pgchem::tigress – The chemoinformatics extension for PostgreSQL: User Guide

Ernst-Georg Schmid

v.7.1 – Summer 2007

The project tigress © Windhund 2004

The barsoi accelerator logo © Windhund 2005

Contents

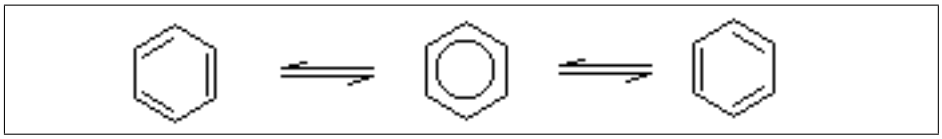

Figure 1: Mesomeric extremes of Benzene

Part I
Caveat

pgchem::tigress is not exhaustively tested and may contain errors in functionality and code. Therefore it should not be used unconsiderate and does not replace the advice of a trained chemist. Notably, pgchem::tigress was not designed to be used in GxP environments, for material safety systems and other safety critical environments.

Part II
Introduction to pgchem::tigress

1 Overview

pgchem::tigress is a chemoinformatics extension to the PostgreSQL object-relational database management system. It enables PostgreSQL to handle chemical datatypes. Pgchem::tigress is basically a wrapper around the checkmol/matchmol molecular analyzer and the OpenBabel computational chemistry package, plus some database functions and tables to access their functionality purely through SQL statements.

Pgchem::tigress supports exact and substructure searching on molecules and reactions with strict and relaxed adherence of atom and bond types, searching by functional groups, calculation of chemical properties like molecular formula and molecular weight, datatype conversion and Tanimoto similarity searching.

As of version 8.x of PostgreSQL, pgchem::tigress compiles and runs natively (at least) on Linux, Solaris, OS X and Win32.

2 The Challenge

A molecule can basically be represented as an undirected graph, where the atoms are the nodes and the bonds are the edges of the graph. A program operating on such graphs must calculate all kinds of chemical information from that graph to determine if, e.g. two molecules are chemically equal or different. In Figure 1 the atoms (nodes) are at the same positions. The bonds (edges) however have changed places. Those two graphs are not topologically equal but are chemically

```
CREATE TABLE example.molecules
(
  iiid serial NOT NULL,
  molecule bytea NOT NULL,
  CONSTRAINT pk_molecules PRIMARY KEY (iiid)
)
```

Figure 2: A basic molecules table

equal to Benzene. To recognize this, a program must know about mesomerism and aromatic bindings and detect, that those are the two mesomeric extremes of the aromatic ring Benzene. Benzene can also be drawn in a third way with a circle in the center to symbolize the delocalized electrons, also shown in Figure 1. This is only one of the many challenges of chemoinformatics.

3 Why PostgreSQL?

Because it is tried and tested, suitable for heavy-duty applications and has a clean interface for custom extensions. So far it has proven to be a good choice, but there are no reasons why there should not be a fbchem::tigress for Firebird or an orachem::tigress for Oracle in the future. Actually, mychem for MySQL has started at http://sourceforge.net/projects/mychem/.

Part III
Designing your schema

Pgchem::tigress needs a few special tables that act as additional indexes for speeding up queries. Those tables have to be present in every database that uses pgchem::tigress. The layout of those tables is very rigid (as not to say: fixed) because all the functions and mainly the maintenance triggers rely on those tables.

4 The molecules table

The molecules table holds the molecules itself, but can be extended to store whatever additional information should be attached to those molecules. It consists of at least two columns, as shown in Figure 2.

The **iiid** is a type *serial* , artificial key to connect each molecule to their corresponding entries in the index tables.

I don't remember what iiid originally meant, but one i stands for internal. That means that you should not rely on this as a key outside the pgchem::tigress subschema. If you need a key for your own application-specific tables, add your own key column, e.g. the CAS-registry-number.

