Design: Entity-Relationship Model

The Varsity International Network of Oenology wishes to computerise the management of the information about its members as well as to record the information they gather about various wines. Your company, Apasaja Private Limited, is commissioned by the Varsity International Network of Oenology to design and implement the relational schema of the database application. The organisation is big enough so that there could be several members with the same name. A card with a unique number is issued to identify each drinker. The contact address of each member is also recorded for the mailing of announcements and calls for meetings.

At most once a week, VINO organises a tasting session. At each session, the attending members taste several bottles. Each member records for each bottle his or her evaluation of the quality (very good, good, average, mediocre, bad, very bad) of each wine that she or he tastes. The evaluation may differ for the same wine from one drinker to another. Actual quality and therefore evaluation also varies from one to another bottle of a given wine. Every bottle that is opened during the tasting session is finished during that session.

Each wine is identified by its name ("Parade D'Amour"), appellation ("Bordeaux") and vintage (1990). Other information of interest about the wine is the degree of alcohol (11.5), where and by whom it has been bottled ("Mis en Bouteille par Amblard-Larolphie Negociant-Eleveur a Saint Andrede Cubzac (Gironde) - France"), the certification of its appellation if available ("Appellation Bordeaux Controlée"), and the country it comes from (produce of "France").

Generally, there are or have been several bottles of the same wine in the cellar. For each wine, the bottles in the wine cellar of VINO are numbered. For instance, the cellar has 20 bottles numbered 1 to 20 of a Semillon from 1996 named Rumbalara. For documentation purposes VINO may also want to record wines for which it does not own bottles. The bottles are either available in the cellar or they have been tasted and emptied.

We first want to design a schema via entity-relationship diagram that most correctly and most completely captures the constraints expressed in the above description of the VINO application.

Questions

Not all questions will be discussed during tutorial. You are expected to attempt them before coming to the tutorial. You may be randomly called to present your answer during tutorial. You are encouraged to discuss them on Canvas Discussion.

1. **Entity-Relationship Design**.

   (a) Identify the entity sets. Justify your choice by quoting the sentences in the text that support it.

Comments:
There are *at least* 3 entity sets:

- `member`: "... *information about its* ***members*** *...*".

  Note: In the context of the description, a *drinker* ("... *A card with a unique number is issued to identify each* ***drinker*** *...*") is the same as a *member*.
- `wine`: "... *record the information they gather about various* ***wines***.".
- `bottle`: "... *there are or have been several* ***bottles*** *of the same wine* ...".

These generally correspond to *nouns* but not *proper nouns* (e.g., The Eiffel Tower). As you will see later, there is another entity set as shown below. However, these first three are sufficient for a *first iteration* of the answer.

- `session`: "... *VINO organises a tasting session.*".

  Note: We can also name this `tasting_session`.

   (b) Identify the relationship sets and the entity sets that they associate. Justify your choice by quoting the sentences in the text that support it.

Comments:
There are *at least* 2 relationship sets:

- `taste`: "... *the attending members* ***taste*** *several bottles.*"

  It relates `member` with `bottle` (i.e., member taste bottle).

  Note: We can also name this `tasted` or `tastes`. However, be consistent in naming.
- `contain`: "... *bottle of a given wine.*"

  It relates `bottle` with `wine` (i.e., bottle contain wine). This relationship set can be inferred from the text. We may use other names such as `in` (i.e., wine in a bottle).

  Note: We can also name this `contained` or `contains`. However, be consistent in naming.

These generally correspond to *verbs*. However, you may need to infer its presence as well as its name. Additionally, there may be other relationship set as shown below.

- `open`: "*Every bottle that is* ***opened*** *during the tasting session ...*".

   (c) For each entity set and relationship set identify its attributes. Justify your choice by quoting the sentences in the text that support it.

Comments:
See the solution.

(d) For each entity set, identify its keys.

Comments:

See the solution.

IMPORTANT: There is no key for relationship sets in the context of entity-relationship diagram. This also applies to aggregates.

