Import of TRANSFAC JSON files into a PostgreSQL database

This document describes a quick way to upload TRANSFAC data in JSON format into a PostgreSQL database, making use of the JSON and JSONB data types in PostgreSQL. While the JSONB type allows faster performance, it has a size limit of 255MB, so for the gene.json file upload the JSON data type is used. The procedure was tested with PostgreSQL version 11.8.

In e.g. psql as user postgres, create a new database with a name and owning user of your choosing:

```
create database DBNAME owner = "USER";
```

```
create schema transfac;
```

Copy the upload SQL scripts contained in the sql folder into the directory where the JSON files are and execute them:

```
psql -v filename="matrix.json" -d DBNAME -f import_matrix.sql

psql -v filename="factor_wo_fragments.json" -d DBNAME -f import_factor.sql

psql -v filename="site.json" -d DBNAME -f import_site.sql

psql -v filename="gene.json" -d DBNAME -f import_gene.sql
```

Each of the scripts will create a database table plus some indexes and will import the data:

import_matrix.sql

```
begin;

\lo_import :filename
\set obj :LASTOID

create table transfac.matrix as
  select *
    from jsonb_to_recordset(convert_from(lo_get(:'obj'),'UTF8')::jsonb)
           as r(accession text,
                identifier text,
                name text,
                type text,
                copyright text,
                description text,
                comments jsonb,
                organism text,
```

```
                "taxClass" text,
                "matrixClass" jsonb,
                "bindingFactors" jsonb,
                "bindingSites" jsonb,
                profiles jsonb,
                "statisticalBasis" jsonb,
                tfmatrix jsonb,
                superfamilies jsonb,
                subfamilies jsonb,
                "preferredVersion" jsonb,
                "oldVersions" jsonb,
                "relatedSpecificMatrices" jsonb,
                "dbReferences" jsonb,
                "references" jsonb,
                "secondaryAccessions" jsonb,
                "secondaryIdentifiers" jsonb
                );

create index ix_mx_accession on transfac.matrix (accession);
create index ix_mx_identifier on transfac.matrix (identifier);
create index ix_mx_name on transfac.matrix (name);
create index ix_mx_bsites on transfac.matrix using gin ("bindingSites"
jsonb_path_ops);
create index ix_mx_bfactors on transfac.matrix using gin ("bindingFactors"
jsonb_path_ops);

\lo_unlink :obj
commit;
```

The table column names and data types match the ones given in the documentation for each JSON file.

The JSONB and JSON data type fields can be queried with SQL using a specific syntax respectively. See also https://www.postgresql.org/docs/11/functions-json.html.

Examples

In case a JSONB column contains an object, the operator ->> retrieves a JSON field as text. (Please note that column names with upper case characters need quotation):

```
select accession, "matrixClass"->>'family'
from matrix
where "matrixClass"->>'family' is not null;
```

If a JSONB column contains an array, the function *jsonb_array_elements* is helpful:

```
select s.identifier as site_id,
jsonb_array_elements("bindingFactors")->>'acc' as factor_acc,
jsonb_array_elements("bindingFactors")->>'name' as factor_name,
jsonb_array_elements("bindingFactors")->>'species' as species,
jsonb_array_elements("bindingFactors")->>'effect' as effect
from site s
order by s.identifier;
```

Table joins involving JSONB columns are possible:

```
select distinct m.accession as matrix_acc, s.accession, s.identifier
from matrix m
join jsonb_array_elements("bindingSites") as r(data) on true
join site s on r.data->>'acc' = s.accession;
```

Alternatively, the schema can be extended with linking tables, e.g.:

```
create table transfac.matrix2site as
select m.accession as matrix_acc,
jsonb_array_elements("bindingSites")->>'acc' as site_acc
from matrix m;
```

Here's an example for querying the JSON data type in the gene table. It retrieves all genes with the transcripton factors (or miRNAs) that regulate them and the DNA binding sites (or mapped mRNA binding sites) that are involved. It also illustrates how to access nested json data:

```
select gf.accession, "shortDesc", json_array_elements(factor)->>'acc' as
factor_acc,
json_array_elements(factor)->>'name' as factor_name,
json_array_elements(factor)->>'effect' as effect, gf.site_acc,
s.sequence ->>'chr' as chromosome,
s.sequence ->>'chrStart' as site_start,
s.sequence ->>'chrEnd' as site_end,
s.sequence ->>'chrStrand' as strand
from
(select accession, "shortDesc", json_array_elements("bindingSites")-
>>'acc' as site_acc,
json_array_elements("bindingSites")->'bindingFactors' as factor
from gene) as gf
join site s on gf.site_acc = s.accession;
```