The molecule column holds the molecule as MDL V2000 molfile. The actual name of the molecule column is unimportant for pgchem::tigress, choose it to your liking, but it has to be of type *bytea*.

```
CREATE TABLE example.molkeys
(
  iiid int4 NOT NULL DEFAULT 0,
  n_atoms int2 NOT NULL DEFAULT 0,
  n_bonds int2 NOT NULL DEFAULT 0,
  n_rings int2 NOT NULL DEFAULT 0,
  n_qa int2 NOT NULL DEFAULT 0,
  n_qb int2 NOT NULL DEFAULT 0,
  n_chg int2 NOT NULL DEFAULT 0,
  n_c1 int2 NOT NULL DEFAULT 0,
  ...
  CONSTRAINT molkeys_pkey PRIMARY KEY (iiid),
  CONSTRAINT fk_mol FOREIGN KEY (iiid)
  REFERENCES example.molecules (iiid) ON UPDATE NO ACTION
  ON DELETE CASCADE
  )
```

Figure 3: The molecular keys table

```
CREATE TABLE example.molfingerprints
(
  iiid integer NOT NULL,
  fp2bitmap bytea NOT NULL,
  CONSTRAINT molsimilarity_pkey PRIMARY KEY (iiid),
  CONSTRAINT fk_id FOREIGN KEY (iiid)
      REFERENCES example.acd2d_moltable (iiid) MATCH SIMPLE
      ON UPDATE RESTRICT ON DELETE CASCADE
)
```

Figure 4: The molecular fingerprints table

5 The molecular keys and fingerprint tables

Chemical matching is a time-consuming process. If all molecules of a database with more than 2-300 entries would have to be individually graph-checked, search performance would severely degrade. So what happens is that for every molecule the molecular keys and/or a molecular fingerprint is generated and stored in the molecular keys and fingerprint tables.

Figure 3 shows an excerpt of the molecular keys table definition. It is linked to the molecules table via the **iiid** and consists of integer columns which store qantitative information about each molecule, e.g. how many atoms, how many bonds etc. Consequently, this table is called molkeys within pgchem::tigress.

Using that table as a sort of index, the workload of chemical matching can be dramatically reduced by matching the molecular keys of the query molecule against this table first and to process only the candidates which came out of this search. For an exact match search, the molecular keys are compared by the '=' operator and for a substructure search by the '>=' operator to filter for possible candidates. See Part V about how to do this in SQL.

```
CREATE TABLE example.molfgroups
(
  iiid int4 NOT NULL,
  code char(8) NOT NULL,
  CONSTRAINT molfgroups_pkey PRIMARY KEY (iiid, code),
  CONSTRAINT fk_mol FOREIGN KEY (iiid) REFERENCES
  example.molecules (iiid)
  ON UPDATE NO ACTION ON DELETE CASCADE
)
```

Figure 5: The functional groups table

Figure 4 shows the fingerprints table definition. It is linked to the molecules table via the **iiid** and consists of a 1024 bit wide bytearray, each bit representing the presence/absence of a pattern derived from the molecule. Like the molecular keys table, this can be used as an index for finding possible match candidates. For an exact match search, the fingerprints are compared by the '=' operator and for a substructure search by the *substruct_screen_fingerprint(bytea, bytea)* function to filter for possible candidates. Experience shows that combining molecular keys and fingerprints for screening yields the best (smallest, fastest) results.

The fingerprints table entires can also be used to calulate the similarity between to molecules, e.g. using the Tanimoto coefficient.

6 The functional groups table

The functional groups table contains 0..n rows per molecule. As shown in Figure 5, each row contains a code that indicates the presence of a specific functional group in this molecule. Checkmol/matchmol currently detects 190 functional groups, thus does pgchem::tigress. This table can be used to directly search for molecules containing a given set of functional groups. See Part V about how to do this in SQL.

7 The similarity fingerprint table

Merged into 5.

