type complex.

See

Refer to "User-defined Types" in "Server Programming" in the PostgreSQL Documentation for information on the data type complex..

The external string representation of that type is (%lf,%lf), which is defined in the functions complex_in() and complex_out() functions. The following example inserts the complex type values (1,1) and (3,3) into the columns a and b, and select them from the table after that.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 A PIC X(64) VARYING.
    01 B PIC X(64) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL INSERT INTO test_complex VALUES ('(1,1)', '(3,3)') END-EXEC.

EXEC SQL DECLARE cur1 CURSOR FOR SELECT a, b FROM test_complex END-EXEC.
EXEC SQL OPEN cur1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO END-FETCH END-EXEC.

PERFORM NO LIMIT
    EXEC SQL FETCH FROM cur1 INTO :A, :B END-EXEC
    DISPLAY "a=" ARR OF A ", b=" ARR OF B
END-PERFORM.

END-FETCH.
EXEC SQL CLOSE cur1 END-EXEC.
```

Another workaround is avoiding the direct use of the user-defined types in ECOBPG and instead create a function or cast that converts between the user-defined type and a primitive type that ECOBPG can handle. Note, however, that type casts, especially implicit ones, should be introduced into the type system very carefully.

For example:

```
CREATE FUNCTION create_complex(r double precision, i double precision) RETURNS complex
LANGUAGE SQL
IMMUTABLE
AS $$ SELECT $1 * complex '(1,0)' + $2 * complex '(0,1)' $$;
```

After this definition, the following:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 A COMP-2.
01 B COMP-2.
01 C COMP-2.
01 D COMP-2.
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE 1 TO A.
MOVE 2 TO B.
MOVE 3 TO C.
MOVE 4 TO D.

EXEC SQL INSERT INTO test_complex VALUES (create_complex(:A, :B), create_complex(:C, :D))
END-EXEC.
```

has the same effect as

```
EXEC SQL INSERT INTO test_complex VALUES ('(1,2)', '(3,4)') END-EXEC.
```

D.4.6 Indicators

The examples above do not handle null values. In fact, the retrieval examples will raise an error if they fetch a null value from the database. To be able to pass null values to the database or retrieve null values from the database, you need to append a second host variable specification to each host variable that contains data. This second host variable is called the *indicator* and contains a flag that tells whether the datum is null, in which case the value of the real host variable is ignored. Here is an example that handles the retrieval of null values correctly:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 VAL PIC X(50) VARYING.
01 VAL_IND PIC S9(9) COMP-5.
EXEC SQL END DECLARE SECTION END-EXEC.

 ...

EXEC SQL SELECT b INTO :VAL :VAL_IND FROM test1 END-EXEC.
```

The indicator variable val_ind will be zero if the value was not null, and it will be negative if the value was null.

The indicator has another function: if the indicator value is positive, it means that the value is not null, but it was truncated when it was stored in the host variable.

D.5 Dynamic SQL

In many cases, the particular SQL statements that an application has to execute are known at the time the application is written. In some cases, however, the SQL statements are composed at run time or provided by an external source. In these cases you cannot embed the SQL statements directly into the COBOL source code, but there is a facility that allows you to call arbitrary SQL statements that you provide in a string variable.

D.5.1 Executing Statements without a Result Set

The simplest way to execute an arbitrary SQL statement is to use the command EXECUTE IMMEDIATE. For example:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STMT PIC X(30) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "CREATE TABLE test1 (...);" TO ARR OF STMT.
COMPUTE LEN OF STMT = FUNCTION STORED-CHAR-LENGTH (ARR OF STMT).
EXEC SQL EXECUTE IMMEDIATE :STMT END-EXEC.
```

EXECUTE IMMEDIATE can be used for SQL statements that do not return a result set (e.g., DDL, INSERT, UPDATE, DELETE). You cannot execute statements that retrieve data (e.g., SELECT) this way. The next section describes how to do that.

D.5.2 Executing a Statement with Input Parameters

A more powerful way to execute arbitrary SQL statements is to prepare them once and execute the prepared statement as often as you like. It is also possible to prepare a generalized version of a statement and then execute specific versions of it by substituting parameters. When preparing the statement, write question marks where you want to substitute parameters later. For example:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STMT PIC X(40) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "INSERT INTO test1 VALUES(?, ?);" TO ARR OF STMT.
COMPUTE LEN OF STMT = FUNCTION STORED-CHAR-LENGTH (ARR OF STMT).
EXEC SQL PREPARE MYSTMT FROM :STMT END-EXEC.
 ...
EXEC SQL EXECUTE MYSTMT USING 42, 'foobar' END-EXEC.
```

When you don't need the prepared statement anymore, you should deallocate it:

```
EXEC SQL DEALLOCATE PREPARE name END-EXEC.
```

D.5.3 Executing a Statement with a Result Set

To execute an SQL statement with a single result row, EXECUTE can be used. To save the result, add an INTO clause.

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 STMT PIC X(50) VARYING.
01 V1 PIC S9(9) COMP.
01 V2 PIC S9(9) COMP.
01 V3 PIC X(50) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "SELECT a, b, c FROM test1 WHERE a > ?" TO ARR OF STMT.
COMPUTE LEN OF STMT = FUNCTION STORED-CHAR-LENGTH (ARR OF STMT).
EXEC SQL PREPARE MYSTMT FROM :STMT END-EXEC.
 ...
EXEC SQL EXECUTE MYSTMT INTO :V1, :V2, :V3 USING 37 END-EXEC.
```

An EXECUTE command can have an INTO clause, a USING clause, both, or neither.

If a query is expected to return more than one result row, a cursor should be used, as in the following example. (See "D.3.2 Using Cursors" for more details about the cursor.)

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 DBANAME PIC X(128) VARYING.
01 DATNAME PIC X(128) VARYING.
01 STMT PIC X(200) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "SELECT u.usename as dbaname, d.datname
-             "  FROM pg_database d, pg_user u
-             "  WHERE d.datdba = u.usesysid"
TO ARR OF STMT.
COMPUTE LEN OF STMT = FUNCTION STORED-CHAR-LENGTH (ARR OF STMT).

EXEC SQL CONNECT TO testdb AS con1 USER testuser END-EXEC.

EXEC SQL PREPARE STMT1 FROM :STMT END-EXEC.
```

```
EXEC SQL DECLARE cursor1 CURSOR FOR STMT1 END-EXEC.
EXEC SQL OPEN cursor1 END-EXEC.

EXEC SQL WHENEVER NOT FOUND GOTO FETCH-END END-EXEC.

PERFORM NO LIMIT
    EXEC SQL FETCH cursor1 INTO :DBANAME,:DATNAME END-EXEC
    DISPLAY "dbaname=" ARR OF DBANAME ", datname=" ARR OF DATNAME
END-PERFORM.

FETCH-END.

EXEC SQL CLOSE cursor1 END-EXEC.

EXEC SQL COMMIT END-EXEC.
EXEC SQL DISCONNECT ALL END-EXEC.
```

D.6 Using Descriptor Areas

An SQL descriptor area is a more sophisticated method for processing the result of a SELECT, FETCH or a DESCRIBE statement. An SQL descriptor area groups the data of one row of data together with metadata items into one data group item. The metadata is particularly useful when executing dynamic SQL statements, where the nature of the result columns might not be known ahead of time. PostgreSQL provides a way to use Descriptor Areas: the named SQL Descriptor Areas.

D.6.1 Named SQL Descriptor Areas

A named SQL descriptor area consists of a header, which contains information concerning the entire descriptor, and one or more item descriptor areas, which basically each describe one column in the result row.

Before you can use an SQL descriptor area, you need to allocate one:

```
EXEC SQL ALLOCATE DESCRIPTOR identifier END-EXEC.
```

The identifier serves as the "variable name" of the descriptor area. When you don't need the descriptor anymore, you should deallocate it:

```
EXEC SQL DEALLOCATE DESCRIPTOR identifier END-EXEC.
```

To use a descriptor area, specify it as the storage target in an INTO clause, instead of listing host variables:

```
EXEC SQL FETCH NEXT FROM mycursor INTO SQL DESCRIPTOR mydesc END-EXEC.
```

If the result set is empty, the Descriptor Area will still contain the metadata from the query, i.e. the field names.

For not yet executed prepared queries, the DESCRIBE statement can be used to get the metadata of the result set:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 SQL-STMT PIC X(30) VARYING.
EXEC SQL END DECLARE SECTION END-EXEC.

MOVE "SELECT * FROM table1" TO ARR OF SQL-STMT.
COMPUTE LEN OF SQL-STMT = FUNCTION STORED-CHAR-LENGTH ( ARR OF SQL-STMT ) .
EXEC SQL PREPARE STMT1 FROM :SQL-STMT END-EXEC.
EXEC SQL DESCRIBE STMT1 INTO SQL DESCRIPTOR MYDESC END-EXEC.
```

Before PostgreSQL 9.0, the SQL keyword was optional, so using DESCRIPTOR and SQL DESCRIPTOR produced named SQL Descriptor Areas. Now it is mandatory, omitting the SQL keyword is regarded as the syntax that produces SQLDA Descriptor Areas. However, ecobpg does not support SQLDA and it causes an error.

In DESCRIBE and FETCH statements, the INTO and USING keywords can be used to similarly: they produce the result set and the metadata in a Descriptor Area.

Now how do you get the data out of the descriptor area? You can think of the descriptor area as a group item with named fields. To retrieve the value of a field from the header and store it into a host variable, use the following command:

```
EXEC SQL GET DESCRIPTOR name :hostvar = field END-EXEC.
```

Currently, there is only one header field defined: COUNT, which tells how many item descriptor areas exist (that is, how many columns are contained in the result). The host variable needs to be of an integer type as PIC S9(9) COMP-5. To get a field from the item descriptor area, use the following command:

```
EXEC SQL GET DESCRIPTOR name VALUE num :hostvar = field END-EXEC.
```

num can be a host variable containing an integer as PIC S9(9) COMP-5.
hostvar must be PIC S9(9) COMP-5 if type of the field is integer. Possible fields are:

CARDINALITY (integer)

number of rows in the result set

DATA

actual data item (therefore, the data type of this field depends on the query)

DATETIME_INTERVAL_CODE (integer)

When TYPE is 9, DATETIME_INTERVAL_CODE will have a value of 1 for DATE, 2 for TIME, 3 for TIMESTAMP, 4 for TIME WITH TIME ZONE, or 5 for TIMESTAMP WITH TIME ZONE.

DATETIME_INTERVAL_PRECISION (integer)

not implemented

INDICATOR (integer)

the indicator (indicating a null value or a value truncation)

KEY_MEMBER (integer)

not implemented

LENGTH (integer)

length of the datum in characters

NAME (string)

name of the column

NULLABLE (integer)

not implemented

OCTET_LENGTH (integer)

length of the character representation of the datum in bytes

PRECISION (integer)

precision (for type numeric)

RETURNED_LENGTH (integer)

length of the datum in characters

RETURNED_OCTET_LENGTH (integer)

length of the character representation of the datum in bytes

SCALE (integer)

scale (for type numeric)

TYPE (integer)

numeric code of the data type of the column

In EXECUTE, DECLARE and OPEN statements, the effect of the INTO and USING keywords are different. A Descriptor Area can also be manually built to provide the input parameters for a query or a cursor and USING SQL DESCRIPTOR name is the way to pass the input parameters into a parametrized query. The statement to build a named SQL Descriptor Area is below:

```
EXEC SQL SET DESCRIPTOR name VALUE num field = :hostvar END-EXEC.
```

PostgreSQL supports retrieving more that one record in one FETCH statement and storing the data in host variables in this case assumes that the variable is an array. E.g.:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01 GROUP-ITEM.
    05 IDNUM PIC S9(9) COMP OCCURS 5.
EXEC SQL END DECLARE SECTION END-EXEC.

EXEC SQL FETCH 5 FROM mycursor INTO SQL DESCRIPTOR mydesc END-EXEC.

EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :IDNUM = DATA END-EXEC.
```

D.7 Error Handling

This section describes how you can handle exceptional conditions and warnings in an embedded SQL program. There are two nonexclusive facilities for this.

- Callbacks can be configured to handle warning and error conditions using the WHENEVER command.
- Detailed information about the error or warning can be obtained from the sqlca variable.

D.7.1 Setting Callbacks

One simple method to catch errors and warnings is to set a specific action to be executed whenever a particular condition occurs. In general:

```
EXEC SQL WHENEVER condition action END-EXEC.
```

condition can be one of the following:

SQLERROR

The specified action is called whenever an error occurs during the execution of an SQL statement.

SQLWARNING

The specified action is called whenever a warning occurs during the execution of an SQL statement.

NOT FOUND

The specified action is called whenever an SQL statement retrieves or affects zero rows. (This condition is not an error, but you might be interested in handling it specially.)

action can be one of the following:

CONTINUE

This effectively means that the condition is ignored. This is the default.

GOTO label
GO TO label

Jump to the specified label (using a COBOL goto statement).

SQLPRINT

Print a message to standard error. This is useful for simple programs or during prototyping. The details of the message cannot be configured.

STOP

Call STOP, which will terminate the program.

CALL name usingargs
DO name usingargs

Call the specified functions with the following characters including arguments. Thus, syntaxes (including compiler depending) are able to be placed as well as the arguments. Though, there are some limitation as following:

- You can't use RETURNING, ON EXCEPTION or OVER FLOW clauses.
- In the called subprogram, You must specify CONTINUE for every action with WHENEVER statement.

The SQL standard only provides for the actions CONTINUE and GOTO (and GO TO).

Here is an example that you might want to use in a simple program. It prints a simple message when a warning occurs and aborts the program when an error happens:

```
EXEC SQL WHENEVER SQLWARNING SQLPRINT END-EXEC.
EXEC SQL WHENEVER SQLERROR STOP END-EXEC.
```

The statement EXEC SQL WHENEVER is a directive of the SQL preprocessor, not a COBOL statement. The error or warning actions that it sets apply to all embedded SQL statements that appear below the point where the handler is set, unless a different action was set for the same condition between the first EXEC SQL WHENEVER and the SQL statement causing the condition, regardless of the flow of control in the COBOL program. So neither of the two following COBOL program excerpts will have the desired effect:

```
*
*   WRONG
*
    ...
    IF VERBOSE = 1 THEN
        EXEC SQL WHENEVER SQLWARNING SQLPRINT END-EXEC
    END-IF.
    ...
    EXEC SQL SELECT ... END-EXEC.
    ...
*
*   WRONG
*
    ...
    CALL SET-ERROR-HANDLER.
*        (and execute "EXEC SQL WHENEVER SQLERROR STOP" in SET-ERROR-HANDLER)
    ...
    EXEC SQL SELECT ... END-EXEC.
    ...
```

D.7.2 sqlca

For more powerful error handling, the embedded SQL interface provides a global variable with the name sqlca (SQL communication area) that has the following group item:

```
01 SQLCA.
    02 SQLCAID PIC X(8).
    02 SQLABC PIC S9(9).
    02 SQLCODE PIC S9(9).
    02 SQLERRM.
      03  SQLERRML PIC S9(9).
      03  SQLERRMC PIC X(150).
    02 SQLERRP PIC X(8).
    02 SQLERRD PIC S9(9) OCCURS 6.
```

```
    02 SQLWARN PIC X(8).
    02 SQLSTATE PIC X(5).
```

(In a multithreaded program, every thread automatically gets its own copy of sqlca. This works similarly to the handling of the standard C global variable errno.)

sqlca covers both warnings and errors. If multiple warnings or errors occur during the execution of a statement, then sqlca will only contain information about the last one.

If no error occurred in the last SQL statement, SQLCODE will be 0 and SQLSTATE will be "00000". If a warning or error occurred, then SQLCODE will be negative and SQLSTATE will be different from "00000". A positive SQLCODE indicates a harmless condition, such as that the last query returned zero rows. SQLCODE and SQLSTATE are two different error code schemes; details appear below.

If the last SQL statement was successful, then SQLERRD(2) contains the OID of the processed row, if applicable, and SQLERRD(3) contains the number of processed or returned rows, if applicable to the command.

In case of an error or warning, SQLERRMC will contain a string that describes the error. The field SQLERRML contains the length of the error message that is stored in SQLERRMC (the result of FUNCTION STORED-CHAR-LENGTH. Note that some messages are too long to fit in the fixed-size sqlerrmc array; they will be truncated.

In case of a warning, the 3rd character of SQLWARN is set to W. (In all other cases, it is set to something different from W.) If the 2nd character of SQLWARN is set to W, then a value was truncated when it was stored in a host variable. The 1st character of SQLWARN is set to W if any of the other elements are set to indicate a warning.

The fields sqlcaid, sqlcabc, sqlerrp, and the remaining elements of sqlerrd and sqlwarn currently contain no useful information.

The structure sqlca is not defined in the SQL standard, but is implemented in several other SQL database systems. The definitions are similar at the core, but if you want to write portable applications, then you should investigate the different implementations carefully.

Here is one example that combines the use of WHENEVER and sqlca, printing out the contents of sqlca when an error occurs. This is perhaps useful for debugging or prototyping applications, before installing a more "user-friendly" error handler.

```
EXEC SQL WHENEVER SQLERROR GOTO PRINT_SQLCA END-EXEC.

