CS 121: Relational Databases

The Entity-Relationship Model I

Designing Database Applications

Database applications are large and complex

A few of the many design areas:

- Database schema (physical/logical/view)
- Programs that access and update data
- Security constraints for data access

Also requires familiarity with the problem domain

- Domain experts *must* help drive requirements

General Approach

Collect user requirements

- Information that needs to be represented
- Operations to perform on that information
- Several techniques for representing this info, e.g. UML

Develop a **conceptual schema** of the database

- A high-level representation of the database's structure and constraints
  - Physical *and* logical design issues are ignored at this stage
- Follows a <u>specific</u> data model
  - Values, relationships, constraints, etc.
- Often represented graphically

Conceptual Schema

Also need to create a <u>specification of functional requirements</u>

- “What operations will be performed against the data?”
- Updating data, adding data, deleting data, …

Designer can use functional requirements to verify the conceptual schema

- Is each operation possible?
- How complicated or involved is it?
- Performance or scalability concerns?

Implementation Phases

Once conceptual schema and functional requirements are verified:

- Convert conceptual schema into an **implementation data model**
- Want to have a simple mapping from conceptual model to implementation model

Finally: any necessary physical design

- Not always present!
- Smaller applications have few physical design concerns
- Larger systems usually need additional design and tuning (e.g. indexes, disk-level partitioning of data)

Importance of Design Phase

Not all changes have the same impact!

Physical-level changes have the least impact

- Typically affect performance, scalability, reliability
- Little to no change in functionality

Logical-level changes are typically *much* bigger

- E.g. changing table schemas, dividing/merging schemas, etc.
- Affects how to interact with the data…
- Also affects what is even *possible* to do with the data

<u>Very important</u> to spend time up front designing the database schema

Design Decisions

Many different ways to represent data

Must avoid two major problems:

1. Unnecessary redundancy
   - Redundant information wastes space
   - Greater potential for inconsistency!
   - Ideally: each fact appears in exactly one place
2. Incomplete representation
   - Schema must be able to fully represent all details and relationships required by the application

More Design Decisions

Even with correct design, usually many other concerns

- How easy/hard is it to access useful information? (e.g. reporting or summary info)
- How hard is it to update the system?
- Performance considerations?
- Scalability considerations?

Schema design requires a good balance between aesthetic and practical concerns

- Frequently need to make compromises between conflicting design principles

The Entity-Relationship Model

A *very* common model for schema design

- Also written as “E-R model”

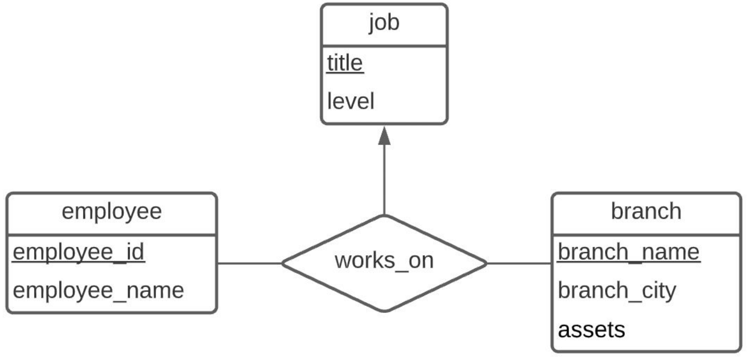
Allows for specification of complex schemas in graphical form

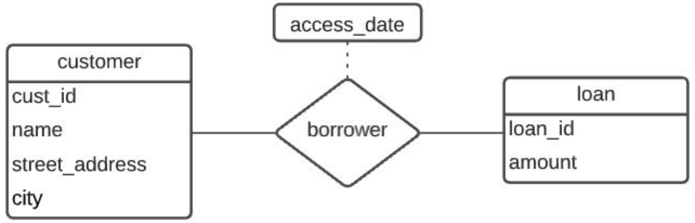
Basic concepts are simple, but can also represent very sophisticated abstractions

- e.g. type hierarchies

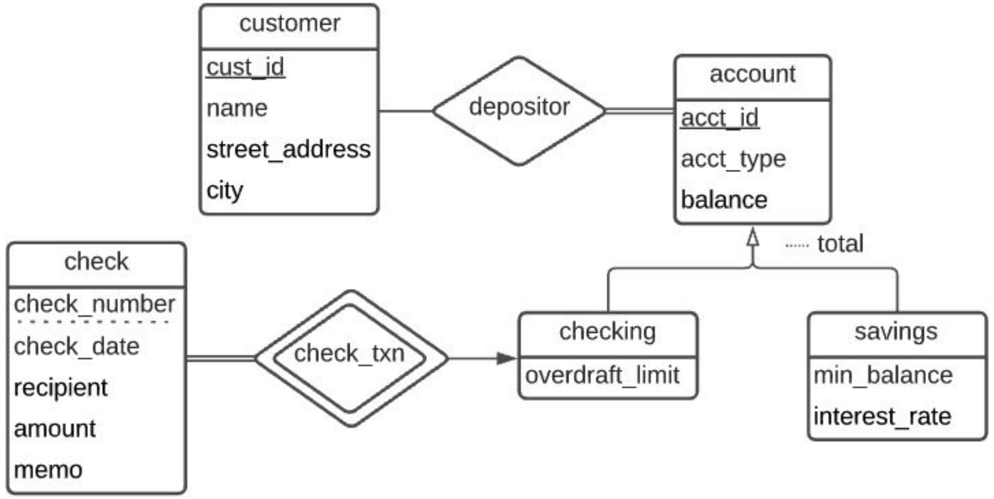
Can be mapped very easily to the relational model!

- Simplifies implementation phase
- Mapping process can be automated by design tools





Example (Preview): Bank Account Diagram



Over the next two lectures, we'll learn the building blocks of the ER model to represent entities, relationships, and constraints for a database schema, using a version of the bank account database we've been seeing so far

Entities and Entity-Sets

An **entity** is any “thing” that can be uniquely represented

- e.g. a product, an employee, an adoption application, a software defect

Each entity has a set of **attributes**

Entities are uniquely identified by some set of attributes

An **entity-set** is a named collection of entities of the same type, with the same attributes

- Can have multiple entity-sets with same entity type, representing different (possibly overlapping) sets

Entities and Entity-Sets (2)

An entity has a set of attributes

- Each attribute has a name (e.g. *artist_name*) and domain (e.g. *string names for all artist names in playlists*)
- Each attribute also has a corresponding value (e.g. *"Moby"*)

