
Data Modeling and Relational Database Design

Volume 1 • Student Guide

Course Code 20000GC12

Edition 1.2

July 2001

D33098

ORACLE®

Authors

Jan Speelpenning
Patrice Daux
Jeff Gallus

Technical Contributors and Reviewers

Simmie Kastner
Sunshine Salmon
Satyajit Ranganathan
Stijn Vanbrabant
Joni Lounsberry
Kate Heap
Gabriella Varga

Publishers

Avril Price-Budgen
Fiona Simpson
Don Griffin

Copyright © Oracle Corporation, 1998, 1999, 2001. All rights reserved.

This documentation contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited. If this documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c) (1) (ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of the Worldwide Education Services group of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Right," as defined in FAR 52.227-14, Rights in Data-General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Education Products, Oracle Corporation, 500 Oracle Parkway, Box 659806, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

Oracle, SQL*Plus, SQL*Net, Oracle Developer, Oracle7, Oracle8, Oracle Designer and PL/SQL are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

Contents

Lesson 1: Introduction to Entities, Attributes, and Relationships

Introduction	1-2
Why Conceptual Modeling?	1-4
Entity Relationship Modeling	1-7
Goals of Entity Relationship Modeling	1-8
Database Types	1-9
Entities	1-10
Entities and Sets	1-12
Attributes	1-13
Relationships	1-15
Entity Relationship Models and Diagrams	1-17
Representation	1-18
Attribute Representation	1-19
Relationship Representation	1-20
Data and Functionality	1-23
Types of Information	1-24
Other Graphical Elements	1-27
Summary	1-28
Practice 1—1: Instance or Entity	1-29
Practice 1—2: Guest	1-30
Practice 1—3: Reading	1-31
Practice 1—4: Read and Comment	1-32
Practice 1—5: Hotel	1-33
Practice 1—6: Recipe	1-34
General Instructor Notes	1-35
Practices	1-38
Suggested Timing	1-41
Workshop Interviewing	1-42

Lesson 2: Entities and Attributes in Detail

Introduction	2-2
Data Compared to Information	2-4
Data	2-5
Tracking Entities	2-7
Electronic Mail Example	2-9
Evolution of an Entity Definition	2-11
Functionality	2-13
Tracking Attributes	2-14
Subtypes and Supertypes	2-17
Summary	2-20
Practice 2—1: Books	2-21
Practice 2—2: Moonlight	2-22
Practice 2—3: Shops	2-23
Practice 2—4: Subtypes	2-24
Practice 2—5: Schedule	2-25
Practice 2—6: Address	2-26
Practice 2—6: Address (continued)	2-27

Lesson 3: Relationships in Detail

Introduction	3-2
Establishing a Relationship	3-4
Relationship Types	3-9
Relationships and Attributes	3-16
Attribute Compared to Relationship	3-18
Relationship Compared to Attribute	3-19
m:m Relationships May Hide Something	3-20
Resolving Relationships	3-25
Normalization During Data Modeling	3-28
Summary	3-32
Practice 3—1: Read the Relationship	3-33
Practice 3—2: Find a Context	3-34
Practice 3—3: Name the Intersection Entity	3-35
Practice 3—4: Receipt	3-36
Practice 3—5: Moonlight P&O	3-37
Practice 3—6: Price List	3-39

Practice 3—7: E-mail	3-40
Practice 3—8: Holiday	3-41
Practice 3—9: Normalize an ER Model	3-42

Lesson 4: Constraints

Introduction	4-2
Identification	4-4
Unique Identifier	4-6
Arcs	4-12
Arc or Subtypes	4-16
More About Arcs and Subtypes	4-17
Hidden Relationships	4-18
Domains	4-19
Some Special Constraints	4-20
Summary	4-24
Practice 4—1: Identification Please	4-25
Practice 4—2: Identification	4-26
Practice 4—3: Moonlight UID	4-28
Practice 4—4: Tables	4-29
Practice 4—5: Modeling Constraints	4-30

Lesson 5: Modeling Change

Introduction	5-2
Time	5-4
Date as Opposed to Day	5-5
Entity DAY	5-6
Modeling Changes Over Time	5-7
A Time Example: Prices	5-10
Current Price	5-16
Journalling	5-17
Summary	5-19
Practice 5—1: Shift	5-20
Practice 5—2: Strawberry Wafer	5-21
Practice 5—3: Bundles	5-22
Practice 5—4: Product Structure	5-24

Lesson 6: Advanced Modeling Topics

Introduction	6-2
Patterns	6-4
Master Detail	6-5
Basket	6-6
Classification	6-7
Hierarchy	6-8
Chain	6-10
Network	6-11
Symmetric Relationships	6-13
Roles	6-14
Fan Trap	6-15
Data Warehouse	6-16
Drawing Conventions	6-17
Generic Modeling	6-19
Generic Models	6-20
More Generic Models	6-21
Most Generic Model	6-22
Summary	6-23
Practice 6—1: Patterns	6-24
Practice 6—2: Data Warehouse	6-25
Practice 6—3: Argos and Erats	6-26
Practice 6—4: Synonym	6-27

Lesson 7: Mapping the ER Model

Introduction	7-2
Why Create a Database Design?	7-4
Transformation Process	7-6
Naming Convention	7-8
Basic Mapping	7-12
Relationship Mapping	7-14
Mapping of Subtypes	7-20
Subtype Implementation	7-23
Summary	7-30
Practice 7—1: Mapping basic Entities, Attributes and Relationships	7-31
Practice 7—2: Mapping Supertype	7-32

Practice 7—3: Quality Check Subtype Implementation	7-33
Practice 7—4: Quality Check Arc Implementation	7-34
Practice 7—5: Mapping Primary Keys and Columns	7-35

Lesson 8: Denormalized Data

Introduction	8-2
Why and When to Denormalize	8-4
Storing Derivable Values	8-6
Pre-Joining Tables	8-8
Hard-Coded Values	8-10
Keeping Details With Master	8-12
Repeating Single Detail with Master	8-14
Short-Circuit Keys	8-16
End Date Columns	8-18
Current Indicator Column	8-20
Hierarchy Level Indicator	8-22
Denormalization Summary	8-24
Practice 8—1: Name that Denormalization	8-25
Practice 8—2: Triggers	8-26
Practice 8—3: Denormalize Price Lists	8-29
Practice 8—4: Global Naming	8-30

Lesson 9: Database Design Considerations

Introduction	9-2
Reconsidering the Database Design	9-4
Oracle Data Types	9-5
Most Commonly-Used Oracle Data Types	9-6
Column Sequence	9-7
Primary Keys and Unique Keys	9-8
Artificial Keys	9-11
Sequences	9-13
Indexes	9-16
Choosing Columns to Index	9-19
When Are Indexes Used?	9-21
Views	9-23
Use of Views	9-24
Old-Fashioned Design	9-25

Distributed Design	9-27
Benefits of Distributed Design	9-28
Oracle Database Structure	9-29
Summary	9-31
Practice 9—1: Data Types	9-32
Practice 9—2: Artificial Keys	9-34
Practice 9—3: Product Pictures	9-35

Appendix A: Solutions

Introduction to Solutions	A-2
Practice 1—1 Instance or Entity: Solution	A-4
Practice 1—2 Guest: Solution	A-5
Practice 1—3 Reading: Solution	A-6
Practice 1—4 Read and Comment: Solution	A-7
Practice 1—5 Hotel: Solution	A-8
Practice 1—6 Recipe: Solution	A-9
Practice 2—1 Books: Solution	A-11
Practice 2—2 Moonlight: Solution	A-12
Practice 2—3 Shops: Solution	A-13
Practice 2—4 Subtypes: Solution	A-14
Practice 2—5 Schedule: Solution	A-15
Practice 2—6 Address: Solution	A-16
Practice 3—1 Read the Relationship: Solution	A-18
Practice 3—2 Find a Context: Solution	A-19
Practice 3—3 Name the Intersection Entity: Solution	A-20
Practice 3—4 Receipt: Solution	A-21
Practice 3—5 Moonlight P&O: Solution	A-23
Practice 3—6 Price List: Solution	A-27
Practice 3—7 E-mail: Solution	A-28
Practice 3—8 Holiday: Solution	A-30
Practice 3—9: Normalize an ER Model: Solution	A-32
Practice 4—1 Identification Please: Solution	A-34
Practice 4—2 Identification: Solution	A-36
Practice 4—3 Moonlight UID: Solution	A-39
Practice 4—4 Tables: Solution	A-40
Practice 4—5 Constraints: Solution	A-41

Practice 5—1 Shift: Solution	A-42
Practice 5—2 Strawberry Wafer: Solution	A-43
Practice 5—3 Bundles: Solution	A-44
Practice 5—4 Product Structure: Solution	A-46
Practice 6—1 Patterns: Solution	A-47
Practice 6—2 Data Warehouse: Solution	A-49
Practice 6—3 Argos and Erats: Solution	A-50
Practice 6—4 Synonym: Solution	A-51
Practice 7—1 Mapping basic Entities, Attributes and Relationships: Solution	A-52
Practice 7—2 Mapping Supertype: Solution	A-53
Practice 7—3 Quality Check Subtype Implementation: Solution	A-54
Practice 7—4 Quality Check Arc Implementation: Solution	A-55
Practice 7—5 Primary Keys and Columns: Solution	A-56
Practice 8—1 Name that Denormalization: Solution	A-57
Practice 8—2 Triggers: Solution	A-58
Practice 8—3 Denormalize Price Lists: Solution	A-61
Practice 8—4 Global Naming: Solution	A-63
Practice 9—1 Data Types: Solution	A-64
Practice 9—2 Artificial Keys: Solution	A-66
Practice 9—3 Product Pictures: Solution	A-67

Appendix B: Normalization

Introduction	B-2
Normalization and its Benefits	B-3
First Normal Form	B-7
Second Normal Form	B-9
Third Normal Form	B-11
Summary	B-13

1

Introduction to Entities, Attributes, and Relationships

Introduction

Lesson Aim

This lesson explains the reasons for conceptual modeling and introduces the key role players: entities, attributes, and relationships.

Overview

- **Why conceptual modeling?**
- **Introduction of the Key role players:**
 - **Entities**
 - **Attributes**
 - **Relationships**

1-2

Topic	See Page
Introduction	2
Why Conceptual Modeling?	4
Entity Relationship Modeling	7
Goals of Entity Relationship Modeling	8
Database Types	9
Entities	10
Entities and Sets	12
Attributes	13
Relationships	15
Entity Relationship Models and Diagrams	17
Representation	18
Attribute Representation	19
Relationship Representation	20
Data and Functionality	23

Topic	See Page
Types of Information	24
Other Graphical Elements	27
Summary	28
Practice 1—1: Instance or Entity	29
Practice 1—2: Guest	30
Practice 1—3: Reading	31
Practice 1—4: Read and Comment	32
Practice 1—5: Hotel	33
Practice 1—6: Recipe	34

Objectives

At the end of this lesson, you should be able to do the following:

- Explain why conceptual modeling is important
- Describe what an entity is and give examples
- Describe what an attribute is and give examples
- Describe what a relationship is and give examples
- Draw a simple diagram
- Read a simple diagram

Why Conceptual Modeling?

This is a course on conceptual data modeling and physical data modeling. Why do you need to learn this? Why invest time in creating entity models when you need tables? Why bother about business functionality and interviews and feedback sessions when you need programs? In this course you learn why. You learn why it is a wise decision to spend time in modeling and why it is a good investment. You will learn even more, including how to create, read, and understand models and how to check them, as well as how to derive table and key definitions from them.

Why Create a Conceptual Model?

- **It describes exactly the information needs of the business**
- **It facilitates discussion**
- **It helps to prevent mistakes, misunderstanding**
- **It forms important “ideal system” documentation**
- **It forms a sound basis for physical database design**
- **It is a very good practice with many practitioners**

1-3

This list shows the reasons for creating a conceptual model. The most important reason is that a conceptual model facilitates the discussion on the shape of the future system. It helps communication between you and your sponsor as well as you and your colleagues. A model also forms a basis for the default design of the physical database. Last but not least, it is relatively cheap to make and very cheap to change.

What You Learn in This Course

In this course you learn how to analyze the requirements of a business, how to represent your findings in an entity relationship diagram and how to define and refine the tables and various other database objects from that model.

In summary, as a result of what you learn in this course you will know:

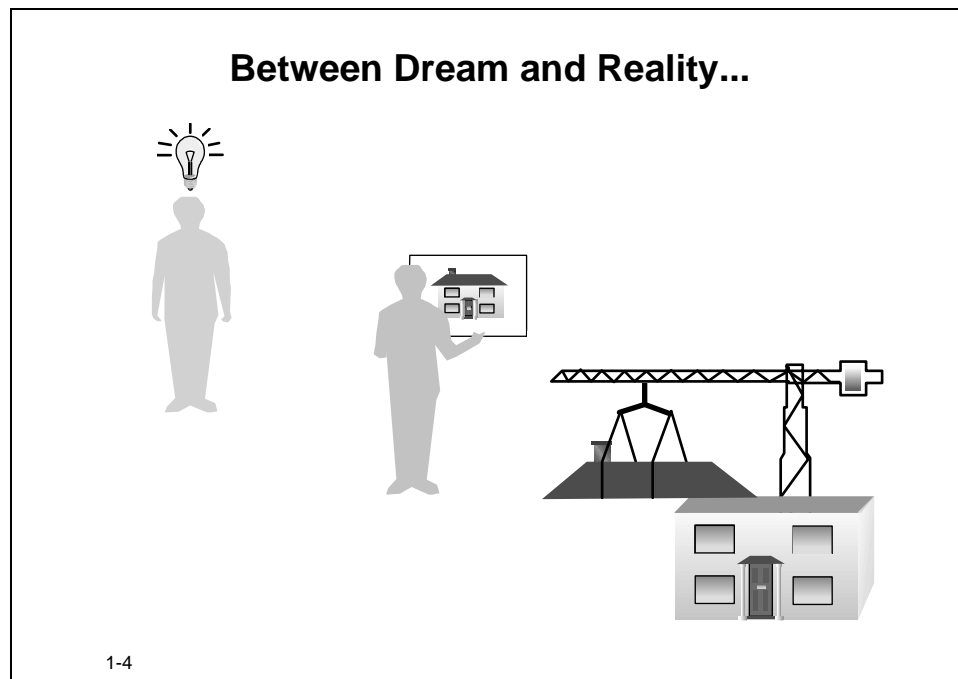
- How to model the information needs of a business and the rules that apply.
- Which tables you need in your database, and why.
- Which columns you need in your tables, and why.
- Which constraints and other database objects you require.

You will also know how to explain this to:

- Your sponsors.
- The developers.
- Your fellow designers.

The House Building Metaphor

Imagine someone who wants to have a house built. Initially, the house only exists in the minds of the future home owners as ideas, or as pieces of various dreams. Sometimes the future inhabitants may not even know what they want, or know if what they want is even feasible. Dreams may be full of internal contradictions and impossibilities. This is not a problem in the dream world, but in the physical realm any inconsistencies and obstacles need to be resolved before someone can construct the house.



A building contractor needs a solid plan, a set of blueprints of the house with a description of the materials to be used, the size of the roof beams, the capacity of the plumbing and many, many other things. The contractor follows the plan, and has the knowledge to construct what is on the blueprint. But how do the ideas of the home owner become the blueprint for contractor? This is where the architect becomes involved.

The Architect

The architects are the intermediary between sponsor and constructor. They are trained in the skills of translating ideas into models. The architect listens to the description of the ideas and asks all kinds of questions. The architect's skills in extracting the ideas, putting it down in a format that allows discussion and analysis, giving advice, describing sensible options, documenting it, and confirming it with the home owners, are the cornerstones to providing the future home-owner with a plan of the home they want.

Sketches

The architect's understanding of the dreams is transformed into sketches of the new house—only sketches! These consist of floor plans and several artist's impressions, and show the functional requirements of the house, not the details of the construction. This is a conceptual model, the first version.

Easy Change

If parts of the model are not satisfactory or are misunderstood, the model can easily be changed. Such a change would only need a little time and an eraser, or a fresh sheet of paper. Remember, it is only changing a model. The cost of change at this stage is very low. Certainly it is far less costly than making changes to the floor plan or roof dimensions after construction has started. The house model is then reviewed again, and further changes are made. The architect continues to explore and clarify the dreams and make alternative suggestions until all controversial issues are settled, and the model is stable and ready for the final approval by the sponsor.

Technical Design

Then the architect converts the model into a technical design, a plan the contractor can use to build the house. Calculations are made to determine, for example, the number of doors, how thick the walls and floor beams must be, the dimensions of the plumbing, and the exact construction of the roof. These are technical issues that need not involve the customer.

What? as Opposed to How?

While the conceptual model addresses the What? phase in the process, the design addresses the question of How? it is to be constructed.

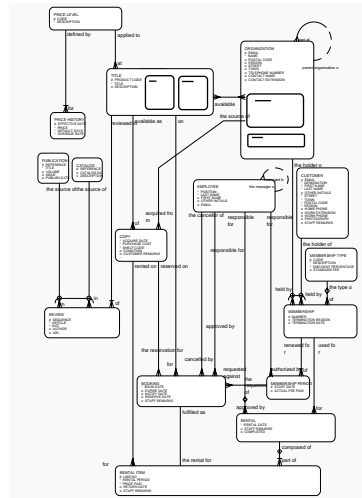
Conceptual modeling is similar to the work of an architect—transforming things that only exist in people's minds into a design that is sufficiently substantial to be created physically.

Entity Relationship Modeling

Entity Relationship Modeling

- **Models business, not implementation**
- **Is a well-established technique**
- **Has a robust syntax**
- **Results in easy-to-read diagrams...**

...although they may look rather complex at first sight



1-5

What is Involved in Modeling?

Entity Relationship modeling is about modeling a business. To be more precise: it is about modeling the data requirements for a business based on the current or desired functionality of the future system.

To model a business you have to understand to a fair degree of detail what the business is about.

Entity Relationship modeling is a technique used to describe the shared understanding of the information needs of a business. It is a well-established technique that leads to diagrams which are quite easy to read and therefore also easy to check.

Goals of Entity Relationship Modeling

Goals of Entity Relationship Modeling

- Capture *all* required information
- Information appears *only once*
- Model *no* information that is derivable from other information already modeled
- Information is in a predictable, logical place

1-6

The goals of conceptual data modeling are to ensure that:

- All pieces of information that are required to run a business properly are recognized.

Models should be complete. Requirements should be known before you start implementing. Dependencies must be clear.

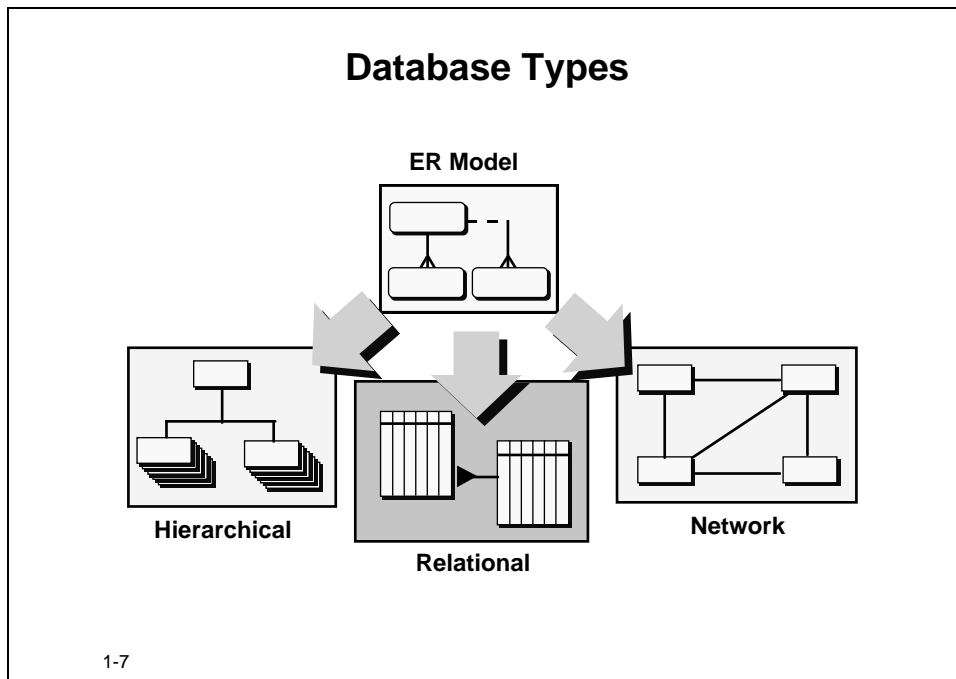
- Every single piece of required information appears only once in the model.

This is an important goal. As soon as a system stores particular information twice, you run into the possibility that this information is not the same in both places. If you are a user of an information system and discover inconsistencies in the data, which information would you trust?

This goal implies that an ideal system does not contain derivable information.

- In the future system, the information is made available in a predictable, logical place; related information is kept together.
- A proper Entity Relationship model leads to a set of logically coherent tables.

Database Types



Entity Relationship modeling is independent of the hardware or software used for implementation. Although you can use an Entity Relationship model as a basis for hierarchical databases, network databases, and relational databases, it is strongly connected to the latter.

Entities

This section gives definitions and examples.

Entity

- **An Entity is:**
 - “**Something**” of significance to the business about which data must be known.
 - **A name for the things that you can list.**
 - **Usually a noun.**
- **Examples: objects, events**
- **Entities have instances.**

1-8

Definition of an Entity

There are many definitions and descriptions of an *entity*. Here are a few; some are quite informal, some are very precise.

- An entity is something of interest.
- An entity is a category of things that are important for a business, about which information must be kept.
- An entity is something you can make a list of, and which is important for the business.
- An entity is a class or type of things.
- An entity is a named thing, usually a noun.

Two important aspects of an entity are that it has instances and that the instances of the entity somehow are of interest to the business.

Note the difference between an entity and an instance of an entity.

More on Entities

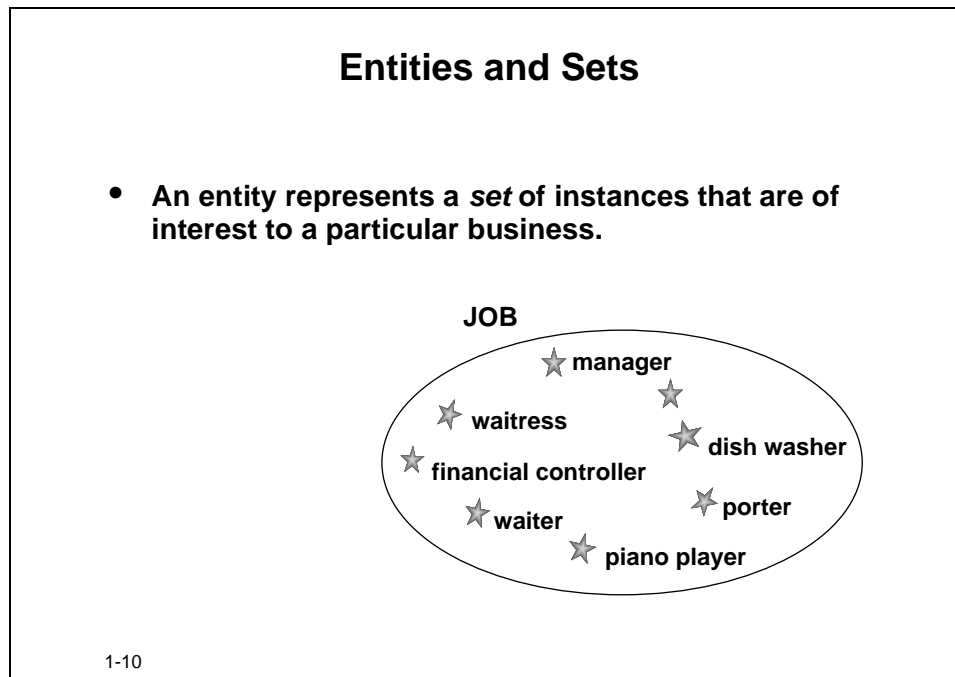
Entities and Instances	
PERSON	Mahatma Gandhi
PRODUCT	2.5 x 35 mm copper nail
PRODUCT TYPE	nail
EMPLOYMENT CONTRACT	my previous contract
JOB	violinist
SKILL LEVEL	fluent
TICKET RESERVATION	tonight: Hamlet in the Royal
PURCHASE	the CD I bought yesterday
ELECTION	for parliament next fall
PRINTER PREFERENCE	...
DOCUMENT VERSION	...

1-9

The illustration shows examples of entities and examples of instances of those entities.
Note:

- There are many entities.
- Some entities have many instances, some have only a few.
- Entities can be:
 - Tangible, like PERSON or PRODUCT.
 - Non-tangible, like REQUIRED SKILL LEVEL.
 - An event, like ELECTION.
- An instance of one entity may be an entity in its own right: the instance “violinist” of entity JOB could be the name of another entity with instances like “David Oistrach”, “Kyung-Wha Chung.”

Entities and Sets



You can regard entities as sets. The illustration shows a set JOB and the set shows some of its instances. At the end of the entity modeling process entities are transformed into tables; the rows of those tables represent an individual instance. During entity modeling you look for properties and rules that are true for the whole set. Often you can decide on the rules by thinking about example instances. The following lessons contain many examples of this.

Set Theory

Entity relationship modeling and the theory of relational databases are both based on a sound mathematical theory, that is, set theory.

Attributes

Attribute

- Also represents something of significance to the business
- Is a *single valued* property detail of an entity
- Is a specific piece of information that:
 - Describes
 - Quantifies
 - Qualifies
 - Classifies
 - Specifies an entity.

1-11

What is an Attribute?

An attribute is a piece of information that in some way describes an entity. An attribute is a property of the entity, a small detail about the entity.

Entities Have Attributes

For now, assume that all entities have at least one attribute. Later, you discover exceptions to this assumption. The attribute describes, quantifies, qualifies, classifies, and specifies an entity. Usually, there are many attributes for an entity, but again, we are only interested in those attributes that are of importance to the business.

Values and Data Types

Attributes have values. An attribute value can be a number, a character string, a date, an image, a sound, and even more. These are called data types or formats. Usually the values for a particular attribute of the instances of an entity all have the same data type. Every attribute has a data type.

Attribute is Single Valued

An attribute for an entity must be single valued. In more precise terms, an entity instance can have only one value for that attribute at any point in time. This is the most important characteristic of an attribute.

The attribute value, however, may change over time.

Attribute Examples

Attribute Examples	
Entity	Attribute
EMPLOYEE	Family Name, Age, Shoe Size, Town of Residence, Email, ...
CAR	Model, Weight, Catalog Price, ...
ORDER	Order Date, Ship Date, ...
JOB	Title, Description, ...
TRANSACTION	Amount, Transaction Date, ...
EMPLOYMENT CONTRACT	Start Date, Salary, ...

1-12

Note:

- Attribute Town of Residence for EMPLOYEE is an example of an attribute that is quite likely to change, but is probably single valued at any point in time.
- Attribute Shoe Size may seem to be of no importance, but that depends on the business: if the business supplies industrial clothing to its employees, this may be a very sensible attribute to take.
- Attribute Family Name may not seem to be single-valued for someone with a double name. This double name, however, can be regarded as a single string of characters that forms just one name.

Volatile Attributes

Some attributes are volatile (unstable). An example is the attribute Age. Always look for nonvolatile, stable, attributes. If there is a choice, use the nonvolatile one. For example, use the attribute Birth Date instead of Age.

Relationships

Relationships

- Also represent something of significance to the business
- Express how entities are mutually *related*
- Always exist between *two* entities (or one entity *twice*)
- Always have two perspectives
- Are named at both ends

1-13

Entities usually have relationships. Here are some examples.

Relationship Examples

EMPLOYEES *have* **JOB**
JOB *are held by* **EMPLOYEES**

PRODUCTS *are classified by a* **PRODUCT TYPE**
PRODUCT TYPE *is a classification for a* **PRODUCT**

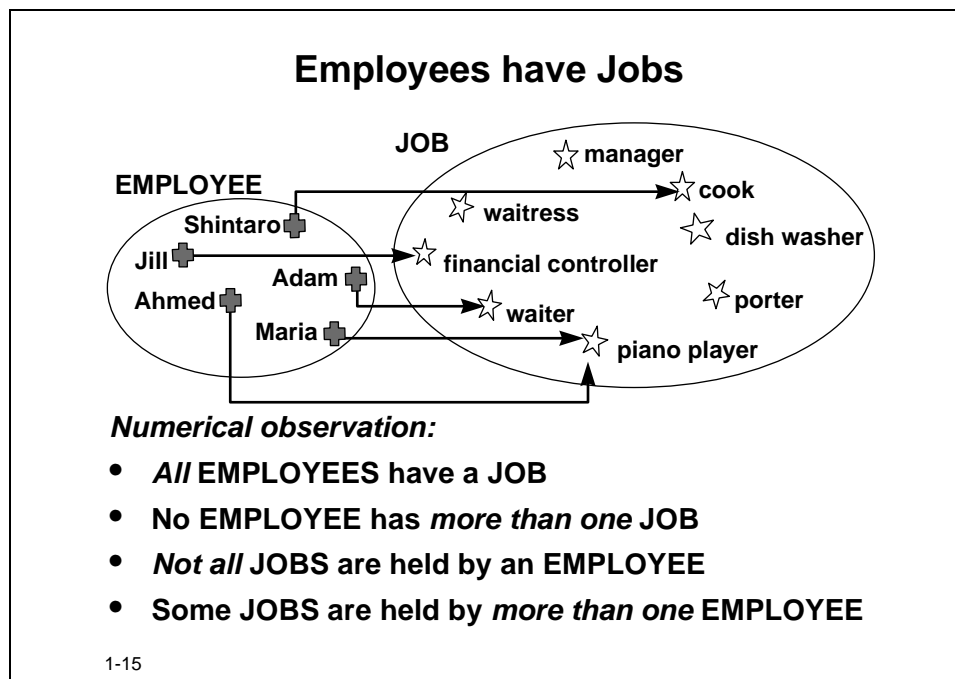
PEOPLE *make* **TICKET RESERVATIONS**
TICKET RESERVATIONS *are made by* **PEOPLE**

1-14

A relationship connects two entities. A relationship represents a significant dependency of two entities—always two entities.

A particular relationship can be worded in many ways: An EMPLOYEE *has* a JOB, or an EMPLOYEE *performs* a JOB, or an EMPLOYEE *holds* a JOB.

An EMPLOYEE *applies* for a JOB expresses a different relationship. Note that this example shows that two entities can have more than one relationship.



Based on what you know about instances of the entities, you can decide on four questions:

- Must *every* employee have a job?
In other words, is this a *mandatory* or *optional* relationship for an employee?
 - Can employees have *more than one* job?
- and
- Must *every* job be done by an employee?
In other words, is this a *mandatory* or *optional* relationship for a job?
 - Can a job be done by *more than one* employee?

Later on we will see why these questions are important and why (and how) the answers have an impact on the table design.

Entity Relationship Models and Diagrams

An Entity Relationship Model (ER Model) is a list of all entities and attributes as well as all relationships between the entities that are of importance. The model also provides background information such as entity descriptions, data types and constraints. The model does not necessarily include a picture, but usually a diagram of the model is very valuable.

An Entity Relationship Diagram (ER Diagram) is a picture, a representation of the model or of a part of the model. Usually one model is represented in several diagrams, showing different business perspectives.

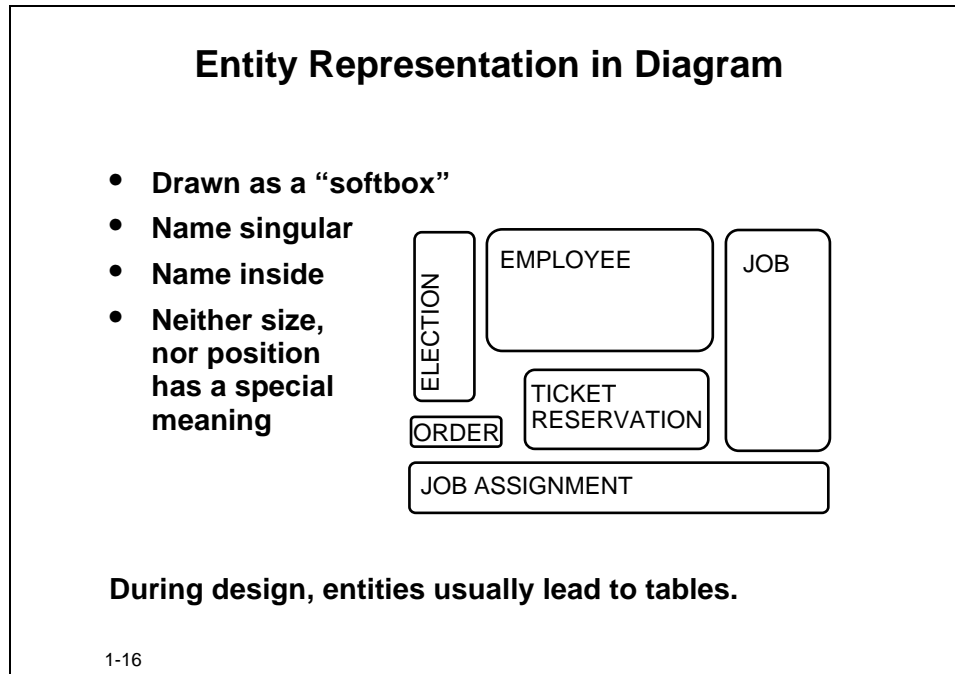
Graphical Elements

Entity Relationship diagramming uses a number of graphical elements. These are discussed in the next pages.

Unfortunately, there is no ISO standard representation of ER diagrams. Oracle has its own convention. In this course we use the Oracle diagramming technique, which is built into the Oracle Designer tool.

Representation

Entity



In an ER diagram entities are drawn as soft boxes with the entity name inside. Borders of the entity boxes never cross each other. Entity boxes are always drawn upright.

Throughout this book, entity names are printed in capitals. Entity names are preferably in the singular form; you will find that diagrams are easier to read this way.

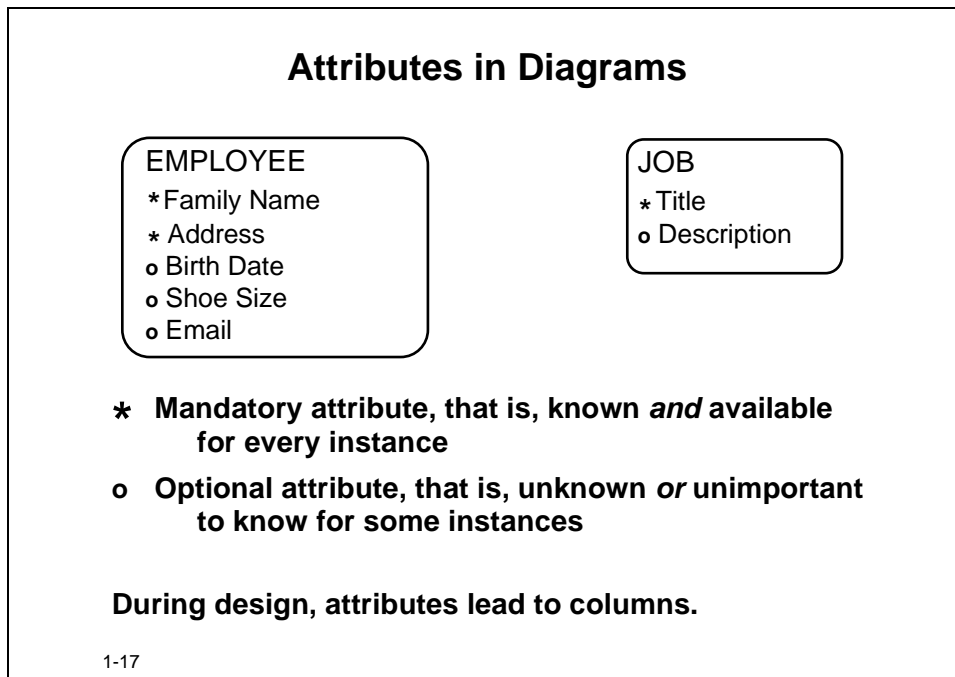
Box Size

Neither the size of an entity, nor its position, has a special meaning. However, a reader might construe a larger entity to be of more importance than a smaller one.

Where Entities Lead

During the design for a relational database, an entity usually leads to a table.

Attribute Representation



Attributes are listed within the entity box. They may be preceded by a * or an ^o. These symbols mean that the attribute is mandatory or optional, respectively. Throughout this book attributes are printed in Initial Capital format.

*** Mandatory:** It is realistic to assume that for every instance of the entity the attribute value is known and available when the entity instance is recorded and that there is a business need to record the value.

o Optional: The value of the attribute for an instance of the entity may be unknown or unavailable when that instance is recorded or the value may be known but of no importance.

Not all attributes of an entity need to be present in the diagram, but all attributes must be known before making the table design. Often only a few attributes are shown in a diagram, for reasons of clarity and readability. Usually you choose those attributes that help understanding of what the entity is about and which more or less “define” the entity.

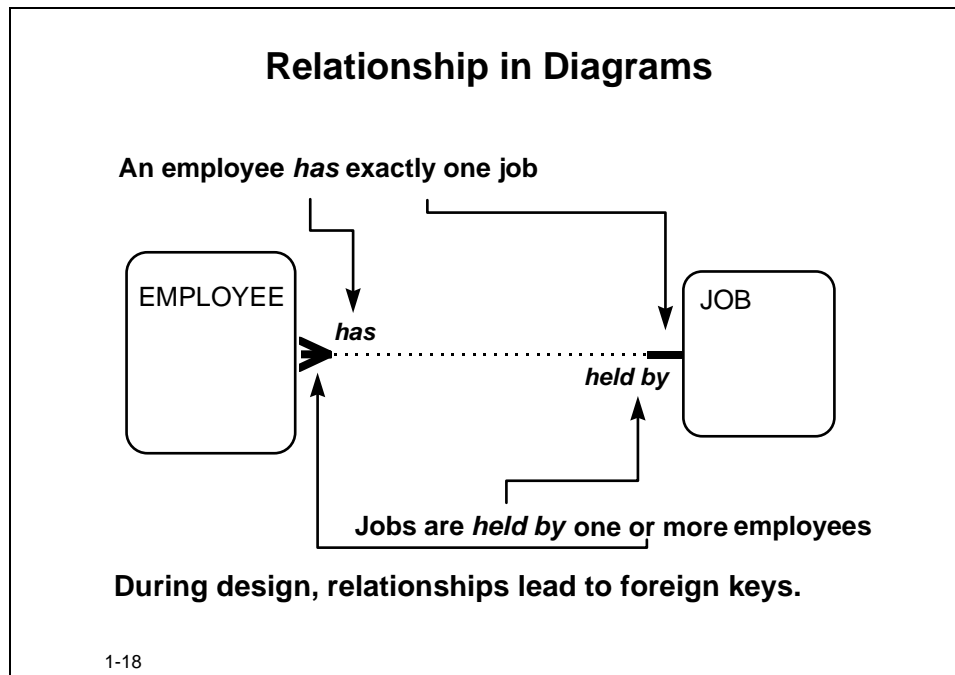
Where Attributes Lead

During design an attribute usually leads to a column. A mandatory attribute leads to a *not null* column.

Relationship Representation

Relationships are represented by a line, connecting the entities. The name of the relationship, from either perspective, is printed near the starting point of the relationship line.

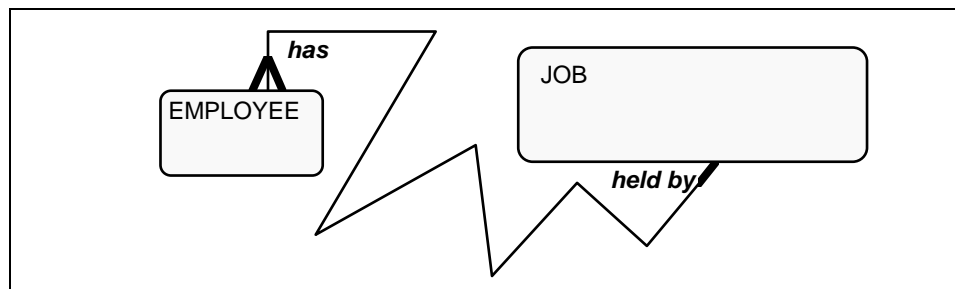
The shape of the end of the relationship line represents the degree of the relationship. This is either *one* or *many*. *One* means exactly one; *many* means one or more.



In the above example, it is assumed that JOBS are held by *one or more* EMPLOYEES. This is shown by the tripod (or crow's foot), at EMPLOYEE.

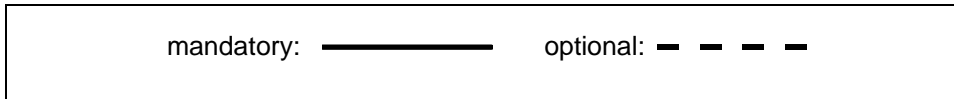
An EMPLOYEE, on the other hand, is assumed here to have *exactly one* JOB. This is represented by the single line at JOB.

The relationship line may be straight, but may also be curved; curves have no special meaning, nor does the position of the starting point of the relationship line. The diagram below represents exactly the same model, but arguably less clearly.



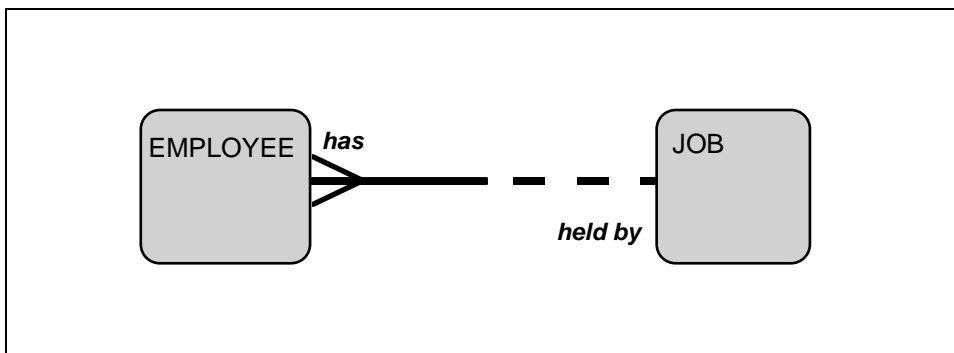
Mandatory and Optional Relationships

Relationships can be mandatory or optional, in the same way as attributes. Mandatory relationships are drawn as a solid line; optional relationships as dotted lines.

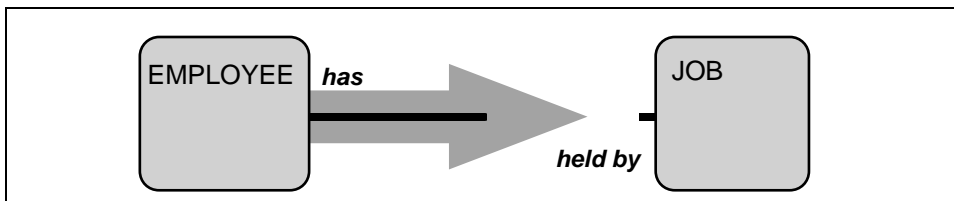


Relationship and Relationship Ends

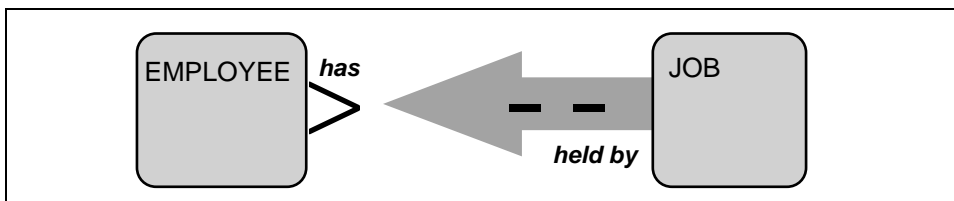
Here, the relationship between EMPLOYEE and JOB is modeled using the optional relationship end and mandatory relationship end notation.



When you read the relationship, imagine it split into two perspectives:

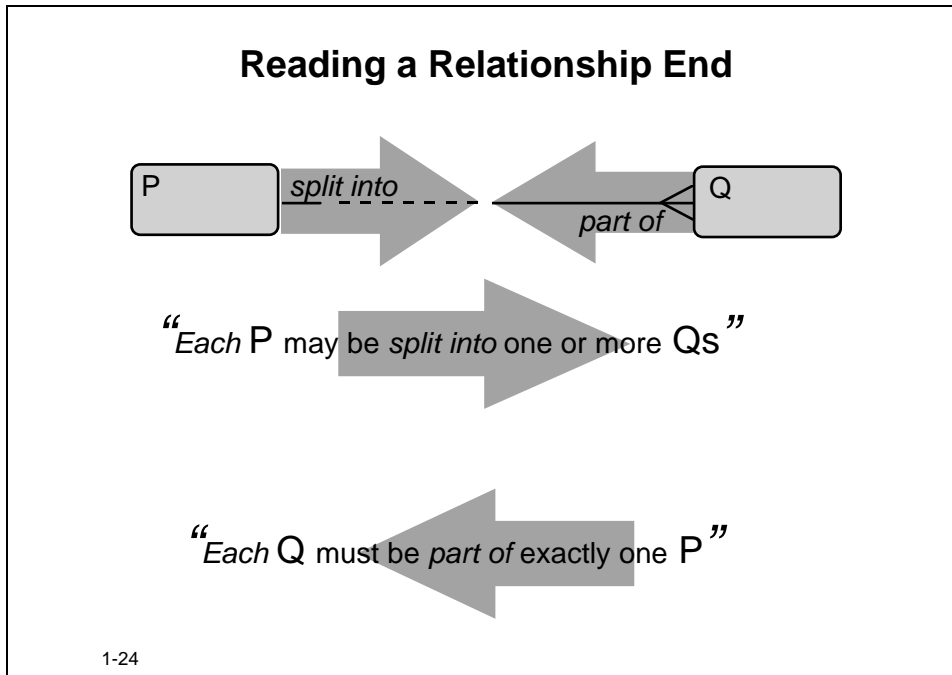


Every EMPLOYEE has exactly one JOB or, alternatively:
An EMPLOYEE *must have* exactly one JOB.



A JOB *may* be held by one or more EMPLOYEES.

Reading a Relationship End



A relationship from **entity1** to **entity2** must be read:

Each entity1 { must be | may be }
relationship_name
{ one or more | exactly one } entity2

Where Relationships Lead

During design relationships lead to foreign keys and foreign key columns. An optional relationship leads to non mandatory foreign key columns.

Relationship Name in the Diagram

Throughout this book relationship names in the diagrams are printed in lower case italics.

For reasons of space and readability of the diagrams in this book, relationship names are sometimes kept very short, and sometimes only a preposition is used.

Data and Functionality

Functions Drive Data

- **Business functions are always present.**
 - **Explicit**
 - **Assumed**
- **Business functions need data.**
- **An entity, attribute, or relationship may be modeled because:**
 - **It is used by a business function.**
 - **The business need may arise in the near future.**

1-25

Functions Drive the Conceptual Data Model

Although this course does not cover the method of function modeling, functions are present at any time, in any discussion on a conceptual data model. You cannot talk about, nor judge a conceptual data model without knowing or assuming the desired functionality of the future system.









Often a conceptual data model discussion may seem to be about the data structure but actually is about functionality, usually unclear or undetermined pieces of functionality. The language used is that of the conceptual data model, the representation used is that of the entity relationship diagram, but the discussion in fact is about functionality.

Functions drive the conceptual data model. The question “Do we need to take Shoe Size for an employee?” can only be answered by answering positively the question “Is there a business function that needs it?”

Consider the conceptual data model as the shadow of the functions of a system.

Most of the time during this course, functionality is only briefly sketched, or merely assumed, to prevent you from reading page after page of functional descriptions.

Types of Information

Weather Forecast			
January 26			
København		1/-5	➤ 3
Bremen		0/-3	➤ 4
Berlin		3/-1	➤ 3
München		5/-3	➤ 3
Amsterdam		8/3	➤ 4
Bruxelles		4/0	➤ 2
Paris		4/1	➤ 3
Bordeaux		7/2	➤ 3

1-26

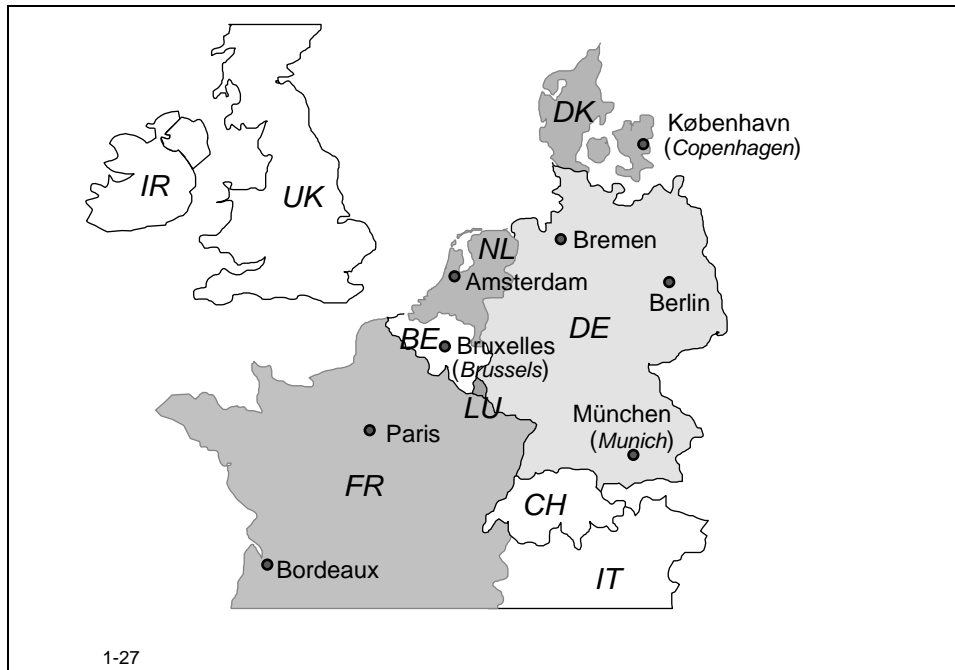
What Information is Available?

The illustration shows a piece of a weather forecast torn from a European newspaper, showing various types of information. What are the types of information? One of the first things you will see are, for example, “København”, “Bremen”. These are cities, or more precisely, names of cities. The little drawings represent the type of weather; these drawings are icons. The next columns are temperatures, probably maximum and minimum; the arrows indicate wind direction and the number next to it is the wind force. Then there is a date on top which is the forecast date. Therefore we have:

- City
- Name of the city (such as “København”)
- Weather type (such as “cloudy with rain”)
- Icon of the weather type
- Minimum temperature
- Maximum temperature
- Wind direction arrow
- Wind force
- Forecast date

Is this all?

No, you can find out even more information. To do this you have to have some “business” knowledge. In this case it is geographical knowledge.



You may notice that the cities in the weather forecast are not printed in a random order. The German cities (Bremen, Berlin and Munchen) are grouped together, just as the French cities are. Moreover, the cities are not ordered alphabetically by name but seem to be ordered North-South. Apparently this report “knows” something to facilitate the grouping and sorting. This could be:

- Country of the city
- Geographical position of the city
- and maybe even
- Geographical position of the country

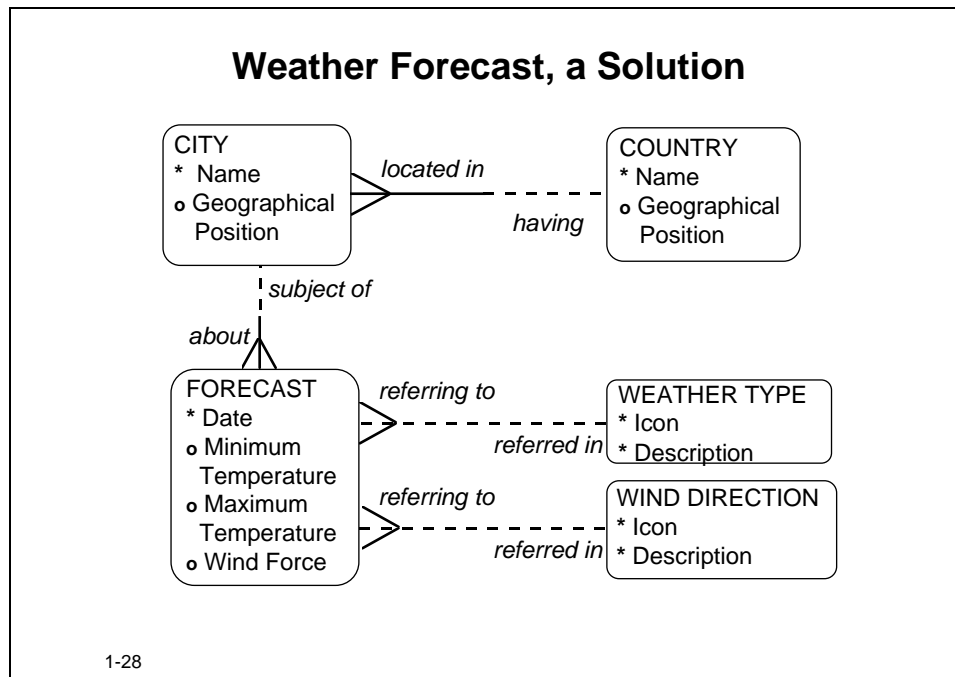
Next Step

Try to identify which of the above types of information is probably an entity, which is an attribute and which is a relationship.

City and Country are easy. These are entities, both with, at least, attribute Name and Geographical Position. Weather Type could also be an entity as there is an attribute available: Icon. For the same reason there could be an entity Wind Direction. Now, where does this leave the temperatures and forecast date? These cannot be attributes of City as the forecast date is not single value for a City: there can be many forecast dates for a city. This is how you discover that there is still one entity missing, such as Forecast, with attributes Date, Minimum and Maximum Temperature, Wind Force.

There are likely to be relationships between:

- COUNTRY and CITY
- CITY and FORECAST
- FORECAST and WEATHER TYPE
- FORECAST and WIND DIRECTION.

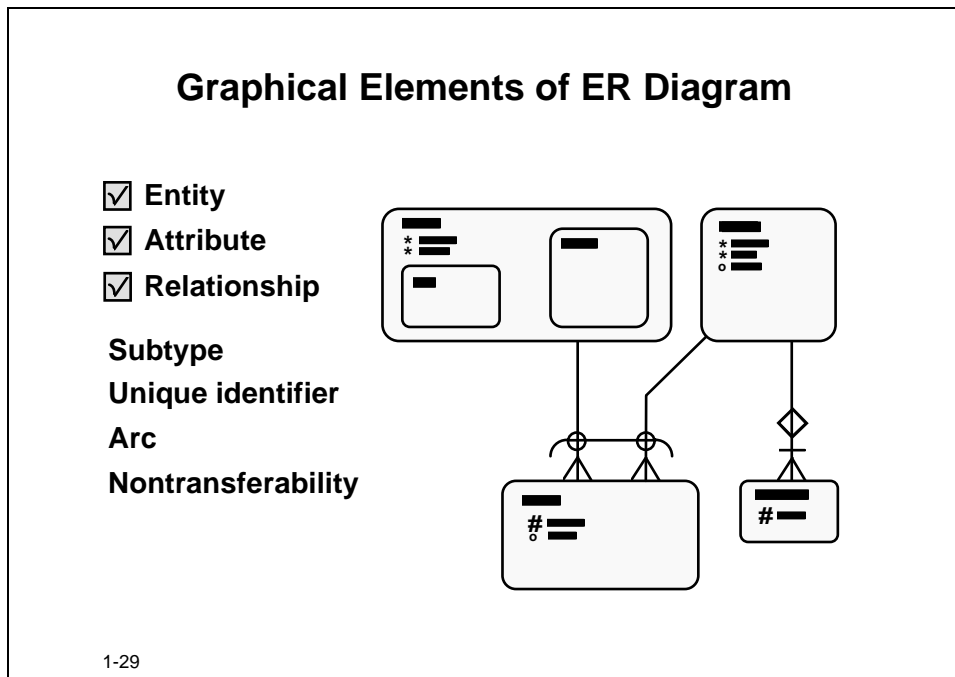


In this entity relationship diagram some assumptions are made about the relationships:

- Every FORECAST must be about one CITY, and not all CITIES must be in a FORECAST—but may be in many
- Every CITY is located in a COUNTRY, and every COUNTRY has one or more CITIES
- A FORECAST must not always contain a WEATHER TYPE, and not all WEATHER TYPES are in a FORECAST—but may be in many
- A FORECAST must not always contain a WIND DIRECTION, and not all WIND DIRECTIONS are in a FORECAST—but may be in many

The rationale behind these assumptions is that we consider an incomplete FORECAST still to be a FORECAST, unless we do not know the date or the CITY the FORECAST refers to.

Other Graphical Elements



The illustration shows all graphical elements you can encounter in a ER diagram. You saw earlier how to represent an entity, an attribute, and a relationship.

The lessons following this one discuss the remaining four types of elements:

- Subtype, represented as an entity within the boundary of another entity
- Unique identifier, represented as a # in front of an attribute or as a bar across a relationship line
- Arc, represented as an arc-shaped line across two or more relationship lines
- Nontransferability symbol, represented as a diamond across a relationship line

Limited Set of Graphical Elements

As you can see, the set of graphical elements in ER diagramming is very limited. The complexity of ER modeling is clearly not in the representation. The main complexity of ER modeling lies in the understanding of the business, in the recognition of the entities that play a role in that business, the relevant attributes that describe the entities, and the relationships that connect them.

Summary

Conceptual models are created to model the functional and information needs of a business. These models may be based on the current needs but can also be a reflection of future needs. This course is about modeling the information needs. Functional needs cannot be ignored while modeling data, as these form the only legitimate basis for the data model. Ideally, the conceptual models are created free of any consideration of the possible technical problems during implementation. Consequently the model is only concerned with what the business does and needs and not with how it can be realized.

Summary

- **ER Modeling models information conceptually**
- **Based on functional business needs**
- **“What”, not “How”**
- **Diagrams provide easy means of communication**
- **Detailed, but not too much**

1-30

Entity Relationship modeling is a well-established technique for catching the information needs. The ER model forms the basis for the technical data model. Technical considerations take place at that level.

Entity Relationship diagrams provide an easy-to-read and relatively easy-to-create diagrammatic representation of the ER model. These diagrams initially form the foundation for the discussion of business needs. Later they provide the best possible map of a future system.

The diagrams show a fair amount of detail, but are not too detailed to become cluttered.

Practice 1—1: Instance or Entity

Goal

The goal of this practice is to learn to make a distinction between an entity, an attribute, and an instance of an entity.

Your Assignment

List which of the following concepts you think is an Entity, Attribute, or Instance. If you mark one as an entity, then give an example instance. If you mark one as an attribute or instance, give an entity. For the last three rows, find a concept that fits.

Practice: Instance or Entity?

Concept	E/A/I?	Example Instance or Entity
PRESIDENT		
ELLA FITZGERALD		
DOG		
ANIMAL		
HEIGHT		
	E	CAR
	A	CAR
	I	CAR

1-32

Practice 1—2: Guest

Goal

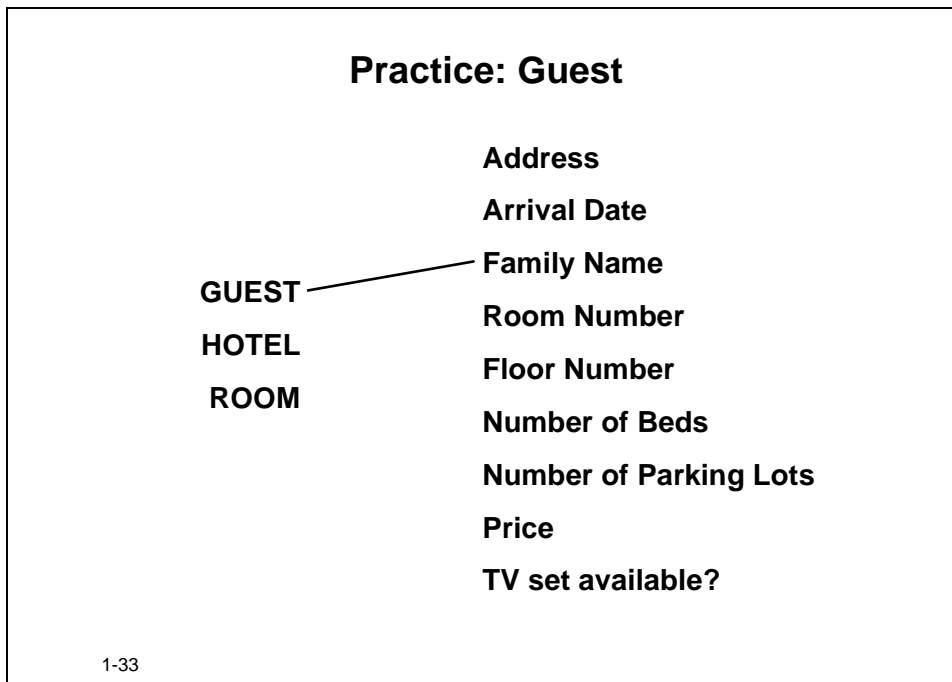
The goal of this practice is to recognize attributes for an entity.

Scenario

On the left side of the illustration are three entities that play a role in a hotel environment: GUEST, HOTEL, and ROOM. On the right is a choice of attributes.

Your Assignment

Draw a line between the attribute and the entity or entities it describes.



Practice 1—3: Reading

Goal

The goal of this practice is to read a relationship.

Your Assignment

Which text corresponds to the diagram?

Practice: Reading

```
graph LR; EMPLOYEE -- "assigned to" --- DEPARTMENT; DEPARTMENT -- "responsible for" --- EMPLOYEE;
```

A Each EMPLOYEE may be assigned to one or more DEPARTMENTS
Each DEPARTMENT must be responsible for one or more EMPLOYEES

B Each EMPLOYEE must be assigned to one or more DEPARTMENTS
Each DEPARTMENT may be responsible for one or more EMPLOYEES

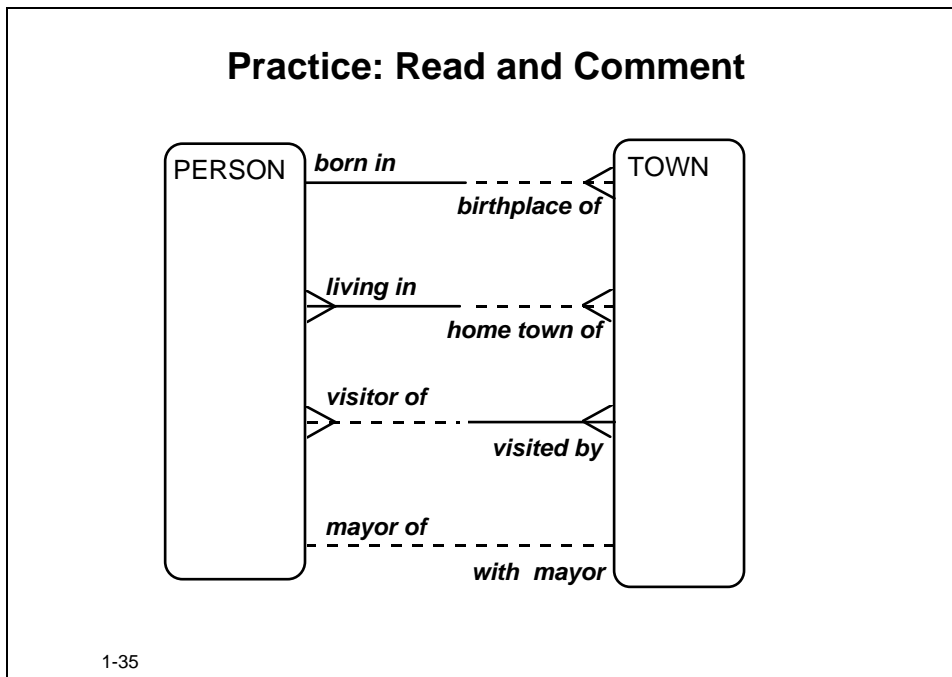
C Each EMPLOYEE must be assigned to exactly one DEPARTMENT
Each DEPARTMENT may be responsible for exactly one EMPLOYEE

1-34

Practice 1—4: Read and Comment

Your Assignment

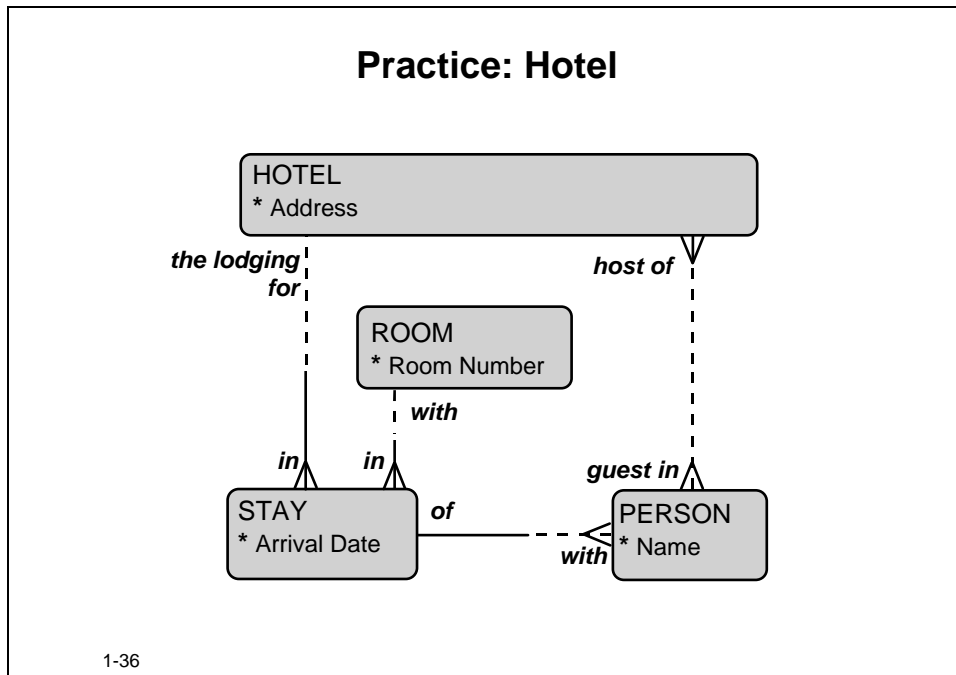
- 1 Read each of the relationships in the model presented here.
- 2 Next, comment on the relationship you just read. Use your knowledge of people and towns.



Practice 1—5: Hotel

Your Assignment

- 1 Comment on the relationships of the model presented here.



- 2 Make up two more possible relationships between **PERSON** and **HOTEL** that might be of some use for the hotel business.

Practice 1—6: Recipe

Goal

The goal of this practice is to discover the various types of information that are present in a given source of information.

Scenario

You work as an analyst for a publishing company that wants to make recipes available on the Web. It wants the public to be able to search for recipes in a very easy way. Your ideas about easy ways are highly esteemed.

Your Assignment

- 1 Analyze the example page from Ralph's famous Raving Recipes book and list as many different types of information that you can find that seem important.

Ralph's Raving Recipes

Soups	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> Açorda alentejana bread soup from Portugal </div> <div style="display: flex;"> <div style="width: 30%; background-color: #d3d3d3; padding: 5px; margin-right: 10px;"> vegetarian 15 min easy </div> <div style="padding: 10px;"> for 4 persons: 1 onion 4 cloves of garlic 1 red pepper 1 liter of vegetable broth 4 tablespoons of olive oil 4 fresh eggs 1 handful of parsley or coriander salt, pepper 9-12 slices of (old) bread </div> </div> <div style="display: flex; margin-top: 10px;"> <div style="width: 30%; background-color: #d3d3d3; padding: 5px;"> preparation </div> <div style="padding: 10px;"> Cut the onion into small pieces and fry together with the garlic. Wash the red pepper, cut it in half, remove the seeds and fry it for at least 15 </div> </div>
--------------	---

page 127

1-37

- 2 Group the various types of information into entities and attributes.
- 3 Name the relationships you discover and draw a diagram.

Entities and Attributes in Detail

Introduction

Lesson Aim

This lesson provides you with a detailed discussion about entities and attributes and how you can track these in various sources of information. The lesson looks at the evolution of an entity definition and the concept of subtype and supertype entity. The lesson also introduces the imaginary business of ElectronicMail Inc. which is used in many examples throughout this book.

Overview

- **Data compared to information**
- **Entities and how to track them down**
- **Attributes**
- **Subtypes and supertypes**

2-2

Topic	See Page
Introduction	2
Data Compared to Information	4
Data	5
Tracking Entities	7
Electronic Mail Example	9
Evolution of an Entity Definition	11
Functionality	13
Tracking Attributes	14
Subtypes and Supertypes	17
Summary	20
Practice 2—1: Books	21

Topic	See Page
Practice 2—2: Moonlight	22
Practice 2—3: Shops	23
Practice 2—4: Subtypes	24
Practice 2—5: Schedule	25
Practice 2—6: Address	26

Objectives

At the end of this lesson, you should be able to do the following:

- Track entities from various sources
- Track attributes from various sources
- Decide when you should model a piece of information as an entity or an attribute
- Model subtypes and supertypes

Data Compared to Information

Data Compared to Information

- **Data**
 - *Facts given from which other facts may be inferred*
 - *Raw material***Example: Telephone Directory**

- **Information**
 - *Knowledge, intelligence***Example: Telephone number of florist**

2-3

The words *data* and *information* are often used as if they are synonyms. Nevertheless, they have a different meaning.

Data: Raw material, from which you can draw conclusions. Facts from which you can infer new facts. A typical example is a telephone directory. This is a huge collection of facts with some internal structure.

Information: Knowledge, intelligence, a particular piece of data with a special meaning or function. Often information leads to data. In reverse, information is often the result of the deriving process from data—this may be a particular piece of data. If data is structured in some way, this is very helpful in the process of finding information. To expand the telephone directory data example, information is the telephone number of your dentist or the home address of a colleague.

Data

Data

Data~

- ***Modeling, Conceptual***
Structuring data concepts into logical, coherent, and mutually related groups
- ***Modeling, Physical***
Modeling the structure of the (future) physical database
- ***Base***
A set of data, usually in a variety of formats, such as paper and electronically-based
- ***Warehouse***
A huge set of organized information

2-4

Conceptual Data Modeling

Conceptual data modeling is the examination of a business and business data in order to determine the structure of business information and the rules that govern it. This structure can later be used as the basis for the definition of the storage of the business data. Conceptual data modeling is independent of possible technical implementations. For that reason, a conceptual data model is relatively stable over longer periods of time, as businesses change, often only gradually, over a period of time. Conceptual Data modeling is also called Information Engineering.

Physical Data Modeling

Physical data modeling is concerned with implementation in a given technical software and hardware environment. The physical implementation is highly dependent on the current state of technology and is subject to change as available technologies rapidly change. A technical design made five years ago is likely to be quite outdated today.

By distinguishing between the conceptual and physical models, you separate the rather stable from the rather unstable parts of a design. This is true for both data models and functional specifications.

Database

A *database* is a set of data. The various parts of the data are usually available in different forms, such as paper and electronic-based. The electronic-based data may reside, for example, in spreadsheets, in all kinds of files, or in a regular data base. Today, relational databases are very common; but many older systems are still around. The older systems are mostly *hierarchical databases* and *network databases*. Systems of more recent date are *semantic databases* and *object oriented databases*.

Data Warehouse

A data warehouse is composed of data from multiple sources placed into one logical database. This data warehouse database, (or, more correctly, this database structure), is optimized for Online Analytical Processing (OLAP) actions.

Often a data warehouse contains summarized data from day-to-day transaction systems with additional information from other sources. An example is a phone company that tracks the traffic load for a routing system. The system does not store the individual telephone calls, but stores the data summarized by hour.

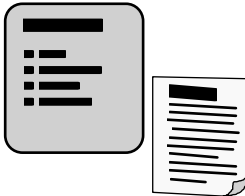
From a data analysis point of view a data warehouse is just a database, like any other, only with very specific and characteristic functional requirements.

Tracking Entities

The nouns in, for example, the texts, notes, brochures, and screens you see concerning a business often refer to entities, attributes of entities, or instances of entities.