PRINT_SQLCA.
    DISPLAY "==== sqlca ====".
    DISPLAY "SQLCODE: " SQLCODE.
    DISPLAY "SQLERRML: " SQLERRML.
    DISPLAY "SQLERRMC: " SQLERRMC.
    DISPLAY "SQLERRD: " SQLERRD(1) " " SQLERRD(2) " " SQLERRD(3)" " SQLERRD(4) " "
SQLERRD(5) " " SQLERRD(6).
    DISPLAY "SQLSTATE: " SQLSTATE.
    DISPLAY "===============".
```

The result could look as follows (here an error due to a misspelled table name):

```
==== sqlca ====
sqlcode: -000000400
SQLERRML: +000000064
SQLERRMC: relation "pg_databasep" does not exist (10292) on line 93
sqlerrd: +000000000 +000000000 +000000000 +000000000 +000000000 +000000000
sqlstate: 42P01
===============
```

D.7.3 SQLSTATE vs. SQLCODE

The fields SQLSTATE and SQLCODE are two different schemes that provide error codes. Both are derived from the SQL standard, but SQLCODE has been marked deprecated in the SQL-92 edition of the standard and has been dropped in later editions. Therefore, new applications are strongly encouraged to use SQLSTATE.

SQLSTATE is a five-character array. The five characters contain digits or upper-case letters that represent codes of various error and warning conditions. SQLSTATE has a hierarchical scheme: the first two characters indicate the general class of the condition, the last three characters indicate a subclass of the general condition. A successful state is indicated by the code 00000. The SQLSTATE codes are for the most part defined in the SQL standard. The PostgreSQL server natively supports SQLSTATE error codes; therefore a high degree of consistency can be achieved by using this error code scheme throughout all applications.

See

Refer to "PostgreSQL Error Codes" in "Appendixes" in the PostgreSQL Documentation for further information.

SQLCODE, the deprecated error code scheme, is a simple integer. A value of 0 indicates success, a positive value indicates success with additional information, a negative value indicates an error. The SQL standard only defines the positive value +100, which indicates that the last command returned or affected zero rows, and no specific negative values. Therefore, this scheme can only achieve poor portability and does not have a hierarchical code assignment. Historically, the embedded SQL processor for PostgreSQL has assigned some specific SQLCODE values for its use, which are listed below with their numeric value and their symbolic name. Remember that these are not portable to other SQL implementations. To simplify the porting of applications to the SQLSTATE scheme, the corresponding SQLSTATE is also listed. There is, however, no one-to-one or one-to-many mapping between the two schemes (indeed it is many-to-many), so you should consult the global SQLSTATE in each case.

See

Refer to "PostgreSQL Error Codes" in "Appendixes" in the PostgreSQL Documentation.

These are the assigned SQLCODE values:

0

Indicates no error. (SQLSTATE 00000)

100

This is a harmless condition indicating that the last command retrieved or processed zero rows, or that you are at the end of the cursor. (SQLSTATE 02000)

When processing a cursor in a loop, you could use this code as a way to detect when to abort the loop, like this:

```
PERFORM NO LIMIT
    EXEC SQL FETCH ... END-EXEC
    IF SQLCODE = 100 THEN
        GO TO FETCH-END
    END-IF
END-PERFORM.
```

But WHENEVER NOT FOUND GOTO ... effectively does this internally, so there is usually no advantage in writing this out explicitly.

-12

Indicates that your virtual memory is exhausted. The numeric value is defined as -ENOMEM. (SQLSTATE YE001)

-200

Indicates the preprocessor has generated something that the library does not know about. Perhaps you are running incompatible versions of the preprocessor and the library. (SQLSTATE YE002)

-201

This means that the command specified more host variables than the command expected. (SQLSTATE 07001 or 07002)

-202

This means that the command specified fewer host variables than the command expected. (SQLSTATE 07001 or 07002)

-203

This means a query has returned multiple rows but the statement was only prepared to store one result row (for example, because the specified variables are not arrays). (SQLSTATE 21000)

-204

The host variable is of type singed int and the datum in the database is of a different type and contains a value that cannot be interpreted as an signed int. The library uses strtol() for this conversion. (SQLSTATE 42804)

-205

The host variable is of type unsigned int and the datum in the database is of a different type and contains a value that cannot be interpreted as an unsigned int. The library uses strtoul() for this conversion. (SQLSTATE 42804)

-206

The host variable is of type float and the datum in the database is of another type and contains a value that cannot be interpreted as a float. The library uses strtod() for this conversion. (SQLSTATE 42804)

-207

The host variable is of type DECIMAL and the datum in the database is of another type and contains a value that cannot be interpreted as a DECIMAL or DISPLAY value. For the case of DISPLAY, this error happens if values in the database is too large for converting to DISPLAY value. (SQLSTATE 42804)

-208

The host variable is of type interval and the datum in the database is of another type and contains a value that cannot be interpreted as an interval value. (SQLSTATE 42804)

-209

The host variable is of type date and the datum in the database is of another type and contains a value that cannot be interpreted as a date value. (SQLSTATE 42804)

-210

The host variable is of type timestamp and the datum in the database is of another type and contains a value that cannot be interpreted as a timestamp value. (SQLSTATE 42804)

-211

This means the host variable is of type bool and the datum in the database is neither 't' nor 'f'. (SQLSTATE 42804)

-212

The statement sent to the PostgreSQL server was empty. (This cannot normally happen in an embedded SQL program, so it might point to an internal error.) (SQLSTATE YE002)

-213

A null value was returned and no null indicator variable was supplied. (SQLSTATE 22002)

-214

An ordinary variable was used in a place that requires an array. (SQLSTATE 42804)

-215

The database returned an ordinary variable in a place that requires array value. (SQLSTATE 42804)

-220

The program tried to access a connection that does not exist. (SQLSTATE 08003)

-221

The program tried to access a connection that does exist but is not open. (This is an internal error.) (SQLSTATE YE002)

-230

The statement you are trying to use has not been prepared. (SQLSTATE 26000)

-240

The descriptor specified was not found. The statement you are trying to use has not been prepared. (SQLSTATE 33000)

-241

The descriptor index specified was out of range. (SQLSTATE 07009)

-242

An invalid descriptor item was requested. (This is an internal error.) (SQLSTATE YE002)

-243

During the execution of a dynamic statement, the database returned a numeric value and the host variable was not numeric. (SQLSTATE 07006)

-244

During the execution of a dynamic statement, the database returned a non-numeric value and the host variable was numeric. (SQLSTATE 07006)

-400

Some error caused by the PostgreSQL server. The message contains the error message from the PostgreSQL server.

-401

The PostgreSQL server signaled that we cannot start, commit, or rollback the transaction. (SQLSTATE 08007)

-402

The connection attempt to the database did not succeed. (SQLSTATE 08001)

-403

Duplicate key error, violation of unique constraint. (SQLSTATE 23505)

-404

A result for the subquery is not single row. (SQLSTATE 21000)

-602

An invalid cursor name was specified. (SQLSTATE 34000)

-603

Transaction is in progress. (SQLSTATE 25001)

-604

There is no active (in-progress) transaction. (SQLSTATE 25P01)

-605

An existing cursor name was specified. (SQLSTATE 42P03)

D.8 Preprocessor Directives

Several preprocessor directives are available that modify how the ecobpg preprocessor parses and processes a file.

D.8.1 Including Files

To include an external file into your embedded SQL program, use:

```
EXEC SQL INCLUDE filename END-EXEC.
EXEC SQL INCLUDE <filename> END-EXEC.
EXEC SQL INCLUDE "filename" END-EXEC.
```

The embedded SQL preprocessor will look for a file named filename.pco, preprocess it, and include it in the resulting COBOL output. Thus, embedded SQL statements in the included file are handled correctly.

By default, the ecobpg preprocessor will search a file at the current directory. This behavior can be changed by the ecobpg commandline option.

First, the preprocessor tries to locate a file by specified file name at the current directory. If it fails and the file name does not end with .pco, the preprocessor also tries to locate a file with the suffix at the same directory.

The difference between EXEC SQL INCLUDE and COPY statement is whether precompiler processes embedded SQLs in the file, or not. If the file contains embedded SQLs, use EXEC SQL INCLUDE.

Note

The include file name is case-sensitive, even though the rest of the EXEC SQL INCLUDE command follows the normal SQL case-sensitivity rules.

D.8.2 The define and undef Directives

Similar to the directive #define that is known from C, embedded SQL has a similar concept:

```
EXEC SQL DEFINE name END-EXEC.
EXEC SQL DEFINE name value END-EXEC.
```

So you can define a name:

```
EXEC SQL DEFINE HAVE_FEATURE END-EXEC.
```

And you can also define constants:

```
EXEC SQL DEFINE MYNUMBER 12 END-EXEC.
EXEC SQL DEFINE MYSTRING 'abc' END-EXEC.
```

Use undef to remove a previous definition:

```
EXEC SQL UNDEF MYNUMBER END-EXEC.
```

Note that a constant in the SQL statement is only replaced by EXEC SQL DEFINE. The replacement may change the number of characters in a line, but ecobpg does not validate it after the replacement. Pay attention to the limitation of the number of characters in a line.

D.8.3 ifdef, ifndef, else, elif, and endif Directives

You can use the following directives to compile code sections conditionally:

```
EXEC SQL ifdef name END-EXEC.
```

Checks a name and processes subsequent lines if name has been created with EXEC SQL define name.

```
EXEC SQL ifndef name END-EXEC.
```

Checks a name and processes subsequent lines if name has not been created with EXEC SQL define name.

```
EXEC SQL else END-EXEC.
```

Starts processing an alternative section to a section introduced by either EXEC SQL ifdef name or EXEC SQL ifndef name.

```
EXEC SQL elif name END-EXEC.
```

Checks name and starts an alternative section if name has been created with EXEC SQL define name.

```
EXEC SQL endif END-EXEC.
```

Ends an alternative section.

Example:

```
EXEC SQL ifndef TZVAR END-EXEC.
EXEC SQL SET TIMEZONE TO 'GMT' END-EXEC.
EXEC SQL elif TZNAME END-EXEC.
EXEC SQL SET TIMEZONE TO TZNAME END-EXEC.
EXEC SQL else END-EXEC.
EXEC SQL SET TIMEZONE TO TZVAR END-EXEC.
EXEC SQL endif END-EXEC.
```

D.9 Processing Embedded SQL Programs

Now that you have an idea how to form embedded SQL COBOL programs, you probably want to know how to compile them. Before compiling you run the file through the embedded SQL COBOL preprocessor, which converts the SQL statements you used to special function calls. After compiling, you must link with a special library that contains the needed functions. These functions fetch information from the arguments, perform the SQL command using the libpq interface, and put the result in the arguments specified for output.

The preprocessor program is called ecobpg and is included in a normal PostgreSQL installation. Embedded SQL programs are typically named with an extension .pco. If you have a program file called prog1.pco, you can preprocess it by simply calling:

```
ecobpg prog1.pco
```

This will create a file called prog1.cob. If your input files do not follow the suggested naming pattern, you can specify the output file explicitly using the -o option.

The preprocessed file can be compiled normally, following the usage of the compiler.

The generated COBOL source files include library files from the PostgreSQL installation, so if you installed PostgreSQL in a location that is not searched by default, you have to add an option such as -I/usr/local/pgsql/include to the compilation command line.

To link an embedded SQL program, you need to include the libecpg library.

Again, you might have to add an option for library search like -L/usr/local/pgsql/lib to that command line.

If you manage the build process of a larger project using make, it might be convenient to include the following implicit rule to your makefiles:

```
ECOBPG = ecobpg

%.cob: %.pco
        $(ECOBPG) $<
```

The complete syntax of the ecobpg command is detailed in "D.12.1 ecobpg".

Currently, ecobpg does not support multi threading.

D.10 Large Objects

Large objects are not supported by ECOBPG.

If you need to access large objects, use large objects interfaces of libpq instead.

D.11 Embedded SQL Commands

This section describes all SQL commands that are specific to embedded SQL.

Command	Description
ALLOCATE DESCRIPTOR	Allocate an SQL descriptor area

Command	Description
CONNECT	Eestablish a database connection
DEALLOCATE DESCRIPTOR	Deallocate an SQL descriptor area
DECLARE	Define a cursor
DESCRIBE	Obtain information about a prepared statement or result set
DISCONNECT	Terminate a database connection
EXECUTE IMMEDIATE	Dynamically prepare and execute a statement
GET DESCRIPTOR	Get information from an SQL descriptor area
OPEN	Open a dynamic cursor
PREPARE	Prepare a statement for execution
SET AUTOCOMMIT	Set the autocommit behavior of the current session
SET CONNECTION	Select a database connection
SET DESCRIPTOR	Set information in an SQL descriptor area
TYPE	Define a new data type
VAR	Define a variable
WHENEVER	Specify the action to be taken when an SQL statement causes a specific class condition to be raised

Refer to the SQL commands listed in "SQL Commands" under "Reference" in the PostgreSQL Documentation, which can also be used in embedded SQL, unless stated otherwise.

D.11.1 ALLOCATE DESCRIPTOR

Name

ALLOCATE DESCRIPTOR -- allocate an SQL descriptor area

Synopsis

```
ALLOCATE DESCRIPTOR name
```

Description

ALLOCATE DESCRIPTOR allocates a new named SQL descriptor area, which can be used to exchange data between the PostgreSQL server and the host program.

Descriptor areas should be freed after use using the DEALLOCATE DESCRIPTOR command.

Parameters

name

A name of SQL descriptor, case sensitive. This can be an SQL identifier or a host variable.

Examples

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc END-EXEC.
```

Compatibility

ALLOCATE DESCRIPTOR is specified in the SQL standard.

See Also

DEALLOCATE DESCRIPTOR, GET DESCRIPTOR, SET DESCRIPTOR

D.11.2 CONNECT

Name

CONNECT -- establish a database connection

Synopsis

```
CONNECT TO connection_target [ AS connection_name ] [ USER connection_user_name ]

CONNECT TO DEFAULT

CONNECT connection_user_name

DATABASE connection_target
```

Description

The CONNECT command establishes a connection between the client and the PostgreSQL server.

Parameters

connection_target

connection_target specifies the target server of the connection on one of several forms.

[database_name] [@host] [:port]

Connect over TCP/IP

unix:postgresql://host [:port] / [database_name] [?connection_option]

Connect over Unix-domain sockets

tcp:postgresql://host [:port] / [database_name] [?connection_option]

Connect over TCP/IP

SQL string constant

containing a value in one of the above forms

host variable

host variable of fixed-length string (trailing spaces are ignored) containing a value in one of the above forms

connection_name

An optional identifier for the connection, so that it can be referred to in other commands. This can be an SQL identifier or a host variable.

connection_user_name

The user name for the database connection.

This parameter can also specify user name and password, using one the forms user_name/password, user_name IDENTIFIED BY password, or user_name USING password.

User name and password can be SQL identifiers, string constants, or host variables(fixed-length string, trailing spaces are ignored).

DEFAULT

Use all default connection parameters, as defined by libpq.

Examples

Here a several variants for specifying connection parameters:

```
EXEC SQL CONNECT TO "connectdb" AS main END-EXEC.
EXEC SQL CONNECT TO "connectdb" AS second END-EXEC.
EXEC SQL CONNECT TO "unix:postgresql://localhost/connectdb" AS main USER connectuser END-
EXEC.
EXEC SQL CONNECT TO 'connectdb' AS main END-EXEC.
EXEC SQL CONNECT TO 'unix:postgresql://localhost/connectdb' AS main USER :user END-EXEC.
EXEC SQL CONNECT TO :dbn AS :idt END-EXEC.
EXEC SQL CONNECT TO :dbn USER connectuser USING :pw END-EXEC.
EXEC SQL CONNECT TO @localhost AS main USER connectdb END-EXEC.
EXEC SQL CONNECT TO REGRESSDB1 as main END-EXEC.
EXEC SQL CONNECT TO connectdb AS :idt END-EXEC.
EXEC SQL CONNECT TO connectdb AS main USER connectuser/connectdb END-EXEC.
EXEC SQL CONNECT TO connectdb AS main END-EXEC.
EXEC SQL CONNECT TO connectdb@localhost AS main END-EXEC.
EXEC SQL CONNECT TO tcp:postgresql://localhost/ USER connectdb END-EXEC.
EXEC SQL CONNECT TO tcp:postgresql://localhost/connectdb USER connectuser IDENTIFIED BY
connectpw END-EXEC.
EXEC SQL CONNECT TO tcp:postgresql://localhost:20/connectdb USER connectuser IDENTIFIED BY
connectpw END-EXEC.
EXEC SQL CONNECT TO unix:postgresql://localhost/ AS main USER connectdb END-EXEC.
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb AS main USER connectuser END-
EXEC.
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER connectuser IDENTIFIED BY
"connectpw" END-EXEC.
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb USER connectuser USING
"connectpw" END-EXEC.
EXEC SQL CONNECT TO unix:postgresql://localhost/connectdb?connect_timeout=14 USER
connectuser END-EXEC.
```

Here is an example program that illustrates the use of host variables to specify connection parameters:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
*   database name
    01 DBNAME PIC X(6).
*   connection user name
    01 USER PIC X(8).
*   connection string
    01 CONNECTION PIC X(38).

    01 VER PIC X(256).
EXEC SQL END DECLARE SECTION END-EXEC.

    MOVE "testdb" TO DBNAME.
    MOVE "testuser" TO USER.
    MOVE "tcp:postgresql://localhost:5432/testdb" TO CONNECTION.

    EXEC SQL CONNECT TO :DBNAME USER :USER END-EXEC.
    EXEC SQL SELECT version() INTO :VER END-EXEC.
    EXEC SQL DISCONNECT END-EXEC.


    DISPLAY "version: " VER.

    EXEC SQL CONNECT TO :CONNECTION USER :USER END-EXEC.
```

```
    EXEC SQL SELECT version() INTO :VER END-EXEC.
    EXEC SQL DISCONNECT END-EXEC.

