FUJITSU

FUJITSU Enterprise Postgres 12 SP1 on IBM LinuxONE™

Application Development Guide

Linux

J2UL-2618-02ZLZ0(00)
February 2021

Preface

Purpose of this document

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

Intended readers

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

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

- PostgreSQL
- SQL
- Linux

Structure of this document

This document is structured as follows:

Chapter 1 Overview of the Application Development Function

Provides an overview of FUJITSU 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 C Library (libpq)

Explains how to use C applications.

Chapter 5 Embedded SQL in C

Explains how to use embedded SQL in C.

Chapter 6 SQL References

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

Chapter 7 Compatibility with Oracle Databases

Explains features that are compatible with Oracle databases.

Chapter 8 Application Connection Switch Feature

Explains the application connection switch feature.

Chapter 9 Performance Tuning

Explains how to tune application performance.

Chapter 10 Scan Using a Vertical Clustered Index (VCI)

Explains how to perform scan using a Vertical Clustered Index (VCI).

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 FUJITSU 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 Quantitative Limits

This appendix explains limitations.

Appendix E 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

Edition 2.0: February 2021 Edition 1.0:
September 2020

Copyright

Copyright 2019-2021 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 2
1.1.3 Functions and Operator 3
1.2 Compatibility with Oracle Database 3
1.3 Application Connection Switch Feature 3
1.3.1 Integration with Database Multiplexing 4
1.4 Notes on Application Compatibility 4
1.4.1 Checking Execution Results 4
1.4.2 Referencing System Catalogs 4
1.4.3 Using Functions 5

Chapter 2 JDBC Driver 6
2.1 Development Environment 6
2.1.1 Combining with JDK or JRE 6
2.2 Setup 6
2.2.1 Environment Settings 6
2.2.2 Message Language and Encoding System Used by Applications Settings 6
2.2.3 Settings for Encrypting Communication Data 7
2.3 Connecting to the Database 8
2.3.1 Using the DriverManager Class 8
2.3.2 Using the PGConnectionPoolDataSource Class 9
2.3.3 Using the PGXADataSource Class 9
2.4 Application Development 10
2.4.1 Relationship between the Application Data Types and Database Data Types 10
2.4.2 Statement Caching Feature 12
2.4.3 Creating Applications while in Database Multiplexing Mode 12
2.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions 12

Chapter 3 ODBC Driver 14
3.1 Development Environment 14
3.2 Setup 14
3.2.1 Registering ODBC Drivers 14
3.2.2 Registering ODBC Data Sources 16
3.2.3 Message Language and Encoding System Used by Applications Settings 18
3.3 Connecting to the Database 19
3.4 Application Development 19
3.4.1 Compiling Applications 19
3.4.2 Creating Applications While in Database Multiplexing Mode 19
3.4.2.1 Errors when an Application Connection Switch Occurs and Corresponding Actions 20

Chapter 4 C Library (libpq) 21
4.1 Development Environment 21
4.2 Setup 21
4.2.1 Environment Settings 21
4.2.2 Message Language and Encoding System Used by Applications Settings 21
4.2.3 Settings for Encrypting Communication Data 22
4.3 Connecting with the Database 23
4.4 Application Development 23
4.4.1 Compiling Applications 23
4.4.2 Creating Applications while in Database Multiplexing Mode 23
4.4.2.1 Errors when an Application Connection Switch Occurs and Corresponding Actions 24

Chapter 5 Embedded SQL in C 25
5.1 Development Environment 25

5.2 Setup ... 25
5.2.1 Environment Settings ... 25
5.2.2 Message Language and Encoding System Used by Applications Settings ... 25
5.2.3 Settings for Encrypting Communication Data ... 25
5.3 Connecting with the Database ... 25
5.4 Application Development ... 27
5.4.1 Support for National Character Data Types ... 28
5.4.2 Compiling Applications ... 28
5.4.3 Bulk INSERT ... 29
5.4.4 DECLARE STATEMENT ... 33
5.4.5 Creating Applications while in Database Multiplexing Mode ... 34
5.4.5.1 Errors when an Application Connection Switch Occurs and Corresponding Actions ... 34
5.4.6 Notes ... 35

Chapter 6 SQL References ... 36
6.1 Expanded Trigger Definition Feature ... 36
6.1.1 CREATE TRIGGER ... 36

Chapter 7 Compatibility with Oracle Databases ... 38
7.1 Overview ... 38
7.2 Precautions when Using the Features Compatible with Oracle Databases ... 38
7.2.1 Notes on SUBSTR ... 38
7.2.2 Notes when Integrating with the Interface for Application Development ... 39
7.3 Queries ... 39
7.3.1 Outer Join Operator (+) ... 39
7.3.2 DUAL Table ... 41
7.4 SQL Function Reference ... 41
7.4.1 DECODE ... 42
7.4.2 SUBSTR ... 43
7.4.3 NVL ... 45
7.5 Package Reference ... 45
7.5.1 DBMS_OUTPUT ... 46
7.5.1.1 Description ... 46
7.5.1.2 Example ... 49
7.5.2 UTL_FILE ... 49
7.5.2.1 Registering and Deleting Directories ... 50
7.5.2.2 Description ... 51
7.5.2.3 Example ... 55
7.5.3 DBMS_SQL ... 56
7.5.3.1 Description ... 57
7.5.3.2 Example ... 61

Chapter 8 Application Connection Switch Feature ... 64
8.1 Connection Information for the Application Connection Switch Feature ... 64
8.2 Using the Application Connection Switch Feature ... 65
8.2.1 Using the JDBC Driver ... 65
8.2.2 Using the ODBC Driver ... 66
8.2.3 Using a Connection Service File ... 68
8.2.4 Using the C Library (libpq) ... 69
8.2.5 Using Embedded SQL ... 70
8.2.6 Using the psql Command ... 72

Chapter 9 Performance Tuning ... 74
9.1 Enhanced Query Plan Stability ... 74
9.1.1 Optimizer Hints ... 74
9.1.2 Locked Statistics ... 76

Chapter 10 Scan Using a Vertical Clustered Index (VCI) ... 80

10.1 Operating Conditions....80
10.2 Usage....81
10.2.1 Designing....81
10.2.2 Checking....82
10.2.3 Evaluating....83
10.3 Usage Notes....83

Appendix A Precautions when Developing Applications....86
A.1 Precautions when Using Functions and Operators....86
A.1.1 General rules of Functions and Operators....86
A.1.2 Errors when Developing Applications that Use Functions and/or Operators....86
A.2 Notes when Using Temporary Tables....87
A.3 Implicit Data Type Conversions....87
A.3.1 Function Argument....89
A.3.2 Operators....89
A.3.3 Storing Values....90
A.4 Notes on Using Index....90
A.4.1 SP-GiST Index....90
A.5 Notes on Using Multibyte Characters in Definition Names....90

Appendix B Conversion Procedures Required due to Differences from Oracle Database....92
B.1 Outer Join Operator (Perform Outer Join)....92
B.1.1 Comparing with the ^= Comparison Operator....92
B.2 DECODE (Compare Values and Return Corresponding Results)....93
B.2.1 Comparing Numeric Data of Character String Types and Numeric Characters....93
B.2.2 Obtaining Comparison Result from more than 50 Conditional Expressions....93
B.2.3 Obtaining Comparison Result from Values with Different Data Types....94
B.3 SUBSTR (Extract a String of the Specified Length from Another String)....95
B.3.1 Specifying a Value Expression with a Data Type Different from the One that can be Specified for Function Arguments....95
B.3.2 Extracting a String with the Specified Format from a Datetime Type Value....96
B.3.3 Concatenating a String Value with a NULL value....96
B.4 NVL (Replace NULL)....97
B.4.1 Obtaining Result from Arguments with Different Data Types....97
B.4.2 Operating on Datetime/Numeric, Including Adding Number of Days to a Particular Day....98
B.4.3 Calculating INTERVAL Values, Including Adding Periods to a Date....98
B.5 DBMS_OUTPUT (Output Messages)....99
B.5.1 Outputting Messages Such As Process Progress Status....99
B.5.2 Receiving a Return Value from a Procedure (PL/SQL) Block (For GET_LINES)....101
B.5.3 Receiving a Return Value from a Procedure (PL/SQL) Block (For GET_LINE)....103
B.6 UTL_FILE (Perform File Operation)....104
B.6.1 Registering a Directory to Load and Write Text Files....104
B.6.2 Checking File Information....105
B.6.3 Copying Files....109
B.6.4 Moving/Renaming Files....110
B.7 DBMS_SQL (Execute Dynamic SQL)....111
B.7.1 Searching Using a Cursor....111

Appendix C Tables Used by the Features Compatible with Oracle Databases....116
C.1 UTL_FILE.UTL_FILE_DIR....116

Appendix D Quantitative Limits....117

Appendix E Reference....122
E.1 JDBC Driver....122
E.2 ODBC Driver....122
E.2.1 List of Supported APIs....122
E.3 C Library (libpq)....125
E.4 Embedded SQL in C....125

