Conceptual Design.
The Entity-Relationship (ER) Model
Relationship Set Representation

Representation Convention:
- Relationship sets: diamonds
- Edges connect relationship sets to entity sets, and relationship sets to relationship set attributes
- E-R with this notation was published by P. Chen in 1976 (at M.I.T. at the time)
- Other notations are also in use: Murach uses “crow’s foot” notation
Key Constraints

- How many other entities can an entity have a relationship with? (case of binary relationships)

Note: many practitioners lump one-to-many and many-to-one into one category “many-to-one” or N-1 for short, i.e. the many side is not specified by this name. On page 33, “sometimes said to be one-to-many” … “is said to be many-to-one”
Example 1

- **Works_In** relationship: an employee can work in many departments; a dept can have many employees.
  
  *many-to-many*
Example 2

- **Manages** relationship: each dept has *at most one* manager

  *one-to-many*

  from Employees to Departments, or

  *many-to-one*

  from Departments to Employees

*No participation constraint holds here.*
Manages relationship visualized: one-to-many, no participation constraint

Key constraint: each dept has \textit{at most one} manager

Many-to-one from Departments to Employees, so arrow goes right to left.
Participation Constraints

- **Total vs Partial Participation**
  - **Partial:** not every employee is a manager
    - “Employees” entity set has partial participation, thin line
    - As shown in previous slides
  - **Total:** every department must have a manager
    - “Departments” entity set has total participation in relationship
    - Represented as thickened line (there is a key constraint as well, represented by the arrow)
Manages relationship visualized: case of total participation by Departments

Key constraint: each dept has *at most one* manager.
With total participation by Departments, each dept has exactly one manager.
Example Problem 1 from last class

Design a database for a bank, including information about customers and their accounts. Information about customers includes their name, address, phone and SSN. Accounts have numbers, types (e.g., savings/checking) and balances.

1. Draw the E/R diagram for this database.
2. Modify the E/R diagram such that each customer must have at least one account. This is a participation constraint.
3. Modify the E/R diagram further such that an account can have at most one customer, i.e., the relationship is many-to-one from Accounts to Customers.
Example Problem 2: Class Exercise

Design a database for a drugstore, including info on customers and their prescriptions. Information about customers includes their name, phone and SSN. Prescriptions have numbers, drugname, doctor name, and dosage.

1. Draw the E/R diagram for this database.
2. Modify the E/R diagram such that each customer must have at least one prescription. This is a participation constraint.
3. Modify the E/R diagram further such that a prescription has exactly one customer. This is both a participation constraint and a key constraint.
Mapping ER to Relational Schemas

- For most part, process is mechanical
  - Some special cases arise in the presence of constraints

- Translation from ER to SQL requires:
  - Mapping entity sets to tables
  - Mapping relationship sets to tables (or avoiding this in some cases)
  - Capturing key constraints on relationships (like “an account can have at most one customer”)
  - Capturing participation constraints (like “each customer must have at least one account”)
CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

Each employee has a unique SSN: this is a key constraint on Employees, an entity table.
“No-constraints” case follows simple rules

- This means primary keys for the entities are known, but no rules such as “only one manager for a department” (key constraint on relationship) or “each customer must have at least one account” (participation constraint)

Relationship set becomes a relation, attributes include:

- Keys for each participating entity set (as foreign keys pointing to respective entity table)
- All descriptive attributes for relationship
- Primary key of relationship set table is the concatenation of primary keys for the entity sets (assuming no “to-one” relationships involved)
  - Later we’ll see that if the relationship is “to-one”, the PK can be smaller
CREATE TABLE Works_In(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
    REFERENCES Employees,
  FOREIGN KEY (did)
    REFERENCES Departments)

Binary N-N relationship: 2 entity keys in PK
CREATE TABLE Works_In3(
  ssn CHAR(11), did INTEGER,
  address CHAR(20), since DATE,
  PRIMARY KEY (ssn, did, address),
  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments,
  FOREIGN KEY (address) REFERENCES Locations)
What if there are “Key Constraints”, e.g., a relationship known to be many-to-one?

- Each department has at most one manager, according to the key constraint on Manages, shown by the arrow, which follows from the Departments-Employees relationship being many-to-one.
“Each dept has *at most one* manager”: why it’s called a key constraint

- The manages relationship table has (ssn, did) pairs specifying the related entities, and without this to-one constraint, (ssn, did) is the PK of Manages.
- Given this many-to-one key constraint, we see that one department has at most one (ssn, did) row, i.e., *did by itself* is a key of Manages.
- This constraint is called a **key constraint** because the manages relationship table now has a specified key (a more restrictive one). It’s a key constraint on the relationship table.
- The needed key for Manages is the key of Departments, did.
- The arrow goes from Departments to Manages, and shows how the key constraint of Departments is now also the key constraint of Manages.
Case of Many-to-one/key constraint