8 The maintenance triggers

Consistency between the molecules table and the various index tables is maintained automagically in pgchem::tigress. `DELETE` is handled by the referential integrity relations between the tables, this avoids orphaned rows in the index tables. `INSERT` and `UPDATE` are a little more complex, as the input molecule has to be analyzed in various ways to fill the index tables. This is handled by two triggers on the molecule table. One trigger adds additional information to the molecule. Checkmol/matchmol has to determine which bonds are truly aromatic, their stereo configuration etc. to do its magic and this is an

```
CREATE TRIGGER precompute_properties
  BEFORE INSERT OR UPDATE
  ON example.molecules
  FOR EACH ROW
  EXECUTE PROCEDURE public.t_precompute_properties();
```

Figure 6: The tweak molecule trigger

```
CREATE TRIGGER maintain_pgchem_indextables
  AFTER INSERT OR UPDATE
  ON example.molecules
  FOR EACH ROW
  EXECUTE PROCEDURE public.t_maintain_pgchem_indextables();
```

Figure 7: The main maintenance trigger

expensive operation, because it cannot rely on the information in the molfile (you can mark every bond as aromatic, wether it is or not), but has to apply the Hückel rule. So checkmol/matchmol has an option to store this information precalculated in unused fields of the molfile to speed up all further operations. The trigger shown in Figure 6 performs that precalculation upon every change of a molecule entry in the database. This is optional, but highly recommended.

The main maintenance trigger shown in Figure 7 however is mandatory. It guarantees, that every change to a molecule entry is propagated to the corresponding entries in the index tables and must not be omitted, except there are no updates to the tables after initial loading and the index tables are filled manually.

Part IV
Loading data

If the Schema is set up correctly and all the maintenance triggers are in place, start loading your molecules table by any means you like. Make sure, that the molfiles are sent to the database as *bytea*, not as *text*, with UN*X or DOS style line endings. For every `INSERT` or `UPDATE`, the content of all index tables will be automatically adjusted. If something did not work the first time, issuing '`UPDATE <molecule_table> SET <molecule_column>=<molecule_column>;`' forces a complete rebuild of the index tables.

A complete rebuild is also necessary, when checkmol/matchmol changes it's fingerprint semantics, e.g. from 0.2i to 0.2j, but this rarely happens.

Pgchem::tigress neither needs nor supports the notion of a MDL No-Structure. Filter No-Structures before or while loading and replace them with `NULL`. All pgchem::tigress functions are declared `STRICT` and handle `NULL` input gracefully.

```
SELECT <molecule_table>.iiid FROM
<molecule_table>,molkeys WHERE
n_atoms=13 AND n_bonds=15 AND
n_rings=6 AND n_C2=11 AND
n_C=11 AND n_CHB1p=4 AND
n_N2=1 AND n_N3=1 AND n_b2=6 AND
n_bar=15 AND n_CN=4 AND
n_rN=5 AND n_rN1=3 AND n_rN2=2 AND
n_rX=5 AND n_rar=6 AND
<molecule_table>.iiid=molkeys.iiid AND
match_exact(<query_molecule>,<molecule_column>,false,false,
false,false,false,false)=TRUE;
```

Figure 8: Exact match with a function

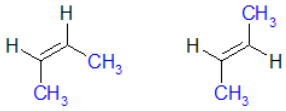

Figure 9: Example search without E/Z geometry checking

Part V
Working with data

All examples use abstract placeholders for table/column names and SQL variables. How they have to look like for a specific programming language/database driver combination is left as an excercise to the reader.

9 Molecule exact searching

Exact searching can be done by using the function `match_exact()`. The function call variant is shown in Figure 8. With E/Z geometry matching switched on, E/Z-stereoisomers are treated as different structures. Figure 9 shows an example for (Z)-2-Butene and (E)-2-Butene.

With R/S geometry matching switched on, R/S-stereoisomers are treated as different structures. Figure 10 shows an example. R/S geometry matching by default **does not** reject otherwise matching structures with flat or bonds! If you need this, either switch on strict adherence to atom and bond types also.