Index.....126

Chapter 1 Overview of the Application Development Function

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

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

- Support for National Characters

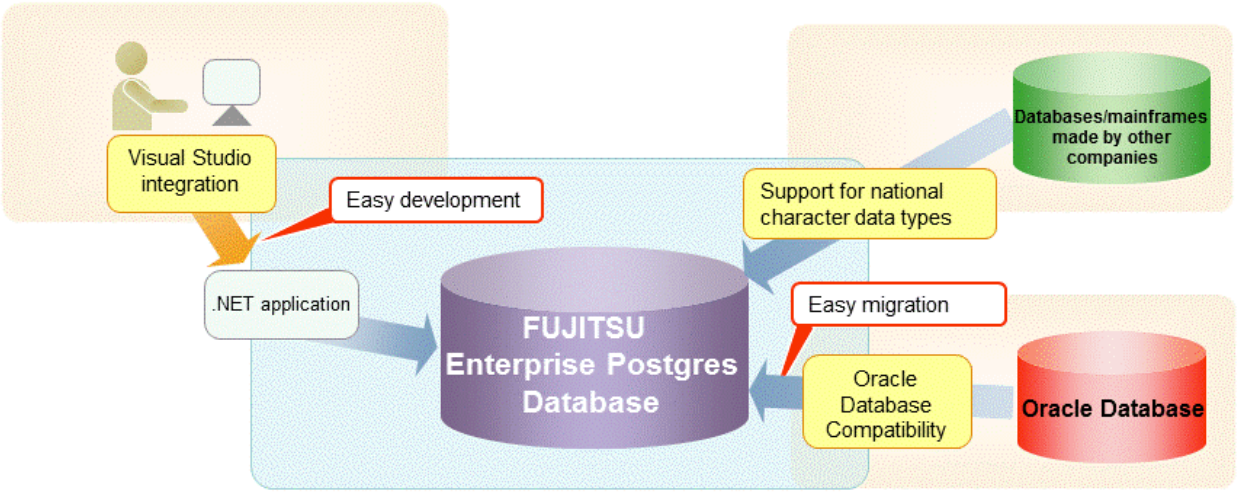
  In order to secure portability from mainframes and databases of other companies, FUJITSU Enterprise Postgres provides data types that support national characters. The national characters are usable from the client application languages.

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

- 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 "1.2 Compatibility with Oracle Database" 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 "1.3 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 "9.1 Enhanced Query Plan Stability" for details.

- Scanning using a Vertical Clustered Index (VCI)

  Scans becomes faster during aggregation of many rows by providing the features below:

  - Vertical clustered index (VCI)
  - In-memory data

  Refer to "Chapter 10 Scan Using a Vertical Clustered Index (VCI)" for details.

1.1 Support for National Characters

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

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 CHARACTER | 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 character string with a fixed length of *n* characters
NATIONAL CHAR(*n*)	This will be the same as (1) if (*n*) is omitted.

Data type specification format	Explanation
NCHAR(*n*)	*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 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 7 Compatibility with Oracle Databases" for information on the features compatible with Oracle databases.

1.3 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 8 Application Connection Switch Feature " for information on the application connection switch feature.

1.3.1 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 the Cluster Operation Guide (Database Multiplexing) for information on database multiplexing.

1.4 Notes on Application Compatibility

FUJITSU 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

1.4.1 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.

1.4.2 Referencing System Catalogs

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

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

1.4.3 Using Functions

The default functions provided with FUJITSU 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 FUJITSU Enterprise Postgres functions, such as those relating to statistical information or for obtaining system-related information, may change as FUJITSU 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 or JRE 7

```
postgresql-jdbc41.jar
```

- If using JDK 8, JRE 8, JDK 11 or JRE 11

```
postgresql-jdbc42.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, JRE 8, JDK 11 or JRE 11 is used, only the name of the JDBC driver file will be different. The method for configuring the CLASSPATH environment variable is the same.

Note that "<*x*>" indicates the product version.

- Setting example (TC shell)

```
setenv CLASSPATH /opt/fsepv<x>client64/jdbc/lib/postgresql-jdbc4.jar:${CLASSPATH}
```

- Setting example (bash)

```
CLASSPATH=/opt/fsepv<x>client64/jdbc/lib/postgresql-jdbc4.jar:$CLASSPATH;export 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".

If ssl is set to "true", sslmode is internally treated as "verify-full".

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");
props.setProperty("sslfactory","org.postgresql.ssl.DefaultJavaSSLFactory");
Connection conn = DriverManager.getConnection(url, props);
```

- Setting example 2

```
String url = "jdbc:postgresql://sv1/test?
user=fsepuser&password=secret&ssl=true&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory";
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. In addition, specify "org.postgresql.ssl.DefaultJavaSSLFactory" for the sslfactory parameter.

Refer to JDK documentation 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 2147483647. 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.

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";
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. 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 2147483647. 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.

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);
...
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.

Argument	Description
	The default is "27500".
setDatabaseName	Specify the database name.
setUser	Specify the username 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.
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 2147483647. 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.

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);...
Connection con = source.getConnection();
```

2.4 Application Development

This section describes the data types required when developing applications that will be connected with FUJITSU 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
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

Data type on the server	Java data type	Data types prescribed by java.sql.Types
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, macaddr8)	org.postgresql.util.PGobject	java.sql.Types.OTHER
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

Although the getString() method of the ResultSet object can be used for all server data types, it is not guaranteed that it will always return a string in the same format for the same data type.

Strings in a format compatible with the JDBC specifications can be obtained using the Java toString() method of the appropriate data type (for example, getInt(), getTimestamp()) to conform to the data type on the server.

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.

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.

2.4.3 Creating Applications while in Database Multiplexing Mode

This section explains points to consider when creating applications while in database multiplexing mode.

See

- Refer to the Cluster Operation Guide (Database Multiplexing) for information on database multiplexing mode.

2.4.3.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs while in database multiplexing mode, 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 (*1)	Action
Server failure or FUJITSU Enterprise Postgres system failure	Failure occurs during access	57P01 08006 08007	After the switch is complete, reestablish the connection, or reexecute the application.
	Accessed during system failure	08001	
Switch to the standby server	Switched during access	57P01 08006 08007	
	Accessed during switch	08001	

*1: Return value of the getSQLState() method of SQLException.

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.

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

FUJITSU 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 FUJITSU Enterprise Postgres. PsqlODBC is included in the FUJITSU 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 Linux platforms, register the ODBC driver using the following procedure:

1. Install the ODBC driver manager (unixODBC)

   Information

   - FUJITSU 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. Register the 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 "FUJITSUEnterprisePostgres<*fujitsuEnterprisePostgresClientVers*>s390x" - Encoding system used by the application - In Unicode (only UTF-8 can be used) "unicode" - Other than Unicode "ansi" Example: The encoding system used by the application is Unicode: "[FUJITSUEnterprisePostgres<*fujitsuEnterprisePostgresClientVers*>s390xunicode]"
Description	Description of the ODBC driver	Specify a supplementary description for the current data source. Any description may be set.

Driver64	Path of the ODBC driver (64-bit)	Set the path of the ODBC driver (64-bit).
		- If the encoding system is Unicode:
		`fujitsuEnterprisePostgresClientInstallDir/odbc/lib/psqlodbcw.so`
		- If the encoding system is other than Unicode:
		`fujitsuEnterprisePostgresClientInstallDir/odbc/lib/psqlodbca.so`

FileUsage	Use of the data source file	Specify 1.
Threading	Level of atomicity secured for connection pooling	Specify 2.

Example

Note that "<x>" indicates the product version.

```
[FUJITSU Enterprise Postgres12s390xunicode]
Description = FUJITSU Enterprise Postgres 12 s390x unicode driver
Driver64    = /opt/fsepv<x>client64/odbc/lib/psqlodbcw.so
FileUsage   = 1
Threading   = 2
```

3.2.2 Registering ODBC Data Sources

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

1. Register the 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 "FUJITSU Enterprise Postgres<fujitsuEnterprisePostgresClientVers>s390x" - Encoding system used by the application - In Unicode (only UTF-8 can be used) "unicode" - Other than Unicode

Definition name	Setting value
	"ansi" Example: The encoding system used by the application is Unicode: "FUJITSU Enterprise Postgres<*fujitsuEnterprisePostgresClientVers*>s390xunicode"
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	Specify the communication encryption method. 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)
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

```
[MyDataSource]
Description     = FUJITSU Enterprise Postgres
Driver          = FUJITSU Enterprise Postgres12s390xansi
Database        = db01
Servername      = sv1
Port            = 27500
ReadOnly        = 0
```

Note

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

2. Configure the 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:

- *fujitsuEnterprisePostgresClientInstallDir*\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.

3.2.3 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.

Messages output by an application may include text from messages sent from the database server. In the resulting text, the text of the application message will use the message locale of the application, and the text of the message sent by the database server will use the message locale of the database server. If the message locales do not match, more than one language or encoding system will be used. Moreover, if the encoding systems do not match, characters in the resulting text can be garbled.

Set the locale for messages (LC_MESSAGES category) to match the message locale of the database server. This can be done in a few different ways, such as using environment variables. Refer to the relevant manual of the operating system for information on the setlocale function.

Example

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

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

Specifying the locale of the LC_ALL category propagates the setting to LC_MESSAGE.

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

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

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

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.

3.4 Application Development

This section describes how to develop applications using ODBC drivers.

3.4.1 Compiling Applications

Specify the following options when compiling applications.

Table 3.1 Include file and library path

Option	How to specify the option
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.

Example

The following are examples of compiling ODBC applications:

```
gcc -m64 -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.

