ПРОТОКОЛ

От работната среща на втора, трета и четвърта работна група по проект ДВУ 01/197 „Биоинформатични изследвания върху структурата и активността на протеините”, проведена в ИМИ – БАН на 10.04.2009 г.

Участваха проф. Янев, доц. Миланов, доц. Пенчева, гл. ас. Иван Тренчев и гл. ас. Живко Велков.

Работната среща беше открита от зам. ръководителя на проекта доц. Миланов. Той изнесе кратко експозе за целта на работната среща: съгласуване на работата на работните групите и определяне текущите задачи. Проф. Янев представи накратко работата на втора работни група и проблематиката, по която работи.

Съществен част от времето бе посветена на проблема „Molecular docking problem”, състоящ се в създаване на метод/алгоритъм за оценяване на афинитета на кандидат-молекула за свързване (binding) с цел-протеин, включващ: профил-профил подреждане (alignment) и число (score), което е функция на подреждането. За profile-profile alignment се предвижда графово описание на подрежданите профили, на основата на 3D структурата на протеина и на молекулата. Подходящите фрагменти (binding sites) се търсят като клики в граф с тегла на върховете, определени с помощта на score функцията. Това беше докладвано от проф. Янев и е резултат от предишно съвместно обсъждане с доц. Миланов.

Подготовката на обзор и решението на този проблем беше възложено съобразно техните компетенции на проф. Янев, проф. Мирчев и доц. Миланов.

Неговата важност е тясно свързана с COST акцията, на която доц. П. Миланов и доц. Н. Пенчева са членове на координациония комитет.

За реализацията на подхода е необходимо (с помощта на специалисти по молекулярна химия) да се :дефинират профилите на протеина и молекулата-кандидат и релация „близост” в дефинираните профили, позволяваща третиране на гъвкави структури; score функция (функции); допустимост в множеството на кликите (кликата съответства на подреждане, но не всяко подреждане е допустимо от физически съображения). Гл. ас. Ж. Велков, който е член на колектив и на работна група I и II, предложи подход апроксимация на 3D структурата на лиганда, който да се оформи по нататък като алгоритъм.

Взето бе решение на следващата среща да се представят конкретни биологични моделни системи за тестване на математическите модели.

MOLECULAR DOCKING PROBLEM

Кратък обзор

Проф. Н. Янев, Проф. Ив. Мирчев, Доц. П. Миланов

The development and implementation of a range of molecular docking algorithms, based on different search methods (Taylor et al. 2002, Halperin et al. 2002) was observed in the last few years. This approach has had several recent successes in drug discovery (Sechi et al., 2005; Liu et al., 2005).

In the field of molecular modeling, docking is a method which predicts the preferred orientation of one molecule to a second when bound to each other to form a stable complex (Lengauer and Rarey, 1996). Knowledge of the preferred orientation in turn may be used to predict the strength of association or binding affinity between two molecules using for example scoring functions.

Docking is frequently used to predict the binding orientation of small molecule drug candidates to their protein targets in order to in turn predict the affinity and activity of the small molecule. Hence docking plays an important role in the rational design of drugs (Kitchen et al., 2004). Given the biological and pharmaceutical significance of molecular docking, considerable efforts have been directed towards improving the methods used to predict docking.

Evaluation of existing docking algorithms can assist in the choice of the must suitable docking programs for any particular study. Effectively, several studies estimating and comparing the accuracies of protein-ligand programs like Dock, ICM, Gold have been reported (Perola, E. et al. , 2004 ; Bursulaya, B. D. et al., 2003).

The goal of this study was to evaluate the ability of ArgusLab, a freely distributed molecular modeling package in which molecular docking is implemented, to reproduce crystallographic binding orientations and to compare its accuracy with that of the widely well established docking packages, Autodock and FlexX.

