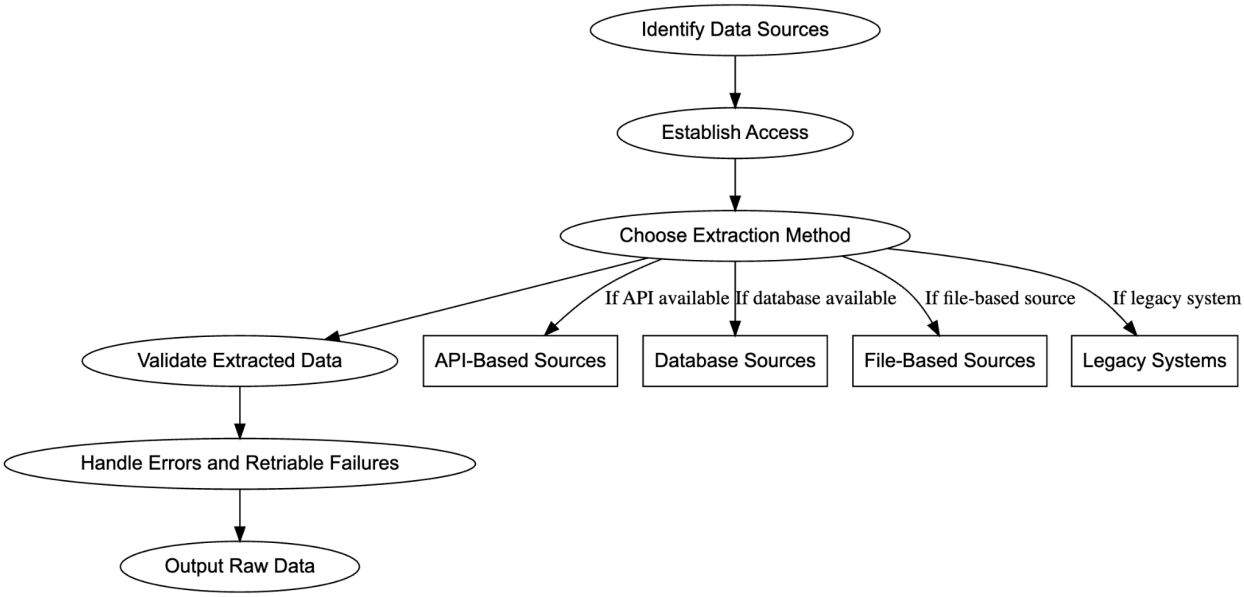
Multi-Entity Vendor Data Consolidation: Technical Document

This document outlines the approach for extracting, cleaning, mapping, deduplicating, and importing vendor data into Light's multi-entity system. The solution is designed to ensure data consistency, accuracy, and seamless integration.

Data Extraction



Identify data Sources: Initially I need to identify the data sources. For example; ERP systems (e.g., SAP, Oracle), Accounting platforms (e.g., QuickBooks, Xero), CRM tools (e.g., Salesforce), Databases (e.g., MySQL, PostgreSQL), Files (e.g., CSV, Excel, JSON)

I'd have to then determine the methods I can use to access each source, for example some sources may only allow for file download/exports, whereas some may allow for APIs and database connections.

- Different environments (on-premise vs. cloud) might affect the extraction approach, for example. For on-premise systems like SAP, I might rely on custom connectors or middleware, whereas cloud systems like QuickBooks or Xero typically offer REST APIs.

Access: I'd then have to establish our access and gain any authentication I require, for APIs, I'd need to obtain API keys, OAuth tokens, or user credentials. For Databases, I'd need to secure database credentials and whitelist IPs if needed. For File based sources, I'd need access to file storage servers so I can export our data.

- Verify permissions to access vendor data.
- Confirm with stakeholders which data fields are required.

Choose Extraction Method:

API: Use REST or GraphQL API's to query our vendor data, Authenticate with OAuth2, API keys or tokens, use filters/pagination for data retrieval.

