Module 2

Database Concepts

Connor Baker

Compiled on February 2, 2020 at 1:55pm

Purposes of DBMS

- Provide support for easy-to-use data
  - Data model (data)
  - Transaction model (operation)
- Provide efficient storage and access of the data in terms of the data model and transactional model

Data Models

- Tools to obtain data *abstraction*
- Necessary to be general and intuitive
- Data model
  - A class of mathematical structures, with description and operations
- Conceptual data model
  - Structural description

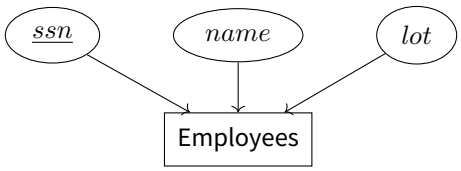
Overview of Database Design

- Conceptual design
  - Use ER model
  - Decide the entities and relationships in the enterprise
  - Decide what information about these entities and relationships we should store in the database
  - Decide the integrity constraints or business rules
- Implementation
  - Map an ER model into a relational schema

ER Model Basics

- Entity
  - A real-world object distinguishable from other objects
- Distinguishable via its description (data)

- Attribute
  - A mapping that maps an object to a value (called the *attribute value*). As an example an age is an attribute of student objects
- An entity is described using a set of attribute values
- Entity set
  - A collection of similar entities (like all employees), where similar is defined as having the same set of attributes



Legend:

- *Entity sets* are boxed
- *Attributes* are italicized (typically circled)
- *Primary keys* are underlined

Different Kinds of Keys

The textbook slides are terrible. All of the following is sourced directly from Wikipedia.

Superkey

A superkey is a set of attributes of a relation for which it holds that in all relations assigned to that variable, there are no two distinct tuples (rows) that have the same values for the attributes in this set. It can be defined as a set of attributes of a relation schema upon which all attributes of the schema are functionally dependent.

The set of **all** attributes is a trivial superkey, because in relational algebra duplicate rows are not permitted: rows are a set (no duplicates), not a multiset (duplicates allowed). The super-key is also known as a superset key.

A superkey is a set of attributes within a table whose values can be used to uniquely identify a tuple. A candidate key is a minimal set of attributes necessary to identify a tuple; this is also called a minimal superkey. Given an employee schema consisting of the attributes employeeID, name, job, and departmentID, where no value in the employeeID attribute is ever repeated, we could use the employeeID in combination with any or all other attributes of this table to uniquely identify a tuple in the table. Examples of superkeys in this schema would be {employeeID,

Name}, {employeeID, Name, job}, and {employeeID, Name, job, departmentID}. The last example is known as trivial superkey, because it uses all attributes of this table to identify the tuple.

In a real database we do not need values for all of those attributes to identify a tuple. We only need, per our example, the set {employeeID}. This is a **minimal superkey**—that is, a minimal set of attributes that can be used to identify a single tuple. employeeID is a candidate key.

Candidate Key

In the relational model of databases, a **candidate key** of a relation is a minimal superkey for that relation; that is, a set of attributes such that:

1. The relation does not have two distinct tuples (i.e. rows or records in common database language) with the same values for these attributes (which means that the set of attributes is a superkey)
2. There is no proper subset of these attributes for which (1) holds (which means that the set is minimal).

Candidate keys are also variously referred to as primary keys, secondary keys or alternate keys.

The constituent attributes are called **prime attributes**. Conversely, an attribute that does not occur in ANY candidate key is called a **non-prime attribute**.

Since a relation contains no duplicate tuples, the set of all its attributes is a superkey if NULL values are not used. It follows that every relation will have at least one candidate key.

The candidate keys of a relation tell us all the possible ways we can identify its tuples. As such they are an important concept for the design of database schema.

Primary Key

In the relational model of databases, a **primary key** is a *specific choice* of a *minimal* set of attributes (columns) that uniquely specify a tuple (row) in a relation (table). Informally, a primary key is “which attributes identify a record”, and in simple cases are simply a single attribute: a unique id. More formally, a primary key is a choice of a candidate key (a minimal superkey); any other candidate key is an **alternate key**.