Entity-sets also specify a set of attributes

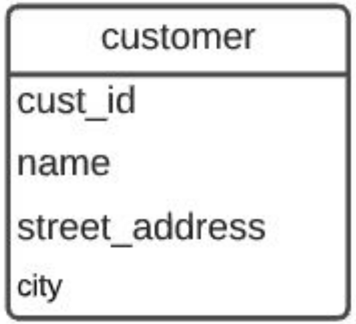
- Every entity in the entity-set has the same set of attributes
- Every entity in the entity-set has its own value for each attribute

Diagramming an Entity-Set

Example: a *customer* entity-set

Attributes:

- *cust_id*
- *name*
- *street_address*
- *city*



Entity-set is denoted by a box

Name of entity-set is given in the top part of box

Attributes are listed in the lower part of the box

Various ERD tools to help diagram, including LucidChart and Visual Paradigm (which you may use on HW)

Relationships

A **relationship** is an association between two or more entities

- e.g. a bank loan, and the customer who owns it

A **relationship-set** is a named collection of relationships of the same type

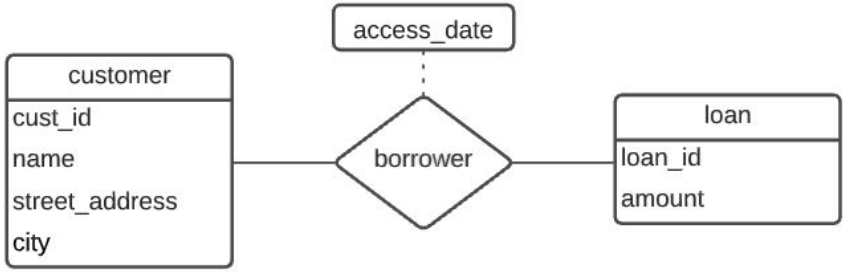
- i.e. involving the same entities

Formally, a relationship-set is a mathematical relation involving n entity-sets, $n \geq 2$

- $E_1, E_2, \ldots, E_n$ are entity sets; $e_1, e_2, \ldots$ are entities in $E_1, E_2, \ldots$
- A relationship set R is a subset of:

  $$\{\, (e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n \,\}$$
- $(e_1, e_2, \ldots, e_n)$ is a specific relationship in R

Relationships (2)



Entity-sets **participate** in relationship-sets

- Specific entities participate in a **relationship instance**

Example: bank loans

- *customer* and *loan* are entity-sets
  - (555-55-5555, Jackson, Woodside) is a *customer* entity
  - (L-14, 1500) is a *loan* entity
- *borrower* is a relationship-set
  - *customer* and *loan* participate in *borrower*
  - *borrower* contains a relationship instance that associates customer “Jackson” and loan “L-14”

(we will go into the details of relationship diagrams shortly)

Relationships and Roles

An entity’s **role** in a relationship is the function that the entity fills

Example: a *purchase* relationship between a *product* and a *customer*

- the product’s role is that it was purchased
- the customer did the purchasing

Roles are usually obvious, and therefore unspecified

- Entities participating in relationships are distinct…
- Names clearly indicate the roles of various entities…
- In these cases, roles are left unstated.

Relationships and Roles (2)

Sometimes the roles of entities are *not* obvious

- Situations where entity-sets in a relationship-set are *not* distinct

Example: a relationship-set named *works_for*, specifying employee/manager assignments

- Relationship involves two entities, and both are *employee* entities

Roles are given names to distinguish entities

The relationship is a set of entities ordered by role:

(*manager*, *worker*)

- First entity's role is named *manager*
- Second entity's role is named *worker*

Relationships and Attributes

Relationships can also have attributes!

- Called **descriptive attributes**
- They describe a particular relationship
- They *do not* identify the relationship!

Example:

- The relationship between a software defect and an employee can have a *date_assigned* attribute

Note: this distinction between entity attributes and relationship attributes is not made by relational model

- Entity-relationship model is a higher level of abstraction than the relational model

Relationships and Attributes (2)

Specific relationships are identified *only* by the participating entities

- …not by any relationship attributes!
- Different relationships are allowed to have the same value for a descriptive attribute
- (This is why entities in an entity-set must be uniquely identifiable.)

Given:

- Entity-sets A and B, both participating in a relationship-set R

Specific entities $a \in A$ and $b \in B$ can only have <u>one</u> relationship instance in R

- Otherwise, we would require more than just the participating entities to uniquely identify relationships

Degree of Relationship Set

Most relationships in a schema are binary

- Two entities are involved in the relationship

Sometimes there are ternary relationships

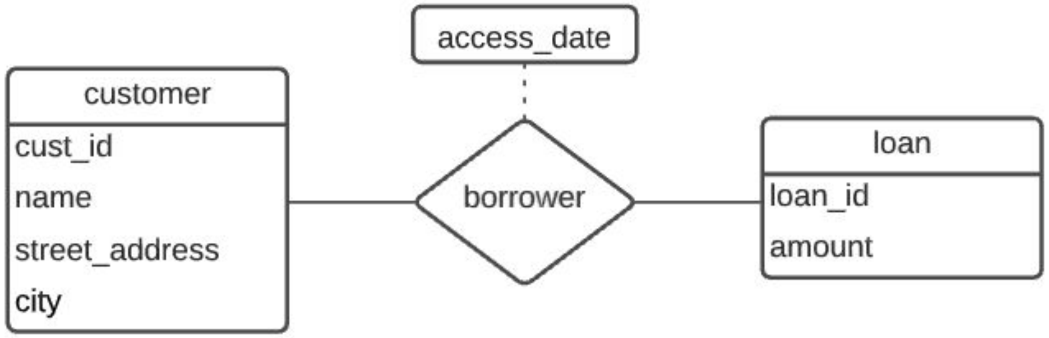
- Three entities are involved
- Far less common, but still useful at times

The number of entity-sets that participate in a relationship-set is called its **degree**

- Binary relationship: degree = 2
- Ternary relationship: degree = 3

Diagramming a Relationship-Set

Example: the *borrower* relationship-set between the *customer* and *loan* entity-sets



Relationship-set is a diamond

- Connected to participating entity-sets by solid lines

Relationship-set can have descriptive attributes

- Listed in another box, connected with a dotted-line

Attribute Structure

Each attribute has a domain or value set

- Values come from that domain or value set

Simple attributes are atomic – they have no subparts

- e.g. *amount* attribute is a single numeric value

Composite attributes have subparts

- Can refer to composite attribute as a whole
- Can also refer to subparts individually
- e.g. *address* attribute, composed of *street*, *city*, *state*, *postal_code* attributes

Attribute Cardinality

Single-valued attributes only store one value

- e.g. a *customer*'s *cust_id* only has one value

Multi-valued attributes store zero or more values

- e.g. a *customer* can have multiple *phone_number* values

A multi-valued attribute stores a set of values, not a multiset

Different *customer* entities can have different sets of phone numbers

Lower and upper bounds can be specified too

- Can set upper bound on *phone_number* to 2

Attribute Source

Base attributes (aka **source** attributes) are stored in the database

- e.g. the *birth_date* of a *customer* entity

Derived attributes are computed from other attributes

- e.g. the *age* of a *customer* entity would be computed from their *birth_date*

Practice: Extending Our *customers* Entity Set

Example: Extend *customers* with more detailed info

Suppose we want to represent a *customer* entity set with the following attributes:

- name
- address
- phone
- birth_date
- age

Structure: Which would make sense as simple (atomic) vs. composite?

