SPSync

Migration planner (PDF)

A planner for moving list data off SharePoint into a database, from list inventory and column audit to the config, cut-over and reconcile.

Your contract register is moving off SharePoint into a purpose-built application, and the project plan has one line for the data: "migrate lists". That line hides most of the risk. Which lists, which columns, what happens to lookups, and how do you prove on cut-over day that the database matches what people were still editing that morning?

This planner covers moving list data into a database with SPSync, which reads SharePoint and never writes to it. The live list stays untouched until you retire it. Moving whole sites between tenants is a job for a tenant migration tool; SPSync's part there is the before and after copies described in keeping a copy through tenant migrations.

List inventory for each site

Start with every list in scope and one row per list. On the command line, spsync discover lists --site-url <url> prints each list's name, display name, web URL, whether it is a document library, whether it is hidden, and its id. Take the item count from Site contents in SharePoint, because the planner needs it for scheduling the first load.

Site | List | Items | Library? | Lookups into | Owner | In scope? | Target table
contracts | Contracts | No | Counterparties | Yes | ct_contracts
contracts | Counterparties | No | Yes | ct_counterparties
contracts | Renewals | No | Contracts | Yes | ct_renewals

Fill the "Lookups into" column carefully. A lookup is stored as {col}_id and {col}_value, and the id only joins to something if the target list is replicated too. Leave a lookup target out of scope and the new application gets an id with nothing to join it to. For a tenant with many sites, spsync discover sites --all --filter <text> finds them by name.

Column audit for each list

Run spsync discover schema <site-id> <list> for every list in scope, taking the site id from spsync discover sites --site-url <url>. It prints each column's internal name, display name, type, and whether it is hidden or required. Internal names become destination column names, so a column displayed as "Due Date" with the internal name Due_x0020_Date arrives as due_date. Mark each column against this table:

Column type | Where it goes | Action before cut-over
Text, number, Yes/No, choice | One column on the list table | None

Lookup | {col}_id and {col}_value | Check the target list is in scope

Person | {col}_user_id, a key into the shared sp_users table | None

Multi-choice, multi-lookup, multi-person | A child table, {prefix}_{list}__{col} | Tell the application team it is one row per value

Date only | A date column | Set the site time zone, or a site east of UTC gets dates a day early

Calculated | The computed value, never the formula | Rebuild the formula in the application if it still matters

Managed metadata | The term GUID and label, for a column that has values | Tell the application team the term hierarchy is not carried

Document library files | File contents are not replicated | Plan the files separately

The term hierarchy and library file rows are the ones that most often surprise a project late. Settle them in week one, while there is still time to plan another route.

Writing the replication config

One export-config call turns the inventory into a YAML file. Commit it and review it like code:

```
spsync export-config --output contracts.yaml \
  --site-url https://contoso.sharepoint.com/sites/contracts \
  --list Contracts --list Counterparties --list Renewals \
  --pipeline-name contracts --prefix ct \
  --destination-kind postgres --connection-string-env CONTRACTS_DB_URL \
  --dataset-name contracts --site-time-zone Europe/London \
  --create-reporting-views --client-id <client-id>
spsync doctor contracts.yaml
```

The connection string is an environment variable name, so the file holds no secret. spsync doctor checks the sign-in, the Graph access to each list and the destination before anything runs. For several sites, repeat --site-url with --lists-by-name, or use spsync batch create with one file covering them all.

First load and parallel running

spsync run contracts.yaml does the first full load, and every later run is a delta. Compare row counts in each table with the item counts in your inventory, then schedule the run, every five minutes by default, and leave it going while the application is built. People keep editing the list for weeks, and the database keeps up. Deleted items stay as rows with sp_deleted_at set, so nothing removed during the project disappears from the record.

Cut-over day and the reconcile

Step | Command or action | Check

1 | Freeze edits on the list, or agree a cut-off time | Owners told

2 | spsync reconcile contracts.yaml | Run completes with no errors

3 | Compare live row counts with SharePoint | Rows where sp_deleted_at is null match the item count

4 | Point the application at the tables or the v_ct_* views | Smoke test passes

5 | Keep the schedule running, or stop it | Decision recorded

The reconcile re-checks every item against SharePoint, so it is the QA step that lets you sign off the copy. It also runs weekly by default on the schedule, which covers recovery after an outage while the two systems run side by side. Keep the config in git after cut-over. If the migration has to happen again in another environment, it is one command.

Script a SharePoint list migration into a database has more on running this from CI. Migrating list data off SharePoint into a database covers the plan from the business side.

Full, up to date version: https://spsync.io/resources/sharepoint-migration-planner/