CSC7072: Databases, fall 2015

Dr. Kim Bauters

SQL assignment & class test feedback

Feedback

SQL assignment

some typical errors:

question 2: retrieve the last name […] and salary […] of every employee in ascending order

```
SELECT last_name, salary
    FROM employee
        ORDER BY last_name ASC
```

fine without, ***but for written exam remember*** ASC or DESC

Feedback

SQL assignment

some typical errors:

question 4: retrieve the name and address of all employees who work on the Database Systems project

```
SELECT name, address
    FROM employee
        JOIN works_on ON employee.id = works_on.employee_id
            JOIN project ON works_on.project_no = project.project_no
                WHERE project.name = “Database Systems”
```

some used *branch* instead of *works_on* !
i.e. employees working in the branch to which
the “Database Systems” project is assigned

Feedback

SQL assignment

some typical errors:

question 5: select those projects to which at least one employee is assigned [;…] use EXISTS […]

```
SELECT project_name, project_no
    FROM project
        WHERE EXISTS( ... );
```

$\hookrightarrow$ all the projects are returned for just about any subquery you put here

most had this one wrong …

Feedback

SQL assignment

some typical errors:

question 5: select those projects to which at least one employee is assigned [;…] use EXISTS […]

```
SELECT project_name, project_no
    FROM project
        WHERE EXISTS(SELECT COUNT(employee_id)>=1
                                FROM works_on);
```

Feedback

SQL assignment

some typical errors:

question 5: select those projects to which at least one employee is assigned [;…] use EXISTS […]

```
SELECT project_name, project_no
    FROM project
        WHERE EXISTS(SELECT COUNT(employee_id)>=1
                                FROM works_on);
```

meaning of subquery: return TRUE if at least one employee works on some project

Feedback

SQL assignment

some typical errors:

question 5: select those projects to which at least one employee is assigned [;…] use EXISTS […]

```
SELECT project_name, project_no
    FROM project
        WHERE EXISTS(SELECT COUNT(employee_id)>=1
                                  FROM works_on);
```

meaning of subquery: return TRUE if at least one employee works on some project

meaning of query: get the project information of each project if *an* employee is working on *a(n) (unrelated)* project

Feedback

SQL assignment

some typical errors:

question 5: select those projects to which at least one employee is assigned [;…] use EXISTS […]

Feedback

SQL assignment

some typical errors:

question 5: select those projects to which at least one employee is assigned [;…] use EXISTS […]

```
SELECT project_name
    FROM project
        WHERE EXISTS(
            SELECT project_no
                FROM works_on
                    WHERE project.project_no=works_on.project_no);
```

Feedback

SQL assignment

some typical errors:

question 8: What is the largest salary in this company and who earns it?

```
SELECT first_name, last_name, MAX(salary)
    FROM employee
```

Feedback

SQL assignment

some typical errors:

question 8: What is the largest salary in this company and who earns it?

```
SELECT first_name, last_name, MAX(salary)
    FROM employee
```

$\hookrightarrow$ you can never have an aggregate function …

Feedback

SQL assignment

some typical errors:

question 8: What is the largest salary in this company and who earns it?

```
SELECT first_name, last_name, MAX(salary)
  FROM employee
```

you can never have an aggregate function …

… with other plain attributes …

Feedback

SQL assignment

some typical errors:

question 8: What is the largest salary in this company and who earns it?

```
SELECT first_name, last_name, MAX(salary)
   FROM employee
```

you can never have an aggregate function …

… with other plain attributes …

… without a GROUP BY to create groups for those plain attributes

Feedback

SQL assignment

some typical errors:

question 8: What is the largest salary in this company and who earns it?

```
SELECT first_name, last_name, MAX(salary)
    FROM employee
```

you can never have an aggregate function …

… with other plain attributes …

… without a GROUP BY to create groups for those plain attributes

meaning: get *some* name, and the maximum salary

Feedback

SQL assignment

some typical errors:

question 8: What is the largest salary in this company and who earns it?

```
SELECT first_name, last_name, salary
    FROM employee
        WHERE salary = (SELECT MAX(salary)
                                FROM employee)
```

Feedback

SQL assignment

some typical errors:

question 12: […] departments with no projects associated with them should also be listed

Feedback

SQL assignment

some typical errors:

question 12: […] departments with no projects associated with them should also be listed

most got it, but sometimes incorrect use of LEFT/RIGHT OUTER JOIN

Feedback

outer join

what is an outer join?