A primary key may consist of real-world observables, in which case it is called a *natural key*, while an attribute created to function as a key and not used for identification outside the database is called a *surrogate key*. For example, for a database of people (of a given nationality), time and location of birth could be a natural key. National identification number is another example of an attribute that may be used as a natural key.

Alternate Key

All candidate keys which are not a primary key.

Natural Key

A **natural key** (also known as **business key**) is a type of unique key in a database formed of attributes that exist and are used in the external world outside the database (i.e. in the business domain or domain of discourse). In the relational model of data, a natural key is a candidate key and is therefore a functional determinant for all attributes in a relation. A natural key is sometimes called a *domain key*.

A natural key serves two complementary purposes: it provides a means of identification for data and it imposes a rule, specifically a *uniqueness constraint*, to ensure that data remains unique within an information system. The uniqueness constraint assures uniqueness of data within a certain technical context (e.g. a set of values in a table, file or relation variable) by rejecting input of any data that would otherwise violate the constraint. This means that the user can rely on a guaranteed correspondence between facts identified by key values recorded in a system and the external domain of discourse (a single version of the truth).

Examples of natural keys could include:

- Purchase Order number
- Flight number
- Login name
- Vehicle registration number
- Social Security number
- Passport number
- University ID number

The presence of a key guarantees uniqueness within an information system but it is not always necessary that the key values be unique or immutable within some wider population of objects or concepts *outside* that system. For example a key on a CITY attribute means that the set of city names assigned to that attribute must be unique at any point in time, so there can only be one city called “Washington” for example. That does not imply that every possible city which might one day be referred to within the system must have a unique name. In logical terms, the proposition being represented by the value “Washington” is that there is a city called Washington *within the domain of discourse* at a point in time, not that there is only *one* city of that name in every conceivable domain or for all time.

Similarly, the potential occurrence of erroneous or unwanted duplicate information does not necessarily rule out the use of an attribute as a natural key. For example in the US there may be instances of duplicate Social Security numbers mistakenly issued to individuals or other instances of a person fraudulently or mistakenly using another person’s SSN. In these situations the use of SSN as a natural key serves the purpose of a data integrity check - detecting potential duplication or fraud by rejecting any duplicate values with the implication that any error should be identified and resolved before entry into the system.

A natural key differs from a surrogate key which has no meaning outside the database itself and is not based on real-world observation or intended as a statement about the reality being modeled. A natural key therefore provides

a certain data quality guarantee whereas a surrogate does not. It is common for elements of data to have several keys, any number of which may be natural or surrogate.

Surrogate Key

A **surrogate key** (or **synthetic key**, **entity identifier**, **system-generated key**, **database sequence number**, **factless key**, **technical key**, or **arbitrary unique identifier**) in a database is a unique identifier for either an *entity* in the modeled world or an *object* in the database. The surrogate key is *not* derived from application data, unlike a natural (or business) key which is derived from application data.

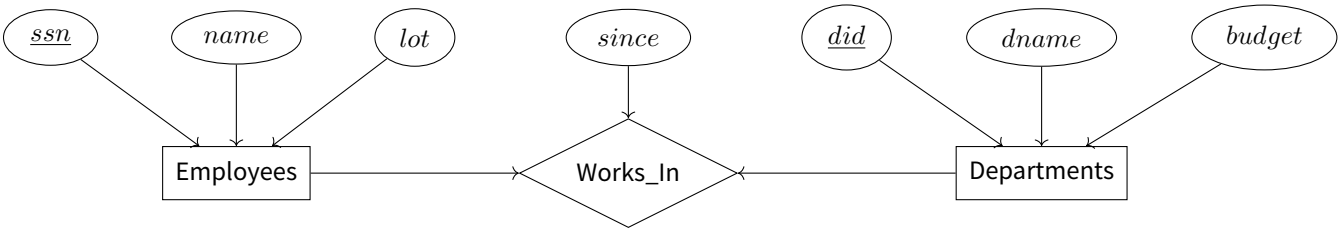
Keys of Entity Sets

- A *superkey* of an entity set is a (sub)set of the attributes such that no two entities in the set are allowed* to have the same values on all (key) attributes
  - * this is a design choice
- A *candidate key* is a superkey which does not have redundant attributes
  - Here, an attribute is called *redundant* if, when it is removed, the set is still a superkey
- A *primary key* is a specially designated candidate key
  - The rest of the candidate keys are called *alternate keys*
- Every entity set must have a key