Entities

- Give the entity a unique name
- Create a formal description of the entity
- Add a few attributes, if possible
- Be aware of homonyms
- Check entity names and descriptions regularly
- Avoid use of reserved words
- Remove relationship name from entity name



2-5

Naming an Entity Uniquely

First distinguish an entity by outlining the concept in your mind. Next, try to find a unique and clear name for an entity. This is not always easy as there are far more concepts than clear names. Use your imagination. Use a dictionary. Use a combination of words, use 'X' if necessary, but do not let the lack of a good name stop you from modeling. Good names evolve over time.

Check the names you used every now and then. The implicit definition of an entity may change during analysis, for instance, as a result of adding an attribute or changing the optionality of a relationship.

Creating a Formal Description

Create a formal description of the entity. This is usually not difficult and the writing helps clarify your thinking about what you are talking about. Check this description regularly. Sometimes concepts evolve during the modeling process. The definitions, of course, should follow that evolution.

Be Aware of Synonyms

In many business contexts one and the same concept is known under different names. Select one and mention the synonyms in the description: "...also known as ...".

Avoid Homonyms

Often in a business one word is used for different concepts. Sometimes even the same person will use the same word but with different meanings as you can see in the next example.

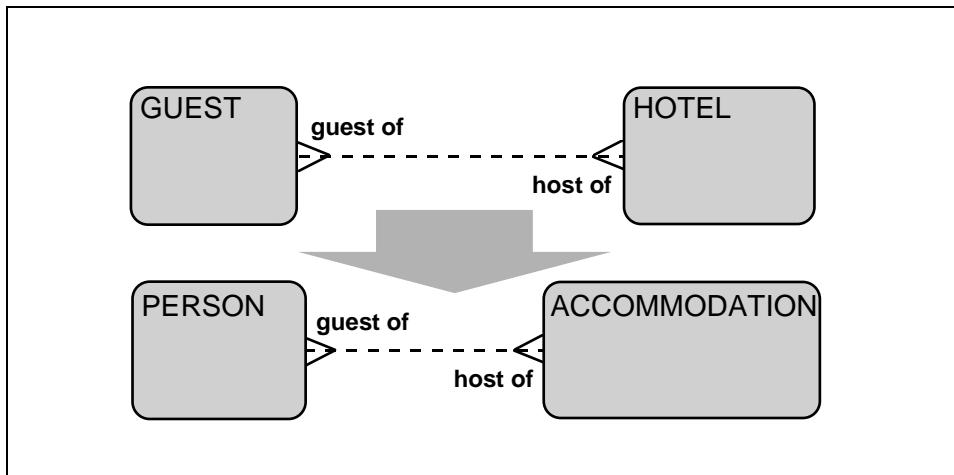
“The data modeling course you attend now was written in 1999 and requires modeling skills to teach.” In this sentence the word “course” refers to three different concepts: a course *event* (like the one you are attending today), a course *text* (which was written in 1999) and the course *type* (that apparently needs particular skills).

Avoid Reserved Words

Although you are free to use any name you want for an entity, try to avoid database and programming terms as entity names if possible. This may prevent naming problems and confusion later on in the design stage.

Remove Relationship Name from Entity Name

Often you can select entity names in a more or less generic way. In the example, both diagrams model the same context. In the first the “guest” aspect is part of the entity name as well as part of the relationship name.



The second model is more general in its naming. There a guest is seen as a PERSON playing the role of being a guest.

As a rule, if there is choice take the more general name. It allows, for example, for the addition of a second relationship between the same entities that shows, for example, *person is working for* or *is owning shares in* the accommodation. The first model would require new entities.

This subject is closely related to the concept of subtypes and roles. You find more on this later in this lesson and when we discuss Patterns.

Electronic Mail Example

In this course we investigate various business contexts. One is that of ElectronicMail, a company that supplies an e-mail service. Here is some background information.

Some Background Information

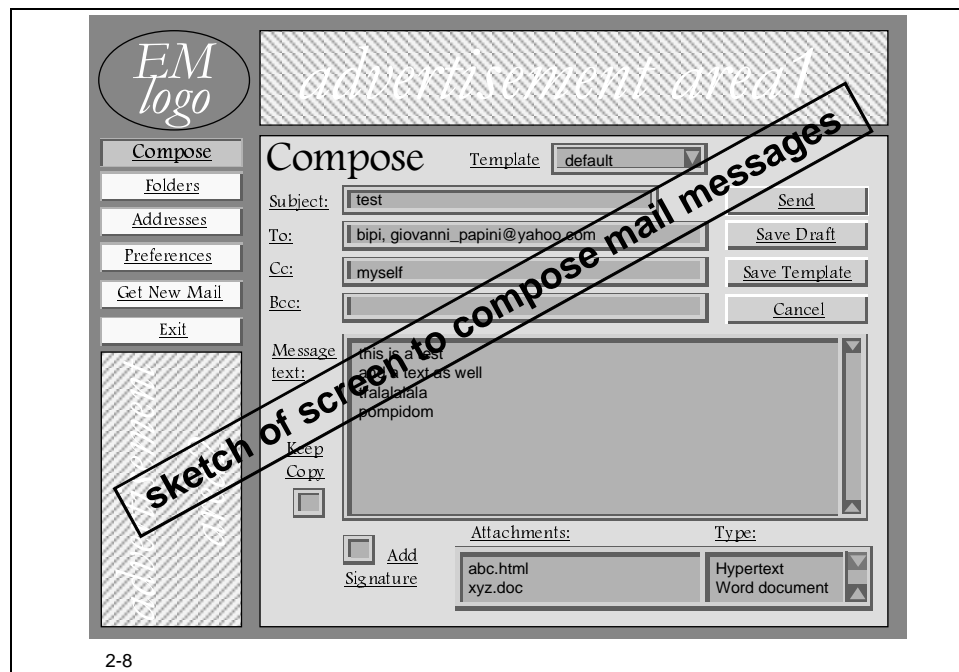
“ElectronicMail (EM) wants to provide an attractive and user- friendly Web-based e-mail system. Important concepts are *user* and *message*.

An EM user has a unique address of 30 characters at most and a password supplied by the person who set up the EM user. Who the person really is, we do not know, although we ask for some additional information, such as the name, country, birth date, line of business, and a few more things.

Users must be able to send and receive mail messages. A mail message is usually a piece of straight text. A message may have attached files. An attachment is a file, like a spreadsheet, that is sent and kept with the message, but not created with our software.

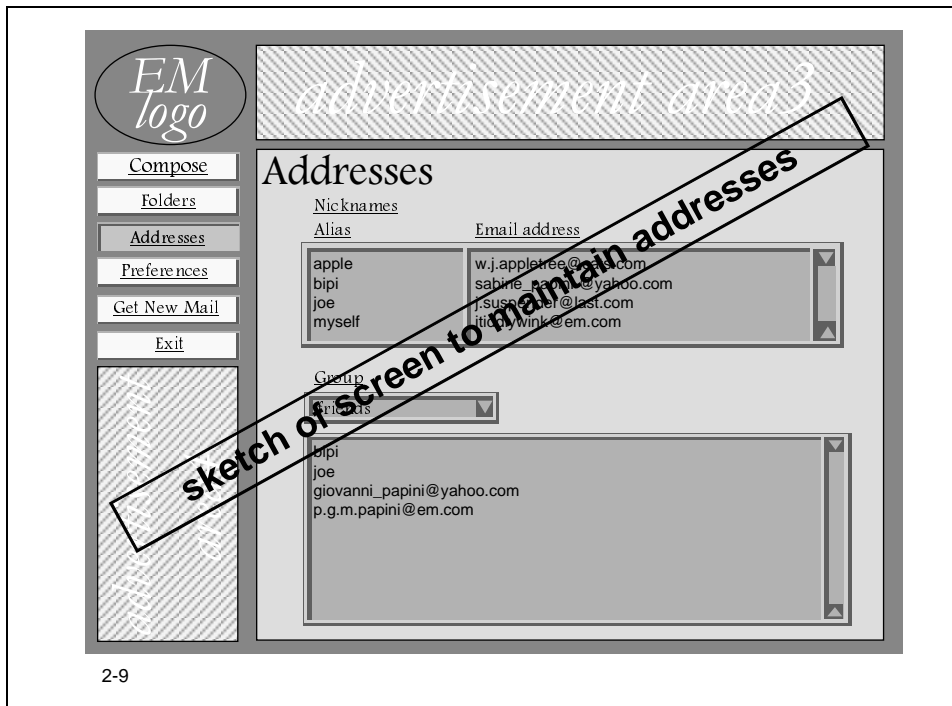
Messages are kept in folders. Every user has three folders to start with: Inbox, Outbox, and Wastebasket. Additional folders can be created by the user.”

2-7



2-8

The screenshots may give an idea of how the Compose a Mail Message screen and the Maintain Addresses screen will look like.



Some Desired Functionality

“Users of ElectronicMail must be able to address messages to a mail list, for example, a group of e-mail addresses. The system should only keep one copy of the message sent (to save database space) plus information about whom the message was sent to.

Users must be able to create templates for their messages. A template must be named and may contain everything a real message contains. A template may be used for new messages.

Users must be able to reply to a message. By replying the user creates a new message to which the old message is added.

Users must be able to create an alias for an e-mail address, to hide the often complex addresses behind an easy-to-remember nickname.”

2-10

Evolution of an Entity Definition

To illustrate the evolution of a concept, consider ElectronicMail's entity MESSAGE. The first intuitive description of this entity may be:

A message is a piece of text sent by a user.

Any user? Well, no.

A message is a piece of text sent by *an EM* user.

Must every message contain text? Should it not be possible to send a message that only transports an attachment, without additional text?

A message is a note that is sent by an EM user. A message does not necessarily contain text, nor a subject, etc.

And what about a message that comes from an external source and is *received* by an EM user? Should those not be kept as well?

A message is a note that is sent by an EM user or received by an EM user or both. A message does not necessarily contain text, nor a subject, etc.

Now suppose a message is sent by an EM user to an external e-mail address only. Suppose the EM user does not want to keep a copy of the mail message. In that case there is no need for the system to keep the message as there is no internal EM user that needs the message. In fact, it is not important at all to keep messages that were *sent* by a EM user; only those that were actually *received* by an EM user are of interest.

A message is a note that is *received* by an EM user. A message does not necessarily contain text, nor a subject, etc.

The thinking process shown here is typical for the change of a definition from the first idea to something that is much more well thought-out—though this does not mean that the definition is final.

Entity Life Cycle

It often helps to make things clear if you think about the *life cycle* of an entity. The life cycle refers to the functional steps of the entity. For example, how can the entity instance come into existence? How can it change? How does it disappear?

In case of entity MESSAGE the questions are:

- When does “something” become a message?
- When does a message change?
- When can a message be removed?

Creating a Message

When I type in some text in the *compose* screen, is that text a message? You will probably agree that it does not make much sense to consider it as a message until some fields are completed, such as the To or Subject field. The checks must take place after I hit the send key. Only after all checks have been made is the message born.

Removing a Message

When can the system remove a message? When a user hits the delete key? But what should the system do when there are other receivers of that same message? It is better to consider the deleting of a message as the signal to the system that you no longer need the right to read the message. When all users that did receive the same message have done this, then the message can be deleted. Apparently, for a message to exist it must have receivers that still need the message.

Changing a Message

Changing a message? As long as the text is not sent, it is no problem as it is not yet considered to be a message. Changing it after sending it? Changing something that is history? This cannot be done. Changing the text should lead to a new message.

Draft

What about a message that is not yet ready for sending? Suppose a user wants to finish a message at a later date. Is there a place for this? Do we want an unsent, or draft, message in the system? Is a DRAFT a special case of entity MESSAGE, or should we treat a DRAFT as a separate entity?

Template

What about the templates? A template is about everything a message can be, but a template is only used as a kind of stamp for a message. Templates are named, messages are not. Is TEMPLATE a special case of entity MESSAGE, or should we look upon it as a separate entity?

Functionality

In the previous evolution of the entity definition, the definition changes were invoked by thinking and rethinking the functionality of the system around messaging. This illustrates the statement made earlier: functions drive the conceptual data model.

Business Functions

“Users of ElectronicMail must be able to **address** messages to a mail list, for example, a group of e-mail addresses. The system should only keep one copy of the message sent (to save data base space) plus information about whom the message was sent to.

Users must be able to **create** templates for their messages. A template must be named and may contain everything a real message contains. A template may be used for new messages.

Users must be able to **reply** to a message. By replying the user **creates** a new message to which the old message is added.

Users must be able to **create** an alias for an e-mail address, to hide the often complex addresses behind an easy-to-remember nickname.”

2-11

The first idea of the functionality of a system, or desired functionality, can be derived from the verbs in, for example, descriptive texts and interview notes. In the above text the functionality is expressed at a high level, without much detail. Nevertheless, you can probably imagine more detailed functionality.

In this course functionality is always present, often implicitly assumed, sometimes in detail.

Tracking Attributes

An Attribute...

- Always answers “of what?”
- Is the property of entity, not of relationship
- Must be single valued
- Has format, for example:
 - Character string
 - Number
 - Date
 - Picture
 - Sound
- Is an elementary piece of data

2-13

As discussed earlier, the nouns in, for example, the texts, notes, brochures, and screens you see used in a business often refer to entities, attributes of entities, or instances of entities. You can usually easily recognize attributes by asking the questions “Of what?” and “Of what format?”. Attributes describe, quantify, qualify, classify, specify or give a status of the entity they belong to. We define an attribute as a property of an entity; this implies there is no concept of a standalone attribute.

In the background information text on ElectronicMail that is shown below, the first occurrence of the (probable) entities are capitalized, the attributes are boxed and instances are shown in italics.

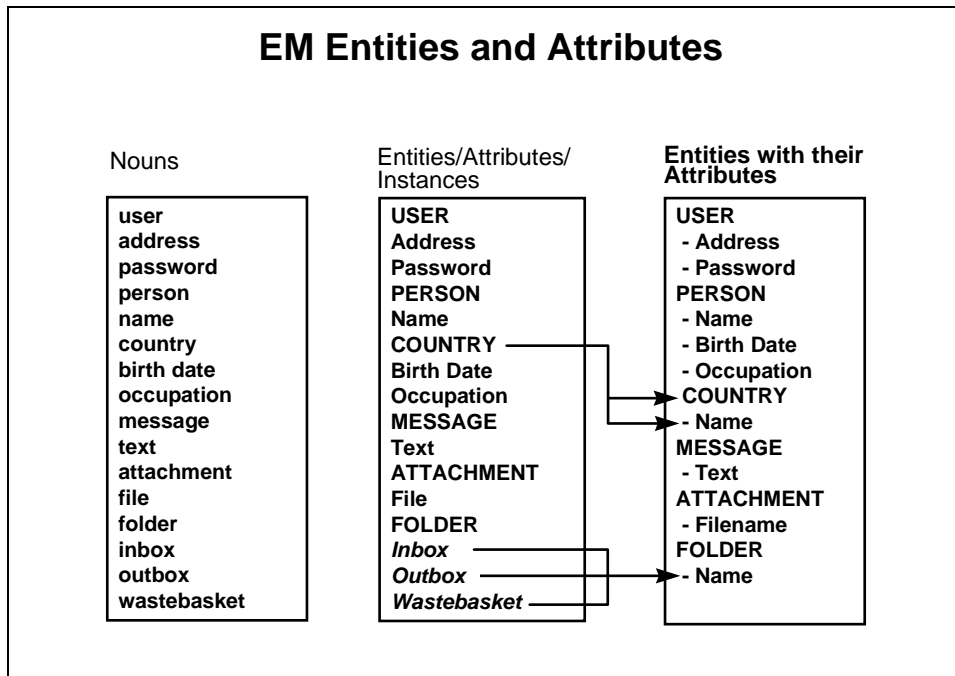
“ElectronicMail (EM) wants to provide an attractive and user friendly Web-based email system. Important concepts are *user* and *message*.

An EM USER has a unique address of 30 characters at most and a password supplied by the PERSON who set up the EM user. Who the person really is, we do not know, although we ask for some additional information, like the name, COUNTRY, birth date, line of business, and a few things more.

Users must be able to send and receive mail MESSAGES. A mail message is usually a piece of straight text. A message may have attached files. An ATTACHMENT is a file, like a *spreadsheet*, that is sent and kept with the message, but not created with our software.

Messages are kept in FOLDERS. Every user has three folders to start with: *Inbox*, *Outbox* and *Wastebasket*. Additional folders can be created by the user.”

List the types of information, distinguish the probable entities and attributes and group them. Add attributes, if necessary, (like Name of COUNTRY) in the example. Distill one or more attributes from the instances (like Name of FOLDER).



Naming Attributes

Attribute names become the candidate column names at a later stage. Column names must follow conventions. Try to name attributes avoiding the use of reserved words.

Do not use abbreviations, unless these were decided beforehand. Examples of frequently-used abbreviations are Id, No, Descr, Ind(icator).

Do *not* use attribute names like Amount, Value, Number. Always add an explanation of the meaning of the attribute name: Amount Paid, Estimated Value, Licence No.

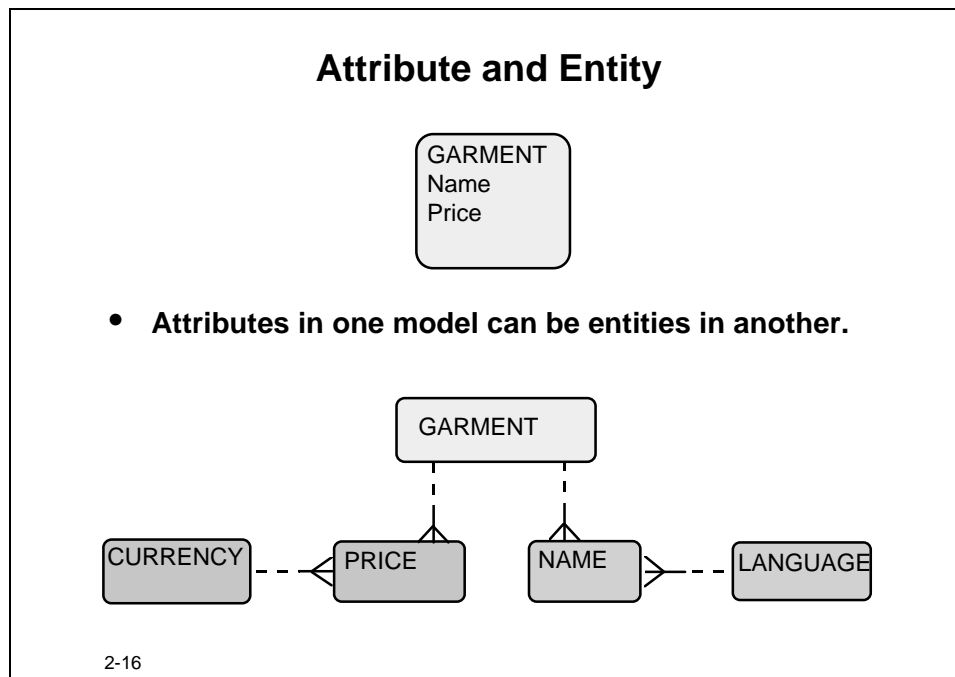
Always put frequently-used name components, such as “date” or “indicator”, of attribute names in the same position, for example, at the end—Start Date, Creation Date, and Purchase Date.

Do not use underscores in attribute names that consist of more than one word. Keep in mind that attribute names, like entity names, must be as clear and understandable as possible.

Entities Compared to Attributes

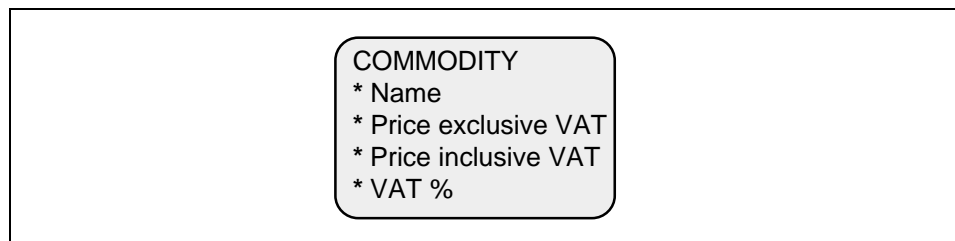
Sometimes a piece of information that is an attribute in one context is an entity in another context. This is purely specific to the business. A typical attribute, like Name, may need to be modeled as an entity. This happens, for example, when the model needs an extra dimension, such as the language. If product names must be kept in several languages and prices must be kept in various currencies, you may suddenly find one product has several names. For example: “This particular article of clothing is named ‘Acapulco swimming trunks’ in English, and ‘Akapulko Badehose’ in German.”

A commonly encountered dimension is *time*. This is discussed later.



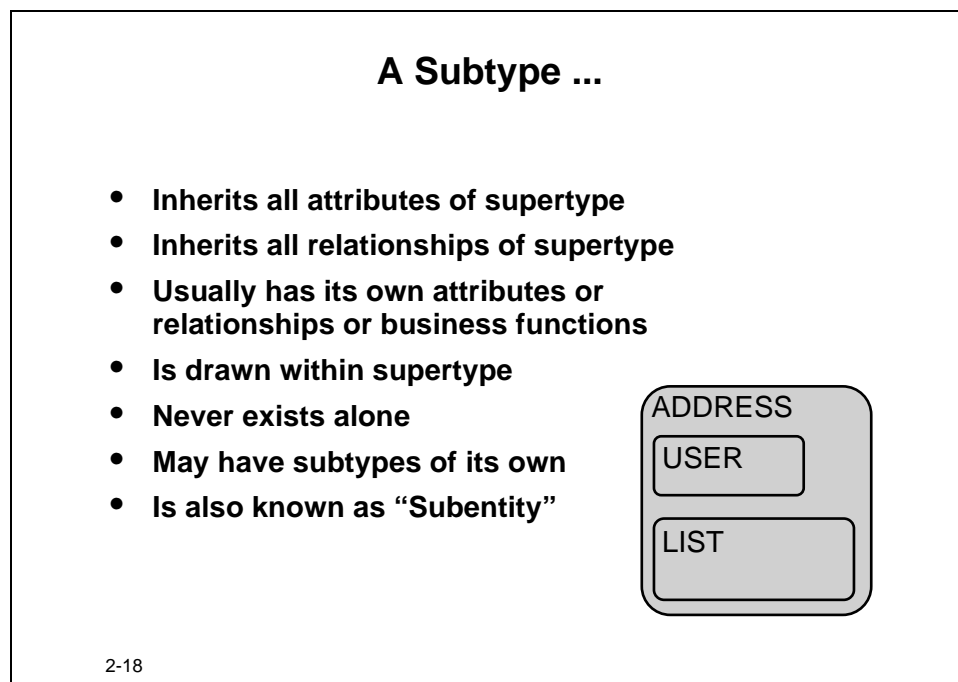
Redundancy

You should take special care to prevent using redundant attributes, that is, attribute values that can be derived from the values of others. An example is shown below. Using derivable information is typically a physical design decision. This is also true for audit type attributes such as Date Instance Created, and User Who Modified.



Subtypes and Supertypes

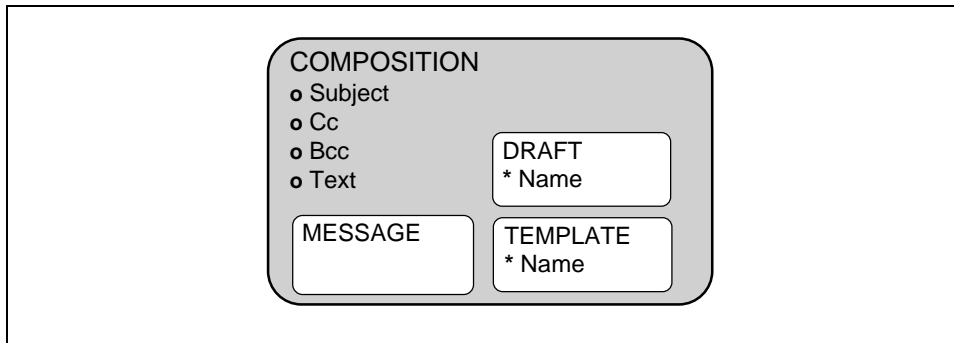
Sometimes it makes sense to subdivide an entity X into subtypes. This may be the case when a group of instances has special properties, such as attributes or relationships that only exist for that group, or a fixed value for one of the attributes, or when there is some functionality that only applies to the group. Such a group is called a subtype of X. Entity X is called the supertype as a consequence. Subtypes are also modeled when particular constraints apply to the subtype only. This is discussed further in the lesson on Constraints.



Subtypes have all properties of X and usually have additional ones. In the example, supertype ADDRESS is divided into two subtypes, USER and LIST. One thing USER and LIST have in common is an attribute NAME and the functional fact that they can both be used in the To field when writing a message.

Inheritance

In the next illustration, is a new entity, COMPOSITION, as a supertype of MESSAGE, DRAFT, and TEMPLATE. The subtypes have several attributes in common. These common attributes are listed at the supertype level. The same applies to relationships. Subtypes inherit all attributes and relationships of the supertype entity.



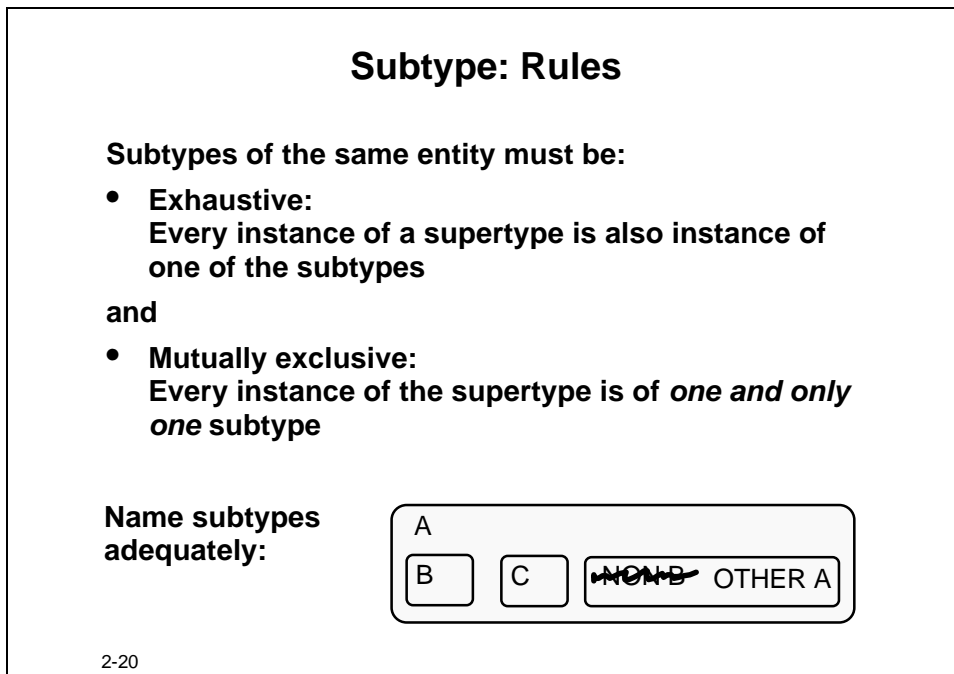
Read the diagram as:

Every MESSAGE (DRAFT, or TEMPLATE) is a COMPOSITION
and thus has attributes like Subject and Text. Conversely:

Every COMPOSITION is either a MESSAGE, a DRAFT, or a TEMPLATE

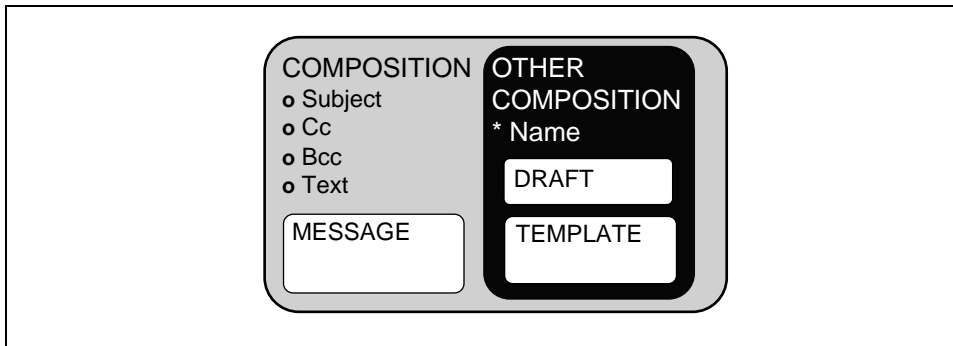
Always More Than One Subtype

Entity relationship modeling prescribes that when an ER model is complete subtypes never stand alone. In other words, if an entity has a subtype, there should always be at least a second subtype. This makes sense. What use would there be for distinguishing between an entity and the single subtype? This idea leads to the two subtype rules.



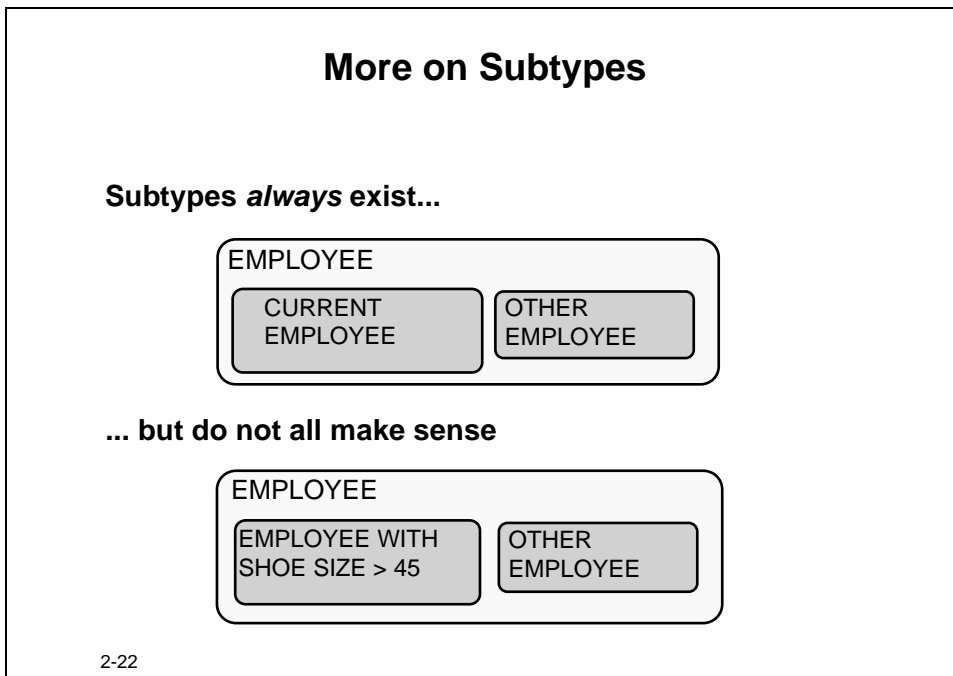
Nested Subtypes

You can nest Subtypes. For readability, you would not usually subtype to more than two levels, but there is no major reason not to do so. Reconsider the placement of the attributes and relationships after creating a new level.



Subtypes Always Exist

Every entity can always be subtyped. You can always make up a rule to subdivide the instances in groups, but that is not the issue. The reason for subtyping should always be that there is a business need to show similarities and differences at the same time.



2-22

Implementing Subtypes

You can implement subtype entities in various ways, for example, as separate tables or as a single table, based on the super entity.

Summary

Entities can often be recognized as nouns in texts that functionally describe a business. Entities can be tangible, intangible, and events. Subtypes of an entity share all attributes and relationships of that entity, but may have additional ones.

Summary

- **Entities**
 - **Nouns in texts**
 - **Tangible, intangible, events**
- **Attributes**
 - **Single-valued qualifiers of entities**
- **Subtypes**
 - **Inherit all attributes and relationships of supertype**
 - **May have their own attributes and relationships**

2-23

Attributes are single-valued elementary pieces of information that describe, qualify, quantify, classify, specify or give a status of the entity they belong to.

Most entities have attributes.

Every attribute can be promoted to a separate entity which is related to the entity the attribute initially belonged to. You must do this when you discover that the attribute is not single valued, for example, when names must be kept in multiple languages or values in multiple currencies.

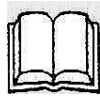
Practice 2—1: Books

Goal

The goal of this practice is to differentiate between various meanings of a word used in a text.

Your Assignment

- 1 In this text the word *book* is used with several meanings. These meanings are different entities in the context of a publishing company or a book reseller. Try to distinguish the various entities, all referred to as *book*. Give more adequate names for these entities and make up one or two attributes to distinguish them.

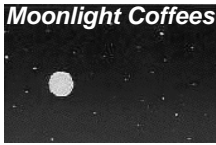


1. I have just finished writing a book. It's a novel about justice and power.
2. We have just published this book. The hard cover edition is available now.
3. Did you read that new book on Picasso? I did. It's great!
4. If you like you can borrow my book.
5. I have just started translating this book into Spanish. I use the modern English text as a basis and not the original, which is 16th century.
6. I ordered that book for my parents.
7. Yes, we have that book available. You should find it in Art books.
8. A second printing of the book *War and Peace* is very rare.
9. I think *My name is Asher Lev* is one of the best books ever written. Mine is autographed.
10. I want to write a book on entity relationship modeling when I retire.

2-25

- 2 Create an ER model based on the text. Put the most general entity at the top of your page and the most specific one at the bottom. Fit the others in between. Do not worry about the relationship names.

Practice 2—2: Moonlight



Scenario

You work as a contractor for Moonlight Coffees Inc. One of your colleagues, who is a business analyst, has prepared some documentation. Below you find an extract from the summary document.

Your Assignment

- 1 Make a list of about 15 different entities that you think are important for Moonlight Coffees. Use your imagination and common sense and, of course, use what you find in the summary that is printed below.



Summary

Moonlight Coffees is a fast growing chain of high quality coffee shops with currently over 500 shops in 12 countries of the world. Shops are located at first-class locations, such as major shopping, entertainment and business areas, airports, railway stations, museums. Moonlight Coffees has some 9,000 employees.

Products

All shops serve coffees, teas, soft drinks, and various kinds of pastries. Most shops sell nonfoods, like postcards and sometimes even theater tickets.

Financial

Shop management reports sales figures on a daily basis to Headquarters, in local currency. Moonlight uses an internal exchange rates list that is changed monthly. Since January 1, 1999, the European Community countries must report in Euros.

Stock

Moonlight Coffees is a public company; stock is traded at NASDAQ, ticker symbol MLTC. Employees can participate in a stock option plan.

2-26

- 2 Write a formal definition of the entity that represents:
 - The coffee shops.
 - The Moonlight employees.

Practice 2—3: Shops

Moonlight Coffees



Scenario

You work as a contractor for Moonlight Coffees. Your task is to create a conceptual data model for their business. You have collected all kinds of documents about Moonlight. Below is a page of a shop list.

Your Assignment

Use the information from the list as a basis for an ER model. Pay special attention to find all attributes.

Shop List

Moonlight Coffees



Shoplist, ordered to date opened

page 4

181 The Flight, JFK Airport terminal 2, New York, USA, 212.866.3410, Airport, 12-oct-97
 182 Hara, Kita Shinagawa, Tokyo, JP, 3581.3603/4, Museum, 25-oct-97
 183 Phillis, 25 Phillis Rd, Atlanta, USA, 405.867.3345, Shopping Centre, 1-nov-97
 184 JFK, JFK Airport terminal 4, New York, USA, 212.866.3766, Airport, 1-nov-97
 185 VanGogh, Museumplein 24, Amsterdam, NL, 76.87.345, Museum, 10-nov-97
 186 The Queen, 60 Victoria Street, London, UK, 203.75.756, Railway Station, 25-nov-97
 187 Wright Bros, JFK Airport terminal 1, New York, USA, 212.866.9852, Airport, 6-jan-98
 188 La Lune, 10 Mont Martre, Paris, FR, 445 145 20, Entertainment, 2-feb-98

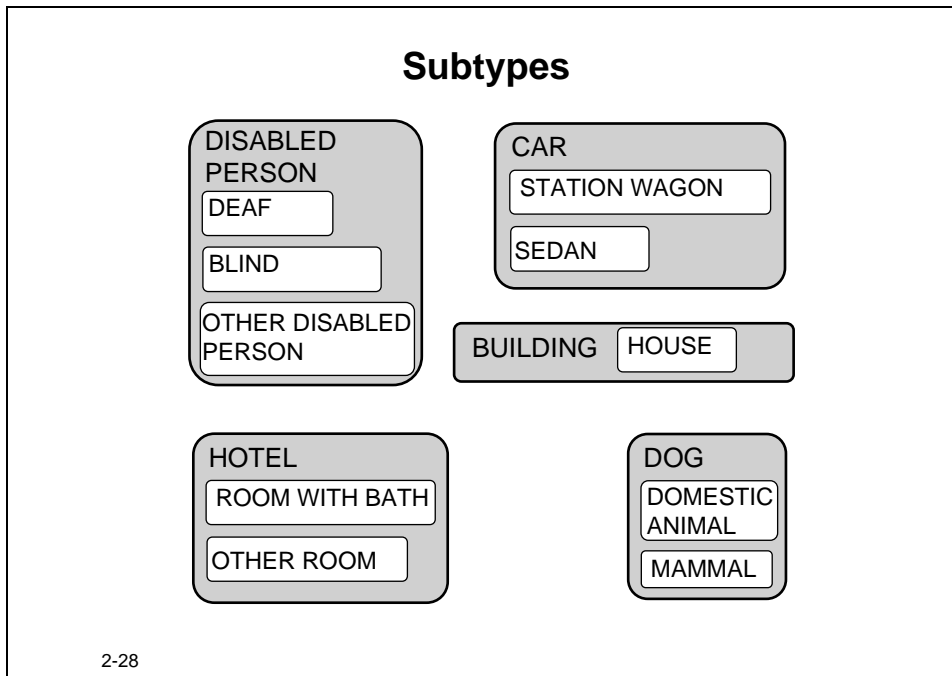
Practice 2—4: Subtypes

Goal

The goal of this practice is to determine correct and incorrect subtyping.

Your Assignment

Find all incorrect subtyping in the illustration. Explain why you think the subtyping is incorrect. Adjust the model to improve it.



Practice 2—5: Schedule



Scenario

You work as a contractor for Moonlight Coffees.

Your Assignment

Use the schedule that is used in one of the shops in Amsterdam as a basis for an entity relationship model. The schedule shows, for example, that in the week of 12 to 18 October Annet B is scheduled for the first shift on Monday, Friday, and Saturday.

van Gogh, Museumplein, Amsterdam

Schedule Oct 12 - Oct 18				prepared by Janet			
Shift	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Annet S			2		2	2	1
Annet B	1				1	1	
Dennis	2	2	1	2	3		
Jürgen						5	4
Kiri		3			4	4	
Wiel							

2-29

The scheme suggests there is only one shift per person per day.

Practice 2—6: Address

Goal

The goal of this practice is to sort out various ways of modeling addresses.

Your Assignment

An entity, possibly PERSON (or ADDRESS) may have attributes that describe the address as in the examples below.

Practice: Address (1/2)

Rheingasse 123 53111 Bonn Germany	34 Oxford Road Reading Berkshire RG1 8JS UK
1020 Maple Drive Kirkland WA 98234 USA	

- 1 How would you model the address information if the future system is required to produce accurate international mailings?

Practice 2—6: Address (continued)

Your Assignment

- 2 Would your model from the previous practice also accept the addresses below?

P.O. Box 66708 Nairobi Kenya	c/o Mrs Smith Maude Street Sandton Johannesburg 2144 South Africa
------------------------------------	---

- 3 Check if your model would be different if the system is also required to have facilities to search addresses in the following categories. Make the necessary changes, if any.

All addresses:

- In Kirkland
- With postal code 53111 in Bonn
- That are P.O. Boxes
- On:
 - Oxford Road or
 - Oxford Rd or
 - OXFORD ROAD or
 - OXFORD RDin Reading

3

Relationships in Detail

Introduction

Lesson Aim

This lesson discusses in detail how to establish a relationship between two entities. You meet the ten types of relationship and examples of the less frequent types. This lesson looks at nontransferable relationships and discusses the differences and similarities between relationships and attributes. It also provides a solution for the situation where a relationship seems to have an attribute. Finally, the rules of normalization are discussed in the context of conceptual models.

Overview

- **Relationships**
- **Ten different relationship types**
- **Nontransferability**
- **Relationships that seem to have attributes**
- **Rules of Normalization**

3-2

Topic	See Page
Introduction	2
Establishing a Relationship	4
Relationship Types	9
Relationships and Attributes	16
Attribute Compared to Relationship	18
Relationship Compared to Attribute	19
m:m Relationships May Hide Something	20
Resolving Relationships	25
Summary	32
Practice 3—1: Read the Relationship	33

Topic	See Page
Practice 3—2: Find a Context	34
Practice 3—3: Name the Intersection Entity	35
Practice 3—4: Receipt	36
Practice 3—5: Moonlight P&O	37
Practice 3—6: Price List	39
Practice 3—7: E-mail	40
Practice 3—8: Holiday	41

Objectives

At the end of this lesson, you should be able to do the following:

- Create a well-defined relationship between entities
- Identify which relationship types are common and which are not
- Give real-life examples of uncommon relationship types
- Choose between using an attribute or a relationship to model particular information
- Resolve a m:m relationship into an intersection entity and two relationships
- Resolve other relationships and know when to do so
- Rules of Normalization

Establishing a Relationship

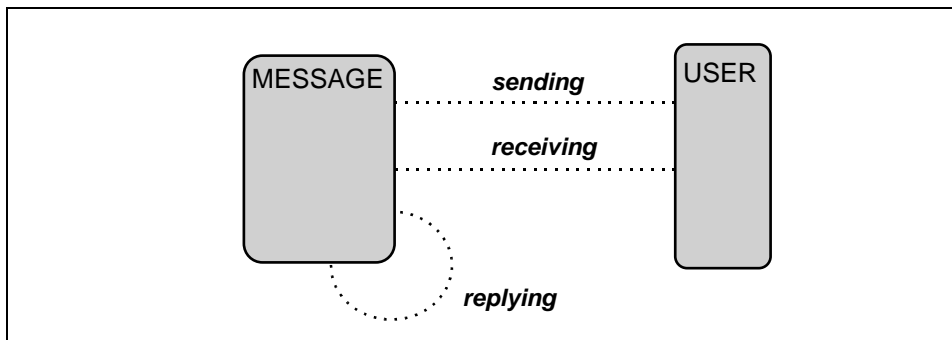
Establishing a Relationship

- **Determine the existence of a relationship**
- **Choose a name for the relationship from both perspectives**
- **Determine optionality**
- **Determine degree**
- **Determine nontransferability**

3-3

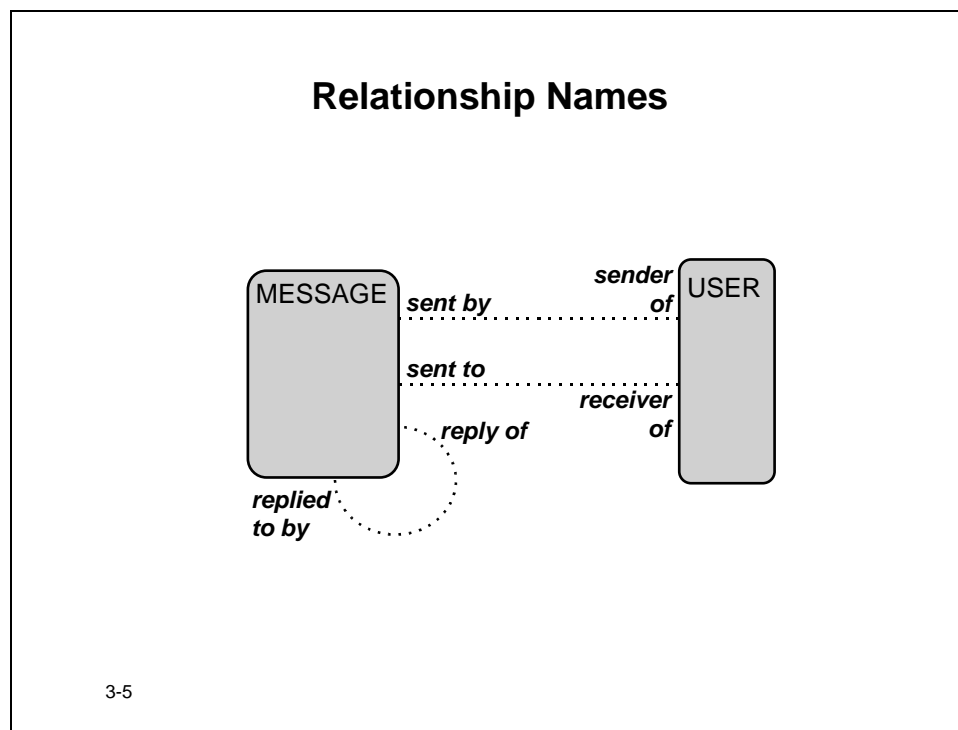
Determining the Existence of a Relationship

- Ask, for each of your entities, if it is somehow related to one or more of the entities in your model, and, if so, draw a dotted “skeleton” relationship line.
- Usually all entities in a model are related to at least one other entity. Exceptions are rare, but they do exist.
- Two entities can be related more than once. For example, in the Electronic Mail system there are two relationships between entities MESSAGE and USER, one is about who is *sending* a MESSAGE and one about who *receives* a MESSAGE.
- An entity can be related to itself. This is called a recursive relationship. For example, a MESSAGE can be a reply to another MESSAGE. See the paragraph on recursive relationships for more details on this.



Choosing a Name for the Relationship

- Sometimes the relationship name for the second perspective is simply the passive tense of the other one, such as *is owner of* and *is owned by*. Sometimes there are distinct words for both concepts, such as *parent of / child of* or *composed of / part of*.
- Try to use names that end in a preposition.
- If you cannot find a name, you may find these relationship names useful:
 - Consists of / is part of
 - Is classified as / is classification for
 - Is assigned to / is assignment of
 - Is referred to / referring to
 - Responsible for / the responsibility of
- Sometimes a very short name is sufficient, for example, with, in, of, for, by, about, at, into.



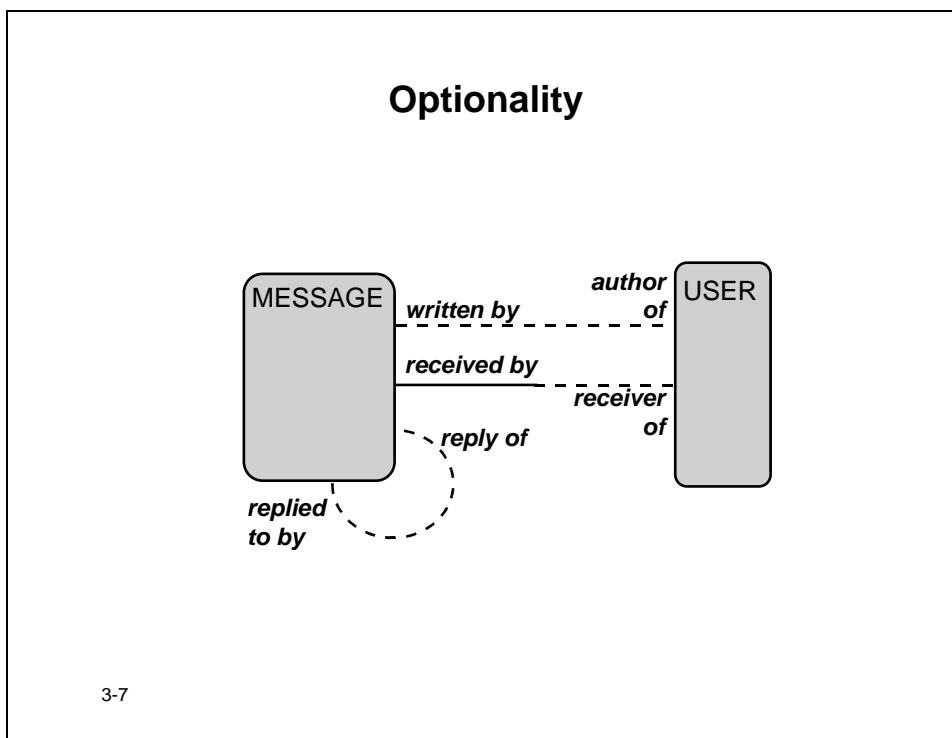
Are *sent to* and *receiver of* really opposite? If so, the assumption is that if a MESSAGE is sent to a USER, it also arrives. Maybe it is safer to name the relationship *received by / receiver of*...

Determining Optionality of Both the Relationship Ends

- Answer the questions:
 - Must every MESSAGE be sent by a USER?
 - Must every USER be sender of an MESSAGE?
 - Must every MESSAGE be sent to a USER?
 - Must every USER be addressed in a MESSAGE?

When an answer is Yes the relationship end is mandatory, otherwise it is optional.

- Be careful at this point. Often a relationship end *seems* to be mandatory, but actually it is not. In the ElectronicMail example it seems that every MESSAGE *must* be sent by a USER. But a MESSAGE that was sent by an external user to an internal USER has no relationship to a USER, unless the system were to keep external users as well.
- Sometimes a relationship is ultimately mandatory, but not initially. Such a relationship should be modeled as optional.

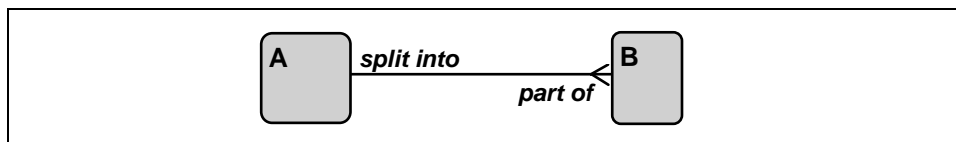


Determining Degree of Both the Relationship Ends

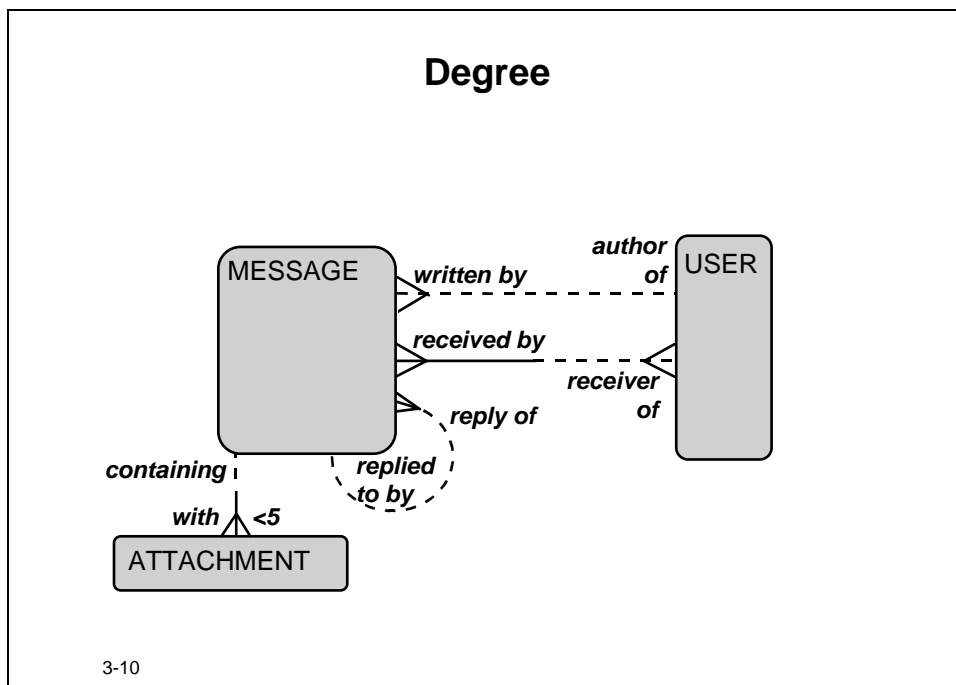
- Answer the questions:
 - Can a MESSAGE be written by more than one USER?
 - Can a USER be author of more than one MESSAGE?

If the answer is No the degree is called “1”.

If the answer is Yes the degree is called “many” or just “m”.
- This must be determined for all relationship ends.
- Note that a **mandatory “many”** relationship end from A to B does not mean that it is mandatory for A to be *split into more* than one B. One B is fine. Read it as: every A must be *split into at least one B*.

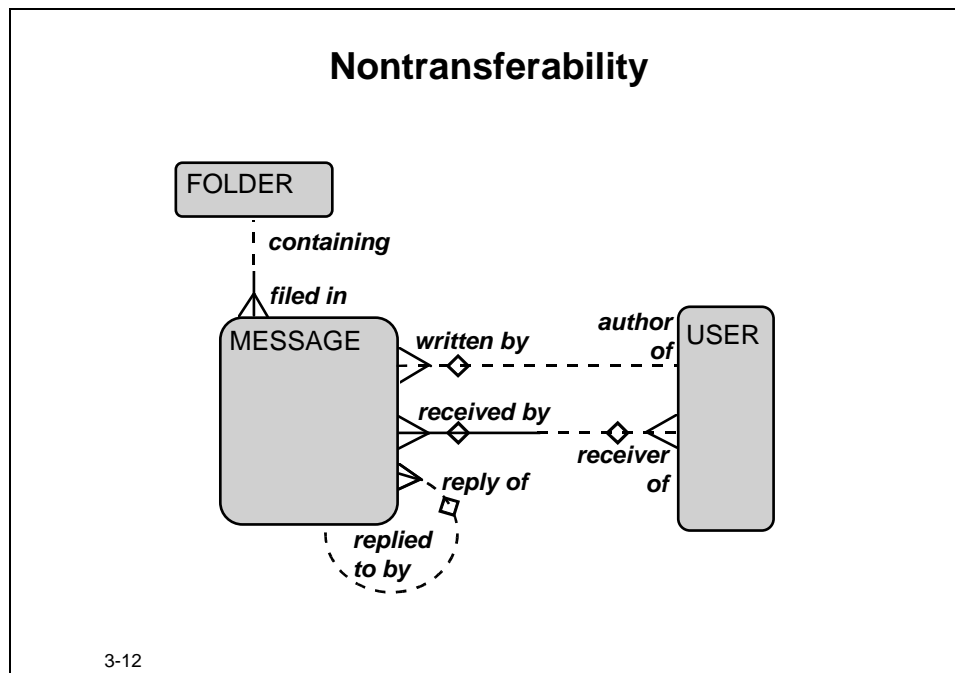


- An **optional “many”** relationship end means *zero, one or more*. In the e-mail example a USER can be author of 0,1 or more MESSAGES.
- Sometimes the degree is a fixed value, or there is a maximum number. Assume a MESSAGE may be *containing* one or more ATTACHMENTS, but for some business reason, the number of ATTACHMENTS per MESSAGE may not exceed 4. The degree then is <5. The diagram, however, shows a crow's foot.



Determine Nontransferability of Both the Relationship Ends

- When a MESSAGE is created, the USER who is the author of the MESSAGE is a fact. It would be strange if a mail system allowed you to change the author after the MESSAGE is completed.
- Often relationships have the following property: you cannot change the connection, once made. That property is called nontransferability. Nontransferability leads to nonupdatable foreign keys. Nontransferability is shown in the diagram with a little diamond-shaped symbol through the line of the relationship end.



- Not all relationships are nontransferable. Assume the mail system allows a user to file a MESSAGE in a FOLDER. This is only a valuable functionality if the user is allowed to change the FOLDER in which a MESSAGE is filed.

Relationship Types

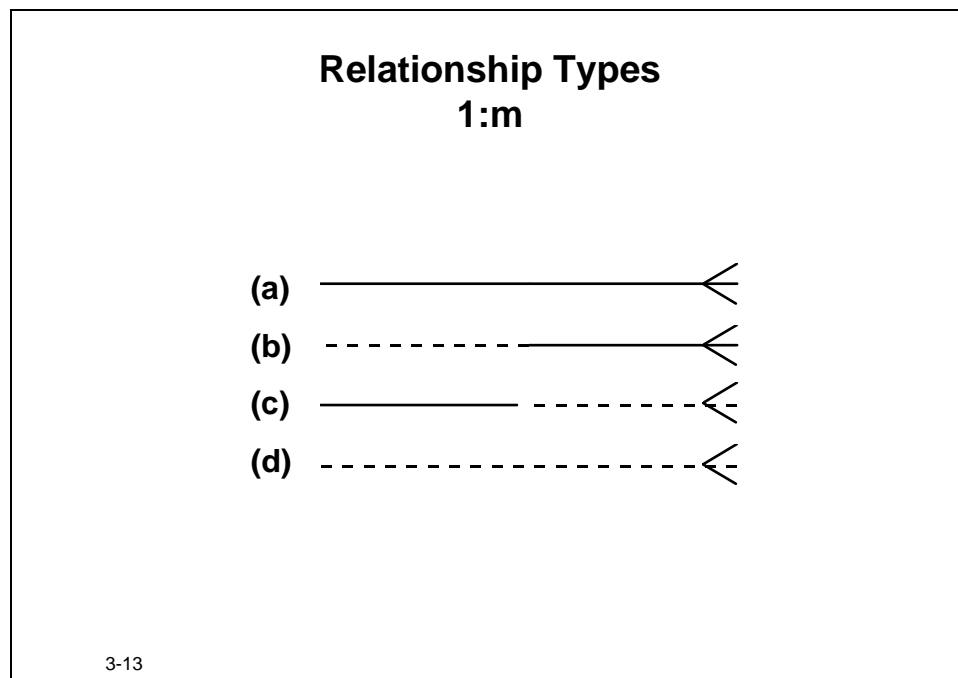
There are three main groups of relationships, named after their degrees:

- One to many (1:m)
- Many to many (m:m)
- One to one (1:1)

This paragraph discusses the various types and gives some examples of their variants.

Relationships—1:m

The various types of 1:m relationships are most common in an ER Model. You have seen several examples already.

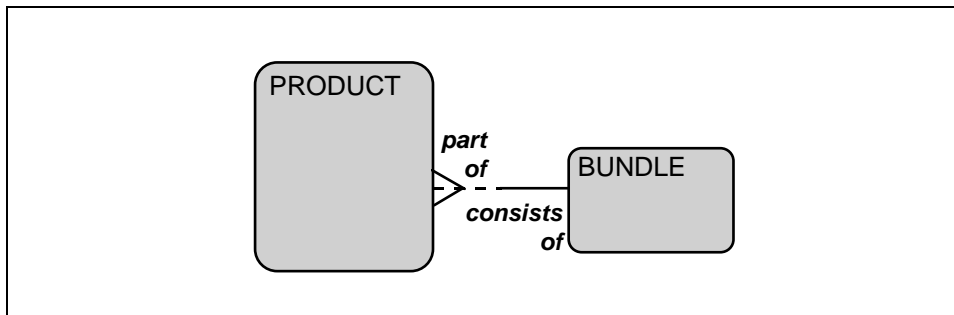


- a** Mandatory at both ends. This type of relationship typically models entities that cannot exist without each other. Often the existence of mandatory details for a master is more wishful thinking than a strict business rule. Often the relationship expresses that an entity is always split into details. Seen from the other perspective, it often expresses an entity that is always classified, assigned.

Circumventing Mandatory 1 to Mandatory m Usually you would try to avoid relationship type (a) in favor of type (b), by taking a different perspective on the subject. For example, suppose an order is defined as something with at least one order item. In other words, an order is regarded as a composed concept. You can avoid modeling order as an entity as you can decide to model a slightly different concept instead, say ORDER HEADER. Next, define an ORDER HEADER to have zero, one or more ORDER ITEMS. An *order* would then be a thing composed of two entities: any ORDER HEADER with one or more ORDER ITEMS. Empty headers would not be considered to be an order.

Why Circumvent? Implementing a 1:m relationship that is mandatory at both ends causes technical problems. In particular it is difficult to make sure details exist for a newly-created record. In most relational database environments it is even impossible.

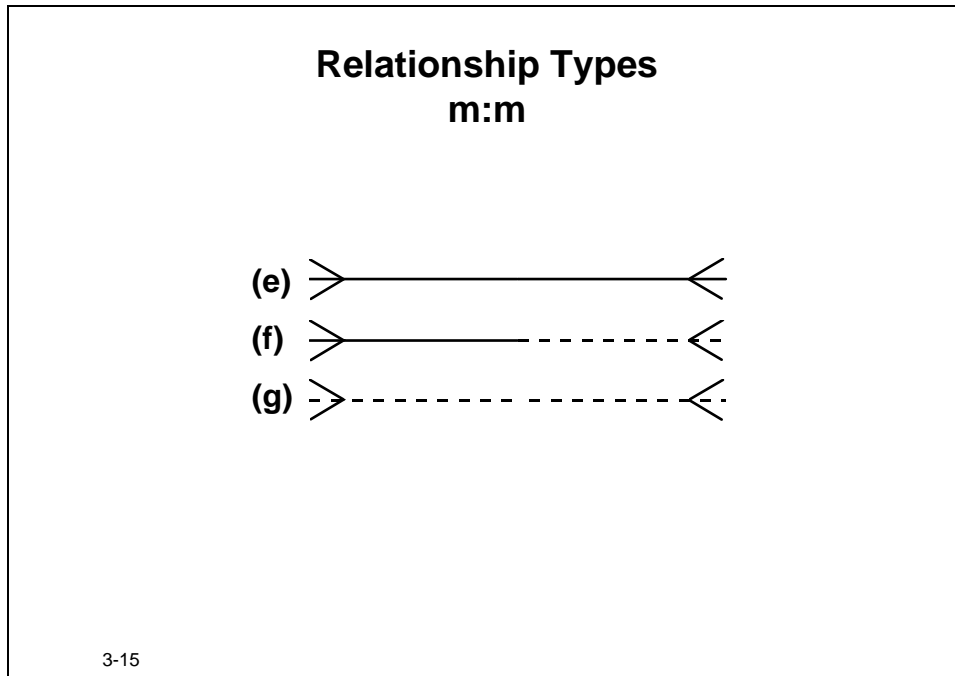
- b** Optional 1: mandatory m. This is a very common type of relationship, together with (d). Normally, at least 90% all relationships are of type (b) and (d). The relationship expresses that the entity at the 1-end can stand alone, whereas the entity at the many end can only exist in the context of the other entity.
- c** Mandatory 1: optional m. This is not common. You will see it only when the relationship expresses that an entity instance only exists when it is a non-empty set, and where the elements of the set can exist independently. In the example below a PRODUCT may be part of one BUNDLE. According to the model, a BUNDLE is of no interest if it is empty.



- d** Optional at both ends. See remarks for (b).

Relationships—m:m

The various types of m:m relationships are common in a first version of an ER Model. In later stages of the model most m:m relationships, and possibly all, will disappear.

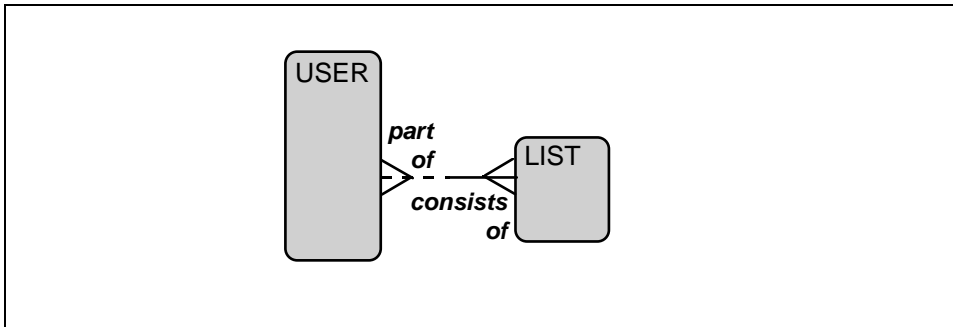


- e Mandatory at both sides is very uncommon in normal circumstances. This relationship seems to mean that an entity instance can only be created if it is immediately assigned to an instance of the other entity, as well as conversely. But how can this occur when we do not have an instance of either entity? Enforcing the mandatory rule from scratch leads to a conflict.

The relationship can, however, be part of a model of a theoretical nature, like the mathematical: a LINE always consists of many POINTS and a POINT is always part of many LINES. It can also describe an existing situation: a DEPARTMENT always has EMPLOYEES and an EMPLOYEE is always assigned to a DEPARTMENT. Here the question may arise if it is guaranteed that the situation will always remain this way.

A m:m relationship that is mandatory at both sides can occur when the relationship is part of an arc. See the lesson on Constraints for more details.

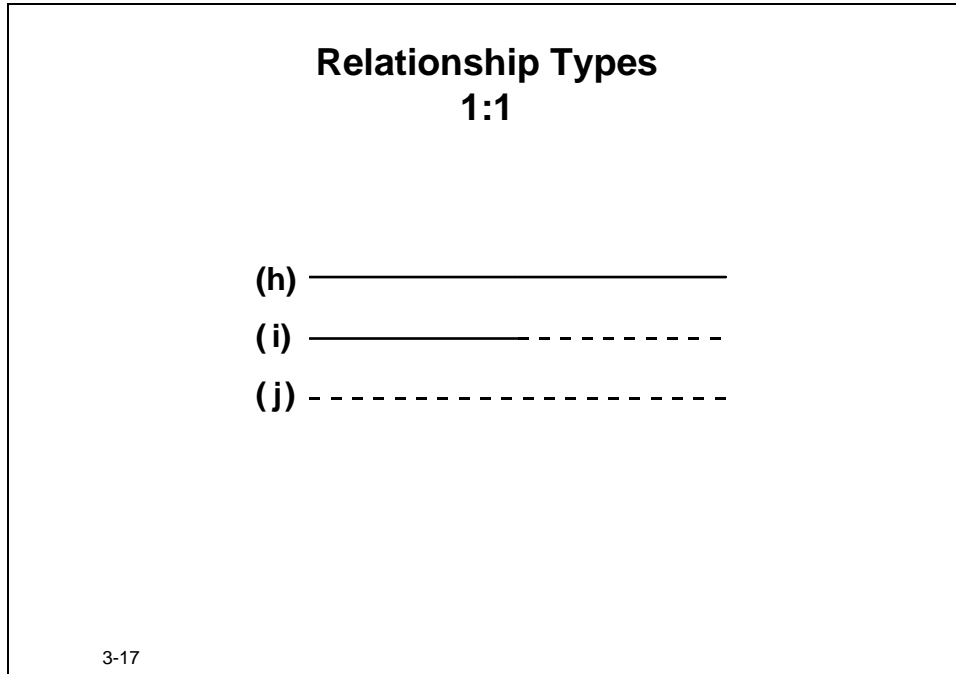
- f** Mandatory at one end is not uncommon in early versions of a model although they usually disappear at a later stage.



- g** Optional at both ends is common in early versions of a model. These also usually disappear at a later stage.

Relationships—1:1

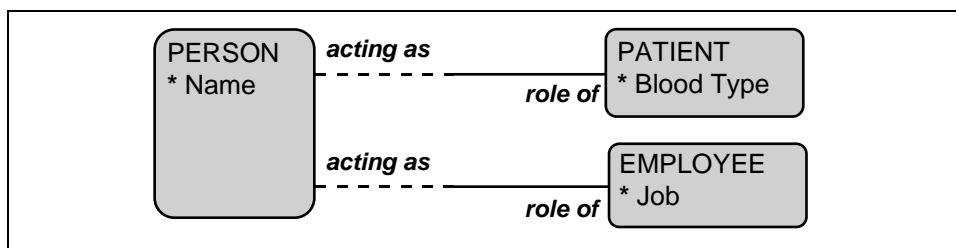
Usually you will find just a few of the various types of 1:1 relationships in every ER Model.



- h** A 1:1 relationship, mandatory at both ends, tightly connects two entities: when you create an instance of one entity there must be exactly one dedicated instance for the other simultaneously; for example, entity PERSON and entity BIRTH. This leads to the question why you want to make a distinction between the two entities anyway. The only acceptable answer is: only if there is a functional need.

If you have this relationship in your model, it is often, possibly always, part of an arc.

- i** Mandatory at one end is often in a model where *roles* are modeled, for example, in this hospital model.

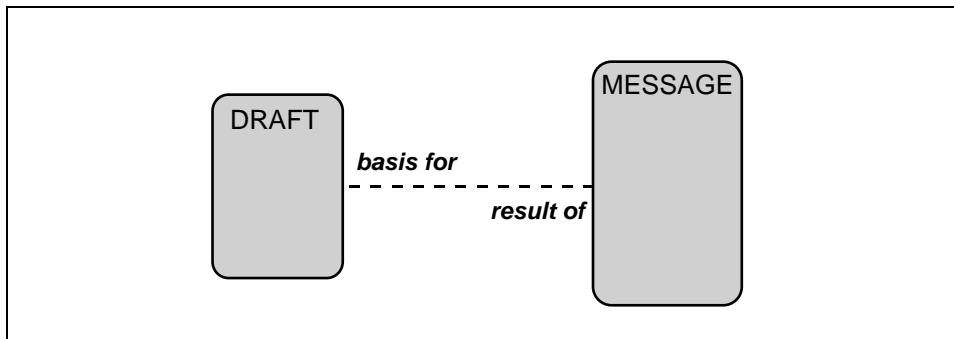


See Page 46

Note: These role-based relationships are often named *is/is type of* or simply *is/is*.

Both PATIENT and EMPLOYEE are roles played by a PERSON. The attribute BLOOD TYPE is, according to this model, only of interest when this person is a PATIENT. Note that PATIENT and EMPLOYEE cannot be modeled as subtypes of PERSON, as a PERSON may play both roles. You meet the concept of roles again in a later lesson.

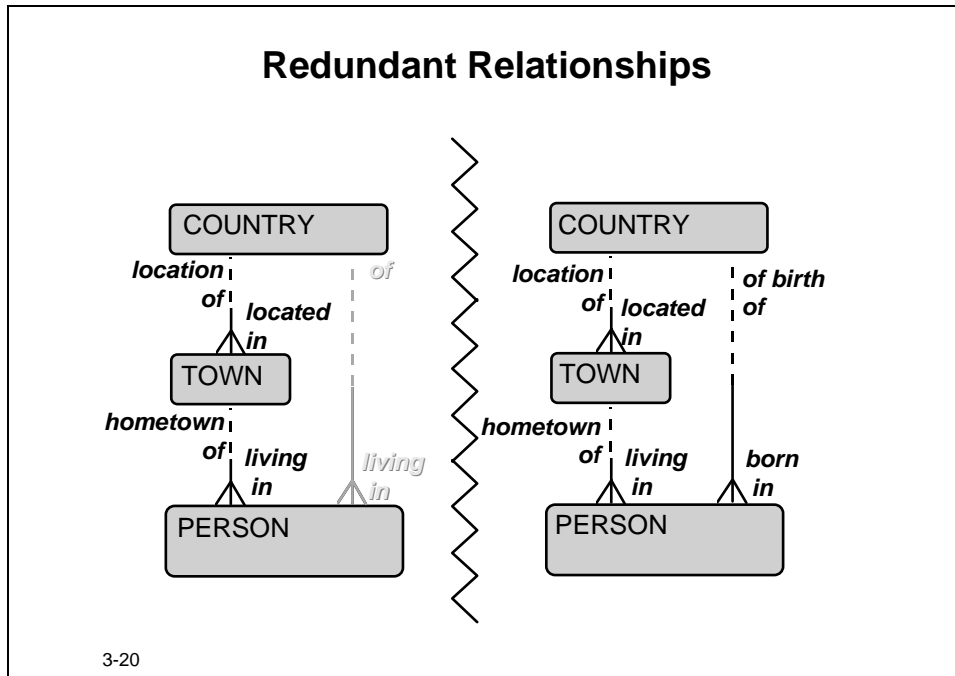
- j Optional at both ends is uncommon. However, they can occur, for example, when there is a relationship between two entities that are conceptually the same but exist in different systems. An example of this is entity EMPLOYEE in one system and entity PERSON in a different, possibly a third-party, system. Many 1:1 relationships (of all three variants) do occur when some of the entities represent various stages in a process, such as in the next example. Relationship names in this case can always be *leads to* / *result of* or something similar.



If you consider a person to be a process as well, the earlier example of BIRTH and PERSON fit nicely into this general idea.

Redundancy

Like attributes, relationships can be redundant.