The test for chemical equality is commutative.

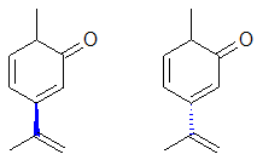

Figure 10: Example search without R/S geometry checking

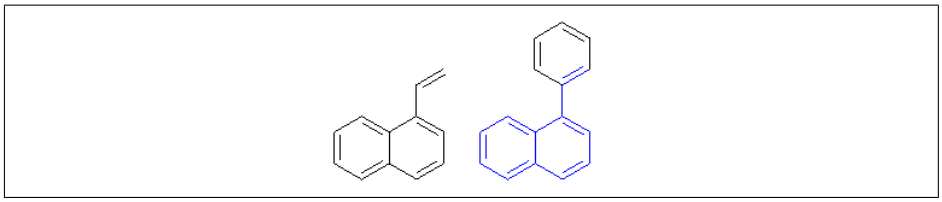

Figure 11: Example substructure search with relaxed atom and bond typing: different bond types

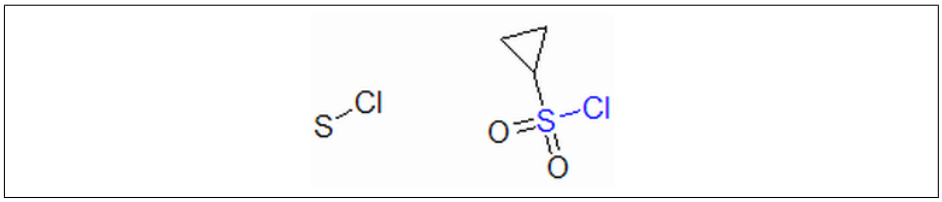

Figure 12: Example substructure search with relaxed atom and bond typing: different atom types

10 Molecule substructure searching

Substructure searching is a bit more complex that exact searching, namely because atom and bond types now become a variable. Figure 11 shows an example. With relaxed adherenece of typing, the search for the substucture on the left finds the result on the right. With strict adherence of typing, it would reject this match, because the uppermost double bond of the query molecule is not aromatic, but the corresponding bond in the benzene ring of the result molecule is.

Figure 12 shows another example. With relaxed adherenece of typing, the search for the substucture on the left finds the result on the right. With strict adherence of typing, it would reject this match, because the left Sulphur atom is an S3 (sp3 sulfid) while the right one is an SO2 (sulfon).

The Figure 13 contains an example how to perform such a search by function. Please note, that now the molecular fingerprint table has to be explicitly refer-

```
SELECT <molecule_table>.iiid FROM
<molecule_table>,molkeys WHERE
n_atoms>=13 AND n_bonds>=15 AND
n_rings>=6 AND n_C2>=11 AND
n_C>=11 AND n_CHB1p>=4 AND
n_N2>=1 AND n_N3>=1 AND n_b2>=6 AND
n_bar>=15 AND n_CN>=4 AND
n_rN>=5 AND n_rN1>=3 AND n_rN2>=2 AND
n_rX>=5 AND n_rar>=6 AND
<molecule_table>.iiid=molkeys.iiid AND
match_substruct<query_molecule>,<molecule_column>,false,
false,false,false,false,false)=TRUE;
```

Figure 13: Substructure match with a function

enced in the SQL statement and that the fingerprint columns are searched as described in Section 5. You can get the fingerprint values of the query molecule from the database by calling the *ms_fingerprint_long()* function and then append it to your SQL statement.

11 Special query features when using MDL V2000 molfiles as query structure

The following special atom symbols may be used in **query** structures:

- 'A' matches all atoms except Hydrogen.
- 'Q' matches all atoms except Hydrogen and Carbon.
- 'X' matches all halogen atoms.

The following special bond types may be used in queries:

- '4' matches aromatic bonds.
- '5' matches single or double bonds.
- '6' matches single or aromatic bonds.
- '7' matches double or aromatic bonds.
- '8' matches any bond type.