(e) For each entity set and each relationship set in which it participates, indicate the minimum and maximum participation constraints.

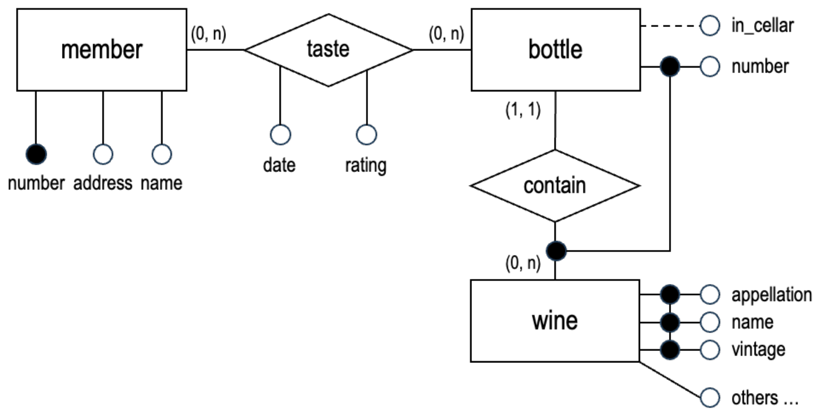
Comments:

The relationship `taste` is an optional many-to-many relationship: both pairs of cardinality constraints are `(0, n)`.

The relationship `contain` is a one-to-many relationship from `wine` to `bottle`. It is mandatory for the bottle entities and optional for the `wine` entities. The participation constraints for the entity set `wine` is `(0, n)`. The participation constraints for the entity set `bottle` is `(1, 1)`. It could not be otherwise since the entity set `bottle` is a **weak entity** (*i.e., it is weakly identified under the relationship set* `contain` *and the dominant entity set* `wine`).

(f) Draw the corresponding entity-relationship diagram with the key and participation constraints. Indicate in English the constraints that cannot be captured, if any.

Comments:



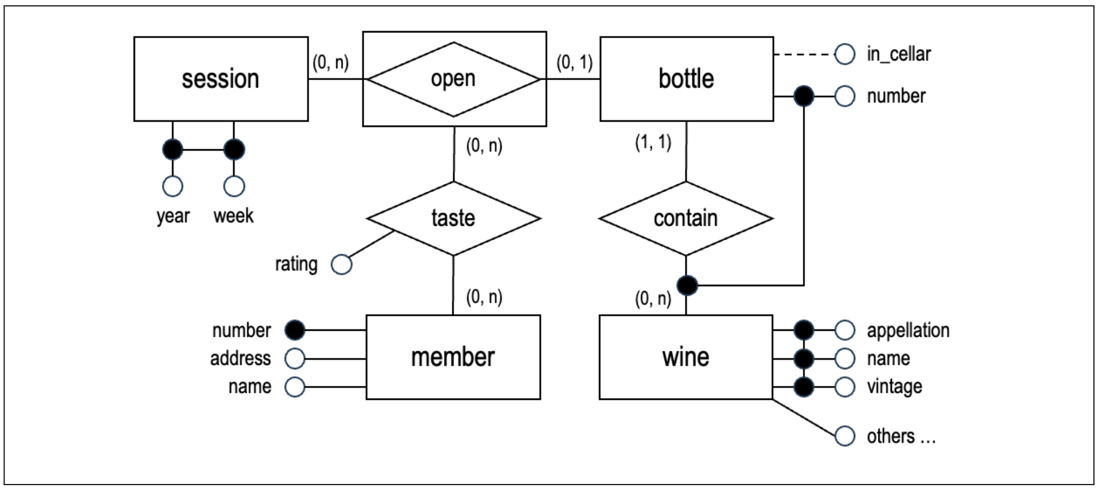
The dotted line for the `in_cellar` attribute indicates that it is *calculated* (a bottle is in cellar if it was not drunk). It can be implemented as a **view**. If it is materialised the corresponding integrity constraints must be implemented.

Note that we use `others ...` as a simplification. You should not do this for your submission.