    DISPLAY "version: " VER.
```

Compatibility

CONNECT is specified in the SQL standard, but the format of the connection parameters is implementation-specific.

See Also

DISCONNECT, SET CONNECTION

D.11.3 DEALLOCATE DESCRIPTOR

Name

DEALLOCATE DESCRIPTOR -- deallocate an SQL descriptor area

Synopsis

```
DEALLOCATE DESCRIPTOR name
```

Description

DEALLOCATE DESCRIPTOR deallocates a named SQL descriptor area.

Parameters

name

The name of the descriptor which is going to be deallocated. It is case sensitive. This can be an SQL identifier or a host variable.

Examples

EXEC SQL DEALLOCATE DESCRIPTOR mydesc END-EXEC.

Compatibility

DEALLOCATE DESCRIPTOR is specified in the SQL standard.

See Also

ALLOCATE DESCRIPTOR, GET DESCRIPTOR, SET DESCRIPTOR

D.11.4 DECLARE

Name

DECLARE -- define a cursor

Synopsis

```
DECLARE cursor_name [ BINARY ] [ INSENSITIVE ] [ [ NO ] SCROLL ] CURSOR [ { WITH | WITHOUT }
HOLD ] FOR prepared_name

DECLARE cursor_name [ BINARY ] [ INSENSITIVE ] [ [ NO ] SCROLL ] CURSOR [ { WITH | WITHOUT }
HOLD ] FOR query
```

Description

DECLARE declares a cursor for iterating over the result set of a prepared statement. This command has slightly different semantics from the direct SQL command DECLARE: Whereas the latter executes a query and prepares the result set for retrieval, this embedded SQL command merely declares a name as a "loop variable" for iterating over the result set of a query; the actual execution happens when the cursor is opened with the OPEN command.

Parameters

cursor_name

A cursor name, case sensitive. This can be an SQL identifier or a host variable.

prepared_name

The name of a prepared query, either as an SQL identifier or a host variable.

query

A SELECT or VALUES command which will provide the rows to be returned by the cursor.

For the meaning of the cursor options, see DECLARE.

See

Refer to "SQL Commands" in "Reference" in the PostgreSQL Documentation for information on the SELECT, VALUES and DECLARE command.

Examples

Examples declaring a cursor for a query:

```
EXEC SQL DECLARE C CURSOR FOR SELECT * FROM My_Table END-EXEC.
EXEC SQL DECLARE C CURSOR FOR SELECT Item1 FROM T END-EXEC.
EXEC SQL DECLARE cur1 CURSOR FOR SELECT version() END-EXEC.
```

An example declaring a cursor for a prepared statement:

```
EXEC SQL PREPARE stmt1 AS SELECT version() END-EXEC.
EXEC SQL DECLARE cur1 CURSOR FOR stmt1 END-EXEC.
```

Compatibility

DECLARE is specified in the SQL standard.

See Also

OPEN, CLOSE, DECLARE

See

Refer to "SQL Commands" in "Reference" in the PostgreSQL Documentation for information on the CLOSE and DECLARE command.

D.11.5 DESCRIBE

Name

DESCRIBE -- obtain information about a prepared statement or result set

Synopsis

```
DESCRIBE [ OUTPUT ] prepared_name USING SQL DESCRIPTOR descriptor_name

DESCRIBE [ OUTPUT ] prepared_name INTO SQL DESCRIPTOR descriptor_name
```

Description

DESCRIBE retrieves metadata information about the result columns contained in a prepared statement, without actually fetching a row.

Parameters

prepared_name

The name of a prepared statement. This can be an SQL identifier or a host variable.

descriptor_name

A descriptor name. It is case sensitive. It can be an SQL identifier or a host variable.

Examples

```
EXEC SQL ALLOCATE DESCRIPTOR mydesc END-EXEC.
EXEC SQL PREPARE stmt1 FROM :sql_stmt END-EXEC.
EXEC SQL DESCRIBE stmt1 INTO SQL DESCRIPTOR mydesc END-EXEC.
EXEC SQL GET DESCRIPTOR mydesc VALUE 1 :charvar = NAME END-EXEC.
EXEC SQL DEALLOCATE DESCRIPTOR mydesc END-EXEC.
```

Compatibility

DESCRIBE is specified in the SQL standard.

See Also

ALLOCATE DESCRIPTOR, GET DESCRIPTOR

D.11.6 DISCONNECT

Name

DISCONNECT -- terminate a database connection

Synopsis

```
DISCONNECT connection_name

DISCONNECT [ CURRENT ]

DISCONNECT DEFAULT

DISCONNECT ALL
```

Description

DISCONNECT closes a connection (or all connections) to the database.

Parameters

connection_name

A database connection name established by the CONNECT command.

CURRENT

Close the "current" connection, which is either the most recently opened connection, or the connection set by the SET CONNECTION command. This is also the default if no argument is given to the DISCONNECT command.

DEFAULT

Close the default connection.

ALL

Close all open connections.

Examples

```
    EXEC SQL CONNECT TO testdb AS DEFAULT USER testuser END-EXEC.
    EXEC SQL CONNECT TO testdb AS con1 USER testuser END-EXEC.
    EXEC SQL CONNECT TO testdb AS con2 USER testuser END-EXEC.
    EXEC SQL CONNECT TO testdb AS con3 USER testuser END-EXEC.

*   close con3
    EXEC SQL DISCONNECT CURRENT END-EXEC.
*   close DEFAULT
    EXEC SQL DISCONNECT DEFAULT END-EXEC.
*   close con2 and con1
    EXEC SQL DISCONNECT ALL END-EXEC.
```

Compatibility

DISCONNECT is specified in the SQL standard.

See Also

CONNECT, SET CONNECTION

D.11.7 EXECUTE IMMEDIATE

Name

EXECUTE IMMEDIATE -- dynamically prepare and execute a statement

Synopsis

```
EXECUTE IMMEDIATE string
```

Description

EXECUTE IMMEDIATE immediately prepares and executes a dynamically specified SQL statement, without retrieving result rows.

Parameters

string

A literal string or a host variable containing the SQL statement to be executed.

Examples

Here is an example that executes an INSERT statement using EXECUTE IMMEDIATE and a host variable named command:

```
MOVE "INSERT INTO test (name, amount, letter) VALUES ('db: ''r1''', 1, 'f')" TO ARR OF cmd.
COMPUTE LEN OF cmd = FUNCTION STORED-CHAR-LENGTH(ARR OF cmd).
EXEC SQL EXECUTE IMMEDIATE :cmd END-EXEC.
```

Compatibility

EXECUTE IMMEDIATE is specified in the SQL standard.

D.11.8 GET DESCRIPTOR

Name

GET DESCRIPTOR -- get information from an SQL descriptor area

Synopsis

```
GET DESCRIPTOR descriptor_name :hostvariable = descriptor_header_item [, ... ]

GET DESCRIPTOR descriptor_name VALUE column_number :hostvariable = descriptor_item [, ... ]
```

Description

GET DESCRIPTOR retrieves information about a query result set from an SQL descriptor area and stores it into host variables. A descriptor area is typically populated using FETCH or SELECT before using this command to transfer the information into host language variables.

This command has two forms: The first form retrieves descriptor "header" items, which apply to the result set in its entirety. One example is the row count. The second form, which requires the column number as additional parameter, retrieves information about a particular column. Examples are the column name and the actual column value.

Parameters

descriptor_name

A descriptor name.

descriptor_header_item

A token identifying which header information item to retrieve. Only COUNT, to get the number of columns in the result set, is currently supported.

column_number

The number of the column about which information is to be retrieved. The count starts at 1.

descriptor_item

A token identifying which item of information about a column to retrieve. See Section 33.7.1 for a list of supported items.

hostvariable

A host variable that will receive the data retrieved from the descriptor area.

Examples

An example to retrieve the number of columns in a result set:

```
EXEC SQL GET DESCRIPTOR d :d_count = COUNT END-EXEC.
```

An example to retrieve a data length in the first column:

```
EXEC SQL GET DESCRIPTOR d VALUE 1 :d_returned_octet_length = RETURNED_OCTET_LENGTH END-
EXEC.
```

An example to retrieve the data body of the second column as a string:

```
EXEC SQL GET DESCRIPTOR d VALUE 2 :d_data = DATA END-EXEC.
```

Here is an example for a whole procedure of executing SELECT current_database(); and showing the number of columns, the column data length, and the column data:

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 D-COUNT PIC S9(9) COMP-5.
    01 D-DATA PIC X(1024).
    01 D-RETURNED-OCTET-LENGTH PIC S9(9) COMP.
EXEC SQL END DECLARE SECTION END-EXEC.

    EXEC SQL CONNECT TO testdb AS con1 USER testuser END-EXEC.
    EXEC SQL ALLOCATE DESCRIPTOR d END-EXEC.

*   Declare, open a cursor, and assign a descriptor to the cursor
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database() END-EXEC.
    EXEC SQL OPEN cur END-EXEC.
    EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d END-EXEC.

*   Get a number of total columns
    EXEC SQL GET DESCRIPTOR d :D-COUNT = COUNT END-EXEC.
    DISPLAY "d_count                 = " D-COUNT.

*   Get length of a returned column
    EXEC SQL GET DESCRIPTOR d VALUE 1 :D-RETURNED-OCTET-LENGTH = RETURNED_OCTET_LENGTH END-
EXEC.
    DISPLAY "d_returned_octet_length = " D-RETURNED-OCTET-LENGTH.

*   Fetch the returned column as a string
    EXEC SQL GET DESCRIPTOR d VALUE 1 :D-DATA = DATA END-EXEC.
    DISPLAY "d_data                  = " D-DATA.

*   Closing
    EXEC SQL CLOSE cur END-EXEC.
    EXEC SQL COMMIT END-EXEC.

    EXEC SQL DEALLOCATE DESCRIPTOR d END-EXEC.
    EXEC SQL DISCONNECT ALL END-EXEC.
```

When the example is executed, the result will look like this:

```
d_count                 = +000000001
d_returned_octet_length = +000000006
d_data                  = testdb
```

Compatibility

GET DESCRIPTOR is specified in the SQL standard.

See Also

ALLOCATE DESCRIPTOR, SET DESCRIPTOR

D.11.9 OPEN

Name

OPEN -- open a dynamic cursor

Synopsis

```
OPEN cursor_name
```

```
OPEN cursor_name USING value [, ... ]

OPEN cursor_name USING SQL DESCRIPTOR descriptor_name
```

Description

OPEN opens a cursor and optionally binds actual values to the placeholders in the cursor's declaration. The cursor must previously have been declared with the DECLARE command. The execution of OPEN causes the query to start executing on the server.

Parameters

cursor_name

The name of the cursor to be opened. This can be an SQL identifier or a host variable.

value

A value to be bound to a placeholder in the cursor. This can be an SQL constant, a host variable, or a host variable with indicator.

descriptor_name

The name of a descriptor containing values to be bound to the placeholders in the cursor. This can be an SQL identifier or a host variable.

Examples

```
EXEC SQL OPEN a END-EXEC.
EXEC SQL OPEN d USING 1, 'test' END-EXEC.
EXEC SQL OPEN c1 USING SQL DESCRIPTOR mydesc END-EXEC.
EXEC SQL OPEN :curname1 END-EXEC.
```

Compatibility

OPEN is specified in the SQL standard.

See Also

DECLARE, CLOSE

Refer to "SQL Commands" in "Reference" in the PostgreSQL Documentation for information on the CLOSE command.

D.11.10 PREPARE

Name

PREPARE -- prepare a statement for execution

Synopsis

```
PREPARE name FROM string
```

Description

PREPARE prepares a statement dynamically specified as a string for execution. This is different from the direct SQL statement PREPARE, which can also be used in embedded programs. The EXECUTE command is used to execute either kind of prepared statement.

Parameters

prepared_name

An identifier for the prepared query.

string

A literal string or a host variable containing a preparable statement, one of the SELECT, INSERT, UPDATE, or DELETE.

Examples

```
MOVE "SELECT * FROM test1 WHERE a = ? AND b = ?" TO ARR OF STMT.
COMPUTE LEN OF STMT = FUNCTION STORED-CHAR-LENGTH (ARR OF STMT).
EXEC SQL ALLOCATE DESCRIPTOR indesc END-EXEC.
EXEC SQL ALLOCATE DESCRIPTOR outdesc END-EXEC.
EXEC SQL PREPARE foo FROM :STMT END-EXEC.

EXEC SQL EXECUTE foo USING SQL DESCRIPTOR indesc INTO SQL DESCRIPTOR outdesc END-EXEC.
```

Compatibility

PREPARE is specified in the SQL standard.

See Also

EXECUTE

Refer to "SQL Commands" in "Reference" in the PostgreSQL Documentation for information on the EXECUTE command.

D.11.11 SET AUTOCOMMIT

Name

SET AUTOCOMMIT -- set the autocommit behavior of the current session

Synopsis

```
SET AUTOCOMMIT { = | TO } { ON | OFF }
```

Description

SET AUTOCOMMIT sets the autocommit behavior of the current database session. By default, embedded SQL programs are not in autocommit mode, so COMMIT needs to be issued explicitly when desired. This command can change the session to autocommit mode, where each individual statement is committed implicitly.

Compatibility

SET AUTOCOMMIT is an extension of PostgreSQL ECOBPG.

D.11.12 SET CONNECTION

Name

SET CONNECTION -- select a database connection

Synopsis

```
SET CONNECTION [ TO | = ] connection_name
```

Description

SET CONNECTION sets the "current" database connection, which is the one that all commands use unless overridden.

Parameters

connection_name

A database connection name established by the CONNECT command.

DEFAULT

Set the connection to the default connection.

Examples

```
EXEC SQL SET CONNECTION TO con2 END-EXEC.
EXEC SQL SET CONNECTION = con1 END-EXEC.
```

Compatibility

SET CONNECTION is specified in the SQL standard.

See Also

CONNECT, DISCONNECT

D.11.13 SET DESCRIPTOR

Name

SET DESCRIPTOR -- set information in an SQL descriptor area

Synopsis

```
SET DESCRIPTOR descriptor_name descriptor_header_item = value [, ... ]

SET DESCRIPTOR descriptor_name VALUE number descriptor_item = value [, ...]
```

Description

SET DESCRIPTOR populates an SQL descriptor area with values. The descriptor area is then typically used to bind parameters in a prepared query execution.

This command has two forms: The first form applies to the descriptor "header", which is independent of a particular datum. The second form assigns values to particular datums, identified by number.

Parameters

descriptor_name

A descriptor name.

descriptor_header_item

A token identifying which header information item to set. Only COUNT, to set the number of descriptor items, is currently supported.

number

The number of the descriptor item to set. The count starts at 1.

descriptor_item

A token identifying which item of information to set in the descriptor. See Section 33.7.1 for a list of supported items.

value

A value to store into the descriptor item. This can be an SQL constant or a host variable.

Examples

```
EXEC SQL SET DESCRIPTOR indesc COUNT = 1 END-EXEC.
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = 2 END-EXEC.
EXEC SQL SET DESCRIPTOR indesc VALUE 1 DATA = :val1 END-EXEC.
EXEC SQL SET DESCRIPTOR indesc VALUE 2 DATA = 'some string', INDICATOR = :val1 END-EXEC.
EXEC SQL SET DESCRIPTOR indesc VALUE 2 INDICATOR = :val2null, DATA = :val2 END-EXEC.
```

Compatibility

SET DESCRIPTOR is specified in the SQL standard.

See Also

ALLOCATE DESCRIPTOR, GET DESCRIPTOR

D.11.14 TYPE

Name

TYPE -- define a new data type

Synopsis

```
TYPE type_name IS ctype
```

Description

The TYPE command defines a new COBOL type. It is equivalent to putting a typedef into a declare section.

This command is only recognized when ecobpgpg is run with the -c option.

A level number of 01 is automatically added to type_name item. Thus, the level number must not to be specified externally. To define a group item, a level number needs to be specified to the each subordinate items.

For reasons of internal implementation, "TYPE" must be placed just after "EXEC SQL", without containing newline. For other place, you can use newline.

Parameters

type_name

The name for the new type. It must be a valid COBOL type name.

ctype

A COBOL type specification (including expression format specification).

Examples

```
EXEC SQL TYPE CUSTOMER IS
      02  NAME PIC X(50) VARYING.
      02 PHONE PIC S9(9) COMP. END-EXEC.

EXEC SQL TYPE CUST-IND IS
      02  NAME_IND PIC S9(4) COMP.
      02 PHONE_IND PIC S9(4) COMP. END-EXEC.

EXEC SQL TYPE INTARRAY IS
      02 INT PIC S9(9) OCCURS 20. END-EXEC.
```

```
EXEC SQL TYPE STR IS PIC X(50) VARYING. END-EXEC.
EXEC SQL TYPE STRING IS PIC X(10). END-EXEC.
```

Here is an example program that uses EXEC SQL TYPE:

```
EXEC SQL TYPE TT IS
     02 V PIC X(256) VARYING.
     02 I PIC S9(9) COMP. END-EXEC.


EXEC SQL TYPE TT-IND IS
     02 V-IND PIC S9(4) COMP.
     02 I-IND PIC S9(4) COMP. END-EXEC.

EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 T TYPE TT.
    01 T-IND TYPE TT-IND.
EXEC SQL END DECLARE SECTION END-EXEC.

    EXEC SQL CONNECT TO testdb AS con1 END-EXEC.

    EXEC SQL SELECT current_database(), 256 INTO :T :T-IND LIMIT 1 END-EXEC.

    DISPLAY "t.v = " ARR OF V OF T.
    DISPLAY "t.i = " I OF T.

    DISPLAY "t_ind.v_ind = " V-IND OF T-IND.
    DISPLAY "t_ind.i_ind = " I-IND OF T-IND.

    EXEC SQL DISCONNECT con1 END-EXEC.
```

Compatibility

The TYPE command is a PostgreSQL extension.

D.11.15 VAR

Name

VAR− define a variable

Synopsis

```
VAR varname IS ctype
```

Description

The VAR command defines a host variable. It is equivalent to an ordinary COBOL variable definition insidea declare section.

When translating , a level number 01 is added. Thus, the level number must not to be specified externally.

To define a group item, a level number needs to be specified to the each subordinate items.

For reasons of internal implementation, "VAR" must be placed just after "EXEC SQL", without containing newline. For other place, you can use newline.

Parameters

varname

A COBOL variable name.

ctype

A COBOL type specification.

Examples

```
EXEC SQL VAR VC IS PIC X(10) VARYING. END-EXEC.
EXEC SQL VAR BOOL-VAR IS BOOL. END-EXEC.
```

Compatibility

The VAR command is a PostgreSQL extension.

D.11.16 WHENEVER

Name

WHENEVER -- specify the action to be taken when an SQL statement causes a specific class condition to be raised

Synopsis

```
WHENEVER { NOT FOUND | SQLERROR | SQLWARNING } action
```

Description

Define a behavior which is called on the special cases (Rows not found, SQL warnings or errors) in the result of SQL execution.

Parameters

See Section "D.7.1 Setting Callbacks" or a description of the parameters.

