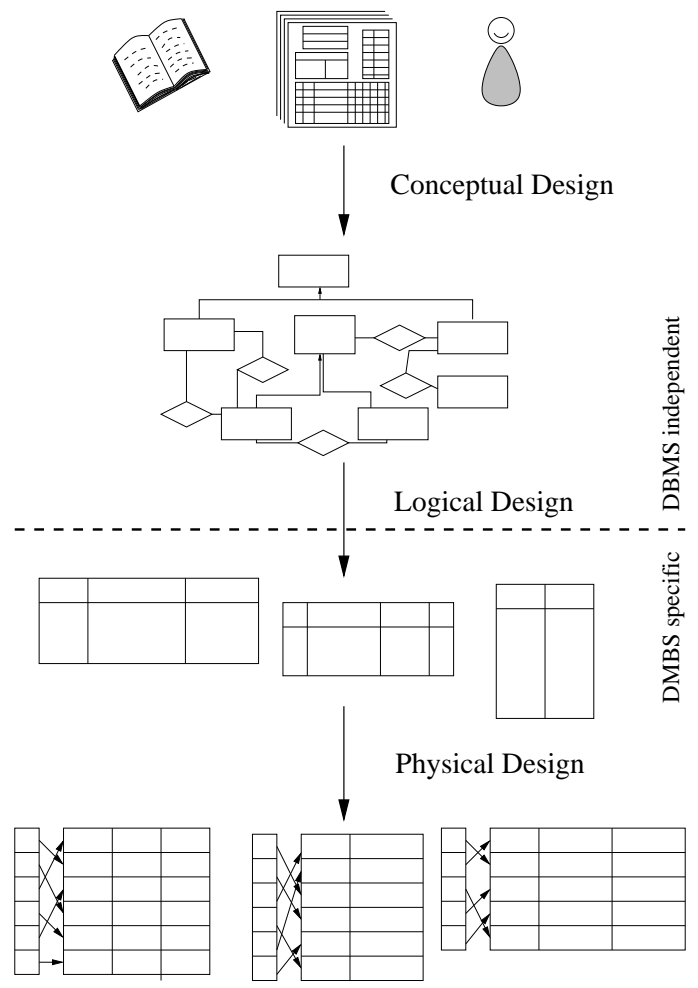


Database Design Process

- there are six stages in the design of a database:
 1. requirement analysis
 2. conceptual database design
 3. choice of the DBMS
 4. data model mapping
 5. physical design
 6. implementation
- not necessarily strictly sequential
 - *feedback loops* exist, i.e. may need to revisit earlier stages during a later stage



1. Requirement Collection and Analysis

- Purpose: to document the data requirements of the users
 - *functional requirements* are the operations that will be applied to the database, including queries and update
- the specification will then be used as the basis for the design of the database
- typical activities:
 - identification of application areas and user groups
 - analysis of existing documentation of application areas, e.g. policy documents, forms, reports, organisation charts
 - analysis of current operating environments and the planned use of the information, e.g. information flow, types of transactions, frequency of transaction types
 - responses to user questionnaires are analysed

... in summary:

- start from a description of the requirements which is:
 - poorly structured,
 - heterogeneous
 - informal
- and use a technique to transform that into a specification of the database requirements which is:
 - formal
 - homogeneous
 - consistent
 - complete

2. Conceptual Design

- two parallel activities

1. **schema design**

- examines the data requirements of the database resulting from the analysis (phase 1) and produces a conceptual schema in a **DBMS-independent** high level data model

2. **transaction design**

- examines the database applications whose requirements were analysed in phase 1 and produces high level specifications for these transactions

2.1. Conceptual Schema Design

- Purpose: to produce a *conceptual schema* of the database
 - expressed using concepts of the high level data model
 - * not including implementational details (has to be understood by non-technical users)
 - * but detailed in terms of the “objects” of the domain the database will represent
- independent of the DBMS to be used (no relational DB-oriented notions!)
- cannot be used **directly** to implement the database
- design is made in terms of a *semantic* or *conceptual* data model

- the goal is to achieve understanding of database structure, semantics, interrelationships and constraints
- need to be expressed in a “language” which offers:
 - *expressiveness*: able to distinguish between different types of data, relationships and constraints
 - *simplicity*: easy for non-specialist users to understand and use concepts
 - *minimality*: small number of basic concepts that are distinct and do not overlap
 - *diagrammatic representation*: for ease of presentation; it should therefore be easy to interpret
 - *formality*: must represent a formal, unambiguous specification of the data
- some of these requirements sometimes conflict
- most popular data model used is the Entity-Relationship (ER) model