БИОЛОГИЧНИ БАЗИ ОТ ДАННИ

<u>Biological databases – Introduction and examples</u>

As the time goes by, the human knowledge is getting larger and larger. The human thirst of discovering is unstoppable. This situation is in every science that exists, including the biology. All that information about all the results of experiments, discoveries, etc… , should be stored and systemized in some data base, in order to make the access to it a lot easier to the scientists or whoever is needed the information.

Biological databases are not just ordinary databases. They are large libraries that contain life science information. All this information are collected from scientific experiments, some published literature etc. Biological databases contain information from many research areas, such as: proteomics, genomics, phylogenetics, metabolomics, etc. This information included in biological databases contain gene localization (both cellular and chromosomal), function, structure, clinical effects of mutations as well as similarities of biological sequences and structures.

Genomic Databases

The best known genomic database is GenBank, the database of genomic series preserved by the NCBI (National Center for Biotechnology Information) . It's full of all explained nucleic acid and amino acid sequences. Its contents are represented by two other databases, the EMBL (European Molecular Biology Laboratory) database and DDBJ (DNA Data Bank of Japan). Distant from presenting and interpreting sequences, these databases suggest many functions related to searching and browsing sequences. They achieve services concerning submitting new sequences to GenBank, and also contain links to various bioinformatic internet sites. Many databases have more specialized information on genomic sequences.

Examples of such databases are the Single Nucleotide Polymorphisms (SNP) Consortium database for biomedical research, the databases of highly conserved DNA motifs, and the cisRED database of gene promoter and regulatory sequences. The database serves for normalization the nomenclature for genes.

Proteomic Databases

Be indebted in communication between amino acid and codon sequences, there are strong links between protein and nucleotide databases. Data on series of amino acids, on the nomenclature, functional characteristics of proteins, protein families and domains, as well as data on known secondary and 3D structures of proteins, are stored in proteomic databases, Swiss-Prot, Uni-Prot, and ExPASy (Expert Protein Analysis System) involving information including the function, taxonomy, amino acid sequences in proteins and structures of proteins.

The PDB (Protein Data Bank) database contains annotated data on the 3D structures of proteins and biological macromolecules. It also includes data on their sequences, functions, and related diseases.

There are also many other databases representing and specialized in particular aspects of proteins, protein functions, experiments involving proteins etc., such as databases of protein 2D gels, limitation enzymes, and secondary structures.

RNA Databases

Information on series of ribonucleotides in RNA, coding and noncoding RNA sequences, the functions of RNA molecules and their spatial configuration, will be available in the databases if necessary. Rfam database stockpiles noncoding RNA (ncRNA) families. Rfam also consist of multiple series databases of Genetic and Proteomic Pathways alignments and covariance models. The GtRNA database stockpiles genomic tRNA ribonucleotide series and secondary structures. The Jena index of RNA structures provides a lot of information about RNA, including indexes of the locations of molecular structures in the PDB database. Records on ribonucleic acid sequences in RNA can also be found in GenBank.

Gene Expression Databases

Many databases include data on expression levels measured in various experiments. Here in this article we list sow some of the best known. They are designed at making possible the distributing of data in the new field of microarray experiments. A component of the NCBI database, NCBI Gene Expression Omnibus (GEO), is a database containing links to microarray-based experiments measuring mRNA, genomic DNA, and protein loads, as well as non array techniques such as serial analysis of gene expression (SAGE), and mass spectrometric proteomic data. The MGED database is full of datasets from many experimental studies occuping gene expression. It also includes links to gene expression data-processing procedures and ontologies. The databases CGED (Cancer Gene Expression Database) and ONCOMINE carry published cancer gene expression data to the research community.

