CS2102: Database Systems

Tutorial 2 — Entity-Relationship Model

Question 1

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.

Question 1

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.

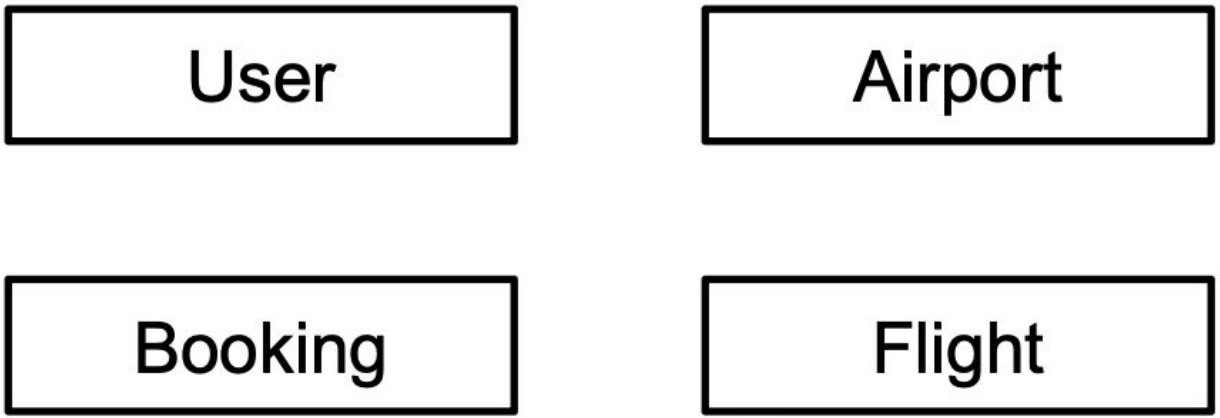
Question 1

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

Recap - Entity Set

Real-world things or objects that are distinguishable from other objects

Typically **nouns**



Question 1

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

member	*"... information about its **members** ..."*
wine	*"... record the information they gather about various **wines**."*
bottle	*"... there are or have been several **bottles** of the same wine ..."*
session	*"... VINO organises a tasting **session**."*

Question 1

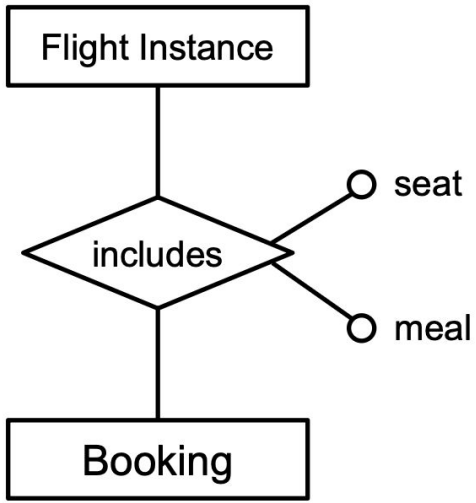
(b) Identify the relationship sets and the entity sets that they associate. Justify your choice by quoting the sentences in the text that support it.

Recap - Relationship Set

Association among two or more entities

Can have their own attributes that further describe the relationship

Names are typically **verbs**

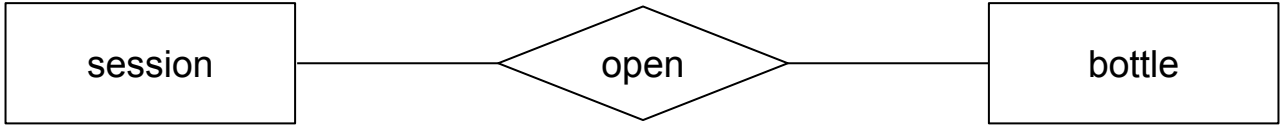


Question 1

(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.

"... the attending members taste several bottles."

"... the attending members taste several bottles."



"... the attending members taste several bottles."

Question 1

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

member

wine

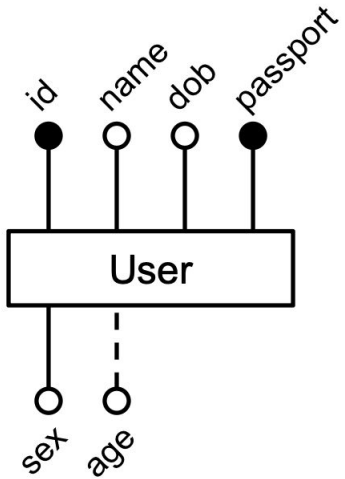
bottle

session

Recap - Attribute

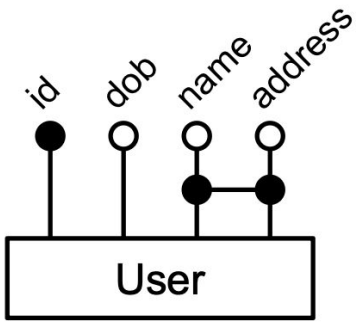
Types of attributes:

- Key attributes
- Derived attributes
- Normal attributes



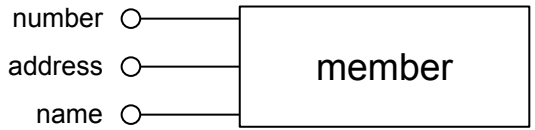
Composite key attributes:

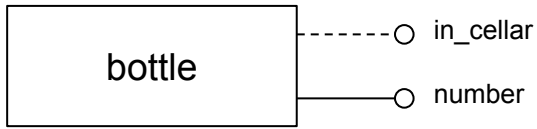
- 2 or more attributes together uniquely identify each entity

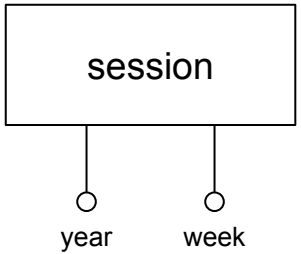


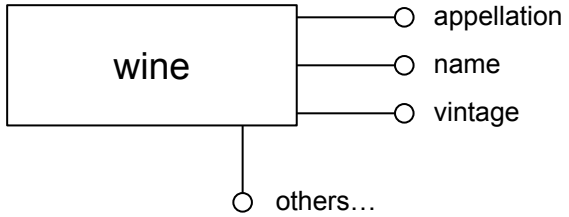
Question 1

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



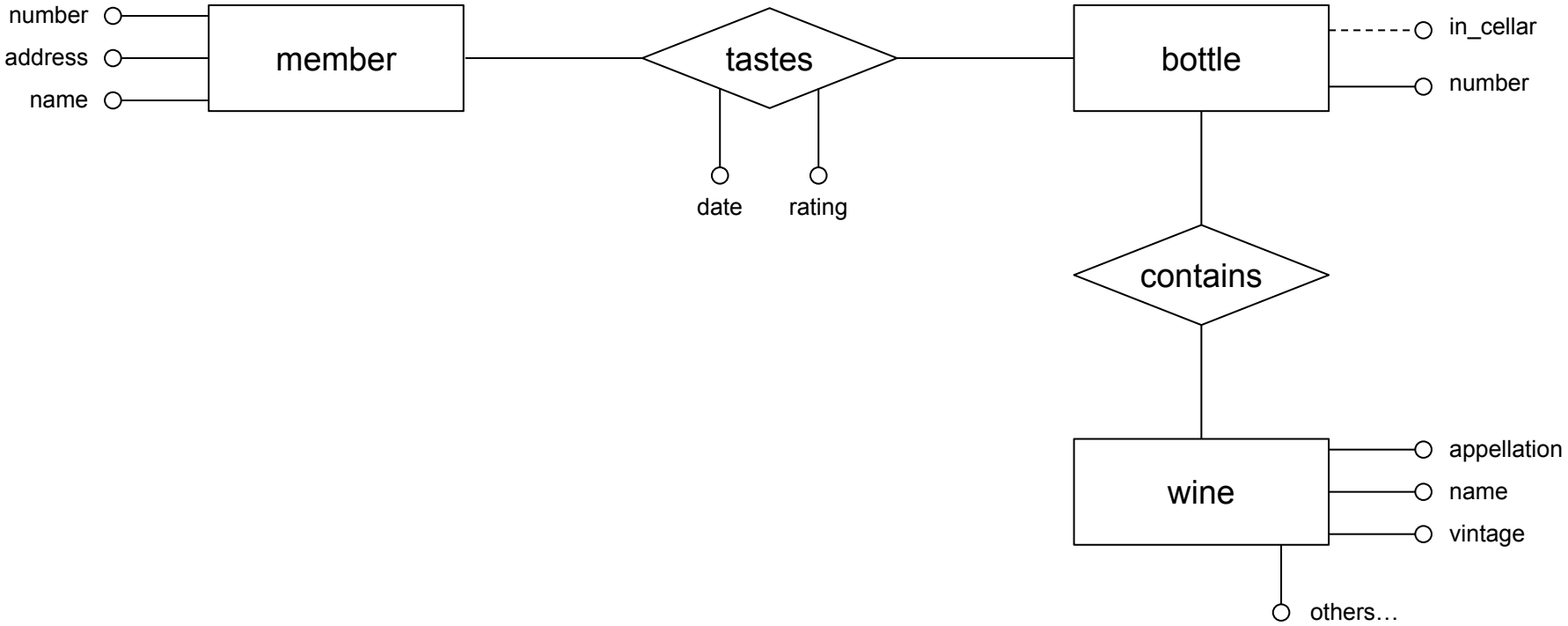






Question 1

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



Recap - Weak Entity Set

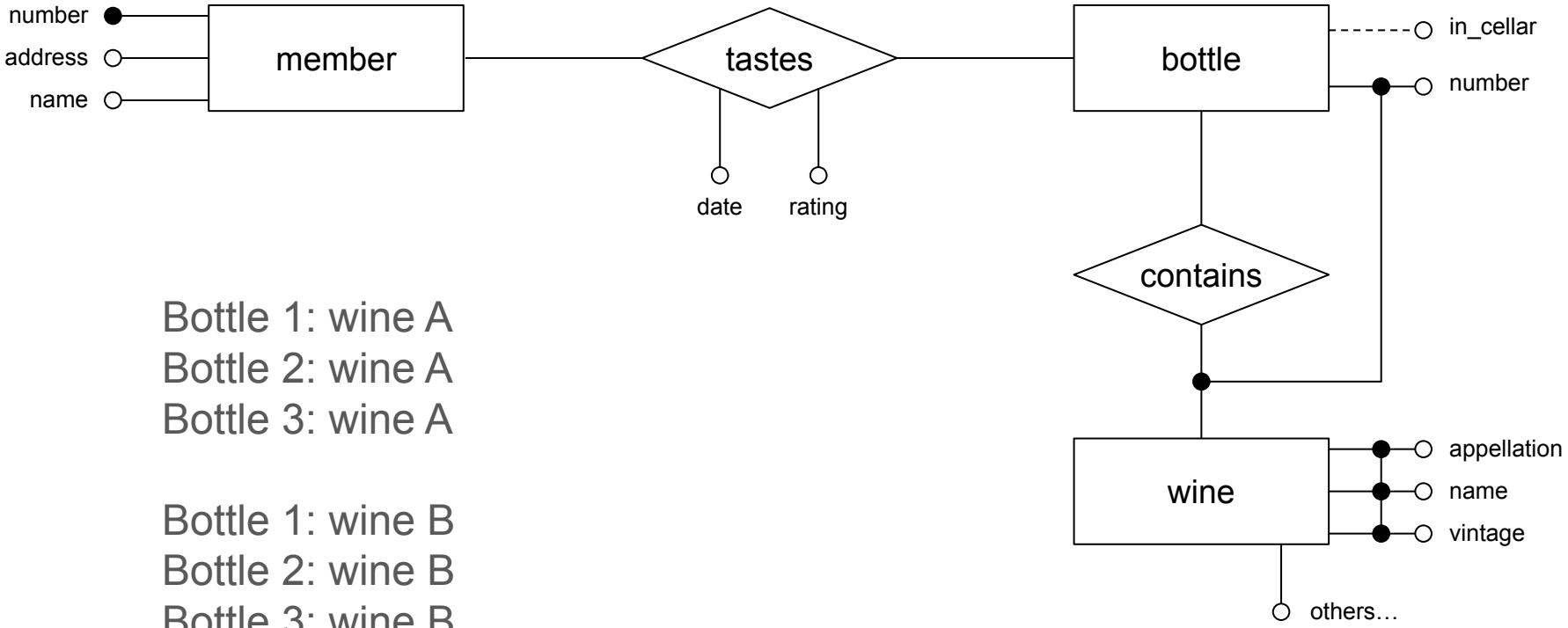
A weak entity can only be uniquely identified by considering the primary key of the owner entity

A weak entity's existence depends on the existence of its owner entity

- Requirements
  - Many-to-one relationship (identifying relationship) from weak entity set to owner entity set
    (one-to-one possible but less common)
  - Weak entity set must have (1, 1) attached to identifying relationship

