CS 325 Week 5-2

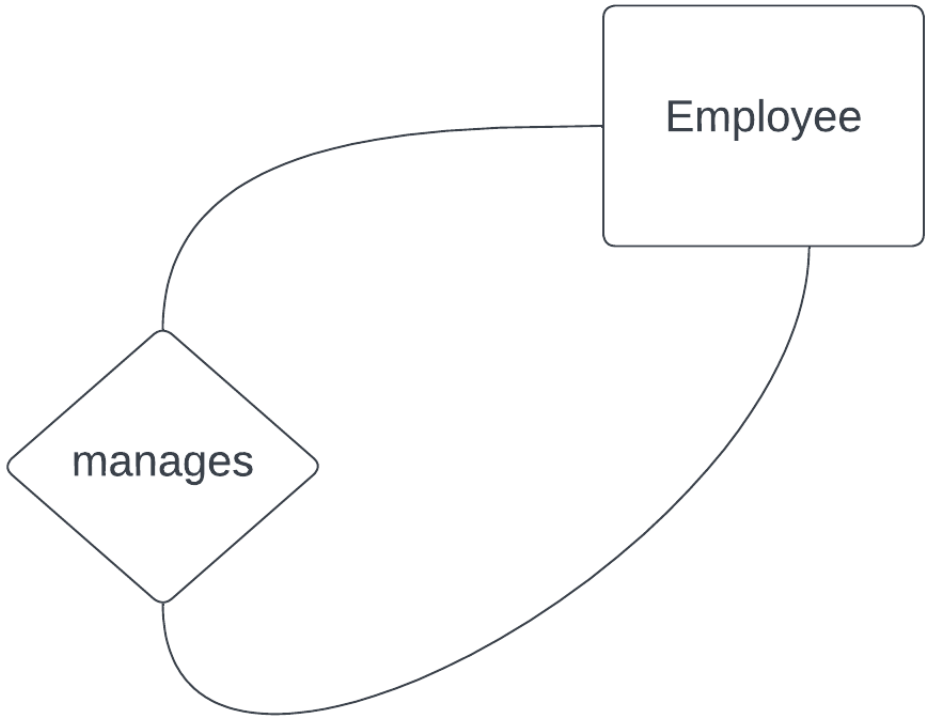
intro to entity-relationship modeling, part 2
(DB Reading Packet 5)

Intro to entity-relationship modeling, part 2

- extensions to the E-R model, and just some additional points to make -- that's the purpose of the packet for E-R modeling, part 2.

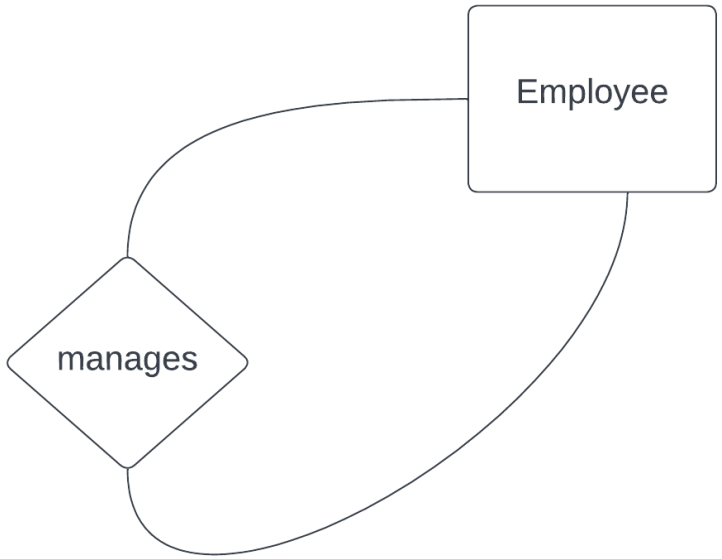
Intro to entity-relationship modeling, part 2

- extensions to the E-R model, and just some additional points to make -- that's the purpose of the packet for E-R modeling, part 2.



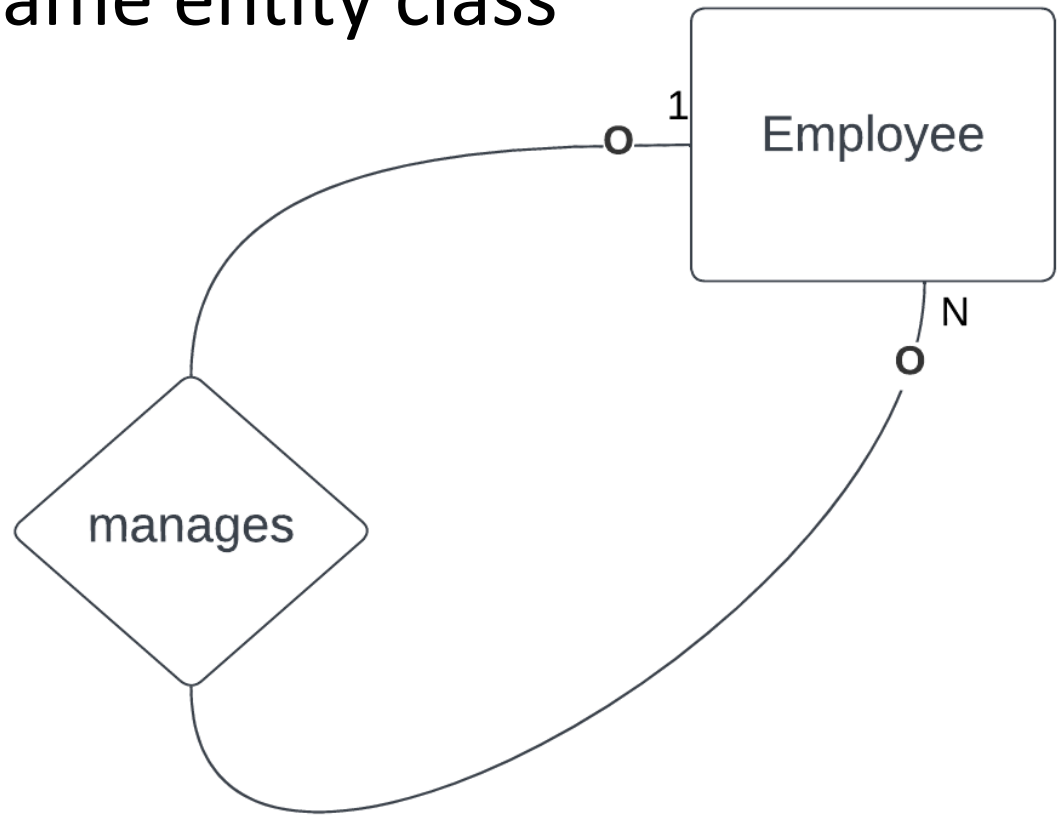
Intro to entity-relationship modeling, part 2

- extensions to the E-R model, and just some additional points to make -- that's the purpose of the packet for E-R modeling, part 2
- for example: recursive relationships! just note that it is OK for a relationship line to begin and end on the same rectangle, to say there is a relationship amongst entity instances of the same entity class.



Intro to entity-relationship modeling, part 2

- for example: recursive relationships! just note that it is OK for a relationship line to begin and end on the same rectangle, to say there is a relationship amongst entity instances of the same entity class

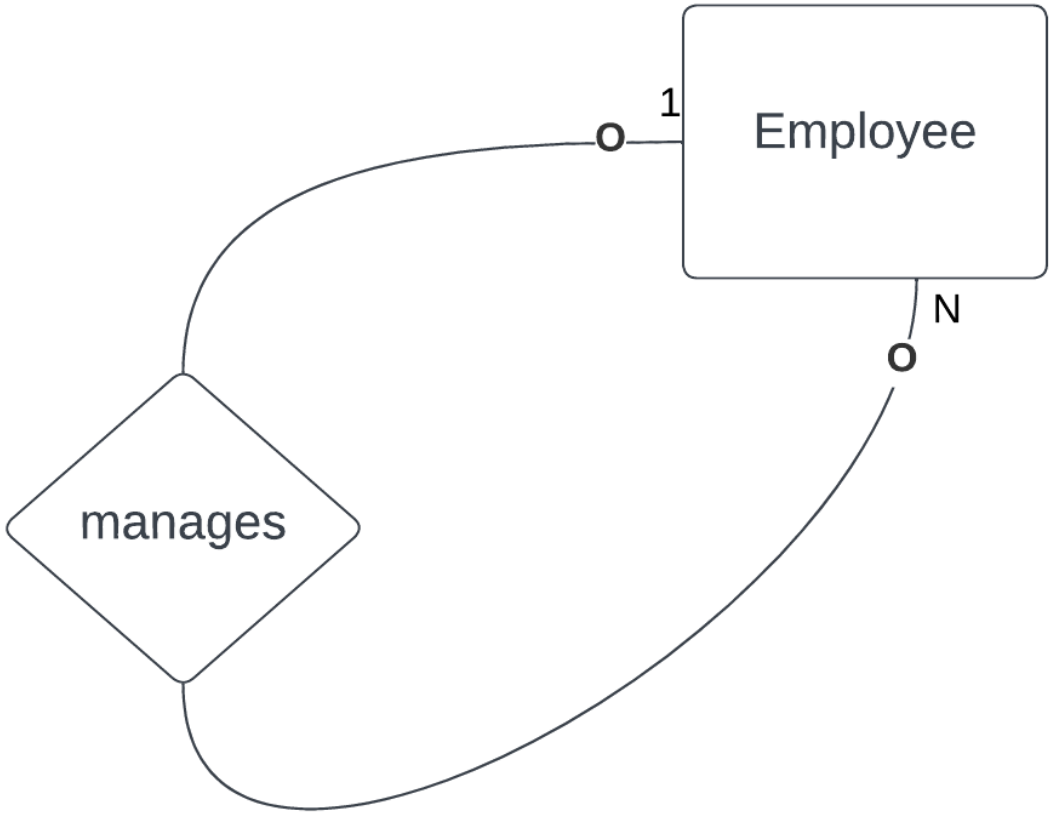


Intro to entity-relationship modeling, part 2

- for example: recursive relationships! just note that it is OK for a relationship line to begin and end on the same rectangle, to say there is a relationship amongst entity instances of the same entity class.

