E-R MODEL & DB DESIGN: OUTLINE

Database Design Process

Entity-Relationship (E-R) Model — Data Structure Types
– Entity
– Attribute
– Entity Specialization/Generalization
– Relationship

Entity-Relationship (E-R) Model — Constraints
– Attribute — Domain, Key
– Entity Specialization/Generalization — Coverage
– Relationship — Cardinality, Participation, Exclusion

Analyzing Application Requirements / Making Design Choices

Reduction of E-R Schemas to Relational Schemas
Database Design Goals

1. Meet the data content requirements of users.

2. Provide a natural and easy-to-understand structuring of data.

3. Support data processing requirements and any performance objectives (e.g., response time, processing time, storage space, etc.).
Database Design Process

Entity-Relationship (E-R) Model — Data Structure Types
- Entity
- Attribute
- Entity Specialization/Generalization
- Relationship

Entity-Relationship (E-R) Model — Constraints
- Attribute — Domain, Key
- Entity Specialization/Generalization — Coverage
- Relationship — Cardinality, Participation, Exclusion

Analyzing Application Requirements / Making Design Choices

Reduction of E-R Schemas to Relational Schemas
E-R MODEL & DATABASE DESIGN

EXERCISE 1
EXERCISE 1: UNIVERSITY APPLICATION

We want to record information about students, departments, courses and course teaching teams.

- For each student we store the student id, name and majors.
- For each department we store a unique code and name.
- For each course we store a unique course id, name, department and prerequisites.
- For each offering of a course, we store the section, semester and year.
- Each student must enroll in one to five course offerings.
- Each course offering can enroll zero to sixty students.
- For each course offering that a student takes we store the grade.
- Each course offering’s teaching team has one or more staff, who is either an instructor or a TA.
- For each staff assigned to a course offering’s teaching team we store the hkid, name, department and office number.
- For each instructor we store their academic title (e.g., professor).

Construct an E-R diagram for the university application.
The entity-relationship (E-R) model is used at the logical level to describe a database's overall structure.

- The E-R model employs three basic concepts to describe data.
  1. **entity** (something about which we want to keep data).
  2. **attribute** (properties of entities).
  3. **relationship** (among entities).

These are shown in an entity-relationship diagram (E-R diagram).

Why E-R model?
- expressiveness
- user communication
- DBMS independent
An **entity (type)** describes a **set of entity instances** with common:
- properties
- relationships
- semantics

Something we want to store data about in the application domain.
(E.g., employee, student, course, product, order, …)

**Notation:**

- **Employee**

**An entity instance**
- has **identity**.
  - It can be distinguished from other entity instances.
- represents some **real-world thing**.
  - It has meaning in the application domain.

**entity (type)**
(a common description for all employees)

**entity (instance)**
(a specific employee)

**entity set**
(the collection of all employees)
EXERCISE 1: UNIVERSITY APPLICATION—ENTITIES

- For each student we store the student id, name and majors.
- For each department we store a unique code and name.
- For each course we store a unique course id, name, department and prerequisites.
- For each offering of a course, we store the section, semester and year.
- Each student must enroll in one to five course offerings.
- Each course offering can enroll zero to sixty students.
- Each course offering that a student takes we store the grade.
- Each course offering’s teaching team has one or more staff, who is either an instructor or a TA.
- For each staff assigned to a course offering’s teaching team we store the hkid, name, department and office number.
- For each instructor we store their academic title (e.g., professor).
An attribute is a property of an entity and describes the data values of that property.

- Each attribute has a name that is unique within an entity (but not across entities).
- Most attribute values are physically stored (base attribute); some may be calculated using stored values (derived attribute).
- An attribute value may be null (missing, unknown, not applicable).
TYPES OF ATTRIBUTES AND NOTATION

- **Single-valued attribute**
- **Composite attribute**
- **Derived attribute**
- **Multivalued attribute**

### Employee
- **hkid**
- **name**
- **birthdate**
- **age ()**
- **address**
  - **streetNo**
  - **streetName**
  - **flatNo**
  - **building**
- **{skill}**
**EXERCISE 1: UNIVERSITY APPLICATION—ENTITY ATTRIBUTES**

- For each student we store the student id, name and majors.
- For each department we store a unique code and name.
- For each course we store a unique course id, name, department and prerequisites.
- For each offering of a course, we store the section, semester and year.
- Each student must enroll in one to five course offerings.
- Each course offering can enroll zero to sixty students.
- For each course offering that a student takes we store the grade.
- Each course offering’s teaching team has one or more staff, who is either an instructor or a TA.
- For each staff assigned to a course offering’s teaching team we store the hkid, name, department and office number.
- For each instructor we store their academic title (e.g., professor).
**ENTITY GENERALIZATION/SPECIALIZATION**

*Generalization/specialization* is a relationship between the same kind of entities playing different roles.