Cardinality: Which would make sense as single- (unique) vs. multi-valued (multiple values possible for one entity)?

Source: Which would make sense as base vs. derived?

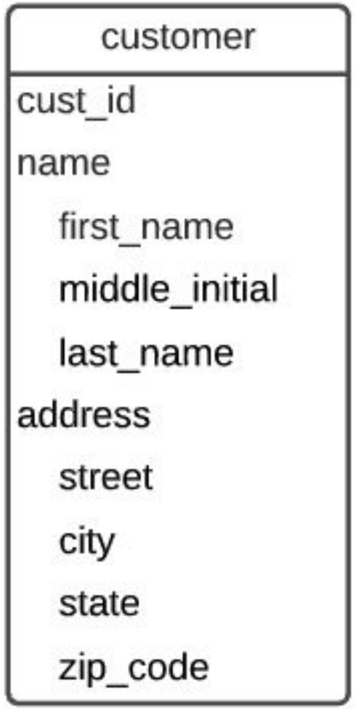
Diagramming Attributes

Example: Extend *customers* with more detailed info

cust_id is a simple attribute

Composite attributes are shown as a hierarchy of values

- Indented values are components of the higher-level value
- e.g. *name* is comprised of *first_name*, *middle_initial*, and *last_name*

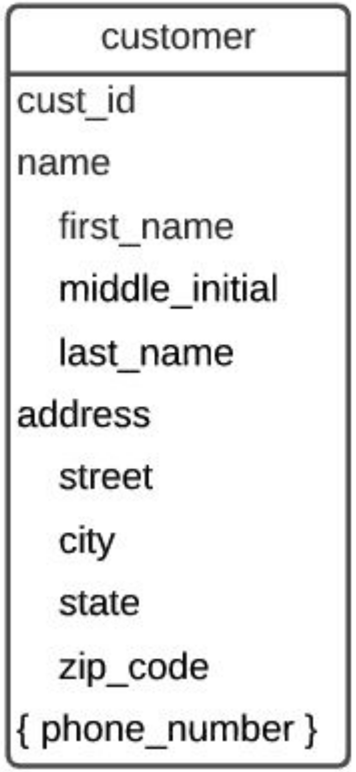


Diagramming Attributes (2)

Example: Extend *customers* with more detailed info

Multivalued attributes are enclosed with curly-braces

- e.g. each customer can have zero or more phone numbers

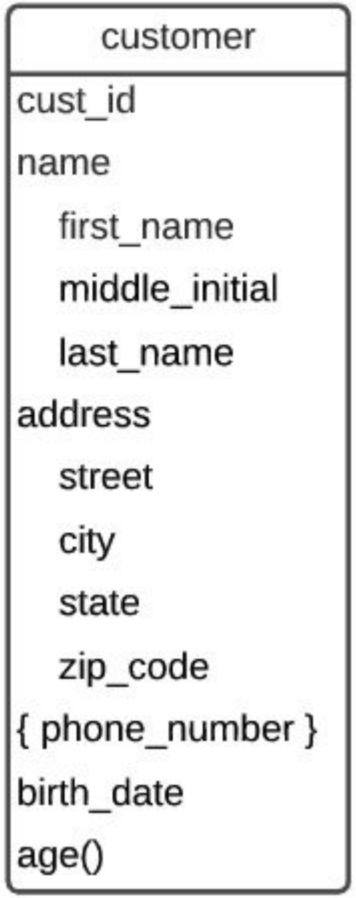


Diagramming Attributes (3)

Example: Extend customers with more detailed info

Derived attributes are indicated by a trailing set of parentheses

- e.g. each customer has a base attribute recording their date of birth
- Also a derived attribute that reports the customer's current age



Representing Constraints

E-R model can represent different kinds of constraints

- Mapping cardinalities
- Key constraints in entity-sets
- Participation constraints

Allows more accurate modeling of application's data requirements

- Constrain design so that schema can only represent valid information

Enforcing constraints can impact performance…

- Still ought to specify them in the design!
- Can always leave out constraints at implementation time (don't prematurely optimize in design stage)

Mapping Cardinalities

Mapping cardinality represents:

“How many other entities can be associated with an entity, via a particular relationship set?”

Example:

- How many *customer* entities can the *borrower* relationship associate with a single *loan* entity?
- How many *loans* can *borrower* relationship associate with a single *customer* entity?
- Specific answer depends on what is being modeled

Also known as the **cardinality ratio**

Easiest to reason about with binary relationships

Application of Cardinalities

In Assignment 6, you are working with an airport database, starting with the ER design process

In the next few slides, we'll learn different types of cardinalities, which will help you identify the mapping cardinality of:

1. Seats to aircraft type
2. Tickets to travelers
3. Aircraft type to flights

Mapping Cardinalities (2)

Given:

- Entity-sets *A* and *B*
- Binary relationship-set *R* associating *A* and *B*

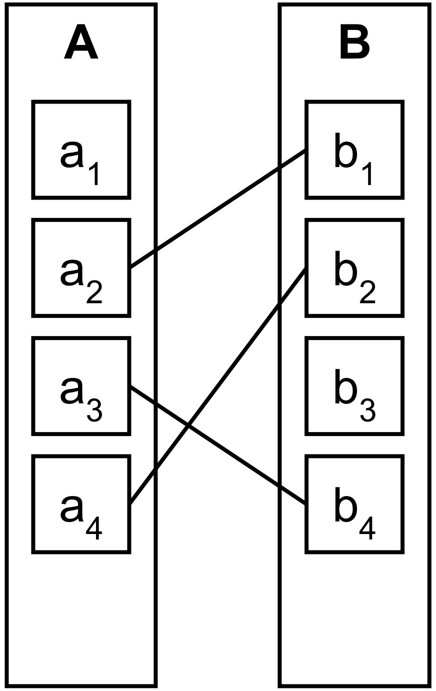
One-to-one mapping (1:1)

- An entity in *A* is associated with *at most* one entity in *B*
- An entity in *B* is associated with *at most* one entity in *A*

Are any of the following 1:1?

