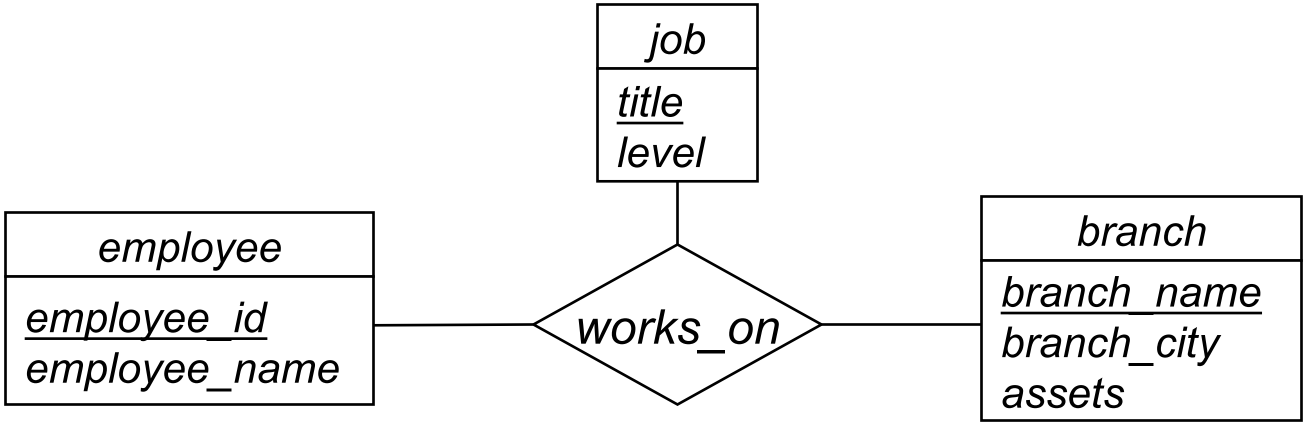
ENTITY-RELATIONSHIP MODEL III

CS121: Relational Databases

Fall 2018 – Lecture 16

N-ary Relationships

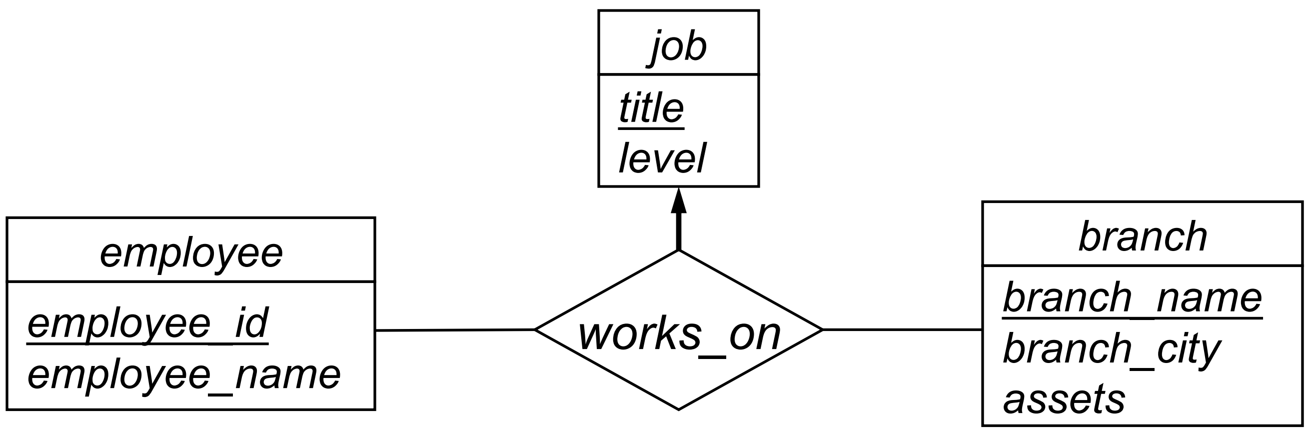
- Can specify relationships of degree > 2 in E-R model
- Example:



  - Employees are assigned to jobs at various branches
  - Many-to-many mapping: any combination of employee, job, and branch is allowed
  - An employee can have several jobs at one branch

N-ary Mapping Cardinalities

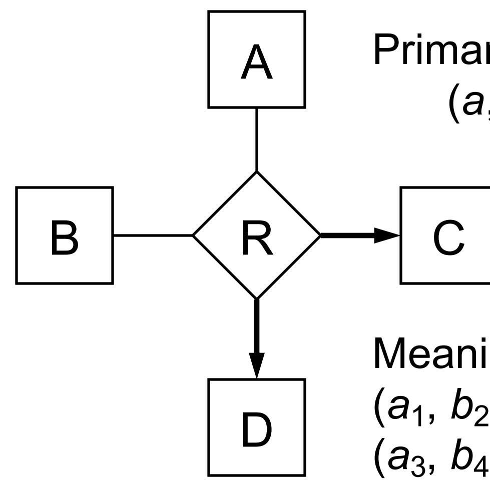
- Can specify *some* mapping cardinalities on relationships with degree > 2
- Each combination of employee and branch can only be associated with <u>one</u> job:



  - Each employee can have only one job at each branch

N-ary Mapping Cardinalities (2)

- For degree > 2 relationships, we only allow at most <u>one</u> edge with an arrow
- <u>Reason</u>: multiple arrows on N-ary relationship-set is ambiguous
  - (several meanings have been defined for this in the past)
- Relationship-set R associating entity-sets $A_1, A_2, \ldots, A_n$
  - No arrows on edges $A_1, \ldots, A_i$
  - Arrows are on edges to $A_{i+1}, \ldots, A_n$
- Meaning 1 (the simpler one):
  - A particular combination of entities in $A_1, \ldots, A_i$ can be associated with at most one set of entities in $A_{i+1}, \ldots, A_n$
  - Primary key of R is union of primary keys from set $\{ A_1, A_2, \ldots, A_i \}$

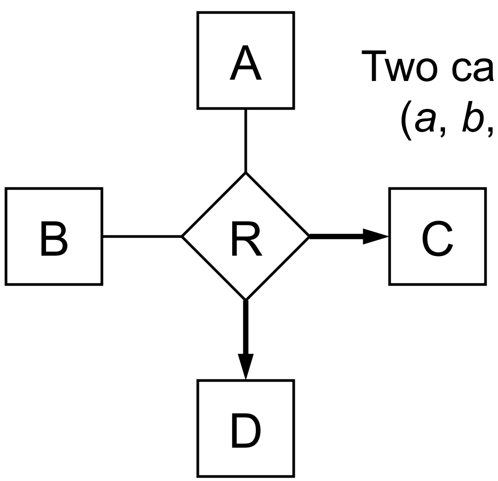


Primary key:
(a, b)

Meaning 1:
(a_1, b_2, c_3, d_5)
(a_3, b_4, c_3, d_2)
$\cancel{(a_3, b_4, c_6, d_9)}$
…

N-ary Mapping Cardinalities (3)

- Relationship-set R associating entity-sets $A_1, A_2, \ldots, A_n$
  - No arrows on edges $A_1, \ldots, A_i$; arrows on edges to $A_{i+1}, \ldots, A_n$
- Meaning 2 (the insane one):
  - For each entity-set A_k ($i < k \leq n$), a particular combination of entities from *all other* entity-sets can be associated with at most one entity in A_k
  - R has a candidate key for each arrow in N-ary relationship-set
  - For each k ($i < k \leq n$), another candidate key of R is union of primary keys from entity-sets $\{ A_1, A_2, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n \}$



Two candidate keys:
(a, b, c), (a, b, d)

Meaning 2:

(a_1, b_2, c_3, d_5)
(a_3, b_4, c_3, d_2)
(a_1, b_2, c_1, d_4)
(a_3, b_4, c_5, d_7)
~~(a_1, b_2, c_3, d_6)~~
~~(a_3, b_4, c_8, d_2)~~
…

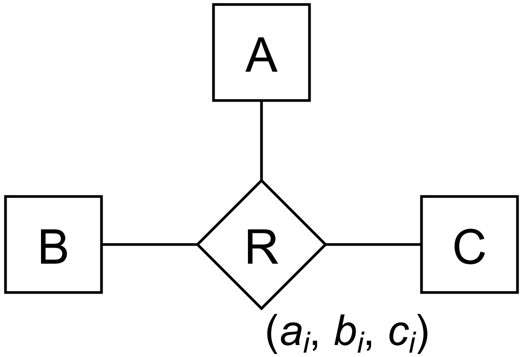
All disallowed by meaning 1!

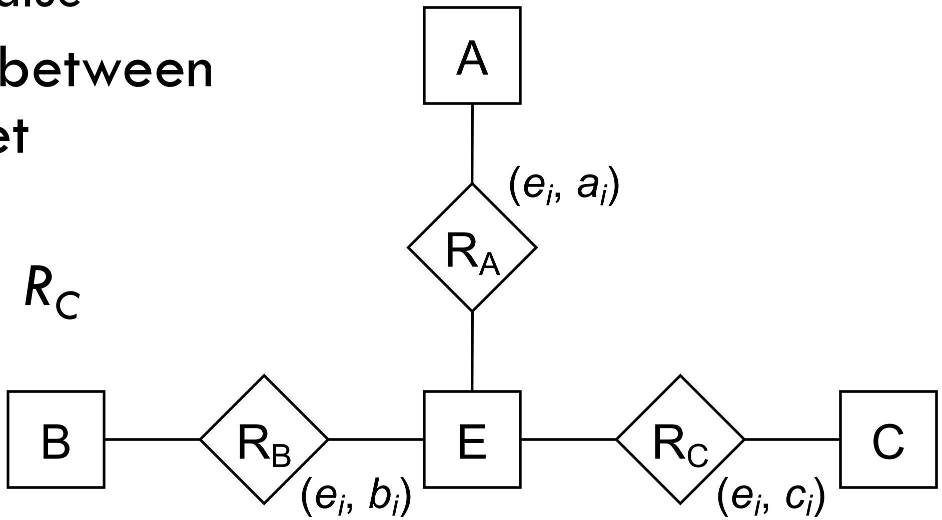
N-ary Mapping Cardinalities (4)