2.2. Transaction Design

- Purpose: to produce a design of the *transactions*, that will run on the database
 1. *retrieval*: retrieve data for display or as part of a report
 2. *update*: enter new data or amend existing data
 3. *mixed*: more complex applications may do both retrieval and update
- Why?
 - need to be sure to include in the conceptual schema all information required by transactions
 - relative importance and frequency of use of transactions will influence physical database design
- ... the software needs to be designed as well as the data!

3. Choosing a DBMS

- Purpose: establish which is the best framework for implementing the produced schema:
 - type of DBMS (relational, network, deductive, ObjectOriented, ...)
 - user and programmer interfaces
 - types of query languages
- choice made on the basis of technical factors
 - the DBMS has to support the required tasks
- of economic factors
 - software acquisition/maintenance, hardware acquisition, creation/conversion, training of staff
- and of organisational factors:
 - platforms supported, availability of vendor services

4. Logical Design

- Purpose: to transform the generic, DBMS independent conceptual schema in the data model of the chosen DBMS (*data model mapping*)
- two stages:
 1. *system independent mapping*: no consideration of any specific characteristics that may apply to the specific DBMS package
 2. *tailoring to DBMS*: different DBMSs may implement the same data model in slightly different ways
- result is a set of DDL statements in the language of the chosen DBMS
 - some CASE tools generate DDL statements from a conceptual design

5. Physical Design

- Purpose: to choose the specific storage structures and access paths for the database files
 - attention to performances
- some relevant criteria:
 - *response time*: may want to minimise database access time for data items referenced by frequently used transactions
 - *space utilisation*: less frequently used data and queries may be archived
 - *transaction throughput*: average number of transactions that can be processed per minute

6. Implementation

- Purpose: to create the database
- compile and execute DDL statements
- populate the database
 - manually/automatically (may need to convert data from a previous format)
- implement application programs (*transactions*)
 - programs are written with embedded DML statements
- operational phase may begin

Entity-Relationship Model

- model to express the conceptual schema of the database
- originally proposed in 1976 by Peter Chen on the *ACM Transactions on Database Systems* journal
 - proposed as a means to unify the network and relational DB models
- many theoretical extensions and practical applications
 - Enhanced Entity Relationship (EER) Model
- used routinely for system analysis and design
 - simple enough to learn and understand the basic concepts
 - powerful enough to be used in the development of complex applications
- conceptual designs using the ER model are called *ER schemas*

ER Model

- The ER model describes data in terms of three primitive notions:
 1. *entities*
 2. *attributes*
 3. *relationships*
- an entity is a “thing”, which can be distinctly identified
 - e.g. a physical thing: a person, a car, a wire
 - e.g. an abstract thing: a university course, an event, a job
- an attribute is a property of an entity
 - e.g. a person has an age, a car has a colour
- a relationship is an association among entities
 - e.g. “*a person owns a car*” is an association between the entities *person* and *car*

Entities

- Entities are the “objects” the database has to store information about
- need to distinguish between
 - entities the database contains information about *currently*
 - world of possible entities the database might contain information about
- the conceptual schema has to capture the changing nature of data
 - need to make decisions based on the world of possible entities, e.g. the *entity class* or *type*
 - the entity class is an abstract description of some set of objects
 - data to be actually stored form *instances* of such abstract description

Attributes

- all instances of an entity class share some common properties named *attributes* of the entity class
 - e.g. attributes of the “employee” class might include name, age, address, salary etc.
- the ER model explicitly classifies attributes according to three criteria:
 1. complexity
 2. cardinality
 3. primitiveness

- *composite* vs *simple (atomic)*

- composite attributes have an overall significance (e.g. an address) but can be subdivided into more basic attributes with independent meaning (city, postal code etc.)
- simple attributes are indivisible (e.g. age)

- *single-valued* vs *multi-valued*

- most attributes can have only one single value for a particular instance (e.g. a person can only have one date of birth)
- some attributes can have one or more values for the same instance (e.g. a car model's colours, a person's names)

- *primitive* vs *derived*

- some attributes can be derived from other attributes of the same entity, e.g. age (derived) from birth date (primitive)
- or can be derived from properties of other entities (e.g. number of lecturers of a department)

