

Data Modeling
Chapter 7

Class 05: Data Modeling 1

Entity Relationship Modeling

- **Entity Relationship Modeling:**
 - The process of visually representing entities, attributes, and relationships
 - Also applicable to many non-relational databases
 - Produces an *Entity Relationship Diagram (ERD)*
 - An iterative process
 - ERDs are platform independent and can be understood by nontechnical people

Class 05: Topic 4.1: Data Modeling 2

ERD Formats

- Original developed by Peter Chen in 1976
- Many variations exist today, all conceptually the same
- Many common elements (next slide)

Class 05: Topic 4.1: Data Modeling 3

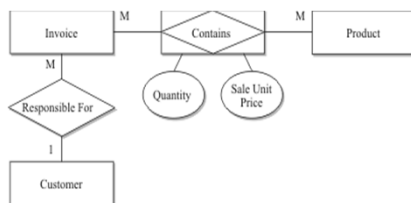
ERD Common Elements

- Entities represented as rectangles or boxes
- Relationships represented as lines
- Symbols on or next to the line ends represent maximum **cardinality** (one or many)
- Symbols near the line ends represent **minimum cardinality** (mandatory or optional relationship participation, which some call *optionality*)
- Attributes may be optionally included

Chen's Format

- Peter Pin-Shan Chen, American computer scientist
 - B.S., electrical engineering, National Taiwan University, 1968
 - Ph.D., computer science and applied mathematics, Harvard University, 1973
 - Distinguished Career Scientist and faculty member at Carnegie Mellon University
- Developed the original ERD format in 1976
 - While an Assistant Professor at MIT Sloan School of Management

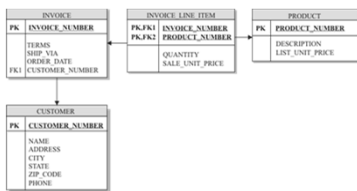
Chen's Format



Conventions in Chen's Format

- Relationship lines contain a diamond into which a descriptive word or phrase is placed
 - For intersection relationships, a rectangle is often drawn around the diamond
 - Proved cumbersome in practice
- Maximum cardinality shown with the symbols "1" (one) and "M" (many)
- Minimum cardinality not shown
- Attributes enclosed in ellipses connected to the entity or relationship (quickly cluttered the landscape)

The Relational Format

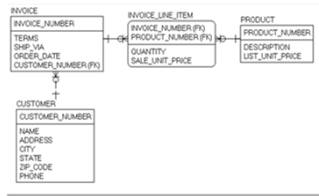


- Supported by ER/Studio and PowerDesigner modeling tools
- Also supported by Visio
- This example is likely a physical model (based on the names)

The Relational Format

- Arrowhead on line end to signify "one"; plain line end to signify "many"
 - Matches object diagram format
 - Arrowhead points to "parent"
- Attributes listed inside entity rectangles
 - Unique identifiers shown above horizontal line; may be underscored and/or noted with "PK"
 - Foreign keys usually noted with "FK"

Information Engineering Format



- Originally developed by Australian Clive Finkelstein in the late 1970s
- Collaborated with James Martin in the early 1980s to publicize the IE format

Class 05: Topic 4.1: Data Modeling

10

Information Engineering (IE) Format

- Generally preferred in the private sector
- Entities:
 - Shown with rectangles
 - Dependent entities have rounded corners
 - Name appears outside the rectangle
 - Foreign keys marked with "(FK)"
 - Rectangle divided with identifying attributes above the dividing line

Class 05: Topic 4.1: Data Modeling

11

IE Format

- Relationships:
 - *Identifying Relationships* (those with FK included in PK) shown with solid line; *Nonidentifying Relationships* shown with dotted line
 - Maximum cardinality shown at line end with short vertical line (one) or "crow's foot" (many)
 - Minimum cardinality shown near line end with circle (zero) or short vertical line (one)