File based sources: I can use automated scripts to download, validate, and parse files in the required format (CSV, JSON), I can use libraries to parse files, such as Pandas for Python.

```
import pandas as pd

df = pd.read_csv('vendors.csv')
```

Data Mapping

Analyse Data Sources and Target Data Structures: Identify the fields available in each source, such as: Vendor Information: Name, ID, address, email, phone. Financial Data: Account balances, outstanding invoices. Transaction Data: Payment terms, history, and credit limits.

I'd also review Light's database schema to understand its entity relationships and mandatory fields. For example; Vendor Table: Unique vendor_id, name, contact_info. Entity Relationships: Parent-child hierarchy for multi-entity support.

Identify Common Fields for Standardisation: I'd make sure to standardise fields to ensure uniformity when I eventually consolidate all of my data from all platforms/across entities.

At this step it is important to establish uniformity, here I standardise our information. See example table below:

Field	Standardisation Suggestions
vendor_id	Use a UUID or external reference key to maintain consistency
Contact Information	Standardise phone numbers (e.g., E.164 format) and email addresses (lowecase, no whitespace)
Dates	Use ISO 8601 format (YYYY-MM-DD) for fields like creation and modification dates
Financial Data	Standardise currencies and amounts to Light's base currency using exchange rates if necessary
Address Format	Split addresses into fields (e.g., street, city, state, zip_code) to match Light's schema

Map Source Fields to Target Structure: Create a mapping file that can be cross referenced upon data uploads detailing how each source field aligns with Light's structure. For example:

Source System	Source Field	Light Field
QuickBooks	Vendor Name	vendor_name
Xero	Contact Name	vendor_name
QuickBooks	Email Address	email_address

Xero	Email	email_address

Address Discrepancies Between Systems

Field Mismatches: QuickBooks may use Vendor Type, while Xero lacks this field. Use default values or derive the type based on other fields.

Data Granularity: QuickBooks might have detailed transaction records, while Xero provides aggregated data. Apply transformations to align with Light's schema.

Duplicate Data: Vendors recorded differently across systems (e.g., "ACME Inc" vs. "ACME Corporation"). Use fuzzy matching or a unique external ID to deduplicate.

Hierarchical Data: Parent-child relationships (e.g., vendors linked to multiple entities in Light's structure) might not exist in source systems. Establish relationships during transformation.

Missing Data: Fill gaps using default values, placeholders, or cross-referencing from another source.

To complete mapping, you can write scripts in Python or SQL to process and map data. For this you can take multiple data entries as an input, collate them into a table of mapping, that I can manually maintain if required (as some fields may not correspond with a field from another source).

De-duplication and Consolidation

Here, I utilise the mapping system that I have created, to cross reference and compare records across entities and use this to create a master sheet of our input data, with reference to our mapping table. I can then identify duplicates across all of our database based on:

- Unique Identifiers: Vendor IDs or external references (if they are consistent across systems).
- Fuzzy Matching: For names and contact information, I can use algorithms like Levenshtein distance or Jaro-Winkler to detect near-matches (e.g., "ACME Inc" vs. "ACME Incorporated").
- Key Fields: Match records on email addresses, tax IDs, or phone numbers, as these are often unique.