Key Attributes

- an important feature of an entity type is the *key* or *uniqueness constraint* on attributes
- an entity type might have an attribute whose values are distinct for each individual entry
- such attribute is called *key attribute* and its value can be used to identify each entity uniquely
- sometimes, several attributes together can form a key, meaning that the combination of them must be distinct for each individual entity
- some entity types have more than one key attribute, for example both National Insurance Number and Staff Number are valid keys for the entity type “lecturer”

Relationships

- a *relationship type* defines an association among entity types
- a relationship has a *degree* that is the number of participating entity types, for example:
 - *binary* relationships (degree two): e.g. a person **owns** a car
 - *ternary* relationships (degree three): e.g. a lecturer **teaches** a course to a student
- relationship types can also have *attributes* (e.g. StartDate attribute on a **supervises** relationship)

- participating entities might have a *role name* in the relationship
 - usually the entity type name (e.g. in a ternary **teaches** relationships, roles are *lecturer*, *course* and *student*)
 - may be needed when entities are related by more than relationship (e.g. in an additional relationship **supervises**, a lecturer has role *supervisor* of a student having role *supervisee*)
 - needed also in recursive relationships (e.g. a student may have role *demonstrator* in the relationship **demonstrate for** with other students)

Structural Constraints on Relationships

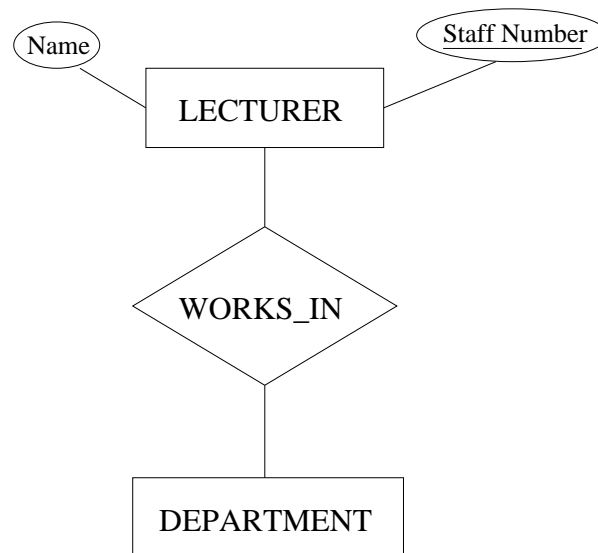
- *structural constraints* limit the possible combinations of entities that can participate in a relationship instance:
 - *cardinality ratio* specifies the number of relationship instances that an entity can participate in (one-to-one, one-to-many, many-to-many)
 - *participation constraint* specifies whether the existence of an entity depends on it being in the relationship:
 - * a *total participation* constraint, or *existence dependency*, specifies that an entity can only exist if it participates in the specified relationship (e.g. every lecturer must work in a specified department)
 - * *partial participation* constraint specifies there may exist an entity which does not participate in the relationship (e.g. not all lecturers supervise students)

