Tutorial for Course of Spatial Database

Spring 2013

Rui Zhu

School of Architecture and the Built Environment
Royal Institute of Technology-KTH
Stockholm, Sweden

April 2013

Lab 1: Introduction to RDBMS and SQL

Due March 27th, 2013

1. Background

This introductory lab is designed for students that are new to databases and have no or limited knowledge about databases and SQL. The instructions first introduce some basic relational databases concepts and SQL, then show a number of tools that allow users to interact with PostgreSQL/PostGIS databases (pgAdmin: administration and management tool administration; QuantumGIS). Lastly, the instructions point to a short and simple SQL tutorial using an online PostgreSQL/PostGIS database and ask the students to write a few simple queries against this database.

2. Tutorial

2.1. What is SQL?

As it is stated in Wikipedia, SQL (Structured Query Language) is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS). SQL is a simple and unified language, which offers many features to make it as a powerfully diverse that people can work with a secure way. Using SQL language, one can retrieve data from a database, insert new records into databases, update/modify and delete existed records. SQL is also a ANSI standard computer language in the field of database, and supported by most commercial and free/open-source database systems, for instances the Oracle, DB2, Informix, SQL Server, MySQL, PostgreSQL, Access (Microsoft Office) and so on. Because of its simplicity, SQL is quite easy to learn and becoming the main interface for both users and developers to manipulate their data stored in relational database management systems (RDBMS).

The first version of SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. It was later formally standardized by the American National Standards Institute (ANSI) in 1986. Subsequent versions of the SQL standard have been released as International Organization for Standardization (ISO) standards (cited from Wikipedia). In the subsequent versions of SQL standards, many other extension have been adopted, such as procedural constructs, control-of-flow statements, user-defined data types, and various other language extensions. Among all the versions, the SQL: 1999 is the most remarkable one which formally accepts many of the above extension as part of the ISO standard for SQL, even the support for geographical data types (SQL99 MM/Spatial). Concerning the SQL standards for spatial databases, OGC and ISO/TC211 have made great efforts and SQL/MM spatial (as well OGC Simple Feature Specification for SQL) is the one you will go along with in the remaining parts of this course.

Although SQL is an ANSI and ISO standard computer language for accessing and manipulating data within database systems, different vendors might have proprietary extensions and supports for special but non-standard data types in their version of SQL. Anyway, in order to be in compliance with the ANSI/ISO standard, they must support the same major keywords in a similar manner (such as **`SELECT`**, **`UPDATE`**, **`DELETE`**, **`INSERT`**, **`WHERE`**, and others).

2.2. Basic Concepts in Relation Database Systems

In relational database systems, the data are organized as tables. In some cases, tables for the same purpose (e. g. tables used in the same application) could be grouped and saved in a database (e.g. the PostgreSQL, but in some other database systems, it might be called a table space). But for the desktop RDB product Access, this feature is not supported. Each Access file (*.mdb) could only host a single database, and all the data tables are saved inside this database.

A data **table** (or simply a table) denotes the information needed to describe a certain kind/class of objects in the real world. Taking the information system of a university for the example, you can create tables for information about students (table 'Student'), schools (table 'School'), departments (table 'Department'), courses (table 'Course') and so on.

A table is made of one or more **columns** (or can be called as **fields** / **attributes**). Each field/column/attribute corresponds to a certain aspect of information used to describe an object. Taking the table of 'Student' for the example, you might need to store the name, gender, ID, date of birth, year of entering the school, the department and school she or he belongs to, and so on. Then for the students, you might have one following table in your database.

Table 1. Example of a Data Table in a Relational Database

First Name	Last Name	ID	Gender	Entering Year	School	Department
Hansen	Ola	198312	Male	1999	CSC	CS
Svendson	Tove	198408	Female	2000	ABE	Physics
Michal	Jackson	196904	Male	1997	ABE	Math.

In order to create a table, you need to provide definitions for all the columns that make up this table. For one particular column, you need to specify its name (there is a certain naming rule for tables and columns in SQL), data type (such as Integer, String/Text, Date/Time, or even floating point numbers), and validation rules for a correct input. For example, the age is always recorded as an Integer, and for human beings, the possible age is within the range of 0 – 150 (this is called validation rule or constraint conditions). If a new input violates the constraints, the database will refuse to let it in and prompt the user an error report about this. A proper data type and constraint setting will help to ensure the quality of your data. A possible specification for the Table 1 might look as the follow. Here we are only talking about the constraints that involving a single column itself. More complex validation rules could be established on the table level involving several columns within the same table or columns from different tables. Such validation rules are also called reference integrity.

Table 2. Definitions for Table 1

Name	Data Type/Length	Constraints	Key	Index
First Name	Varchar(50)	NOT NULL	/	/
Last Name	Varchar(50)	NOT NULL	/	/
ID	Varchar(12)	Unique	Primary Key	B-Tree Index
Gender	Char(1)	'M' or 'F'	/	/
Entering Year	Integer(4)	1980<Year<2050	/	/
School	Varchar(50)	Should be an existing school	/	/
Department	Varchar(50)	Should be an existing dept.	/	/

You might have two fields, one of which could be derived from the other one. For example, the year of birth and age is such a pair of fields. In order to avoid data redundancy and waste of space, you are always required to choose one from them. Which one to be chosen depends on which one is really needed by the customer or which one is most needed according the application scenario. Furthermore, when one field is obligatory and another field (optional) could be derived from the obligatory one, should we keep the optional one or just compute it on the air when it is needed? In fact, it is also another kind of data redundancy. Data redundancy will bring extra space of storage and special effort to maintain the consistency among fields. You can't have students with a birthday of 19800710 but an age of 18. Such inconsistency among data needs to be cleared out of your database. But computing the needed information on the air could have its own problem. It might cost more time to prepare the needed information than just simply getting them from the database; and this is vitally important for those real-time applications or those which are quite critical of response time. The key problem here is that you have to find a balance between the time and space. If an optional field is queried frequently but troublesome to compute it from existing obligatory fields (e.g. compute the date of birth from the person number), just compute it once and then save it as an optional field of your table. In most case, these optional fields seldom change after they are computed and saved, but will bring you improvement of system performance. In contrary, if an optional field is seldom used and could be easily derived from existing obligatory fields, you can just skip it to release yourself from consequent work of maintaining the consistency among data fields. For example, suppose the field of 'YearOfBirth' is obligatory and it is used more frequently than the field 'Age', you can keep the field of 'YearOfBirth' and compute the age of students when it is necessary.

A row of data in the table (e.g. Table 1), which is also called a **record**, refers to a complete set of information for a concrete object in the real world. For example, each row of the 'Students' table represents all the information you need for a student in the system. In most cases, we need a unique ID or serial number for each record in the data table in order to easily and quickly identify the object you refer to in a transaction. Then you can set a single column or a collection of several columns, whose value or combination of values in different record is unique throughout the table, as the **unique key** (or called **primary key**) of this data table. For example, the 'ID' column is set as the primary key of table 'Students'. Through a valid primary key, you can uniquely refer to a single record in the data table. There are also other kinds of keys in relational database, such as the Foreign Key. You will learn it in the coming lectures and labs.

Index is an extra data structure beyond data table in relational (also in other kinds) database systems to speed up the query processing. For example, it is easier and faster to look for a number within an ordered sequence than a disordered sequence. The ordered sequence could be viewed as some kind of index in the database. Without index, one has to sequentially scan all the data records until she or he find the student whose ID is '19830710-1716'. This process is definitely slower than the retrieval with index, and this case is especially true when the number of records is large but you only want to locate one or two of them. In modern relational database systems, there are many kinds of built-in indexes ready for use upon various data types, and the indexes could be established on a single column (then it will spend up your query using this column) or a bundle of column within the same table. For example, you can build a B-Tree index on the column 'Age', and a Hash index on the two columns of 'FirstName' and 'LastName'. You can even create a spatial index on the column of your table which hosts the location information of objects that interest you.

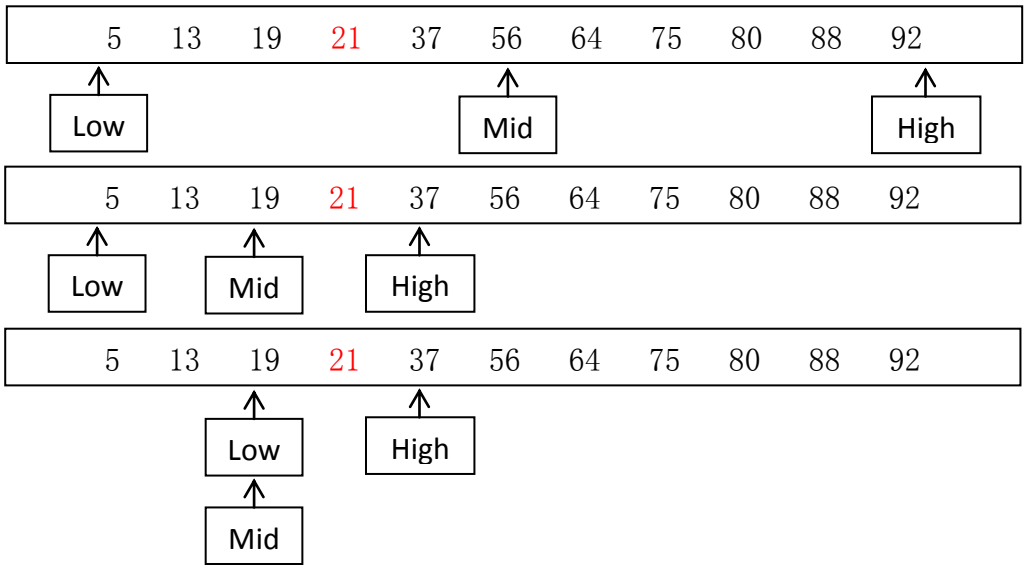


Figure 1. Retrieval within an Ordered Sequence of Numbers

However, index will cost extra space of storage and extra efforts for the database to maintain it. It will work the best when there is no or seldom change to the data table after the records are created from outside. That whether and when to create an index upon your data table is determined by many factors. But in most cases, an index on the most frequent queried columns is always helpful to speed up your system. You will learn more about the indexing techniques and how to use it in database systems in the coming lectures and labs. You can also search for 'index database' or 'B-Tree database' in popular search engine and Wikipedia to explore more information yourself.