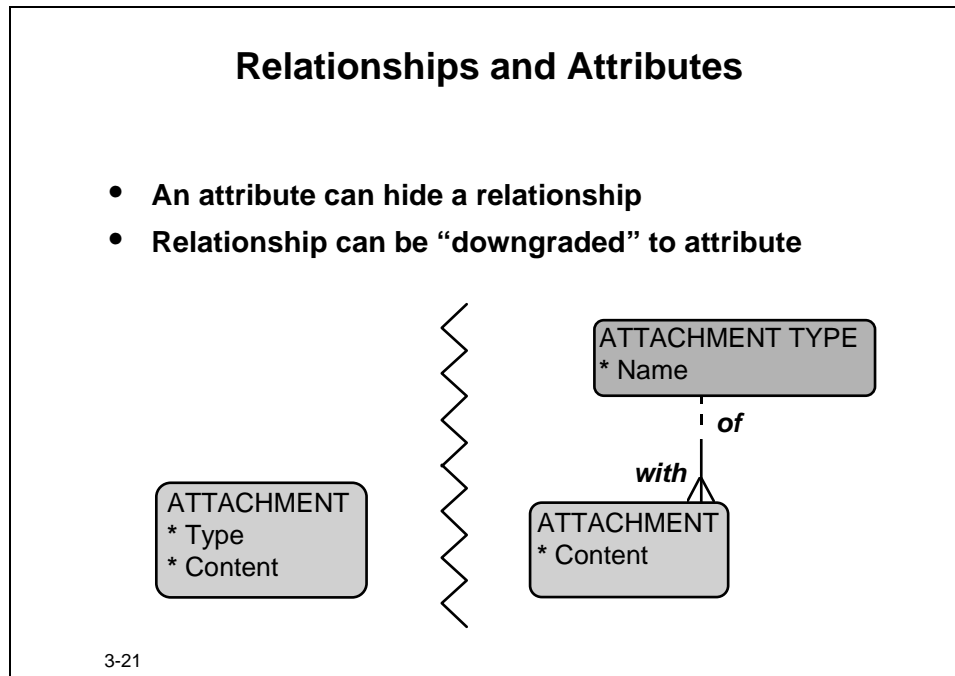
In the left-hand example you can derive the relationship from PERSON to COUNTRY from the other two relationships and you should remove them from the model.

This is a semantic issue and cannot be concluded from the structure alone, as the right-hand example shows.

Relationships and Attributes

Attributes can hide a relationship. In fact, any attribute can hide a relationship.

See Page 48



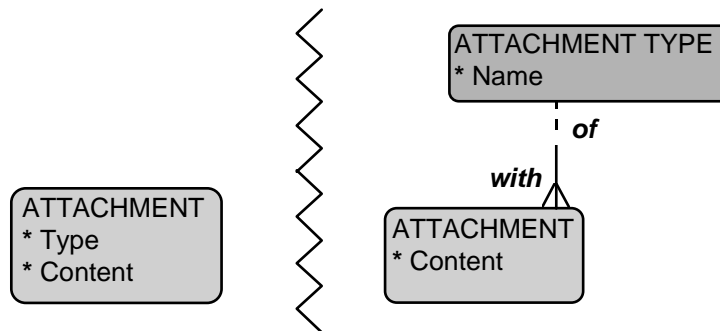
In the example, attribute TYPE of entity ATTACHMENT can be replaced by an entity ATTACHMENT TYPE plus a relationship from ATTACHMENT to ATTACHMENT TYPE.

You would have no choice other than to model it this way as soon as you need to keep extra attributes for ATTACHMENT TYPE. If there are no important attributes for ATTACHMENT TYPE to keep other than the Name of the type, you could consider removing the entity and take Type as an attribute of ATTACHMENT.

You could also consider using the left-hand option when the number of types is a *fixed* and *small* amount, such as in the context of a chain of hotels where there are only three types of rooms: single, double, and suite.

Attribute Compared to Relationship

- **Easy model**
- **Fewer tables**
- **No join**
- **Value control**
- **List of values**
- **Other relationships**



3-22

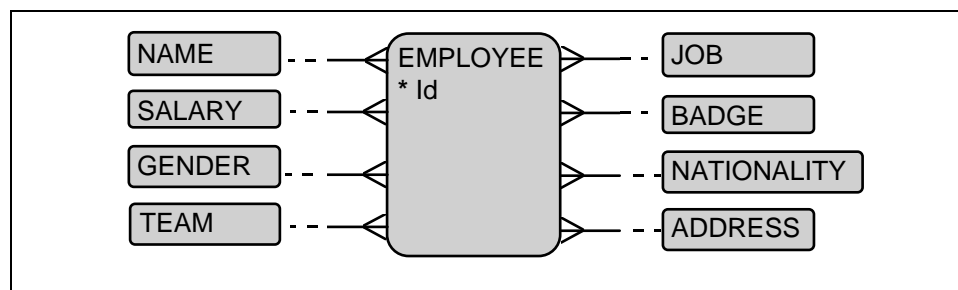
The table based on entity **ATTACHMENT** would contain the same columns in both situations, but the Attachment Type Name column would be a foreign key column in the second implementation. This would mean that an Attachment Type Name entered for an **ATTACHMENT** can only be taken from the types listed in the table based on entity **ATTACHMENT TYPE**. The list serves as a pick list and spelling check.

There are advantages and disadvantages for both models.

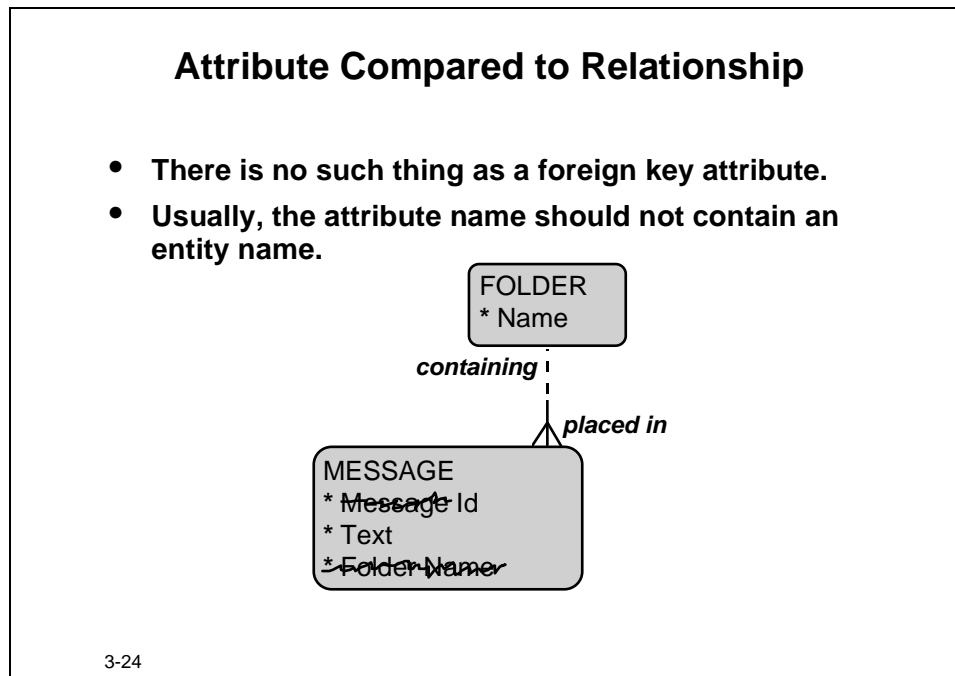
The one entity model is somewhat easier to read because it is less packed with lines. In the table implementation you would need no joins to get the required information.

However, a two-entity model is usually far more flexible. It leaves the option open to create relationships from other entities to the new entity. You would have control over the values entered as they are checked against a given set. Usually, the two-table implementation takes less (sometimes even much less) space in the database.

Use your common sense when you select the attributes and entities.



Attribute Compared to Relationship



Nonexistence of Foreign Key Attributes

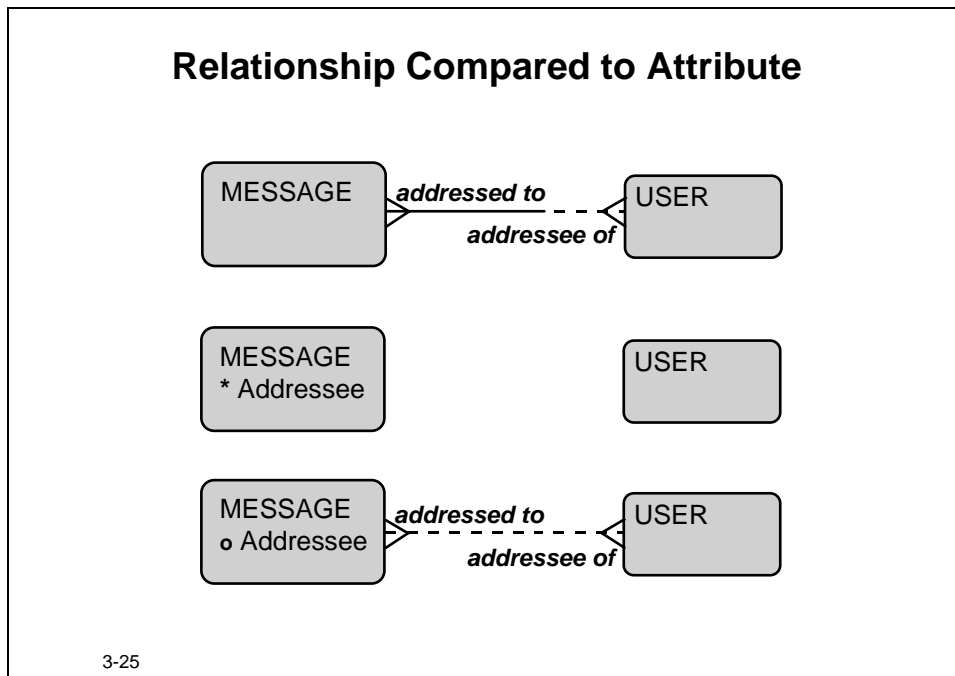
Be aware of foreign key attributes such as attribute Folder Name of entity MESSAGE in the example. In ER modeling there is no such thing as a foreign key attribute. The future foreign key is represented by the relationship between MESSAGE and FOLDER. A foreign key column (or columns) will result from the primary unique identifier of the entity FOLDER. See the lesson on CONSTRAINTS for more details on unique identifiers.

No Entity Name in Attribute Name

When an attribute name contains an entity name, it usually comes from one of the following situations:

- The attribute hides a relationship to an entity, as in the above example. The second entity was probably added in a later stage.
- The attribute hides an entity. A typical example is an attribute Employment Date of entity EMPLOYEE. This might hide the entity EMPLOYMENT, as there is probably no rule that an employee may be employed by the same company only once.
- The entity name in the attribute name is redundant. A typical example is attribute Message Id of entity MESSAGE. The name “Id” would suffice.
- The attribute is the result of a one-to-one relationship that is not modeled, for example, attributes Birth Date and Birthplace of entity EMPLOYEE. These are in fact attributes of an entity BIRTH that is not (and probably will never be) modeled.

Relationship Compared to Attribute



Sometimes a piece of information looks like a relationship between entities, but actually is not a relationship.

In ElectronicMail's Compose Message screen there is a field labeled "To" where the user is supposed to enter the names of the addressees. Initially you may want to model that as a relationship *addressed to / addressee of* between MESSAGE and USER, but this is a questionable approach. If a message is sent to an external user would it make sense for ElectronicMail to keep track of all external user addresses that were used to send messages to, just for the sake of maintaining the relationship? Would this be possible?

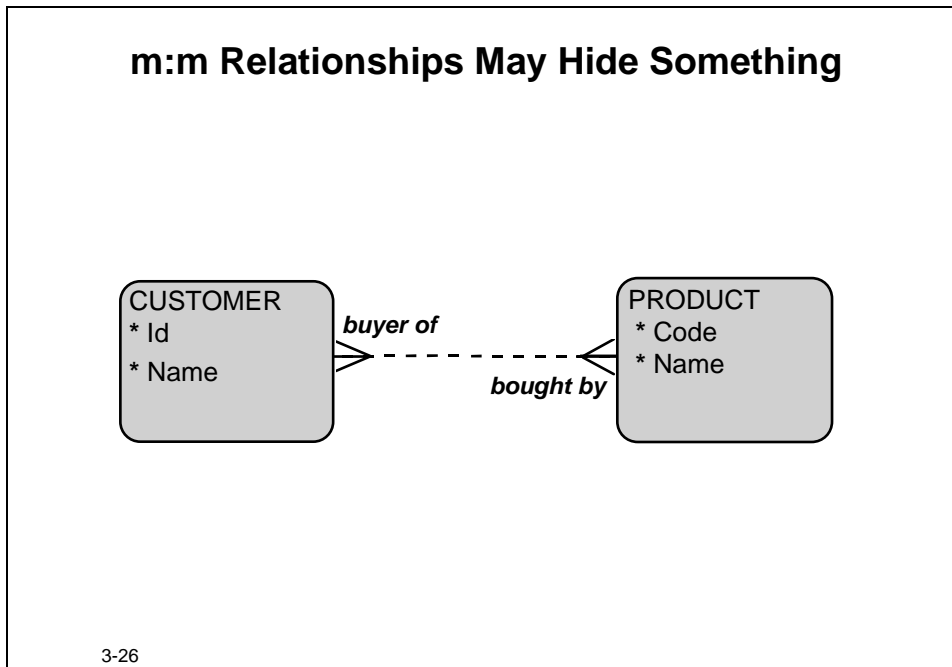
In this case it would be a better choice to see the Addressee as an attribute of the MESSAGE. This attribute *may* contain a value that is also known as a USER. In other words, entity USER contains only suggestions for addressees.

Another possibility is to do both—model an optional relationship and an optional attribute that cooperatively handle the addressee. An extra constraint (which cannot be shown in the diagram) must then make sure that at least one of the attributes or the relationship is actually given a value for a MESSAGE.

m:m Relationships May Hide Something

During the process of modeling you will find many relationships to be of type m:m. Often this is a temporary thing. After you have been able to add more details to the model, a lot of the m:m relationships will disappear as, after consideration, they simply do not model the business properly.

A typical example is about the CUSTOMER/PRODUCT relationship.

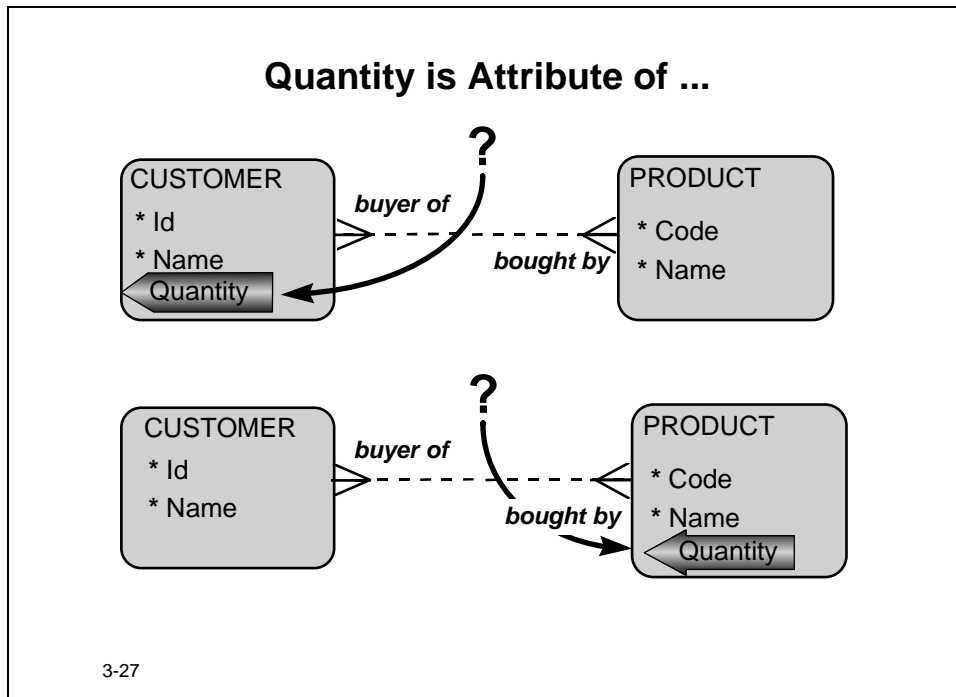


Suppose you make a model for a retail company that sells PRODUCTS. A CUSTOMER *buys* PRODUCTS. Suppose future customers are accepted into the system as well. This would mean:

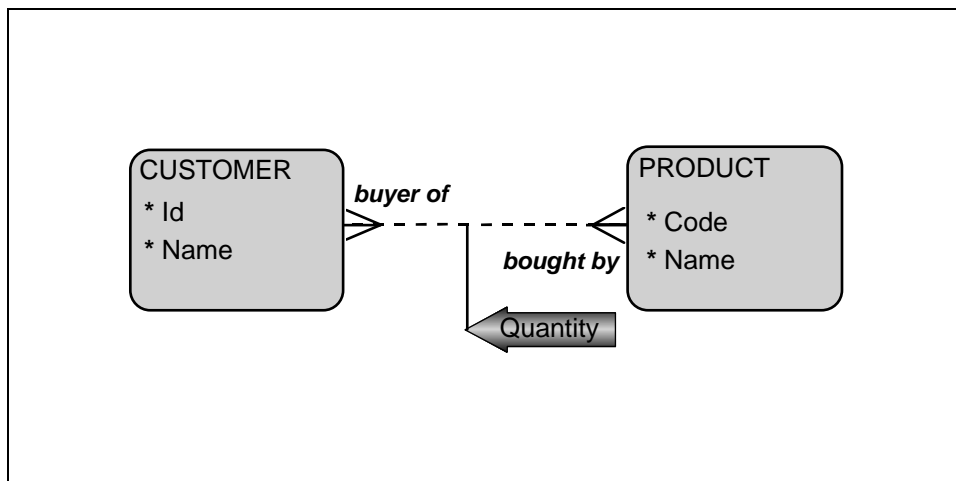
A CUSTOMER may *buy* one or more PRODUCTS

A PRODUCT may be *bought by* one or more CUSTOMERS

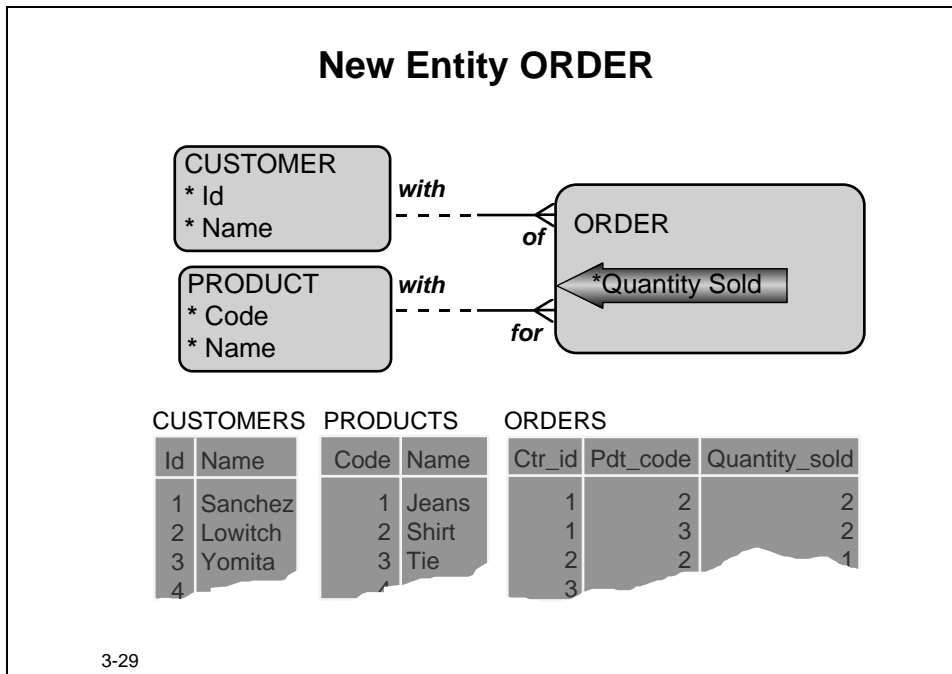
A typical event for this company would be customer Nick Sanchez buying two shirts. “Nick Sanchez” is a CUSTOMER Name, “shirt” is a PRODUCT Name. This leaves the question of where to put the “two”, the quantity information.



It is clear that Quantity is neither a property of CUSTOMER nor of PRODUCT. Quantity seems to be an attribute of the relationship between CUSTOMER and PRODUCT.



Relationships do not and cannot have attributes. Apparently an entity of which quantity is a property, is missing. For that reason we need to change the model. Entity ORDER (or SALE or PURCHASE) enters the scene.

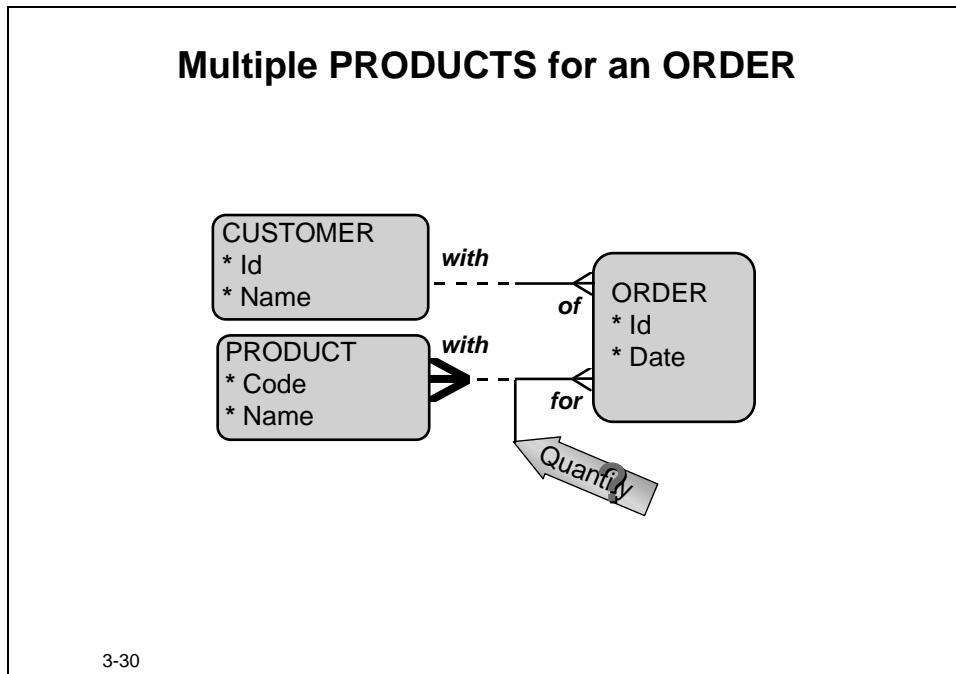


The table design here is the default design for implementing the model. Note the two foreign key columns in the ORDERS table, Ctr_id (foreign key to CUSTOMERS) and Pdt_code (to PRODUCTS).

Now suppose Pepe Yomita enters the store and buys one pair of jeans, two shirts, and *one* silk tie. Given the current model this would mean that Pepe places three orders: one for the jeans, one for the shirts and one for the tie. Three orders, all at the same time, from one and the same customer. No problem so far as the model allows for this.

Now suppose the store wants to automate the billing of the orders. (This is probably one of the reasons for making the model anyway.) Using the above model, this would mean three orders and, as a consequence, three bills, as the system has no way of knowing these three orders somehow belong to each other.

It is better to change the model in such a way that one order can be for more than one product. That means we should have a m:m relationship between ORDER and PRODUCT, which we should investigate next.



Then there is the question again: where do you put quantity? Quantity can now no longer be an attribute of an order because the attribute must be single-valued and cannot contain three values 1, 2 and 1 at the same time. Quantity has become a property of the m:m relationship between PRODUCT and ORDER.

Resolving Relationships

Relationships and Intersection Entities

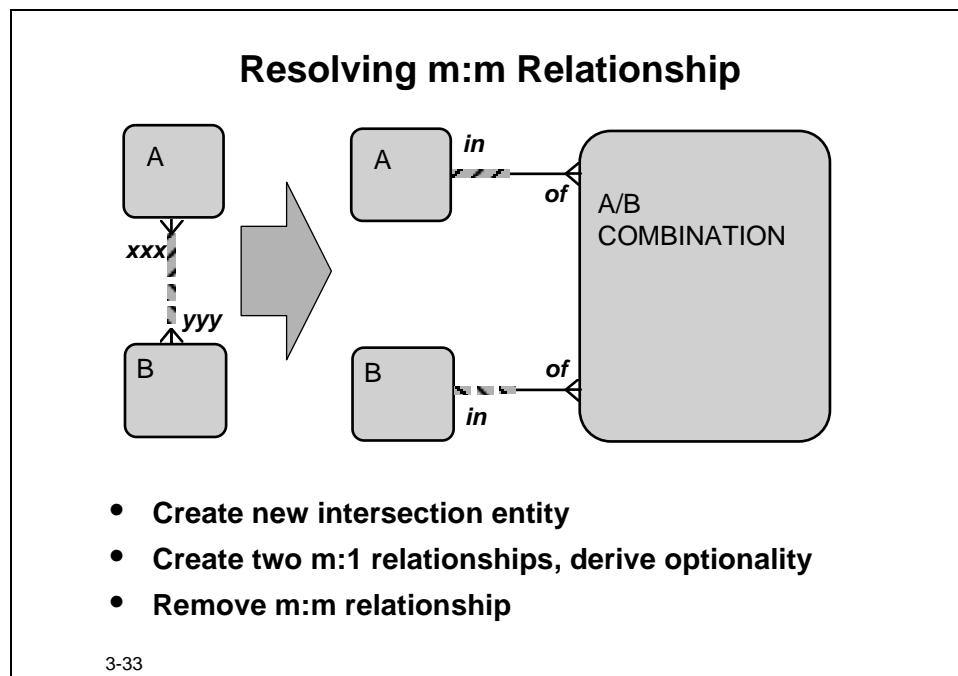
Earlier in this lesson you saw a typical example of relationships seeming to have attributes. The relationships in the example were many-to-many relationships. You deal with the situation by creating a new entity, an intersection entity, that replaces the relationship and can hold attributes.

This leads to the following questions:

- What are the steps in resolving a relationship in general?
- Should every m:m relationship be resolved?
- Can other relationships than m:m be resolved?

Resolving a Relationship

Suppose we want to resolve the m:m relationship between entities A and B.



- 1 First create a new intersection entity. You will experience that sometimes there is no suitable word available for the concept you are modeling. The new entity can always be named with the neologism “A/B COMBINATION”, or a name that is somehow derived from the name of the original m:m relationship. Do not let the unavailability of a proper name for the entity stop you from modeling it.
- 2 Next create two new m:1 relationships from entity A/B COMBINATION, one to A and one to B. Initially, draw these as mandatory at A/B COMBINATION, as you will probably only be interested in complete pairs of A and B. If the original

m:m relationship was optional (or mandatory) at A's side, then the new relationship from A to A/B COMBINATION is also optional (or mandatory).

- 3 Name the relationships. You can often name both relationships "in / of".
- 4 The next step is to remove the m:m relationship you started with.
- 5 Finally, reconsider the newly-drawn relationships. They may be optional at the A/B COMBINATION side. Also, they may turn out to be of type m:m and require resolving, as you have seen in the example of customers buying products.

Should Every m:m Relationship be Resolved?

The answer depends on a number of factors.

Given the usual scenario, when you start creating an ER model you will discover that many of the relationships you draw are of type m:m. Most of these will appear to hide entities that you need in a later stage as you need to have a place in which to put specific attributes. Finally, you will have only a few "genuine m:m" relationships left.

No Purely from a conceptual data modeling point of view, there is no need to resolve these genuine m:m relationships. The model is rich enough to be the basis for table design. A m:m relationship will transform into a binary table; this is a table that consists of the columns of two foreign keys only. This is exactly the same table as the one that would result from the intersection entity when you resolved the m:m relationship.

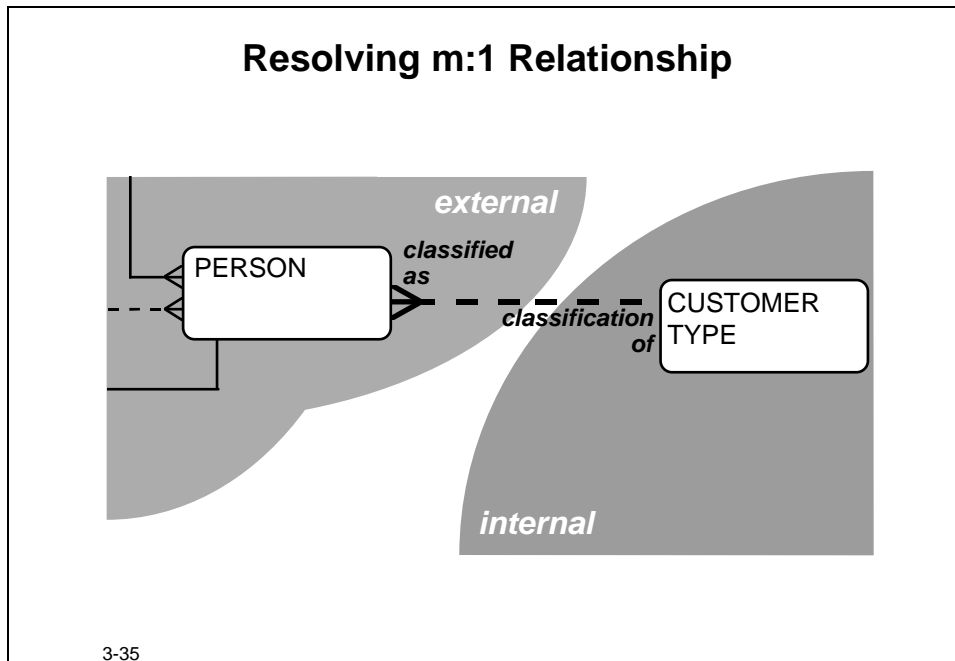
A m:m relationship in a conceptual data diagram needs less space than a separate entity plus two relationships. For this reason a diagram with unresolved m:m relationships is more transparent and easier to read.

Yes From a function modeling point of view the answer is different. If your model contains a true m:m relationship there is apparently a business need to keep information on the combinations of, say, entity A and B. In other words, the system would contain at least one business function that creates the relationship. This "create relationship" cannot be expressed as a usage of entities or attributes, although this is usually what design tools require of the functional model. Oracle Designer is no exception. This means that when you create an ER model in Oracle Designer you would always resolve the m:m relationships in order to create a fully-defined functional model with all data usages included.

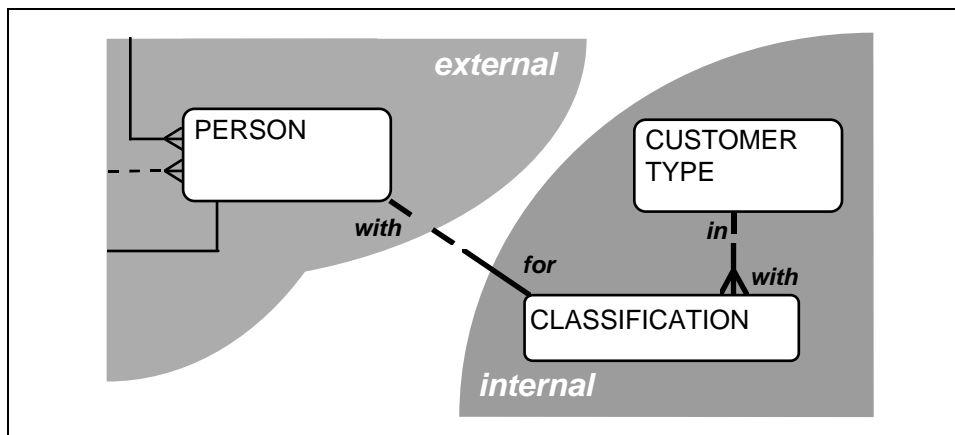
Resolving Other Relationships

Can relationships other than m:m be resolved? Yes. Every relationship, even a 1:1, can be resolved into an intersection entity and two relationships, just like a m:m relationship. When would you want to do this? It is quite rare to find a situation where you have to do this. A typical situation where you may like to resolve a non m:m relationship is when one entity represents something that is outside your system, for example, when the entity is part of a third-party package.

Suppose you need your system to create a m:1 relationship from external entity **PERSON** to **CUSTOMER TYPE**, one of your internal entities (as in the diagram below):



This would result later on in a change of the table structure of the third-party **PERSONS** table. This is undesirable (third parties often ask you to you sign a contract that simply forbids you to do that) and sometimes even impossible if you have no authority over that table.



The above model leaves the external entity **PERSON** as is and does the referencing from inside. The m:1 relationship is replaced by an entity **CLASSIFICATION** and two relationships.

Normalization During Data Modeling

Normalization is a relational database concept. However, if you have created a correct entity model, then the tables created during design will conform to the rules of normalization. Each formal normalization rule from relational database design has a corresponding data model interpretation. The interpretations which can be used to validate the placement of attributes in an ER Model are as follows.

Normalization Rules

Normal Form Rule	Description
First Normal Form	All attributes are single valued.
Second Normal Form (2NF)	An attribute must be dependent upon entity's entire unique identifier.
Third Normal Form (3NF)	No non-UID attribute can be dependent on another non-UID attribute.

"A normalized entity-relationship data model automatically translates into a normalized relational database design"

"Third normal form is the generally accepted goal for a database design that eliminated redundancy"

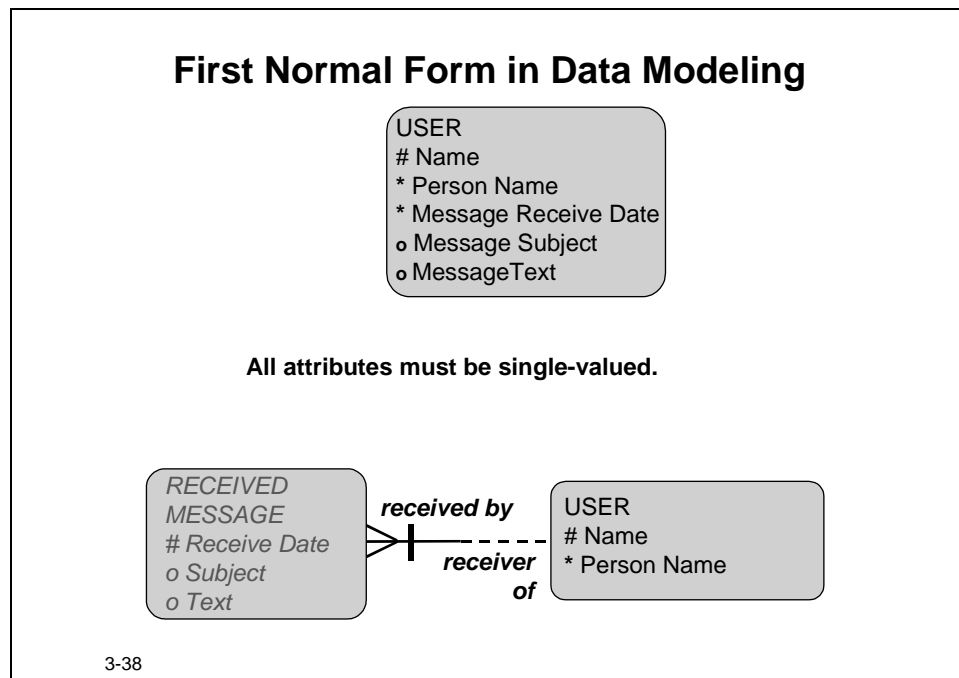
First Normal Form in Data Modeling

All attributes must be single-valued.

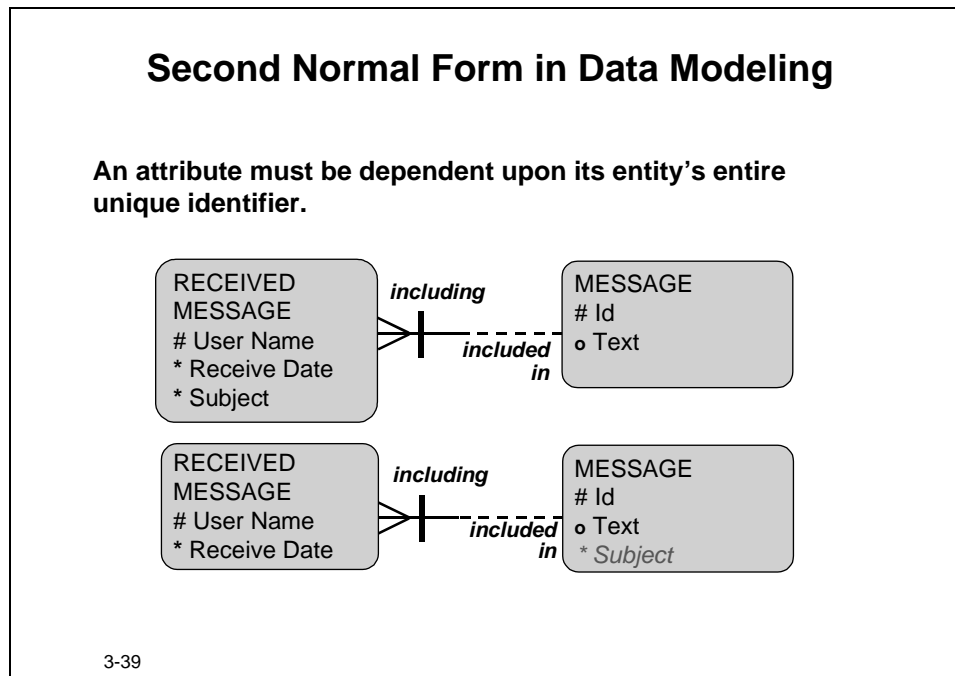
Validate that each attribute has a single value for each occurrence of the entity. No attribute should have repeating values.

You can often recognize the misplaced attributes by the fact that there is the same (entity) name in the attribute name, such as *Message Subject* and *Message Text*.

If the attribute has multiple values, create an additional entity and relate it to the original entity with a m:1 relationship.



Second Normal Form in Data Modeling

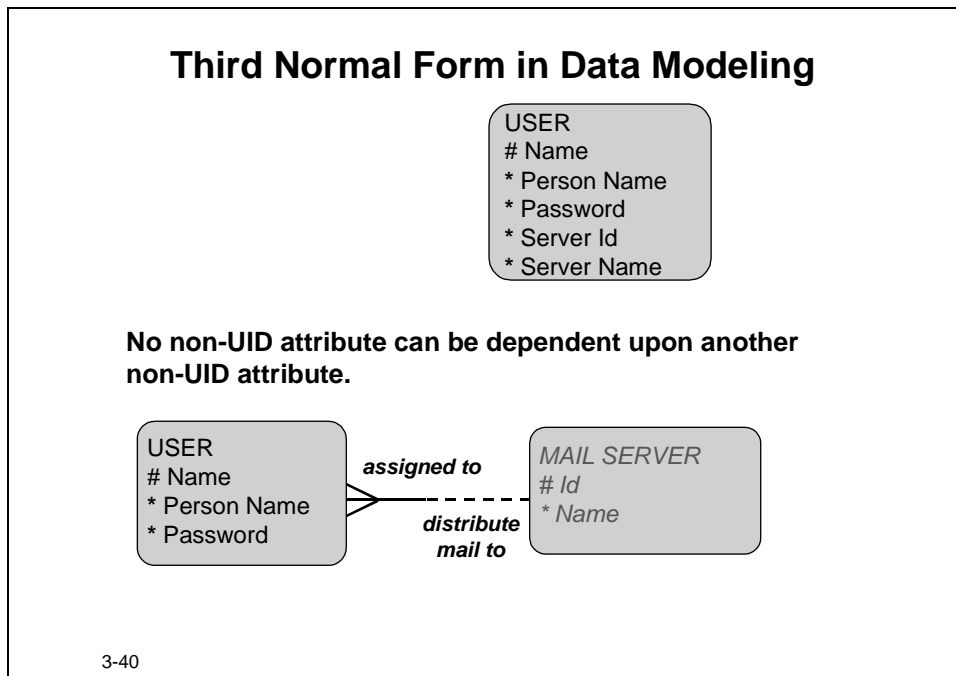


An attribute must be dependent upon its entity's entire unique identifier.

Validate that each attribute is dependent upon its entity's entire unique identifier. Each specific instance of the UID must determine a single instance of each attribute.

Validate that an attribute does not depend upon only part of its entity's UID. If it does, then it is misplaced and you must move it.

Third Normal Form in Data Modeling



No non-UID attribute can be dependent upon another non-UID attribute. If an attribute is dependent upon a non-UID attribute, then move both the dependent attribute and the attribute it is dependent upon to a new, related entity.

Summary

Summary

- **Relationships express how entities are connected.**
- **Initially relationships often seem to be of type m:m.**
- **Finally relationships are most often of type m:1.**
- **Relationships can be resolved into:**
 - **Two new relationships.**
 - **One intersection entity.**
- **Third Normal form is generally accepted standard.**

3-41

Relationships connect entities and express how they are connected. There are ten types of relationships, 4 of type 1:m, 3 of type m:m and 3 of type 1:1.

The m:1 relationship that is optional at the 1 side is by far the most common type in finished ER models. This one is very easy to implement in a relational database.

At the beginning of the process of creating an ER model there are often many m:m relationships. Many of these disappear after closer investigation.

Relationships cannot have attributes. If this seems to be the case, you need to resolve the relationship into an intersection entity plus two relationships.

The other types are less common—some express more a desired situation rather than reality, such as the m:1 relationship that is mandatory at both ends.

A normalized data model yields a normalized relational database design. Third normal form is the generally accepted standard.

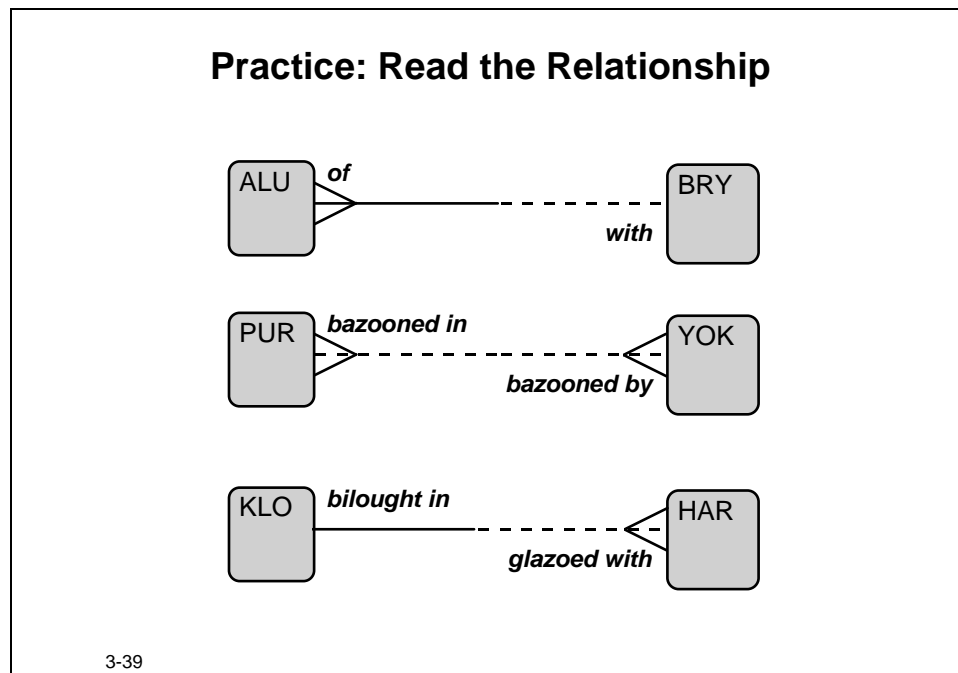
Practice 3—1: Read the Relationship

Goal

The goal of this practice is to learn to read relationships from an ER diagram.

Your Assignment

Read the diagrams aloud, from both perspectives. Make sentences that can be understood and verified by people who know the business area, but do not know how to read ER models.



Practice 3—2: Find a Context

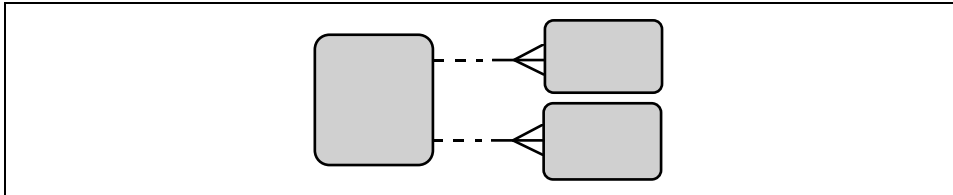
Goal

The purpose of this practice is to use your modeling skills.

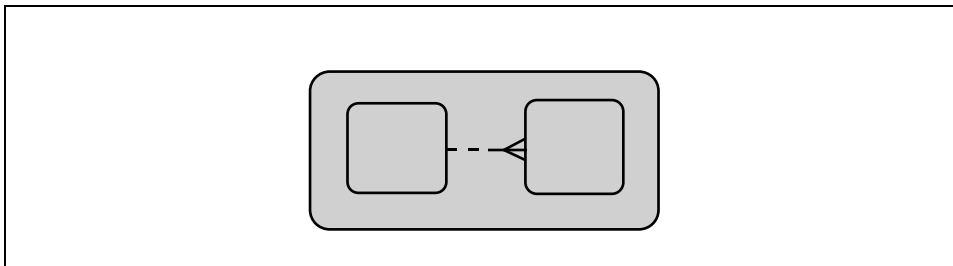
Your Assignment

Given the following ER diagrams, find a context that fits the model.

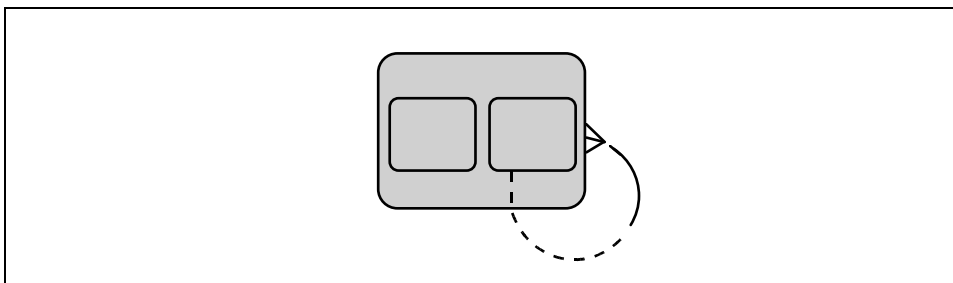
1



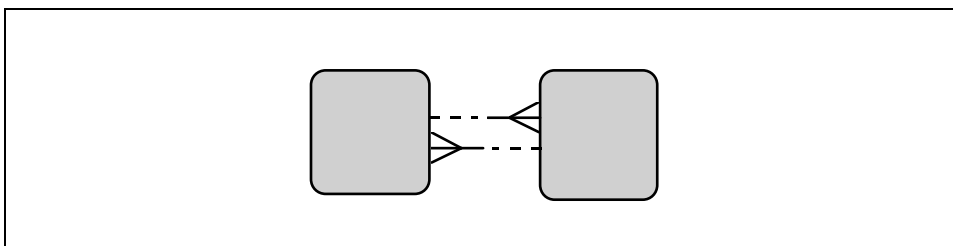
2



3



4



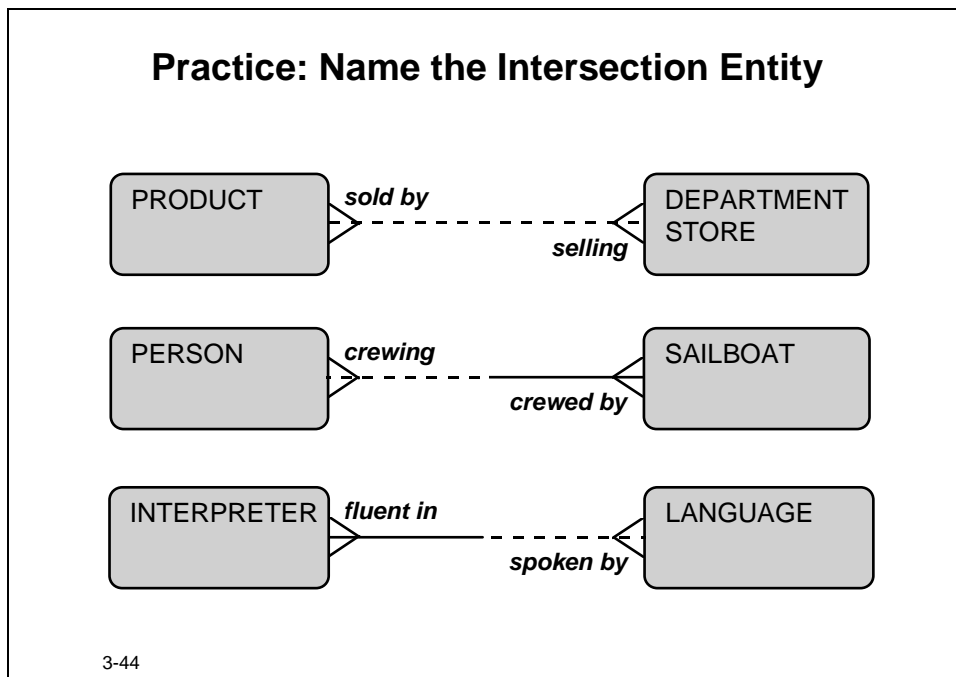
Practice 3—3: Name the Intersection Entity

Goal

The goal of this practice is to find a proper name for the intersection entity after resolving the m:m relationship.

Your Assignment

- 1 Resolve the following m:m relationships. Find an acceptable name for the intersection entity.



- 2 Invent at least one attribute per intersection entity that could make sense in some serious business context. Give it a clear name.

Practice 3—4: Receipt



Goal

The purpose of this practice is to use a simple source of real life data as a basis for a conceptual data model.

Scenario

You work as a contractor for Moonlight Coffees. Your task is to create a conceptual data model for their business. You have collected all kinds of documents about Moonlight. Below you see an example of a receipt given at one of the shops.

Your Assignment

Use the information from the receipt and make a list of entities and attributes.

```

Served by: Dennis
Till: 3 Dec 8, 4:35 pm
-----
CAPPUCC M 3.60
          * 2 7.20
CREAM      .75
          * 2 1.50
APPLE PIE  3.50
BLACKB MUF 4.50
<SUB>      16.70
tax 12%    2.00
<TOTAL>    18.70
          =====
CASH       20.00
RETURN     1.30
-----
Hope to serve you again
@MOONLIGHT COFFEES
25 Phillis Rd, Atlanta
    
```

3-45

Practice 3—5: Moonlight P&O



Goal

The purpose of this practice is to create a ER model iteratively, based on new pieces of information and new requirements.

Scenario

You are still working as a contractor for Moonlight Coffees—apparently you are doing very well!

Your Assignment

- 1 Create a entity relationship model based on the following personnel and organization information:

All Moonlight Coffee employees work for a department such as “Global Pricing” or “HQ”, or for a shop. All employees are at the payroll of one of our country organizations. Jill, for example, works as a shop manager in London; Werner is a financial administrator working for Accounting and is located in Germany.

- 2 Extend or modify the diagram based on this information:

All shops belong to one country organization (“the countries”). There is only one country organization per country. All countries and departments report to HQ, except HQ itself.

- 3 And again:

Employees can work part time. Lynn has had an 80% assignment for Product Development since the 1st September. Before that she had a full-time position.

- 4** Change the model—if necessary and if possible—to allow for the following new information.
 - a** Jan takes shifts in two different shops in Prague.
 - b** Last year Tess resigned in Brazil as a shop manager and moved to Toronto. Recently she joined the shop at Toronto Airport.
 - c** To reduce the number of direct reports, departments and country organizations may also report to another department instead of Headquarters.
 - d** The shops in Luxembourg report to Belgium.
 - e** To prevent conflicting responsibilities, employees are not allowed to work for a department and for a shop at the same time.
- 5** Would your model be able to answer the next questions?
 - a** Who is currently working for Operations?
 - b** Who is currently working for Moonlight La Lune at the Mont Martre, France?
 - c** Are there currently any employees working for Marketing in France?
 - d** What is the largest country in terms of number of employees? In terms of managers? In terms of part-timers?
 - e** When can we celebrate Lynn's fifth year with the company? When can we do the same with Tess' fifth year with Moonlight?
 - f** What country has the lowest number of resignations?

Practice 3—6: Price List

Moonlight Coffees



Goal

The purpose of this practice is to use a simple source of real life data as a basis for a conceptual data model.

Scenario

You work as a contractor for Moonlight Coffees.

Your Assignment

Make a ER model based on the pricelist from one of the Moonlight Coffee Stores.

price list

25 Phillis Road, Atlanta
visit us at www.moonlight.com

	small	medium	large	
regular coffee	2.25	2.90	3.50	
cappuccino	2.90	3.60	4.20	
café latte	2.60	3.20	3.90	
special coffee	3.10	3.70	4.40	
espresso	2.25	2.90	3.50	
coffee of the day	2.00	2.50	3.00	
<i>decaffeinated</i>	<i>.25</i>	<i>.50</i>	<i>.75</i>	<i>extra</i>
black tea	2.25	2.90	3.50	
infusions	2.60	3.20	3.90	
herbal teas	2.90	3.60	4.20	
tea of the day	2.00	2.50	3.00	
<i>decaffeinated</i>	<i>.25</i>	<i>.50</i>	<i>.75</i>	<i>extra</i>
milk	1.25	1.90	2.50	
soft drinks	2.25	2.90	3.50	
soda water	2.25	2.90	3.50	
mineral water	2.90	3.60	4.20	
apple pie				3.50
strawberry cheesecake				3.50
whole wheat oats muffin with almonds				3.90
blackberry muffin				4.50
fruitcake				4.50
cake of the day				4.00
additional whipped cream				.75

Sales Tax included
September 16

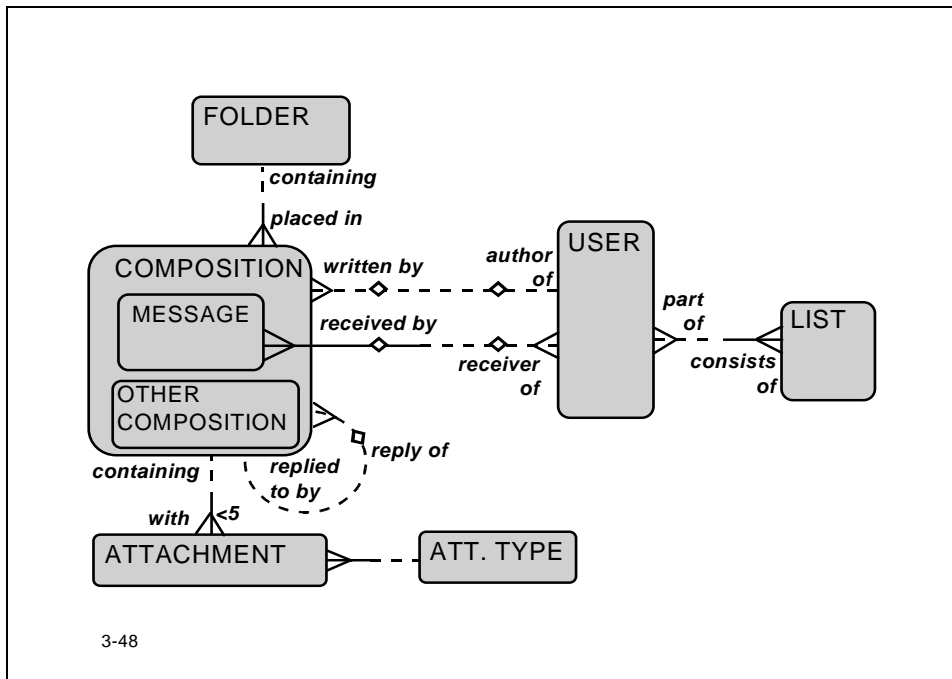
3-47

Practice 3—7: E-mail

Goal

The goal of this practice is to extend an existing conceptual data model.

Scenario.



Your Assignment

Take the given model as starting point. Add, delete, or change any entities, attributes, and relationships so that you can facilitate the following functionality:

- 1 A user must be able to create nick names (aliases) for other users.
- 2 A folder may contain other folders.
- 3 A user must be able to forward a composition. A forward is a new message that is automatically sent together with the forwarded message.
- 4 All folders and lists are owned by a user.

Challenge:

- 5 A mail list may contain both users and other lists.
- 6 A mail list may contain external addresses, like “giovanni_papini@yahoo.com”.
- 7 A nickname may be an alias for an external address.

Practice 3—8: Holiday

Goal

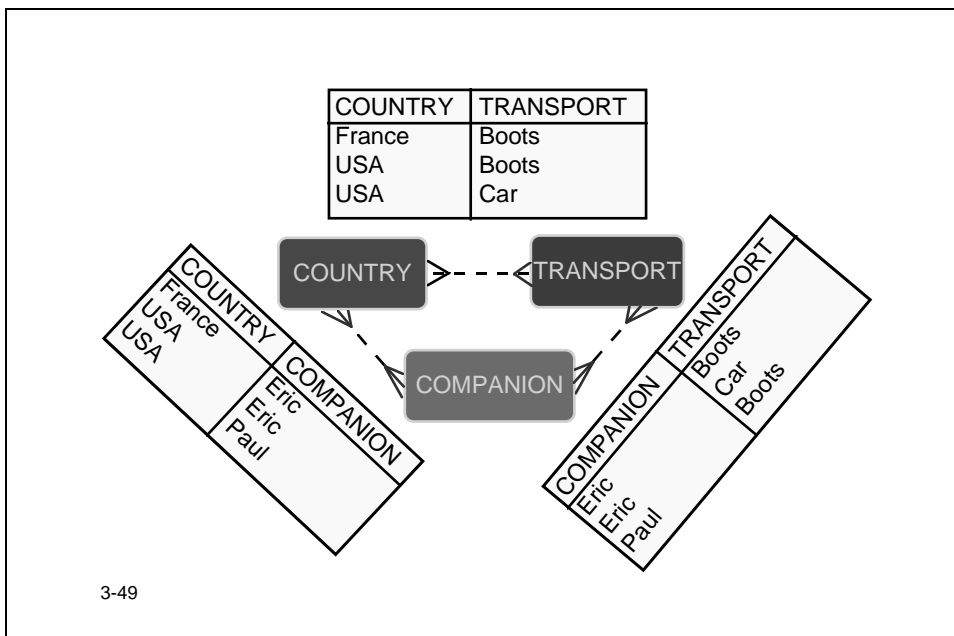
The purpose of this practice is to do a quality check on a conceptual data model.

Scenario

“Paul and I hiked in the USA. Eric and I hiked in France and we rented a car in the USA last year”.

Your Assignment

Comment on the model given below that was based on the scenario text.

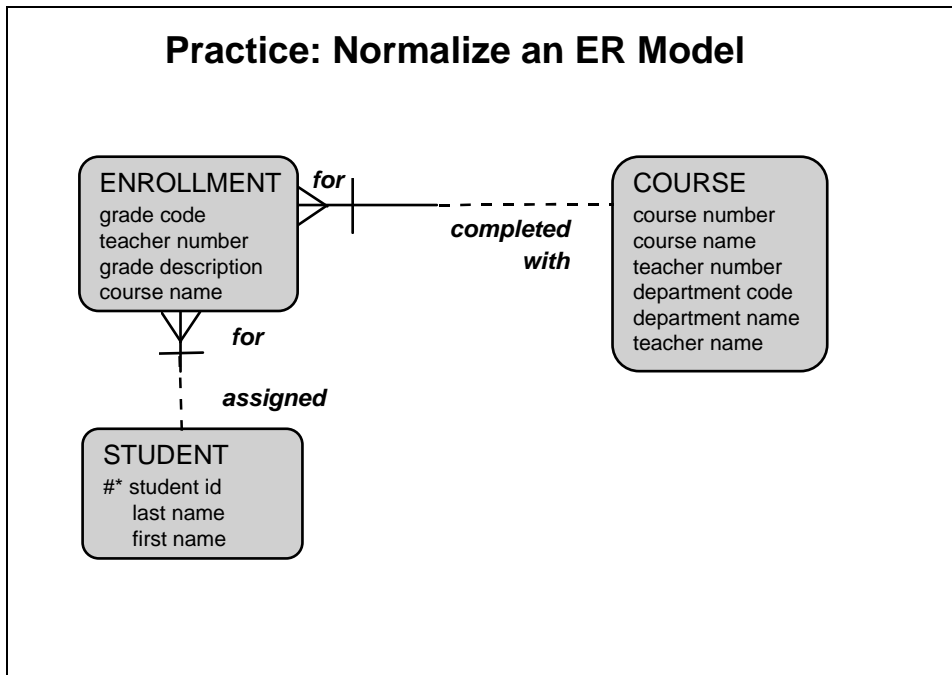


Practice 3—9: Normalize an ER Model

Goal The purpose of this practice is to place an unnormalized ER Model into third Normal Form.

Your Assignment

- 1 For the following ER Model, evaluate each entity against the rules of normalization, identify the misplaced attributes and explain what rule of normalization each misplaced attribute violates.
- 2 Optionally, redraw the ER diagram in third normal form.



4

Constraints

Introduction

This lesson is about constraints that apply to a business. Constraints are also known as business rules. Some of these constraints can be easily modeled. Some can be diagrammed but the resulting decreased clarity may not be acceptable. Some constraints cannot be modeled at all. These should be listed in a separate document.

Overview

- **Unique Identifiers**
- **Arcs**
- **Domains**
- **Various other constraints**

4-2

Topic	See Page
Introduction	2
Identification	4
Unique Identifier	6
Arcs	12
Arc or Subtypes	16
More About Arcs and Subtypes	17
Hidden Relationships	18
Domains	19
Some Special Constraints	20
Summary	24
Practice 4—1: Identification Please	25
Practice 4—2: Identification	26
Practice 4—3: Moonlight UID	28

Topic	See Page
Practice 4—4: Tables	29
Practice 4—5: Modeling Constraints	30

Objectives

At the end of this lesson, you should be able to do the following:

- Describe the problem of identification in the real world
- Add unique identifiers to your model and know how they are represented
- Recognize correct and incorrect unique identifiers
- Decide when an arc is needed in your model
- Describe the similarities between arcs and subtypes
- Describe various types of business constraints that cannot be represented in an ER diagram

Identification

What Are We Talking About?

It is not unreasonable to assume everybody knows Rembrandt was born in the Netherlands. What most people probably do not know is that Rembrandt was born on a farm as the son of Pajamas and an unknown father. Rembrandt had a twin sister. Although Rembrandt never married, he was the father of numerous children. You can easily recognize Rembrandt and his offspring as they all have four white stripes at the end of their tails.



Identification is about knowing what or who you are talking about. Obviously, the name Rembrandt is not unique to the famous painter; other human beings and even cats have the same name.

In day-to-day conversations, you can usually assume that you and the people you talk to share enough of the same context and know enough about each other's jobs and interests, to understand what you are both talking about. Language is always a rather nonspecific way to communicate, with lots of ambiguities, but people are very capable of interpretation. Computers must communicate in a more specific way that is not open to much interpretation. It would help a system to be told "Rembrandt the painter" or "Rembrandt van Rijn, born in 1606" or maybe even the combination of all: "Rembrandt van Rijn, the painter, born in 1606", to distinguish this Rembrandt from the other famous creatures with the same name.

The Problem of Identification

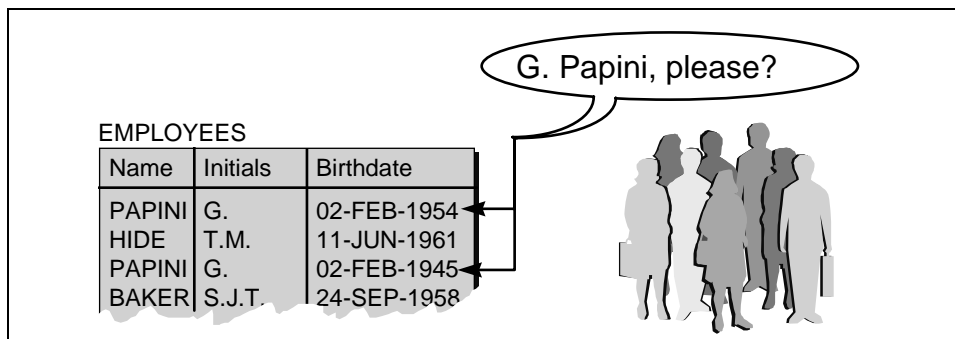
There are three sides to the problem of identification. One is identification in the real world—how do we distinguish two real world things that have very similar properties? This is the most difficult side. The second is identification within a database system—how do we distinguish rows in tables? This one is far less complex. A third issue deals with representation: how do we know what real world thing a row in a table represents?

Identification in the Real World Many things in the real world are difficult, if not impossible, to identify—distinguishing between two cabs, two customers, two versions of a contract, or two performances of the fourth string quartet by Shostakovich. As a general rule, real world things cannot be identified with certainty. You have to live with a substantial level of ambiguity. For example, how can I be sure that the car at the other side of the street with license plate MN4606 is the same car as the one I saw last week with that number? I cannot even be sure it is the same license plate. In normal circumstances there is no reason for doubt, but that is not the same as certainty. Sometimes people have their reasons for creating confusion.

Fortunately, some things in the real world are easier as they are within your reach. There you can define the rules. When a company sends out, for example, invoices, it can give every single invoice a unique number. When a business lets people create ElectronicMail usernames (identities), they can force these names to be unique.

Identification Within a Database Usually, database systems can make sure that a row is not stored twice, or, to be more exact, that a particular combination of values is not stored twice, within the same table. The technical problem is solved for you by the standard software you use.

Representation The remaining problem is to make sure that you can always know what real world thing is represented by a particular row in a table. The solution to this problem depends highly on the context. How likely do you consider it to be that two different employees for the same company have the same family name, or the same family name plus initials, or the same family name plus initials plus birthdate?

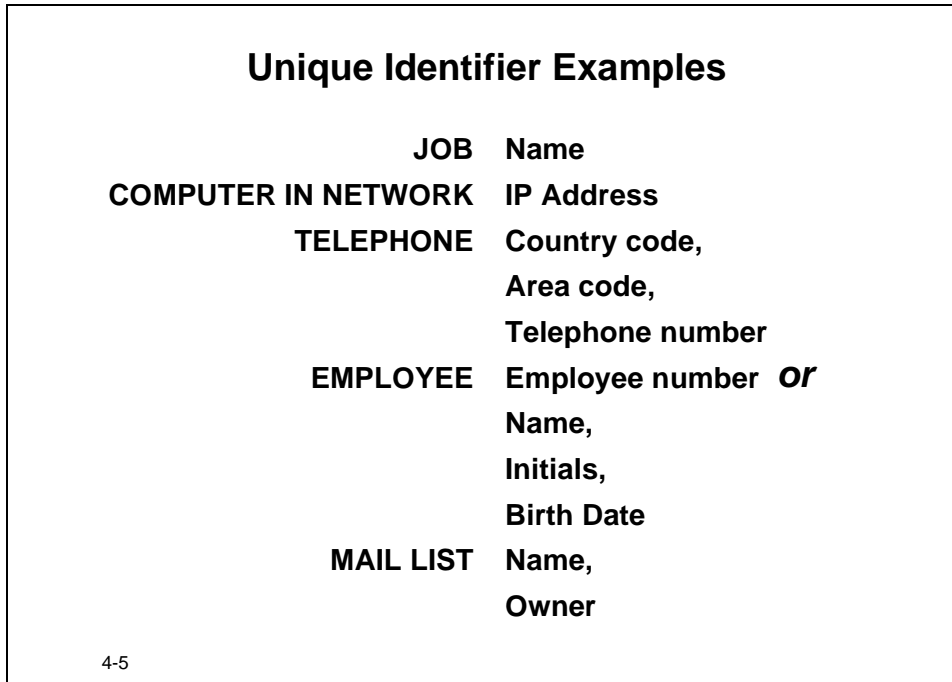


Clearly, the answer could be different when your company employs five or 50,000 employees.

Be aware that adding a new identifying attribute for EMPLOYEE, say, Id, only partially solves the above problem. It would be very useful *within* the database. It would not help much in the real world where employees usually would not know their IDs, let alone the IDs of others. This kind of Id attribute often works only as an internal, but not as an external identification.

Unique Identifier

To know what you are talking about, you need to find, for every entity, a value, or a combination of values, that uniquely identifies the entity instance. This value or combination is called the Unique Identifier for the entity.

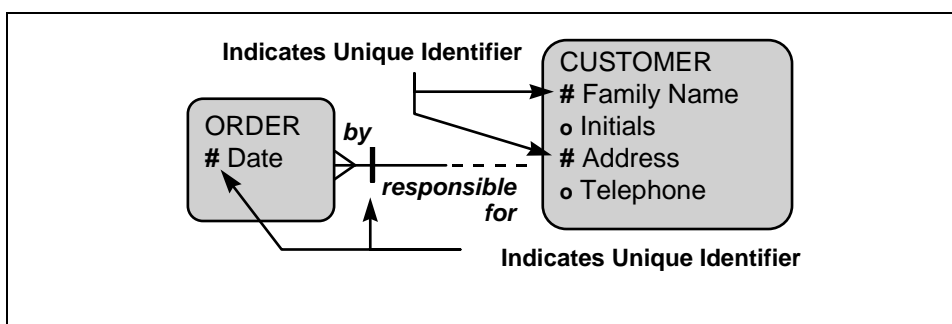


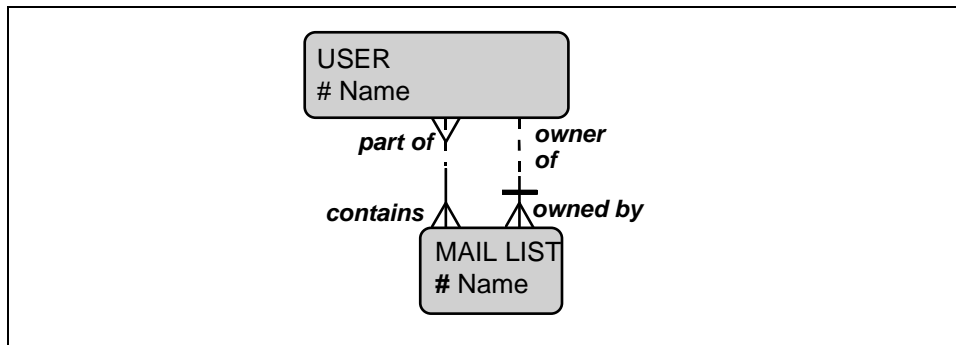
The MAIL LIST example shows that a unique identifier is not necessarily a combination of attributes: the *owner* of a MAIL LIST is actually represented by a *relationship*.

UID Representation

In an ER diagram, the components of the UID of an entity are marked:

- # for attributes.
- With a small bar across the relationship end for relationships (a barred relationship).





Single Attribute UID

The model shows that a USER of ElectronicMail is identified by attribute Name only. Many entities will be identified by a single attribute. Typical candidate attributes, if available, for single attribute UIDs are: Id, Code, Name, Description, Reference.

Multiple Attribute UID

An entity may have a UID that consists of multiple attributes; for example, a software package can usually be identified by its Name and its Version, such as Oracle Designer, version 7.0.

Composed UID

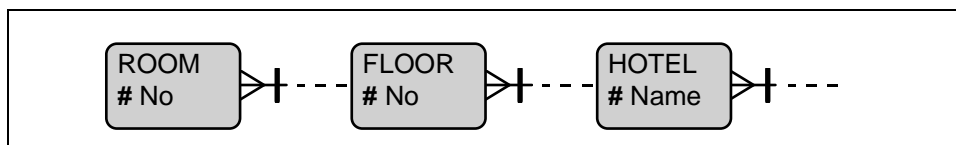
A MAIL LIST, illustrated above, is identified by the Name of the LIST plus the USER that owns the LIST. That means that the combination of OWNER and a Name of a list must be a unique pair.

This means that every USER must name their LIST instances uniquely, but need not worry about names given by other users. It also means that the system may have many LIST instances with the same name, as long as they are owned by different USERS.

