Data Modeling and Relational Database Design

Student Guide – Volume 2

20000GC13
Edition 1.3
August 2002
D37097

ORACLE®

Author

Patrice Daux
Jeff Gallus
Jan Speelpenning

Technical Contributors and Reviewers

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

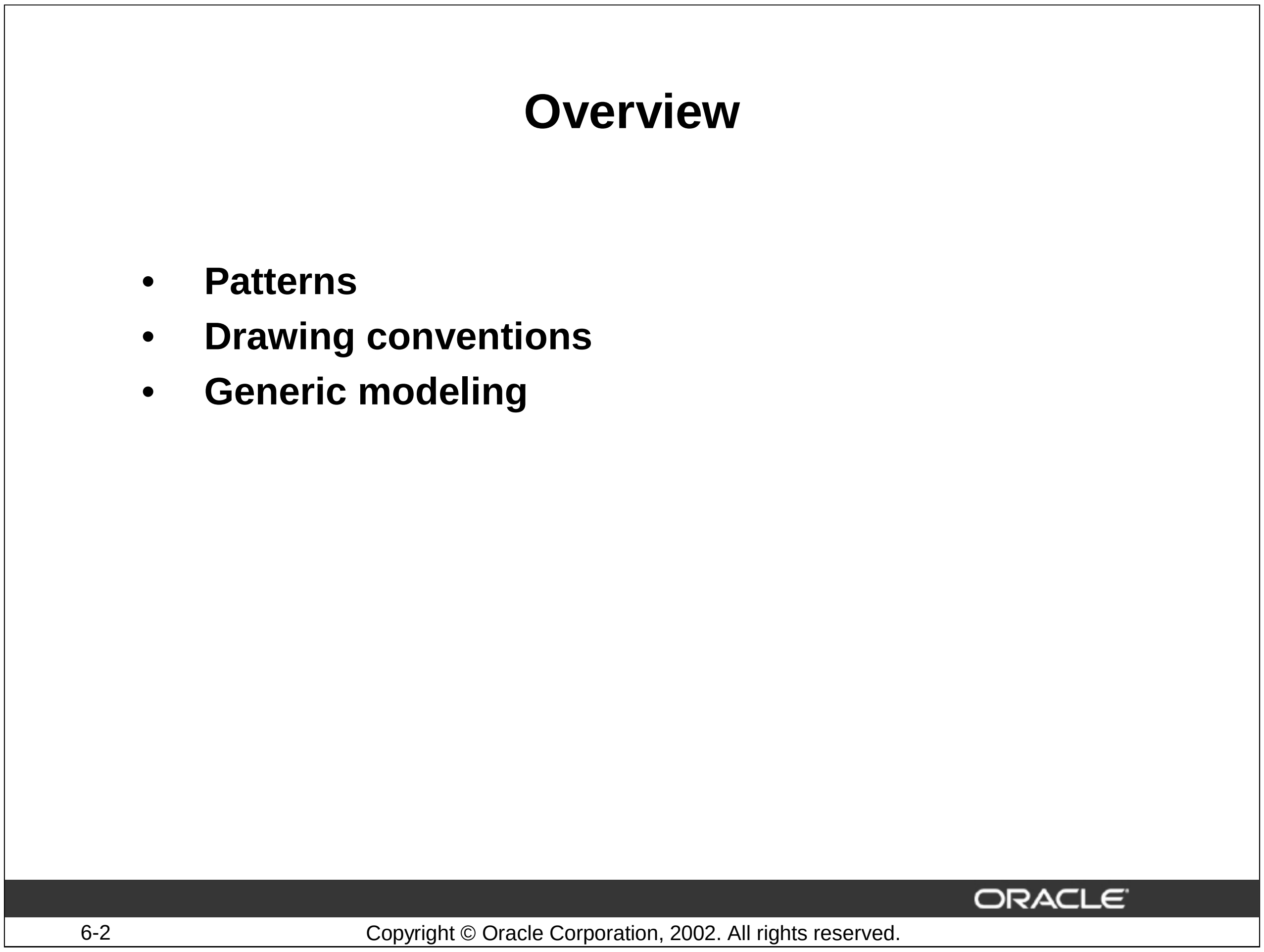
Publisher

Christine Markusic

Contents

6 Advanced Modeling Topics

ORACLE



Overview

- Patterns
- Drawing conventions
- Generic modeling

6-2

Introduction

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.

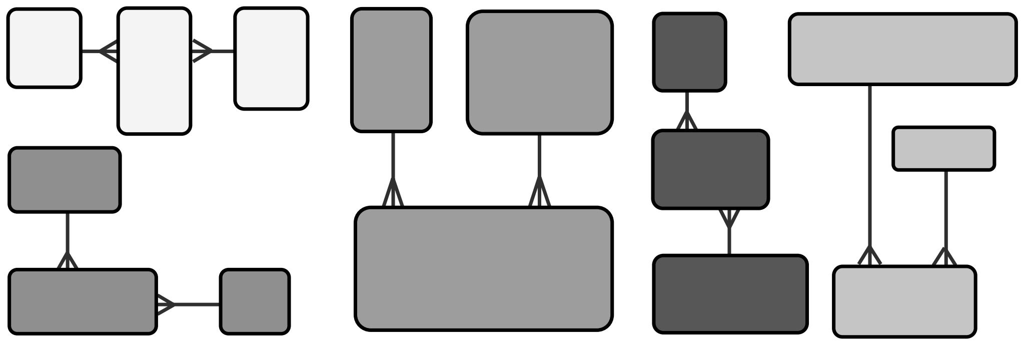
Objectives

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
- Similar rules and constraints?



ORACLE

6-3

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.

Patterns

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.

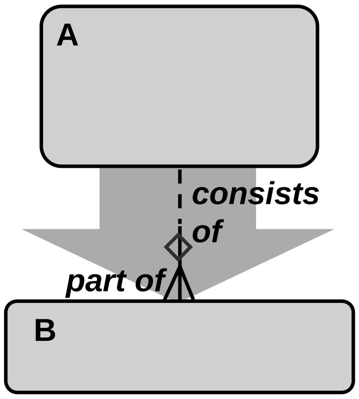
Are there similarities between marriage and job assignment? Of course, the business rules in the context of a marriage are different, because they are determined differently compared to those of a job assignment. But when you are aware of the similarities, you can easily check if business rules of the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

Patterns: Master–Detail

- **Characteristic: *consists of***
  An instance of B only exists in the context of an A
- **Metaphor: Master – Detail**



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.

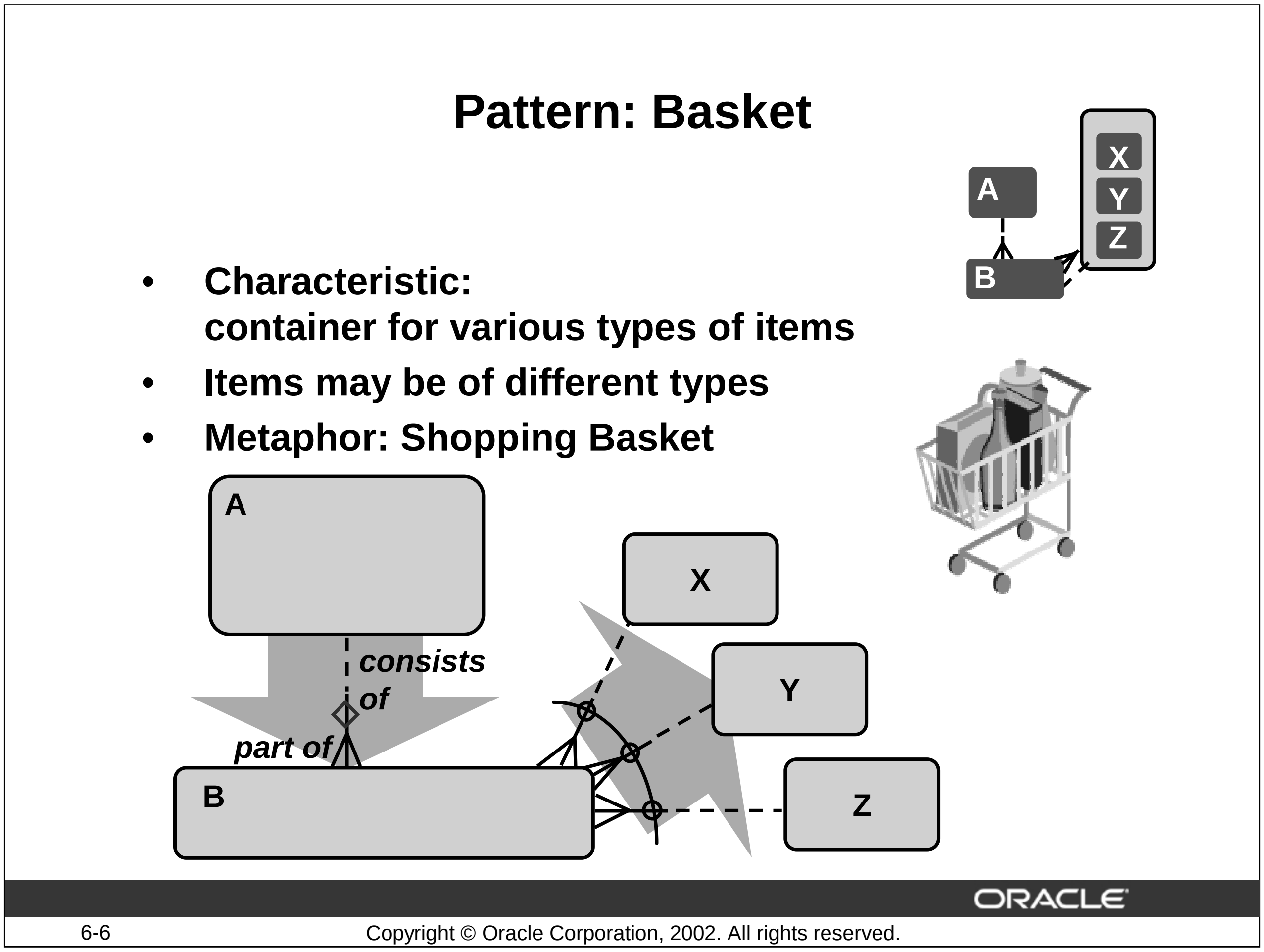
It is very rare that these relationships are transferable; if an instance of B is connected to the wrong instance of A, it is far more likely that the instance of B is deleted and then recreated in the context of the correct A.

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.

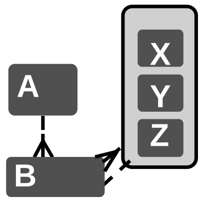
Implementation

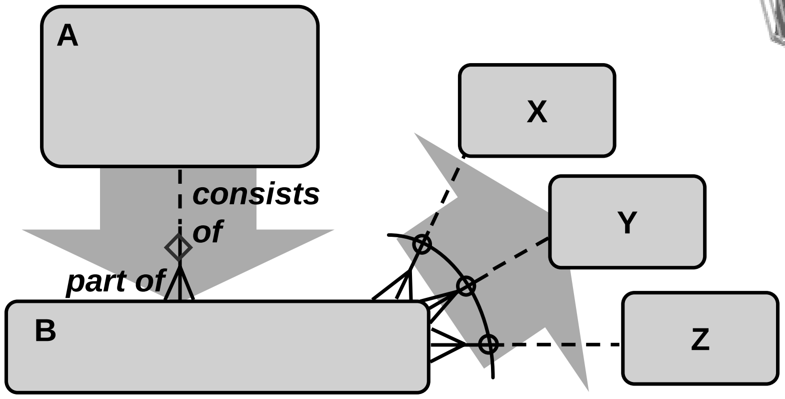
The tables that come from this master-detail pattern should be considered as clustered.



Pattern: Basket

- **Characteristic: container for various types of items**
- **Items may be of different types**
- **Metaphor: Shopping Basket**



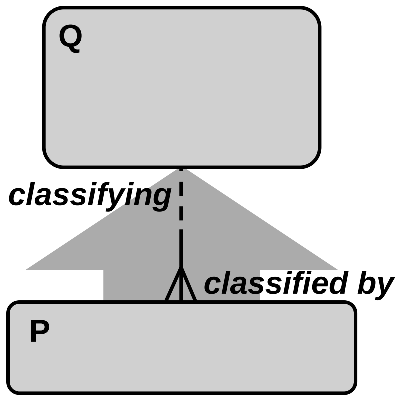


Basket

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.

Patterns: Classification

- **Characteristic: *classified by, grouped by***
  Q exists independently, may be related
- **Metaphor: EMPLOYEE–DEPARTMENT**



Classification

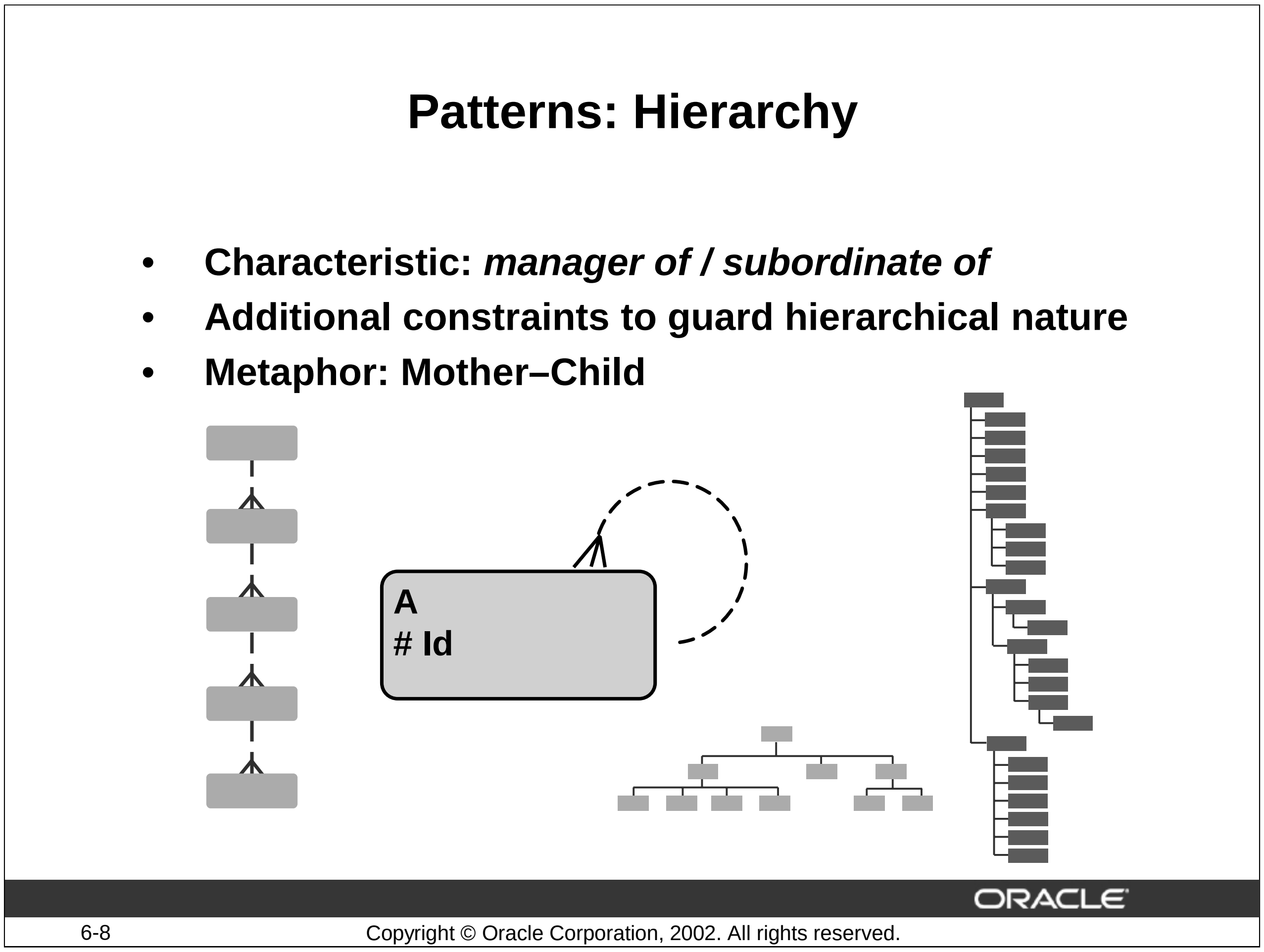
This is again a 1:m relationship, but now the main orientation is from P to Q.

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.

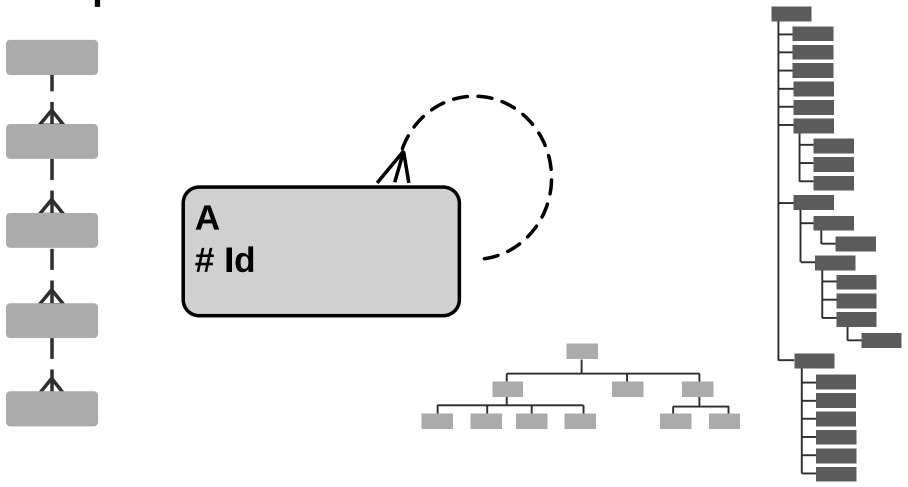
Typical classification-type relationship names:
- Classified by
- Grouped by
- Assigned to
- (Exists) In

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



Patterns: Hierarchy

- **Characteristic: *manager of / subordinate of***
- **Additional constraints to guard hierarchical nature**
- **Metaphor: Mother–Child**



ORACLE

6-8 Copyright © Oracle Corporation, 2002. All rights reserved.

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 C1 owns company C2, company C2 owns company C3, could it be that company C3 owns the shares of company C1? 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.

Hierarchy

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 center of the illustration, only requires an instance of A to refer to a valid instance of A. A1 referring to A1 is fine, according to the model. A2 referring to A3 and A3 referring to A2 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.

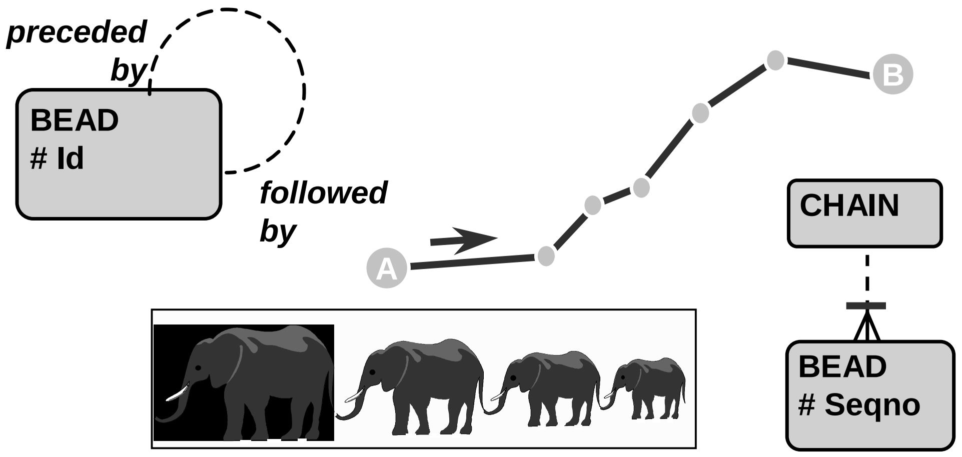
Implementation

The first constraint, A1 may not refer to A1, 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.

Patterns: Chain

- **Characteristic: *preceded by / followed by***
- **Sequence is important**
- **Metaphor: Elephants**



Chain

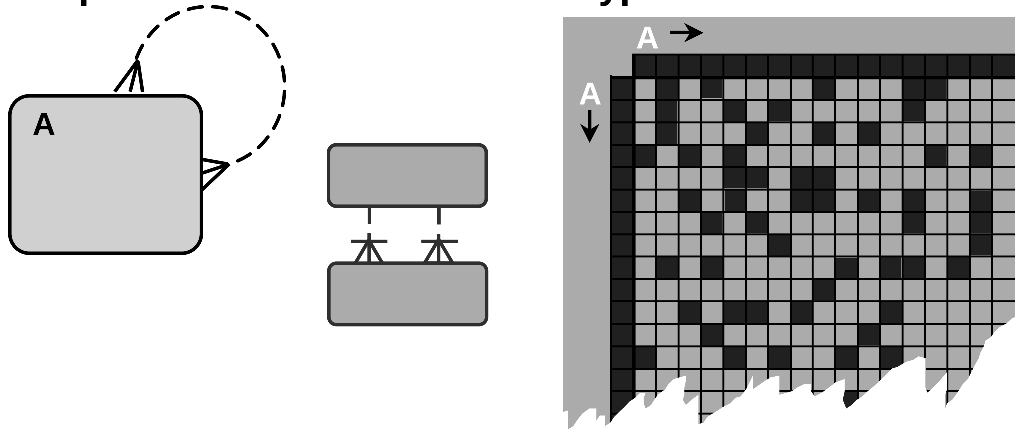
A Chain (of beads) can be regarded as a special kind of hierarchy. A chain is a recursive relationship of an entity. The relationship of the chain is a 1:1 relationship as a chain is characterized by the fact that an object in the chain is preceded and followed by one object at most.

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.

Patterns: Network

- **Characteristic: *pairs***
  Every A can be connected to every A
  (sometimes: to every *other* A)
- **Metaphor: Web Document with Hyperlinks**



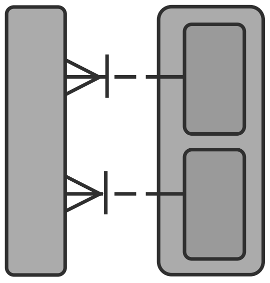
Network

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:

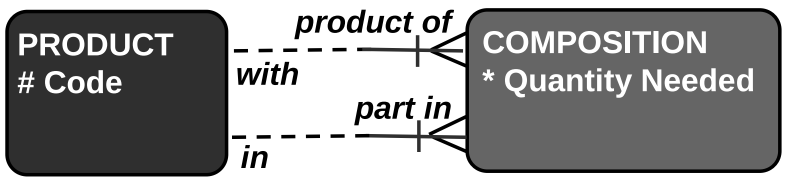


6-12 Copyright © Oracle Corporation, 2002. All rights reserved.

Network

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

Bill of Material



Code	Name
914.53	AAAAAAAA
914.54	AAA
914.55	BBBBBBBB
914.56	B
[illegible]	[illegible]
	DDDDD

COMPOSITIONS

Prod_code	Part_code	Quantity
854.01	604.18	1
854.01	604.19	1
854.01	914.54	2
914.54	914.55	1
914.54	914.56	1
[illegible]	[illegible]	[illegible]

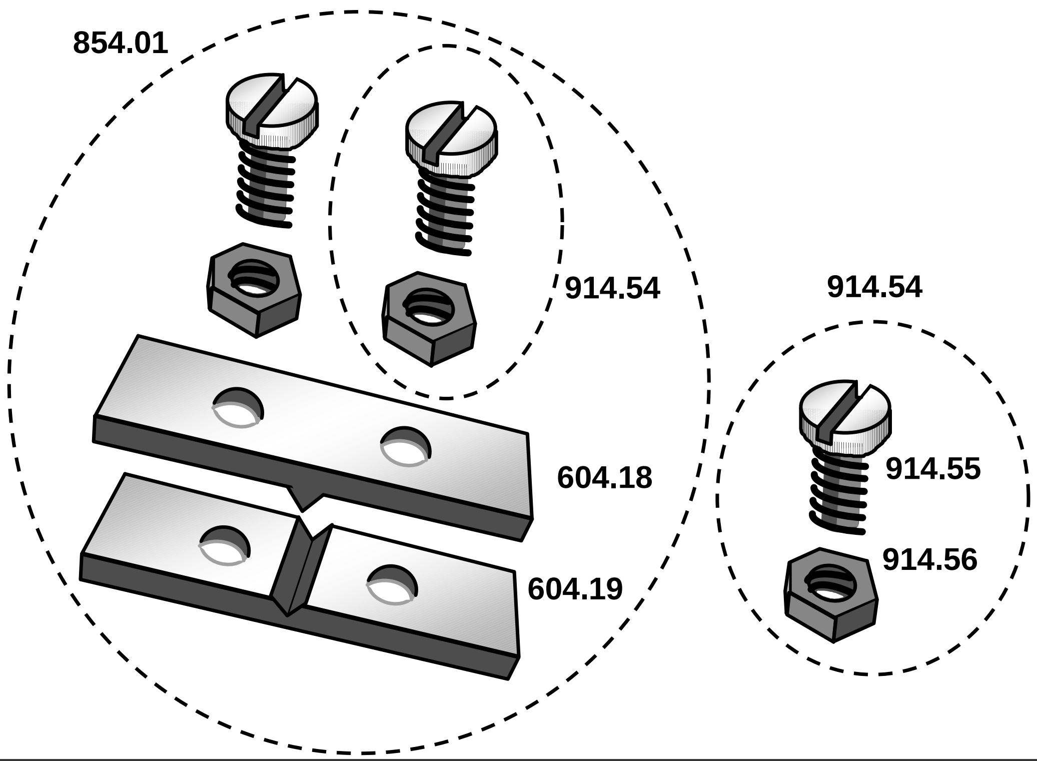
ORACLE

Network

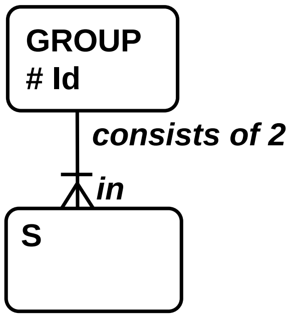
Bill of Material

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.

Bill of Material - Example



Symmetric Relationship



Group_id	S
1	S_1
1	S_2
2	S_3
2	S_4
3	S_5
3	S_6

Symmetric Relationships

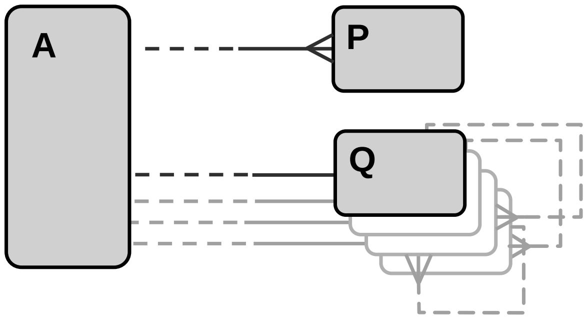
Symmetric recursive relationships cause a very special kind of problem which is more complex than you would assume. In most contexts a record of a pair (A1, A2) has a different meaning when referred to as (A2, A1). For example, if the model is about entity PERSON and the relationship is mother of /daughter of, then the existence of person pair (P1, P2) would mean the exclusion of the possibility of pair (P2, P1). The recursive relationship of PERSON and family of / family of. Here, if (P1, P2) is true, then (P2, P1) is equally true. This is called a symmetric relationship. There are other symmetric recursive relationships such as: STATION directly connected by rail with STATION,

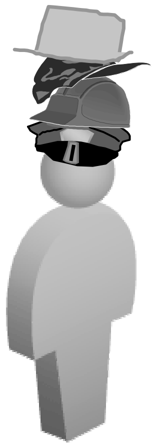
Symmetric Relationships: Problem When in a symmetric relationship the pair (S1, S2) is valid, the pair (S2, S1) must be valid as well. Nevertheless, it would not make much sense to record both pairs as that would essentially store the same information twice—which would oppose one of the basic principles 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 (S1, S2) as Group1, (S3, S4) 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.