Examples

```
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
EXEC SQL WHENEVER SQLWARNING SQLPRINT END-EXEC.
EXEC SQL WHENEVER SQLWARNING DO "warn" END-EXEC.
EXEC SQL WHENEVER SQLERROR sqlprint END-EXEC.
EXEC SQL WHENEVER SQLERROR CALL "print2" END-EXEC.
EXEC SQL WHENEVER SQLERROR DO handle_error USING "select" END-EXEC.
EXEC SQL WHENEVER SQLERROR DO sqlnotice USING 0 1 END-EXEC.
EXEC SQL WHENEVER SQLERROR DO "sqlprint" END-EXEC.
EXEC SQL WHENEVER SQLERROR GOTO error_label END-EXEC.
EXEC SQL WHENEVER SQLERROR STOP END-EXEC.
```

A typical application is the use of WHENEVER NOT FOUND GOTO to handle looping through result sets:

```
    EXEC SQL CONNECT TO testdb AS con1 END-EXEC.
    EXEC SQL ALLOCATE DESCRIPTOR d END-EXEC.
    EXEC SQL DECLARE cur CURSOR FOR SELECT current_database(), 'hoge', 256 END-EXEC.
    EXEC SQL OPEN cur END-EXEC.

*   when end of result set reached, break out of while loop
    EXEC SQL WHENEVER NOT FOUND GOTO NOTFOUND END-EXEC.

    PERFORM NO LIMIT
        EXEC SQL FETCH NEXT FROM cur INTO SQL DESCRIPTOR d END-EXEC
        ...
    END-PERFORM.

    NOTFOUND.
```

```
EXEC SQL CLOSE cur END-EXEC.
EXEC SQL COMMIT END-EXEC.

EXEC SQL DEALLOCATE DESCRIPTOR d END-EXEC.
EXEC SQL DISCONNECT ALL END-EXEC.
```

Compatibility

WHENEVER is specified in the SQL standard, but most of the actions are PostgreSQL extensions.

D.12 PostgreSQL Client Applications

This part contains reference information for PostgreSQL client applications and utilities. Not all oft these commands are of general utility; some might require special privileges. The common feature of these applications is that they can be run on any host, independent of where the database server resides.

When specified on the command line, user and database names have their case preserved－the presence of spaces or special characters might require quoting. Table names and other identifiers do not have their case preserved, except where documented, and might require quoting.

D.12.1 ecobpg

Name

ecobpg -- embedded SQL COBOL preprocessor

Synopsis

ecobpg [option...] file...

Description

ecobpg is the embedded SQL preprocessor for COBOL programs. It converts COBOL programs with embedded SQL statements to normal COBOL code by replacing the SQL invocations with special function calls. The output files can then be processed with any COBOL compiler tool chain.

ecobpg will convert each input file given on the command line to the corresponding COBOL output file. Input files preferably have the extension .pco, in which case the extension will be replaced by .cob to determine the output file name. If the extension of the input file is not .pco, then the output file name is computed by appending .cob to the full file name. The output file name can also be overridden using the -o option.

Options

ecobpg accepts the following command-line arguments:

-c

Automatically generate certain COBOL code from SQL code. Currently, this works for EXEC SQL TYPE.

-I directory

Specify an additional include path, used to find files included via EXEC SQL INCLUDE. Defaults are . (current directory), /usr/local/include, the PostgreSQL include directory which is defined at compile time (default: /usr/local/pgsql/include), and /usr/include, in that order.

-o filename

Specifies that ecobpg should write all its output to the given filename.

-r option

Selects run-time behavior. Option can be one of the following:

prepare

Prepare all statements before using them. Libecpg will keep a cache of prepared statements and reuse a statement if it gets executed again. If the cache runs full, libecpg will free the least used statement.

questionmarks

Allow question mark as placeholder for compatibility reasons. This used to be the default long ago.

-t

Turn on autocommit of transactions. In this mode, each SQL command is automatically committed unless it is inside an explicit transaction block. In the default mode, commands are committed only when EXEC SQL COMMIT is issued.

--varchar-with-named-member

When converting VARCHAR host variable, adding name of the variable to members as prefix. Instead of LEN and ARR, (varname)-ARR and (varname)-LEN will be used.

-v

Print additional information including the version and the "include" path.

--version

Print the ecobpg version and exit.

-?
--help

Show help about ecobpg command line arguments, and exit.

Notes

When compiling the preprocessed COBOL code files, the compiler needs to be able to find the ECOBPG library text files in the PostgreSQL include directory. Therefore, you might have to use the -I option when invoking the compiler.

Programs using COBOL code with embedded SQL have to be linked against the libecpg library, for example using the linker options.

The value of either of these directories that is appropriate for the installation can be found out using pg_config.

See

Refer to "pg_config" in "Reference" in the PostgreSQL Documentation.

Examples

If you have an embedded SQL COBOL source file named prog1.pco, you can create an executable program using the following command:

```
ecobpg prog1.pco
```

Appendix E Quantitative Limits

This appendix lists the quantitative limits of Enterprise Postgres.

Table E.1 Length of identifier

Item	Limit
Database name	Up to 63 bytes (*1)(*2)(*3)
Schema name	Up to 63 bytes (*1)(*2)
Table name	Up to 63 bytes (*1)(*2)
View name	Up to 63 bytes (*1)(*2)
Index name	Up to 63 bytes (*1)(*2)
Table space name	Up to 63 bytes (*1)(*2)
Cursor name	Up to 63 bytes (*1)(*2)
Function name	Up to 63 bytes (*1)(*2)
Aggregate function name	Up to 63 bytes (*1)(*2)
Trigger name	Up to 63 bytes (*1)(*2)
Constraint name	Up to 63 bytes (*1)(*2)
Conversion name	Up to 63 bytes (*1)(*2)
Role name	Up to 63 bytes (*1)(*2)(*4)
Cast name	Up to 63 bytes (*1)(*2)
Collation sequence name	Up to 63 bytes (*1)(*2)
Encoding method conversion name	Up to 63 bytes (*1)(*2)
Domain name	Up to 63 bytes (*1)(*2)
Extension name	Up to 63 bytes (*1)(*2)
Operator name	Up to 63 bytes (*1)(*2)
Operator class name	Up to 63 bytes (*1)(*2)
Operator family name	Up to 63 bytes (*1)(*2)
Rewrite rule name	Up to 63 bytes (*1)(*2)
Sequence name	Up to 63 bytes (*1)(*2)
Text search settings name	Up to 63 bytes (*1)(*2)
Text search dictionary name	Up to 63 bytes (*1)(*2)
Text search parser name	Up to 63 bytes (*1)(*2)
Text search template name	Up to 63 bytes (*1)(*2)
Data type name	Up to 63 bytes (*1)(*2)
Enumerator type label	Up to 63 bytes (*1)(*2)

*1: This is the character string byte length when converted by the server character set character code.

*2: If an identifier that exceeds 63 bytes in length is specified, the excess characters are truncated and it is processed.

*3: Names of databases that use PL/extJava must be 28 bytes or less.

*4: Names of roles for connecting from the PL/extJava application server must be 8 bytes or less.

Table E.2 Database object

Item	Limit
Number of databases	Less than 4,294,967,296 (*1)
Number of schemas	Less than 4,294,967,296 (*1)
Number of tables	Less than 4,294,967,296 (*1)
Number of views	Less than 4,294,967,296 (*1)
Number of indexes	Less than 4,294,967,296 (*1)
Number of table spaces	Less than 4,294,967,296 (*1)
Number of functions	Less than 4,294,967,296 (*1)
Number of aggregate functions	Less than 4,294,967,296 (*1)
Number of triggers	Less than 4,294,967,296 (*1)
Number of constraints	Less than 4,294,967,296 (*1)
Number of conversion	Less than 4,294,967,296 (*1)
Number of roles	Less than 4,294,967,296 (*1)
Number of casts	Less than 4,294,967,296 (*1)
Number of collation sequences	Less than 4,294,967,296 (*1)
Number of encoding method conversions	Less than 4,294,967,296 (*1)
Number of domains	Less than 4,294,967,296 (*1)
Number of extensions	Less than 4,294,967,296 (*1)
Number of operators	Less than 4,294,967,296 (*1)
Number of operator classes	Less than 4,294,967,296 (*1)
Number of operator families	Less than 4,294,967,296 (*1)
Number of rewrite rules	Less than 4,294,967,296 (*1)
Number of sequences	Less than 4,294,967,296 (*1)
Number of text search settings	Less than 4,294,967,296 (*1)
Number of text search dictionaries	Less than 4,294,967,296 (*1)
Number of text search parsers	Less than 4,294,967,296 (*1)
Number of text search templates	Less than 4,294,967,296 (*1)
Number of data types	Less than 4,294,967,296 (*1)
Number of enumerator type labels	Less than 4,294,967,296 (*1)
Number of default access privileges defined in the ALTER DEFAULT PRIVILEGES statement	Less than 4,294,967,296 (*1)
Number of large objects	Less than 4,294,967,296 (*1)
Number of rows in tables defined by WITH OIDS	Less than 4,294,967,296 (*1)
Number of index access methods	Less than 4,294,967,296 (*1)

*1: The total number of all database objects must be less than 4,294,967,296.

Table E.3 Schema element

Item	Limit
Number of columns that can be defined in one table	From 250 to 1600 (according to the data type)

Item	Limit
Table row length	Up to 400 gigabytes
Number of columns comprising a unique constraint	Up to 32 columns
Data length comprising a unique constraint	Less than 2,000 bytes (*1)(*2)
Table size	Up to one terabyte
Search condition character string length in a trigger definition statement	Up to 800 megabytes (*1)(*2)
Item size	Up to 1 gigabyte

***1: Operation might proceed correctly even if operations are performed with a quantity outside the limits.**

*2: This is the character string byte length when converted by the server character set character code.

Table E.4 Index

Item	Limit
Number of columns comprising a key (B-tree, GiST, GIN index)	Up to 32 columns
Key length	Less than 2,000 bytes (*1)

*1: This is the character string byte length when converted by the server character set character code.

Table E.5 Data types and attributes that can be handled

Item			Limit
Character	Data length		Data types and attributes that can be handled (*1)
	Specification length (n)		Up to 10,485,760 characters (*1)
Numeric	External decimal expression		Up to 131,072 digits before the decimal point, and up to 16,383 digits after the decimal point
	Internal binary expression	2 bytes	From -32,768 to 32,767
		4 bytes	From -2,147,483,648 to 2,147,483,647
		8 bytes	From -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
	Internal decimal expression		Up to 13,1072 digits before the decimal point, and up to 16,383 digits after the decimal point
	Floating point expression	4 bytes	From -3.4E+38 to -7.1E-46, 0, or from 7.1E-46 to 3.4E+38
		8 bytes	From -1.7E+308 to -2.5E-324, 0, or from 2.5E-324 to 1.7E+308
bytea			Up to one gigabyte minus 53 bytes
Large object			Up to two gigabytes

*1: This is the character string byte length when converted by the server character set character code.

Table E.6 Function definition

Item	Limit
Number of arguments that can be specified	Up to 100
Number of variable names that can be specified in the declarations section	No limit
Number of SQL statements or control statements that can be specified in a function processing implementation	No limit

Table E.7 Data operation statement

Item	Limit
Maximum number of connections for one process in an application (remote access)	4,000 connections
Number of expressions that can be specified in a selection list	Up to 1,664
Number of tables that can be specified in a FROM clause	No limit
Number of unique expressions that can be specified in a selection list/DISTINCT clause/ORDER BY clause/GROUP BY clause within one SELECT statement	Up to 1,664
Number of expressions that can be specified in a GROUP BY clause	No limit
Number of expressions that can be specified in an ORDER BY clause	No limit
Number of SELECT statements that can be specified in a UNION clause/INTERSECT clause/EXCEPT clause	Up to 4,000 (*1)
Number of nestings in joined tables that can be specified in one view	Up to 4,000 (*1)
Number of functions or operator expressions that can be specified in one expression	Up to 4,000 (*1)
Number of expressions that can be specified in one row constructor	Up to 1,664
Number of expressions that can be specified in an UPDATE statement SET clause	Up to 1,664
Number of expressions that can be specified in one row of a VALUES list	Up to 1,664
Number of expressions that can be specified in a RETURNING clause	Up to 1,664
Total expression length that can be specified in the argument list of one function specification	Up to 800 megabytes (*2)
Number of cursors that can be processed simultaneously by one session	No limit
Character string length of one SQL statement	Up to 800 megabytes (*1) (*3)
Number of input parameter specifications that can be specified in one dynamic SQL statement	No limit
Number of tokens that can be specified in one SQL statement	Up to 10,000
Number of values that can be specified as a list in a WHERE clause IN syntax	No limit
Number of expressions that can be specified in a USING clause	No limit

Item	Limit
Number of JOINs that can be specified in a joined table	Up to 4,000 (*1)
Number of expressions that can be specified in COALESCE	No limit
Number of WHEN clauses that can be specified for CASE in a simple format or a searched format	No limit
Data size per record that can be updated or inserted by one SQL statement	Up to one gigabyte minus 53 bytes
Number of objects that can share a lock simultaneously	Up to 256,000 (*1)

*1: Operation might proceed correctly even if operations are performed with a quantity outside the limits.

*2: The total number of all database objects must be less than 4,294,967,296.

*3: This is the character string byte length when converted by the server character set character code.

Table E.8 Data sizes

Item	Limit
Data size per record for input data files (COPY statement, psql command \copy meta command)	Up to 800 megabytes (*1)
Data size per record for output data files (COPY statement, psql command \copy meta command)	Up to 800 megabytes (*1)

*1: Operation might proceed correctly even if operations are performed with a quantity outside the limits.

Appendix F Reference

F.1 JDBC Driver

F.1.1 Java Programming Language API

java.sql

Interface name	Method name	jdbc4/jdbc41
Array	free()	N
	getArray()	Y
	getArray(long index, int count)	Y
	getArray(long index, int count, Map<String,Class<?>> map)	Y
	getArray(Map<String,Class<?>> map)	Y
	getBaseType()	Y
	getBaseTypeName()	Y
	getResultSet()	Y
	getResultSet(long index, int count)	Y
	getResultSet(long index, int count, Map<String,Class<?>> map)	Y
	getResultSet(Map<String,Class<?>> map)	Y
Blob	free()	Y
	getBinaryStream()	Y
	getBinaryStream(long pos, long length)	N
	getBytes(long pos, int length)	Y
	length()	Y
	position(Blob pattern, long start)	Y
	position(byte[] pattern, long start)	Y
	setBinaryStream(long pos)	Y
	setBytes(long pos, byte[] bytes)	Y
	setBytes(long pos, byte[] bytes, int offset, int len)	Y
	truncate(long len)	Y
CallableStatement	getArray(int parameterIndex)	Y
	getArray(String parameterName)	N
	getBigDecimal(int parameterIndex)	Y
	getBigDecimal(int parameterIndex, int scale)	Y
	getBigDecimal(String parameterName)	N
	getBlob(int parameterIndex)	N
	getBlob(String parameterName)	N
	getBoolean(int parameterIndex)	Y
	getBoolean(String parameterName)	N

Interface name	Method name	jdbc4/jdbc41
	getByte(int parameterIndex)	Y
	getByte(String parameterName)	N
	getBytes(int parameterIndex)	Y
	getBytes(String parameterName)	N
	getCharacterStream(int parameterIndex)	N
	getCharacterStream(String parameterName)	N
	getClob(int parameterIndex)	N
	getClob(String parameterName)	N
	getDate(int parameterIndex)	Y
	getDate(int parameterIndex, Calendar cal)	Y
	getDate(String parameterName)	N
	getDate(String parameterName, Calendar cal)	N
	getDouble(int parameterIndex)	Y
	getDouble(String parameterName)	N
	getFloat(int parameterIndex)	Y
	getFloat(String parameterName)	N
	getInt(int parameterIndex)	Y
	getInt(String parameterName)	N
	getLong(int parameterIndex)	Y
	getLong(String parameterName)	N
	getNCharacterStream(int parameterIndex)	Y
	getNCharacterStream(String parameterName)	N
	getNClob(int parameterIndex)	N
	getNClob(String parameterName)	N
	getNString(int parameterIndex)	Y
	getNString(String parameterName)	N
	getObject(int parameterIndex)	Y
	getObject(int parameterIndex, Map<String,Class<?>> map)	Y
	getObject(String parameterName)	N
	getObject(String parameterName, Map<String,Class<?>> map)	Y
	getRef(int parameterIndex)	N
	getRef(String parameterName)	N
	getRowId(int parameterIndex)	N
	getRowId(String parameterName)	N
	getShort(int parameterIndex)	Y
	getShort(String parameterName)	N
	getSQLXML(int parameterIndex)	Y
	getSQLXML(String parameterName)	N

Interface name	Method name	jdbc4/jdbc41
	getString(int parameterIndex)	Y
	getString(String parameterName)	N
	getTime(int parameterIndex)	Y
	getTime(int parameterIndex, Calendar cal)	Y
	getTime(String parameterName)	N
	getTime(String parameterName, Calendar cal)	N
	getTimestamp(int parameterIndex)	Y
	getTimestamp(int parameterIndex, Calendar cal)	Y
	getTimestamp(String parameterName)	N
	getTimestamp(String parameterName, Calendar cal)	N
	getURL(int parameterIndex)	N
	getURL(String parameterName)	N
	registerOutParameter(int parameterIndex, int sqlType)	Y
	registerOutParameter(int parameterIndex, int sqlType, int scale)	Y
	registerOutParameter(int parameterIndex, int sqlType, String typeName)	N
	registerOutParameter(String parameterName, int sqlType)	N
	registerOutParameter(String parameterName, int sqlType, int scale)	N
	registerOutParameter(String parameterName, int sqlType, String typeName)	N
	setAsciiStream(String parameterName, InputStream x)	N
	setAsciiStream(String parameterName, InputStream x, int length)	N
	setAsciiStream(String parameterName, InputStream x, long length)	N
	setBigDecimal(String parameterName, BigDecimal x)	N
	setBinaryStream(String parameterName, InputStream x)	N
	setBinaryStream(String parameterName, InputStream x, int length)	N
	setBinaryStream(String parameterName, InputStream x, long length)	N
	setBlob(String parameterName, Blob x)	N
	setBlob(String parameterName, InputStream inputStream)	N
	setBlob(String parameterName, InputStream inputStream, long length)	N
	setBoolean(String parameterName, boolean x)	N

