0. Introduction

PostgreSQL is an object-relational database management system (ORDBMS) based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department. POSTGRES pioneered many concepts that only became available in some commercial database systems much later. PostgreSQL is an open-source descendant of this original Berkeley code. It supports SQL92 and SQL99 and offers many modern features:

- complex queries
- foreign keys
- triggers
- views
- transactional integrity
- multiversion concurrency control

Also, PostgreSQL can be extended by the user in many ways, for example by adding new

- data types
- functions
- operators
- aggregate functions
- index methods
- procedural languages

And because of the liberal license, PostgreSQL can be used, modified, and distributed by everyone free of charge for any purpose, be it private, commercial, or academic.

1. Getting started

1.1 Architecture of Postgresql

Using a client/server model, PostgreSQL session consists of a server process and client applications. The server process, which manages the database files, accepts connections to the database from client applications, and performs actions on the database on behalf of the clients. The database server program is called postmaster.

1.2 Installing PostgreSQL on Windows

PostgreSQL doesn't yet natively support Windows, but it will with the (soon) upcoming version 8 release. The free option at present is to use PostgreSQL running through Cygwin (http://sources.redhat.com/cygwin). If you're not familiar with cygwin, it's a unix emulation layer that allows many Unix specific programs to be compiled for and run on Windows. It's very easy to install, coming with its own installation program, and one of the options at installation time is to install

PostgreSQL. Also, you can build PostgreSQL in Cygwin by yourself and use all of PostgreSQL "the Unix way".

1.3 Installing PostgreSQL

1. Getting the source code

The primary anonymous ftp site for PostgreSQL is ftp://ftp.PostgreSQL.org/pub.

2. Installing PostgreSQL

To install PosgreSQL, GNU make and an ISO/ANSI C compiler is required. gzip is also needed to unpack the distribution in the first place. By default, the GNU Readline library will be used.

The installation of PosgreSQL consists of four steps shown below:

- Configuration
  For a default installation simply enter "./configure".
- Build
  To start the build, type "gmake".
- Regression Tests
  Type "gmake check". (This won't work as root; do it as an unprivileged user.) You can repeat this test at any later time by issuing the same command.
- Installing The Files
  To do a full installation of PostgreSQL enter "gmake install"

To install only the client applications and interface libraries, then you can use these commands:

gmake -C src/bin install
gmake -C src/include install
gmake -C src/interfaces install
gmake -C doc install

3. Creating the Postgresql USER

To create a user, you should use the CREATE USER SQL command: CREATE USER name;

To remove an existing user, use the analogous DROP USER command: DROP USER name;

For convenience, the programs createuser and dropuser are provided as wrappers around these SQL commands that can be called from the shell command line: createuser name/dropuser name

In order to bootstrap the database system, a freshly initialized system always contains one predefined user. This user will have the same name as the operating system user that initialized the database cluster. Customarily, this user will be named postgres. In order to create more users you first have to connect as this initial user.

4. Configuration and compilation issues

- Shared Libraries

On some systems that have shared libraries (which most systems do) you need to tell your system how to find the newly installed shared libraries. The method to set the shared library search path varies between platforms, but the most widely usable method is to set the environment variable LD_LIBRARY_PATH.

- Environment Variables

If you installed into /usr/local/pgsql or some other location that is not searched for programs by default, you should add the location information into your PATH.

- Initializing Postgresql

Firstly, create a new user in system specially for PostgreSQL

```
adduser postgres
```

Secondly, add a data directory and change its owner to the newly created one

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
```

Thirdly, run initdb to initialize a database storage area on disk, a database cluster

```
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

Forthly, start Postgresql server

```
/usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
```

Now, you may test your PostgreSQL

```
/usr/local/pgsql/bin/createdb test
/usr/local/pgsql/bin/psql test
```

1.4 Uninstalling Posgresql

To undo the installation use the command gmake uninstall. However, this will not remove any created directories.

1.5 Running a Postgresql server

Before anyone can access the database, you must start the database server. The database server program is called postmaster. The postmaster must know where to find the data it is supposed to use. This is done with the -D option. Thus, the simplest way to start the server is:

```
$ postmaster -D /usr/local/pgsql/data
```

which will leave the server running in the foreground. To start the postmaster in the background, use the usual shell syntax:

```
$ postmaster -D /usr/local/pgsql/data >logfile 2>&1 &
```

In particular, in order for the server to accept TCP/IP connections (rather than just Unix-domain socket ones), you must specify the -i option.