Question 1

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

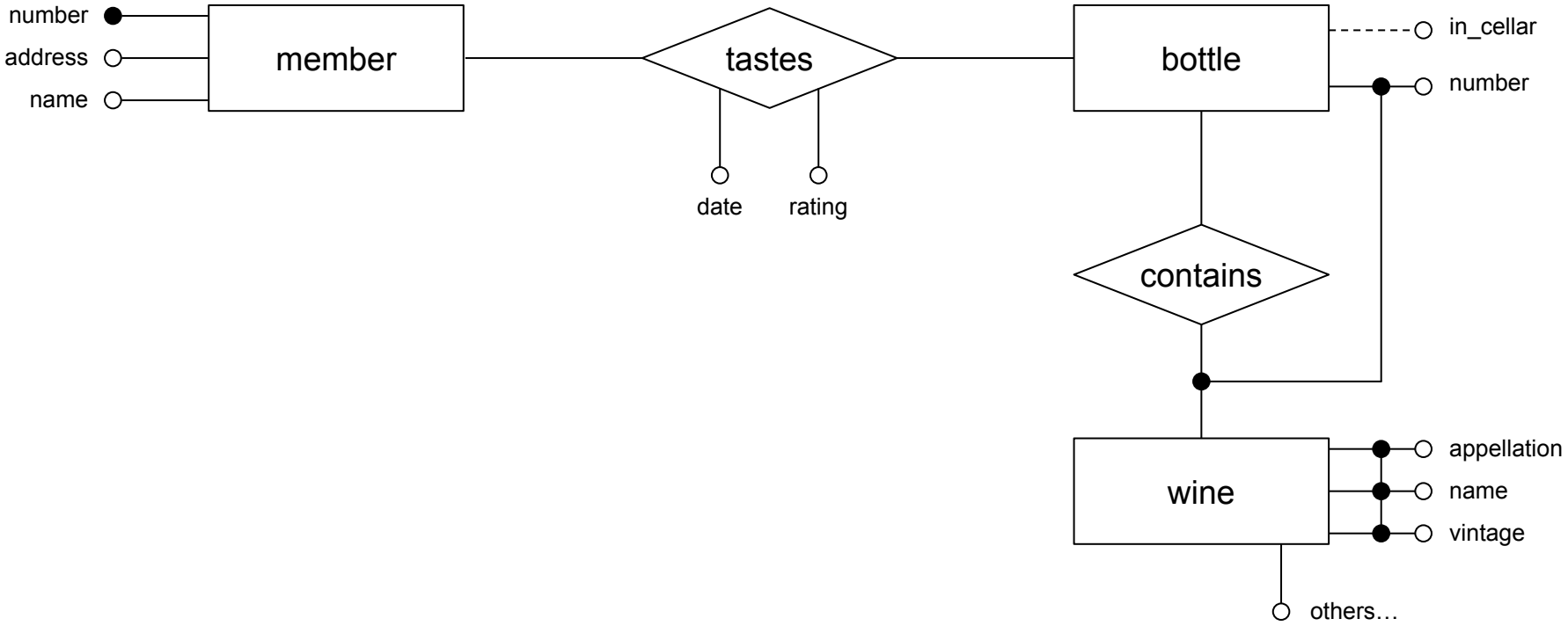


Bottle 1: wine A
Bottle 2: wine A
Bottle 3: wine A

Bottle 1: wine B
Bottle 2: wine B
Bottle 3: wine B

Question 1

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

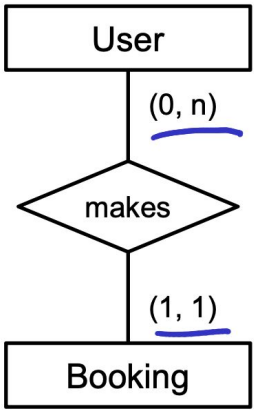


Recap - Cardinality

Participation constraints → describe how often an entity can participate in a relationship

3 basic cardinality constraints:

- Many-to-many
- Many-to-one
- One-to-one

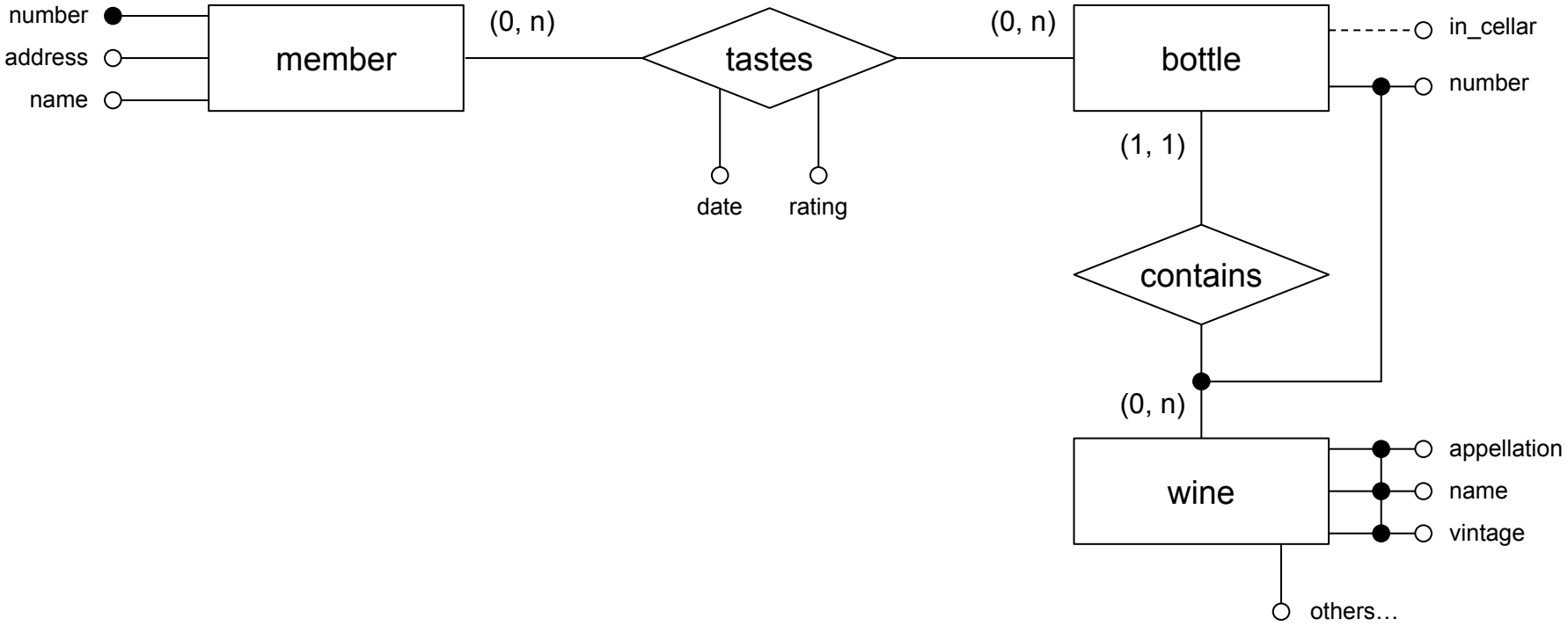


Interpretation

- Each user can make multiple bookings
  (but not every user must have made a booking)
- Each booking was done by exactly one user
  (implies that each booking is associated with a user)

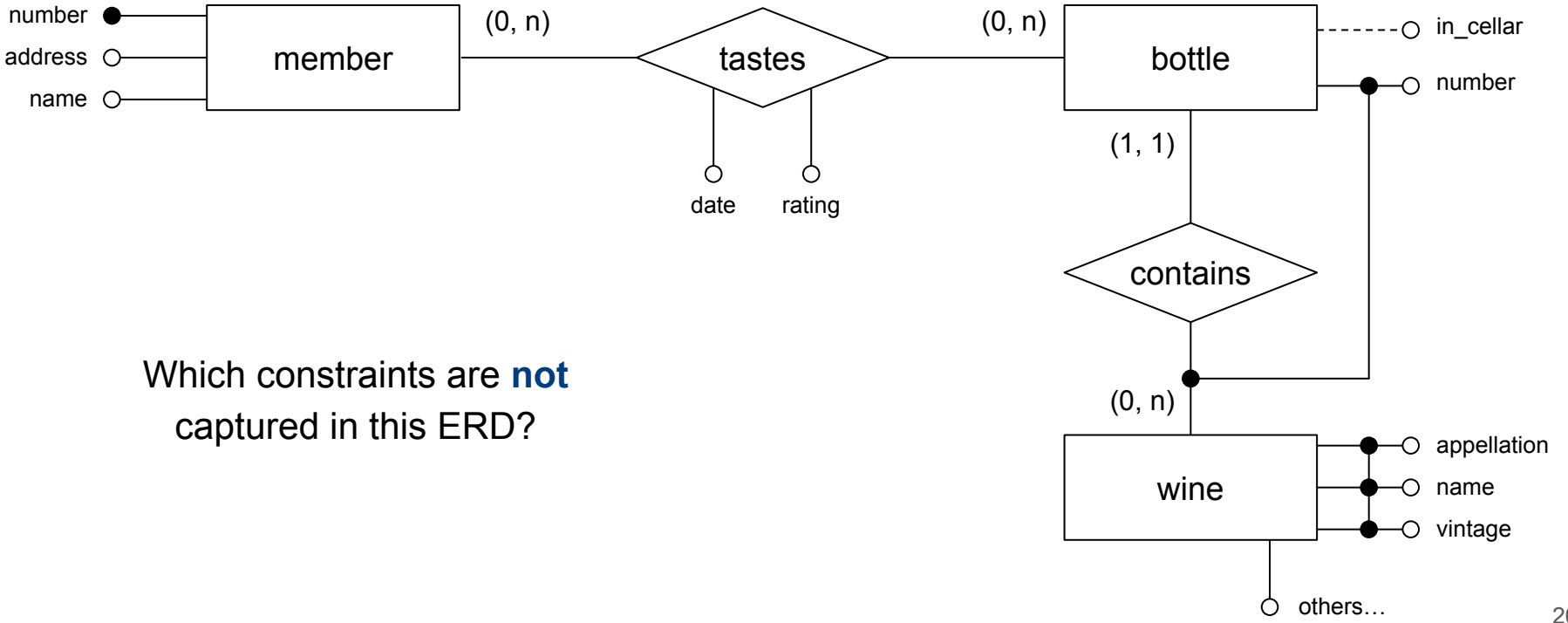
Question 1

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



Question 1

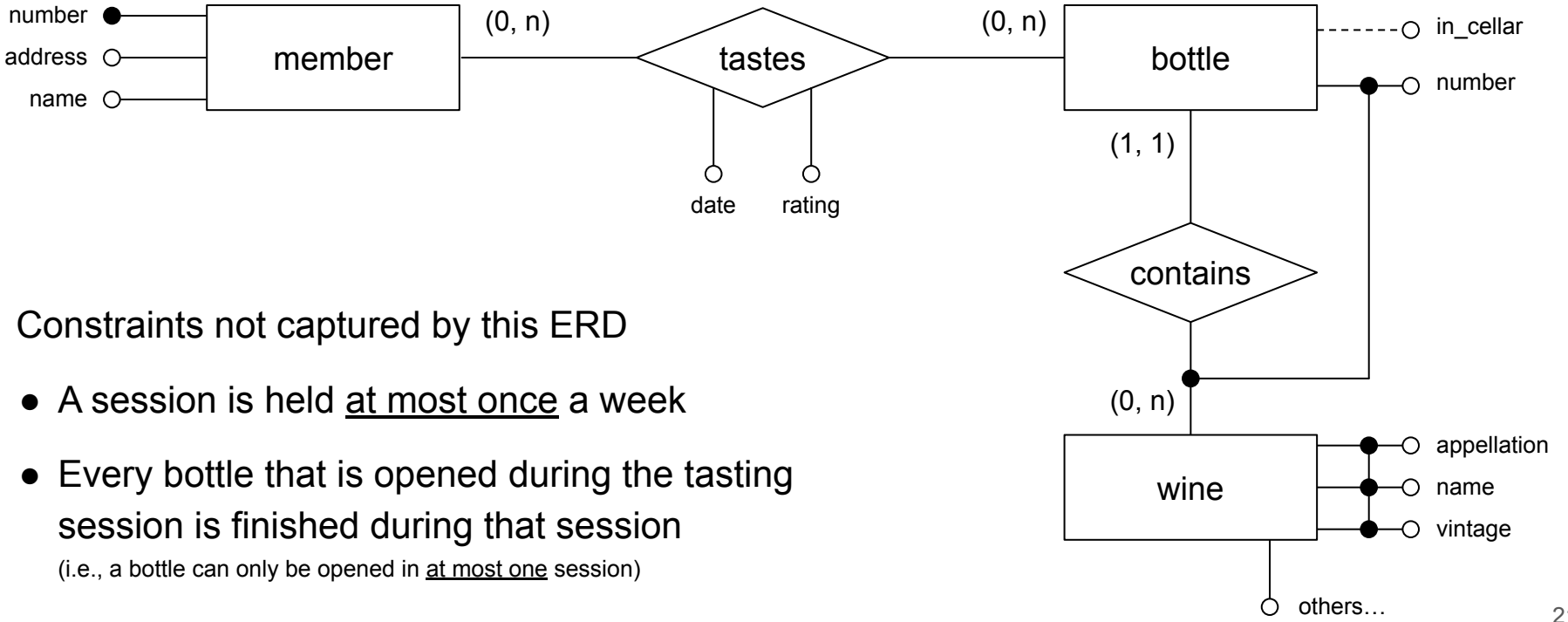
(f) Draw the corresponding entity-relationship diagram with the key and participation constraints. Indicate in English the constraints that cannot be captured, if any.



Which constraints are **not** captured in this ERD?

Question 1

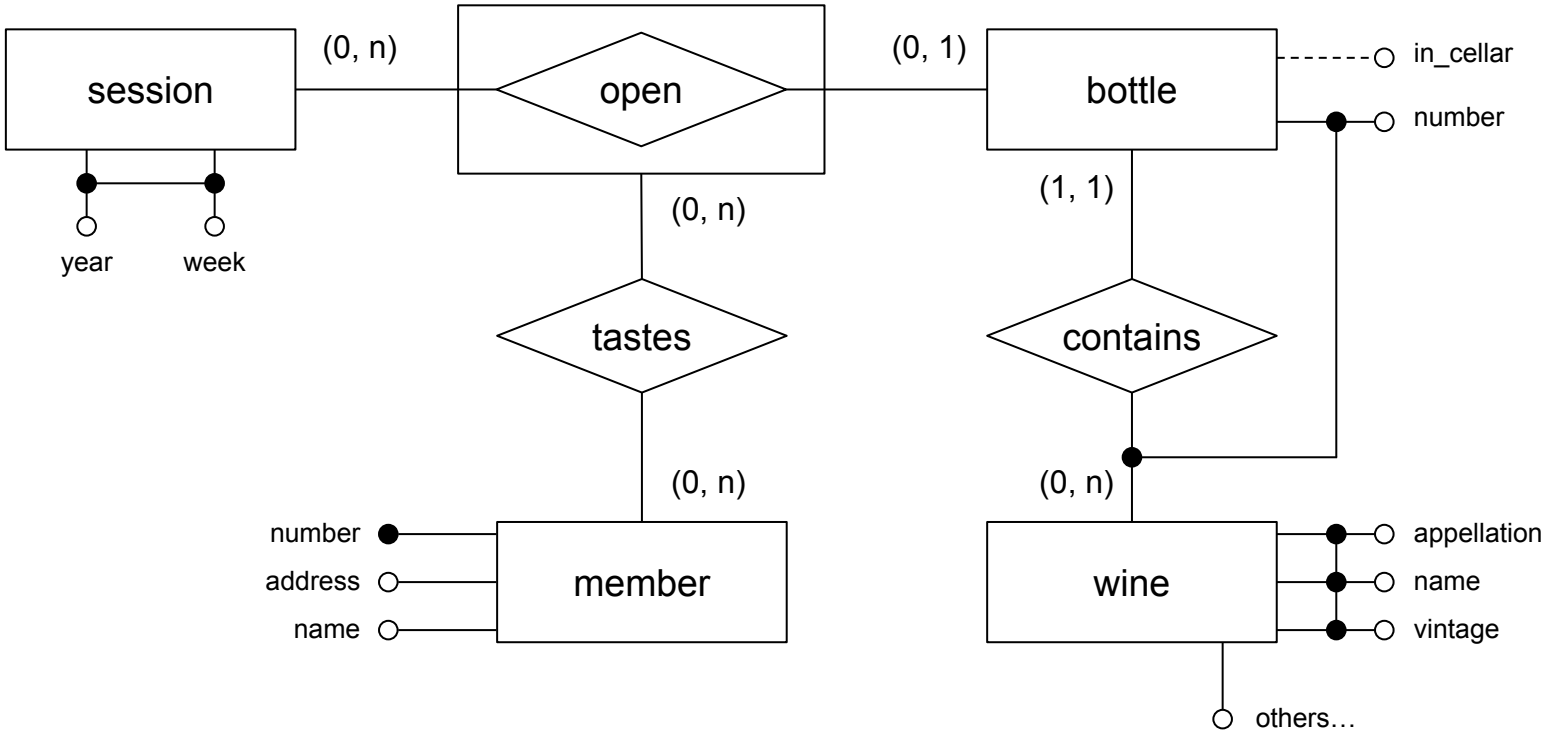
(f) Draw the corresponding entity-relationship diagram with the key and participation constraints. Indicate in English the constraints that cannot be captured, if any.