3.4.2 Creating Applications While in Database Multiplexing Mode

This section explains points to consider when creating applications while in database multiplexing mode.

See

- Refer to the Cluster Operation Guide (Database Multiplexing) for information on database multiplexing mode.

3.4.2.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs while in database multiplexing mode, 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 (*1)	Action
Server failure or FUJITSU Enterprise Postgres system failure	Failure occurs during access	57P01 08S01	After the switch is complete, reestablish the connection, or reexecute the application.
	Accessed during system failure	08001	
Switch to the standby server	Switched during access	57P01 08S01	
	Accessed during switch	08001	

*1: Return value of SQLSTATE.

Chapter 4 C Library (libpq)

This chapter describes how to use C libraries.

4.1 Development Environment

Install FUJITSU 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.

4.2 Setup

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

4.2.1 Environment Settings

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

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

Example

Note that "<*x*>" indicates the product version.

```
> LD_LIBRARY_PATH=/opt/fsepv<x>client64/lib:$LD_LIBRARY_PATH;export LD_LIBRARY_PATH
> PGLOCALEDIR=/opt/fsepv<x>client64/share/locale;export PGLOCALEDIR
```

4.2.2 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.

Messages output by an application may include text from messages sent from the database server. In the resulting text, the text of the application message will use the message locale of the application, and the text of the message sent by the database server will use the message locale of the database server. If the message locales do not match, more than one language or encoding system will be used. Moreover, if the encoding systems do not match, characters in the resulting text can be garbled.

Set the locale for messages (LC_MESSAGES category) to match the message locale of the database server. This can be done in a few different ways, such as using environment variables. Refer to the relevant manual of the operating system for information on the setlocale function.

Example

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

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

Specifying the locale of the LC_ALL category propagates the setting to LC_MESSAGE.

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.

4.2.3 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.

4.3 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 "5.3 Connecting with the Database" in "Embedded SQL in C " for information on connection string.

4.4 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).

4.4.1 Compiling Applications

Specify the following paths when compiling applications.

Refer to your compiler documentation for information on how to specify the path.

Table 4.1 Include file and library path

Type of path	Path name
Path of the include file	*fujitsuEnterprisePostgresClientInstallDir*/include
Path of the library	*fujitsuEnterprisePostgresClientInstallDir*/lib

Table 4.2 C Library (libpq library)

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

4.4.2 Creating Applications while in Database Multiplexing Mode

This section explains points to consider when creating applications while in database multiplexing mode.

See

- Refer to the Cluster Operation Guide (Database Multiplexing) for information on database multiplexing mode.

4.4.2.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs while in database multiplexing mode, 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
Server failure or FUJITSU Enterprise Postgres system failure	Failure occurs during access	PGRES_FATAL_ERROR(*1) 57P01(*2) NULL(*2)	After the switch is complete, reestablish the connection, or reexecute the application.
	Accessed during system failure	CONNECTION_BAD(*3)	
Switch to the standby server	Switched during access	PGRES_FATAL_ERROR(*1) 57P01(*2) NULL(*2)	
	Accessed during switch	CONNECTION_BAD(*3)	

*1: Return value of PQresultStatus().

*2: Return value of PQresultErrorField() PG_DIAG_SQLSTATE.

*3: Return value of PQstatus().

Chapter 5 Embedded SQL in C

This chapter describes application development using embedded SQL in C.

5.1 Development Environment

Install FUJITSU 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++.

5.2 Setup

5.2.1 Environment Settings

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

Refer to "4.2.1 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 in the PATH environment variable:

```
fujitsuEnterprisePostgresClientInstallDir/bin
```

5.2.2 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 "4.2.2 Message Language and Encoding System Used by Applications Settings" in "C Library (libpq)" for information on the settings for the library for C.

5.2.3 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 "4.2.3 Settings for Encrypting Communication Data" in "C Library (libpq)" for information on the environment settings for the C library.

5.3 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]
- 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

Argument	Description
	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. Specify a value between 1 and 32767 (in seconds). The default value of the system is used if this is not specified. - keepalives_interval Specify the interval between resends when there is no response to keepalive messages. Specify a value between 1 and 32767 (in seconds). The default value of the system is used if this is not specified. - keepalives_count Specify the number of resends for keepalive messages. Specify a value between 1 and 127. The default value of the system is used if this is not specified. - tcp_user_timeout After establishing the connection, when sending from the client to the server, if the TCP resend process operates, specify the time until it is considered to be disconnected. Specify a value between 0 and 2147483647 (in millseconds). The default value of the system is used if 0. 0 will be set as default if nothing is specified.

If a value other than 0 is specified for the tcp_user_timeout parameter, the waiting time set by the tcp_keepalives_idle parameter and tcp_keepalives_interval parameter will be invalid and the waiting time specified by the tcp_user_timeout parameter will be used.

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;
```

5.4 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.

5.4.1 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.

5.4.2 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 "9.1.1 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 paths when compiling a C application output with precompiling.

Refer to your compiler documentation for information on how to specify the path.

Table 5.1 Include file and library path

Type of path	Path name
Path of the include file	*fujitsuEnterprisePostgresClientInstallDir*/include
Path of the library	*fujitsuEnterprisePostgresClientInstallDir*/lib

Table 5.2 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

5.4.3 Bulk INSERT

This section describes the bulk INSERT.

Synopsis

```
EXEC SQL [ AT conn ] [ FOR { numOfRows | ARRAY_SIZE } ]
    INSERT INTO tableName [ ( colName [, ...] ) ]
    { VALUES ( { expr | DEFAULT } [, ...] ) [, ...] | query }
    [ RETURNING * | outputExpr [ [ AS ] outputName ] [, ...]
    INTO outputHostVar [ [ INDICATOR ] indicatorVar  ]  [, ...] ];
```

Description

Bulk INSERT is a feature that inserts multiple rows of data in bulk.

By specifying the array host variable that stored the data in the VALUES clause of the INSERT statement, the data for each element in the array can be inserted in bulk. This feature is used by specifying the insertion count in the FOR clause immediately before the INSERT statement.

FOR Clause

Specify the insertion count using numOfRows or ARRAY_SIZE in the FOR clause. The FOR clause can be specified only in the INSERT statement, not in other update statements.

numOfRows and ARRAY_SIZE

Insertion processing will be executed only for the specified count. However, if the count is 1, it will be assumed that the FOR clause was omitted when the application is executed. In this case, proceed according to the INSERT specification in the PostgreSQL Documentation.

Specify the FOR clause as an integer host variable or as a literal.

Specify ARRAY_SIZE to insert all elements of the array in the table. When specifying ARRAY_SIZE, specify at least one array in *expr*.

If two or more arrays were specified in expr, it will be assumed that ARRAY_SIZE is the minimum number of elements in the array.

numOfRows or ARRAY_SIZE must exceed the minimum number of elements in all arrays specified in *expr*, *outputHostVar*, and *indicatorVal*.

The following example shows how to specify the FOR clause.

```
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);
```

expr

Specify the value to be inserted in the table. Array host variables, host variable literals, strings, and pointer variables can be specified. Structure type arrays and pointer variable arrays cannot be specified.

Do not use pointer variables and ARRAY_SIZE at the same time. The reason for this is that the number of elements in the area represented by the pointer variable cannot be determined.

query

A query (SELECT statement) that supplies the rows to be inserted. The number of rows returned by *query* must be 1. If two or more rows are returned, an error will occur. This cannot be used at the same time as ARRAY_SIZE.

outputHostVar, indicatorVal

These must be array host variables or pointer variables.

Error Messages

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

Message

invalid statement name "FOR value should be positive integer"

Cause

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

Solution

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

Message

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

Message

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 *numOfRows* 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, *numOfRows* should be less than or equal to the number of elements in the pointer. Otherwise, junk data will be inserted into the table.
- If an error occurs, all bulk INSERT actions will be rolled back, therefore, no rows are inserted. However, if the RETURNING clause was used, and the error occurred while obtaining the rows after the insertion was successful, the insertion processing will not be rolled back.

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.

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 example,

```
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'.

5.4.4 DECLARE STATEMENT

This section describes the DECLARE STATEMENT.

Synopsis

```
EXEC SQL [ AT connName] DECLARE statementName STATEMENT;
```

Description

DECLARE STATEMENT is an embedded SQL command that declares an identifier for a prepared statement. The declared identifier can be used as an identifier in a prepared statement for the following SQL commands:

- EXECUTE
- DECLARE
- DESCRIBE
- PREPARE (Embedded SQL commands)
- PREPARE (SQL commands)

You can associate an identifier in a prepared statement with a connection by executing a DECLARE STATEMENT that specifies the connection. If you specify an identifier associated with the connection in a later SQL command, the SQL command is executed using the connection associated with the identifier. The association between the connection and the identifier in the prepared statement is shared throughout the process.

Only one connection can be associated with a prepared statement of the same name. If you make multiple associations across files, subsequent DECLARE STATEMENT are ignored. If you associate the same file more than once, precompiling the file fails.

If you use the identifier associated with a connection in a SQL command, do not use the AT clause. If a connection different from the connection linked to the identifier using the AT clause is selected, a runtime error will occur.

Parameters

connName

A database connection name established by the CONNECT command.

If AT clause is omitted, no association is made between the connection and the identifier. DECLARE STATEMENT is executed, but has no effect.

statementName

Specify the identifier of the prepared statement. You can specify either a SQL identifier or a host variable.