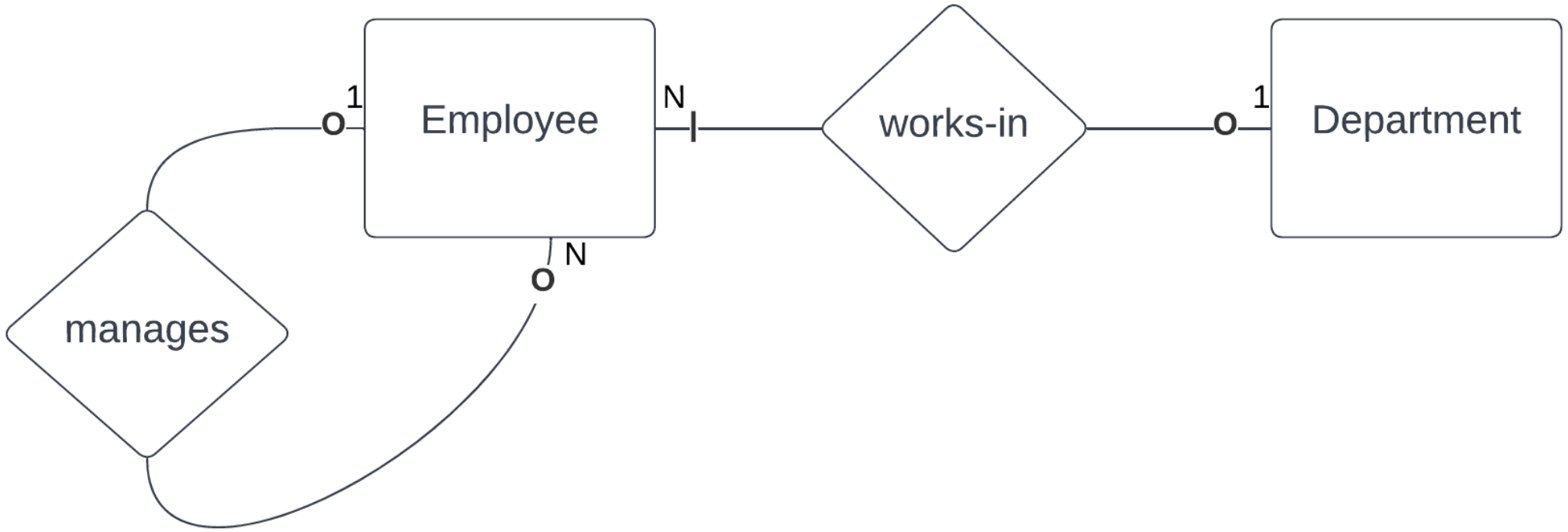
Employee

EMPL_NUM
empl_last_name
salary
hiredate
Job_title
commission



Intro to entity-relationship modeling, part 2

- For example, we can have a “department” entity class, and between “employee” and “department”, we have the “works-in” relationship.



Question

What is a recursive relationship?

1) a relationship that calls itself
2) a relationship amongst instances of the same entity class
3) a relationship between entity class A and entity class B that is also between entity class B and entity class A
4) a relationship between relationships

Intro to entity-relationship modeling, part 2

- Weak entities

Intro to entity-relationship modeling, part 2

- Weak entities
  - is one whose presence is REALLLLLY strongly dependent on the existence of another entity class (typically one)

Intro to entity-relationship modeling, part 2

- Weak entities
  - is one whose presence is REALLLLLY strongly dependent on the existence of another entity class (typically one)
  - don't take this too far; not really for association entity class situations (We will get to those)
  - Imagine an office scenario: Employee and Project

Intro to entity-relationship modeling, part 2

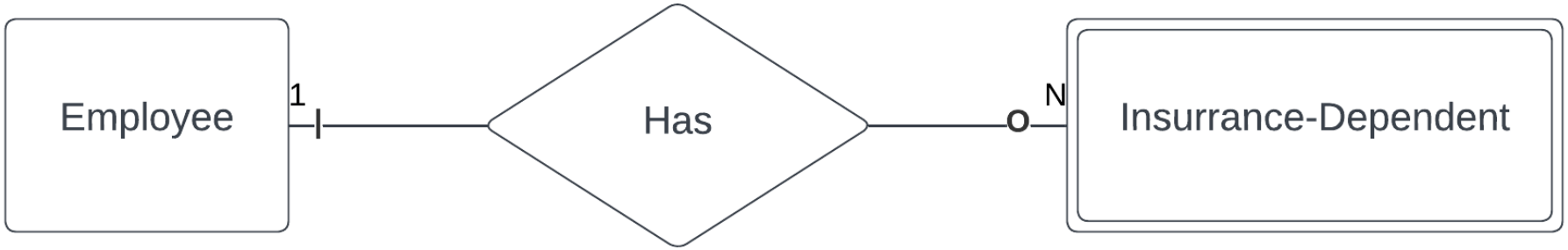
- Weak entities
  - is one whose presence is REALLLLLY strongly dependent on the existence of another entity class (typically one)
  - don't take this too far; not really for association entity class situations (We will get to those)
  - Imagine an office scenario: Employee and Project
    - So, in this scenario, both Employees and Projects are significant in their own right, and neither would be considered a weak entity class

Intro to entity-relationship modeling, part 2

- Weak entities
  - is one whose presence is REALLLLLY strongly dependent on the existence of another entity class (typically one)
  - don't take this too far; not really for association entity class situations (We will get to those)
  - Imagine an office scenario: Employee and Project
  - But perhaps this office allows Employees to designate Insurance-Dependents to receive health-care coverage.

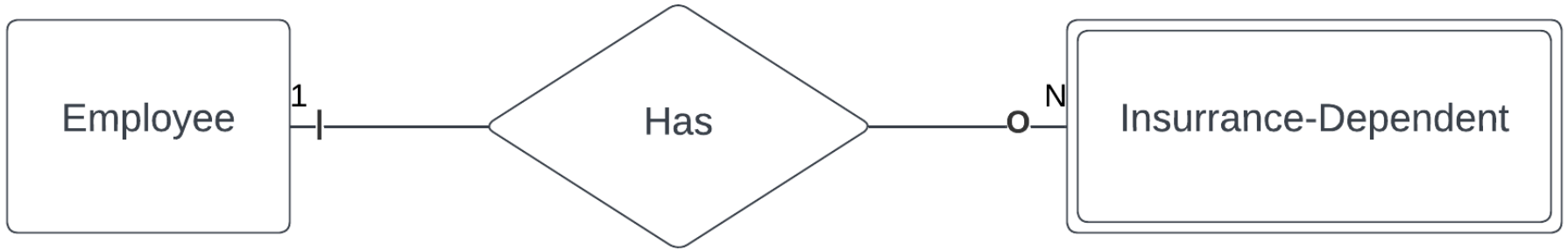
Intro to entity-relationship modeling, part 2

- Weak entities
  - is one whose presence is REALLLLLY strongly dependent on the existence of another entity class (typically one)
  - don't take this too far; not really for association entity class situations (We will get to those)
  - Imagine an office scenario: Employee and Project
  - But perhaps this office allows Employees to designate Insurance-Dependents to receive health-care coverage.



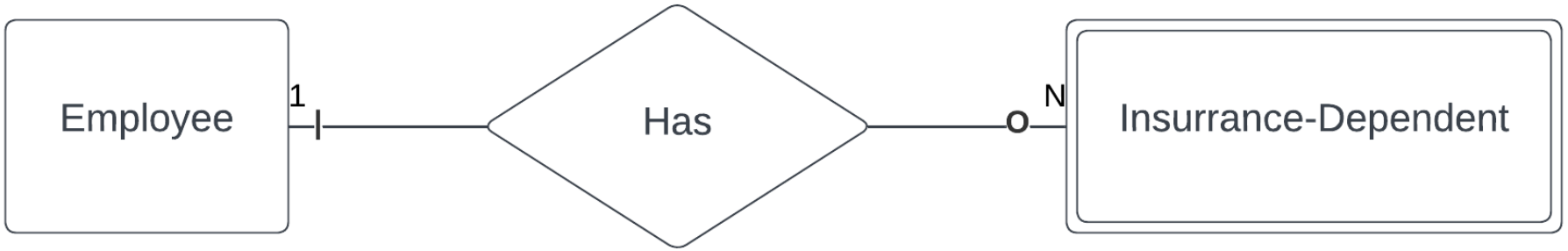
Intro to entity-relationship modeling, part 2

- Weak entities
  - interesting rule of thumb: if it is truly a weak entity class, its instances should be existence-dependent on at least one instance of its parent class (better be a hash/1 on the relationship line and on the parent end of the relationship line...)
  - different notations exist for this, our CS 325 standard will be to put these in a double-bordered rectangle



Intro to entity-relationship modeling, part 2

- Weak entities
  - interesting rule of thumb: if it is truly a weak entity class, its instances should be existence-dependent on at least one instance of its parent class (better be a hash/1 on the relationship line and on the parent end of the relationship line...)
  - different notations exist for this, our CS 325 standard will be to put these in a double-bordered rectangle
  - it is pretty common for a weak entity class to not have any identifying attributes (but that's also true for association entity classes)



Question

Following CS 325's ERD style standards, how are you expected to depict a weak entity class in an ERD?

1) As a rectangle with a double-border
2) As a circle
3) With (Weak) next to its name
4) As a rectangle with a dotted-line border

Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!
  - Supertype entity classes and subtype entity classes were brought in as a significant part of the so-called extended E-R model (added after Peter Chen's initial 1976 paper)

Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!
- Those are better choices for modeling so-called IS-A relationships -- a Savings_Account IS An Account, etc.

Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!
- Little definition
- Those are better choices for modeling so-called IS-A relationships -- a Savings_Account IS An Account, etc.
  - these are also sometimes called GENERALIZATION/SPECIALIZATION relationships

Intro to entity-relationship modeling, part 2

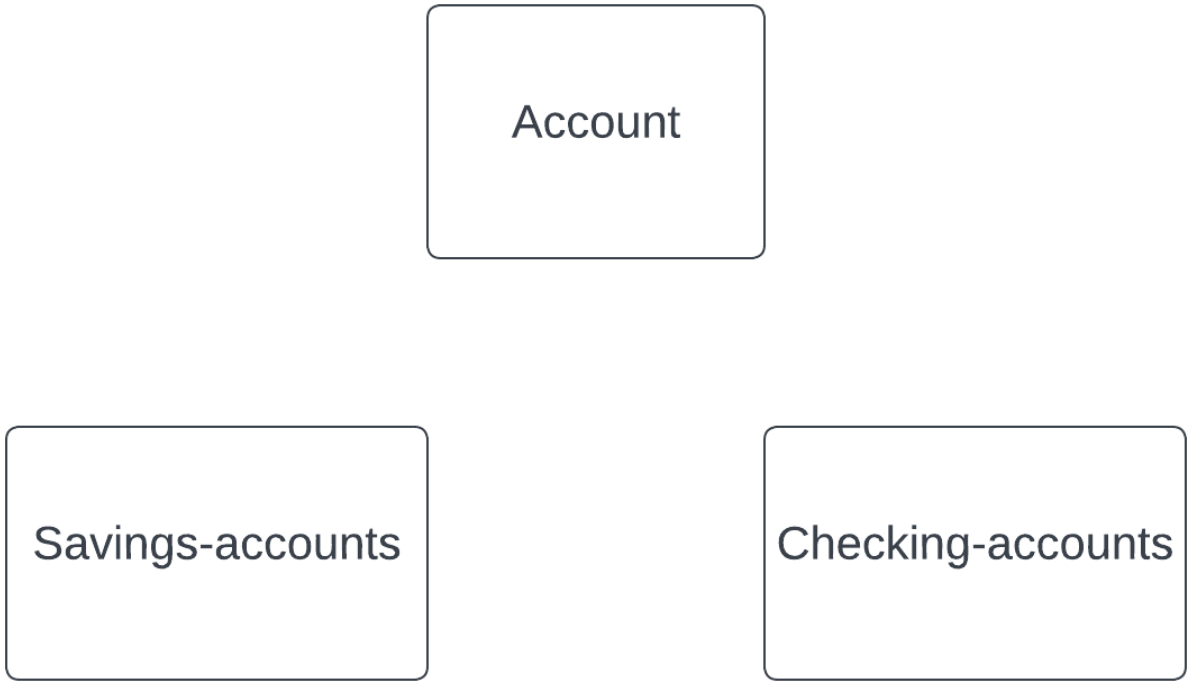
- Supertypes and Subtypes!
- how do you know when these would be useful for your model?
  - sometimes you just realize it -- hey, courses include pass/fall courses and graded courses
  - Sometimes, as you're trying to figure out the attributes for an entity class, you notice DISTINCT subsets of seemingly "optional“ attributes –
  - that is, some distinct subsets of attributes that seem to be "optional", but make more sense when you think of them as characteristics of a subtype entity class instead;

Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!
- how do you know when these would be useful for your model?
  - sometimes the relationships give you clue –
  - you might have some relationships all instances of an entity class can have,
  - but some you really want to limit to only some members of an entity class –
  - and you realize these make more sense if you think of some relationships being with a supertype entity class and some being with a subtype entity class

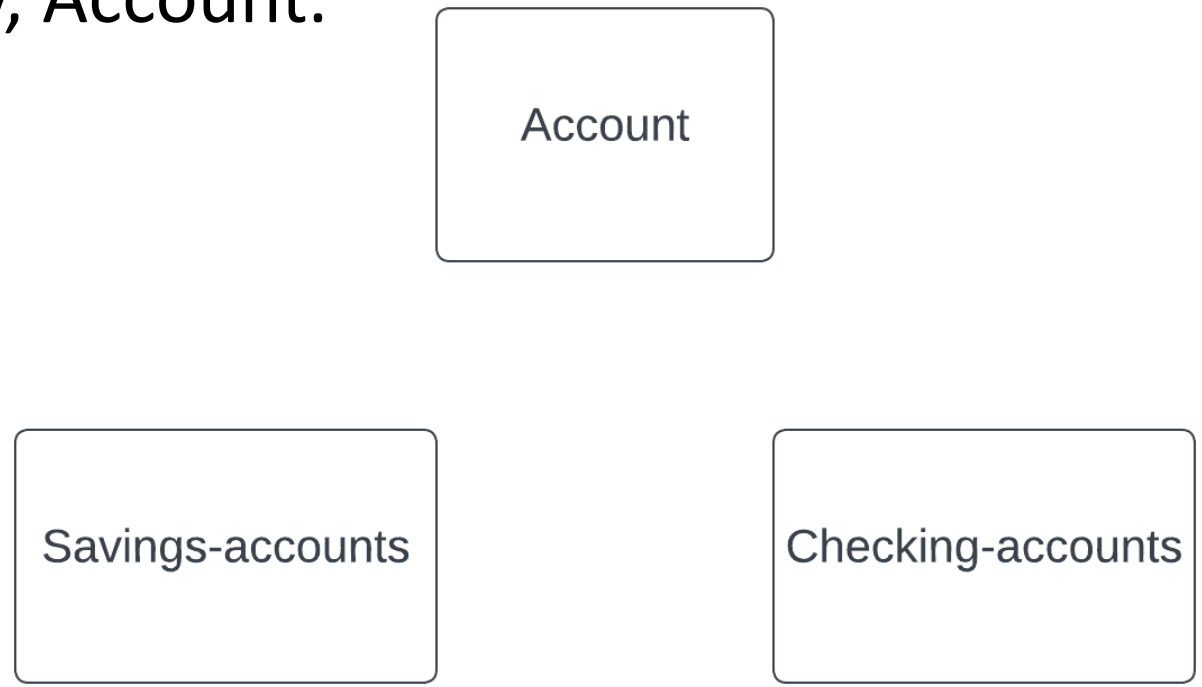
Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!



Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!
  - You wouldn’t say an account has a savings account, but rather, a savings account is a type of account, and a checking account is another type of account. They are variations of the same general category, Account.

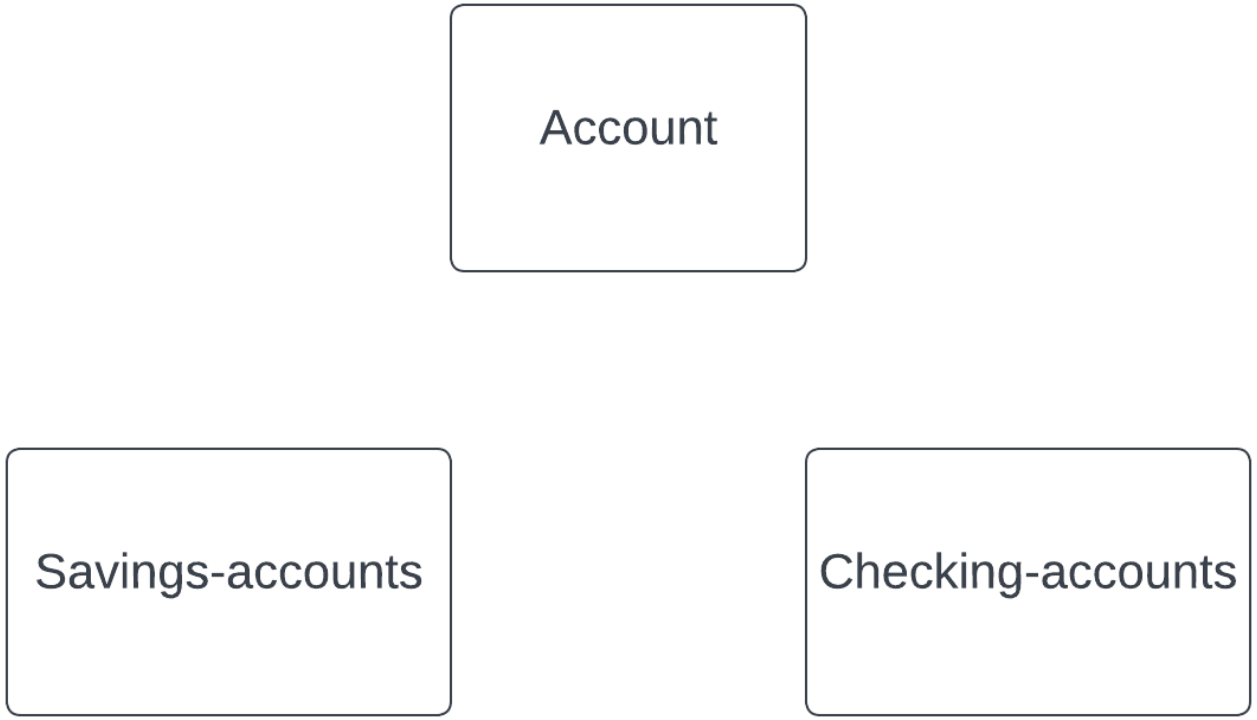


Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!

Account

ACCT-NUM
acct_date_opened



Intro to entity-relationship modeling, part 2

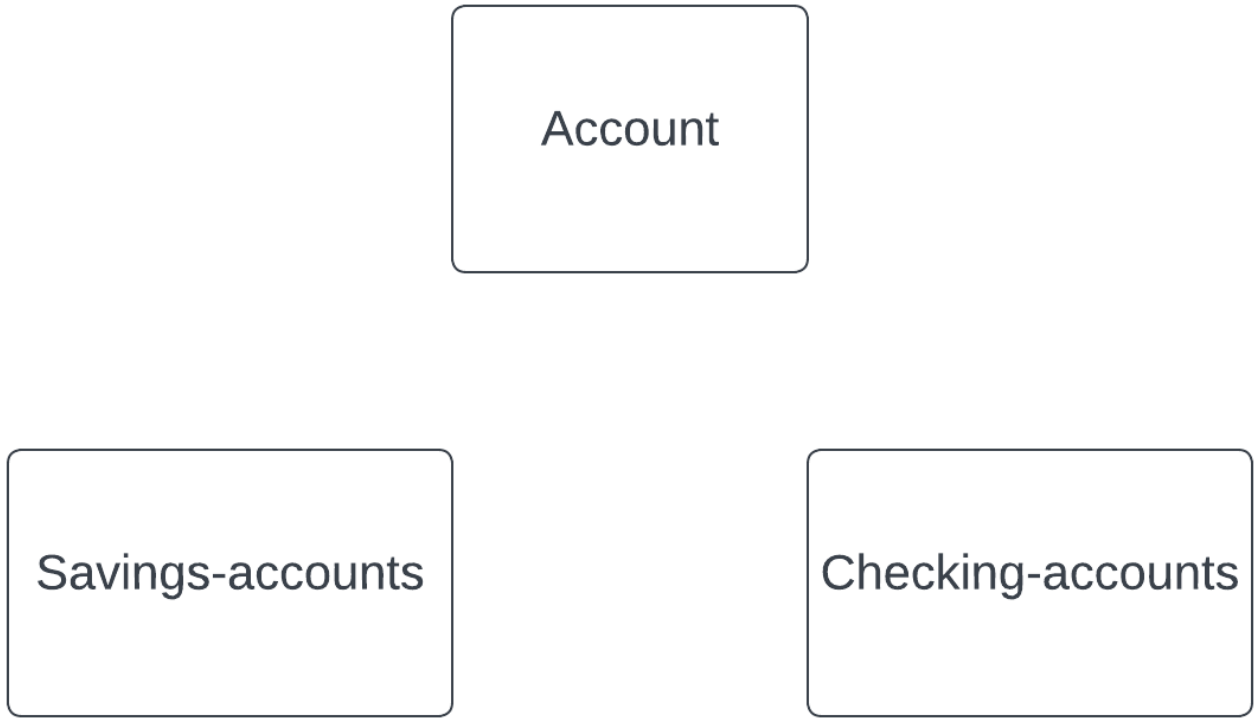
- Supertypes and Subtypes!