Ontology Databases

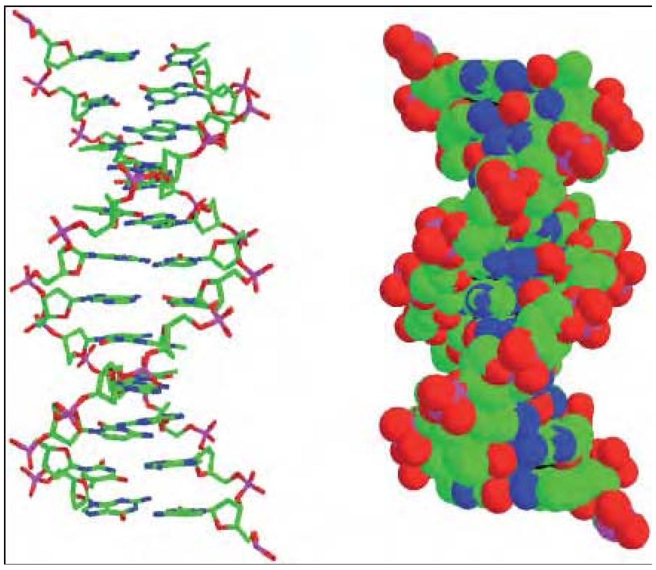
Perhaps the most extensively used ontology database is GO (Gene Ontology), which provides controlled vocabularies for supporting analyses of gene expression measurements and other molecular-biology experiments. But other ontologies are also developing at a fast rate. One good example of database containing links to many Internet ontologies is OBO (Open Biomedical Ontologies). It supplies Web addresses of many sites full of biomedical structured vocabularies, containing GO, the Generic Model Organism Project (GMOD), Microarray Gene Expression Data (MGED), The National Center for Biomedical Ontology vocabulary.

Databases of Genetic and Proteomic Pathways

The area of genetic trails is a very fast-growing field for bioinformatic data. One large database containing such data on the genetic lane is KEGG, the Kyoto Encyclopedia of Genes and Genomes, a Web site that categorizes databases and associated software. The BioCarta database supports proteomic studies by granting information on proteomic lanes, as well as on reagents, antibodies, proteins, cells and cell-based tests. Trail databases suggest a graphical presentation of their contents, which provides 352 12 Bioinformatic Databases and Bioinformatic Internet Resources a useful support for qualitative understanding of signaling, regulatory, and other mechanisms.

Clinical Databases

In this last section we refer to the repositories of clinical data. These databases are growing at a very fast rate owing to their value in developing knowledge in relation to



disease etiologies, therapies, treatments, and so forth. They accumulate data explaining clinical cases, diagnoses, therapy protocols, recovery, and Fig.12.1.

The illustration of the spatial structure of atoms and bonds in a DNA polymer took with the use of publicly available programs 3DNA and Ras Mol. The representation on the left, called "sticks", shows atomic bonds, while the one on the right, called "balls" gives a picture of the approximate atomic radii. The colors match to atom types in the following way: red, oxygen; green, carbon; purple, phosphorus; blue, nitrogen

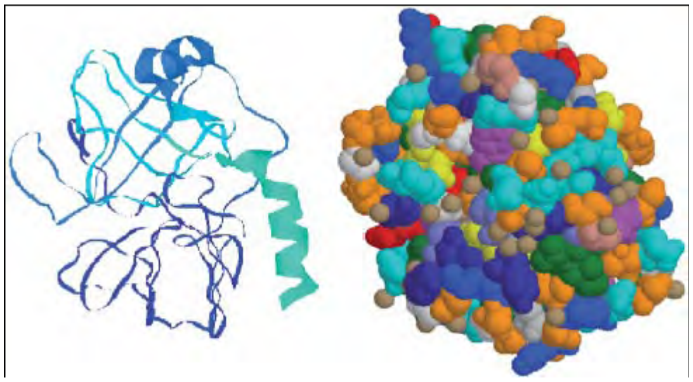


Fig.12.2. Graphical arrangement of the enzyme trypsin took with the use of spatial coordinates of atoms from Protein Data Bank (accession symbol 2ptn), and the molecular-graphics program Ras Mol.

Left: A view resulting from choosing the Ras Mol option "ribbons" for boosting secondary structures.
Right: A view resulting from using "spacefill" option.

Different colors represent diverse amino acids.