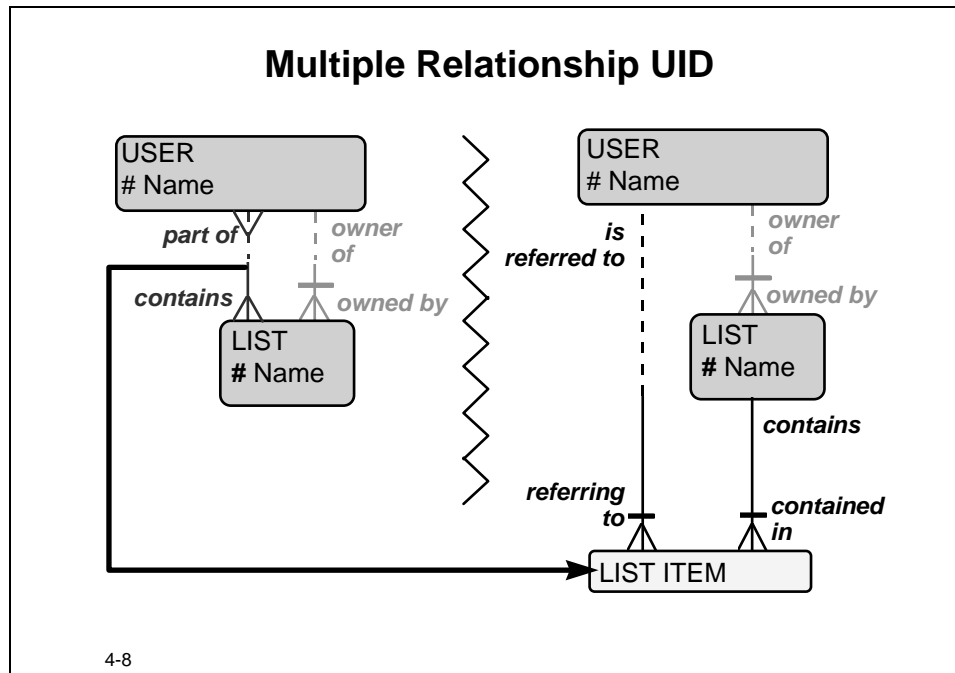
You may argue that a USER also has a composed UID, as the Name must be unique, *within* this mail system. To show this, you could add an extra high level entity, MAIL PROVIDER, plus a relationship from USER to PROVIDER. The relationship then is part of the UID of a USER.

Cascade Composed UID

It is not uncommon that an entity has a barred relationship to another entity that has a barred relationship to a third entity, and so on.



UID: Relationships Only



A Unique Identifier can also consist of relationships only.

At the lower right side of the diagram, entity **LIST ITEM** is shown, which resulted from the resolved m:m relationship between **LIST** and **USER**.

The model shows that a **LIST ITEM** is identified by the combination of the **USER** and the **LIST**. In other words, the model says that a **LIST** may contain as many **ITEMS** as you like, as long as they refer to different **USERS**.

This results in the next definition:

A Unique Identifier (UID) of an entity is a constraint that declares the uniqueness of values; a UID is composed of one or more attributes, one or more relationships, or a combination of attributes and relationships of the entity.

Consequently, not all components of the UID may be optional.

Indirect Identification

Identification regularly takes place using an indirect construction, that is, when the instance of an entity is identified only by the instance of another entity it refers to.

Examples

- In many office buildings employees are identified by their badge, which is identified by a code.
- Around the world a person is identified by the picture on their passport.

- All cows in the European Community are identified by the number of the tag they are supposed to wear in their ear.
- When you park a car at Amsterdam International Airport you enter the parking lot by inserting a credit card into a slot at the gate. The parking event is identified by the credit card of the person that parked the car. This is a double indirect identification.

Clearly, these identification constructions are not 100% reliable, but are probably as far as you can go in a situation.

The model of these indirect identifications is shown in the next illustration, at the right bottom corner. An instance of S is identified by the single instance of T it refers to. In other words, the UID consists of one relationship only.

Multiple UIDs

Entities may have multiple UIDs. Earlier, you saw the example of entity EMPLOYEE that can be identified by an Employee Number, and possibly by a combination of, for example, Name, Initials and Birth Date.

At some point in time, usually at the end of your analysis, you promote one of the UIDs to be the primary UID. All the other UIDs are called secondary UIDs.

You would usually select the UID that is most compact or easy to remember to become primary UID. The reason, of course, is that the UID leads to one or more foreign key columns in related tables. These columns should not be too sizeable. Preferably, the primary UID of an entity does not consist of optional elements.

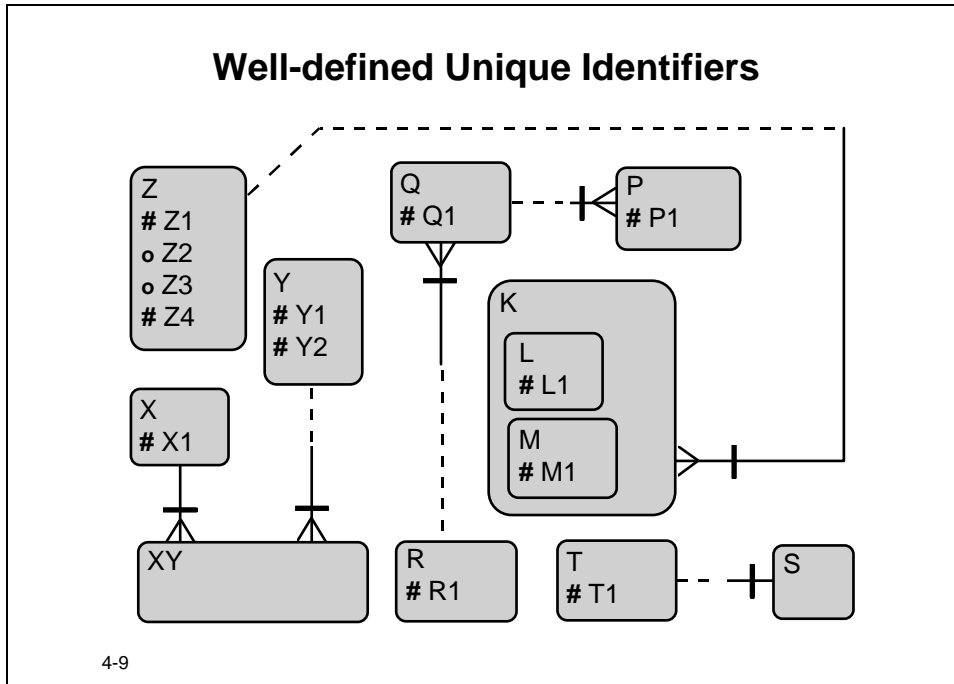
UID in Diagram

Only the primary UID is shown in ER Diagrams.

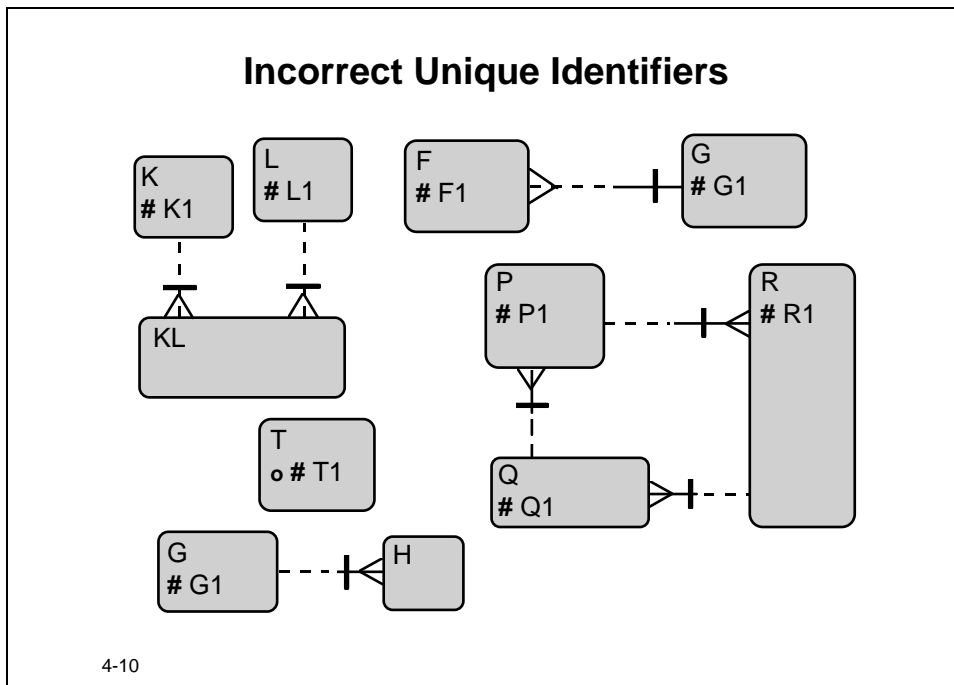
Where UIDs Lead

Unique Identifiers lead to Primary Key and Unique Key constraints.

Unique Identifier Examples

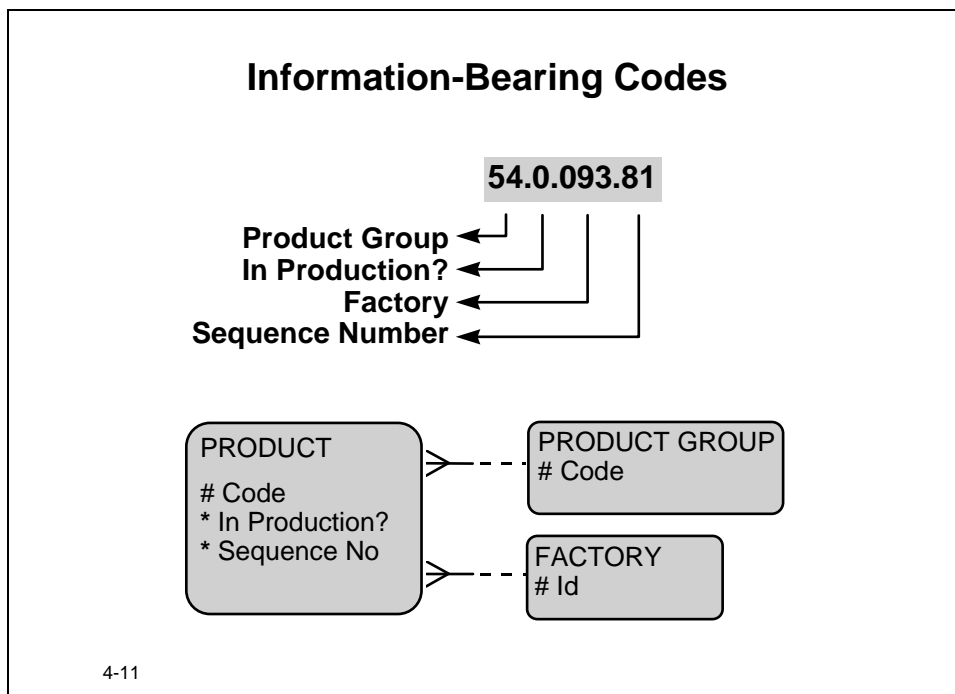


Examples of Incorrect Unique Identifiers



Information-Bearing Identifiers

When things in the real world are coded, you need to be especially careful. Codes that have been used for some time are often information bearing. An example is a company that uses product codes like 54.0.093.81, where 54 refers to the product group, 0 shows that the product is still in production, 093 identifies the factory where the product is made and 81 is a sequence number. These codes come from the time when a maximum amount of information had to be squeezed into a minimum number of bits. The example above would be modeled conceptually:



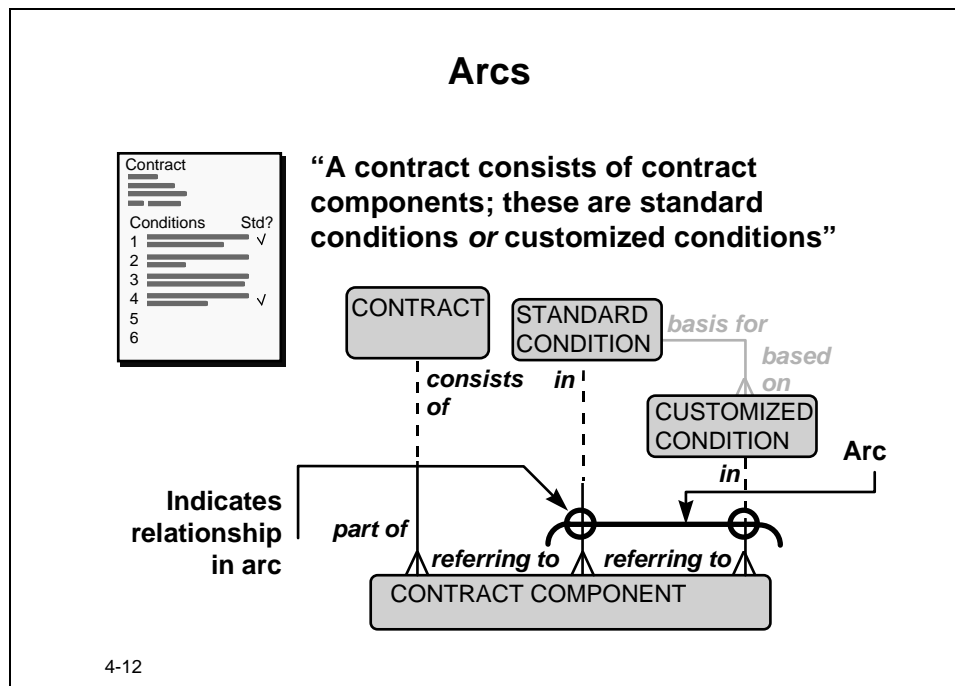
The Code attribute would contain the same codes, for reasons of compatibility, but now without meaning, as the old meaning is transferred to the attributes and relationships. Product 54.0.093.81 may now be produced by factory 123 and may no longer be in Product Group 54.

Arcs

Suppose ElectronicMail rents the Advertisement Areas that are located in their various mail screens on the Web. This renting is controlled by contracts; contracts consist of one or more standard conditions and customized conditions. This can be modeled with four entities: CONTRACT, CONTRACT COMPONENT, STANDARD CONDITION and CUSTOMIZED CONDITION. See the model below. How do we model the following constraint: every instance of CONTRACT COMPONENT refers to either a STANDARD CONDITION or a CUSTOMIZED CONDITION, but not to both at the same time?

An *arc* is a constraint about two or more relationships of an entity. An arc indicates that any instance of that entity can have only one valid relationship of the relationships in the arc at any one time. An arc models an *exclusive or* across the relationships. An arc is therefor also called exclusive arc.

There is no similar constraint construct for attributes of an entity.



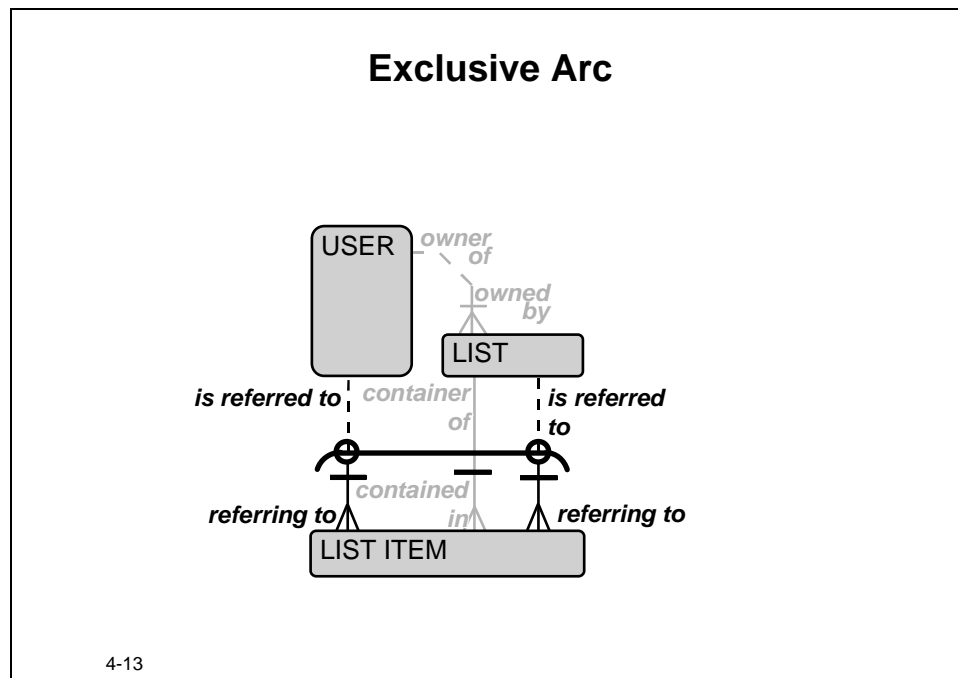
Arc Representation

The arc is drawn as an arc-shaped line, around an entity. Where the arc crosses a relationship line a small circle is drawn, but only if the relationship participates in the arc.

Mandatory Compared to Optional Relationships in an Arc

When the arc is drawn across two mandatory relationships, as in the example above, it means that every instance of CONTRACT COMPONENT must have one valid relationship. When the arc is drawn across two optional relationships, it would mean that an instance *may* have one valid relationship.

Another Arc Example



Suppose a MAIL LIST may contain USERS as well as other MAIL LISTS. This means that a particular LIST ITEM may refer to a USER or a LIST. To be more precise, it must be a reference to a USER or to a LIST, but not to both at the same time.

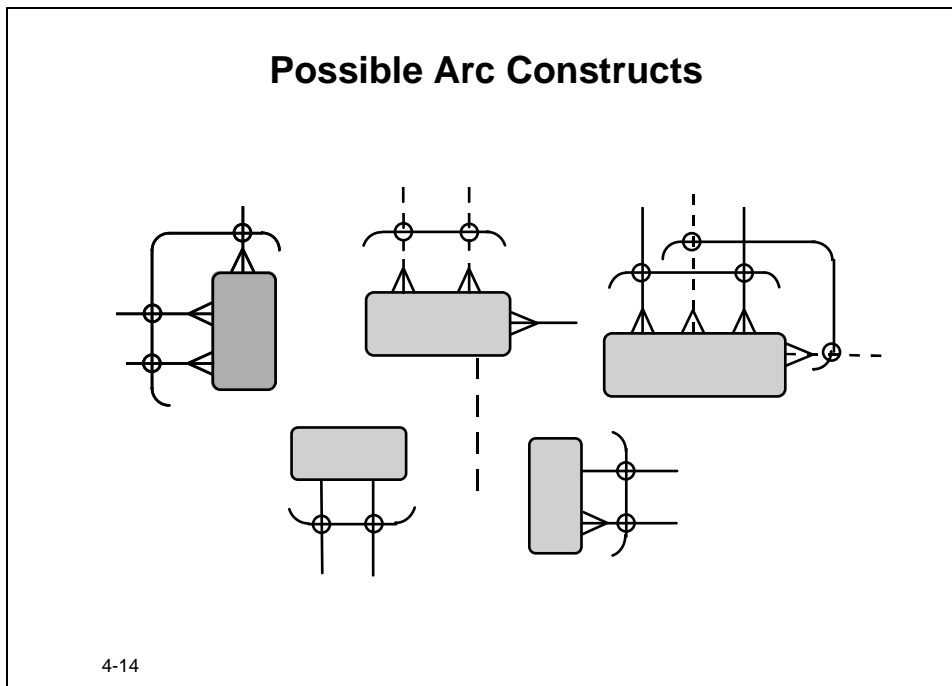
Note

- The relationship *contained in/container of* from LIST ITEM to LIST (the one that is printed in gray) is *not* part of the arc as there is no small circle at the intersection with the arc.
- A relationship that is part of a UID may also be part of an arc.
- The constraint that a LIST may only contain LISTS other than itself cannot be shown in the model.

Where Arcs Lead

An arc is normally implemented as a *check constraint* in an Oracle database. Note that a check constraint is not an ISO standard relational database object. In other words, an arc must be implemented differently in other database systems.

See Page 36



Some Rules About Arcs

- An arc always belongs to one entity.
- Arcs can include more than two relationships.
- Not all relationships of an entity need to be included in an arc.
- An entity may have several arcs.
- An arc should always consist of relationships of the same optionality: all relationships in an arc must be mandatory or all must be optional.
- Relationships in an arc may be of different degree, although this is rare.

Tips About Arcs

- Do not include a relationship in more than one arc, for clarity reasons.
- Consider modeling subtypes instead of arcs (see the next paragraph).

Incorrect Arcs

Some Incorrect Arc Constructs

- The arc “belongs” to one entity
- Relationships in the arc must be of the same optionality
- Arcs must contain at least two relationships

An arc may be correct, but is quite difficult to implement ...

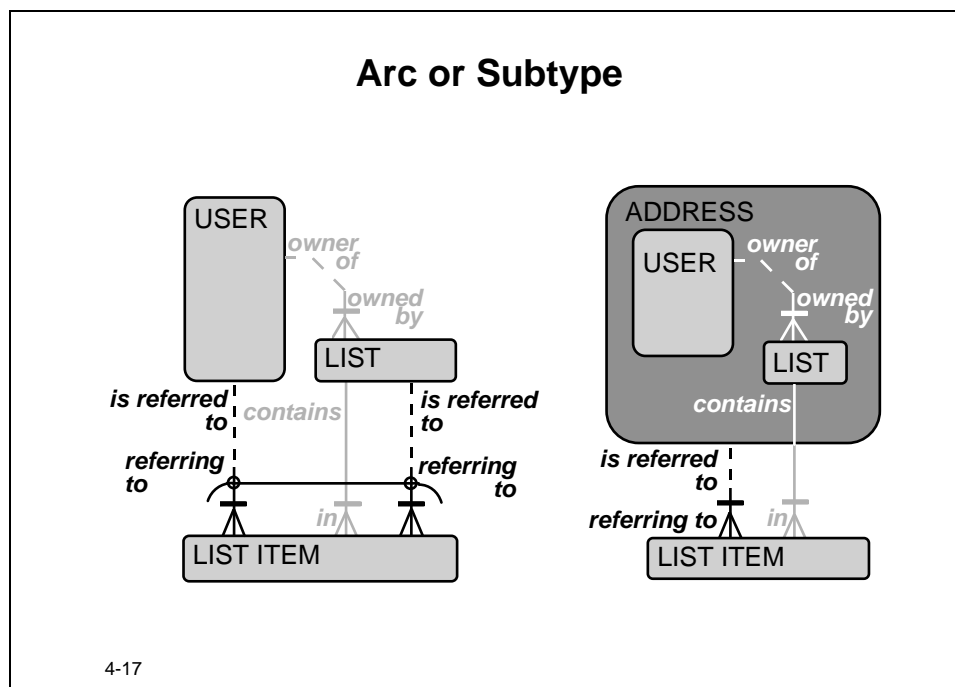
You cannot capture all possible relationship constraints with arcs. For example, if two out of three relationships must be valid, this cannot be represented. The table below shows what an arc can express.

Number of Valid Relationships in Arc Per Entity Instance			Minimum	Maximum
		$\} n$	n	n
		$\} n$	1	1
		$\} n$	0	n
		$\} n$	0	1

Arc or Subtypes

Relationships within an arc are often of a very similar nature. They frequently carry exactly the same names. If that is the case, an arc can often be replaced by a subtype construction, as the illustration shows. On the left you see the arc that contains both *referring to* relationships of LIST ITEM. In the model on the right there is only one relationship left, now connected to an entity ADDRESS, a new supertype entity of USER and LIST.

Both models are equivalent.



The model on the left emphasizes the difference between USER and LIST, which clearly exists; the other model emphasizes the commonality. This commonality is mainly a functional issue. Both USERS and LISTS can be part of a LIST and both can be used as the address in the To, Cc or Bcc field in the screen for composing a message.

Generally speaking, you can replace every arc with a supertype/subtype construction and every supertype/subtype construction with an arc.

More About Arcs and Subtypes

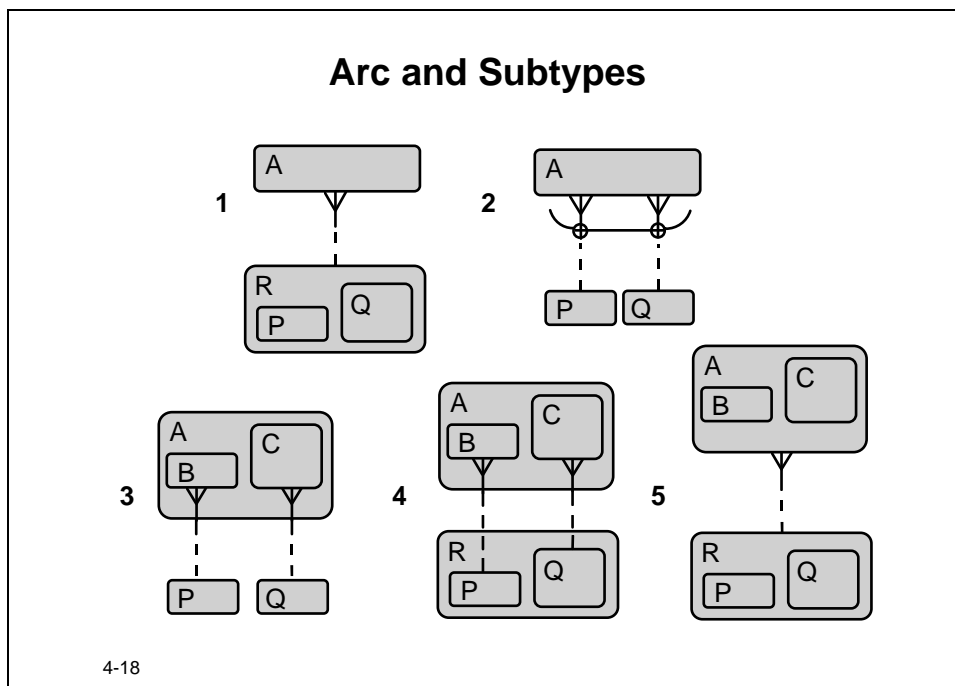
Arcs and Subtypes are similar notions. The five models that are printed below all show the same context.

Model 1 and 2 are equivalent models to what you have seen before.

If every instance of A is related to a P or a Q, then you could say there are P-related-A's and Q-related-A's. These two subtypes of A are shown in model 3.

Model 4 goes one step beyond this and shows subtypes of entity A and a supertype R of P and Q.

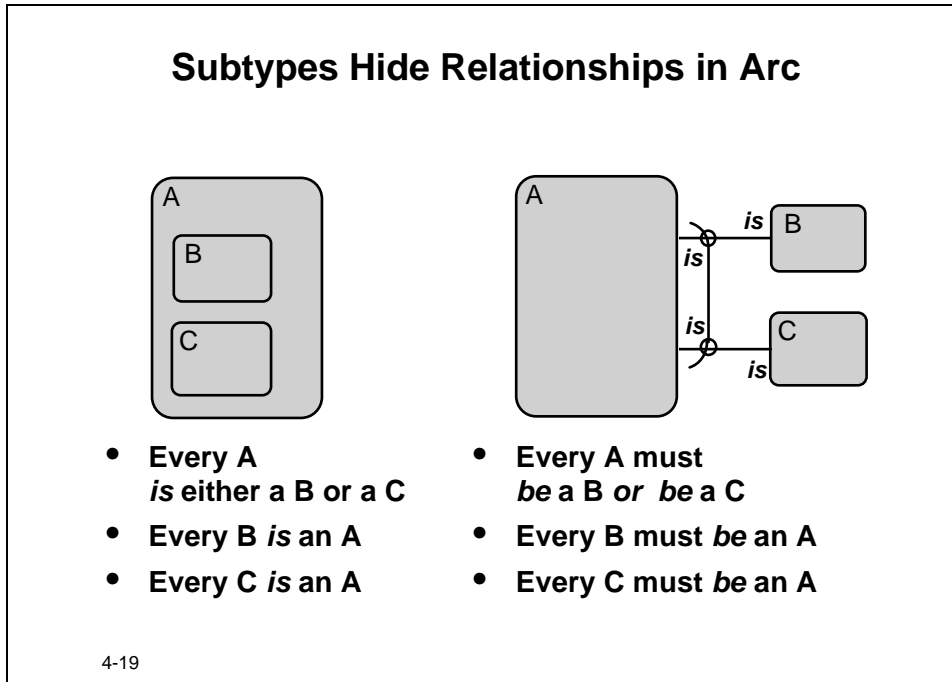
Though models 3 and 4 are completely correct, it is likely they both model something twice.



Note that only model 5 does not present the same information. In model 5, an instance of B may be related to an instance of Q, unlike that which is modeled in 3 and 4.

Hidden Relationships

Every subtype hides a relationship between the subtype and its supertype. Moreover, the relationships are in an arc, as the next illustration shows. Both relationships are mandatory 1:1 *is/is* relationships.



Domains

A very common type of *attribute constraint* is a set of values that shows the possible values an attribute can have. Such a set is called a *domain*.

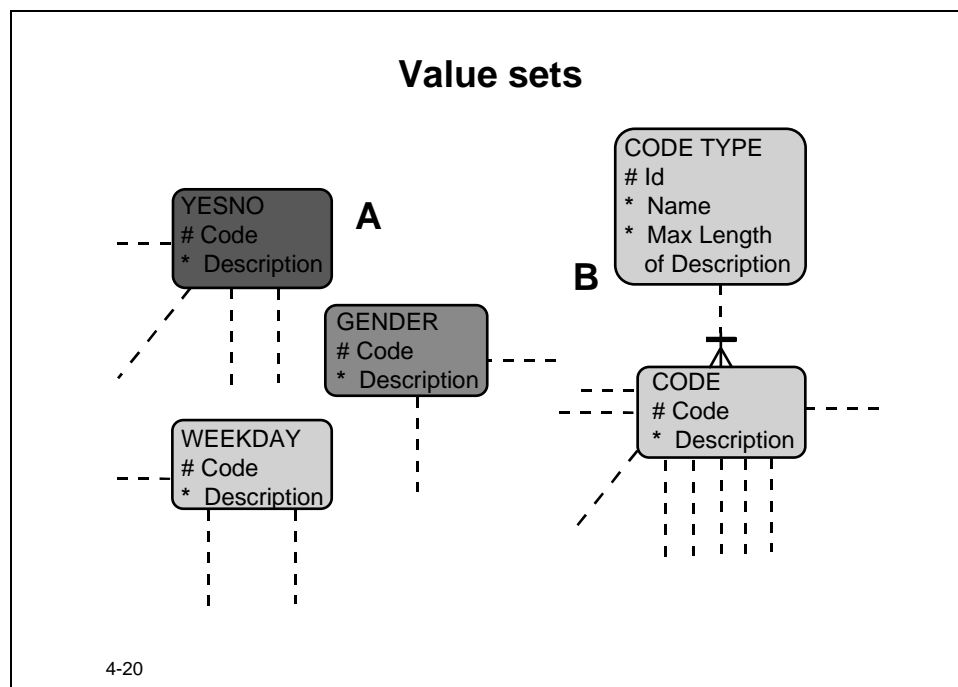
Very common domains are, for example:

- Yesno: Yes, No
- Gender: Male, Female, Unknown
- Weekday: Sun, Mon, Tue, Wed, Thu, Fri, Sat

In a conceptual data model you can recognize these as entities with, usually, only two attributes: Code and Description. These domain entities are referred to frequently but do not have any “many” relationships of their own, (see model A below). Typically, you would know all the values before the system is built. The number of values is normally low. Often you would deliver such a system with non-empty code tables

An alternative model for the (sometimes many) code entities is a more generic, two-entity approach: CODE and CODE TYPE, model B.

Model A has the advantage of fewer relationships per entity as well as easy-to-understand entities; B has obviously fewer entities and therefore will lead to fewer tables.



Domains that have a large number of values, such as all positive integers up to a particular value, are usually not modeled.

You should list and describe such a constraint in a separate document.

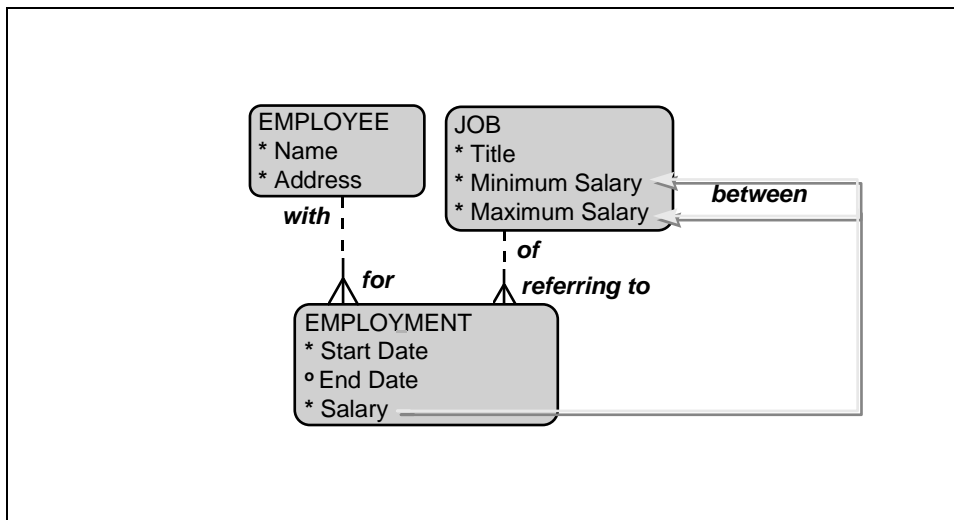
Some Special Constraints

Although an entity relationship model can express many of the constraints that are not too complex, there are many types of constraints left that cannot be modeled. These constraints must be listed on a separate document and often need to be handled programmatically.

Categories: Examples

- **Conditional domain:** The domain for an attribute depends on the value of one or more attributes of the same entity.
- **State value transition:** The set of values an attribute may be *changed to* depends on the current value of that attribute.
- **Range check:** A numeric attribute must be between attribute values of a related instance.
- **Front door check:** A valid relationship must only exist at creation time.
- **Conditional relationship:** A relationship must exist or may not exist, if an attribute (of a related entity) has a special value.
- **State value triggered check:** A check must take place when an attribute is given a value that indicates a certain state.
- There are also combinations of the above.

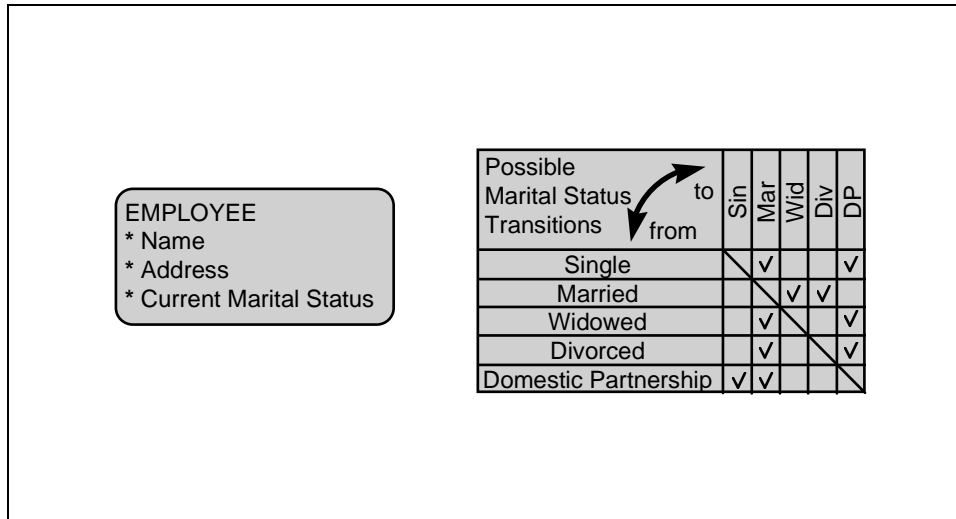
Range Check: Example



See Page 37

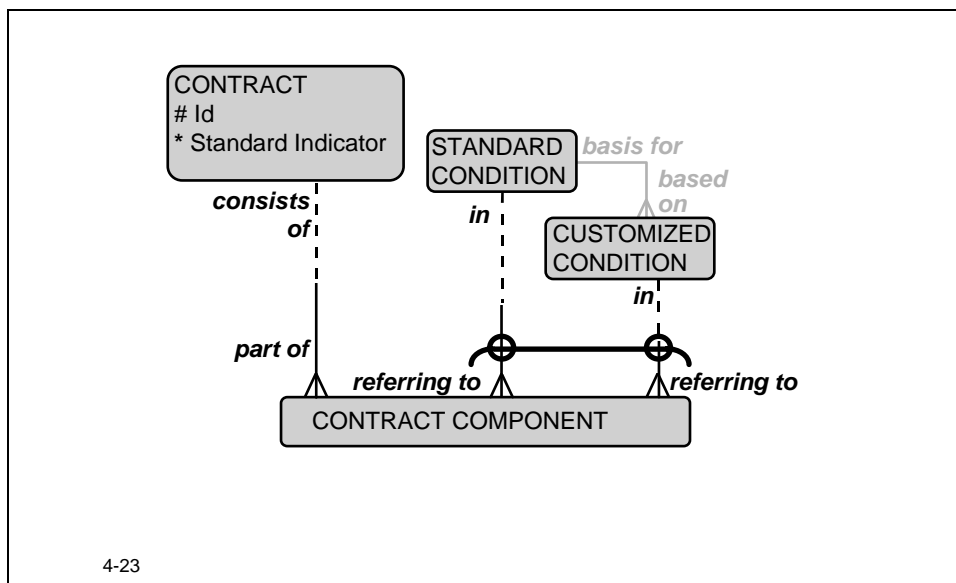
Constraint: Employee salary must be within the salary range of the job of the employee.

State Value Transition: Example



Constraint: Marital Status of employees cannot change from any value to all other values.

Conditional Relationship: Example



Constraint: If a CONTRACT has Standard Indicator set to Yes, the CONTRACT COMPONENT may not refer to a CUSTOMIZED CONDITION.

Derived Attribute?

You may argue that the attribute Standard Indicator of CONTRACT is derivable. If the contract contains CUSTOMIZED CONDITIONS, it is, by consequence, not a standard CONTRACT. This may be true, but it is not necessarily so. Suppose the contract is created in various steps, by various people with different responsibilities. Then, the creation of a CONTRACT is a process that may take days. The Standard Indicator, then, is an attribute of that process. Only when the CONTRACT is finalized, should a check be made that the Indicator corresponds with the actual STANDARD and CUSTOMIZED CONDITIONS. In those situations, the entity CONTRACT will usually have an attribute Completed Indicator that triggers the check when set to Yes.

Rules May Lead to Attributes

If you cannot capture a constraint in the model, the best you can do within the model is make the model rich enough so that a program for constraint checking performs well. Consider the rule:

If the Standard Indicator is set to No, and there is no CUSTOMIZED CONDITION, then the CONTRACT is not yet ready for being sent to the CUSTOMER.

This rule deals with a procedure and cannot be modeled as such, but it calls for an indicator at entity CONTRACT to indicate something like a Ready To Send status.

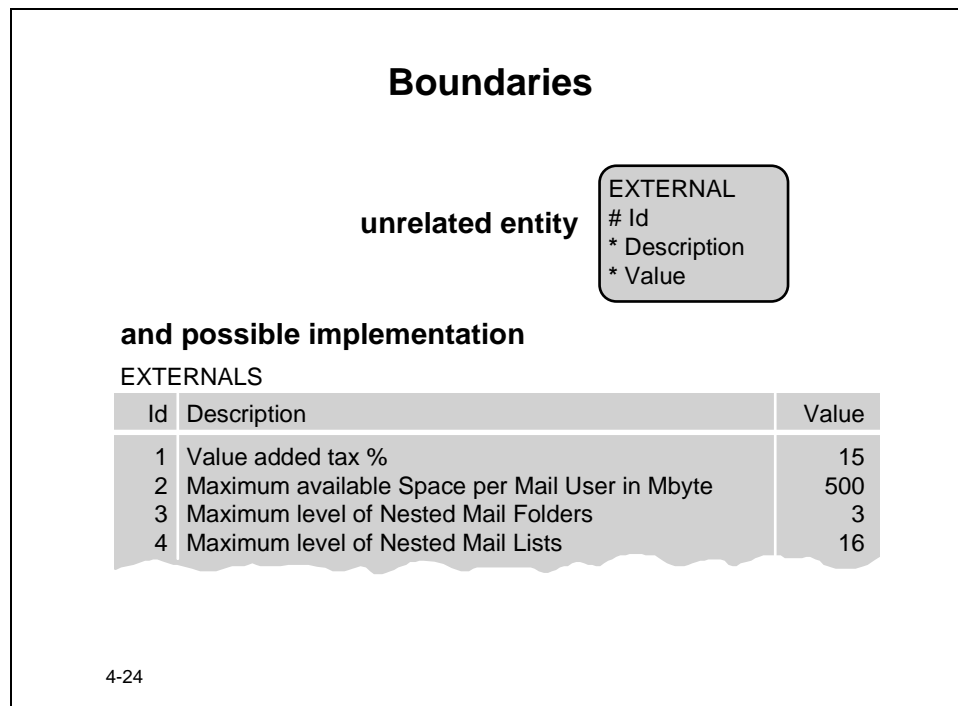
Model for Overview

An analyst often runs into constraints that cannot be modeled and thus must be documented separately. This is not a weakness of the model. An important goal of a diagram is to give an overall picture, not all the details. The model should let you view the key areas clearly.

Boundaries

More than once the checking of constraints or special rules needs to use information that is not directly related to one of the entities in the model.

Typical examples are rules and boundaries set by external sources, like a mother company or national legislation. If reasonably possible, these rules should be part of your conceptual data model, and should not be hard coded in your programs. The reason is obvious: if the rule changes, which is beyond your power, there is a chance you do not have to make changes to your programs. Only an update of a value in a table would be necessary. The time spent developing a complete model is fully justified by the programming time saved.



Summary

Entities in the real world must be individually identified before they can be represented in a database. You would not know what you are talking about, otherwise. Some entities are really difficult to identify, such as people and paintings. Some are more easy, especially when they are part of the domain as you can make up the rules, such as a unique number for each of the invoices you send to your customers. Some unique identifiers are already present in the real world, often as a combination of attributes and relationships of the entity.

Summary

- **Identification**
 - **Can be a real problem in the real world**
 - **Models cannot overcome this**
- **Entities must have at least one Unique Identifier**
- **Unique Identifiers consist of attributes or relationships or both**
- **Arcs**
- **Many types of constraint are not represented in ER model**

4-25

Arcs in a diagram represent a particular type of constraint for the relationships of one entity.

Many business constraints cannot be represented in a diagram and must be listed separately. This way the model remains clear and not too full of graphical elements.

Practice 4—1: Identification Please

Your Assignment

Describe how you would identify the following entities, making up any attributes and relationships you consider appropriate.

Practice: Identification Please

- A city
- A contact person for a customer
- A train
- A road
- A financial transaction
- An Academy Award (Oscar)
- A painting
- A T.V. show

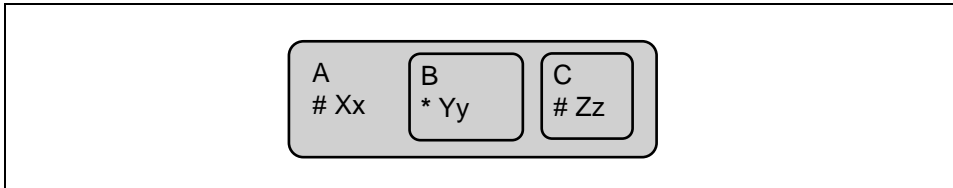
4-27

Practice 4—2: Identification

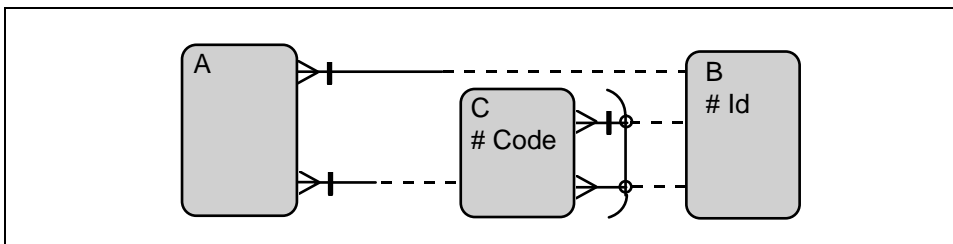
Your Assignment

Are the entities in the next diagrams identifiable?

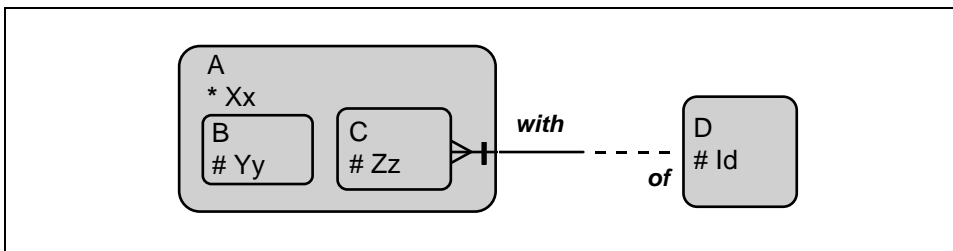
1



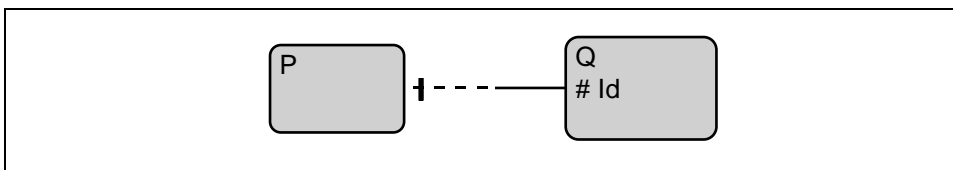
2



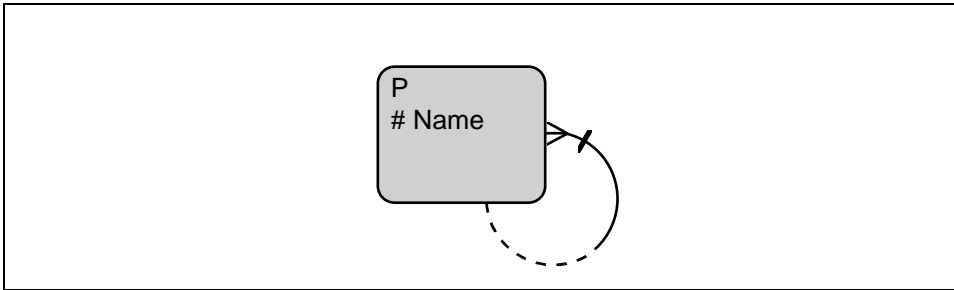
3



4

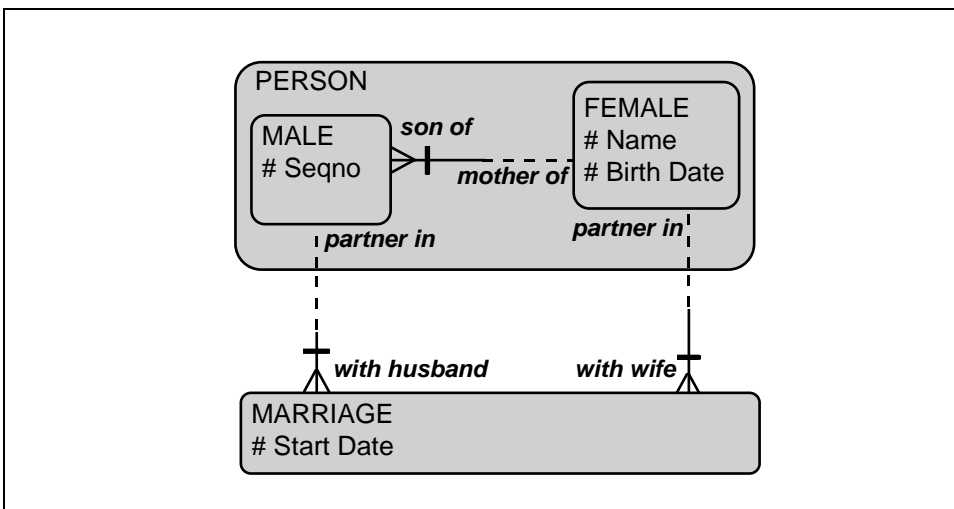


5



6

Note: the next model describes a context that may be different from the world you are familiar with.



7 Given the above model, answer the following questions.

- a Can person A marry twice?
- b Can person A marry twice on the same day?
- c Can person A marry with person B twice?
- d Can person A marry with person B twice on the same day?
- e Can person A be married to person B and person C simultaneously?
- f Can person A be married to person A?

Practice 4—3: Moonlight UID



Goal

The purpose of this practice is to define UIDs for given entities.

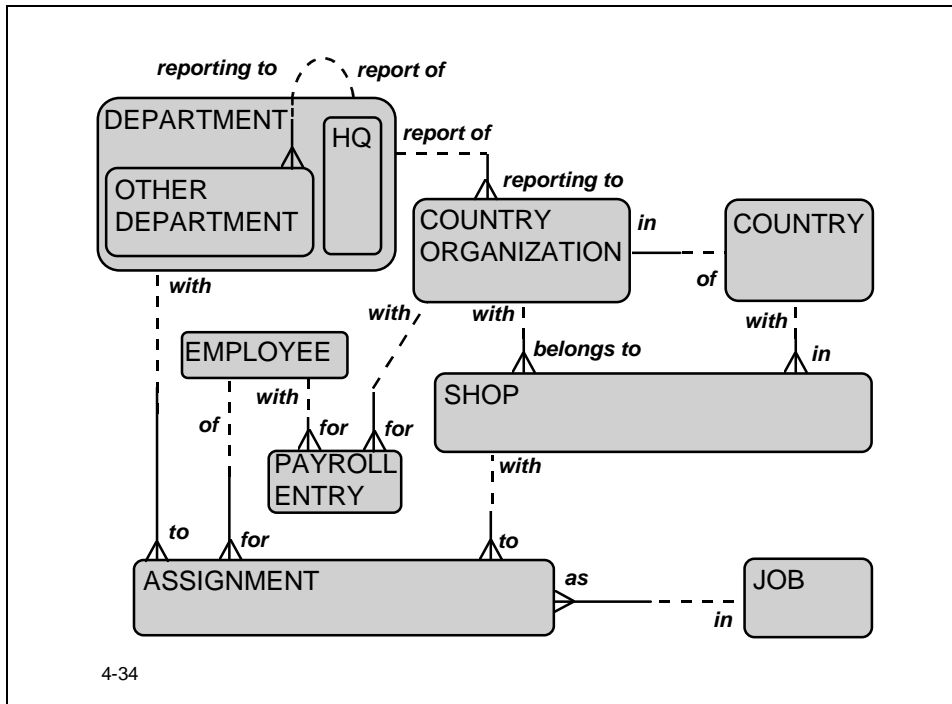
Scenario

Moonlight Coffees, organization model.

Your Assignment

Use what you know about Moonlight Coffees by now, and, most importantly, use your imagination.

- 1 Given the model below, indicate UIDs for the various entities. Add whatever attributes you consider appropriate. Country organizations have a unique “tax registration number” in their countries.
- 2 Are there any arcs missing?



Practice 4—4: Tables

Goal

The purpose of this practice is to match a given context with a ER model.

Your Assignment

Read the text on ISO Relational tables.

Do a quality check on the ER model based on the quoted text and what you know about this subject. Also list constraints that are mentioned in the text but not modeled.

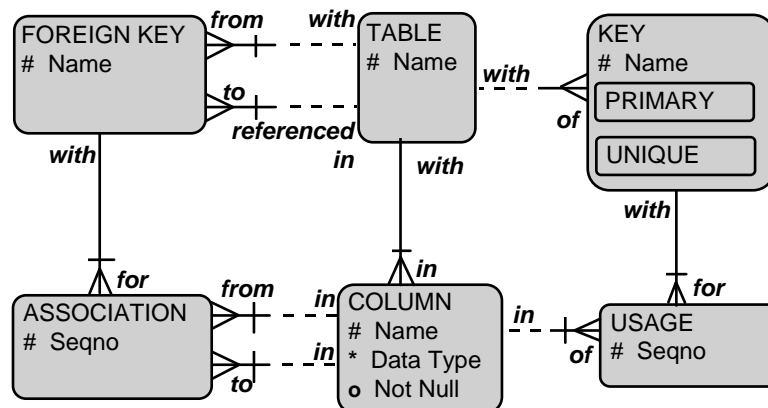
Practice: Table 1

“In a relational database system, data is stored in tables. Tables of a database user must have a unique name. A table must have at least one column. A column has a unique name within the table. A column must have a data type and may be Not Null.

Tables can have one primary key and any number of unique keys. A key contains one or more columns of the table. A column can be part of more than one key.

A table can have foreign keys. A foreign key always connects one table with another. A foreign key consists of one or more columns of the one table that refers to key columns of the other table.

The sequence of columns within the key and foreign key is important.”



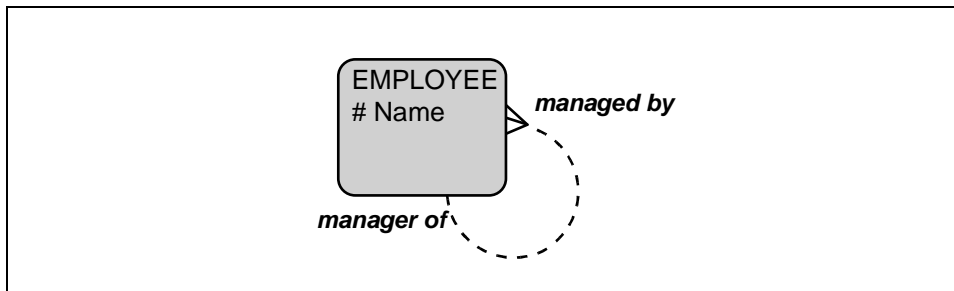
Practice 4—5: Modeling Constraints

Goal

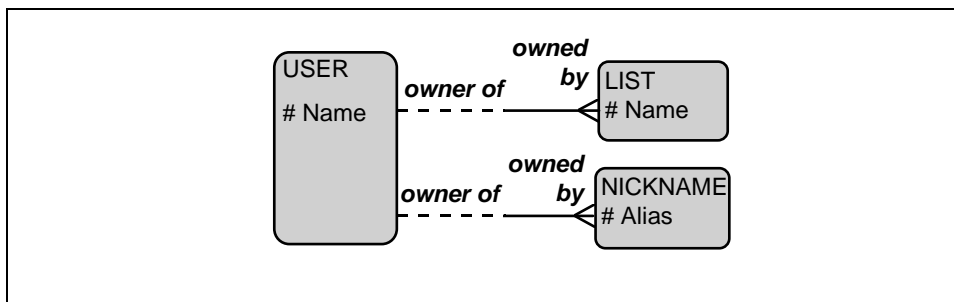
The purpose of this practice is to learn what constraints can be modeled and how, and which cannot be modeled.

Your Assignment

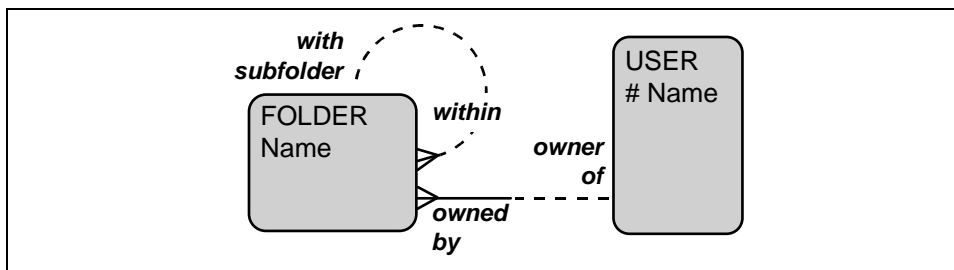
Change the diagrams to model the constraint given.



- 1 Every EMPLOYEE must have a manager, except the Chief Executive Officer.



- 2 A user may not use the same name for both NICKNAME and LIST name.



- 3 A top level FOLDER must have a unique name per user; sub folders must have a unique name within the folder where they are located.

Modeling Change

Introduction

Every update of an attribute or transfer of a relationship means loss of information. Often that information is no longer of use, but some systems need to keep track of some or all of the old values of an attribute. This may lead to an explicit time dimension in the model which is usually quite a complicated issue.

Lesson Aim

Time is often present in a business context, as many entities are in fact a representation of an event. This lesson discusses the possibilities and difficulties that arise when you incorporate time in your entity model.

Overview

- **Date and time**
- **Modeling change over time**
- **Prices change**
- **Journalling**

5-2

Topic	See Page
Introduction	2
Time	4
Date as Opposed to Day	5
Entity DAY	6
Modeling Changes Over Time	7
A Time Example: Prices	10
Journalling	17
Summary	19
Practice 5—1: Shift	20

Topic	See Page
Practice 5—2: Strawberry Wafer	21
Practice 5—3: Bundles	22
Practice 5—4: Product Structure	24

Objectives

At the end of this lesson, you should be able to do the following:

- Make a well considered decision about using entity DATE or attribute Date
- Model life cycle attributes to all entities that need them
- List all constraints that arise from using a time dimension
- Cope with journalling

Time

Modeling Time

In many models time plays a role. Often entities that are essentially *events* are part of a model, for example, PURCHASE, ASSIGNMENT. One of the properties you record about these entities is the date or date and time of the event. Often the date and time are part of a unique identifier.

Change and Time

- **Every update means loss of information.**
- **Time in your model makes the model more complex.**
- **There are often complex join conditions.**
- **Users can work in advance.**
- **When do you model date/time as an entity?**
- **What constraints do arise?**
- **How do you handle journalling?**

A second time-related issue often helps to increase the usability of a system dramatically. By adding dates like Start, Expiry, End Date, to data in the system, you allow users to work in advance. Suppose a particular value, say the price of gas or diesel, will change as of January 1. It is very useful to be able to tell the system the new value long before New Year's Eve. By adding a time dimension to the model you make the system independent of the *now*.

As always, there is a price for adding things such as this. Adding a time dimension to your conceptual data model makes the model considerably more complex. In particular, the number of constraints and business rules that must be checked will increase.

A third time-related issue in conceptual data models is connected to the concept of *logging* or *journalling*. Suppose you allow values to be updated, but you want to keep track of some of the old values. In other words, what do you do when you need to keep a record of the history of attribute values, of relationships, of entire entities?

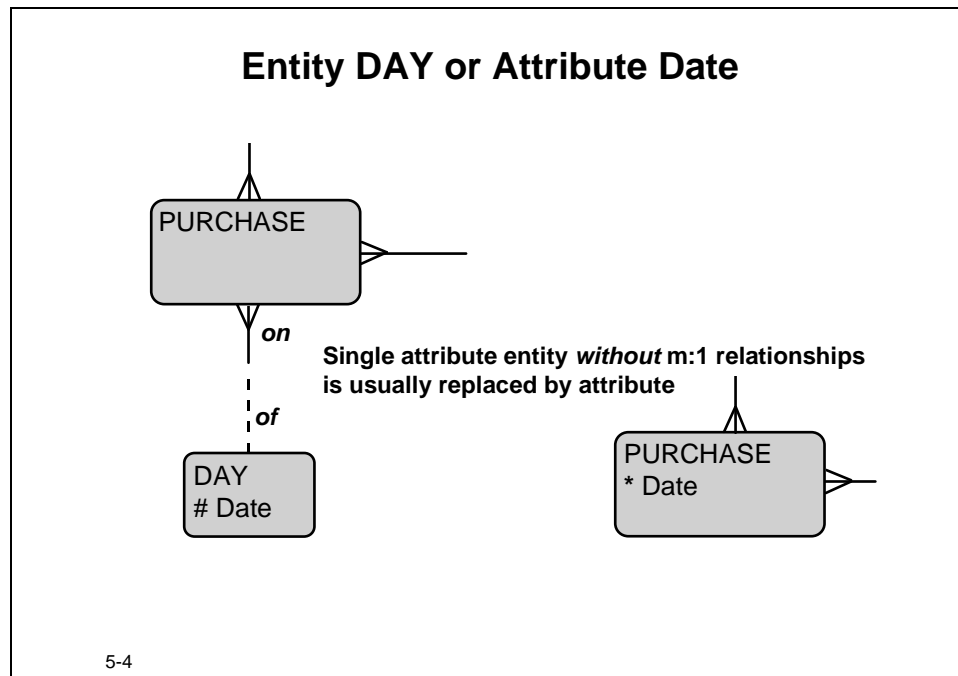
The following issues arise:

- When do you model date/time as an entity, and when as an attribute?
- How do you handle the constraints that arise in systems that deal with time-related data?
- How do you handle journalling?

Date as Opposed to Day

Probably all current operating systems and database systems have types “date” and “time” available that know, for example, that 29-OCT-1983 was a Saturday in the 10th month, called October, of 1983.

Some database systems, like Oracle, see time as a component of a date and store them in one. Knowing that, you are likely to decide that dates can be modeled as attributes with the format date.



A day, however, is not just a date. My great-grand father was born on a day in 1852, but the exact date is unknown. A Genealogical Register System should therefore be able to store fragments of a date, such as “1852”, or even a description, such as “around 1765”.

Systems that store historical information often have to deal with several dates for one event, according to multiple sources with nonidentical information.

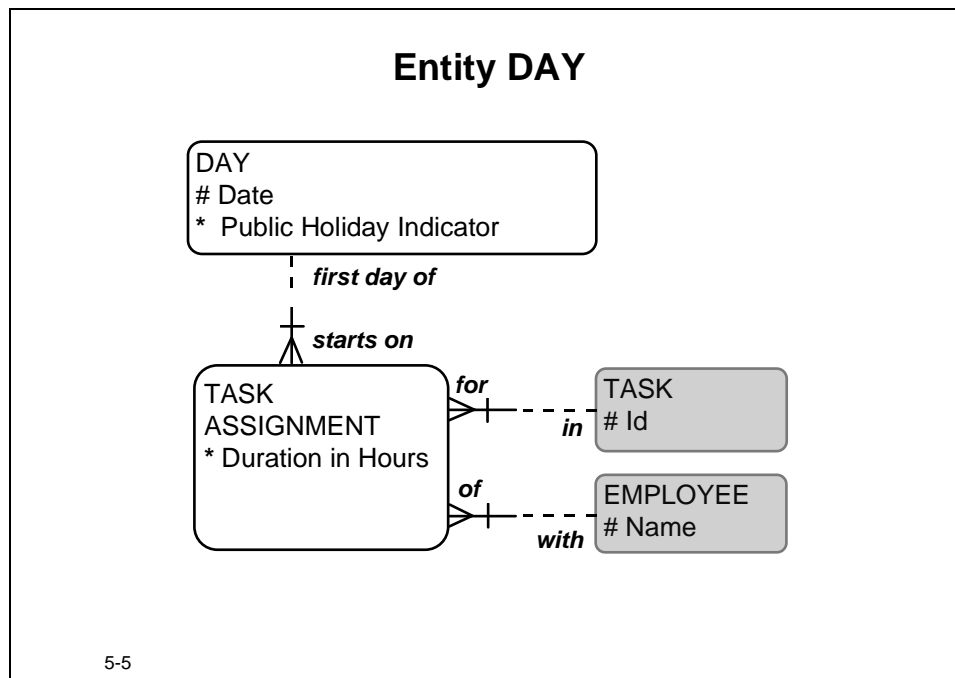
Some systems have to take dates in conjunction with the reliability of that date. Clearly, in these cases a simple attribute would not suffice.

Loosely speaking, when you are interested in the date only, and these dates are known to the user, model an attribute; on the other hand, when you are interested in the day, model it as an entity with attribute Date, which is possibly a unique identifying attribute.

Entity DAY

It is not only systems that deal with historical information that struggle with dates. Sometimes a system needs to know more about a day than can be derived from its date. A planning system, for example, often needs to know if a particular day is a public holiday. Many data warehouse systems use a calendar that is different from the normal one, for example, where a year is divided into four-week periods or 30 day Months or Quarters where Q1 starts in the middle of May.

Some warehouses need weather information about days in order to do statistical analysis about the influence of the weather on, for example, their sales. In these cases a day has attributes or relationships of its own and should be modeled as entity DAY.



The above model shows part of a planning system where tasks are assigned to employees. Tasks may take from a few hours to, at maximum, several days.

Based on this model, table TASK_ASSIGNMENTS will contain a date column that is a foreign key column to the DAYS table.

Date and Time

As stated earlier, an Oracle DATE column always contains date and time. This needs some special attention as two DATE columns may apparently contain the same date but they are not equal because of a difference in their time component.

While modeling, always make explicitly clear when time of the day is an issue, for instance, by naming the attribute DateTime. As soon as hours and minutes play a role, the concepts of “time zone” and “daylight saving time” may become important.

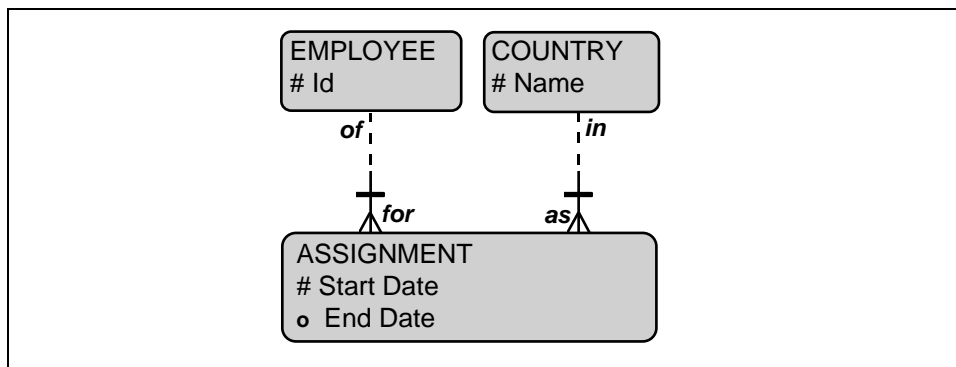
Modeling Changes Over Time

Date and Time in your models may substantially increase the complexity of your system, as the next example shows.

The context for this example is that of an Embassy Information System, but could have been chosen from almost any business area.

Embassy employees have an assignment for a country, but, of course, the assignments may change over time. Therefore, the model would need an entity ASSIGNMENT with a mandatory attribute Start Date and an optional End Date. Start Date is modeled as part of the UID for ASSIGNMENT. This means that the model allows an employee to have two assignments in the same country, as long as they start on different days. It also allows the employee to have two assignments that start on the same day, as long as these are for different countries.

Suppose we know today that Jacqueline will switch from Chili to Morocco on the first of next month. This fact can be fed into the system immediately, by creating a new instance of ASSIGNMENT with a Start Date that is still in the future at create time. The future users will appreciate this kind of functionality.



End Date Redundant?

You may argue that attribute End Date of ASSIGNMENT is redundant because Jacqueline's assignments follow each other: the End Date of Jacqueline's assignment in Chili matches the Start Date of the one in Morocco. This may be true, but it does not take into consideration that embassy people may take a leave and return after a couple of years. In other words, if you do not model attribute End Date you ignore the possibility that the assigned periods of a person are not contiguous.

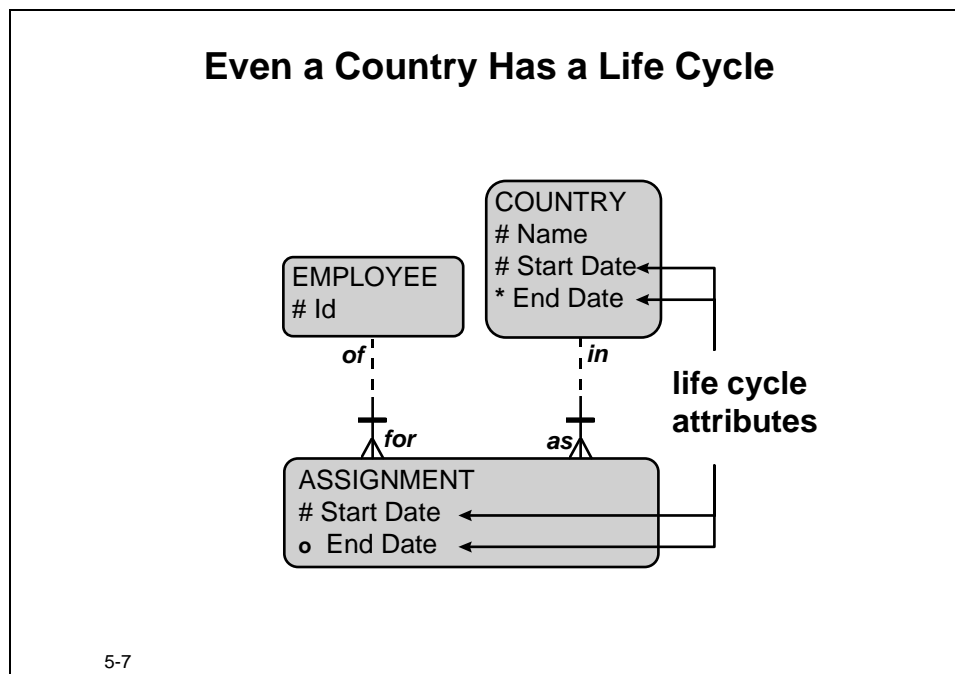
Note that the model does allow an employee to have two assignments in, for example, Honduras, that overlap! The unique identifier does not protect the data against overlapping periods. Adding End Date to the UID does not help.

You would need a whole series of extra constraints to cope with this.

Countries Have a Life Cycle Too

Suppose the Embassy Information System contains data that goes back to at least the late eighties. In those days the USSR and Zaire were still countries. Suppose there are ASSIGNMENTS that refer to the USSR and Zaire. In the case of Zaire, you could consider an update of the Name of the COUNTRY: Democratic Republic Congo is essentially just the new name for Zaire. In case of the USSR this would not make sense. There is not a new name for the old country. The old country simply ceased to exist when it broke into several countries. Although the concept of a country seems very stable, countries may change fundamentally during the lifetime of the information system.

This leads to the next model.



Time-related Constraints

Be aware of the numerous constraints that result from the time dimension! Here is a selection:

- An ASSIGNMENT may only refer to a COUNTRY that *is valid* at the Start Date of the ASSIGNMENT.
- The obvious one: End Date must be past Start Date.
- A business rule: ASSIGNMENT periods may not overlap. The Start Date of an ASSIGNMENT for an EMPLOYEE may not be between any Start Date and End Date of an other ASSIGNMENT for the same EMPLOYEE.
- As for the previous constraint, but for End Date.

- You would probably not allow an ASSIGNMENT to be transferred to another COUNTRY, unless the ASSIGNMENT has not yet started, that is, the Start Date of the ASSIGNMENT is still in the future.

This is an example of *conditional nontransferability*.

For updates of the attribute Start Date here are some possible constraints:

- A Start Date of an ASSIGNMENT may be updated to a *later* date, unless this date is later than the End Date (if any) of the COUNTRY it refers to.
- A Start Date of an ASSIGNMENT may be updated to a *later* date, if the current Start Date is still in the future.
- A Start Date of an ASSIGNMENT may be updated to an *earlier* date, unless this date is earlier than the Start Date of the COUNTRY it refers to.
- A Start Date of an ASSIGNMENT may be updated to an *earlier* date, if this new date is still in the future.
- A Start Date of a COUNTRY may be updated to a *later* date, if there are no ASSIGNMENTS that would get disconnected.

Similar constraints apply to attribute End Date.

Referential Logic

Note that, except for two, these constraints result from *referential logic* only. There may be more additional business constraints.

Imagine the sheer number of constraints if a time-affected entity is related to several other time-affected entities! Fortunately, these constraints all have a similar pattern; these result from the referential, time related, logic.

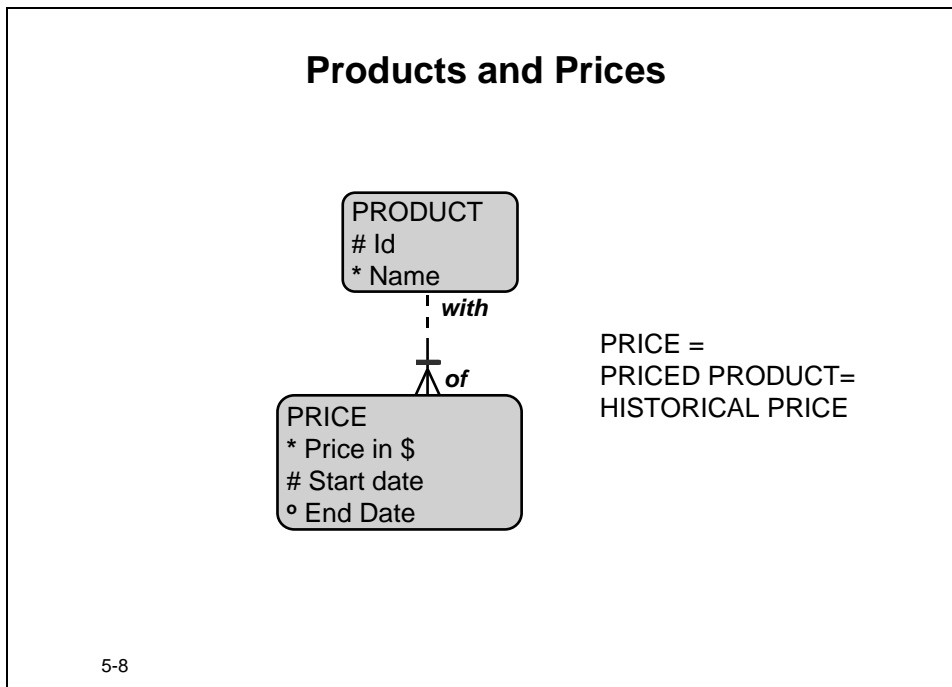
Not in Diagram

You cannot model any of these constraints in the diagram as they all have to be listed separately.