- Recall from last time:
- Given a key constraint, or equivalently, its many-to-one expression, there are two major ways to design relational tables:
  - Variant 1: use a relationship table with PK = PK of the "one" side of relationship
  - Variant 2: no relationship table, but instead add column(s) for key attribute(s) of the one-side table and any attributes of the relationship to the many-side table, and make the added key column(s) be a foreign key to the one-side table.
- Let's look again at the example…
Variant 1 for “Each department has at most one manager” key constraint on Manages

Map relationship to its own table:

- Note that did is the key now!
  - That means we can’t have two rows here for one department, pointing to two managers for that department.
- Note this setup allows a department without a manager: it would simply be missing from this table.

```
CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    FOREIGN KEY (did) REFERENCES Departments)
```
Variant 2 for “Each department has at most one manager” key constraint on Manages

- Since each department has at most one manager, we could instead combine Manages and Departments.
- And use a nullable FK on ssn (no participation constraint is given)
- A department without a manager would have a null ssn

CREATE TABLE Dept_Mgr(  -- or simply Dept
did INTEGER,
dname CHAR(20),
budget INTEGER,
ssn CHAR(11),  -- note nullable
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn) REFERENCES Employees)
Review: Participation Constraints

- Does every department have a manager?
  - If yes, the participation of Departments in Manages is *total*

- Turns out that it is **NOT** possible to capture this with Variant 1
  - The Dept_Mgr variant is the only way!
Participation Constraints in SQL: “every department has one manager”

CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget INTEGER,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees
    ON DELETE NO ACTION)
Participation Constraints Summary

- General case (not known to be many-to-one or one-to-many)
  - Total participation cannot be enforced unless we use complex constraints not supported in Oracle

- What if there is also a key constraint in place?
  - If the entity set with total participation also has a relationship key constraint deserving an outbound arrow from it, then it is possible to capture total participation
  - i.e., if a FK can do the relationship key constraint, a not-null FK can enforce total participation of the entity with the FK in its table.
  - But only if “combined” table construction is used!
Design a database for a bank, including information about customers and their accounts. Information about customers includes their name, address, phone and SSN. Accounts have numbers, types (e.g., savings/checking) and balances.

1. Draw the E/R diagram for this database.
2. Modify the E/R diagram such that each customer must have at least one account. This is a participation constraint.
3. Modify the E/R diagram further such that an account can have at most one customer, i.e., the relationship is many-to-one from Accounts to Customers.
4. Now specify tables.
Example 2 revisited

Design a database for a drugstore, including info on customers and their prescriptions. Information about customers includes their name, phone and SSN. Prescriptions have numbers, drug name, doctor name, and dosage.

1. Draw the E/R diagram for this database.
2. Modify the E/R diagram such that each customer must have at least one prescription. This is a participation constraint.
3. Modify the E/R diagram further such that a prescription has exactly one customer. This is both a participation constraint and a key constraint.
4. Now specify tables.
Look at our standard tables

Student(snum, sname, major, standing, age)
Faculty(fid, fname, deptid)
Class(name, meets_at, room, fid) FK fid to faculty
Enrolled(snum, name) both FKs

- Enrolled fits the rules for a binary many-to-many relationship table: PK of two entity ids, FKs on them.
- Make sense: each class has many students, each student has many classes (i.e. can have multiple classes)
- Class has (nullable) fid column, with FK to faculty. This corresponds to a many-to-one relationship “is_taught_by” from class to faculty, with no participation constraint on class.
- Draw the ER diagram...
More cases in our standard tables

- **Emp-dept-works**: similarly many-to-many relationship works.

- The FK on managerid in dept indicates a many-to-one relationship “manages”, and nullable managerid means there is no participation requirement on dept in this.

- **Flights-aircraft-employees-certified**: similar many-to-many relationship “certified for” between employees and aircraft.

- **Suppliers-parts-catalog**: similar many-to-many relationship “supplies” between supplier and part.

- Unusual prevalence of many-to-many relationships here
#### Murach Example: The relationships between the tables

**vendors**
- vendor_id
- vendor_name
- vendor_address
- vendor_city
- vendor_state
- vendor_zip_code
- vendor_phone
- vendor_contact_first_name
- vendor_contact_last_name
- terms
- account_no

**invoices**
- invoice_id
- vendor_id
- invoice_number
- invoice_date
- invoice_total
- payment_total
- credit_total
- terms
- invoice_due_date
- payment_date
- account_no

**invoice_line_items**
- invoice_id
- invoice_sequence
- account_no
- line_item_description
- item_quantity
- item_unit_price
- line_item_amount

- This shows two many-to-one relationships (right to left in the diagram), with a “crow’s foot” at the many end.
  - The vendor_id in invoices is a not-null FK
  - The invoice_id in invoice_line_items is a not null FK
- So there are two many-to-one relationships worthy of the thick arrow in R&G notation.
Example of an entity that looks like a relationship

Concept: a sailor reserves a boat for a certain day (pg. 102)
Use of a verb as its name makes it sound like a pure relationship, but is it?

```sql
create table reserves(
sid int not null, -- key value of sailor table
bid int not null, -- key value of boat table
day varchar(10), -- day of reservation
foreign key (sid) references sailors(sid),
foreign key (bid) references boats(bid)
);
```