Class 05: Topic 4.1: Data Modeling

12

IDEF1X Format

• Relationships

- Symbols are asymmetric (different set for the "one" side than the "many" side)
- Symbols show both optionality and cardinality
- If relationship is part of an entity's identifier, a solid line is used; else a dotted line is used
- On the "many" side:
 - A solid circle denotes cardinality of zero, one or more.
 - A "P" denotes a mandatory relationship (cardinality of at least 1); "1" means "one and only one"
- On the "one" side:
 - No symbol on/near the line means "one and only one"
 - A diamond means "zero or one and only one" (optional)

Class 05: Topic 4.1: Data Modeling

16

Unified Modeling Language (UML)

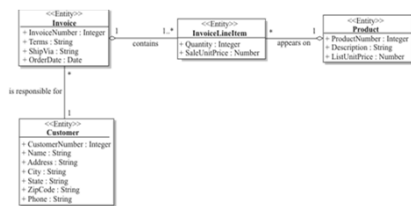
- A general-purpose modeling language for providing a standard way to visualize the design of a software system.
- Developed by Grady Booch, Ivar Jacobson, and James Rumbaugh, all of Rational Software in 1994-95.
 - An integral part of the Rational Unified Process (RUP)
 - Additional development followed
 - Object Management Group (OMG) adopted UML as a standard in 1997
 - UML has 13 types of diagrams that can be used to model the behavior and structure of a software system.

Class 05: Topic 4.1: Data Modeling

17

UML Object Class Diagram

- Class Diagram is the one of interest to data modelers



Class 05: Topic 4.1: Data Modeling

18


Class Diagram Conventions

- Entities shown in a rectangle that represents its object class.
 - Symbol <<Entity>> included with the class name
- Unique identifiers (primary keys) not shown
 - Defined elsewhere with the UML model.
- Foreign keys not shown
 - Not used in object-oriented systems.
- Attributes, called *variables* in o-o terminology, shown with a name followed by a type (separated with a colon).

Class Diagram Conventions

- Relationships are shown with lines.
- Cardinality and optionality of the relationships shown using a combined symbol near the end of the line:

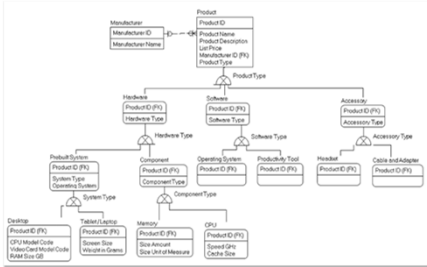
1	One and only one
*	Zero, one, or more
1..*	One or more
1..y	Between x and y occurrences, where: x can be 0 or any positive integer y can be a positive integer, or the symbol * to denote "or more" x must be greater than y (if y and x are the same, then y is simply omitted)

- Diamond  aggregation, a dependency between two entity types
- Specialization and generalization (super types and subtypes) denoted using a line between the two entities
 - Hollow arrow points toward the general class (the super type)

Supertypes and Subtypes

- Some entities can be broken into more specific categories or types
 - More detailed entities are *subtypes* (*subclass* in object technology)
 - More general entities are *supertypes* (*superclasses* in object technology)
- Must be broken down by *type*, not *state*
- Tradeoff between generalization and specialization

ERD with Supertypes and Subtypes



Class 05: Topic 4.1: Data Modeling

22

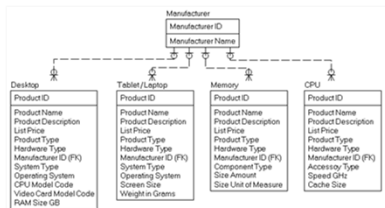
Modeling Supertypes and Subtypes

- Define attributes at the appropriate level
 - Placed at a level so they apply to the entity in which they are stored, plus all of the subtypes below it
- Attributes next to subtype symbols are **type discriminators**
 - Contains a value that indicates which of the subtypes applies to each tuple (row) that will be stored in the supertype
- Getting the balance right can be a challenge
 - Too general hampers business rule enforcement
 - Too specialized hampers ease of use (too many joins)

Class 05: Topic 4.1: Data Modeling

23

Model Variant – Highly Specialized



- Partial listing of entities shown (there would be 8 in all)
- Everything is collapsed into the lowest level entity