Weak Entity Types

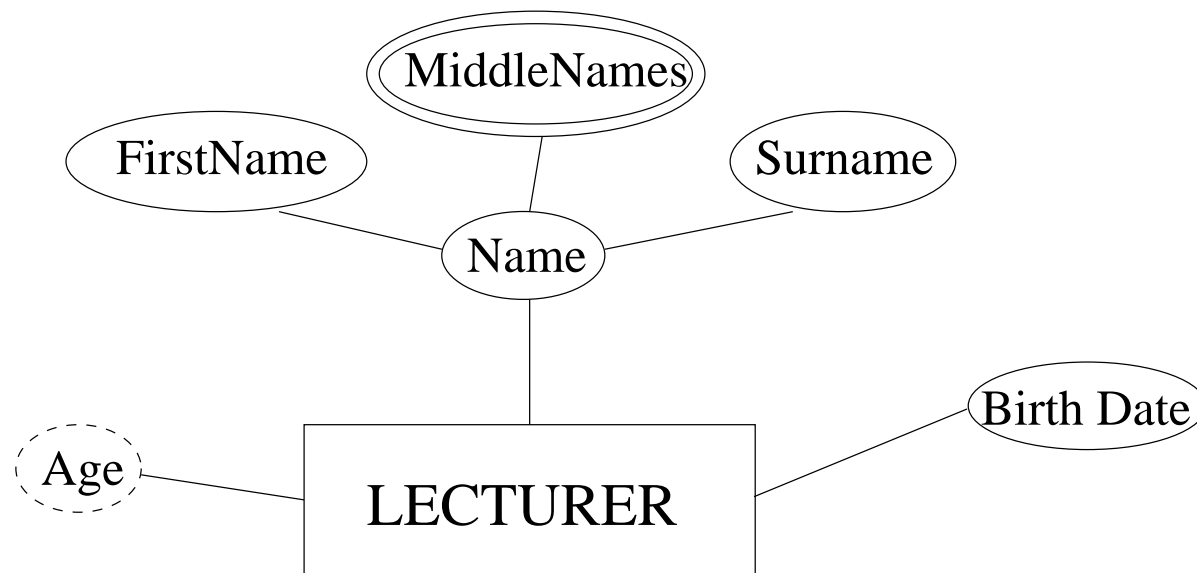
- these are entity types which cannot exist in isolation
- instances are identified because they “belong” to specific entities from another entity type, known as *identifying owner*
 - for instance, the content of a lecture theatre (white boards, desks, etc.) cannot typically be identified as such
 - the lecture theatre is their identifying owner, so we can talk about “the desk which **is in** RB8”
- the relationship type that relates the weak entity to its owner is the weak entity’s *identifying relationship*
 - in the example above, the **is in** relationship
- weak entity types might have a *partial key*, to distinguish one weak entity from other weak entities *related to the same owner*
 - for example “the desk 1 (or 2, 3 etc.) which **is in** RB8”

ER Diagrams

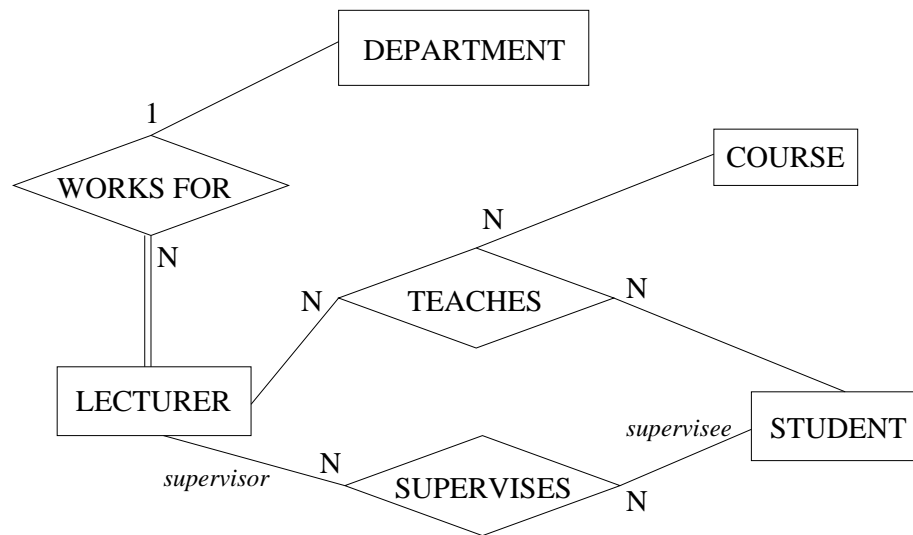
- entity types are represented as boxes
- relationship types are represented as diamonds connected with each participating entity types
- attributes are shown as ovals connected to the relevant entity or relation type (key attributes are underlined)