Implementation

In an Oracle environment, one of these constraints can be implemented as a check constraint, (End Date must be later than Start Date). All the others will be implemented as database triggers.

A Time Example: Prices

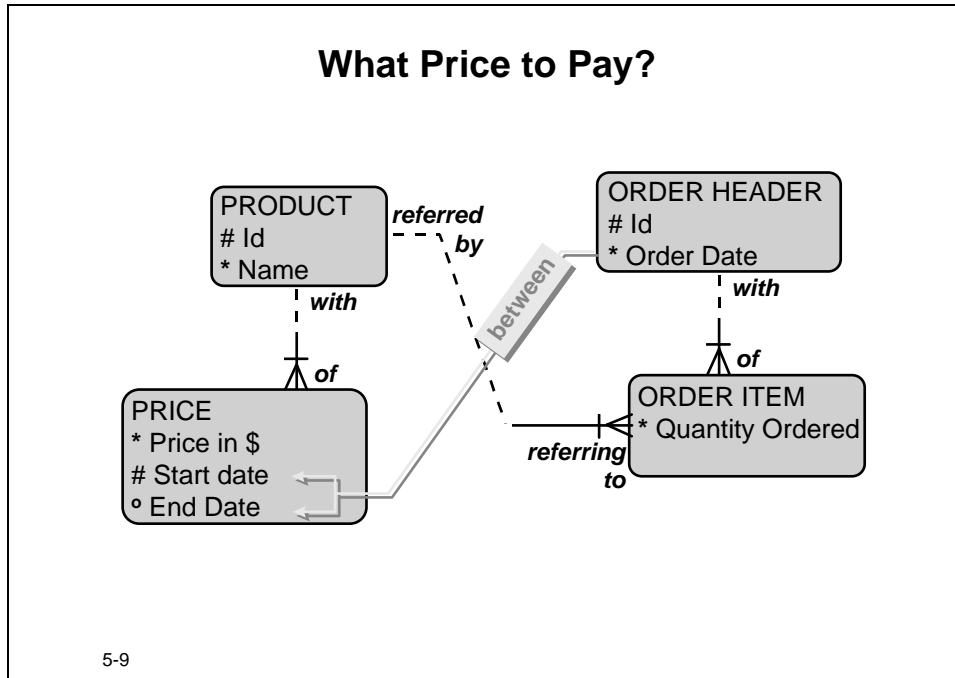


Products have a price. Prices change. Old prices are probably of interest. That leads to a model with entities **PRODUCT** and **PRICE**. The latter entity contains the prices and the time periods they are applicable. In real-life situations you find the concept of **PRICE** also named **PRICED PRODUCT**, **HISTORICAL PRICE** (and less appropriate: price list or price history); all these names more or less describe the concept.

You may argue the need for an **End Date** attribute. If the various periods of a product price are contiguous, **End Date** is obsolete. If, on the other hand, the products are not always available, as in the fruit and vegetable market, the periods should have an explicit **End Date**.

Introducing Order Header and Order Item

See Page 27



Here, entities ORDER HEADER and ORDER ITEM are introduced. An ORDER HEADER holds the information that applies to all items, like the Order Date and the relationship to the CUSTOMER that placed the order or the EMPLOYEE that handled it. (For clarity, these relationships are not drawn here.) The ORDER ITEM holds the Quantity Ordered and refers to the PRODUCT ordered. The price that must be paid can be found by matching the Order Date between Start Date and End Date of PRICE. Note that you cannot model this “between relationship”.

This model is a fairly straightforward product pricing model and is often used.

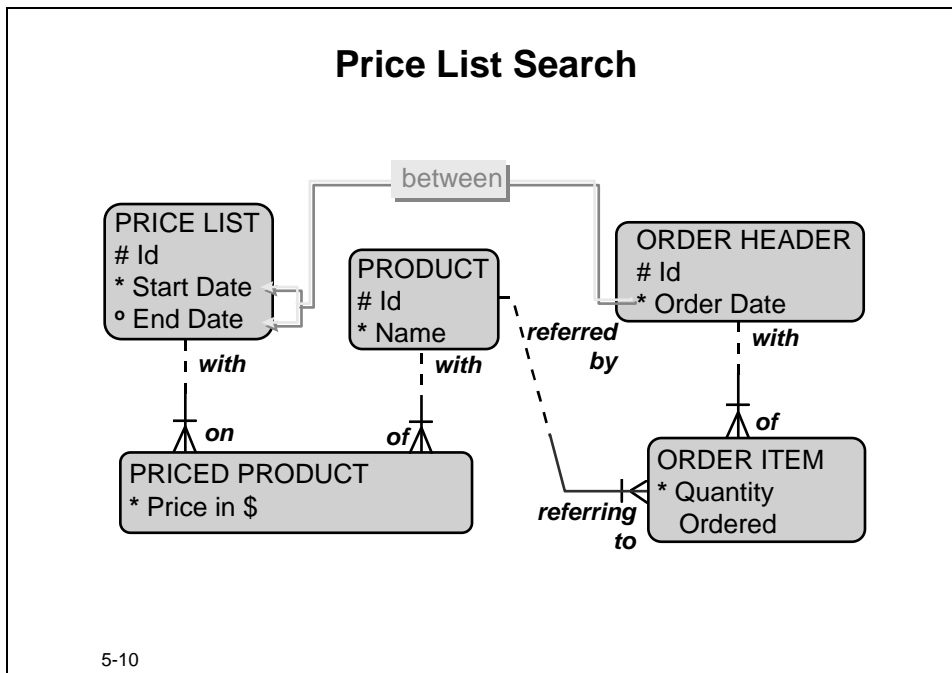
Order

Note that the concept of an order in this model is composed of ORDER HEADER and ORDER ITEM.

To find the order total for an order, it would need a join over four tables.

Price List

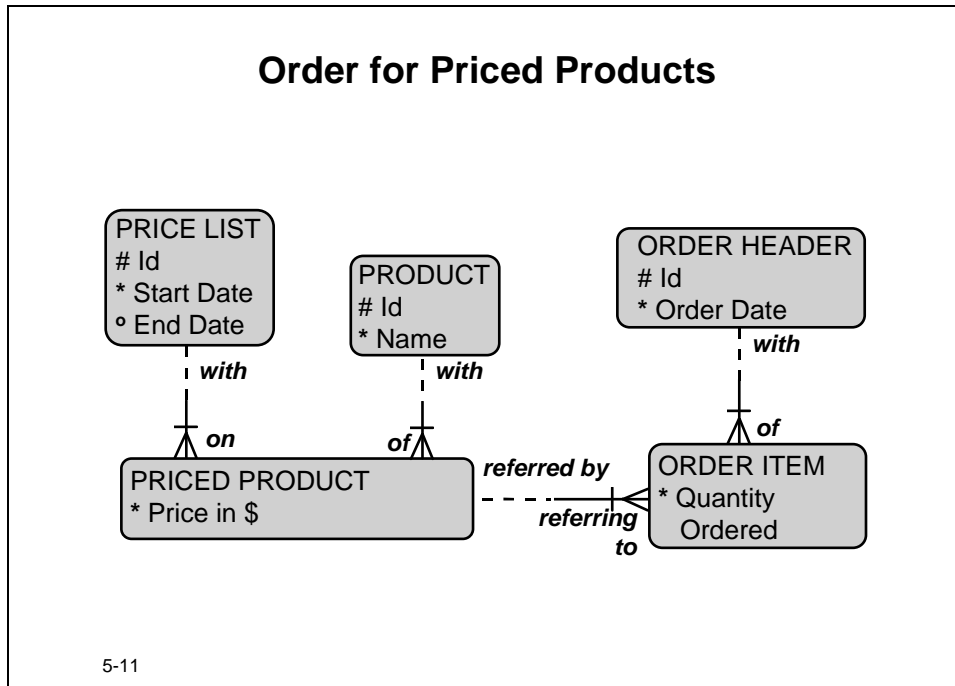
A variant on the above model is often used when prices as a group are usually changed at the same time. The period that prices are valid is the same for many prices; that would lead to this model:



Entity **PRICE LIST** represents the set of prices for the various products; **PRICED PRODUCT** represents the price list items. To know the price paid for an ordered item, you take the Order Date of the **ORDER HEADER**, and take the **PRICE LIST** that is applicable at that date. Next, you go from **ORDER ITEM** to the **PRODUCT** that is referred to and from there to the **PRICED PRODUCT** of the **PRICE LIST** you have just found. To find the order total for an order, it would need a join over five tables.

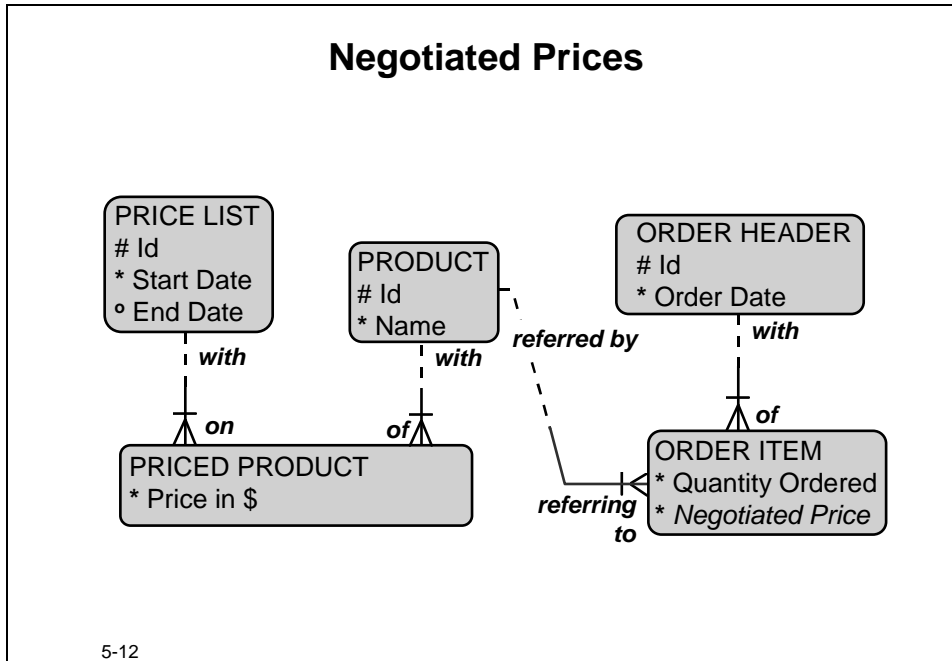
Buying a PRODUCT or a PRICED PRODUCT?

Another variant of a pricing model is shown here.



Here an ORDER ITEM refers directly to a PRICED PRODUCT. At create time of the ORDER ITEM the constraint is applied that the Order Date must match the correct PRICE LIST period. To find the order total for an order now only requires three tables.

Negotiated Prices



When prices are subject to negotiation, the model becomes simpler. Negotiated Price is now an attribute of entity ORDER ITEM; ORDER ITEM refers to PRODUCT. Every referential constraint can be modeled.

This model may seem to hold derivable information, but this is not true. Even in the case that almost all Negotiated Prices are equal to the current product price, you have to model Negotiated Price at ORDER ITEM level, just because of the small chance of an exception. To find the order total you require only two tables. You can imagine that many analysts choose this variant of the model as a safeguard, even if there is nothing to negotiate at present.

Which Variant to Use and When?

Typically, the model with the negotiated prices will occur where the number of ORDER ITEMS per ORDER HEADER is low, often just a single one, and where the value is high, as, for example, in the context of a used car business.

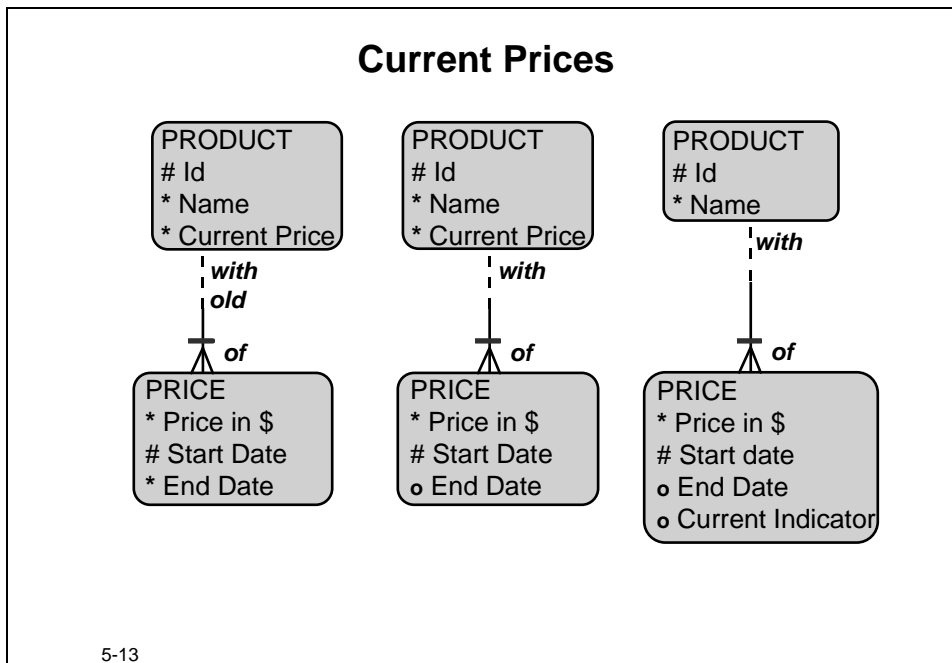
You see ORDER ITEM referring to a PRODUCT most often in the situation where prices do not change frequently. The number of items per ORDER HEADER is often well over one, and the overall value limited. Typical examples are the fashion industry and grocery stores.

The model with ORDER ITEM referring to PRICED PRODUCT is often used in businesses where prices often change, as in the fresh fruit and vegetable markets. Prices there may even change during the day.

The model with attribute Current Price for a PRODUCT is typically the model for the supermarket environment where instant availability of prices at the checkouts is vital.

As stated earlier, the best model for a particular context depends on functional needs. See more on this in the chapters on Denormalized Data and Design Considerations.

Current Price



These models are variants on the **PRODUCT-PRICE** model you have seen before.

In the left-hand model the 1:m relationship between **PRODUCT** and **PRICE** shows the real historical prices only. You can guess that only historical prices are kept because attribute **End Date** is mandatory; an additional constraint is that this value should always be in the past. The **Current Price** of a **PRODUCT** is represented as an attribute. This model does not have any redundancies.

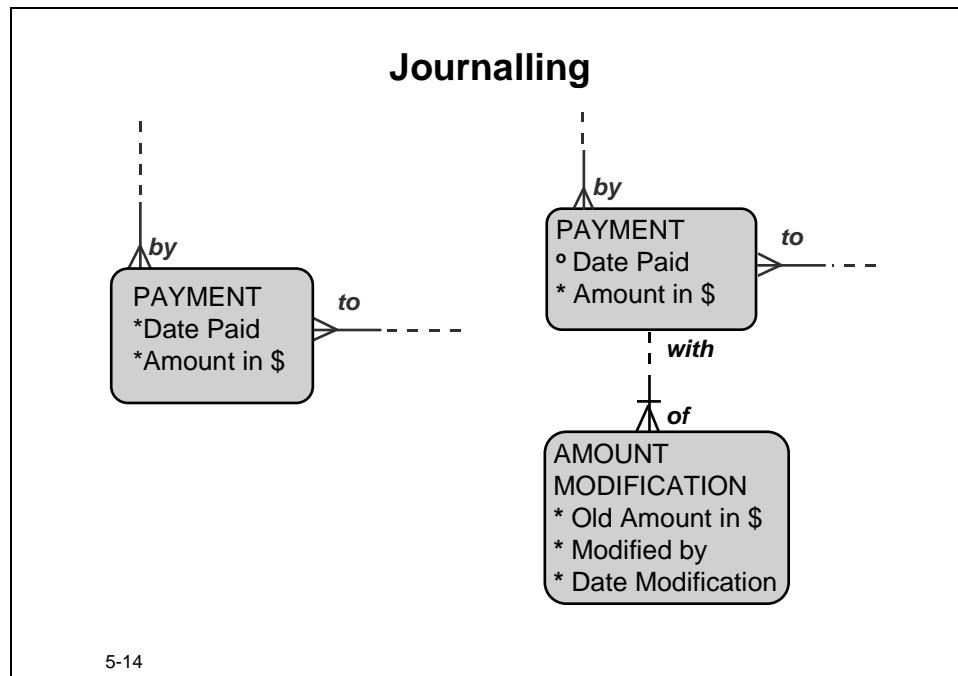
In many situations it would be a good *design* decision to keep the current product prices as well as the old prices in one table based on entity **PRICE**. The middle model is an ER representation of that situation. Note that **End Date** is now optional.

The right-hand model is another model that contains a subtle redundancy. See more on this type of redundancy in the lesson on **Denormalized Data**.

Journalling

When a system allows a user to modify or remove particular information, the question should arise if the old values must be kept on record. This is called **logging** or **journalling**. You will often encounter this when the information is of a financial nature.

Consequences for the Model



A journal usually consists of both the modified value and the information about who did the modification and when it was done. This extra information can, of course, be expanded if you wish.

Apart from the consequences for the conceptual data model, the system needs special journalling functionality: any business function that allows an update of Amount In should result in the requested update, plus the creation of an entity instance AMOUNT MODIFICATION with the proper values. Of course, the system would need special functions as well in order to do something with the logged data.

No Journal Entity

When several, or all, attributes of an entity need to be journalled, it is often implemented by maintaining a full shadow table that has the same columns as the original plus some extra to store information about the who, when, and what of the change. This table does not result from a separate entity; it is just a second, special, implementation of one and the same entity.

Journalling Registers Only

Note that logging does not prevent a user from making updates. Preventing updates entirely is a functional issue and is invisible in the conceptual data model. Be aware that preventing updates altogether would also block the possibility to change typos or other mistakes.

At this stage, decisions must be made about the behavior of the system with respect to updates; sometimes this leads to modifications in the conceptual data model.

For example, suppose that in a particular business context a certain group of users is allowed to create instances of PAYMENT but is not allowed to change them. Changes can only be made by, say, a financial manager. Suppose you just created a PAYMENT instance and you discover you made a mistake. For those cases the business would need some mechanism to stop the erroneous instance. One mechanism would be to ask one of the financial managers to make the change. A far better mechanism would be to add functionality so that a payment can be neutralized. This may be represented in the model as an attribute Neutralized Indicator that users can set to Yes.

Summary

Every update in a system means loss of information. To avoid that you can create your model to keep a history of the old situations. Sometimes relationships refer to a time-dependent state of an entity. In other words, the updated entity is in fact a new instance of the entity and not an updated existing instance. If this is the case, the time-dependent referential constraints cannot be modeled by a relationship only.

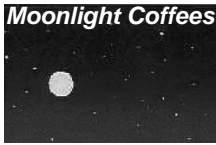
Time in your model is a complicated issue. Many models have some time-related entities.

Summary

- **Consider the need for keeping old values.**
- **Time in your model is complicated:**
 - **Implicit versions**
 - **References**
- **Journalling**

5-15

Practice 5—1: Shift



Goal

The purpose of this practice is to model various aspects of time.

Scenario

Some shops are open 24 hours a day, seven days a week. Others close at night. Employees work in shifts. Shifts are subject to local legislation. Below you see the shifts that are defined in one of the shops in Amsterdam.

Your Assignment

List the various date/time elements you find in this Shift scheme and make a conceptual data model.

Practice: Shift

Museumplein, Amsterdam, March 21

Shift	1	2	3	4	5
Mon	6:30 11:30	11:30 16:00	16:00 20:30	20:30 23:00	–
Tue	7:00 11:30	11:30 16:00	16:00 20:30	20:30 23:00	–
Wed	7:00 11:30	11:30 16:00	16:00 20:30	20:30 23:00	–
Thu	7:00 11:30	11:30 16:00	16:00 20:30	20:30 23:00	–
Fri	7:00 11:30	11:30 16:00	16:00 20:30	20:30 24:00	–
Sat/Sun	8:00 11:30	11:30 15:00	15:00 18:00	18:00 21:00	21:00 24:00

5-17

Practice 5—2: Strawberry Wafer

Moonlight Coffees



Scenario

You have modeled a price list in an earlier lesson. Now some new information is available.

Your Assignment

Revisit your model and make changes, if necessary, given this extra information.

Prices are at the same level within a country; prices are determined by the Global Pricing Department. Usually the prices for regular, global products are re-established once a year.

Prices and availability for local specialties are determined by the individual shops. For example, the famous Norwegian *Vafler med Jordbær* (a delicious wafer with fresh strawberries) is only available in summer. Its price depends on the current local market price of fresh strawberries.

prijslijst

de Keyzer, Keyzerlei 15, Antwerpen
bezoekt ons op 't Web: www.moonlight.com

	klein	middel	groot	
gewone koffie	60	90	120	
cappuccino	90	110	140	
koffie verkeerd	75	100	130	
speciale koffies	99	125	150	
espresso	60	95	110	
koffie van de dag	45	75	100	
<i>caffeine vrij</i>	5	10	15	<i>toeslag</i>
zwarte thees	60	100	120	
vruchten thees	75	110	130	
kruiden thees	80	120	140	
dag thee	50	85	100	
<i>caffeine vrij</i>	5	10	15	<i>toeslag</i>
frisdranken	60	100	130	
diverse sodas	60	100	130	
mineraal water	75	120	140	
appel taart				180
brusselse wafel				150
portie chocolade bonbons				150
koekje van eigen deeg				120
portie slagroom				30

inclusief BTW
16 September

5-19

Practice 5—3: Bundles



Goal

The purpose of this practice is to expand the concept of an old entity.

Scenario

As a test, Moonlight sells bundled products in some shops, for a special price. Here are some examples.

A *SweetTreat^(tm)* consists of a large soft drink plus cake of the day.

A *BigBox^(tm)* consist of a large coffee of the day plus two cakes of the day.

A *SuperSweetTreat^(tm)* consists of a *SweetTreat^(tm)* plus whipped cream (on the cake).

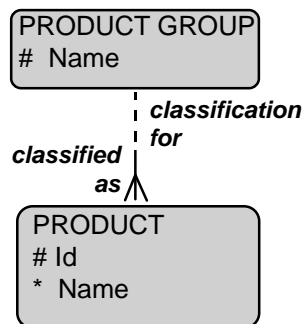
A *FamilyFeast^(tm)* consists of two *BigBoxes^(tm)* plus two *SweetTreatsTM* plus a small surprise.

Bundles sell very well; all kinds of new bundles are expected to come.

The system should know how all these products are composed, in order to complete various calculations.

Your Assignment

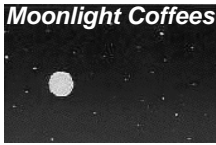
- 1 Modify the product part of the model in such a way that the desired calculations can be completed.



2 Change the model in such a way that it allows for:

**A DecafPunch^(tm) consists of a regular decaffeinated coffee
or a regular decaffeinated tea, plus a blackberry muffin.**

Practice 5—4: Product Structure



Goal

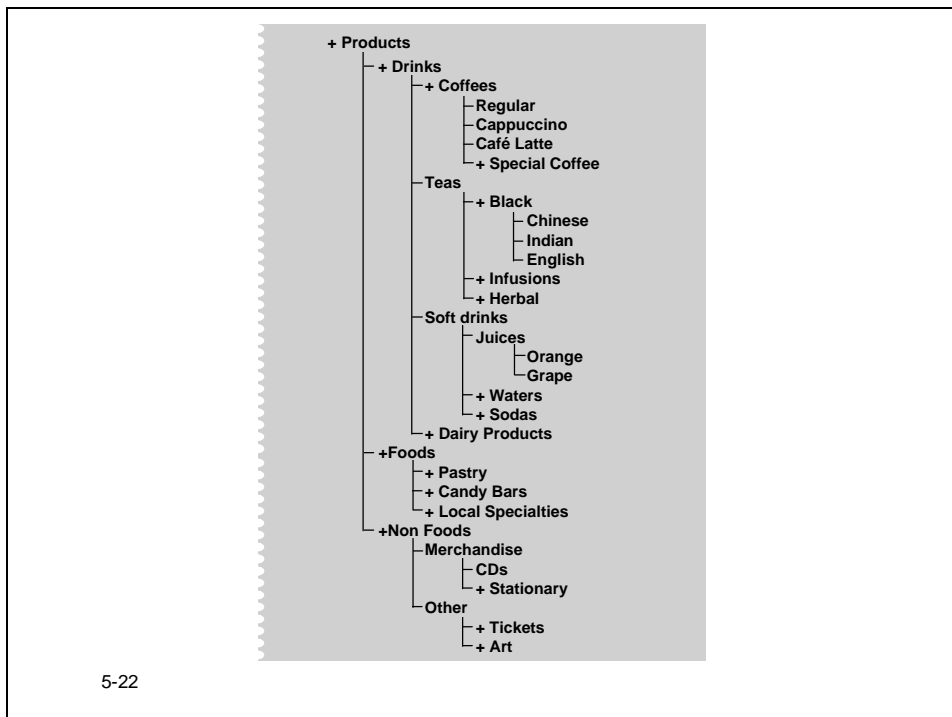
The purpose of this practice is to model a hierarchical structure.

Scenario

Moonlight needs to make sales information available as a tool to optimize its business. A hierarchical product structure is being developed to be able to report on different summary levels. This hierarchical structure should replace the single level product group classification. Below you see the current idea about a product structure. This structure is far from complete, but it should give you an idea of the shape the structure will take. The + signs mean that the structure will be expanded at that point.

Your Assignment

- 1 Create a model for a product classification structure.



- 2 (Optional) How would you treat the bundled products?

6

Advanced Modeling Topics

Introduction

Lesson Aim

This lesson gives an overview of patterns you can discover in data models. This lesson introduces some generic models. You can use these to make your model withstand future changes that are predictable but not yet known.

Objectives

Overview

- Patterns
- Drawing conventions
- Generic modeling

6-2

Topic	See Page
Introduction	2
Patterns	4
Master Detail	5
Basket	6
Classification	7
Hierarchy	8
Chain	10
Network	11
Symmetric Relationships	13
Roles	14
Fan Trap	15

Topic	See Page
Data Warehouse	16
Drawing Conventions	17
Generic Modeling	19
Generic Models	20
Summary	23
Practice 6—1: Patterns	24
Practice 6—2: Data Warehouse	25
Practice 6—3: Argos and Erats	26
Practice 6—4: Synonym	27

At the end of this lesson, you should be able to do the following:

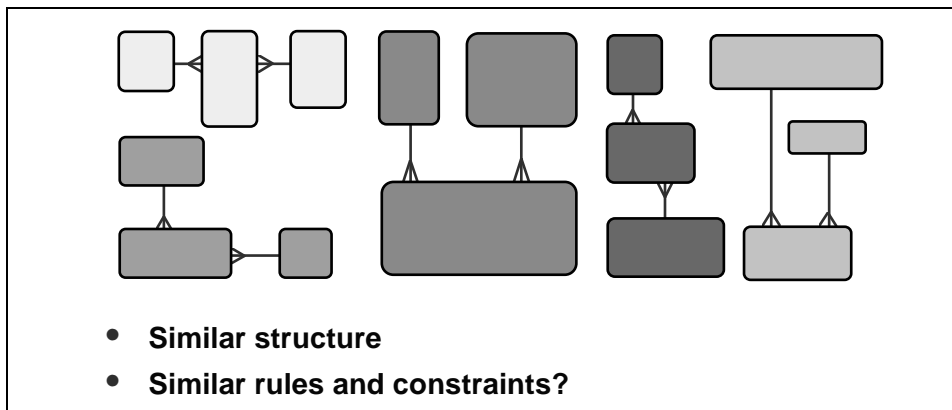
- Recognize common patterns in conceptual data models
- Know the general behavior, such as the common constraints, of these patterns
- Use particular drawing conventions
- Create a more generic model for selected sections of a conceptual data model

Patterns

Similar Structure

Many models contain parts that have a similar structure, although the context may be completely different. For example, the structure of a conceptual data model in the context of a dictionary that deals with concepts such as headword, entry, meaning, synonym is, surprisingly, almost identical to the structure of a railroad with track, station, connection, and also to the structure of a baseball or soccer competition.

Easier to see are the similarities between, for example, ORDER HEADER with ORDER ITEM and QUOTATION HEADER with QUOTATION ITEM, or between MARRIAGE and JOB ASSIGNMENT.



Why Search for Similarities?

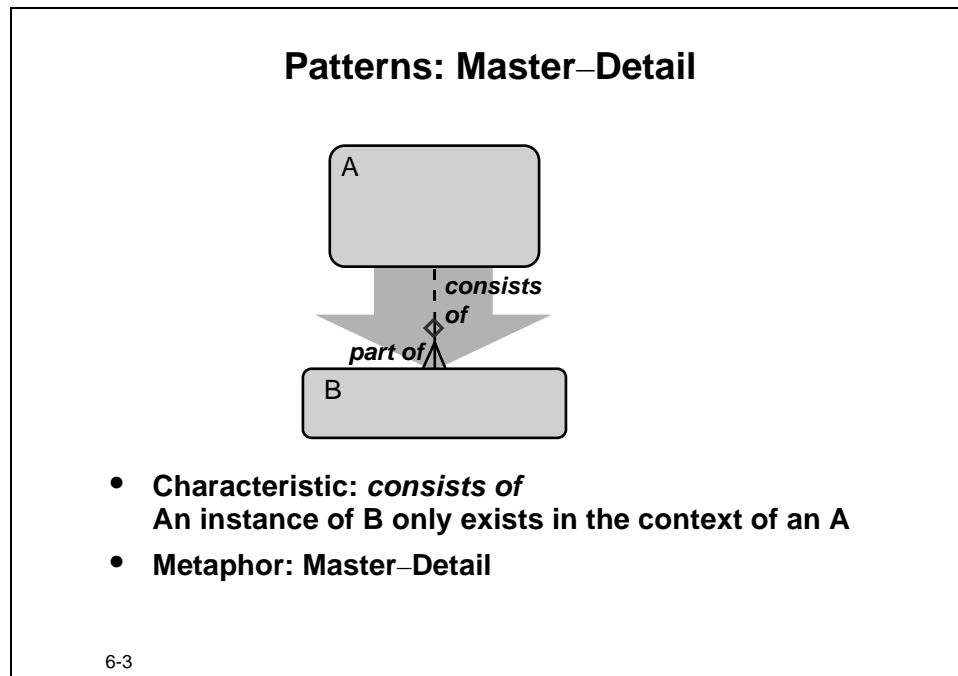
The main reason why it is important to look for similarities is that it will save you time. If you have solved a problem in a particular context and you can apply the solution to another, it obviously saves time. Moreover, you will feel confident that you know about the situation. It will help you to ask the right questions. It will help you identify the really complex and unpleasant things and will prevent you from making the same mistakes twice.

Are there similarities between marriage and job assignment? Of course, the business rules in the context of a marriage are different, because they are determined differently compared to those of a job assignment. But when you are aware of the similarities, you can easily check if business rules of the first context apply in the second, by asking, for example:

- Can an assignment be for more than one job?
- Can someone have two assignments simultaneously? Unofficially?
- How does an assignment start? How does it end?

The following paragraphs discuss a series of patterns that you will encounter while creating your models. For all these patterns you will see the characteristics and the rules that usually apply.

Master Detail



Master-detail constructions are very common, as 1:m relationships are very common. Distinguish between a 1:m relationship that is typically directed from the 1 to the many and a relationship that is directed the other way around (see below). Master-detail is characterized by the fact that the master **A** is divided into **B**'s. **B**'s do not exist alone; they are always in the context of an **A**.

It is very rare that these relationships are transferable; if an instance of **B** is connected to the wrong instance of **A**, it is far more likely that the instance of **B** is deleted and then recreated in the context of the correct **A**.

Typical master-detail relationship names:

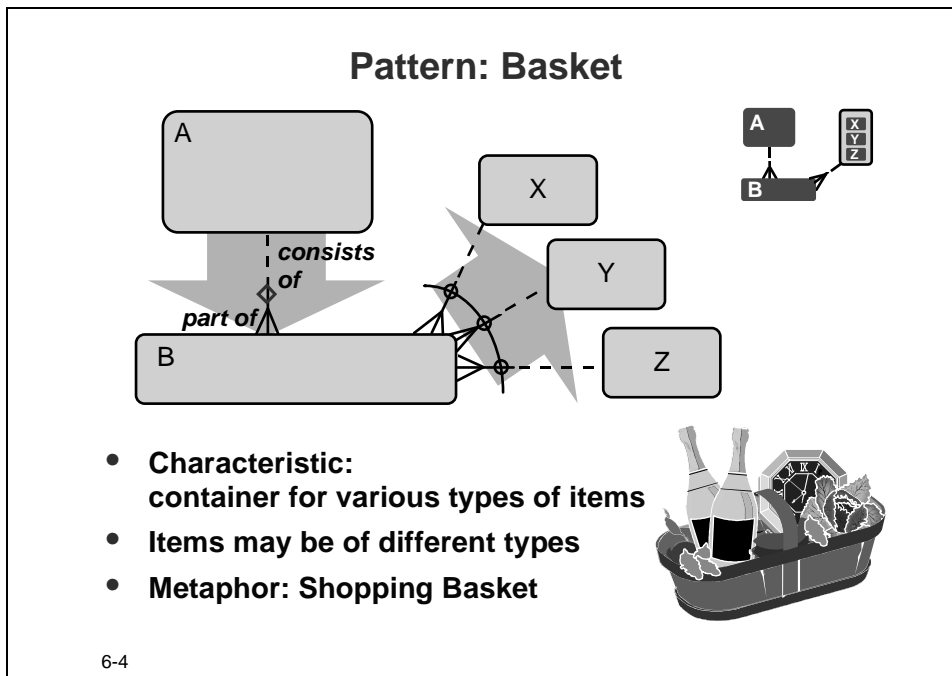
- Consists of
- Divided into
- Made of
- (Exists) With

Often a master **A** is of no value when it has no **B**'s, for example, the relationship is mandatory at the 1 side. This mandatory relationship end can usually be circumvented, as you have seen before.

Implementation

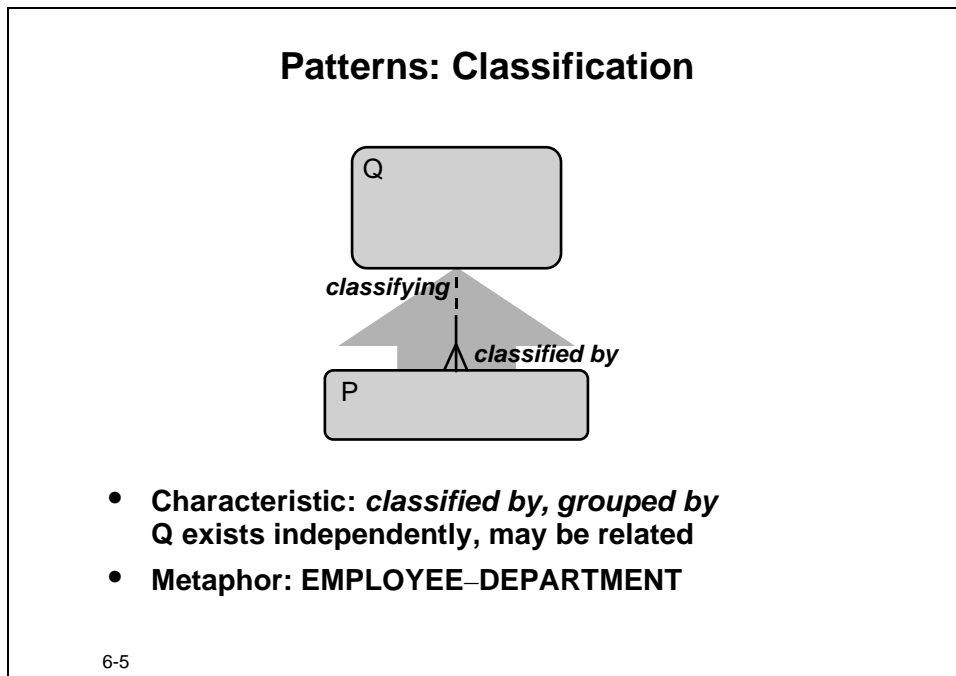
The tables that come from this master-detail pattern should be considered as clustered.

Basket



A ***basket*** construction is a special case of a master-detail pattern. A basket can contain one or more things, but these things (often named: items) can be of different types. A single item is always of one type only. That is the reason for the arc. The arc shows that an item must be of one and only one of the types.

Classification



This is again a 1:m relationship, but now the main orientation is from P to Q.

This is typically the case when Q can exist independently from P. Q acts as a class for P, something with which to group P's.

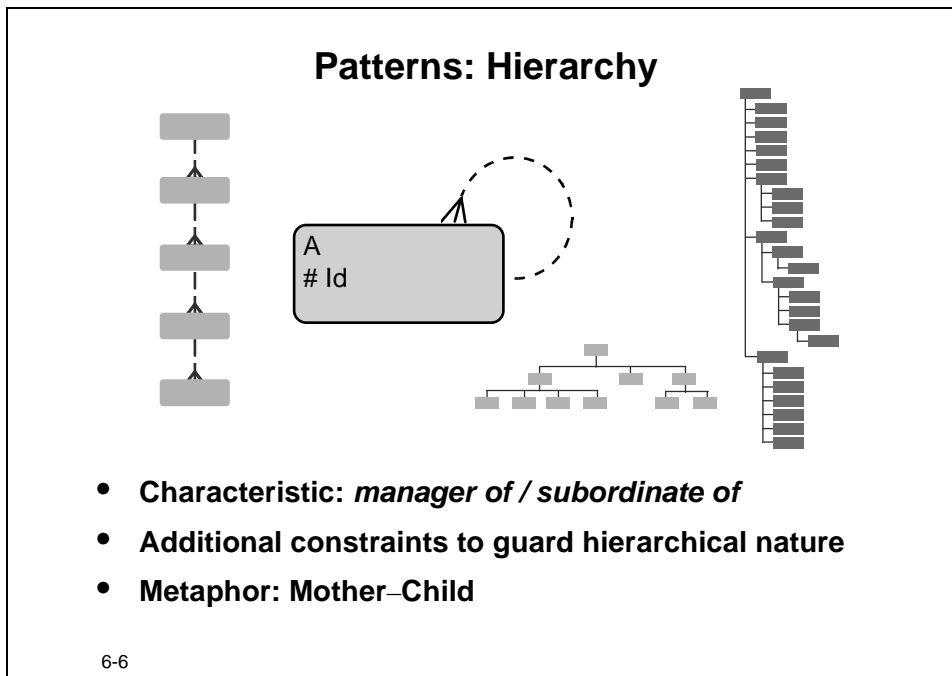
Usually entities in a conceptual data model have several of these classes.

Typical classification-type relationship names:

- Classified by
- Grouped by
- Assigned to
- (Exists) In

The relationship is usually transferable as classifications may change over time.

Hierarchy



Most hierarchical structures have a known limit for the maximum number of levels. If that is the case and the limit is a low number of 5, for example, then usually the best model is the one that is shown in the left of the illustration, one entity per level.

Model the structure with the recursive relationship if:

- The structure has no known level limit.
- The structure has a level limit, but the limit is high, say six or more.
- An instance of the structure can easily have a change of position, thus changing its level.
- You like maintaining constraints.

Disputable or False Hierarchies

Often structures should be hierarchical but you cannot be sure. Sometimes they seem hierarchical but actually are not so. You can have, for example, the *is owner of* relationship between companies. Suppose company C_1 owns company C_2 , company C_2 owns company C_3 , could it be that company C_3 owns the shares of company C_1 ? Even if legislation would prohibit such strange constructions, would you be sure?

Many people see the parent/child relationship as a metaphor for a hierarchical relationship. Clearly this is wrong as a child usually has two parents and can have step-parents as well.

Also the hierarchical structure of a FILE SYSTEM with files and folders, which are files of a particular type, is a disputable hierarchy when you think of the concept of a shortcut in Windows (or a Link in UNIX). These shortcuts transform the hierarchy conceptually into a network although technically a shortcut and a link are just files with a special role.

Recursive Relationship and Optionality

Recursive relationships that describe a real hierarchy are usually optional at both ends, as the hierarchy must start or end somewhere.

Constraints Applying to a Hierarchy

The recursive model, as you see in the centre of the illustration, only requires an instance of A to refer to a valid instance of A. A_1 referring to A_1 is fine, according to the model. A_2 referring to A_3 and A_3 referring to A_2 is fine as well. These are the only obvious diversions from a real hierarchy.

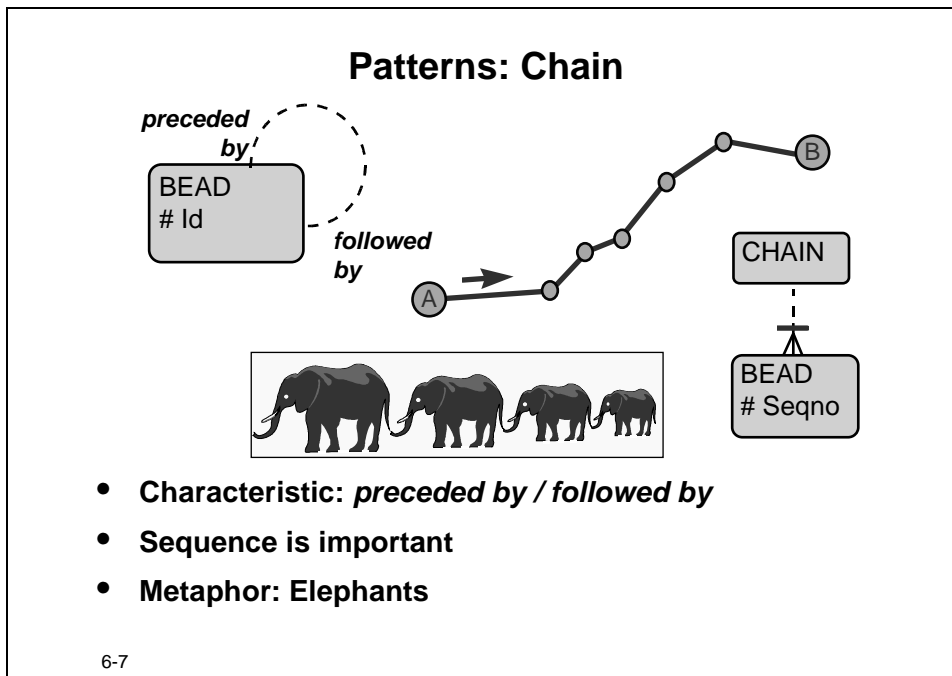
Constraints that apply in a hierarchical structure deal with safeguarding the hierarchy and should prevent the table from containing the above kind of data.

Implementation

The first constraint, A_1 may not refer to A_1 , and you can easily check this with an Oracle check constraint. The others need some programming and lead to database triggers.

Possibly you may have to check extra business rules, for example, when the number of levels may not exceed a given value.

Chain

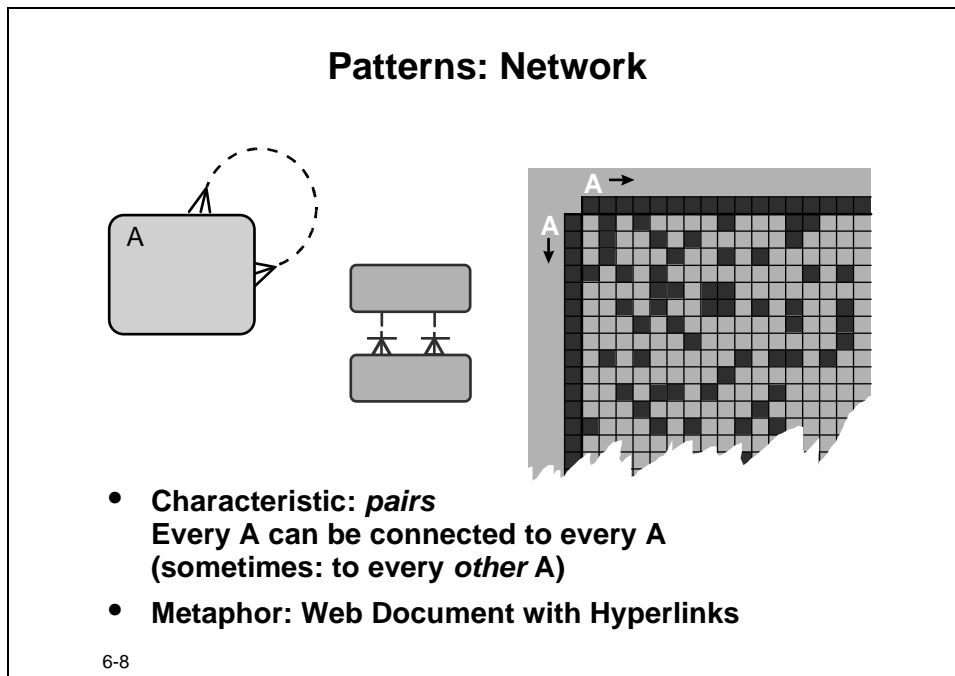


A **Chain (of beads)** can be regarded as a special kind of hierarchy. A chain is a recursive relationship of an entity. The relationship of the chain is a 1:1 relationship as a chain is characterized by the fact that an object in the chain is preceded and followed by one object at most.

A chain is a structure where *sequence* is of importance, for example, the sequence of the pages in a chapter and of the chapters in a document, of the critical path in a procedure, of the *preferred* road from A to B.

A chain can also be modeled as a master-detail. The recursive model allows an easy insertion in the chain. The right-hand model with entity CHAIN and BEAD may need to change the sequence numbers of all the beads behind the inserted one.

Network

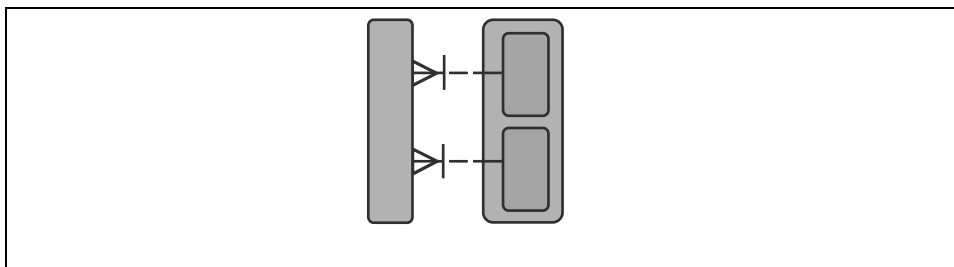


Network structures typically describe pairs of things of the same type, for example, marriage, railroad track (pair of start and end stations), synonyms (two words with the same meaning), and Web documents with hyperlinks to other Web documents.

Characteristics

Often:

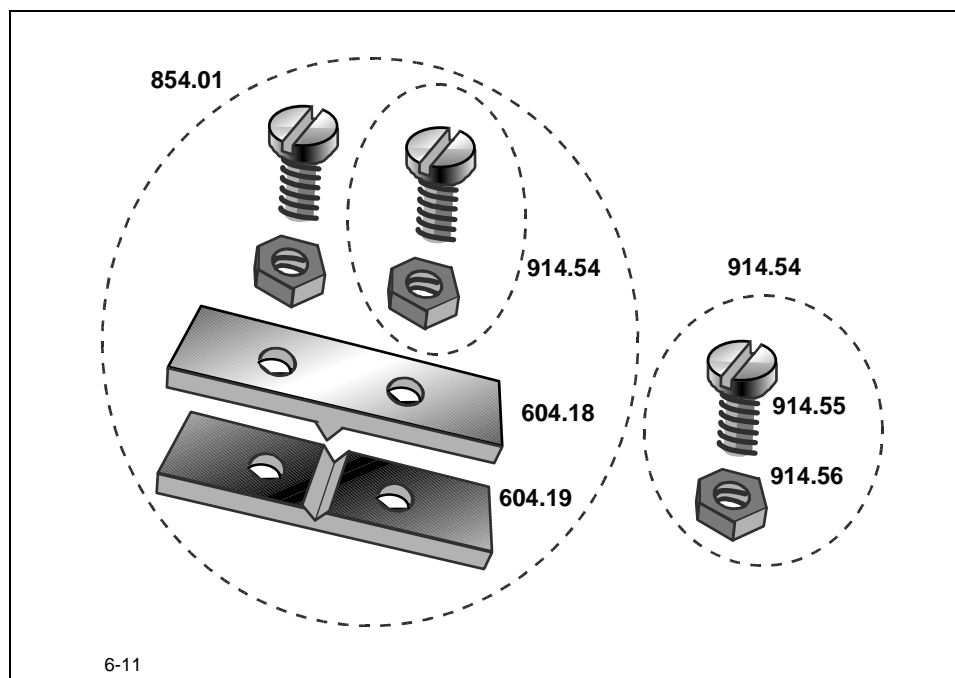
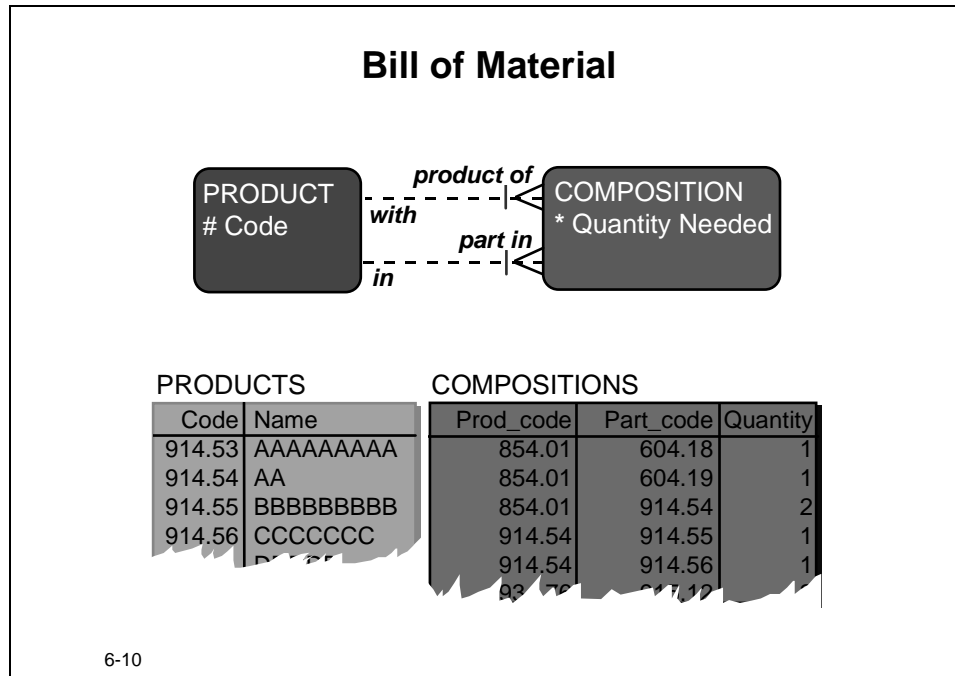
- The m:m relationship must be resolved to hold specific information about the pair such as the date of the marriage, or the length of the railroad track.
- The two relationships of the intersection entity form the unique identifier.
- Time-related constraints apply in networks that must guard, for example, the kind of rules that deal with “sequentially monogamous”.
- The two relationships refer to different subtypes of the entity:



Note that a hierarchy is a network where a particular set of business rules apply.

Bill of Material

A special example of a network structure is a Bill of Material (BOM). A BOM describes the way things are composed of other things, and how many of these other things (here it is instances of PRODUCT) are needed. Entity COMPOSITION is the intersection entity with attribute Quantity Needed.



Symmetric Relationships

Symmetric recursive relationships cause a very special kind of problem which is more complex than you would assume.

In most contexts a record of a pair (A_1, A_2) has a different meaning when referred to as (A_2, A_1). For example, if the model is about entity PERSON and the relationship is *mother of / daughter of*, then the existence of person pair (P_1, P_2) would mean the exclusion of the possibility of pair (P_2, P_1).

The recursive relationship of PERSON and *family of / family of*. Here, if (P_1, P_2) is true, then (P_2, P_1) is equally true. This is called a symmetric relationship. There are other symmetric recursive relationships such as: STATION *directly connected by rail with* STATION,

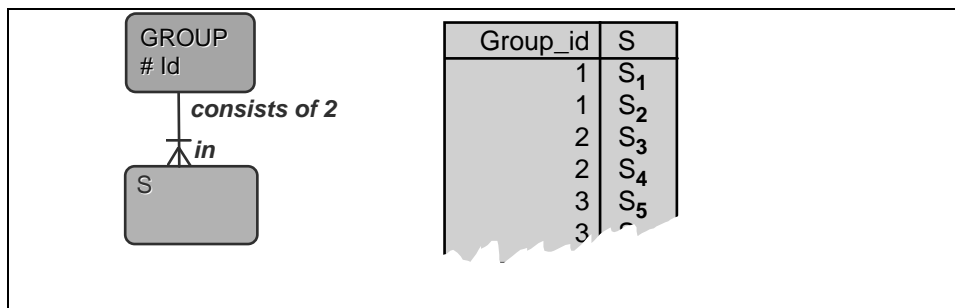
Symmetric Relationships: Problem

When in a symmetric relationship the pair (S_1, S_2) is valid, the pair (S_2, S_1) must be valid as well. Nevertheless, it would not make much sense to record both pairs as that would essentially store the same information twice—which would oppose one of the basic principles of database design.

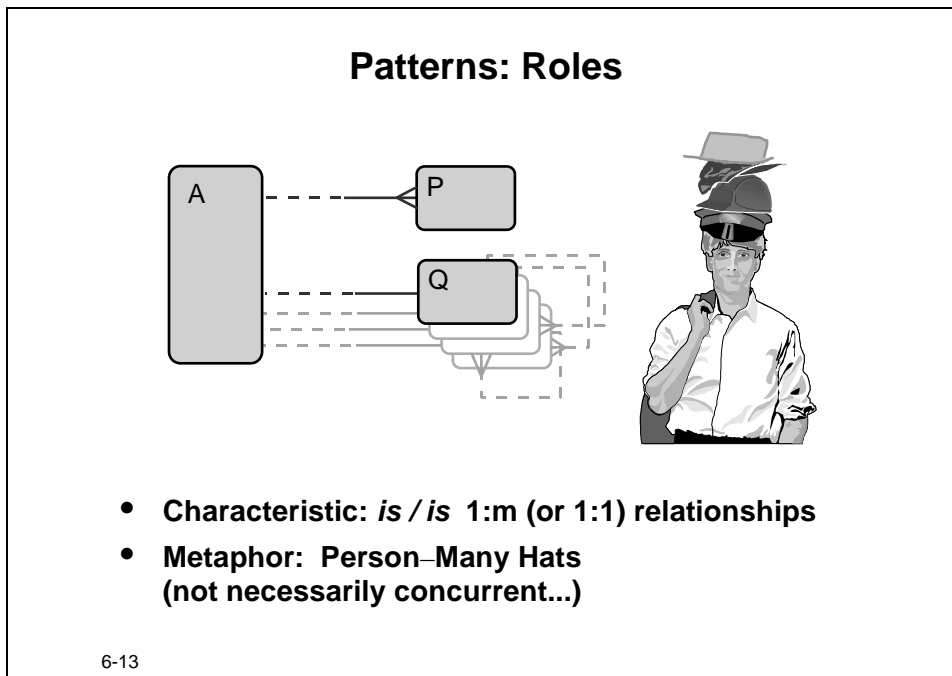
But if we record only one pair, which should we record? And how would you know which of the two pairs was used if someone else had recorded it?

Symmetric Relationships: Solution

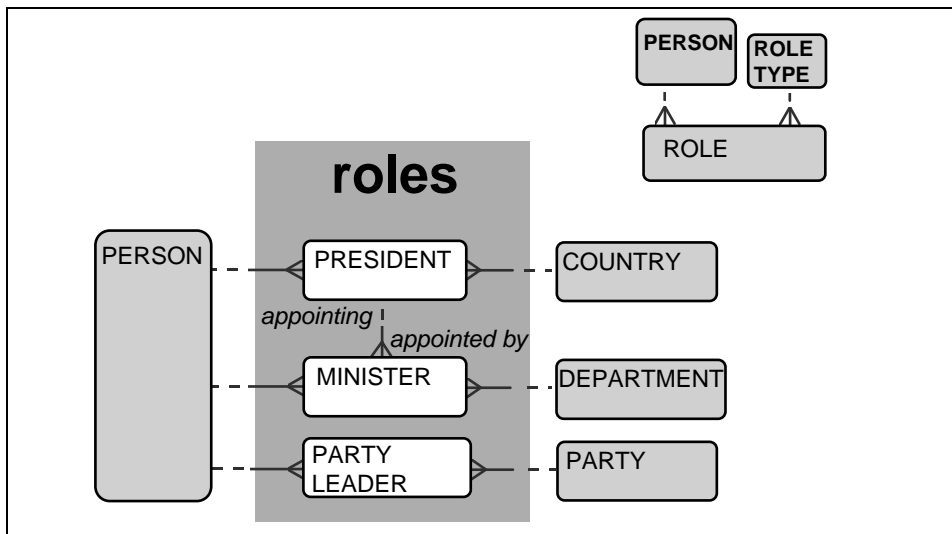
A way which is often used to model these symmetric situations is based on the following idea: think of (S_1, S_2) as Group₁, (S_3, S_4) as Group₂ and so on. Looking at the relationship this way, you can say that a GROUP always consists of exactly two instances of S. The model and the table implementation are shown below.



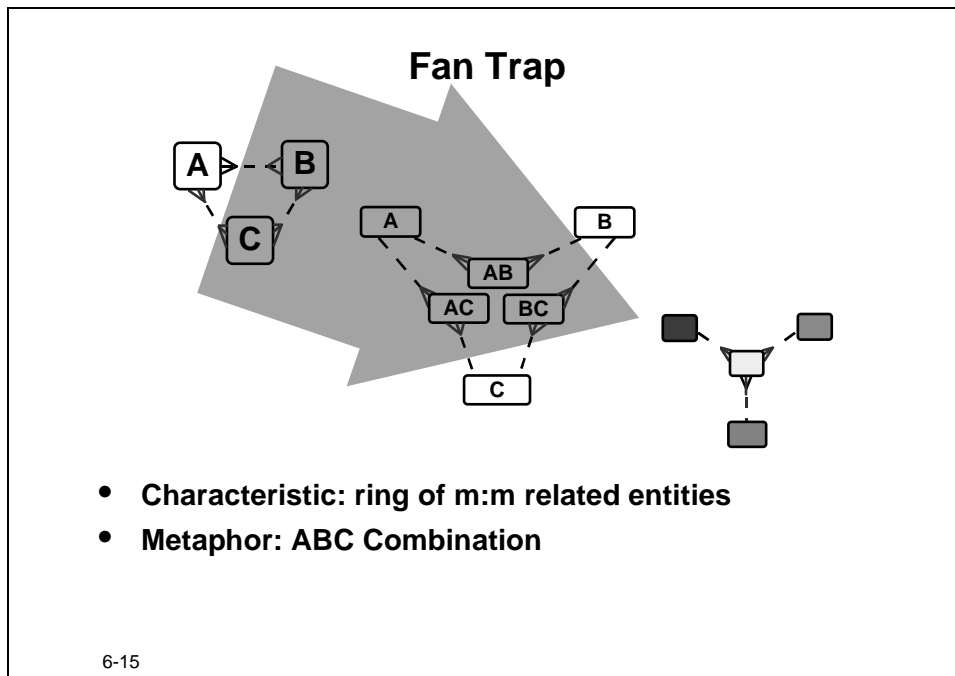
Roles



Roles often occur when a system needs to know more about people than the basic Name/Address/City information. Modeling the roles as separate entities offers the possibility to show which attributes are mandatory for a particular role, and, if necessary, to show relationships between the various roles. The example below shows that a person in their role as president of a country can appoint a person in the role of minister of a department. Possibly the words “presidency” and “ministership” are closer to the concepts than the ones in the diagram.



Fan Trap

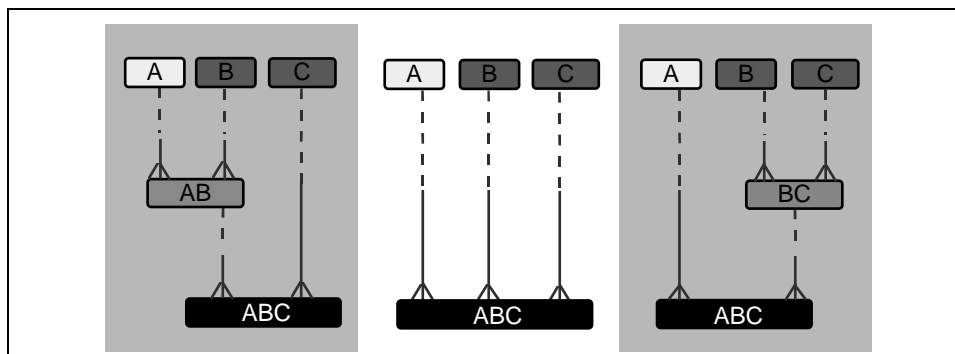


A **Fan Trap** (named after the characteristic shape of the solution) occurs when three or more entities are related through m:m relationships and form a ring. Usually you should replace the relationships with a central entity having several m:1 relationships. Preventing a fan trap is similar to resolving a m:m relationship between two entities.

Why Traps Occur

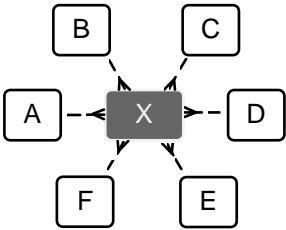
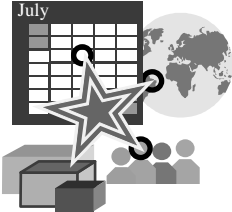
Resolving the three m:m relationships results into three intersection entities, AB, BC and AC. These will contain related pairs. Joining AB and BC may, however, result in different information to what AC contains which you may have seen in practice 3-8.

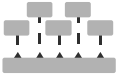
Note there are various ways of avoiding the trap, as is shown in the illustration. All can be correct, depending on the context.



Data Warehouse

Patterns: Data Warehouse

- **Characteristic:** *multidimensional*, many, many detail instances
- **Metaphor:** *star model*
Stars may be strangely shaped: 
- **Snowflake model**

6-17

A data warehouse system can be modeled as any system. Data warehouses contain the same sort of information as any straightforward transaction processing information system. Data warehouses usually contain less detailed, summarized, information as warehouses are mainly built for overview and statistical analysis. However, Data warehouses in general receive the input from online transaction systems that do contain details.

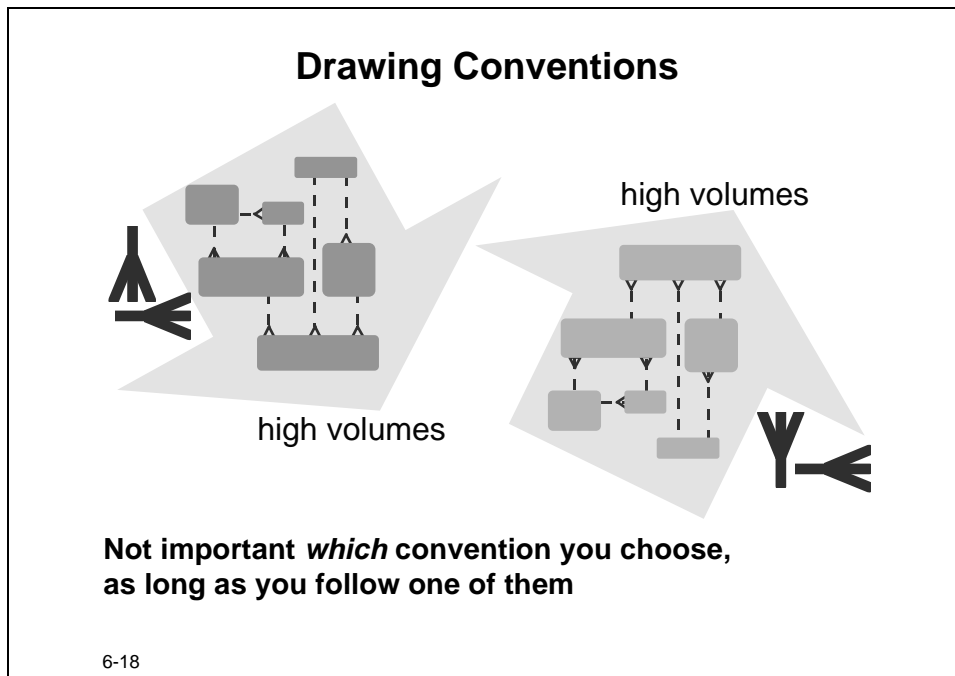
Data warehouses often have a star-shaped model: this is made up of one central entity (the facts) containing the condensed, summarized, information, and several dimensions that classify and group the details.

Common dimensions represent entities such as:

- Time
- Geography
- Actor (for example, salesperson, patient, customer, instructor)
- Product (for example, article, medical treatment, course)

Often the dimensions are classified as well. Time may be structured in day, week, month, quarter, year. You can classify products in various ways as you have seen in earlier examples. If this is the case, the model is usually described as the ***Snowflake*** model, as it looks like the crystal shape of a snowflake.

Drawing Conventions



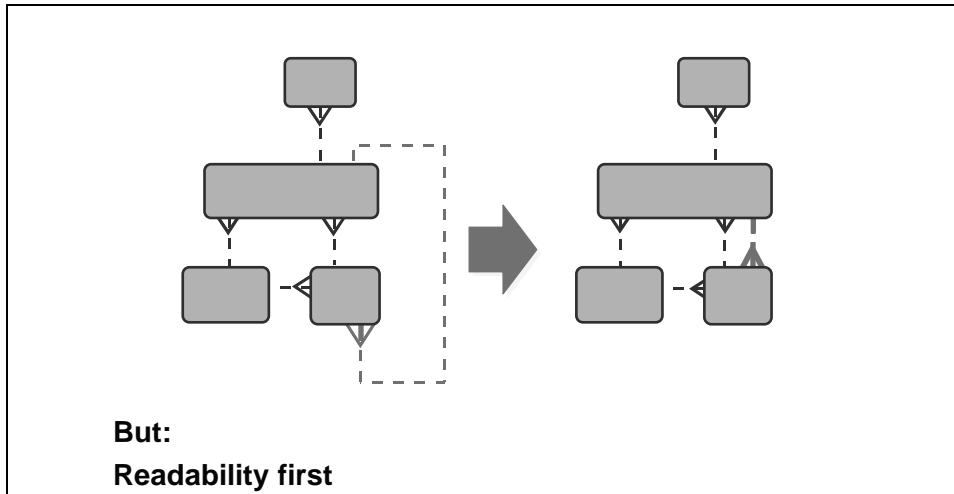
Two drawing conventions are widely in use: one that positions the entities with the high volumes at the top of the paper and one that does the opposite. Both try to avoid crossing relationship lines, partially overlapping entities, and relationship lines that cross entities. Whatever convention you choose, choose one and use it consistently. This will prevent errors and make the reading of large diagrams much easier.

Keep the overall structure of the layout unchanged during the modeling project as many people are disoriented when you change the structure.

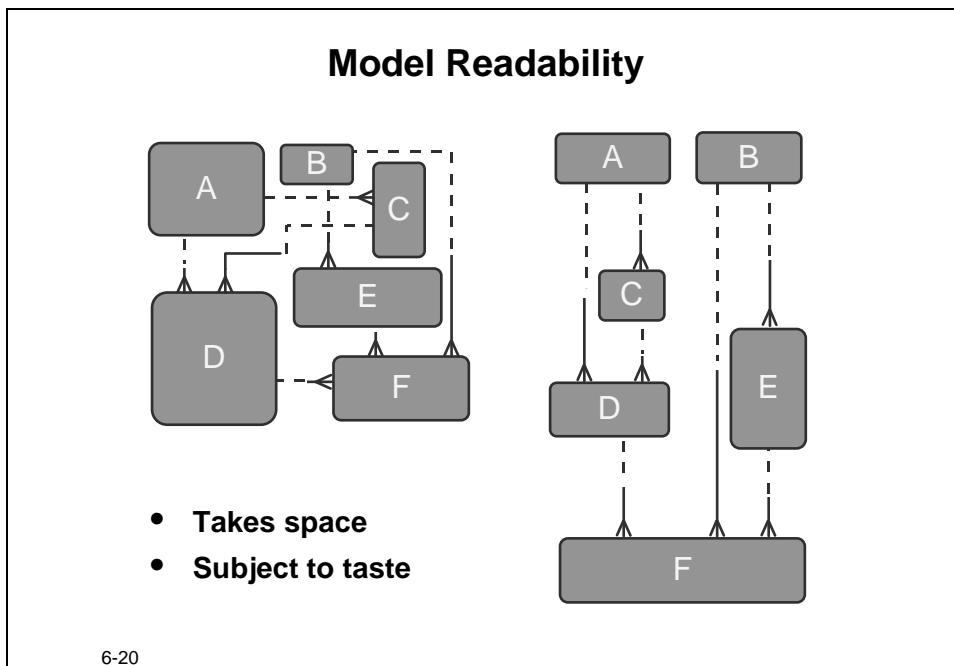
Make separate diagrams for every business area. These may have a different layout; these diagrams are mainly used for communication with subject matter experts.

At the end of this course, you should be able to read models created in any drawing convention, and you should be able to complete a model following any convention used.

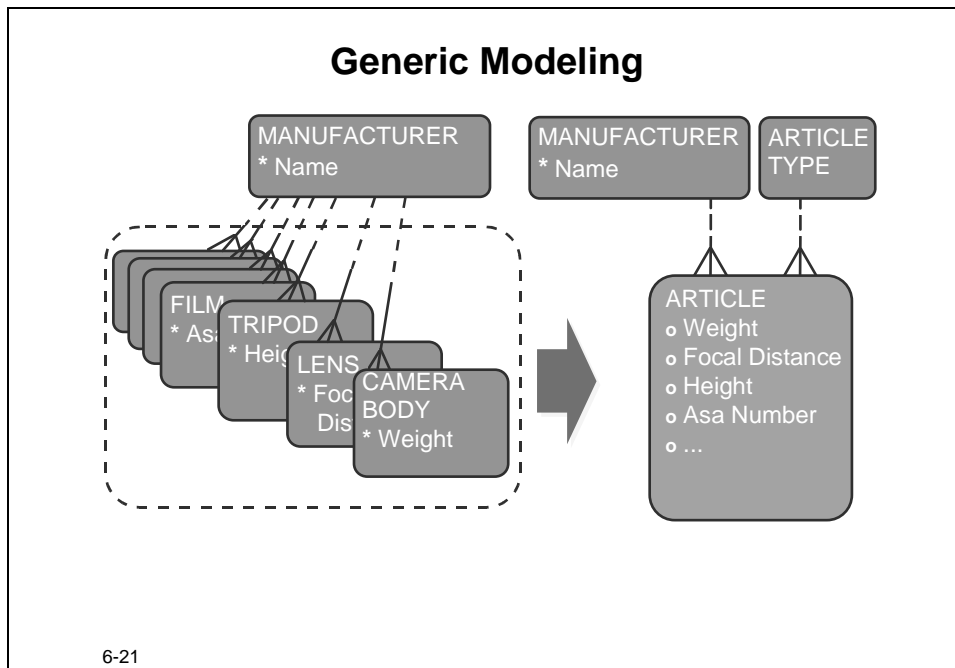
Use Conventions Sensibly



The major goal of creating the diagram (but not the model) is to give a representation of the model that can be used for communication purposes. This means that you must never let a convention interfere with readability and clarity. Do not be concerned that readability takes space. Usually an entity model is represented by several diagrams that show only the entities and relationships that deal with a particular functional part of the future system. Splitting the model over various diagrams adds to the readability.