- Both interpretations of multiple arrows have been used in books and papers…
- If we only allow one edge to have an arrow, both definitions are equivalent
  - The ambiguity disappears

Binary vs. N-ary Relationships

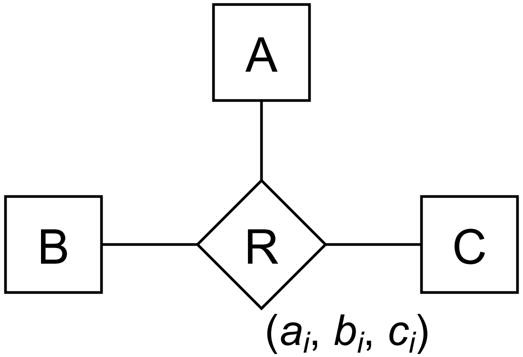
- Often have only binary relationships in DB schemas
- For degree > 2 relationships, *could* replace with binary relationships
  - Replace N-ary relationship-set with a new entity-set *E*
    - Create an identifying attribute for *E*
    - e.g. an auto-generated ID value
  - Create a relationship-set between *E* and each other entity-set
  - Relationships in *R* must be represented in R_A, R_B, and R_C



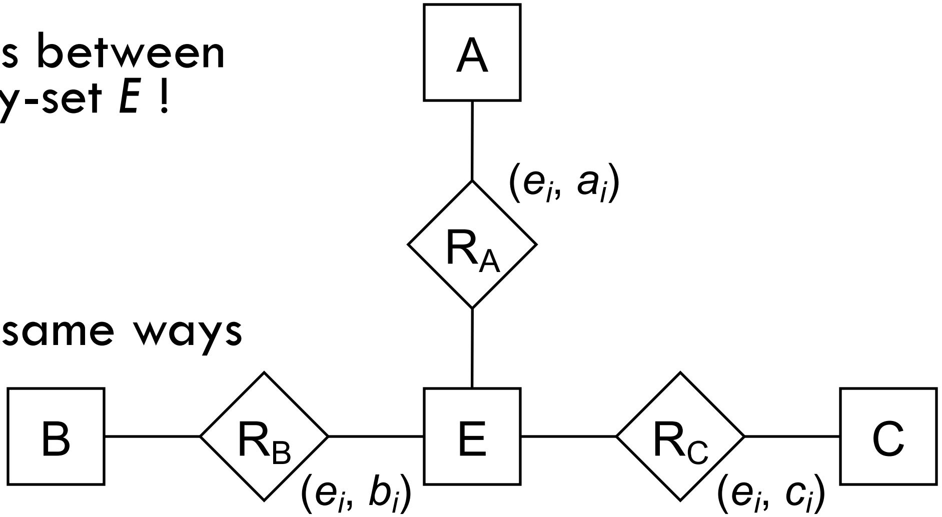


Binary vs. N-ary Relationships (2)

- **Are these representations identical?**
- Example: Want to represent a relationship between entities a_5, b_1 and c_2
  - How many relationships can we actually have between these three entities?
- Ternary relationship set:
  - Can only store one relationship between a_5, b_1 and c_2, due to primary key of *R*



- Alternate approach:
  - Can create <u>many</u> relationships between these entities, due to the entity-set *E* !
    - (a_5, e_1), (b_1, e_1), (c_2, e_1)
    - (a_5, e_2), (b_1, e_2), (c_2, e_2)
    - …
  - Can't constrain in exactly the same ways

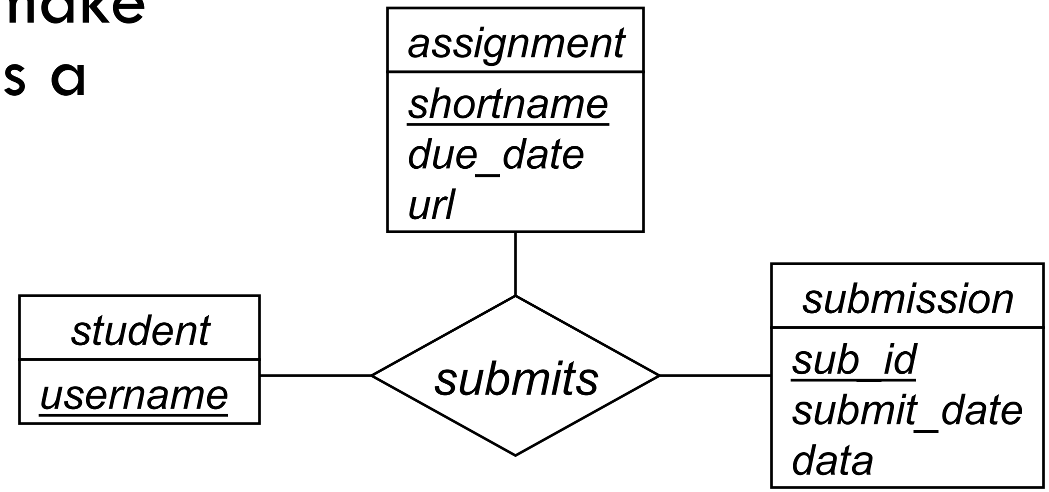


Binary vs. N-ary Relationships (3)