Examples

Dynamic SQL statement

DECLARE STATEMENT is primarily used to execute dynamic SQL statements.

```
EXEC SQL BEGIN DECLARE SECTION;
char dbname[128];
EXEC SQL END DECLARE SECTION;
...
```

```
EXEC SQL CONNECT TO postgres AS con1;
EXEC SQL CONNECT TO another_database AS con2;
EXEC SQL AT con1 DECLARE sql_stmt STATEMENT;
EXEC SQL DECLARE cursor_name CURSOR FOR sql_stmt;
EXEC SQL PREPARE sql_stmt FROM "SELECT current_database()";
EXEC SQL OPEN cursor_name;
EXEC SQL FETCH cursor_name INTO :dbname;
EXEC SQL CLOSE cursor_name;
```

In the example above, the connection 'con1' is associated with the prepared statement identifier 'sql_stmt'. So the current connection is 'con2', but the embedded SQL commands that follow are executed on 'con1'.

PREPARE AS statement

The following is an example using a PREPARE statement in the SQL command:

```
EXEC SQL AT db1 DECLARE stmt STATEMENT;
EXEC SQL PREPARE stmt (int) AS
    SELECT * FROM employee WHERE id = $1;
EXEC SQL EXECUTE stmt USING 1;
```

The above SELECT statement is executed on the connection 'db1'.

Compatibility

DECLARE STATEMENT is not specified in the SQL standard.

See Also

EXECUTE, DECLARE, DESCLIBE, PREPARE (Embedded SQL commands), PREPARE (SQL commands)

See

- Refer to "Embedded SQL Commands" in "Client Interfaces" in the PostgreSQL documentation for information on the embedde SQL commands.
- Refer to "SQL Commands" in "Reference" in the PostgreSQL documentation for information on the SQL commands.

5.4.5 Creating Applications while in Database Multiplexing Mode

This section explains points to consider when creating applications while in database multiplexing mode.

See

- Refer to the Cluster Operation Guide (Database Multiplexing) for information on database multiplexing mode.

5.4.5.1 Errors when an Application Connection Switch Occurs and Corresponding Actions

If an application connection switch occurs while in database multiplexing mode, 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 (*1)	Action
Server failure or	Failure occurs during access	57P01 57P02	After the switch is complete, reestablish the

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

*1: Return value of SQLSTATE.

5.4.6 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 6 SQL References

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

6.1 Expanded Trigger Definition Feature

This section explains the expanded trigger definition feature.

6.1.1 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 ] ]
    [ REFERENCING { { OLD | NEW } TABLE [ AS ] transition_relation_name } [ ... ] ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( condition ) ]
    { EXECUTE { FUNCTION | 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.

- 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).

Chapter 7 Compatibility with Oracle Databases

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

7.1 Overview

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

The table below lists features compatible with Oracle databases.

Table 7.1 Features compatible with Oracle databases

Category		Feature	
		Item	Overview
SQL	Queries	Outer join operator (+)	Operator for outer joins
		DUAL table	Table provided by the system
	Functions	DECODE	Compares values, and if they match, returns a corresponding value
		SUBSTR	Extracts part of a string using characters to specify position and length
		NVL	Returns a substitute value when a value is NULL
Package		DBMS_OUTPUT	Sends messages to clients
		UTL_FILE	Enables text file operations
		DBMS_SQL	Enables dynamic SQL execution

See

In addition to the above, refer to the file below for information on the Orafce function.

fujitsuEnterprisePostgresInstallDir/share/doc/extension/README.asciidoc

7.2 Precautions when Using the Features Compatible with Oracle Databases

This section provides notes on using the features compatible with oracle databases.

7.2.1 Notes on SUBSTR

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

For this reason, when using SUBSTR, define which specification is to take precedence. In the default configuration of FUJITSU Enterprise Postgres, the specifications of FUJITSU Enterprise Postgres take precedence.

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" before "pg_catalog" when doing this.

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

Information

- The search_path parameter specifies the order in which schemas are searched. The SUBSTR function in Oracle databases is defined in the oracle schema.

- Refer to "Statement Behavior" in "Client Connection Defaults" in "Server Administration" in the PostgreSQL Documentation for information on search_path.

7.2.2 Notes when Integrating with the Interface for Application Development

The SQL noted in "Table 7.1 Features compatible with Oracle databases" can be used in the interface for application development.

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.

7.3 Queries

The following queries are supported:

- Outer Join Operator (+)
- DUAL Table

7.3.1 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).

Syntax

SELECT statement

```
SELECT … [WHERE [NOT] joinCond ...] …
SELECT … [WHERE srchCond ]... ] …
```

Join condition

```
{ colSpec(+) = colSpec | colSpec = colSpec(+) }
```

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 syntax 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 rows that do not meet these conditions are excluded from the output. When the values of the actual rows are assigned to variables and if the expression returns TRUE, those rows 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 rows that meet the join conditions, including rows that do not meet all the join conditions.
- Join conditions take precedence over search conditions. For this reason, all rows returned by the join conditions are subject to the search conditions.
- The following rules and restrictions apply to queries that use outer join operators. It is therefore recommended to use 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 7.2 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: Return all rows in table t2, including those that do not exist in table t1.

```
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;
```

7.3.2 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

In the following example, the current system date is returned.

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

7.4 SQL Function Reference

The following SQL functions are supported:

- DECODE
- SUBSTR
- NVL

7.4.1 DECODE

Description

Compares values and if they match, returns a corresponding value.

Syntax

```
DECODE(expr, srch, result [, srch, result ]... [, default ])
```

General rules

- DECODE compares values of the value expression to be converted and the search values one by one. If the values match, a corresponding result value is returned. If no values match, the default value is returned if it 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 the values 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 expression to be converted contains data types that can be converted. When specifying 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 the data types that can be specified.
- 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 string literals, all will become character 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. When specifying 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 the 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 7.3 Data type combinations that can be converted by DECODE (summary)

		Other result values or default value		
		Numeric type	Character type	Date/time type
Result value (any)	Numeric type	Y	N	N
	Character 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 7.4 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 does not match any of search values 1, 2, or 3, the default value "other number" is returned.

```
SELECT col1, DECODE(col3, 1000, 'one',
                          2000, 'two',
                          3000, 'three',
                          'other number') "num-word"
       FROM t1;
col1  | num-word
------+----------
 1001 | one
 1002 | two
 1003 | three
(3 rows)
```

7.4.2 SUBSTR

Description

Extracts part of a string using characters to specify position and length.

Syntax

```
SUBSTR(str, startPos [, len ])
```

General rules

- SUBSTR extracts and returns a substring of string *str*, beginning at position *startPos*, for number of characters *len*.
- When *startPos* is positive, it will be the number of characters from the beginning of the string.
- When *startPos* is 0, it will be treated as 1.
- When *startPos* is negative, it will be the number of characters from the end of the string.
- When *len* is not specified, all characters to the end of the string are returned. NULL is returned when *len* is less than 1.
- For *startPos* and *len*, specify a SMALLINT or INTEGER type. When specifying 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 the data types that can be 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 parameter must 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 "7.2.1 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. In this example, user1 is used as the username.

    ```
    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. In this example, db1 will be used as the database name.

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

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

See

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

Information

The general rules for SUBSTRING are as follows:

- The start position will be from the beginning of the string, whether positive, 0, or negative.
- When *len* is not specified, all characters to the end of the string are returned.
- An empty string is returned if no string is extracted or *len* 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
-----------
(1 row)
```

7.4.3 NVL

Description

Returns a substitute value when a value is NULL.

Syntax

```
NVL(expr1, expr2)
```

General rules

- NVL returns a substitute value when the specified value is NULL. When *expr1* is NULL, *expr2* is returned. When *expr1* is not NULL, *expr1* is returned.
- Specify the same data types for *expr1* and *expr2*. However, if a constant is specified in *expr2*, and the data type can also be converted by *expr1*, different data types can be specified. When this happens, the conversion by *expr2* is done to suit the data type in *expr1*, so the value of *expr2* returned when *expr1* is a NULL value will be the value converted in the data type of *expr1*.
- When specifying 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 the data types that can be converted.

Example

In the following example, "IS NULL" is returned 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)
```

7.5 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.

7.5.1 DBMS_OUTPUT

Overview

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

Features

Features	Description
ENABLE	Enables features of this package.
DISABLE	Disables features of this package.
SERVEROUTPUT	Controls whether messages are sent.
PUT	Sends messages.
PUT_LINE	Sends messages with a newline character appended.
NEW_LINE	Sends a newline character.
GET_LINE	Retrieves a line from the message buffer.
GET_LINES	Retrieves multiple lines from the message buffer.

Syntax

```
{ ENABLE( [buffSize ])
| DISABLE()
| SERVEROUTPUT(sendMsgs)
| PUT(str)
| PUT_LINE(str)
| NEW_LINE()
| GET_LINE()
| GET_LINES(maxLineNum)
}
```

7.5.1.1 Description

This section explains each feature of DBMS_OUTPUT.

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 a buffer size from 2000 to 1000000.
- 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.
- Specify TRUE or FALSE for *sendMsgs*.
- If TRUE is specified, 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 FALSE is specified, 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 "Data Types" in "The SQL Language" in the PostgreSQL Documentation for information on boolean type (TRUE/FALSE) values.