To work with a real database systems you might also encounter with other aspects of interacting with a database system, such as the users and privileges, access interface, performance tuning, backup/crash recovery, and so on. In addition to this lab, there will be some other lectures and labs of this course talking about these topics.

It is necessary to clarify the access interface and SQL language in order to figure what SQL can do and what it cannot do. The access interfaces are the approaches client computers used to remotely connect to the database system through network, but SQL is the language the users or developers used to communicate with databases. The access interfaces will transfer the SQL command from users to the database and then bring back the results that database prepare for each request. You can use a client application (could be a desktop application or web page) or API (application programming interface) to connect the database. Taking the open source database system PostgreSQL for the example, you have 'psql' and 'pgAdmin' desktop clients and various kinds of API (ODBC, OLE/DB Provider, JDBC, .NET, PHP, python, C/C++ API, etc.). Oracle database systems used to have a desktop client application and now after 10g it is replaced by a web based application. There are also various kinds of API for Oracle database systems. You will find similar information about MS SQL, Access, MySQL, DB2 and so on. (Please do explore more over the internet). You figure out your request using SQL language and submit it to database through this access interface, and then just wait for the database to response. Finally, you will also need the help of access interface to interpret the results from database into the form you or the computer could understand. So the relationship between database systems, access interface and SQL could be illustrated by the following figure (Figure 2).

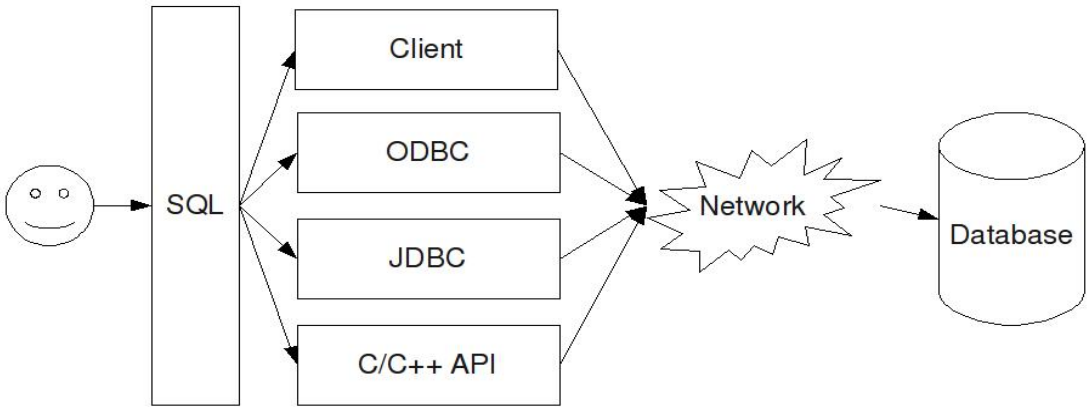


Figure 2. SQL, Access Interfaces and Database Systems

2.3. Fundamental Operations of SQL

SQL is the language you can use to express your request on the database systems. You could do various kinds of tasks on database systems using SQL, including the data creation, modification (insert/delete/update), information retrieval and database maintaining jobs. When using SQL, one only needs to express what she/he wants rather than how to guide the database to do it. For example, you might write a SQL command (or called a SQL statement) to find the highest student in the university, but you don't need to tell the database system to go through each record of the student table, pick up the height column and then compare the height of one student with that of another to find the largest height, and finally return the information of highest student to the client.

In this lab, we only provide some very simple example of SQL to give you a brief idea of what SQL looks like and how it works. You will receive a thorough training on the fundamental and advanced level of SQL programming in the future lectures and labs. Generally speaking, the SQL statements could be classified as three groups.

(1) The Data Manipulation Language (DML)

As the name tells, the DML is used to manipulate the data within database. One can send out data retrieval request using **`SELECT`** command, or modify the existing data stored in tables. The Data Manipulation Language (DML) part of SQL mainly includes:

- **`SELECT`** - extracts data from a database table
- **`UPDATE`** - updates data in a database table
- **`DELETE`** - deletes data from a database table
- **`INSERT INTO`** - inserts new data into a database table

(2) The Data Definition Language (DDL)

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. You can also define indexes (keys), specify links between tables, and impose constraints between database tables. The most important DDL statements in SQL are:

- **`CREATE TABLE`** - creates a new database table
- **`ALTER TABLE`** - alters (changes) a database table
- **`DROP TABLE`** - deletes a database table
- **`CREATE INDEX`** - creates an index (search key)
- **`DROP INDEX`** - deletes an index

(3) The Data Control Language (DCL)

DCL is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

- **`CREATE ROLE/USER`** – create a user or role
- **`GRANT DELETE/UPDATE/CREATE`** – grant certain privilege to users or roles
- **`REVOKE DELETE/UPDATE/CREATE`** – revoke certain privilege from users or roles

3. Interaction with the DBMS and the Database

3.1. Login to the Database

pgAdmin is the most popular and feature rich Open Source administration and development platform for PostgreSQL, the most advanced Open Source database in the world (reference official website of pgAdmin: http://www.pgadmin.org/). We are going to use pgAdmin as a PostgreSQL administration and management tool in this course. Every group has your own database, username and password to login to the database. Open pgAdmin, click the plug and input your initial database and username, as it is shown below in Table 3 and Figure 3.

Table 3. Login Information for Each Group

Group	Database	Username	Password
1	g1_sdb	g1_user	g1_pswd
2	g2_sdb	g2_user	g2_pswd
3	g3_sdb	g3_user	g3_pswd

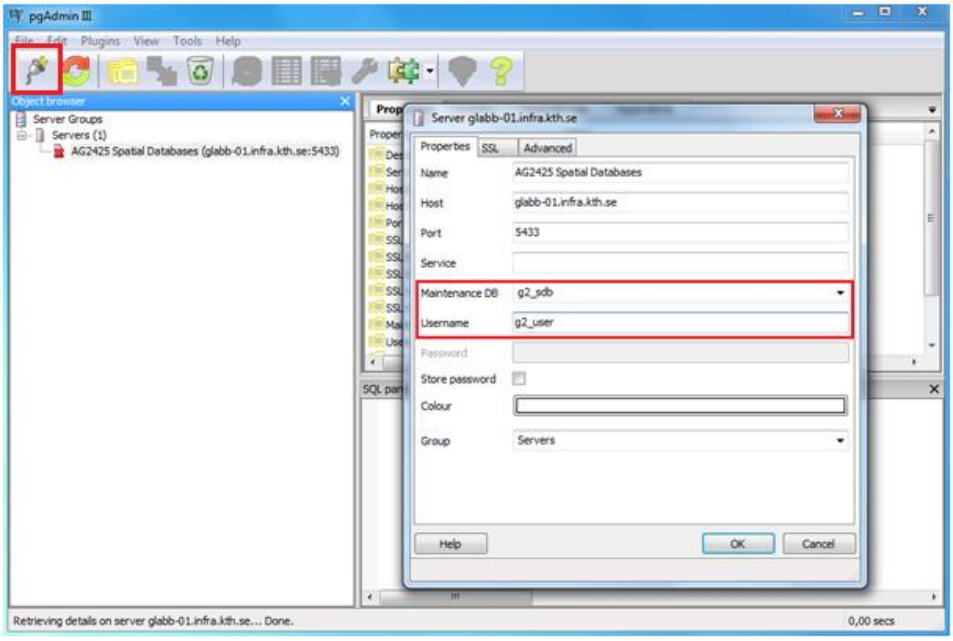


Figure 3. Login to the pgAdmin

Then, double click **AG2425 Spatial Databases** with the Red Cross and enter your password (Figure 4). Now you are in your own spatial database. Only your own database and *postgres* database is accessible. However, please ensure that you are always staying in your own database.

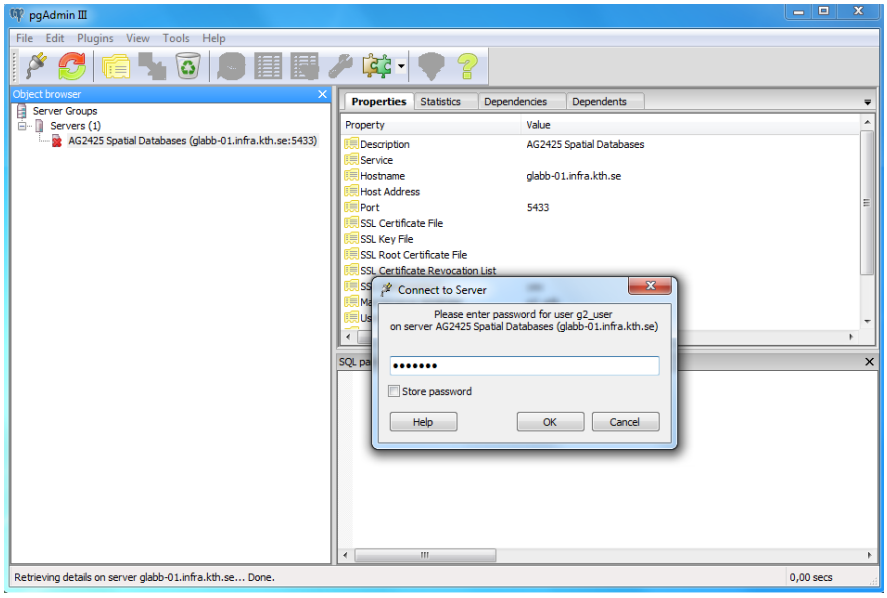


Figure 4. Enter Your Password

Now it is wise for you to change your password. Highlight your database, click the SQL button (used for executing arbitrary SQL queries), type the following SQL statement and click **SQL** (or F5) to execute (Figure 5).

```
ALTER USER YOUR_USER_NAME WITH PASSWORD 'YOUR_OWN_PASSWORD';
```