Interface name	Method name	jdbc4/jdbc41
	setByte(String parameterName, byte x)	N
	setBytes(String parameterName, byte[] x)	N
	setCharacterStream(String parameterName, Reader reader)	N
	setCharacterStream(String parameterName, Reader reader, int length)	N
	setCharacterStream(String parameterName, Reader reader, long length)	N
	setClob(String parameterName, Clob x)	N
	setClob(String parameterName, Reader reader)	N
	setClob(String parameterName, Reader reader, long length)	N
	setDate(String parameterName, Date x)	N
	setDate(String parameterName, Date x, Calendar cal)	N
	setDouble(String parameterName, double x)	N
	setFloat(String parameterName, float x)	N
	setInt(String parameterName, int x)	N
	setLong(String parameterName, long x)	N
	setNCharacterStream(String parameterName, Reader value)	N
	setNCharacterStream(String parameterName, Reader value, long length)	Y
	setNClob(String parameterName, NClob value)	N
	setNClob(String parameterName, Reader reader)	N
	setNClob(String parameterName, Reader reader, long length)	N
	setNString(String parameterName, String value)	Y
	setNull(String parameterName, int sqlType)	N
	setNull(String parameterName, int sqlType, String typeName)	N
	setObject(String parameterName, Object x)	N
	setObject(String parameterName, Object x, int targetSqlType)	N
	setObject(String parameterName, Object x, int targetSqlType, int scale)	N
	setRowId(String parameterName, RowId x)	N
	setShort(String parameterName, short x)	N
	setSQLXML(String parameterName, SQLXML xmlObject)	N
	setString(String parameterName, String x)	N
	setTime(String parameterName, Time x)	N
	setTime(String parameterName, Time x, Calendar cal)	N

Interface name	Method name	jdbc4/jdbc41
	setTimestamp(String parameterName, Timestamp x)	N
	setTimestamp(String parameterName, Timestamp x, Calendar cal)	N
	setURL(String parameterName, URL val)	N
	wasNull()	Y
Clob	free()	Y
	getAsciiStream()	Y
	getCharacterStream()	Y
	getCharacterStream(long pos, long length)	N
	getSubString(long pos, int length)	Y
	length()	Y
	position(Clob searchstr, long start)	N
	position(String searchstr, long start)	N
	setAsciiStream(long pos)	N
	setCharacterStream(long pos)	N
	setString(long pos, String str)	N
	setString(long pos, String str, int offset, int len)	N
	truncate(long len)	Y
Connection	clearWarnings()	Y
	close()	Y
	commit()	Y
	createArrayOf(String typeName, Object[] elements)	Y
	createBlob()	N
	createClob()	N
	createNClob()	N
	createQueryObject(Class ifc)	N
	createQueryObject(Class ifc, Connection con)	N
	createSQLXML()	Y
	createStatement()	Y
	createStatement(int resultSetType, int resultSetConcurrency)	Y
	createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Y
	createStruct(String typeName, Object[] attributes)	N
	nativeSQL(String sql)	Y
	prepareCall(String sql)	Y
	prepareCall(String sql, int resultSetType, int resultSetConcurrency)	Y
	prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Y
	prepareStatement(String sql)	Y

Interface name	Method name	jdbc4/jdbc41
	prepareStatement(String sql, int resultSetType, int resultSetConcurrency)	Y
	prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)	Y
	prepareStatement(String sql, int autoGeneratedKeys)	Y
	prepareStatement(String sql, int[] columnIndexes)	Y
	prepareStatement(String sql, String[] columnNames)	Y
	releaseSavepoint(Savepoint savepoint)	Y
	rollback()	Y
	rollback(Savepoint savepoint)	Y
	setAutoCommit(boolean autoCommit)	Y
	getAutoCommit()	N
	setCatalog(String catalog)	Y
	getCatalog()	N
	setClientInfo(String name, String value)	Y
	setClientInfo(Properties properties)	Y
	getClientInfo(String name)	N
	getClientInfo()	N
	isClosed()	Y
	setHoldability(int holdability)	Y
	getHoldability()	N
	getMetaData()	Y
	setReadOnly(boolean readOnly)	Y
	isReadOnly()	Y
	setSavepoint()	Y
	setSavepoint(String name)	Y
	setTransactionIsolation(int level)	Y
	getTransactionIsolation()	N
	setTypeMap(Map map)	Y
	getTypeMap()	Y
	isValid(int timeout)	Y
	getWarnings()	Y
DatabaseMetaData	allProceduresAreCallable()	Y
	allTablesAreSelectable()	Y
	autoCommitFailureClosesAllResultSets()	Y
	dataDefinitionCausesTransactionCommit()	Y
	dataDefinitionIgnoredInTransactions()	Y
	deletesAreDetected(int type)	Y
	doesMaxRowSizeIncludeBlobs()	Y

Interface name	Method name	jdbc4/jdbc41
	getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern)	N
	getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	Y
	getCatalogs()	Y
	getCatalogSeparator()	Y
	getCatalogTerm()	Y
	getClientInfoProperties()	Y
	getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern)	Y
	getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	Y
	getConnection()	Y
	getCrossReference(String parentCatalog, String parentSchema, String parentTable, String foreignCatalog, String foreignSchema, String foreignTable)	Y
	getDatabaseMajorVersion()	Y
	getDatabaseMinorVersion()	Y
	getDatabaseProductName()	Y
	getDatabaseProductVersion()	Y
	getDefaultTransactionIsolation()	Y
	getDriverMajorVersion()	Y
	getDriverMinorVersion()	Y
	getDriverName()	Y
	getDriverVersion()	Y
	getExportedKeys(String catalog, String schema, String table)	Y
	getExtraNameCharacters()	Y
	getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern, String columnNamePattern)	N
	getFunctions(String catalog, String schemaPattern, String functionNamePattern)	N
	getIdentifierQuoteString()	Y
	getImportedKeys(String catalog, String schema, String table)	Y
	getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate)	Y
	getJDBCMajorVersion()	Y
	getJDBCMinorVersion()	Y
	getMaxBinaryLiteralLength()	Y

Interface name	Method name	jdbc4/jdbc41
	getMaxCatalogNameLength()	Y
	getMaxCharLiteralLength()	Y
	getMaxColumnNameLength()	Y
	getMaxColumnsInGroupBy()	Y
	getMaxColumnsInIndex()	Y
	getMaxColumnsInOrderBy()	Y
	getMaxColumnsInSelect()	Y
	getMaxColumnsInTable()	Y
	getMaxConnections()	Y
	getMaxCursorNameLength()	Y
	getMaxIndexLength()	Y
	getMaxProcedureNameLength()	Y
	getMaxRowSize()	Y
	getMaxSchemaNameLength()	Y
	getMaxStatementLength()	Y
	getMaxStatements()	Y
	getMaxTableNameLength()	Y
	getMaxTablesInSelect()	Y
	getMaxUserNameLength()	Y
	getNumericFunctions()	Y
	getPrimaryKeys(String catalog, String schema, String table)	Y
	getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	Y
	getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	Y
	getProcedureTerm()	Y
	getResultSetHoldability()	Y
	getRowIdLifetime()	N
	getSchemas()	Y
	getSchemas(String catalog, String schemaPattern)	Y
	getSchemaTerm()	Y
	getSearchStringEscape()	Y
	getSQLKeywords()	Y
	getSQLStateType()	Y
	getStringFunctions()	Y
	getSuperTables(String catalog, String schemaPattern, String tableNamePattern)	N
	getSuperTypes(String catalog, String schemaPattern, String typeNamePattern)	N

Interface name	Method name	jdbc4/jdbc41
	getSystemFunctions()	Y
	getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)	Y
	getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	Y
	getTableTypes()	Y
	getTimeDateFunctions()	Y
	getTypeInfo()	Y
	getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)	Y
	getURL()	Y
	getUserName()	Y
	getVersionColumns(String catalog, String schema, String table)	Y
	insertsAreDetected(int type)	Y
	isCatalogAtStart()	Y
	isReadOnly()	Y
	locatorsUpdateCopy()	Y
	nullPlusNonNullIsNull()	Y
	nullsAreSortedAtEnd()	Y
	nullsAreSortedAtStart()	Y
	nullsAreSortedHigh()	Y
	nullsAreSortedLow()	Y
	othersDeletesAreVisible(int type)	Y
	othersInsertsAreVisible(int type)	Y
	othersUpdatesAreVisible(int type)	Y
	ownDeletesAreVisible(int type)	Y
	ownInsertsAreVisible(int type)	Y
	ownUpdatesAreVisible(int type)	Y
	storesLowerCaseIdentifiers()	Y
	storesLowerCaseQuotedIdentifiers()	Y
	storesMixedCaseIdentifiers()	Y
	storesMixedCaseQuotedIdentifiers()	Y
	storesUpperCaseIdentifiers()	Y
	storesUpperCaseQuotedIdentifiers()	Y
	supportsAlterTableWithAddColumn()	Y
	supportsAlterTableWithDropColumn()	Y
	supportsANSI92EntryLevelSQL()	Y
	supportsANSI92FullSQL()	Y
	supportsANSI92IntermediateSQL()	Y

Interface name	Method name	jdbc4/jdbc41
	supportsBatchUpdates()	Y
	supportsCatalogsInDataManipulation()	Y
	supportsCatalogsInIndexDefinitions()	Y
	supportsCatalogsInPrivilegeDefinitions()	Y
	supportsCatalogsInProcedureCalls()	Y
	supportsCatalogsInTableDefinitions()	Y
	supportsColumnAliasing()	Y
	supportsConvert()	Y
	supportsConvert(int fromType, int toType)	Y
	supportsCoreSQLGrammar()	Y
	supportsCorrelatedSubqueries()	Y
	supportsDataDefinitionAndDataManipulationTransactions()	Y
	supportsDataManipulationTransactionsOnly()	Y
	supportsDifferentTableCorrelationNames()	Y
	supportsExpressionsInOrderBy()	Y
	supportsExtendedSQLGrammar()	Y
	supportsFullOuterJoins()	Y
	supportsGetGeneratedKeys()	Y
	supportsGroupBy()	Y
	supportsGroupByBeyondSelect()	Y
	supportsGroupByUnrelated()	Y
	supportsIntegrityEnhancementFacility()	Y
	supportsLikeEscapeClause()	Y
	supportsLimitedOuterJoins()	Y
	supportsMinimumSQLGrammar()	Y
	supportsMixedCaseIdentifiers()	Y
	supportsMixedCaseQuotedIdentifiers()	Y
	supportsMultipleOpenResults()	Y
	supportsMultipleResultSets()	Y
	supportsMultipleTransactions()	Y
	supportsNamedParameters()	Y
	supportsNonNullableColumns()	Y
	supportsOpenCursorsAcrossCommit()	Y
	supportsOpenCursorsAcrossRollback()	Y
	supportsOpenStatementsAcrossCommit()	Y
	supportsOpenStatementsAcrossRollback()	Y
	supportsOrderByUnrelated()	Y
	supportsOuterJoins()	Y
	supportsPositionedDelete()	Y

Interface name	Method name	jdbc4/jdbc41
	supportsPositionedUpdate()	Y
	supportsResultSetConcurrency(int type, int concurrency)	Y
	supportsResultSetHoldability(int holdability)	Y
	supportsResultSetType(int type)	Y
	supportsSavepoints()	Y
	supportsSchemasInDataManipulation()	Y
	supportsSchemasInIndexDefinitions()	Y
	supportsSchemasInPrivilegeDefinitions()	Y
	supportsSchemasInProcedureCalls()	Y
	supportsSchemasInTableDefinitions()	Y
	supportsSelectForUpdate()	Y
	supportsStatementPooling()	Y
	supportsStoredFunctionsUsingCallSyntax()	Y
	supportsStoredProcedures()	Y
	supportsSubqueriesInComparisons()	Y
	supportsSubqueriesInExists()	Y
	supportsSubqueriesInIns()	Y
	supportsSubqueriesInQuantifieds()	Y
	supportsTableCorrelationNames()	Y
	supportsTransactionIsolationLevel(int level)	Y
	supportsTransactions()	Y
	supportsUnion()	Y
	supportsUnionAll()	Y
	updatesAreDetected(int type)	Y
	usesLocalFilePerTable()	Y
	usesLocalFiles()	Y
Driver	acceptsURL(String url)	Y
	connect(String url, Properties info)	Y
	getMajorVersion()	Y
	getMinorVersion()	Y
	getPropertyInfo(String url, Properties info)	Y
	jdbcCompliant()	Y
ParameterMetaData	getParameterClassName(int param)	Y
	getParameterCount()	Y
	getParameterMode(int param)	Y
	getParameterType(int param)	Y
	getParameterTypeName(int param)	Y
	getPrecision(int param)	Y
	getScale(int param)	Y

Interface name	Method name	jdbc4/jdbc41
	isNullable(int param)	Y
	isSigned(int param)	Y
PreparedStatement	addBatch()	Y
	clearParameters()	Y
	execute()	Y
	executeQuery()	Y
	executeUpdate()	Y
	getMetaData()	Y
	getParameterMetaData()	Y
	setArray(int parameterIndex, Array x)	Y
	setAsciiStream(int parameterIndex, InputStream x)	N
	setAsciiStream(int parameterIndex, InputStream x, int length)	Y
	setAsciiStream(int parameterIndex, InputStream x, long length)	N
	setBigDecimal(int parameterIndex, BigDecimal x)	Y
	setBinaryStream(int parameterIndex, InputStream x)	N
	setBinaryStream(int parameterIndex, InputStream x, int length)	Y
	setBinaryStream(int parameterIndex, InputStream x, long length)	Y
	setBlob(int parameterIndex, Blob x)	Y
	setBlob(int parameterIndex, InputStream inputStream)	N
	setBlob(int parameterIndex, InputStream inputStream, long length)	N
	setBoolean(int parameterIndex, boolean x)	Y
	setByte(int parameterIndex, byte x)	Y
	setBytes(int parameterIndex, byte[] x)	Y
	setCharacterStream(int parameterIndex, Reader reader)	N
	setCharacterStream(int parameterIndex, Reader reader, int length)	Y
	setCharacterStream(int parameterIndex, Reader reader, long length)	N
	setClob(int parameterIndex, Clob x)	Y
	setClob(int parameterIndex, Reader reader)	N
	setClob(int parameterIndex, Reader reader, long length)	N
	setDate(int parameterIndex, Date x)	Y
	setDate(int parameterIndex, Date x, Calendar cal)	Y
	setDouble(int parameterIndex, double x)	Y

Interface name	Method name	jdbc4/jdbc41
	setFloat(int parameterIndex, float x)	Y
	setInt(int parameterIndex, int x)	Y
	setLong(int parameterIndex, long x)	Y
	setNCharacterStream(int parameterIndex, Reader value)	N
	setNCharacterStream(int parameterIndex, Reader value, long length)	Y
	setNClob(int parameterIndex, NClob value)	N
	setNClob(int parameterIndex, Reader reader)	N
	setNClob(int parameterIndex, Reader reader, long length)	N
	setNString(int parameterIndex, String value)	Y
	setNull(int parameterIndex, int sqlType)	Y
	setNull(int parameterIndex, int sqlType, String typeName)	Y
	setObject(int parameterIndex, Object x)	Y
	setObject(int parameterIndex, Object x, int targetSqlType)	Y
	setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength)	Y
	setRef(int parameterIndex, Ref x)	N
	setRowId(int parameterIndex, RowId x)	N
	setShort(int parameterIndex, short x)	Y
	setSQLXML(int parameterIndex, SQLXML xmlObject)	Y
	setString(int parameterIndex, String x)	Y
	setTime(int parameterIndex, Time x)	Y
	setTime(int parameterIndex, Time x, Calendar cal)	Y
	setTimestamp(int parameterIndex, Timestamp x)	Y
	setTimestamp(int parameterIndex, Timestamp x, Calendar cal)	Y
	setUnicodeStream(int parameterIndex, InputStream x, int length)	Y
	setURL(int parameterIndex, URL x)	Y
ResultSet	absolute(int row)	Y
	afterLast()	Y
	beforeFirst()	Y
	cancelRowUpdates()	Y
	clearWarnings()	Y
	close()	Y
	deleteRow()	Y
	findColumn(String columnLabel)	Y

Interface name	Method name	jdbc4/jdbc41
	first()	Y
	getArray(int columnIndex)	Y
	getArray(String columnLabel)	Y
	getAsciiStream(int columnIndex)	Y
	getAsciiStream(String columnLabel)	Y
	getBigDecimal(int columnIndex)	Y
	getBigDecimal(int columnIndex, int scale)	Y
	getBigDecimal(String columnLabel)	Y
	getBigDecimal(String columnLabel, int scale)	Y
	getBinaryStream(int columnIndex)	Y
	getBinaryStream(String columnLabel)	Y
	getBlob(int columnIndex)	Y
	getBlob(String columnLabel)	Y
	getBoolean(int columnIndex)	Y
	getBoolean(String columnLabel)	Y
	getByte(int columnIndex)	Y
	getByte(String columnLabel)	Y
	getBytes(int columnIndex)	Y
	getBytes(String columnLabel)	Y
	getCharacterStream(int columnIndex)	Y
	getCharacterStream(String columnLabel)	Y
	getClob(int columnIndex)	Y
	getClob(String columnLabel)	Y
	getConcurrency()	Y
	getCursorName()	Y
	getDate(int columnIndex)	Y
	getDate(int columnIndex, Calendar cal)	Y
	getDate(String columnLabel)	Y
	getDate(String columnLabel, Calendar cal)	Y
	getDouble(int columnIndex)	Y
	getDouble(String columnLabel)	Y
	getFetchDirection()	Y
	getFetchSize()	Y
	getFloat(int columnIndex)	Y
	getFloat(String columnLabel)	Y
	getHoldability()	N
	getInt(int columnIndex)	Y
	getInt(String columnLabel)	Y
	getLong(int columnIndex)	Y

Interface name	Method name	jdbc4/jdbc41
	getLong(String columnLabel)	Y
	getMetaData()	Y
	getNCharacterStream(int columnIndex)	Y
	getNCharacterStream(String columnLabel)	N
	getNClob(int columnIndex)	N
	getNClob(String columnLabel)	N
	getNString(int columnIndex)	Y
	getNString(String columnLabel)	N
	getObject(int columnIndex)	Y
	getObject(int columnIndex, Map<String,Class<?>> map)	Y
	getObject(String columnLabel)	Y
	getObject(String columnLabel, Map<String,Class<?>> map)	Y
	getRef(int columnIndex)	Y
	getRef(String columnLabel)	Y
	getRow()	Y
	getRowId(int columnIndex)	N
	getRowId(String columnLabel)	N
	getShort(int columnIndex)	Y
	getShort(String columnLabel)	Y
	getSQLXML(int columnIndex)	Y
	getSQLXML(String columnLabel)	Y
	getStatement()	Y
	getString(int columnIndex)	Y
	getString(String columnLabel)	Y
	getTime(int columnIndex)	Y
	getTime(int columnIndex, Calendar cal)	Y
	getTime(String columnLabel)	Y
	getTime(String columnLabel, Calendar cal)	Y
	getTimestamp(int columnIndex)	Y
	getTimestamp(int columnIndex, Calendar cal)	Y
	getTimestamp(String columnLabel)	Y
	getTimestamp(String columnLabel, Calendar cal)	Y
	getType()	Y
	getUnicodeStream(int columnIndex)	Y
	getUnicodeStream(String columnLabel)	Y
	getURL(int columnIndex)	N
	getURL(String columnLabel)	N
	getWarnings()	Y