Constraints not captured by this ERD

- A session is held <u>at most once</u> 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 <u>at most one</u> session)

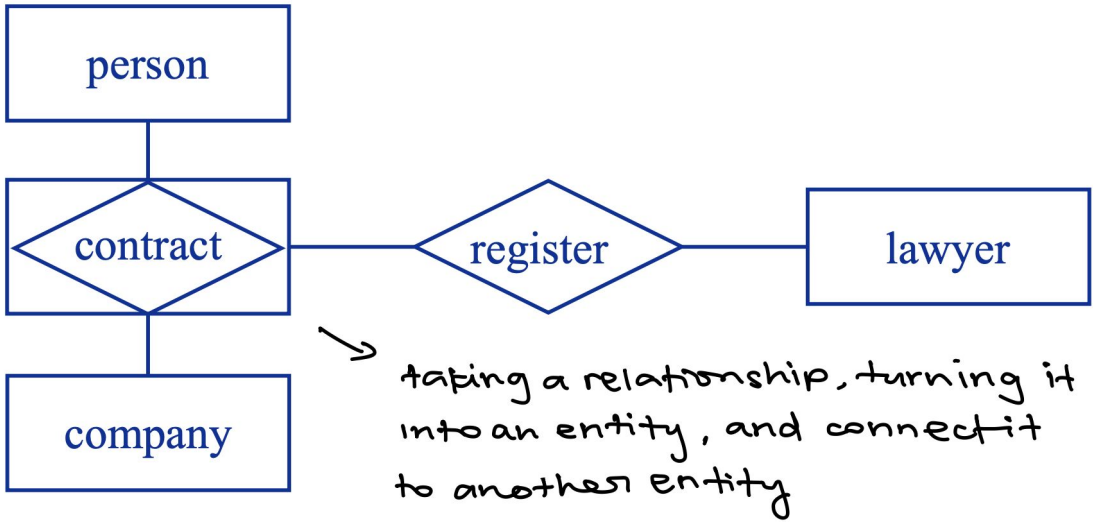
Question 1


session
(0, n)
open
(0, 1)
bottle
in_cellar
number
year
week
(0, n)
(1, 1)
tastes
contains
(0, n)
(0, n)
number
address
name
member
wine
appellation
name
vintage
others…

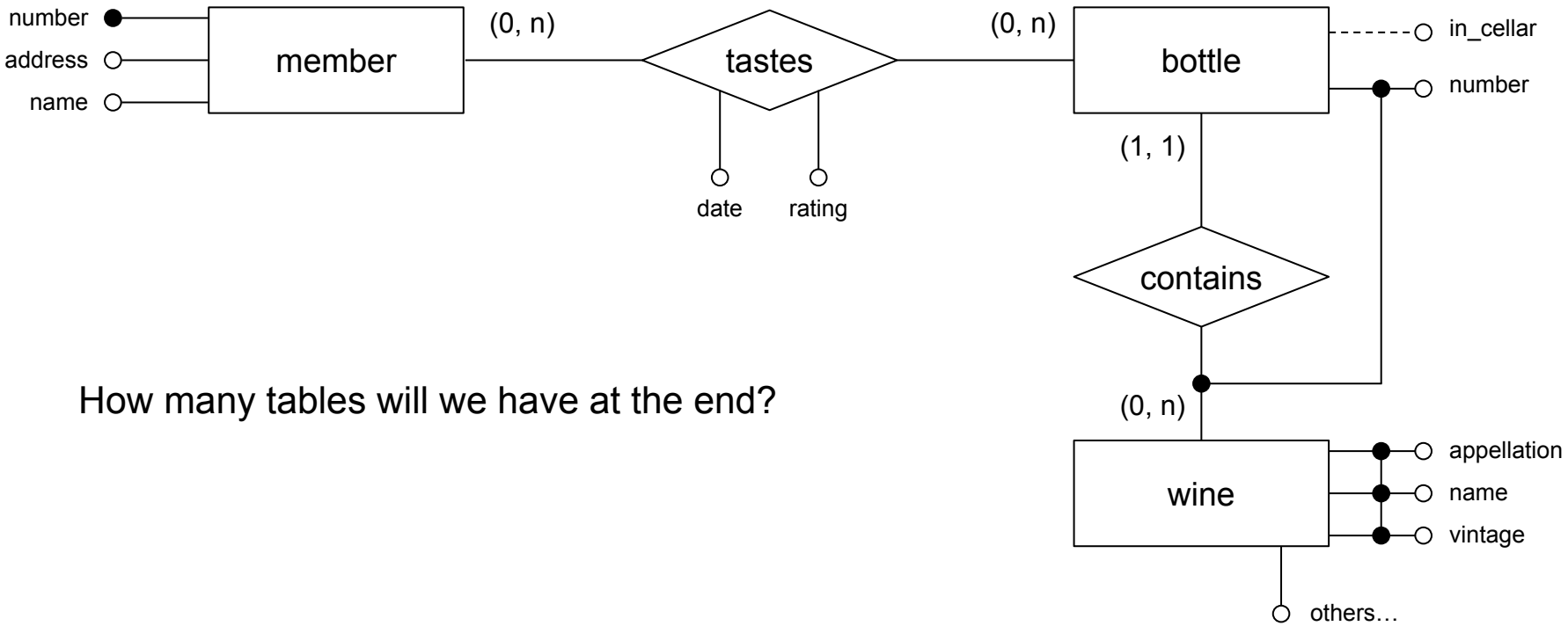
Recap - Aggregation

Taking a relationship, turning it into an entity, and connecting it to another entity

The box around the *contract* diamond should not touch

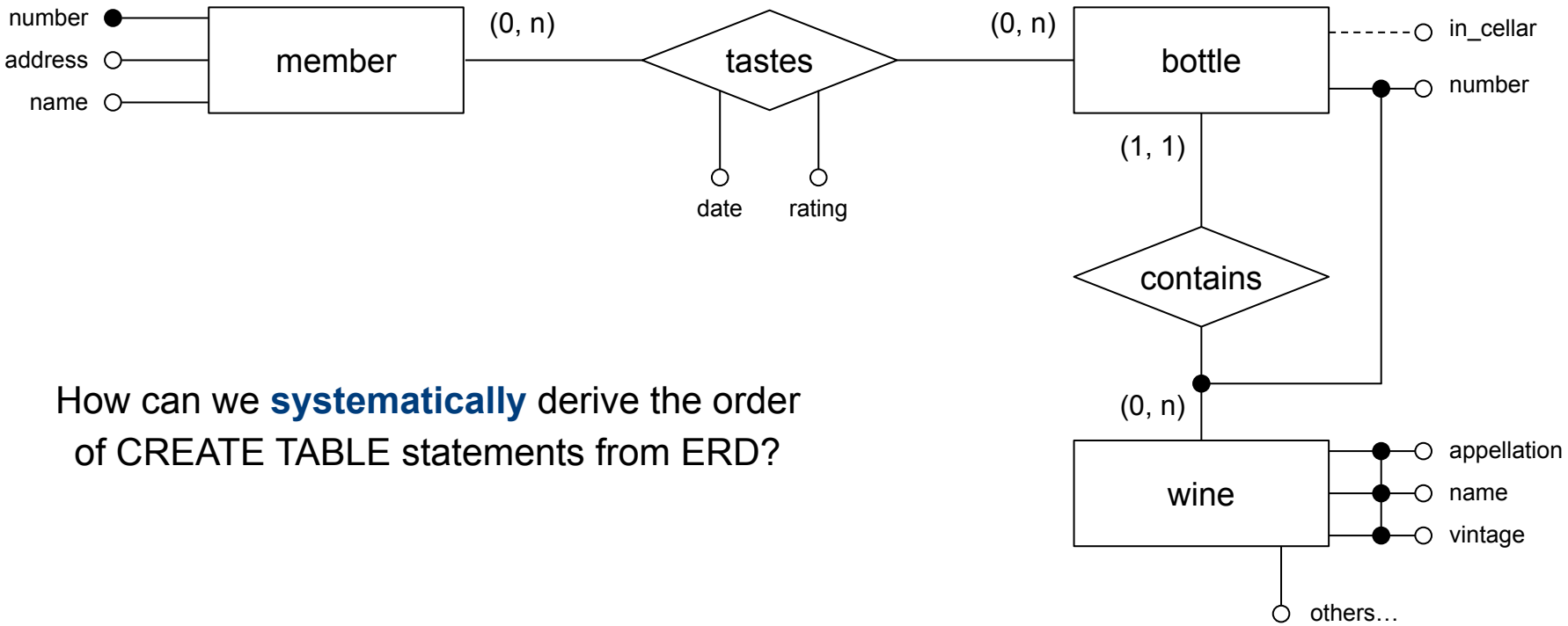


Question 2



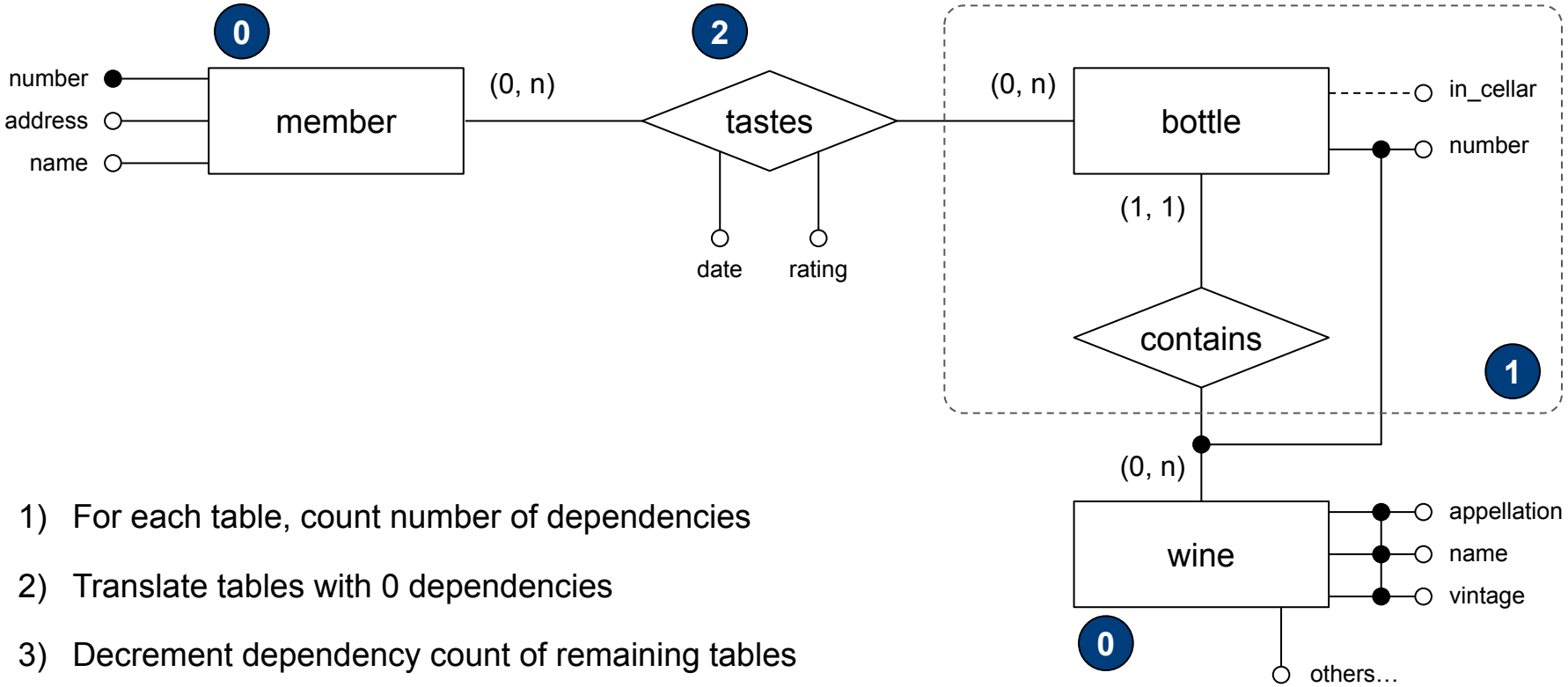
How many tables will we have at the end?