Account

ACCT-NUM
acct_date_opened

Saving_Account
---
int_rate

Checking_Account
---
per_check_fee
Monthly_fee



Intro to entity-relationship modeling, part 2

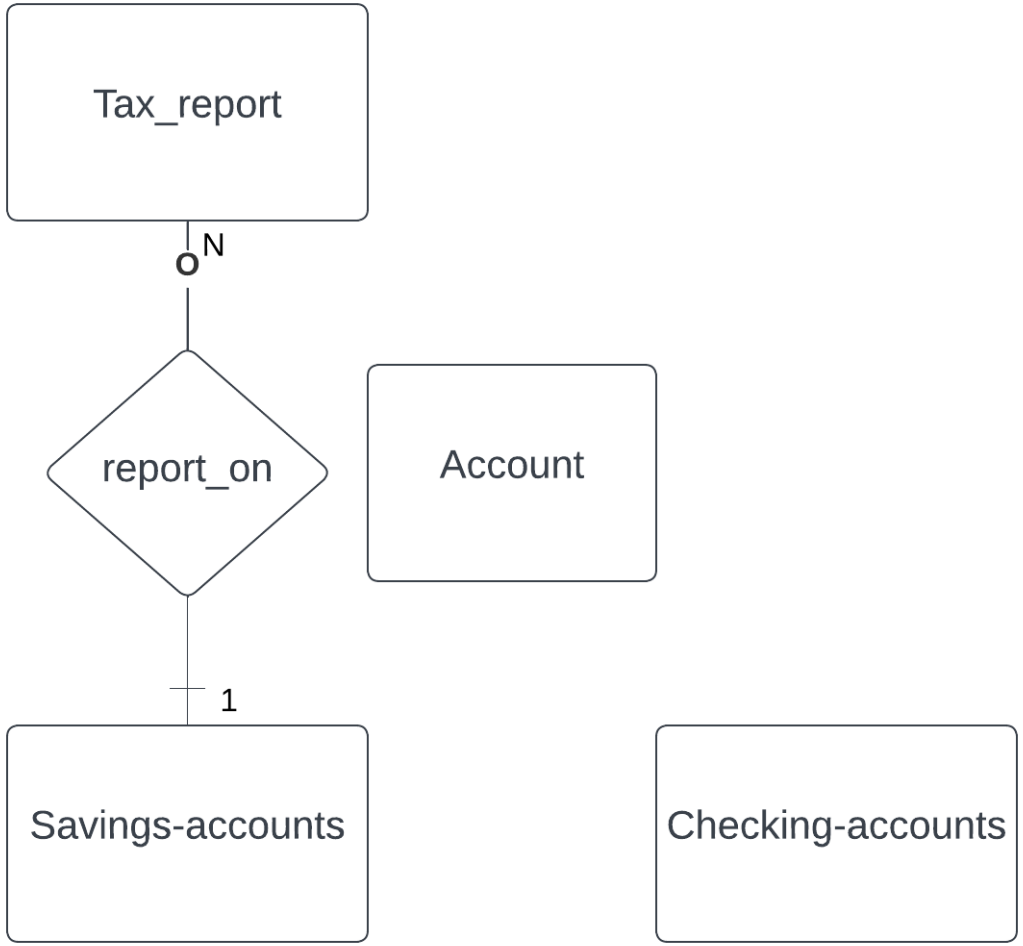
- Supertypes and Subtypes!

Account

ACCT-NUM
acct_date_opened

Saving_Account
---
int_rate

Checking_Account
---
per_check_fee
Monthly_fee



Intro to entity-relationship modeling, part 2

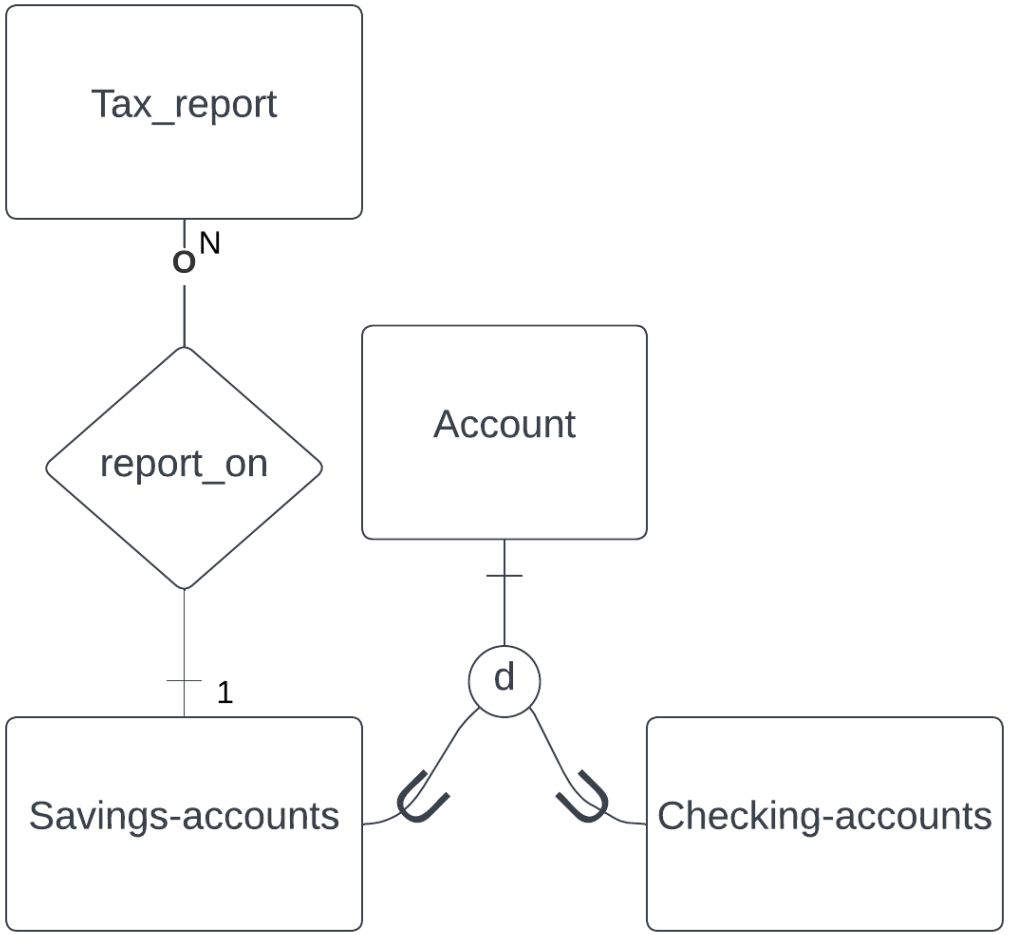
- Supertypes and Subtypes!

Account

ACCT-NUM
acct_date_opened

Saving_Account
---
int_rate

Checking_Account
---
per_check_fee
Monthly_fee



Intro to entity-relationship modeling, part 2

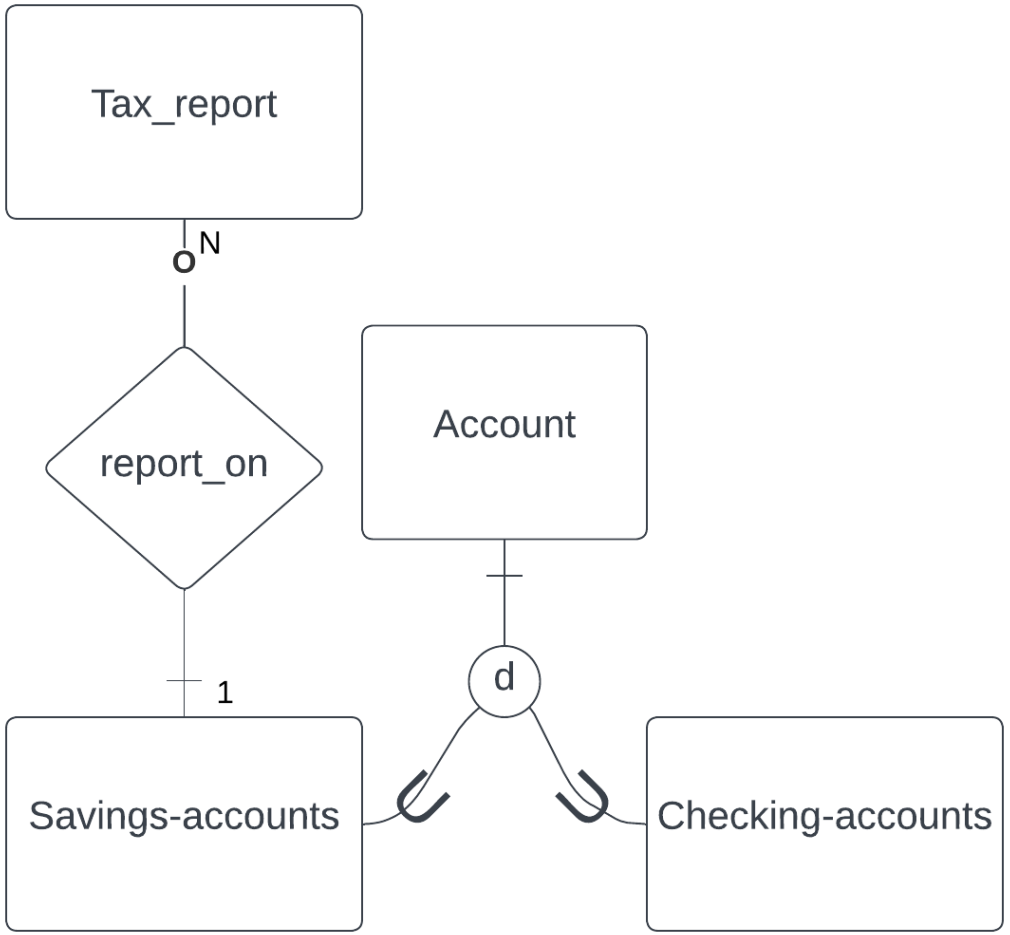
- Supertypes and Subtypes!
  - “d“ - the subtypes are mutually exclusive.

Account

ACCT-NUM
acct_date_opened

Saving_Account
---
int_rate

Checking_Account
---
per_check_fee
Monthly_fee



Intro to entity-relationship modeling, part 2

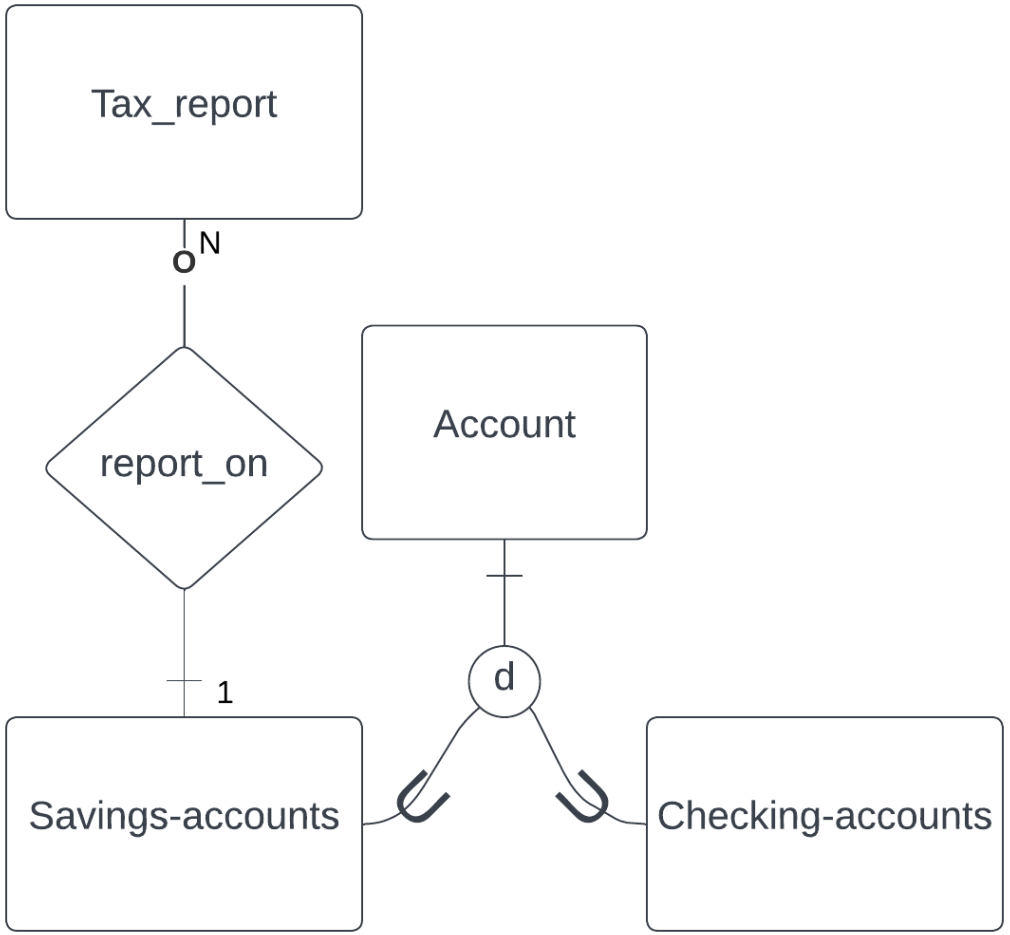
- Supertypes and Subtypes!
  - no maximum cardinalities