Example

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

PUT

- PUT sets the message to be sent.
- The string is the message to be sent.
- When TRUE is specified for SERVEROUTPUT, the messages are sent to clients such as psql.
- When FALSE is specified for SERVEROUTPUT, the messages are retained in the buffer.
- PUT does not append a newline character. To append a newline character, 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 sets the message to be sent appended with a newline character.
- The string is the message that is sent.

- When TRUE is specified for SERVEROUTPUT, the messages are sent to clients such as psql.
- When FALSE is specified for SERVEROUTPUT, the messages are retained in the buffer.
- PUT_LINE appends a newline character 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 appends a newline character to the message created with PUT.
- When TRUE is specified for SERVEROUTPUT, the messages are sent to clients such as psql.
- When FALSE is specified for SERVEROUTPUT, the messages are retained in the buffer.

Example

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

GET_LINE

- GET_LINE retrieves a line from the message buffer.
- Use a SELECT statement to obtain the retrieved line and status code returned by the operation, which are stored in the line and status columns.
- The line column stores the line retrieved from the buffer. The data type of line is TEXT.
- The status column stores the status code returned by the operation: 0-completed successfully; 1-failed because there are no more lines in the buffer. The data type of status is INTEGER.
- If GET_LINE or GET_LINES is executed and then PUT, PUT_LINE, or NEW_LINE is while messages that have not been retrieved from the buffer still exist, the messages not retrieved 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 retrieves multiple lines from the message buffer.
- Use a SELECT statement to obtain the retrieved lines and the number of lines retrieved, which are stored in the lines and numlines columns.
- The lines column stores the lines retrieved from the buffer. The data type of lines is TEXT.
- The numlines column stores the number of lines retrieved from the buffer. The data type of numlines is INTEGER.

- *maxLineNum* is the maximum number of lines to retrieve from the buffer. The data type is INTEGER.
- If GET_LINE or GET_LINES is executed and then PUT, PUT_LINE, or NEW_LINE is executed while messages that have not been retrieved from the buffer still exist, the messages not retrieved 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);
```

7.5.1.2 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();
```

7.5.2 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 "7.5.2.1 Registering and Deleting Directories" for information on the how to register and delete the directory.

Note

- When performing file operations, access privileges for the UTL_FILEUTL_FILE_DIR table are required according to the operation. However, because the user performing file operations can access any file regardless of the operating system access privileges for the target file, set the access privileges for the table appropriately so that important files are not accessed by any user.

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 a file.
FCLOSE_ALL	Closes all files open in a session.
FCOPY	Copies a whole file or a contiguous portion thereof.
FFLUSH	Flushes the buffer.
FGETATTR	Retrieves the attributes of a file.
FOPEN	Opens a file.
FRENAME	Renames a file.
GET_LINE	Reads one line from a text file.
IS_OPEN	Checks if a file is open.
NEW_LINE	Writes newline characters.
PUT	Writes a string.
PUT_LINE	Appends a newline character to a string and writes the string.
PUTF	Writes a formatted string.

Syntax

```
{ FCLOSE(fileHandle)
| FCLOSE_ALL()
| FCOPY(srcDir, srcFileName, destDir, destFileName
    [ {,startLine | ,startLine ,endLine } ])
| FFLUSH(fileHandle)
| FGETATTR(dir, filename)
| FOPEN(dir, fileName, openMode [, maxLineSize ])
| FRENAME(srcDir, srcFileName, destDir, destFileName [ ,overwrite ])
| GET_LINES(fileHandle [,len ])
| IS_OPEN(fileHandle)
| NEW_LINE(fileHandle [, numOfNewLines ])
| PUT(fileHandle, str)
| PUT_LINE(fileHandle, str [, writeToFile ])
| PUTF(fileHandle, fmt [, args ]... ])
}
```

7.5.2.1 Registering and Deleting Directories

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';
```

7.5.2.2 Description

This section explains each feature of UTL_FILE.

FCLOSE

- FCLOSE closes a file that is open.
- Specify an open file handle.
- 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 a whole file or a contiguous portion thereof. The whole file is copied if *startLine* and *endLine* are not specified.
- Specify the directory location of the source file.
- Specify the source file.
- Specify the directory where the destination file will be created.
- Specify the file name of the destination file.
- Specify the line number at which to begin copying. Specify a value greater than 0. If not specified, 1 is used.
- Specify the line number at which to stop copying. If not specified, the last line number of the file is used.

Example

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

FFLUSH

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

Example

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

FGETATTR

- FGETATTR retrieves file attributes: file existence, file size, and information about the block size of the file.
- Specify the directory where the file exists.
- Specify the file name.
- Use a SELECT statement to obtain the file attributes, which are stored in the fexists, file_length, and blocksize columns.
- The fexists column stores a boolean (TRUE/FALSE) value. If the file exists, fexists is set to TRUE. If the file does not exist, fexists is set to FALSE. The data type of fexists is BOOLEAN.
- The file_length column stores the length of the file in bytes. If the file does not exist, file_length is NULL. The data type of file_length is INTEGER.
- The blocksize column stores the block size of the file in bytes. If the file does not exist, blocksize is NULL. The data type of blocksize is INTEGER.

Example

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

FOPEN

- FOPEN opens a file.
- Specify the directory where the file exists.
- Specify the file name.
- Specify the mode for opening the file:
  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 a value from 1 to 32767.
- Up to 50 files per session can be open at the same time.

Example

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

FRENAME

- FRENAME renames a file.
- Specify the directory location of the source file.
- Specify the source file to be renamed.

- Specify the directory where the renamed file will be created.
- Specify the new name of the file.
- Specify whether to overwrite a file if one exists with the same name and in the same location as the renamed file. If TRUE is specified, the existing file will be overwritten. If FALSE is specified, an error occurs. If omitted, FALSE is set.

Refer to "Boolean Type" in "Data Types" in "The SQL Language" in the PostgreSQL Documentation for information on boolean type (TRUE/FALSE) values.

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

GET_LINE

- GET_LINE reads one line from a file.
- Specify the file handle returned by FOPEN using the r (read) mode.
- Specify the number of bytes to read from the file. If not specified, the maximum string length specified at FOPEN will be used.
- The return value is the buffer that receives the line read 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 length (in bytes) of the data to be read. Specify a value from 1 to 32767. If not specified, the maximum string length specified at FOPEN is set. If no maximum string length is specified at FOPEN. 1024 is set.
- If the line length is greater than the specified number of bytes to read, the remainder of the line is read on the next call.
- A NO_DATA_FOUND exception will occur when trying to read past the last line.

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

IS_OPEN

- IS_OPEN checks if a file is open.
- Specify the file handle.
- The return value is a BOOLEAN type. TRUE represents an open state and FALSE represents a closed state.

Refer to "Boolean Type" in "Data Types" in "The SQL Language" in the PostgreSQL Documentation for information on boolean type (TRUE/FALSE) values.

Example

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

NEW_LINE

- NEW_LINE writes one or more newline characters.
- Specify an open file handle.
- Specify the number of newline characters to be written to the file. If omitted, "1" is used.

Example

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

PUT

- PUT writes a string to a file.
- Specify the file handle that was opened with FOPEN using w (write) or a (append).
- Specify the string to be written to the file.
- The maximum length (in bytes) of the string to be written 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 appends a newline character to a string and writes the string.
- Specify the file handle that was opened with FOPEN w (write) or a (append).
- Specify whether to forcibly write to the file. If TRUE is specified, file writing is forced. If FALSE is specified, file writing is asynchronous. If omitted, FALSE will be set.
- The maximum length of the string (in bytes) is the maximum string length specified at FOPEN.

Example

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

PUTF

- PUTF writes a formatted string.
- Specify the file handle that was opened with FOPEN w (write) or a (append).

- Specify the format, which is a string that includes the formatting characters \n and %s.
- The \n in the format is code for a newline character.
- 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.

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

7.5.2.3 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 "7.5.2.1 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;

SELECT gen_file('/home/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 "7.5.2.1 Registering and Deleting Directories" for information on how to delete the directory.

7.5.3 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	Retrieves the value of the column in the select list extracted with FETCH_ROWS.
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 row and extracts values from the row.
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 consider this. The supported data types 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" in "Functions and Operators" in "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 a wide range such as when no interval qualifier is specified, 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
```

Syntax

```
{ BIND_VARIABLE(cursor, varName, val [, len ])
| CLOSE_CURSOR(cursor)
| COLUMN_VALUE(cursor, colPos, varName)
| DEFINE_COLUMN(cursor, colPos, varName [, len ])
| EXECUTE(cursor)
| FETCH_ROWS(cursor)
| OPEN_CURSOR([parm1 ])
| PARSE(cursor, sqlStmt, parm1 [, parm2, parm3, parm4 ])
}
```

7.5.3.1 Description

This section explains each feature of DBMS_SQL.

BIND_VARIABLE

- BIND_VARIABLE sets values in the host variable within the SQL statement.
- Specify 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.
- Specify 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 is a character 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.
- Specify the cursor number to be processed.
- The value returned is a NULL value.