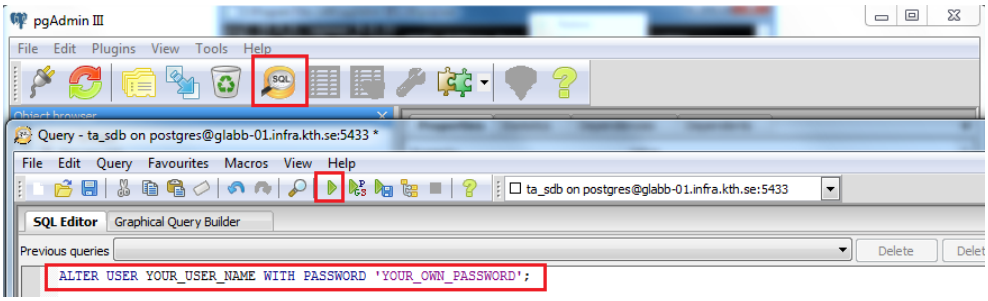


Figure 5. Execute SQL Query to Change Your Password

3.2. Import and Export Textual Data to and from the Database

There are two plugins for pgAdmin that come with the latest version of 1.16 pre-packaged (Figure 6): **PSQL Console (psql)** is a terminal-based interactive front-end to PostgreSQL that allows users to perform various database administration and management tasks from the command line; **PostGIS Shapefile and DBF loader2.0** is a tool to import/export shapefile into the database. Here you will use **PSQL Console (psql)** to import and export textual data from the database. Click the button for the **PSQL Console** plugin.

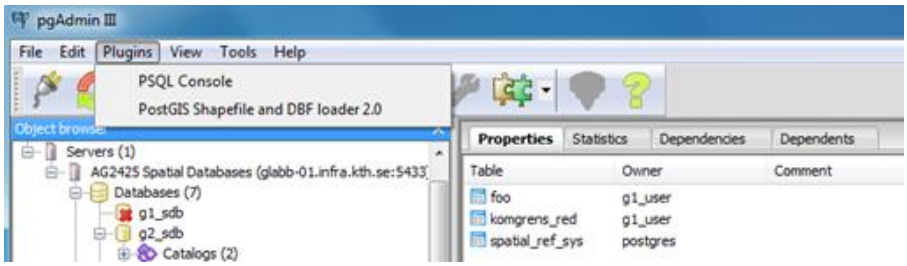
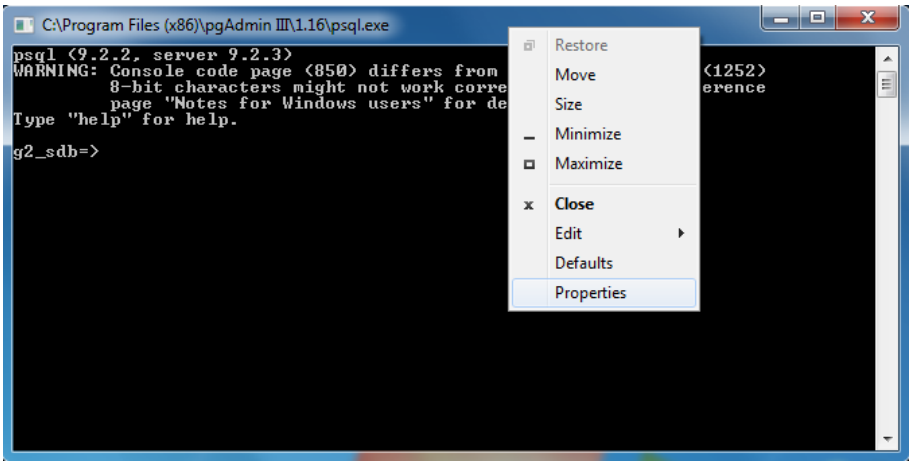


Figure 6. Plugins for pgAdmin

Put your mouse at the top boundary of the *psql.exe*, right click your mouse, and click **Properties** (Figure 7), check the **QuickEdit Mode**. QuickEdit Mode allows you to copy **and** paste by right clicking (there is no message-box shown even when it is successfully copied and pasted).



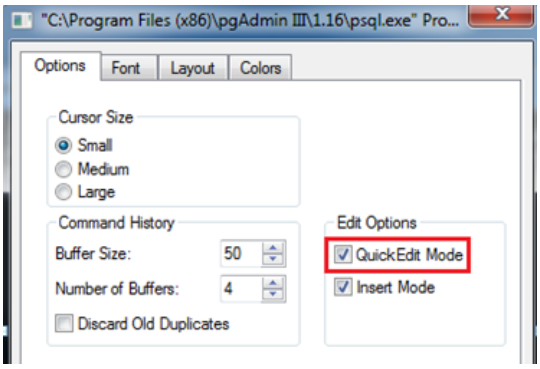


Figure 7. Check QuickEdit Mode for Quick Editing

Now follow the next three steps to import a sample file *foo1.txt* into your own database, as it is shown in Figure 8.

Step 1. Create an empty table *foo1* with two columns *i* and *j*.

```
CREATE TABLE foo1(i INT, j INT);
```

Step 2. Create a *.txt file with the name of *foo1* and input some sample values with as delimiters. Be sure that there are no empty lines.

Step 3. Copy the file from the specific directory that *foo1.txt* located to the DB on the server using psql console.

```
\COPY foo1 FROM 'YOUR_OWN_DIRECTORY' DELIMITERS ',' CSV;
```

(e.g. `YOUR_OWN_DIRECTORY` such as `C:/Temp/foovalues.txt`)

Then, you will see the *foo1.txt* has already successfully been imported into your own DB.

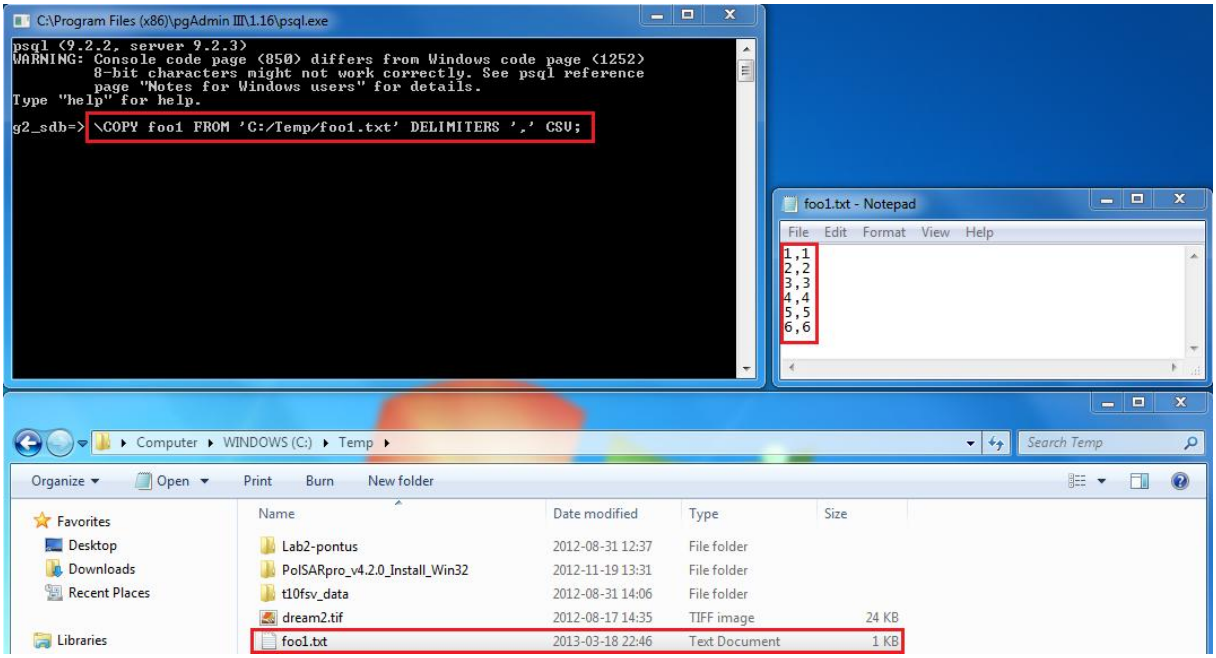


Figure 8. Import an *.txt File into Your Own DB

It is also available that copy a file from the DB to a specific folder with the following SQL statement, as it is shown in Figure 9.

```
\COPY foo TO 'C:/Temp/foo.csv' DELIMITER ',' CSV HEADER;
```

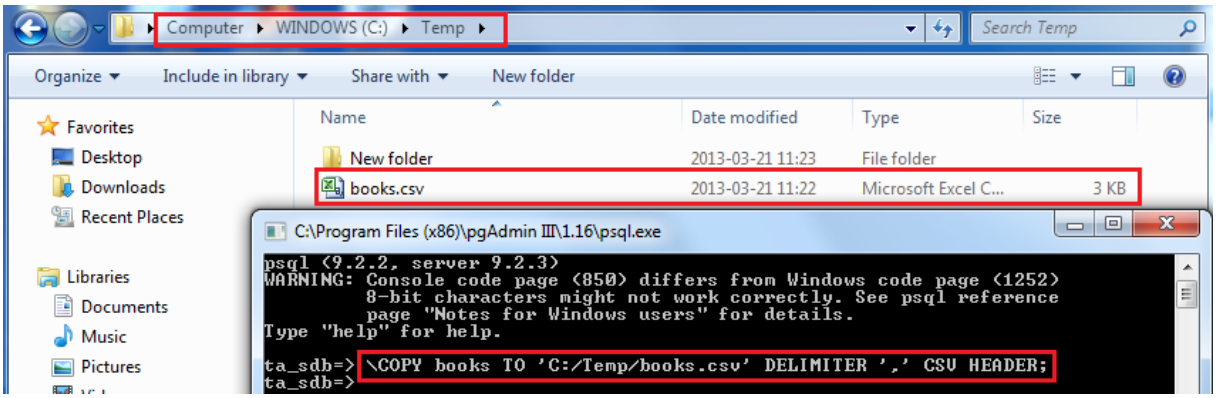


Figure 9. Export an *.csv File into Your Own DB

3.3 Import and Export a Shapefile to and from the Database

In pgAdmin III, click the button for the **PostGIS Shapefile and DBF loader2.0** plugin, click **Add File**, select three shapefiles (data is provided with the tutorial of Lab1) and click **Open** (Figure 10). Before import, be sure to specify *DBF file character encoding* as **LATIN1** in the **Options…**. It is an opposite operation to export a table (with spatial information, e.g. with geometry columns) from DB to a specific folder (Figure 11).