1. Seats to aircraft type
2. **Tickets to travelers** - one traveler can only have one ticket, one ticket can only be for one traveler
3. Aircraft type to flights

2. Tickets to travelers
3. Aircraft type to flights



Mapping Cardinalities (3)

One-to-many mapping (1:M)

- An entity in *A* is associated with *zero or more* entities in *B*
- An entity in *B* is associated with *at most* one entity in *A*

Many-to-one mapping (M:1)

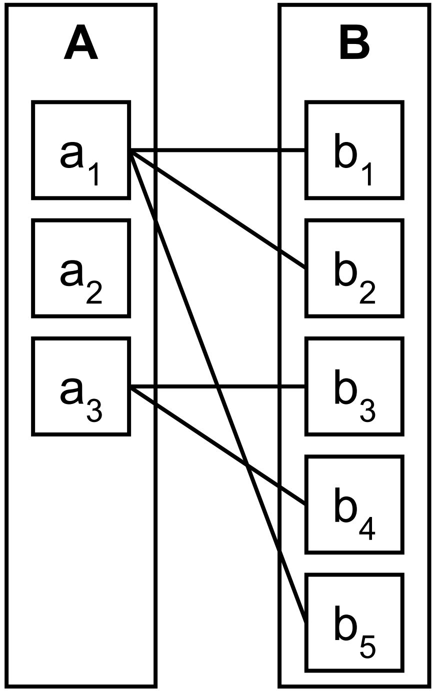
- Opposite of one-to-many
- An entity in *A* is associated with *at most* one entity in *B*
- An entity in *B* is associated with *zero or more* entities in *A*

Are any of the following 1:M or M:1?

Are any of the following 1:M or M:1?

1. Seats to aircraft type
2. Tickets to travelers
3. Aircraft type to flights

1. **Seats to aircraft type - M:1, a seat cannot belong to more than one aircraft, but an aircraft can have more than one seat**
2. Tickets to travelers
3. **Aircraft type to flights - 1:M, an aircraft can participate in many flights, but each flight can only have one aircraft**

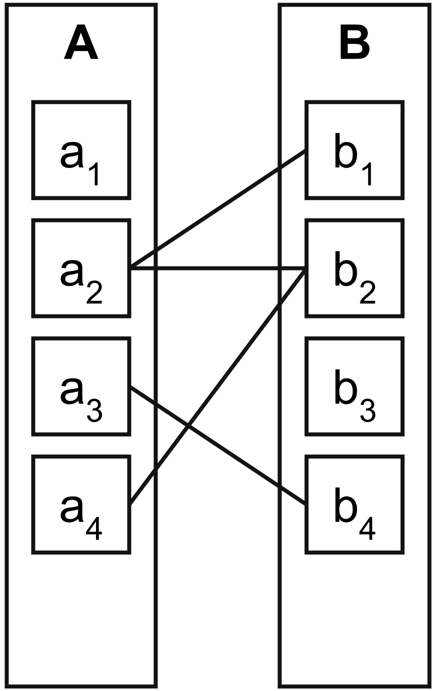


Mapping Cardinalities (4)

Many-to-many mapping

- An entity in A is associated with *zero or more* entities in B
- An entity in B is associated with *zero or more* entities in A

Extending the airport database, we could imagine an M:M cardinality for an employee working at more than one airport and an airport having more than one employee



Mapping Cardinalities (5)

Which mapping cardinality is most appropriate for a given relationship?

- Answer depends on what you are trying to model!
- Could just use many-to-many relationships everywhere, but that wouldn't make sense.

Goal:

- Constrain the mapping cardinality to most accurately reflect what should be allowed
- Database can enforce these constraints automatically
- Good schema design reduces or eliminates the *possibility* of storing bad data

Another Example: Deciding on *borrower* relationship between *customer* and *loan*

One-to-one mapping:

- Each customer can have only one loan
- Customers can't share loans
  (e.g. with spouse or business partner)

One-to-many mapping:

- A customer can have multiple loans
- Customers still can't share loans

Many-to-one mapping:

- Each customer can have only one loan
- Customers can share loans

Many-to-many mapping:

- A customer can have multiple loans
- Customers can share loans too

Best choice for *borrower*: many-to-many mapping

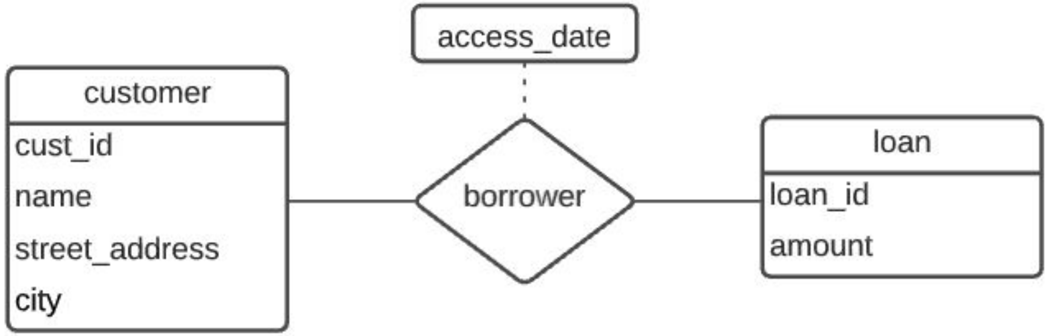
- ***Handles real-world needs!***

Diagramming Cardinalities

In relationship-set diagrams:

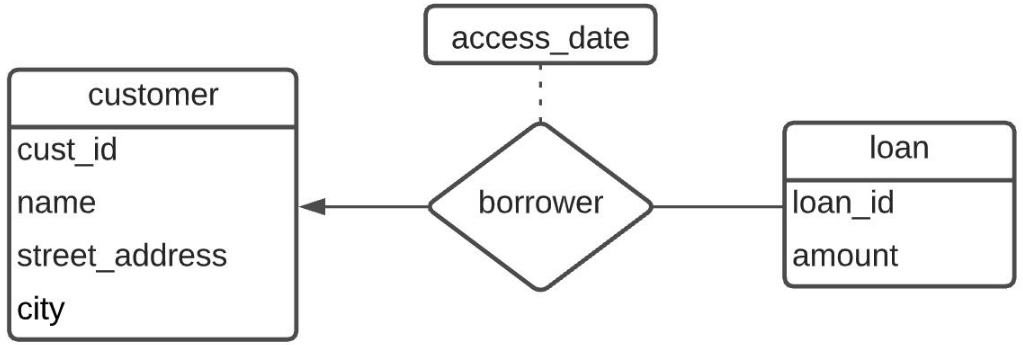
- an arrow towards an entity represents “one”
- a simple line represents “many”
- arrow is *always* towards the entity