ER Model Basics (continued)

- *Relationship*: An association among two or more entities
  - Gandalf *works in* the Pharmacy department
- *Relationship Set:* A collection of *similar* relationships
- *Similarity:* in terms of entity sets where the entities are from.
  - A person (from the `Employees` entity set) *works in* a department (from the `Departments` entity set)
- An n-ary relationship set R relates n entity sets $E_1, \dots, E_n$; each relationship in R involves entities $e_i \in E_i$
- The same entity set could participate in different relationship sets, or in different "roles" in the same set

Relationship Set Example



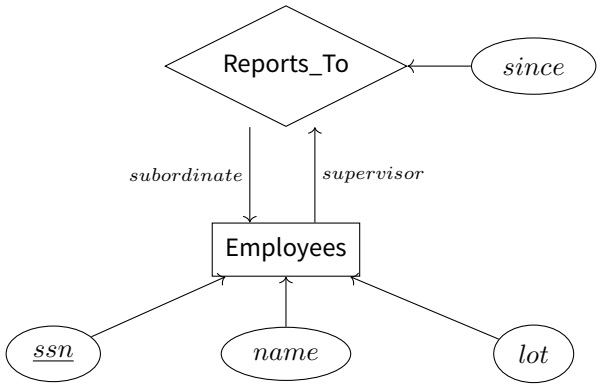
Legend:

- *Entity sets* are boxed
- *Attributes* are italicized and circled
- *Primary keys* are underlined
- *Relationship sets* are written inside a diamond

Descriptive Attributes

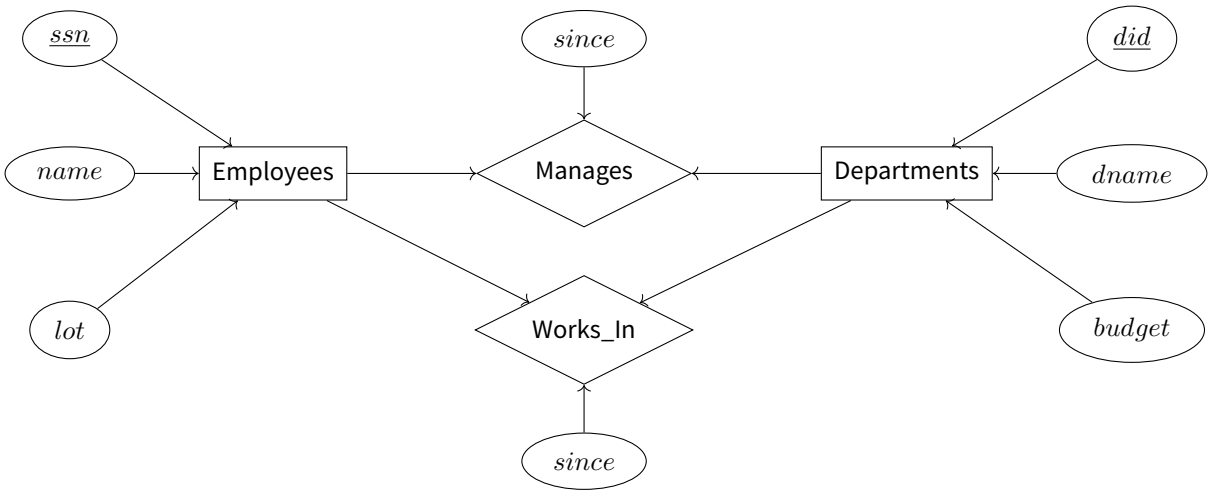
- Relationships can have attributes
  - These attributes are called *descriptive attributes* because they only describe the relationship and cannot be used to distinguish between relationships
- A relationship can only be distinguished by the participating entities
  - As such, there can't be more than one relationship involving the same entities

Another Relationship Set



Key Constraints

- Consider `Works_In`
  - an employee can work in many departments; a department can have many employees
  - In contrast, each department has at most one manager, according to the *key constraint* on `Manages`