Question 2



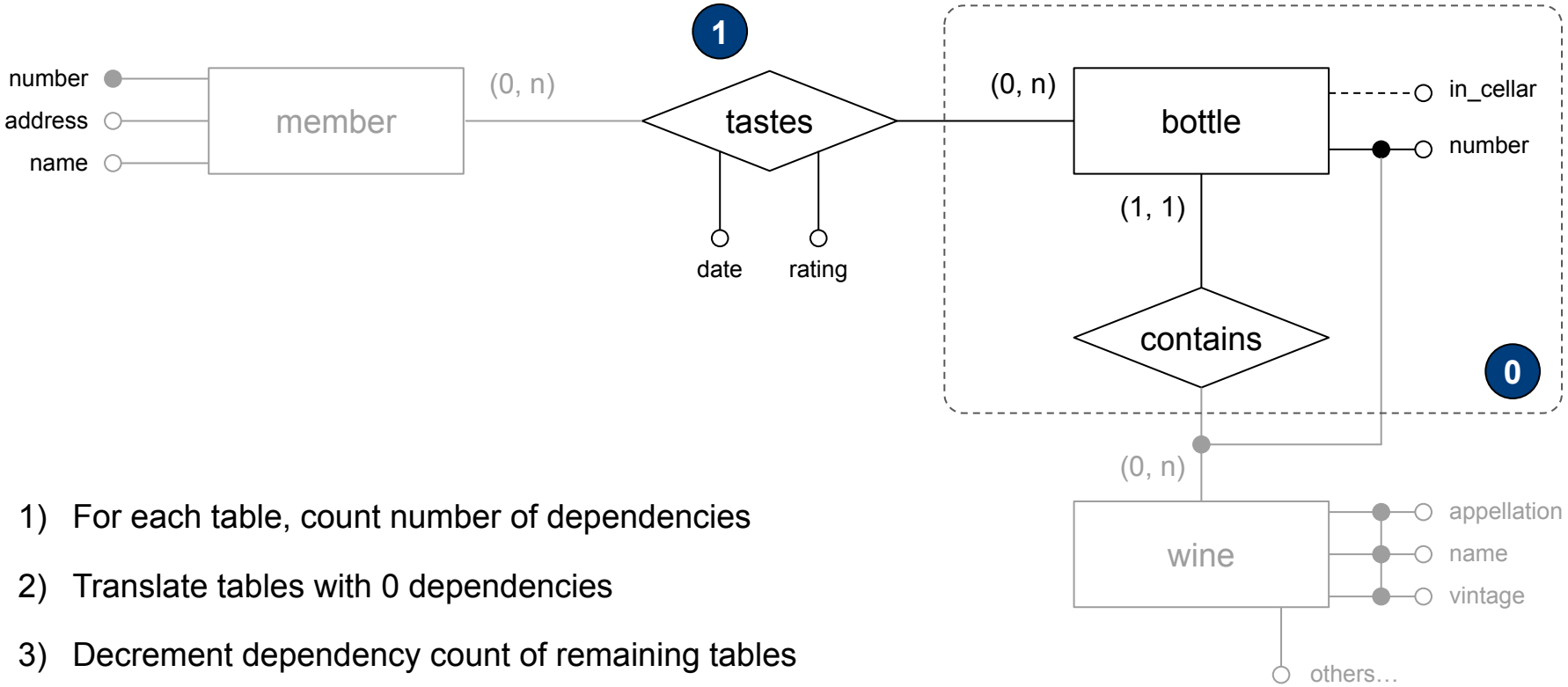
How can we **systematically** derive the order of CREATE TABLE statements from ERD?

Question 2



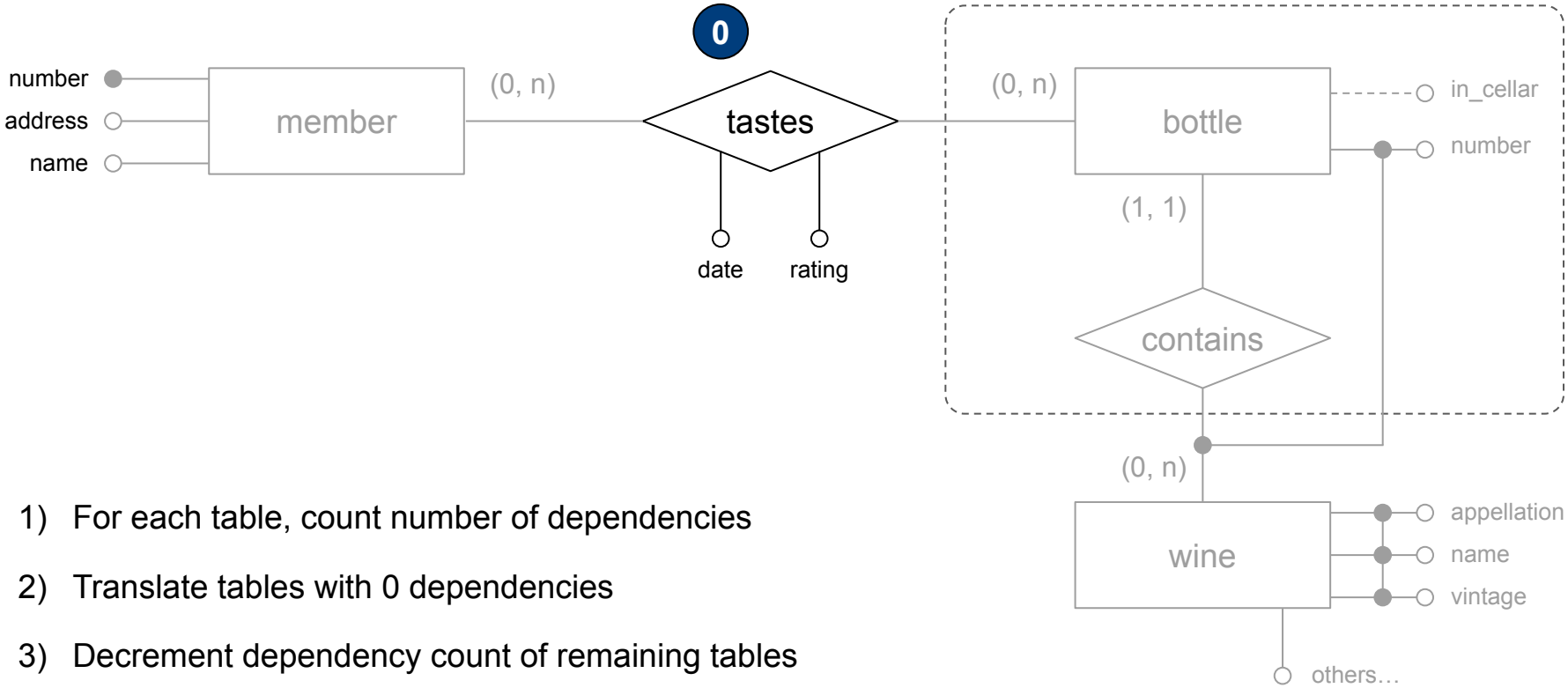
1) For each table, count number of dependencies
2) Translate tables with 0 dependencies
3) Decrement dependency count of remaining tables

Question 2



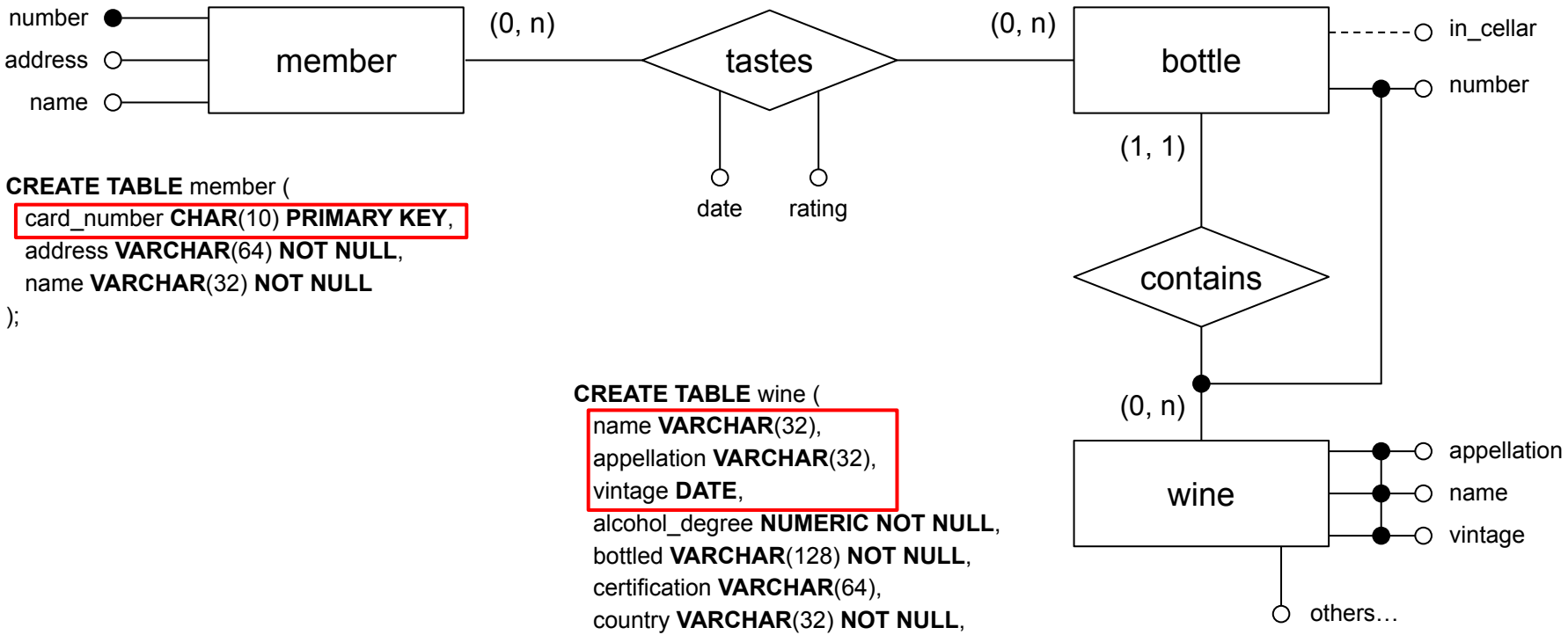
1) For each table, count number of dependencies
2) Translate tables with 0 dependencies
3) Decrement dependency count of remaining tables

Question 2



1) For each table, count number of dependencies
2) Translate tables with 0 dependencies
3) Decrement dependency count of remaining tables

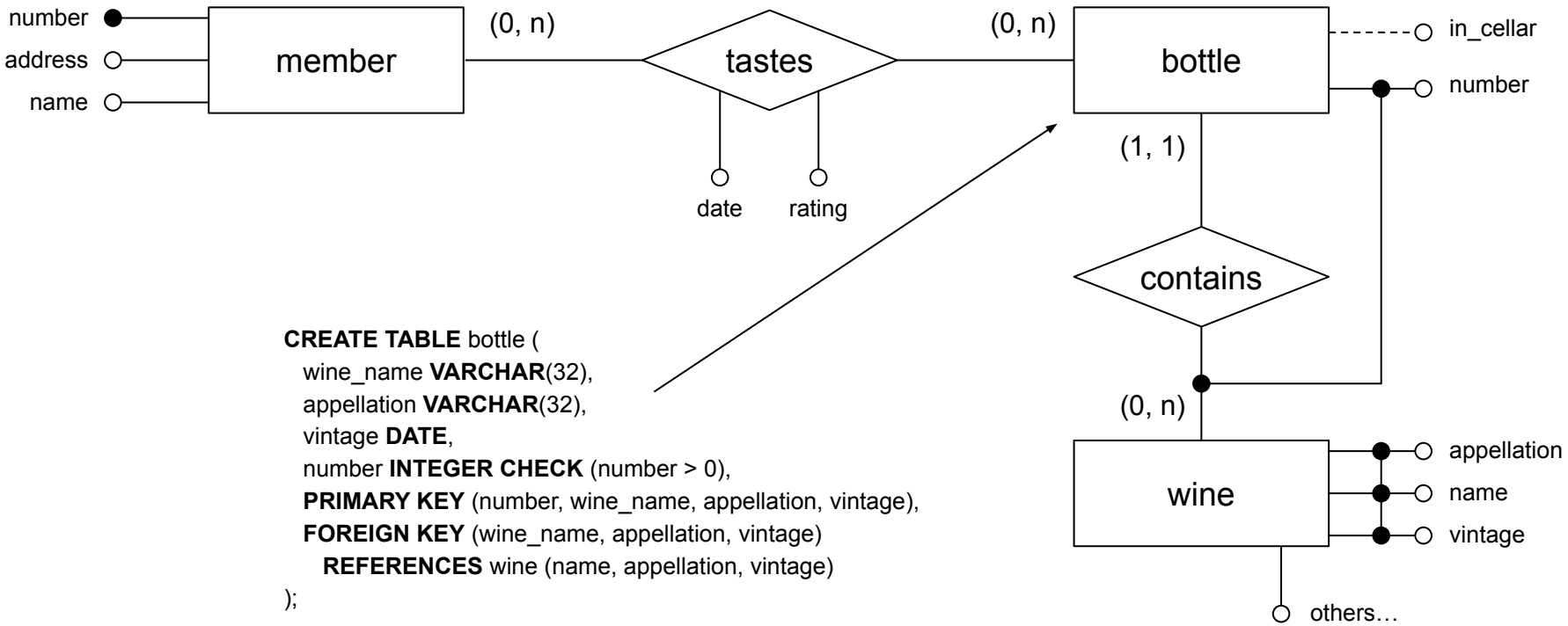
Question 2




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

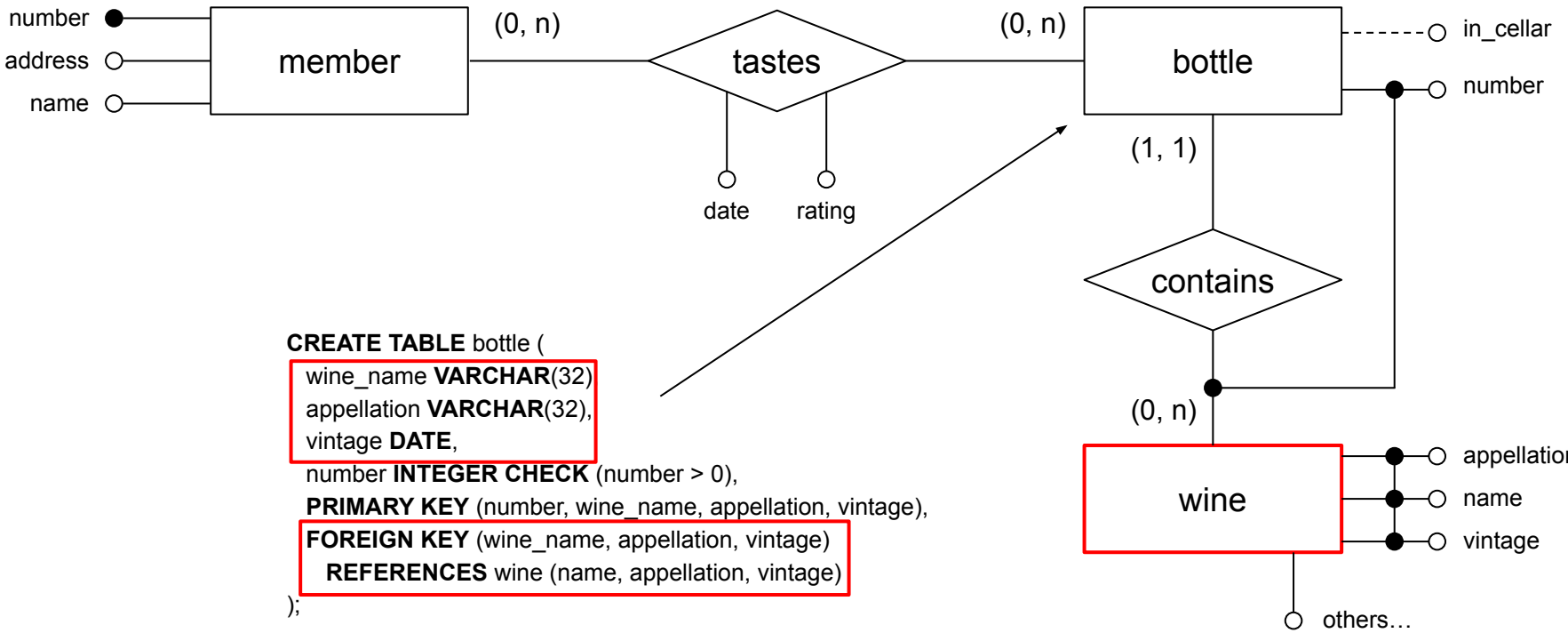
```
CREATE TABLE wine (
  name VARCHAR(32),
  appellation VARCHAR(32),
  vintage DATE,
  alcohol_degree NUMERIC NOT NULL,
  bottled VARCHAR(128) NOT NULL,
  certification VARCHAR(64),
  country VARCHAR(32) NOT NULL,
  PRIMARY KEY (name, appellation, vintage)
);
```