Generic Modeling



What is Generic Modeling?

Generic modeling is looking at the same context from another, more distant perspective. From a distance many things look the same.

Suppose you are to make a model for a photographer's shop. The business typically sells many different articles, for example, camera bodies, compact cameras, lenses, films. For each type of article, there are between, say, 10 and 500 different types. You can model every type as an entity, for example, CAMERA BODY, LENS, FILM.

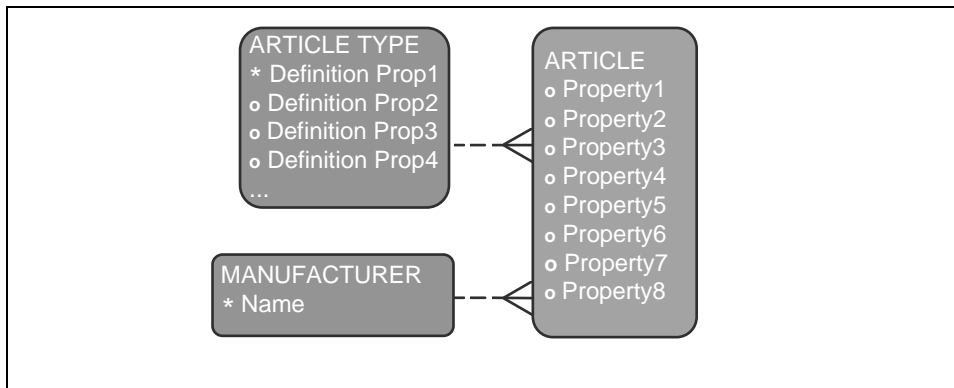
You could also model them all as subtypes of the entity ARTICLE, or all as just ARTICLE, without the subtypes.

This, however, would not work. For example, there is the fact that every now and then new kinds of articles are stocked in the shop. Every time this happens it leads to a new entity with its own attributes in the model.

The model with entity ARTICLE would only be a workable model if there were no (or possibly only very few) new instances of ARTICLE TYPE during the life cycle of the system.

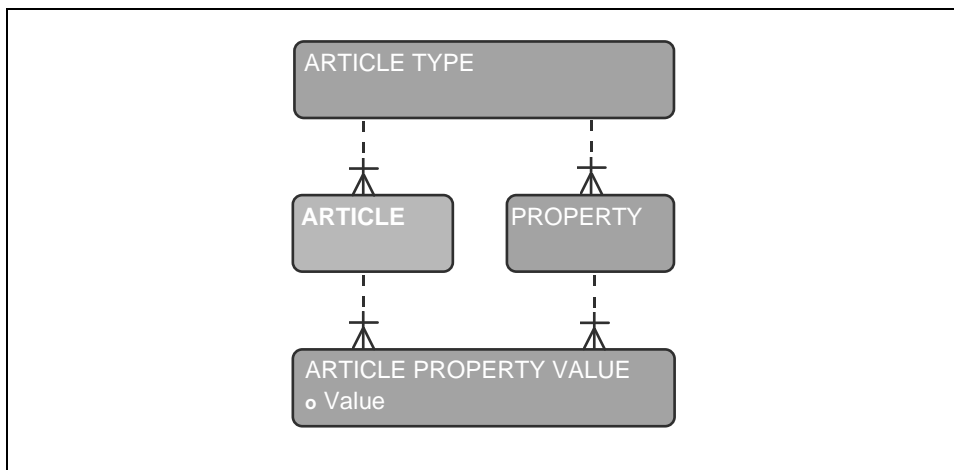
Generic Models

More generic models are shown below. They may be useful in particular situations.



Recycling of Attributes You can use this model if it is safe to assume the articles will have a limited number of attributes. This limit may be a high number but must be set beforehand. Property1 may contain the Asa Number for instances of ARTICLE of TYPE Film and may contain Weight for instances of ARTICLE of TYPE Camera Body and so on. The major advantage of this model is the possibility of adding new instances of ARTICLE TYPE without the need to change the model.

The type of information that should be entered for Property1, Property2, and so on can be described by using, for example, the Definition Prop1, attributes of ARTICLE TYPE. Here you can also store information about the data type of these properties.

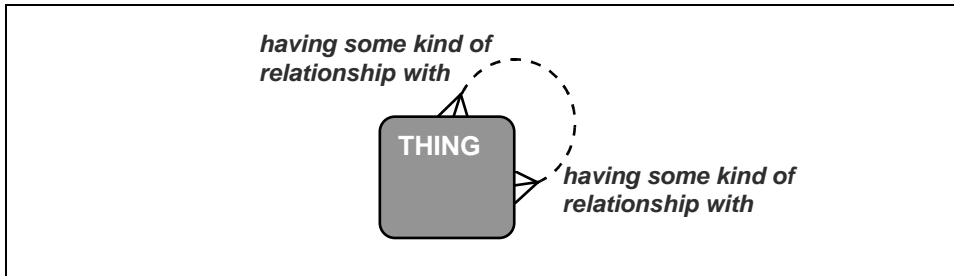


Attributes Modeled as PROPERTY Instance This model takes another approach. Every value for a PROPERTY of an ARTICLE is stored separately. This model gives a lot of freedom to define new articles and properties during the life cycle of the system.

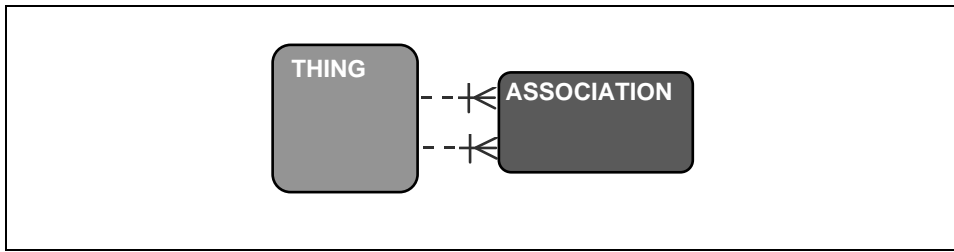
More Generic Models

Everything is a “Thing”

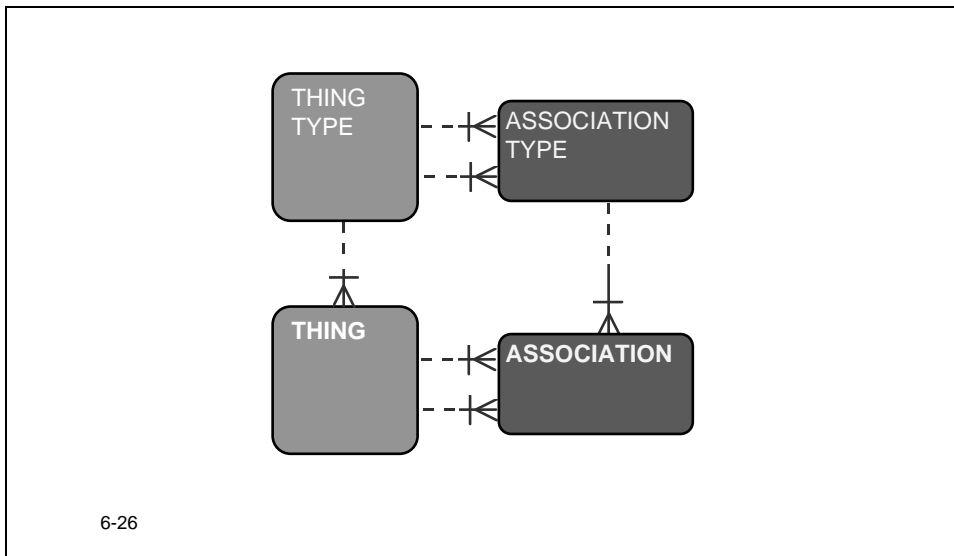
The world is full of things that may be related to things:



Resolving the m:m relationship:

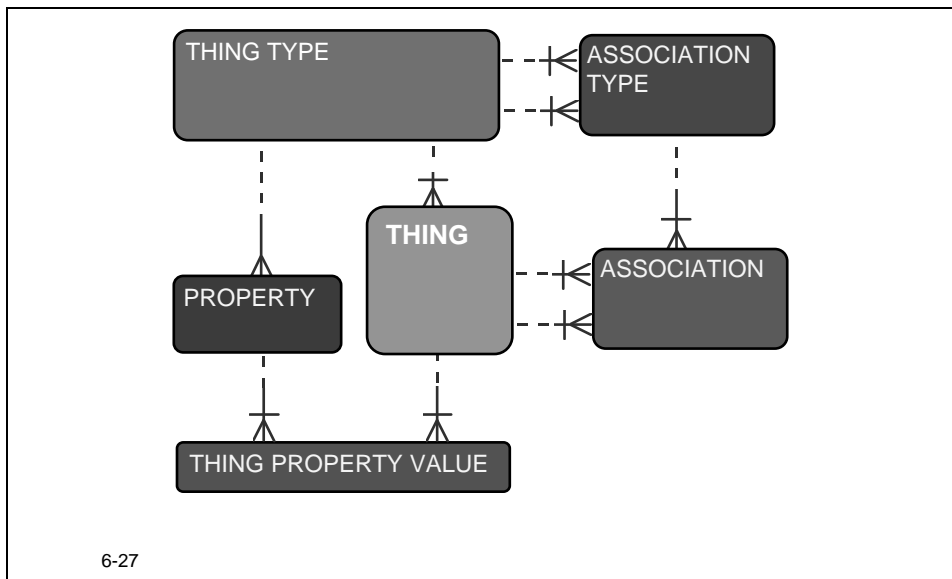


Now add some definition information:



This is a rather generic model. In fact, it is a model of the universe and beyond. Note that the number of attributes for entity **THING** may be substantial.

Most Generic Model



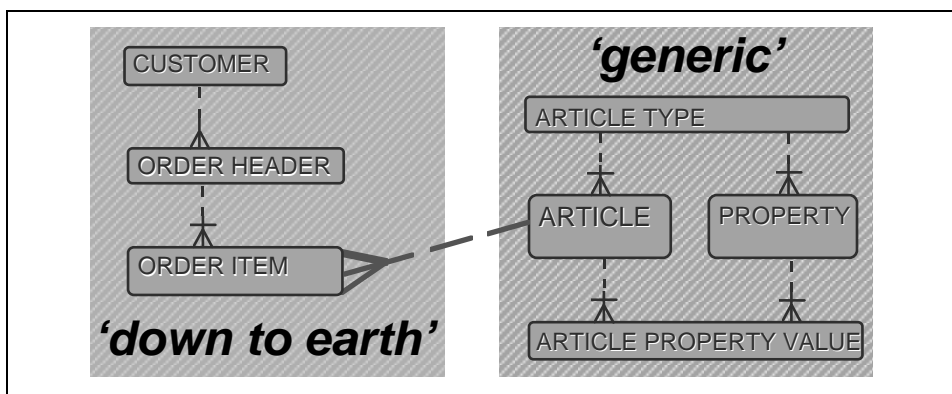
This model combines the concepts of “thing” and the *property/property value* and thus allows everything to be represented with a free number of properties per type.

Value of Generic Modeling

The use of generic modeling is mainly to reduce to a minimum the number of possible future changes of the conceptual data model. This can be an enormous advantage as it cuts maintenance costs during the lifetime of a system. The other side of the coin is that the initial coding of the programs is more complex as the entities are not “down-to-earth” things.

Best of Two Worlds

In many models you would use a mix of the easy-to-understand, straightforward entities and the more generic thing-like entities.



Summary

Summary

- **Patterns**
 - **Show similarities**
 - **Invent your wheel only once**
- **Generic models**
 - **Reduce the number of entities dramatically**
 - **Are more complex to implement**
 - **Are very flexible**
 - **Are usually the best choice in unstable situations**

6-29

Thinking in terms of patterns forms a valuable way of doing quality checks on a conceptual data model. Often constraints and considerations in one context can be transferred to the other context with a simple translation.

Using a drawing convention in your models helps to improve readability and clarity. This may prevent mistakes and inaccuracies.

Generic modeling can prevent the need to change data structures in the future and can reduce the number of tables and programs dramatically. The price is increased complexity in both data model and programs.

Practice 6—1: Patterns

Goal

The purpose of this practice is to predict the main pattern in a given context.

Your Assignment

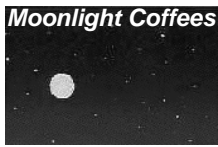
What pattern do you expect to find in the given contexts? If you do not see it, make a quick sketch of the model. Use your imagination and common sense.

Practice: Patterns

- **Model of moves in a chess game**
- **Model of tenders (quotations)**
- **Model of recipes**
- **Model of all people involved in college: students, teachers, parents, ...**
- **Rentals in a video shop**
- **Model of phases in a process**

6-31

Practice 6—2: Data Warehouse



Goal

In this practice you create a conceptual data model for a data warehouse for Moonlight Coffees Inc.

Scenario

Moonlight wants to build a data warehouse based on the detailed sales figures the shops report back on a daily basis. Examples of questions Moonlight wants the data warehouse to answer are printed below.

- What is the sales volume in \$ of coffee last month compared with the coffee sales volume same month last year?
- What is the sales volume in \$ of coffee per head in Japan compared with the average coffee sales volume in the Moonlight countries around the world?
- What is the growth of the sales volume in \$ of coffee in Sweden compared with the growth of sales volume of all products in the same geographical area? What is the growth in local currency?
- What was the total sales volume in \$ of coffee last month, compared with the total coffee sales volume in the same month last year, for the shops that have been open for at least 18 months?
- What is the growth of the sales volume in \$ of nonfoods compared to that of foods?
- What is the best day of the week for total sales in the various countries? How is that related to the average? Is the best day of the week dependent on the type of location?
- What products are most profitable per country? Globally?
- Does the service level (#employees per 1000 items sold) have influence on sales?

6-32

Your Assignment

- 1 Check the Moonlight models you created so far. Do they cater for answering the listed questions. If not, make the appropriate changes.
- 2 For a data warehouse data model, suggest the central “facts” entity.

Practice 6—3: Argos and Erats

Goal

When you model information, you make a lot of assumptions, often without being aware of this. Most of these assumptions are likely to be correct as they are usually based on experience in similar contexts or common.

This practice helps to increase your awareness of this.

Scenario

The scenario for this practice is Stranger in a Strange Land. Lost in Darkness. The Wanderer in the Mist. You name it!

Your Assignment

Make a conceptual data model based on the information in the text. Mark all the pieces in the diagram that can be confirmed from the text.

"Erats have names that are unique. Erats can have argos. Argos have names as well. The name of an argo must be unique within the erat it belongs to. Erats mutually have rondels. There are only a few different types of rondels. Erats can have one or more ubins. A ubin always consists of one or more argos of the erat, one or more rondels of the erat, or combinations of the two."

Practice 6—4: Synonym

Scenario

A synonym is, according to a dictionary, “a word having the same meaning with another (usually *almost* the same).”

Examples:

practice	- exercise
order	- command
entity	- being
order	- sequence
order	- arrangement
Command	- demand

Your Assignment

Make a conceptual data model that could be the basis for a dictionary of synonyms.

Mapping the ER Model

Introduction

Lesson Aim

This lesson describes some principles of relational databases and presents the various techniques that you can use to transform your Entity Relationship model into a physical database design.

Overview

- **Why use design modeling?**
- **Introduction to the components:**
 - **Tables**
 - **Columns**
 - **Constraints**
- **Basic Mapping**
- **Complex mapping**

7-2

Topic	See Page
Introduction	2
Why Create a Database Design?	4
Transformation Process	6
Naming Convention	8
Basic Mapping	12
Relationship Mapping	14
Mapping of Subtypes	20
Summary	30
Practice 7-1: Mapping basic Entities, Attributes and Relationships	31
Practice 7—2: Mapping Supertype	32

Topic	See Page
Practice 7—3: Quality Check Subtype Implementation	33
Practice 7—4: Quality Check Arc Implementation	34
Practice 7—5: Mapping Primary Keys and Columns	35

Objectives

At the end of this lesson, you should be able to do the following:

- Explain the need of a physical database design
- Know the concepts of the relational model
- Agree on the necessity of naming rules
- Perform a basic mapping
- Decide how to transform complex concepts

Why Create a Database Design?

The Entity Relationship model describes the data required for the business. This model should be totally independent from any implementation considerations. This same ER model could also be used as a basis for implementation of any type of DBMS or even a file system.

Why Create a Data Design Model?

- **Closer to the implementation solution**
- **Facilitates discussion**
- **Ideal model can be adapted to an RDBMS model**
- **Sound basis for physical database design**

7-3

A New Starting Point An Entity Relationship model is a high-level representation which cannot be implemented as is.

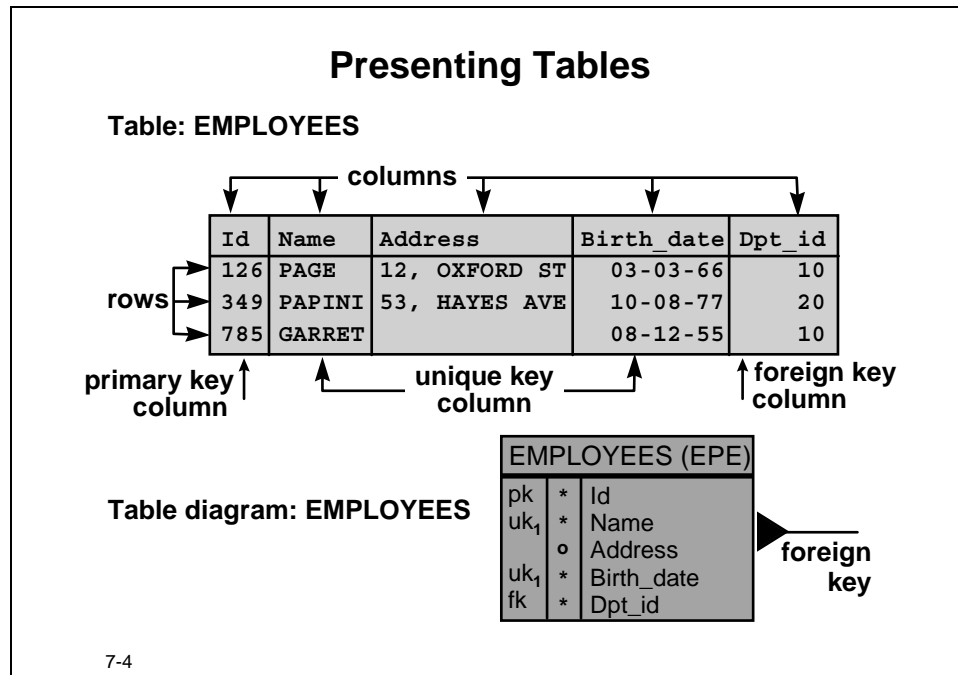
People creating these models may not be aware of physical and database constraints, but they still have to provide a conceptually “workable” solution. This is why it is important to have a validated and agreed ER model before going into the physical database design.

Transforming the ER model, creates a “first-cut” database design. This first-cut design is intended to serve as a new basis for defining the physical implementation of the database.

This new model can easily be used for further discussions between designers, developers, and database administrators.

Presenting Tables

Tables are supported by integrity rules that protect the data and the structures of the database. Integrity rules require each table to have a primary key and each foreign key to be consistent with its corresponding primary key.



Tables A table is a very simple structure in which data is organized and stored. Tables have columns and rows. Each column is used to store a specific type of value. In the above example, the EMPLOYEES table is the structure used to store employees' information.

Rows Each row describes an occurrence of an employee. In the example, each row describes in full all properties required by the system.

Columns Each column holds information of a specific type like Id, Name, Address, Birth Date, and the Id of the department the employee is assigned to.

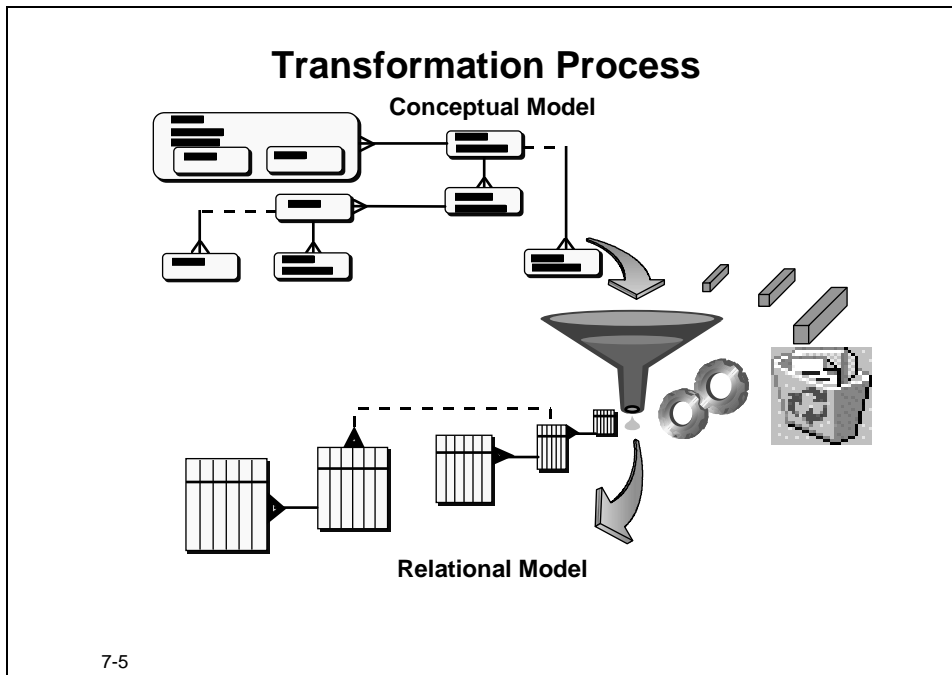
Primary keys The Id column is a primary key, that is, every employee has a unique identification number in this table which distinguishes each individual row.

Unique keys Both columns Name and Birth_date are associated with a Unique key constraint which means that the system does not allow two rows with the same name and Birth_date. This restriction defines the limits of the system.

Foreign keys The foreign key column enables the use of the Dpt_id value to retrieve the department properties for which a specific employee is working.

Transformation Process

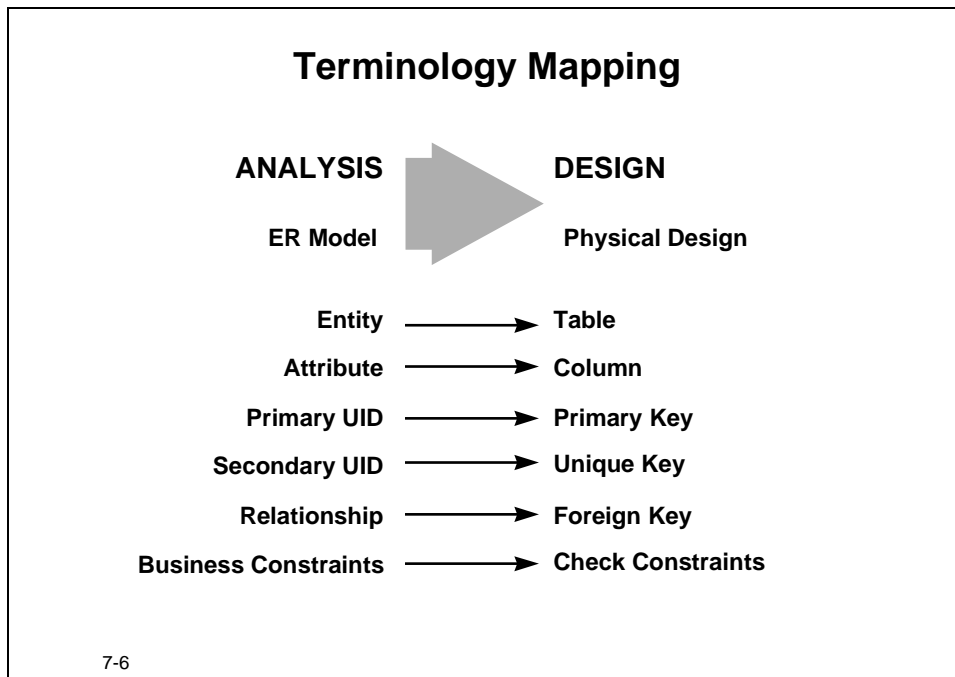
Using transformation rules you create a new model based on the conceptual model.



Conceptual Model The way you can describe requirements for the data business requires using a semantically rich syntax through graphical representation. As you have seen in previous chapters, you can describe many of the business rules with graphical elements such as subtypes, arcs, and relationships (barred and nontransferable ones). The only constraints in expressing business complexity that you have encountered so far are the graphical limitations. We know that this model acts as a generic one, because it is not related to any physical considerations. Therefore you can use it for any type of database. Nevertheless, it may be that the DBMS type you want to use (relational or others) does not support all of the semantic rules graphically expressed in your ER model.

Relational Model The Relational model is based on mathematical rules. This means that when you try to fit all of the syntax from the ER model into the physical database model, some of it may not have any correspondence in the relational model. To preserve these specified rules, you have to keep track of them and find the correct way to implement them.

Terminology Mapping



Changing from analysis to design also means changing terminology.

Using a very simple basis:

- An entity leads to a table.
- An attribute becomes a column.
- A primary unique identifier produces a Primary key.
- A secondary unique identifier produces a Unique key.
- A relationship is transformed into a Foreign key and foreign key columns.
- Constraints are the rules with which the database must cope to be consistent. Some of the business rules are translated into Check Constraints, other complex ones require additional programming and you can implement them at client side or server side or both.

This initial mapping of an ER model is limited to the design of tables, columns, and constraints that can be declared. A ***declarative constraint*** is a business constraint that can be ensured at the server level using database language statements only and requires no coding.

Naming Convention

Before transforming the ER diagram you probably need to define a naming convention so that people working on the project use the same standards and produce the same model from the same source. Rules explained here are the ones used within Oracle. Even though they are efficient, they are not the only ones that you can use. You or your company can provide the company's own standard as part of its method.

General Naming Topics

Decide on a convention for:

- Table names
- Special characters (% , * , # , - , space , ...)
- Table short names
- Column names
- Primary and Unique Key Constraint names
- Foreign Key Constraint names
- Foreign Key Column names

7-7

Naming of Tables

The plural of the entity name is used as the corresponding table name. The idea is that the Entity is the concept of an abstract thing—you can talk about EMPLOYEE, CUSTOMER, and so on, so singular is a good naming rule, but a table is made up of rows (the EMPLOYEES table, or CUSTOMERS table) where the plural is more appropriate.

Naming of Columns

Column names are identical to the attribute names, with a few exceptions. Replace special characters with an underscore character. In particular, remove the spaces from attribute names, as SQL does not allow spaces in the names of relational elements. Attribute Start Date converts to column Start_date; attribute Delivered Y/N transforms to Delivered_y_n (or preferably Delivered_Ind). Often column names use more abbreviations than attribute names.

Short Names

A unique short name for every table is a very useful element for the naming of foreign key columns or foreign key constraints. A suggested way to make these short names is based on the following rules:

- For entity names of more than one word, take the:
 - First character of the first word.
 - First character of the second word.
 - Last character of the last word.

For example entity PRICED PRODUCT produces PPT as a short table name.
- For entity names of one word but more than one syllable, take the:
 - First character of the first syllable.
 - First character of the second syllable.
 - Last character of the last syllable.

For example EMPLOYEE gives EPE as a short name.
- For entity names of one syllable, but more than one character, take the:
 - First character.
 - Second character.
 - Last character.

For example FLIGHT gives FLT.

This short name construction rule does not guarantee uniqueness among short names but experience has proved that duplicated names are relatively rare.

In case two short names happen to be the same, just add a number to the one that is used less often giving, for example, CTR for the most frequently used one and then CTR1 for the second one.

Naming of Foreign Key Constraints

The recommended rule for naming foreign key constraints is
<short name of the from table> _ <short name of the to table> _ <fk>.

For example, a foreign key between tables EMPLOYEES and DEPARTMENT results in constraint name *epe_dpt_fk*.

Naming of Foreign Key Columns

Foreign key columns are prefixed with the short name of the table they refer to. This leads to foreign key column names like *dpt_no*. Limiting the attribute name to 22 characters enables you to add two prefixes plus two underscores to the column name. This may occur in the event of cascade barred relationships. This is discussed later in the lesson.

Multiple Foreign Keys Between Two Tables

If there are two (or more) foreign keys between two tables then the foreign keys and foreign key columns would be entitled to the same name. In this situation, add the name of the relationship to *both* foreign key names. Do the same with the foreign key columns. This way you will never mistake one foreign key for the other.

For example, in the model of Electronic Mail entity LIST ITEM has two relationships with ALIAS (one of them is at the subtype level). The naming would result in the two foreign key names: *lim_als_in* and *lim_als_referring_to*. The foreign key columns would be named *Als_id_in* and *Als_id_referring_to*.

Naming of Check Constraints

Check Constraints are named <table short name>_ck_<sequence_number>, such as epe_ck_1, epe_ck_2 for the first and second check constraint on table EMPLOYEES.

Naming Restrictions with Oracle

Each RDBMS can have its own naming restrictions. You need to know if the convention you decide to use is compatible with it.

Naming Restrictions with Oracle

- **Table and column names:**
 - **Must start with a letter**
 - **May contain up to 30 alphanumeric characters**
 - **Cannot contain space or special characters**
- **Table names must be unique within a schema.**
- **Column names must be unique within a table.**

7-8

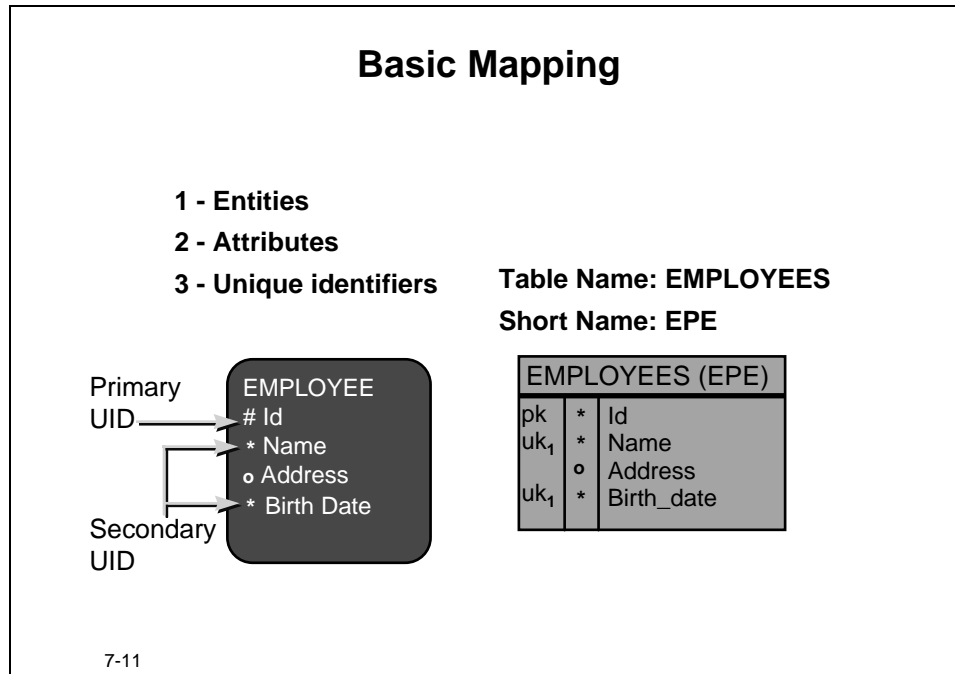
- You can use any alpha-numeric character for naming tables and columns as long as the name:
 - Starts with a letter.
 - Is up to 30 characters long.
 - Does not include special characters such as “!” but “\$”, “#” and “_” permitted.

- Table names must be unique within the schema that is shared with views and synonyms.
 - Within the same table two columns cannot have the same name.
 - Be aware also of the reserved programming language words that are not allowed for naming objects. Avoid names like:
 - Number
 - Sequence
 - Values
 - Level
 - Type
- for naming tables or columns. Refer to the RDBMS reference books for these.

Basic Mapping

Entity Mapping

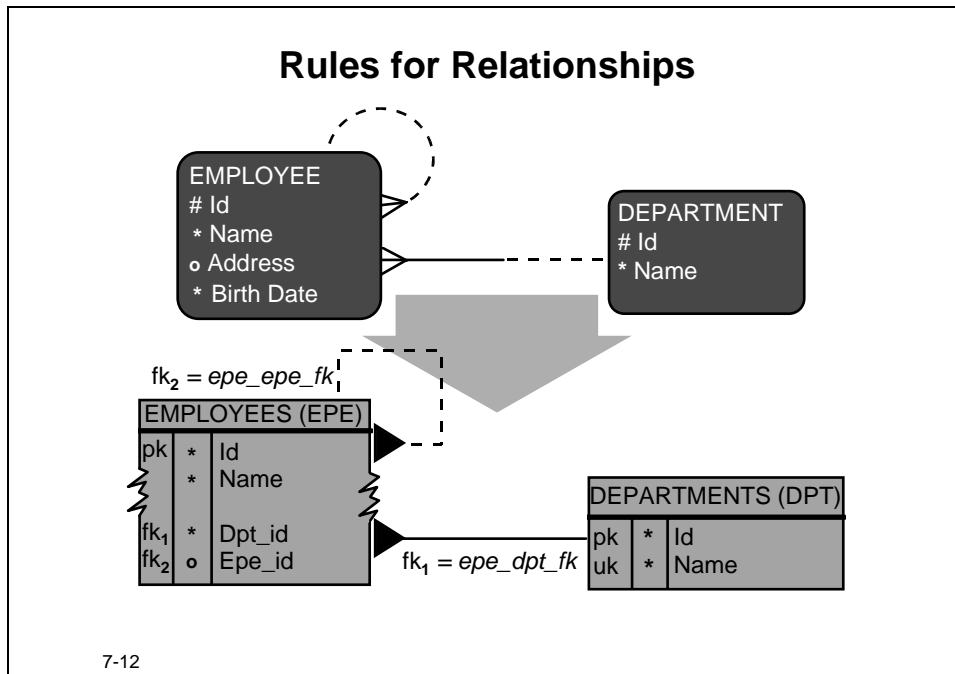
Before going into complex transformation we will look at the way to transform simple entities.



- 1** Transform entities into tables using your own naming convention or the one previously described.
 In this example the entity EMPLOYEE produces a table name EMPLOYEES and a short name EPE.
 Use a box to represent tables on a diagram.
- 2** Each attribute creates a column in the table and the characteristics such as mandatory or optional have to be kept for each column. Using the same notation “*” or “o” facilitates recognition of these characteristics on a diagram.
- 3** All unique identifiers are transformed. A primary unique identifier is transformed into a Primary key. The notation “pk” next to the column name indicates the Primary key property. If more than one column is part of the primary key, use the “pk” notation for each column.

You need to implement secondary unique identifiers, even if they do not appear on your ER diagram. To preserve this property, secondary UIDs are transformed as unique keys. In the above example, the values for the combination of two columns must be unique. They belong to the same unique key and each column has a uk₁ notation to indicate this. If, in future, another unique key comes to exist for that table, it would be notated as uk₂.

Rules for Relationships



Foreign Key Columns: A relationship creates one or more foreign key columns in the table at the many side. Using previous naming rules, the name of this foreign key column is *Dpt_id* for the relationship with **Department** and *Epe_id* for the recursive relationship. This ensures that column names such as *Id*, coming from different tables, still provide a unique column name in the table.

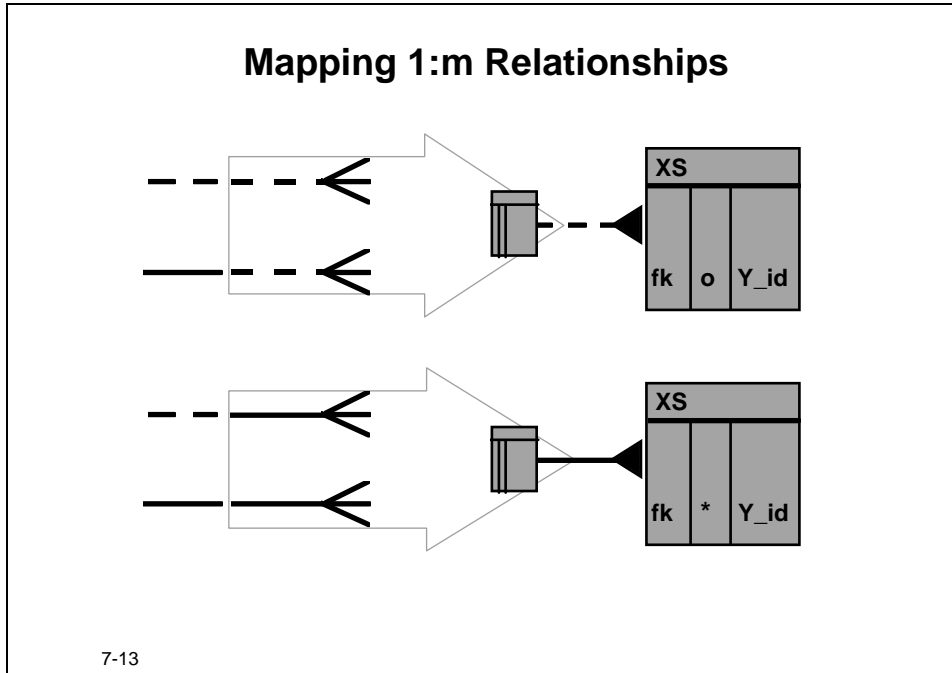
Depending on whether or not the relationship is required, the foreign key column is mandatory or optional.

Foreign Key Constraints: The foreign key constraints between **EMPLOYEES** and **DEPARTMENTS** is *epe_dpt_fk*. The recursive one between **EMPLOYEES** and **EMPLOYEES** is called *epe_epe_fk*.

Relationship Mapping

Mapping of One-to-Many Relationships

As previously mentioned, some of the meaning that is expressed in an ERD cannot be reproduced in the physical database design.



A relationship in an ER Diagram expresses the rules that apply between two entities, from two points of view. The notation used in the ERD is rich enough to tell, for example, that the relationship is mandatory on **both** sides. The illustration shows that the 1:m relationships that are mandatory at the one side are implemented in exactly the same way as the ones that are optional at the one side. This means that part of the content of the ER model is lost during transformation, due to the relational model limitations. You need to keep track of these incomplete transformations; they must be implemented using a mechanism other than a declarative constraint.

Mapping of Mandatory Relationship at the One Side

In case of the implementation of a relationship that is mandatory at the one side you need to check two things.

- You cannot create any master record without at least one detail record.
- When deleting details you must be sure that you do not delete the last detail for a master record, or alternatively, you must delete the master record together with its last detail.

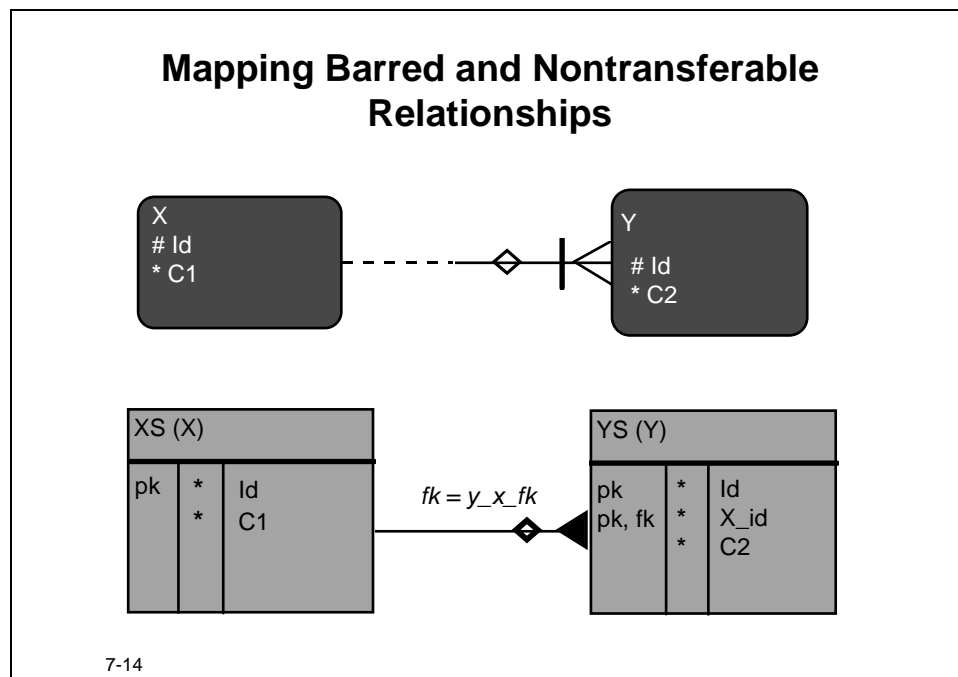
You can implement code to check this on the server side or on the client side. In an Oracle environment this was usually done at the client side. Since Oracle 8, on the server side Oracle offers implementation possibilities that were not available in previous releases.

Optional Composed Foreign Keys

When a foreign key is made of two or more columns, and the foreign key is optional, all foreign key columns must be defined as optional. Note that if you enter a value in one of the foreign key columns, but not in the other one, Oracle will not fire the foreign key constraint check.

You would need additional code to check that either all or none of the foreign key columns have a value, but exclude the possibility of a partially-entered key.

Mapping of Nontransferable Relationships



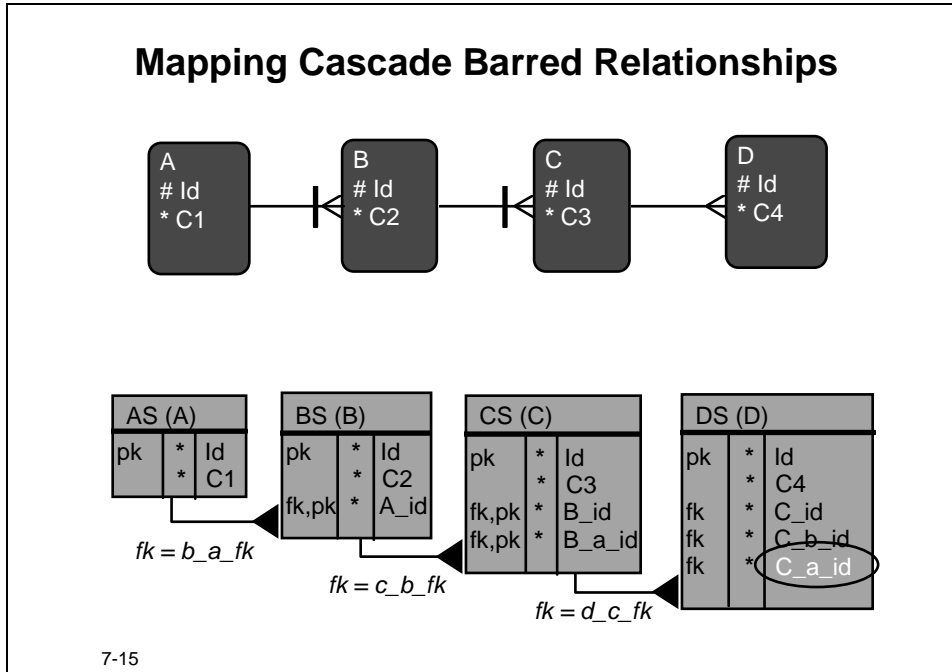
This relationship property does not migrate to the physical database design because it has no natural counterpart in an RDBMS, although you can code a solution at the server side. In the example, you would create an update trigger at table YS that fails when the foreign key column X_id is updated.

Mapping Barred Relationships

A barred relationship, like any other relationship, is mapped into a foreign key. The foreign key column is also part of the primary key, and thus plays a double role.

Mapping of Cascade Barred Relationships

A Cascade Barred relationship may lead to long column names as the illustration shows.



To avoid column names that could end up with more than 30 characters, the suggested convention is never to use more than two table prefixes.

The usual choice for the foreign key column names is:

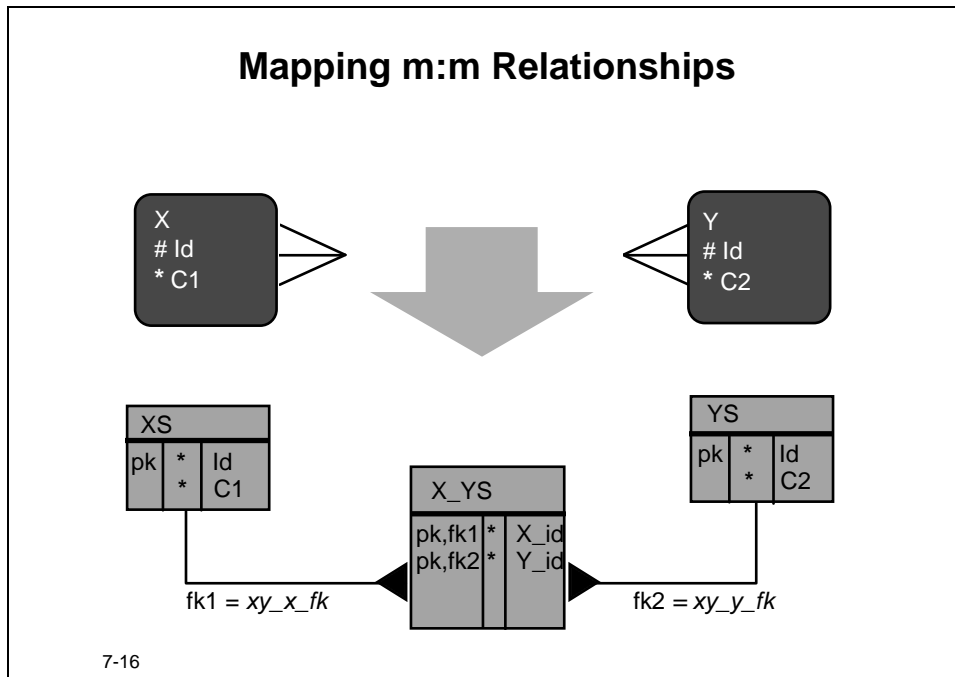
<nearest by table short name> _ <farthest table short name> _ <column name>

In the above example the foreign key column in DS that comes all the way from AS through BS and CS is named C_a_id instead of C_b_a_id.

As the short names are usually three characters long, this rule explains why attribute names should not have more than 22 characters.

Mapping of Many-to-Many Relationships

When transforming a many-to-many relationship, you create an *intersection table*.

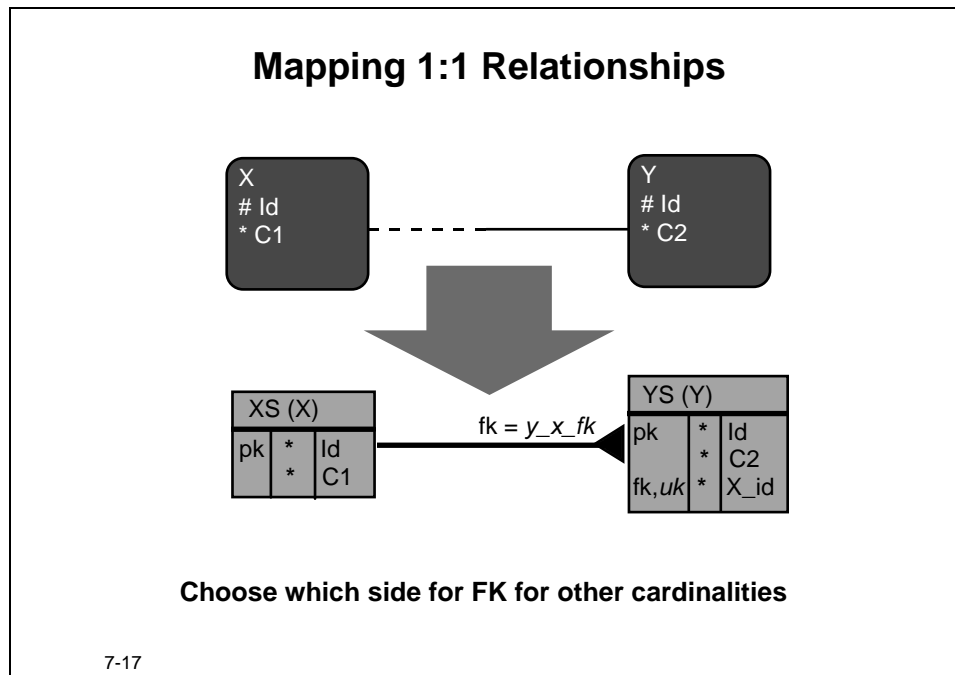


The intersection table contains all the combinations that exist between XS and YS.

- This table has no columns other than foreign key columns. These columns together form the primary key.
- The rule for naming this table is short name of the first table (in alphabetical order) and full name of the second one. This would give a many-to-many relationship between tables EMPLOYEES and PROJECTS an intersection table named EPE_PROJECTS.
- Whether the relationship was mandatory or not, the foreign key columns are always mandatory.

Note this table is identical (except, possibly, for its name) to the table that would result from an intersection entity that could replace the m:m relationship.

Mapping of One-to-One Relationships



When transforming a one-to-one relationship, you create a foreign key and a unique key. All columns of this foreign key are also part of a *unique key*.

If the relationship is mandatory on one side, the foreign key is created at the corresponding table. If the relationship is mandatory on both sides or optional on both sides, you can choose on which table you want to create the foreign key. There is no absolute rule for deciding on which side to implement it.

If the relationship is optional on both sides you may decide to implement the foreign key in the table with fewer numbers of rows, as this would save space.

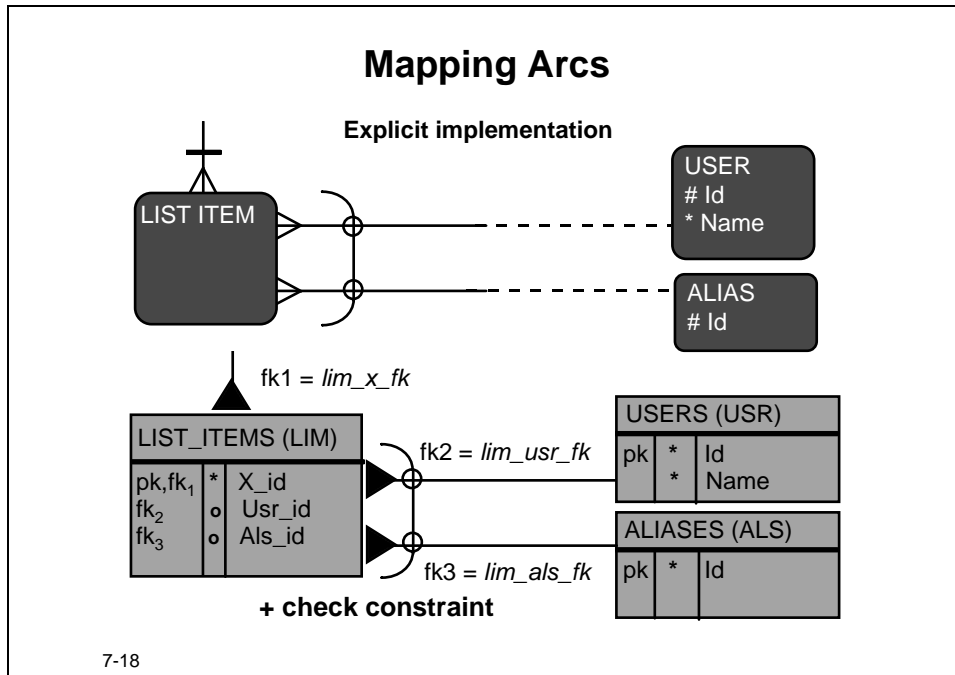
If the relationship is mandatory at both ends, we are facing the same RDBMS limitation you saw earlier. Therefore, you need to write code to check the mandatory one at the other side, just as you did to implement m:1 relationships that are mandatory at the one end.

Alternative Implementations

A 1:1 relationship between two entities can be implemented by a single table. This is probably the first implementation to consider. It would not need a foreign key constraint.

A third possible implementation is to create an intersection table, as if the relationship was of type m:m. The columns of each of the foreign keys of the intersection table would be part of unique keys as well.

Mapping of Arcs



The first solution illustrated above shows that there are as many foreign keys created as there are relationships. Therefore a rule must be set to verify that if one of the foreign keys is populated, the others must not be populated (which is the exclusivity principle of the relationships in an arc) and that one foreign key value must always exist (to implement the mandatory condition).

From a diagram point of view, all foreign keys must be optional, but additional code will perform the logical control. One solution on the server side is to create a check constraint at **LIST_ITEMS** as is:

```
CHECK      (      usr_id IS NOT NULL
              AND   als_id IS NULL)
OR         (      usr_id IS NULL
              AND   als_id IS NOT NULL) .
```

This controls the exclusivity of mandatory relationships.

In case the relationships are optional, you need to add:

```
OR      (usr_id IS NULL AND als_id IS NULL)
```

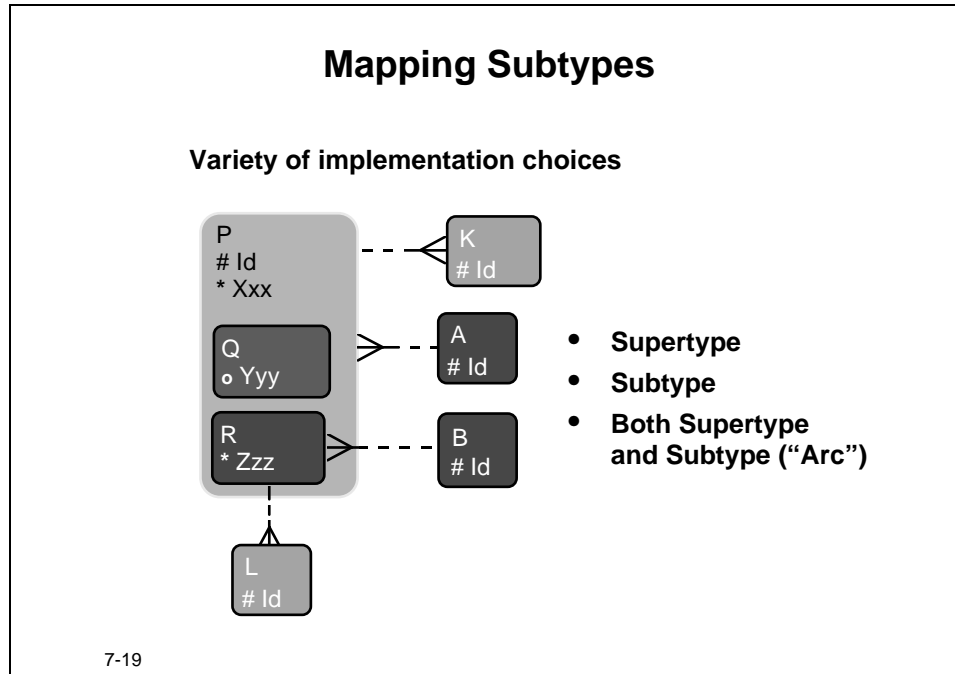
An other syntax that is often used:

```
DECODE (usr_id, NULL, 0, 1)
+ DECODE (als_id, NULL, 0, 1) = 1;
(or =<1 for optional relationship) .
```

You can also map arcs in a different way using the *generic arc* implementation. This is a historical solution that you may encounter in old systems. You should not use it in new systems. It is discussed in the lesson on Design Considerations.

Mapping of Subtypes

In mapping subtypes, you must make a choice between three different types of implementations. All three are discussed in detail.



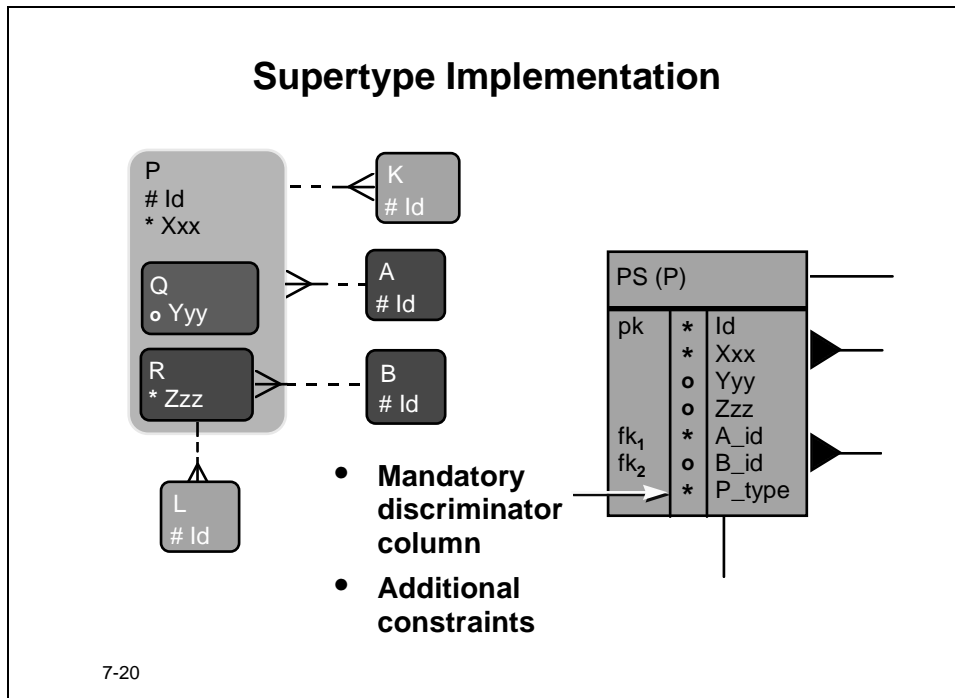
Supertype Implementation

This choice produces one single table for the implementation of the entities P, Q, and R. The supertype implementation is also called single (or one) table implementation.

Rules

- 1 Tables:
 - Independent of the number of subtypes, only one single table is created.
- 2 Columns:
 - The table gets a column for all attributes of the supertype, with the original optionality.
 - The table also gets a column for each attribute belonging to the subtype but the columns are all switched to optional.
 - Additionally, a mandatory column should be created to act as a *discriminator* column to distinguish between the different subtypes of the entity. The value it can take is from the set of all the subtype short names (DBE, DBU in the example). This discriminator column is usually called <table_short_name> _type, in the example Dba_type.
- 3 Identifiers:
 - Unique identifiers translate into primary and unique keys.

- Unique identifiers at subtype level usually translate into a unique key or check constraint only.



4 Relationships:

- Relationships at the supertype level transform as usual. Relationships at subtype level are implemented as foreign keys, but the foreign key columns all become optional.

5 Integrity constraints:

- For each particular subtype, all columns that come from mandatory attributes must be checked to be NOT NULL.
- For each particular subtype, all columns that come from attributes or relationships of other subtypes must be checked to be NULL.

Note: You may avoid the use of the discriminator column if you have one mandatory attribute in each subtype. The check is done directly on these columns to find out what type a specific row belongs to.

When to Consider Supertype Implementation

The single table implementation is a common and flexible implementation. It is the one you are likely to consider first and is specially appropriate when:

- Most of the attributes are at the supertype level.
- Most of the relationships are at the supertype level.
- The various subtypes overlap in the required functionality.

- The access path to the data of the various types is the same.
- Business rules are globally the same for the subtypes.
- The number of instances per subtype does not differ too much, for example, one type having more than, say, 1000 times the number of instances of the other.
- An instance of one subtype can become an instance of another, for example, imagine an entity ORDER with subtypes OPEN ORDER and PROCESSED ORDER, each subtype having its own properties. An OPEN ORDER may eventually become a PROCESSED ORDER.

Additional Objects

Usually you would create a view for every subtype, showing only the columns that belong to that particular subtype. The correct rows are selected using a condition based on the discriminator column. These views are used for all data operations, including inserts and updates. All applications can be based on the view, without loss of performance.

The supertype table plus subtype views is an elegant and appropriate implementation and should be considered as first choice.

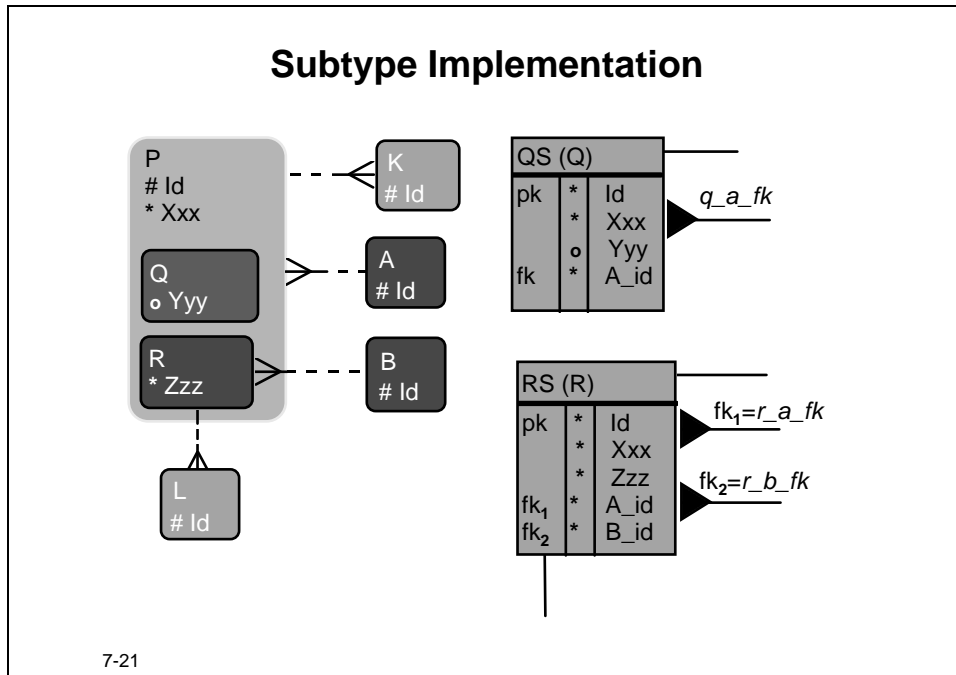
Consequences for Tables Based on K and L

The foreign key in the table based on K is straightforward.

The foreign key of the table based on L is more complex. The supertype implementation would mean that the foreign key refers to a valid P, not to the more limited set of R's. This must be checked with an additional constraint.

Subtype Implementation

This subtype table implementation (often loosely referred to as *two-table* implementation) produces one table for each of the subtypes, assuming there are only two subtypes, such as Q and R.



Rules

1 Tables:

- One table per first level subtype.

2 Columns:

- Each table gets a column for all attributes of the supertype, with the original optionality.
- Each table also gets a column for each attribute belonging to the subtype, also with the original optionality.

3 Identifiers:

- The primary unique identifier at the supertype level creates a primary key for each of the tables. Alternatively, if the subtypes had their own UID, this one are used as the basis for the primary key.
- Secondary identifiers of the supertype become unique keys within each table.

4 Relationships:

- All tables get a foreign key for a relationship at the supertype level with the original optionality.

- For the relationships at the subtype levels, the foreign key is implemented in the table it is mapped to. The original optionality is retained.

5 Integrity constraints:

- No specific additional checks are required. Only when the Id values must be unique across all subtypes would it need further attention.

When to Consider a Subtype Implementation

You can regard this implementation as a horizontal partitioning of the supertype. It may be appropriate when:

- The resulting tables will reside in different databases (distribution). This may occur when different business locations are only interested in a specific part of the information.
- When the common access paths for the subtypes are different.
- Subtypes have almost nothing in common. This may occur when there are few attributes at the supertype and many at the subtype levels. An example can be found in the Electronic Mail model. Entity ADDRESS has two subtypes: MAIL LIST and ALIAS. These subtypes only share the fact that they can be used as addressee for a message, but their other properties are completely different.
- Most of the relationships are at the subtype level. This is the case especially if both tables are to be implemented in different databases, and the foreign key integrity constraint for the supertype may not be verified in all cases.
- Business functionality and business rules are quite different between subtypes.
- The way tables are used is different, for example, one table being queried while the other one is being updated. A one-table solution could result in performance problems.
- The number of instances of one subtype is very small compared to the other one.

Additional Objects

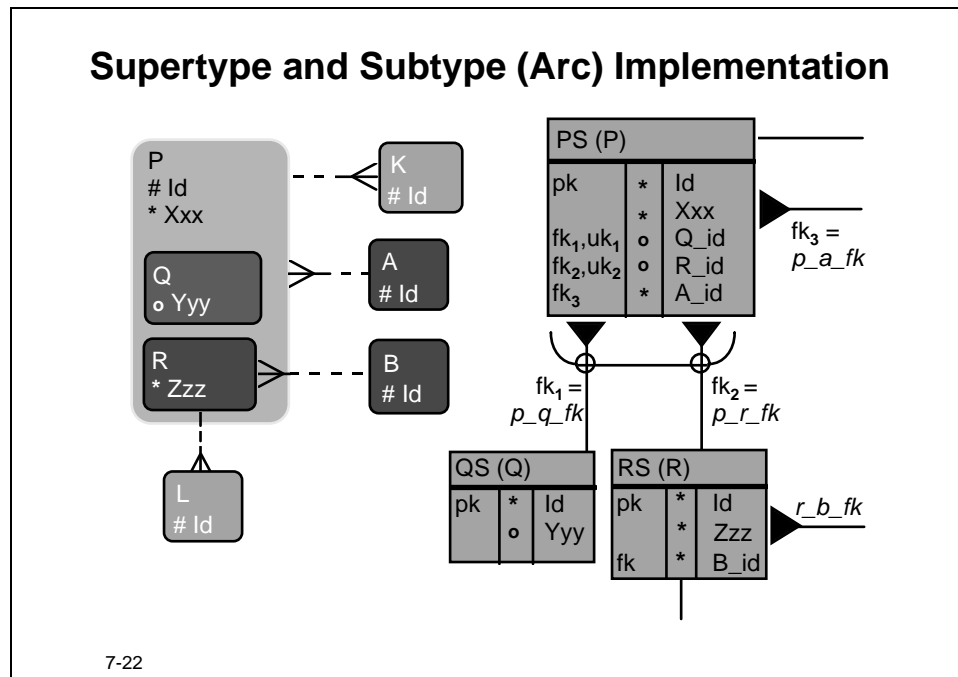
Usually you would create an additional view that represents the supertype showing all columns of the supertype and various subtypes. The view select statement must use the union operator. The view can be used for queries only, not for data manipulation.

Consequences for Tables Based on K and L

The foreign key in the table based on L is straightforward and should refer to the table based on R.

The foreign key of the table based on K is now more complex. This must be implemented as two optional foreign keys, one to each of the tables based on Q and R. An extra check is needed to make sure that both foreign keys do not have a value at the same time; this is identical to an ordinary arc check.

Both Supertype and Subtype “Arc” Implementation



This choice produces one table for every entity, linked to foreign keys in an exclusive arc at the PS side. It is the implementation of the model as if the subtypes were modeled as standalone entities with each one having an *is subtype of / is supertype of* relationship to the supertype. These relationships are in an arc. Therefore this implementation is also called ***Arc Implementation***. See also the chapter on Constraints for more details about subtypes compared to the arc.

Rules

- 1 Tables:
 - As many tables are created as there are subtypes, as well as one for the supertype.
- 2 Columns:
 - Each table gets a column for all attributes of the entity it is based on, with the original optionality.
- 3 Identifiers:
 - The primary UID at the supertype level creates a primary key for each of the tables.
 - All other unique identifiers transform to unique keys in their corresponding tables.
- 4 Relationships:
 - All tables get a foreign key for a relevant relationship at the entity level with

the original optionality.

5 Integrity constraints:

- Two additional columns are created in the table based on the supertype. They are foreign key columns referring to the tables that implement the subtypes. The columns are clearly optional as the foreign keys are in an arc. The foreign key columns are also part of the unique keys because, in fact, they implement a mandatory one-to-one relationship.
- An additional check constraint is needed to implement the arc.

When to Consider a Both Supertype and Subtype Implementation

This solution performs a double partitioning. It is used relatively rarely, but could be appropriate when:

- The resulting tables reside in different databases (distribution). This may occur when different business locations are only interested in a specific part of the information.
- Subtypes have almost nothing in common and each table represents information that can be used independently, for example, when the PS table gives all global information and both QS and RS give specific information, and the combination of global and specific information is hardly ever needed.
- Business rules are quite different between all types.
- The way tables are used and accessed is different.
- Users from different business areas need to work with the same rows at the same time, but with different parts of the rows, which could result in locking problems and a performance issue.

Additional Objects

Although you would hardly use them, you could consider creating additional views that represent the supertype and various subtypes in full.

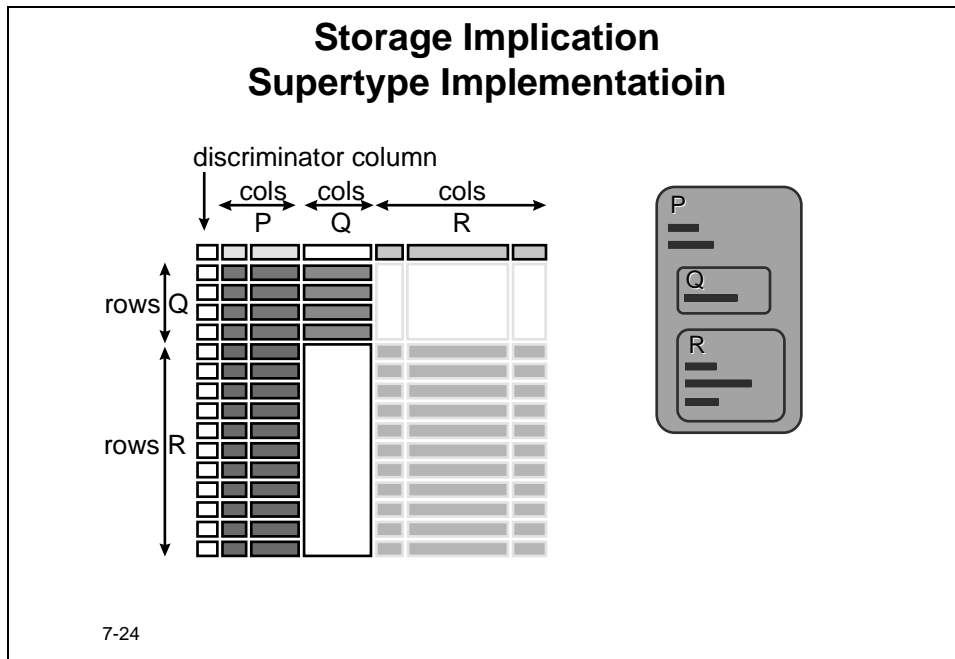
Consequences for Tables Based on K and L

Both foreign keys can be implemented straightforwardly without additional checks.

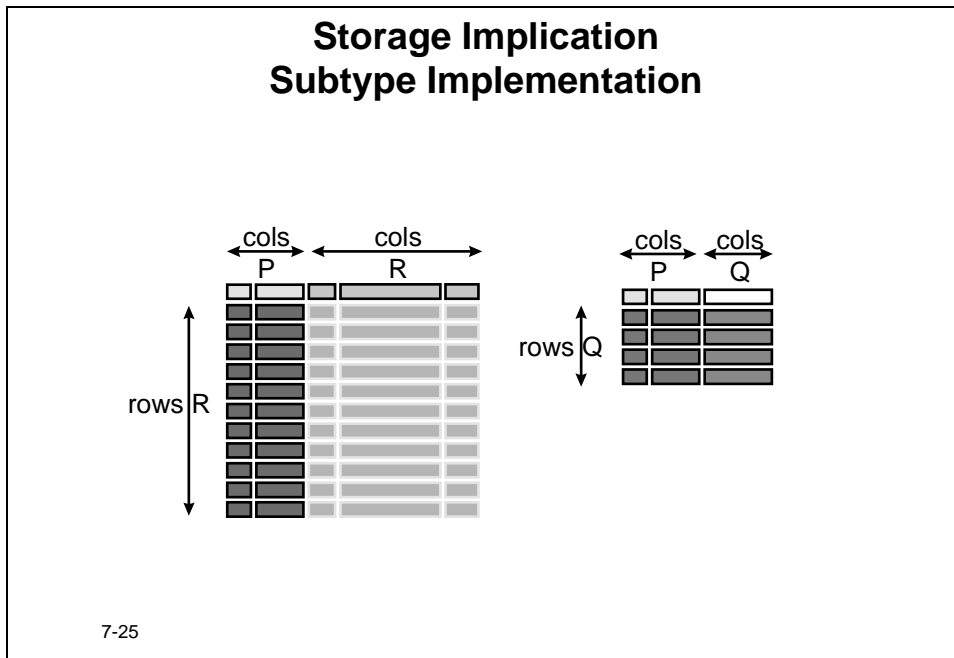
Storage Implication

The illustrations show the differences between the one, two, and three table implementations. In most database systems empty column values do take some bytes of database space (although this sounds contradictory). In Oracle this is very low when the empty columns are at the end of the table and when the data type is of variable size.