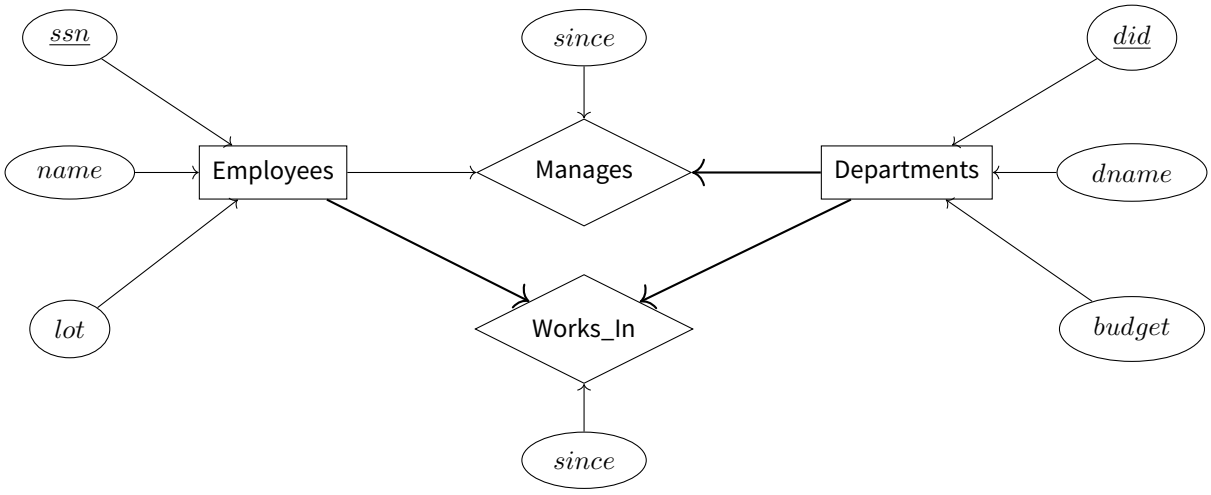


Key Constraint

- An entity set may participate in a relationship set as a "key" participant
- What it means is that each entity of the "key" entity set can only participate *at most once* in the relationship set
- More than one relationship set can be key participant (e.g., a one-to-one relationship set)

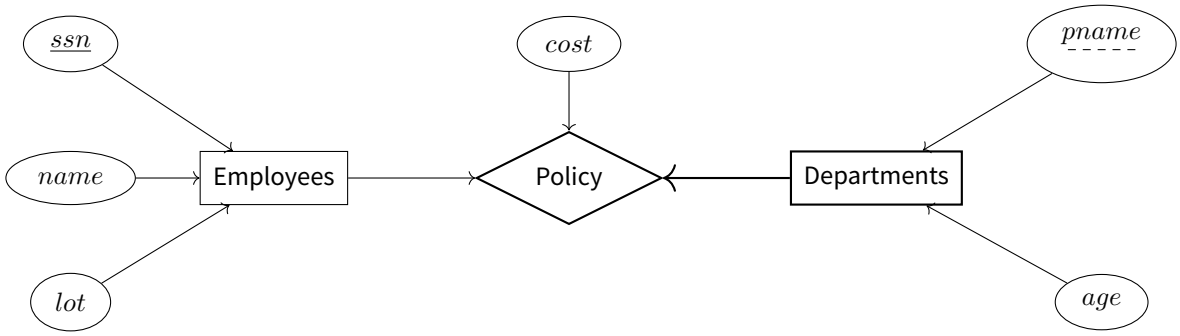
Participation Constraints

- Does every department have a manager?
  - If so, this is a *participation constraint*
    - The participation of `Departments` in `Manages` is said to be *total* (as opposed to *partial*)
  - Every `did` value in the Departments table must appear in a row of the Manages table (with a non-null `ssn` value)
- See the bolded components below



Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity
  - The owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities)
  - Weak entity set must have total participation in this *identifying* relationship set
- See the bolded components below