- Using binary relationships is sometimes more intuitive for particular designs
- Example: office-equipment inventory database
  - Ternary relationship-set *inventory*, associating *department*, *machine*, and *vendor* entity-sets
- What if vendor info is unknown for some machines?
  - For ternary relationship, must use *null* values to represent missing vendor details
  - With binary relationships, can simply not have a relationship between *machine* and *vendor*
- For cases like these, use binary relationships
  - If it makes sense to model as separate binary relationships, do it that way!

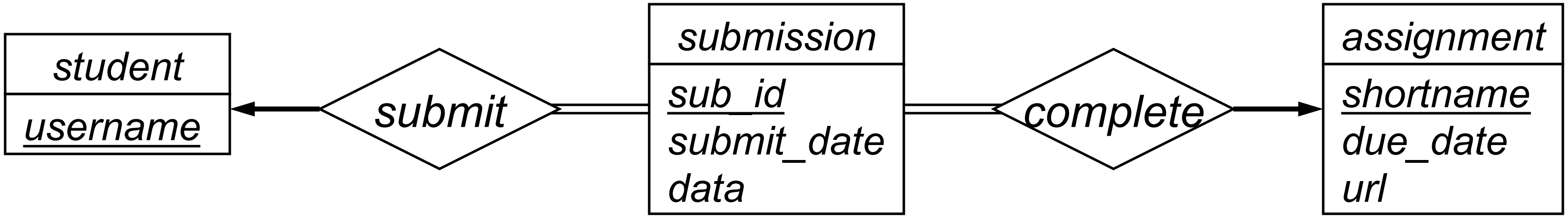
Course Database Example

- What about this case:
  - Ternary relationship between *student*, *assignment*, and *submission*
  - Need to allow multiple submissions for a particular assignment, from a particular student
- In this case, it could make sense to represent as a ternary relationship
  - Doesn't make sense to have only two of these three entities in a relationship



Course Database Example (2)

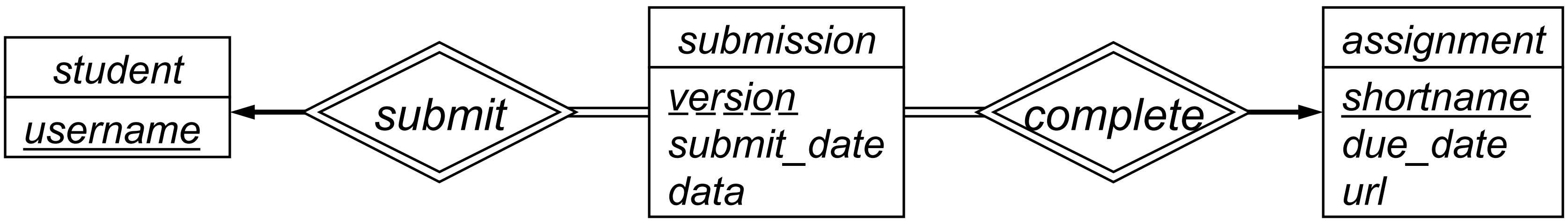
- Other ways to represent students, assignments and submissions?
- Can also represent as two binary relationships



- Note the total participation constraints!
  - Required to ensure that every *submission* has an associated *student*, and an associated *assignment*
  - Also, two one-to-many constraints

Course Database Example (3)

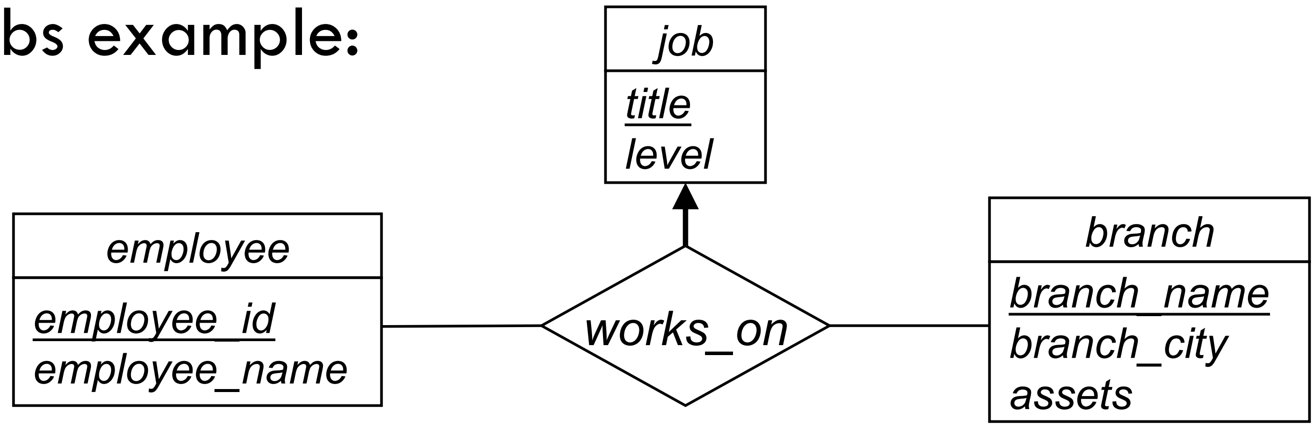
- Could even make *submission* a weak entity-set
  - Both *student* and *assignment* are identifying entities!