Example

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

COLUMN_VALUE

- COLUMN_VALUE retrieves the value of the column in the select list extracted with FETCH_ROWS.
- Specify the cursor number to be processed.
- Specify the position of the column of the select list in the SELECT statement. The position of the first column is 1.
- Specify the destination variable name.
- Use a SELECT statement to obtain the values of the value, column_error, and actual_length columns.
- The value column returns the value of the column specified at the column position. The data type of the variable name must match that 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.
- The data type of the column_error column is NUMERIC. If the column value could not be set correctly in the value column, a value other than 0 will be returned:

22001: The extracted string has been truncated
22002: The extracted value contains a NULL value

- The data type of the actual_length column is INTEGER. If the extracted value is a character 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 retrieving 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 retrieving just 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.
- Specify the cursor number to be processed.
- Specify the position of the column in the select list in the SELECT statement. The position of the first column is 1.
- Specify the destination variable name. The data type 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.
- Specify the maximum number of characters of character 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.
- Specify 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 rows processed. Anything else is invalid.

Example

```
ret := DBMS_SQL.EXECUTE(cursor);
```

FETCH_ROWS

- FETCH_ROWS positions at the next row and extracts values from the row.
- Specify the cursor number to be processed.
- The return value is an INTEGER type and is the number of rows extracted. 0 is returned if all are extracted.
- The extracted information is retrieved 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.
- The parameter is used for compatibility with Oracle databases only, and is ignored by FUJITSU Enterprise Postgres. An INTEGER type can be specified, but it will be ignored. If migrating from an Oracle database, specify 1.
- 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.
- Specify the cursor number to be processed.
- Specify the SQL statement to be parsed.
- Parameters 1, 2, 3, and 4 are used for compatibility with Oracle databases only, and are ignored by FUJITSU Enterprise Postgres. If you are specifying values anyway, specify the following:
  - Parameter 1 is an INTEGER type. Specify 1.
  - Parameters 2 and 3 are TEXT types. Specify NULL.
  - Parameter 4 is a BOOLEAN type. 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 is reset, and the SQL statement is parsed anew.

Example

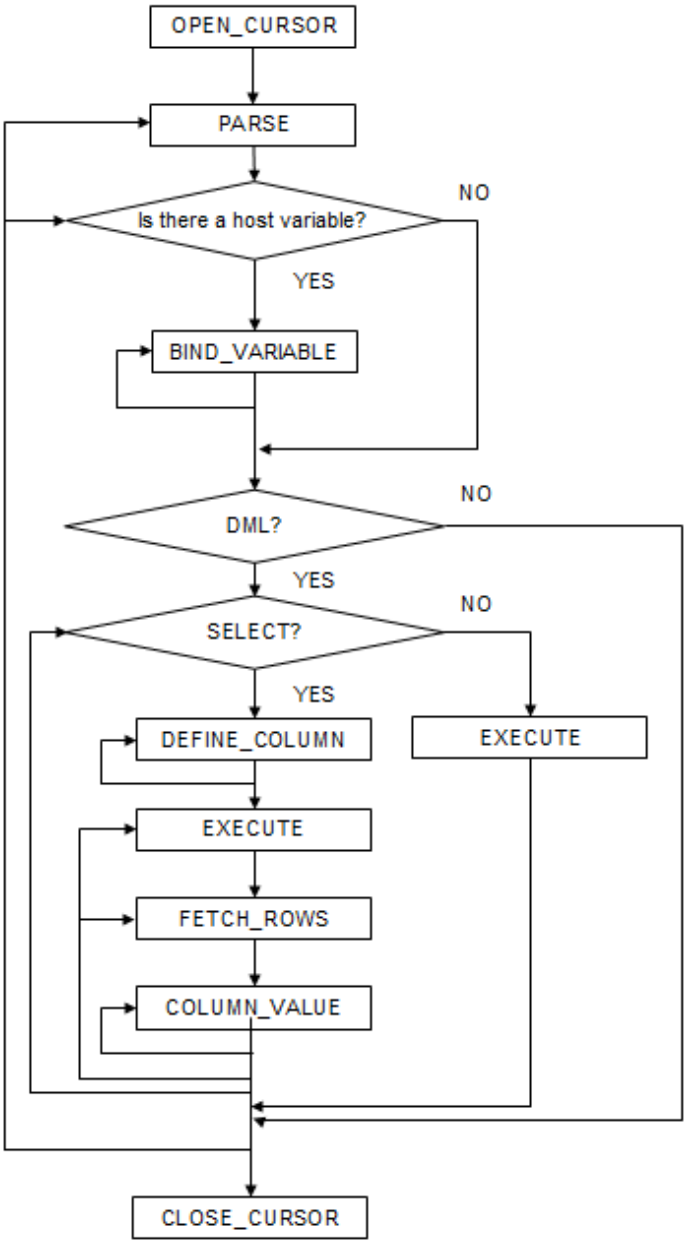
```
PERFORM DBMS_SQL.PARSE(cursor, 'SELECT memid, memnm FROM member WHERE memid = :NO', 1);
```

7.5.3.2 Example

This section explains the flow of DBMS_SQL and provides an example.

Flow of DBMS_SQL

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 8 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. A standby server can optionally be prioritized over the primary server as the target server.

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.

8.1 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:

host1,host2:port2

JDBC

If only one port number is specified, it will be assumed that host1: 27500 (the default value) and host2:port2 were specified.
Omit all port numbers, or specify only one per server.

Others

If only one port number is specified, it will be assumed that the same port is used for all the hosts.

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, "any" will be assumed.

Primary server

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.

Standby server (this value can be used only when the JDBC driver is used)

The standby server is selected as the connection target from the specified "IP addresses or host names". On standby server, the update will always fail. If the target server is not standby, the JDBC driver will throw an error stating that it is unable to find a server with the specified targetServerType.

Priority given to a standby server

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.

Any

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 write operations will always fail.

The table below shows the server selection order values to set for each driver:

Server selection order	JDBC drivers	Other drivers
Primary server	"master"	"read-write"
Standby server	"slave"	-
Priority given to a standby server	"preferSlave"	"prefer-read"
Any	"any"	"any"

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.

8.2 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.

8.2.1 Using the JDBC Driver

Set the following information in the connection string of the DriverManager class, or in the data source.

Table 8.1 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.
targetServerType	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]/dbName[?targetServerType={master | slave |
preferSlave | any}][&sslmode=verify-
full&sslrootcert=cACertificateFile&sslservercertcn=targetServerCertificateCN]
```

- If the target server is omitted, the default value "any" 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/dbName
```

When using the data source

Specify the properties of the data source in the following format:

```
source.setServerName("host1[:port1],host2[:port2]");
source.setTargetServerType("master");
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 "any".
- 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.

8.2.2 Using the ODBC Driver

Set the following information in the connection string or data source.

Table 8.2 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.

Parameter	Explanation
	If *n* server names are specified, and m ports are specified then there will be error reported. The only exceptions are where m=n or m=1. In case only one port is specified, then the same is applied for all the hosts.
target_session_attrs	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 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};[target_session_attrs={read-write | prefer-read | any}];[ 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}
target_session_attrs={read-write | prefer-read | any }
SSLMode=verify-full
SSLServerCertCN=targetServerCertificateCN
```

- When using IPV6, specify the host in the "*host*" format.

  [Example]

```
Servername={2001:Db8::1234,192.168.1.1}
```

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.

8.2.3 Using a Connection Service File

Set the connection parameters as follows.

Table 8.3 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, and m ports are specified then there will be error reported. The only exceptions are where m=n or m=1. In case only one port is specified, then the same is applied for all the hosts.
target_session_attrs	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*

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.

8.2.4 Using the C Library (libpq)

It is recommended that you use a connection service file. Refer to "8.2.3 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 8.4 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, and m ports are specified then there will be error reported. The only exceptions are where m=n or m=1. In case only one port is specified, then the same is applied for all the hosts.
target_session_attrs(PGTARGETSESSIONATTRS)	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_session_attrs={read-write | prefer-read | any }]
```

- 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_session_attrs={read-write| prefer-read | any }]
```

- 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
```

8.2.5 Using Embedded SQL

It is recommended that you use a connection service file. Refer to "8.2.3 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_session_attrs={read-write |
prefer-read | any}][&sslmode=verify-full&sslservercertcn=targetServerCertificateCN]
```

- The above format cannot be specified directly without using a literal or variable.

Table 8.5 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.
target_session_attrs	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 "8.2.4 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.

8.2.6 Using the psql Command

It is recommended that you use a connection service file. Refer to "8.2.3 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 8.6 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.
-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, and m -p options are specified then there will be error reported. The only exception is where m=n or m=1. In case only one port is specified, then the same is applied for all the hosts.
(PGTARGETSESSIONATTRS)	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 9 Performance Tuning

This chapter explains how to tune application performance.

9.1 Enhanced Query Plan Stability

FUJITSU Enterprise Postgres estimates the cost of query plans based on SQL statements and database statistical information, and selects the least expensive query plan. However, like other databases, FUJITSU Enterprise Postgres does not necessarily select the most suitable query plan. For example, it may suddenly select unsuitable query plan due to changes in the data conditions.

In mission-critical systems, stable performance is more important than improved performance, and changes in query plans case to be avoided. In this situation, by stabilizing the SQL statement query plan so that it does not change, deterioration of the application performance is suppressed.

9.1.1 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 FUJITSU 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* ...])

- 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*)

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.

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 "5.4.2 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 ()

9.1.2 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 so that the currently selected query plan remains selected.
	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	Outputs the current statistics to an external file (binary format).
	Import	Reads the statistics from an external file created by the export feature, 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.