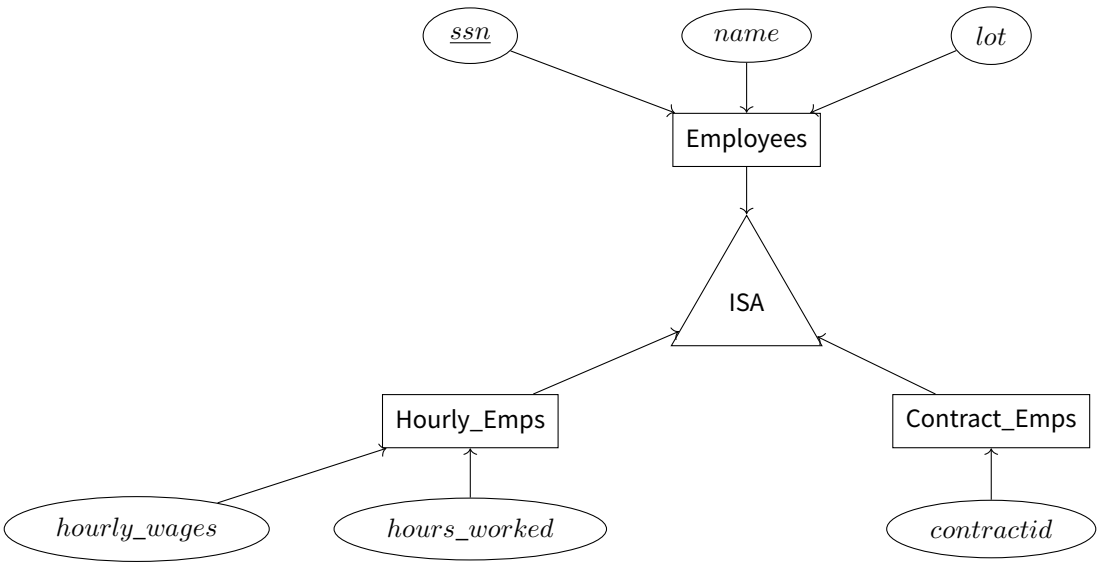


ISA (is-a) Hierarchies

- As in C++ or other programming languages, attributes are inherited
- If we declare `A isa B`, every `A` entity is also considered to be a `B` entity
  - *Overlap constraints*: Can Joe be an `Hourly_Emps` as well as a `Contract_Emps` entity? (Allowed/disallowed)

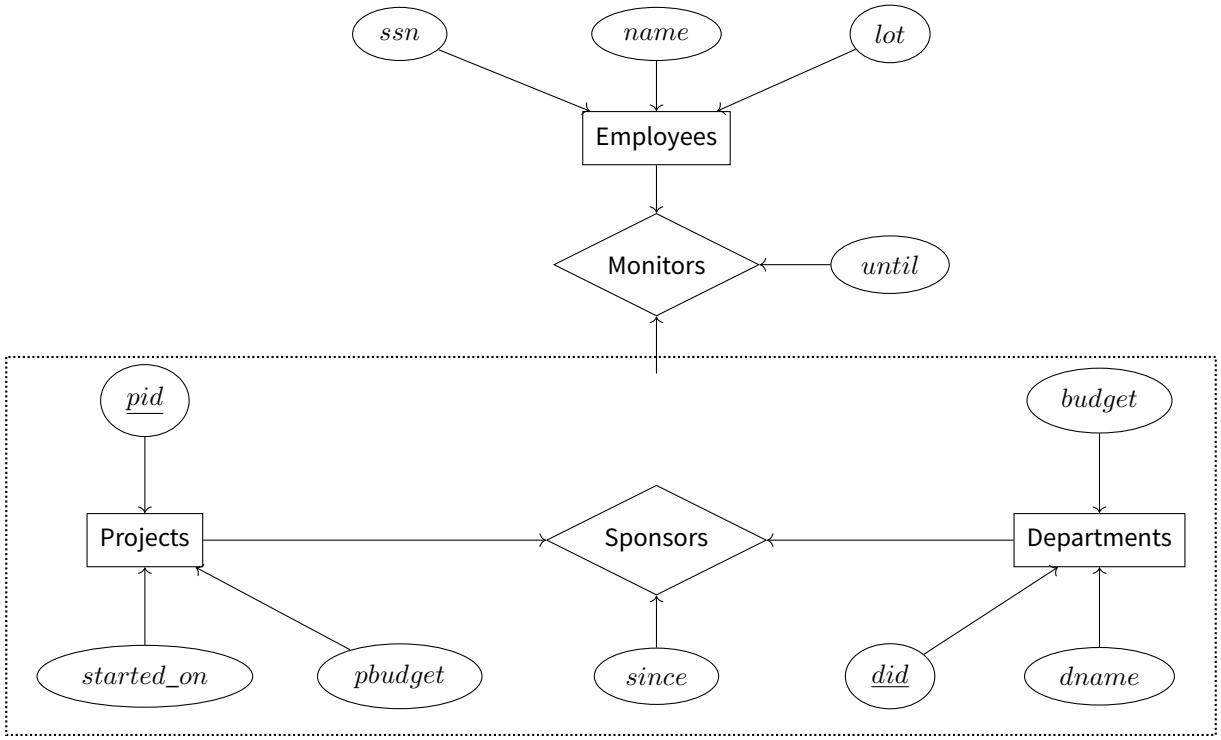
  - *Covering constraints:* Does every `Employees` entity also have to be an `Hourly_Emps` or a `Contract_Emps` entity? (Yes/no)