Patterns: Roles

- **Characteristic: *is / is* 1:m (or 1:1) relationships**
- **Metaphor: Person–Many Hats (not necessarily concurrent...)**

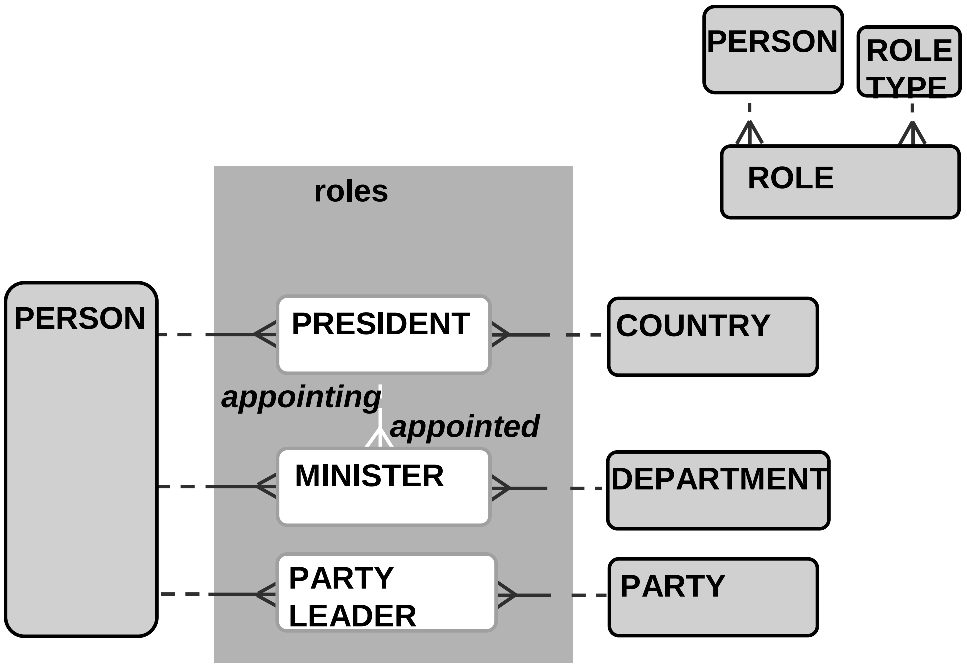




Roles

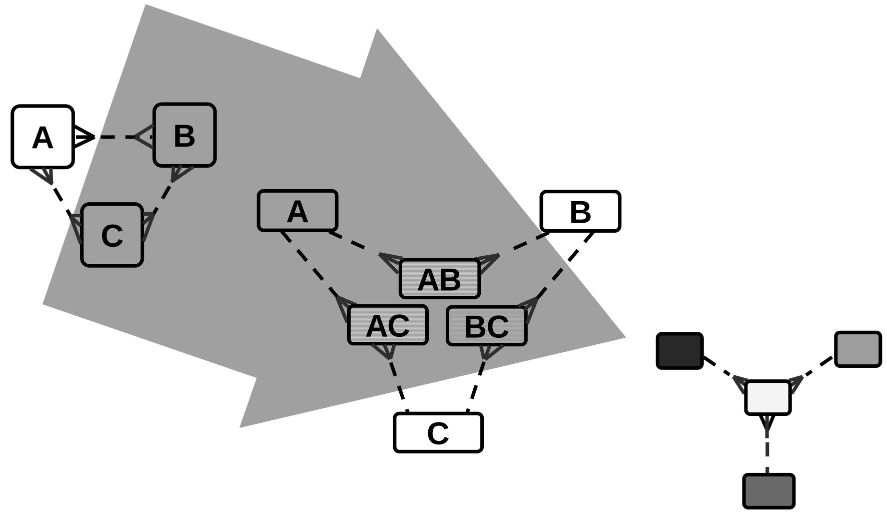
Roles often occur when a system needs to know more about people than the basic Name/Address/City information. Modeling the roles as separate entities offers the possibility to show which attributes are mandatory for a particular role, and, if necessary, to show relationships between the various roles. 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.

Roles


PERSON
ROLE TYPE
ROLE
roles
PERSON
PRESIDENT
COUNTRY
appointing
appointed
MINISTER
DEPARTMENT
PARTY LEADER
PARTY

Fan Trap

- Characteristic: ring of m:m related entities
- Metaphor: ABC Combination



Fan Trap

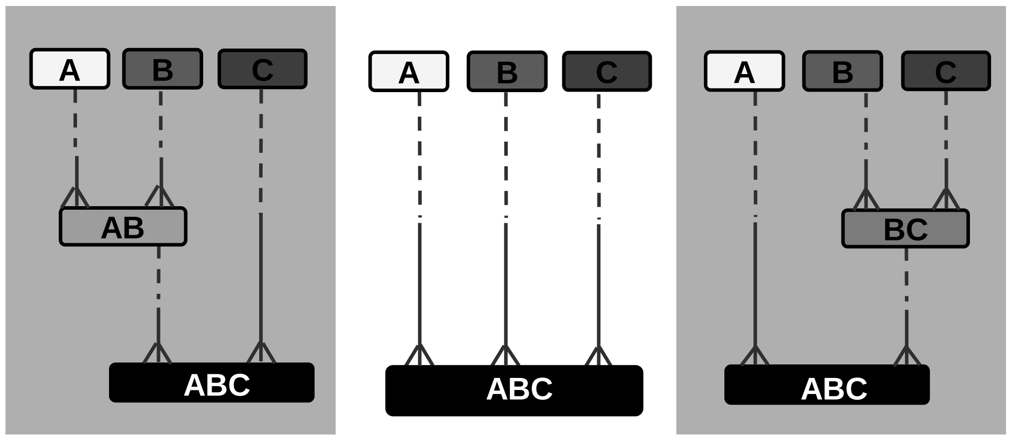
A Fan Trap (named after the characteristic shape of the solution) occurs when three or more entities are related through m:m relationships and form a ring. Usually you should replace the relationships with a central entity having several m:1 relationships. Preventing a fan trap is similar to resolving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

Fan Trap Resolved

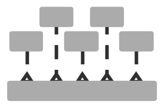


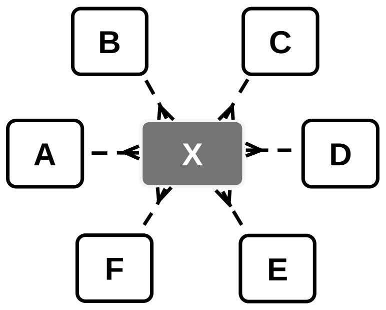
AB functions as
list of values

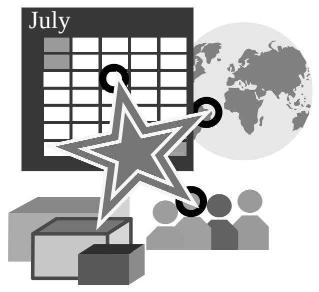
BC functions as
list of values

Patterns: Data Warehouse

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







ORACLE

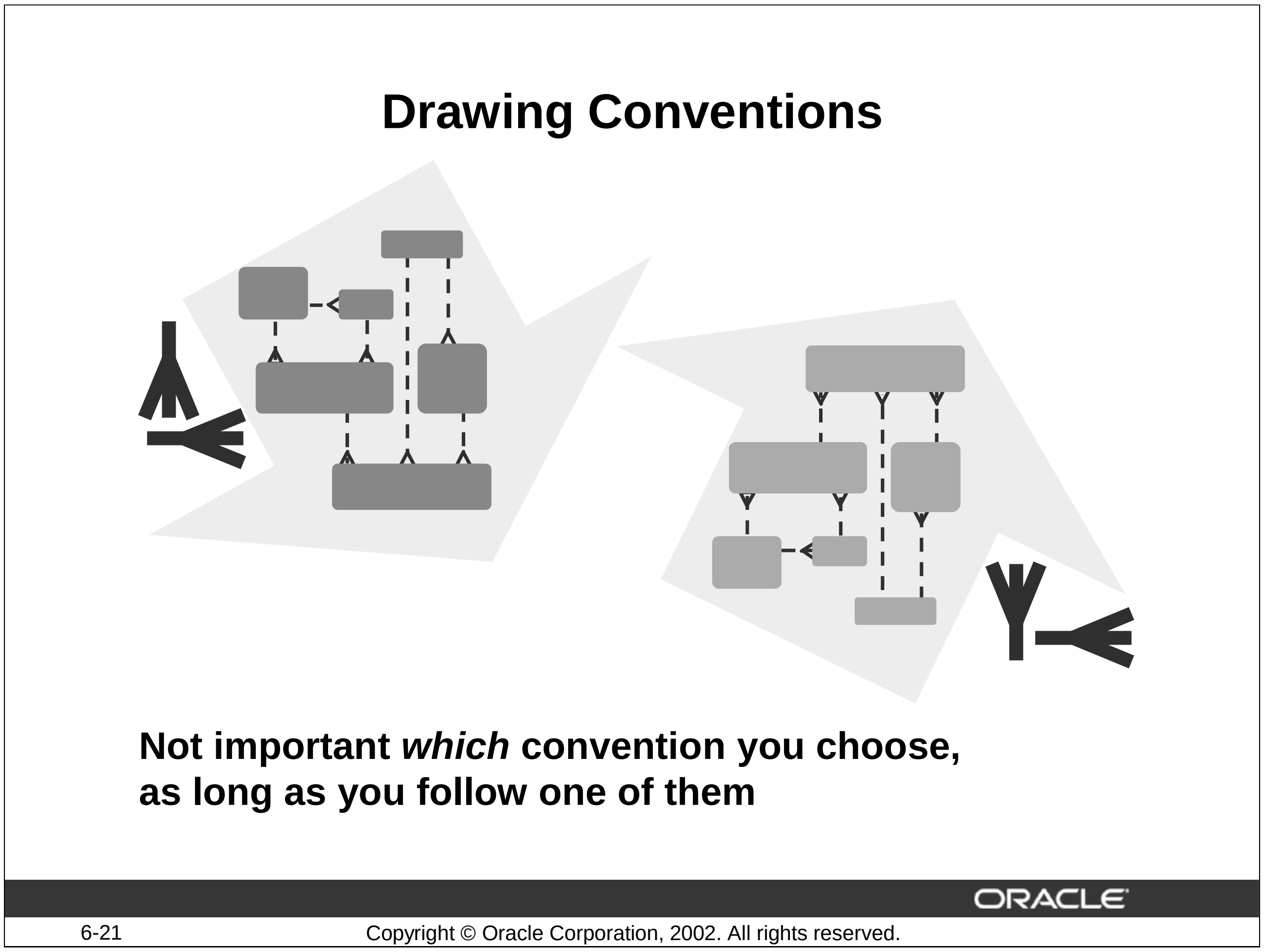
Data Warehouse

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.

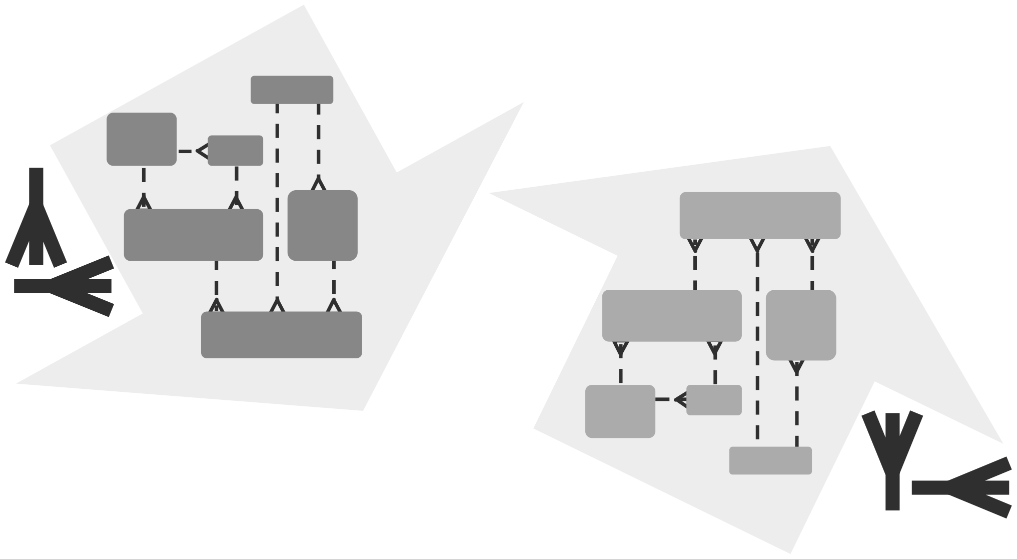
Common dimensions represent entities such as:

- 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



Not important *which* convention you choose, as long as you follow one of them

Drawing Conventions

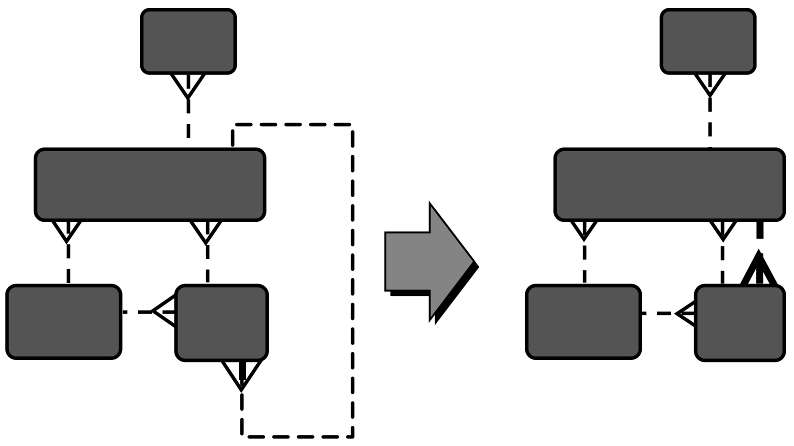
Two drawing conventions are widely in use: one that positions the entities with the high volumes at the top of the paper and one that does the opposite. Both try to avoid crossing relationship lines, partially overlapping entities, and relationship lines that cross entities. Whatever convention you choose, 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



But:

Readability first

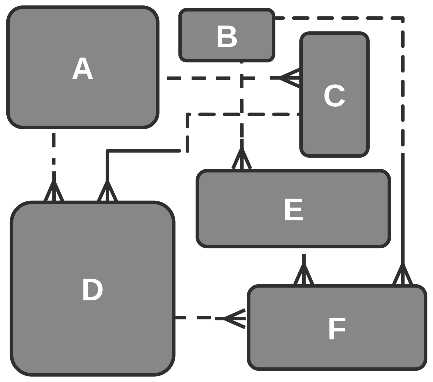
ORACLE

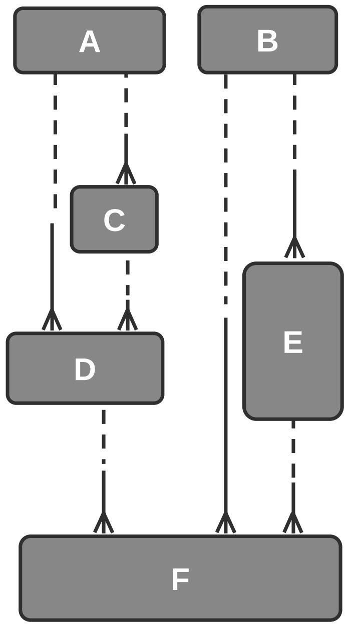
6-22

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.

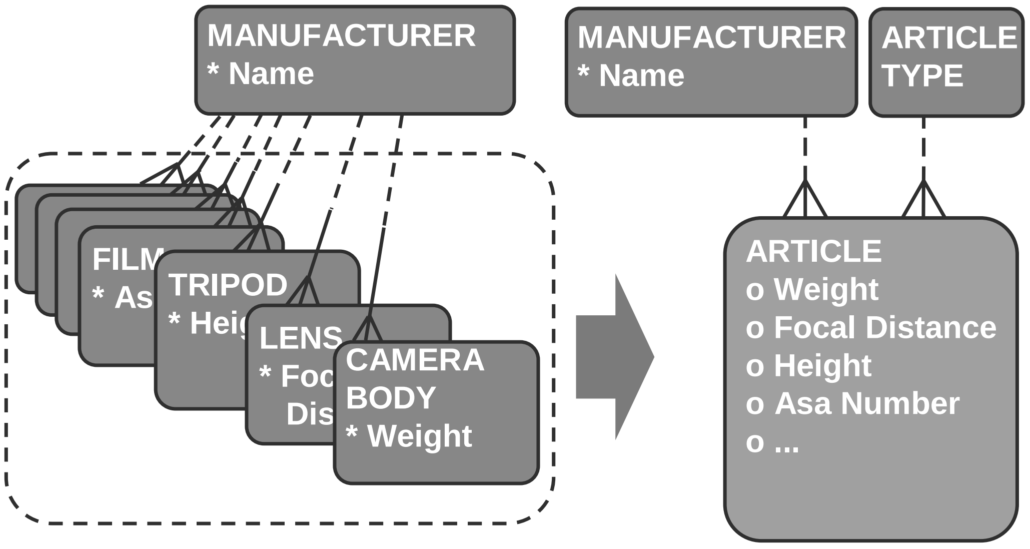
Model Readability





- Takes space
- Subject to taste

Generic Modeling



What is Generic Modeling?

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

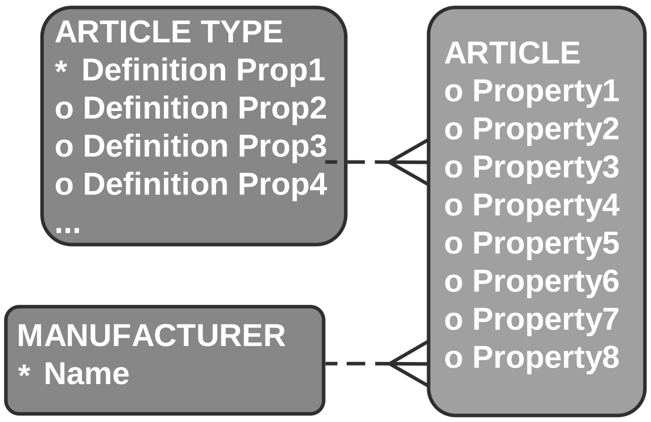
Suppose you are to make a model for a photographer's shop. The business typically sells many different articles, for example, camera bodies, compact cameras, lenses, films. For each type of article, there are between, say, 10 and 500 different types. You can model every type as an entity, for example,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

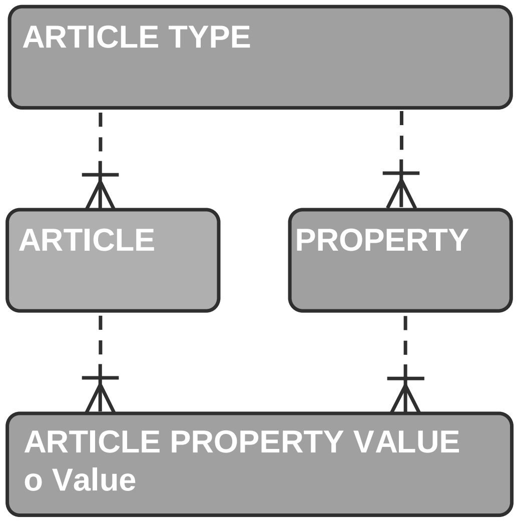
Generic Modeling



Generic Models

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

Generic Model



Generic Models

Recycling of Attributes

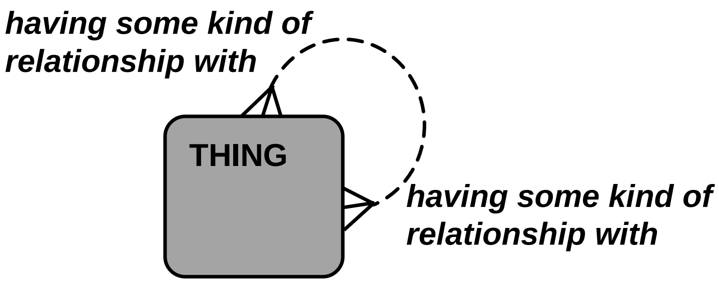
You can use this model if it is safe to assume the articles will have a limited number of attributes. This limit may be a high number but must be set beforehand. Property1 may contain the Asa Number for instances of ARTICLE of TYPE Film and may contain Weight for instances of ARTICLE of TYPE Camera Body 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.

Generic

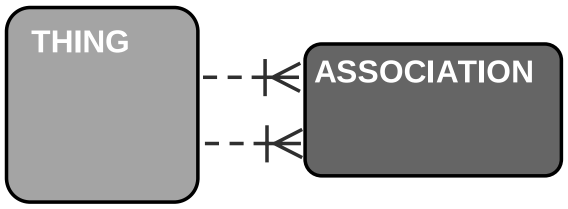

having some kind of relationship with
THING
having some kind of relationship with

More Generic Models

Everything is a “Thing”

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

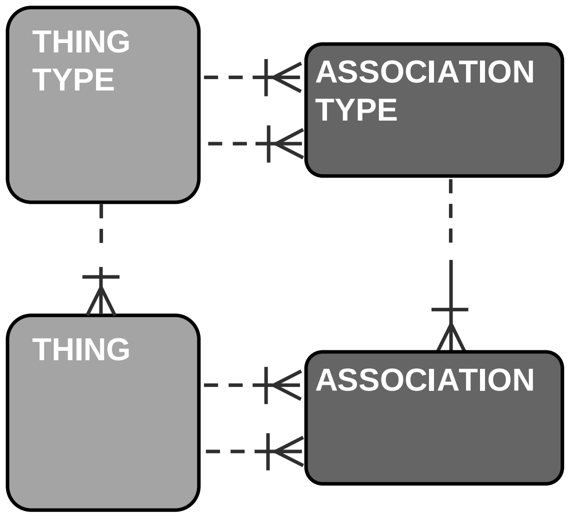
More Generic



More Generic Models

Resolving the m:m relationship:

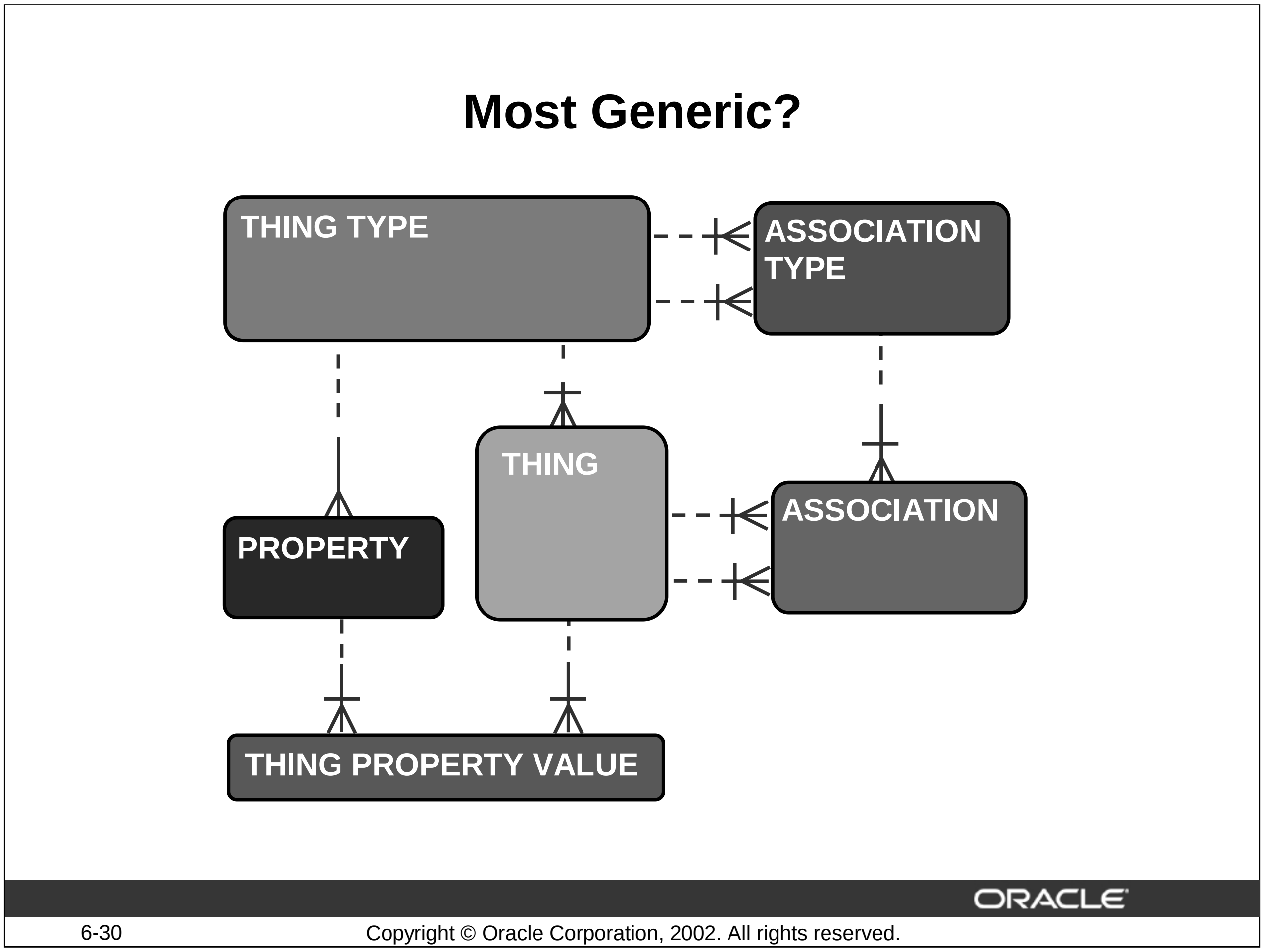
More Generic Plus



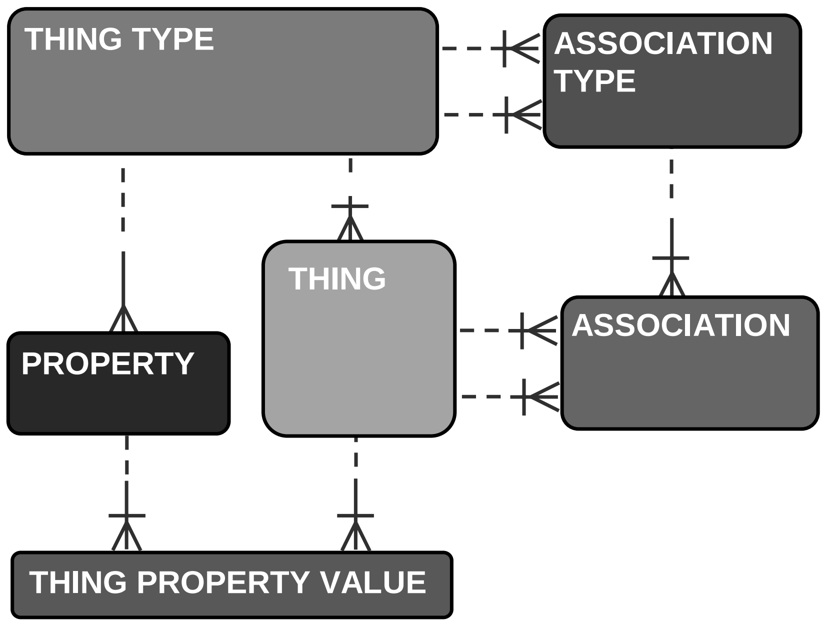
More Generic Models

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?



ORACLE

6-30 Copyright © Oracle Corporation, 2002. All rights reserved.

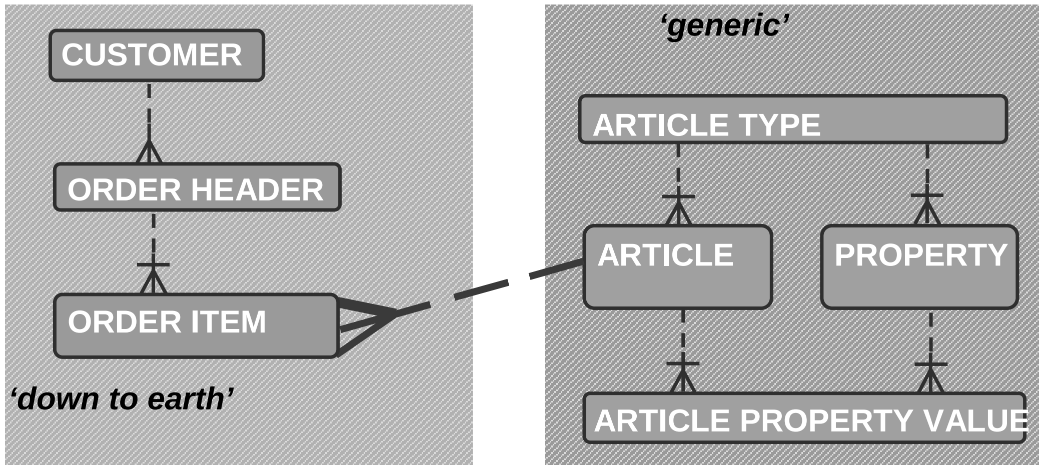
Most Generic Model

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.