Question 2



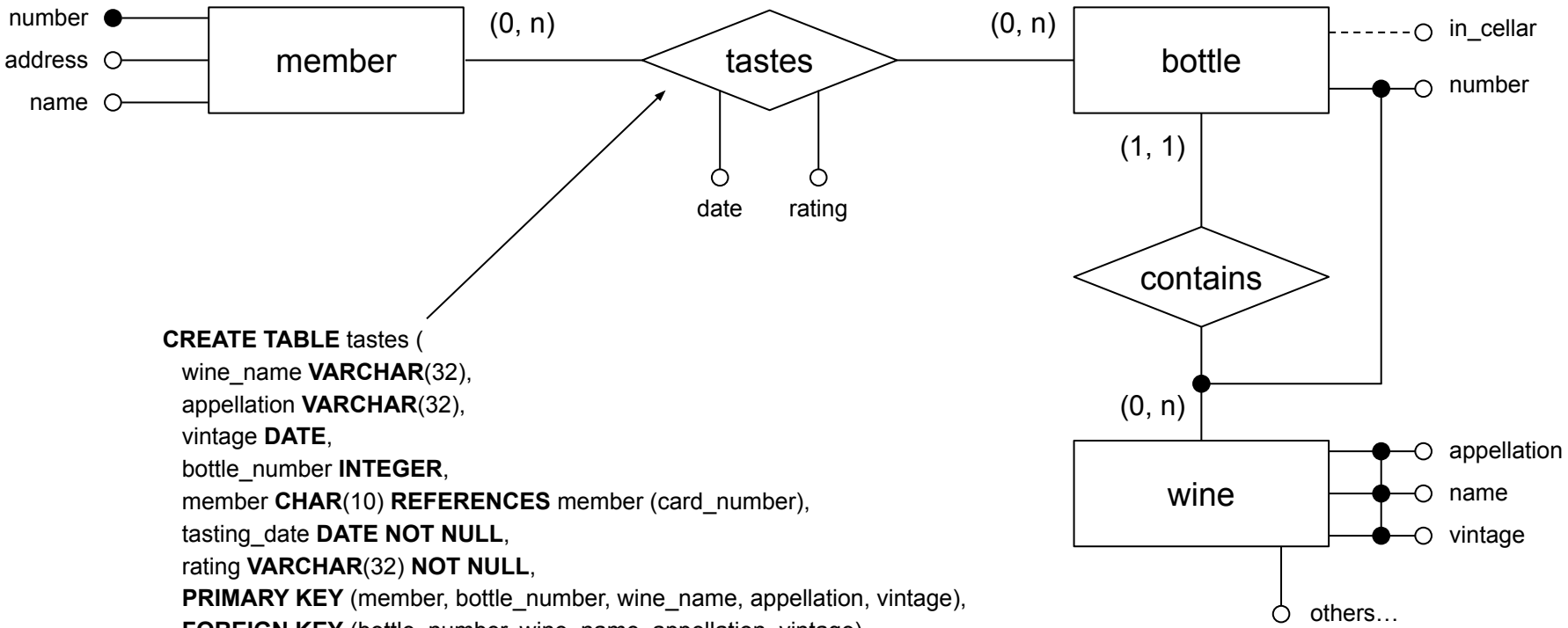

```
CREATE TABLE bottle (
  wine_name VARCHAR(32),
  appellation VARCHAR(32),
  vintage DATE,
  number INTEGER CHECK (number > 0),
  PRIMARY KEY (number, wine_name, appellation, vintage),
  FOREIGN KEY (wine_name, appellation, vintage)
    REFERENCES wine (name, appellation, vintage)
);
```

Question 2



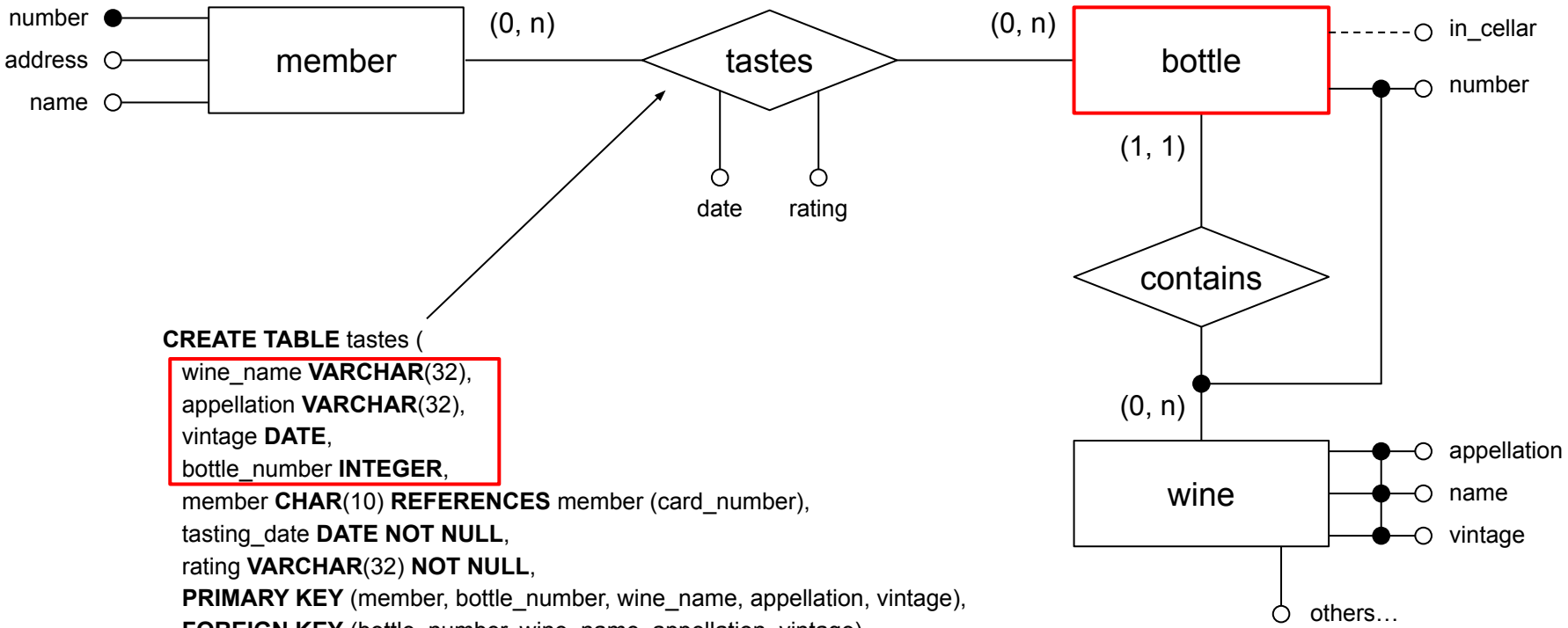

```
CREATE TABLE bottle (
  wine_name VARCHAR(32)
  appellation VARCHAR(32),
  vintage DATE,
  number INTEGER CHECK (number > 0),
  PRIMARY KEY (number, wine_name, appellation, vintage),
  FOREIGN KEY (wine_name, appellation, vintage)
    REFERENCES wine (name, appellation, vintage)
);
```

Question 2



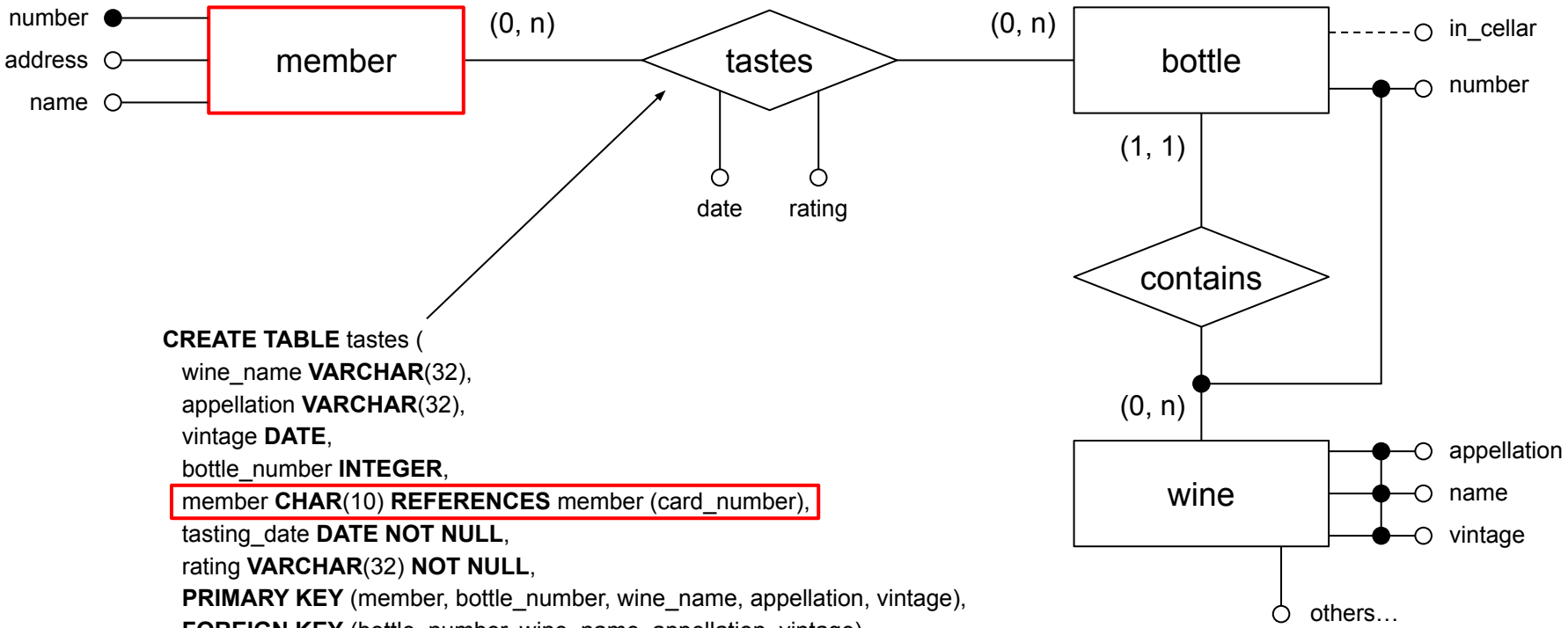

```
CREATE TABLE tastes (
   wine_name VARCHAR(32),
   appellation VARCHAR(32),
   vintage DATE,
   bottle_number INTEGER,
   member CHAR(10) REFERENCES member (card_number),
   tasting_date DATE NOT NULL,
   rating VARCHAR(32) NOT NULL,
   PRIMARY KEY (member, bottle_number, wine_name, appellation,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

Question 2




```
CREATE TABLE tastes (
  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)
);
```

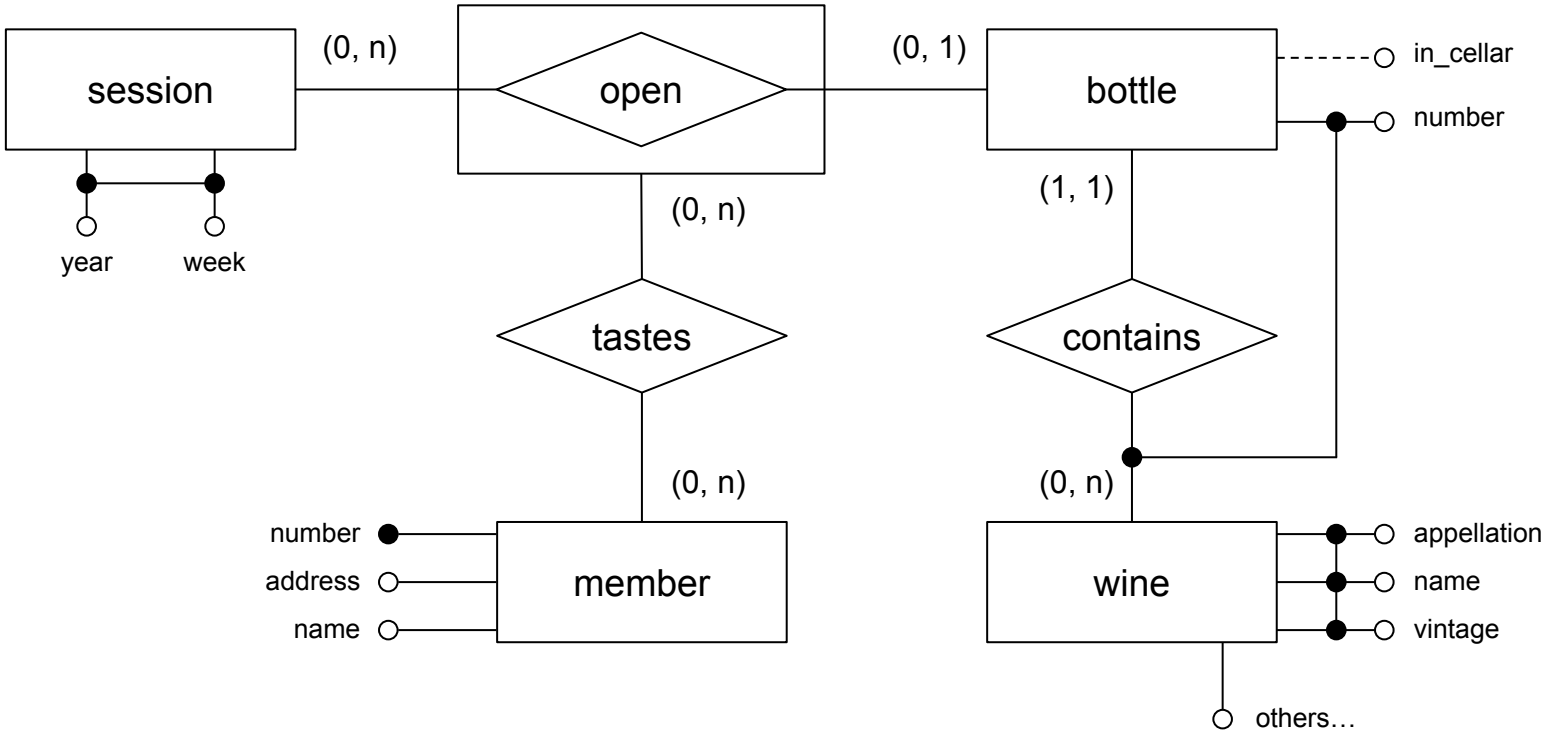
Question 2