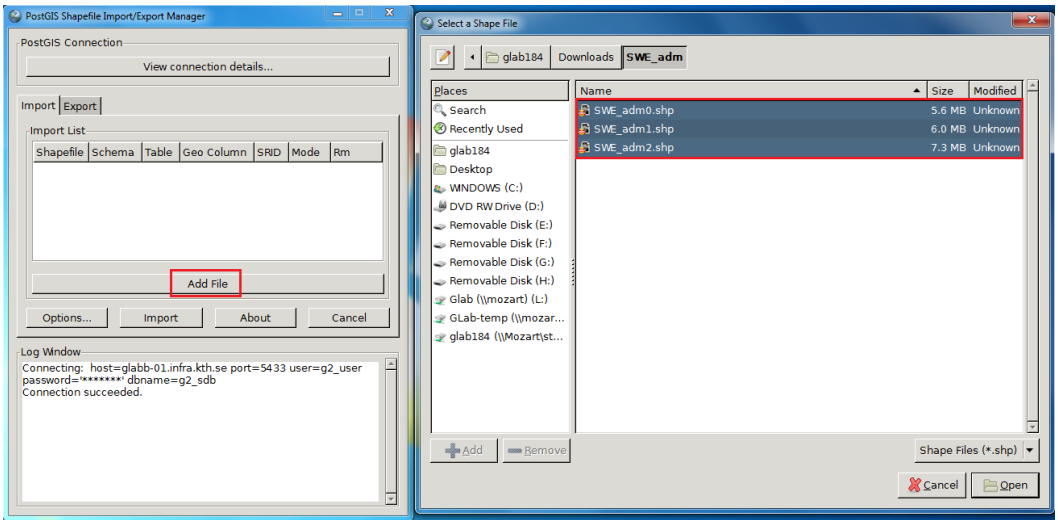


Figure 10. Import Shapfiles into the DB

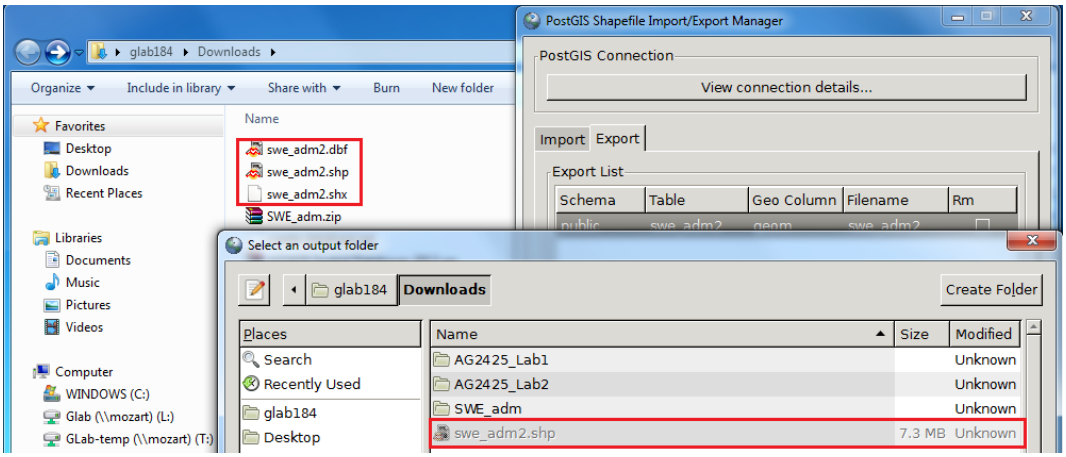


Figure 11. Export a Table (with Shapefile Characters, e.g. Geometry Column) from DB to a Specific Folder

3.4. Connect Quantum GIS to the Database

There is a mode in Quantum GIS (http://www.qgis.org/) that allows you to connect Quantum GIS to a PostGIS database, build queries, and show geometric features read from the PostGIS database. Firstly, open Quantum GIS, click the connection mode, click **Ny** and input required parameters. Click **OK** when test connection to the database is successful (Figure 12).

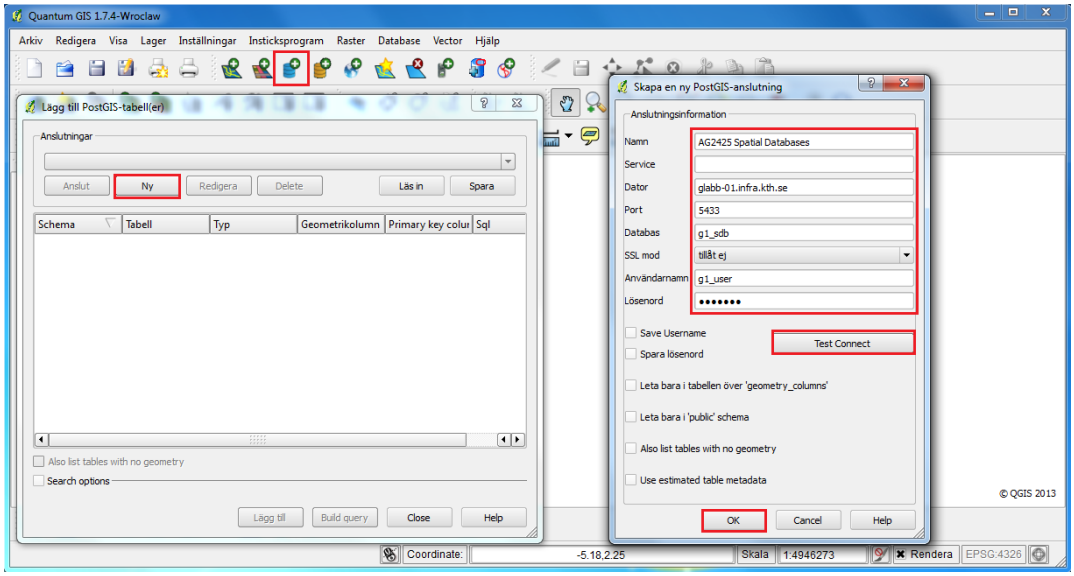


Figure 12. Connect Quantum GIS to PostGIS

Then, click **Anslut** (possibly re-enter username and password once more), select geometric features in your DB (it is Swedish administration regions in this particular situation.) that you are willing to display and click *Lägg till* (Figure 13). You will see maps of Sweden.

OBS: There exists an ArcGIS plugin called **ST-Links** SpatialKit that allows ArcGIS to connect to PostGIS and other OGC Simple Feature Access compliant spatial databases. A trial version that works only for a limited number of features per layer / theme can be downloaded here. Students are encouraged to investigate ST-Links SpatialKit: http://www.st-links.com/Pages/default.aspx

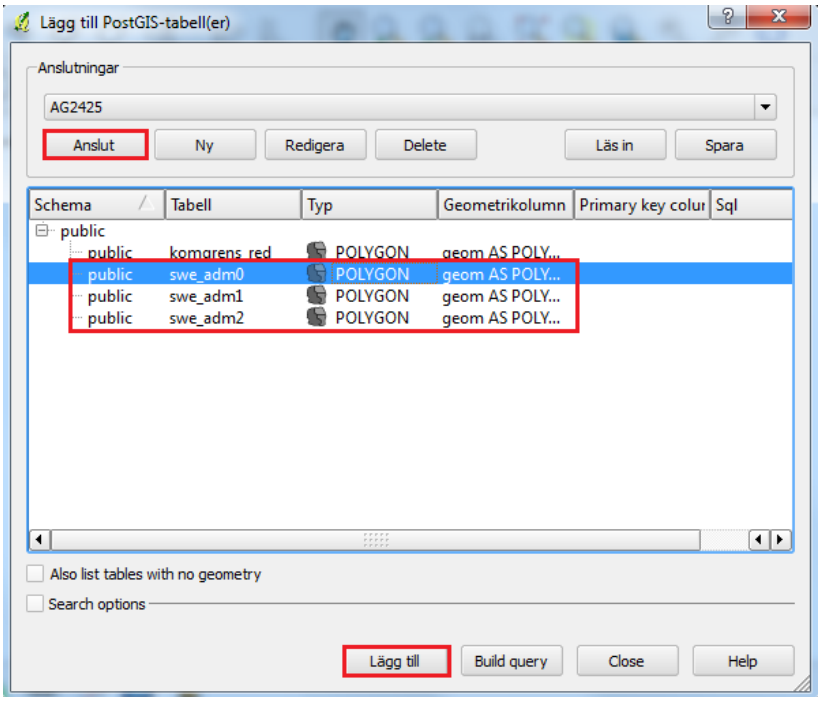


Figure 13 Select Geometric Features to Display

3.5. Create Tables and Insert Values

The following example illustrates how to create and query on database using SQL statements. Taking the school information system for instance, we will use SQL command to create the 'Students' table according to its definition in Table 2. (Note: All the SQL commands have been tested on PostgreSQL server, but they might be slightly syntactically incorrect in another DBMSes that use different SQL dialects.)

- **SQL command to create table (DDL)**

```
CREATE TABLE Students(
   firstname VARCHAR(50) NOT NULL,
   lastname VARCHAR(50) NOT NULL,
   id VARCHAR(12) PRIMARY KEY,
   gender CHAR(1) NOT NULL,
   entering_year INTEGER CHECK(
      enteringyear > 1980 AND enteringyear <2050),
   school VARCHAR(50)REFERNCE schools(Name),
   department VARCHAR(50) REFERENCEN departments (Name)
);
```

--This SQL creates the Students table according its definition in Table 2.