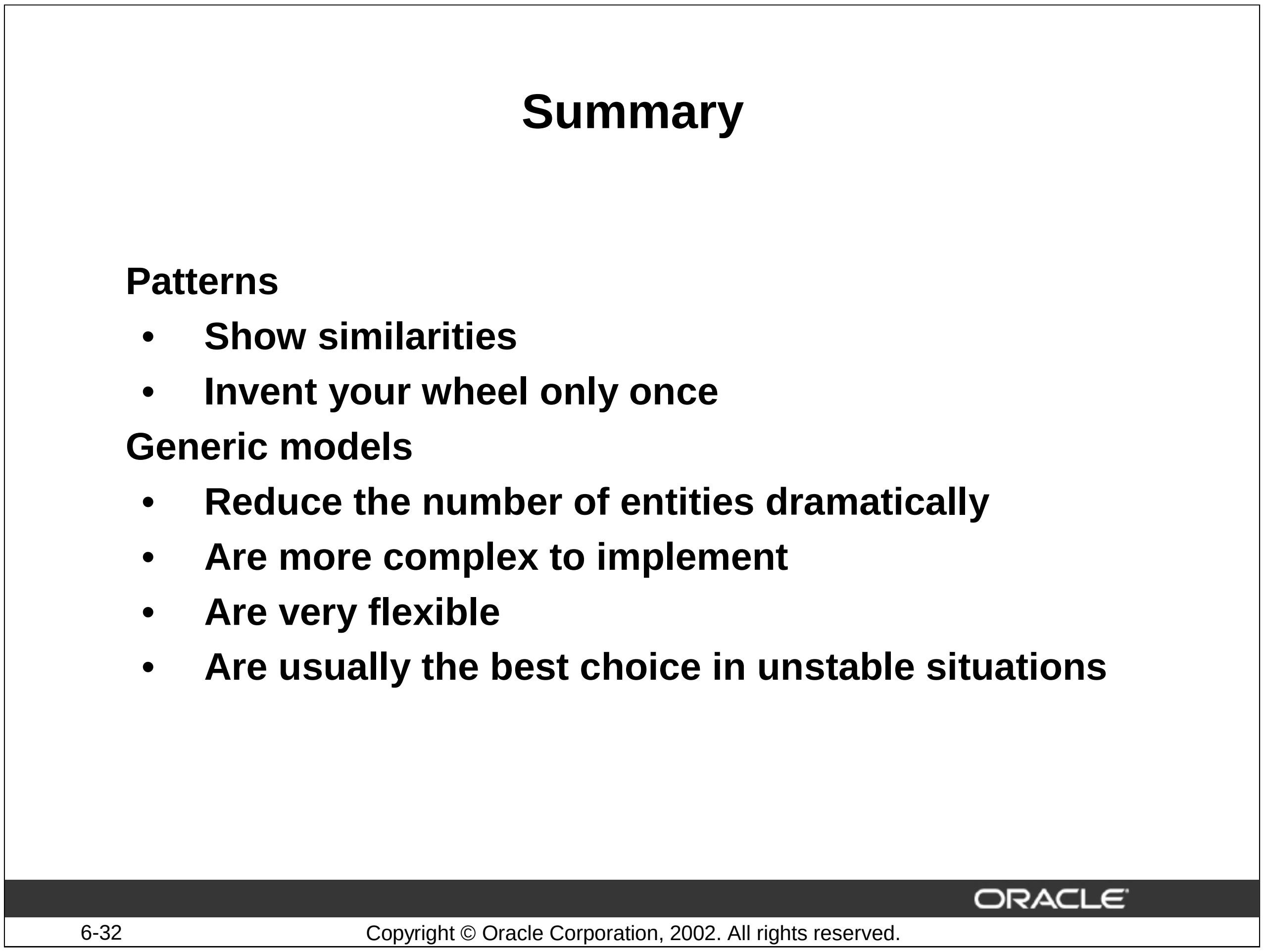
Best of Two Worlds



Most Generic Model

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

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

6-32

Summary

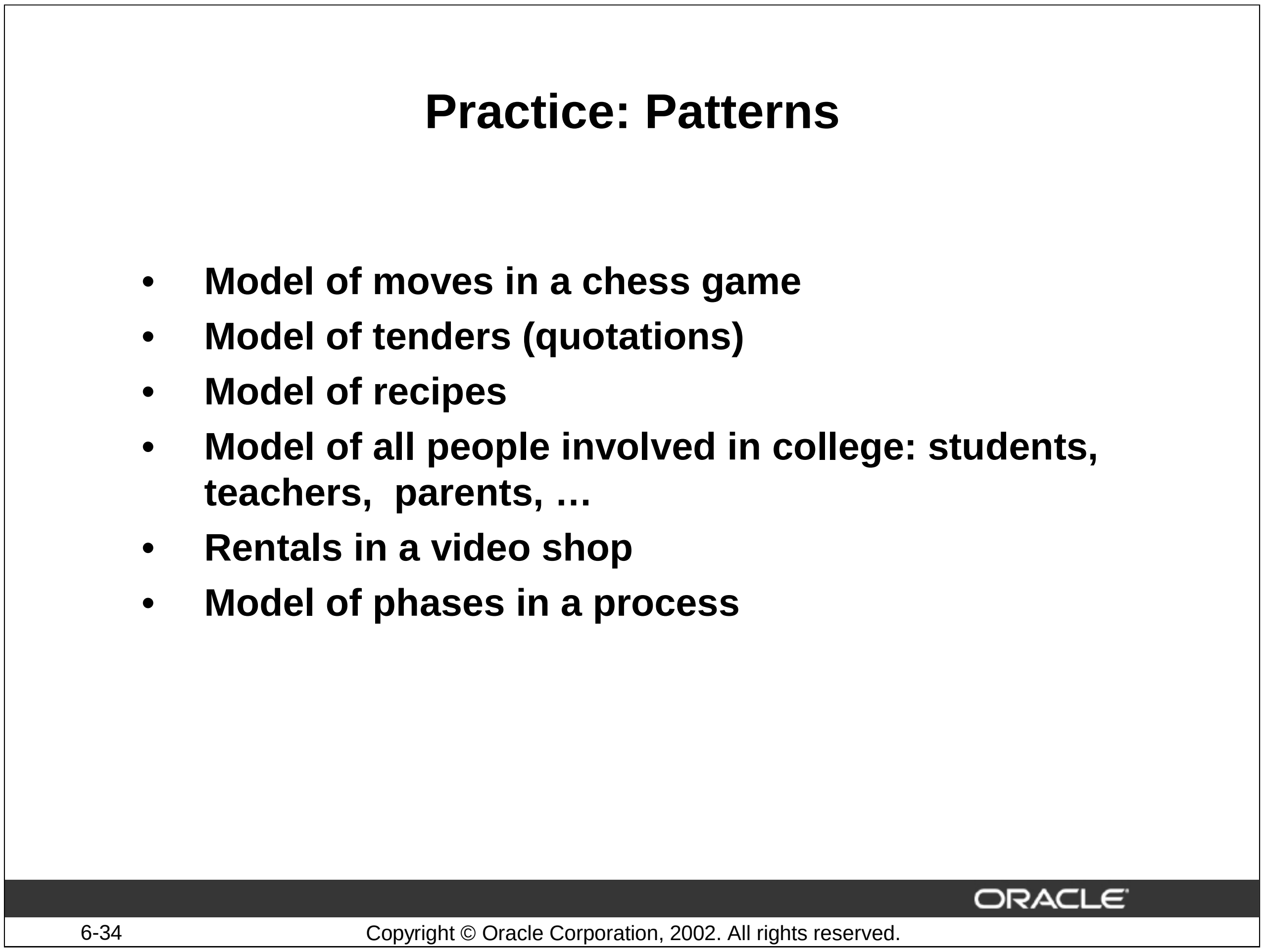
Thinking in terms of patterns forms a valuable way of doing quality checks on a conceptual data model. Often constraints and considerations in one context can be transferred to the other context with a simple translation.

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.

Practices

- Patterns
- Data Warehouse
- Argos and Erats
- Synonym



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

6-34

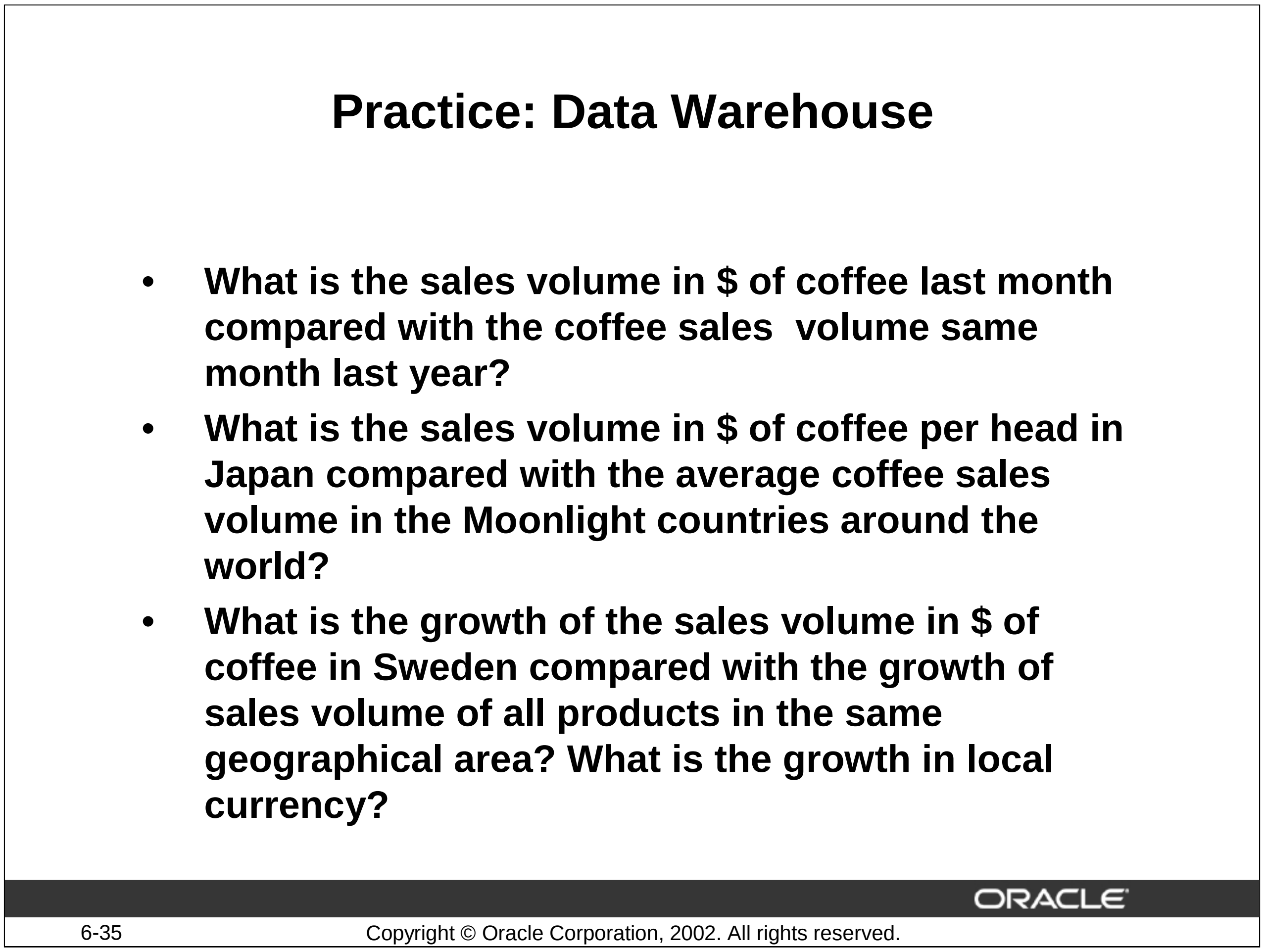
Practice 6-1: Patterns

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: Data Warehouse

- 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?

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.

Your Assignment

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

Practice: Data Warehouse

- **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?**

6-36

Practice 6-2: Data Warehouse

Goal

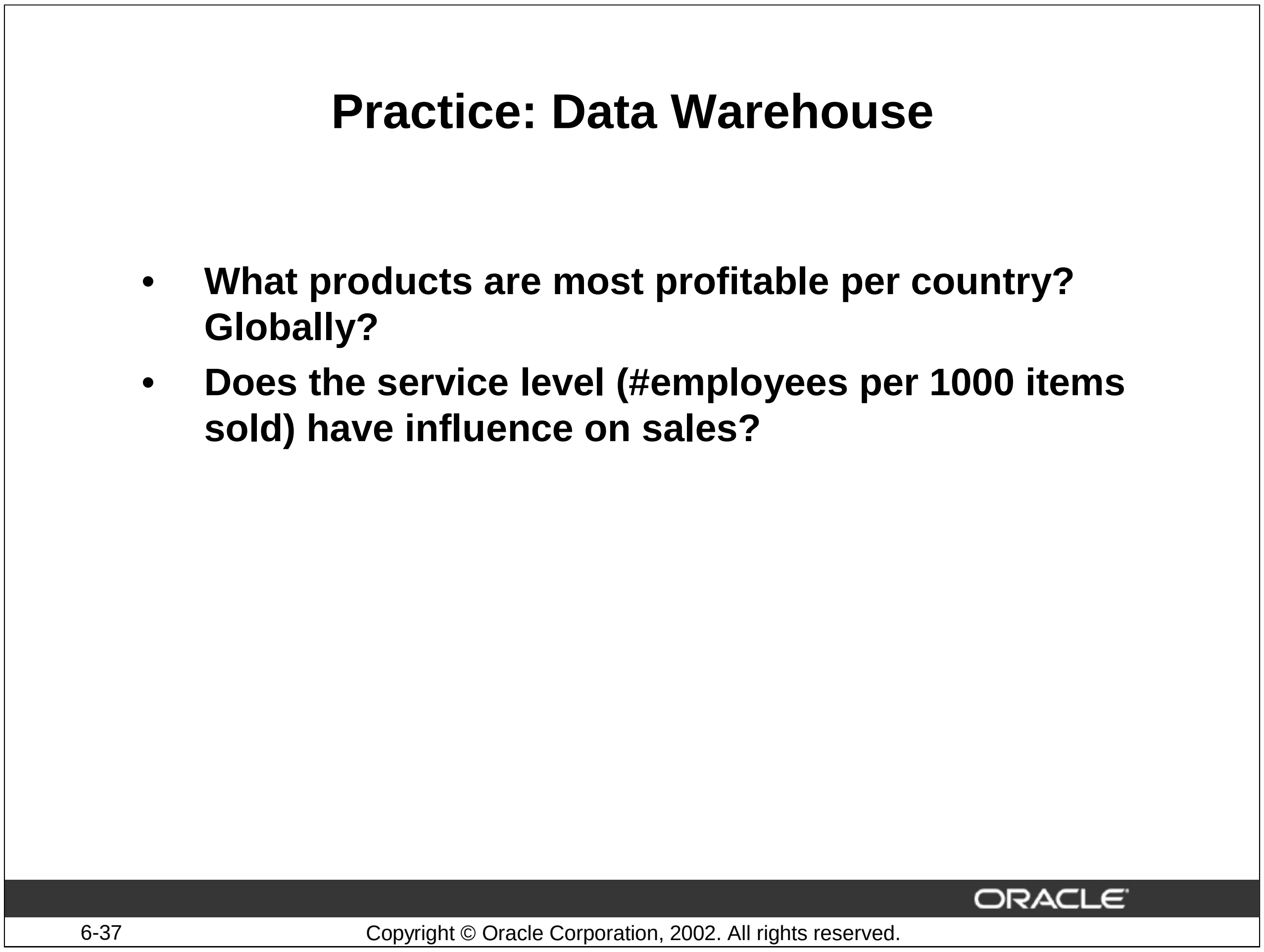
In this practice you create a conceptual data model for a data warehouse for Moonlight Coffees Inc.

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.

Your Assignment

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



Practice: Data Warehouse

- **What products are most profitable per country? Globally?**
- **Does the service level (#employees per 1000 items sold) have influence on sales?**

6-37

Practice 6-2: Data Warehouse

Goal

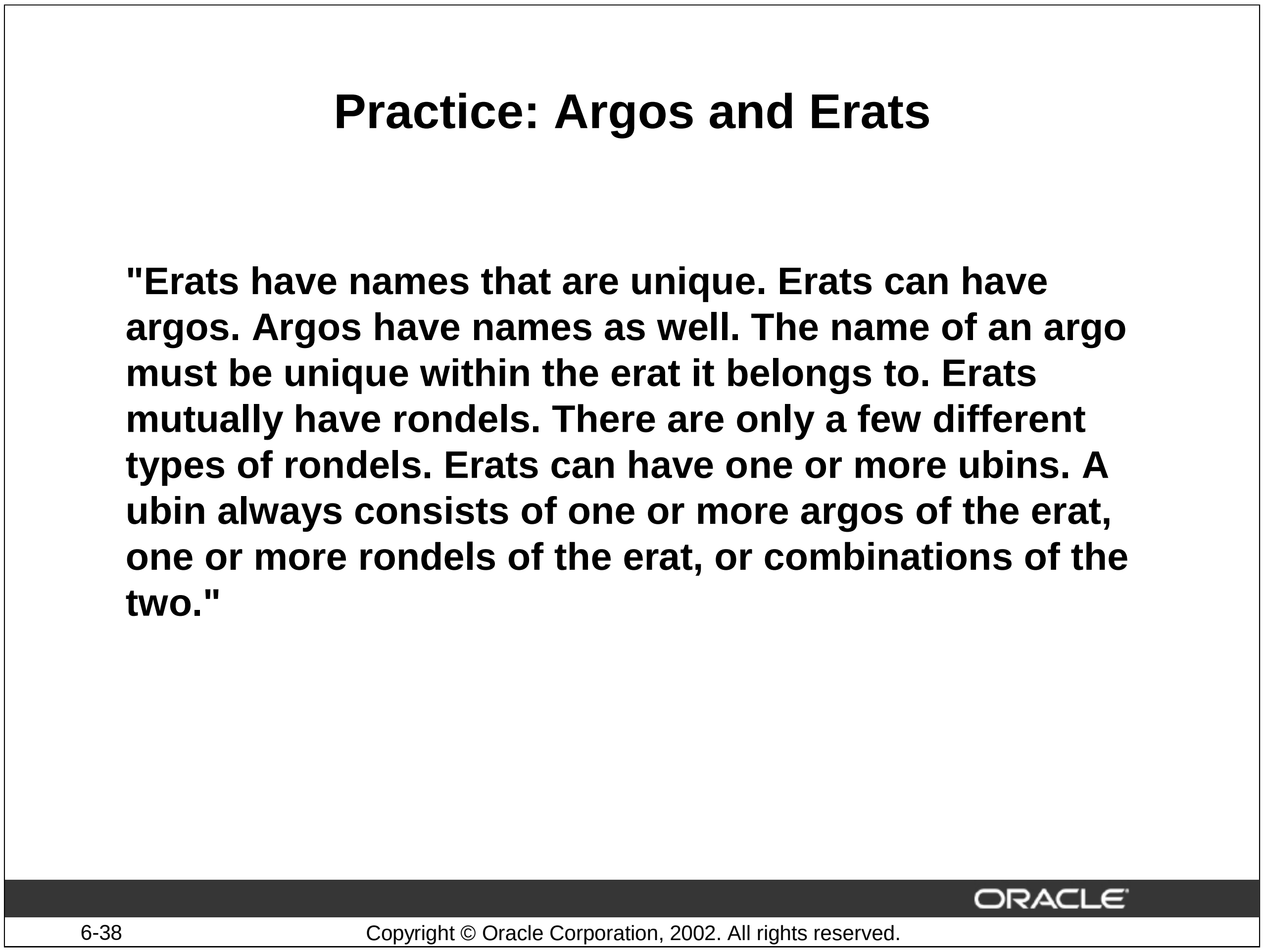
In this practice you create a conceptual data model for a data warehouse for Moonlight Coffees Inc.

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.

Your Assignment

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



Practice: Argos and Erats

"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."

ORACLE

6-38

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.

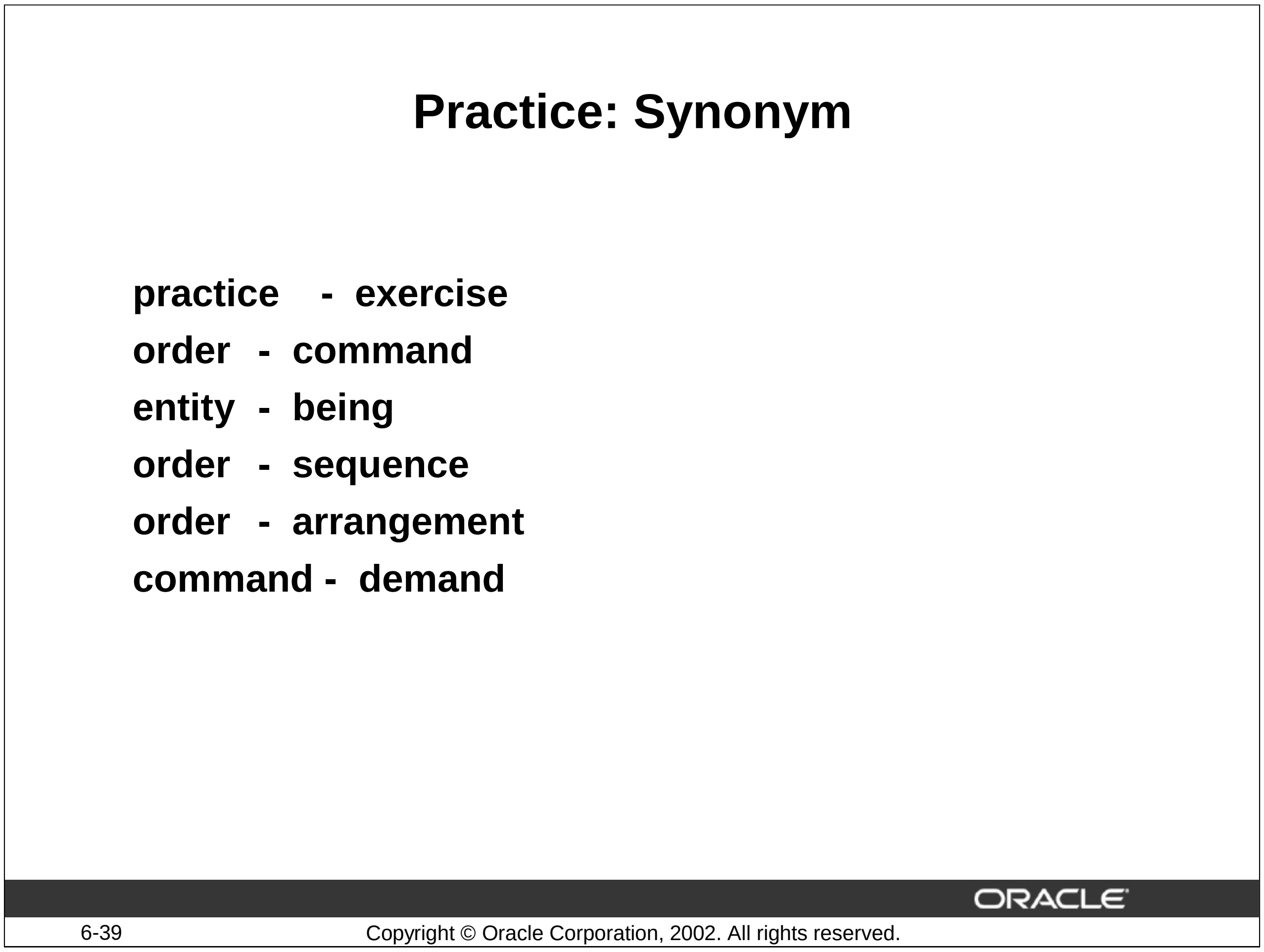
This practice helps to increase your awareness of this.

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.



Practice: Synonym

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

Practice 6-4: Synonym

Scenario

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

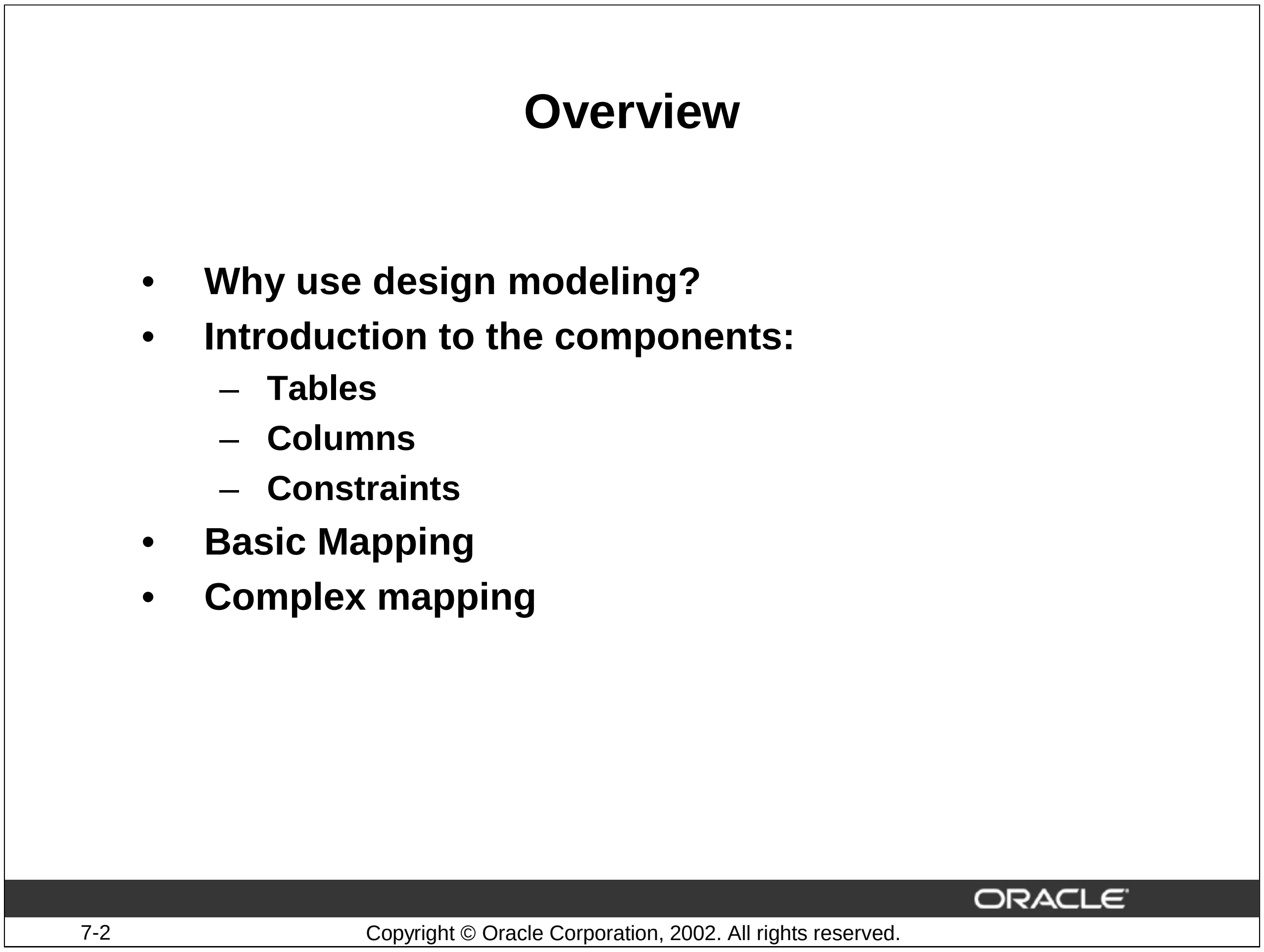
Your Assignment

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

7

Mapping the Entity Model

ORACLE



Overview

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

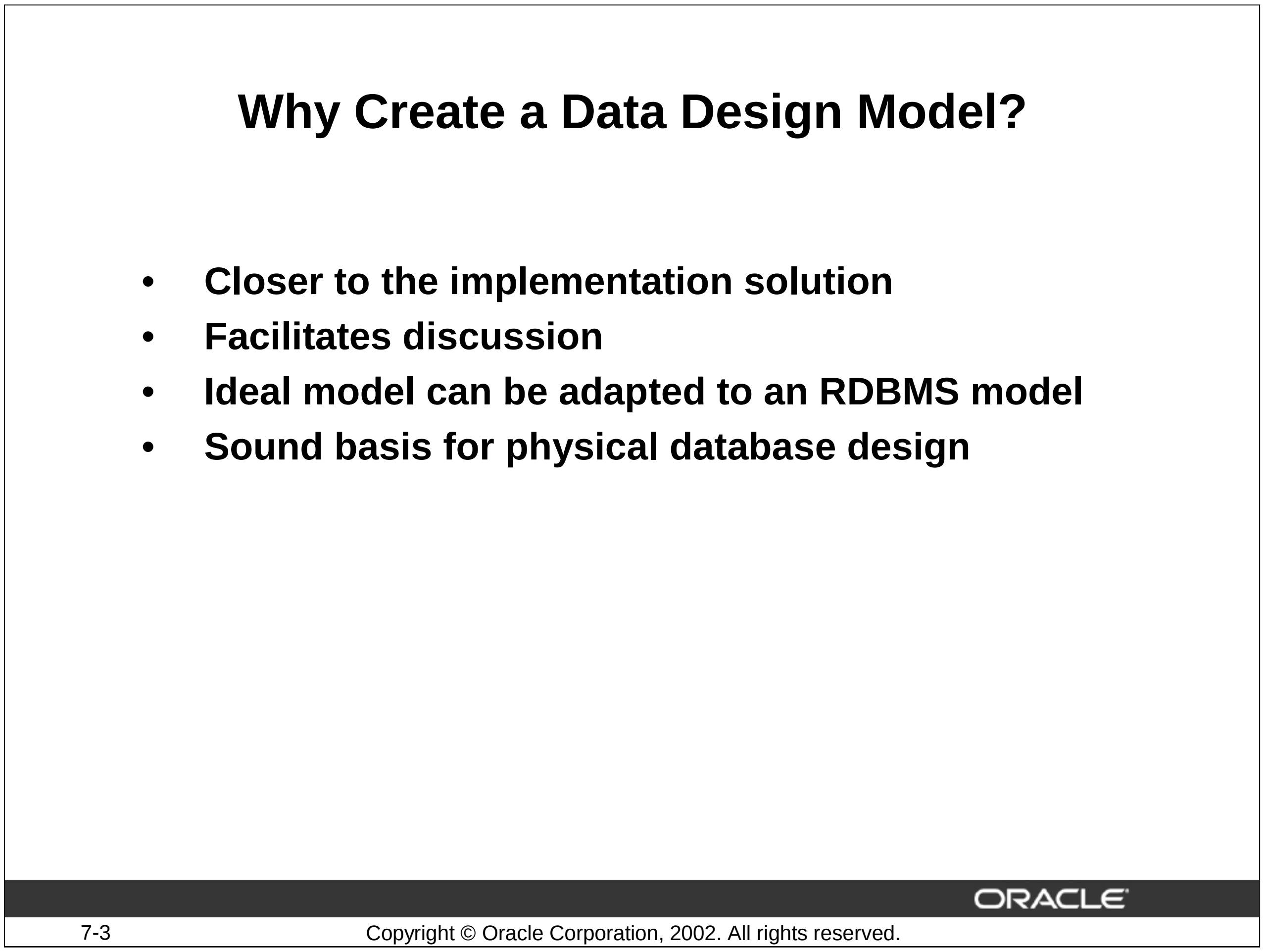
Introduction

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.