- extends the normal join operation to avoid loss of information
- computes the normal join; then adds tuples from one relation that do not match tuples in the other relation to the result of the join using *null* values
- can be both a *left*, *right* or *full* join

returns *all rows from left table*, with matching rows from right table and padded with nulls if necessary

returns *all rows from right table*, with matching rows from left table and padded with nulls if necessary

Feedback

outer join examples

course

<u>course_id</u>	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

<u>course_id</u>	<u>prereq_id</u>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*SELECT * FROM course LEFT OUTER JOIN prereq USING(course_id)*

<u>course_id</u>	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

Feedback

outer join examples

course

<u>**course_id**</u>	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

<u>**course_id**</u>	<u>**prereq_id**</u>
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*SELECT * FROM course RIGHT OUTER JOIN prereq USING(course_id)*

<u>**course_id**</u>	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	*CS-101*

Feedback

outer join examples

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	**prereq_id**
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*SELECT * FROM course FULL OUTER JOIN prereq USING(course_id)*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	*CS-101*
CS-315	Robotics	Comp. Sci.	3	*null*

Feedback

ER class test

time management is everything!

  remember that exam is bigger problem, *and* schema conversion

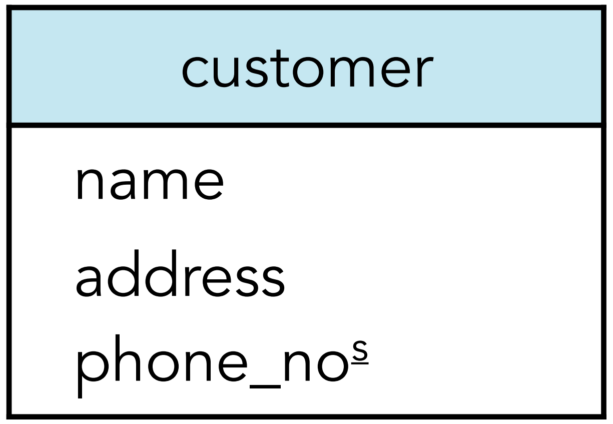
very forgiving for class test, but exam will be tougher

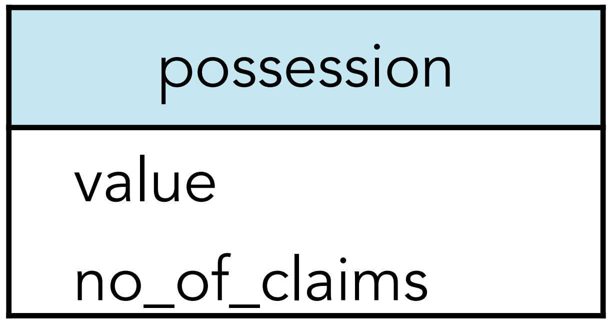
Feedback

ER class test

how I corrected:

- $^1/_3$ of points on entities
- about half of those points lost if missing entities
- lose 25% if you mark both *accident* and *claim* as entities



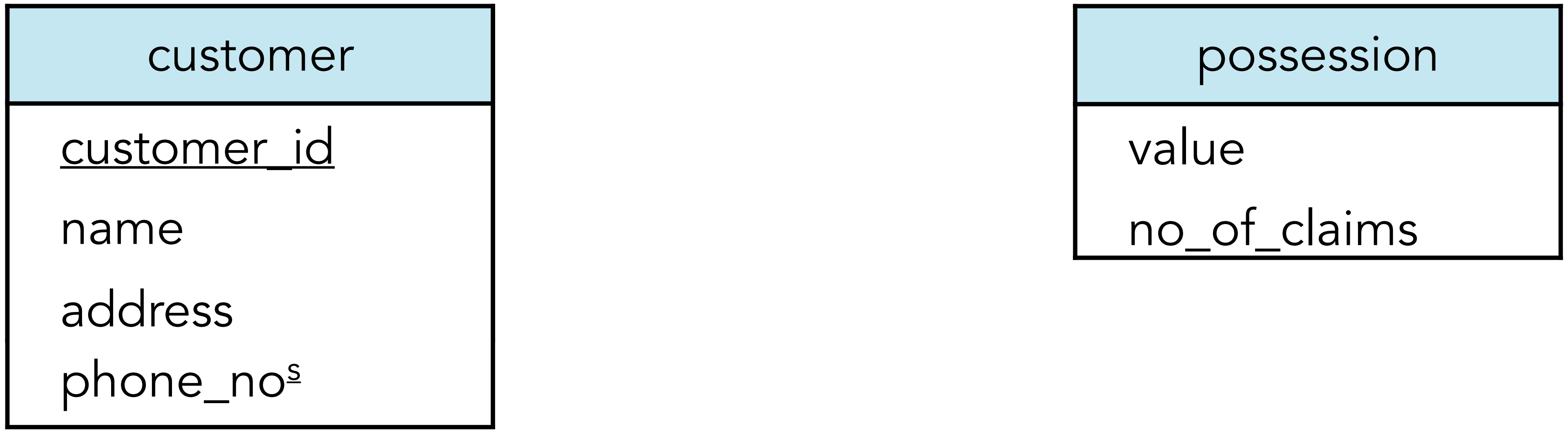


Feedback

ER class test

how I corrected:

- $^1/_3$ of points on entities
- about half of those points lost if missing entities
- lose 25% if you mark both *accident* and *claim* as entities