The following special bond topology markers may be used in queries:

- '0' matches any bond topology.
- '1' matches ring bonds.
- '2' matches chain bonds.
- '00' matches any bond topology even in strict mode.
- '01' matches only ring bonds that are member of more rings than in the query (for annulated systems).
- '02' matches only ring bonds that are member of exactly the same amount of rings than in the query.

The following special stereo markers may be used in queries:

- 'Stereo care flag' switches on E/Z geometry matching.
- 'dblEither' type bonds switch off E/Z geometry matching.
- 'Chiral flag' switches on R/S geometry matching.

ACHTUNG: Structures containing query features can be registered, but may not be fully searcheable. The database server will write a warning referring to this in the server log.

```
SELECT <molecule_table>.iiid FROM
<molecule_table> WHERE
instant_tanimoto(<query_molecule>,<molecule_column>) > 0.9
```

Figure 14: On-the-fly similarity search

```
SELECT <molecule_table>.iiid FROM
<molecule_table>,molsimilarity WHERE
<molecule_table>.iiid=molsimilarity.iiid AND
tanimoto(fingerprint2(<query_molecule>),<molecule_column>) > 0.9
```

Figure 15: Precomputed similarity search

12 Functional group searching

To search for molecules containing one or more functional groups, just select the desired functional group codes from the functional groups table and join it with the molecules table. The example scripts that come with pgchem::tigress contain a lookup table, containing all known codes with their english names. This can be used to search by names instead of codes.

13 Tanimoto similarity searching

Currently there are two ways of similarity caclulation, on-the-fly and precomputed. On-the-fly is done by the *instant_tanimoto(bytea,bytea)* function, which takes two molecules and returns their tanimoto coefficient (Figure 14). Precomputed is done by the *tanimoto(bytea,bytea)* function, which takes two fingerprints and returns their tanimoto coefficient (Figure 15). The fingerprints can be obtained from the similarity fingerprint table or dynamically from the *fingerprint2(bytea)* function (Figure 15).

Precomputed searching is typically much faster than on-the-fly.

14 Molecule search functions

- *precompute_properties(bytea,bool)* modifies the input molecule and adds information about aromaticity etc. Such 'tweaked' molecules are faster to compare, because most information was precomputed and stored in the molecule. The flag toggles whether the molecule is always 'tweaked', regardless if it was 'tweaked' before. This function is mainly used in the precompute_properties trigger (Figure 6).

- *match_exact(bytea,bytea,bool,bool,bool,bool,bool,bool)* takes a query molecule, a molecule to exact match the query against, and six flags. The first flag toggles strict comparison of atom and bond types (particularly aromatic), the second flag globally toggles E/Z geometry matching of double bonds, the third flag globally toggles R/S matching of chiral centers[1]. The fourth

[1] R/S and E/Z geometry matching can also be specified *per query*, as described in Section 11.

flag toggles strict checking of charge, the fifth flag toggles strict checking of isotope and the the sixth flag toggles strict checking of radical. A return value of TRUE indicates a match.

- *match_substruct(bytea,bytea,bool,bool,bool,bool,bool,bool)* takes a query molecule, a molecule to substructure match the query against, and three flags. The first flag toggles strict comparison of atom and bond types (particularly aromatic), the second flag globally toggles E/Z geometry matching of double bonds and the third flag globally toggles R/S matching of chiral centers[2]. The fourth flag toggles strict checking of charge, the fifth flag toggles strict checking of isotope and the the sixth flag toggles strict checking of radical. A return value of TRUE indicates a match.

- *match_substruct_smarts(text,bytea)* takes a query SMARTS, a molecule to substructure match the query against. A return value of TRUE indicates a match. This function uses an alternative subgraph isomorphism checking algorithm which is less precise but about 2 times faster. I assume that it respects stereo features in the SMARTS, but I haven't checked that.