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 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

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.

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.

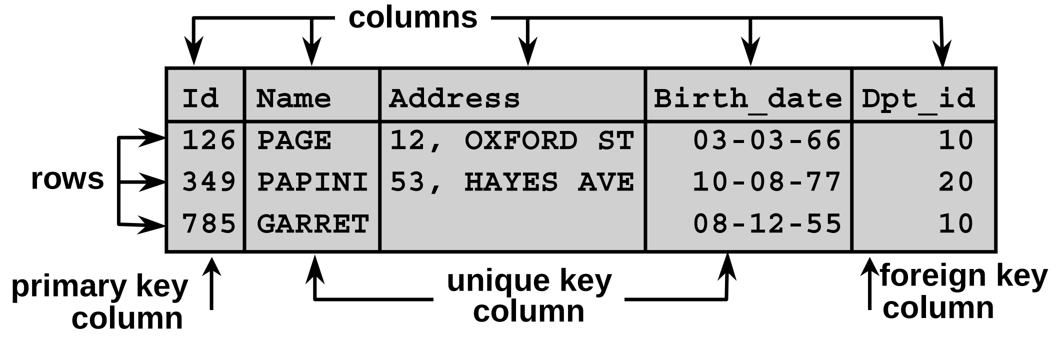
Transforming the ER model, creates a “first-cut” database design. This first-cut design is intended to serve as a new basis for defining the physical implementation of the database.

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

A data design model can be derived from an ER model (which is the flow down approach we advise and follow in this course), but it could also be created directly from an existing database.

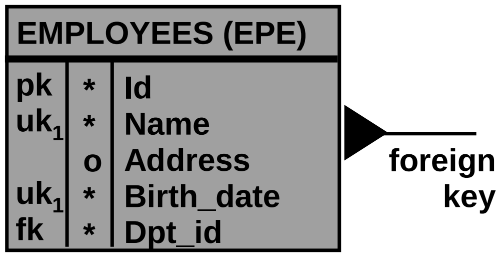
Presenting Tables

Table: EMPLOYEES



Id	Name	Address	Birth_date	Dpt_id
126	PAGE	12, OXFORD ST	03-03-66	10
349	PAPINI	53, HAYES AVE	10-08-77	20
785	GARRET		08-12-55	10

Table diagram: EMPLOYEES



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.

Presenting Tables

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.

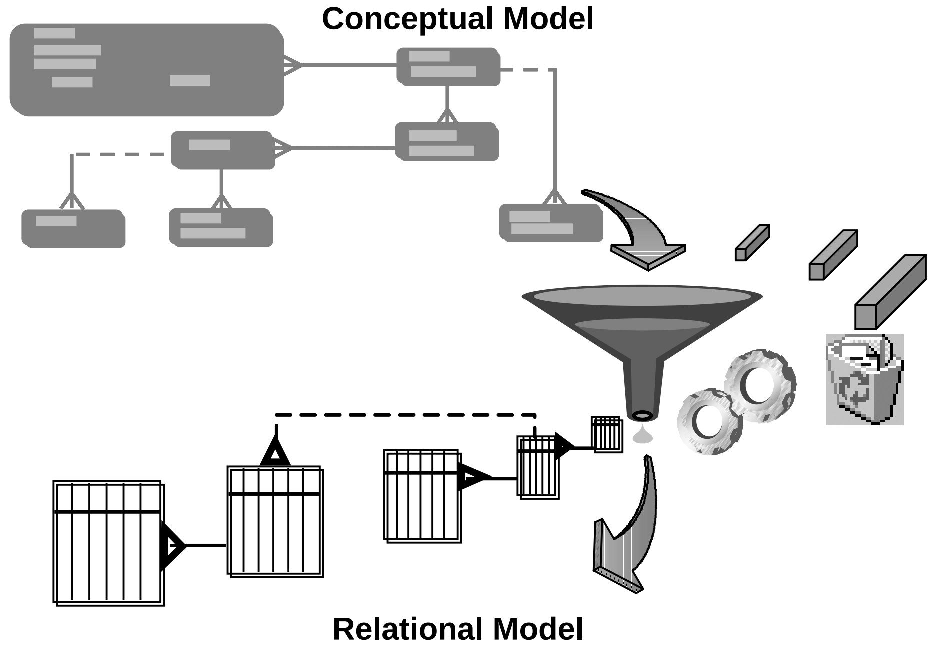
Unique keys:

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



Transformation Process

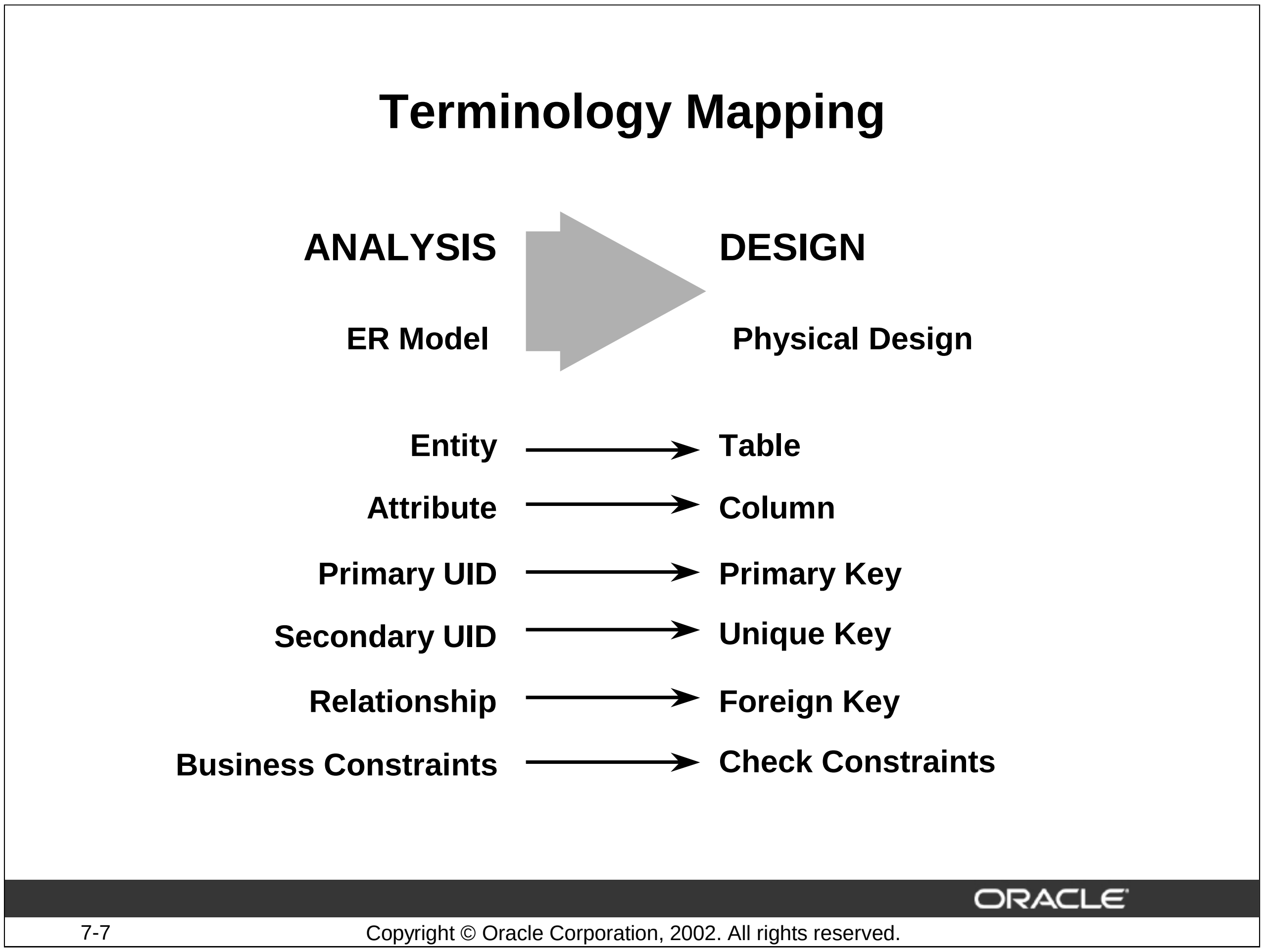
Using transformation rules you create a new model based on the conceptual model.

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

ANALYSIS		DESIGN
ER Model		Physical Design
Entity	$\longrightarrow$	Table
Attribute	$\longrightarrow$	Column
Primary UID	$\longrightarrow$	Primary Key
Secondary UID	$\longrightarrow$	Unique Key
Relationship	$\longrightarrow$	Foreign Key
Business Constraints	$\longrightarrow$	Check Constraints

7-7

Terminology Mapping

Changing from one world to another 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.

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

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.

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.

Naming Convention

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.

Naming Convention

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

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

Naming Restrictions with Oracle

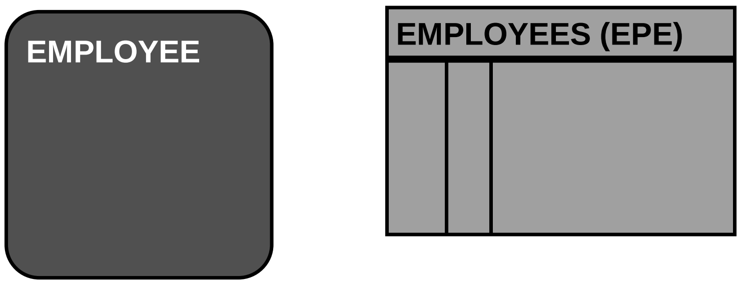
Each RDBMS can have its own naming restrictions. You need to know if the convention you decide to use is compatible with it.

- You can use any alpha-numeric character for naming 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 for Entities

1 - Entities

Table Name: EMPLOYEES
Short Name: EPE



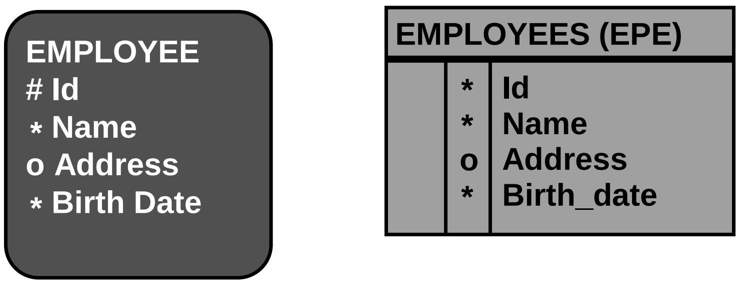
Basic Mapping for Attributes

1 - Entities

2 - Attributes

Table Name: EMPLOYEES

Short Name: EPE



Basic Mapping for Unique Identifiers

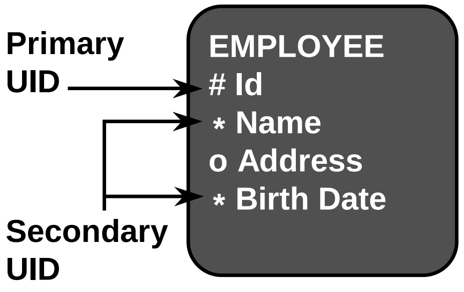
1 - Entities

2 - Attributes

3 - Unique identifiers

Table Name: EMPLOYEES

Short Name: EPE



EMPLOYEES (EPE)		
pk	*	Id
uk_1	*	Name
	o	Address
uk_1	*	Birth_date

Basic Mapping

Entity Mapping

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

- Transform entities into tables using your own naming convention or the one previously described.

In this exam the entity EMPLOYEE produces a table name EMPLOYEES and a short name EPE.

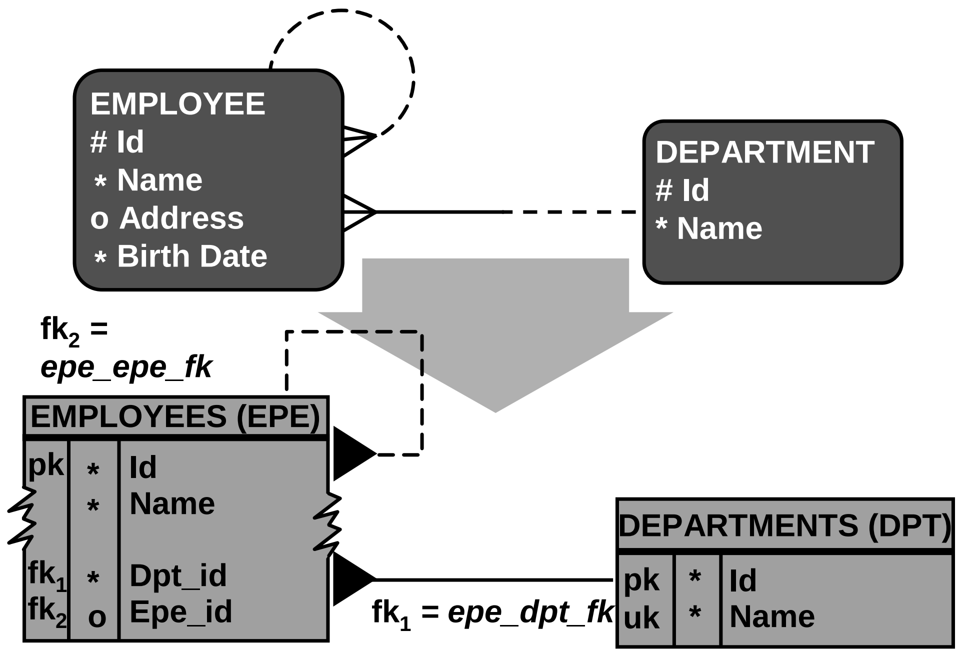
- Use a box to represent tables on a diagram.
- Each attribute creates a column in the table and the characteristics such as mandatory or optional have to be kept for each column. Using the same notation “*” or “o” facilitates recognition of these characteristics on a diagram.
- All unique identifiers 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.

Basic Mapping

Secondary Unique Identifiers

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



Rules for Relationships

Foreign Key Columns:

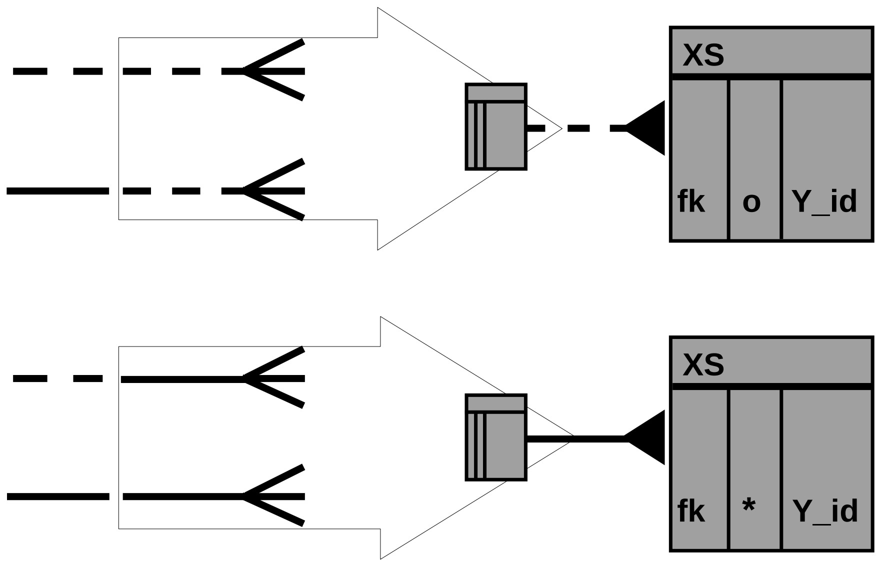
A relationship creates one or more foreign key columns in the table at the many side. Using previous naming rules, the name of this foreign key column is Dpt_id for the relationship with Department and Epe_id for the recursive relationship. This ensures that column names such as Id, coming from different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

Mapping 1:m Relationships



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.

Relationship Mapping

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.

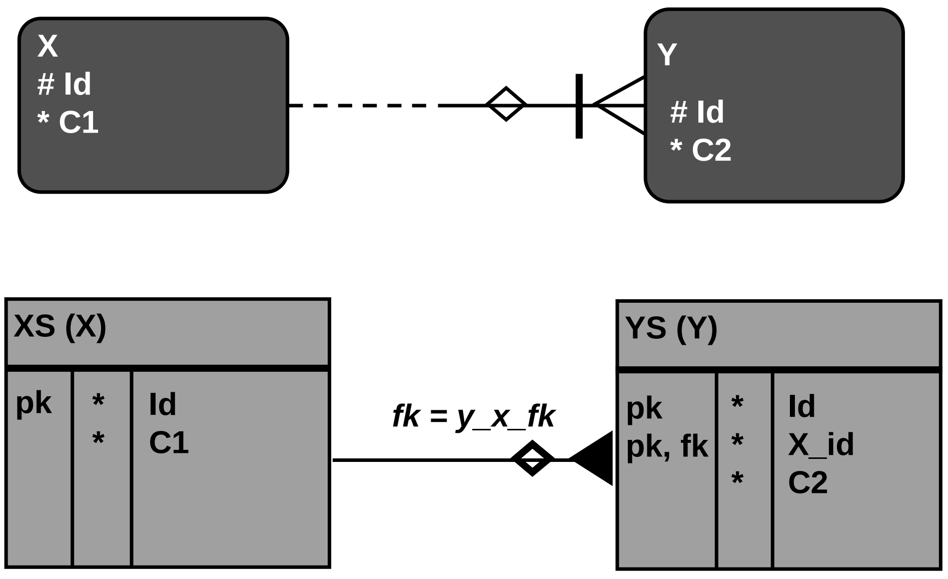
You can implement code to check this on the server side or on the client side. In an Oracle environment this was usually done at the client side. Since Oracle 8, on the server side Oracle offers implementation possibilities that were not available in previous releases.

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 Barred and Nontransferable Relationships



Relationship Mapping

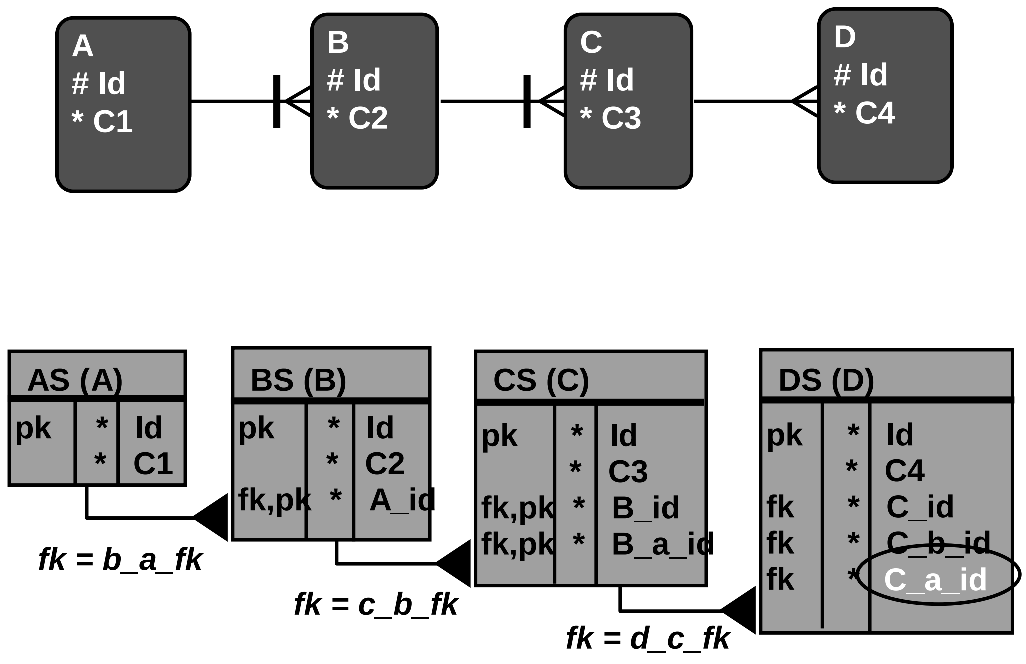
Mapping of Nontransferable Relationships

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 Cascade Barred Relationships



Relationship Mapping

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.

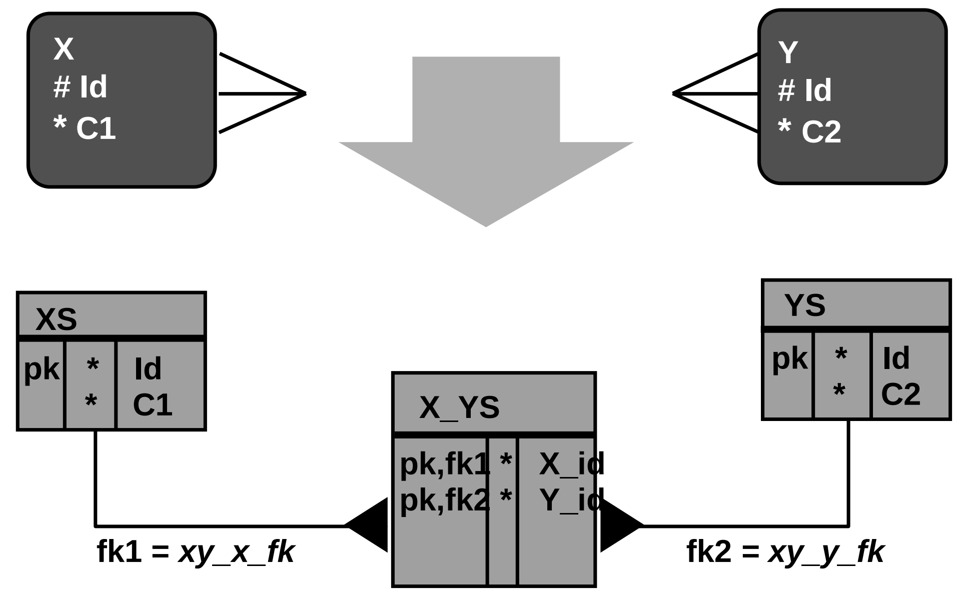
The usual choice for the foreign key column names is:

<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 m:m Relationships



Relationship Mapping

Mapping of Many-to-Many Relationships

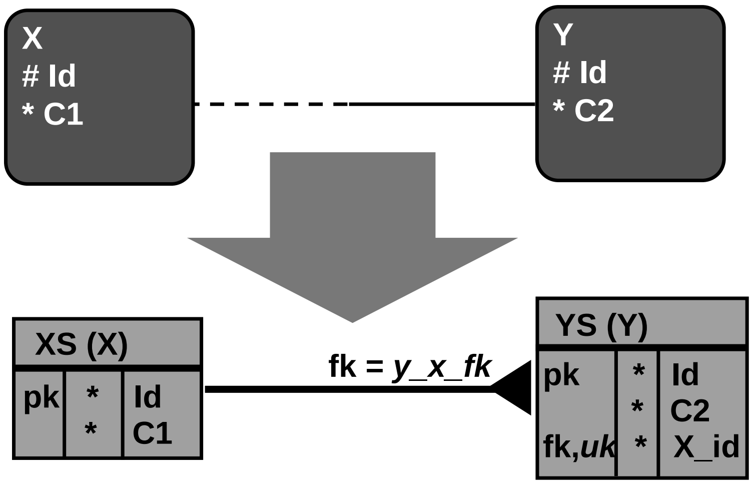
When transforming a many-to-many relationship, you create an intersection table.