- Discriminator for *submission* is version number
- Primary key for *submission* ?
  - Union of primary keys from all owner entity-sets, plus discriminator
  - (*username*, *shortname*, *version*)

Binary vs. N-ary Relationships

- Sometimes ternary relationships are best
  - Clearly indicates all entities involved in relationship
  - Only way to represent certain constraints!
- Bank jobs example:



  - Each (*employee*, *branch*) pair can have only one job
  - Simply <u>cannot</u> construct the same constraint using only binary relationships
    - *(Reason is related to issue identified on slide 8)*

E-R Model and Real Databases

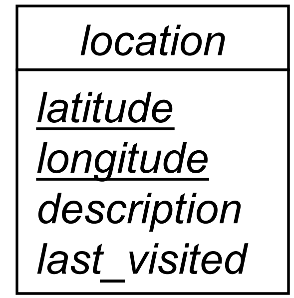
- For E-R model to be useful, need to be able to convert diagrams into an implementation schema
- Turns out to be very easy to do this!
  - Big overlaps between E-R model and relational model
  - Biggest difference is E-R composite/multivalued attributes, vs. relational model atomic attributes
- Three components of conversion process:
  - Specify schema of the relation itself
  - Specify primary key on the relation
  - Specify any foreign key references to other relations

Strong Entity-Sets

- Strong entity-set *E* with attributes $a_1, a_2, \ldots, a_n$
  - Assume simple, single-valued attributes for now
- Create a relation schema with same name *E*, and same attributes $a_1, a_2, \ldots, a_n$
- Primary key of relation schema is same as primary key of entity-set
  - Strong entity-sets require no foreign keys to other things
- Every entity in *E* is represented by a tuple in the corresponding relation

Entity-Set Examples

- Geocache location E-R diagram:
  - Entity-set named *location*

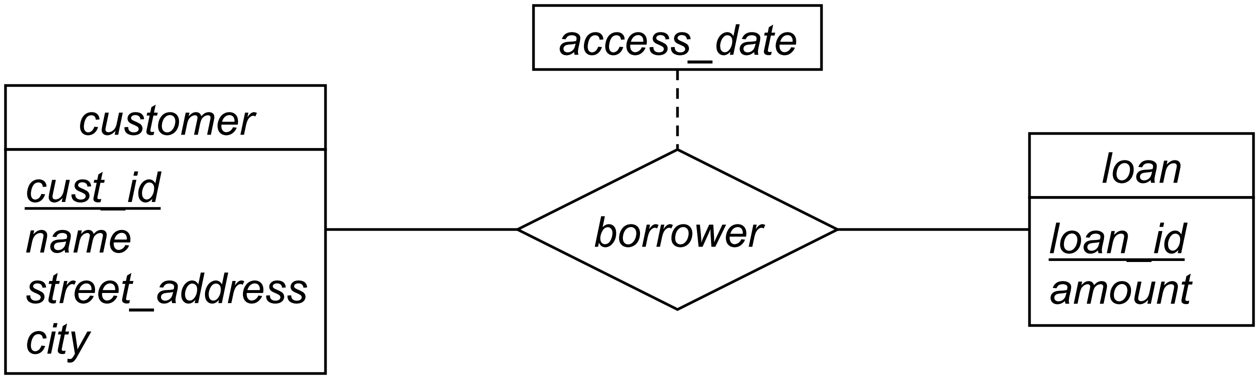


- Convert to relation schema:

location(*latitude*, *longitude*, *description*, *last_visited*)

Entity-Set Examples (2)

- E-R diagram for customers and loans:



- Convert *customer* and *loan* entity-sets:

  customer(*<u>cust_id</u>*, *name*, *street_address*, *city*)

  loan(*<u>loan_id</u>*, *amount*)

Relationship-Sets

- Relationship-set R
  - For now, assume that all participating entity-sets are strong entity-sets
  - $a_1, a_2, \ldots, a_m$ is the union of all participating entity-sets' primary key attributes
  - $b_1, b_2, \ldots, b_n$ are descriptive attributes on R (if any)
- Relational model schema for R is:
  - $\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$
- $\{a_1, a_2, \ldots, a_m\}$ is a superkey, but not necessarily a candidate key
  - Primary key of R depends on R's mapping cardinality

Relationship-Sets: Primary Keys

- For binary relationship-sets:
  - e.g. between strong entity-sets *A* and *B*
  - If many-to-many mapping:
    - Primary key of relationship-set is union of all entity-set primary keys
    - *primary_key*(*A*) ∪ *primary_key*(*B*)
  - If one-to-one mapping:
    - Either entity-set's primary key is acceptable
    - *primary_key*(*A*), or *primary_key*(*B*)
    - Enforce <u>both</u> candidate keys in DB schema!