- Reasons for using ISA
  - To add descriptive attributes specific to a subclass
  - To identify entities that participate in a relationship



Aggregation

- Used when we have to model a relationship involving (entity sets and) a *relationship set*
- *Aggregation* allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships



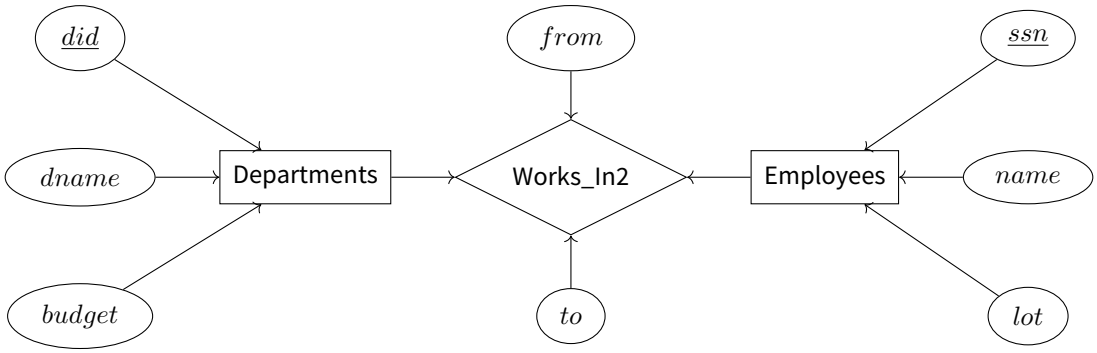
NB: On aggregations vs. the ternary relationship. Monitors is a distinct relationship, with a descriptive attribute. Furthermore, we can say that each sponsorship is monitored by at most one employee.

Conceptual Design Using the ER Model

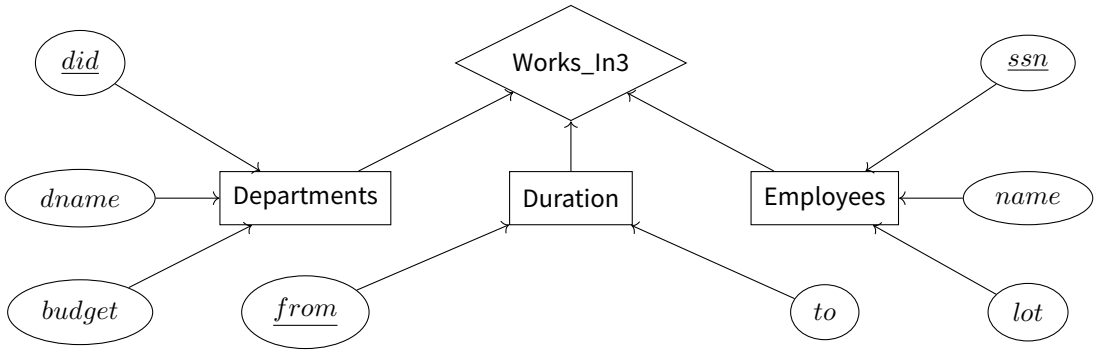
- Design choices
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - Identifying relationships
    - Binary or ternary?
    - Aggregation?
- Constraints in the ER model
  - A lot of data semantics can (and should) be captured
  - But some constraints cannot be captured in ER diagrams

