1. DATABASE

Grape interacts heavily with a database. Users, sessions and other data is stored in the database. Functions are defined here, and most business logic happen in the database. The following subdirectories can be found in a project's db/ directory.

- *schema/* - The database schema file (**.dbm file** - created in pgModeler, and exported DDL to SQL files).
- *function/* - Directories and files containing database functions.
- *process/* - Files containing database functions related to background processes.
- *data/* - Files containing initial data for the system.
- *deployments/* - Containing subdirectories for specific deployments. See the section Deployments for more information regarding this.

1.1 Defining the database structure

1.1.1 Database model

pgModeler is a database modelling tool specifically designed for PostgreSQL. The model are saved in a DBM (*.dbm) file, and exported to a *.sql file. Usually, both *.dbm* files and the corresponding *.sql* files are added to the repositories. The sql files are always generated by pgModeler and should never be edited manually. The DBM file is saved in db/schema/PROJECTNAME.dbm. From within pgModeler, the SQL should be exported to a file db/schema/PROJECTNAME.sql. When the database is created, this file will be loaded.

1.1.2 Database functions

Database functions (like API call handlers or business logic) usually resides in db/function/. Files should be properly named, and functions grouped together in files, in a way that is understandable to someone who is not familiar with the code. File names should be all lowercase, and words split by a underscore. For example, *file_name.sql*.

1.1.3 Initial data

Initial data for the system usually resides in db/data/. These sql functions includes data such as user/access roles, settings and whitelist tables specific to the system.

1.1.4 Indexes

> **Note!** Remember that primary key columns automatically have indexes assigned to them.

Always make indexes on the following columns:

- Any column that is referenced by a foreign key.
- Any column that references a foreign key.
- Any column that is used often (in the WHERE clause of queries).

1.1.5 Naming conventions

- Keep table names lowercase, with words separated by a underscore. For example, *table_name*.
- If a relation (table) has an id/unique column (which should be the case in most tables), add '_id' to the table name. For example, table policy has an ID column called *policy_id*.

> **Note!** There are security risks involved in using an auto-incrementing column where the value of such a column is exposed to clients, or untrusted personnel.

- Use a 'v_' prefix for views. For example, ***v_policy***
- Use a 'mv_' prefix for materialized views. For example, ***mv_policy_address***
- Use a '_idx' suffix for indexes. For example, ***policy_id_idx***
- Use a '_fk' suffix for foreign key constraints, with an abbreviation of the tablename as prefix. For example, ***po_policy_id_fk***

1.1.6 Pitfalls to avoid

- Avoid multi-column indexes (and primary keys) unless it is really suitable for the situation.
- Avoid too many indexes on a table.

1.1.7 Schemas

By default, grape applications will use the following schemas:

- public
- grape
- proc

1.2 Using NOTIFY/LISTEN

Grape includes a utility function, ***new_notify_handler(channel_name, callback)***, which will keep track of callbacks for database NOTIFY events.

1.3 grape-db-setup

grape-db-setup is a tool used to load SQL files into a database. This is typically done during initial creation of a database, or when applying patches to a database. ***grape-db-setup*** accepts the following options:

- ***-d, --dburi [dburi]*** Connection parameters for the target database. If dburi is not specified, the generated SQL will be printed to stdout.
- ***-s, --superdburi [superdburi]*** Connection parameters that will be used when creating and dropping the database.
- ***-c, --create*** Create the database before attempting to create objects.
- ***-r, --drop*** Drop and recreate the database before attempting to create objects.
- ***-i, --continue*** Continue processing when an error occurs (by default, processing will stop).
- ***-e, --schema*** The default schema to use when creating objects (defaults to "public"). If "none" is specified, search_path will not be set.

In addition to the above mentioned options, one or more files/directories should be provided. Each of these additional arguments will be processed:

- If the entry is a directory, the files in this directory will be recursively processed and **.sql** files be included in the output.
- If the entry is a **.sql** file, the file will be included in the output.
- If the entry is a **.manifest** file, the file will be read and lines will be processed accordingly.

1.3.1 File order

grape-db-setup loads SQL files alphabetically. Sometimes it might be necessary to explicitly change this load order. This can be done by adding a *-- **Require:*** pre-processing instruction in the SQL file. The file that is required, will be loaded before the file containing the instruction. For example, having the following in an SQL file called file.sql, will force the loading of other_file.sql before file.sql.
SQL files can contain a:

```
-- Require: other_file.sql
```

1.3.2 Manifest files

Manifest files contains a list of files (sql and other manifest files) and directories. If a manifest file is provided to ***grape-db-setup***, each line will be processed. Manifest files can contain comments using the # character.

```
# Load grape files
../node_modules/ps-grape/db/initial.manifest

db/function/
```

Most projects will include 2 manifest files usually:

- ***initial.manifest*** - To be used during initial creation of database. This file will create the database schema and tables (as exported from pgModeler).
- ***functions.manifest*** - To be used when recreating functions. This file will load all PL/pgSQL functions and views.

Additionally, deployments can include manifest files in order to apply changes to target live databases.