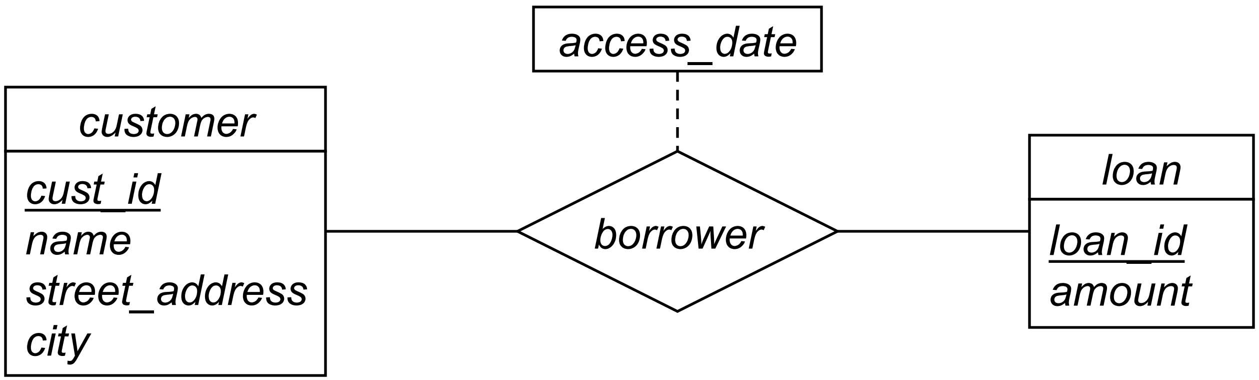
Relationship-Sets: Primary Keys (2)

- For many-to-one or one-to-many mappings:
  - e.g. between strong entity-sets *A* and B
  - Primary key of entity-set on "many" side is primary key of relationship
- Example: relationship *R* between *A* and *B*
  - One-to-many mapping, with *B* on "many" side
  - Schema contains *primary_key*(*A*) ∪ *primary_key*(*B*), plus any descriptive attributes on *R*
  - *primary_key*(*B*) is primary key of *R*
    - Each $a \in A$ can map to many $b \in B$
    - Each value for *primary_key*(*B*) can appear only once in *R*

Relationship-Set Foreign Keys

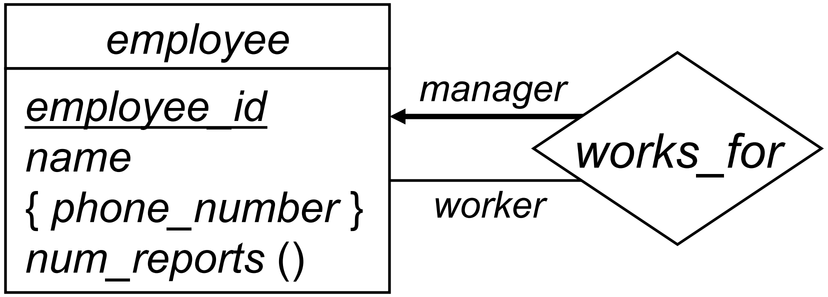
- Relationship-sets associate entities in entity-sets
  - We need foreign-key constraints on relation schema for *R* !
- For each entity-set E_i participating in *R* :
  - Relation schema for *R* has a foreign-key constraint on E_i relation, for *primary_key*(E_i) attributes
- Relation schema notation doesn't provide mechanism for indicating foreign key constraints
  - Don't forget about foreign keys and candidate keys!
    - Making notes on your relational model schema is a <u>very</u> good idea
  - Can specify both foreign key constraints and candidate keys in the SQL DDL

Relationship-Set Example



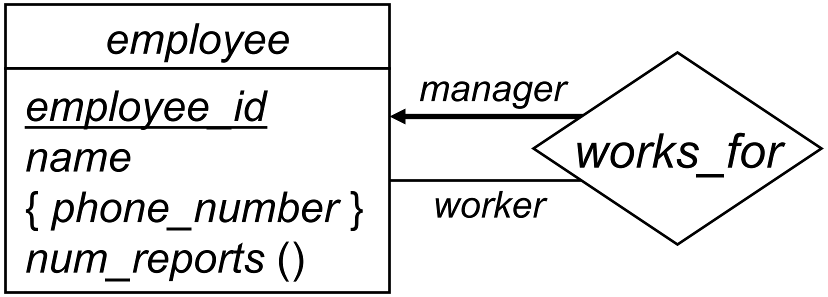
- Relation schema for *borrower*:
  - Primary key of *customer* is *cust_id*
  - Primary key of *loan* is *loan_id*
  - Descriptive attribute *access_date*
  - *borrower* mapping cardinality is many-to-many
  - Result: *borrower*(<u>*cust_id*</u>, <u>*loan_id*</u>, *access_date*)