- *molkeys_long(bytea,bool,bool,bool)* takes a query molecule and returns its fingerprint in long form. Long means, that for every column a name/value pair *name:value;* is generated. This can be used to obtain the screening fingerprint for exact or substructure matches by replacing the *:* with = or >= and the *;* with *AND* . The first flag toggles strict checking of charge, the second flag toggles strict checking of isotope and the the third flag toggles strict checking of radical. **When used in combination with a match function, the settings of these flags must be identical to the settings of the corresponding flags of the match function.**

- *fgroup_codes(bytea)* takes a query molecule and returns its functional group codes. This can be used to obtain the codes for a functional group search by drawn example.

- *substruct_screen_fingerprint(bytea, bytea)* takes a query fingerprint, a fingerprint to substructure screen the query against. A return value of TRUE indicates a possible substructure match candidate.

For convenience, all search functions taking boolean flags have overloaded siblings without those flags, assuming FALSE for all of them.

15 Calculating properties

- *molweight(bytea)* takes a molecule and returns the standard molar mass given by IUPAC atomic masses, including all implicit hydrogens.

- *exactmass(bytea)* takes a molecule and returns the the mass given by isotopes (or most abundant isotope, if not specified), including all implicit hydrogens .

[2] R/S and E/Z geometry matching can also be specified *per query*, as described in Section 11.

- *total_charge(bytea)* takes a molecule and returns the total charge (0=neutral), including all implicit hydrogens.
- *number_of_atoms(bytea)* takes a molecule and returns the number of atoms, including all implicit hydrogens.
- *number_of_heavytoms(bytea)* takes a molecule and returns the number of heavy atoms. This also counts Deuterium an Tritium!
- *number_of_bonds(bytea)* takes a molecule and returns the number of bonds, including all implicit hydrogens.
- *number_of_rotatable_bonds(bytea)* takes a molecule and returns the number of rotatable bonds[3].
- *is_chiral(bytea)* takes a molecule and tries to perceive its chirality.
- *is_2D(bytea)* takes a molecule and returns true if 2D coordinates are present.
- *is_3D(bytea)* takes a molecule and returns true if 3D coordinates are present.
- *molformula(bytea)* takes a molecule and returns the molformula, including all implicit hydrogens.
- *fingerprint2(bytea)* takes a molecule and returns the OpenBabel Type 2 binary fingerprint.

16 Conversions

- *molecule_to_molfile(bytea)* takes a molecule and converts it to a V2000 molfile.
- *molfile_to_molecule(text)* takes a V2000 molfile and converts it to a molecule.
- *molecule_to_V3000(bytea)* takes a molecule and converts it to a V3000 molfile.
- *V3000_to_molecule(text)* takes a V3000 molfile and converts it to a molecule.
- *molecule_to_smiles(bytea)* takes a molecule and converts it to a SMILES string.
- *molecule_to_canonical_smiles(bytea,bool)* takes a molecule and converts it to a canonical SMILES string. If parameter two is false, any additional stereo information is discarded.
- *smiles_to_molecule(text)* takes a SMILES/canonical SMILES string and converts it to a molecule.
- *molecule_to_inchi(bytea)* takes a molecule and converts it to a IUPAC InChI string.

[3]Any non-ring bond with hybridization of sp2 or sp3 is considered a potentially rotatable bond. There is no special bond-typing, e.g. for amide C-N bonds with their high rotational energy barrier.

17 Modifications

- *strip_salts(bytea)* takes a molecule and strips all atoms except for the largest contiguous fragment.
- *add_hydrogens(bytea,bool,bool)* takes a molecule and adds hydrogens. Parameter one controls if all or only polar hydrogens are added and parameter two if a correction for Ph=7 should be done.
- *remove_hydrogens(bytea,bool)* takes a molecule and removes hydrogens. Parameter two controls if all or only non-polar hydrogens (true) shall be removed. This also removes Deuterium and Tritium!