The importance of colors is as follows: ASP, GLU, bright red; LYS, ARG, blue; PHE, TYR, mid blue; ALA, dark grey; HIS, pale blue; CYS, MET; yellow; SER, THR, orange; ASN, GLN, cyan; LEU, VAL, ILE, green; TRP, pink; PRO, flesh continuing existence.

Different types of data are organized: patients records, diagnostic tests containing medical images, treatment plans and follow up data on patient cohorts concerning recovery, complications and survival. There are many sides of the development of clinical databases: quality, completeness and volume of data, availability and so forth.

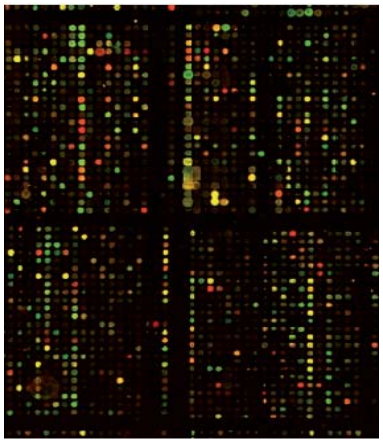


Fig.12.3. An example of an image of a cDNA microarray, from the Website containing the data accompanying. An example of a clinical database is the European EURODIAB database related to type 1, childhood diabetes, resulting from many years of collecting clinical cases of type 1 diabetes mellitus. Along with storing patient records, treatments, and survival data, clinical databases can also put in order and help with access to tissue banks. An example is the GENEPI (GENEtic Pathways for the Prediction of the Effects of Ionising Radiation) database, which stores clinical data concerning therapeutic radiation and, at the same time, can help in arranging access to tissue samples.

BioSQL

BioSQL (Biological Structured Query Language) is a generic relational model covering sequences, features, sequence and feature annotation, a reference taxonomy, and ontologies (or controlled vocabularies).

The beginnings of this project start in 2001, mainly by Ewan Birney's idea. This project became collaboration between the BioJAVA, BioPerl, BioPython and BioRuby projects. The idea, or should we say: the goal of this project is to build a generic schema for permanent storage of features, sequences and annotation in a way that is interoperable between the Bio projects. Each Bio* project has a language binding to BioSQL.

BioPerl

What is BioPerl? It is a group of modules from the high-level dynamic programming language Perl, that is making development of Perl scripts for bioinformatics applications a lot easier. This project is an active open source software project supported by the OBF (Open Bioinformatics Foundation).

The initial release date of this project is June 11, 2002. The latest release date of the latest version (v. 1.6.0) is not so long ago: 2009-01-25. This last version 1.6.0 is considered to be the most stable version of BioPerl, and therefore is recommended to use this version.

Of course, if a user wants to use BioPerl, he needs to have some basic experience with the programming language Perl. It means that the user needs to understand how to use Perl modules, objects, methods and references. It also provides software modules for many of the typical tasks of bioinformatics programming. These include: creating and manipulating sequence alignments, accessing nucleotide and peptide sequence data from local and remote databases, searching for similar sequences etc.

BioJava

BioJava also is an open-source project that is providing a Java framework for the processing of biological data. Java is a Object-Oriented language from high level. BioJava, however, contains objects for manipulating file parsers, biological sequences, access to BioSQL. One of the most powerful advantages that BioJava has is the tools for making sequence analysis GUIs and powerful analysis and statistical routines including a dynamic programming toolkit.

BioPython

Similar as the previous projects, the BioPython project is an international association of developers. This project is about freely available Python tools for computational molecular biology. These developers work along the OBF, which provides web space, and CVS space as well. This project is an effort to develop Python libraries and applications along to the needs of the bioinformatics.

What we need for creating, installing and configuration our biological databases. At the beginning we need the following:

- the latest version of BioJava to enjoy the full functionality of BioSQL (downloadable from biojava.org).
- need the latest Oracle BioSQL schema.(there are 2 options to work fine)

Formerly a different schema was presented, but *BioJava* recommends using only the official schema.

Original: by Hilmar Lapp, the original BioSQL schema makes it possible to use all the security mechanisms of Oracle and produces a high quality schema.