```
CREATE TABLE tastes (
  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)
);
```

Question 2

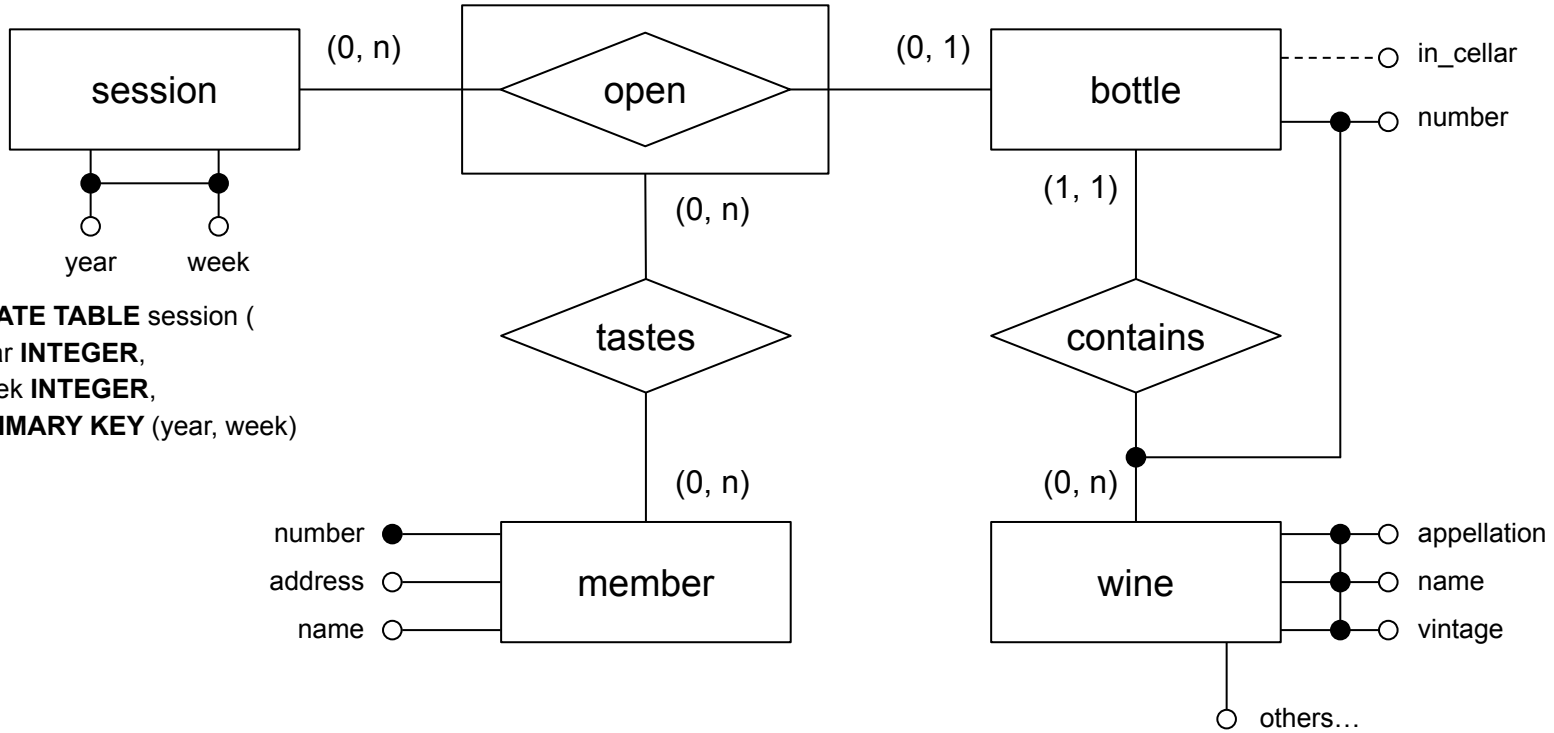
How many tables will we have at the end?



Which tables will be the same as before?

Question 2

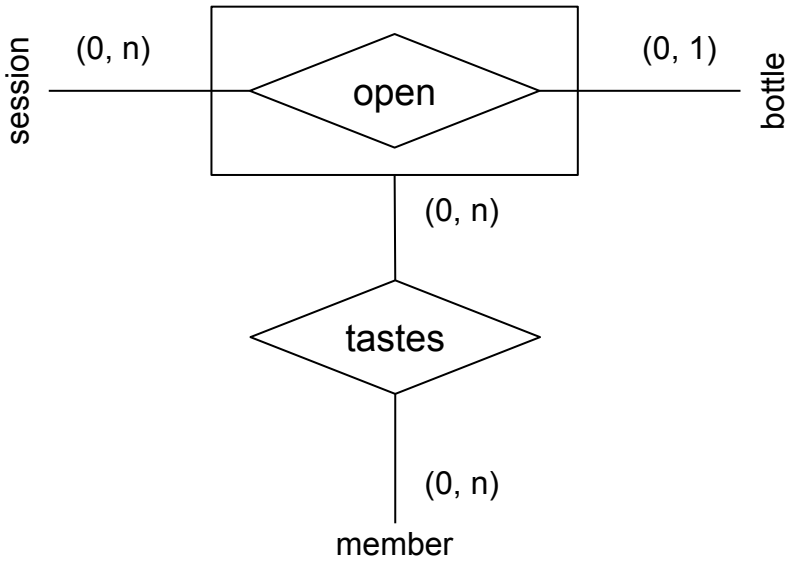
How many table will we have at the end?




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

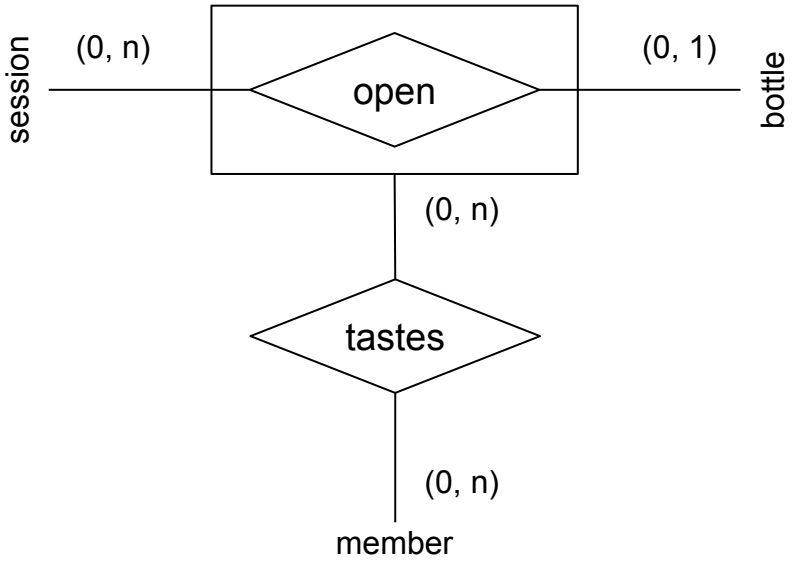
Which tables will be the same as before?

Question 2



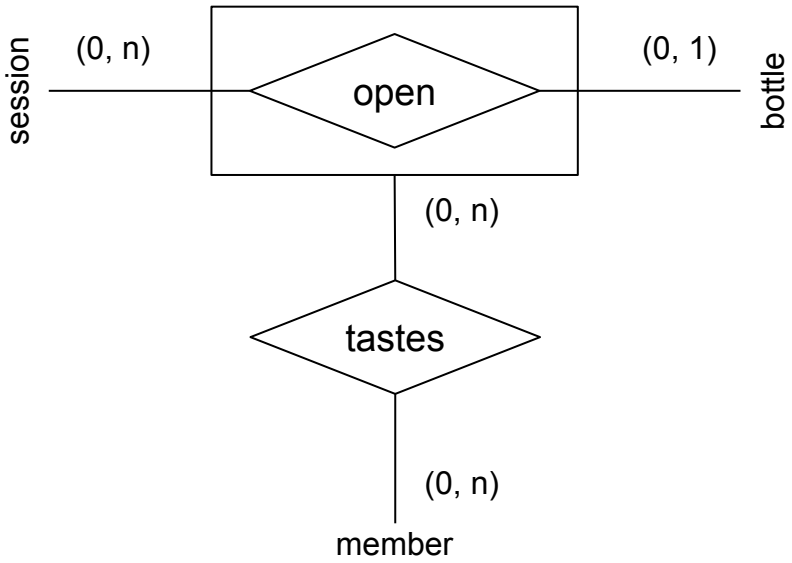

```
CREATE TABLE open (
  wine_name VARCHAR(32),
  appellation VARCHAR(32),
  vintage DATE,
  bottle_number INTEGER,
  session_year INTEGER NOT NULL,
  session_week INTEGER NOT NULL,
  PRIMARY KEY (bottle_number, wine_name, appellation, vintage),
  FOREIGN KEY (session_year, session_week)
    REFERENCES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

Question 2




```
CREATE TABLE tastes (
  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)
);
```

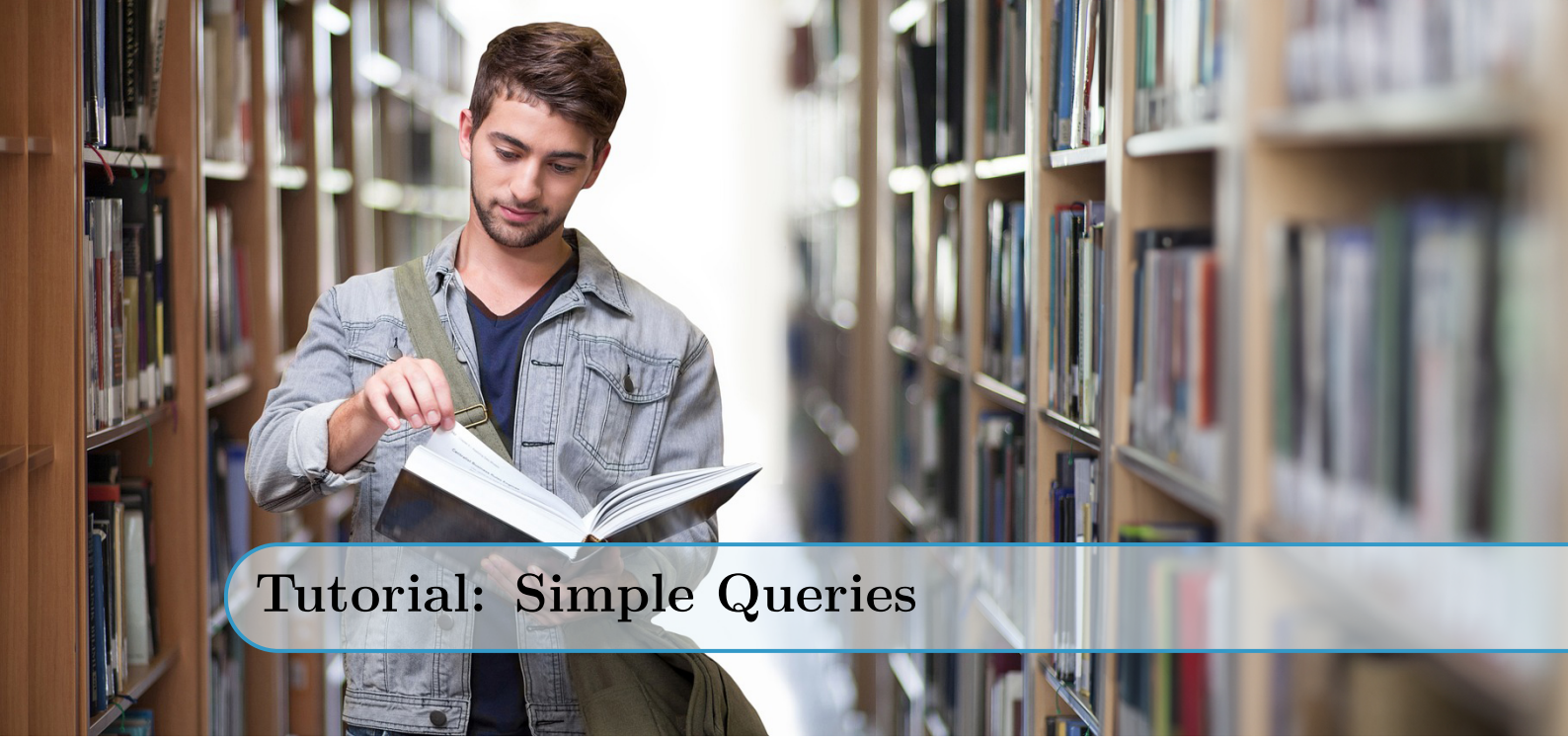
Question 2




```
CREATE TABLE tastes (
  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)
);
```

```
CREATE TABLE tastes (
  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)
);
```

See you next week!