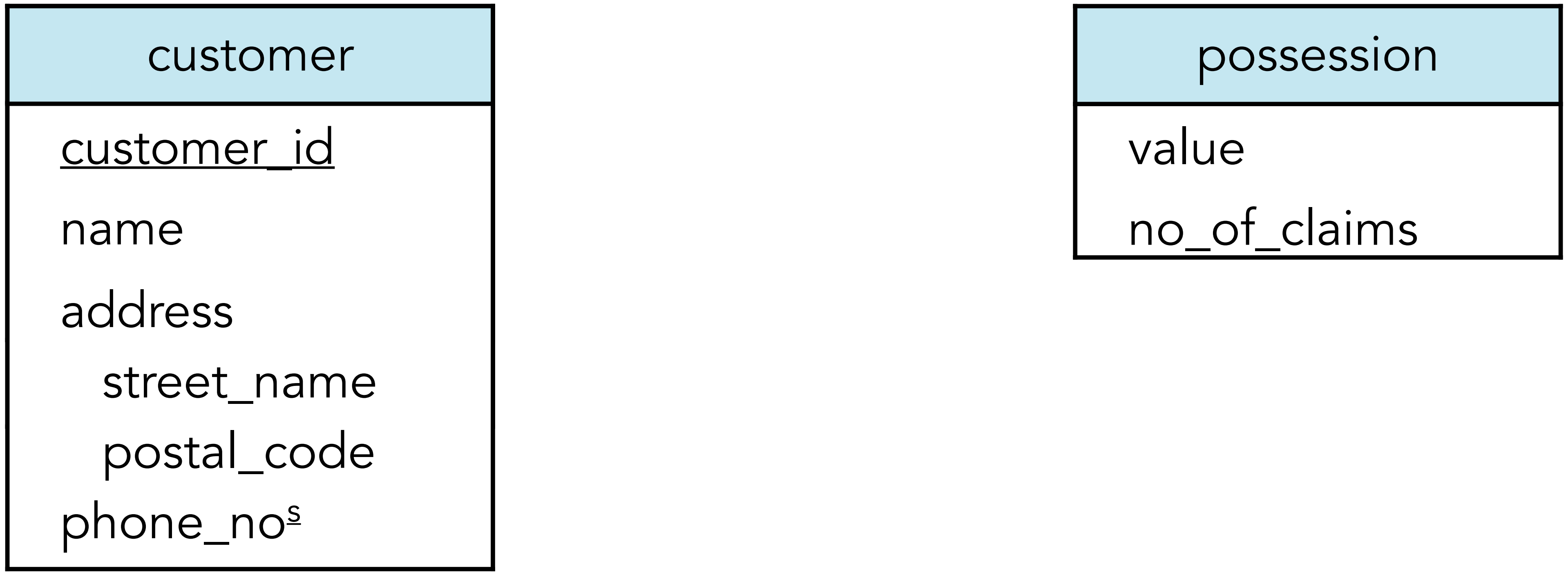


Feedback

ER class test

how I corrected:

- $^1/_3$ of points on entities
- about half of those points lost if missing entities
- lose 25% if you mark both *accident* and *claim* as entities

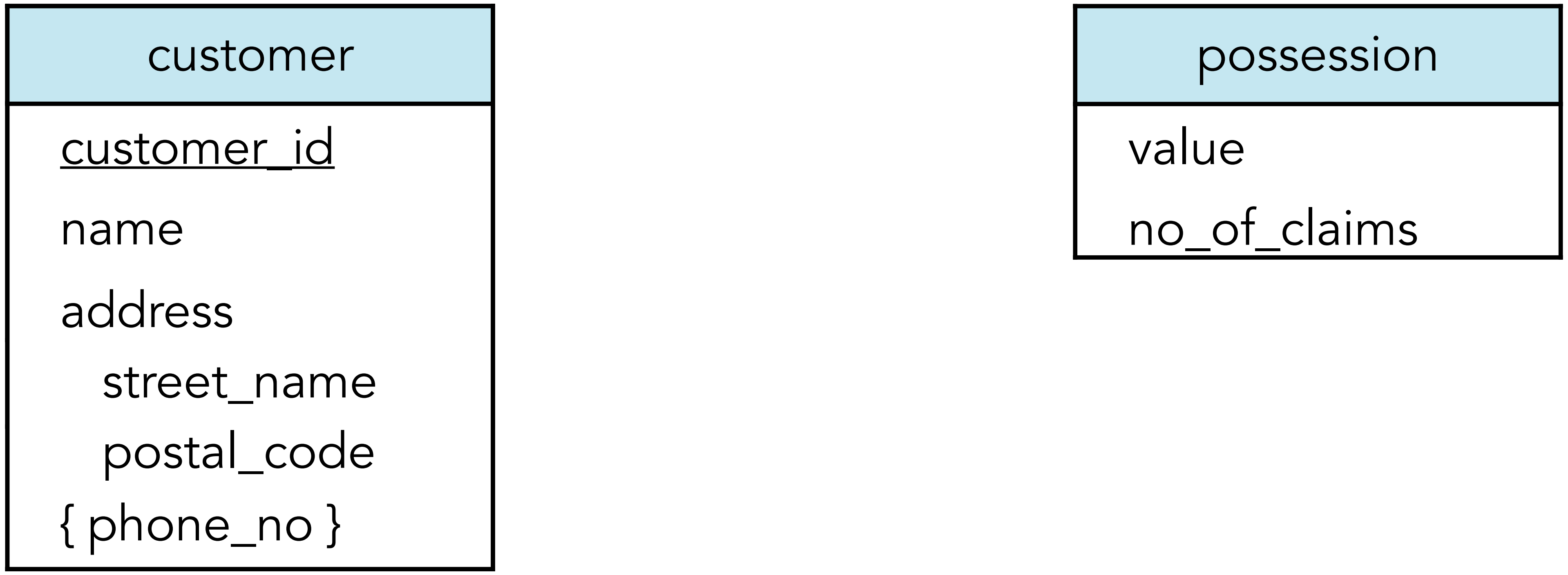


Feedback

ER class test

how I corrected:

- $^1/_3$ of points on entities
- about half of those points lost if missing entities
- lose 25% if you mark both *accident* and *claim* as entities

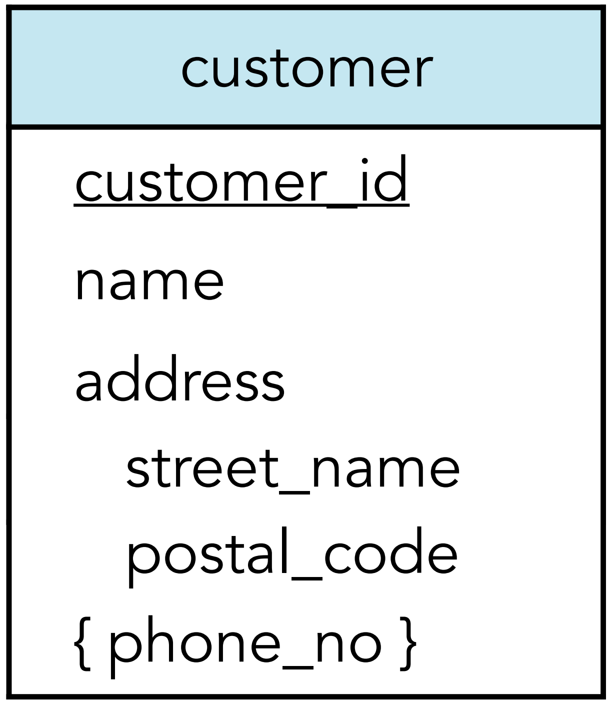


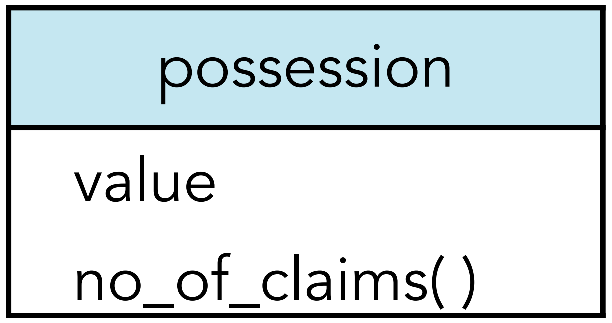
Feedback

ER class test

how I corrected:

- $^{1}/_{3}$ of points on entities
- about half of those points lost if missing entities
- lose 25% if you mark both *accident* and *claim* as entities





Feedback

ER class test

how I corrected:

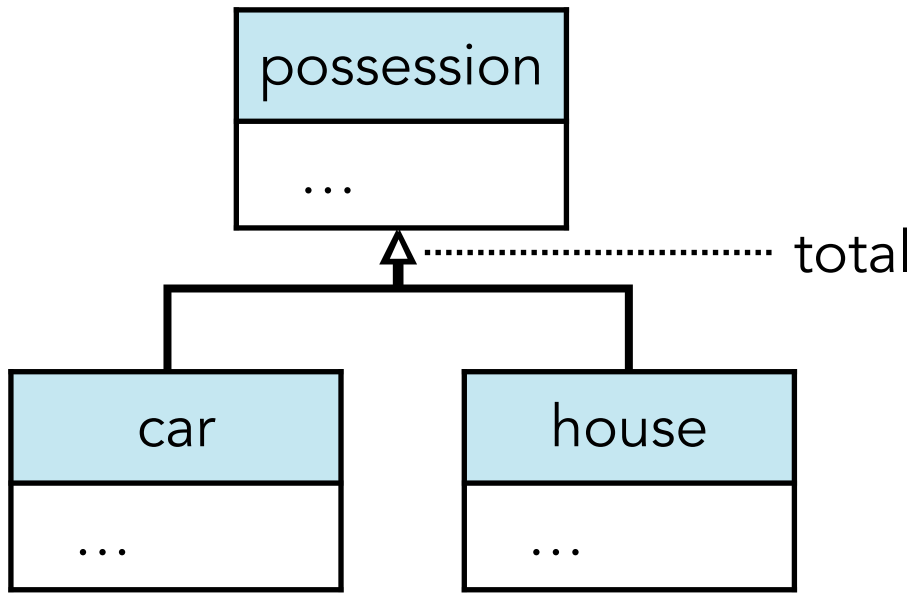
- $^1/_3$ of remaining points on relationships
- about half of those points lost if you missed total is-a relationship
- other half on incorrect notations/missing relationships/attributes

Feedback

ER class test

how I corrected:

- $^1/_3$ of remaining points on relationships
- about half of those points lost if you missed total is-a relationship
- other half on incorrect notations/missing relationships/attributes