For downloading the schema, go to the Downloads link on the BioSQL website, or otherwise download the latest revision of the Oracle version from anonymous svn:

```
$ svn co svn://code.open-bio.org/biosql/biosql-schema/trunk/sql/biosqldb-ora biosqldb-
ora
```

If we are using a BioSQL release then we need to unpack the file and look for the Oracle version in the sql/biosqldb-ora directory

Alternative: ***(deprecated)***: by Len Trigg, this version is working fine and has a build in single user account, and we don't have to install the sysdba access to work with. If we need the sysdba we have to ask for a copy of the script from the biosql-l mailing lists.

Both options (the orginal and the alternative) are completely efficient and well-matched with both BioJava and BioPerl. In this article we are using only the first one.

ORACLE

To use an Oracle database we need to install the latest version of the JDBC drivers. BioJava is tested with BioSQL using Oracle 9i and 10g. For the **Original** schema, we will also need sysdba access, or get our DBA to help us if we do not have this ourself. Things that require sysdba access/DBA assistance incorporate creating tablespaces (or being assigned one to use), creating or assigning tasks, and creating or assigning additional user accounts other than our own, if we are going to install BioSQL outside of our own account. If our DBA does any of this for us, then we will need to mention out the correct steps in BS-create-all.sql before starting the installer.

- Tablespaces are shaped in BS-create-tablespaces.sql.
- Roles are created in BS-create-roles.sql.
- Users are created in BS-create-users.sql.
- Global roles and users are defined in BS-defs-local.sql (see below on how to set this up).

BUGFIXING

During the creating of this article all possible problems that were recognized with the default BioSQL setup scripts were resolved. But be careful when installing, there may still be problems unique to your environment.

One exciting bug that is not linked to BioSQL but may cause you grief is to do with the built-in ODM Blast functionality in Oracle 10g. ODM Blast will throw "table or view does not exist" faults if you pass it a cursor over a table that is in fact a synonym (eg. biosequence and bioentry in any of the users you have granted biosql_user or biosql_loader to).

We can only run ODM Blast over actual physical tables or views, and not synonyms of them.

INSTALLATION

Before running any script we need to be sure that we have set $ORACLE_SID environment variable to the correct database.

There may be special necessities to reconnect to the database, and if it is not set, we may end up running the scripts against the incorrect database. Instead, we can attach "@SID" to our passwords each time we are asked for them during setup, where "SID" is the SID of our database.

The installation requires the making of three tablespaces *(May be created or allocated for us by our DBA)* - one for data, one for indexes, one for LOB objects. We must decide where we will be keeping the database files for these, and what we will call the tablespaces. We don't create them yet though, we just write the names down. As always it is good practice to keep the data and index tablespaces on separate disks to avoid IO blockages, but we can perhaps safely put the data and LOB tablespaces on the same disk.

We will also require to make a decision on names for the two basic roles that BioSQL makes use of *(May be created or assigned for us by our DBA)* - the base_user role which includes just enough advantages to connect to the database, and the schema_creator role, which includes the privileges required to create database objects in a schema. For a second time, we don't create them just yet.

Now, we must copy BS-defs.sql to BS-defs-local.sql and edit it. We should check every entry in it carefully, particularly the names and locations of the tablespace files to be created, and the names of the two roles we just decided on above. We will also pick up names for the various default BioSQL roles and users. biosql_owner is the actual owner of the schema that should already exist and have had the schema_creator role granted to it, you'll need to define its password here too. biosql_user is a role to be granted to people who need read-only access to the BioSQL database, biosql_loader is a role designed for common read/write access, whilst biosql_admin has full read-write authorization on the schema.