Account

ACCT-NUM
acct_date_opened

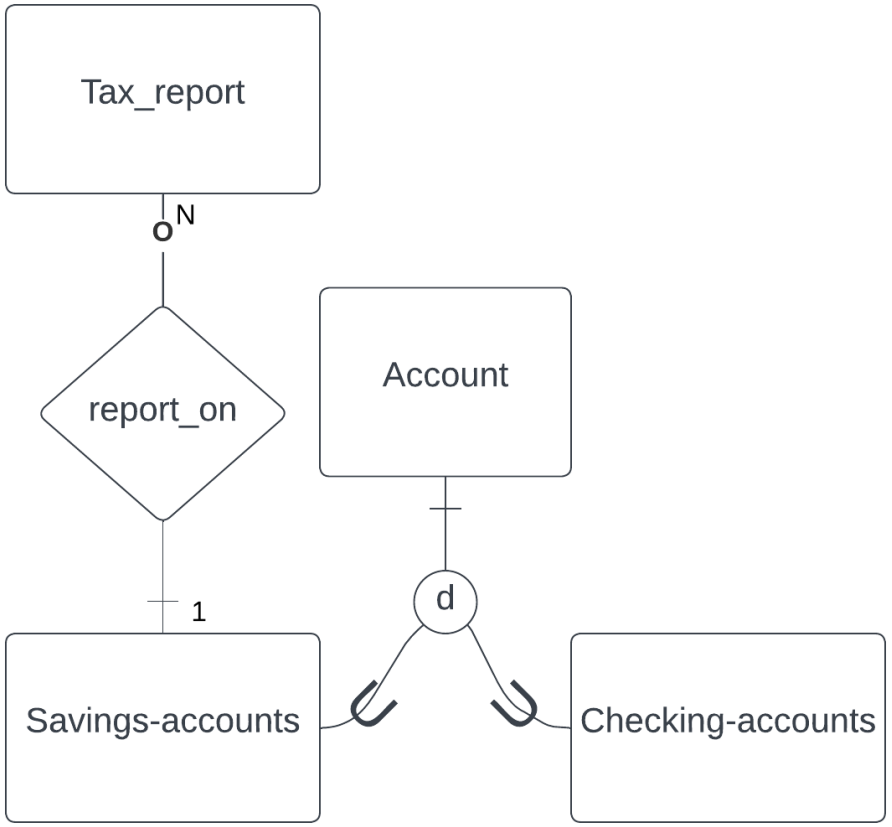
Saving_Account
---
int_rate

Checking_Account
---
per_check_fee
Monthly_fee



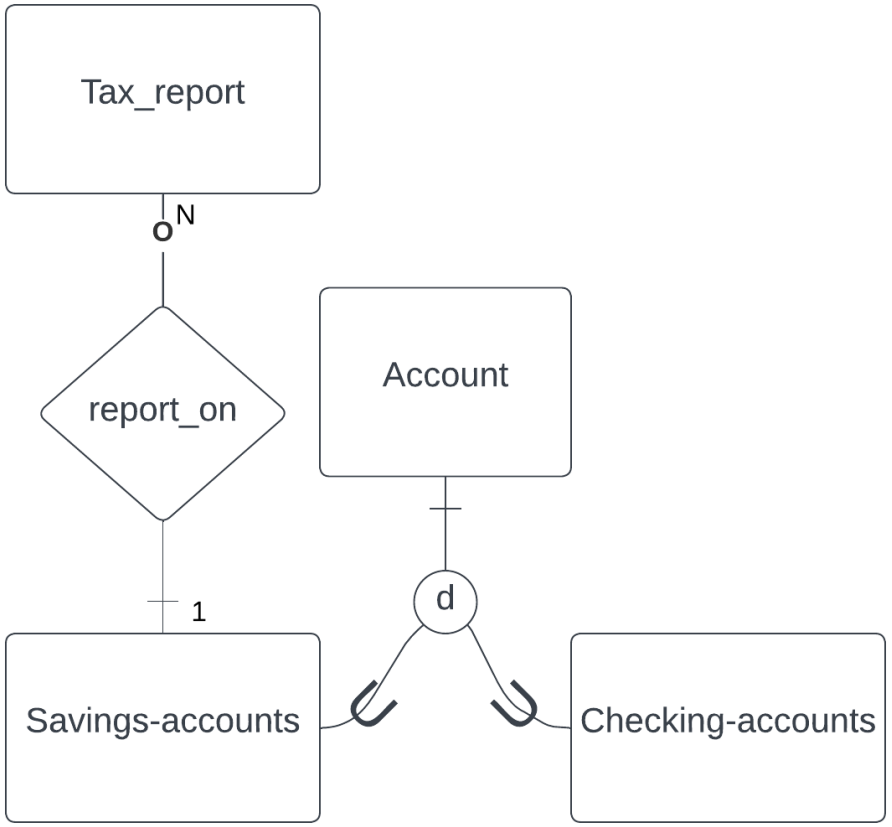
Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!
  - “d” shows that an account is either savings or checking, but never both
  - "U" shape indicates that savings and checking accounts are subtypes of the broader account type



Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes!
  - “d” shows that an account is either savings or checking, but never both
  - "U" shape indicates that savings and checking accounts are subtypes of the broader account type
  - the “hash” symbol ensures that every instance of the supertype is also an instance of at least one subtype. This enforces that there are no "unclassified" instances in the supertype



Question

The entity supertype contains common characteristics, and the entity subtypes each contain their own unique characteristics.

1) True
2) False

Intro to entity-relationship modeling, part 2

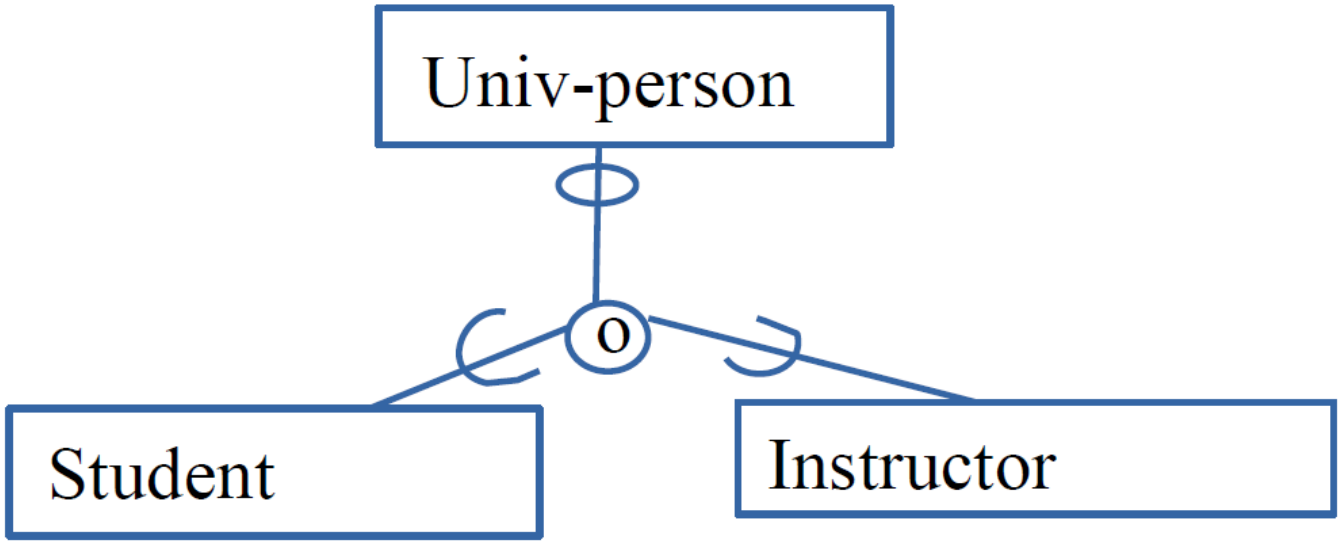
- Supertypes and Subtypes! CS 325 standard format for these:
  - an entity class rectangle for EACH supertype and EACH subtype entity class
  - BUT the relationship lines are different here -- draw a line from each EACH supertype and EACH subtype entity class TO a small circle,
  - labeled with **d** if the subclass entity classes are DISJOINT and labeled with **o** if the subclass entity classes are OVERLAPPING and labeled with **u** if the subclass entity classes are part of a UNION supertype-subtype situation

Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes! CS 325 standard format for these:
  - an entity class rectangle for EACH supertype and EACH subtype entity class
  - BUT the relationship lines are different here -- draw a line from each EACH supertype and EACH subtype entity class TO a small circle,
  - labeled with **d** if the subclass entity classes are DISJOINT and labeled with **o** if the subclass entity classes are OVERLAPPING and labeled with **u** if the subclass entity classes are part of a UNION supertype-subtype situation
    - put a small u-shaped curve on the line connecting each subtype entity class to that circle,
    - put a hash or oval on the line connecting the supertype entity class to that circle, indicating whether a supertype instance MUST be one of subtype instances, also, or not;

Intro to entity-relationship modeling, part 2

- Supertypes and Subtypes! CS 325 standard format for these:



Intro to entity-relationship modeling, part 2

- handling ATTRIBUTES for supertype/subtype entity classes

Intro to entity-relationship modeling, part 2

handling ATTRIBUTES for supertype/subtype entity classes

- generally, we'd like a characteristic or attribute in a model to only appear ONCE, in ONE entity class's attribute list

Intro to entity-relationship modeling, part 2

handling ATTRIBUTES for supertype/subtype entity classes

- generally, we'd like a characteristic or attribute in a model to only appear ONCE, in ONE entity class's attribute list
- if an attribute is meant to be for ALL instances of that supertype entity class, it goes in the supertype entity class's attribute list

Intro to entity-relationship modeling, part 2

handling ATTRIBUTES for supertype/subtype entity classes

- generally, we'd like a characteristic or attribute in a model to only appear ONCE, in ONE entity class's attribute list
- if an attribute is meant to be for ALL instances of that supertype entity class, it goes in the supertype entity class's attribute list
- SO -- for the Account scenario:

```
Account                 Savings_acct        Checking_acct
--------                -------------       --------------
ACCT_NUM                int_rate            per_ck_charge
acct_date_opened        min_balance         monthly_fee
acct_balance
```

Intro to entity-relationship modeling, part 2

handling ATTRIBUTES for supertype/subtype entity classes

- generally, we'd like a characteristic or attribute in a model to only appear ONCE, in ONE entity class's attribute list
- if an attribute is meant to be for ALL instances of that supertype entity class, it goes in the supertype entity class's attribute list
- NOTICE these distinctive things here:
  - there aren't repeated attributes here!