Supertype Implementation All rows for both types are in one table. Note the empty space in the Q rows at the R columns and vice-versa.

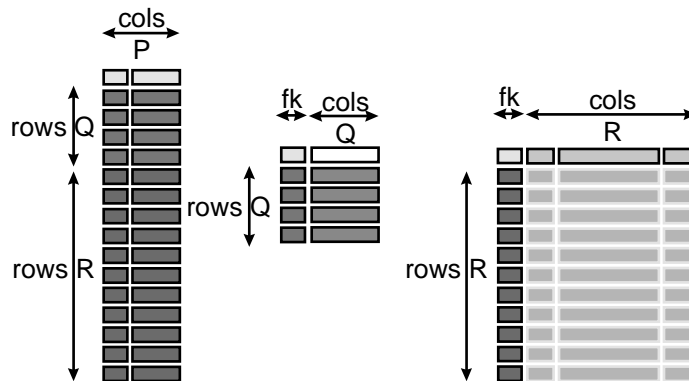


Subtype Implementation In the two table implementation the “empty space” of the one-table implementation is gone. This is a horizontal split of the table.



Arc Implementation In this three table implementation the one table is sliced vertically into a P-columns-only portion. The remaining part is horizontally split into the Q and R columns and rows. An additional foreign key column at P, or a foreign key column at both Q and R is needed to connect all the pieces together.

Storage Implication Supertype and Subtype (Arc) Implementation



7-26

Summary

Summary

- **Relational concepts**
- **Naming rules convention**
- **Basic mapping**
- **Complex mapping**

7-27

Relational databases implement the relational theory they are based on.

A coherent naming rule can prevent many errors and frustrations and adds to the understanding of the structure of the database schema.

You have seen how to map basic elements from an ER model such as entities and relationships. You can do this very simply. There are also complex structures which require decisions on how to transform them. Some ER model elements can only be implemented by coding check constraints or database triggers. These are specific to Oracle and not part of the ISO standard for relational databases.

Practice 7—1: Mapping basic Entities, Attributes and Relationships

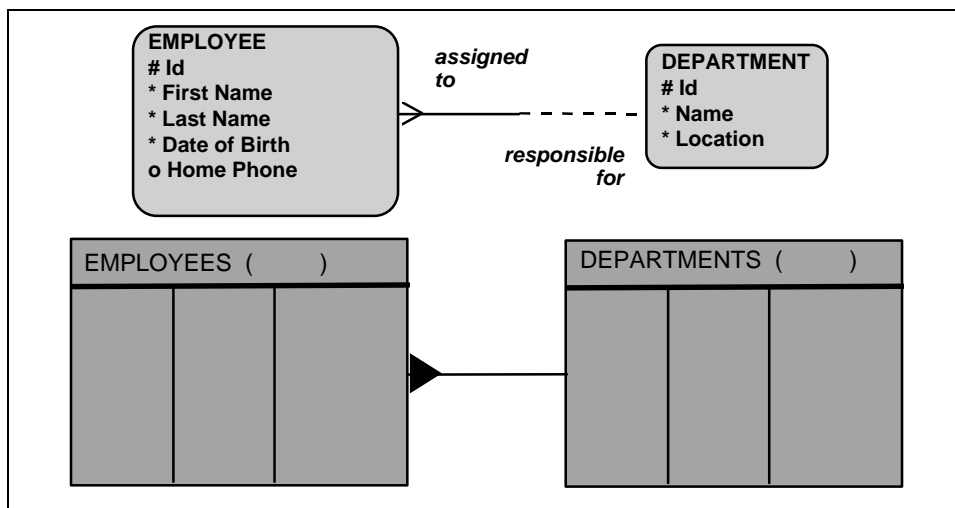


Goal

In this practice, you are to create a basic mapping of a conceptual model into a first cut logical mapping of your database.

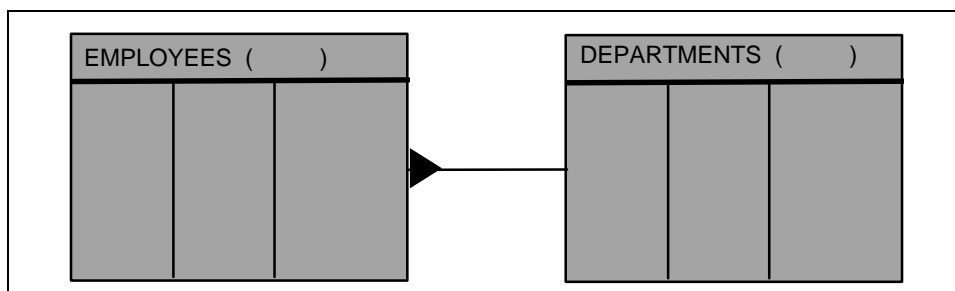
Scenario

The following is part of the simple Moonlight ER model showing the entities of DEPARTMENT and EMPLOYEE. Map the entities, attributes, relationships, optionality, and keys of the following diagram.



Your Assignment

- 1 Map both entities to tables and all attributes to columns.
- 2 Map relationships to foreign keys columns and mark as (fk).
- 3 Map all optionality tags to not nulls (*).
- 4 Map UID tags to primary keys (pk).
- 5 On the table diagram, name all the elements that must be created following this implementation. Use the naming convention as described in this lesson, or use your own rules. Give proper names to the columns and foreign key constraints.



Practice 7—2: Mapping Supertype

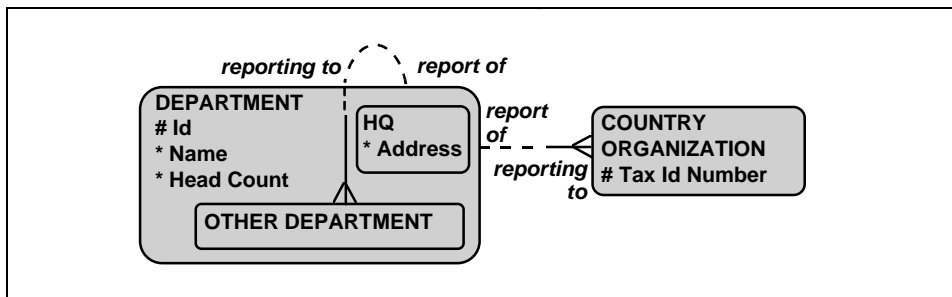


Goal

In this practice, you create a complex mapping and test your understanding of the transformation process.

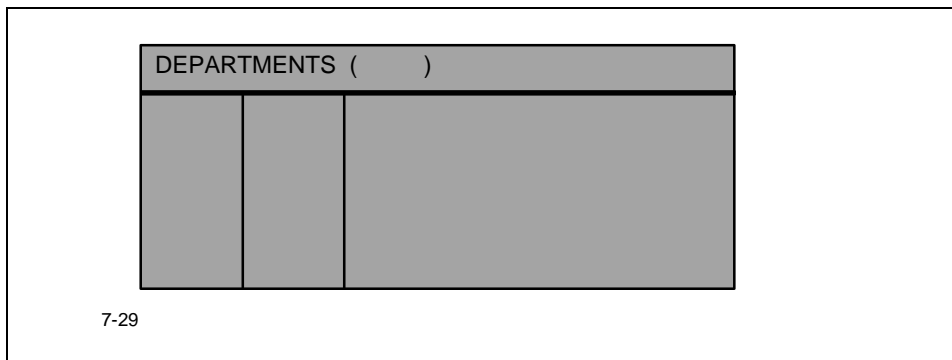
Scenario

Here is part of the Moonlight ER model showing the entity DEPARTMENT. One of the analysts has decided to implement the DEPARTMENT entity and its subtypes as a single table.



Your Assignment

- 1 What would have been the rationale of this choice?
- 2 On the table diagram, name all the elements that must be created following this supertype implementation. Use the naming convention as described in this lesson, or use your own rules. Give proper names to the columns and foreign key constraints and identify check constraints, if any.



7-29

Practice 7—3: Quality Check Subtype Implementation

Moonlight Coffees

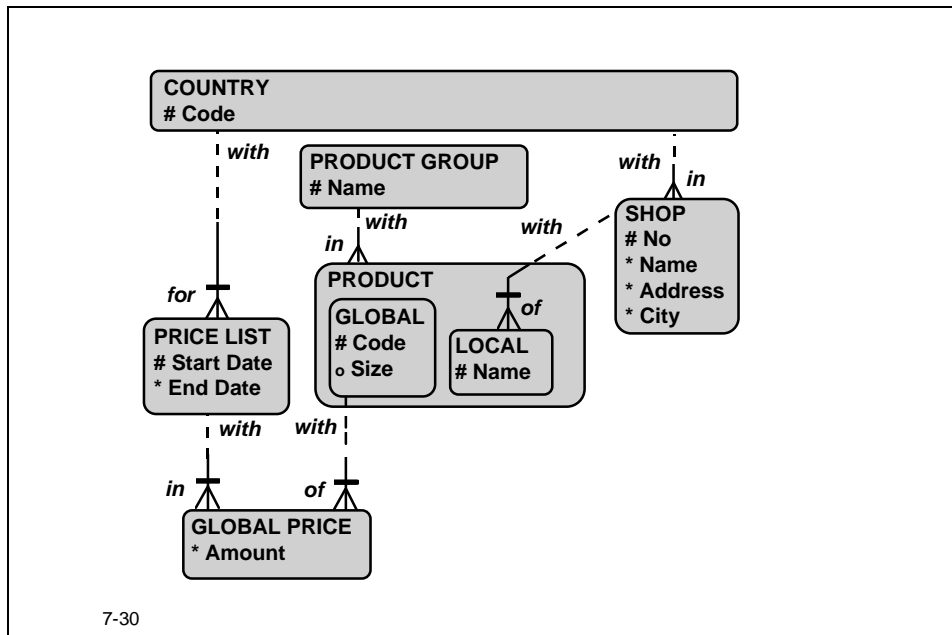


Goal

In this practice you perform a quality check on table mappings that were created by someone who is supposed to use the naming convention that is described in this lesson.

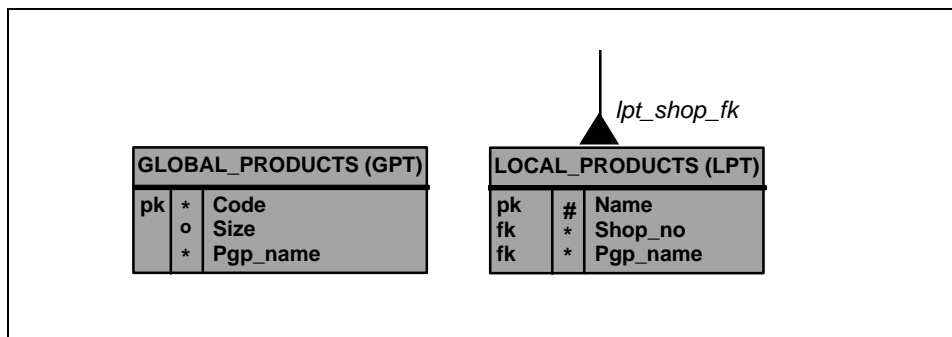
Scenario

Here is a part of the Moonlight ER model.



Your Assignment

Perform a quality check on the proposed subtype implementation of entity PRODUCT.



Practice 7—4: Quality Check Arc Implementation

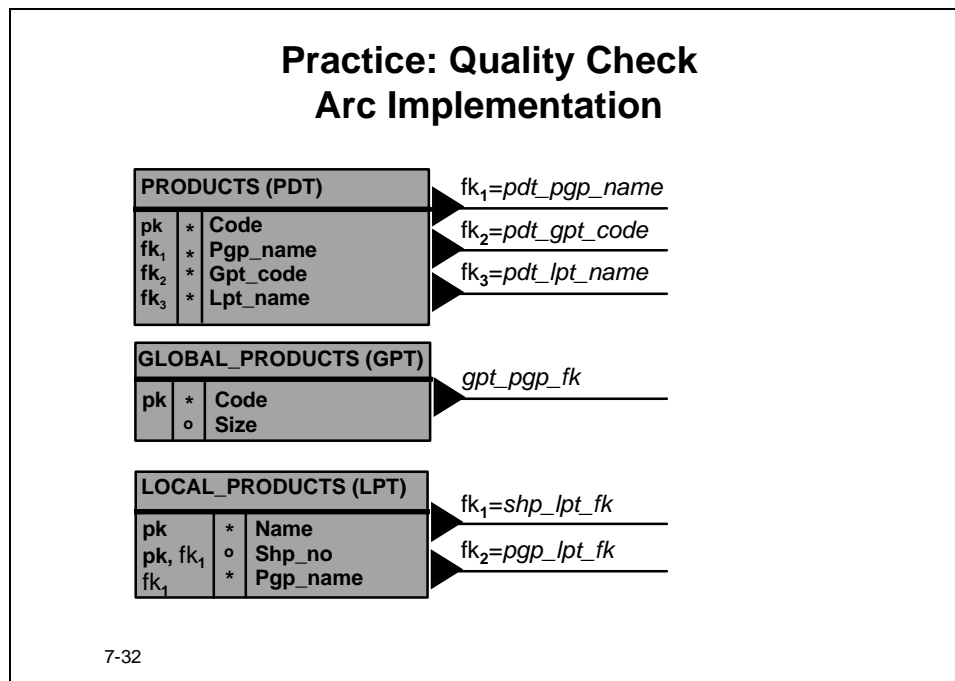


Goal

The purpose of this practice is to do a quality check on table mappings that were created by someone else who is supposed to use the naming convention that is described in this lesson.

Scenario

This practice is based on the same ER diagram as the previous practice.



Your Assignment

Perform a quality check on the proposed supertype and subtype implementation of the entity PRODUCT and its subtypes. Also, check the selected names.

Practice 7—5: Mapping Primary Keys and Columns



Goal

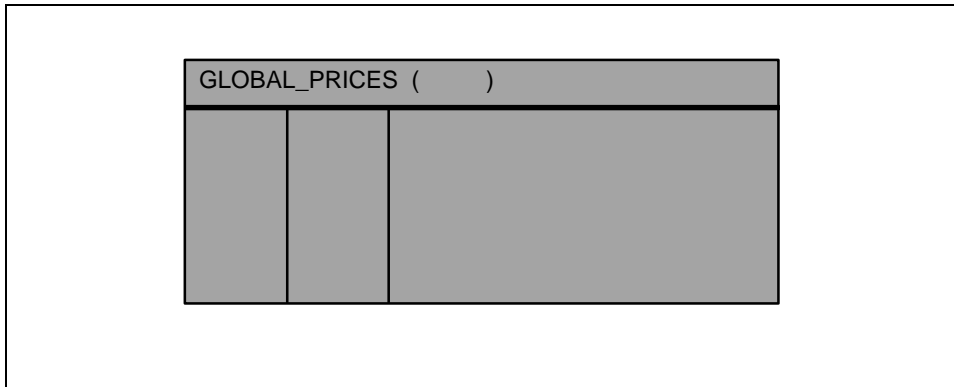
The purpose of this practice is to do a complex mapping of primary keys and columns.

Scenario

This practice is based on the same model that was used in the previous practice.

Your Assignment

Identify the Primary key columns and names resulting from the transformation of the GLOBAL PRICE entity. Give the short name.



Denormalized Data

Introduction

Lesson aim

This lesson shows you the most common types of denormalization with examples.

Overview

- Denormalization
- Benefits
- Types of denormalization

8-2

Topic	See Page
Why and When to Denormalize	4
Storing Derivable Values	6
Pre-Joining Tables	8
Hard-Coded Values	10
Keeping Details With Master	12
Repeating Single Detail with Master	14
Short-Circuit Keys	16
End Date Columns	18
Current Indicator Column	20
Hierarchy Level Indicator	22
Denormalization Summary	24
Practice 8—1: Name that Denormalization	25
Practice 8—3: Denormalize Price Lists	29
Practice 8—4: Global Naming	30

Objectives

At the end of this lesson, you should be able to do the following:

- Define denormalization and explain its benefits
- Differentiate and describe the different circumstances where denormalization is appropriate

Why and When to Denormalize

Definition of Denormalization

Denormalization aids the process of systematically adding redundancy to the database to improve performance after other possibilities, such as indexing, have failed. You will read more on indexing in the lesson on Design Considerations.

Denormalization can improve certain types of data access dramatically, but there is no success guaranteed and there is always a cost. The data model becomes less robust, and it will always slow DML down. It complicates processing and introduces the possibility of data integrity problems. It always requires additional programming to maintain the denormalized data.

Denormalization Overview

Denormalization

- **Starts with a “normalized” model**
- **Adds “redundancy” to the design**
- **Reduces the “integrity” of the design**
- **Application code added to compensate**

8-3

Hints for Denormalizing

- Always create a conceptual data model that is completely normalized.
- Consider denormalization as the last option to boost performance.
- Never presume denormalization will be required.
- To meet performance objectives, denormalization should be done during the database design.
- Once performance objectives have been met, do not implement any further denormalization.
- Fully document all denormalization, stating what was done to the tables, what application code was added to compensate for the denormalization, and the reasons for and against doing it.

Denormalization Techniques and Issues

In the next pages you see a number of denormalization techniques that are used regularly. For every type of denormalization you see an indication of when it is appropriate to use it and what the advantages and disadvantages are.

Denormalization Techniques

- **Storing Derivable Values**
- **Pre-joining Tables**
- **Hard-Coded Values**
- **Keeping Details with Master**
- **Repeating Single Detail with Master**
- **Short-Circuit Keys**

8-4

The following topics are covered:

- Storing Derivable Values
- Pre-joining Tables
- Hard-Coded Values
- Keeping Details with Master
- Repeating Single Detail with Master
- Short-Circuit Keys

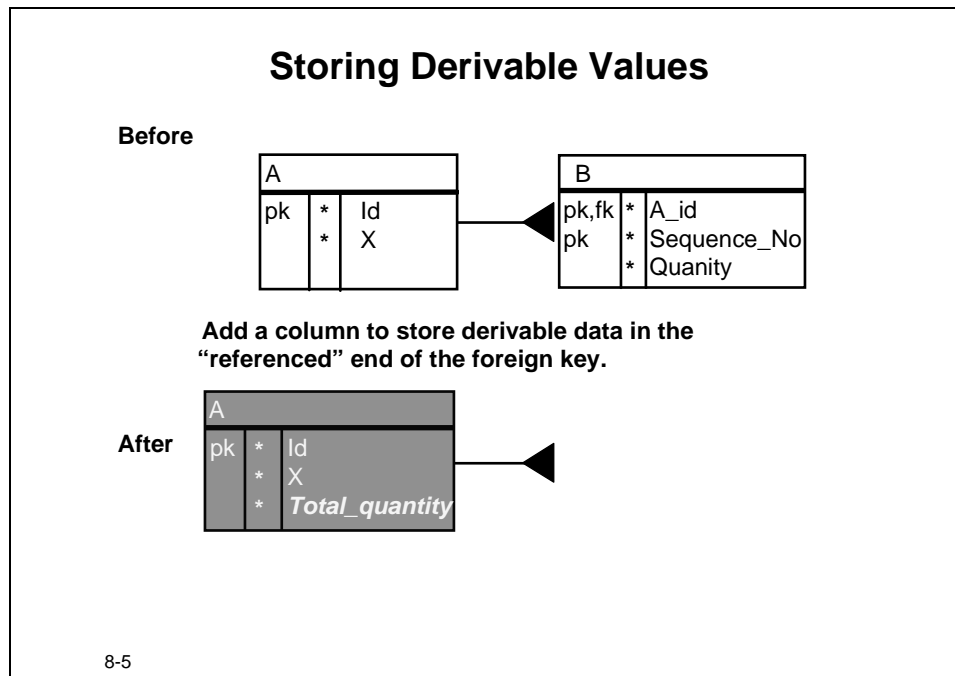
and the most common specific examples:

- Derivable End Date Column
- Derivable Current Indicator column
- Hierarchy Level Indicator

Storing Derivable Values

When a calculation is frequently executed during queries, it can be worthwhile storing the results of the calculation. If the calculation involves detail records, then store the derived calculation in the master table. Make sure to write application code to re-calculate the value, each time that DML is executed against the detail records.

In all situations of storing derivable values, make sure that the denormalized values cannot be directly updated. They should always be recalculated by the system.



Appropriate:

- When the source values are in multiple records or tables
- When derivable values are frequently needed and when the source values are not
- When the source values are infrequently changed

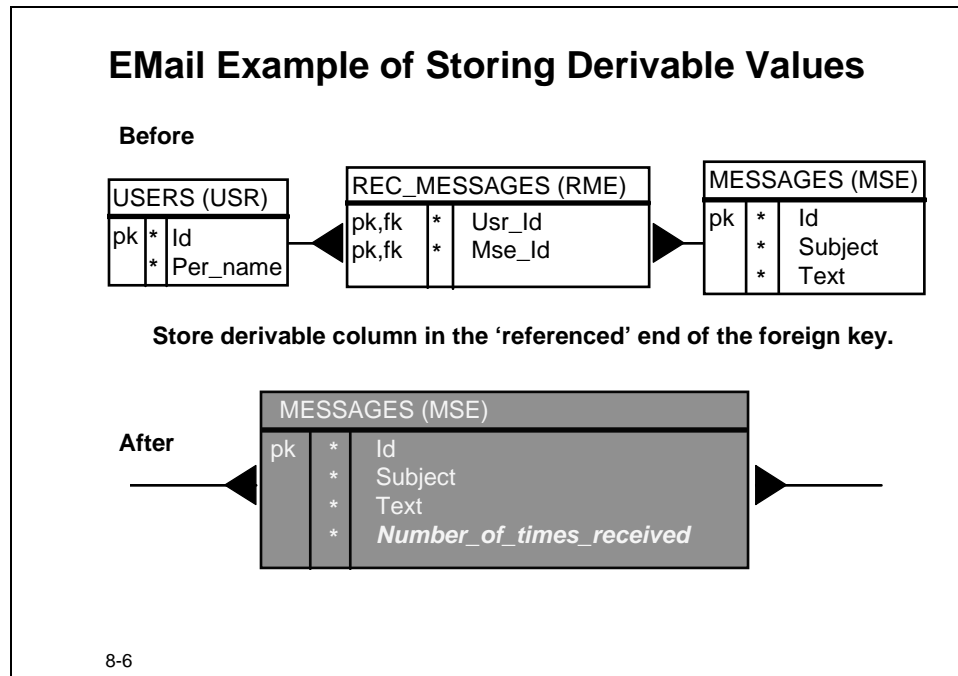
Advantages:

- Source values do not need to be looked up every time the derivable value is required
- The calculation does not need to be performed during a query or report

Disadvantages:

- DML against the source data will require recalculation or adjustment of the derivable data
- Data duplication introduces the possibility of data inconsistencies

E-mail Example of Storing Derivable Values



When a message is delivered to a recipient, the user only receives a pointer to that message, which is recorded in RECEIVED_MESSAGES. The reason for this, of course, is to prevent the mail system from storing a hundred copies of the same message when one message is sent to a hundred recipients.

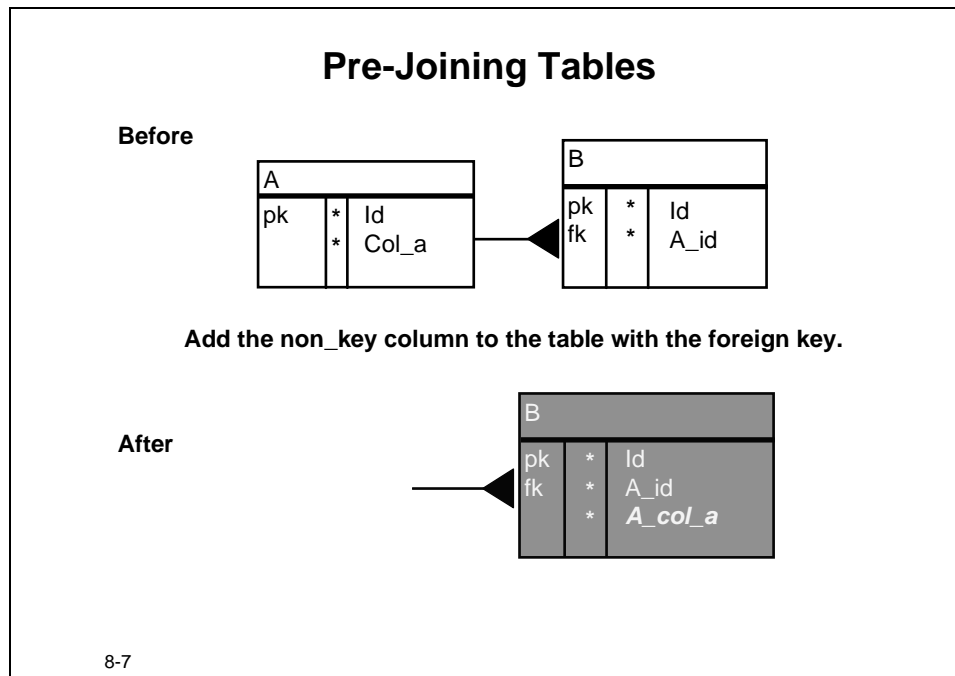
Then, when someone deletes a message from their account, only the entry in the RECEIVED_MESSAGES table is removed. Only after all RECEIVED_MESSAGE entries, for a specific message, have been deleted, the should the actual message be deleted too.

We could consider adding a denormalized column to the MESSAGES table to keep track of the total number of RECEIVED_MESSAGES that are still kept for a particular message. Then each time users delete a row in RECEIVED_MESSAGES, in other words, they delete a pointer to the message, the Number_of_times_received column can be decremented. When the value of the denormalized column equals zero, then we know the message can also be deleted from the MESSAGES table.

Pre-Joining Tables

You can pre-join tables by including a nonkey column in a table, when the actual value of the primary key, and consequentially the foreign key, has no business meaning. By including a nonkey column that has business meaning, you can avoid joining tables, thus speeding up specific queries.

You must include application code that updates the denormalized column, each time the “master” column value changes in the referenced record.



Appropriate:

- When frequent queries against many tables are required
- When slightly stale data is acceptable

Advantages

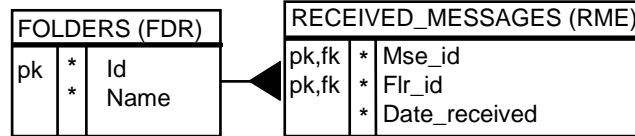
- Time-consuming joins can be avoided
- Updates may be postponed when stale data is acceptable

Disadvantages

- Extra DML needed to update original nondenormalized column
- Extra column and possibly larger indices require more working space and disk space

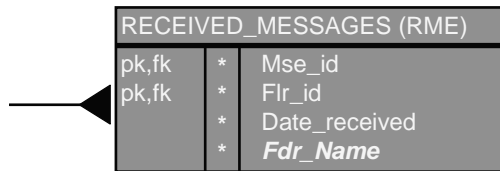
Email Example of Pre-Joining Tables

Before



Create a table with all the frequently queried columns.

After



8-8

Example

Suppose users often need to query RECEIVED_MESSAGES, using the name of the folder where the received message is filed. In this case it saves time when the name of the folder is available in the RECEIVED_MESSAGES table.

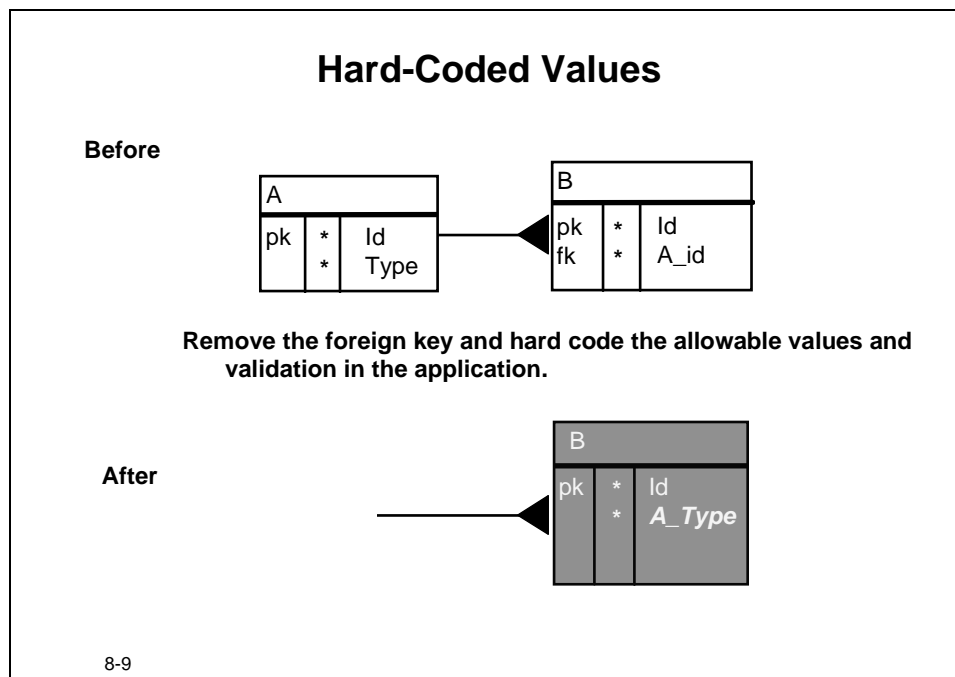
Now, if a user needs to find all messages in a particular folder, only a query on RECEIVED_MESSAGES is needed.

Clearly, the disadvantage is extra storage space for the extra column in a, potentially, very large table.

Hard-Coded Values

If a reference table contains records that remain constant, then you can consider hard-coding those values into the *application* code. This will mean that you will not need to join tables to retrieve the list of reference values. This is a special type of denormalization, when values are kept outside a table in the database. In the example, you should consider creating a check constraint to the B table in the database that will validate values against the allowable reference values. Note that a check constraint, though it resides in the database, is still a form of hardcoding.

Whenever a new value of A is needed the constraint must be rewritten.



Appropriate

- When the set of allowable values can reasonably be considered to be static during the life cycle of the system
- When the set of possible values is small, say, less than 30

Advantages

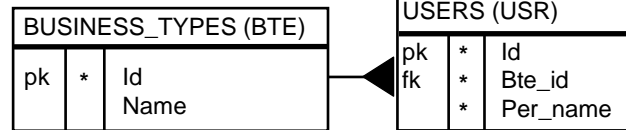
- Avoids implementing a look-up table
- Avoids joins to a look-up table

Disadvantages

- Changing look-up values requires recoding and retesting

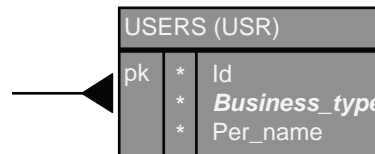
Email Example of Hard-Coded Values

Before



Hard code the allowable values and validation in the application.

After



8-10

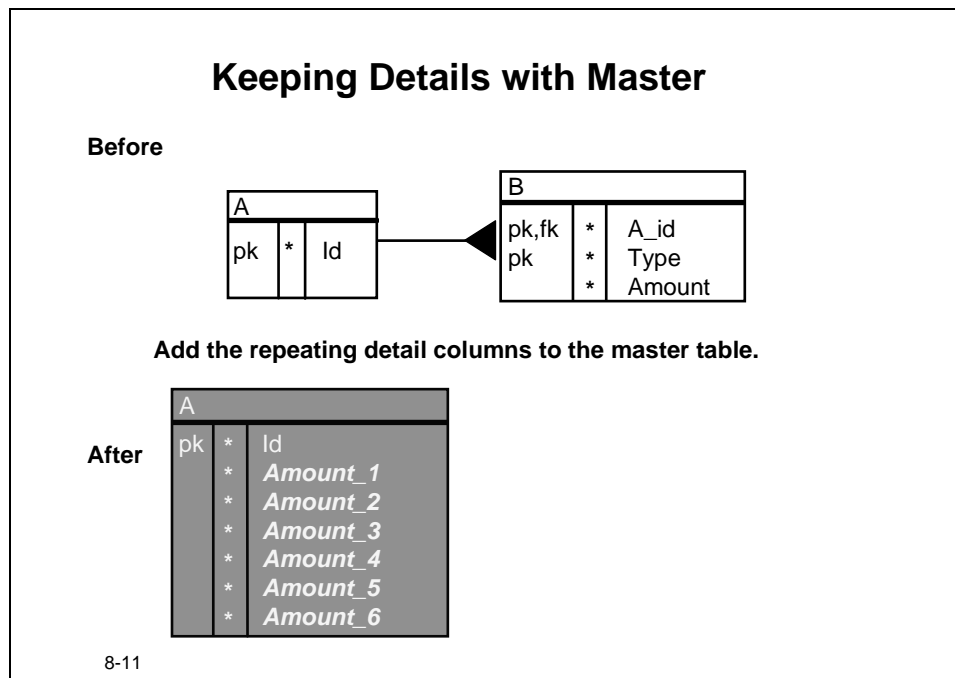
Example

ElectronicMail would like to know some background information about their users, such as the type of business they work in. Therefore EM have created a table to store all the valid BUSINESS_TYPES they want to distinguish. The values in this table are set up front and not likely to change.

This is a candidate for hard-coding the allowable values. You could consider placing a check constraint on the column in the database. In addition to that, or instead of that, you could build the check into the field validation for the screen application where users can sign in to the EM service.

Keeping Details With Master

In a situation where the number of detail records per master is a fixed value (or has a fixed maximum) and where usually all detail records are queried with the master, you may consider adding the detail columns to the master table. This denormalization works best when the number of records in the detail table are small. This way you will reduce the number of joins during queries. An example is a planning system where there is one record per person per day. This could be replaced by one record per person per month, the table containing a column for each day of the month.



Appropriate

- When the number of detail records for all masters is fixed and static
- When the number of detail records multiplied by the number of columns of the detail is small, say less than 30

Advantages

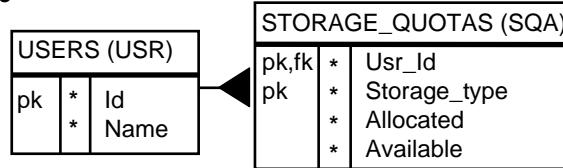
- No joins are required
- Saves space, as keys are not propagated

Disadvantages

- Increases complexity of data manipulation language (DML) and SELECTs across detail values
- Checks for Amount column must be repeated for Amount1, Amount2 and so on
- Table *name* A might no longer match the actual content of the table

Email Example Keeping Detail with Master

Before



Add the repeating detail columns to the master table.

After

USERS (USR)		
pk	*	Id
	*	Name
	*	<i>Message_Quota_Allocated</i>
	*	<i>Message_Quota_Available</i>
	*	<i>File_Quota_Allocated</i>
	*	<i>File_Quota_Available</i>

8-12

Example

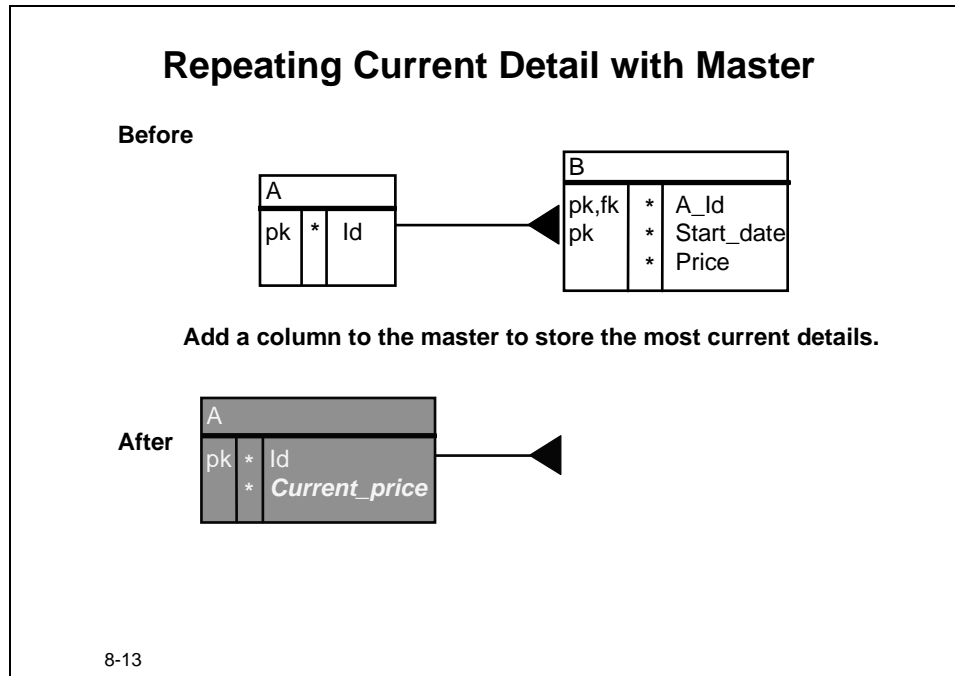
Suppose each e-mail user is assigned two quotas—one for messages and one for files. The amount of each quota is different, so both have to be tracked individually. The quota does not change very frequently. To be relationally pure, we would create a two-record STORAGE_TYPES table and a STORAGE_QUOTAS table with records for each user, one for each quota type. Instead, we can create the following denormalized columns in the USER table:

- Message_Quota_Allocated
- Message_Quota_Available
- File_Quota_Allocated
- File_Quota_Available

Note that the name of table USERS does not really match the data in the denormalized table.

Repeating Single Detail with Master

Often when the storage of historical data is necessary, many queries require only the most current record. You can add a new foreign key column to store this single detail with its master. Make sure you add code to change the denormalized column any time a new record is added to the history table.



Appropriate

- When detail records per master have a property such that one record can be considered “current” and others “historical”
- When queries frequently need this specific single detail, and only occasionally need the other details
- When the Master often has only one single detail record

Advantages

- No join is required for queries that only need the specific single detail

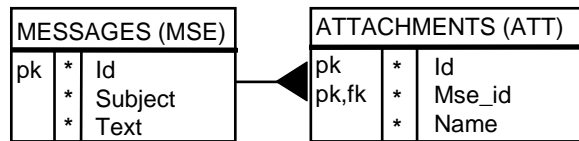
Disadvantages

- Detail value must be repeated, with the possibility of data inconsistencies

Additional code must be written to maintain the duplicated single detail value at the master record.

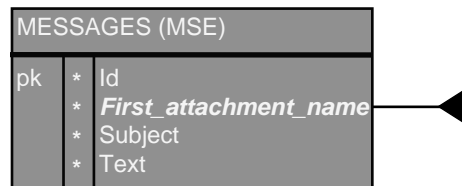
Email Example of Repeating Single Detail with Master

Before



Add a column to the master to store the most current details.

After



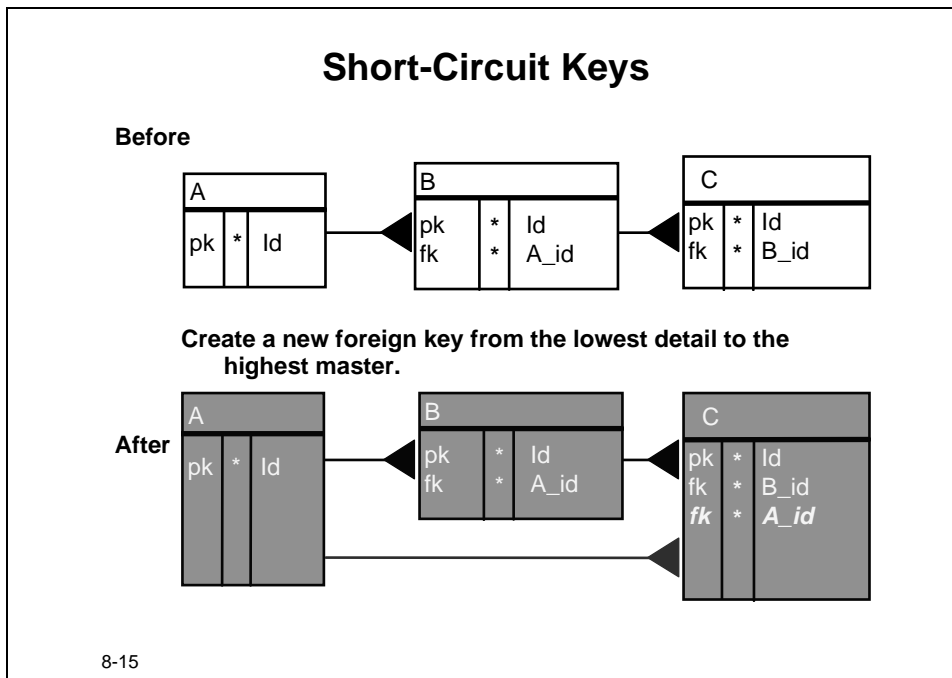
8-14

Example

Any time a message is sent, it can be sent with attachments included. Messages can have more than one attachment. Suppose in the majority of the messages that there is no or only one attachment. To avoid a table join, you could store the attachment name in the MESSAGES table. For those messages containing more than one attachment, only the first attachment would be taken. The remaining attachments would be in the ATTACHMENTS table.

Short-Circuit Keys

For database designs that contain three (or more) levels of master detail, and there is a need to query the lowest and highest level records only, consider creating short-circuit keys. These new foreign key definitions directly link the lowest level detail records to higher level grandparent records. The result can produce fewer table joins when queries execute.



Appropriate

- When queries frequently require values from a grandparent and grandchild, but not from the parent

Advantages

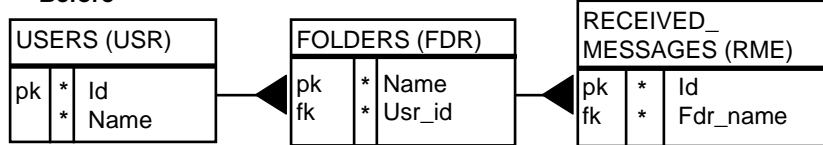
- Queries join fewer tables together

Disadvantages

- Extra foreign keys are required
- Extra code is required to make sure that the value of the denormalized column `A_id` is consistent with the value you would find after a join with table B.

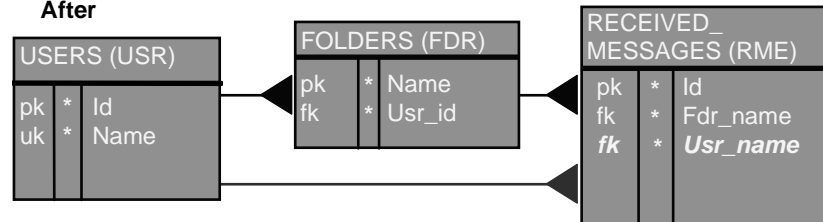
Email Example of Short-Circuit Keys

Before



Create a new foreign key from the lowest detail to the highest master.

After



8-16

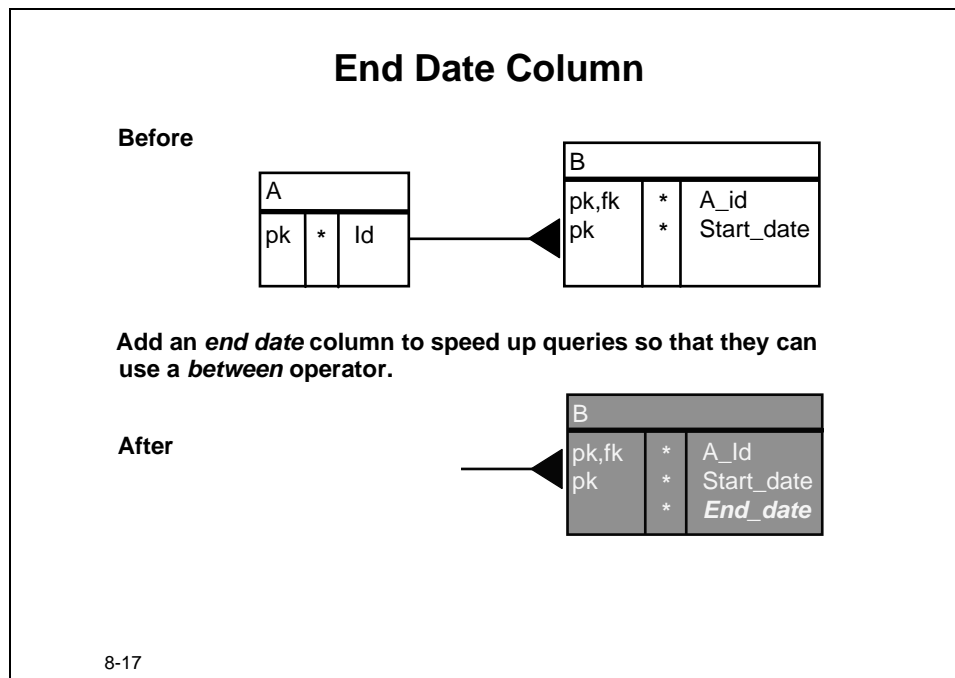
Example

Suppose frequent queries are submitted that require data from the RECEIVED_MESSAGES table and the USERS table, but not from the FOLDERS table. To avoid having to join USERS and FOLDERS, the primary or a unique key of the USERS table can be migrated to the RECEIVED_MESSAGES table, to provide information about USERS and RECEIVED_MESSAGES with one less, or no, table join.

End Date Columns

The most common denormalization decision is to store the end date for periods that are *consecutive*; then the end date for a period can be derived from the start date of the previous period.

If you do this, to find a detail record for a particular date you avoid the need to use a complex subquery.



Appropriate

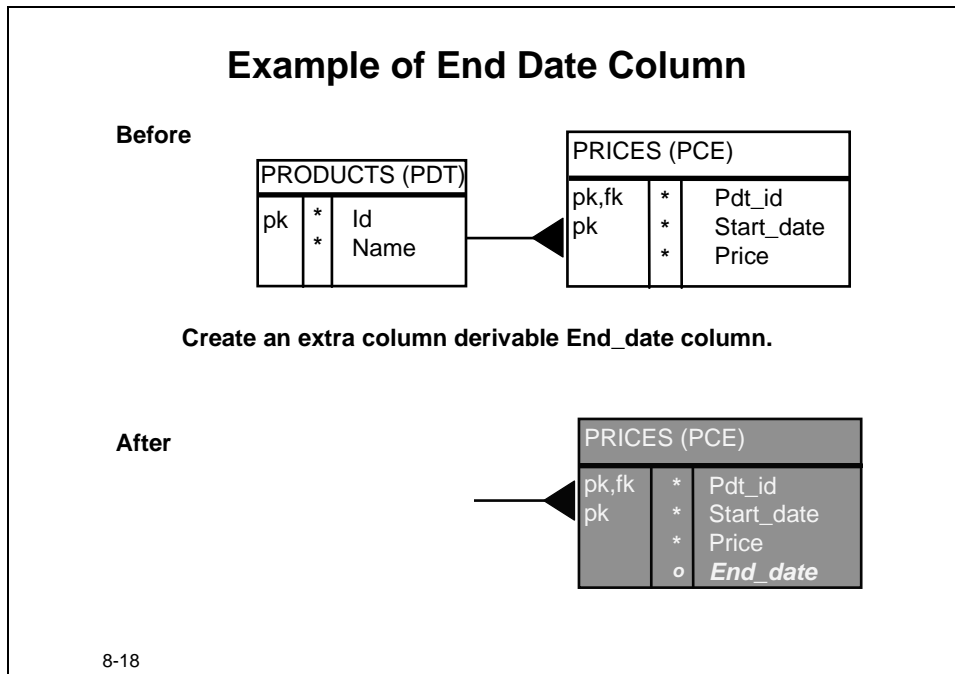
- When queries are needed from tables with long lists or records that are historical and you are interested in the most current record

Advantages

- Can use the between operator for date selection queries instead of potentially time-consuming synchronized subquery

Disadvantages

- Extra code needed to populate the end date column with the value found in the previous start date record



Example

When a business wishes to track the price history of a product, they may use a PRICES table that contains columns for the price and its start date and a foreign key to the PRODUCTS table. To avoid using a subquery when looking for the price on a specific date, you could consider adding an end date column. You should then write some application code to update the end date each time a new price is inserted.

Compare:

```
...WHERE    pdt_id = ...
          AND    start_date = ( SELECT    max(start_date)
                                FROM      prices
                                WHERE     start_date <= sysdate
                                AND       pdt_id = ...
                                )
```

and

```
...WHERE pdt_id = ...
          AND    sysdate between start_date and nvl(end_date, sysdate)
```

Note that the first table structure presupposes that products always have a price since the first price start date of that product. This may very well be desirable but not always the case in many business situations.

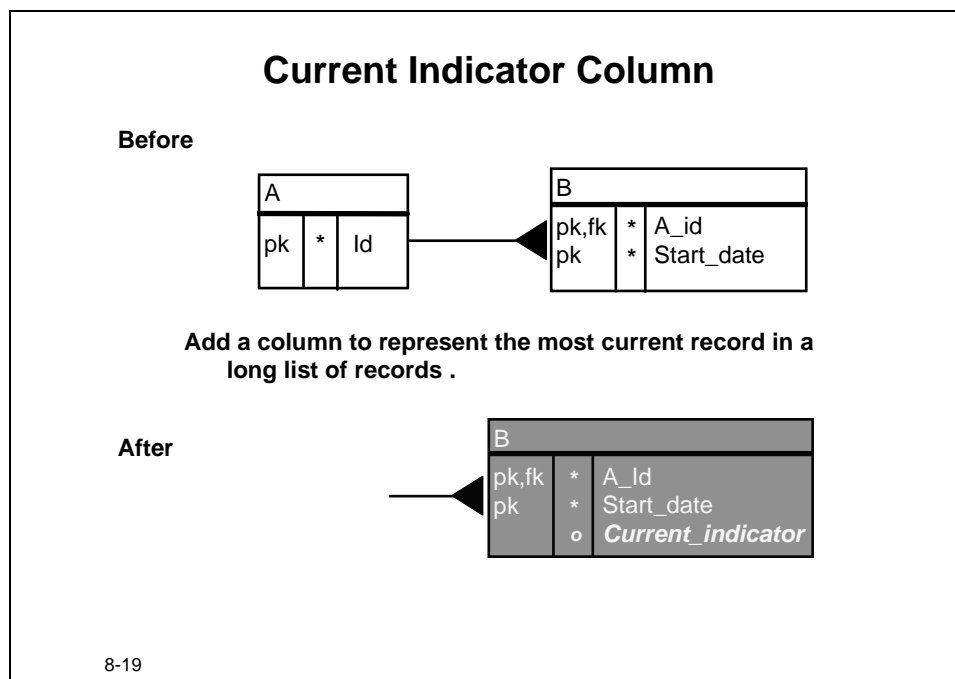
Note also that you would need code to make sure periods do not overlap.

Current Indicator Column

This type of denormalization can be used in similar situations to the end date column technique. It can even be used in addition to an end date. It is a very common type of denormalization.

Suppose most of the queries are to find the most current detail record. With this type of requirement, you could consider adding a new column to the details table to represent the currently active record.

You would need to add code to update that column each time you insert a new record.



Appropriate

- When the situation requires retrieving the most current record from a long list

Advantages

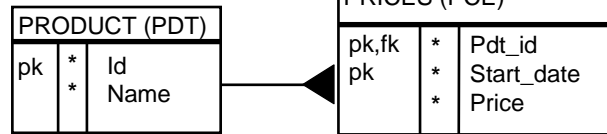
- Less complicated queries or subqueries

Disadvantages

- Extra column and application code to maintain it
- The concept of “current” makes it impossible to make data adjustments ahead of time

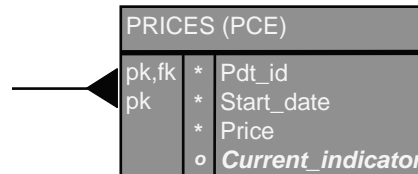
Example of Current Indicator Column

Before



Add a column to represent the most current record, in a long list of records.

After



8-20

Example

In the first table structure, when the current price of a product is needed, you need to query the PRICES table using:

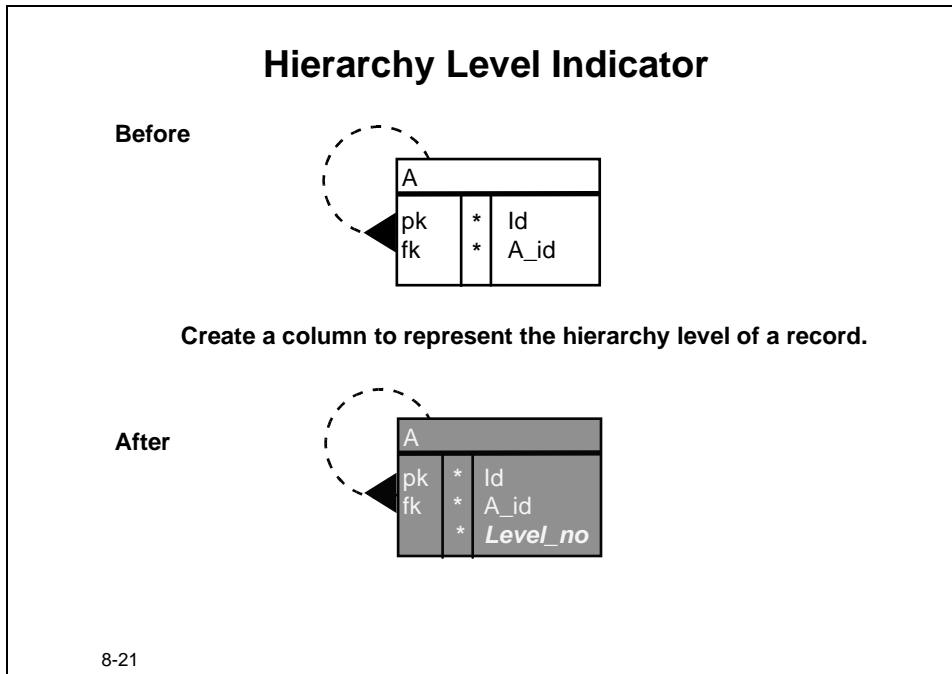
```
...WHERE pdt_id = ...
      AND start_date = ( SELECT max(start_date)
                        FROM   prices
                        WHERE  start_date <= sysdate
                        AND    pdt_id = ...
                        )
```

The query in the second situation would simply be:

```
...WHERE pdt_id = ...
      AND current_indicator = 'Y'
```

Hierarchy Level Indicator

Suppose there is a business limit to the number of levels a particular hierarchy may contain. Or suppose in many situations you need to know records that have the same level in a hierarchy. In both these situations, you will need to use a connect-by clause to traverse the hierarchy. This type of clause can be costly on performance. You could add a column to represent the level of a record in the hierarchy, and then just use that value instead of the connect-by clause in SQL.



Appropriate

- When there are limits to the number of levels within a hierarchy, and you do not want to use a connect-by search to see if the limit has been reached
- When you want to find records located at the same level in the hierarchy
- When the level value is often used for particular business reasons

Advantages

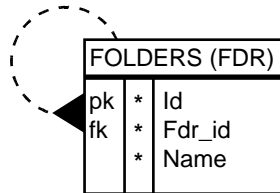
- No need to use the connect-by clause in query code

Disadvantages

- Each time a foreign key is updated, the level indicator needs to be recalculated, and you may need to cascade the changes

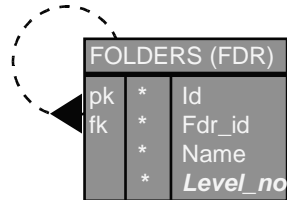
Example of Hierarchy Level Indicator

Before



Create a column to represent the hierarchy level of a record.

After



8-22

Example

Imagine that because of storage limitations, a limit has been placed on the number of nested folders. Each time a user wants to create a new instance of a folder within an existing folder instance, code must decide if that limit has been reached. This can be a slow process.

If you add a column to indicate at what nested level a FOLDER is, then when you create a new folder in it, you can decide immediately if this is allowed. If it is, the level of the new folder is simply one more than the level of the folder it resides in.

Denormalization Summary

Denormalization is a structured process and should not be done lightly. Every denormalization step will require additional application code. Be confident you do want to introduce this redundant data.

Denormalization Summary

Denormalization Techniques

- **Storing Derivable Information**
 - End Date Column
 - Current Indicator
 - Hierarchy Level Indicator
- **Pre-Joining Tables**
- **Hard-Coded Values**
- **Keeping Detail with Master**
- **Repeating Single Detail with Master**
- **Short-Circuit Keys**

8-23

Practice 8—1: Name that Denormalization



Goal

Learn to discriminate the type of denormalization depicted.

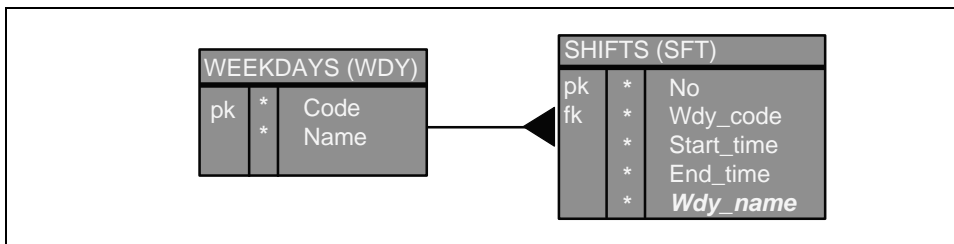
Your Assignment

For the following table diagrams, decide what type of denormalization is used and explain why the diagram depicts the denormalization you have listed.

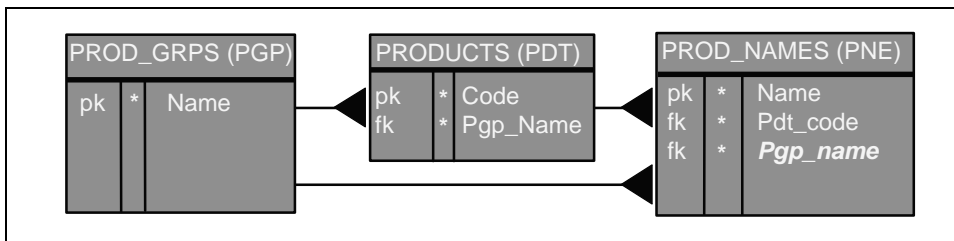
Use one of:

- Storing derivable information
- Pre-Joining Tables
- Hard-Coded Values
- Keeping Details with Master
- Repeating Single Detail with Master
- Short-Circuit Keys

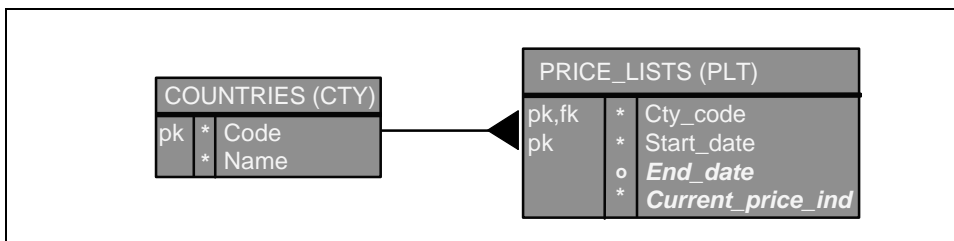
1



2



3



Practice 8—2: Triggers

Goal

The purpose of this practice is to investigate which database triggers are needed to handle a suggested denormalization.

Your Assignment

- 1 Indicate which triggers are needed and what they should do to handle the denormalized column `Order_total` of `ORDER_HEADERS`.

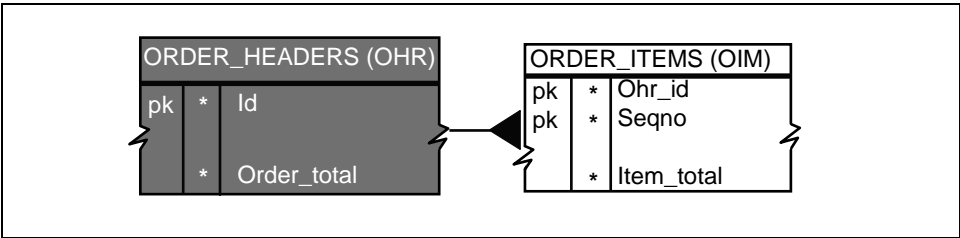


Table	Trg Type	Column	Needed?	What should it do?
OHR	Insert			
	Delete			
	Update	Id		
		Order_total		
OIM	Insert			
	Delete			
	Update	Ohr_id		
		Item_total		

2 Indicate which triggers are needed and what they should do to handle the denormalized column Lcn_address of EMPLOYEES.

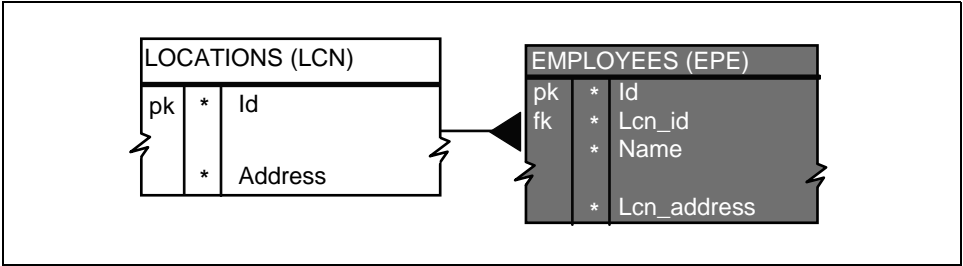


Table	Trg Type	Column	Needed?	What should it do?
LCN	Insert			
	Delete			
	Update	Address		
		<i>other cols</i>		
EPE	Insert			
	Delete			
	Update	Lcn_id		
		Lcn_address		

8-31

- 3 Indicate which triggers are needed and what they should do to handle the denormalized column `Curr_price_ind` of table `PRICES`.

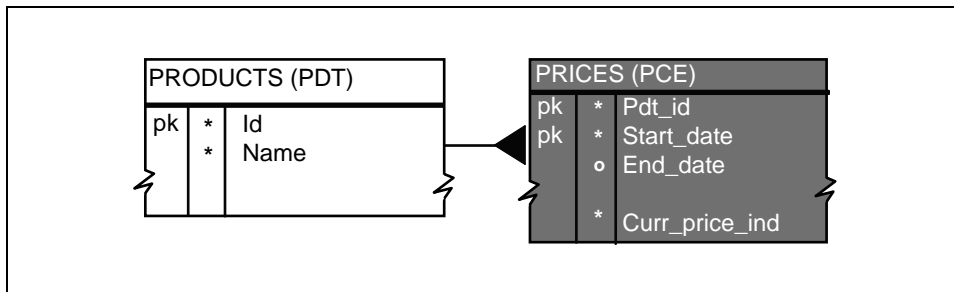


Table	Trg Type	Column	Needed?	What should it do?
PDT	Insert			
	Delete			
PCE	Insert			
	Delete			
	Update	Pdt_id		
		Start_date		
		End_date		
		Curr_price_ind		

8-33

Practice 8—3: Denormalize Price Lists

Moonlight Coffees



Goal

The aim of this practice is to decide on the type of denormalization you could use, and what code is needed to ensure database integrity.

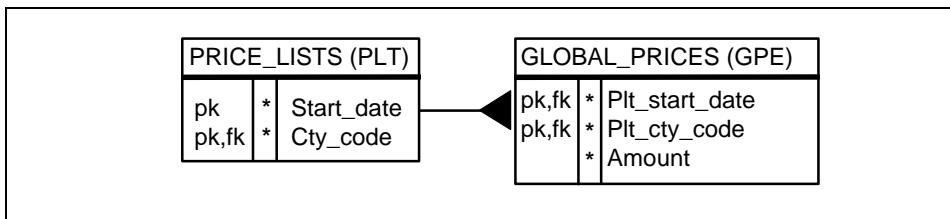
Scenario

End users have started to complain about query performance. One of the areas where this is particularly noticeable is when querying the price of a global product. Since there is a large list of records in the GLOBAL_PRICES table, and it needs to be joined with the PRICE_LISTS table, it is not surprising the queries can take a long time. Optimizing the queries using other techniques have failed to result in acceptable response times. Therefore the decision is to use some denormalization to correct this problem.

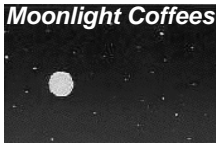
The corporate office also has another concern. They would like to notify the local shops of any new price list changes of global products, prior to their effective date. They would like to enter the new price list information when it is decided, *not* when the start date is reached. You need to add provision to alleviate this restriction.

Your Assignment

Describe what type of denormalization you would implement and what code you would add to ensure the database does not lose any integrity. The next diagram shows the current table schema. Consider both issues described above when deciding which types of denormalization to implement.



Practice 8—4: Global Naming



Goal

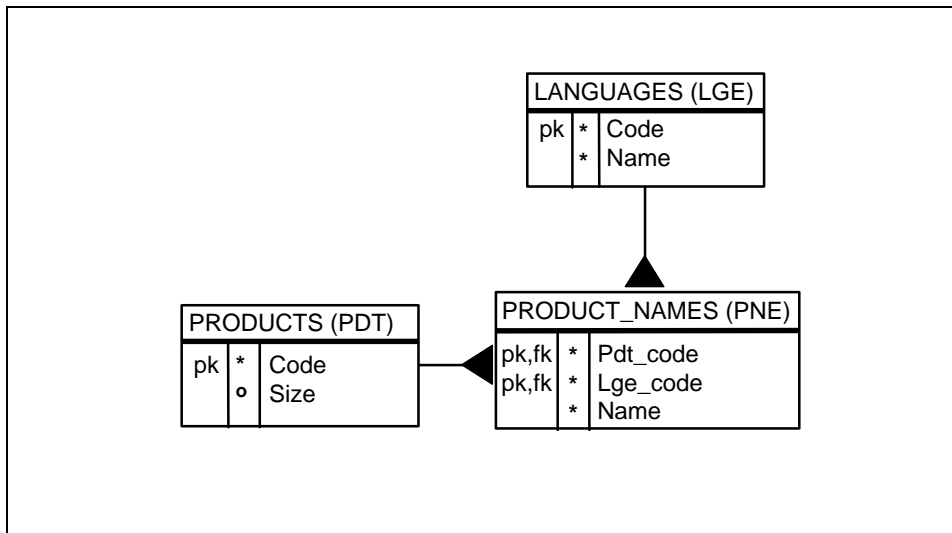
To convert user requirements into denormalized table designs

Scenario

The corporate office has decided to formalize English as the corporate language. Headquarters has asked the IS department to arrange for all global products to store their names in English. On the other hand, countries must be able to store their native language equivalent.

Your Assignment

Using the design below, denormalize the table design and describe the additional code that will allow this requirement to be implemented.



Database Design Considerations

Introduction

Lesson Aim

This lesson illustrates some principles of the Oracle RDBMS and presents the various techniques that can be used to refine the physical design.

Overview

- **Oracle specific Design Considerations**
- **Data Integrity Issues**
- **Performance Considerations**
- **Storage Issues**

9-2

Topic	See Page
Introduction	2
Reconsidering the Database Design	4
Oracle Data Types	5
Most Commonly-Used Oracle Data Types	6
Column Sequence	7
Primary Keys and Unique Keys	8
Artificial Keys	11
Sequences	13
Indexes	16
Choosing Columns to Index	19
When Are Indexes Used?	21

Topic	See Page
Views	23
Use of Views	24
Old-Fashioned Design	25
Distributed Design	27
Benefits of Distributed Design	28
Oracle Database Structure	29
Summary	31
Practice 9—1: Data Types	32
Practice 9—2: Artificial Keys	34
Practice 9—3: Product Pictures	35

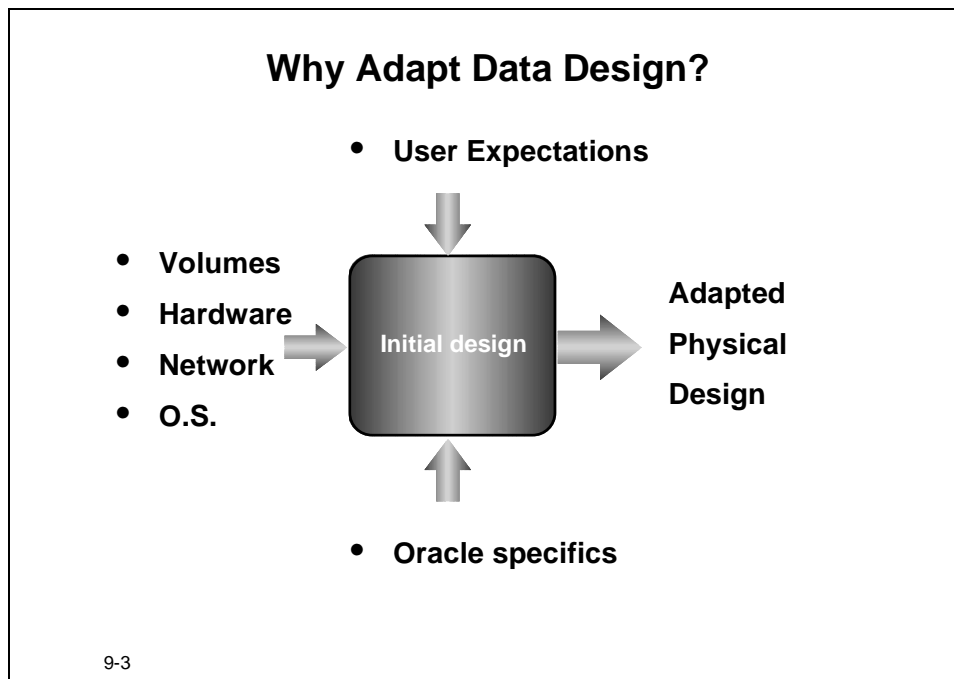
Objectives

At the end of this lesson, you should be able to do the following:

- Describe which data types to use for columns
- Evaluate the quality of the Primary key
- Use artificial keys and sequences where appropriate
- Define rules for referential integrity
- Explain the use of indexes
- Discuss partitioning and views
- Recognize old-fashioned database techniques
- Explain the principle of distributed databases
- Describe the Oracle database model

Reconsidering the Database Design

Each RDBMS has its own internal mechanism. This lesson discusses the major features provided by Oracle to get the best RDBMS performance.



You have to analyze a large number of parameters to obtain a correct adapted physical design from the initial design. Note the “a correct”, not “the correct”. Like many design issues, there is no absolute truth here.

The points noted here are the most important ones—there are others.

- The expected volume of tables, the hardware characteristics like CPU speed, memory size, number of disks and corresponding space, the architecture—client/server or three tier, the network bandwidth, speed, and the operating systems are determinants.
- User requirements are an other big issue. Depending on the response time, the GUI and the frequency of use of modules, they influence the objects that can be used in Oracle to cope with user expectations.
- Depending on the version of Oracle you are using, some elements may or may not exist.

Oracle Data Types

Oracle Data Types

- **Depending on:**
 - **Domains**
 - **Storage issue**
 - **Performance**
 - **Use**
- **Select a data type for columns:**
 - **Character**
 - **Number**
 - **Date**
 - **Large Objects**

9-4

When you create a table or cluster, you must specify an internal data type for each of its columns. These data types define a generic domain of values that each column can contain.

- Some data types have a narrow focus, like number and date. Some data types are general purpose data types, like the various character data types.
- Some data types allow for variable length, some do not.

Choosing a large fixed length for a column to store very few bytes for most of the rows can result in a huge table size. This may affect performance as a row may actually contain only a few bytes and yet be stored on multiple blocks, resulting in a great number of I/O's, and therefore decreasing performance.

- One cannot search against the Large Object Data Types; they cannot be used in a where clause. They are only retrievable by searching against other columns.