Once we have edited the BS-defs-local.sql script suitably, we need to generate the two base roles of base_user and schema_creator manually. We are creating them by running something similar to the next script whilst logged in as sysdba, from inside the biosqldb-ora directory:

```
@BS-defs-local
  create role &base_user;
  grant
  CREATE SESSION,
  CREATE SYNONYM
  to &base_user;
  create role &schema_creator;
  grant
  CREATE PROCEDURE,
  CREATE ROLE,
  CREATE SEQUENCE,
  CREATE SESSION,
  CREATE SYNONYM,
  CREATE TRIGGER,
  CREATE TYPE,
  CREATE VIEW,
  CREATE TABLE,
  CREATE PUBLIC SYNONYM,
  DROP PUBLIC SYNONYM
  to &schema_creator
  with admin option;
```

If we desire to set up some basic users, we must edit the BS-create-users.sql script to look at the sample users it will create for us without human intervention. If we don't want them, or want special names etc., comment them out or edit them.

The final step before the real installation is to edit the BS-create-all.sql script to make sure that only the steps we need are carried out. If we already have predefined tablespaces and don't want it to create new ones, comment out the line that reads @BS-create-tablespaces. We should do the same for @BS-create-users and @BS-create-roles as necessary. The same if we don't want any default data loaded into the database, comment out the line near the end that reads @BS-prepopulate-db.

We are checking if we have commented/uncommented the appropriate components of section 9 of BS-create-all.sql. The BS-create-Biosql-API2 script it refers to is an alternative to BS-create-Biosql-API, which works much better with BioJava. This is because BioJava has no flexibility about column names in tables. The API2 version of the script guarantees that the column names are exactly the same as what BioJava expects by using views. But, no matter which we run, everything will still work fine with BioPerl.

At this moment, logging into our Oracle database and creating the BioSQL database by typing:

```
@BS-create-all
```

It is prompting us for the sysdba user and password if necessary (unless we mentioned out these pieces), maybe a couple of times. We would like to pin the output to see what happens, but we'll discover that half of it doesn't appear in the spool file, because BioSQL is using spool itself to make dynamic scripts on the fly. When we have done everything right, the only messages we should get are a few Table or view does not exist style messages, referring to the attempts by the script to drop old objects before recreating new ones.

During the installation process we may be asked for the sysdba username and password many times. This will happen and is required only to create roles, tablespaces, and users.

If something goes wrong, we can safely repeat the script without dropping anything first as it will drop the database objects from the previous attempt first. It will still leave after the tablespaces, users, and roles. We can always just drop the users and tablespaces that have been created if it really messes up, and start again from scratch.

At this time, our database has been installed! The only step we must do is to log in to each user who will be using BioSQL, and run the usersyns.sql script that the installation generated for us in the biosqldb-ora directory. This script makes the synonyms for the BioSQL objects and permits the users to see them. This script should not have any errors at all. If it does, edit it and check it closely for things like misplaced linebreaks etc.

Note that if our users can't join or can't get the suitable authorization to do what we want them to do, try re-running the BS-create-roles script as sysdba, then the BS-grants script as the biosql_owner user. Disconnect and reconnect as the user having trouble and it should be fixed.

TESTING: Any BioJava script should work fine.

PostgreSQL

PostgreSQL is open source object-relational DBMS (database management system). Its development dates from about 15 years ago, and it’s still active. This database system has earned a strong reputation over the years. Its architecture is very strong and provides good qualities for data integrity, reliability and correctness. This DBMS runs on all major operating systems, such as Linux, UNIX and Windows. It includes SQL92 and SQL99 data types, including, BOOLEAN, CHAR, DATE, INTEGER, NUMERIC, INTERVAL, VARCHAR, and TIMESTAMP. This DBMS also supports the storage of binary large objects, such as pictures, videos and sounds. It has programming interfaces for Java, .Net, C/C++, Perl, Python, Ruby, ODBC, Tcl etc.

PostgreSQL is proud to posses sophisticated features like: tablespaces, online/hot backups, query planner/optimizer, etc, also supports international character sets, multibyte character encodings, Unicode, and it is locale-aware for sorting, case-sensitivity, and formatting. It is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL systems in production environments that manage in excess of 4 TB of data. It also has won Linux New Media Award for Best Database System and it is five time winner of the The Linux Journal Editors' Choice Award for best DBMS.