Intro to entity-relationship modeling, part 2

handling ATTRIBUTES for supertype/subtype entity classes

- generally, we'd like a characteristic or attribute in a model to only appear ONCE, in ONE entity class's attribute list
- if an attribute is meant to be for ALL instances of that supertype entity class, it goes in the supertype entity class's attribute list
- NOTICE these distinctive things here:
  - there aren't repeated attributes here!
  - it is QUITE common that the subtype entity classes do not have identifying attributes! (those tend to be in the supertype entity class's attribute lists)

Intro to entity-relationship modeling, part 2

handling ATTRIBUTES for supertype/subtype entity classes

- generally, we'd like a characteristic or attribute in a model to only appear ONCE, in ONE entity class's attribute list
- if an attribute is meant to be for ALL instances of that supertype entity class, it goes in the supertype entity class's attribute list
- SO -- for the Univ_person scenario:

```
Univ_person       Student       Instructor
-----------       --------      ----------
UNIV_ID           curr_gpa      salary_per_course
Last_name
First_name
Campus_email
```

Question

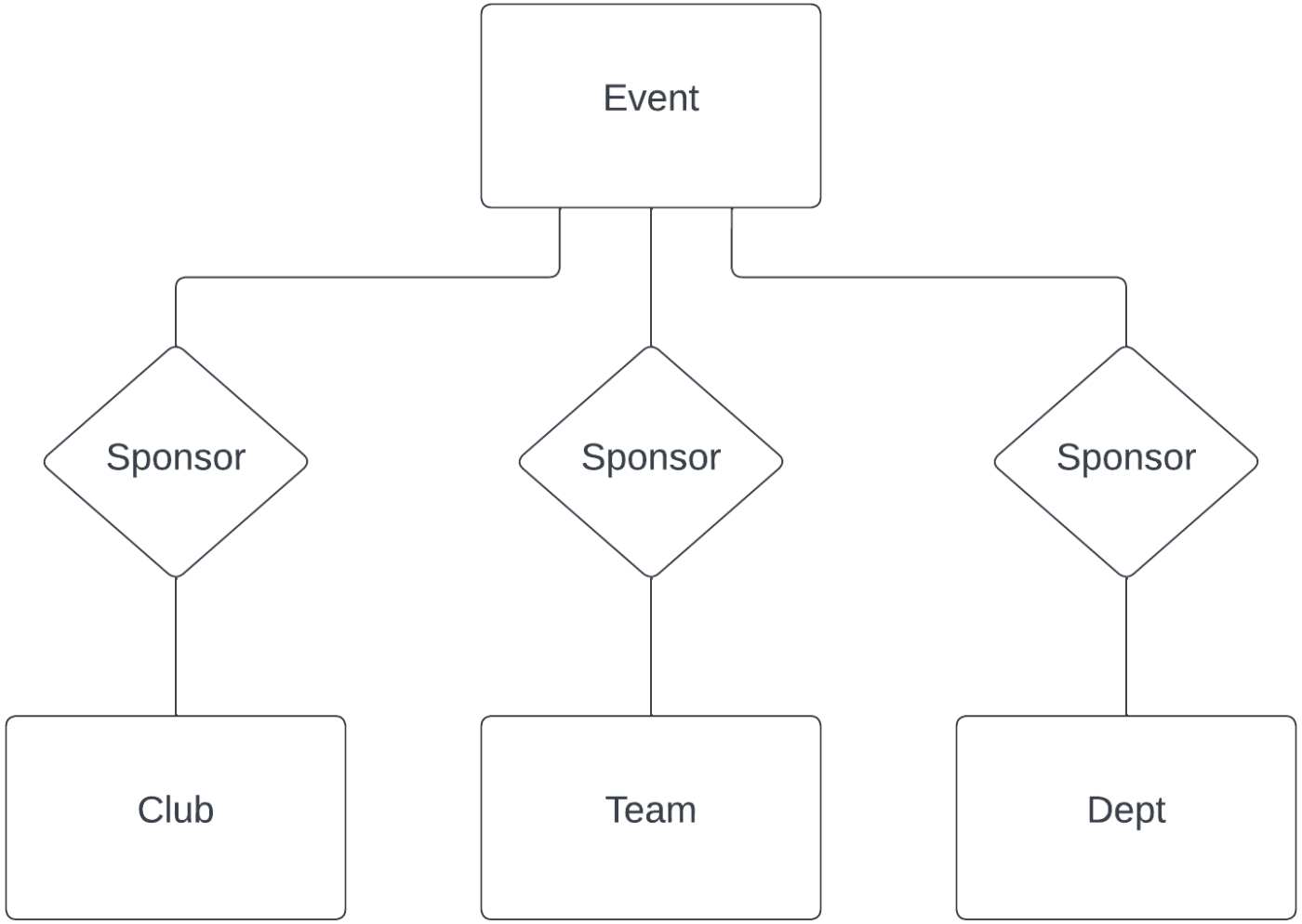
Again modeling that scenario in which you have a supertype entity class `Widget`, with subtype entity classes `Cooking_Widget`, `Transport_Widget`, and `Art_Widget`, and in which it turns out that every widget instance must be either for cooking or for transport or for art -- no widget is for more than one of those, and every widget is for one of those.

`Widget` instances happen to be uniquely identified via a `widget ID number`. Following CS 325's ERD style standards, which entity attribute list(s) should contain `WIDG_ID_NUM`?

1. Just `Widget`'s entity attribute list.
2. Just the entity attribute lists for `Cooking_Widget`, `Transport_Widget`, and `Art_Widget`.
3. The entity attribute lists for `Widget`, `Cooking_Widget`, `Transport_Widget`, and `Art_Widget`

Intro to entity-relationship modeling, part 2

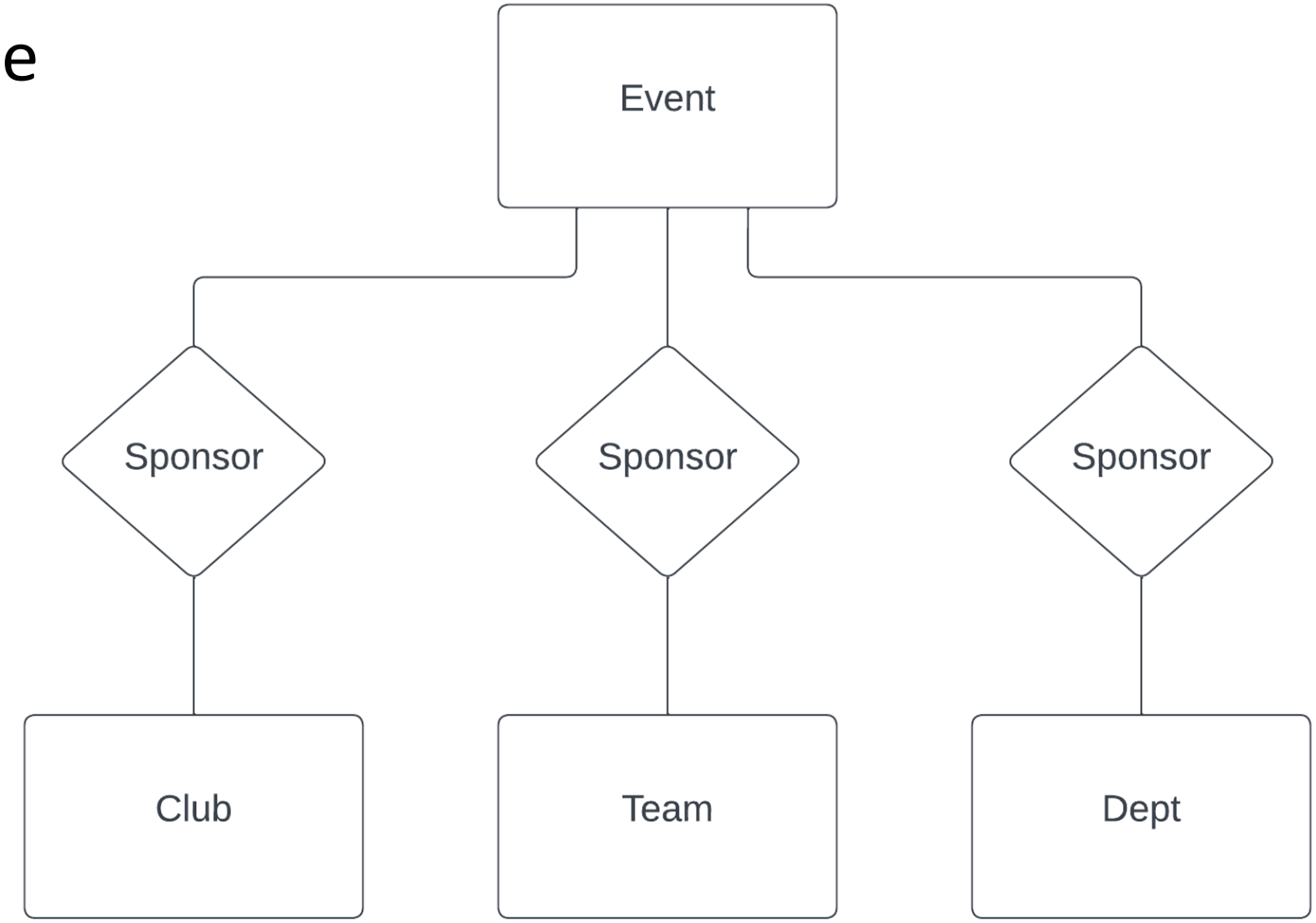
what about those UNION supertype/subtypes?