2. Using the pgsql and pgadmin clients

2.1 Creating a database in pgsql

You need to be connected to the database server in order to create databases. There are two ways to create databases. One is the SQL command CREATE DATABASE in SQL environment, the other

is the command createdb in the shell.

CREATE DATABASE dbname;	create a database for the current user.
CREATE DATABASE dbname OWNER username;	create a database for some else
createdb dbname;	create a database for the current user.
createdb -O username dbname;	create a database for some else

2.2 Accessing a database (a session) in pgsql

Once you have created a database, you can access it by:

- Running the PostgreSQL interactive terminal program, called psql, which allows you to interactively enter, edit, and execute SQL commands.
- Using an existing graphical frontend tool like PgAccess or an office suite with ODBC support to create and manipulate a database. These possibilities are not covered in this tutorial.
- Writing a custom application, using one of the several available language bindings.

2.3 Controling and exiting a session in pgsql

The psql program has a number of internal commands that are not SQL commands. They begin with the backslash character, "\". Some of these commands were listed in the welcome message.
To get out of psql, type “mydb=> \q” and psql will quit and return you to your command shell.

2.4 Using pgadmin

pgAdmin III is a powerful administration and development platform for the PostgreSQL database, free for any use. The application is running under GNU/Linux, FreeBSD and Windows 2000/XP.
pgAdmin III is designed to answer the needs of all users, from writing simple SQL queries to developing complex databases. The graphical interface supports all PostgreSQL features and makes administration easy. The application also includes a query builder, an SQL editor, a server-side code editor and much more. pgAdmin III is released with an installer and does not require any additional driver to communicate with the database server. For more information and download, visit http://www.pgadmin.org.

3. The Postgresql SQL language

3.1 Basic SQL commands in Postgresql

To create a table, you use the aptly named CREATE TABLE command. For example:

```
CREATE TABLE table1 (name text, price integer);
```

An example command to insert a row would be:

```
INSERT INTO products VALUES ('Cheese', 9.99);
```

For example, this command updates all products that have a price of 5 to have a price of 10:

UPDATE products SET price = 10 WHERE price = 5;

To remove all rows from the products table that have a price of 10, use

DELETE FROM products WHERE price = 10;

An example of query is

SELECT * FROM table1 WHERE price = 10 ;

4. Server administration

4.1 Server Run-time Environment

There are a lot of configuration parameters that affect the behavior of the database system. Every parameter takes a value of one of the four types: boolean, integer, floating point, and string. Boolean values are ON, OFF, TRUE, FALSE, YES, NO, 1, 0 (case-insensitive) or any non-ambiguous prefix of these. One way to set these parameters is to edit the file postgresql.conf in the data directory. (A default file is installed there.) A second way to set these configuration parameters is to give them as a command line option to the postmaster, such as: “postmaster -c log_connections=yes -c syslog=2”

4.2 Database Users and Privileges

A database user may have a number of attributes that define its privileges and interact with the client authentication system. A user's attributes can be modified after creation with ALTER USER. A user can also set personal defaults for many of the run-time configuration settings. For example, if for some reason you want to disable index scans (hint: not a good idea) anytime you connect, you can use

ALTER USER myname SET enable_indexscan TO off

There are several different privileges: SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE, TEMPORARY, EXECUTE, USAGE, and ALL PRIVILEGES. To assign privileges, the GRANT command is used. So, if joe is an existing user, and accounts is an existing table, the privilege to update the table can be granted with

GRANT UPDATE ON accounts TO joe;

To revoke a privilege, use the fittingly named REVOKE command:

REVOKE ALL ON accounts FROM PUBLIC;

The special privileges of the table owner (i.e., the right to do DROP, GRANT, REVOKE, etc) are always implicit in being the owner, and cannot be granted or revoked. But the table owner can choose to revoke his own ordinary privileges, for example to make a table read-only for himself as well as others.

4.3 Managing Databases

1. Backup

One obvious maintenance task is creation of backup copies of the data on a regular schedule.

Without a recent backup, you have no chance of recovery after a catastrophe (disk failure, fire, mistakenly dropping a critical table, etc.).

2. **Routine Vacuuming**

PostgreSQL's VACUUM command must be run on a regular basis for several reasons:

- To recover disk space occupied by updated or deleted rows.
- To update data statistics used by the PostgreSQL query planner.
- To protect against loss of very old data due to transaction ID wraparound.

The frequency and scope of the VACUUM operations performed for each of these reasons will vary depending on the needs of each site.

3. **Routine Reindexing**

In some situations it is worthwhile to rebuild indexes periodically with the REINDEX command. (There is also contrib/reindexdb which can reindex an entire database.) However, PostgreSQL 7.4 has substantially reduced the need for this activity compared to earlier releases.