Tutorial: Simple Queries

Students at the National University of Ngendipura (NUN) buy books for their studies. They also lend and borrow books to and from other students. Your company, Apasaja Private Limited, is commissioned by NUN Students Association (NUNStA) to implement an online book exchange system that records information about students, books that they own and books that they lend and borrow.

The database records the name, faculty, and department of each student. Each student is identified in the system by her email. The database also records the date at which the student joined the university (year attribute).

The database records the title, authors, publisher, year and edition and the ISBN-10 and ISBN-13 for each book. The International Standard Book Number, ISBN-10 or -13, is an industry standard for the unique identification of books. It is possible that the database records books that are not owned by any students (because the owners of a copy graduated or because the book was advised by a lecturer for a course but not yet purchased by any student.)

The database records the date at which a book copy is borrowed and the date at which it is returned. We refer to this information as a loan record.

For auditing purposes the database records information about the books, the copies and the owners of the copies as long as the owners are students or as there are loan records concerning the copies. For auditing purposes the database records information about graduated students as long as there are loan records concerning books that they owned.

This tutorial uses the schema and data for the database created in "Tutorial: Creating and Populating Tables" including all the updates done during the tutorial.

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.

Important: This tutorial is designed to be solved using **simple queries only**. This means your answers should not contain nested or aggregate queries.

1. **Single-Table Queries**.

   (a) Print the different departments.

   Comments:
   One alternative is the following.

   Code:

   ```
   SELECT d.department
   FROM department d;
   ```

   Notice that the query does not require `DISTINCT` keyword to eliminate duplicate. Duplicates are guaranteed not to occur because `department` is the `PRIMARY KEY` of the table `department`.

   Further notice that we are *aliasing* the table `department` with the name `d`. Additionally, we use `d.department` to refer to the column and not just `department`. Both are good practices to ensure the code's readability.

   (b) Print the different departments in which students are enrolled.

   Comments:
   There could be departments in which no student is enrolled. This is the case for the department of `"Undecidable Computation"`. We need to look into the `student` table instead.

   Code:

   ```
   SELECT DISTINCT s.department
   FROM student s;
   ```

   Notice that the query requires the `DISTINCT` keyword to eliminate duplicates since it is very likely that there is more than one student in most departments.

   It is also important to understand the wording of the question. We require **different departments**. Hence, there should be no duplicate department. Consider an alternative wording below.

   > Print the department of the different students that are enrolled.

   Here, the emphasis is on the **different students**. The query will be similar to above *except* that `DISTINCT` should not be used. A student is identified by their `email` and not `department`. So we should not make the `department` distinct.

(c) For each copy that has been borrowed and returned, print the ISBN13 of the book and the duration of the loan. Order the results in ascending order of the ISBN13 and descending order of duration. Remember to use only one single table.

Comments:

One possible answer:

Code: Alternative #1

```
SELECT l.book, l.returned - l.borrowed + 1 AS duration
FROM loan l
WHERE l.returned IS NOT NULL -- equivalently NOT (l.returned ISNULL)
ORDER BY l.book ASC, duration DESC;
```

Since `x IS NOT NULL` is equivalent to `NOT (x ISNULL)`, we have an alternative solution as shown in the comment above. Note that for sorting, `ASC` is the default but we *highly recommend* indicating it for clarity.

Extra challenge: Can you modify the query to also print the loan duration of copies that have **NOT** been returned, where the duration is calculated until the *current date*?

Answer: To handle `NULL` returned dates for unreturned copies, we can use `COALESCE` or `CASE-WHEN`. To get the current date, use the **variable** `CURRENT_DATE`.

Code: Alternative #2: COALESCE

```
SELECT l.book,
  (COALESCE(l.returned, CURRENT_DATE) - l.borrowed + 1) AS duration
FROM loan l
ORDER BY l.book ASC, duration DESC;
```

Code: Alternative #3: CASE-WHEN

```
SELECT l.book,
  ((CASE
    WHEN l.returned ISNULL THEN CURRENT_DATE
    ELSE l.returned
  END) - l.borrowed + 1) AS duration
FROM loan l
ORDER BY l.book ASC, duration DESC;
```

2. **Multi-Table Queries**.

   (a) For each loan of a book published by Wiley that has not been returned, print the title of the book, the name and faculty of the owner and the name and faculty of the borrower.

Comments:
We join the `PRIMARY KEY` and the `FOREIGN KEY` to stitch the tables together.

Code: Alternative #1: Joining all five tables

```
SELECT b.title,
  s1.name AS ownerName,
  d1.faculty AS ownerFaculty,
  s2.name AS borrowerName,
  d2.faculty AS borrowerFaculty
FROM loan l, book b, copy c,
  student s1, student s2,
  department d1, department d2
WHERE l.book = b.ISBN13
  AND c.book = l.book
  AND c.copy = l.copy
  AND c.owner = l.owner
  AND l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = d1.department
  AND s2.department = d2.department
  AND b.publisher = 'Wiley'
  AND l.returned ISNULL;
```

You can omit the table `copy` and the `copy` column since the existence of the corresponding rows and values is guaranteed by design (i.e., by the schema) and `PRIMARY KEY` constraints.

Code: Alternative #2: Omit table "copy"

```
SELECT b.title,
  s1.name AS ownerName,
  d1.faculty AS ownerFaculty,
  s2.name AS borrowerName,
  d2.faculty AS borrowerFaculty
FROM loan l, book b, -- no more copy table
  student s1, student s2,
  department d1, department d2
WHERE l.book = b.ISBN13
  AND l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = d1.department
  AND s2.department = d2.department
  AND b.publisher = 'Wiley'
  AND l.returned ISNULL;
```

The above query is also equivalent to:

Code: Alternative #3: INNER JOIN instead of cross product

```
SELECT b.title,
  s1.name AS ownerName,
  d1.faculty AS ownerFaculty,
  s2.name AS borrowerName,
  d2.faculty AS borrowerFaculty
FROM loan l
  INNER JOIN book b ON l.book = b.ISBN13
  INNER JOIN student s1 ON l.owner = s1.email
  INNER JOIN student s2 ON l.borrower = s2.email
  INNER JOIN department d1 ON s1.department = d1.department
  INNER JOIN department d2 ON s2.department = d2.department
WHERE b.publisher = 'Wiley' AND l.returned ISNULL;
```

In the last solution, the `ON` clause is used for joining the tables (i.e., connecting `PRIMARY KEY` to `FOREIGN KEY`) while the `WHERE` clause is used for additional filters. This is a good practice to improve code clarity. When using `INNER JOIN`, `ON` clause and `WHERE` clause can be interchangeable as long as the attributes make sense.

This is not the case for `OUTER JOIN`. You will need to think very carefully about the join condition for `OUTER JOIN`.

(b) Let us check the integrity of the data. Print the different emails of the students who borrowed or lent a copy of a book before they joined the University. There should not be any.

Comments:

One alternative is the following.

Code: Alternative #1

```
SELECT DISTINCT s.email
FROM loan l, student s
WHERE (s.email = l.borrower OR s.email = l.owner)
  AND l.borrowed < s.year;
```

Equivalently, we have the following solution by distributing `l.borrowed < s.year` into the disjunction.

Code: Alternative #2

```
SELECT DISTINCT s.email
FROM loan l, student s
WHERE (s.email = l.borrower AND l.borrowed < s.year)
   OR (s.email = l.owner AND l.borrowed < s.year);
   -- (x OR y) AND z  ===  (x AND z) OR (x AND y)
```

Other correct solutions can use `CROSS JOIN`, `INNER JOIN`, or `UNION`.

(c) Print the emails of the different students who borrowed or lent a copy of a book on the day that they joined the university.

Comments:
The following query is generally preferrable **BUT** it requires an explicit `DISTINCT` keyword. Additionally, it is not easily extensible to the next 2 questions.

Code:

```
SELECT DISTINCT s.email
FROM loan l, student s
WHERE (s.email = l.borrower OR s.email = l.owner)
  AND l.borrowed = s.year;
  -- Can you rewrite this with INNER JOIN?
```

Alternatively, we can use `UNION`. The `DISTINCT` keyword is not needed because `UNION` (as well as `INTERSECT` and `EXCEPT`) already eliminates duplicates.

Code: using UNION

```
SELECT s.email
FROM loan l, student s
WHERE s.email = l.borrower AND l.borrowed = s.year
UNION
SELECT s.email
FROM loan l, student s
WHERE s.email = l.owner AND l.borrowed = s.year;
```

(d) Print the emails of the different students who borrowed and lent a copy of a book on the day that they joined the university.

Comments:
We could also make use of set operators like the previous part, using `INTERSECT` here.

Code: using INTERSECT

```
SELECT s.email
FROM loan l, student s
WHERE s.email = l.borrower AND l.borrowed = s.year
INTERSECT
SELECT s.email
FROM loan l, student s
WHERE s.email = l.owner AND l.borrowed = s.year;
```

An equivalent query without `INTERSECT` is more complicated as it requires two `loan` tables.

Code: no INTERSECT

```
SELECT DISTINCT s.email
FROM loan l1, loan l2, student s
WHERE s.email = l1.borrower AND l1.borrowed = s.year
  AND s.email = l2.owner AND l2.borrowed = s.year;
  -- Can you rewrite this with INNER JOIN?
```

We cannot simply use a single `loan` table (let's alias that table as `l`) because the following condition has a different meaning.

```
s.email = l.borrower AND s.email = l.owner
```

The above condition implies `l.borrower = l.owner` by *transitivity* of equality. This means, the query only selects students who borrowed their own books, which is only a subset of the solution.

(e) Print the emails of the different students who borrowed but did not lend a copy of a book on the day that they joined the university.

Comments:

Again we can use a set operator, `EXCEPT`.

Code: using EXCEPT

```
SELECT s.email
FROM loan l, student s
WHERE s.email = l.borrower AND l.borrowed = s.year
EXCEPT
SELECT s.email
FROM loan l, student s
WHERE s.email = l.owner AND l.borrowed = s.year;
```

There is no alternative simple query without using `EXCEPT`. We need to use *nested* or *aggregate* queries to write alternative answers to this type of question.

(f) Print the different ISBN13 of the books that have never been borrowed.

Comments:

There is no such book. You may create some records yourself if you want a non-empty result.

Code: Alternative #1: EXCEPT

```
SELECT b.ISBN13
FROM book b
EXCEPT
SELECT l.book
FROM loan l;
```

We can also use `OUTER JOIN`. Note that this introduces `NULL` values. We can then check for those `NULL` values using `ISNULL`.

Code: Alternative #2: OUTER JOIN

```
SELECT b.ISBN13
FROM book b LEFT OUTER JOIN loan l ON b.ISBN13 = l.book
WHERE l.book ISNULL;
```

Comments:

Queries should solve the corresponding question regardless of the current data set being used. This means your code should still work with other data sets. We will test with other data sets satisfying the schema.

This means *hardcoding* is strictly not allowed. Use of constants is prohibited *unless* they are explicitly mentioned in the question. For example, if "Changi Airport" is mentioned, you may use it (e.g., `'Changi Airport'`). However, other related constants (e.g., `'Singapore'` because Changi Airport is in Singapore or `'SIN'` as the IATA code for Changi Airport) are not allowed.

Note that SQL queries tend to be short with small changes affecting the entire output. This means, it is difficult to argue about the "logic" of the query especially when there are partial marks. For partial marks to be awarded (*if any*), (i) the query must be executable, (ii) the columns must be correct (i.e., correct name after aliasing, correct data type, correct order, etc), and (iii) the difference in the rows must be minimal.

Codes that cannot be executed may receive immediate 0 marks.

Finally, although SQL queries are case-insensitive, we highly recommend using uppercase for keywords. This eases reading the query. However, in the industry, please follow the convention set by the company.

References

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

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

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

[4] *W3schools Online Web Tutorials*. `https://www.w3schools.com/`. [Online; last accessed 2025].

Tutorial: Aggregate and Nested Queries

Students at the National University of Ngendipura (NUN) buy books for their studies. They also lend and borrow books to and from other students. Your company, Apasaja Private Limited, is commissioned by NUN Students Association (NUNStA) to implement an online book exchange system that records information about students, books that they own and books that they lend and borrow.

The database records the name, faculty, and department of each student. Each student is identified in the system by her email. The database also records the date at which the student joined the university (year attribute).

The database records the title, authors, publisher, year and edition and the ISBN-10 and ISBN-13 for each book. The International Standard Book Number, ISBN-10 or -13, is an industry standard for the unique identification of books. It is possible that the database records books that are not owned by any students (because the owners of a copy graduated or because the book was advised by a lecturer for a course but not yet purchased by any student.)

The database records the date at which a book copy is borrowed and the date at which it is returned. We refer to this information as a loan record.

For auditing purposes the database records information about the books, the copies and the owners of the copies as long as the owners are students or as there are loan records concerning the copies. For auditing purposes the database records information about graduated students as long as there are loan records concerning books that they owned.

This tutorial uses the schema and data for the database created in "Tutorial: Creating and Populating Tables" including all the updates done during the tutorial.

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. **Aggregate Queries**.

   (a) How many loans involve an owner and a borrower from the same department?

   Comments:
   One alternative is the following.

   Code: Query

```
SELECT COUNT(*)
FROM loan l, student s1, student s2
WHERE l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = s2.department;
```

   (b) For each faculty, print the number of loans that involve an owner and a borrower from this faculty?

   Comments:
   One alternative is the following.

   Code: Query

```
SELECT d1.faculty, COUNT(*)
FROM loan l, student s1, student s2, department d1, department d2
WHERE l.owner = s1.email
  AND l.borrower = s2.email
  AND s1.department = d1.department
  AND s2.department = d2.department
  AND d1.faculty = d2.faculty
GROUP by d1.faculty;
```

   (c) What are the average and the standard deviation [3] of the duration of a loan in days?

   Comments:
   One alternative is the following.

   Code: Alternative #1

```
SELECT CEIL(AVG((CASE
      WHEN l.returned ISNULL THEN CURRENT_DATE
      ELSE l.returned
    END) - l.borrowed + 1)),
  CEIL(STDDEV_POP((CASE
      WHEN l.returned ISNULL THEN CURRENT_DATE
      ELSE l.returned
    END ) - l.borrowed + 1))
FROM loan l;
```

   Another is the following.

Code: Alternative #2

```
SELECT CEIL(AVG(temp.duration)), CEIL(STDDEV_POP(temp.duration))
FROM (
    SELECT((CASE
        WHEN l.returned ISNULL THEN CURRENT_DATE
        ELSE l.returned
      END ) - l.borrowed + 1) AS duration
    FROM loan l
) AS temp;
```

There is another alternative using `COALESCE` but it is left as an exercise.