- **Insert new records into the above table (DML)**

```
INSERT INTO Students VALUES(
   'Johan', 'Nelson', '198403121203', 'M', 2000,
   'Computer Science', 'Software Engineering');
```

--This SQL intends to insert the information of Johan Nelson into the Student table.

- **Update the data table (DML)**

```
UPDATE Students
SET LastName='Nilson'
WHERE id='198403121203';
```

--This SQL is going to change the lastname of Johan Nelson to 'Nilson'.

- **Data retrieval in the data table (DML)**

```
SELECT firstname, lastname
FROM Students
WHERE enteringyear = 2000;
```

--*This SQL is a query to find out the students' name that entered the school at the year 2000.*

As it is shown in the above examples, SQL commands are very easy to understand since they are quite close to the human language. It is also quite powerful which could ask the database to do a complex task through a simple command.

3.6. Query Tasks

Go the http://www.postgisonline.org/. This is a user-friendly website that allows users to exercise their newly learned SQL skills on a number of predefined tables. When you are at the front page, you can find a green bar at the top. Go to **Tutorials** on the bar. After you click on it, you can see there are two sections. Go through all links under the section **Plain SQL without anything spatial** and follow the instructions in each of the link so that you can familiarize yourself of different SQL commands. Then:

1) Write an SQL to find out which house (number) Kalle Andersson belongs to.
2) Write an SQL to specify the name and age of the 5 oldest people in descending order.
3) Try the query "`SELECT DISTINCT familyname FROM people;`" and describe your findings.
4) Suppose each family has the same family name in the table. Write your SQL commands to list the average age of each family in descending order.
5) Write an SQL query to find out the people's name and their age in houses where the average age of the residents is 65 or above.

3.7. Answer Questions

1) Describe the process of submitting a SQL at the client side till the interpreting the result from database. Note: the access interface, SQL command, and database systems should be involved in your description.
2) Explain what SQL is in short using your own words.
3) What is 'Primary Key' in relational databases?
4) Describe the difference between SQL language and other programming language you know (e. g. C/C++/Java)

4. Report

A report is not required for this lab. However, this does not mean that this lab is not important; essential knowledge will build a stable foundation for the coming labs with many challenges.

OBS: Useful Online References
PostgreSQL Documentation 9.2: http://www.postgresql.org/docs/9.2/static/sql.html (Last Accessed: March 18, 2013)

Tizag SQL Tutotial: http://www.tizag.com/sqlTutorial/ (Last Accessed: March 18, 2013)

SQL Tutorial: http://www.sql-tutorial.net/ (Last Accessed: March 18, 2013)

Lab 2: Introduction to Relational Database and Modeling

Due April 10th, 2013

1. Task

In this lab, we are going to create a relational database for a Book Shop. Based on the given information of seven entities, you have to draw an Entity-Relationship model. Then, create seven tables with auto-increment constraints (e.g. primary keys, foreign keys, etc.) in the DB based on the provided seven *.txt data sets and import the files into your own DB. Lastly, finish optional query tasks.

2. Entities Description

BOOK

ISBN	TITLE	IN_STOCK	PRICE	AUTHOR_ID	TYPE_ID	SUPPLIER_ID

AUTHOR

ID	NAME	DATE_OF_BIRTH

TYPE_OF_BOOK

ID	NAME

INVOICE

INVOICE_NR	INVOICE_DATE	PAYMENT_DATE	AMOUNT	CUSTOMER_ID

CUSTOMER

ID	NAME	ADDRESS	CONTACT

ITEM_OF_INVOICE

INVOICE_NR	ITEM_ID	QUANTITY	ISBN

SUPPLIER

ID	NAME	ADDRESS	CONTACT

Columns for each of the seven *.files are as follows.

1. TYPES_OF_BOOKS(ID, NAME)
   [TYPES_OF_BOOKS: TXT]

2. AUTHORS(ID, NAME)
   [AUTHORS: TXT]

3. BOOKS(ISBN, TITLE, IN_STOCK, PRICE, AUTHOR_ID, TYPE_ID, SUPPLIER_ID)

```
   [BOOKS: TXT]

4. CUSTOMERS(ID, NAME, ADDRESS, CONTACT)
   [CUSTOMERS: TXT]

5. INVOICES(INVOICE_NR, INVOICE_DATE, PAYMENT_DATE, AMOUNT,
   CUSTOMER_ID)
   [INVOICES: TXT]

6. ITEMS_OF_INVOICES(INVOICE_NR, ITEM_ID, QUANTITY, ISBN)
   [ITEMS_OF_INVOICES: TXT]

7. SUPPLIERS(ID, NAME)
   [SUPPLIERS: TXT]
```

3. Optional Tasks

1. List `name` and `address` of customers whose name begins with *B*.
   Hit: substring function is needed.

2. List `name` of customers who bought more than 10 books.
   Hit: query result of one table can be a precondition for the other query. There tables should be considered at the same time.

4. Report

Submit a professional report in format of .pdf with name of your team member(s) before the deadline to the Bilda System. The report should include an E-R model, necessary screenshots and explanations, SQL statements for the DB creation and queries and selection results.

Lab 3: SQL and Data Indexing Techniques in Relational Database

Due April 17th, 2013

1. Task

This lab is a continuous exercise of lab 2. For this lab, we are going to create views and functions with indexing techniques after reading a short but valuable material.

2. Indexing Techniques

2.1. Indexes

When accessing a table, POSTGRESQL normally reads from the beginning of the table to the end, looking for relevant rows. With an index, it can quickly find specific values in the index, and then go directly to matching rows. In this way, indexes allow fast retrieval of specific rows from a table.

For example, consider the query `SELECT * FROM customer WHERE col = 43`. Without an index, POSTGRESQL must scan the entire table looking for rows where *col* equals 43. With an index on *col*, POSTGRESQL can go directly to rows where *col* equals 43, bypassing all other rows.

For a large table, it can take minutes to check every row. Using an index, finding a specific row takes fractions of a second.

Internally, POSTGRESQL stores data in operating system files. Each table has its own file, and data rows are stored one after another in the file. An index is a separate file that is sorted by one or more columns. It contains pointers into the table file, allowing rapid access to specific values in the table.

POSTGRESQL does not create indexes automatically. Instead, users should create them for columns frequently used in `WHERE` clauses.

To create an index, use the `CREATE INDEX` command, such as `CREATE INDEX customer_custid_idx ON customer (customer_id)`. In this example, *customer_custid_idx* is the name of the index, *customer* is the table being indexed, and *customer_id* is the column being indexed. Although you can use any name for the index, it is good practice to use the table and column names as part of the index name – for example, *customer_customer_id_idx* or *i_customer_custid*. This index is useful only for finding rows in *customer* for *specific customer_ids*. It cannot help when you are accessing other columns, because indexes are sorted by a specific column.

You can create as many indexes as you wish. Of course, an index on a seldom-used column is a waste of disk space. Also, performance can suffer if too many indexes exist, because row changes require an update to each index.

It is possible to create an index spanning multiple columns. Multicolumn indexes are sorted by the first indexed column. When the first column contains several equal values, sorting continues using the second indexed column. Multicolumn indexes are useful only on columns with many duplicate values.

The command `CREATE INDEX customer_age_gender_idx ON customer (age, gender)` creates an index that is sorted by *age* and, when several *age* rows have the same value, then sorted on *gender*. This index can be used by the query `SELECT * FROM customer WHERE age = 36 AND gender = 'F'` and the query `SELECT * FROM customer WHERE age = 36`.

The index *customer_age_gender_idx* is useless if you wish to find rows based only on gender, however. The *gender* component of the index can be used only after the *age* value has been specified. Thus, the query `SELECT * FROM customer WHERE gender = 'F'` cannot use the index because it does not place a restriction on *age*, which is the first part of the index.

Indexes can be useful for columns involved in joins, too. They can even be employed to speed up some `ORDER BY` clauses. To remove an index, use the `DROP INDEX` command.

2.2. Unique Indexes

Unique indexes resemble ordinary indexes, except that they prevent duplicate values from occurring in the table. Example below shows the creation of one table and a unique index. The index is unique because of the keyword `UNIQUE`. The remaining queries try to insert a duplicate value, but the unique index prevents this and displays an appropriate error message.

Sometimes unique indexes are created only to prevent duplicate values, not for performance reasons. Multicolumn unique indexes ensure that the combination of indexed columns remains unique. Unique indexes do allow multiple `NULL` values, however. Unique indexes both speed data access and prevent duplicates.

```
test=> CREATE TABLE duptest (channel INTEGER);
CREATE
test=> CREATE UNIQUE INDEX duptest_channel_idx ON duptest
(channel);
CREATE
test=> INSERT INTO duptest VALUES (1);
INSERT 130220 1
test=> INSERT INTO duptest VALUES (1);
ERROR: Cannot insert a duplicate key into unique index
duptest_channel_idx
```

3. Create Views

NO.	View
View 1	Create a view that displays what has been sold ordered by customer. Each customer should be represented with as many rows as he/she has bought book titles. Observe that if more than one purchase has been made of the same book title only one summarised row should be displayed.
View 2	Create a view that displays the titles that are not in stock, ordered by supplier.
View 3	Create a view that displays how many books have been sold of each category (i.e. `ISBN`). Tip: look at Left/Right Join and Decode.
View 4	Create a view with the columns ID and NAME from the table CUSTOMER and a column COUNT which displays the number of invoices they have received. Customers who have not been invoiced should also be included in the COUNT. Tip: look at Left/Right Join and Decode
View 5	Create a view that displays MIN, MAX, AVG and SUM for the column PRICE in the table BOOK grouped by category (type). The average price should be rounded into two decimals.

4. Create Functions

NO.	Function
Fun. 1	Create a function with the input parameter INVOICE_NR that returns the total amount for the corresponding invoice.
Fun. 2	Create a corresponding procedure with the input parameter INVOICE_NR and with the output parameter TOTAL_SUM containing the total amount of the invoice. For a given invoice number the procedure is to return the total amount of the invoice.
Fun. 3	Create a function with the input parameter CUSTOMER_ID, and that returns the total amount for all of the invoices for the specified customer.
Fun. 4	Create a corresponding procedure with the input parameter CUSTOMER_ID, and with the output parameter TOTAL_SUM containing the total amount of the invoices of the specified customer.