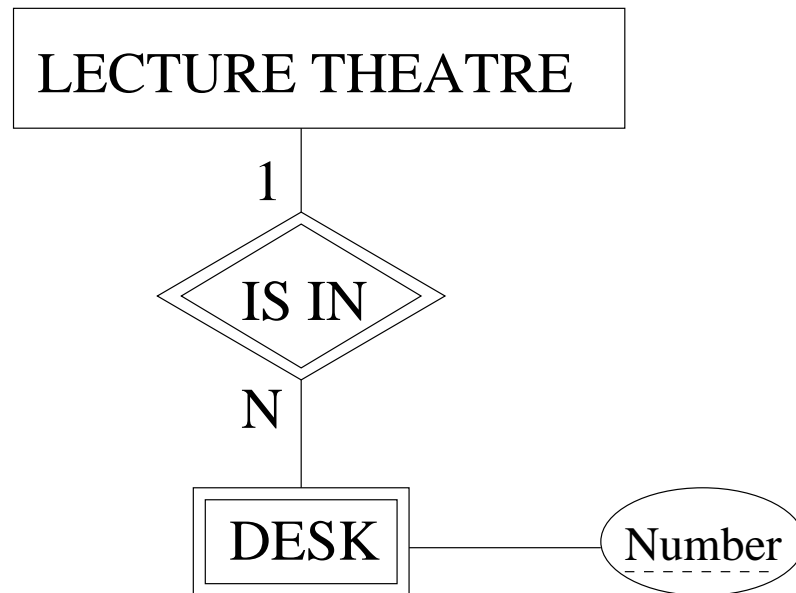
- component attributes are connected to the composite attribute
- multivalued attributes are indicated by double ovals
- derived attributes are indicated by dashed lines



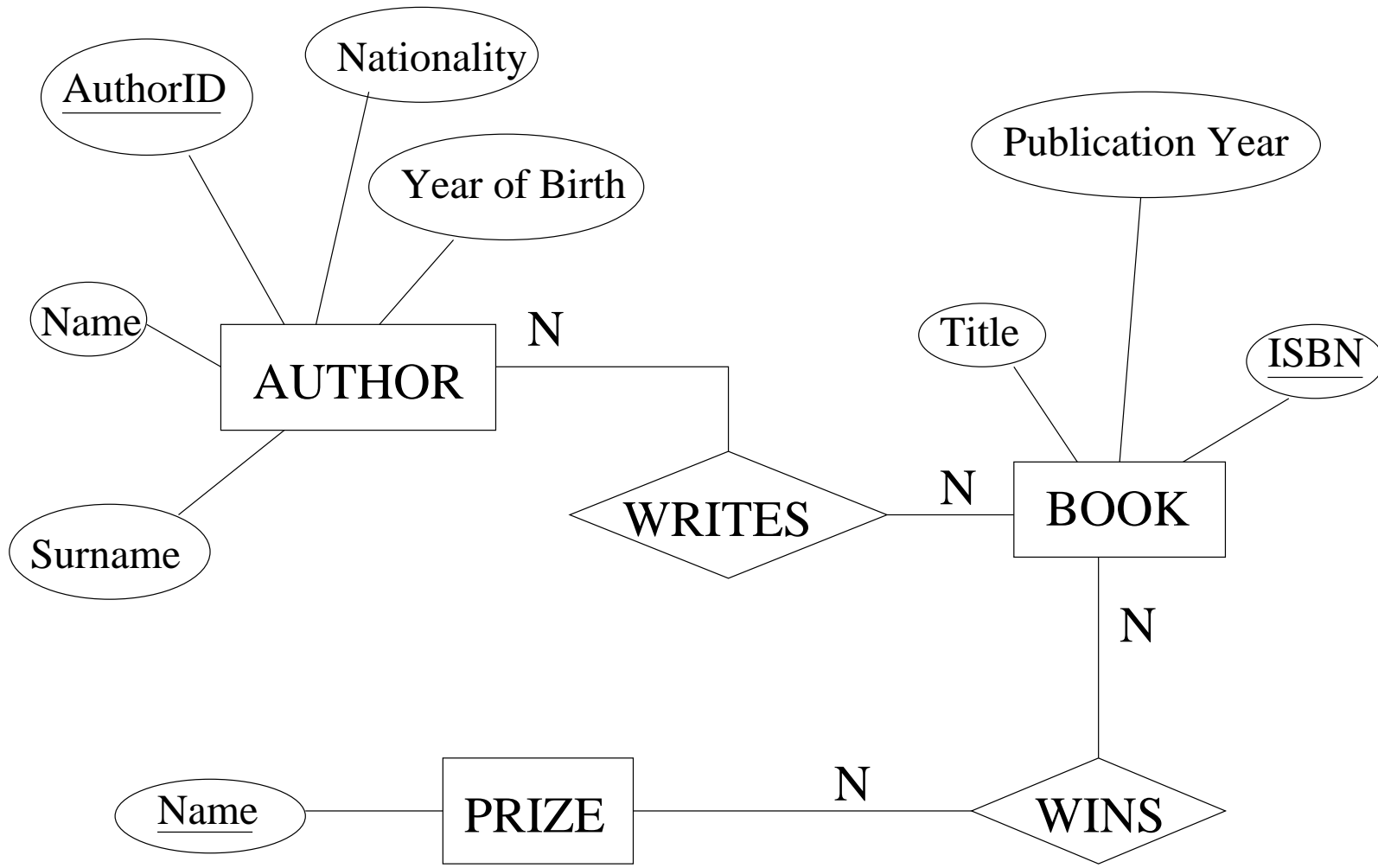
- the *cardinality* of the relationship is written by the line
- total participation of an entity E in a relationship R is indicated by a double line between E and R
- role names are attached to relationship connectors