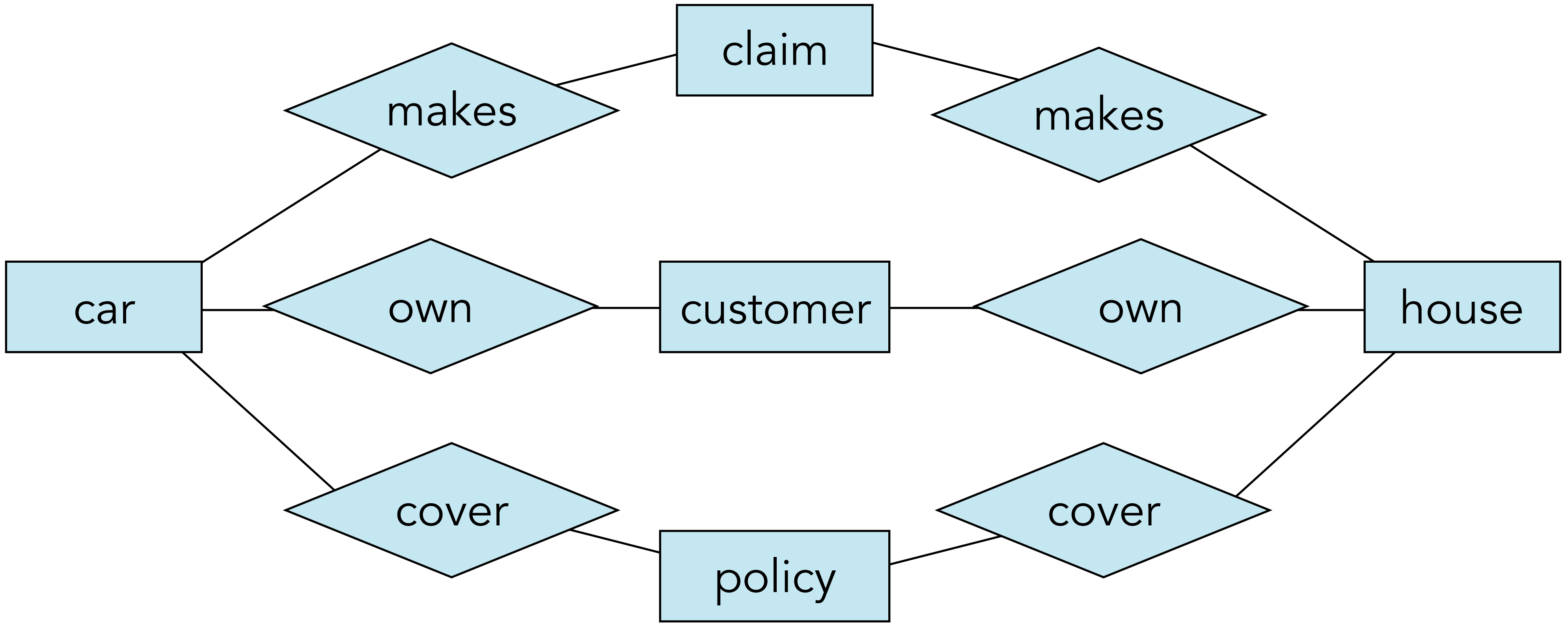


Feedback

ER class test

how I corrected:

- $^1/_3$ of remaining points on relationships
- about half of those points lost if you missed total is-a relationship
- other half on incorrect notations/missing relationships/attributes

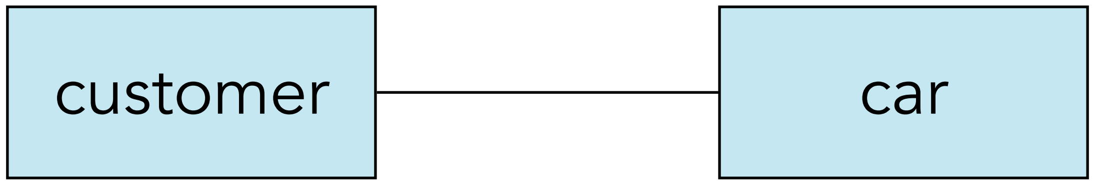


Feedback

ER class test

how I corrected:

- $^1/_3$ of remaining points on relationships
- about half of those points lost if you missed total is-a relationship
- other half on incorrect notations/missing relationships/attributes



Feedback

ER class test

how I corrected:

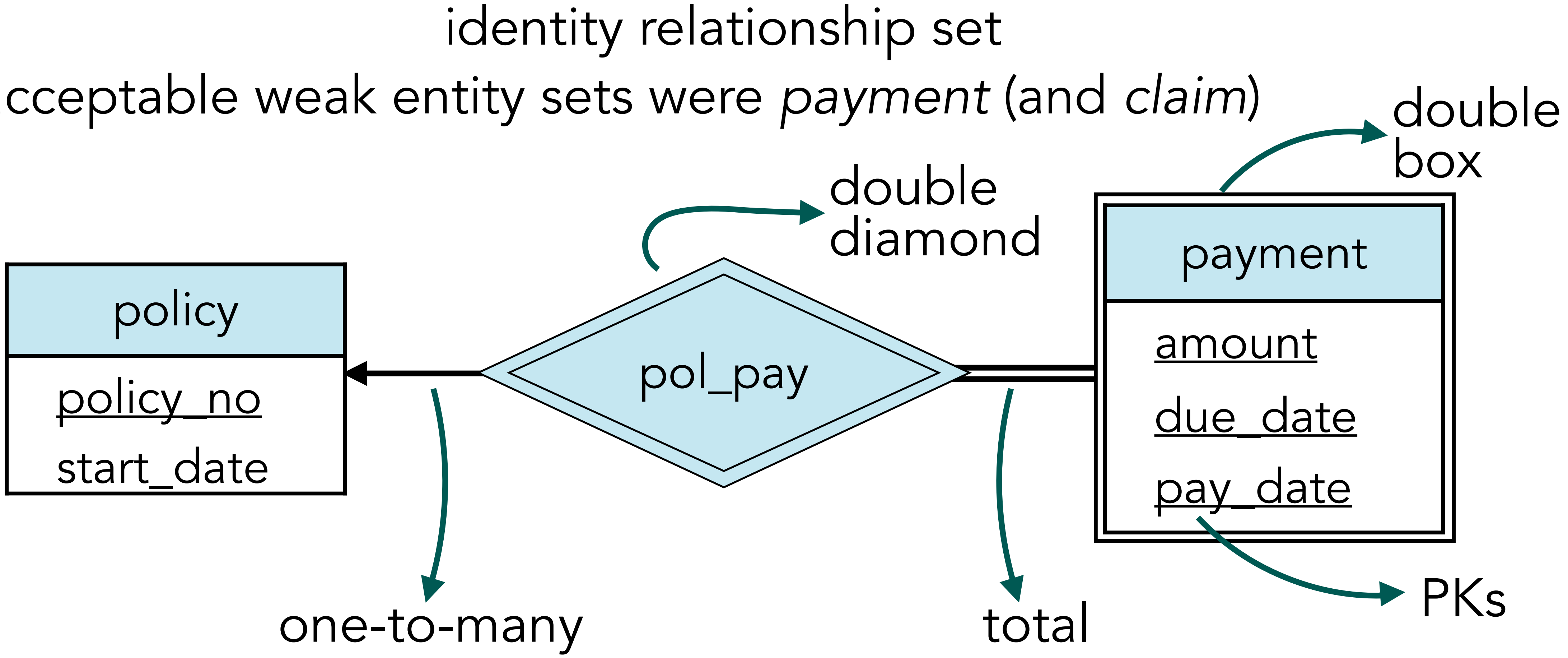
- $^1/_3$ of remaining points on weak entity sets
- only question where you scored 25% for not trying!
- identity relationship set
- acceptable weak entity sets were *payment* (and *claim*)

Feedback

ER class test

how I corrected:

- $^{1}/_{3}$ of remaining points on weak entity sets
- only question where you scored 25% for not trying!
- acceptable weak entity sets were *payment* (and *claim*)