5. Report

Submit a report in format of .pdf with the name of your team member(s) before the deadline to the Bilda System. The report should include the SQL statement (please be well formatted and clearly commented), explanations, and necessary screenshots for each task.

Reference

The rest reading material is referenced from the book of *PostgreSQL Introduction and Concepts* written by Bruce Momjian.

Lab 4: Spatial Data Modeling with SDBMS (1)
Conceptual and Logical Modeling

Due April 21st, 2013

1. Task

This lab is a new exercise for spatial data modeling. In this lab, we are going to do conceptual and logical modeling for a website after a brief reading of the introduction part.

2. Introduction

The first step to use spatial database for your data management is to figure out what kind of information (both spatial and non-spatial information) you are going to store and how these data could be organized and related to each other. This process is also called spatial data modeling. Similar to the object-oriented modeling for spatial information, the spatial data modeling with SDBMS will also need to find out the objects or spatial information that are required to fulfill the task of an application. The main difference is that one has to transform the result of object-oriented spatial data modeling (or E-R diagram) into tables, since most spatial databases are implemented on the basis of object-relational database (ORDB) systems. Various terms of relationship between objects (more exactly classes during the modeling process) need to be realized as relationships between tables, as well as integrity constraints which might not be able to be expressed using simple SQL statements. In this case, you need to do some programming to make stored procedures/triggers using the SQL/PL (procedural language) provided by the database systems (e.g. Oracle, PostGIS).

The purpose of this exercise is to offer you the opportunity to carry out a spatial data modeling with SDBMS, given the scenario of a small demo application. You are required to find out all the entities or object classes (conceptual model) needed to build the information system, and convert them into tables (logical/physical model). You are also required to provide the definition of the table structures, the relationships between tables and (spatial and non-spatial) data integrity constraints. Please note that, the SDBMS provided in this exercise have a full support for the OGC Simple Feature Specification for SQL. Hence, you can use the Geometry type in the definition of your data tables, as well as various functions for spatial data types.

NOTE: We omit the difference between logical and physical models here for this exercise, because for standard ORDBMS or RDBMS, there is little difference among various implementation of the same logical data model. For example, if you use the standard SQL language to convert a logical model into physical model, the SQL script should be able to run in both Oracle and PostgreSQL, and finally produce the same result.

You are required to express your conceptual data model using either E-R diagram or UML diagram. You can draw the diagram by hand or by any kind of software you are familiar with. In your report, attach the graphic conceptual model and submit it to your TA. Regarding the definition of your tables as well the relationships between them, you are allowed to do it in natural language for this exercise. But it is strongly recommended that you try to transform your design into tables using SQL statements (**You will run these SQL scripts in the next exercise to create your tables**). Concerning the spatial data integrity constraints, you can describe it in natural language in this exercise first. We will go through with you on how to implement these spatial data integrity constraints in SDBMS using SQL/PL language and triggers/stored procedures in the next exercise.

3. A demo Application of Spatial Data Modeling

A Website for Advertisements and Sales of Houses

Suppose you are required to build a website for sales and advertisements of houses and apartments. Those people who want to sell their houses and apartments could register their information as well as the houses on your website. Then the potential buyers could search for a house they like through the online system. They could specify various kinds of conditions/filters (locational or non-locational) to describe the house of their interest, and then the online system could list all the candidates that meet the buyers' requirements. After choosing the house to purchase, the online system will require buyer's contact information, and at the same time provide them the seller's contact information, in order to be fair and avoid spam requests. Finally, the online system will inform the seller that somebody shows interest on requests. Finally, the online system will inform the seller that somebody shows interest on her/his house as well.

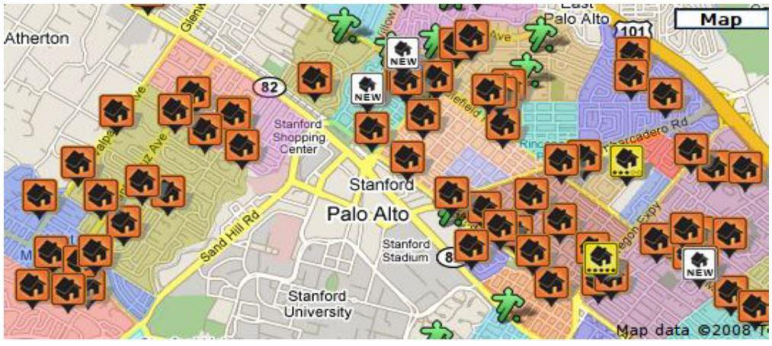


Figure 1. Houses and Parcels

Figure 2. Parcels, Roads and Houses

According to the local law, the land for building houses and apartments are divided and organized as parcels, which are surrounded and connected by roads. Houses and apartments could only be constructed within a certain parcel. It is not allowed to build them in an open area which is located outside all the parcels.

Anybody including the real estate retailers could register the houses and apartments information on this web site. So it is common that one seller could have more than one house or apartment for sale. Sometimes, the seller might need to change his contact information. On this website, she/he just needs to do it once, and then all his houses and apartments will be attached with her/his latest contact information.

In order to find a proper house or apartment, the buyers will need to investigate all the houses and compare their price, location, area, layout, structures, out-looking, age, the parcel where the house is located, and the (traveling) accessibility of each house and apartment. They might even care about the credit record of the seller. In order to better support the decision making for potential buyers, the web site should be able to visualize all the candidates on a digital map. Then buyer could have a better overview on the neighborhood of their house, as well as where it is located in the city. When they click a house in the map, detailed information about this house and seller will be displayed automatically.

Please design a spatial database for this web application with those information and constraints mentioned in the above text. The spatial database should be able to handle all the spatial and non-spatial information in an integrated way. In addition, the database system itself should have some automatic approaches to let out "dirty" data (hint: using integrity constraint rules) and keep the whole system in a consistent status. In order to build such a web site with good performance, your design should take into account the rapid access of spatial and non-spatial information.

4. Modeling Task

Based on the above demo case of spatial data management, the task is to build a spatial database which could meet the requirements to fulfill the objectives of this application. Your design and definition of this spatial database should include the following aspects:

1. The required entities (or object classes) and their responsibility in the database (**Conceptual**)
   Hint: you can use either E-R diagram (E-R data model) or UML diagram (OO data model), depending on which kind of conceptual data model you are using.
2. The definition of data tables, including the table structures and keys (**Logical/Physical**)
   Hint: Manually convert the conceptual model into logical/physical data model.
3. Data integrity constraints (both spatial and non-spatial)
4. SQL commands for creating the tables (optional)

5. Report

Submit a report in format of .pdf with the name of your team member(s) before the deadline

to the Bilda System. The report should include the SQL statement (please be well formatted and clearly commented), explanations, and necessary screenshots for each task.

Reference

1. Geodatabase Design.
   http://www.fargeo.com/geodatabase-design?services/ (Last Accessed: April 17, 2013)
2. Spatial Analysis and Modeling.
   http://www.gisdevelopment.net/technology/gis/techgi0039.htm (Last Accessed: April 17, 2013)
3. Spatial Data Modeling Standards.
   http://www.for.gov.bc.ca/his/datadmin/spatproj.htm (Last Accessed: April 17, 2013)

Lab 5: Spatial Data Modeling with SDBMS(2) Implementation in SDB

Due April 25th, 2013

1. Task

This lab is a continuation exercise of Lab 4 for spatial data modeling. In this lab, we are going to physically implement a spatial database for a website after a brief reading for the introduction and tutorials.

2. Introduction

As the second part of spatial data modeling with SDBMS, this exercise 5 will give you chance to learn how to implement your design into the database. Given the PostGIS/PostgreSQL, which is the fully OGC standard compatible spatial database, the students will create tables and data integrity constraints using SQL commands and SQL/PL programming language, according to their data model design in last exercise. A simple test (provided either by the TA or students themselves) will also be carried out to see whether the constraints rules work well on controlling the quality of spatial and non-spatial data within your database.

NOTE: The implementation of spatial data modeling within SDBMS is more or less the same among different spatial database products, especially for those who are compatible with OGC and ISO standards. The main difference is perhaps the syntax of proprietary SQL, procedural language, and some special functionalities provided by different database products. After this exercise, you should be able to do the task easily within other spatial database products, such as the Oracle 9i/10g Spatial, IBM DB2, MS SQL Server 2008, MySQL 5.1 and etc. To be simplified, we will use the free and open-source spatial database PostGIS/PostgreSQL as the database platform in this exercise.

3. Tutorials

3.1. A Basic Tutorial on PostgreSQL Client: pgAdmin

For database systems like PostgreSQL, clients are standalone desktop or online applications that could be used by DBA or software developers to access the database. Normally, the client applications will provide various functionalities to aid the DBA or developers to examine the status of database, to manage the data and metadata within the database. Together with PostgreSQL, there are two kinds of client applications provided in this purpose, and they are the command based client “psql” and the GUI based client application named “pgAdmin III”. In this part, we will introduce the basic usage of pgAdmin III on how to monitor and manage the data within PostgreSQL database.

3.2. An instruction on “How to create customized data integrity constraint in PostgreSQL”

(1) PostgreSQL Triggers

“A trigger is a specification that the database should automatically execute a particular function whenever a certain type of operation is performed. Triggers can be defined to execute either before or after any `INSERT`, `UPDATE`, or `DELETE` operation, either once per modified row, or once per SQL statement. If a trigger event occurs, the trigger's function is called at the appropriate time to handle the event.

The trigger function must be defined before the trigger itself can be created. The trigger function must be declared as a function taking no arguments and returning type trigger. (The trigger function receives its input through a specially-passed **TriggerData** structure, not in the form of ordinary function arguments.)

Once a suitable trigger function has been created, the trigger is established with `CREATE TRIGGER` statement. The same trigger function can be used for multiple triggers.