Entity vs. Attribute

- Should `address` be an attribute of `Employees` or an entity (connected to `Employees` by a relationship?)
- Depends on what we want to do with address information and the semantics of the data
  - If we have several addresses per employee, `address` must be an entity (since attributes cannot be set-valued)
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, `address` must be modeled as an entity (since attribute values are atomic)
- `Works_In2` does not allow an employee to work in a department for two or more periods
- Similar to the problem of wanting to record several addresses for an employee
  - We want to record several values of the descriptive attributes for each instance of this relationship



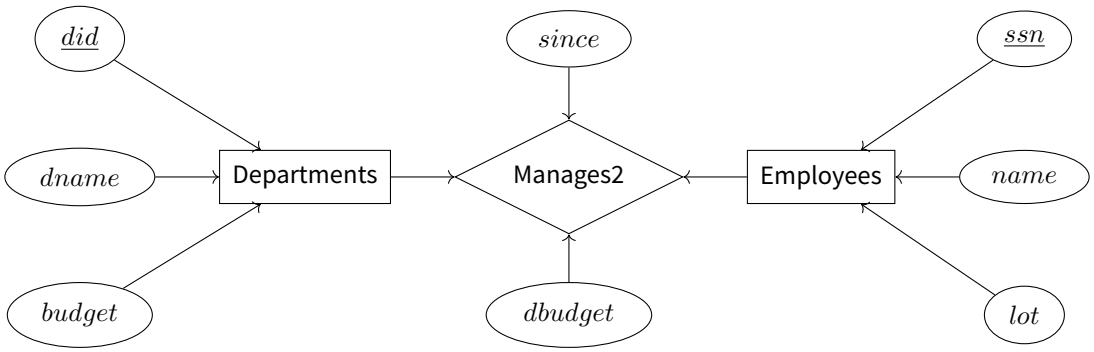
vs.



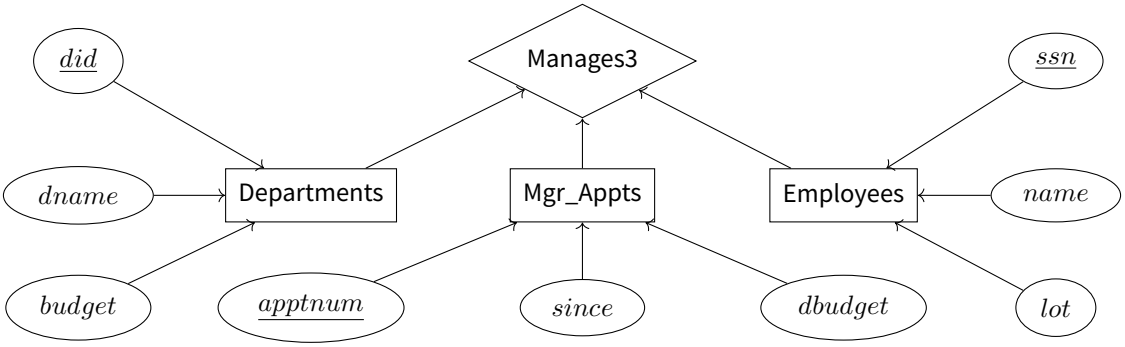
Entity vs. Relationship

- First ER diagram is okay if a manager gets a separate discretionary budget for each department

- What if a manager gets a discretionary budget that covers all managed departments?
  - *Redundancy* of `dbudget` which is stored for each department managed by the manager
  - *Misleading*: suggests that `dbudget` is tied to a managed department



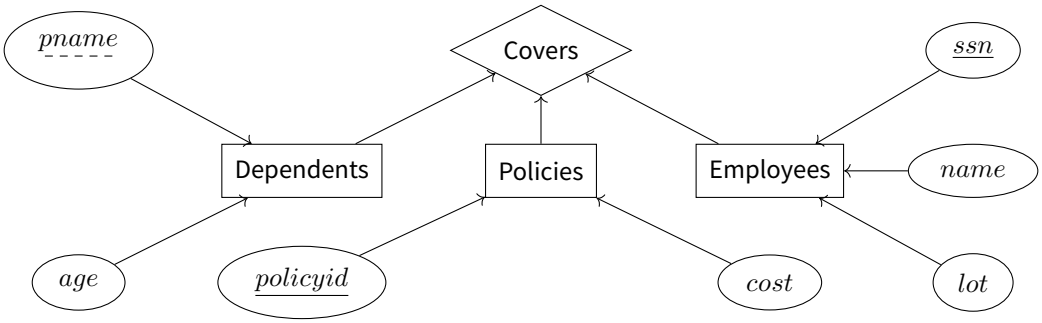
vs.