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 1:1 Relationships



Choose which side for FK for other cardinalities

Relationship Mapping

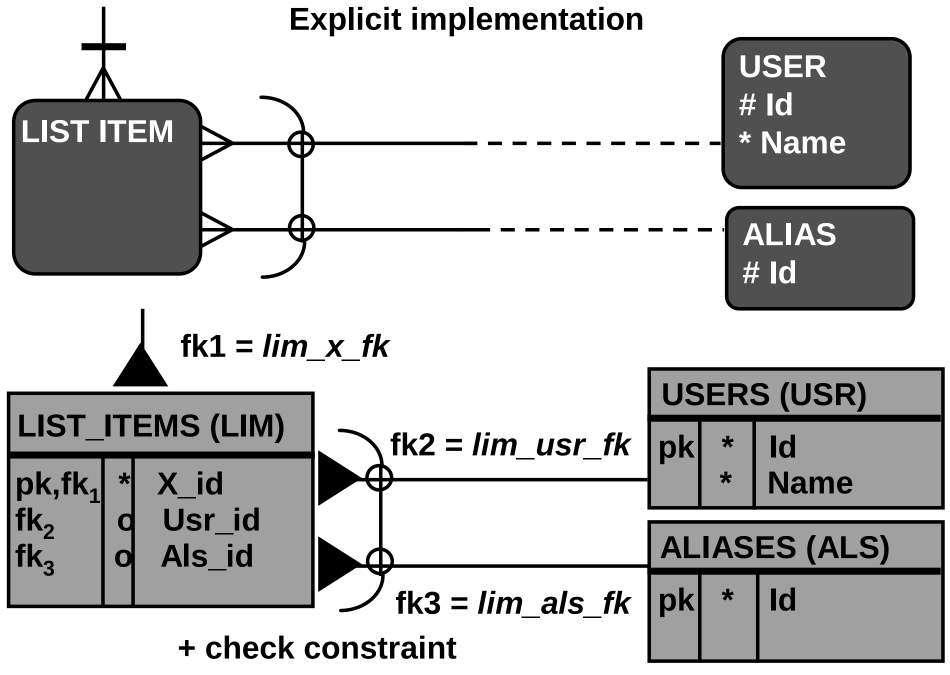
Mapping of One-to-One Relationships

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 Arcs



Relationship Mapping

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)

Relationship Mapping

Implementing Arcs

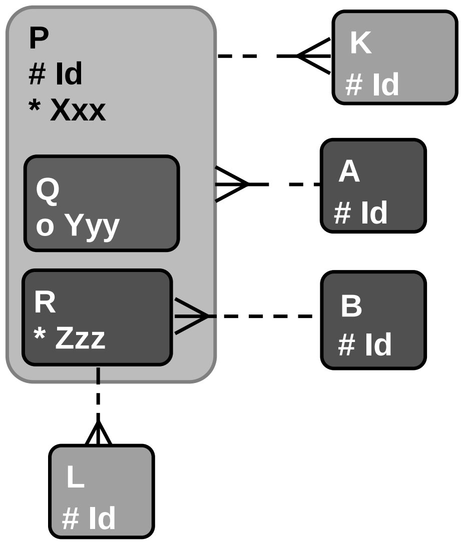
An other syntax that is often used:

- 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 Subtypes

Variety of implementation choices



- **Supertype**
- **Subtype**
- **Both Supertype and Subtype ("Arc")**

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

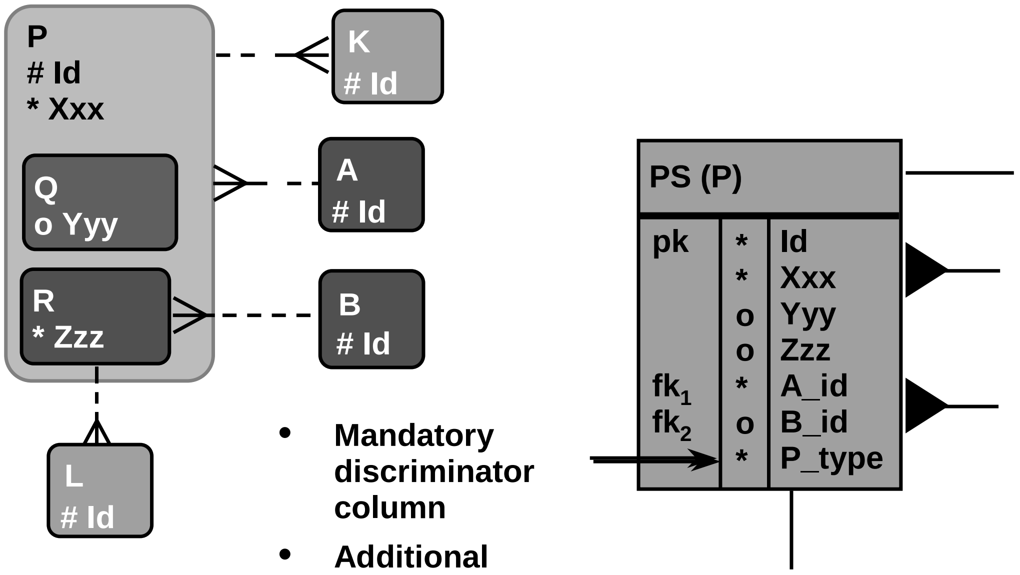
This choice produces one single table for the implementation of the entities P, Q, and R. The supertype implementation is also called single (or one) table implementation.

Mapping of Subtypes

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.

Supertype Implementation



- **Mandatory discriminator column**
- **Additional constraints**

Mapping of Subtypes

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.

Mapping of Subtypes

- 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.

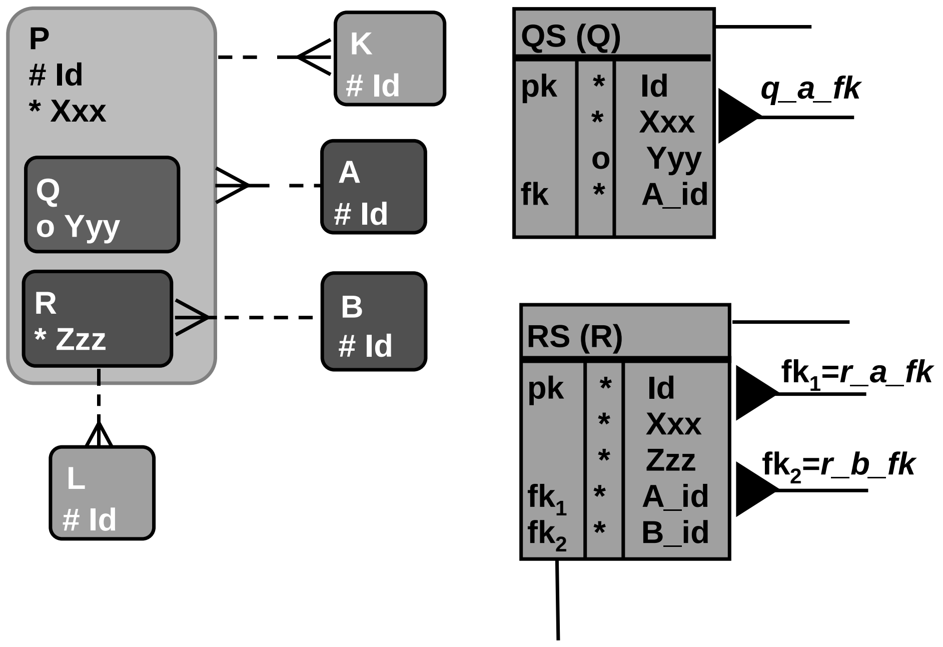
The supertype table plus subtype views is an elegant and appropriate implementation and should be considered as first choice.

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



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.

Subtype Implementation

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.

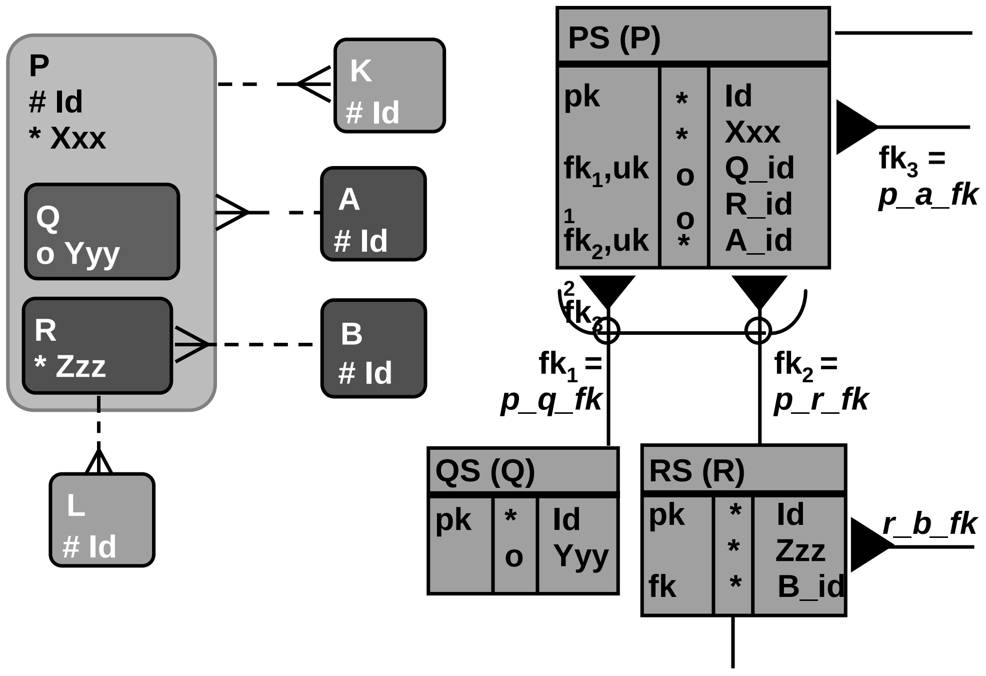
Consequences for Tables Based on K and L

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.

Supertype and Subtype (Arc) Implementation



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.

Both Supertype and Subtype “Arc” Implementation

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

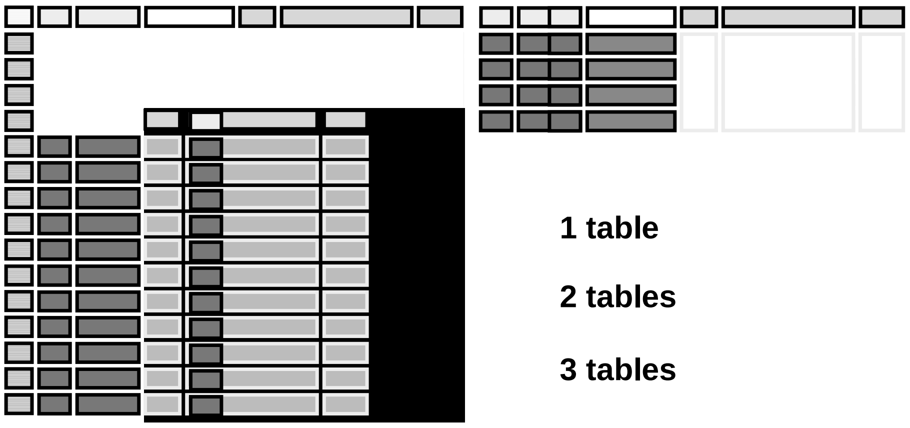
This solution performs a double partitioning. It is used relatively rarely, but could be appropriate when:

- 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



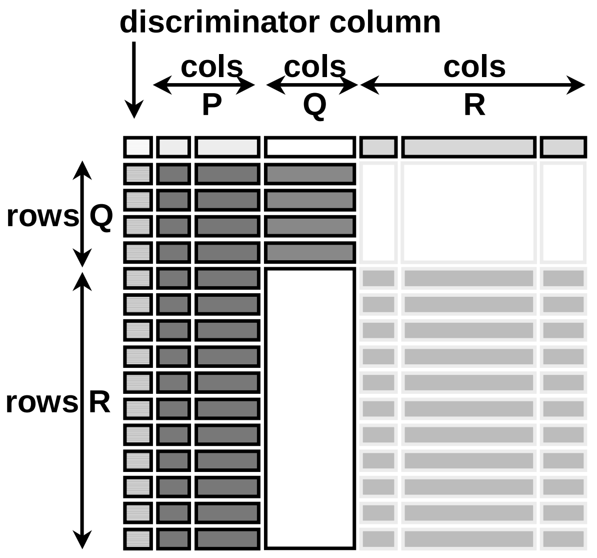
1 table

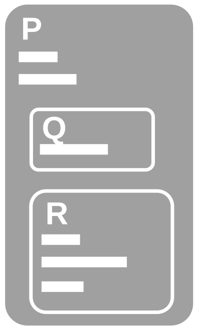
2 tables

3 tables

ORACLE

Storage Implication
Supertype Implementation





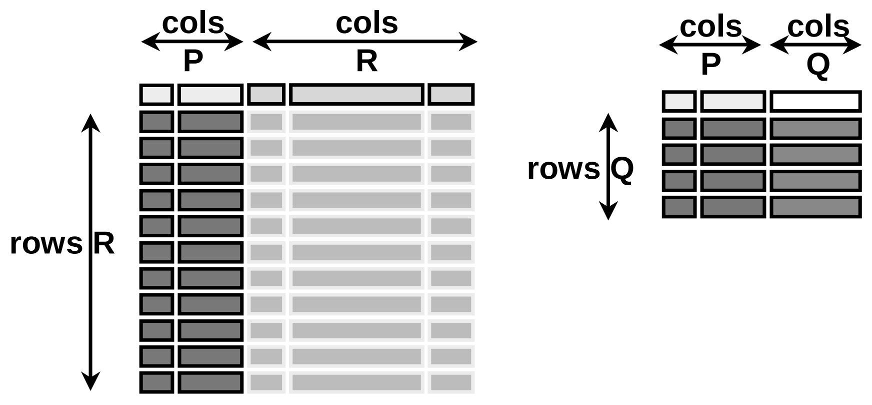
Storage Implication

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.

Storage Implication Subtype Implementation

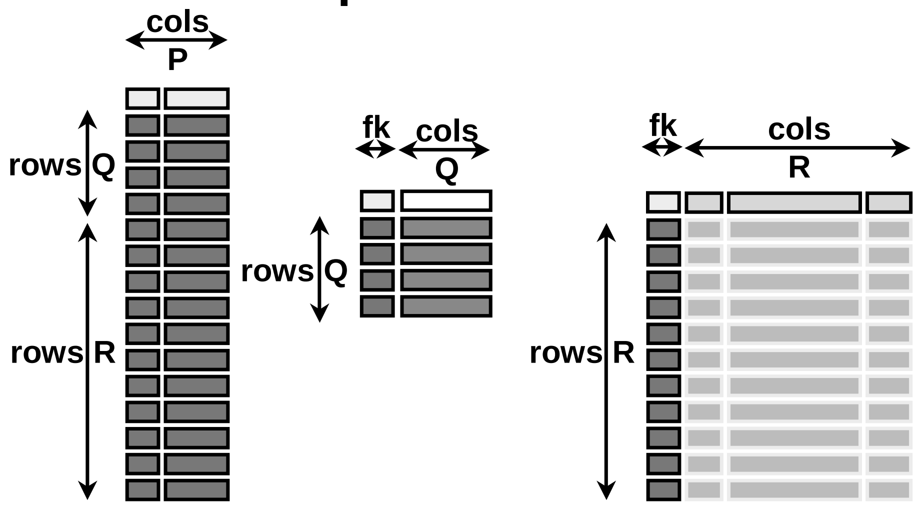


Storage Implication

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.

Storage Implication Supertype and Subtype (Arc) Implementation



Storage Implication

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.

Summary

- Relational concepts
- Naming rules convention
- Basic mapping
- Complex mapping

Summary

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.

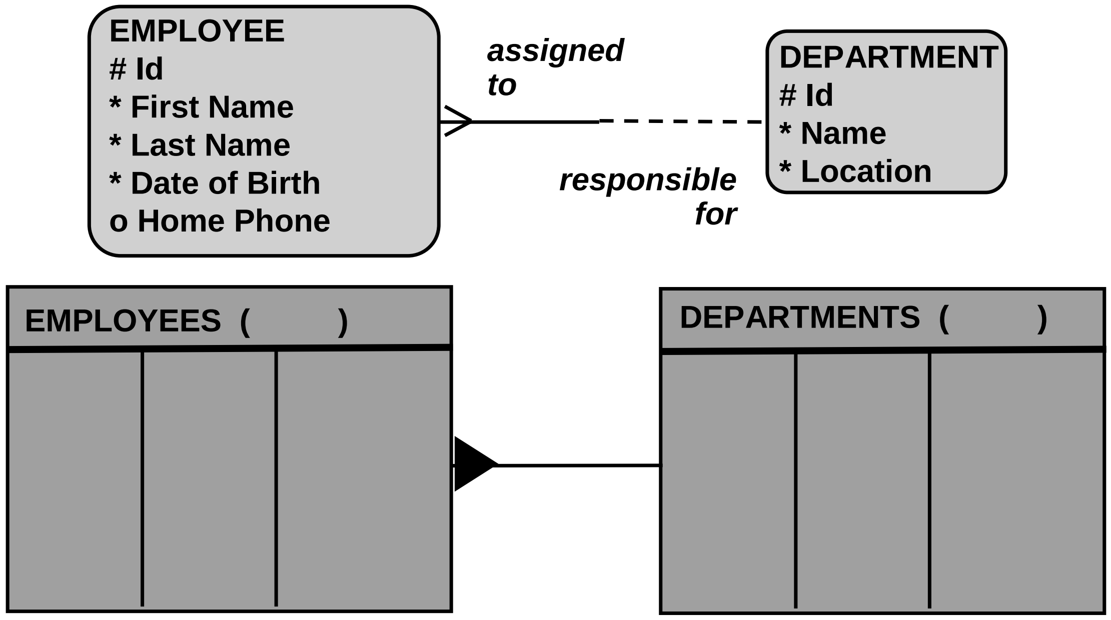
You have seen how to map basic elements from an ER model such as entities and relationships. You can do this very simply. There are also complex structures which require decisions on how to transform them. Some ER model elements can only be implemented by coding check constraints or database triggers. These are specific to Oracle and not part of the ISO standard for relational databases.

Practice

- Mapping basic Entities, Attributes and Relationships
- Mapping Supertype
- Quality Check
  Subtype Implementation
- Quality Check
  Supertype and Subtype (Arc) Implementation
- Mapping Primary Keys and Columns

ORACLE

Practice: Mapping basic Entities, Attributes and Relationships



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.

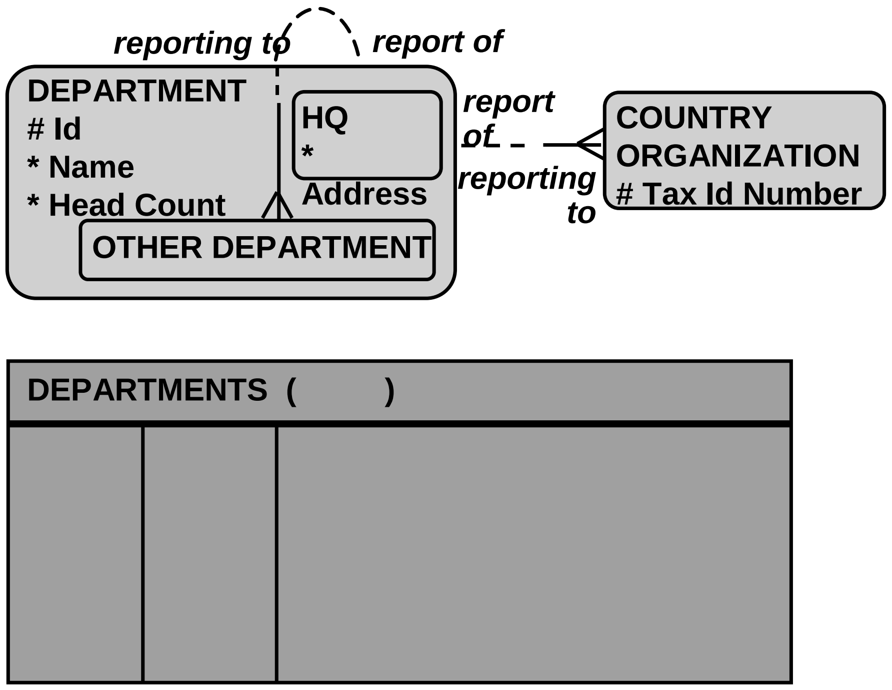
Scenario

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

- Map both entities to tables and all attributes to columns.
- Map relationships to foreign keys columns and mark as (fk).
- Map all optionality tags to not nulls (*).
- Map UID tags to primary keys (pk).
- 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: Mapping Supertype



Practice 7-2: Mapping Supertype

Goal

In this practice, you create a complex mapping and test your understanding of the transformation process.

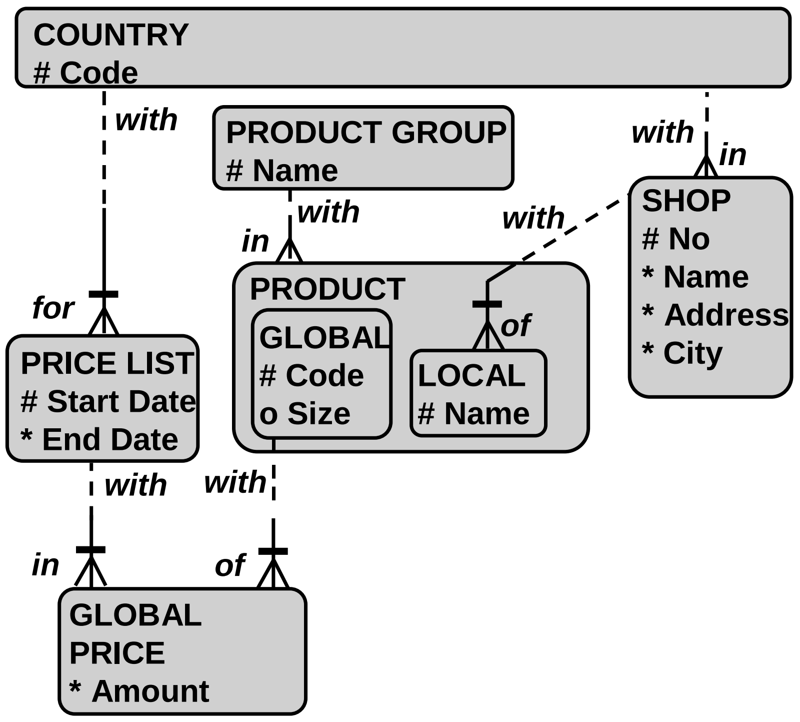
Scenario

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

- What would have been the rationale of this choice?
- 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.

Partial ER model Moonlight



ORACLE

Practice 7-3: Quality Check Subtype Implementation

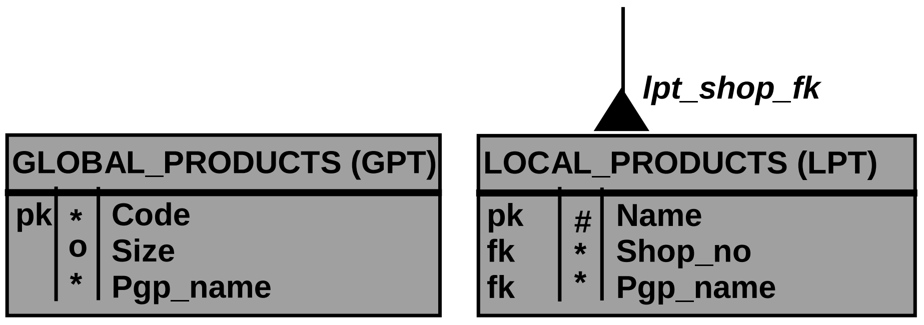
Goal

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.

Practice: Quality Check Subtype Implementation

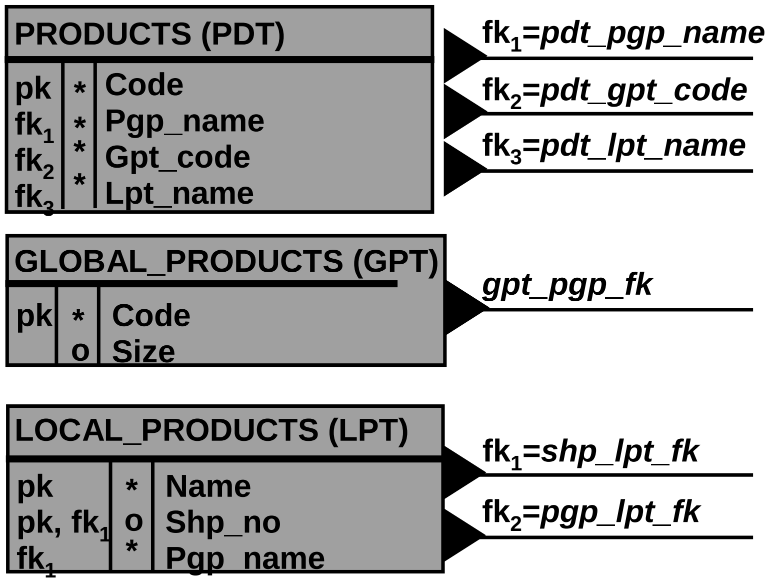


Practice 7-3: Quality Check Subtype Implementation

Your Assignment

Perform a quality check on the proposed subtype implementation of entity PRODUCT.

Practice: Quality Check Arc Implementation



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.

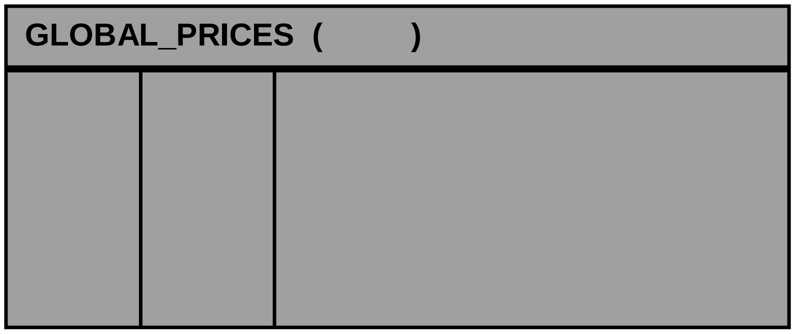
Scenario

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: Mapping Primary Keys and Columns



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.

8

Denormalized Data

Overview

- Denormalization
- Benefits
- Types of denormalization

Introduction

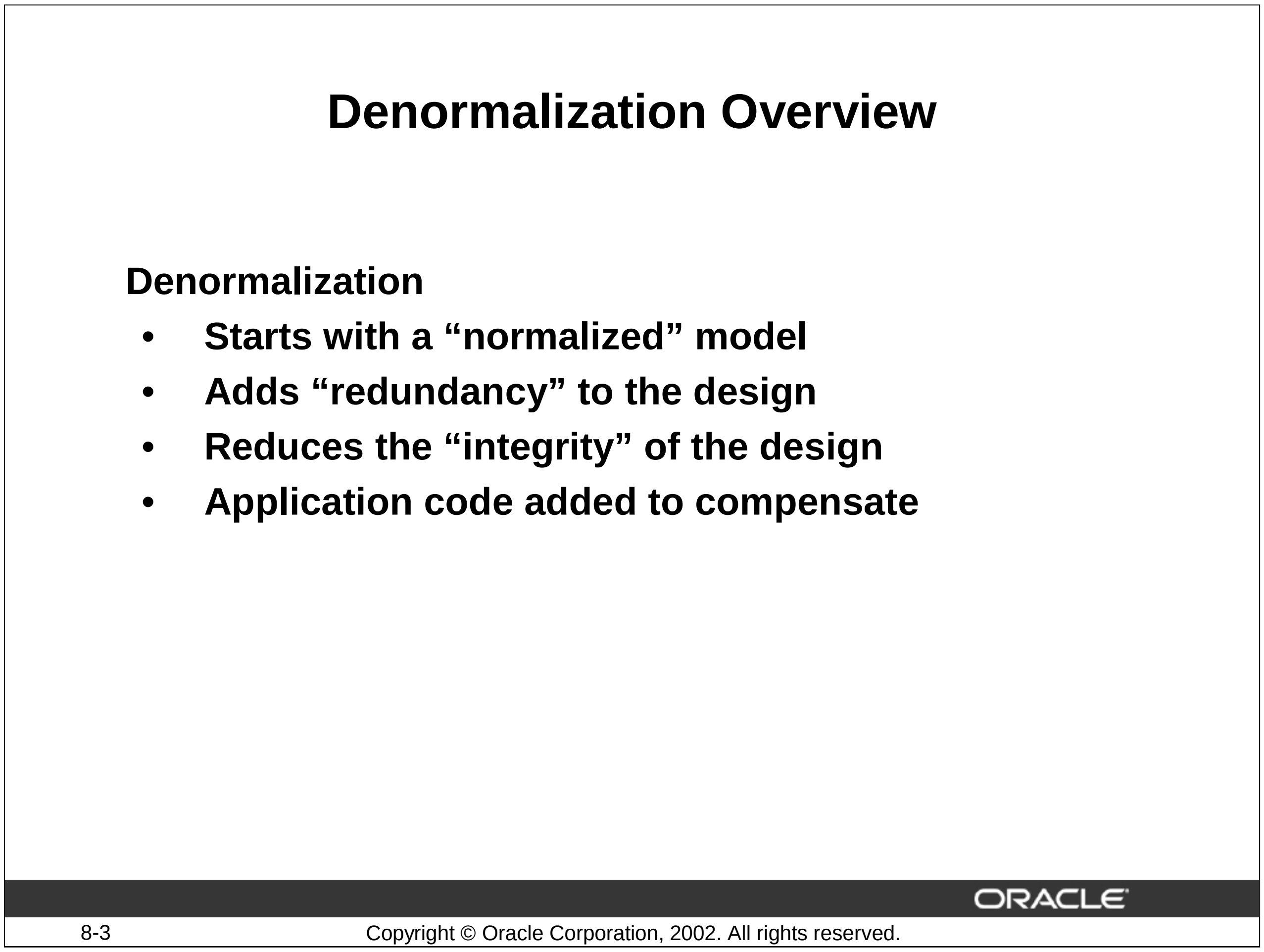
Lesson aim

This lesson shows you the most common types of denormalization with examples.