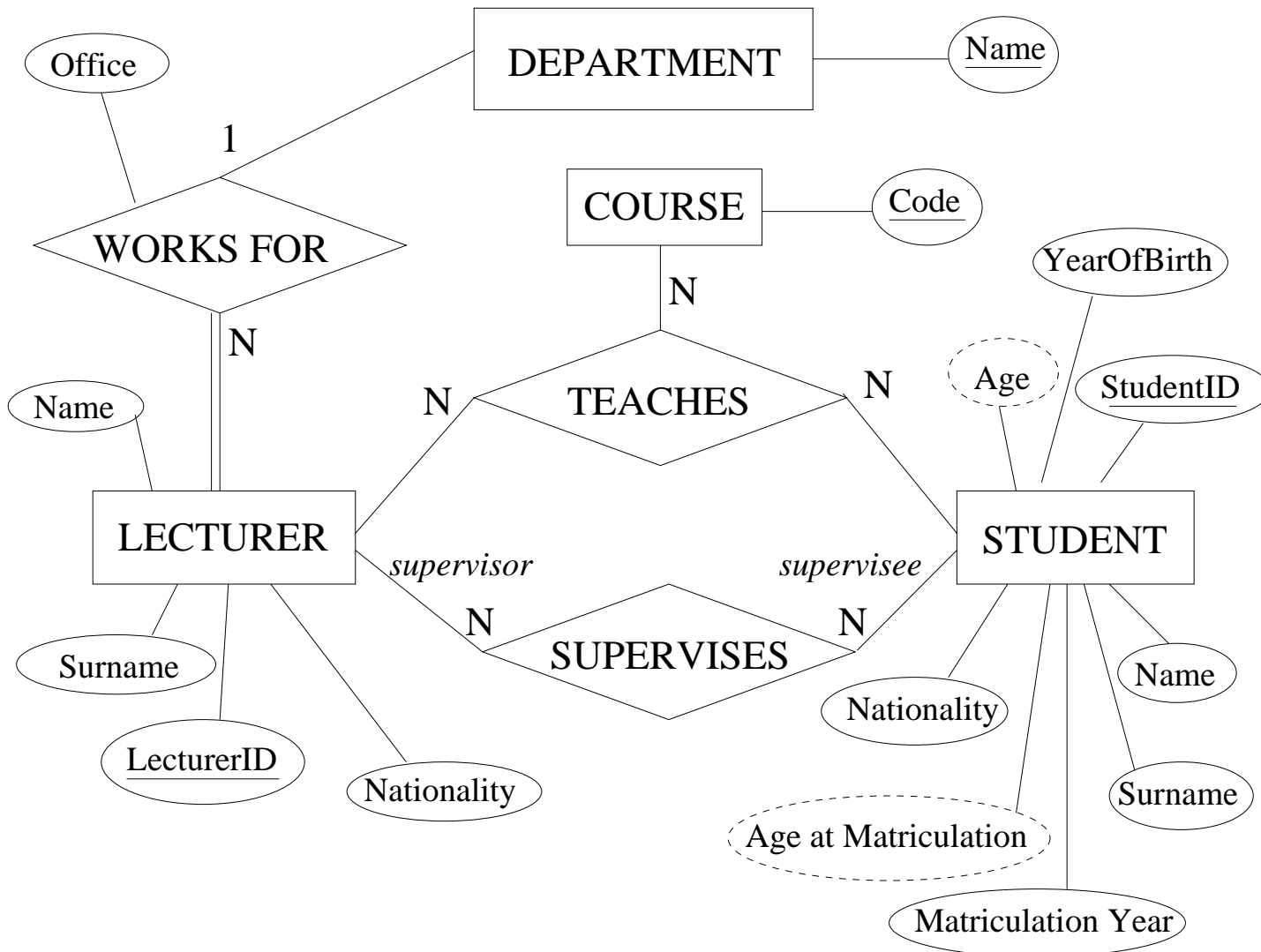
- weak entities are indicated by double boxed rectangles
- identifying relationship types are indicated by double boxed diamonds
- partial keys are indicated with a dashed underline



