SPSync

Warehouse modelling guide (PDF)

Using SPSync's normalised SharePoint schema as a dbt source, with keys, soft deletes, the generated v_ views and a star schema on BigQuery or Databricks.

Cost centres, project codes and approval states live in SharePoint lists, and the analytics engineer has been asked to get them into the warehouse model next to the ERP data. The first attempt is usually a flattened export, with lookups as text and multi-choice values packed into one cell, and every staging model then spends its first fifty lines undoing that. SPSync replicates the lists as a normalised schema, which is a better starting point for dbt and needs different handling in a few places.

What the replicated schema looks like

Object | Name | Primary key | Notes
List table | {prefix}_{list} | sp_item_id | One row per item, deleted items included
Child table | {prefix}_{list}__{col} | sp_item_id, ordinal | One row per value of a multi-value column
Users | sp_users | id | Shared by every list and site
Reporting view | v_{prefix}_{list} | Live rows, people joined to names

Every list table carries metadata columns including sp_item_id, sp_list_id, sp_site_id, sp_created, sp_modified, sp_author_id, sp_editor_id, sp_deleted_at and sp_synced_at. A lookup becomes {col}_id plus {col}_value, and a person column becomes {col}_user_id pointing at sp_users.id. Those are logical keys; SPSync does not declare foreign key constraints, so add dbt relationships tests where you rely on them.

Declaring the tables as dbt sources

Declare each table in sources.yml. Only the list tables carry sp_synced_at. The child tables and sp_users have no such column, so put the freshness check on each list table and leave it off the others, or dbt source freshness fails on them:

```
version: 2
sources:
  - name: sharepoint
    schema: reporting
    tables:
      - name: sp_projects
        loaded_at_field: sp_synced_at
        freshness:
          warn_after: {count: 6, period: hour}
      - name: sp_projects__stakeholders
      - name: sp_departments
      - name: sp_users
```

SPSync sets sp_synced_at when it writes a row, and a sync round that finds no changes writes nothing. On a quiet list the newest sp_synced_at is the last time someone edited an item, which on a reference list such as departments can be weeks ago, and that is expected. So a stale timestamp does not tell you the schedule has stopped. Alert on the replication run itself, through the CLI's exit status in your scheduler or the run history in the desktop app, and keep freshness as a sanity check on busy lists like projects, with a threshold set from how often people edit them.

Keys: natural, surrogate and per site

sp_item_id is SharePoint's item id. It is unique within one list, which is why each list gets its own table, and that is enough for most models. Two cases need more care.

Situation | Key to use

One site, one table per list | sp_item_id

Several sites merged into one table | sp_site_id plus sp_item_id; SPSync adds sp_site_id to the primary key itself

A dimension shared across lists or sources | A surrogate key hashed from sp_list_id and sp_item_id in a staging model

People | sp_users.id, already stable

sp_users.id is a UUID derived from the person's login, so the same person gets the same id in every list and on every run. It can be the people dimension's key as it stands. For lists, generate the surrogate key in staging with your usual macro, such as dbt_utils.generate_surrogate_key, so the warehouse key does not depend on a SharePoint id that means nothing elsewhere.

Soft deletes in staging models

When an item is deleted in SharePoint, SPSync keeps the row and sets sp_deleted_at. That gives you a choice the source system never offered. Most staging models filter it out:

```
select *
from {{ source('sharepoint', 'sp_projects') }}
where sp_deleted_at is null
```

A snapshot or audit model can keep deleted rows and use sp_deleted_at as the end date. Child tables have no sp_deleted_at of their own, so join them to their parent row and filter there. Child rows always mirror the item's current values; when someone unticks a choice, its row goes on the next run.

The generated v_ views

With reporting views switched on, SPSync creates v_{prefix}_{list} for each list. The view leaves out deleted rows and joins each person column to sp_users to add display name and login columns. Lookup display values are already on the row as {col}_value. Multi-value columns are not in the view, because a one-to-many join has no single right shape, so read the child tables for those.

The views make a good source for a quick report. For a governed model, read the base tables, which carry every column and the deleted rows. Views are available on BigQuery, Databricks, ClickHouse, SQL Server, Postgres, MySQL, Oracle, SQLite and DuckDB.

A star schema from lists

SharePoint object | Warehouse role | Example
A list of events or transactions | Fact | Tasks, Renewals, Incidents
A list that other lists look up | Dimension | Departments, Counterparties
sp_users | Dimension | People, with is_group for groups
A child table | Bridge | Project to stakeholder
sp_created, sp_modified, date columns | Role-playing date keys | Created date, due date

The lookup {col}_id on the fact joins to the dimension's sp_item_id. Replicate every lookup target list, or the fact carries ids with nothing behind them.

When SharePoint columns change

A new column appears on the next run and a removed one is never dropped, so existing models keep compiling. When a column changes type, SQL Server, PostgreSQL, MySQL, ClickHouse, Oracle, SQLite and DuckDB write the new values to {col}_v2 and leave the original as it was. BigQuery and Databricks do not fork a retyped column today, so on those warehouses agree with list owners that column types stay fixed.

dbt models over replicated SharePoint tables covers running dbt after the sync. The schema reference lists every column type.

Full, up to date version: https://spsync.io/resources/warehouse-modelling-guide/