Interface name	Method name	jdbc4/jdbc41
	insertRow()	Y
	isAfterLast()	Y
	isBeforeFirst()	Y
	isClosed()	Y
	isFirst()	Y
	isLast()	Y
	last()	Y
	moveToCurrentRow()	Y
	moveToInsertRow()	Y
	next()	Y
	previous()	Y
	refreshRow()	Y
	relative(int rows)	Y
	rowDeleted()	Y
	rowInserted()	Y
	rowUpdated()	Y
	setFetchDirection(int direction)	Y
	setFetchSize(int rows)	Y
	updateArray(int columnIndex, Array x)	Y
	updateArray(String columnLabel, Array x)	Y
	updateAsciiStream(int columnIndex, InputStream x)	Y
	updateAsciiStream(int columnIndex, InputStream x, int length)	N
	updateAsciiStream(int columnIndex, InputStream x, long length)	N
	updateAsciiStream(String columnLabel, InputStream x)	Y
	updateAsciiStream(String columnLabel, InputStream x, int length)	N
	updateAsciiStream(String columnLabel, InputStream x, long length)	Y
	updateBigDecimal(int columnIndex, BigDecimal x)	Y
	updateBigDecimal(String columnLabel, BigDecimal x)	N
	updateBinaryStream(int columnIndex, InputStream x)	N
	updateBinaryStream(int columnIndex, InputStream x, int length)	Y
	updateBinaryStream(int columnIndex, InputStream x, long length)	N
	updateBinaryStream(String columnLabel, InputStream x)	N

Interface name	Method name	jdbc4/jdbc41
	updateBinaryStream(String columnLabel, InputStream x, int length)	N
	updateBinaryStream(String columnLabel, InputStream x, long length)	N
	updateBlob(int columnIndex, Blob x)	N
	updateBlob(int columnIndex, InputStream inputStream)	N
	updateBlob(int columnIndex, InputStream inputStream, long length)	N
	updateBlob(String columnLabel, Blob x)	N
	updateBlob(String columnLabel, InputStream inputStream)	N
	updateBlob(String columnLabel, InputStream inputStream, long length)	N
	updateBoolean(int columnIndex, boolean x)	Y
	updateBoolean(String columnLabel, boolean x)	Y
	updateByte(int columnIndex, byte x)	Y
	updateByte(String columnLabel, byte x)	Y
	updateBytes(int columnIndex, byte[] x)	Y
	updateBytes(String columnLabel, byte[] x)	Y
	updateCharacterStream(int columnIndex, Reader x)	N
	updateCharacterStream(int columnIndex, Reader x, int length)	Y
	updateCharacterStream(int columnIndex, Reader x, long length)	N
	updateCharacterStream(String columnLabel, Reader reader)	N
	updateCharacterStream(String columnLabel, Reader reader, int length)	Y
	updateCharacterStream(String columnLabel, Reader reader, long length)	N
	updateClob(int columnIndex, Clob x)	N
	updateClob(int columnIndex, Reader reader)	N
	updateClob(int columnIndex, Reader reader, long length)	N
	updateClob(String columnLabel, Clob x)	N
	updateClob(String columnLabel, Reader reader)	N
	updateClob(String columnLabel, Reader reader, long length)	N
	updateDate(int columnIndex, Date x)	Y
	updateDate(String columnLabel, Date x)	Y
	updateDouble(int columnIndex, double x)	Y
	updateDouble(String columnLabel, double x)	Y

Interface name	Method name	jdbc4/jdbc41
	updateFloat(int columnIndex, float x)	Y
	updateFloat(String columnLabel, float x)	Y
	updateInt(int columnIndex, int x)	Y
	updateInt(String columnLabel, int x)	Y
	updateLong(int columnIndex, long x)	Y
	updateLong(String columnLabel, long x)	Y
	updateNCharacterStream(int columnIndex, Reader x)	N
	updateNCharacterStream(int columnIndex, Reader x, long length)	Y
	updateNCharacterStream(String columnLabel, Reader reader)	N
	updateNCharacterStream(String columnLabel, Reader reader, long length)	N
	updateNClob(int columnIndex, NClob nClob)	N
	updateNClob(int columnIndex, Reader reader)	N
	updateNClob(int columnIndex, Reader reader, long length)	N
	updateNClob(String columnLabel, NClob nClob)	N
	updateNClob(String columnLabel, Reader reader)	N
	updateNClob(String columnLabel, Reader reader, long length)	N
	updateNString(int columnIndex, String nString)	Y
	updateNString(String columnLabel, String nString)	N
	updateNull(int columnIndex)	Y
	updateNull(String columnLabel)	Y
	updateObject(int columnIndex, Object x)	Y
	updateObject(int columnIndex, Object x, int scaleOrLength)	Y
	updateObject(String columnLabel, Object x)	Y
	updateObject(String columnLabel, Object x, int scaleOrLength)	Y
	updateRef(int columnIndex, Ref x)	N
	updateRef(String columnLabel, Ref x)	N
	updateRow()	Y
	updateRowId(int columnIndex, RowId x)	N
	updateRowId(String columnLabel, RowId x)	N
	updateShort(int columnIndex, short x)	Y
	updateShort(String columnLabel, short x)	Y
	updateSQLXML(int columnIndex, SQLXML xmlObject)	Y
	updateSQLXML(String columnLabel, SQLXML xmlObject)	Y

Interface name	Method name	jdbc4/jdbc41
	updateString(int columnIndex, String x)	Y
	updateString(String columnLabel, String x)	Y
	updateTime(int columnIndex, Time x)	Y
	updateTime(String columnLabel, Time x)	Y
	updateTimestamp(int columnIndex, Timestamp x)	Y
	updateTimestamp(String columnLabel, Timestamp x)	Y
	wasNull()	Y
ResultSetMetaData	isAutoIncrement(int column)	Y
	isCaseSensitive(int column)	Y
	getCatalogName(int column)	Y
	getColumnClassName(int column)	Y
	getColumnCount()	Y
	getColumnDisplaySize(int column)	Y
	getColumnLabel(int column)	Y
	getColumnName(int column)	Y
	getColumnType(int column)	Y
	getColumnTypeName(int column)	Y
	isCurrency(int column)	Y
	isDefinitelyWritable(int column)	Y
	isNullable(int column)	Y
	getPrecision(int column)	Y
	isReadOnly(int column)	Y
	getScale(int column)	Y
	getSchemaName(int column)	Y
	isSearchable(int column)	Y
	isSigned(int column)	Y
	getTableName(int column)	Y
	isWritable(int column)	Y
RowId	equals(Object obj)	N
	hashCode()	N
	toString()	N
	getBytes()	N
Savepoint	getSavepointId()	Y
	getSavepointName()	Y
SQLXML	free()	Y
	setBinaryStream()	Y
	getBinaryStream()	Y
	setCharacterStream()	Y
	getCharacterStream()	Y

Interface name	Method name	jdbc4/jdbc41
	setResult(Class resultClass)	Y
	getSource(Class sourceClass)	Y
	setString(String value)	Y
	getString()	Y
Statement	addBatch(String sql)	Y
	cancel()	Y
	clearBatch()	Y
	clearWarnings()	Y
	close()	Y
	execute(String sql)	Y
	execute(String sql, int autoGeneratedKeys)	Y
	execute(String sql, int[] columnIndexes)	Y
	execute(String sql, String[] columnNames)	Y
	executeBatch()	Y
	executeQuery(String sql)	Y
	executeUpdate(String sql)	Y
	executeUpdate(String sql, int autoGeneratedKeys)	Y
	executeUpdate(String sql, int[] columnIndexes)	Y
	executeUpdate(String sql, String[] columnNames)	Y
	isClosed()	Y
	getConnection()	Y
	setCursorName(String name)	Y
	setEscapeProcessing(boolean enable)	Y
	setFetchDirection(int direction)	Y
	getFetchDirection()	Y
	setFetchSize(int rows)	Y
	getFetchSize()	Y
	getGeneratedKeys()	Y
	setMaxFieldSize(int max)	Y
	getMaxFieldSize()	Y
	setMaxRows(int max)	Y
	getMaxRows()	Y
	getMoreResults()	Y
	getMoreResults(int current)	Y
	setPoolable(boolean poolable)	Y
	isPoolable()	Y
	setQueryTimeout(int seconds)	Y
	getQueryTimeout()	Y
	getResultSet()	Y

Interface name	Method name	jdbc4/jdbc41
	getResultSetConcurrency()	Y
	getResultSetHoldability()	Y
	getResultSetType()	Y
	getUpdateCount()	Y
	getWarnings()	Y

Y: Supported
N: Not supported

javax.sql

Interface name	Method name	jdbc4/jdbc41
ConnectionPool DataSource	getPooledConnection()	Y
	getPooledConnection(String user, String password)	Y
DataSource	createQueryObject(Class ifc)	N
	createQueryObject(Class ifc, DataSource ds)	N
	getConnection()	Y
	getConnection(String username, String password)	Y
PooledConnection	addConnectionEventListener(ConnectionEventListener listener)	Y
	addStatementEventListener(StatementEventListener listener)	Y
	close()	Y
	removeConnectionEventListener(ConnectionEventListener listener)	Y
	removeStatementEventListener(StatementEventListener listener)	N
	getConnection()	Y

Y: Supported
N: Not supported

F.1.2 PostgreSQL Fixed API

org.postgresql

Interface name	Method name	jdbc4/jdbc41
PGConnection	addDataType(java.lang.String type, java.lang.Class klass)	Y
	addDataType(java.lang.String type, java.lang.String name)	Y
	getBackendPID()	Y
	getCopyAPI()	Y
	getFastpathAPI()	Y
	getLargeObjectAPI()	Y
	getmaxStatements()	Y

Interface name	Method name	jdbc4/jdbc41
	getNotifications()	Y
	getPrepareThreshold()	Y
	setPrepareThreshold(int threshold)	Y
PGNotification	getName()	Y
	getParameter()	Y
	getPID()	Y
PGRefCursorResultSet	getRefCursor()	Y
PGResultSetMetaData	getBaseColumnName(int column)	Y
	getBaseSchemaName(int column)	Y
	getBaseTableName(int column)	Y
	getFormat(int column)	Y
PGStatement	getLastOID()	Y
	getPrepareThreshold()	Y
	isUseServerPrepare()	Y
	setPrepareThreshold(int threshold)	Y
	setUseServerPrepare(boolean flag)	Y

Y: Supported

org.postgresql.copy

Interface name	Method name	jdbc4/jdbc41
CopyIn	endCopy()	Y
	flushCopy()	Y
	writeToCopy(byte[] buf, int off, int siz)	Y
CopyOperation	cancelCopy()	Y
	getFieldCount()	Y
	getFieldFormat(int field)	Y
	getFormat()	Y
	getHandledRowCount()	Y
	isActive()	Y
CopyOut	readFromCopy()	Y
CopyManager	copyIn(java.lang.String sql)	Y
	copyIn(java.lang.String sql, java.io.InputStream from)	Y
	copyIn(java.lang.String sql, java.io.InputStream from, int bufferSize)	Y
	copyIn(java.lang.String sql, java.io.Reader from)	Y
	copyIn(java.lang.String sql, java.io.Reader from, int bufferSize)	Y
	copyOut(java.lang.String sql)	Y

Interface name	Method name	jdbc4/jdbc41
	copyOut(java.lang.String sql, java.io.OutputStream to)	Y
	copyOut(java.lang.String sql, java.io.Writer to)	Y
PGCopyInputStream	available()	Y
	cancelCopy()	Y
	close()	Y
	getFieldCount()	Y
	getFieldFormat(int field)	Y
	getFormat()	Y
	getHandledRowCount()	Y
	isActive()	Y
	read()	Y
	read(byte[] buf)	Y
	read(byte[] buf, int off, int siz)	Y
	readFromCopy()	Y
PGCopyOutputStream	cancelCopy()	Y
	close()	Y
	endCopy()	Y
	flush()	Y
	flushCopy()	Y
	getFieldCount()	Y
	getFieldFormat(int field)	Y
	getFormat()	Y
	getHandledRowCount()	Y
	isActive()	Y
	write(byte[] buf)	Y
	write(byte[] buf, int off, int siz)	Y
	write(int b)	Y
	writeToCopy(byte[] buf, int off, int siz)	Y

Y: Supported

org.postgresql.ds

Interface name	Method name	jdbc4/jdbc41
PGConnectionPoolDataSource		Y
PGPooledConnection	createConnectionEvent(java.sql.SQLException sqle)	Y
PGPoolingDataSource	addDataSource(java.lang.String dataSourceName)	Y
PGSimpleDataSource		Y

Y: Supported

org.postgresql.ds.common

Interface name	Method name	jdbc4/jdbc41
BaseDataSource	createReference()	Y
	getApplicationName()	Y
	getBinaryTransfer()	Y
	getBinaryTransferDisable()	Y
	getBinaryTransferEnable()	Y
	getCompatible()	Y
	getConnection()	Y
	getConnection(java.lang.String user, java.lang.String password)	Y
	getDatabaseName()	Y
	getDescription()	Y
	getLoginTimeout()	Y
	getLogLevel()	Y
	getLogWriter()	Y
	getmaxStatements()	Y
	getPassword()	Y
	getPortNumber()	Y
	getPrepareThreshold()	Y
	getProtocolVersion()	Y
	getReceiveBufferSize()	Y
	getReference()	Y
	getSendBufferSize()	Y
	getServerName()	Y
	getSocketTimeout()	Y
	getSsl()	Y
	getSslfactory()	Y
	getSslmode()	Y
	getSslrootcert()	Y
	getSslservercertcn()	Y
	getStringType()	Y
	getTargetServer()	Y
	getTcpKeepAlive()	Y
	getUnknownLength()	Y
	getUrl()	Y
	getUser()	Y
	initializeFrom(BaseDataSource source)	Y
	isColumnSanitiserDisabled()	Y

Interface name	Method name	jdbc4/jdbc41
	readBaseObject(java.io.ObjectInputStream in)	Y
	setApplicationName(java.lang.String applicationName)	Y
	setBinaryTransfer(boolean enabled)	Y
	setBinaryTransferDisable(java.lang.String oidList)	Y
	setBinaryTransferEnable(java.lang.String oidList)	Y
	setCompatible(java.lang.String compatible)	Y
	setDatabaseName(java.lang.String databaseName)	Y
	setDisableColumnSanitiser(boolean disableColumnSanitiser)	Y
	setLoginTimeout(int i)	Y
	setLogLevel(int logLevel)	Y
	setLogWriter(java.io.PrintWriter printWriter)	Y
	setmaxStatements(int casheSize)	Y
	setPassword(java.lang.String password)	Y
	setPortNumber(int portNumber)	Y
	setPrepareThreshold(int count)	Y
	setProtocolVersion(int protocolVersion)	Y
	setReceiveBufferSize(int nbytes)	Y
	setSendBufferSize(int nbytes)	Y
	setServerName(java.lang.String serverName)	Y
	setSocketTimeout(int seconds)	Y
	setSsl(boolean enabled)	Y
	setSslfactory(java.lang.String classname)	Y
	setSslmode(java.lang.String sslmode)	Y
	setSslrootcert(java.lang.String sslrootcert)	Y
	setSslservercertcn(java.lang.String sslservercertcn)	Y
	setStringType(java.lang.String stringType)	Y
	setTargetServer(java.lang.String targetServer)	Y
	setTcpKeepAlive(boolean enabled)	Y
	setUnknownLength(int unknownLength)	Y
	setUrl(String url)	Y
	setUser(java.lang.String user)	Y
	writeBaseObject(java.io.ObjectOutputStream out)	Y

Y: Supported

org.postgresql.fastpath

Interface name	Method name	jdbc4/jdbc41
Fastpath	addFunction(java.lang.String name, int fnid)	Y
	addFunctions(java.sql.ResultSet rs)	Y
	createOIDArg(long oid)	Y
	fastpath(int fnId, boolean resultType, FastpathArg[] args)	Y
	fastpath(java.lang.String name, boolean resulttype, FastpathArg[] args)	Y
	getData(java.lang.String name, FastpathArg[] args)	Y
	getID(java.lang.String name)	Y
	getInteger(java.lang.String name, FastpathArg[] args)	Y
	getOID(java.lang.String name, FastpathArg[] args)	Y
FastpathArg		Y

Y: Supported

org.postgresql.geometric

Interface name	Method name	jdbc4/jdbc41
PGbox	clone()	Y
	equals(java.lang.Object obj)	Y
	getValue()	Y
	hashCode()	Y
	lengthInBytes()	Y
	setByteValue(byte[] b, int offset)	Y
	setValue(java.lang.String value)	Y
	toBytes(byte[] bytes, int offset)	Y
PGcircle	clone()	Y
	equals(java.lang.Object obj)	Y
	getValue()	Y
	hashCode()	Y
	setValue(java.lang.String s)	Y
PGline	clone()	Y
	equals(java.lang.Object obj)	Y
	getValue()	Y
	hashCode()	Y
	setValue(java.lang.String s)	Y
PGlseg	clone()	Y
	equals(java.lang.Object obj)	Y
	getValue()	Y

Interface name	Method name	jdbc4/jdbc41
	hashCode()	Y
	setValue(java.lang.String s)	Y
PGpath	clone()	Y
	closePath()	Y
	equals(java.lang.Object obj)	Y
	getValue()	Y
	hashCode()	Y
	isClosed()	Y
	isOpen()	Y
	openPath()	Y
	setValue(java.lang.String s)	Y
PGpoint	equals(java.lang.Object obj)	Y
	getValue()	Y
	hashCode()	Y
	lengthInBytes()	Y
	move(double x, double y)	Y
	move(int x, int y)	Y
	setByteValue(byte[] b, int offset)	Y
	setLocation(int x, int y)	Y
	setLocation(java.awt.Point p)	Y
	setValue(java.lang.String s)	Y
	toBytes(byte[] b, int offset)	Y
	translate(double x, double y)	Y
	translate(int x, int y)	Y
PGpolygon	clone()	Y
	equals(java.lang.Object obj)	Y
	getValue()	Y
	hashCode()	Y
	setValue(java.lang.String s)	Y