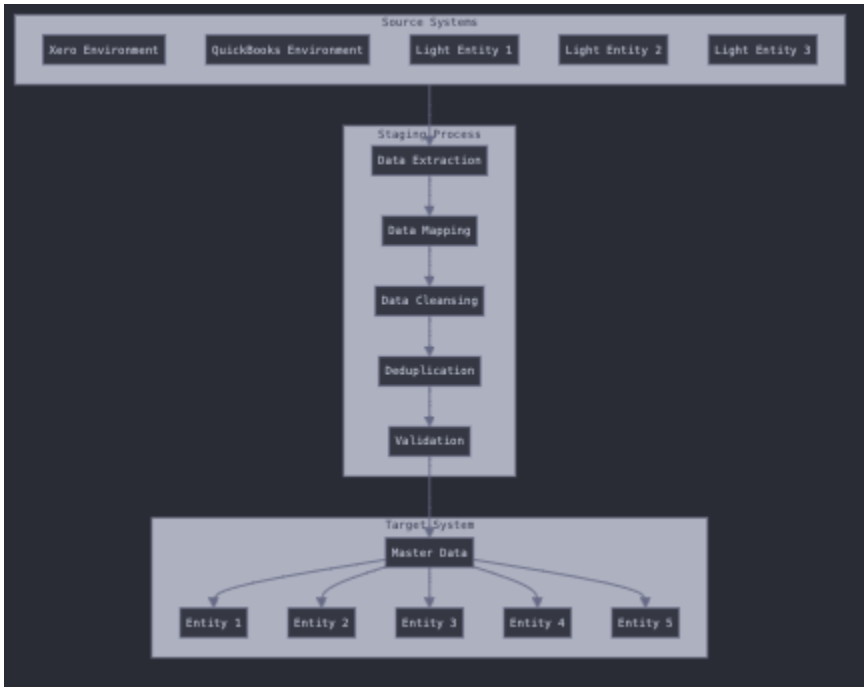
I must also define set rules for consolidation, in order to do this, it makes sense to use our master sheet collated from our cross referencing. I have to also carry out Field Prioritisation: For fields like address, phone, or email, decide which source system takes precedence, or merge non-conflicting data. For example, for phone numbers, I'd be likely to use the most recent entry, and for addresses I'd use the complete address if available; otherwise, take the most frequent entry across systems.

Unified record creation/tracking of source systems should also be carried out, in which a field/identifier is added to each source data file when it is extracted as this will allow for better traceability for auditing our data and allows us to quickly identify and reprocess data if discrepancies are found. An example of how I'd set out to do this is below:

Consolidated Vendor ID	Source	Source Vendor ID
123456	QuickBooks	QB_001
123456	Xero	Xero_002

I would also use Python libraries like fuzzywuzzy or RapidFuzz in Python for name matching. This would allow us to automate our deduplication process. You could also use SQL queries for this.

Import Process



First, I'd dive into mapping the data structures from Xero, QuickBooks, and the Light environments. Understanding exactly what we have in each system would let me create a standardised mapping system that works across all entities while preserving their unique configurations.

Next, I'd tackle the data extraction and cleanup. Using tax ID or Vendor ID's as primary keys with fuzzy name matching as backup, I'd identify and merge duplicate vendors across systems. Each vendor would get a unique identifier while maintaining their entity-specific settings like payment terms and credit limits. The goal here is one master record per vendor, but with preserved relationships to each entity they work with.

The import process would be staged - moving master vendor data first, followed by entity configurations. I'd run everything through a test environment initially, with validation checks between each phase. This lets us catch and fix issues before touching the live system.

Finally, I'd validate everything through reconciliation checks, making sure vendor counts match, entity relationships are intact, and all configurations are preserved. Any exceptions or special cases would be documented and handled according to predefined rules.

Validation

I'd write Python scripts to handle three main validation layers and flag where there are discrepancies:

Data Integrity Checks

- Compare total vendor counts between source and target
- Verify all required fields are populated
- Check format consistency (tax IDs, phone numbers, etc.)

Flag any duplicate vendors across entities

- Entity Relationship Testing
- Confirm each vendor maintains correct entity links
- Validate entity-specific settings (payment terms, credit limits)
- Test cross-entity scenarios
- Verify access controls per entity

Financial Validation

- Match historical transaction totals
- Check payment terms are correctly applied
- Verify credit limits carried over
- Test standard vendor transactions in each entity

I'd run these checks first in a staging environment with a sample dataset, fix any issues, then use the same validation suite for the full import. Each validation run would generate a report highlighting any discrepancies for review before proceeding.