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_table_stats('*schemaName.tableName*')
Unlock	Database	dbms_stats.unlock_database_stats()
	Schema	dbms_stats.unlock_schema_stats('*schemaName*')
	Table	dbms_stats.unlock_table_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-<*x*>.sql_sample" for information on the content of the COPY statement. "<*x*>" indicates the product version.

"export_effective_stats-<*x*>.sql_sample" is stored as follows:
fujitsuEnterprisePostgresInstallDir/share/doc/extension

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.

Chapter 10 Scan Using a Vertical Clustered Index (VCI)

This chapter describes scanning using a VCI.

10.1 Operating Conditions

Faster aggregation can be achieved by using a VCI defined for all columns to be referenced.

This section describes the conditions under which a scan can use a VCI.

Whether to use VCI is determined based on cost estimation in the same way as normal indexes. Therefore, another execution plan will be selected if it is cheaper than a VCI even if a VCI is available.

SQL statements that can use VCIs

In addition to general SELECT statements, VCIs can be used for the SQL statements below (as long as they do not specify any of the elements listed in "SQL statements that cannot use VCIs" below):

- SELECT INTO
- CREATE TABLE AS SELECT
- CREATE MATERIALIZED VIEW ... AS SELECT
- CREATE VIEW ... AS SELECT
- COPY (SELECT ...) TO

SQL statements that cannot use VCIs

VCIs cannot be used for SQL statements that specify any of the following:

- Subquery to reference the column in which the parent query is referencing is specified
- Lock clause (such as FOR UPDATE)
- Cursor declared with WITH HOLD or scrollable
- SERIALIZABLE transaction isolation level
- Function or operator listed in "Functions and operators that do not use a VCI"
- User-defined function

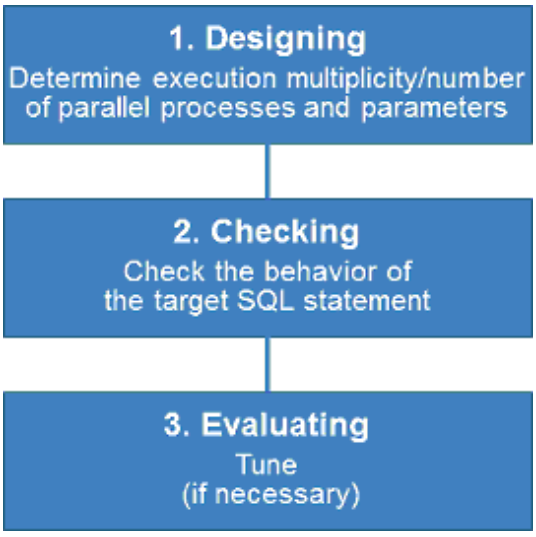
Table 10.1 Functions and operators that cannot use VCIs

Classification		Function/operator
Mathematical functions and operators	Random functions	random and setseed
String functions and operators	String functions	format (if the *format* argument is specified), regexp_matches, regexp_split_to_array and regexp_split_to_table
Date/time functions and operators	Date/time functions	age(timestamp), current_date, current_time, current_timestamp, localtime, localtimestamp, statement_timestamp and transaction_timestamp
	Delaying execution functions	pg_sleep, pg_sleep_for, and pg_sleep_until
Enum support functions		All functions and operators
Geometric functions and operators		All functions and operators
Network address functions and operators		All functions and operators
Text search functions and operators		All functions and operators
XML functions		All functions

Classification		Function/operator
JSON functions and operators		All functions and operators
Sequence manipulation functions		All functions
Array functions and operators		All functions and operators
Range functions and operators		All functions and operators
Aggregate functions	General-purpose aggregate functions	array_agg, json_agg, json_object_agg, string_agg and xmlagg
	Aggregate functions for statistics	corr, covar_pop, covar_samp, regr_avgx, regr_avgy, regr_count, regr_intercept, regr_r2, regr_slope, regr_sxx, regr_sxy and regr_syy
	Ordered-set aggregate functions	All functions
	Hypothetical-set aggregate functions	All functions
Window functions		All functions
Subquery expressions		Subquery expressions with its row constructor specified on the left side
Row and array comparisons		Row constructor and composite type comparisons
Set returning functions		All functions
System information functions		All functions
System administration functions		All functions
Trigger functions		All functions
Session information functions		current_role and current_user

10.2 Usage

This section describes how to use a VCI in line with the following steps:



10.2.1 Designing

Design as follows before using a VCI.

- Execution multiplicity and number of parallel processes

- Parameters

Execution multiplicity and number of parallel processes

Determine the maximum number of SQL statements that can be executed simultaneously and the number of parallel processes based on the number of CPU cores that can be allocated for scans that use VCI to perform aggregate processing. Design in advance the multiplicity of SQL statements for executing scans that use VCI and the number of parallel processes for scans that use VCI.

For example, if the number of CPUs that can be allocated is 32 cores, then the maximum number of SQL statements that can be executed simultaneously is 8 and the number of parallel processes is 4.

Note

A temporary file is created in /dev/shm or in a directory specified for the vci.smc_directory parameter as the dynamic shared memory for each SQL statement during a scan using a VCI.

Ensure that this directory has sufficient space to meet the memory requirements estimated for the execution multiplicity and number of parallel processes of SQL statements (refer to "Memory used per scanning" in "VCI Memory Requirements" in the Installation and Setup Guide for Server for details). If it does not have sufficient space when a scan is performed, SQL statements will return errors due to the insufficient memory.

Parameters

The VCI parallel scan feature cannot be used for setting parameters immediately after creating an instance.

Therefore, set the parameters below based on the values determined in "Execution multiplicity and number of parallel processes of SQL statements" above.

Parameter name	Description	Default	Value index
vci.max_parallel_degree	Maximum number of VCI parallel processes (background processes) to be used per SQL statement.	0	Specify the number of parallel processes.
vci.smc_directory	Directory name in which a temporary file is created as the dynamic shared memory during a scan using a VCI.	/dev/shm	Specify a directory that has enough free space for the memory used for each query during the scan.
max_worker_processes	Maximum number of background processes that the system supports.	8	Add the value of the maximum number of SQL statements that can be executed simultaneously for scans that use VCI multiplied by vci.max_parallel_degree.

See

Refer to "Parameters" in the Operation Guide for information on the details of and how to set the parameters.

10.2.2 Checking

Execute the SQL statement with "EXPLAIN ANALYZE" to check the following:

- If a VCI was used
  "Custom Scan (VCI...)" is displayed in the plan if a VCI was used.
- Number of parallel processes
  The number of parallel processes when the SQL statement is executed is displayed in "Allocated Workers". Check that it is running the designed number of parallel processes.

- Response
  Check if the execution time displayed in "Execution time" is as estimated.

The following shows an example of the output result of EXPLAIN ANALYZE:

```
EXPLAIN ANALYZE SELECT COUNT(*) FROM test WHERE x > 10000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
--------------
Custom Scan (VCI Aggregate)  (cost=19403.15..19403.16 rows=1 width=0) (actual time=58.505..58.506
rows=1 loops=1)
       Allocated Workers: 4
   ->  Custom Scan (VCI Scan) using test_x_idx on test (cost=0.00..16925.00 rows=991261 width=0)
(never executed)
         Filter: (x > 10000)
Planning time: 0.151 ms
Execution time: 86.910 ms
(6 rows)
```

A cost output by the execution plan that uses a VCI may be inaccurate. A VCI works if all or part of the best execution plan when the SQL statement was executed is replaced with an execution plan that uses a VCI. If the cost of the execution plan to be replaced is lower than a certain value (vci.cost_threshold parameter), it will not be replaced or recalculated. Therefore, the cost of the original execution plan is output as is.

10.2.3 Evaluating

If the results in "10.2.2 Checking" is any of the following, tune accordingly:

If a VCI is not used

- Check if the "10.1 Operating Conditions" are met.
- Check if vci.enable is set to "on".
- A VCI may not be appropriately used when statistics are outdated, such as immediately after inserting a large amount of data. In such cases, execute the VACUUM ANALYZE statement or the ANALYZE statement.
- A VCI is not used if there is insufficient memory for VCI scan. This may occur during time-consuming transactions involving tables for which VCIs were defined. Set vci.log_query to "on", and check if either "could not use VCI: local ROS size (%zu) exceeds limit (%zu)" or "out of memory during local ROS generation" is output. If it is, then increase the value of the vci.max_local_ros.

Response is not as expected

Tuning may improve response. Check the following:

- If vci.max_parallel_degree is not set or is set to 0, set an appropriate value according to "10.2.1 Designing".
- If there is a margin in the CPU usage, increase the value of vci.max_parallel_degree and check again. In addition, if the value that of max_worker_processes is lower than the maximum number of SQL statements that can be executed simultaneously for parallel scan multiplied by vci.max_parallel_degree, increase it and check again.

10.3 Usage Notes

This section provides notes on using VCI.

- Regardless of whether VCI is used, the content of the result does not change. However, records may be returned in a different order if the ORDER BY clause is not specified.
- To reduce resource consumption, edit postgresql.conf or use the SET statement to enable/disable vci.enable when you use this feature only for specific times or jobs (SQL applications).