Y: Supported

org.postgresql.largeobject

Interface name	Method name	jdbc4/jdbc41
BlobInputStream	close()	Y
	mark(int readlimit)	Y
	markSupported()	Y
	read()	Y
	reset()	Y
BlobOutputStream	close()	Y

Interface name	Method name	jdbc4/jdbc41
	flush()	Y
	write(byte[] buf, int off, int len)	Y
	write(int b)	Y
LargeObject	close()	Y
	copy()	Y
	getInputStream()	Y
	getLongOID()	Y
	getOID()	Y
	getOutputStream()	Y
	read(byte[] buf, int off, int len)	Y
	read(int len)	Y
	seek(int pos)	Y
	seek(int pos, int ref)	Y
	size()	Y
	tell()	Y
	truncate(int len)	Y
	write(byte[] buf)	Y
	write(byte[] buf, int off, int len)	Y
LargeObjectManager	create()	Y
	create(int mode)	Y
	createLO()	Y
	createLO(int mode)	Y
	delete(int oid)	Y
	delete(long oid)	Y
	open(int oid)	Y
	open(int oid, boolean commitOnClose)	Y
	open(int oid, int mode)	Y
	open(int oid, int mode, boolean commitOnClose)	Y
	open(long oid)	Y
	open(long oid, boolean commitOnClose)	Y
	open(long oid, int mode)	Y
	open(long oid, int mode, boolean commitOnClose)	Y
	unlink(int oid)	Y
	unlink(long oid)	Y

Y: Supported

org.postgresql.ssl

Interface name	Method name	jdbc4/jdbc41
DbKeyStoreSocketFactory	getKeyStorePassword()	Y
	getKeyStoreStream()	Y
DbKeyStoreSocketFactory.DbKeyStoreSocketException		Y
NonValidatingFactory		Y
NonValidatingFactory.NonValidatingTM	checkClientTrusted(java.security.cert.X509Certificate[] certs, java.lang.String authType)	Y
	checkServerTrusted(java.security.cert.X509Certificate[] certs, java.lang.String authType)	Y
	getAcceptedIssuers()	Y
WrappedFactory	createSocket(java.net.InetAddress host, int port)	Y
	createSocket(java.net.InetAddress address, int port, java.net.InetAddress localAddress, int localPort)	Y
	createSocket(java.net.Socket socket, java.lang.String host, int port, boolean autoClose)	Y
	createSocket(java.lang.String host, int port)	Y
	createSocket(java.lang.String host, int port, java.net.InetAddress localHost, int localPort)	Y
	getDefaultCipherSuites()	Y
	getSupportedCipherSuites()	Y

Y: Supported

org.postgresql.util

Interface name	Method name	jdbc4/jdbc41
PGInterval	add(java.util.Calendar cal)	Y
	add(java.util.Date date)	Y
	add(PGInterval interval)	Y
	equals(java.lang.Object obj)	Y
	getDays()	Y
	getHours()	Y
	getMinutes()	Y
	getMonths()	Y
	getSeconds()	Y
	getValue()	Y
	getYears()	Y
	hashCode()	Y
	scale(int factor)	Y
	setDays(int days)	Y

Interface name	Method name	jdbc4/jdbc41
	setHours(int hours)	Y
	setMinutes(int minutes)	Y
	setMonths(int months)	Y
	setSeconds(double seconds)	Y
	setValue(int years, int months, int days, int hours, int minutes, double seconds)	Y
	setValue(java.lang.String value)	Y
	setYears(int years)	Y
PGJDBCMain	main(java.lang.String[] args)	Y
PGmoney	equals(java.lang.Object obj)	Y
	getValue()	Y
	setValue(java.lang.String s)	Y
PGobject	clone()	Y
	equals(java.lang.Object obj)	Y
	getType()	Y
	getValue()	Y
	setType(java.lang.String type)	Y
	setValue(java.lang.String value)	Y
	toString()	Y
ServerErrorMessage	getColumn()	Y
	getConstraint()	Y
	getDatatype()	Y
	getDetail()	Y
	getFile()	Y
	getHint()	Y
	getInternalPosition()	Y
	getInternalQuery()	Y
	getLine()	Y
	getMessage()	Y
	getPosition()	Y
	getRoutine()	Y
	getSchema()	Y
	getSeverity()	Y
	getSQLState()	Y
	getTable()	Y
	getWhere()	Y
	toString()	Y

Y: Supported

org.postgresql.xa

Interface name	Method name	jdbc4/jdbc41
PGXADataSource		Y

Y: Supported

ConnectionPoolDataSource

Interface name	Method name	jdbc4/jdbc41
ConnectionPoolDataSource	getLoginTimeout()	Y
	getLogWriter()	Y
	setLoginTimeout(int seconds)	Y
	setLogWriter(PrintWriter out)	Y

Y: Supported

F.2 ODBC Driver

F.2.1 List of Supported APIs

The following table shows the support status of APIs:

Function name	Support status
SQLAllocConnect	Y
SQLAllocEnv	Y
SQLAllocHandle	Y
SQLAllocStmt	Y
SQLBindCol	Y
SQLBindParameter	Y
SQLBindParam	Y
SQLBrowseConnect	N
SQLBulkOperations	Y
SQLCancel	Y
SQLCancelHandle	N
SQLCloseCursor	Y
SQLColAttribute	Y
SQLColAttributeW	Y
SQLColAttributes	Y
SQLColAttributesW	Y
SQLColumnPrivileges	Y
SQLColumnPrivilegesW	Y
SQLColumns	Y
SQLColumnsW	Y
SQLCompleteAsync	N
SQLConnect	Y

Function name	Support status
SQLConnectW	Y
SQLCopyDesc	Y
SQLDataSources	Y
SQLDataSourcesW	Y
SQLDescribeCol	Y
SQLDescribeColW	Y
SQLDescribeParam	Y
SQLDisconnect	Y
SQLDriverConnect	Y
SQLDriverConnectW	Y
SQLDrivers	N
SQLEndTran	Y
SQLError	Y
SQLErrorW	Y
SQLExecDirect	Y
SQLExecDirectW	Y
SQLExecute	Y
SQLExtendedFetch	Y
SQLFetch	Y
SQLFetchScroll	Y
SQLForeignKeys	Y
SQLForeignKeysW	Y
SQLFreeConnect	Y
SQLFreeEnv	Y
SQLFreeHandle	Y
SQLFreeStmt	Y
SQLGetConnectAttr	Y
SQLGetConnectAttrW	Y
SQLGetConnectOption	Y
SQLGetConnectOptionW	Y
SQLGetCursorName	Y
SQLGetCursorNameW	Y
SQLGetData	Y
SQLGetDescField	Y
SQLGetDescFieldW	Y
SQLGetDescRec	Y
SQLGetDescRecW	Y
SQLGetDiagField	Y
SQLGetDiagFieldW	Y

Function name	Support status
SQLGetDiagRec	Y
SQLGetDiagRecW	Y
SQLGetEnvAttr	Y
SQLGetFunctions	Y
SQLGetInfo	Y
SQLGetInfoW	Y
SQLGetStmtAttr	Y
SQLGetStmtAttrW	Y
SQLGetStmtOption	Y
SQLGetTypeInfo	Y
SQLGetTypeInfoW	Y
SQLMoreResults	Y
SQLNativeSql	Y
SQLNativeSqlW	Y
SQLNumParams	Y
SQLNumResultCols	Y
SQLParamData	Y
SQLParamOptions	Y
SQLPrepare	Y
SQLPrepareW	Y
SQLPrimaryKeys	Y
SQLPrimaryKeysW	Y
SQLProcedureColumns	Y
SQLProcedureColumnsW	Y
SQLProcedures	Y
SQLProceduresW	Y
SQLPutData	Y
SQLRowCount	Y
SQLSetConnectAttr	Y
SQLSetConnectAttrW	Y
SQLSetConnectOption	Y
SQLSetConnectOptionW	Y
SQLSetCursorName	Y
SQLSetCursorNameW	Y
SQLSetDescField	Y
SQLSetDescRec	Y
SQLSetEnvAttr	Y
SQLSetParam	Y
SQLSetPos	Y

Function name	Support status
SQLSetScrollOptions	N
SQLSetStmtAttr	Y
SQLSetStmtAttrW	Y
SQLSetStmtOption	Y
SQLSpecialColumns	Y
SQLSpecialColumnsW	Y
SQLStatistics	Y
SQLStatisticsW	Y
SQLTablePrivileges	Y
SQLTablePrivilegesW	Y
SQLTables	Y
SQLTablesW	Y
SQLTransact	Y

Y: Supported
N: Not supported

F.3 .NET Data Provider

There are the following two ways to develop applications using Fujitsu Npgsql .NET Data Provider:

- Use the Fujitsu Npgsql.NET Data Provider API (classes and methods) directly.

  Fujitsu Npgsql .NET Data Provider is created based on the open source software Npgsql. Refer to the "Npgsql - .Net Data Provider for Postgresql" for information on the APIs:

  See

  Refer to the Installation and Setup Guide for Client for the version of Npgsql that Fujitsu Npgsql .NET Data Provider is based on.

- Use the API of the .NET System.Data.Common namespace

  It is possible to create applications that do not rely on a provider when you use the System.Data.Common namespace. Refer to the "Writing Provider-Independent Code in ADO.NET" in MSDN Library for information.

The following table indicates whether the System.Data.Common namespace API is supported. Note that API marked "Npgsql extension API" do not exist in the System.Data.Common namespace , but rather are API extended by Npgsql.

Table F.1 Reference

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
DbCommand	NpgsqlCommand	CommandText	Public Property	Y	
		CommandTimeout	Public Property	Y	
		CommandType	Public Property	Y	
		Connection	Public Property	Y	
		Container	Public Property	Y	

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		DesignTimeVisible	Public Property	Y	
		Parameters	Public Property	Y	
		Site	Public Property	Y	
		Transaction	Public Property	Y	
		UpdatedRowSource	Public Property	Y	
		LastInsertedOID	Public Property	Y	Npgsql extension API
		NpgsqlCommand	Public Constructor	Y	Npgsql extension API
		Cancel	Public method	Y	
		Clone	Public method	Y	
		CreateObjRef	Public method	Y	
		CreateParameter	Public method	Y	
		Dispose	Public method	Y	
		Equals	Public method	Y	
		ExecuteNnQuery	Public method	Y	
		ExecuteReader	Public method	Y	
		ExecuteScalar	Public method	Y	
		GetHashCode	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetType	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		Prepare	Public method	Y	
		ResetCommandTimeout	Public method	N	
		ToString	Public method	Y	
		Disposed	Public event	Y	
DbCommandBuilder	NpgsqlCommandBuilder	CatalogLocation	Public Property	Y	
		CatalogSeparator	Public Property	Y	
		ConflictOption	Public Property	Y	
		Container	Public Property	Y	
		DataAdapter	Public Property	Y	
		SchemaSeparator	Public Property	Y	
		SetAllValues	Public Property	Y	
		QuotePrefix	Public Property	Y	
		QuoteSuffix	Public Property	Y	
		Site	Public Property	Y	
		NpgsqlCommandBuilder	Public Constructor	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		CreateObjRef	Public method	Y	
		Dispose	Public method	Y	
		Equals	Public method	Y	
		GetDeleteCommand	Public method	Y	
		GetHashCode	Public method	Y	
		GetInsertCommand	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetType	Public method	Y	
		GetUpdateCommand	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		QuoteIdentifier	Public method	Y	
		RefreshSchema	Public method	Y	
		ToString	Public method	Y	
		UnquoteIdentifier	Public method	Y	
		Disposed	Public event	Y	
DbConnection	NpgsqlConnection	ConnectionString	Public Property	Y	
		ConnectionTimeout	Public Property	Y	
		Container	Public Property	Y	
		Database	Public Property	Y	
		Datasource	Public Property	Y	
		Host	Public Property	Y	
		Port	Public Property	Y	
		ServerVersion	Public Property	Y	
		Site	Public Property	Y	
		State	Public Property	Y	
		BackendProtocolVersion	Public Property	Y	Npgsql extension API
		ConnectionLifeTime	Public Property	Y	Npgsql extension API
		CommandTimeout	Public Property	Y	Npgsql extension API
		FullState	Public Property	Y	Npgsql extension API
		NpgsqlCompatibilityVersion	Public Property	Y	Npgsql extension API
		PostgreSqlVersion	Public Property	Y	Npgsql extension API
		PreloadReader	Public Property	Y	Npgsql extension API
		ProcessID	Public Property	Y	Npgsql extension API
		SSL	Public Property	Y	Npgsql extension API
		SyncNotification	Public Property	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		UseExtendedTypes	Public Property	Y	Npgsql extension API
		NpgsqlConnection	Public Constructor	Y	Npgsql extension API
		BeginTransaction	Public method	Y	
		ChangeDatabase	Public method	Y	
		ClearPool	Public method	Y	
		Clone	Public method	Y	
		Close	Public method	Y	
		CreateCommand	Public method	Y	
		CreateObjRef	Public method	Y	
		Dispose	Public method	Y	
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetSchema	Public method	Y	
		GetType	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		Open	Public method	Y	
		ToString	Public method	Y	
		EnlistTransaction	Public method	Y	
		Disposed	Public event	Y	
		StateChange	Public event	Y	
		CertificateSelectionCallback	Public event	Y	Npgsql extension API
		CertificateValidationCallback	Public event	Y	Npgsql extension API
		Notice	Public event	Y	Npgsql extension API
		Notification	Public event	Y	Npgsql extension API
		PrivateKeySelectionCallback	Public event	Y	Npgsql extension API
		ProvideClientCertificatesCallback	Public event	Y	Npgsql extension API
DbConnectionStringBuilder	DbConnectionStringBuilder	BrowsableConnectionString	Public Property	Y	
		ConnectionLifetime	Public Property	Y	
		ConnectionString	Public Property	Y	
		Count	Public Property	Y	
		IsFixedSize	Public Property	Y	
		IsReadOnly	Public Property	Y	

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		Item	Public Property	Y	
		Keys	Public Property	Y	
		MaxPoolSize	Public Property	Y	
		MinPoolSize	Public Property	Y	
		Password	Public Property	Y	
		Pooling	Public Property	Y	
		Port	Public Property	Y	
		Values	Public Property	Y	
		CommandTimeout	Public Property	Y	Npgsql extension API
		Compatible	Public Property	Y	Npgsql extension API
		Database	Public Property	Y	Npgsql extension API
		Encoding	Public Property	Y	Npgsql extension API
		Enlist	Public Property	Y	Npgsql extension API
		Host	Public Property	Y	Npgsql extension API
		IntegratedSecurity	Public Property	Y	Npgsql extension API
		PreloadReader	Public Property	Y	Npgsql extension API
		Protocol	Public Property	Y	Npgsql extension API
		SearchPath	Public Property	Y	Npgsql extension API
		SSL	Public Property	Y	Npgsql extension API
		SslMode	Public Property	Y	Npgsql extension API
		SyncNotification	Public Property	Y	Npgsql extension API
		Timeout	Public Property	Y	Npgsql extension API
		UseExtendedTypes	Public Property	Y	Npgsql extension API
		UserName	Public Property	Y	Npgsql extension API
		NpgsqlConnectionStringBuilder	Public Constructor	Y	Npgsql extension API
		Add	Public method	Y	
		Clear	Public method	Y	
		ContainsKey	Public method	Y	
		Equals	Public method	Y	
		EquivalentTo	Public method	Y	
		GetHashCode	Public method	Y	
		GetType	Public method	Y	
		Remove	Public method	Y	
		ShouldSerialize	Public method	Y	
		ToString	Public method	Y	
		TryGetValue	Public method	Y	

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		Clone	Public method	Y	Npgsql extension API
DbDataAdapter	NpgsqlDataAdapter	AcceptChangesDuringFill	Public Property	Y	
		AcceptChangesDuringUpdate	Public Property	Y	
		Container	Public Property	Y	
		ContinueUpdateOnError	Public Property	Y	
		DeleteCommand	Public Property	Y	
		FillLoadOption	Public Property	Y	
		InsertCommand	Public Property	Y	
		MissingMappingAction	Public Property	Y	
		MissingSchemaAction	Public Property	Y	
		ReturnProviderSpecificTypes	Public Property	Y	
		SelectCommand	Public Property	Y	
		Site	Public Property	Y	
		TableMappings	Public Property	Y	
		UpdateBatchSize	Public Property	Y	
		UpdateCommand	Public Property	Y	
		NpgsqlDataAdapter	Public Constructor	Y	Npgsql extension API
		CreateObjRef	Public method	Y	
		Dispose	Public method	Y	
		Equals	Public method	Y	
		Fill	Public method	Y	
		FillSchema	Public method	Y	
		GetFillParameters	Public method	Y	
		GetHashCode	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetType	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		ToString	Public method	Y	
		Update	Public method	Y	
		ResetFillLoadOption	Public method	Y	Npgsql extension API
		ShouldSerializeAcceptChangesDuringFill	Public method	Y	Npgsql extension API
		ShouldSerializeFillLoadOption	Public method	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		Disposed	Public event	Y	
		FillError	Public event	Y	
		RowUpdated	Public event	Y	
		RowUpdating	Public event	Y	
DbDataReader	NpgsqlDataReader	Depth	Public Property	Y	
		IsClosed	Public Property	Y	
		RecordsAffected	Public Property	Y	
		FieldCount	Public Property	Y	
		HasRows	Public Property	Y	
		Item	Public Property	Y	
		VisibleFieldCount	Public Property	Y	
		Public Constructor	Public method	Y	
		Close	Public method	Y	
		CreateObjRef	Public method	Y	
		Equals	Public method	Y	
		GetBoolean	Public method	Y	
		GetByte	Public method	N	
		GetBytes	Public method	Y	
		GetChar	Public method	Y	
		GetChars	Public method	Y	
		GetDataTypeName	Public method	Y	
		GetDateTime	Public method	Y	
		GetDecimal	Public method	Y	
		GetDouble	Public method	Y	
		GetFieldType	Public method	Y	
		GetFloat	Public method	Y	
		GetGuid	Public method	Y	
		GetHashCode	Public method	Y	
		GetInt16	Public method	Y	
		GetInt32	Public method	Y	
		GetInt64	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetName	Public method	Y	
		GetOrdinal	Public method	Y	
		GetSchemaTable	Public method	Y	
		GetString	Public method	Y	
		GetType	Public method	Y	