Relationship-Set Example (2)



- In cases like this, must use roles to distinguish between the entities involved in the relationship-set
  - *employee* participates in *works_for* relationship-set twice
  - Can't create a schema (*employee_id*, *employee_id*) !
- Change names of key-attributes to distinguish roles
  - e.g. (*manager_employee_id*, *worker_employee_id*)
  - e.g. (*manager_id*, *employee_id*)

Relationship-Set Example (2)



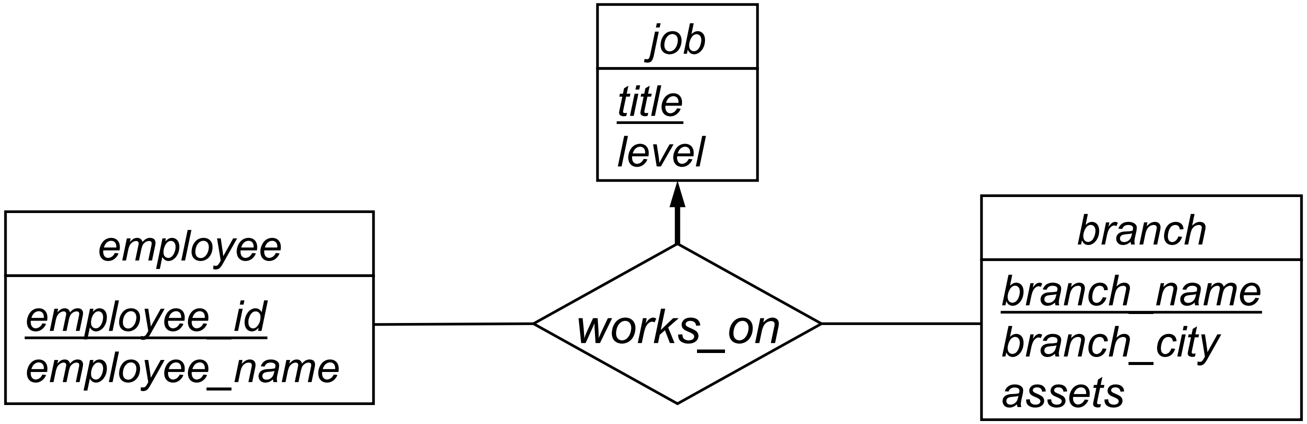
- Relation schema for *employee* entity-set:
  - (For now, ignore *phone_number* and *num_reports*…)

    employee(<u>*employee_id*</u>, *name*)
- Relation schema for *works_for*:
  - One-to-many mapping from *manager* to *worker*
  - "Many" side is used for primary key
  - Result: *works_for*(<u>*employee_id*</u>, *manager_id*)

N-ary Relationship Primary Keys

- For degree > 2 relationship-sets:
  - If no arrows ("many-to-many" mapping), relationship-set primary key is union of <u>all</u> participating entity-sets' primary keys
  - If one arrow ("one-to-many" mapping), relationship-set primary key is union of primary keys of entity-sets without an arrow
  - Don't allow more than one arrow for relationship-sets with degree > 2

N-ary Relationship-Set Example



- Entity-set schemas:

  job(*title*, *level*)

  employee(*employee_id*, *employee_name*)

  branch(*branch_name*, *branch_city*, *assets*)

- Relationship-set schema:
  - Primary key includes entity-sets on non-arrow links

    works_on(*employee_id*, *branch_name*, *title*)