What is missing from this compared to the N-N relationship table?
Example of an entity that looks like a relationship, continued

Concept: a sailor reserves a boat for a certain day. One sailor can reserve the same boat for different days.

```sql
create table reserves(
sid int not null,  -- key value of sailor table
bid int not null,  -- key value of boat table
day varchar(10),  -- day of reservation
foreign key (sid) references sailors(sid),
foreign key (bid) references boats(bid)
);
```

- What is missing from this compared to the N-N relationship table????
  - Answer: Primary key(sid, bid).
- What is the PK here?
  - Answer: the set of all 3 columns, the PK of last resort.
  - That means it should be a ternary relationship: sailors, boats, days, but days don’t have their own entity table
  - Could say days are simple or degenerate entities (no attributes)
- Another approach: call this table “reservation” or “reservations”, using a noun to go with its being an entity, not a pure relationship.
Pure relationships can have attributes

**Concept:** a certain part supplied by a certain supplier has a certain cost.

```
create table catalog(
sid int, -- key value in suppliers
pid int, -- key value in parts
cost decimal(10,2),
primary key(sid,pid),
foreign key(sid) references suppliers(sid),
foreign key(pid) references parts(pid)
);
```

- Here we see the pure relationship-table pattern, so we can classify this as a pure binary relationship table, N-N, in spite of its noun name. So what’s cost doing in here?

- It’s an attribute of the relationship: suppliers supply parts, each with its particular cost. Different suppliers can charge different amounts for the same generic part. The PK (sid,pid) means that for a certain pid and sid, we have just one cost.
Do we have to call catalog a relationship?

• No, it’s just that we can call it that. Any N-N relationship with attributes can alternatively be considered an entity with two N-1 relationships outbound from it.

• This flexibility is often used in data modeling environments that don’t support attributes of relationships.

• We can draw two E-R diagrams for this…on the board anyway.
A weak entity can be identified uniquely only by considering the key of another (owner) entity.

- 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.
- The weak entity set and the identifying relationship set are marked with thickened lines.

Partial key
Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```sql
CREATE TABLE Dep_Policy (  
  pname CHAR(20),  
  age INTEGER,  
  cost REAL,  
  ssn CHAR(11) NOT NULL,  
  PRIMARY KEY (pname, ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees,  
  ON DELETE CASCADE)
```
Murach Example: line items are weak entities

- This shows two many-to-one relationships, with a “crow’s foot” at the many end.
  - The vendor_id in invoices is a not-null FK
  - The invoice_id in invoice_line_items is a not null FK
- The PK of invoice_line_items (in bold) contains invoice_id, so invoice_sequence is a partial key
ISA ("is a") Hierarchies

- Reasons for using ISA:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.
  - If A ISA B, every A entity is also considered to be a B entity …
  - … and A inherits attributes from B
ISA is hard to do in a DB

- Unlike in programming languages, there is no built-in support for inheritance in relational databases.
- We can use table-supported relationships and end up with multiple tables, hard to understand and query.
- Simplest solution, most common in practice:
  - Use one big table with all attributes of superclass and all subclasses, one PK.
  - Use nullable attributes for attributes like hourly_wages and contractid, so they can be null if not appropriate.
- Modeling systems (object-relational mapping) can handle complexity of more sophisticated approaches.
Example of use of ISA where a subclass has its own relationship (FYI)

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.
- What if a manager gets a discretionary budget that covers all managed depts?
- Manager has attribute dbudget, relationship to his/her departments.
Skipped Sections of R&G

- From home page, ER box:

  R&G: Chapter 2 - all except 2.4.5, 2.5.3, 2.5.4, 2.6-2.8; Chapter 3 - 3.5 (up to 3.5.5)

- So we’re skipping ER aggregation (non-standard dashed box notation), sec. 2.4.5

- We’re skipping Binary vs. ternary relationships, sec. 2.5.3, 2.5.4. We are concentrating on binary relationships.

- We’re skipping Sec. 3.5.6, Translation of ISA relationships. The book covers two methods, but there are more. It’s an advanced topic.
Summary of ER Model

- **Conceptual design** follows **requirements analysis**
  - Yields a high-level description of data to be stored

- ER model popular for conceptual design
  - Constructs are expressive, similar to human thinking

- Basic constructs: *entities, relationships, and attributes*

- Additional constructs: *weak entities, ISA hierarchies*
Summary of ER Model (contd.)

- Several kinds of constraints:
  - *key constraints*
  - *participation constraints*

- Constraints play an important role in determining the best database design for an enterprise.

- ER design is *subjective*
  - Many ways to model a given scenario!
  - Entity vs. attribute, entity vs. relationship
  - Binary vs. multi-way relationships
  - Whether or not to use ISA hierarchies
Oracle SQL Developer Diagrams: Relational Model
Oracle SQL Developer Diagrams: ER Model in “Bachman Notation” No many-to-many relationships