Many-to-many mapping between *customer* and *loan*:



Diagramming Cardinalities (2)

One-to-many mapping between *customer* and *loan*:



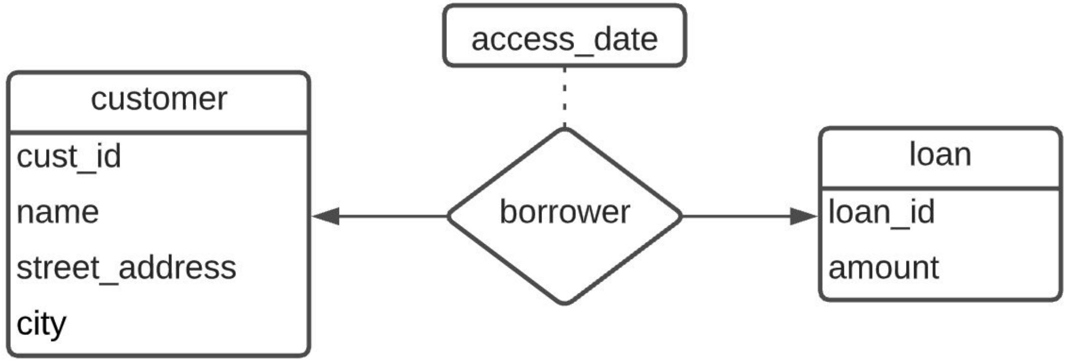
Each customer can have multiple loans

A loan is owned by exactly one customer

- (Actually, this is technically "at most one". Participation constraints will allow us to say "exactly one.")

Diagramming Cardinalities (3)

One-to-one mapping between *customer* and *loan*:



Each customer can have only one loan

A loan is owned by exactly one customer

Entity-Set Keys

Entities in an entity-set must be uniquely distinguishable using their values

- Entity-set: each entity is unique

E-R model also includes the notion of keys:

- Superkey: a set of one or more attributes that can uniquely identify an entity
- Candidate key: a *minimal* superkey
- Primary key: a candidate key chosen by DB designer as the primary means of accessing entities (ideally something that is robust and rarely, if ever, changes)

Keys are a property of the entity-set (not of the relationship-set)

- They apply to *all* entities in the entity-set

Choosing Candidate Keys

Candidate keys constrain the values of the key attributes

- No two entities can have the same values for those attributes
- Need to ensure that database can actually represent all expected circumstances

Simple example: *customer* entity-set

- Using customer name as a candidate key is clearly bad design: different customers can have the same name

A less-simple example: *applicants* entity-set (from an animal shelter database)

- Should *phone* be a candidate key?

Choosing Primary Keys

An entity-set may have multiple candidate keys

The primary key is the candidate key most often used to reference entities in the set

- In logical/physical design, primary key values will be used to represent relationships
- External systems may also use primary key values to reference entities in the database (e.g. an application referencing a student id number)

The primary key attributes should *never* change!

- If ever, it should be *extremely* rare.

Choosing Keys: Performance

Large, complicated, or multiple-attribute keys are generally slower

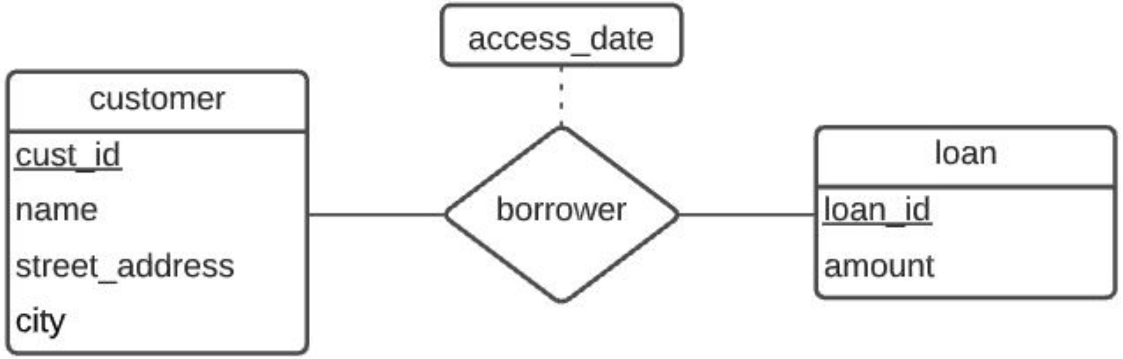
- Use smaller, single-attribute keys
    - (You can always generate them…)
- Use faster, fixed-size types
    - e.g. **`INT`** or **`BIGINT`**

Especially true for primary keys!

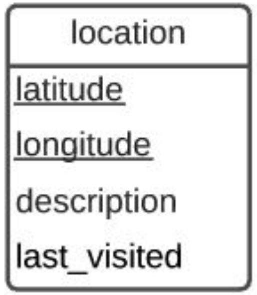
- Values used in both database and in access code
- Use something small and simple, if possible

Diagramming Primary Keys

In an entity-set diagram, all attributes in the primary key have an underlined name



Another example: a geocache *location* entity-set



Keys and Relationship-Sets

Need to be able to distinguish between individual relationships in a relationship-set as well

- Relationships aren't distinguished by their descriptive attributes
- (They might not even have descriptive attributes)

Relationships are identified by the *entities* participating in the relationship

- Specific relationship instances are uniquely identified by the primary keys of the participating entities
- Example: *borrower* relates *customer_id* and *loan_id* primary keys

Keys and Relationship-Sets (2)

Given:

- *R* is a relationship-set with no descriptive attributes
- Entity-sets $E_1, E_2, \ldots, E_n$ participate in *R*
- *primary_key*(E_i) denotes set of attributes in E_i that represent the primary key of E_i

A relationship instance in *R* is identified by

$$\textit{primary\_key}(E_1) \cup \textit{primary\_key}(E_2) \cup \ldots \cup \textit{primary\_key}(E_n)$$

- This is a superkey
- Is it a candidate key?
  - Depends on the *mapping cardinality* of the relationship set!

Keys and Relationship-Sets (3)

If R also has descriptive attributes $\{a_1, a_2, \ldots\}$, a relationship instance is **described** by:

$$primary\_key(E_1) \cup primary\_key(E_2) \cup \ldots \cup primary\_key(E_n) \cup \{a_1, a_2, \ldots\}$$

- Not a minimal superkey!
- By definition, there can only be one relationship between $\{E_1, E_2, \ldots, E_n\}$ in the relationship-set
  - i.e. the descriptive attributes **do not identify** specific relationships

Thus, just as before, this is also a superkey:

$$primary\_key(E_1) \cup primary\_key(E_2) \cup \ldots \cup primary\_key(E_n)$$

Relationship-Set Primary Keys

What is the primary key for a binary relationship-set?