- The optimizer hint (pg_hint_plan) cannot be specified for a VCI. The hint clause is ignored if it is specified.
- If a plan other than VCI is specified for the optimizer hint (pg_hint_plan), a VCI may be used. Therefore, if you specify a query plan with the hint clause, use the SET statement to set vci.enable to "off".
- The message below may be output when a scan that uses VCI is performed on the streaming replication standby server:

```
"LOG: recovery has paused"
"HINT: Execute pg_wal_replay_resume() to continue."
```

  This message is output because application of the WAL to the VCI temporarily pauses due to the scan being performed.

- Even if a scan is performed using a VCI, information in the idx_scan, idx_tup_read, and idx_tup_fetch columns of the collected statistics views, pg_stat_all_indexes and pg_stat_user_indexes, will not be updated.
- Currently, it is not possible to replace the query plan for parallel aggregation with the query plan using VCI. Therefore, if you create a VCI on a column of a partition table and aggregate (sum () etc.) on that column, one of the following plans will be selected. Use different setting parameters according to the situation of the target table.
  - Plan of the parallel aggregations using scan methods other than VCI scan

    It is selected when max_parallel_workers_per_gather is 1 or more.

```
explain select sum(value) from test;
                                        QUERY PLAN
-------------------------------------------------------------------------------
 Finalize Aggregate  (cost=99906.30..99906.31 rows=1 width=8)
   ->  Gather  (cost=99906.08..99906.29 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=98906.08..98906.09 rows=1 width=8)
               ->  Parallel Append  (cost=0.00..94739.83 rows=1666500 width=4)
                     ->  Parallel Seq Scan on test_1  (cost=0.00..43203.67 rows=833250 width=4)
                     ->  Parallel Seq Scan on test_2  (cost=0.00..43203.67 rows=833250 width=4)
```

    This plan is fast when the number of records to be aggregated (number of records that hit the search conditions) is very large. This is because the benefit of parallelizing aggregation is important, not the performance of scanning. For example, each parallel worker will perform a sequential scan and aggregate most of the scanned records.

  - Plan that aggregates VCI scan results by a single aggregator node

    It is selected by setting max_parallel_workers_per_gather to 0 and not creating a query plan of parallel aggregate.

```
explain select sum(value) from test;
                                        QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=145571.00..145571.01 rows=1 width=8)
   ->  Append  (cost=0.00..135572.00 rows=3999600 width=4)
         ->  Custom Scan (VCI Scan) using test_1_id_value_idx on test_1  (cost=0.00..57787.00
rows=1999800 width=4)
               Allocated Workers: 2
         ->  Custom Scan (VCI Scan) using test_2_id_value_idx on test_2  (cost=0.00..57787.00
rows=1999800 width=4)
               Allocated Workers: 2
```

    This plan is fast when the number of aggregated items is not large or when the size of the aggregated column is smaller than the record size. This is because the scan performance is more important, so it is faster to aggregate the results of VCI scans of each partition.

- Originally, if there is only one partition to be accessed, the following VCI aggregation plan can be used. Below is an example of scanning only one partition with partition pruning.

```
explain select sum(value) from test where id < 1000001;
                                        QUERY PLAN
-------------------------------------------------------------------------------
 Custom Scan (VCI Aggregate)  (cost=62786.50..62786.51 rows=1 width=8)
   Allocated Workers: 2
```

```
   ->  Custom Scan (VCI Scan) using test_1_id_value_idx on test_1  (cost=0.00..57787.00
rows=1999800 width=4)
         Filter: (id < 1000001)
```

However, the current planner does not try to choose VCI aggregation because it creates a plan for parallel aggregation if the table is partitioned. So in this case, set max_parallel_workers_per_gather to 0 to force the planner to choose VCI aggregation.

Appendix A Precautions when Developing Applications

This appendix describes precautions when developing applications with FUJITSU 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 FUJITSU 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 FUJITSU 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.

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 FUJITSU 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
	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
	MACADDR8	Y	N	N
Bit string type	BIT	Y	N	Y

Conversion target		Conversion source		
		Character literal (*1)	Numeric literal(*2)	Bit character string literal
	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.

See

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.

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:

- SP-GiST index

A.4.1 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 Multibyte Characters in Definition Names

Multibyte characters must not be used in database names or user names, 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.

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 FUJITSU 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
JDBC driver	Cannot be connected
ODBC driver	No connection constraints
SQLEmbedded SQL in C	No connection constraints
psql command	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 FUJITSU Enterprise Postgres, within the scope noted in "Chapter 7 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

FUJITSU 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.

FUJITSU 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

FUJITSU 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.

FUJITSU 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

FUJITSU 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.

FUJITSU 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

FUJITSU 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.

FUJITSU 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 "7.2.1 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

FUJITSU 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.

FUJITSU 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;
```

FUJITSU 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.

FUJITSU 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 FUJITSU 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

FUJITSU 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.

FUJITSU 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

FUJITSU 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.

FUJITSU Enterprise Postgres

Value expressions with different data types cannot be specified.

Conversion procedure

Since the data types that can be specified for the expressions in the two arguments are unknown, use the following steps to convert:

1. Check the data types specified for each of the two expressions.
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

FUJITSU 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.

FUJITSU 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

FUJITSU 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.

FUJITSU 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;
/
```

FUJITSU 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.

FUJITSU 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.

FUJITSU 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;
/
```

FUJITSU 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.

FUJITSU 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

FUJITSU 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.

FUJITSU 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)
```

FUJITSU 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.

FUJITSU 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');
```

FUJITSU 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.

FUJITSU 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.

FUJITSU 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.

FUJITSU 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.

FUJITSU 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.

FUJITSU 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');
```

FUJITSU 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.

FUJITSU 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');
```

FUJITSU 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');
```

FUJITSU 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.

FUJITSU 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

FUJITSU 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.

FUJITSU 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.

FUJITSU 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 Quantitative Limits

This appendix lists the quantitative limits of FUJITSU Enterprise Postgres.

Table D.1 Length of identifier

Item	Limit
Database name	Up to 63 bytes (*1) (*2)
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)
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.

Table D.2 Database object

Item	Limit
Number of databases	Less than 4,294,967,296 (*1)

Item	Limit
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 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 D.3 Schema element

Item	Limit
Number of columns that can be defined in one table	From 250 to 1600 (according to the data type)
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)

Item	Limit
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 D.4 Index

Item	Limit
Number of columns comprising a key (including VCI)	Up to 32 columns
Key length (other than VCI)	Less than 2,000 bytes (*1)

*1: This is the character string byte length when converted by the server character set character code.

Table D.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 D.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 D.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
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 D.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 E Reference

E.1 JDBC Driver

See

Refer to the Java API Reference for information on PostgreSQL JDBC driver.

E.2 ODBC Driver

E.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	Y
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
SQLConnectW	Y
SQLCopyDesc	Y
SQLDataSources	Y
SQLDataSourcesW	Y
SQLDescribeCol	Y

Function name	Support status
SQLDescribeColW	Y
SQLDescribeParam	Y
SQLDisconnect	Y
SQLDriverConnect	Y
SQLDriverConnectW	Y
SQLDrivers	Y
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
SQLGetDiagRec	Y
SQLGetDiagRecW	Y
SQLGetEnvAttr	Y
SQLGetFunctions	Y
SQLGetInfo	Y

Function name	Support status
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
SQLSetScrollOptions	N
SQLSetStmtAttr	Y
SQLSetStmtAttrW	Y
SQLSetStmtOption	Y
SQLSpecialColumns	Y

Function name	Support status
SQLSpecialColumnsW	Y
SQLStatistics	Y
SQLStatisticsW	Y
SQLTablePrivileges	Y
SQLTablePrivilegesW	Y
SQLTables	Y
SQLTablesW	Y
SQLTransact	Y

Y: Supported
N: Not supported

E.3 C Library (libpq)

See

Refer to "libpq - C Library" in "Client Interfaces" in the PostgreSQL Documentation.

E.4 Embedded SQL in C

See

Refer to "ECPG - Embedded SQL in C" in "Client Interfaces" in the PostgreSQL Documentation.

Index

[B]

BIND_VARIABLE..... 57

[C]

CLOSE_CURSOR.....58
Code examples for applications.....27
COLUMN_VALUE..... 58
Comparison operator..... 3

[D]

DBMS_OUTPUT..... 46
DBMS_SQL..... 56
DECODE..... 42
DEFINE_COLUMN.....59
DISABLE..... 47
DUAL Table..... 41

[E]

ENABLE..... 46
Encoding System Settings..... 18,22
Example of specifying the hint clause.....28
EXECUTE..... 59

[F]

FCLOSE..... 51
FCLOSE_ALL.....51
FCOPY..... 51
FETCH_ROWS.....60
FFLUSH..... 51
FGETATTR.....52
FOPEN.....52
FRENAME..... 52

[G]

GET_LINE..... 48

[I]

IS_OPEN..... 53

[L]

Language settings..... 7,18,21

[N]

NEW_LINE..... 48,54
NVL..... 45

[O]

OPEN_CURSOR.....60
Outer Join Operator (+)..... 39

[P]

PARSE.....60
Pattern matching..... 3
Precompiling example..... 28
PUT.....47,54
PUTF..... 54
PUT_LINE.....47,54

[S]

Scan Using a Vertical Clustered Index (VCI)..... 80
Settings for encrypting communication data for connection to the server..... 7
String functions and operators..... 3
SUBSTR..... 43

[U]

UTL_FILE..... 49

[W]

When setting from outside with environment variables.....22
When specifying in the connection URI.....22