18 Helper functions

- *validate_cas_no(varchar)* takes a CAS-No. and checks its validity with the official CAS checksum algorithm.
- *validate_molecule(bytea)* checks if a molecule appears to be valid (with a very simple algorithm).
- *is_nostruct(bytea)* checks if a molecule is a MDL NoStruct.
- *pgchem_version()* returns the pgchem::tigress version identifier.
- *pgchem_barsoi_version()* returns the barsoi version identifier.

Part VI
Running with Barsoi

Removed since 7.0. All search functions now use Barsoi by default.

19 FastMatch searching

Removed since 7.1. All search functions now use FastMatch internally where applicable.

Part VII
Miscellaneous

20 Rejecting duplicate molecules

In order to emulate a unique constraint on molecules, a row level `INSERT` and `UPDATE` trigger can be used. First create a trigger function like that in Figure 16. As the exact search in this function is dependent on the name and layout of the specific molecules table, do not put this in the public schema. Then attach a

trigger like Figure 17 to that molecule table. Every new molecule will now be compared to those already in the table and rejected if it is a duplicate.

21 Tuning

- Use Barsoi if you can (Standard since pgchem::tigress version 7.0).
- Use PostgeSQL 8.1.x or better, because its performance enhancements really speed up the fingerprint screening stage.
- If speed is paramount to precision for substructure searching, use *match_substruct_smarts()* instead of *match_substruct()*.
- Encapsulate searches in stored procedures to keep them close to the database. A full example can be found in the examples/searching directory: chembank_find_matching_molecules.sql.
- Combine molecular keys and fingerprints for screening.
- Avoid on-the-fly similarity calculation for searching.
- Frequently update the statistics on the tables.
- Configure PostgreSQL correctly for your type and size of application.
- Query the database as precise as possible, especially for substructure searches, e.g. avoid to search for Benzene or Naphthalene as substructures without further constraints. An application can use a screening only search to guess the worst case return scenario and ask the user, e.g. *'This search will probably yield 23432 rows (87%) of the database. Do you really want to continue?'*.
- Use the LIMIT option for PostgreSQL queries if you want to limit the number of hits returned. LIMIT kills the entire query at once when the specified result set limit has been reached, effectively reducing the workload for high-yield queries.
- Always keep in mind, that the database optimizer has no information whatsoever about the selectivity of pgchem::tigress's functions, except, that he has no information whatsoever about the selectivity of pgchem::tigress's functions. That means it will probably apply them after all known result set reduction mechanisms have been applied, which is good, but may fail to do so. So take an occasional look at EXPLAIN and the query plan.
- Pgchem::tigress is generally a bit faster on UN*X than on Win32.

22 Security

- Validating all data in an application on input and output is always a good idea.
- pgchem::tigress and barsoi have be hardened against classical buffer overflows, but may not be immune.

23 Limitations

- Stereochemistry is absolute.
- If R/S geometry checking is on together with strict adherence to atom and bond types, 'squiggle' bonds are treated as flat bonds, i.e. rejected.
- The double bond stereo query feature (MDL: bond stereo dblEither) for E/Z stereochemistry is experimentally supported.
- Strict adherence to atom and bond types does not always work as expected. Handle this option with care.
- MDL NoStructs are very weakly supported. Avoid them if you can at any cost. The concept of a special non-structure is grotesque anyway when you can use NULL.

24 Links & Things

- checkmol/matchmol: http://merian.pch.univie.ac.at/~nhaider/cheminf/cmmm.html
- OpenBabel: http://openbabel.sourceforge.net/
- PostgreSQL: http://www.postgresql.org/
- pgchem::tigress + barsoi: http://pgfoundry.org/projects/pgchem/

A The OpenBabel Type2 molecular fingerprint format

Since the Type2 fingerprint works like a Daylight fingerprint, see http://www.daylight.com/dayhtml/doc/theory/theory.finger.html#RTFToC80 for how it works. The unfolded Type2 fingerprint used in pgchem::tigress is 1024 bits wide.