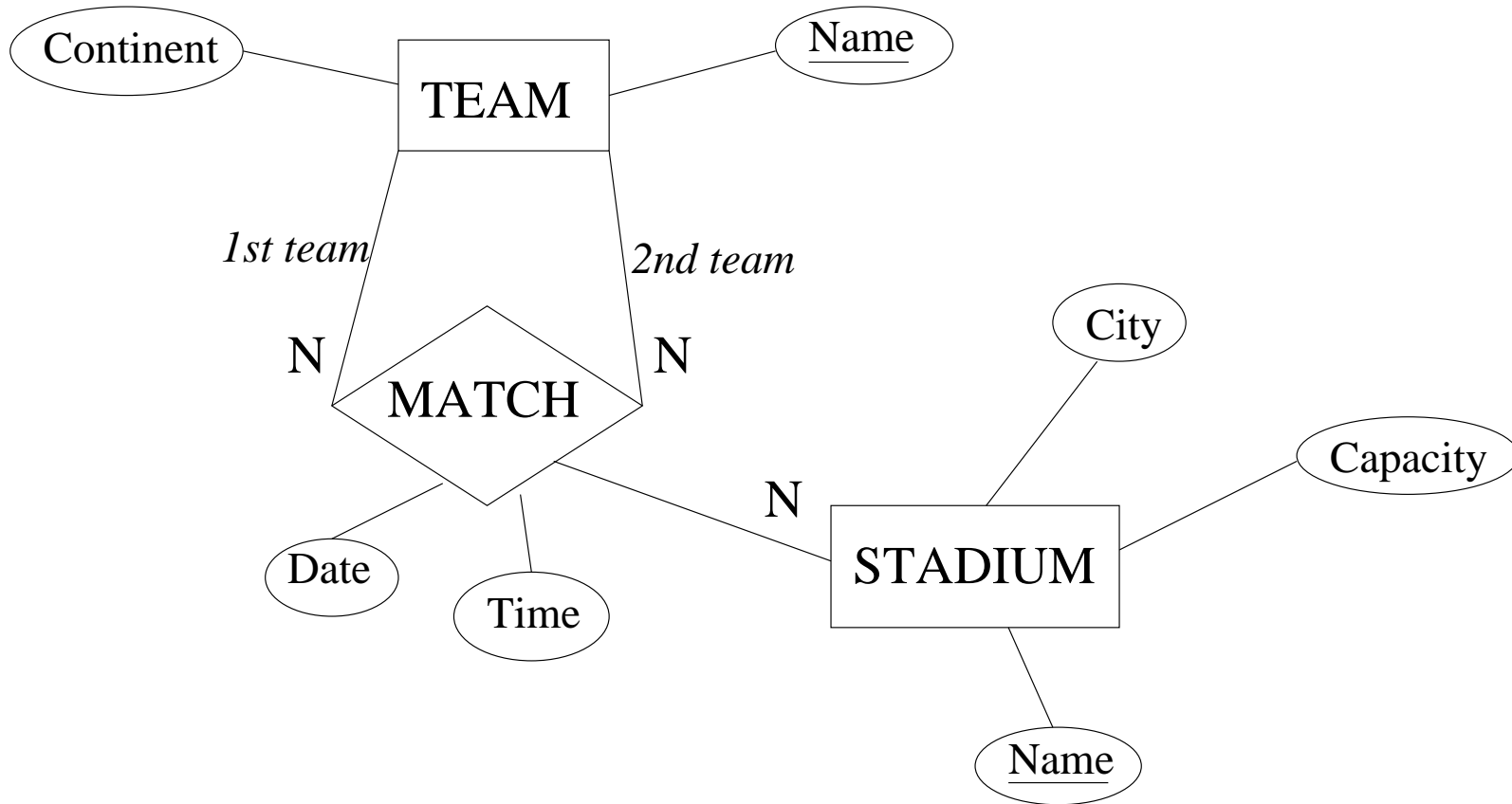
Example: Books



Example: University



Example: Championship



Example: Championship - 2