![Diagram showing generalization/specialization relationships between classes.](image)

In this example, subclass membership is user-defined (i.e., determined by the schema designer and not based on any attribute).
Can also be applied top-down \textit{(attribute-defined)}. 

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>accountNo</td>
</tr>
<tr>
<td>amount</td>
</tr>
<tr>
<td>serviceCharge</td>
</tr>
<tr>
<td>interestRate</td>
</tr>
<tr>
<td>accountType</td>
</tr>
</tbody>
</table>

- Applies only to checking accounts.
- Applies only to saving accounts.
- Values are \{checking, saving\}.
Can also be applied top-down (attribute-defined).

**discriminator**: An attribute of enumeration type that indicates which property of an entity is being abstracted by a generalization/specialization.

In this example, subclass membership is determined by a predicate on an attribute (i.e., the discriminator attribute) of the superclass.
INHERITANCE

_Inheritance is the taking up of properties by a subclass from its superclass._

- We extract the _common_ attributes and relationships, associate them with the superclass and _inherit_ them to the subclass(es).
  - Reduces redundancy of descriptions.
  - Promotes reusability of descriptions.
  - Simplifies modification of descriptions.
  - We only _define_ an entity's properties _in one place._

- A subclass may _add_ new properties (attributes, relationships).

**Design Guideline:** Inheritance should not exceed _2-3 levels._
Multiple inheritance:
a subclass inherits properties from more than one superclass.

Multiple inheritance may result in conflicts, which can be resolved by redefining an attribute's name.

Inherits from both Checking and Saving entities.

For multiple inheritance, a property from the same ancestor entity found along more than one path is inherited only once.
Each course offering’s teaching team has one or more staff, who is either an instructor or a TA.

What should be the generalization?

⟹ Staff superclass; Instructor, TA subclasses.
A relationship (type) is a description of a set of relationships with common properties and semantics.

A relationship (instance) (a specific relationship) (a common description for a relationship set)

relationship (type) (a common description for a relationship set)

relationship set (the collection of all relationships of one type)
The number of entities that participate in a relationship.

- **Unary (reflexive)** — relates the same entity (to itself)
  - Employee ManagerOf

- **Binary** — relates two different entities
  - Employee WorksOn Project

- **Ternary** — relates three different entities
  - Employee Uses Language

Higher degrees are extremely rare!

A ternary relationship often can be expressed as two binary relationships, but not always.

In practice, the vast majority of relationships are binary.
(We will use only unary or binary relationships in this course.)
RELATIONSHIP EXAMPLES

There can be several relationships between entities.

- **Person**
  - OwnsSharesIn
  - WorksFor

- **Company**
  - OwnsSharesIn
  - WorksFor

- **Employee**
  - MarriedTo
  - ManagerOf

- **Albert**
  - OwnsSharesIn
  - WorksFor

- **IBM**

- **Sam**

- **John**

- **Jessie**
  - MarriedTo
  - ManagerOf
EXERCISE 1: UNIVERSITY APPLICATION—RELATIONSHIPS

- For each course we store a unique course id, name, department and prerequisites.

What should be related?

⇒ Course related to Department.
For each course we store a unique course id, name, department and prerequisites.

What should be related?

\[ \implies \text{Course related to Course} \]

(unary relationship).

![Diagram of university application relationships]

- Department
- Offers
- Course
- HasPrerequisite
- Staff
- Instructor
- TA
- Student
- Offering
- Course
- Instructor
- TA
- Department
- Staff
EXERCISE 1: UNIVERSITY APPLICATION—RELATIONSHIPS

- For each offering of a course, we store the section, semester and year.

What should be related?
  \[\implies\text{Offering related to Course.}\]
Each student must enroll in one to five course offerings.
Each course offering can enroll zero to sixty students.
For each course that a student takes we store the grade.

What should be related?
\[\implies\text{Student related to Offering.}\]
EXERCISE 1: UNIVERSITY APPLICATION—RELATIONSHIPS

- For each staff assigned to a course offering’s teaching team we store the hkid, name, department and office number.

What should be related?

⟹ Staff related to Offering.
EXERCISE 1: UNIVERSITY APPLICATION—RELATIONSHIPS

- For each staff assigned to a course offering’s teaching team we store the hkid, name, department and office number.

What should be related?

⟹ Staff related to Department.
We want to represent the percentage time worked on a project.