Intro to entity-relationship modeling, part 2

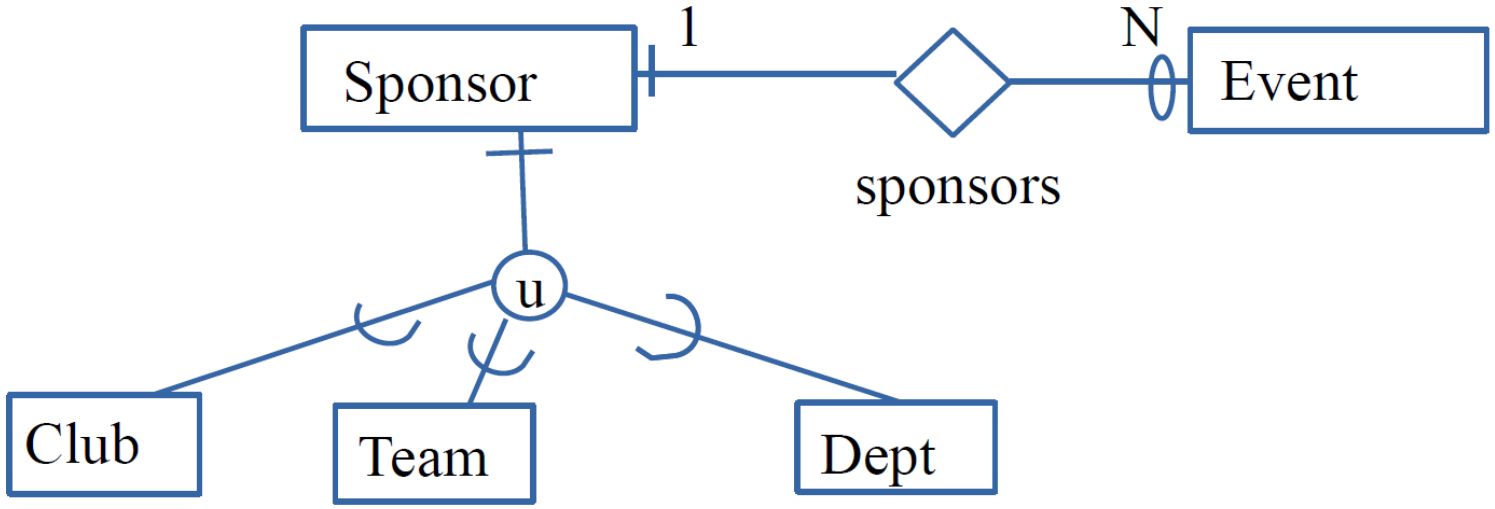
what about those UNION supertype/subtypes?

- there's a concept of a sponsor of an event
- need for a union supertype



Intro to entity-relationship modeling, part 2

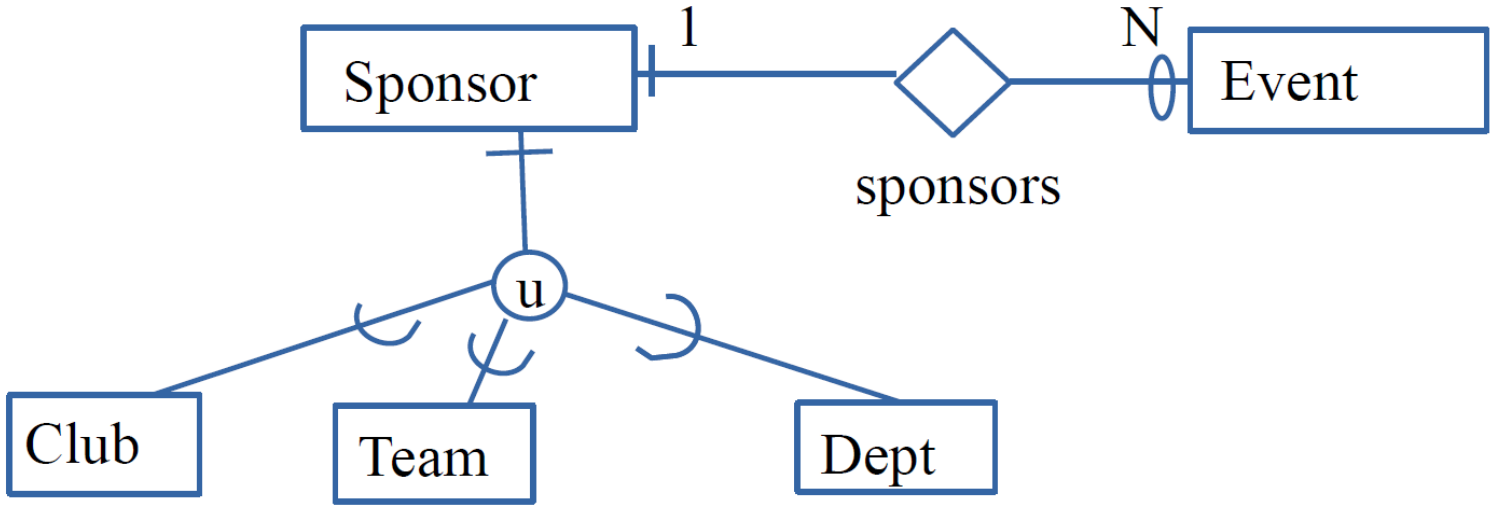
what about those UNION supertype/subtypes?



Intro to entity-relationship modeling, part 2

what about those UNION supertype/subtypes?

- a sponsor can sponsor zero to many events, while an event must have at least one sponsor



Intro to entity-relationship modeling, part 2

what about those UNION supertype/subtypes?

- when you have a business rule that says "certain <really distinct categories> can do <certain important thing>",
- and so it turns out that making a UNION supertype entity class for "those that can do this thing" can be very useful in a model;
- (see the Sponsor-of-Events model example in DB Reading Packet 5)

Intro to entity-relationship modeling, part 2

what about those UNION supertype/subtypes?

```
Sponsor Club        Team       Dept        Event
------- --------    --------   ----------  -----
        CLUB_NUM    TEAM_CODE  DEPT_CODE   EVENT_NUM
        Club_Name   Sport      Dept_title  Event_title
        Is_active   Season     Office_num  Event_date
```

Intro to entity-relationship modeling, part 2

what about those UNION supertype/subtypes?

- it is QUITE common if the union supertype has few or even no attributes at the model stage!
- it is QUITE common if the union subtypes all have VERY different and distinct attributes, and distinct identifying attributes!
- BUT: IF there ARE common attributes for union supertypes, put those in the union supertype's entity attribute list...

```
Sponsor Club        Team       Dept        Event
------- --------    --------   ----------  -----
        CLUB_NUM    TEAM_CODE  DEPT_CODE   EVENT_NUM
        Club_Name   Sport      Dept_title  Event_title
        Is_active   Season     Office_num  Event_date
```

Question

Modeling a scenario, you have a supertype entity class `Widget`, with subtype entity classes `Cooking_Widget`, `Transport_Widget`, and `Art_Widget`. It turns out that every widget instance must be either for cooking or for transport or for art -- no widget is for more than one of those, and every widget is for one of those.

This being the case, following CS 325's ERD style standards, what letter should be put on the circle connecting the relationship lines connecting `Widget`, `Cooking_Widget`, `Transport_Widget`, and `Art_Widget`?

1) The letter o, for overlapping
2) The letter u, for union
3) The letter s, for subtype
4) The letter d, for disjoint

Question

Again modeling that scenario in which you have a supertype entity class `Widget`, with subtype entity classes `Cooking_Widget`, `Transport_Widget`, and `Art_Widget`, and in which it turns out that every widget instance must be either for cooking or for transport or for art -- no widget is for more than one of those, and every widget is for one of those.

This still being the case, following CS 325's ERD style standards, what should be put on the relationship line next to Widget's entity class rectangle?

1) An Sb, for subtype
2) An Sp, for supertype
3) An oval, or a zero
4) A line, or a 1, or a hash

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
[dl313@nrs-projects ~]$ mkdir f24-325lect05-2
[dl313@nrs-projects ~]$ chmod 700 f24-325lect05-2
[dl313@nrs-projects ~]$ cd f24-325lect05-2
[dl313@nrs-projects f24-325lect05-2]$
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
[dl313@nrs-projects ~]$ mkdir f24-325lect05-2
[dl313@nrs-projects ~]$ chmod 700 f24-325lect05-2
[dl313@nrs-projects ~]$ cd f24-325lect05-2
[dl313@nrs-projects f24-325lect05-2]$ vim 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
/*=====
    a SQL select statement can do MORE than just relational operations...

    for example, a SELECT clause can project more than just columns.

    for example, you can specify that a computation on a column be
    projected, and then it will be done for each selected row from that
    query
    *   THIS DOES NOT CHANGE the DATA IN THAT TABLE!!!!! it just
        PROJECTS a result with this computation showing

=====*/
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                                    12,8          All
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
/*=====
    a SQL select statement can do MORE than just relational operations...

    for example, a SELECT clause can project more than just columns.

    for example, you can specify that a computation on a column be
    projected, and then it will be done for each selected row from that
    query
    *   THIS DOES NOT CHANGE the DATA IN THAT TABLE!!!!! it just
        PROJECTS a result with this computation showing

=====*/

prompt ========
prompt project empl last names and salaries:

select empl_last_name, salary
from   empl;
~
~
~
~
~
-- INSERT --                                                18,13          All
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
========
project empl last names and salaries:

EMPL_LAST_NAME      SALARY
--------------- ----------
King                  5000
Jones                 2975
Blake                 2850
Raimi                 2450
Ford                  3000
Smith                  800
Michaels              1600
Ward                  1250
Martin                1250
Scott                 3000
Turner                1500

EMPL_LAST_NAME      SALARY
--------------- ----------
Adams                 1100
James                  950
Miller                1300
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
prompt ========
prompt project empl last names and salaries:

select empl_last_name, salary
from   empl;

prompt ========
prompt project empl last names and 2 * salaries:

select empl_last_name, salary * 2
from   empl;
```

-- INSERT -- 35,1 Bot

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
========
project empl last names and 2 * salaries:

EMPL_LAST_NAME    SALARY*2
--------------- ----------
King                 10000
Jones                 5950
Blake                 5700
Raimi                 4900
Ford                  6000
Smith                 1600
Michaels              3200
Ward                  2500
Martin                2500
Scott                 6000
Turner                3000

EMPL_LAST_NAME    SALARY*2
--------------- ----------
Adams                 2200
James                 1900
Miller                2600

14 rows selected.
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
prompt ========
prompt project empl last names and 2 * salaries:

select empl_last_name, salary * 2
from   empl;

-- empl salaries have NOT changed as a result!!!!!!!!

prompt ========
prompt project empl last names and salaries
prompt (note that projecting a computed column does not change
prompt that column!)

select empl_last_name, salary
from   empl;
```

-- INSERT -- 37,1 95%

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
========
project empl last names and salaries
(note that projecting a computed column does not change
that column!)