Featureful and Standards Compliant

PostgreSQL prides itself in standards compliance. Its SQL implementation strongly conforms to the ANSI-SQL 92/99 standards. It has full support for subqueries (including subselects in the FROM clause), read-committed and serializable transaction isolation levels. And while PostgreSQL has a fully relational system catalog which itself supports multiple schemas per database, its catalog is also accessible through the Information Schema as defined in the SQL standard.

Data integrity features include (compound) primary keys, foreign keys with restricting and cascading updates/deletes, check constraints, unique constraints, and not null constraints.

It also has a host of extensions and advanced features. Among the conveniences are auto-increment columns through sequences, and LIMIT/OFFSET allowing the return of partial result sets. PostgreSQL supports compound, unique, partial, and functional indexes which can use any of its B-tree, R-tree, hash, or GiST storage methods.

GiST (*Generalized Search Tree*) indexing is an advanced system which brings together a wide array of different sorting and searching algorithms including B-tree, B+-tree, R-tree, partial sum trees, ranked B+-trees and many others. It also provides an interface which allows both the creation of custom data types as well as extensible query methods with which to search them. Thus, GiST offers the flexibility to specify *what* you store, *how* you store it, and *the ability to define new ways* to search through it --- ways that far exceed those offered by standard B-tree, R-tree and other generalized search algorithms.

GiST serves as a foundation for many public projects that use PostgreSQL such as OpenFTS and PostGIS. OpenFTS (Open Source Full Text Search engine) provides online indexing of data and relevance ranking for database searching. PostGIS is a project which adds support for geographic objects in PostgreSQL, allowing it to be used as a spatial database for geographic information systems (GIS), much like ESRI's SDE or Oracle's Spatial extension.

Other advanced features include table inheritance, a rules systems, and database events. Table inheritance puts an object oriented slant on table creation, allowing database designers to *derive* new tables from other tables, treating them as base classes. Even better, PostgreSQL supports both single and multiple inheritance in this manner.

The rules system, also called the *query rewrite system*, allows the database designer to create rules which identify specific operations for a given table or view, and dynamically transform them into alternate operations when they are processed.

The events system is an interprocess communication system in which messages and events can be transmitted between clients using the LISTEN and NOTIFY commands, allowing both simple peer to peer communication and advanced coordination on database events. Since notifications can be issued from triggers and stored procedures, PostgreSQL clients can monitor database events such as table updates, inserts, or deletes as they happen.

Highly Customizable

PostgreSQL runs stored procedures in more than a dozen programming languages, including Java, Perl, Python, Ruby, Tcl, C/C++, and its own PL/pgSQL, which is similar to Oracle's PL/SQL. Included with its standard function library are hundreds of built-in functions that range from basic math and string operations to cryptography and Oracle compatibility. Triggers and stored procedures can be written in C and loaded into the database as a library, allowing great flexibility in extending its capabilities. Similarly, PostgreSQL includes a framework that allows developers to define and create their own custom data types along with supporting functions and operators that define their behavior. As a result, a host of advanced data types have been created that range from geometric and spatial primitives to network addresses to even ISBN/ISSN (International Standard Book Number/International Standard Serial Number) data types, all of which can be optionally added to the system.

Just as there are many procedure languages supported by PostgreSQL, there are also many library interfaces as well, allowing various languages both compiled and interpreted to interface with PostgreSQL. There are interfaces for Java (JDBC), ODBC, Perl, Python, Ruby, C, C++, PHP, Lisp, Scheme, and Qt just to name a few.

Best of all, PostgreSQL's source code is available under the most liberal open source license: the BSD license. This license gives you the freedom to use, modify and distribute PostgreSQL in any form you like, open or closed source. Any modifications, enhancements, or changes you make are yours to do with as you please. As such, PostgreSQL is not only a powerful database system capable of running the enterprise, it is a development platform upon which to develop in-house, web, or commercial software products that require a capable RDBMS.