There are other constraints not captured by the entity-relationship above such as the following.

- A session is held at most once a week.
- Every bottle that is opened during the tasting session is finished during that session (i.e., a bottle can only be opened in at most one session).

These constraints can be captured if we use **aggregate** below.



Comments:

Some common mistakes:

- Having identifying attributes (i.e., black dot) for relationship set (including aggregate).
- Not having additional black dots for *composite* identifying attributes.
- Missing non-default participation constraints (i.e., those not `(0, n)`). Similarly, we cannot have other complicated constraints (e.g., having `(0, x)` and `(0, y)` on two different participation with additional constraint as text `x + y = 1`).
- Having other participation constraints beside `(1, 1)` (or not listing it, which means it becomes the default `(0, n)`) for weak entity set (i.e., `bottle`) to identifying relationship set (i.e., `contain`). On that note, the identifying attribute of the weak entity set (i.e., the partial key) should be an additional black dot and connected to the line between the dominant entity set (i.e., `wine`) and the identifying relationship set.
- Having the rectangle of aggregates touching the diamond. This may cause ambiguity. On that note, when connected to the aggregate as entity set, the line should end on the rectangle.

Some common design issues:

- Adding an artificial key (e.g., `session_id`). This does not enforce the original constraint (i.e., one session per week) and there is a natural key (i.e., `(year, week)`) that can be used.
- Using ternary relationship set when aggregate is needed.
- Not using weak entity set when it is needed.
- Using entity set when relationship set is needed. This is typically when the word can be both noun and verb (e.g., rent). If it requires artificial key (e.g., not mentioned in the description), then relationship set is more appropriate.

- Having the same attribute name in different entity/relationship set that corresponds to the same item. For instance, instead of using weak entity set concept for `bottle`, we use `number` attribute in `wine` and claim that it should refer to the bottle number.

  This is incorrect. Each attribute appearing in entity-relationship diagram refers to different item regardless of the name. While the best practice is to use different attribute names for different items, it may lead to a long and complicated name.

  For instance, we use `number` for both `member` and `bottle` and we use `name` for both `member` and `wine`. An alternative would be `member number`, `bottle number`, `member name`, and `wine name` respectively.

This variant of entity-relationship diagram is designed carefully in such a way that we can express multiple candidate keys. It is then the task during the schema translation to decide which one is the `PRIMARY KEY`. The rest of the candidate keys will then be enforced with a combination of `UNIQUE` and `NOT NULL` constraints.

Given that the focus is on the keys, we need to first identify the identifying attributes. Once we have recognized the candidate keys, the rest of the attributes should be quite straightforward.

2. **Logical Design**.

   (a) Translate your entity-relationship diagram into a relational schema. Give the SQL DDL statements to create the schema. Declare the necessary integrity constraints. Indicate in English the constraints that cannot be captured, if any.

Comments:

The translation for solution **without aggregate** is shown below.

Code: Without Aggregate

```sql
CREATE TABLE member (
  card_number  CHAR(10)     PRIMARY KEY,
  address      VARCHAR(64)  NOT NULL,
  name         VARCHAR(32)  NOT NULL
);

CREATE TABLE wine (
  name            VARCHAR(32),
  appellation     VARCHAR(32),
  vintage         DATE,
  alcohol_degree  NUMERIC NOT NULL,
  bottled         VARCHAR(128) NOT NULL,
  certification   VARCHAR(64),
  country         VARCHAR(32) NOT NULL,
  PRIMARY KEY (name, appellation, vintage)
);

CREATE TABLE bottle (
  wine_name       VARCHAR(32),
  appellation     VARCHAR(32),
  vintage         DATE,
  number          INTEGER CHECK (number > 0),
  PRIMARY KEY (number, wine_name, appellation, vintage),
  FOREIGN KEY (wine_name, appellation, vintage)
    REFERENCES wine (name, appellation, vintage)
);

CREATE TABLE taste (
  wine_name      VARCHAR(32),
  appellation    VARCHAR(32),
  vintage        DATE,
  bottle_number  INTEGER,
  member         CHAR(10)
    REFERENCES member (card_number),
  tasting_date   DATE NOT NULL,
  rating         VARCHAR(32) NOT NULL,
  PRIMARY KEY (member, bottle_number, wine_name, appellation, vintage),
  FOREIGN KEY (bottle_number, wine_name, appellation, vintage)
    REFERENCES bottle (bottle_number, wine_name, appellation, vintage)
);
```

The translation for solution **with aggregate** is shown below.