Most Commonly-Used Oracle Data Types

- **CHAR(size)** These are fixed-length character data of length-sized bytes. Maximum size is 2000 bytes.
Typical use: for official International Currency Codes which are a fixed three characters in length such as USD, FFR.
- **VARCHAR2(size)** Variable-length character string having maximum length-sized bytes. Maximum size is 4000, and minimum is 1. This is the most commonly-used data type and you should use it if you are not sure which one to use. It replaces the old Oracle version 6 CHAR data type.
Typical use: for storing individual ASCII text *lines* of unlimited length ASCII texts on which you need to be able to search using a wildcard.
- **NUMBER** This data type is used for numerical values, with or without a decimal, of virtually unlimited size. Use this data type for data on which calculation or sorting should be possible. Avoid its use for numbers like a phone number, where the value does not have any meaning.
Typical use: amount of money, quantities, generated unique key values.
- **DATE** Valid date range from January 1, 4712 BC to December 31, 4712 AD. A date data type also contains time components. You should use it only when you know the full date including day, month, and year. The time component is often set to 00:00 (midnight) in normal use of dates.
Typical use: any date where the full date is known.
- **LONG** Character data of variable length up to 2 gigabytes. Obsolete since Oracle8. Was used for ASCII text files where you do not need to search using the wildcard or substring functionality. Use CLOB data type instead.
Typical use: for storing the source code of HTML pages.
- **LONG RAW** Raw binary data of variable length up to 2 gigabytes. Obsolete since Oracle8. Was used for large object types where the database should not try to interpret the data. Use BLOB data type instead.
Typical use: images or video clips.
- **CLOB** Character large object type. Replaces LONG. Major difference: a table can have more than one CLOB column where there was only one LONG allowed. Maximum size is 4 gigabytes.
Typical use: see LONG.
- **BLOB** Character large object type. Replaces LONG RAW. Major difference: a table can have more than one BLOB column where there was only one LONGRAW allowed. Maximum size is 4 gigabytes.
Typical use: see LONG RAW.
- **BFILE** Contains a locator to a large binary file stored outside the database to enable byte stream I/O access to external LOBs residing on the database server.
Typical use: movies

Column Sequence

The sequence of columns in a table is relevant, although any column sequence would allow all table operations. The column sequence can influence, in particular, the performance of data manipulation operations. It may also influence the size of a table.

Suggested Column Sequence

- **Primary key columns**
- **Unique Key columns**
- **Foreign key columns**
- **Mandatory columns**
- **Optional columns**

Large object columns always at the end

9-5

The suggested optimal column sequence is the following:

- 1 Primary key columns
- 2 Unique key columns
- 3 Foreign key columns
- 4 Remaining mandatory columns *
- 5 Remaining optional columns *


* In cases where the table contains a LONG or LONG RAW column, even if it is a mandatory column, make it the *last* column of the table.

The rationale is that null columns should be at the end of the table; columns that are often used in search conditions should be up front. This is for both storage and performance reasons.

Primary Keys and Unique Keys

Primary Keys

```
CREATE TABLE countries
( code      NUMBER(6)      NOT NULL
, name      VARCHAR2(25)   NOT NULL
, currency  NUMBER (10,2)  NOT NULL
);
ALTER TABLE countries
ADD CONSTRAINT cty_pk PRIMARY KEY
(code);
```



Constraint *and* Index name

9-6

Primary Keys

They are a strong concept that is usually enforced for every table.

- They can be made up of one or more columns; each has to be mandatory.
- They are declarative as a constraint and can be named. When creating a primary key constraint, Oracle automatically creates a unique index in association with it.
- A foreign key usually refers to the primary key of a table, but may also refer to a unique key.

Tables that do not have a primary key should have a unique key.

Note: Although Oracle allows a primary key to be updated, relational theory strongly advises against this.

Unique Keys

A unique key is a key that for some reason was not selected to be the primary key. The reasons may have been:

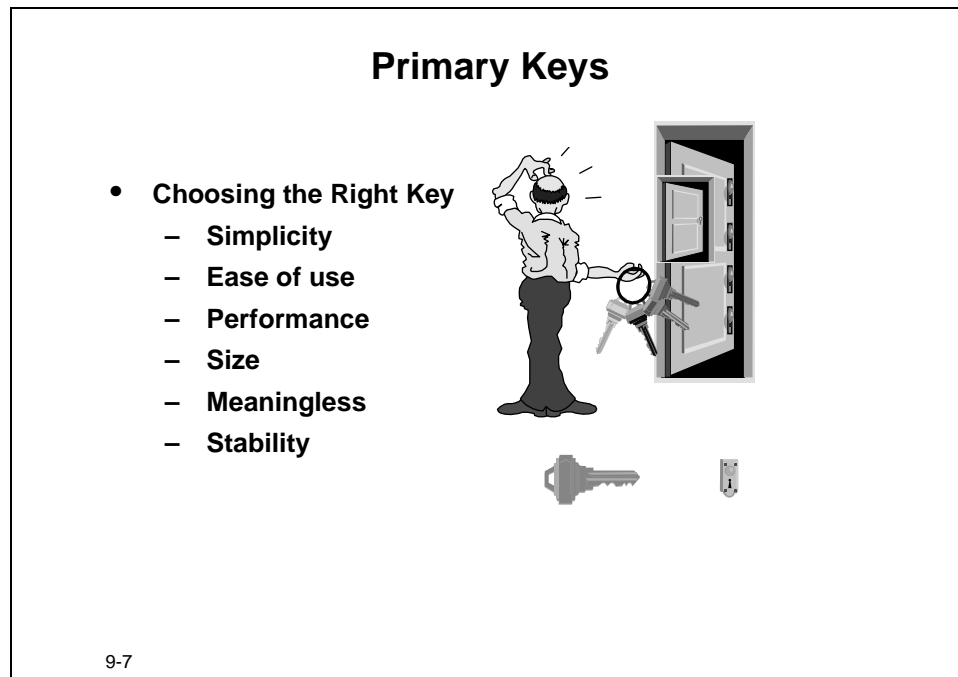
- Allowed nulls. Nulls may be allowed in Unique keys columns.
- Updatable. Unique key values may change but still need to remain unique. For example, the home phone number of an employee or the license plate for a car.

There may be more than one unique key for each table.

Note: A *Unique index* is the additional structure Oracle uses to check the uniqueness of values for primary keys and unique keys. Creating a unique key results automatically in the creation of a unique index.

How to Choose the Primary Key

Following analysis there is a choice of what you want to use for a primary key. It does not have to be seen or known by the user—it can do its work completely in the background.



Desirable Properties for Primary Key

Simple: A primary key should be as simple as possible although Oracle8 allows it to consist of up to 32 columns. Primary key columns can be of various data types. Note that UIDs, as they arise from data analysis, are often composed, not simple. You need to consider replacing such a primary key by a simple key.

Easy to Use: Primary keys are normally used in join statements, so a primary key should be easy to use. Writing a SQL statement to create a join between two tables is easier if two columns only, rather than a large number, are involved in the join predicate.

Does Not Kill Performance: A join operation using a single key usually performs much better than a join using four key columns.

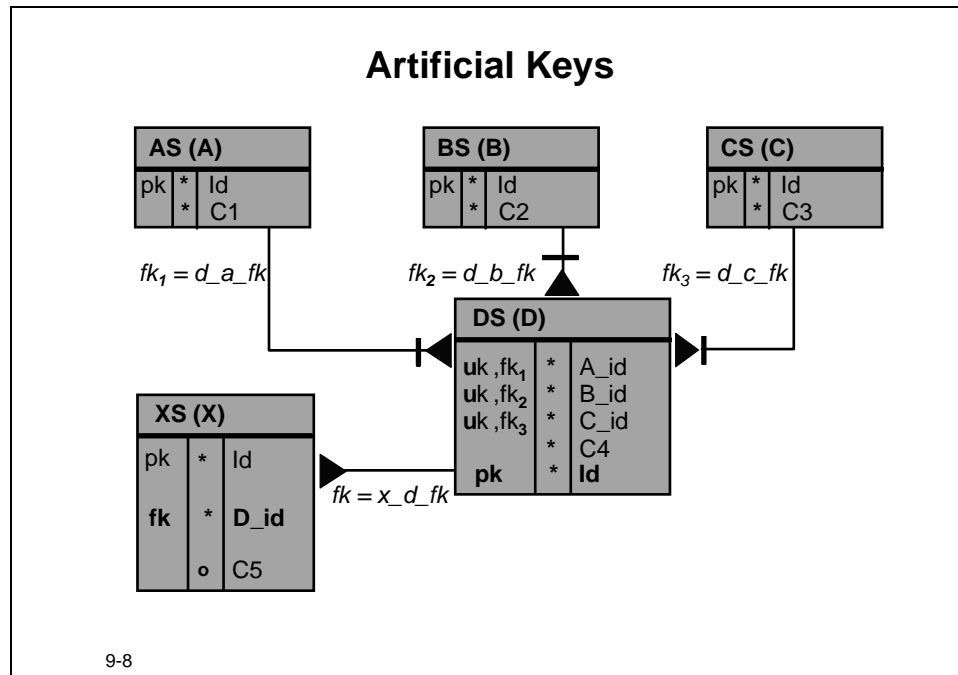
Small Size: Large-sized primary keys lead to large-sized foreign keys referencing them. In general, the referencing table contains far more rows than the referenced table. An oversized primary key can lead to a multiple of unnecessary bytes.

Meaningless: You could, for example, choose to use the name of a country as a primary key, but even recent history has shown that countries may change their names. Opt for numeric values rather than character values, and if using numbers, avoid numbers with any particular meaning.

Stable: You should try to avoid selecting a primary key that is likely to be updated. Bear in mind that it is very rare for real world things to stay stable for ever.

Artificial Keys

An *artificial key* is a meaningless, usually numeric, value that is assigned to a record which functions as the primary key for the table. Artificial keys provide an interesting alternative to complex primary keys. Artificial keys are also called surrogate keys.



Advantages

Artificial keys have the following advantages over composed keys:

- The extra space that is needed for the artificial key column and index is less, often far less, than the space you save for the foreign key columns of referring tables.
- Join conditions consist of a single equation.
- The joins perform better.
- Internal references, which are completely invisible to the user, can be managed. The modeled UID can then be implemented as a unique key, and made updatable without needing cascade updates.
- Because they are meaningless, it is difficult to memorize them. Users will not even attempt this.
- Some people really like them.

Disadvantages

Disadvantages of artificial keys are:

- Because they are meaningless, they always require joins to collect the meaning of the foreign key column.

- More space is required for the indexes, if you decide to create an additional unique key that consists of the original primary key columns.
- Because they are meaningless, it is difficult to memorize them. Users always need a list of values or other help for entering the foreign key values.
- Some people really hate them.

Deciding About Artificial Keys?

Before Design

Negative: It would corrupt your data model, as you would add elements that have no business meaning.

Positive: There is a close mapping between the conceptual and technical model that reduces the chances of misunderstanding.

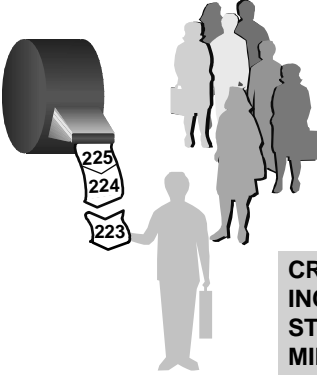
After Design

Positive: It really is a design decision based on current performance considerations.

Tools like Oracle Designer let you decide about artificial keys *during* the initial mapping of the ER model. This is a nice compromise.

Sequences

Sequences



```
CREATE SEQUENCE sequence_name
INCREMENT BY      number
START WITH        number
MINVALUE          number
MAXVALUE          number
CACHE number / NOCACHE
CYCLE | NOCYCLE;
```

9-9

Some Sequence Characteristics

- A sequence is a database object that can generate a serial list of unique numbers for columns of database tables.
- A sequence provides the quickest way of generating unique numbers.
- Sequences simplify application programming by automatically generating unique numerical values that can be used as artificial key values.
- A sequence may be used to generate sequence numbers for any number of tables. Usually a separate sequence is created for each table with an artificial key, although there is no special need for that.
- A sequence guarantees generation of unique ascending or descending numbers. A sequence does not guarantee that all consecutive numbers are actually used.

Foreign Key

By definition, Foreign Keys must refer to primary key or unique key values. You should consider what should happen if the primary key (or unique key) value changes.

Foreign Key Behavior		
	Delete	Update
Restrict	✓	✓
Cascade	✓	
Default / Nullify		

✓ Supported by Oracle through declaration

9-10

Referential Integrity

There are two aspects to consider:

- The rules you want to implement to support business constraints
- The functionalities Oracle provides for these rules

Relational theory describes four possible kinds of behavior for a foreign key. For every foreign key decide what kind of behavior you want it to have.

The behaviors describe what the foreign key should do when the value of the key it refers to changes.

Restrict Delete

Restrict delete means that no deletes of a primary (or unique) key value are allowed when referencing values exist. This is supported by Oracle. This is the most commonly used foreign key behavior.

Restrict Update

Restrict update means that no updates of a primary (or unique) key value are allowed when referencing values exist. This is supported by Oracle. Note that this behavior is unnecessary in the case of artificial keys as these are probably never updated.

Note that restrict update is not the same concept as nontransferability. Restrict update prevents the update of a referenced primary key value. Nontransferability means that the foreign key columns are not updatable.

Cascade Delete

Cascade delete means that deletion of a row causes all rows that reference that row through a foreign key marked as “cascade” will be deleted automatically. Cascade delete is an option that Oracle supports.

The complete delete operation will fail if, during the cascade, there is a record somewhere that cannot be deleted. This may happen if the record to be deleted is referred to through a restrict delete foreign key.

Cascade delete is a very powerful mechanism that should be used with care.

Cascade Update

Cascade update means that after a primary key value is updated, this change is propagated to all the foreign key columns referencing it.

Cascade update and nontransferability often come together.

Default and Nullify

The default and the nullify option mean that on delete or update of the primary key value, the related foreign key values will acquire a default value or will be set to NULL.

These options can be implemented by creating an update database trigger on the table referred to by the foreign key. Clearly, the nullify option is only valid if the foreign key is optional.

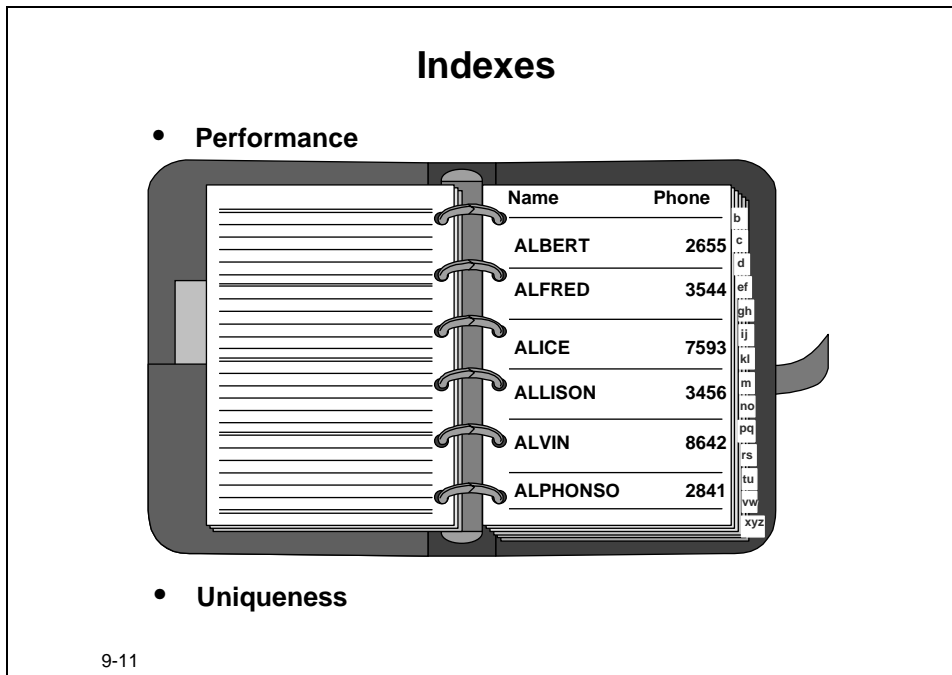
Typical Use

Usually, many foreign keys are defined as restrict delete. This does not prevent the referred record being deleted; it just forces the user to consciously remove or transfer all referring rows.

Of course, when you use artificial keys you can set all foreign key update properties to “restrict” as there will never be a good reason for updating an artificial key value.

Indexes

Indexes are database structures that are stored separately from the tables they depend on. In a relational database you can query any column, independently of the existence of an index on that column.



Indexes are used for two reasons:

- To speed up queries
- To ensure uniqueness if required

Performance

Indexes are created to provide a fast method to retrieve values. However, indexes can slow down performance on DML statements.

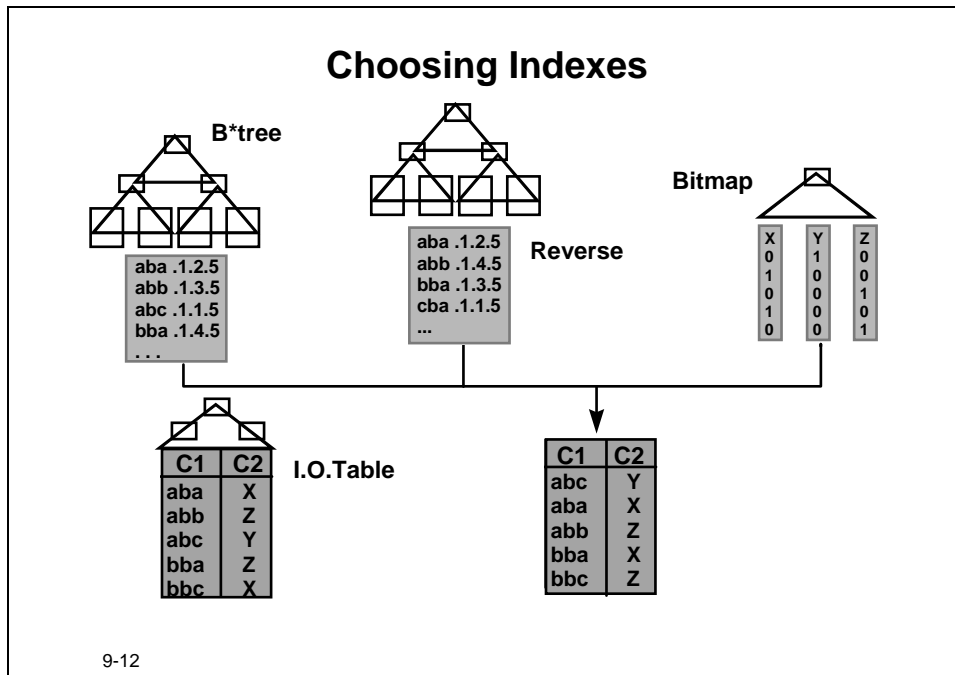
Oracle provides a wide range of index types. You must choose the type which is suitable for its intended use.

Uniqueness

A unique index is an efficient structure to ensure that the values are not duplicated within the set of columns included in the index. Unique indexes are automatically created when you create a primary or unique key. The name of the index in that case is the same as the name of the key constraint.

Index Types

See page 38



B*Tree

The classical structure of an index, if not explicitly specified otherwise, is the B*Tree (also known as **Tree balanced**) index. It is specially designed for online transaction processing systems. They have a proven efficiency and Oracle has offered them for some time. They easily support insert, update, and delete.

Typical use: General purpose

Reverse Key

Based on that classical structure of the B*Tree, Oracle offers a reverse key index which has most of the properties of the B*Tree but in which the bytes of each indexed column are reversed.

Typical use: In an Oracle Parallel Server environment, where such an arrangement can help avoid performance degradation in indexes

Bitmap

A bitmap index stores for each individual value of the indexed column, if a row contains this value or not.

Typical use: Data warehouse environment. Bitmap indexes have a proven efficiency in On Line Analytical Process systems when ad-hoc queries can be intensive and the number of distinct values for the indexed column is not high.

Bitmap indexes require less space than a B*Tree index but they do not support inserts, updates, and deletes as well as a B*Tree.

Index Organized Table

An index organized table is a table that contains rows that are stored in an ordered way, using the B*Tree technique. It provides the speed that indexes provide and does not require a separate index. The only restriction in its use is that you cannot create additional indexes for this Index Organized table.

Typical use: Tables that are always accessed through exactly the same path, in particular when storing large objects.

Concatenated Index

You can create an index that includes more than one column. These are called concatenated indexes. The order in which you specify the columns has a strong impact on the way Oracle can use the index. Set the column that is always in a Where clause as the first column of the index. This is called the leading part of the index.

Function Based Index

Since Oracle8i it is possible to create an index based on a SQL function.

Typical use: Create an index on the first three characters of a name using the *substr* function or the year component of a date using the *to_char* function.

Choosing Columns to Index

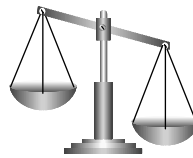
Which Columns to Index?

- Primary key columns and Unique Key columns (Up to Version 6)
- Foreign Key columns
- When significant better performance can be observed in SELECT statements



Avoid indexing:

- Small tables
- Columns frequently updated



9-13

Candidate Columns for Regular B*Tree Indexing

- Columns used in join conditions to improve performance on joins
- Columns that contain a wide range of values
- Columns that are often used in the Where clause of query
- Columns that are often used in an Order By clause of a query

Candidate Columns for Bitmap Indexing

- Columns that have few distinct values such as, for example, a column containing indicator values (Y/N) or a column for gender

Columns Less Suitable for Indexing

- Columns that contain many NULL values where you usually search rows with the NULL values

Columns that Cannot or Should Not be Indexed

- LONG and LONG RAW columns cannot be indexed
- Columns that are hardly ever used in Where / Order By clauses
- Small tables occupying only few data blocks

Temporary Indexes

- Indexes can be created and dropped for a particular incidental use. For example, you can decide to create an index right before a report is run and then drop it afterwards.

General Recommendations

- Limit the number of indexes per table. Although a table can have any number of indexes this does not necessarily improve performance; the more indexes, the more overhead is incurred when there are updates or deletes.
- As a rule of thumb, if there is any doubt, do not create the index. You can always create it later.
- It is very likely that the initial set of indexes will have to change after some time, because of changes of the characteristics of the system. Typically, the number of different values in a column can initially be very low but increase during the life cycle of a system. Initially, an index would not be of value but it would be later.

When Are Indexes Used?

When Can Indexes be Used?

- When referenced in a Where clause or Order By
- When the Where clause does not include some operators
- When the optimizer decides
- With hints in the SQL statement

9-14

You may have created an index to improve performance but without seeing any benefits.

For Oracle to use them, indexed columns need to be referenced in the Where clause of a SQL statement, or in the order by, while the Where clause must not include the following:

- IS NULL
- IS NOT NULL
- !=
- LIKE
- When the column is affected by an operation or function (unless you use a function-based index and the condition uses the same function)

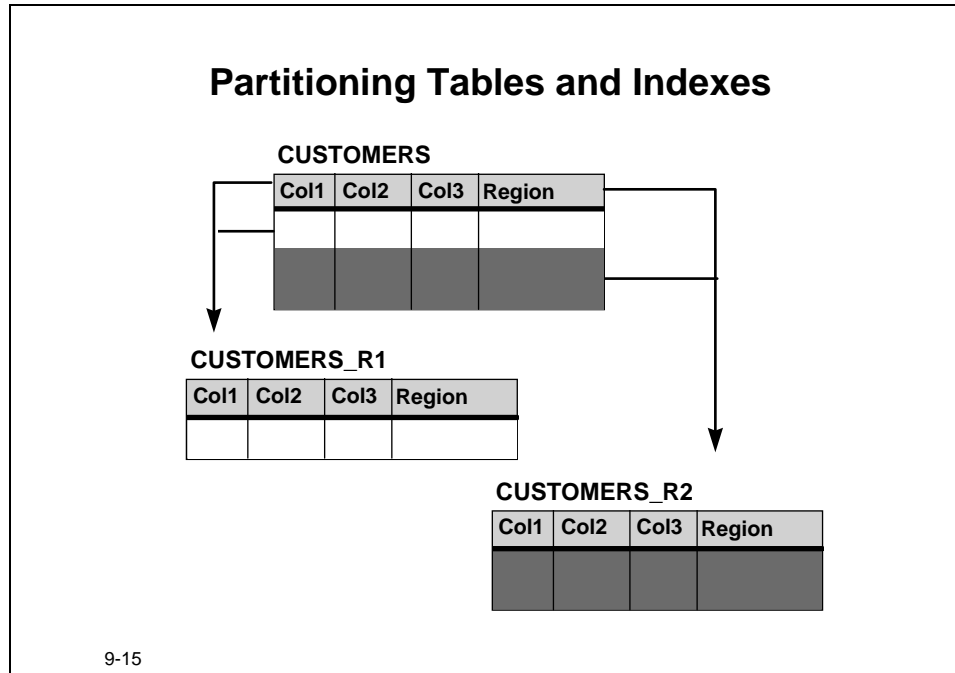
For example, suppose column X contains many nulls and a few numeric, positive values. Suppose queries often select all rows having a NOT NULL value. Finally, suppose an index is created on X.

In this case, the condition WHERE X > 0 is preferable to WHERE X IS NOT NULL because in the first situation Oracle would use an index on X and in the second Oracle would not.

Yet, even if it was written in this way, it is the optimizer's choice to decide whether to use indexes or not. The decision is based on rules or on statistics. You can stimulate the optimizer to use indexes using hints in your SQL statements.

Table Partitioning

Oracle provides an interesting feature to solve performance and administration problems on tables with a large number of rows.



Partitioned Table

Since Oracle8, when creating a table, you can specify the criteria on which you want to divide the table and make a horizontal partitioning. There are then as many partitioned tables as there are distinct values in the column. Each partitioned table has a specific name but access is made referring to the global name of the table. The optimizer then decides which partition to access, depending on the value of the Where clause.

The main issue of this feature is to manipulate considerably smaller pieces of data and then improve the speed of SQL statements. Suppose you want to query on customers located in a specific region, Oracle does not need to access all rows of the **CUSTOMERS** table but can limit its search to the piece holding all customers of this region only.

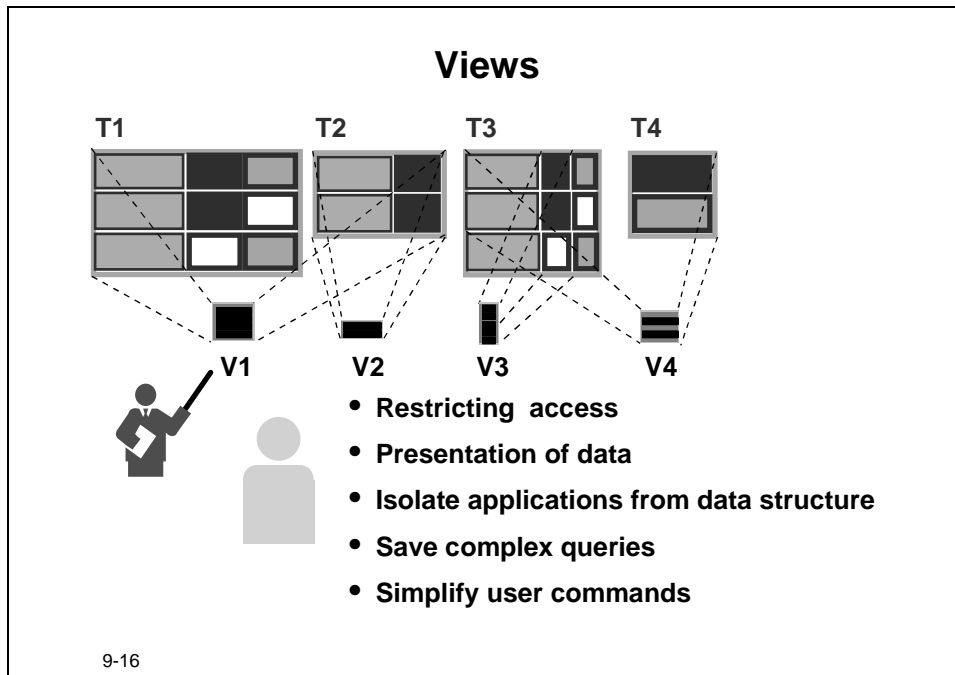
Logically, the table behaves as one object; physically, data is stored in different places.

Partitioned Index

Using the same idea, an index may be partitioned. It does not need to match with the table partitioning. It may have different partitioning criteria and have a different number of partitions to the table. This may be useful in the situation where the answer to particular queries can always be found in the partitioned index.

Views

A view is a window onto the database. It is defined by a `SELECT` statement which is named and stored in the database. Therefore a view has no data of its own—it relays information from underlying tables.



Usages of Views

- Restricting access: The view mechanism is one of the possible ways to hide columns and rows from the tables it is based on.
- Presenting data: A view can be used to present data in a more understandable way to end-users. For example, a view can present calculated data built from elementary information that is stored in tables.
- Isolating application from data structures: Applications may be based on views rather than tables, where there is a high risk that the structure might change. If a view is used, the application would need no maintenance providing the view remains untouched, even though the underlying tables were modified.
- Saving complex queries and simplifying commands: Views can be used to hide the complexity of the data structure, allowing users to create queries over multiple tables without having to know how to join the tables together.
- Simplifying user commands.

Use of Views

Reasons for Views

- **Advantages**
 - **Dynamic views**
 - **Present denormalized data from normalized tables**
 - **Simplify SQL statements**
- **Disadvantages**
 - **May affect performances**
 - **Restricted DML in some cases**

9-17

Advantages

- You can use a view to present derived data to end users without having to store them in the database. Typically, you would show completely denormalized, pre-joined information in views that would allow end users to write simple SELECT statements like `SELECT * FROM ... WHERE ...`.
- Views can be made dynamic, for example, showing data that depend on which user you are or what day it is.

For example, you could create a view that shows localized help messages.

According to the user name, the system can find the preferred language in a PREFERENCES table and next return a message in this language. A single view returns different values depending on the name of the user.

Another example type of view can be used to allow a user to access data between 8:00 am and 6:00 pm on weekdays only.

Disadvantages

- Views are always somewhat slower, which is due to the fact that the parse time is slightly longer. Once a table and its columns are found, the query can be immediately executed. Query criteria are linked with “and” to the criteria of the view. This can affect the execution plan generated by the optimizer.
- Even if views behave almost like tables, there are still some restrictions when using views for insert, update, and delete statements.

Old-Fashioned Design

Going through existing systems, you may find some old-fashioned design techniques. These techniques were used at the time the RDBMS features were not so advanced.

See page 40

Old Fashioned Design

- **Unique index**
- **Views with “Check option” clause**
- **Generic Arc implementation**

9-18

Unique Index

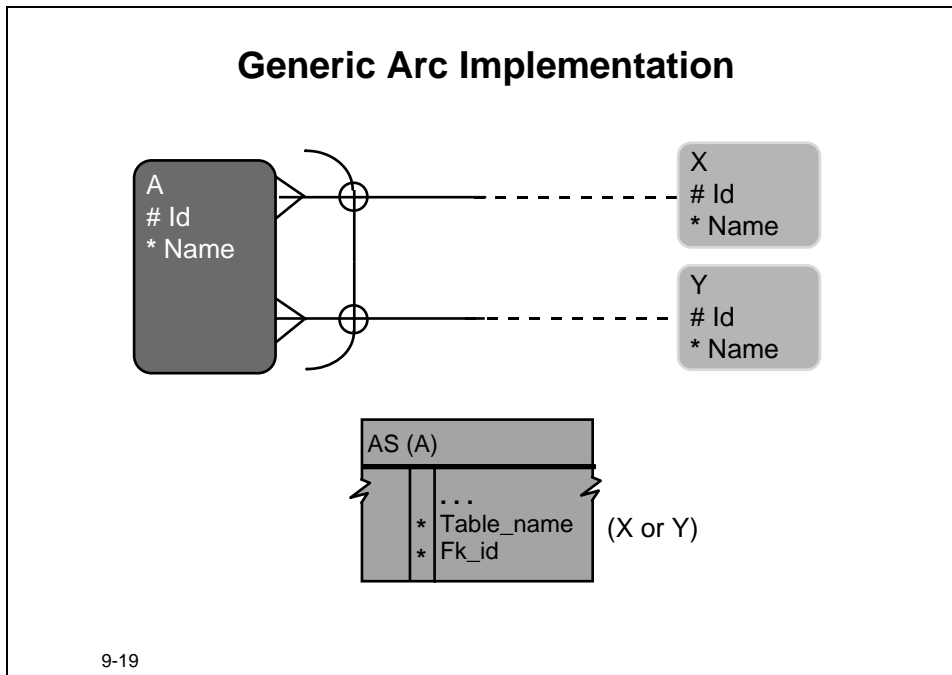
Unique Indexes used to be created manually on the primary key columns because the primary key constraint could not be declared up to Oracle7.

Check Option Views

In earlier versions of Oracle, it was not unusual to create a view “with a check option”. These views, now obsolete, could be used to some extent to enforce data integrity and referential integrity before Oracle7.

There is no functionality in a view with a check option that cannot be coded in a database trigger. The declaration of integrity constraints and coding of database triggers is now the preferred way to handle this.

Generic Arc Implementation



The generic arc implementation is a fossil construction you may find in old systems. In the implementation of the arc of entity **A** in the example, the three relationships in the arc were merged into one generic foreign key column **Fk_id**. Added to table **AS** is a NOT NULL column that keeps the information about which table the foreign key value refers to. This used to be a popular technique because it could make use of a NOT NULL constraint on **Fk_id** when the arc was mandatory.

This solution for implementing arcs should now be avoided for the following limitations:

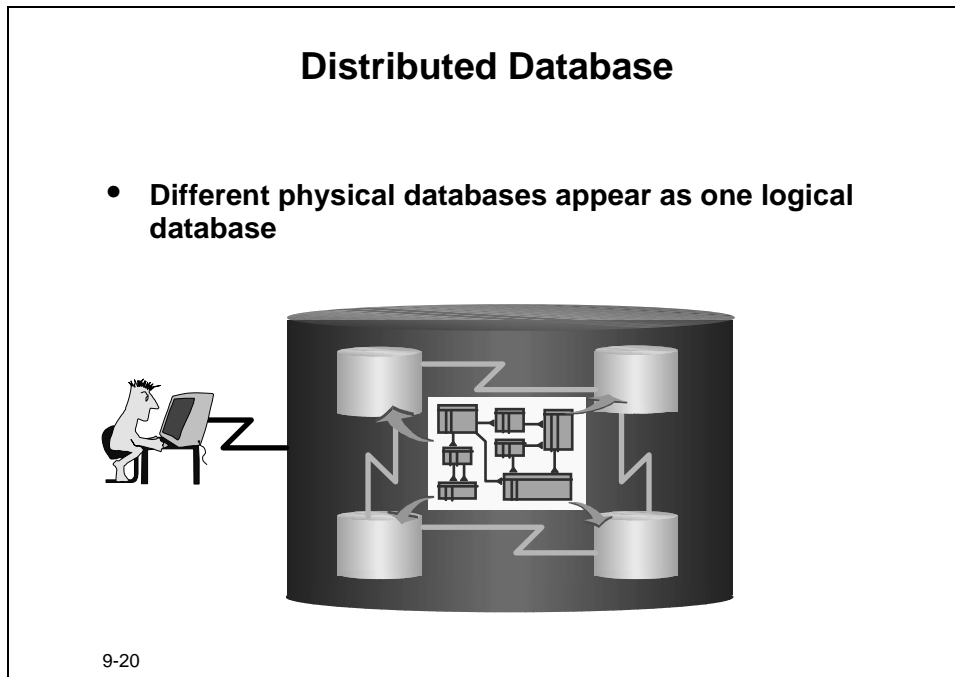
- Since Oracle7 the arc can now be implemented by simply declaring two foreign keys and writing one check constraint.
- The joins may be very inefficient as, in many cases, you would need the time-consuming *union* operator:

```
select  A.Name, X.Name, 'X' Type
from    AS A, XS X
where   ...
union
select  A.Name, Y.Name, 'Y'
from    AS A, YS Y
where   ...
```

- Foreign key constraint for the foreign key column cannot be declared since it cannot reference more than one primary key.

Distributed Design

This is characterized as many physical databases, located at different nodes, but appearing to be a single “logical database”.



Characteristics

- Multiple physical databases
- One logical database view
- Possibly dissimilar processors
- Kernel runs wherever a part of the database exists

The multiple physical databases are not necessarily copies of each other or part of each other.

You can decide on how to spread the individual table content across the different databases on the different partitioning principles. You can decide for a vertical or horizontal technique, or a combination of both.

Benefits of Distributed Design

Benefits of Distributed Databases

- **Resilience**
 - **Reduced line traffic**
 - **Location transparency**
 - **Local autonomy**
 - **Easier growth path**
- but**
- **Increased, distributed, complexity**

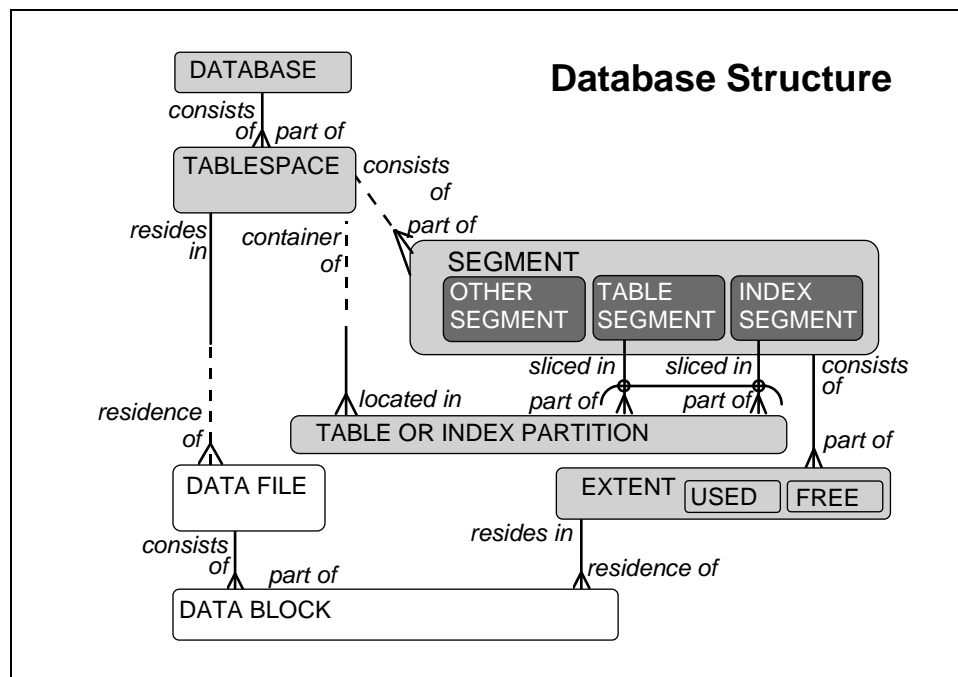
9-21

- Improved flexibility and resilience. Access to data is not dependent on only one machine or link. If there is any failure then some data is still accessible on the local nodes. A failing link can automatically be rerouted via alternative links.
- Improved response time by having the data close to the usual users of the data. This may reduce the line traffic dramatically. For example, in the model of ElectronicMail, it is very likely that each country will have its own database. This database will store in its own messages table the messages that belong to the people registered in that country.
- Location transparency allows the physical data to be moved without the need to change applications or notify users.
- Local autonomy allows each of the physical databases:
 - To be managed independently.
 - To have definitions and access rights created and controlled locally.
- An easier growth path is achieved:
 - More processes can be added to the network
 - More databases can be included on a node.
 - Software update is independent of physical structure.

Disadvantage

A major disadvantage of distributed design is the often very complex configuration: with the data the complexity is also distributed. System maintenance is complicated.

Oracle Database Structure



Tablespaces

The diagram shows the structure of a Oracle database.

An Oracle database consists of one or more tablespaces. Each tablespace can hold a number of segments, and each segment must be wholly contained in its tablespaces. The SYSTEM tablespace is created as part of the database creation, and should be reserved for the Oracle Data Dictionary and related tables only. You should not create application data structures in this tablespace. You are advised to create separate tablespaces for different types of segments.

Segments

A segment is the space occupied by a database object. There are three types of segments: a table segment, an index segment or an other segment, that is used for clusters. Only the other segments must be part of one tablespace.

Partitions

Usually, a segment is assigned to a single tablespace. However, with Oracle8 it is possible to spread a table or index segment into more than one tablespace. This technique is called partitioning. A partition is the part of a table segment (or index segment) that resides in one tablespace.

Extents

Each time more space is needed by a segment, a number of contiguous blocks is allocated as an extent. There is no maximum limit on the number of extents that can be allocated to a segment. It is usually preferable to avoid an excessive number of small extents by ensuring that the segment has a sufficiently large initial extent.

Data Files

Data files are the operating system files that physically contain the database data. Data files consist of data blocks.

Data Blocks

A data block is the smallest amount of data Oracle reads in one read operation. A data block always contains information from one extent only.

There is a distinction between the logical table, made up of rows with columns, and the physical table, taking space that is made up of database blocks organized in extents and located in data files.

Summary

Summary

- **Data Types**
- **Primary, Foreign, and Artificial Keys**
- **Indexes**
- **Partitioning**
- **Views**
- **Distributed design**

9-23

- Oracle provides a large choice of data types for the columns of the tables.
- Primary keys are needed for tables. Artificial keys can be a good solution to implement complex primary keys.
- Indexes improve performance of queries and provide a mechanism for guaranteeing unique values.
- Partitioning tables can also be a solution to performance problems.
- Views are a flexible, secure, and convenient object for users.
- Distributed Design is a complex technique. It allows data to be located closer to the user.

Practice 9—1: Data Types

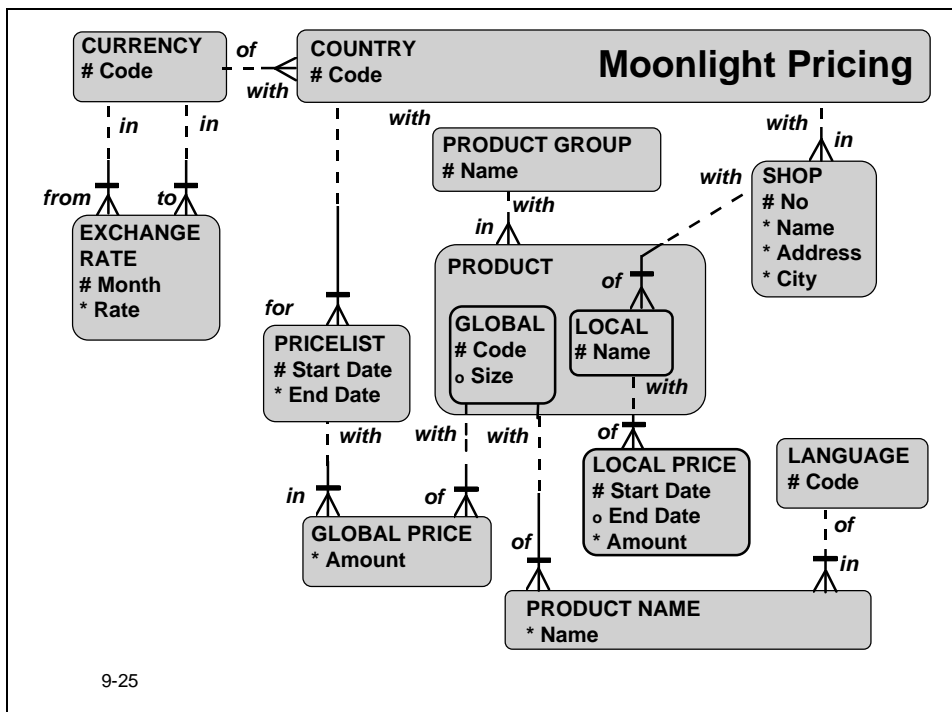


Goal

The purpose of this practice is to perform a quality check on proposed data types.

Scenario

Use the model that illustrates Moonlight pricing.



Your Assignment

- 1 Here you see table names and column names and the suggested data type. Do a quality check on these. If you think it is appropriate, suggest an alternative.

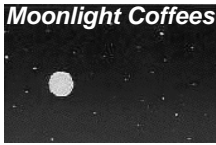
Table	Column	Suggested Data Type	Your Choice Data Type
COUNTRIES	Code	Varchar2(2)	
CURRENCIES	Code	Varchar2(3)	
EXCHANGE_RATES	Month	Date	
	Rate	Number(8,4)	
PRICE_LISTS	Start_date	Date	
	End_date	Date	
PRODUCT_GROUPS	Name	Char(8)	
PRODUCTS	Code	Char(10)	
	Size	Number(4,2)	
	Pdt_type	Number(1)	

- 2 Suggest data types for the following columns. They are all based on previous practices.

Table	Column	Your Choice Data Type
GLOBAL_PRICES	Amount	
LOCAL_PRICES	Start_date	
	End_date	
	Amount	
SHOPS	Name	
	Address	
	City	

- 3 What data type would you use for a column that contains *times* only?

Practice 9—2: Artificial Keys



Goal

You are coming to the end of your contract for Moonlight Coffees. The job is almost finished!

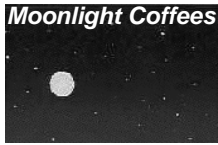
Scenario

You need to make decisions on possible artificial keys for some of the Moonlight tables. The model is the same as the one used in the previous practice.

Your Assignment

- 1 Indicate for each table if you see benefits of creating an artificial key and why.
COUNTRIES
GLOBAL_PRICES
PRICE_LISTS
- 2 For which tables (if any) based on the Moonlight model does it not make any sense at all to create artificial keys?

Practice 9—3: Product Pictures



Goal

The purpose of this practice is to modify a design to serve new requirements.

Scenario

This is your last task for Moonlight coffees. Tomorrow you are free to forget all about Moonlight and only drink coffee!

The decision has been made to make the first steps into the e-commerce market. One objective is to allow customers to consult Moonlight's website. This site should provide product information. For each product at least two additional attributes have been identified.

The first is the attribute Picture for images of the products. The second is an attribute HTML Document that holds the product description that can be displayed with a browser. Other attributes may follow.

Your Assignment

- 1 Decide what data type you would advise to be used for each column.
- 2 You have heard that an old Oracle version would not accept more than one long type column per table. You are not sure if this is still a limitation. Advise about the implementation.

B

Normalization

Introduction

Lesson aim

This lesson describes the steps involved in order to normalize table data to the third normal form for cases when there is no possibility of performing a full data analysis.

Overview

- **Table Normalization**
- **Normal Forms of Tables**

B-2

Topic	See Page
Introduction	2
Normalization and its Benefits	3
First Normal Form	7
Second Normal Form	9
Third Normal Form	11
Summary	13

Objectives

At the end of this lesson, you should be able to do the following:

- Define normalization and explain its benefits
- Place tables in Third Normal Form

Normalization and its Benefits

Why and When to Normalize Tables

Before we even talk about why you should normalize, first consider when you should normalize. If you are developing an application and use the techniques of entity relationship (ER) modeling, then you will not need to normalize. One of the advantages of entity relationship modeling is that the resulting table design is already normalized, provided there are no obvious errors in the ER model.

The only time you will need to normalize the data is if there has been no time to build an entity model and when a set of tables is already available. You can then employ the normalization techniques following the initial database design as a *last chance* to check for existing database integrity.

History of Normalization

Normalization is a technique established by the originator of the relational model, E.F. Codd. The complete set of normalization techniques, include twelve rules that databases need to follow in order to be described as truly normalized. It is a technique that was created in support of relational theory, years before entity relationship modeling was developed. The entity relationship modeling process has incorporated many of the normalization techniques to produce a normalized entity relationship diagram.

Two terms that have their origins in the normalization technique are still widely in use. One is ***normalized data***, the other is ***denormalization***.

Objective of Normalization

The major objective of normalization is to remove redundant data from an existing set of tables or table definitions, thereby increasing the integrity of the database design and to maximize flexibility of data storage. Removing redundant data helps to eliminate update anomalies. The first three normal forms progress in a systematic manner to achieve this objective.

There are many other normal forms in addition to the first three, and they deal with more subtle anomalies. In general, the IT industry considers normalization to the Third form an acceptable level to remove redundancy. With a few exceptions, higher normalization levels are not widely used.

The major subject of normalization is tables, not entities.

Why Normalize?

- **An Entity Model is not always available as a starting point for design**
- **To reduce redundant data in existing design**
- **To increase integrity of data, and stability of design**
- **To identify missing tables, columns and constraints**

Note: Third normal form is the generally-accepted goal for a database design that eliminates redundancy.

B-3

Normalization Compared to Normalized Data

Normalized data is data that contains no redundancies. This is important as data redundancy may cause integrity problems. Normalization is the activity, the process, that leads to a normalized data structure as does entity relationship modeling.

Benefits of Normalized Data

The major benefits of a correctly normalized database from an Information Systems perspective include:

- Refinement of the strategy for constructing tables and selecting keys.
- Improved communication with the end-users' application activities.
- Reduced problems associated with inserting and deleting data.
- Reduced enhancement and modification time associated with changing the data structure.
- Improved information for decisions relating to the physical database design.
- Identification of potential problems that may have been overlooked during analysis.

Recognize Unnormalized Data

USER _ID	USER _NAME	MSE _ID	REC DATE	SUBJECT	TEXT	SRVR _ID	SERVER _NAME
2301	Smith	54101	05/07	Meeting Today	There is..	3786	IMAP05
2301	Smith	54098	07/12	Promotions	I like to.	3786	IMAP05
2301	Smith	54445	10/06	Next Assignment	Your next.	3786	IMAP05
5607	Jones	54101	05/07	Meeting Today	There is..	6001	IMAP08
5607	Jones	54512	06/07	Lunch?	Can you...	6001	IMAP08
5607	Jones	54660	12/01	Jogging Today?	Can you...	6001	IMAP08
7773	Walsh	54101	05/07	Meeting Today	There is..	9988	EMEA01
7773	Walsh	54554	03/17	Stock Quote	The latest	9988	EMEA01
0022	Patel	54101	05/07	Meeting Today	There is..	2201	EMEA09
0022	Patel	54512	06/07	Lunch?	Can we ...	2201	EMEA09

B-4

Unnormalized Data

Data that has not been “normalized” is considered to be “unnormalized” data or data in *zero-normal* form. This data is not to be confused with data that is *denormalized*. If no ER Model was created at the start of a database design project, you are likely to have unnormalized data, not denormalized data. If you want to add redundancy, for faster performance or other reasons, you follow the rules defined during the process of denormalization. But, to denormalize data you must start with normalized data. You cannot denormalize an unnormalized design, just as you cannot de-ice your car, if there is no ice on it.

Normalization

Normalization consists of a series of rules that must be applied to move from a supposedly unnormalized set of data to a normalized structure. The process is described in various steps which lead to a “higher” level of normalization. These levels are called normal forms.

Normalization Rules	
Normal Form Rule	Description
First Normal Form (1NF)	The table must express a set of unordered, two-dimensional tables. The table cannot contain repeating groups.
Second Normal Form (2NF)	The table must be in 1NF. Every non-key column must be dependent on all parts of the primary key.
Third Normal Form (3NF)	The table must be in 2NF. No non-key column may be functionally dependent on another non-key column.
“Each non-primary key value MUST be dependent on the key, the whole key, and nothing but the key.”	
B-5	

First Normal Form

Definition of First Normal Form (1NF)

The table must express a set of unordered, two-dimensional table structures. A table is considered in the first normal form if it contains no repeating groups.

Steps to Remove Repeating Groups

- 1 Remove the repeating columns from the original unnormalized table.
- 2 Create a new table with the primary key of the base table and the repeating columns.
- 3 Add another appropriate column to the primary key, which ensures uniqueness.
- 4 Create a foreign key in the new table to link back to the original unnormalized table.

Converting to First Normal Form

USER _ID	USER _NAME	MSE _ID	REC_ DATE	SUBJECT	TEXT	SRVR _ID	SERVER _NAME
2301	Smith	54101	05/07	Meeting Today	There is..	3786	IMAP05
2301	Smith	54098	07/12	Promotions	I like to.	3786	IMAP05
2301	Smith	54445	10/06	Next Assignment	Your next.	3786	IMAP05
5607	Jones	54512	06/07	Lunch?	Can you...	6001	IMAP08
5607	Jones	54101	05/07	Meeting Today	There is..	6001	IMAP08
5607	Jones	54660	12/01	Jogging Today?	Can you...	6001	IMAP08
7773	Walsh	54101	05/07	Meeting Today	There is..	9988	EMEA01
7773	Walsh	54554	03/17	Stock Quote	The latest	9988	EMEA01
0022	Patel	54101	05/07	Meeting Today	There is..	9988	EMEA01
0022	Patel	54512	06/07	Lunch?	Can we ...	9988	EMEA01

1. Remove repeating group from the base table.
2. Create a new table with the PK of the base table and the repeating group.

B-6

First Normal Form—Single Record

USERS

USER _ID	USER _NAME	MSE _ID	REC DATE	SUBJECT	TEXT	SRVR _ID	SERVER _NAME
2301	Smith	54101	05/07	Meeting Today	There is..	3786	IMAP05
5607	Jones	54512	06/07	Lunch?	Can you...	6001	IMAP08
7773	Walsh	54101	05/07	Meeting Today	There is..	9988	EMEA01
0022	Patel	54101	05/07	Meeting Today	There is..	9988	EMEA01



USERS

USER _ID	USER _NAME	SRVR _ID	SERVER _NAME
2301	Smith	3786	IMAP05
5607	Jones	6001	IMAP08
7773	Walsh	9988	EMEA01
0022	Patel	9988	EMEA01

B-7

First create a second table to contain the repeating group columns. Then create a primary key composed of the primary key from the unnormalized table and another column that is unique. Finally create a foreign key to link back to the first table.

First Normal Form—Repeating Groups

RECEIVED MESSAGES (1NF)

USER	MSE	REC			
_ID	_ID	DATE	SUBJECT		TEXT
-----	-----	-----	-----	-----	-----
2301	54101	05/07	Meeting Today		There is..
2301	54098	07/12	Promotions		I like to.
2301	54445	10/06	Next Assignment		Your next.
5607	54101	05/07	Meeting Today		There is..
5607	54512	06/07	Lunch?		Can you...
5607	54660	12/01	Jogging Today?		Can you...
7773	54101	05/07	Meeting Today		There is..
SRVR	SERVER	/17	Stock Quote		The latest
_ID	_NAME	/07	Meeting Today		There is..
		/07	Lunch?		Can we ...

USER _ID	USER _NAME	SRVR _ID	SERVER _NAME
2301	Smith	3786	IMAP05
5607	Jones	6001	IMAP08
7773	Walsh	9988	EMEA01
0022	Patel	9988	EMEA01

USERS (1NF)

B-8

Second Normal Form

Definition of Second Normal Form (2NF)

A table is in second normal form if the table is in the first normal form and every non-primary key column is functionally dependent upon the entire primary key. No non-primary key column can be functionally dependent on part of the primary key.

Depends on is defined as: a column B depends on column A means that B must be re-evaluated whenever A changes.

A table in the first normal form will be in second normal form if any one of the following apply:

- The primary key is composed of only one column.
- No nonkeyed columns exist in the table.
- Every nonkeyed attribute is dependent on all of the columns contained in the primary key.

Converting to Second Normal Form

1. **Determine which non-key columns are not dependent upon the table's entire primary key.**
2. **Remove those columns from the base table.**
3. **Create a second table with those columns and the columns from the *PK* that they are dependent upon.**

B-9

Steps to Remove Partial Dependencies

- 1 Determine which nonkey columns are dependent upon the table's entire primary key.
- 2 Remove those columns from the base table. Create a second table with those nonkeyed columns and a copy of the columns from the primary key that they are dependent upon.
- 3 Create a foreign key from the original base table to the new table, linking to the new primary key.

Tables Already in Second Normal Form

USERS

USER _ID	USER _NAME	SRVR _ID	SERVER _NAME
2301	Smith	3786	IMAP05
5607	Jones	6001	IMAP08
7773	Walsh	9988	EMEA01
0022	Patel	9988	EMEA01

Is the USERS table already in 2NF?

B-10

Convert to Second Normal Form

RECEIVED_ MESSAGES (1NF)	USER _ID	MSE _ID	REC DATE	SUBJECT	TEXT
	2301	54101	05/07	Meeting Today	There is.
	2301	54098	07/12	Promotions	I like to
	2301	54445	10/06	Next Assignmen	Your next
	5607	54101	05/07	Meeting Today	There is.
	5607	54512	06/07	Lunch?	Can you..
	7773	54660	12/01	Jogging Today?	Can you..
	7773	54101	05/07	Meeting Today	There is.
	7773	54554	03/17	Stock Quote	The lates
	0022	54101	05/07	Meeting Today	There is.
	0022	54512	06/07	Lunch?	Can we ..

RECEIVED_ MESSAGES (2NF)	USER _ID	MSE _ID	REC DATE
	2301	54101	05/07
	2301	54098	07/12
	2301	54445	10/06
	5607	54101	05/07
	5607	54512	06/07
	5607	54660	12/01
	7773	54101	05/07
	7773	54554	03/17
	0022	54101	05/07
	0022	54512	06/07

MESSAGES (2NF)	MSE _ID	SUBJECT	TEXT
	54101	Meeting Toda	There is.
	54098	Promotions	I like to
	54445	Next Assignm	Your next
	54512	Lunch?	Can you..
	54660	Jogging Toda	Can you..
	54554	Stock Quote	The lates

B-11

Third Normal Form

Definition of Third Normal Form (3NF)

A table is in third normal form if every nonkeyed column is directly dependent on the primary key, and not dependent on another nonkeyed column. If the table is in second normal form and all of the “transitive dependencies” are removed, then every non-keyed column is said to be “dependent upon the key, the whole key, and nothing but the key”.

Converting to Third Normal Form

Remove any columns that are dependent upon another non-key column:

- 1. Determine which columns are dependent upon another non-key column.**
- 2. Remove those columns from the base table.**
- 3. Create a second table with those columns and the non-key columns that they are dependent upon.**

B-12

Steps to Remove Transitive Dependencies

- 1** Determine which columns are dependent on another non-keyed column.
- 2** Remove those columns from the base table.
- 3** Create a second table with those columns and the non-key columns that they are dependent upon.
- 4** Create a foreign key in the original table linking to the primary key of the new table.

Tables Already in Third Normal Form

No non-key column can be functionally dependent upon another non-key column.

RECEIVED_
MESSAGES
(2NF)

USER _ID	MSE REC _ID DATE
2301	54101 05/07
2301	54098 07/12
2301	54445 10/06
5607	54101 05/07
5607	54512 06/07
5607	54660 12/01
7773	54101 05/07
7773	54554 03/17
0022	54101 05/07
0022	54512 06/07

MESSAGES
(2NF)

ID	SUBJECT	TEXT
54101	Meeting Today	There is.
54098	Promotions	I like to
54445	Next Assignmen	Your next
54512	Lunch?	Can you..
54660	Jogging Today?	Can you..
54554	Stock Quote	The lates

Are these two tables in third normal form? Why?

B-13

Converting to Third Normal Form

USERS

USER _ID	USER _NAME	SRVR _ID	SERVER _NAME
2301	Smith	3786	IMAP05
5607	Jones	6001	IMAP08
7773	Walsh	9988	EMEA01
0022	Patel	9988	EMEA01

USERS

ID	NAME	SRVR _ID
2301	Smith	3786
5607	Jones	6001
7773	Walsh	9988
0022	Patel	9988

MAIL_
SERVER

ID	NAME
3786	IMAP05
6001	IMAP08
9988	EMEA01

B-14

The theory of normalization goes further than the third normal form to cater for several problematic constructions that may remain. Those normal forms are outside the scope of this lesson.

Summary

Summary

1NF The table must express a set of unordered, two-dimensional tables. The table cannot contain repeating groups.

2NF The table must be in 1NF. Every non-key column must be dependent on all parts of the primary key.

3NF The table must be in 2NF. No non-key column may be functionally dependent on another non-key column.

An entity relationship model transforms into normalized data design.

B-15

Index

A

- arc 1-27, 4-12
 - both supertype and subtype implementation 7-25
 - exclusive 4-12
 - rules 4-14
- arc implementation 7-25
 - generic 9-26
 - rules 7-25
- arc or subtypes 4-16
- arcs
 - incorrect 4-15
 - mapping 7-19
- artificial key 9-11
- attribute 1-13
 - multiple
 - UID 4-7
 - redundancy 2-16
 - single
 - UID 4-7
 - single valued 1-13
- attribute constraint 4-19
- attribute representation
 - mandatory 1-19
 - optional 1-19
- attributes 3-19
 - naming 2-15
 - recycling 6-20
 - tracking 2-14
 - volatile 1-14
- attributes modeled as PROPERTY instance 6-20

B

- B*Tree
 - index 9-17
- barred relationship 4-6
- basket, pattern 6-6
- BFILE 9-6
- bill of material
 - pattern 6-12
- binary table 3-26
- bitmap index 9-17

- BLOB 9-6
- business function 1-23
- business rules 4-2

C

- cascade composed UID 4-7
- cascade delete 9-15
- cascade update 9-15
- chain
 - pattern 6-10
- CHAR 9-6
- check
 - conditional domain 4-20
 - conditional relationship 4-20
 - front door 4-20
 - range 4-20
 - state value transition 4-20
 - state value triggered 4-20
- check constraint 4-14
- classification, pattern 6-7
- CLOB 9-6
- column
 - current indicator 8-20
- column sequence 9-7
- columns 7-5
 - choosing for index 9-19
 - end date 8-18
 - foreign key
 - naming 7-9
- composed
 - UID 4-7
- concatenated index 9-18
- concept
 - evolution 2-11
- conceptual data modeling 1-8, 2-5, 3-26
- conceptual model 7-6
- conceptual modeling 1-4
- conceptual models 1-28
- conditional domain check 4-20
- conditional nontransferability 5-9
- conditional relationship 4-20
- constraint
 - check 4-14
 - declarative 7-7
- constraints 4-2
 - check

-
- naming 7-10
 - foreign key
 - naming 7-9
 - hierarchy 6-9
 - special 4-20
 - time-related 5-8
 - convention
 - naming 7-8
 - conventions
 - sensible use 6-18
 - crowsfoot 3-7
 - current indicator column 8-20
- D**
- data 2-4
 - normalized B-3
 - unnormalized B-5
 - data blocks 9-30
 - data files 9-30
 - data modeling
 - conceptual 2-5
 - physical 2-5
 - data type
 - BFILE 9-6
 - BLOB 9-6
 - CHAR 9-6
 - CLOB 9-6
 - DATE 9-6
 - LONG 9-6
 - LONG RAW 9-6
 - NUMBER 9-6
 - VARCHAR2 9-6
 - data types
 - Oracle 9-5
 - data warehouse 2-6
 - pattern 6-16
 - data warehouse system
 - design strategy 7-8
 - star model 7-10, 9-14
 - database
 - hierarchical 2-6
 - network 2-6
 - object oriented 2-6
 - relational 2-6
 - semantic 2-6
 - database structure
 - data blocks 9-30
 - data files 9-30
 - extents 9-30
 - Oracle 9-29
 - partitions 9-29
 - segments 9-29
 - tablespaces 9-29
 - DATE 9-6
 - date
 - end 5-7
 - start 5-7
 - date as Opposed to day 5-5
 - date or day 5-5
 - date time 5-6
 - declarative constraint 7-7
 - default and nullify 9-15
 - definition
 - denormalization 8-4
 - definition of an entity 1-10
 - degree 3-7
 - delete
 - cascade 9-15
 - restrict 9-14
 - denormalization
 - definition 8-4
 - denormalization techniques
 - derivable values 8-5
 - hard-coded values 8-5
 - pre-joining tables 8-5
 - derivable 1-8
 - derivable values
 - storing 8-6
 - design
 - distributed 9-27
 - old fashioned 9-25
 - design strategy
 - client-server 7-12
 - data warehouse approach 7-8
 - discriminator column 7-20
 - distributed design 9-27
 - benefits 9-28
 - domain 4-19
 - conditional 4-20
 - drawing conventions 6-17

E

- electronic mail 2-9
- elements
 - arc 1-27
 - nontransferability 1-27
 - subtype 1-27
 - unique identifier 1-27
- end date 5-7
- end date columns 8-18
- entities
 - event 2-20
 - intangible 2-20
 - tangible 2-20
- entity 3-25
 - formal description 2-7
 - inheritance 2-17
 - intersection 3-25
 - naming 2-7
 - subtypes 2-17
 - supertype 2-17
- entity DAY 5-6
- entity definition
 - evolution 2-11
- entity life cycle 2-12
- entity relationship diagram 1-17
- entity relationship model 1-17
- entity relationship modeling 1-7, 1-28
- ER diagram
 - soft box 1-18
- ER model
 - transform 7-4
- evolution of a concept 2-11
- exclusive arc 4-12
- extents 9-30

F

- fan trap
 - pattern 6-15
- first normal form B-7
- foreign key
 - cascade delete 9-15
 - cascade update 9-15
 - columns 7-13
 - constraints 7-13
 - default and nullify 9-15

- optional composed 7-15
- foreign keys 7-5
- form
 - first normal B-7
 - second normal B-9
- formal description of the entity 2-7
- front door check 4-20
- function
 - business 1-23
 - modeling 1-23
- function based index 9-18
- functionality 1-23, 2-13

G

- generic arc implementation 9-26
- generic model 6-22
- generic modeling 6-19
- generic models 6-20, 6-21
- graphical elements 1-17

H

- hard-coded values 8-10
- hidden relationships 4-18
- hierarchies
 - disputable 6-8
 - false 6-8
- hierarchy
 - constraints 6-9
 - pattern 6-8
- hierarchy level indicator 8-22
- historical price 5-10
- homonyms 2-8
- house building metaphor 1-5

I

- identification 4-4
 - in database 4-5
 - indirect 4-8
 - problems 4-4
 - real world 4-5
- identifiers
 - information-bearing 4-11
- incorrect arcs 4-15
- incorrect UUIDs 4-10

-
- index
 - choosing columns 9-19
 - partitioned 9-22
 - unique 9-8
 - when used 9-21
 - index organized table 9-18
 - index types 9-17
 - B*Tree 9-17
 - bitmap 9-17
 - concatenated index 9-18
 - function based index 9-18
 - reverse key 9-17
 - tree balanced 9-17
 - indexes 9-16
 - indicator
 - hierarchy level 8-22
 - indirect identification 4-8
 - information 2-4
 - types 1-24
 - information-bearing identifiers 4-11
 - inheritance 2-17
 - instances 1-10, 1-11
 - integrity
 - referential 9-14
 - intersection entity 3-25
- J**
- journalling 5-4, 5-17
- K**
- keeping details with master 8-12
 - key
 - artificial 9-11
 - foreign 9-14
 - primary
 - desirable properties 9-9
 - keys
 - foreign 7-5
 - primary 7-5, 9-8
 - short-circuit 8-16
 - unique 7-5, 9-8
- L**
- life cycle
 - entity 2-12
 - logging 5-4, 5-17
 - logic
 - referential 5-9
 - LONG 9-6
 - LONG RAW 9-6
- M**
- mandatory 3-7, 3-10
 - many to many (m:m) 3-9
 - mapping
 - basic 7-12
 - entity 7-12
 - nontransferable relationships 7-15
 - relationship 7-14
 - terminology 7-7
 - mapping arcs 7-19
 - mapping barred relationships 7-15
 - mapping many-to-many relationships 7-17
 - mapping one-to-one relationships 7-18
 - mapping subtypes 7-20
 - master
 - keeping details 8-12
 - repeating single detail 8-14
 - master detail, pattern 6-5
 - model
 - conceptual 7-6
 - relational 7-6
 - modeling
 - generic 6-19
 - modeling time 5-4
 - multiple attribute
 - UID 4-7
- N**
- name space 7-11
 - naming
 - attributes 2-15
 - check constraints 7-10
 - convention 7-8
 - entities 2-7
 - foreign key columns 7-9
 - foreign key constraints 7-9
 - relationships 3-5

-
- restrictions with Oracle 7-10
 - tables 7-8
 - naming relationships 3-5
 - negotiated prices 5-14
 - nested subtypes 2-19
 - network
 - pattern 6-11
 - network structures 6-11
 - nontransferability 1-27, 3-8
 - conditional 5-9
 - normalization B-3
 - normalized data B-3
 - nouns 2-7
 - NUMBER 9-6
- O**
- old fashioned design
 - generic arc implementation 9-26
 - unique index 9-25
 - old fashioned design
 - check option views 9-25
 - OLTP system 7-6
 - one to many (1:m) 3-9
 - one to one (1:1) 3-9
 - onstraint
 - attribute 4-19
 - optional 3-7
 - optionality 3-6
 - Oracle data types 9-5
 - Oracle database structure 9-29
- P**
- partitioned index 9-22
 - partitioning tables 9-22
 - partitions 9-29
 - pattern
 - basket 6-6
 - bill of material 6-12
 - chain 6-10
 - classification 6-7
 - data warehouse 6-16
 - fan trap 6-15
 - hierarchy 6-8
 - master detail 6-5
 - network 6-11
 - roles 6-14
 - patterns 6-4
 - physical data modeling 2-5
 - pre-joining tables 8-8
 - price 5-10
 - negotiated 5-14
 - price history 5-10
 - price list 5-10, 5-12
 - priced product 5-10
 - primary key
 - desirable properties 9-9
 - primary keys 7-5, 9-8
 - primary UID 4-9
 - primary unique identifier 3-18
 - product 5-10
 - properties
 - primary key 9-9
- R**
- range check 4-20
 - recursive relationship 3-4
 - recycling of attributes 6-20
 - redundancy 2-16
 - relationships 3-15
 - referential integrity 9-14
 - referential logic 5-9
 - relational databases 2-6
 - relational model 7-6
 - relationship
 - conditional 4-20
 - many to many 3-11
 - mapping 7-14
 - master-detail 6-5
 - one to many 6-5
 - recursive 3-4
 - relationship ends
 - degree 3-7
 - optionality 3-6
 - relationship name 3-5
 - relationship representation 1-20
 - relationships 1-15, 3-19
 - barred
 - mapping 7-15
 - hidden 4-18
 - mandatory 1-21
 - many to many 3-9

-
- mapping 7-17
 - many to one 3-9
 - mapping nontransferable 7-15
 - mapping one to many 7-14
 - one to many 3-9
 - one to one 3-13
 - one-to-one
 - mapping 7-18
 - resolving 3-25
 - resolving other 3-26
 - symmetric 6-13
 - UID 4-8
 - relationships
 - optional 1-21
 - repeating single detail with master 8-14
 - representation 4-4
 - reserved words. 2-15
 - resolving other relationships 3-26
 - resolving relationships 3-25
 - restrict
 - delete 9-14
 - update 9-14
 - reverse key index 9-17
 - roles
 - pattern 6-14
 - rows 7-5
 - rules
 - about arcs 4-14
 - business 4-2
 - subtype 2-18
 - transformation 7-6
- S**
- second normal form B-9
 - secondary UID 4-9
 - segments 9-29
 - sequences 9-13
 - set theory 1-12
 - sets. 1-12
 - short-circuit keys 8-16
 - similar structure 6-4
 - single attribute
 - UID 4-7
 - Snowflake model 6-16
 - soft box 1-18
 - special constraints 4-20
 - start date 5-7
 - state value transition check 4-20
 - state value triggered check 4-20
 - storage implication 7-27
 - arc implementation 7-29
 - subtype implementation 7-27
 - supertype implementation 7-27
 - storing derivable values 8-6
 - subtype 1-27
 - implementation
 - rules 7-23
 - implementation 7-23
 - rules 2-18
 - subtypes 2-17
 - mapping 7-20
 - nested 2-19
 - subtypes or arcs 4-16
 - supertype 2-17
 - implementation
 - rules 7-20
 - implementation 7-20
 - supertype and subtype implementation
 - arc 7-25
 - symmetric relationships 6-13
 - problem 6-13
 - solution 6-13
 - synonyms 2-7
- T**
- table
 - binary 3-26
 - index organized 9-18
 - naming 7-8
 - tables 7-5
 - partitioning 9-22
 - pre-joining 8-8
 - tablespaces 9-29
 - terminology mapping 7-7
 - three-tiered architecture 7-13
 - time
 - modeling 5-4
 - time-related constraints 5-8
 - tracking attributes 2-14
 - transformation rules 7-6
 - transforming the ER model 7-4
 - tree balanced index 9-17

types of information 1-24

U

UID

- cascade composed 4-7
- composed 4-7
- multiple attribute 4-7
- primary 4-9
- relationships 4-8
- secondary 4-9
- single attribute 4-7
- unique identifier 1-27, 4-6
 - primary 3-18
- unique index 9-8
- unique key 7-18
- unique keys 7-5, 9-8
- unnormalized data B-5
- update
 - cascade 9-15
 - restrict 9-14

V

- values 1-13
 - derivable
 - storing 8-6
 - hard-coded 8-10
- VARCHAR2 9-6
- views
 - usage 9-23
- volatile attributes 1-14

W

- words
 - reserved 2-15