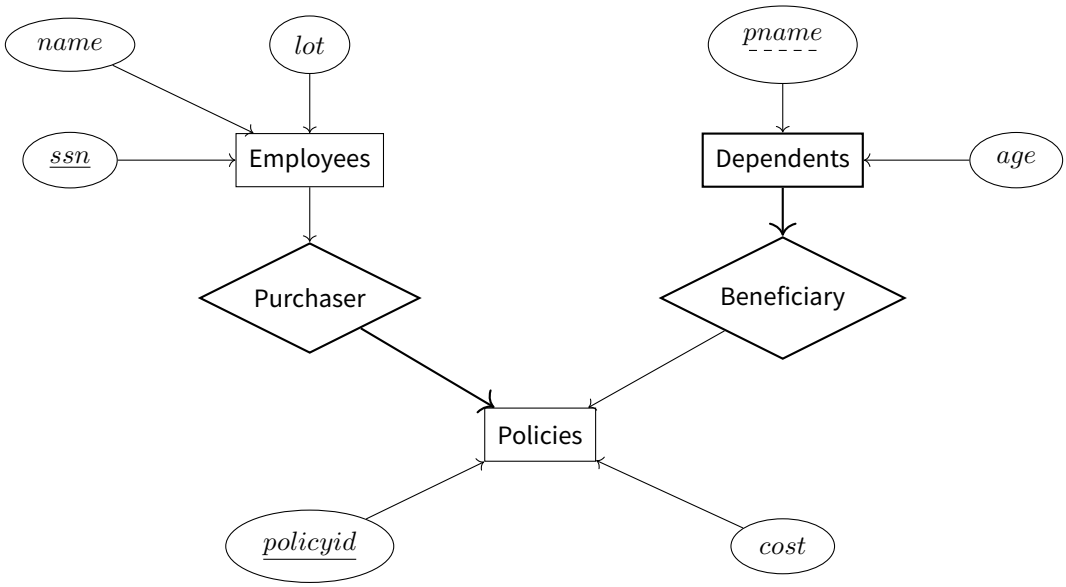
Binary vs. Ternary Relationships

- If each policy is owned by just one employee
  - Key constraints on `Policies` would mean that policy can only cover one dependent
- What are the additional constraints in the second diagram?

Bad Design



Better Design



Binary vs. Ternary Relationships (cont.)

- Previous example illustrated a case when two binary relationships were better than one ternary relationship
- An example in the other direction: a ternary relation `Contracts` relates entity sets `Parts`, `Departments`, and `Suppliers`, and has descriptive attribute `qty`. No combination of binary relationships is an adequate substitute
  - `Suppliers` "can supply" `Parts`, `Departments` "need" `Parts`, and `Departments` "deals with" `Suppliers` does not imply that `Departments` has agreed to buy `Parts` from `Suppliers`
  - How do we record `qty`?

Summary of Conceptual Design

- *Conceptual design* follows *requirements analysis*
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
- Basic constructs
  - *Entities*
  - *Relationships*
  - *Attributes* (of both entities and relationships)
- Note: there are many variations on the ER model
- Several kinds of integrity constraints can be expressed in the ER model
  - *Key constraints*
  - *Participation constraints*
  - *Overlap/covering constraints* for ISA hierarchies
  - *Foreign key constraints* are implicit in the definition of a relationship set
    - Some constraints (notably, *functional dependencies*) cannot be expressed in the ER model
    - Constraints play an important role in determining the best database design for an enterprise
- ER design is *subjective*
  - There are typically many ways to model a given scenario
  - Analyzing alternatives can be tricky, especially for large enterprise
  - Common choices include
    - Entity vs. attribute
    - Entity vs. relationship
    - Binary or n-ary relationship
    - Whether or not to use ISA hierarchies
    - Whether or not to use aggregation
- Ensuring good database design
  - Resulting relational schema should be analyzed and further refined
  - Functional Dependency information and normalization techniques are especially useful