Code: With Aggregate

```
CREATE TABLE member (
  card_number  CHAR(10)     PRIMARY KEY,
  address      VARCHAR(64)  NOT NULL,
  name         VARCHAR(32)  NOT NULL
);

CREATE TABLE wine (
  name            VARCHAR(32),
  appellation     VARCHAR(32),
  vintage         DATE,
  alcohol_degree  NUMERIC NOT NULL,
  bottled         VARCHAR(128) NOT NULL,
  certification   VARCHAR(64),
  country         VARCHAR(32) NOT NULL,
  PRIMARY KEY (name, appellation, vintage)
);

CREATE TABLE session (
  year  INTEGER,
  week  INTEGER,
  PRIMARY KEY (year, week)
);

CREATE TABLE bottle (
  wine_name       VARCHAR(32),
  appellation     VARCHAR(32),
  vintage         DATE,
  number          INTEGER CHECK (number > 0),
  PRIMARY KEY (number, wine_name, appellation, vintage),
  FOREIGN KEY (wine_name, appellation, vintage)
    REFERENCES wine (name, appellation, vintage)
);

CREATE TABLE open (
  wine_name       VARCHAR(32),
  appellation     VARCHAR(32),
  vintage         DATE,
  bottle_number   INTEGER,
  session_year    INTEGER NOT NULL,
  session_week    INTEGER NOT NULL,
  PRIMARY KEY (bottle_number, wine_name, appellation, vintage),
  FOREIGN KEY (session_year, session_week)
    REFERENCES session (year, week),
  FOREIGN KEY (bottle_number, wine_name, appellation, vintage)
    REFERENCES bottle (number, wine_name, appellation, vintage)
);
```

continued on next page ...

... *continued from previous page*

```
CREATE TABLE taste (
  wine_name      VARCHAR(32),
  appellation    VARCHAR(32),
  vintage        DATE,
  bottle_number  INTEGER,
  member         CHAR(10)
    REFERENCES member (card_number),
  rating         VARCHAR(32) NOT NULL,
  PRIMARY KEY (member, bottle_number, wine_name, appellation, vintage),
  FOREIGN KEY (bottle_number, wine_name, appellation, vintage)
    REFERENCES open (bottle_number, wine_name, appellation, vintage)
);
```

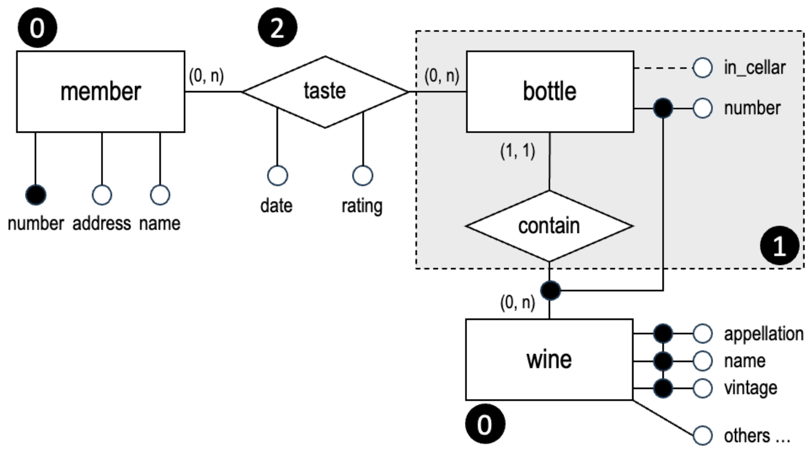
Note: Our translation scheme is designed in such a way to minimize the number of possible `NULL` values. By default, all attributes should be guarded with `NOT NULL` constraints. You will need to justify each place where `NOT NULL` constraints is not present unless the attribute is already part of `PRIMARY KEY` which by default is `NOT NULL`.