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



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 Copyright © Oracle Corporation, 2002. All rights reserved.

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.

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.
- 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, and what application code was added to compensate for the denormalization.

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

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.

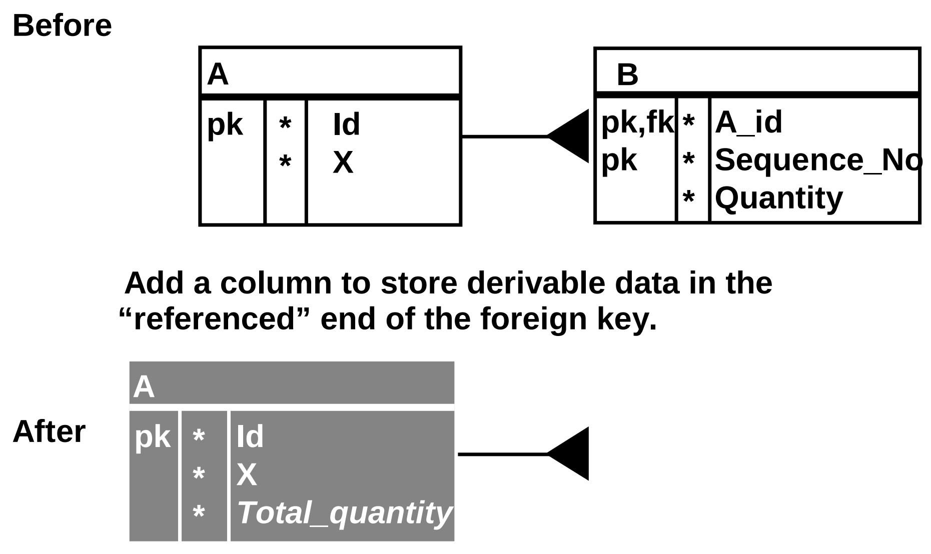
The following topics are covered:

- 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



Add a column to store derivable data in the "referenced" end of the foreign key.

8-5 Copyright © Oracle Corporation, 2002. All rights reserved.

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:

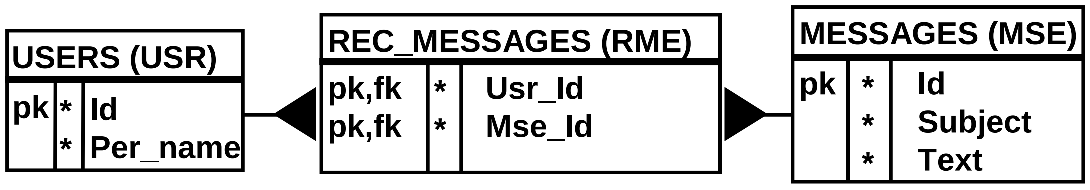
- Source values do not need to be looked up every time the derivable value is required
- The calculation does not need to be performed during a query or report

Disadvantages:

- DML against the source data will require recalculation or adjustment of the derivable data
- Data duplication introduces the possibility of data inconsistencies

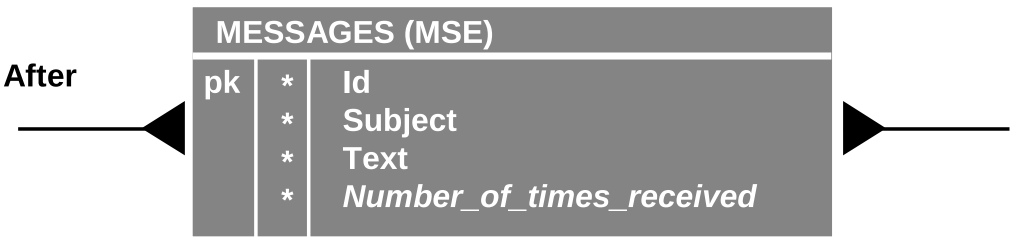
EMail Example of Storing Derivable Values

Before



Store derivable column in the 'referenced' end of the foreign key.

After



E-mail Example of Storing Derivable Values

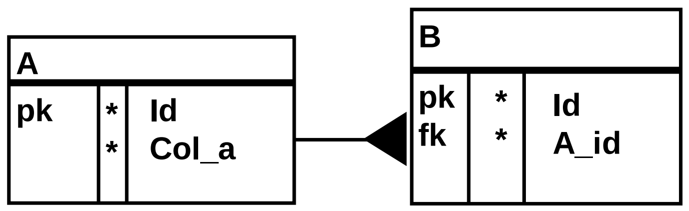
When a message is delivered to a recipient, the user only receives a pointer to that message, which is recorded in RECEIVED_MESSAGES. The reason for this, of course, is to prevent the mail system from storing a hundred copies of the same message when one message is sent to a hundred recipients.

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.

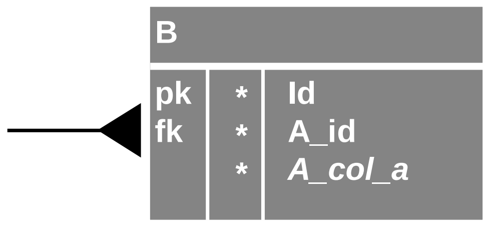
Pre-Joining Tables

Before



Add the non_key column to the table with the foreign key.

After



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

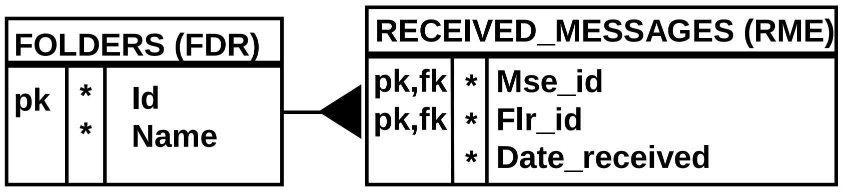
- Time-consuming joins can be avoided
- Updates may be postponed when stale data is acceptable

Disadvantages

- Extra DML needed to update original nondenormalized column
- Extra column and possibly larger indices require more working space and disk space

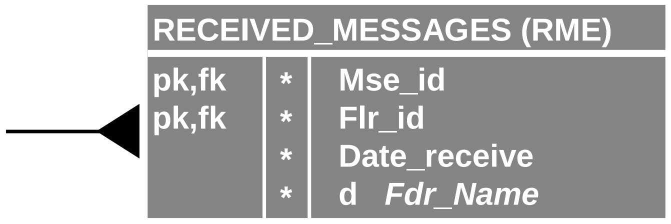
EMail Example of Pre-Joining Tables

Before



Create a table with all the frequently queried columns.

After



Pre-Joining Tables Example

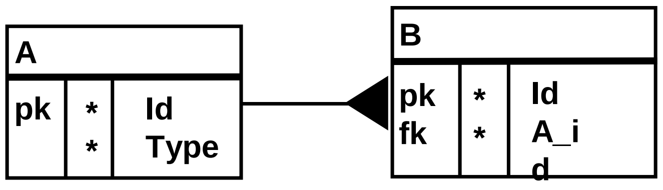
Suppose users often need to query RECEIVED_MESSAGES, using the name of the folder where the received message is filed. In this case it saves time when the name of the folder is available in the RECEIVED_MESSAGES table.

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.

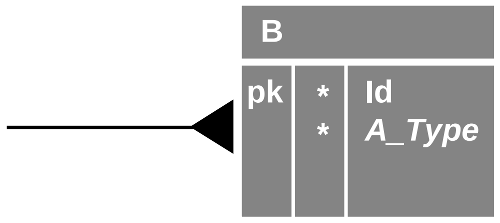
Hard-Coded Values

Before



Remove the foreign key and hard code the allowable values and validation in the application.

After



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

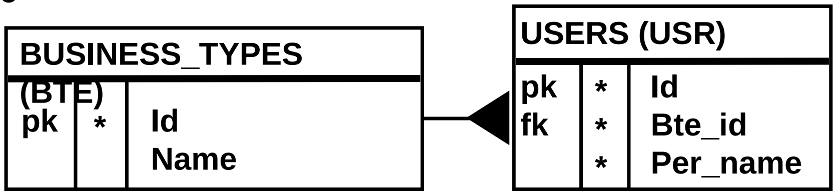
- Avoids implementing a look-up table
- Avoids joins to a look-up table

Disadvantages

- Changing look-up values requires recoding and retesting

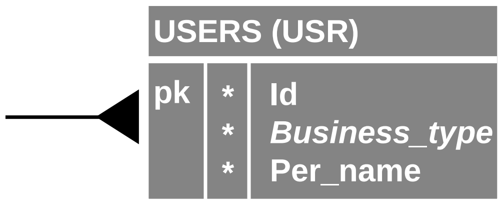
Email Example of Hard-Coded Values

Before



Hard code the allowable values and validation in the application.

After



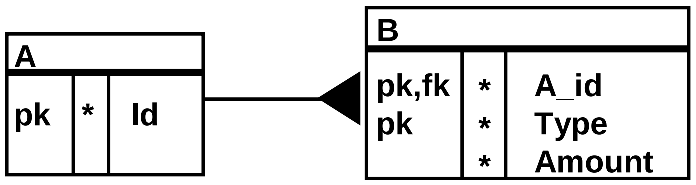
Hard-Coded Values Example

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.

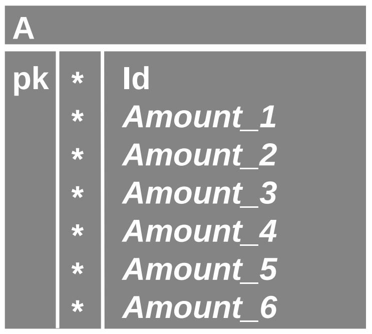
Keeping Details with Master

Before



Add the repeating detail columns to the master table.

After



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

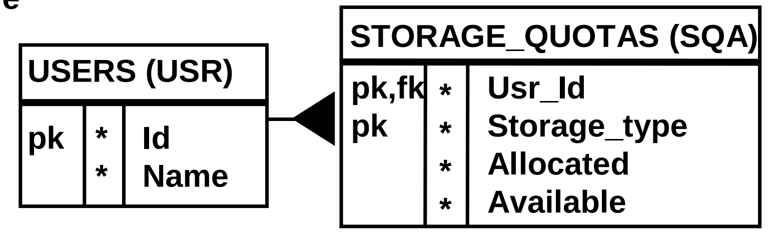
- When the number of detail records for all masters is fixed and static
- When the number of detail records multiplied by the number of columns of the detail is small, say less than 30

Advantages

- No joins are required
- Saves space, as keys are not propagated

EMail Example Keeping Detail with Master

Before



Add the repeating detail columns to the master table.

After

USERS (USR)		
pk	*	Id
	*	Name
	*	*Message_Quota_Allocated*
	*	*Message_Quota_Available*
	*	*File_Quota_Allocated*
	*	*File_Quota_Available*

Keeping Details With Master (Example)

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

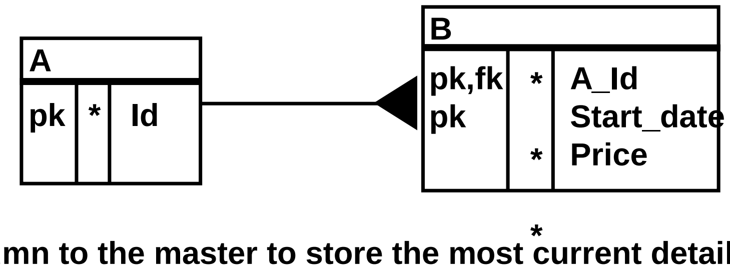
Suppose each e-mail user is assigned two quotas—one for messages and one for files. The amount of each quota is different, so both have to be tracked individually. The quota does not change very frequently. To be relationally pure, we would create a two-record STORAGE_TYPES table and a STORAGE_QUOTAS 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.

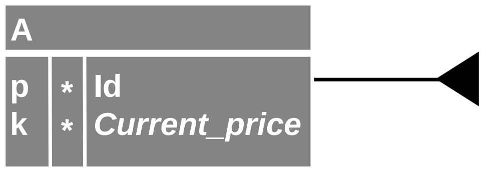
Repeating Current Detail with Master

Before



Add a column to the master to store the most current details.

After



8-13 Copyright © Oracle Corporation, 2002. All rights reserved.

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. Additional code must be written to maintain the duplicated single detail value at the master record.

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

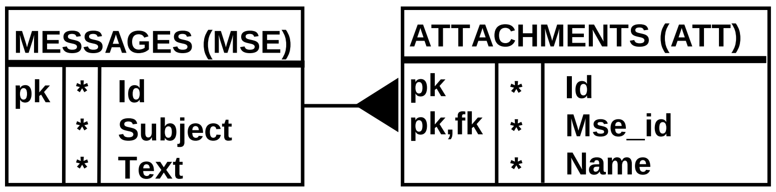
- No join is required for queries that only need the specific single detail

Disadvantages

- Detail value must be repeated, with the possibility of data inconsistencies

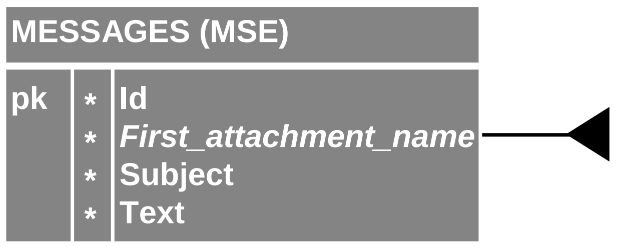
EMail Example of Repeating Single Detail with Master

Before



Add a column to the master to store the most current details.

After

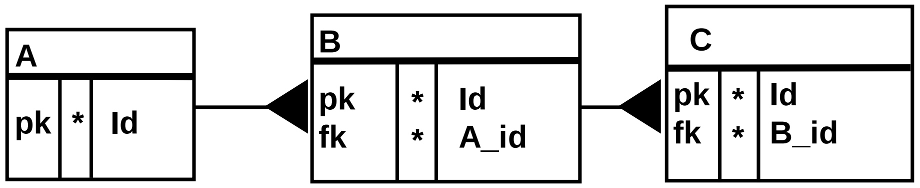


Repeating Single Detail with Master Example

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.

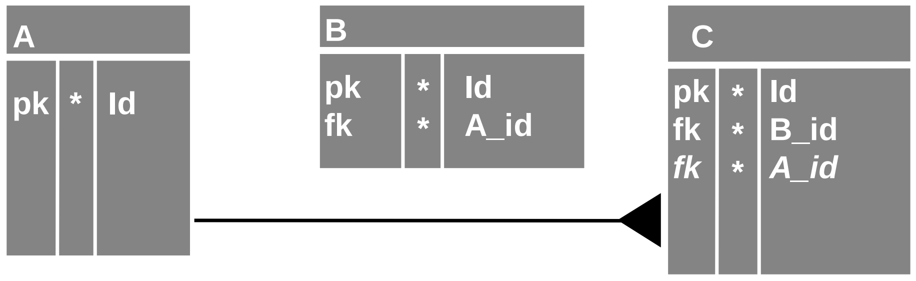
Short-Circuit Keys

Before



Create a new foreign key from the lowest detail to the highest master.

After



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

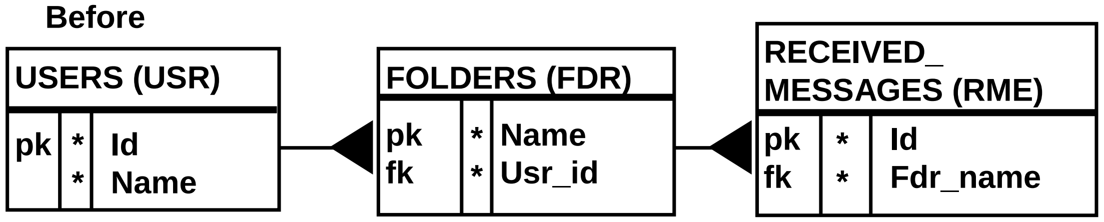
- Queries join fewer tables together

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.

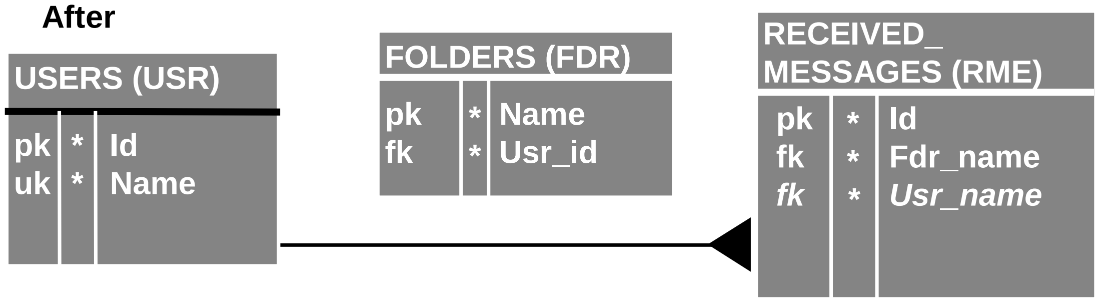
EMail Example of Short-Circuit Keys

Before



Create a new foreign key from the lowest detail to the highest master.

After



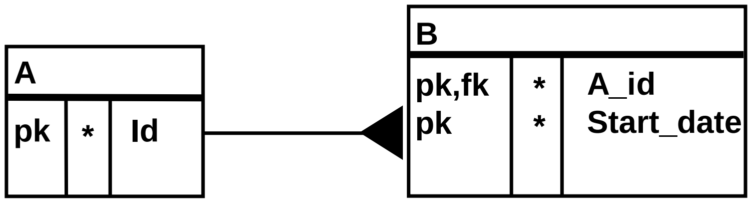
ORACLE

Short-Circuit Keys 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 been migrated to the RECEIVED_MESSAGES table, to provide information about USERS and RECEIVED_MESSAGES with one less, or no, table join.

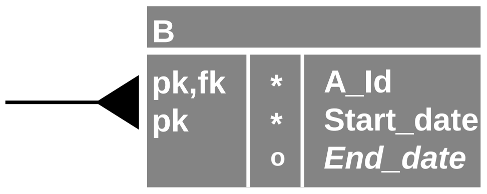
End Date Column

Before



Add an *end date* column to speed up queries so that they can use a *between* operator.

After



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

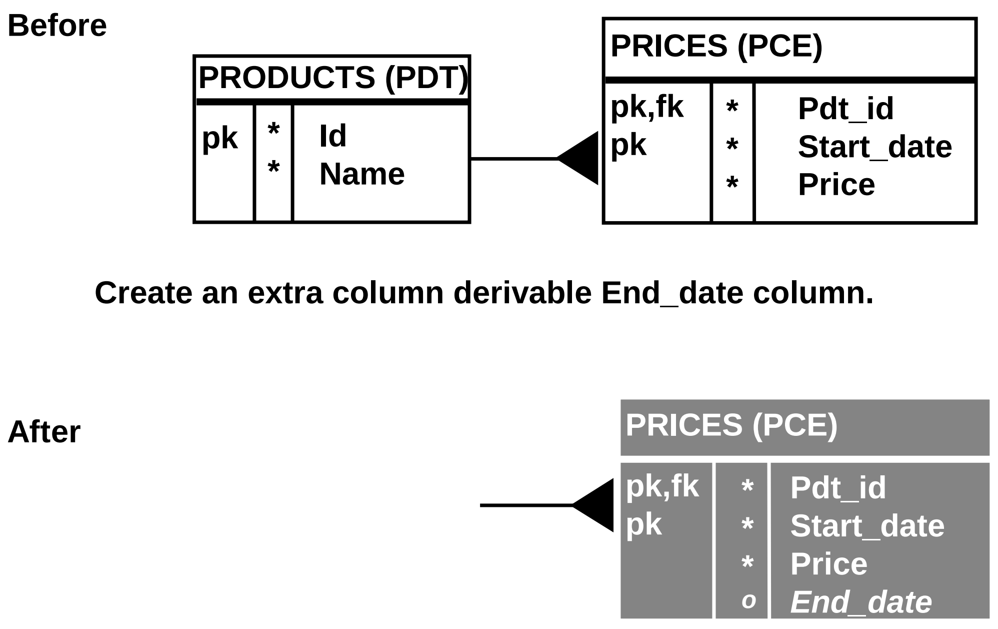
Advantages

- 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 of End Date Column



End Date Columns 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

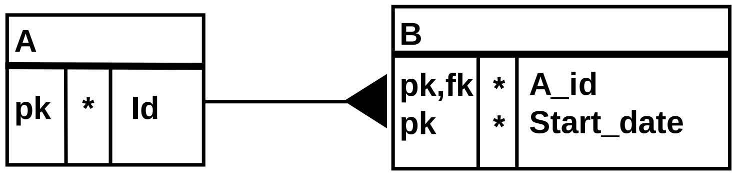
...WHERE pdt_id = ... AND sysdate between start_date and nvl(end_date, sysdate)

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.

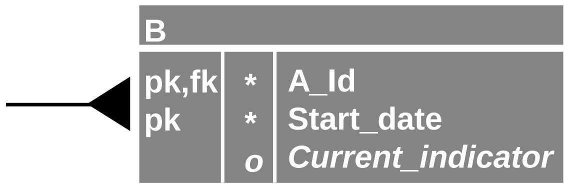
Current Indicator Column

Before



Add a column to represent the most current record in a long list of records.

After



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

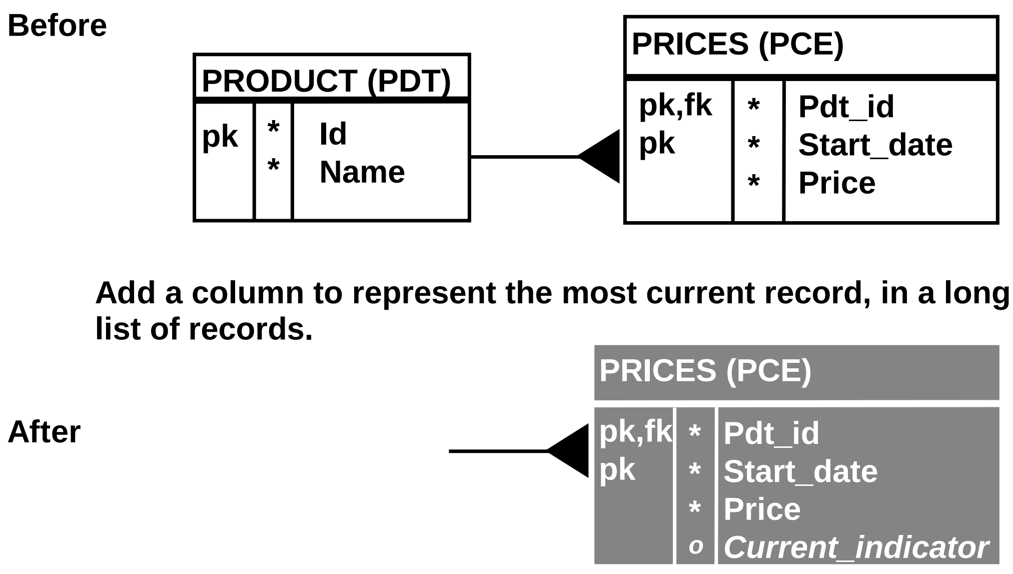
Advantages

- 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



Current Indicator Column Example

In the first table structure, when the current price of a product is needed, you need to query the PRICES table using:

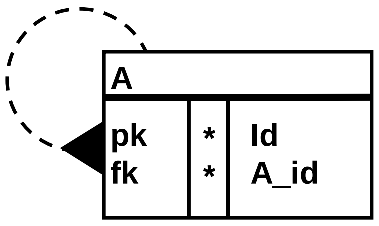
...WHERE pdt_id = ... AND start_date = (SELECT max(start_date) FROM prices WHERE start_date <= sysdate AND pdt_id = ...)

The query in the second situation would simply be:

- ...WHERE pdt_id = ...
- AND current_indicator = ’Y’

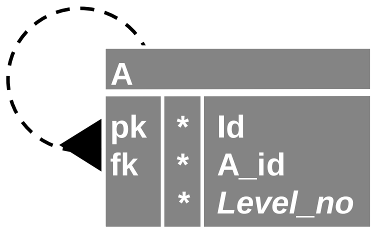
Hierarchy Level Indicator

Before



Create a column to represent the hierarchy level of a record.

After



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

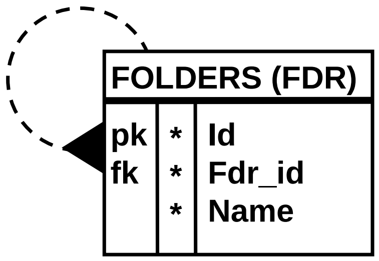
- No need to use the connect-by clause in query code

Disadvantages

- Each time a foreign key is updated, the level indicator needs to be recalculated, and you may need to cascade the changes

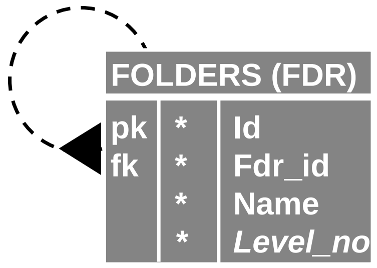
Example of Hierarchy Level Indicator

Before



Create a column to represent the hierarchy level of a record.

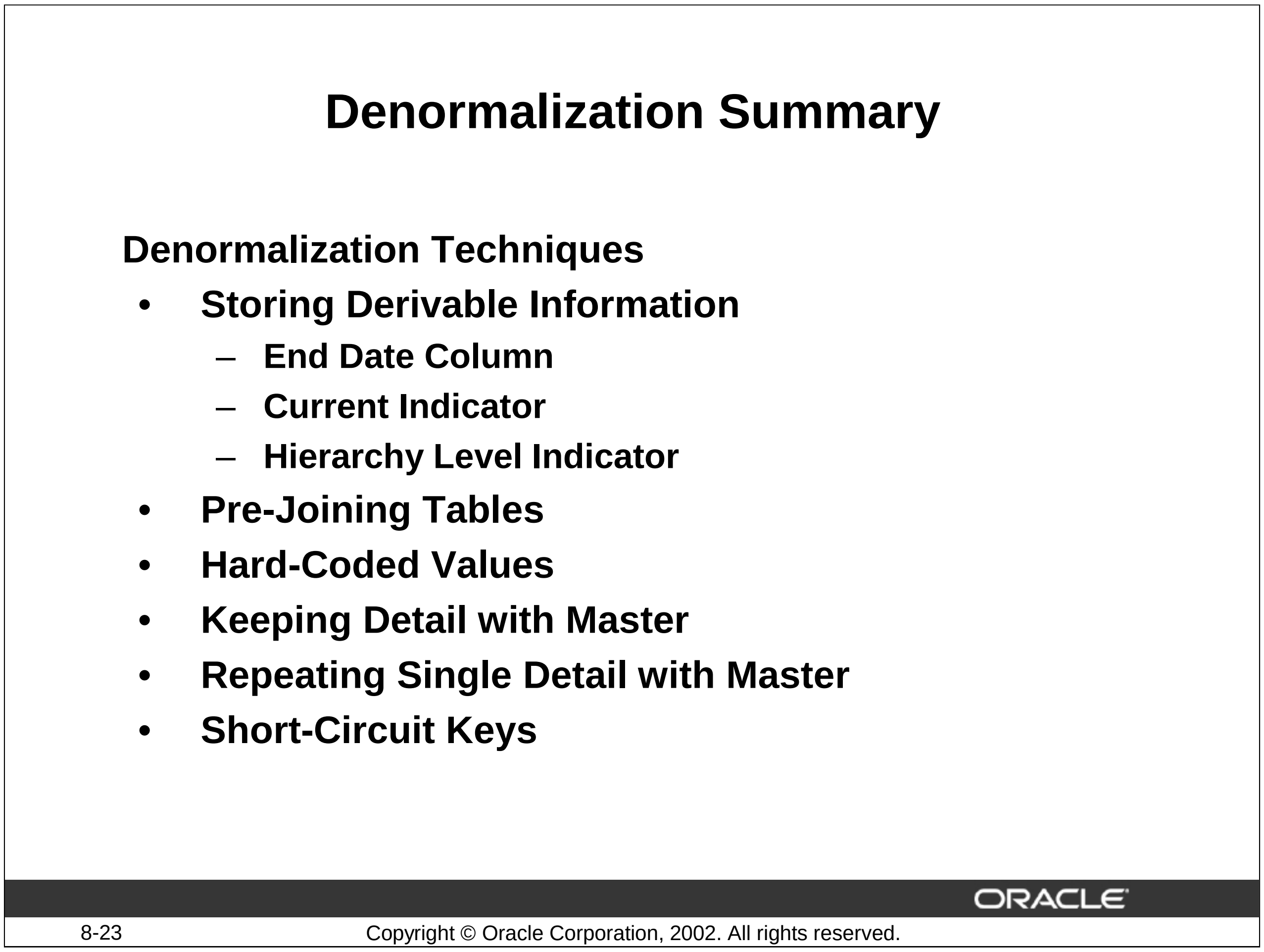
After



Hierarchy Level Indicator 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 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**

ORACLE

8-23

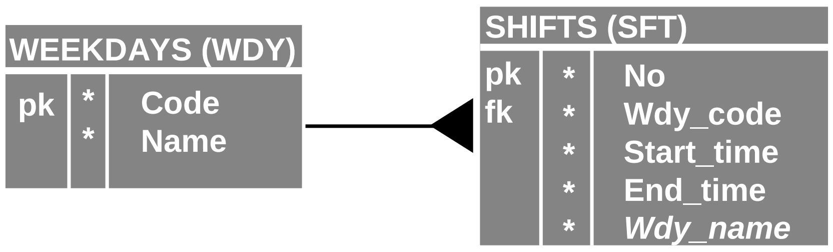
Denormalization Summary

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.