- Must also be a candidate key
- Depends on the mapping cardinalities

Relationship-set *R*, involving entity-sets *A* and *B*

- If mapping is **many-to-many**, primary key is:

  $primary\_key(A) \cup primary\_key(B)$
- Any given entity's primary-key values can appear multiple times in *R*
- We need both entity-sets' primary key attributes to uniquely identify relationship instances

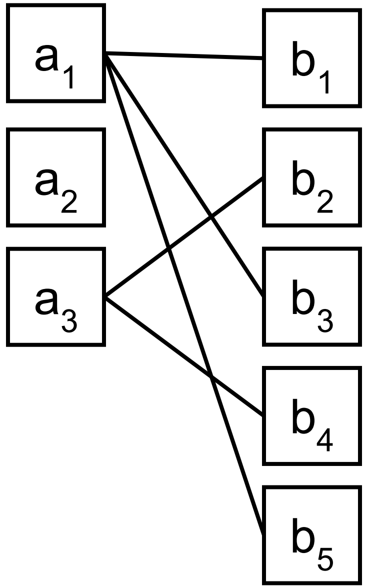
Relationship-Set Primary Keys (2)

Relationship-set *R*, involving entity-sets *A* and *B*

- Individual relationships are described by *primary_key*(*A*) ∪ *primary_key*(*B*)

If mapping is **one-to-many**:

- Entities in *B* associated with *at most* one entity in *A*
- A given value of *primary_key*(*A*) can appear in multiple relationships
- Each value of *primary_key*(*B*) can appear only once
- Relationships in *R* are uniquely identified by *primary_key*(*B*)
- *primary_key*(*B*) is primary key of relationship-set

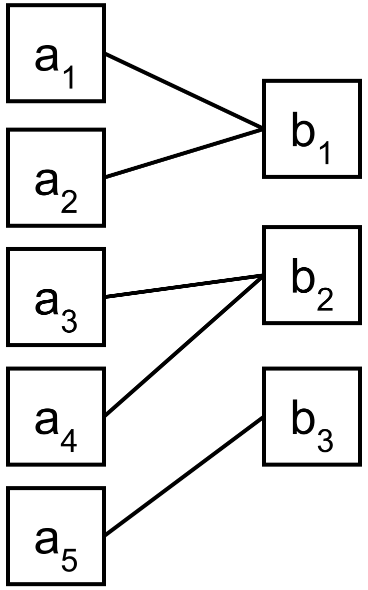


Relationship-Set Primary Keys (3)

Relationship-set *R*, involving entity-sets *A* and *B*

Many-to-one is exactly the opposite of one-to-many

- *primary_key*(*A*) uniquely identifies relationships in *R*



Relationship-Set Primary Keys (4)

Relationship-set *R*, involving entity-sets *A* and *B*

If mapping is **one-to-one**:

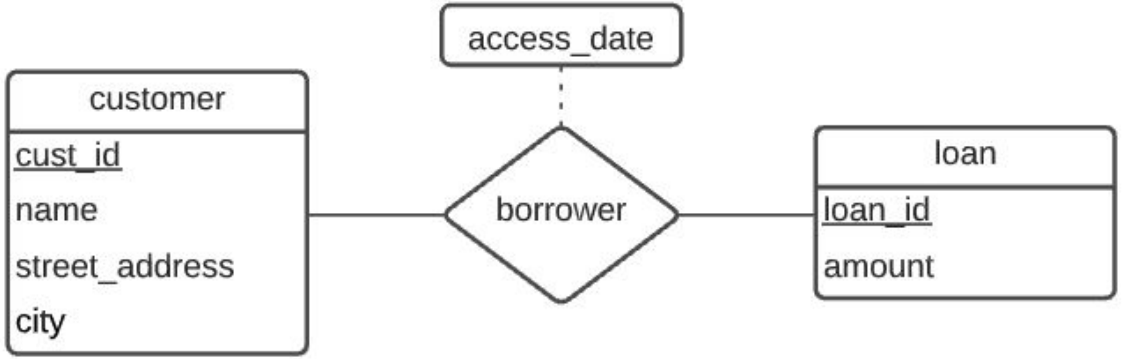
- Entities in *A* associated with *at most* one entity in *B*
- Entities in *B* associated with *at most* one entity in *A*
- Each entity's key-value can appear only once in *R*
- *Either* entity-set's primary key can be primary key of *R*

For one-to-one mapping, *primary_key*(*A*) and *primary_key*(*B*) are <u>both</u> candidate keys

- Make sure to enforce both candidate keys in the implementation schema!

Example

What is the primary key for *borrower* ?



borrower is a many-to-many mapping

- Relationship instances are described by (*cust_id*, *loan_id*, *access_date*)
- Primary key for relationship-set is (*cust_id*, *loan_id*)

Participation Constraints

Given entity-set *E*, relationship-set *R*

- How many entities in *E* participate in *R* ?
- In other words, what is minimum number of relationships that each entity in *E* *must* participate in?

If <u>every</u> entity in *E* participates in at least one relationship in *R*, then:

- *E*'s participation in *R* is **total**

If only some entities in *E* participate in relationships in *R*, then:

- *E*'s participation in *R* is **partial**

Participation Constraints (2)

Example: *borrower* relationship between *customer* and *loan*

A customer might not have a bank loan

- Could have a bank account instead
- Could be a new customer
- Participation of *customer* in *borrower* is **partial**

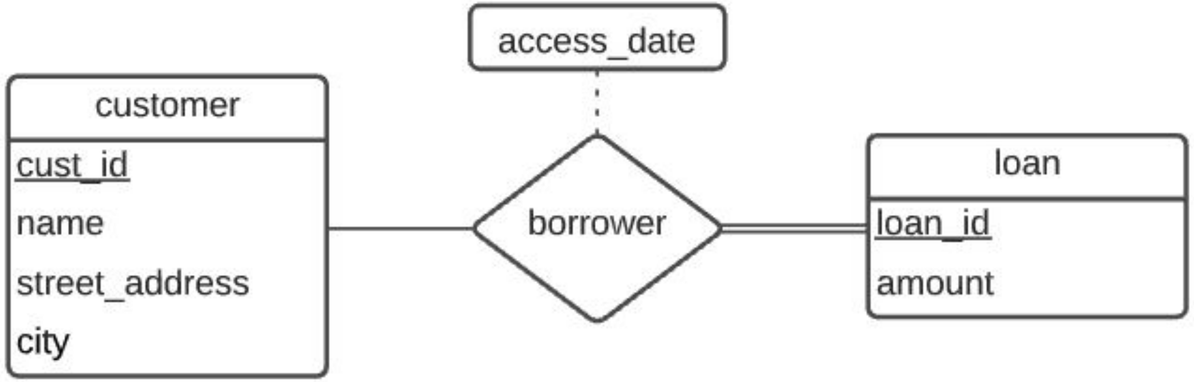
Every loan definitely has at least one customer