Comments:

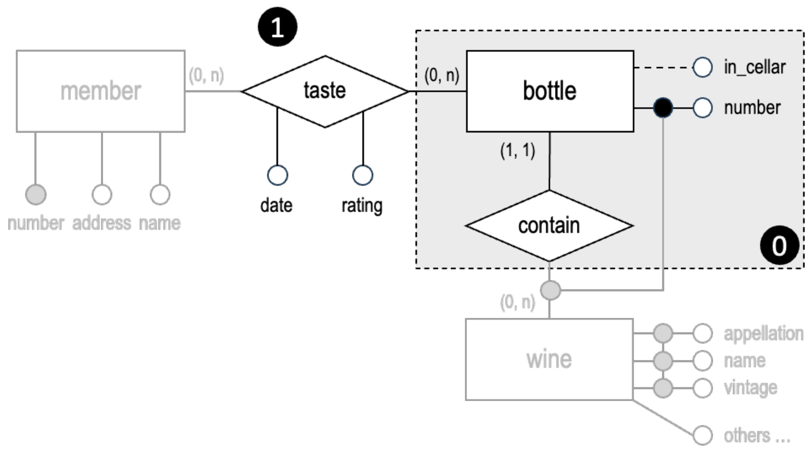
In general, translation should start from entity set without any `FOREIGN KEY` references to another table. However, we should also identify relationship sets that are merged with one or more entity sets. In the example without aggregate, we merge `bottle` and `contain`.

We will count the number of `FOREIGN KEY` references to another table for each entity/relationship set. Then we convert all that has the count of 0 and update the count. The conversion may look like the following.

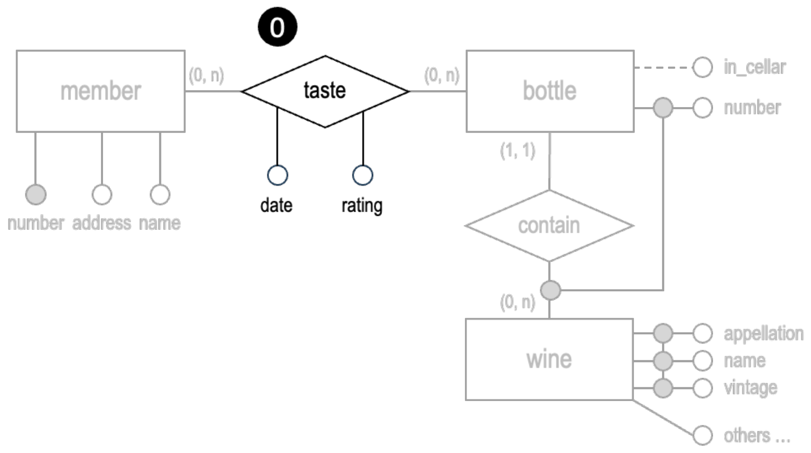
1. Translate `member` and `wine` in any order.



2. Translate `bottle` and `contain` (i.e., they are merged). We name this table `bottle` for simplicity but a better name is `bottle_contain` to show that this comes from the merging of `bottle` and `contain`.



3. Translate `taste`.



References

[1] P. Atzeni et al. *Database Systems - Concepts, Languages and Architectures.* [Online; last accessed 2025]. `http://dbbook.dia.uniroma3.it/`.

[2] S. Bressan and B. Catania. *Introduction to Database Systems.* McGraw-Hill Education, 2006. ISBN: 9780071246507.

[3] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book.* 2nd ed. Prentice Hall Press, 2008. ISBN: 9780131873254.

[4] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems.* 2nd. USA: McGraw-Hill, Inc., 2000. ISBN: 0072440422.