Practices

- Name that Denormalization
- Triggers
- Denormalize Price Lists
- Global Naming

Practice: Name that Denormalization *(1/3)*



Practice 8-1: Name that Denormalization

Goal

Learn to discriminate the type of denormalization depicted.

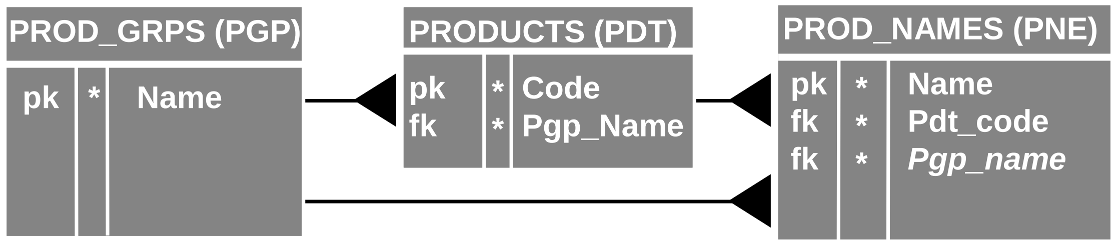
Your Assignment

For the following table diagrams, decide what type of denormalization is used and explain why the diagram depicts the denormalization you have listed.

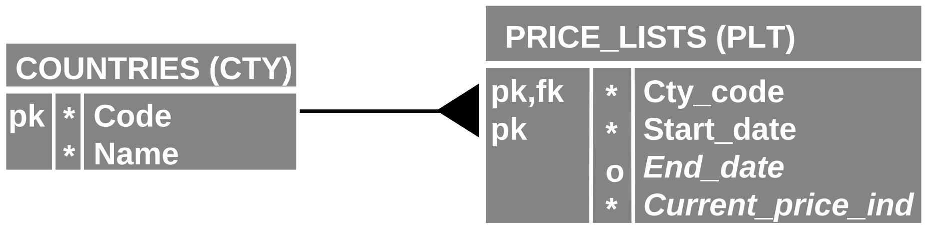
Use one of:

- Storing derivable information
- Pre-Joining Tables
- Hard-Coded Values
- Keeping Details with Master
- Repeating Single Detail with Master
- Short-Circuit Keys

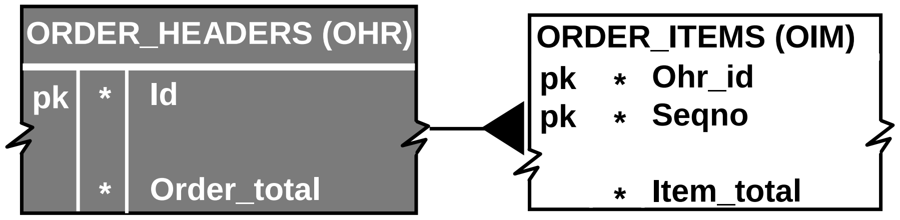
Practice: Name that Denormalization *(2/3)*



Practice: Name that Denormalization *(3/3)*



Practice: Triggers *(1/6)*



Practice 8-2: Triggers

Goal

The purpose of this practice is to investigate which database triggers are needed to handle a suggested denormalization.

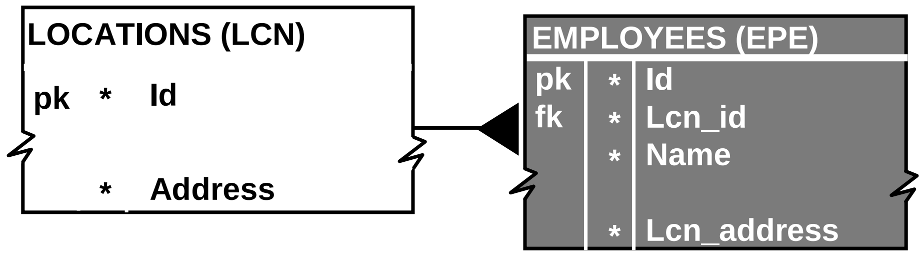
Your Assignment

1. Indicate which triggers are needed and what they should do to handle the denormalized column Order_total of ORDER_HEADERS.

Practice: Triggers *(2/6)*

Table	Trg Type	Column	Needed?	What should it do?
OHR	Insert			
	Delete			
	Update	Id		
		Order_total		
OIM	Insert			
	Delete			
	Update	Ohr_id		
		Item_total		

Practice: Triggers *(3/6)*



Practice 8-2: Triggers

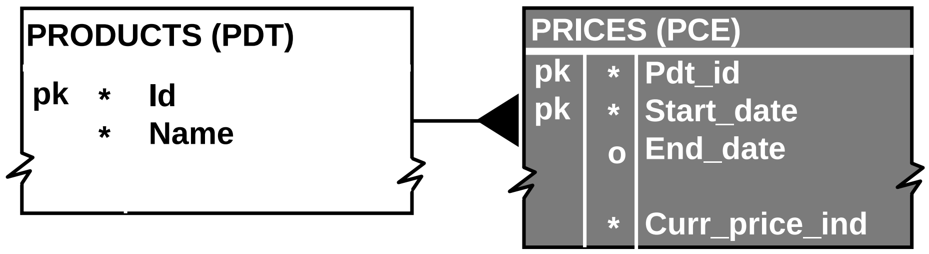
2. Indicate which triggers are needed and what they should do to handle the denormalized column Lcn_address of EMPLOYEES.

Practice: Triggers *(4/6)*

Table	Trg Type	Column	Needed?	What should it do?
LCN	Insert			
	Delete			
	Update	Address		
		other cols		
EPE	Insert			
	Delete			
	Update	Lcn_id		
		Lcn_address		

ORACLE

Practice: Triggers *(5/6)*



Practice 8-2: Triggers

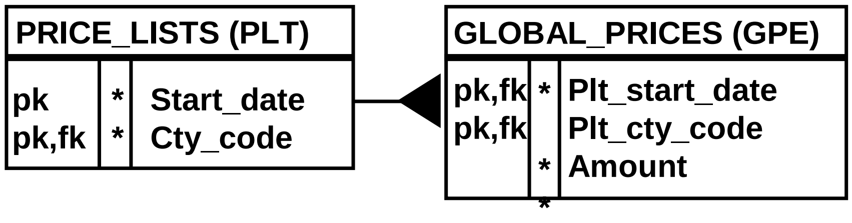
3. Indicate which triggers are needed and what they should do to handle the denormalized column Curr_price_ind of table PRICES.

Practice: Triggers *(6/6)*

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		

Practice: Denormalize Price Lists

- Speed up performance for queries on Amount.
- Insert new price lists before their effective date.



Practice 8-3: Denormalize Price Lists

Goal

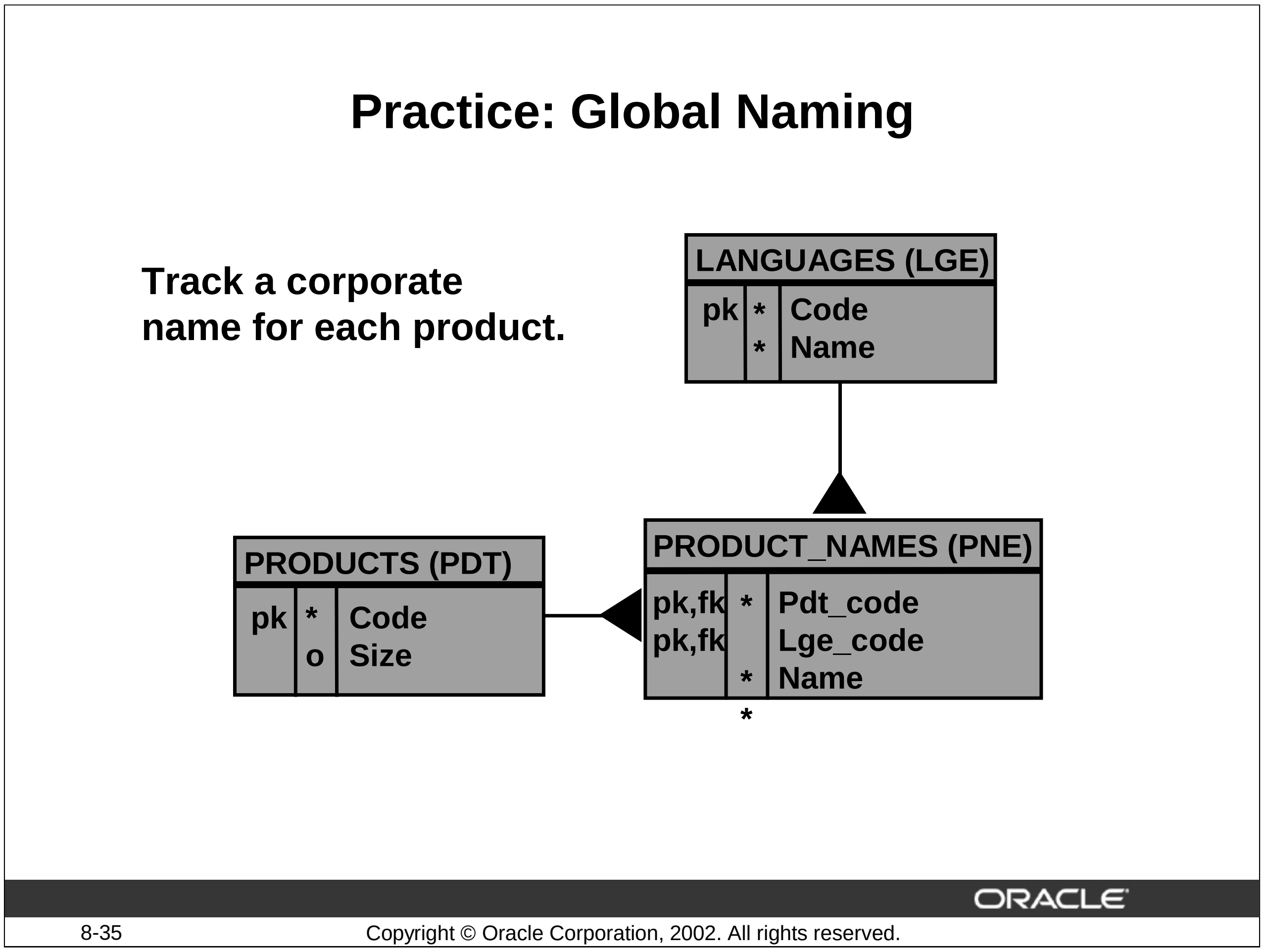
The aim of this practice is to decide on the type of denormalization you could use, and what code is needed to ensure database integrity.

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: Global Naming

Track a corporate name for each product.

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.

9

Database Design Considerations

ORACLE

Overview

- **Oracle specific Design Considerations**
- **Data Integrity Issues**
- **Performance Considerations**
- **Storage Issues**

Introduction

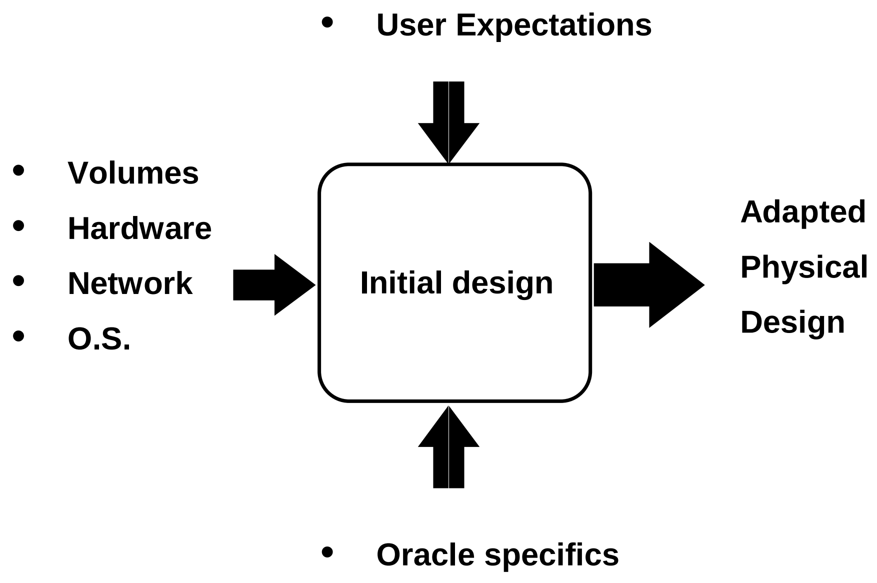
This lesson illustrates some principles of the Oracle RDBMS and presents the various techniques that can be used to refine the physical design.

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

Why Adapt Data Design?



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

Depending on:

- **Domains**
- **Storage issue**
- **Performance**
- **Use**

Select a data type for columns:

- **Character**
- **Number**
- **Date**
- **Large Objects**

ORACLE

Oracle Data Types

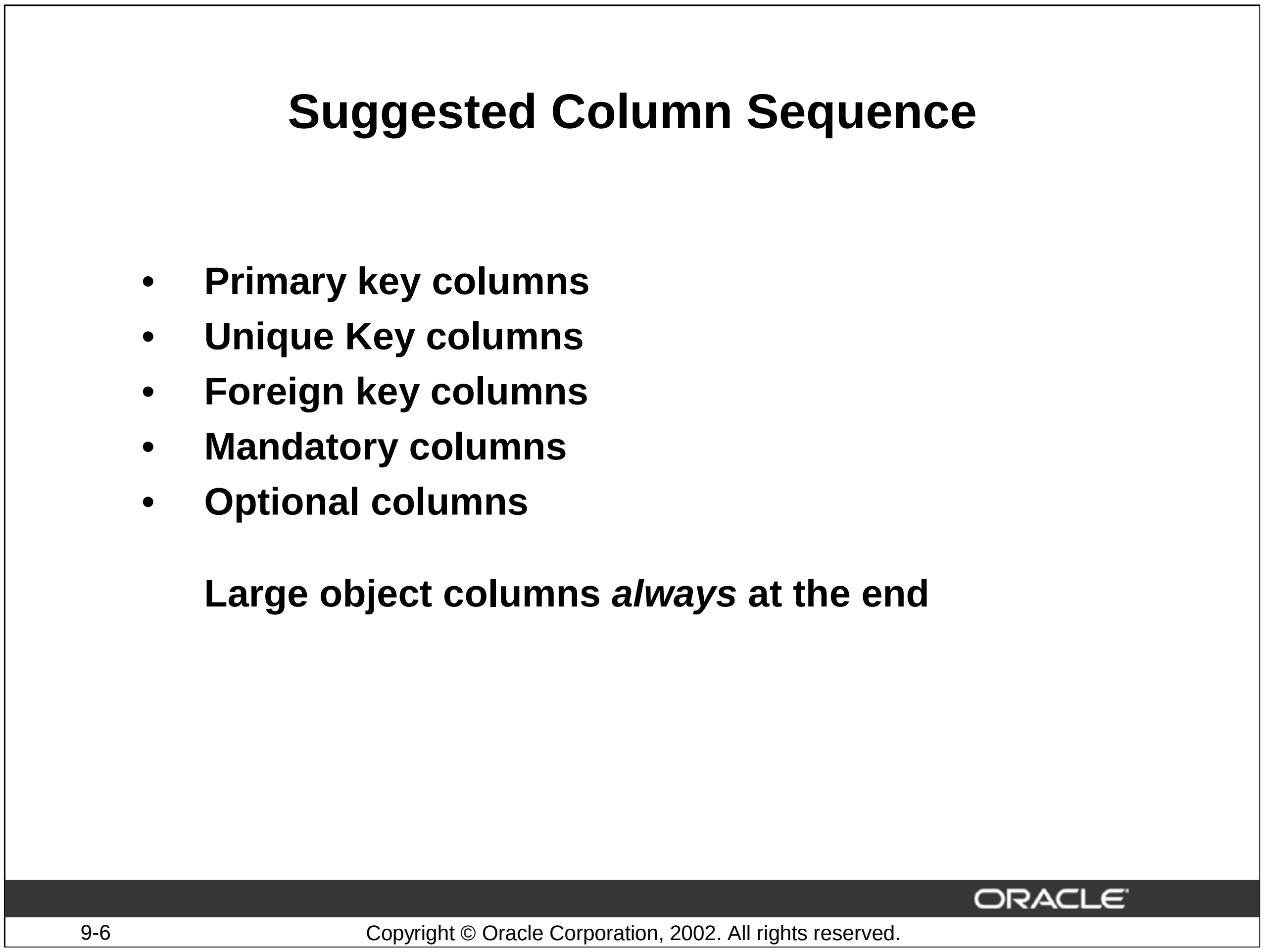
When you create a table or cluster, you must specify an internal data type for each of its columns. These data types define a generic domain of values that each column can contain.

- 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.

Oracle Data Types

Most Commonly-Used Oracle Data Types

1. **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.
2. **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.
3. **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.
4. **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.
5. **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.
6. **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.
7. **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.
8. **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.
9. **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



Suggested Column Sequence

- **Primary key columns**
- **Unique Key columns**
- **Foreign key columns**
- **Mandatory columns**
- **Optional columns**

Large object columns *always* at the end

ORACLE

9-6

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.

The suggested optimal column sequence is the following:

- Primary key columns
- Unique key columns
- Foreign key columns
- Remaining mandatory columns
- 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

```
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

Primary Keys and Unique Keys

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.

Primary Keys and Unique Keys

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.

Primary Keys

Choosing the Right Key

- **Simplicity**
- **Ease of use**
- **Performance**
- **Size**
- **Meaningless**
- **Stability**

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.

How to Choose the Primary Key

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.

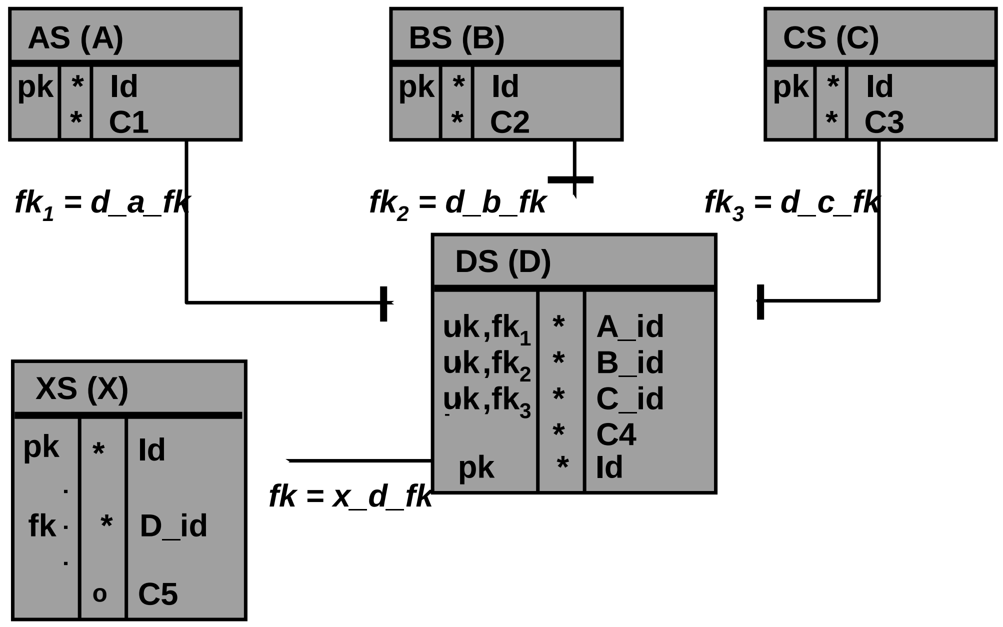
Meaningless:

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



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 than 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.

Artificial Keys

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

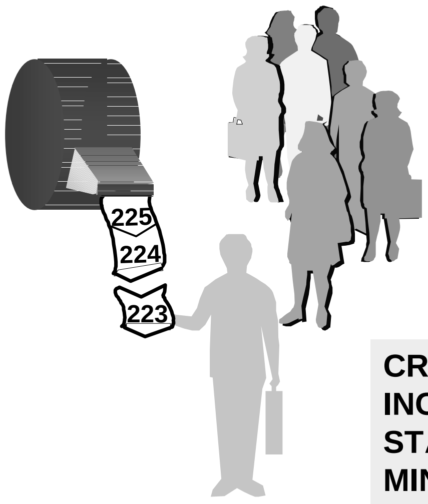
- **Negative:** It would corrupt your data model, as you would add elements that have no business meaning.
- **Positive:** There is a close mapping between the conceptual and technical model that reduces the chances of misunderstanding.

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