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		GetValue	Public method	Y	
		GetValues	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		IsDBNull	Public method	Y	
		NextResult	Public method	Y	
		Read	Public method	Y	
		ToString	Public method	Y	
		Dispose	Public method	Y	
		GetBitString	Public method	Y	Npgsql extension API
		GetData	Public method	Y	
		GetDataTypeOID	Public method	Y	Npgsql extension API
		GetDate	Public method	Y	Npgsql extension API
		GetEnumerator	Public method	Y	
		GetFieldDbType	Public method	Y	Npgsql extension API
		GetFieldNpgsqlDbType	Public method	Y	Npgsql extension API
		GetInterval	Public method	Y	Npgsql extension API
		GetProviderSpecificFieldType	Public method	Y	
		GetProviderSpecificValue	Public method	Y	
		GetProviderSpecificValues	Public method	Y	
		GetTime	Public method	Y	Npgsql extension API
		GetTimeStamp	Public method	Y	Npgsql extension API
		GetTimeStampTZ	Public method	Y	Npgsql extension API
		GetTimeTZ	Public method	Y	Npgsql extension API
		HasOrdinal	Public method	Y	Npgsql extension API
		ReaderClosed	Public event	Y	Npgsql extension API
		Finalize	Protect method	Y	
		MemberwiseClone	Protect method	Y	
		Dispose	Protect method	Y	Npgsql extension API
		GetDbDataReader	Protect method	Y	Npgsql extension API
		SendClosedEvent	Protect method	Y	Npgsql extension API
DBError	NpgsqlError	Message	Public Property	Y	
		Number	Public Property	N	
		Source	Public Property	N	
		Code	Public Property	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		Detail	Public Property	Y	Npgsql extension API
		ErrorSql	Public Property	Y	Npgsql extension API
		File	Public Property	Y	Npgsql extension API
		Hint	Public Property	Y	Npgsql extension API
		InternalPosition	Public Property	Y	Npgsql extension API
		InternalQuery	Public Property	Y	Npgsql extension API
		Line	Public Property	Y	Npgsql extension API
		Position	Public Property	Y	Npgsql extension API
		Routine	Public Property	Y	Npgsql extension API
		Severity	Public Property	Y	Npgsql extension API
		Where	Public Property	Y	Npgsql extension API
		NpgsqlError	Public Constructor	Y	Npgsql extension API
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetType	Public method	Y	
		ToString	Public method	Y	
DBErrorCollection	Not supported	Count	Public Property	N	
		Item	Public Property	N	
		CopyTo	Public method	N	
		Equals	Public method	N	
		GetHashCode	Public method	N	
		GetType	Public method	N	
		ToString	Public method	N	
		Finalize	Protect method	N	
		MemberwiseClone	Protect method	N	
DBException	NpgsqlException	Data	Public Property	Y	
		ErrorCode	Public Property	Y	
		Errors	Public Property	Y	
		HelpLink	Public Property	Y	
		InnerException	Public Property	Y	
		Message	Public Property	Y	
		Number	Public Property	N	
		Source	Public Property	Y	
		StackTrace	Public Property	Y	
		TargetSite	Public Property	Y	
		BaseMessage	Public Property	Y	Npgsql extension API
		Code	Public Property	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		Detail	Public Property	Y	Npgsql extension API
		ErrorSql	Public Property	Y	Npgsql extension API
		File	Public Property	Y	Npgsql extension API
		Hint	Public Property	Y	Npgsql extension API
		Item	Public Property	Y	Npgsql extension API
		Line	Public Property	Y	Npgsql extension API
		Position	Public Property	Y	Npgsql extension API
		Routine	Public Property	Y	Npgsql extension API
		Severity	Public Property	Y	Npgsql extension API
		Where	Public Property	Y	Npgsql extension API
		NpgsqlException	Public Constructor	Y	Npgsql extension API
		Equals	Public method	Y	
		GetBaseException	Public method	Y	
		GetHashCode	Public method	Y	
		GetObjectData	Public method	Y	
		GetType	Public method	Y	
		ToString	Public method	Y	
DbProviderFactory	NpgsqlProviderFactory	Instance	Public field	Y	
		CanCreateDataSourceEnumerator	Public Property	Y	
		NpgsqlFactory	Public Constructor	Y	Npgsql extension API
		CreateCommand	Public method	Y	
		CreateCommandBuilder	Public method	Y	
		CreateConnection	Public method	Y	
		CreateConnectionStringBuilder	Public method	Y	
		CreateDataAdapter	Public method	Y	
		CreateDataSourceEnumerator	Public method	Y	
		CreateParameter	Public method	Y	
		CreatePermission	Public method	Y	
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetType	Public method	Y	
		ToString	Public method	Y	
		GetService	Public method	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
DBInfoMessageEventArgs	Not supported	Errors	Public Property	N	
		Message	Public Property	N	
		Source	Public Property	N	
DBParameter	NpgsqlParameter	DbType	Public Property	Y	
		Direction	Public Property	Y	
		IsNullable	Public Property	Y	
		ParameterName	Public Property	Y	
		Precision	Public Property	Y	
		Scale	Public Property	Y	
		Size	Public Property	Y	
		SourceColumn	Public Property	Y	
		SourceVersion	Public Property	Y	
		Value	Public Property	Y	
		NpgsqlDbType	Public Property	Y	Npgsql extension API
		NpgsqlValue	Public Property	Y	Npgsql extension API
		SourceColumnNullMapping	Public Property	Y	
		UseCast	Public Property	Y	Npgsql extension API
		NpgsqlParameter	Public Constructor	Y	Npgsql extension API
		CreateObjRef	Public method	Y	
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetType	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		ToString	Public method	Y	
		Clone	Public method	Y	
		ResetDbType	Public method	Y	
		Finalize	Protect method	N	
		MemberwiseClone	Protect method	N	
DBParameterCollection	NpgsqlParameterCollection	Count	Public Property	Y	
		Item	Public Property	Y	
		IsFixedSize	Public Property	Y	Npgsql extension API
		IsReadOnly	Public Property	Y	Npgsql extension API
		IsSynchronized	Public Property	Y	Npgsql extension API
		SyncRoot	Public Property	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		NpgsqlParameterCollection	Public Constructor	Y	Npgsql extension API
		Add	Public method	Y	
		AddRange	Public method	Y	
		Clear	Public method	Y	
		Contains	Public method	Y	
		CopyTo	Public method	Y	
		IndexOf	Public method	Y	
		Insert	Public method	Y	
		Remove	Public method	Y	
		RemoveAt	Public method	Y	
		CreateObjRef	Public method	Y	
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetType	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		ToString	Public method	Y	
		AddWithValue	Public method	Y	Npgsql extension API
		TryGetValue	Public method	Y	Npgsql extension API
		GetEnumerator	Public method	Y	Npgsql extension API
		Finalize	Protect method	N	
		MemberwiseClone	Protect method	N	
DBDataPermission	Not supported	AllowBlankPassword	Public Property	N	
		Add	Public method	N	
		Assert	Public method	N	
		Copy	Public method	N	
		Demand	Public method	N	
		Deny	Public method	N	
		Equals	Public method	N	
		FromXml	Public method	N	
		GetHashCode	Public method	N	
		GetType	Public method	N	
		Intersect	Public method	N	
		IsSubsetOf	Public method	N	
		IsUnrestricted	Public method	N	
		PermitOnly	Public method	N	

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		ToString	Public method	N	
		ToXml	Public method	N	
		Union	Public method	N	
DBDataPermissionAttribute	Not supported	Action	Public Property	N	
		AllowBlankPassword	Public Property	N	
		ConnectionString	Public Property	N	
		KeyRestrictionBehavior	Public Property	N	
		KeyRestrictions	Public Property	N	
		TypeId	Public Property	N	
		Unrestricted	Public Property	N	
		CreatePermission	Public method	N	
		Equals	Public method	N	
		GetHashCode	Public method	N	
		GetType	Public method	N	
		IsDefaultAttribute	Public method	N	
		Match	Public method	N	
		ShouldSerializeConnectionString	Public method	N	
		ShouldSerializeKeyRestrictions	Public method	N	
		ToString	Public method	N	
RowUpdatedEventArgs	NpgsqlRowUpdatedEventArgs	Command	Public Property	Y	
		Errors	Public Property	Y	
		Row	Public Property	Y	
		StatementType	Public Property	Y	
		Status	Public Property	Y	
		TableMapping	Public Property	Y	
		RecordsAffected	Public Property	Y	
		RowCount	Public Property	Y	Npgsql extension API
		NpgsqlRowUpdatedEventArgs	Public Constructor	Y	Npgsql extension API
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetType	Public method	Y	
		ToString	Public method	Y	
		CopyToRows	Public method	Y	Npgsql extension API
		Finalize	Protect method	Y	

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		MemberwiseClone	Protect method	Y	
RowUpdatingEventArgs	NpgsqlRowUpdatingEventArgs	Command	Public Property	Y	
		Errors	Public Property	Y	
		Row	Public Property	Y	
		StatementType	Public Property	Y	
		Status	Public Property	Y	
		TableMapping	Public Property	Y	
		NpgsqlRowUpdatingEventArgs	Public Constructor	Y	Npgsql extension API
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetType	Public method	Y	
		ToString	Public method	Y	
		BaseCommand	Protect Property	Y	
		Finalize	Protect method	Y	
		MemberwiseClone	Protect method	Y	
DBTransaction	NpgsqlTransaction	Connection	Public Property	Y	
		IsolationLevel	Public Property	Y	
		NpgsqlTransaction	Public Constructor	Y	Npgsql extension API
		Commit	Public method	Y	
		CreateObjRef	Public method	Y	
		Dispose	Public method	Y	
		Equals	Public method	Y	
		GetHashCode	Public method	Y	
		GetLifetimeService	Public method	Y	
		GetType	Public method	Y	
		InitializeLifetimeService	Public method	Y	
		Rollback	Public method	Y	
		ToString	Public method	Y	
		Save	Public method	Y	
		Finalize	Protect method	N	
		MemberwiseClone	Protect method	N	
None applicable	NpgsqlNotificationEventArgs	AdditionalInformation	Public Property	Y	Npgsql extension API
		Condition	Public Property	Y	Npgsql extension API
		PID	Public Property	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		GetType	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API
		Finalize	Protect method	Y	Npgsql extension API
		MemberwiseClone	Protect method	Y	Npgsql extension API
None applicable	NpgsqlEventLog	EchoMessages	Public Property	Y	Npgsql extension API
		Level	Public Property	Y	Npgsql extension API
		LogName	Public Property	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API
		GetType	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API
		Finalize	Protect method	Y	Npgsql extension API
		MemberwiseClone	Protect method	Y	Npgsql extension API
None applicable	NpgsqlCopyFormat	FieldCount	Public Property	Y	Npgsql extension API
		IsBinary	Public Property	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		FieldIsBinary	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API
		GetType	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API
None applicable	NpgsqlCopyIn	CopyBufferSize	Public Property	Y	Npgsql extension API
		CopyStream	Public Property	Y	Npgsql extension API
		FieldCount	Public Property	Y	Npgsql extension API
		IsActive	Public Property	Y	Npgsql extension API
		IsBinary	Public Property	Y	Npgsql extension API
		NpgsqlCommand	Public Property	Y	Npgsql extension API
		Cancel	Public method	Y	Npgsql extension API
		End	Public method	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		FieldIsBinary	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API
		GetType	Public method	Y	Npgsql extension API
		Start	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API
		Finalize	Protect method	Y	Npgsql extension API
		MemberwiseClone	Protect method	Y	Npgsql extension API
None applicable	NpgsqlCopyOut	CopyStream	Public Property	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		FieldCount	Public Property	Y	Npgsql extension API
		IsActive	Public Property	Y	Npgsql extension API
		IsBinary	Public Property	Y	Npgsql extension API
		NpgsqlCommand	Public Property	Y	Npgsql extension API
		Read	Public Property	Y	Npgsql extension API
		End	Public method	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		FieldIsBinary	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API
		GetType	Public method	Y	Npgsql extension API
		Start	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API
		Finalize	Protect method	Y	Npgsql extension API
		MemberwiseClone	Protect method	Y	Npgsql extension API
None applicable	NpgsqlCopySerializer	DEFAULT_BUFFER_SIZE	Public field	Y	Npgsql extension API
		DEFAULT_DELIMITER	Public field	Y	Npgsql extension API
		DEFAULT_ESCAPE	Public field	Y	Npgsql extension API
		DEFAULT_NULL	Public field	Y	Npgsql extension API
		DEFAULT_QUOTE	Public field	Y	Npgsql extension API
		DEFAULT_SEPARATOR	Public field	Y	Npgsql extension API
		EscapeSequenceFor	Protect method	Y	Npgsql extension API
		BufferSize	Public Property	Y	Npgsql extension API
		Delimiter	Public Property	Y	Npgsql extension API
		Escape	Public Property	Y	Npgsql extension API
		IsActive	Public Property	Y	Npgsql extension API
		Null	Public Property	Y	Npgsql extension API
		Separator	Public Property	Y	Npgsql extension API
		ToStream	Public Property	Y	Npgsql extension API
		AddBool	Public method	Y	Npgsql extension API
		AddDateTime	Public method	Y	Npgsql extension API
		AddInt32	Public method	Y	Npgsql extension API
		AddInt64	Public method	Y	Npgsql extension API
		AddNull	Public method	Y	Npgsql extension API
		AddNumber	Public method	Y	Npgsql extension API
		AddString	Public method	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		Close	Public method	Y	Npgsql extension API
		EndRow	Public method	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		Flush	Public method	Y	Npgsql extension API
		FlushFields	Public method	Y	Npgsql extension API
		FlushRows	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API
		GetType	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API
		EscapeSequenceBytes	Protect Property	Y	Npgsql extension API
		SpaceInBuffer	Protect Property	Y	Npgsql extension API
		StringsToEscape	Protect Property	Y	Npgsql extension API
		AddBytes	Protect method	Y	Npgsql extension API
		FieldAdded	Protect method	Y	Npgsql extension API
		Finalize	Protect method	Y	Npgsql extension API
		MakeRoomForBytes	Protect method	Y	Npgsql extension API
		MemberwiseClone	Protect method	Y	Npgsql extension API
		PrefixField	Protect method	Y	Npgsql extension API
None applicable	NpgsqlNoticeEventArgs	Notice	Public field	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API
		GetType	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API
		Finalize	Protect method	Y	Npgsql extension API
		MemberwiseClone	Protect method	Y	Npgsql extension API
None applicable	ServerVersion	ProtocolVersion2	Public field	Y	Npgsql extension API
		ProtocolVersion3	Public field	Y	Npgsql extension API
		Equality Operator	Public	Y	Npgsql extension API
		Greater Than Operator	Public	Y	Npgsql extension API
		Greater Than Or Equal Operator	Public	Y	Npgsql extension API
		Inequality Operator	Public	Y	Npgsql extension API
		Less Than Operator	Public	Y	Npgsql extension API
		Less Than Or Equal Operator	Public	Y	Npgsql extension API
		Implicit ServerVersion to Version Conversion	Public	Y	Npgsql extension API

System.Data.Common class name	Fujitsu Npgsql .NET Data Provider class name	Method (M)/Function (F)/Property (P)		Support status	Note
		Name	Qualifier		
		Implicit Version to ServerVersion Conversion	Public	Y	Npgsql extension API
		Major	Public Property	Y	Npgsql extension API
		Minor	Public Property	Y	Npgsql extension API
		Patch	Public Property	Y	Npgsql extension API
		Clone	Public method	Y	Npgsql extension API
		CompareTo	Public method	Y	Npgsql extension API
		Equals	Public method	Y	Npgsql extension API
		GetHashCode	Public method	Y	Npgsql extension API
		GetType	Public method	Y	Npgsql extension API
		ToString	Public method	Y	Npgsql extension API

Y: Supported

N: Not supported

F.4 C Library (libpq)

See

Refer to "libpq - C Library" in "Client Interfaces" in the PostgreSQL Documentation.

F.5 Embedded SQL in C

See

Refer to "ECPG - Embedded SQL in C" in "Client Interfaces" in the PostgreSQL Documentation.

Index

[B]

BIND_VARIABLE....97

[C]

CLOSE_CURSOR....98
Code examples for applications....56,67
COLUMN_VALUE....98
Comparison operator....3
ConnectionPoolDataSource....237

[D]

DBMS_OUTPUT....84
DBMS_SQL....95
DECODE....79
DEFINE_COLUMN....99
DISABLE....85
DUAL Table....79

[E]

ENABLE....85
Encoding System Settings....31,49
Example of specifying the hint clause....56
EXECUTE....99

[F]

FCLOSE....90
FCLOSE_ALL....90
FCOPY....90
FETCH_ROWS....100
FFLUSH....91
FGETATTR....91
FOPEN....91
FRENAME....92

[G]

GET_LINE....86

[I]

IS_OPEN....93

[J]

java.sql....207
Java Applications using PL/extJava....104
javax.sql....227

[L]

Language settings....10,30,37,48
List of data types belonging to base data types....43

[N]

NEW_LINE....86,93
Notes on automatically generating update-type SQL statements 46
Notes on metadata....46
Notes on Server Explorer....46
Notes on TableAdapter....45
Notes on the Query Builder....45
NVL....83

[O]

OPEN_CURSOR....100
org.postgresql....227
org.postgresql.copy....228
org.postgresql.ds....229
org.postgresql.ds.common....230
org.postgresql.fastpath....232
org.postgresql.geometric....232
org.postgresql.largeobject....233
org.postgresql.ssl....234
org.postgresql.util....235
org.postgresql.xa....236
Outer Join Operator (+)....76

[P]

PARSE....100
Pattern matching....3
Precompiling example....56,68
PUT....85,93
PUTF....94
PUT_LINE....86,94

[S]

Settings for encrypting communication data for connection to the server....10
String functions and operators....3
SUBSTR....81

[U]

Use the API of the .NET System.Data.Common namespace.240
Use the Fujitsu Npgsql.NET Data Provider API (classes and methods) directly....240
UTL_FILE....88

[W]

When setting from outside with environment variables....49
When specifying in the connection URI....49
When using Add-OdbcDsn....27
When using ODBCConf.exe....26