PostgreSQL offers both **per-row** triggers and **per-statement** triggers. With a per-row trigger, the trigger function is invoked once for each row that is affected by the statement which fired the trigger. In contrast, a per-statement trigger is invoked only once when an appropriate statement is executed, regardless of the number of rows affected by that statement. In particular, a statement that affects zero rows will still result in the execution of any applicable per-statement triggers. These two types of triggers are sometimes also called **row-level** triggers and **statement-level** triggers, respectively.

Triggers can also be classified as **before** triggers and **after** triggers. Statement-level before triggers naturally fire before the statement starts to do anything, while statement-level after triggers fire at the very end of the statement. Row-level before triggers fire immediately before a particular row is operated on, while row-level after triggers fire at the end of the statement (but before any statement-level after triggers).

(2) PostgreSQL Procedural Language

PostgreSQL allows user-defined functions to be written in other languages besides SQL and C. These other languages are generically called procedural languages (PLs). For a function written in a procedural language, the database server has no built-in knowledge about how to interpret the function's source text. Instead, the task is passed to a special handler that knows the details of the language. The handler could either do all the work of parsing, syntax analysis, execution, etc. itself, or it could serve as "glue" between PostgreSQL and an existing implementation of a programming language. The handler itself is a C language function compiled into a shared object and loaded on demand, just like any other C function. There are currently four procedural languages available in the standard PostgreSQL distribution: PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python.

Since the PL/pgSQL procedural language is based on the SQL, it is relatively much easier to learn for those who have already some knowledge on SQL. As a result, in this exercise we willuse the PL/pgSQL language to create functions, which could then be in the creation of customized data integrity constraints.

(3) Create customized data integrity constraint using triggers and PL/pgSQL

In this part, we will guide you the process of creating a trigger and its handler function using PostgreSQL PL/pgSQL language.

Depending the policy of your data integrity control, you have to choose either a `BEFORE` or `AFTER` trigger on the level of either `ROW` or `STATEMENT`, as well as which kinds of event are of your interest (`INSERT`, `UPDATE` or `DELETE`). For instance, we have a table designed for road information within the online store of House and Apartment application. The road could be designed as below.

```
CREATE TABLE road(
    id SERIAL,
    name TEXT NOT NULL,
    shape GEOMETRY NOT NULL,
    length FLOAT
);
```

For the data table of road information, we only allow geometry of type OGC `LineString` be inserted into the road table. Furthermore, we want the shape of road **without any self-intersection**. Finally, we want to ensure the **length field of road be identical to the geometric length** of this road's geometry. Based on the above analysis, we can create a `BEFORE INSERT/UPDATE` trigger on the `ROW` level.

First, we create a new function to check the type and existence of self-intersection of the road's shape. If none, update the field length with road's geometric length automatically. Please look into PostgreSQL documentation about the PL/pgSQL Basic Statements, Control Structure and Trigger Procedure when writing your code (of course, the PostGIS reference for SQL geometric functions). There is an example as below.

```
CREATE OR REPLACE FUNCTION check_road_interity()
RETURNS trigger AS
    $check_road_interity$ --label of outset section
DECLARE
BEGIN
    --only road table is of interest
    IF TG_TABLE_NAME <> 'road' THEN
        RETURN NEW; --do no change to the new record
    END IF;
    --only update and insert event
    IF TG_OP <> 'UPDATE' AND TG_OP <> 'INSERT' THEN
        RETURN NEW; --do no change to the new record
    END IF;
    --check whether the input the
    DECLARE
        geoType TEXT;
        selfInserted BOOLEAN;
        geoLength NUMERIC(10);
    BEGIN
        SELECT GeometryType(NEW.shape) INTO geoType;
```

```
        SELECT ST_IsSimple(NEW.shape) INTO selfInserted;
        RAISE LOG 'GeoType %s', geoType; --Print Debug Info
        IF geoType <> 'LINESTRING' OR selfInserted <> 't' THEN
           RETURN NULL; --skip the change and return
        ELSE
            SELECT ST_LENGTH2D(NEW.shape) INTO geoLength;
            NEW.length :=geoLength;
            RETURN NEW;
        END IF;
        END;
END;
$check_road_interity$ --label of outset section
LANGUAGE plpgsql;
```

Secondly, a trigger will be created on the data table of 'road' to listen on any INSERT or UPDATE event happened to this table, using the following PL/SQL code.

```
CREATE TRIGGER road_integrity_trigger
BEFORE INSERT OR UPDATE ON road
FOR EACH ROW EXECUTE PROCEDURE check_road_interity();
```

Finally, we use some script to test whether this trigger and function works to ensure our integrity rule.

```
--Correct INSERT
DELETE FROM road;
INSERT INTO road(name,shape,length)
VALUES('E17', 'LineString(10 10, 20 20, 30 30, 50 70)', 200);
SELECT id, name, ST_AsText(shape), length FROM road;
--value of the length is 73, not the input of 200

--Correct INSERT with no input for Length field
DELETE FROM road;
INSERT INTO road(name,shape)
VALUES('E20', 'LineString(10 10,20 20)');
SELECT id, name, ST_AsText(shape), length FROM road;

--Incorrect INSERT of non-LineString type
DELETE FROM road;
INSERT INTO road(name,shape,length)
VALUES('E18', 'Point(10 10)', 200);
SELECT id, name, ST_AsText(shape), length FROM road;

--Incorrect INSERT of self-intersection
DELETE FROM road;
INSERT INTO road(name,shape,length)
VALUES('E19', 'LineString(10 10, 20 20, 20 10, 10 20)', 200);
SELECT id, name, ST_AsText(shape), length FROM road;
```

4. Implementation Task

1. Prepare the SQL to create tables and simple integrity rules, and then execute them in your own database.
2. Prepare the SQL/PL program to create advanced integrity constraint rules, and then execute them in your database.
3. Provide some small examples to test your spatial database by adding some new records, updating existing records, and deleting certain records from your DB. Investigate the behaviors of your spatial database when these updating operations are executed.
4. **Optional:** You might also want to run some other SQL command within PostgreSQL to get used to it.

5. Report

Submit a report in format of .pdf with the name of your team member(s) before the deadline to the Bilda System. The report should include the SQL statement (please be well formatted and clearly commented), necessary screenshots, and explanations.

Reference

1. The GEOMETRY_COLUMNS VIEW. http://postgis.net/docs/manual-2.0/using_postgis_dbmanagement.html#geometry_columns
2. Using OpenGIS Standards. (Please Read Section 4.3.1 – 4.3.4) http://postgis.net/docs/manual-2.0/using_postgis_dbmanagement.html#id358093
3. PostGIS Geography Type. http://postgis.net/docs/manual-2.0/using_postgis_dbmanagement.html#PostGIS_Geography
4. PostgreSQL Documentation: Tutorial on SQL Language. http://www.postgresql.org/docs/9.2/static/sql.html
5. PostgreSQL Documentation: PL/pgSQL Procedural Language. http://www.postgresql.org/docs/9.2/static/plpgsql.html
6. PostgreSQL Documentation: Create trigger functions using PL/pgSQL. http://www.postgresql.org/docs/9.2/static/plpgsql-trigger.html
7. Using pgAdmin III. http://www.pgadmin.org/docs/1.8/using.html

All the links were last accessed on April 17, 2013.

Lab 6: Spatial Queries and Indices

Due May 2nd, 2013

1. Task

The purpose of this exercise is to give you experience on how to carry out spatial queries within spatial databases to get what you want to know from the spatial data, to understand how much a proper configured spatial index could help to improve the performance of spatial queries processing, and to get acknowledge of using the spatial indices to improve the performance of spatial query processing

In this lab, you are required to finish the following task: use SQL language to carry out complex spatial query and spatial data processing upon the spatial information stored in SDB (i.e. PostGIS for this lab exercise).

2. Tutorials

2.1. Spatial Query Language

In this exercise, you will be given the chance to use OGC and ISO standard GSQL language to query, analyze and retrieve spatial data from the remote spatial database. The same data set, which has once been used in the GIS Architecture course, will be used again in this Exercise 7. But more complicate queries and discussions will be prepared and prompted for you to get better understanding on the GSQL query language and the spatial database.

This sample spatial database contains four tables: cities, countries, rivers, continents in this planet, and definition of these four tables are listed as tables in the end of this document. They are stored in PostGIS SDB, which is compatible with the OGC Standard for SQL. Please answer the question using spatial query language (GSQL) upon these three tables. (You can choose to use either the OGC standard query language or other private functions provided only by PostGIS. But the OGC standard GSQL is recommended in this part, because it will make your codes able to work on other SDB products.)

The Danube River is one of the longest rivers in Europe continent and at the same time an important waterway for this land. And the place we are standing now is Sweden.

(1) Basic Input/Output Functions

Q1: How do the geometries of Danube River and the country of Sweden are recorded in the PostGIS spatial database used in this exercise? Please calculate the WKT representation of this two geometries using SQL language.

Q2: Is the Danube river recorded as one whole Line String object in the database?

(2) Spatial Relationship Predicates

Q3: Can you explain the difference between spatial relationship of *Crosses* and *Intersects*? Hints: Check the relation between Danube River and Germany, and the relation between Turkey and Europe.

Q4: How could we know the spatial relationship between two objects like the Danube River and the country of Germany? There is a function in OGC standard called `ST_Relate` and might be applied in this example. Could you explain the output of this function?

(3) Spatial Join in Query Language

Q6: Given the two data tables Country and River, can you find out the countries that the Danube River crosses? You can simply use the OGC function `ST_Intersects`.

Q7: Similarly, you can do another query based on an attribute join. For example, find out all the cities whose country has a population more than 5,000,000. Write down your SQL command and compare it with last question with spatial join (Do the same query using spatial queries). Can you guess which query will be generally processed faster? Explain why.

Write down your SQL commands, results, and explanation on your report.

(4) Spatial Measuring Operators

Using spatial measuring operators, you are able to calculate the geometric properties of spatial objects, such as the length, perimeter, area, volume and so on. Use these operators to answer the following question.

