AI declaration: After I have finalized the tables I used AI for help with name conventions and relationships between tables. I also generated the DBML data structure and used dbdiagram.io.com to build the data model.

**DBML data structure in dbdiagram.io.com:**

// Use DBML to define your database structure

// Docs: https://dbml.dbdiagram.io/docs

// Use DBML to define your database structure

// Docs: https://dbml.dbdiagram.io/docs

Table member {

  memberId integer [pk, increment]

  memberName varchar(255) [not null]

  memberSize integer

  memberStatus enum('Good Standing', 'Overdue Payment', 'Cancelled') [not null]

  memberAccountsPayableEmail varchar(255) [not null] // Email for accounts payable

  memberStartDate date [not null]

  memberEndDate date

  memberLastContactDate date

  memberNotes text

}

Table address {

  addressId integer [pk, increment]

  memberId integer [ref: > member.memberId]

  addressLine1 varchar(255) [not null]

  addressLine2 varchar(255)

  addressCity varchar(100) [not null]

  addressProvince enum("Alberta", "British Columbia", "Manitoba", "New Brunswick", "Newfoundland and Labrador", "Nova Scotia", "Ontario", "Prince Edward Island", "Quebec", "Saskatchewan")

  addressPostalCode varchar(20)

}

Table industry {

  industryId integer [pk, increment]

  industryName varchar(255) [not null]

  industryNAICSCode varchar(6)

  industryDescription text

}

Table memberIndustry {

  memberId integer [ref: > member.memberId]

  industryId integer [ref: > industry.industryId]

  primary key (memberId, industryId)

}

Table membershipType {

  membershipTypeId integer [pk, increment]

  membershipTypeName enum('Local Industrial', 'Non-Local Industrial', 'In-Kind', 'Government & Education', 'Chamber', 'Associate') [not null]

  membershipTypeDescription text

  membershipTypeFee decimal(10, 2)

  membershipTypeBenefits text

}

Table memberMembershipType {

  memberId integer [ref: > member.memberId]

  membershipTypeId integer [ref: > membershipType.membershipTypeId]

  primary key (memberId, membershipTypeId)

}

Table cancellation {

  cancellationId integer [pk, increment]

  memberId integer [ref: > member.memberId]

  cancellationDate date [not null]

  cancellationReason varchar(255)

  cancellationNotes text

}

Table contact {

  contactId int [pk, increment]

  firstName varchar(50) [not null]

  middleName varchar(50)

  lastName varchar(100) [not null]

  contactTitleRole varchar(100)

  contactPhone varchar(10)

  contactWebsite varchar(100)

  contactInteractions varchar(255)

  contactNotes text

}

Table contactEmail {

  contactEmailId int [pk, increment]

  contactId int  [ref: > contact.contactId]

  emailType enum('VIP', 'Work', 'Personal') [not null]

  emailAddress varchar(255) [not null]

}

Table memberContact {

  memberId integer [ref: > member.memberId]

  contactId integer [ref: > contact.contactId]

  memberContactRelationshipType varchar(100)

  primary key (memberId, contactId)

}

Table opportunity {

  opportunityId integer [pk, increment]

  opportunityName varchar(255) [not null]

  opportunityStatus enum ('Qualification', 'Negotiating', 'Closed-New Member', ' Closed-Not Interested')

  opportunityPriority varchar(50)

  opportunityAction varchar(255)

  opportunityContact varchar(50)

  opportunityAccount varchar(50)

  opportunityLastContactDate date

  opportunityInteractions varchar(255)

}

**Summary of Relationships:**

* **member ↔ address: One-to-many**
* **member ↔ industry: Many-to-many (via memberIndustry table)**
* **member ↔ membershipType: Many-to-many (via memberMembershipType table)**
* **member ↔ cancellation: One-to-many**
* **member ↔ contact: Many-to-many (via memberContact table)**
* **contact ↔ contactEmail: One-to-many**
* **opportunity a table not related to others. Consist of potential future members.**

**Explanation**

1. member and address tables

* Relationship: One-to-many
* Explanation: A member can have multiple addresses (e.g., headquarters, branches, warehouses), but each address is associated with exactly one member.
  + Foreign Key: address.memberId references member.memberId.

2. member and industry tables

* Relationship: Many-to-many
* Explanation: A member can belong to multiple industries, and each industry can have multiple members. This many-to-many relationship is facilitated by the memberIndustry junction table.
  + Foreign Keys: memberIndustry.memberId references member.memberId, and memberIndustry.industryId references industry.industryId.

3. member and membershipType tables

* Relationship: Many-to-many
* Explanation: A member can have multiple membership types, and each membership type can be associated with multiple members. The memberMembershipType junction table handles this many-to-many relationship.
  + Foreign Keys: memberMembershipType.memberId references member.memberId, and memberMembershipType.membershipTypeId references membershipType.membershipTypeId.

4. member and cancellation tables

* Relationship: One-to-many
* Explanation: A member can have multiple cancellations (if a member cancels their membership and then reactivates, this could result in multiple cancellation records). However, each cancellation is associated with one specific member.
  + Foreign Key: cancellation.memberId references member.memberId.

6. member and contact tables

* Relationship: Many-to-many
* Explanation: A member can have multiple contacts (such as primary contacts, billing contacts, etc.), and each contact can represent multiple members. This relationship is managed through the memberContact junction table.
  + Foreign Keys: memberContact.memberId references member.memberId, and memberContact.contactId references contact.contactId.