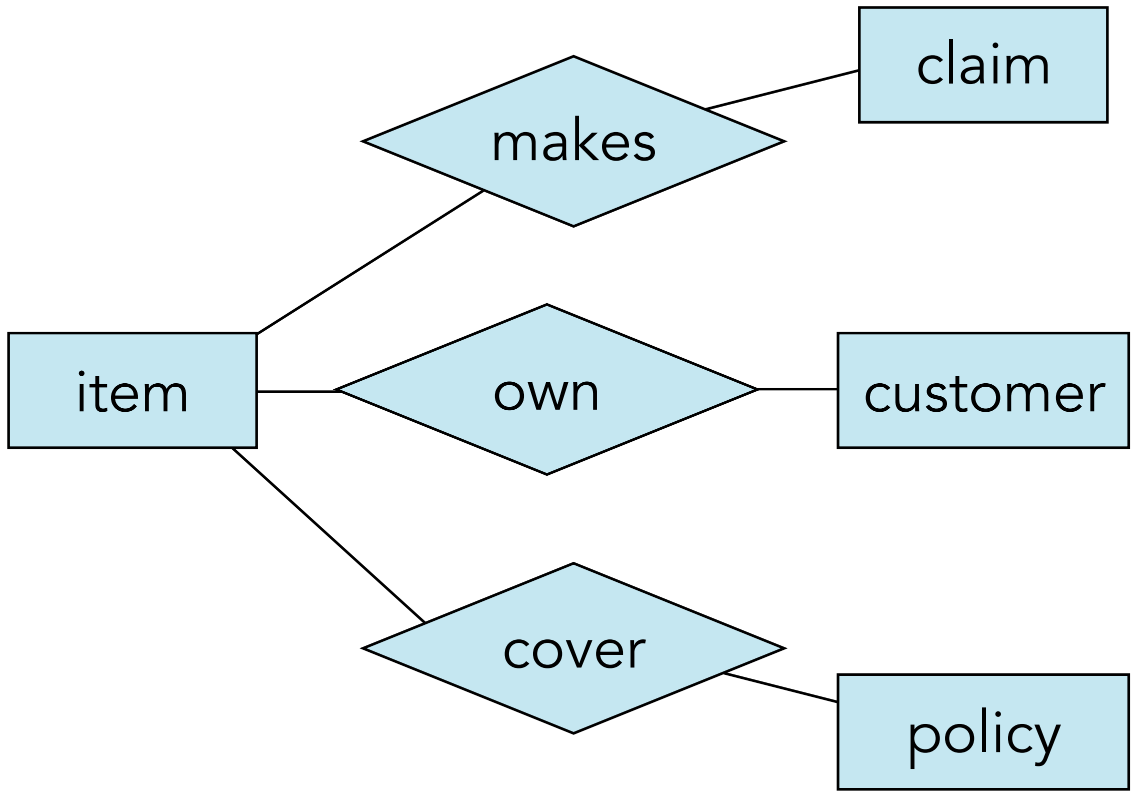


Feedback

ER class test

how I correcte:

$^1/_3$ of remaining points on cardinality/totality

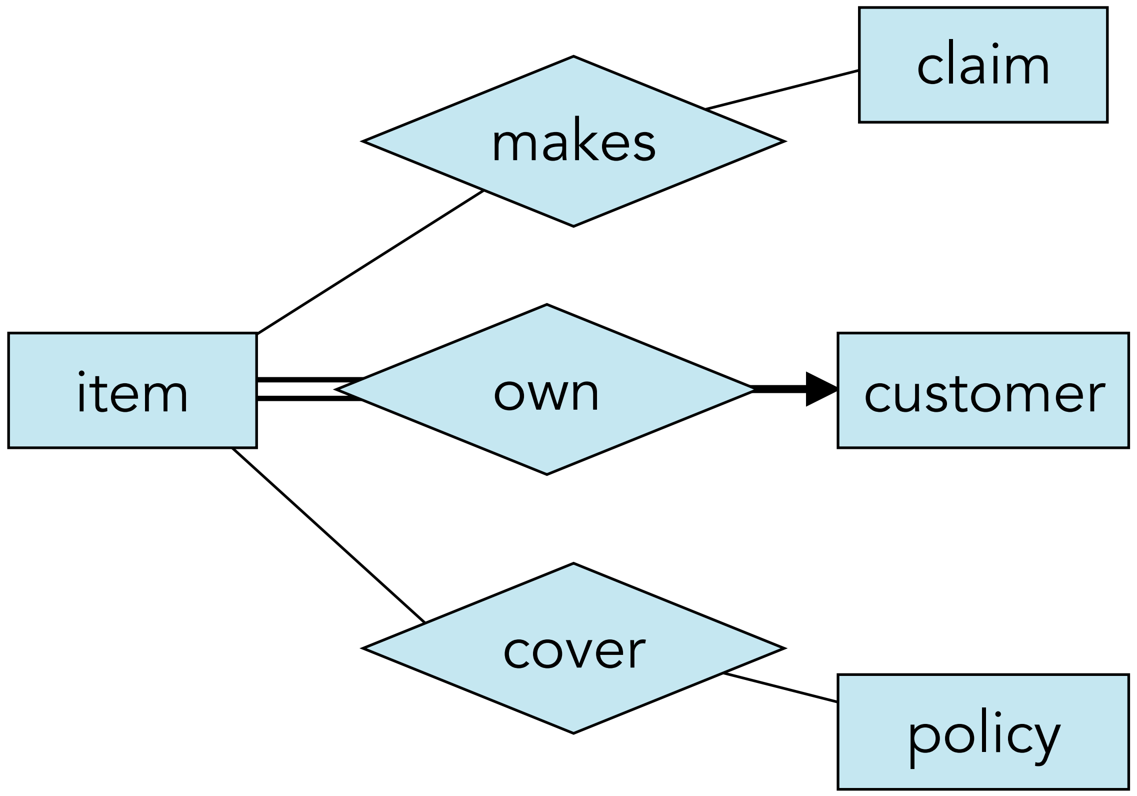


Feedback

ER class test

how I corrected:

$^1/_3$ of remaining points on cardinality/totality

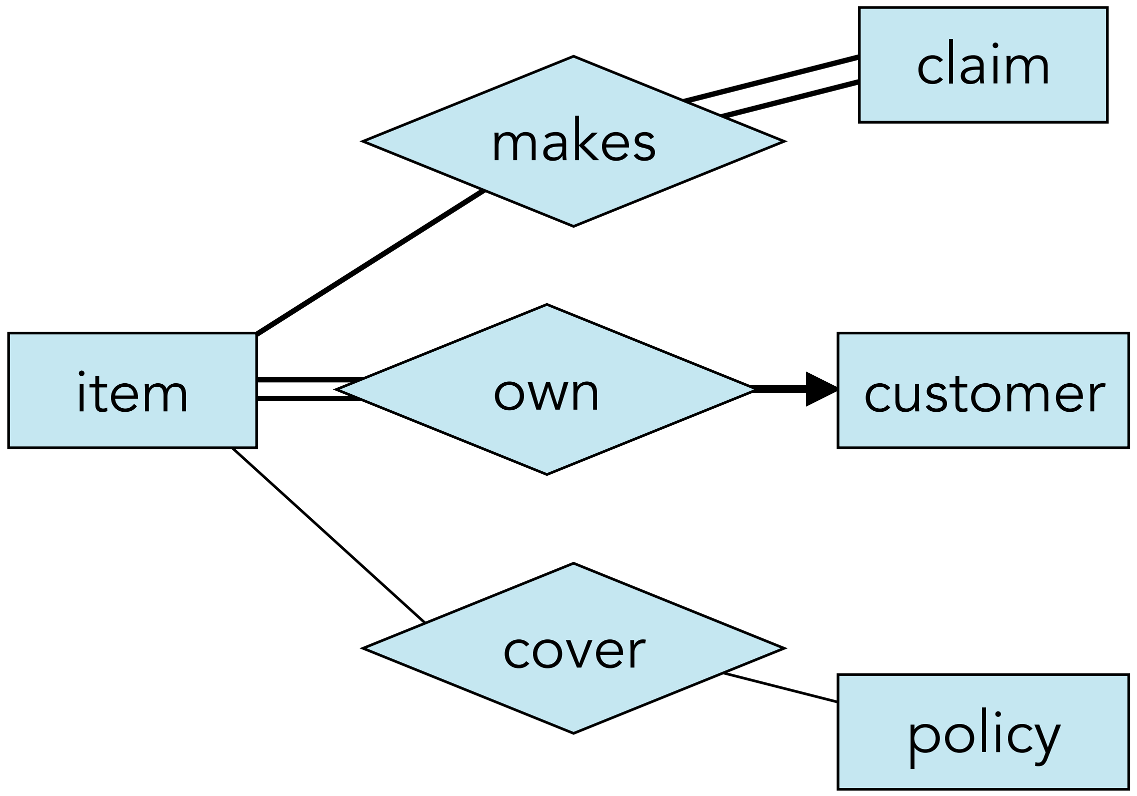


Feedback

ER class test

how I corrected:

- $^1/_3$ of remaining points on cardinality/totality

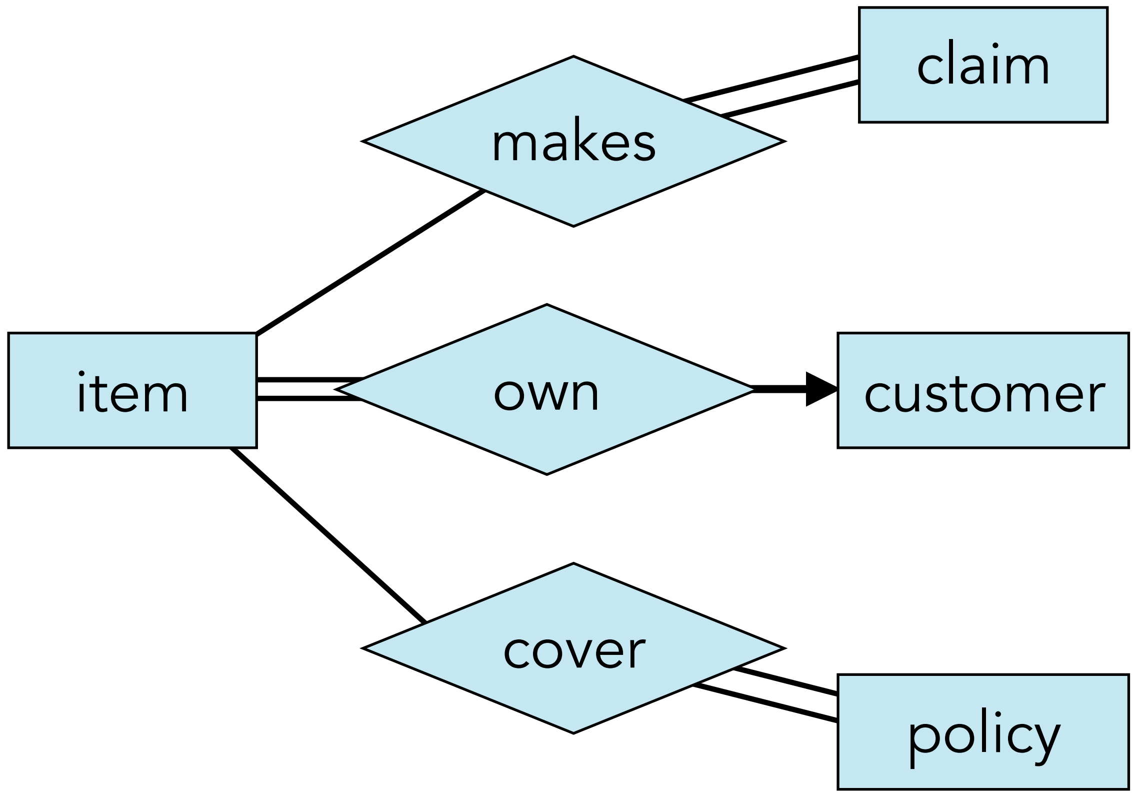


Feedback

ER class test

how I corrected:

- $^1/_3$ of remaining points on cardinality/totality


claim
makes
item
own
customer
cover
policy