Q8: What is the area of Sweden in km^2? Use the `SELECT` ... `AS` ... clause in this example. Think about at least two ways to answer this question using SQL command. Is your result coincident to the true value? Write down your observation and try to explain why. (Hints: which reference systems and projection are all these tables using? Find out the way on how to make projection transformation in PostGIS documentation.)

Q9: Which are the world's top 10 longest rivers? Use the “limit 10” clause in your query, as well as the spatial measuring function named `ST_Length`. Is there other method that you can use to the Q9 without `ST_Length` function?

Q10: Find out all the rivers of Europe whose length is larger than 1000 km. Use both `ST_Intersects` and `ST_Within` functions in your query. Try to find out the difference between the results and explain why.

Write down your SQL commands, results, and explanation on your report.

(5) Complex Spatial Query mixed with Attribute Condition

You can use spatial functions and operators together with those that could be applied on simple alphanumeric data. Try to answer the following questions by mixing up the spatial and attribute functions and operators.

Q11: Find out the countries (as well as the number of cities) which have more than 5 city with a population more than 1,000,000 (or pop_rank <= 2). Give at least two ways (both spatial join and attribute join) to answer this question. Do they produce the same result? Write down your observation and briefly explain why this happens. Can you write a query to find out the difference for the country of China, who meets the condition of this query and has different number of cities from the two types of join operation?

Q12: Find out the countries which is crossed by the Danube River and has a population larger than 10,000,000.

Q13: Find out the nearest Swedish city to the city of London. Use self-join and `limit N` clause for this question.

(6) Spatial Analysis Operators

Using the spatial query language, you can even carry out spatial analysis upon the data stored in remote SDB, and of course the result will be saved temporally on the SDB until user terminate the session between client and server. But the functions of spatial analysis vary from one (SDB software) to another, since it is not strictly required by the OGC or ISO standard.

Q14: Find out the cities in Sweden which has a distance of more than 1000 KM (=how many degree? you can use 10 degrees) from the city of Stockholm. Remember that the coordinates of city dataset is recorded in latitude/longitude degrees. Please use the `ST_Distance` and `ST_Buffer` functions to find out the answer. Which approach will be faster?

Q15: Find out the countries in Europe which has longest part of Danube River inside its territory.

2.2. Spatial Indices

Indexes are what make using a spatial database for large data sets management and retrieval possible. Without indexing, any search for a feature would require a "sequential scan" of every record in the database. Indexing speeds up searching by organizing the data (or location of data on disk) into a search tree which can be quickly traversed to find a particular record. PostgreSQL supports three kinds of indexes by default: B-Tree indexes, R-Tree indexes, and GiST indexes (the GiST spatial index for highly recommended by PostGIS/PostgreSQL developers).

In this example, we will learn the basic usage of spatial index within spatial indexes (Note: it is similar to use spatial index within other spatial database platform, such as Oracle Spatial, MySQL, etc.)

The spatial datasets used in this part are the topological maps of USA, and they are `us_counties` (counties), `us_high` (highway), `us_states` (states), `us_capitals` (capitals of states), and `us_lakes` (lakes).

(1) Create a Spatial Index within PostgreSQL
The SQL command to create a spatial index in PostgreSQL is illustrated as follow:

```
CREATE INDEX [indexname] ON [tablename] USING GIST
([geometryfield]);
```

After building an index, it is important to force PostgreSQL to collect table statistics, which are used to optimize query plans. You can use the SQL command as follow:

```
VACUUM FULL ANALYZE [tablename];
```

Then it is ready to usethe spatial indices in your queries.

Q16: Please create a GiST index on table `us_counties` on the column `the_geom` using the above two SQL commands.

(2) Use Created Spatial Index in Spatial Queries
In most case, you don't need to be aware of the existence of spatial indices while carrying outspatial queries. The query planner of PostgreSQL will find the fast way (it think) by looking into various statistics of tables and indices. You can now just write a normal SQL with spatial query operators and then PostGIS/PostgreSQL will find the best solution for your query.

Q17: Carry out a spatial query on the table of `us_counties` and `us_high` to find out all the counties that are crossed by the “Interstate 20” highway.

(3) Investigate the Advantage of Spatial Indices
In this exercise, we will compare the spatial query processing time of two data table, one with a GiST R-Tree spatial index and another without any spatial index.

Q18: Please carry out the same query as the previous task (Q17), but replace `us_counties` with `us_counties_noindex` which contains exactly the same data as `us_counties` but without any spatial index. Compare the query processing time and write down your observation. Is the query processing on table with spatial index always faster than that of un-indexed tables?

Write down your answer and explain why.

Note: For PostgreSQL, you should use the following SQL command for this question:

```
SELECT a.Name as NameOfCounty
FROM us_counties_noindex a, us_highway b
WHERE ST_Intersects(a.the_geom, b.the_geom)
AND b.route like '%Interstate%20';
```

5. Report

Submit a report in format of .pdf with the name of your team member(s) before the deadline to the Bilda System. The report should include the SQL statement (please be well formatted and clearly commented), necessary screenshots, and explanations.

Reference

8. PostgreSQL: SQL Performance Tips.
   http://www.postgresql.org/docs/9.2/static/performance-tips.html
9. PostgreSQL: Indices Types.
   http://www.postgresql.org/docs/9.2/static/indexes.html
10. Spatial Index in PostGIS.
    http://postgis.refractions.net/documentation/manual-1.4/ch04.html#id2761109
11. PostGIS Performance Tips.
    http://postgis.refractions.net/documentation/manual-1.4/ch06.html

All the links were last accessed on April 24, 2013.

Appendix – Tables

City

Column	Type
gid	integer
city_name	character varying (30)
admin_name	character varying (42)
fips_cnty	character varying (2)
cntry_name	character varying (30)
status	character varying (30)
pop_rank	smallint
pop_class	character varying (22)
port_id	smallint
the_geom	geometry

Country

Column	Type
fips_cntry	integer
iso_2digit	character varying (2)
iso_3digit	character varying (2)
iso_num	character varying (3)
cntry_name	smaillint
long_name	character varying (40)
isoshrtnam	character varying (45)
unshrtnam	character varying (55)
locshrtnam	character varying (43)
loclngnam	character varying (74)
status	character varying (60)
pop_cntry	integer
sqkm	double precision
sqmi	double precision
colormap	smailint
the_geom	geometry

River

Column	Type
gid	integer
name	character varying (25)
system	character varying (16)
miles	double precision
kilometers	double precision
the_geom	geometry

Continent

Column	Type
gid	integer
continent	character varying (13)
sqmi	double precision
sqkm	double precision
the_geom	geometry

Help File to Install the Latest Versions of pgAdmin and Shapefile Import/Export Plugin

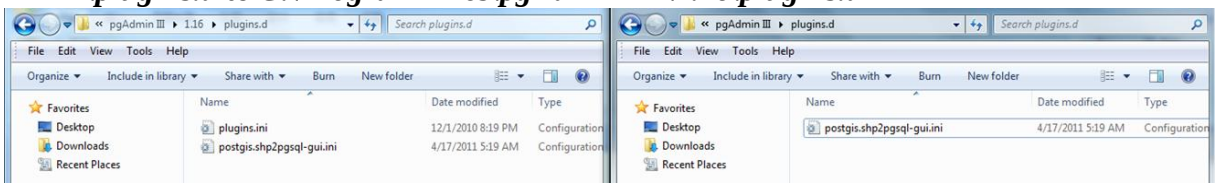
1. Download shapefile import/export plug-in
   http://www.postgis.org/download/windows/pg92/

2. Download the latest version of pgAdmin
   http://www.pgadmin.org/download/windows.php

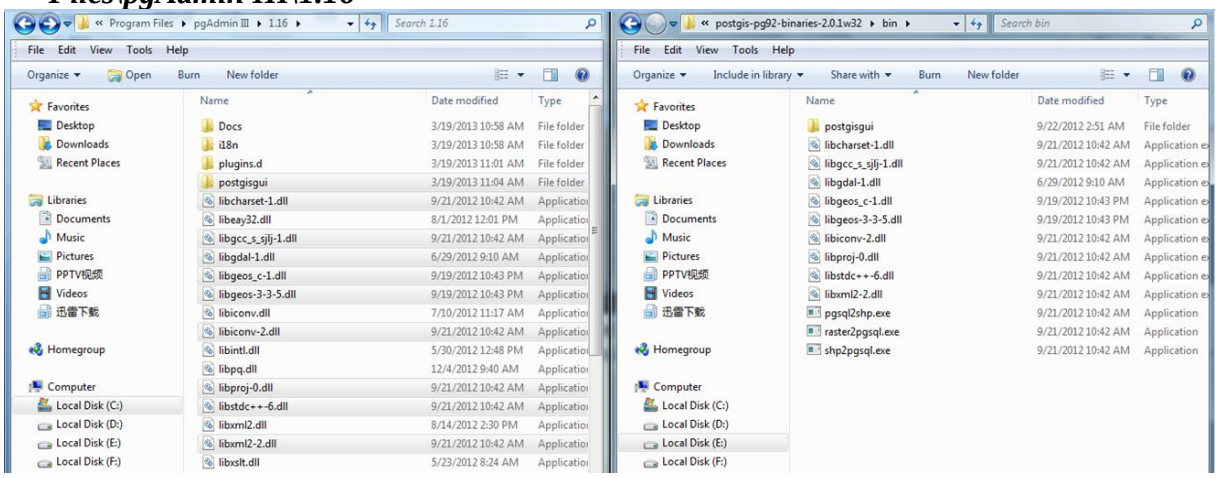
3. Install the latest version of pgAdmin.

4. Unzip the postgis-pg92-binaries-2.0.1w32.zip

5. Copy ***postgis.shp2pgsql-gui.ini*** from ***…\postgis-pg92-binaries-2.0.1w32\pgAdmin III\plugins.d*** to ***C:\Program Files\pgAdmin III\1.16\plugins.d***



6. Copy all the files under ***…\postgis-pg92-binaries-2.0.1w32\bin*** to ***C:\Program Files\pgAdmin III\1.16***



7. Be ended.