This manual is under development

The PostgreSQL switch

WebHare is switching from its builtin database server ('dbserver') to PostgreSQL. Experimental PostgreSQL support will be available in version 4.27.

You can already start preparing for the switchover to make the switch as painless as possible (even though WebHare will keep support for both database servers for a while). Here's what you should do:

Stay up-to-date!

WebHare 4.26 is an *unskippable upgrade* so you should work towards updating all your servers to this version.

PostgreSQL will be a lot stricter than dbserver about text in the database and will generally reject invalid UTF-8 data unless a column is explicitly declared as binary. WebHare will try to avoid declaring binary columns as much as possible as invalid UTF-8 data is painful to work with.. and if we can avoid inserting it into the database and troublesome data will be stopped at insertion.

You should scan your installation for invalid UTF-8 data and fix it. Often you can just strip out incorrect data and reinsert it - if it was meant to be text, it was probably already corrupted anyway.

One source of invalid UTF-8 data are old hashed passwords which contained raw binary data. These may be present in WRD data, and you should scan your existing schemas using 'wh wrd:scannonutf8' (added in 4.26)

Watch the warnings!

WebHare 4.27 adds a lot of deprecations for dbserver-specific code. Make sure you 'wh checkmodule' all your modules to find these warnings. Even better: set up continuous integration to keep your modules tested against the development master for early deprecation warning.

Migration

Before you start migrating, you should:

- Ensure all your modules are up to date, especially if they use custom tables (schema definitions must properly mark NULL and BINARY columns)
- If you manually dropped tables before conversion, softreset first to make sure all rights tables are in sync

Dry run

We recommed doing a dry-run of the migration first. A dryrun will generate the postgres tables but not move then into place for a switchover:

```
wh convert-to-postgresql --dryrun
```

Any records with invalid UTF-8 data will be printed.You may consider whether you can fix them or just accept a best-effort ISO-8859-15 -> UTF-8 for these fields. If some fields are still incorrect even after this fix, migration will fail.

Any converted cells will have their original values saved
to webhare_internal.postgresql_migration_issues after the final migration.

Running the conversion

To actually do the conversion, remove the --dryrun parameter. This will prepare the postgres tables and WebHare will switch over to postgres database once it's restarted. If you're using docker and want to automatically restart after the conversion is done:

```
wh db setserver readonly && wh convert-to-postgresql && sv restart webhare
```

The time migration takes may vary (do a dry-run to have an estimate) and has taken anywhere between 5 seconds and 15 minutes for our own servers - database size and I/O speed matters a lot of course.

We recommend switching to readonly mode during migration, and accepting that this may break some webpages or applications which try to write to the database. If you do not disable database writes, you will lose any changes that were made since the start of the migration.

You can always revert back to the old database engine by deleting the /opt/whdata/postgresql directory and restarting WebHare, but you will lose all changes made since the migration. If you are satisfied with the results of your migration and have verified your backup/restore procedures, you can delete /opt/whdata/dbase.

PostgreSQL implementation details

Blobs

Blobs are stored in files in folders outside of the database. Internally, they are stored as the composite type `webhare_blob`, which consists of a text id identifying the storage location, and an int8 length with the length of the blob. All these blob references are stored in the table `webhare_internal.blobs`. Blob columns in normal tables are created with a foreign reference to that table. In HareScript, creation of blob files and registration in the blob table is taken care of automatically.

References to the blobs are stored as a webhare_blob type (CREATE TYPE webhare_blob AS (id text, size int8)). When using psq you would select '(columnname).id' and '(columname).size', eg:

```
# Get the biggest file from_fsobjects
SELECT id, (data).size AS len FROM system.fs_objects
      ORDER BY (data).size DESC NULLS LAST LIMIT 1;
```

Mapping of NULLs

HareScript doesn't have the concept NULL baked into the language. To store these values, a distinction is made between value columns and reference columns. For value columns, NULL is never used. All values are stored as-is. For reference columns (columns that are a foreign reference to another table) NULL is stored for the default value of the keys. These must be annotated in the HareScript table definitions with `NULL := <value>`.

UTF-8

PostgreSQL enforces that only UTF-8 valid data can be stored in VARCHAR columns (which are used to store strings in WebHare). If string data needs to be stored that is not valid UTF-8, a BYTEA column is used. This is done when the column is marked binary in the schemadefinition. Also, the HareScript column needs to be annotated with `__ATTRIBUTE__(BINARY)`.

Prepared statements

If a SELECT or INSERT with the same query and argument types is repeated a number of times, a prepared statement is created for that query, and used to speed up execution of the statement.

Transaction management

The WebHare PostgreSQL database is configured to have a read-only session by default. The BeginWork calls will open a new, writable transaction. This can be a problem when connecting manually to the PostgreSQL database, use the statement `SET SESSION CHARACTERISTICS AS TRANSACTION READ WRITE` to enable auto-opening of writable transactions.

Isolation

The legacy database server did use a isolation mode roughly equivalent to the 'READ COMMITTED' isolation of PostgreSQL, and without the row-level locking, and some leaking of

the effects of other transactions. Transactions for PostgreSQL now use the isolation level 'READ COMMITTED', where the individual statements in a transactions can see the effects of transactions that were committed between those statements. Make sure this doesn't affect the correctness of your code!

Read-only mode

The database supports a read-only mode which can be used during upgrades or for failover/verification.

Read-only mode is separate from restore mode - restore mode generally disables tasks that have external effects or trigger modifications, but does not block updates to the database itself.

Read-only mode works at the HareScript level and is checked at work begin and commit. Direct connections to the database can still modify it.

To enter read-only mode: `wh db setserver readonly`

To exit read-only mode: `wh db setserver readwrite`

Effects of readonly mode

`GetPrimary()->BeginWork()` will throw immediately. You can check %IsDatabaseReadonly to help prevent this situation, but to handle this in a race-free manner you need to properly handle the %DatabaseReadonlyException

`ScreenBase::BeginWork` will work, but Finish will report that the database is in readonly mode and fail (Similar to deadlock/unique constraint violations that aren't caught until Finish)

Upgrading Posgresql

To upgrade the WebHare database on macOS from 12 to 13:

```
cd $WEBHARE_DATAROOT  # see 'wh dirs' fo the proper value if needed
mv postgresql/db postgresql/db.bak
brew install postgresql@12
initdb -D postgresql/db --auth-local=trust -E 'UTF-8' --locale='en_US.UTF-8'
pg_upgrade -b /usr/local/Cellar/postgresql@12/12.4/bin/ -d postgresql/db.bak -D postgresql/db
```

If all seems fine, you can delete $WHDATA/postgresql/db.bak at some point