2. **Nested Queries**.

   (a) Print the titles of the different books that have never been borrowed. Use a nested query.

Comments:
There is no such book if you did not inserted one during the previous tutorial. You can create some records if you want a non-empty result.

Code: Alternative #1

```
SELECT b.title
FROM book b
WHERE b.ISBN13 NOT IN (
  SELECT l.book
  FROM loan l);
```

Equivalently by definition of `NOT IN`, we have the following.

Code: Alternative #2

```
SELECT b.title
FROM book b
WHERE b.ISBN13 <> ALL (
  SELECT l.book
  FROM loan l);
-- x NOT IN s  ===  forall y IN S : x <> y
```

Always use one of the quantifiers `ALL` or `ANY` in front of subqueries wherever possible even though some systems may be lenient with this requirement. You should still use the quantifiers `ALL` or `ANY` even if the result is only a single value (i.e., scalar subquery). This may prevent careless mistakes.

Note that it is possible for the result to contain the same title several times (*since there could be different books with the same title*). There is no need to use `DISTINCT` as the query asks for the different books, not for the different titles. In fact, using `DISTINCT` is a mistake here.

(b) Print the name of the different students who own a copy of a book that they have never lent to anybody.

Comments:

One alternative is the following.

Code: Alternative #1

```
SELECT s.name
FROM student s
WHERE s.email IN (
  SELECT c.owner
  FROM copy c
  WHERE NOT EXISTS (
    SELECT *
    FROM loan l
    WHERE l.owner = c.owner
      AND l.book = c.book
      AND l.copy = c.copy));
```

Equivalently by definition of `IN`, we have the following.

Code: Alternative #2

```
SELECT s.name
FROM student s
WHERE s.email = ANY (
  SELECT c.owner
  FROM copy c
  WHERE NOT EXISTS (
    SELECT *
    FROM loan l
    WHERE l.owner = c.owner
      AND l.book = c.book
      AND l.copy = c.copy));
```

The query can also be written as follows but the highlighted tuple construction does not always work on other systems than PostgreSQL.

Code: Alternative #3: Tuple Construction

```
SELECT s.name
FROM student s
WHERE s.email IN (
  SELECT c.owner
  FROM copy c
  WHERE (c.owner, c.book, c.copy) NOT IN ( -- this line may not work
    SELECT l.owner, l.book, l.copy
    FROM loan l));
```

The following **incorrect** query would print several times the names of students who own several copies that have never been borrowed if such cases occurred. We would not be able to differentiate the repeated names of students who own several copies that have never been borrowed from the repeated names of different students

with the same name.

Code: Incorrect Query

```
-- INCORRECT
SELECT s.name
FROM student s, copy c
WHERE s.email = c.owner
AND NOT EXISTS (
  SELECT *
  FROM loan l
  WHERE l.owner = c.owner
    AND l.book = c.book
    AND l.copy = c.copy);
```

(c) For each department, print the names of the students who lent the most.

Comments:

One alternative is the following.

Code: Query

```
SELECT s.department, s.name, COUNT(*)
FROM student s, loan l
WHERE l.owner = s.email
GROUP BY s.department, s.email, s.name
HAVING COUNT(*) >= ALL (
  SELECT COUNT(*)
  FROM student s1, loan l1
  WHERE l1.owner = s1.email
    AND s.department = s1.department
  GROUP BY s1.email);
```

Notice that there are multiple such students in the department of Language (that is why one should almost never use *TOP N* queries, for instance, using `LIMIT` [4]).

If we create a new department called Undecidable Computations with some students who never lent any book, what would happen? If there were students in the department of Undecidable Computations, should we print all of them or none of them? They would all have lent zero books, which would be the maximum in the department... We should print them all (using `OUTER JOIN`, `CASE-WHEN`, and `ISNULL` to consider the cases of 0 loan). Are there students who never lent a book?

Note that we need to group by `s.name` in order to print the name although there is no ambiguity. Some systems, like PostgreSQL, relax this rule. It is recommended not to use this relaxation for the sake of portability.

(d) Print the emails and the names of the different students who borrowed all the books authored by Adam Smith.

Comments:

There is no such student. We can create one student with the corresponding records as follows.

Code: Test Case

```
INSERT INTO loan VALUES
  ('nihanran1989@msn.com', 'choyweixiang2011@gmail.com',
   '978-0553585971', 1, '2024-03-10', NULL);
```

Code: Query

```
SELECT s.email, s.name
FROM student s
WHERE NOT EXISTS (
  SELECT *
  FROM book b
  WHERE authors = 'Adam Smith'
    AND NOT EXISTS (
      SELECT *
      FROM loan l
      WHERE l.book = b.ISBN13
      AND l.borrower = s.email));
```

This is the typical query for *universal quantification*.

Comments:

As a good practice, try to make your query easily readable and understandable. One way is to use *indentation* to "group" the nested queries in such a way that queries in the same nesting have the same indentation level.

References

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

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

[3] *PostgreSQL Docs: Aggregate Functions*. `https://www.postgresql.org/docs/current/functions-aggregate.html`. [Online; last accessed 2025].

[4] *PostgreSQL Docs: SELECT (LIMIT)*. `https://www.postgresql.org/docs/current/sql-select.html#SQL-LIMIT`. [Online; last accessed 2025].

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

[6] *W3schools Online Web Tutorials*. `https://www.w3schools.com/`. [Online; last accessed 2025].

Tutorial: Relational Algebra

Students at the National University of Ngendipura (NUN) buy books for their studies. They also lend and borrow books to and from other students. Your company, Apasaja Private Limited, is commissioned by NUN Students Association (NUNStA) to implement an online book exchange system that records information about students, books that they own and books that they lend and borrow.

The database records the name, faculty, and department of each student. Each student is identified in the system by her email. The database also records the date at which the student joined the university (year attribute).

The database records the title, authors, publisher, year and edition and the ISBN-10 and ISBN-13 for each book. The International Standard Book Number, ISBN-10 or -13, is an industry standard for the unique identification of books. It is possible that the database records books that are not owned by any students (because the owners of a copy graduated or because the book was advised by a lecturer for a course but not yet purchased by any student.)

The database records the date at which a book copy is borrowed and the date at which it is returned. We refer to this information as a loan record.

For auditing purposes the database records information about the books, the copies and the owners of the copies as long as the owners are students or as there are loan records concerning the copies. For auditing purposes the database records information about graduated students as long as there are loan records concerning books that they owned.

This tutorial uses the schema from "Tutorial: Creating and Populating Tables" including all the updates done during the tutorial.

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.

> **Comments:**
> Relational algebra expressions below are written in a way to ease reading. We added square brackets to clearly show the conditions/attributes/etc.

1. **Relational Algebra**.

   (a) Find the different departments in School of Computing.

   Comments:

   One possible relational algebra expression.

   $\pi_{\texttt{[d.department]}}(\sigma_{\texttt{[d.faculty='School of Computing']}}(\rho(\texttt{department},\texttt{d})))$

   This corresponds to the following SQL query.

   Code: Query

   ```
   SELECT d.department
   FROM department d
   WHERE d.faculty = 'School of Computing';
   ```

   Non-symmetric set difference can be used to implement other universally quantified queries in algebra but these queries are quite complicated to write.

   (b) Let us check the integrity of the data. Find the emails of the students who borrowed or lent a copy of a book before they joined the university. There should not be any.

   Comments:

   One possible relational algebra expression.

   $$\pi_{\texttt{[s.email]}}(\sigma_{\texttt{[(s.email=l.borrower}\vee\texttt{s.email=l.owner)}\wedge\texttt{(l.borrowed<s.year)]}}(\rho(\texttt{student},\texttt{s}) \times \rho(\texttt{loan},\texttt{l})))$$

   Code: Alternative #1

   ```
   SELECT DISTINCT s.email
   FROM student s, loan l
   WHERE (s.email = l.borrower OR s.email = l.owner)
     AND l.borrowed < s.year;
   ```

   An alternative is to use $\bowtie$ (i.e., `INNER JOIN`).

   $$\pi_{\texttt{[s.email]}}(\rho(\texttt{student},\texttt{s}) \bowtie_{\texttt{[(s.email=l.borrower}\vee\texttt{s.email=l.owner)}\wedge\texttt{(l.borrowed<s.year)]}} \rho(\texttt{loan},\texttt{l}))$$

   Code: Alternative #2: INNER JOIN

   ```
   SELECT DISTINCT s.email
   FROM student s
     INNER JOIN loan l ON (s.email = l.borrower OR s.email = l.owner)
                       AND l.borrowed < s.year;
   ```

   We can also combine the $\bowtie$ with σ by moving some of the conditions around. This is fine as $\bowtie$ corresponds to `INNER JOIN`. Such technique may not be possible for `OUTER JOIN`.

   Finally, we can use `UNION`.

$\pi_{\texttt{[s1.email]}}(\sigma_{\texttt{[s1.email=l1.borrower}\wedge\texttt{l1.borrowed<s1.year]}}(\rho(\texttt{student},\texttt{s1}) \times \rho(\texttt{loan},\texttt{l1})))$
$\cup$
$\pi_{\texttt{[s2.email]}}(\sigma_{\texttt{[s2.email=l2.owner}\wedge\texttt{l2.borrowed<s2.year]}}(\rho(\texttt{student},\texttt{s2}) \times \rho(\texttt{loan},\texttt{l2})))$

Code: Alternative #3: UNION

```
SELECT s1.email
FROM loan l1, student s1
WHERE s1.email = l1.borrower
  AND l1.borrowed < s1.year
UNION
SELECT s2.email
FROM loan l2, student s2
WHERE s2.email = l2.owner
  AND l2.borrowed < s2.year;
```

(c) Print the emails of the students who borrowed but did not lend a copy of a book on the day that they joined the university.

Comments:

We use non-symmetric set difference. There are two convention for this operation. The first is $x-y$ and the second is $x \setminus y$. Both mean the same thing. Our convention is typically the first (i.e., $x - y$).

$\pi_{\texttt{[s1.email]}}(\sigma_{\texttt{[s1.email=l1.borrower}\wedge\texttt{l1.borrowed=s1.year]}}(\rho(\texttt{student},\texttt{s1}) \times \rho(\texttt{loan},\texttt{l1})))$
$-$
$\pi_{\texttt{[s2.email]}}(\sigma_{\texttt{[s2.email=l2.owner}\wedge\texttt{l2.borrowed=s2.year]}}(\rho(\texttt{student},\texttt{s2}) \times \rho(\texttt{loan},\texttt{l2})))$

Code: Query

```
SELECT s1.email
FROM loan l1, student s1
WHERE s1.email = l1.borrower
  AND l1.borrowed = s1.year
EXCEPT
SELECT s2.email
FROM loan l2, student s2
WHERE s2.email = l2.owner
  AND l2.borrowed = s2.year;
```

2. **Universal Quantification**.

   (a) Print the emails and the names of the different students who borrowed all the books authored by Adam Smith.

Comments:

We know one possible SQL query.

Code: Nested Query

```
SELECT s.email, s.name
FROM student s
WHERE NOT EXISTS (
  SELECT *
  FROM book b
  WHERE authors = 'Adam Smith'
    AND NOT EXISTS (
      SELECT *
      FROM loan l
      WHERE l.book = b.ISBN13
      AND l.borrower = s.email));
```

However, this SQL query contains `NOT EXISTS`, which is not directly translatable to relational algebra. What we can do is to break down the problem into the following subproblems.

1. Find the emails and names of the different students such that there is at least one book authored by Adam Smith that the student has not borrowed.
2. The solution is then the emails and names of the different students (including those who have not borrowed any books) *except* the students in Step 1.

Step 1 can be split even further.

a. Find all combinations of emails, names, and ISBN13 where emails and names are all students and ISBN13 are all books by Adam Smith.

b. Step 1's answer is then the result of Step a *except* the combinations for the students who borrowed books by Adam Smith.

c. We need to project the attributes from Step b to include only emails and names.

Putting everything together, we get the following relational algebra expressions. To ease reading, we add $:=$ symbol to split queries into smaller subqueries that we can refer to at a later time.

$Q_1 := \pi_{[\texttt{b1.ISBN13}]}(\sigma_{[\texttt{b1.authors='Adam Smith'}]}(\rho(\texttt{book}, \texttt{b1})))$

$\Rightarrow$ The ISBN13 of all books authored by Adam Smith.

$Q_2 := \pi_{[\texttt{s1.email,s1.name}]}(\rho(\texttt{student}, \texttt{s1})) \times Q_1$

$\Rightarrow$ Relational algebra query for Step a. **Note:** The columns are `[email, name, ISBN13]`.

$$Q_3 := \pi_{[\texttt{s2.email,s2.name,b2.ISBN13}]}(\rho(\texttt{loan}, \texttt{l2}) \bowtie_{[\texttt{l2.book=b2.ISBN13} \wedge \texttt{b2.authors='Adam Smith'}]} \rho(\texttt{book}, \texttt{b2}) \bowtie_{[\texttt{l2.borrower=s2.email}]} \rho(\texttt{student}, \texttt{s2}))$$

$\Rightarrow$ Students who have borrowed books by Adam Smith.

$Q_4 \coloneqq \pi_{[\texttt{s2.email,s2.name}]}(Q_2 - Q_3)$

$\Rightarrow$ Relational algebra query for Step b followed by projection. This is also the final answer for Step 1. **Note:** Q_2 and Q_3 are *union-compatible*. Now, the columns are `[email, name]`.

$Q_5 \coloneqq \pi_{[\texttt{s3.email,s3.name}]}(\rho(\texttt{student}, \texttt{s3})) - Q_4$

$\Rightarrow$ **Note:** The projection on `s3` is to ensure *union-compatibility* with Q_4.

The answer is then Q_5.

The corresponding SQL query is shown below.

Code: Query from Relational Algebra

```
SELECT s3.email, s3.name
FROM student s3
EXCEPT
SELECT tmp.email, tmp.name
FROM (
  SELECT s1.email, s1.name, b1.ISBN13
  FROM student s1, book b1
  WHERE b1.authors = 'Adam Smith'
  EXCEPT
  SELECT s2.email, s2.name, b2.ISBN13
  FROM student s2, book b2, loan l2
  WHERE b2.authors = 'Adam Smith'
    AND s2.email = l2.borrower
    AND b2.ISBN13 = l2.book
) AS tmp;
```

References

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

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