4. **Log File Maintenance**

The simplest production-grade approach to managing log output is to send it all to syslog and let syslog deal with file rotation. To do so, you should set the configurations parameter syslog to 2 (to log to syslog only) in postgresql.conf. However, syslog is not very reliable in some cases, then piping the stderr of the postmaster to some type of log rotation program is very usful. If you start the server with pg_ctl, then the stderr of the postmaster is already redirected to stdout, so you just need a pipe command: “pg_ctl start | logrotate”

4.4 Client Authentication

PostgreSQL offers a number of different client authentication methods. The method used to authenticate a particular client connection can be selected on the basis of (client) host address, database, and user. Client authentication is controlled by the file pg_hba.conf in the data directory, e.g., /usr/local/pgsql/data/pg_hba.conf. (HBA stands for host-based authentication.) A default pg_hba.conf file is installed when the data directory is initialized by initdb.

4.5 Backup and Restore

There are two fundamentally different approaches to backing up PostgreSQL data, SQL dump and File system level backup. The idea behind the SQL-dump method is to generate a text file with SQL commands that, when fed back to the server, will recreate the database in the same state as it was at the time of the dump. PostgreSQL provides the utility program pg_dump for this purpose.

4.6 Monitoring Database Activity and Disk Usage

On most platforms, PostgreSQL modifies its command title as reported by ps, so that individual server processes can readily be identified using ps. (The appropriate invocation of ps varies across different platforms). You can monitor disk space from three places: from psql using VACUUM

information, from psql using the tools in contrib/dbsize, and from the command line using the tools in contrib/oid2name.

5. Programming Interfaces

5.1 C/C++

libpq is the C application programmer's interface to PostgreSQL. libpq is a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries. libpq is also the underlying engine for several other PostgreSQL application interfaces, including libpq++ (C++), libpgtcl (Tcl), Perl, and ECPG. So some aspects of libpq's behavior will be important to you if you use one of those packages. For example, PQconnectdb or PQsetdbLogin deasl with making a connection to a PostgreSQL backend server; the PQstatus function should be called to check whether a connection was successfully made before queries are sent via the connection object; function PQexec Submits a command to the server and waits for the result.

5.2 JDBC

1. **Getting the Driver**

Precompiled versions of the driver can be downloaded from the PostgreSQL JDBC web site.

2. **Setting up the Class Path**

To use the driver, the JAR archive (named postgresql.jar if you built from source, otherwise it will likely be named pg7.4jdbc1.jar, pg7.4jdbc2.jar, or pg7.4jdbc3.jar for the JDBC 1, JDBC 2, and JDBC 3 versions respectively) needs to be included in the class path, either by putting it in the CLASSPATH environment variable, or by using flags on the java command line.

For instance, assume we have an application that uses the JDBC driver to access a database, and that application is installed as /usr/local/lib/myapp.jar. The PostgreSQL JDBC driver installed as /usr/local/pgsql/share/java/postgresql.jar. To run the application, we would use:

```
export CLASSPATH=/usr/local/lib/myapp.jar:/usr/local/pgsql/share/java/postgresql.jar:.
java MyApp
```

3. **Preparing the Database Server for JDBC**

Because Java only uses TCP/IP connections, the PostgreSQL server must be configured to accept TCP/IP connections. This can be done by setting tcpip_socket = true in the postgresql.conf file or by supplying the -i option flag when starting postmaster.

4. **Initializing the Driver**

- Importing JDBC

Any source that uses JDBC needs to import the java.sql package, using: “import java.sql.*;” Note: Do not import the org.postgresql package. If you do, your source will not compile, as javac will get confused.

- Loading the Driver

Before you can connect to a database, you need to load the driver. There are two methods available, and it depends on your code which is the best one to use. In the first method, your code implicitly loads the driver using the Class.forName() method. For PostgreSQL, you would use:

Class.forName("org.postgresql.Driver");

This will load the driver, and while loading, the driver will automatically register itself with JDBC. Note: The forName() method can throw a ClassNotFoundException if the driver is not available. This is the most common method to use, but restricts your code to use just PostgreSQL. If your code may access another database system in the future, and you do not use any PostgreSQL-specific extensions, then the second method is advisable. The second method passes the driver as a parameter to the JVM as it starts, using the -D argument. Example:

java -Djdbc.drivers=org.postgresql.Driver example.ImageViewer

In this example, the JVM will attempt to load the driver as part of its initialization. Once done, the ImageViewer is started. Now, this method is the better one to use because it allows your code to be used with other database packages without recompiling the code. The only thing that would also change is the connection URL, which is covered next. One last thing: When your code then tries to open a Connection, and you get a No driver available SQLException being thrown, this is probably caused by the driver not being in the class path, or the value in the parameter not being correct.