EMPL_LAST_NAME      SALARY
--------------- ----------
King                  5000
Jones                 2975
Blake                 2850
Raimi                 2450
Ford                  3000
Smith                  800
Michaels              1600
Ward                  1250
Martin                1250
Scott                 3000
Turner                1500

EMPL_LAST_NAME      SALARY
--------------- ----------
Adams                 1100
James                  950
Miller                1300
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
/*=======
    hey, guess what?
    you can ASK the select statement to use a DIFFERENT column
    heading for the columns in its result;

    ONE way to do this (using SQL and not other means)
    is with a COLUMN ALIAS

    IN a select clause,
    after a projected expression, you can put a blank and then
       the desired alternate title <-- that's the column alias

    *   if you don't surround it with DOUBLE (!!) quotes,
        it will be displayed in all-caps
    *   if you DO surround it with DOUBLE (!!) quotes,
        it will be displayed in the case shown within the quotes
    *   (it CANNOT contain blanks unless it is in double quotes)

    *   THIS DOES NOT CHANGE the COLUMN NAMES IN THAT TABLE!!!!!
        it just projects a DIFFERENT column heading for THAT
        select's results!
=====*/

-- INSERT --                                          53,1          85%
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
    IN a select clause,
    after a projected expression, you can put a blank and then
       the desired alternate title <-- that's the column alias

    *   if you don't surround it with DOUBLE (!!) quotes,
        it will be displayed in all-caps
    *   if you DO surround it with DOUBLE (!!) quotes,
        it will be displayed in the case shown within the quotes
    *   (it CANNOT contain blanks unless it is in double quotes)

    *   THIS DOES NOT CHANGE the COLUMN NAMES IN THAT TABLE!!!!!
        it just projects a DIFFERENT column heading for THAT
        select's results!
=====*/

prompt ========
prompt playing with column aliases!

select empl_last_name, salary * 2 if_raise
from empl;
```

-- INSERT -- 61,1 95%

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
========
playing with column aliases!

EMPL_LAST_NAME    IF_RAISE
--------------- ----------
King                 10000
Jones                 5950
Blake                 5700
Raimi                 4900
Ford                  6000
Smith                 1600
Michaels              3200
Ward                  2500
Martin                2500
Scott                 6000
Turner                3000

EMPL_LAST_NAME    IF_RAISE
--------------- ----------
Adams                 2200
James                 1900
Miller                2600

14 rows selected.
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
    *   if you don't surround it with DOUBLE (!!) quotes,
        it will be displayed in all-caps
    *   if you DO surround it with DOUBLE (!!) quotes,
        it will be displayed in the case shown within the quotes
    *   (it CANNOT contain blanks unless it is in double quotes)

    *   THIS DOES NOT CHANGE the COLUMN NAMES IN THAT TABLE!!!!!
        it just projects a DIFFERENT column heading for THAT
        select's results!
=====*/

prompt ========
prompt playing with column aliases!

select empl_last_name, salary * 2 if_raise
from empl;

select empl_last_name, salary * 2 "if Raise"
from empl;
```

-- INSERT -- 66,11 Bot

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
EMPL_LAST_NAME    if Raise
--------------- ----------
King                 10000
Jones                 5950
Blake                 5700
Raimi                 4900
Ford                  6000
Smith                 1600
Michaels              3200
Ward                  2500
Martin                2500
Scott                 6000
Turner                3000

EMPL_LAST_NAME    if Raise
--------------- ----------
Adams                 2200
James                 1900
Miller                2600

14 rows selected.

SQL>
```

```
    *   if you DO surround it with DOUBLE (!!) quotes,
        it will be displayed in the case shown within the quotes
    *   (it CANNOT contain blanks unless it is in double quotes)

    *   THIS DOES NOT CHANGE the COLUMN NAMES IN THAT TABLE!!!!!
        it just projects a DIFFERENT column heading for THAT
        select's results!
=====*/

prompt ========
prompt playing with column aliases!

select empl_last_name, salary * 2 if_raise
from empl;

select empl_last_name, salary * 2 "if Raise"
from empl;

select empl_last_name "Employee", salary * 1.1 "if 10% raise"
from   empl;
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
Employee        if 10% raise
--------------- ------------
King                    5500
Jones                 3272.5
Blake                   3135
Raimi                   2695
Ford                    3300
Smith                    880
Michaels                1760
Ward                    1375
Martin                  1375
Scott                   3300
Turner                  1650

Employee        if 10% raise
--------------- ------------
Adams                   1210
James                   1045
Miller                  1430

14 rows selected.

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
        select's results!
=====*/

prompt ========
prompt playing with column aliases!

select empl_last_name, salary * 2 if_raise
from empl;

select empl_last_name, salary * 2 "if Raise"
from empl;

select empl_last_name "Employee", salary * 1.1 "if 10% raise"
from   empl;

prompt ========
prompt WILL GET ERROR, column alias must be in NO quotes or DOUBLE quotes:

select empl_last_name, salary * 2 'if Raise'
from empl;

-- INSERT --                                          77,1          Bot
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
Smith                     880
Michaels                 1760
Ward                     1375
Martin                   1375
Scott                    3300
Turner                   1650

Employee        if 10% raise
--------------- ------------
Adams                    1210
James                    1045
Miller                   1430

14 rows selected.

========
WILL GET ERROR, column alias must be in NO quotes or DOUBLE quotes:
select empl_last_name, salary * 2 'if Raise'
                                  *
ERROR at line 1:
ORA-00923: FROM keyword not found where expected


SQL>
```

Question

Which of the following correctly creates a column alias without changing the actual column name in the table?

A) SELECT salary * 2 AS new_salary FROM empl;

B) SELECT salary * 2 new_salary FROM empl;

C) SELECT salary * 2 "New Salary" FROM empl;

D) All of the above

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
/*=====
     a caveat!

     computations on columns are NOT done for NULL values of that column!
=====*/

prompt ========
prompt computations will NOT be done on NULL values:

select empl_last_name, salary + commission "Total pay"
from  empl;
```

-- INSERT -- 99,1 Bot

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
========
computations will NOT be done on NULL values:

EMPL_LAST_NAME   Total pay
--------------- ----------
King
Jones
Blake
Raimi
Ford
Smith
Michaels              1900
Ward                  1750
Martin                2650
Scott
Turner                1500

EMPL_LAST_NAME   Total pay
--------------- ----------
Adams
James
Miller

14 rows selected.
```

dl313@nrs-projects:~

```
SQL> select empl_last_name, COALESCE(salary + commission, 0) "Total pay"
  2  from empl;

EMPL_LAST_NAME   Total pay
--------------- ----------
King                     0
Jones                    0
Blake                    0
Raimi                    0
Ford                     0
Smith                    0
Michaels              1900
Ward                  1750
Martin                2650
Scott                    0
Turner                1500

EMPL_LAST_NAME   Total pay
--------------- ----------
Adams                    0
James                    0
Miller                   0

14 rows selected.

SQL>
```

dl313@nrs-projects:~

```
SQL> select empl_last_name, NVL(salary + commission, 0) "Total pay"
  2  from empl;

EMPL_LAST_NAME   Total pay
-------------- ----------
King                    0
Jones                   0
Blake                   0
Raimi                   0
Ford                    0
Smith                   0
Michaels             1900
Ward                 1750
Martin               2650
Scott                   0
Turner               1500

EMPL_LAST_NAME   Total pay
-------------- ----------
Adams                   0
James                   0
Miller                  0

14 rows selected.

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
/*======
    TABLES can also have aliases!

    IN the from clause,
    you can follow a table name (or table EXPRESSEION) with a blank and
       a name (or something in double-quotes),
       and that becomes the alias for this tables THROUGHOUT that ONE
       SELECT statement!

    *   why??????
        *   it can make join conditions and references to
            columns with the same name shorter to type...
        *   it can allow joining a table to itself (!!)

    *   CLASS STYLE STANDARD: make the alias at least
        SOMEWHAT related to the table name (even if just the first
        letter of its name)
======*/
```

-- INSERT -- 106,9 94%

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
    IN the from clause,
    you can follow a table name (or table EXPRESSEION) with a blank and
       a name (or something in double-quotes),
       and that becomes the alias for this tables THROUGHOUT that ONE
       SELECT statement!

    *   why??????
        *   it can make join conditions and references to
            columns with the same name shorter to type...
        *   it can allow joining a table to itself (!!)

    *   CLASS STYLE STANDARD: make the alias at least
        SOMEWHAT related to the table name (even if just the first
        letter of its name)
======*/

prompt ========
prompt empl last names and dept names, NO table aliases:

select empl_last_name, dept_name
from   empl, dept
where  empl.dept_num = dept.dept_num;

-- INSERT --                                                109,57        91%
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
========
empl last names and dept names, NO table aliases:

EMPL_LAST_NAME  DEPT_NAME
--------------- ---------------
Miller          Accounting
Raimi           Accounting
Scott           Research
Jones           Research
Ford            Research
Smith           Research
Martin          Sales
Ward            Sales
Blake           Sales
Michaels        Sales
James           Sales

EMPL_LAST_NAME  DEPT_NAME
--------------- ---------------
Turner          Sales
Adams           Operations
King            Management

14 rows selected.
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
prompt ========
prompt empl last names and dept names, NO table aliases:

select empl_last_name, dept_name
from   empl, dept
where  empl.dept_num = dept.dept_num;

prompt ========
prompt empl last names and dept names, WITH table aliases
prompt (no difference in the output!):

select empl_last_name, dept_name
from   empl e, dept d
where  e.dept_num = d.dept_num;
```

125,0-1 Bot

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
========
empl last names and dept names, WITH table aliases
(no difference in the output!):

EMPL_LAST_NAME  DEPT_NAME
--------------- ---------------
Miller          Accounting
Raimi           Accounting
Scott           Research
Jones           Research
Ford            Research
Smith           Research
Martin          Sales
Ward            Sales
Blake           Sales
Michaels        Sales
James           Sales

EMPL_LAST_NAME  DEPT_NAME
--------------- ---------------
Turner          Sales
Adams           Operations
King            Management
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
prompt ========
prompt empl last names and dept names, WITH table aliases
prompt (no difference in the output!):

select empl_last_name, dept_name
from   empl e, dept d
where  e.dept_num = d.dept_num;

prompt ========
prompt WILL GET ERROR; once you have a table alias,
prompt you must USE it THROUGHOUT that select,
prompt EVEN in its SELECT clause...!

select empl_last_name, dept.dept_num, dept_name
from   empl e, dept d
where  e.dept_num = d.dept_num;
```

-- INSERT -- 137,1 Bot

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
Ward                Sales
Blake               Sales
Michaels            Sales
James               Sales

EMPL_LAST_NAME  DEPT_NAME
--------------- ---------------
Turner              Sales
Adams               Operations
King                Management

14 rows selected.

========
WILL GET ERROR; once you have a table alias,
you must USE it THROUGHOUT that select,
EVEN in its SELECT clause...!
select empl_last_name, dept.dept_num, dept_name
                       *
ERROR at line 1:
ORA-00904: "DEPT"."DEPT_NUM": invalid identifier


SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
prompt ========
prompt WILL GET ERROR; once you have a table alias,
prompt you must USE it THROUGHOUT that select,
prompt EVEN in its SELECT clause...!

select empl_last_name, dept.dept_num, dept_name
from   empl e, dept d
where  e.dept_num = d.dept_num;

prompt ========
prompt now using the needed table alias in select clause:

select empl_last_name, d.dept_num, dept_name
from   empl e, dept d
where  e.dept_num = d.dept_num;
```

-- INSERT -- 145,1 Bot

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect05-2.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect05-2 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
========
now using the needed table alias in select clause:

EMPL_LAST_NAME  DEP DEPT_NAME
--------------- --- --------------
Miller          100 Accounting
Raimi           100 Accounting
Scott           200 Research
Jones           200 Research
Ford            200 Research
Smith           200 Research
Martin          300 Sales
Ward            300 Sales
Blake           300 Sales
Michaels        300 Sales
James           300 Sales

EMPL_LAST_NAME  DEP DEPT_NAME
--------------- --- --------------
Turner          300 Sales
Adams           400 Operations
King            500 Management

14 rows selected.
```

Question

What happens when you try to perform a computation on a column that contains NULL values?

A) The computation will return 0 for NULL values.

B) The computation will be performed, and the result will replace the NULL.

C) The computation will be skipped, and NULL will remain as the result.

D) The query will return an error if NULL values are encountered.