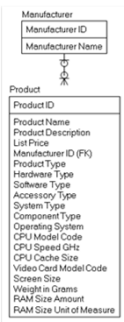
Class 05: Topic 4.1: Data Modeling

24

Highly Specialized Models

- **Benefits:**
 - **Simplicity:** No hierarchy to navigate; limited joins required
- **Challenges:**
 - **Must know product type to find the data** (can be partly mitigated using views with a UNION)
 - **Primary key values must be controlled to prevent duplicates**
 - **Must add a new table for any new type of product**

Model Variant – Highly Generalized

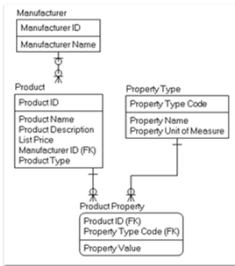


- **Everything is rolled up into the highest level entity**

Highly Generalized Models

- **Benefits:**
 - **Simplicity:** No hierarchy to navigate; limited joins required
- **Challenges:**
 - **Impossible to know which attributes apply to a given product** – all this must be handled in the application
 - **As the design evolves, the lone product entity could contain a very large number of attributes**
 - **Wide table can present performance issues in row-oriented relational database tables**

Object-like Generalization



- Optional attributes defined as properties
- Property values stored as rows in the Product Property entity
 - Much like name-value pairs in a NoSQL database

Class 05: Topic 4.1: Data Modeling

28

Example of Data Values

Product ID	Product Name	Product Description	List Price	Manufacturer ID	Product Type
10056708	Crucial 8 GB Kit	Crucial 8GB Kit (4GBx2) DDR3/DDR3L 1066 MT/s (PC3-8500) SODIMM 204-Pin Mac Memory	89.99	127498	Hardware
10056712	Kingston	Kingston HyperX FURY 16GB Kit (2x8GB) 1866MHz DDR3 CL10 DIMM - Black	105.0	128642	Hardware

Property Type Code	Property Name	Property Unit of Measure
HWDR TYPE	Hardware Type	<NULL>
COMP TYPE	Component Type	<NULL>
RAM SIZE	RAM Size Amount	Gigabyte
---	---	---

Product ID	Property Type Code	Property Value
10056708	HWDR TYPE	Component
10056708	COMP TYPE	Memory
10056708	RAM SIZE	8
---	---	---
10056712	HWDR TYPE	Component
10056712	COMP TYPE	Memory
10056712	RAM SIZE	16
---	---	---

Class 05: Topic 4.1: Data Modeling

29

Advantages of Object-like Model

- Highly flexible.
 - Adding new attributes (properties) is simply a matter of inserting a new row into the Property Type entity (table), which makes it immediately available for use.
- Complex searches involving combinations of properties are quite simple to write because all the properties are in one table.
- Business users generally have little difficulty understanding designs of this sort
 - Especially compared to the original design we looked at with multiple layers of supertype and subtype entities

Class 05: Topic 4.1: Data Modeling

30

Disadvantages of Object-like Model

- Without adding an elaborate set of cross-reference tables, the data in the database cannot tell you which properties are appropriate for a given product subtype.
- Queries must join a lot of rows in order to retrieve all the possible properties for products.
 - For databases of this type, several dozen properties per product would not be unusual.

Disadvantages of Object-like Model

- The DBMS will not be able to enforce mandatory attributes for a given product subtype.
 - A database constraint cannot be used to enforce mandatory attribute business rules
- Queries that assemble all the properties of a product together in a single row in the query result set are quite complex.

Guidelines for Drawing ERDs

- Do not try to relate every entity to every other entity
 - Entities are related *only* when the *entire* primary key in one entity appears as a foreign key in the other
- Except for subtypes, avoid relationships involving more than two entities
- Be consistent with entity and attribute names
- Use abbreviations only when necessary; use a standard list of abbreviations
- Name primary and foreign keys consistently
- Strive for action words in relationship names



"That's the last bug. Tomorrow we go production..."

Class 05: Topic 4.1: Data Modeling

34