- Connecting to the Database

With JDBC, a database is represented by a URL (Uniform Resource Locator). With PostgreSQL, this takes one of the following forms:

jdbc:postgresql:database

jdbc:postgresql://host/database

jdbc:postgresql://host:port/database

To connect, you need to get a Connection instance from JDBC. To do this, you use the DriverManager.getConnection() method:

Connection db = DriverManager.getConnection(url, username, password);

- Closing the Connection

To close the database connection, simply call the close() method to the Connection: “db.close();”

5. Issuing a Query and Processing the Result

Any time you want to issue SQL statements to the database, you require a Statement or PreparedStatement instance. Once you have a Statement or PreparedStatement, you can use issue a query. This will return a ResultSet instance, which contains the entire result.

- Getting results based on a cursor

By default the driver collects all the results for the query at once. This can be inconvenient for large data sets so the JDBC driver provides a means of basing a ResultSet on a database cursor and only fetching a small number of rows. A small number of rows are cached on the client side of the connection and when exhausted the next block of rows is retrieved by repositioning the cursor.

- Using the Statement or PreparedStatement Interface

You can use a single Statement instance as many times as you want. You could create one as soon as you open the connection and use it for the connection's lifetime. But you have to remember that only one ResultSet can exist per Statement or PreparedStatement at a given time. If you need to

perform a query while processing a ResultSet, you can simply create and use another Statement. If you are using threads, and several are using the database, you must use a separate Statement for each thread. When you are done using the Statement or PreparedStatement you should close it.

- Using the ResultSet Interface

Before reading any values, you must call next(). This returns true if there is a result, but more importantly, it prepares the row for processing. Under the JDBC specification, you should access a field only once. It is safest to stick to this rule, although at the current time, the PostgreSQL driver will allow you to access a field as many times as you want. You must close a ResultSet by calling close() once you have finished using it. Once you make another query with the Statement used to create a ResultSet, the currently open ResultSet instance is closed automatically.

6. Performing Updates

To change data (perform an INSERT, UPDATE, or DELETE) you use the executeUpdate() method. This method is similar to the method executeQuery() used to issue a SELECT statement, but it doesn't return a ResultSet; instead it returns the number of rows affected by the INSERT, UPDATE, or DELETE statement.

7. Calling Stored Functions

PostgreSQL's JDBC driver fully supports calling PostgreSQL stored functions. The following is an example for calling a built in stored function.

```
// Turn transactions off.
con.setAutoCommit(false);
// Procedure call.
CallableStatement upperProc = con.prepareCall("{ ? = call upper( ? ) }");
upperProc.registerOutParameter(1, Types.VARCHAR);
upperProc.setString(2, "lowercase to uppercase");
upperProc.execute();
String upperCased = upperProc.getString(1);
upperProc.close();
```

- Using the CallableStatement Interface

All the considerations that apply for Statement and PreparedStatement apply for CallableStatement but in addition you must also consider one extra restriction:
You can only call a stored function from within a transaction.

- Obtaining ResultSet from a stored function

PostgreSQL's stored function can return results by means of a refcursor value. A refcursor.
As an extension to JDBC, the PostgreSQL JDBC driver can return refcursor values as ResultSet values. It is also possible to treat the refcursor return value as a distinct type in itself. The JDBC driver provides the org.postgresql.PGRefCursorResultSet class for this purpose.

8. Creating and Modifying Database Objects

To create, modify or drop a database object like a table or view you use the execute() method. This method is similar to the method executeQuery(), but it doesn't return a result. For example,

```
Statement st = db.createStatement();
st.execute("DROP TABLE mytable");
```

```
st.close();
```

5.3 PHP

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Instead of writing a program with lots of commands to output HTML, you write an HTML script with some embedded code to do something. The PHP code is enclosed in special start and end tags that allow you to jump into and out of "PHP mode". What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server. If you were to have a script similar to the above on your server, the client would receive the results of running that script, with no way of determining what the underlying code may be. You can even configure your web server to process all your HTML files with PHP, and then there's really no way that users can tell what you have up your sleeve. The best things in using PHP are that it is extremely simple for a newcomer, but offers many advanced features for a professional programmer. Don't be afraid reading the long list of PHP's features. You can jump in, in a short time, and start writing simple scripts in a few hours. Although PHP's development is focused on server-side scripting, you can do much more with it. For more information, please visit http://www.php.net.