- Doesn't make any sense not to!
- Participation of *loan* in *borrower* is **total**

Diagramming Participation

Can indicate participation constraints in entity-relationship diagrams

- Partial participation shown with a single line
- Total participation shown with a double line



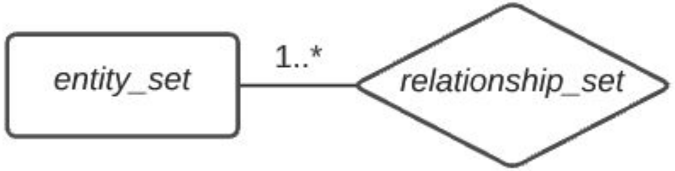
Numerical Constraints

Can also state numerical participation constraints

- Specifies how many different relationship instances each entity in the entity-set can participate in
- Indicated on link between entity and relationship

Form: lower..upper (inclusive)

- * means “unlimited”
- 1..* = one or more
- 0..3 = between zero and three, inclusive
- etc.



Numerical Constraints (2)

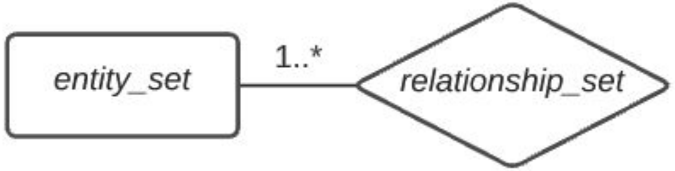
Can also state mapping constraints with numerical participation constraints

Total participation:

- Lower bound at least 1

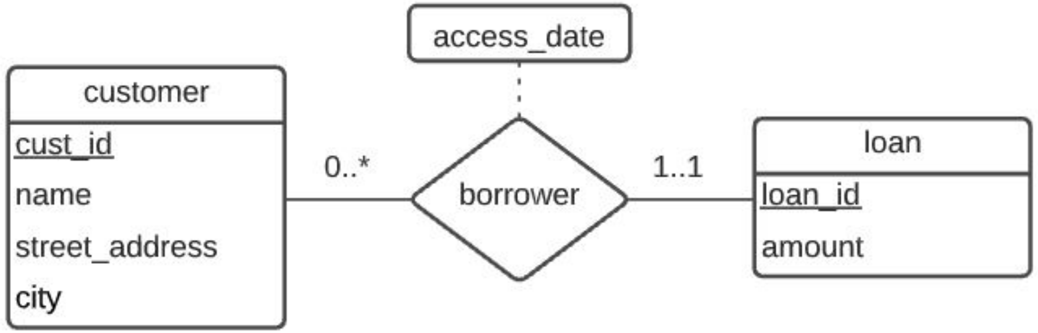
Partial participation:

- Lower bound is 0



Numerical Constraint Example

What does this mean?



Each *customer* entity may participate in zero or more relationships in this relationship-set

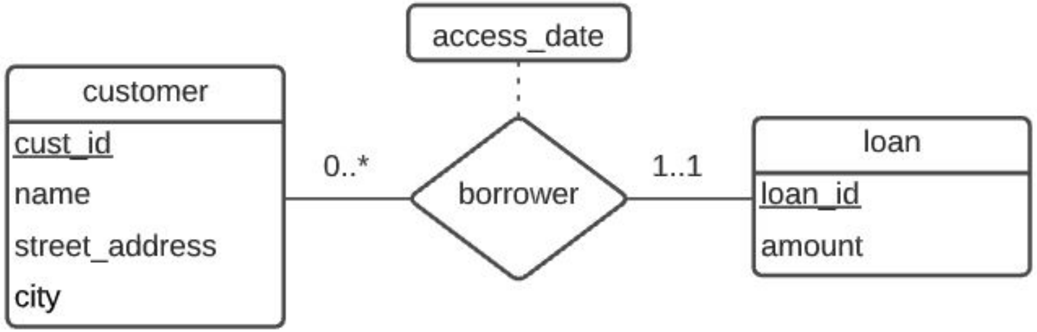
- *A customer can have zero or more loans.*

Each *loan* entity must participate in *exactly one* relationship (no more, no less) in this relationship-set

- *Each loan must be owned by exactly one customer.*

Numerical Constraint Example (2)

What is the mapping cardinality of *borrower* regarding *customer* to *loan*?



From last slide:

- *A customer can have zero or more loans*
- *Each loan must be owned by exactly one customer.*

This is a <u>one-to-many</u> mapping from *customer* to *loan*

Diagramming Roles

Entities have **roles** in relationships

- An entity's role indicates the entity's function in the relationship
- e.g. role of customer in *borrower* relationship-set is that they own the loan

Sometimes roles are ambiguous

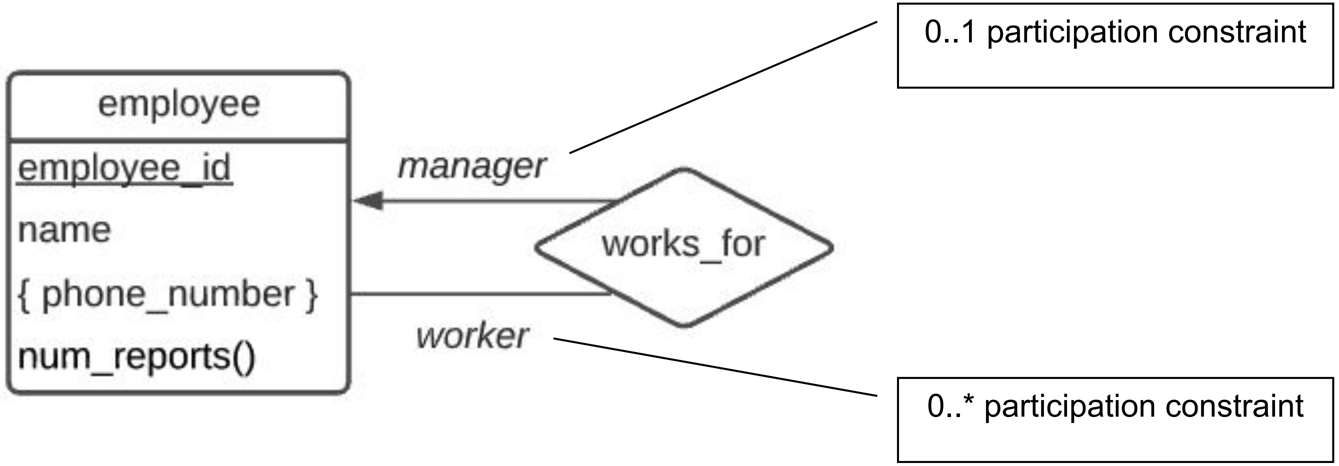
- e.g. when the same kind of entity is involved in a relationship multiple times