B The molecular keys format

```
CREATE OR REPLACE FUNCTION example.t_is_molecule_unique()
  RETURNS "trigger" AS
$BODY$
DECLARE is_not_unique bool;
DECLARE mol_fp text;
BEGIN

is_not_unique:=false;

IF TG_OP='INSERT' OR TG_OP='UPDATE' THEN

mol_fp:=ms_fingerprint_long(NEW.molecule,false,false,false);

mol_fp:=replace(mol_fp,':','=');
mol_fp:=replace(mol_fp,';',' AND ');

is_not_unique := EXECUTE 'EXISTS (SELECT iiid FROM
example.molecules WHERE' || mol_fp ||
'match_exact(decode(''''||encode(NEW.molecule,
'hex')||''',''hex''),molecule,FALSE,FALSE,FALSE,FALSE,FALSE,
FALSE)=TRUE';

  IF is_not_unique THEN  RAISE EXCEPTION 'MOLECULE IS NOT
  UNIQUE ON INSERT OR UPDATE!'; END IF;

ELSE

  RAISE EXCEPTION 'PGCHEM IS-MOLECULE-UNIQUE TRIGGER CALLED
  OUTSIDE INSERT OR UPDATE!';

END IF;
RETURN NEW;
END;
$BODY$
LANGUAGE 'plpgsql' VOLATILE;
```

Figure 16: The unique molecules trigger function

```
CREATE TRIGGER is_molecule_unique
  BEFORE INSERT OR UPDATE
  ON example.molecules
  FOR EACH ROW
  EXECUTE PROCEDURE example.t_is_molecule_unique();
```

Figure 17: The unique molecules trigger

Field(s)	**Descriptor**
ntoms, n_bonds, n_rings	number of atoms, bonds, rings
n_QA, n_QB, n_chg	number of query atoms, query bonds, charges
n_C1, n_C2, n_C	number of sp, sp2 hybridized, and total no. of carbons
n_CHB1p, n_CHB2p, n_CHB3p, n_CHB4	number of C atoms with at least 1, 2, 3 hetero bonds
n_O2, n_O3	number of sp2 and sp3 oxygens
n_N1, n_N2, n_N3	number of sp, sp2, and sp3 nitrogens
n_S, n_SeTe	number of sulfur atoms and selenium/tellurium atoms
n_F, n_Cl, n_Br, n_I	number of fluorine, chlorine, bromine, iodine atoms
n_P, n_B	number of phosphorus and boron atoms
n_Met, n_X	number of metal and "other" atoms (not listed elsewhere)
n_b1, n_b2, n_b3, n_bar	number single, double, triple, and aromatic bonds
n_C1O, n_C2O, n_CN, n_XY	number of C-O single bonds, C=O double bonds, CN bonds (any type), hetero/hetero bonds
n_r3, n_r4, n_r5, n_r6, n_r7, n_r8	number of 3-, 4-, 5-, 6-, 7-, and 8-membered rings
n_r9, n_r10, n_r11, n_r12, n_r13p	number of 9-, 10-, 11-, 12-, and 13plus-membered rings
n_rN, n_rN1, n_rN2, n_rN3p	number of rings containing N (any number), 1 N, 2 N, and 3 N or more
n_rO, n_rO1, n_rO2p	number of rings containing O (any number), 1 O, and 2 O or more
n_rS, n_rX, n_rAr, n_rBz	number of rings containing S (any number), any heteroatom (any number), number of aromatic rings, number of benzene rings
n_br2p	number of bonds belonging to more than one ring
n_psg01, n_psg02, n_psg13, n_psg14	number of atoms belonging to elements of group 1, 2, etc.
n_psg15, n_psg16, n_psg17, n_psg18	number of atoms belonging to elements of group 15, 16, etc.
n_pstm, n_psla	number of transition metals, lanthanides/actinides
n_iso, n_rad	number of isotopes, radicals

Table 1: The molecular keys format, all descriptors are integers