- Employee
  - hkid
  - name
- Project
  - projectNo
  - title
- WorksOn

<table>
<thead>
<tr>
<th>Employee</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>hkid</td>
<td>projectNo</td>
</tr>
<tr>
<td>name</td>
<td>title</td>
</tr>
</tbody>
</table>

- Employee
  - ..., John, ...
  - ..., James, ...
  - ..., Alan, ...
  - ..., Bill, ...
- Project
  - ..., E-commerce, ...
  - ..., Accounting, ...
  - ..., Stock control, ...
  - ..., Web store, ...

- Percentage:
  - 30%
  - 40%
  - 60%
  - 20%
  - 70%
  - 80%
**Option 1:** Use many attributes (e.g., in Employee). Is this OK?

- **Employee**
  - hkid
  - name
  - %time1
  - %time2

- **Project**
  - projectName
  - title

The relationship between Employee and Project is shown by the `WorksOn` association. The Employee table contains attributes such as name and time percentages, while the Project table contains project names. The diagram illustrates how different employees work on various projects, with time percentages indicating their contribution to each project.
Option 2: Use a multivalued attribute (e.g., in Employee). Is this OK?

Employee

<table>
<thead>
<tr>
<th>hkid</th>
<th>name</th>
<th>WorksOn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td></td>
<td></td>
<td>projectNo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>title</td>
</tr>
</tbody>
</table>

{time}
Option 3: Allow relationships to have attributes. Is this OK?

Option 3: Allow relationships to have attributes. Is this OK?
EXERCISE 1: UNIVERSITY APPLICATION—RELATIONSHIP ATTRIBUTES

- Each student must enroll in one to five course offerings.
- Each course offering can enroll zero to sixty students.
- For each course that a student takes we store the grade.

Any relationship attributes?
⟹ Yes — grade attribute
Add where?
⟹ To EnrollsIn relationship.
A role name is assigned to one end of a relationship to identify the role that the entity at that end plays in the relationship.

Who is the boss and who is the worker?

A role name disambiguates the role that an entity plays in a relationship.

It is necessary to use role names for unary relationships (i.e., when a relationship relates instances from the same entity).
EXERCISE 1: UNIVERSITY APPLICATION—RELATIONSHIP ROLE NAMES

- For each course we store a unique course id, name, department and prerequisites.

Any role names?
⟹ Yes — add role names to HasPrerequisite.
EXERCISE 1: UNIVERSITY APPLICATION—E-R DIAGRAM

**Entities:**
- **Department**
  - Offers (Course)
- **Course**
  - HasPrerequisite
- **Staff**
  - Has (Offering)
  - Appoints (Instructor, TA)
- **Instructor**
- **TA**
- **Offering**
  - AssignedTo (Staff)
  - EnrollsIn (Student)
- **Student**
  - EnrollsIn (Offering)

**Attributes:**
- **Student**
  - studentId
  - name
  - {major}
- **Department**
  - code
  - name
- **Course**
  - courseId
  - name
  - prerequisite
- **Offering**
  - section
  - semester
  - year
- **Staff**
  - hkid
  - name
  - officeNumber
- **Instructor**
  - title
- **TA**
EXERCISE 2: BUS COMPANY

We want to keep track of bus routes and schedules for a bus company.

- Each bus route has a unique route number, a departure station and a destination station.
- For each bus route, there is a schedule, which records all the departure times of buses.
- For each departure time of each route, a driver and a bus can be assigned; however, information about the driver or the bus may sometimes be missing.
- A driver has a unique employee id, a name and a phone number.
- A bus is identified by its license number and has a maximum seating capacity.

Construct an E-R diagram for the bus company application.

What is a Schedule?

<table>
<thead>
<tr>
<th>Route 1</th>
<th>Route 2</th>
<th>Route 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Departure time</td>
<td>Driver</td>
</tr>
<tr>
<td></td>
<td>11:00</td>
<td>Bill</td>
</tr>
<tr>
<td></td>
<td>12:00</td>
<td>Sarah</td>
</tr>
<tr>
<td></td>
<td>13:00</td>
<td>Bill</td>
</tr>
<tr>
<td></td>
<td>Departure time</td>
<td>Driver</td>
</tr>
<tr>
<td></td>
<td>9:00</td>
<td>Al</td>
</tr>
<tr>
<td></td>
<td>11:00</td>
<td>Cindy</td>
</tr>
<tr>
<td></td>
<td>13:00</td>
<td>Al</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
<td>Mark</td>
</tr>
<tr>
<td></td>
<td>Departure time</td>
<td>Driver</td>
</tr>
<tr>
<td></td>
<td>9:00</td>
<td>John</td>
</tr>
<tr>
<td></td>
<td>15:00</td>
<td>Sarah</td>
</tr>
</tbody>
</table>