Example: *works_for* relationship

- Relationship is between two *employee* entities
- One is the *manager*; the other is the *worker*

Diagramming Roles (2)

If roles need to be indicated, put labels on the lines connecting entity to relationship



works_for relationship-set is one-to-many from managers to workers

Weak Entity-Sets

Sometimes an entity-set doesn't have distinguishing attributes

- Can't define a primary key for the entity-set!
- Called a **weak entity-set**

Example:

- Checking accounts have a unique account number
- Checks have a check number
    - Unique for a given account, but not across all accounts!
    - Number only makes sense in context of a particular account
- Want to store check transactions in the database

Weak Entity-Sets (2)

Weak entity-sets *must* be associated with another (strong) entity-set

- Called the **identifying entity-set**, or **owner entity-set**
- The identifying entity-set owns the weak entity-set
- Association called the **identifying relationship**

Every weak entity *must* be associated with an identifying entity

- Weak entity's participation in relationship-set is total
- The weak entity-set is **existence dependent** on the identifying entity-set
- If the identifying entity is removed, its weak entities should also cease to exist
- *(this is where cascade-deletes may be appropriate…)*

Weak Entity-Set Keys

Weak entity-sets don’t have a primary key

- Still need to distinguish between weak entities associated with a particular strong entity

Weak entities have a **discriminator**

- A set of attributes that distinguishes between weak entities associated with a strong entity
- Also known as a **partial key**

Checking account example:

- The check number is the discriminator for check transactions

Weak Entity-Set Keys (2)

Using discriminator, can define a primary key for weak entity-sets

For a weak entity-set *W*, and an identifying entity-set *S*, primary key of *W* is:

$$primary\_key(S) \cup discriminator(W)$$

Checking account example:

- *account_number* is primary key for checking accounts
- *check_number* is discriminator (partial key) for checks
- Primary key for check transactions would be (*account_number*, *check_number*)

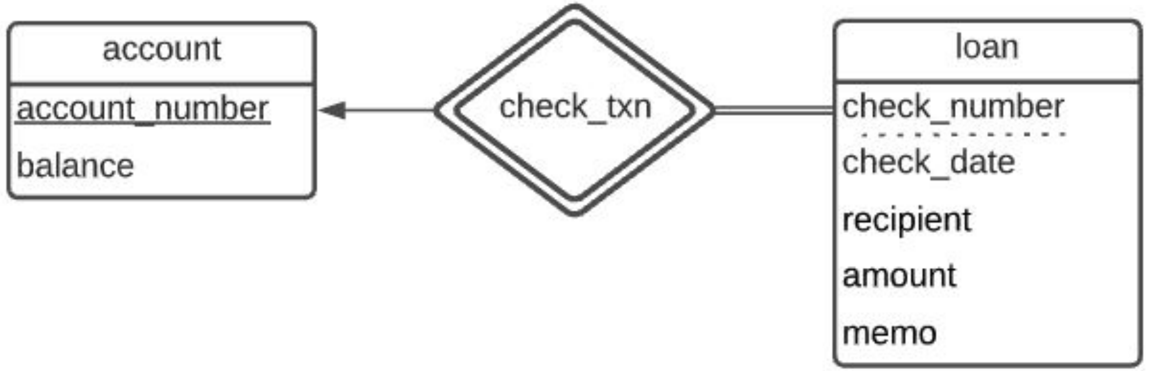
Diagramming Weak Entity-Sets

Weak entity-sets drawn similarly to strong entity-sets

- Difference: discriminator attributes are underlined with a dashed underline

Identifying relationship to the owning entity-set is indicated with a double diamond

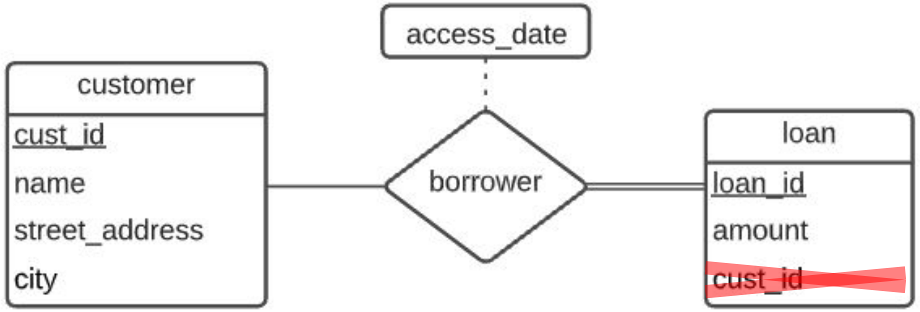
- One-to-many mapping
- Total participation on weak entity side



Common Attribute Mistakes

Don't include entity-set primary key attributes on other entity-sets!

- e.g. customers and loans, in a one-to-many mapping



Even if every loan is owned by only one customer, this is still wrong

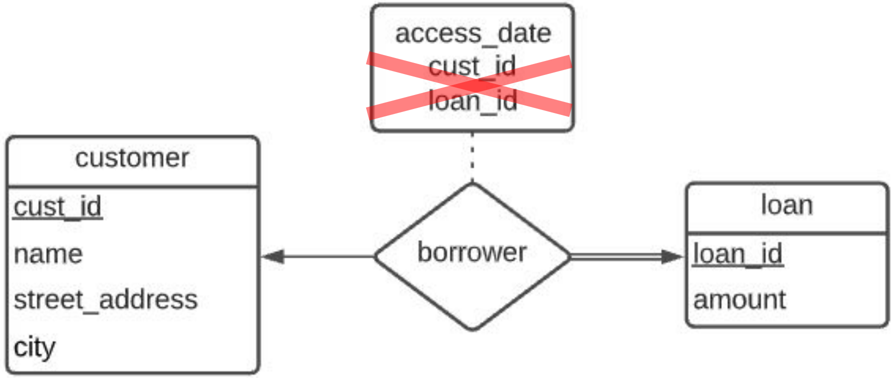
- The association is recorded by the *relationship*, so specifying foreign key attributes on the entity-set is redundant

Common Attribute Mistakes (2)

Don’t include primary key attributes as descriptive attributes on relationship-set, either!

This time, assume *borrower* is a 1:1 mapping

- IDs used as descriptive attributes on *borrower*



Again, this is implicit in the relationship