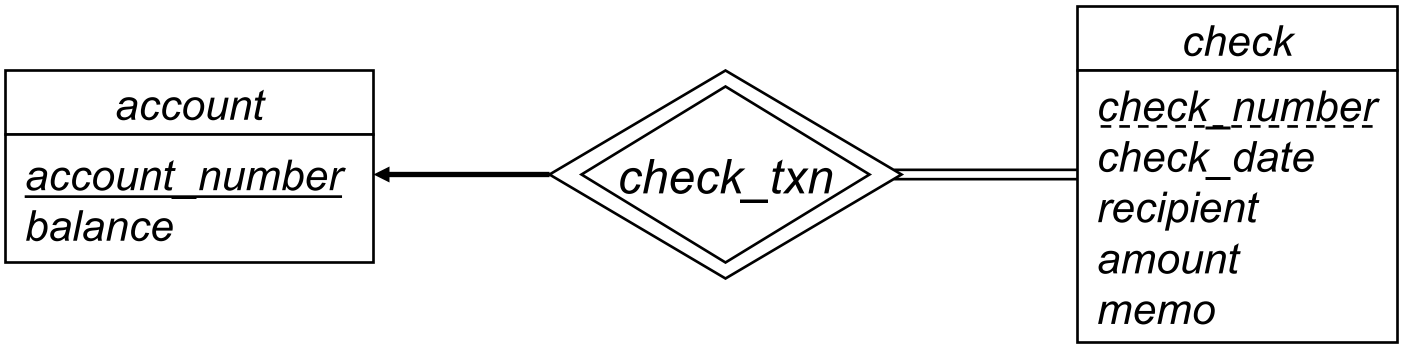
Weak Entity-Sets

- Weak entity-sets depend on at least one strong entity-set
  - The identifying entity-set, or owner entity-set
  - Relationship between the two is called the identifying relationship
- Weak entity-set *A* owned by strong entity-set *B*
  - Attributes of *A* are $\{a_1, a_2, \ldots, a_m\}$
    - Some subset of these attributes comprises the discriminator of *A*
  - $primary\_key(B) = \{b_1, b_2, \ldots, b_n\}$
  - Relation schema for *A*: $\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$
  - Primary key of *A* is $discriminator(A) \cup primary\_key(B)$
  - *A* has a foreign key constraint on $primary\_key(B)$, to *B*

Identifying Relationship?

- The identifying relationship is many-to-one, with no descriptive attributes
- Relation schema for weak entity-set already includes primary key for strong entity-set
  - Foreign key constraint is imposed, too
- No need to create relational model schema for the identifying relationship
  - Would be redundant to the weak entity-set's relational model schema!

Weak Entity-Set Example



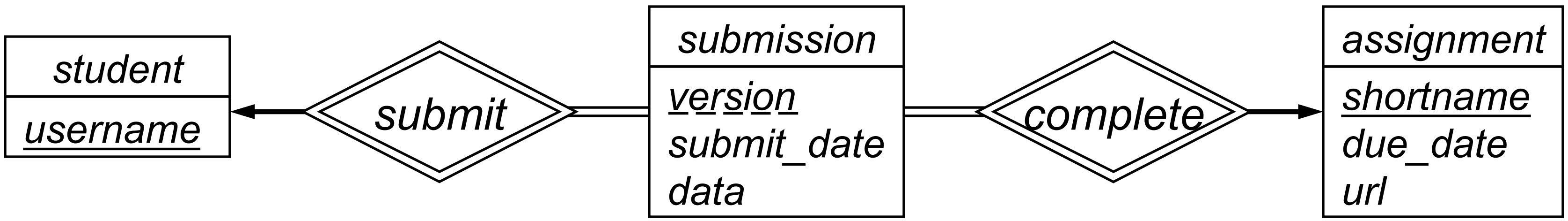
- *account* schema:

  account(<u>*account_number*</u>, *balance*)

- *check* schema:
  - Discriminator is *check_number*
  - Primary key for *check* is: (*account_number*, *check_number*)

  check(<u>*account_number*</u>, <u>*check_number*</u>, *check_date*, *recipient*, *amount*, *memo*)

Weak Entity-Set Example (2)



- Schemas for strong entity-sets:

  student(<u>*username*</u>)

  assignment(<u>*shortname*</u>, *due_date*, *url*)
- Schema for *submission* weak entity-set:
  - Discriminator is *version*
  - Both *student* and *assignment* are owners!

  submission(<u>*username*</u>, <u>*shortname*</u>, <u>*version*</u>, *submit_date*, *data*)
    - Two foreign keys in this relation as well

Composite Attributes

- Relational model simply doesn't handle composite attributes
  - All attribute domains are *atomic* in the relational model
- When mapping E-R composite attributes to relation schema: simply flatten the composite
  - Each component attribute maps to a separate attribute in relation schema
  - In relation schema, simply can't refer to the composite as a whole
  - (Can adjust this mapping for databases that support composite types)

Composite Attribute Example

- Customers with addresses:

customer
<u>cust_id</u>
name
address
street
city
state
zip_code

- Each component of *address* becomes a separate attribute

  customer(*<u>cust_id</u>*, *name*, *street*, *city*, *state*, *zip_code*)

Multivalued Attributes

- Multivalued attributes require a separate relation
  - Again, no such thing as a multivalued attribute in the relational model
  - E-R constraint on multivalued attributes: in a specific entity's multivalued attribute, each value may only appear once
- For a multivalued attribute *M* in entity-set *E*
  - Create a relation schema *R* to store *M*, with attribute(s) *A* corresponding to the single-valued version of *M*
  - Attributes of *R* are: *primary_key*(*E*) ∪ *A*
  - Primary key of *R* includes all attributes of *R*
    - Each value in *M* for an entity e must be unique
  - Foreign key from *R* to *E*, on *primary_key*(*E*) attributes

Multivalued Attribute Example

- Change our E-R diagram to allow customers to have multiple addresses:

customer

cust_id
name
{ *address*
    street
    city
    state
    zip_code }

- Now, must create a separate relation to store the addresses

  customer(*cust_id*, *name*)

  cust_addrs(*cust_id*, *street*, *city*, *state*, *zipcode*)

  - Large primary keys aren't ideal – tend to be costly