```
CREATE SEQUENCE sequence_name
INCREMENT BY         number
START WITH           number
MINVALUE             number
MAXVALUE            number
CACHE number | NOCACHE
CYCLE | NOCYCLE;
```

Sequences

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 Behavior

	Delete	Update
Restrict	✓	✓
Cascade	✓	
Default Nullify		

✓ **Supported by Oracle through declaration**

9-14

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.

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.

Foreign Key

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.

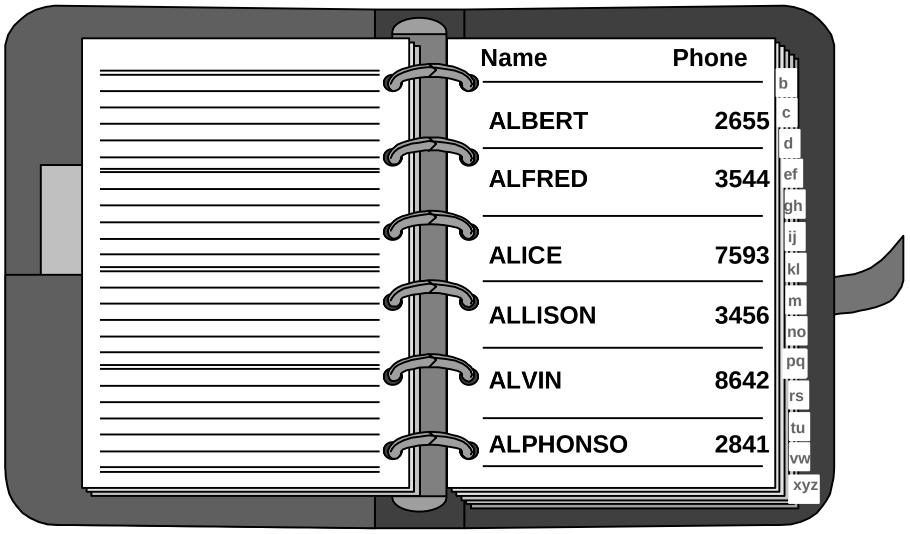
Typical Use

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

- **Performance**



- **Uniqueness**

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

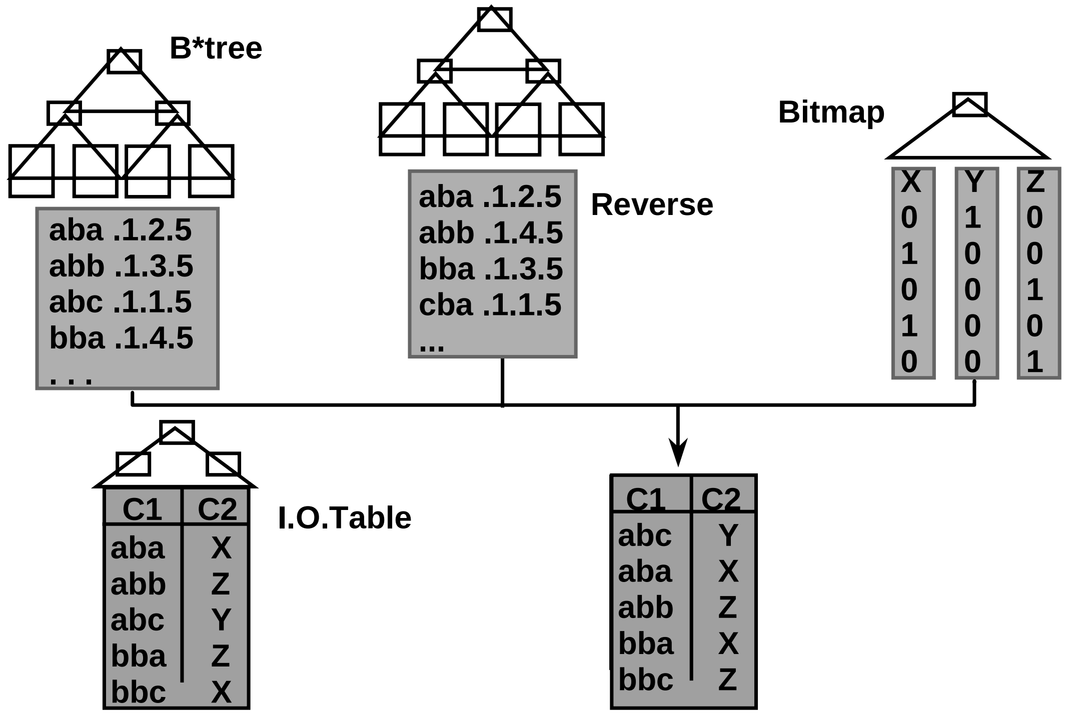
Performance

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.

Choosing Indexes



Index Types

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

Index Types

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

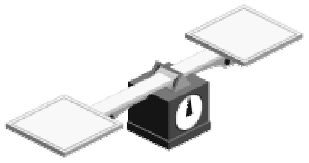
Since Oracle8i it is possible to create an index based on a SQL function.

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.

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**

ORACLE

9-19 Copyright © Oracle Corporation, 2002. All rights reserved.

Choosing Columns to Index

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

Choosing Columns to Index

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 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

When Are Indexes Used?

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.

Partitioning Tables and Indexes

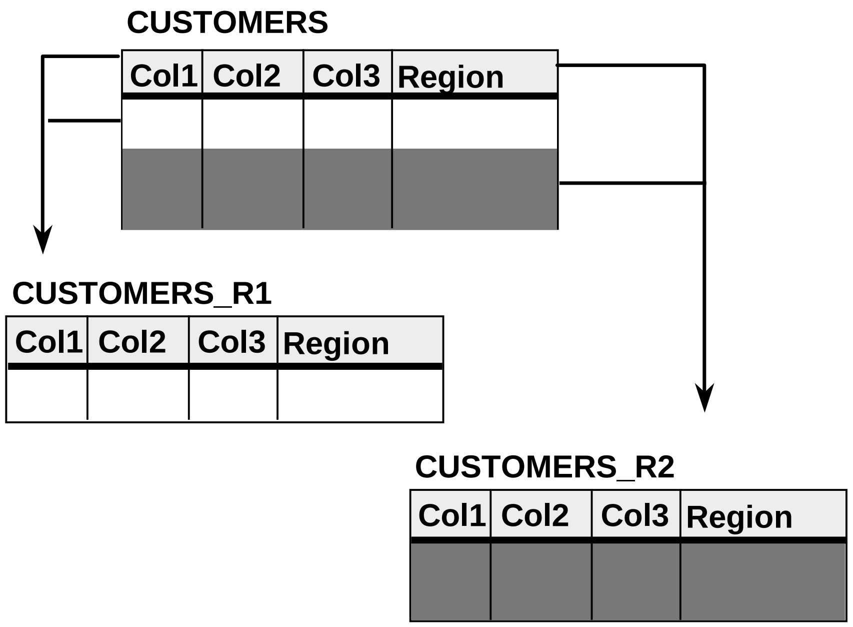


Table and Index Partitioning

Partitioned Table

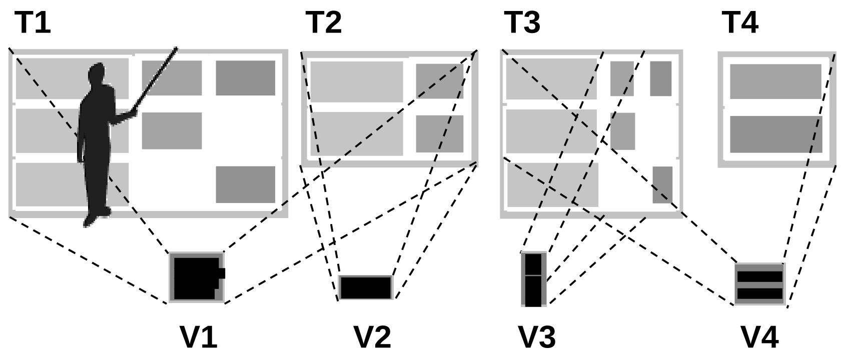
Since Oracle8, when creating a table, you can specify the criteria on which you want to divide the table and make a horizontal partitioning. There are then as many partitioned tables as there are distinct values in the column. Each partitioned table has a specific name but access is made referring to 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

- Restricting access
- Presentation of data
- Isolate applications from data structure
- Save complex queries
- Simplify user commands



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.

Reasons for Views

- **Advantages**
  - **Dynamic views**
  - **Present denormalized data from normalized tables**
  - **Simplify SQL statements**
- **Disadvantages**
  - **May affect performances**
  - **Restricted DML in some cases**

Use of Views

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.
- An 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

- Unique index
- Views with “Check option” clause
- Generic Arc implementation

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.

Unique Index

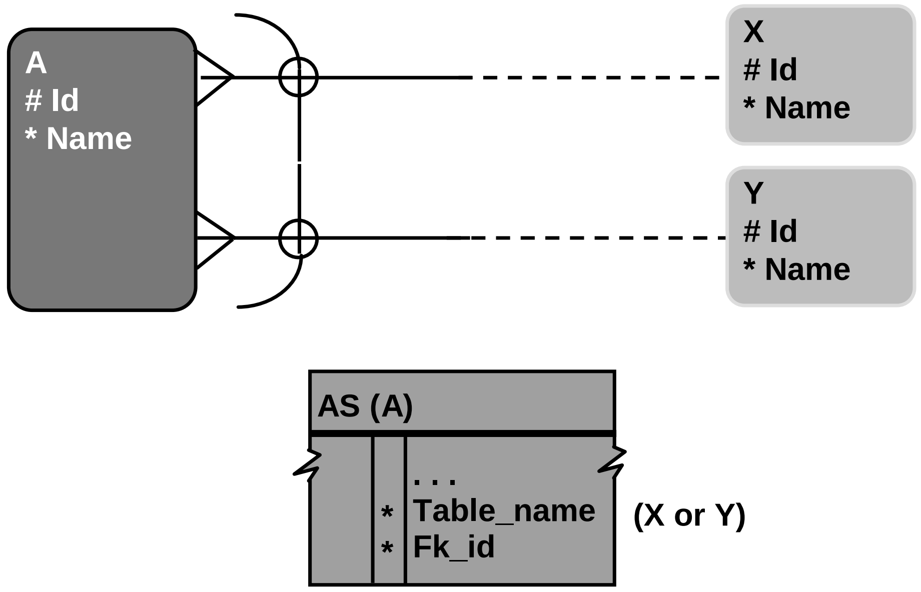
Unique Indexes used to be created manually on the primary key columns because the primary key constraint could not be declared up to Oracle7.

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



Generic Arc Implementation

The generic arc implementation is a fossil construction you may find in old systems.

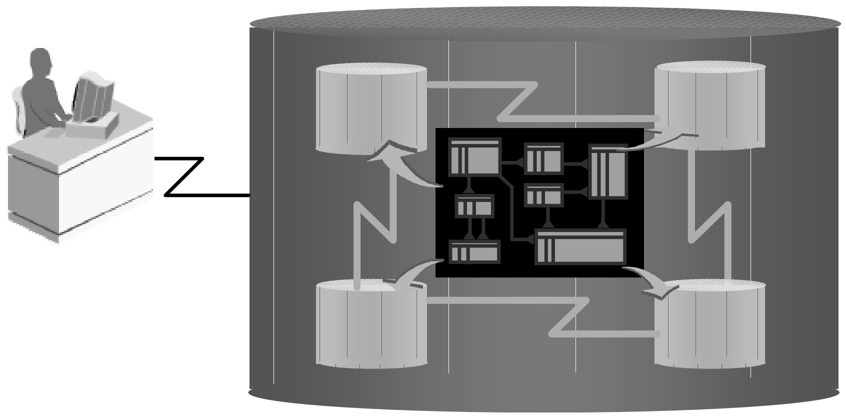
In the implementation of the arc of entity A in the example, the three relationships in the arc were merged into one generic foreign key column Fk_id. Added to table AS is a NOT NULL column that keeps the information about which table the foreign key value refers to. This used to be a popular technique 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 Database

Different physical databases appear as one logical database.



ORACLE

9-27 Copyright © Oracle Corporation, 2002. All rights reserved.

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 Databases

- Resilience
- Reduced line traffic
- Location transparency
- Local autonomy
- Easier growth path

but

- Increased, distributed, complexity

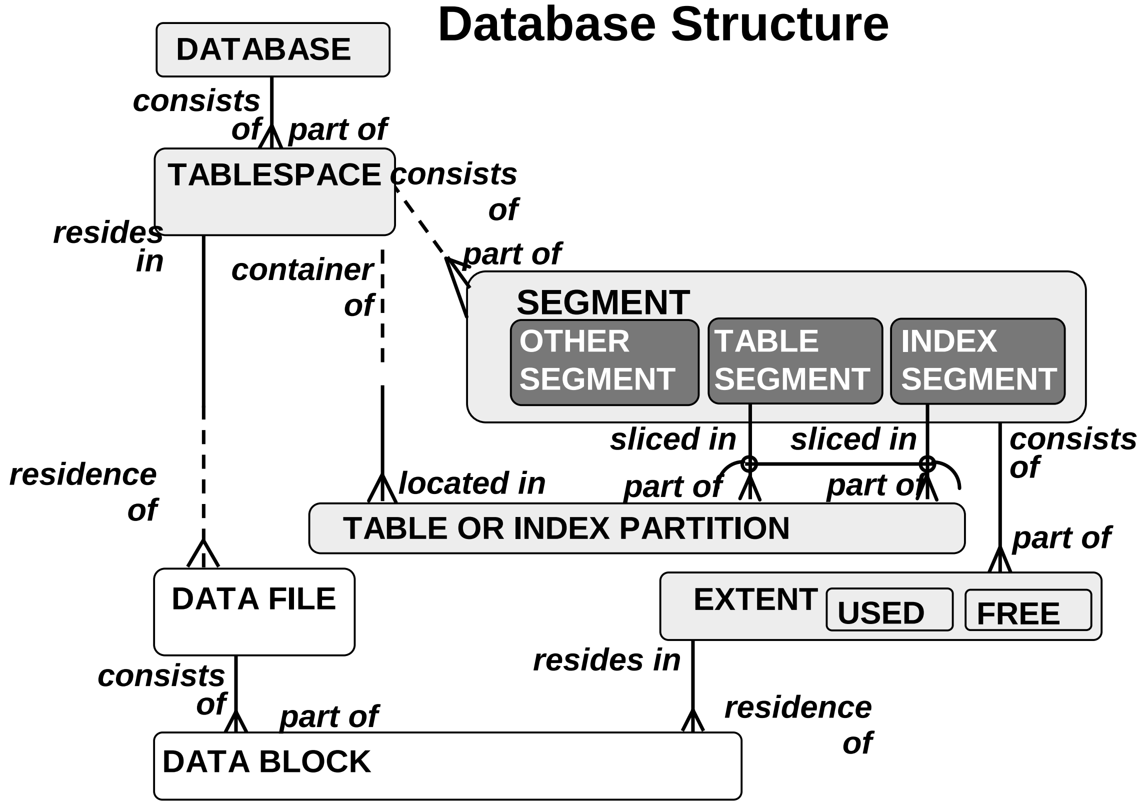
Benefits of Distributed Design

- 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.
- 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.

Database Structure



Oracle Database Structure

Tablespaces

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.

Oracle Database Structure

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.

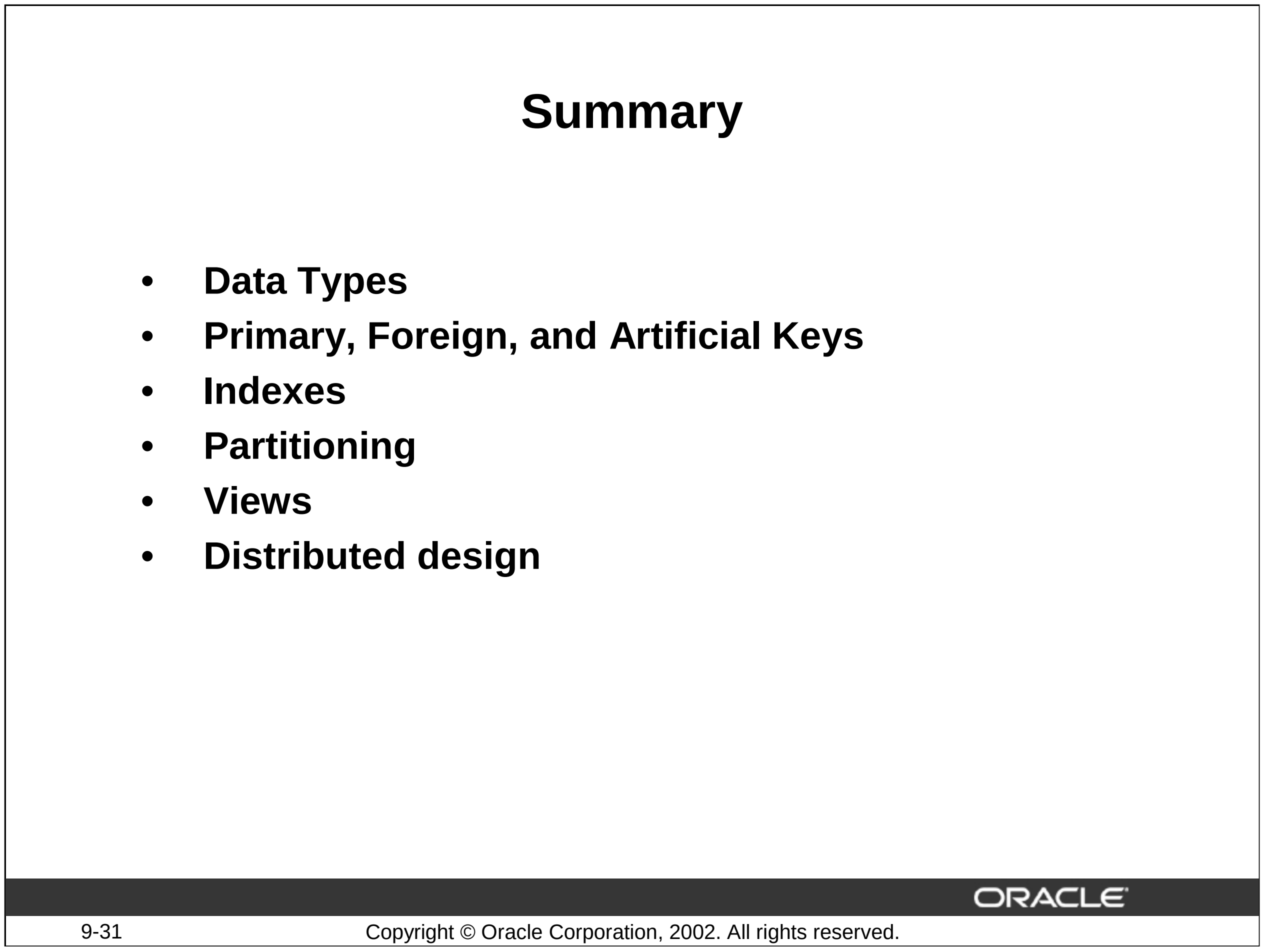
Data Files

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

- Data Types
- Primary, Foreign, and Artificial Keys
- Indexes
- Partitioning
- Views
- Distributed design

9-31

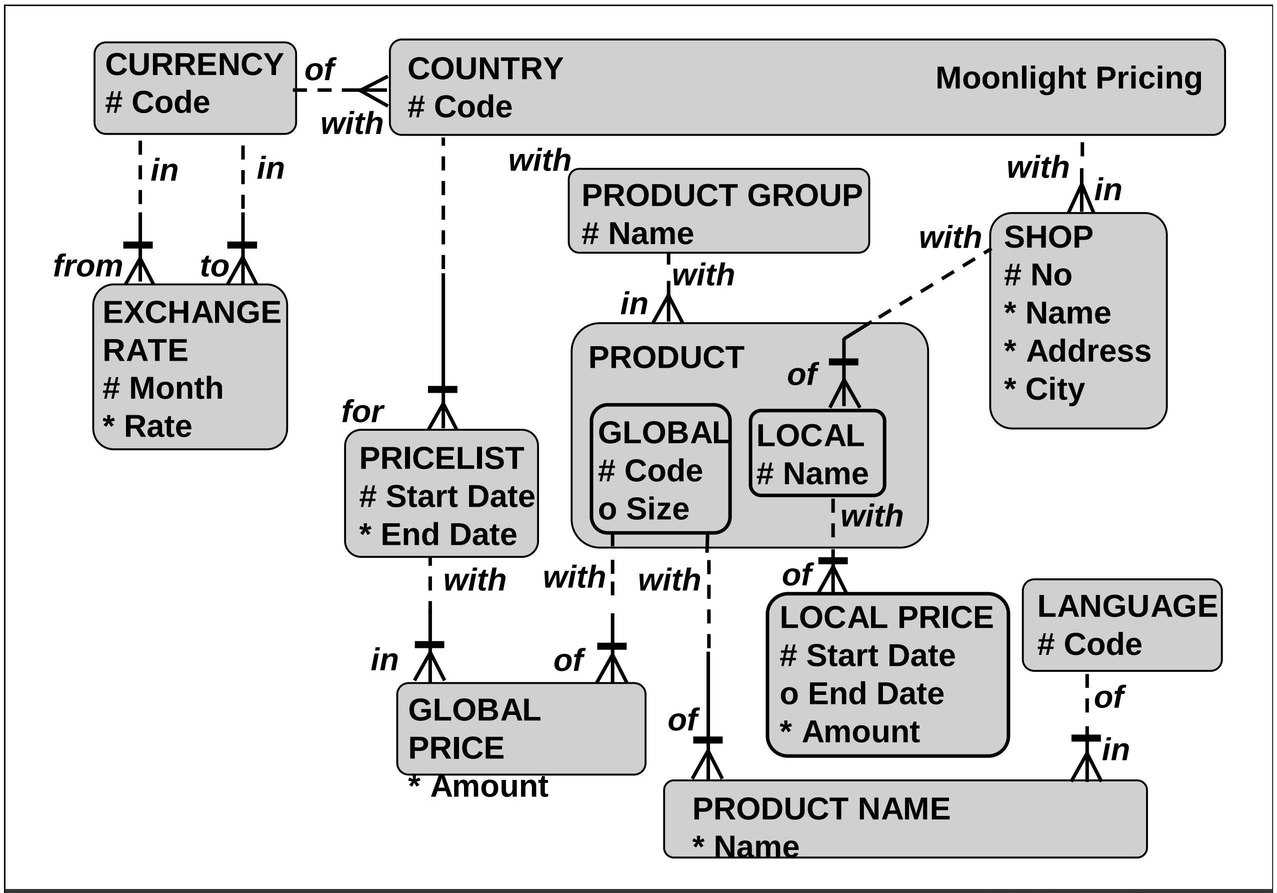
Summary

- 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.

Practices

- Data Types
- Artificial Keys
- Product Pictures

9-32



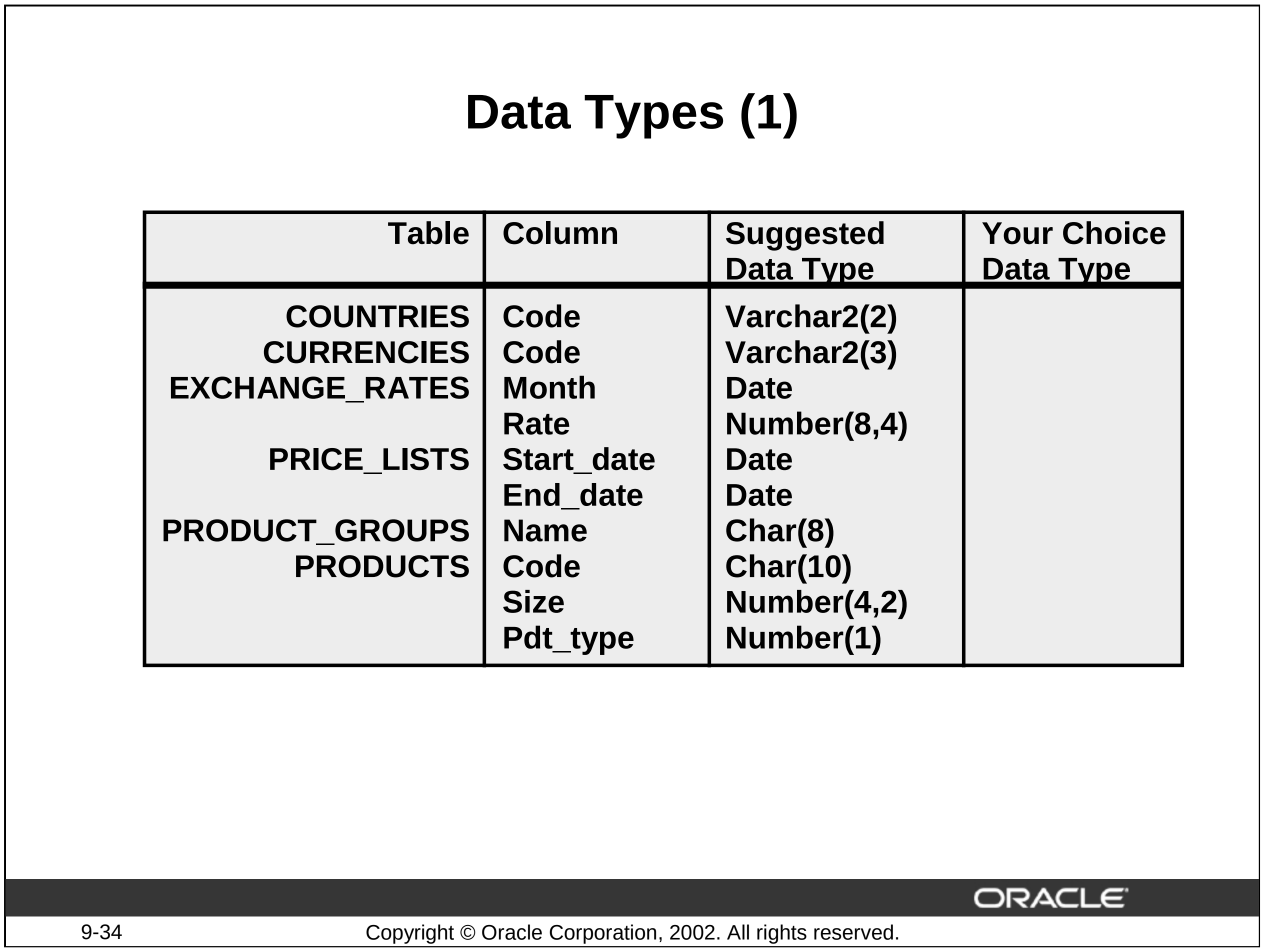
Practice 9-1: Data Types

Goal

The purpose of this practice is to perform a quality check on proposed data types.

Scenario

Use the model that illustrates Moonlight pricing.



Data Types (1)

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)	

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.

Data Types (2)

Table	Column	Your Choice Data Type
GLOBAL_PRICES	Amount	
LOCAL_PRICES	Start_date	
	End_date	
	Amount	
SHOPS	Name	
	Address	
	City	

Your Assignment

2. Suggest data types for the following columns. They are all based on previous practices.
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

- Indicate for each table if you see benefits of creating an artificial key and why.
  - COUNTRIES
  - GLOBAL_PRICES
  - PRICE_LISTS
- 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

- Decide what data type you would advise to be used for each column.
- 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.

Solutions

Solution: Instance or Entity?

Concept	E/A/I?	Example Instance or Entity
PRESIDENT	E	Lincoln, Washington, Gorbachev
ELLA FITZGERALD	I	STAR, SINGER, PERSON
DOG	E	Snoopy
ANIMAL	E	Cat, Dog, ...
HEIGHT	A	PERSON, BUILDING, ...
TYPE OF TRANSPORT	*E*	*CAR*
Number of Wheels	*A*	*CAR*
My current car	*I*	*CAR*

ORACLE

Practice 1-1 Instance or Entity: Solution

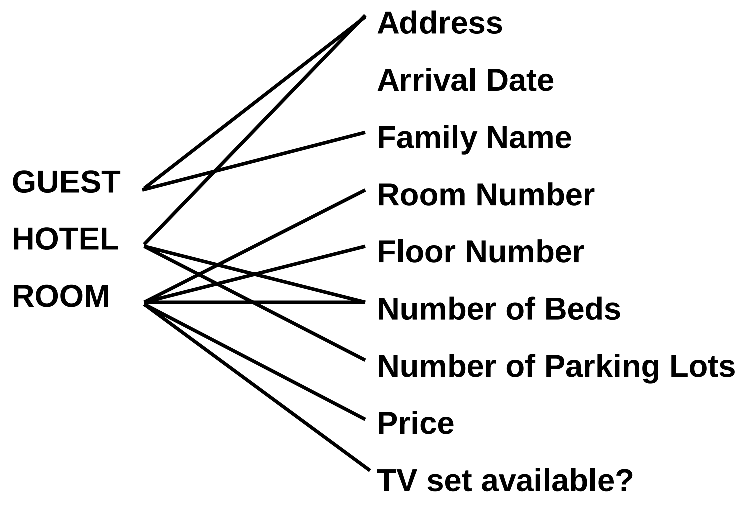
List which of the following concepts you think is an Entity, Attribute, or Instance. If you mark one as an entity, then give an example instance. If you mark one as an attribute or instance, give an entity. For the last three rows, find a concept that fits.

A Possible Solution:

In fact, all concepts can be both an entity and an instance of an entity. This depends on the business perspective. Some businesses have strange perspectives...

Even Ella Fitzgerald could be an entity. Think in the context of a look-alike contest or the context of an art collection where there may be several instances of an Ella Fitzgerald.

Solution: Guest



Practice 1-2 Guest: Solution

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

A Possible Solution:

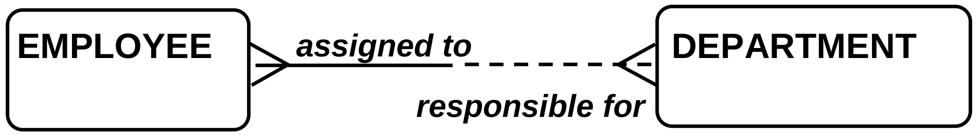
Note that several entities seem to have the same attribute. In fact you should view this as entities having different attributes that, by coincidence, have the same name.

The rule is that an attribute always belongs to one entity. Per entity the data type or length of the attribute may be different.

Some of the presented answers are disputable. You could argue that a GUEST has an Arrival Date when old arrival dates are not of interest and may be overwritten.

If you want to keep the number of beds per room, you do not need to keep it at the level of the hotel, as you can derive the value based on the sum of all beds per room.

Solution: Reading



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

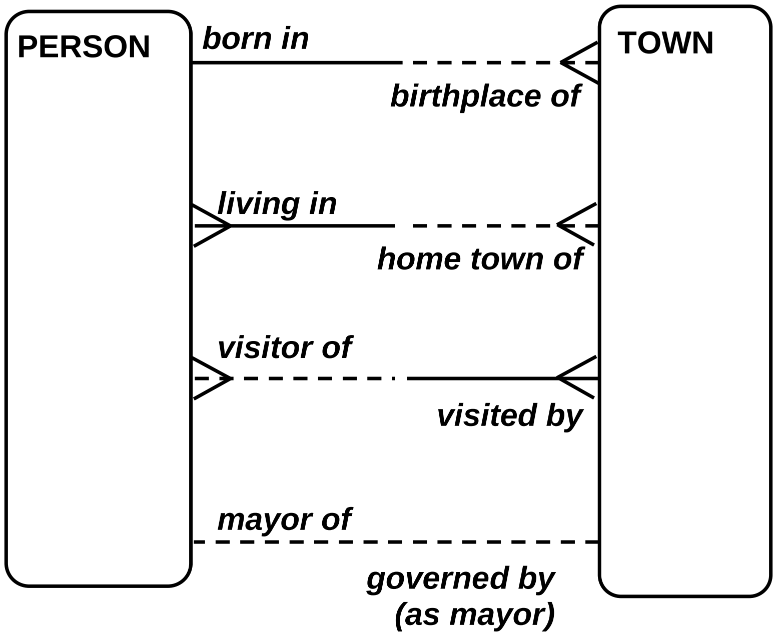
A-4

Practice 1-3 Reading: Solution

Which text corresponds to the diagram?

Option B is the correct reading.

Practice: Read and Comment



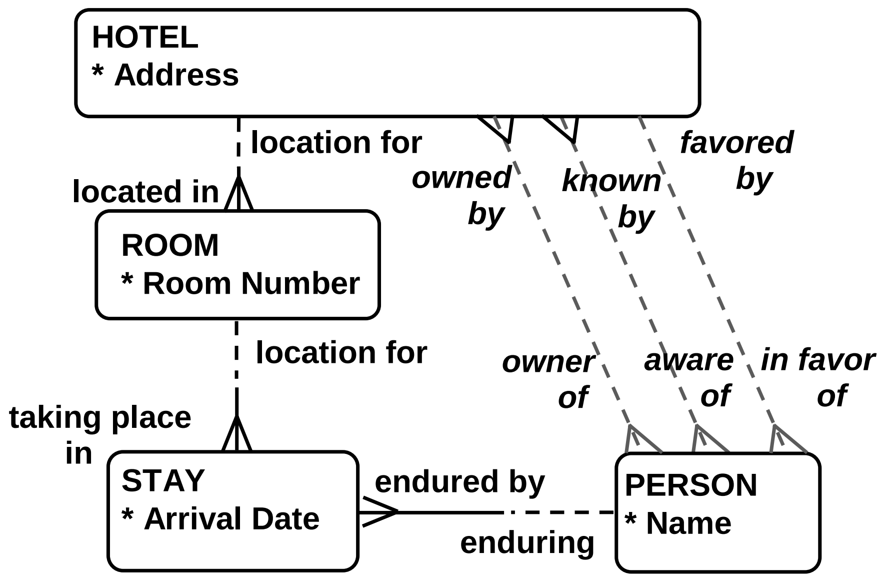
Practice 1-4 Read and Comment: Solution

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.

From Top to Bottom:

- Each Person must be born in one or more Towns.
  Each Town may be birthplace of exactly one Person.
  Comment: both sides seem te be of the wrong degree.
- Each Person must be living in one or more Towns.
  Each Town may be home town of one or more Persons.
  Comment: The first certainly has a wrong degree. The optionality seems wrong as well.
- Each Person may be visitor of one or more Towns.
  Each Town must be visited by one or more Persons.
  Comment: The optionality of the second seems very likely, but is probably wrong, depending of the definition of Town.
- Each Person may be mayor of exactly one Town.
  Each Town may be governed by (as mayor) exactly one Person.
  Comment: Both seem fine, except that if you need to keep historical information, then both sides of the relationship must be “many”.

Solution: Hotel



Practice 1-5 Hotel: Solution

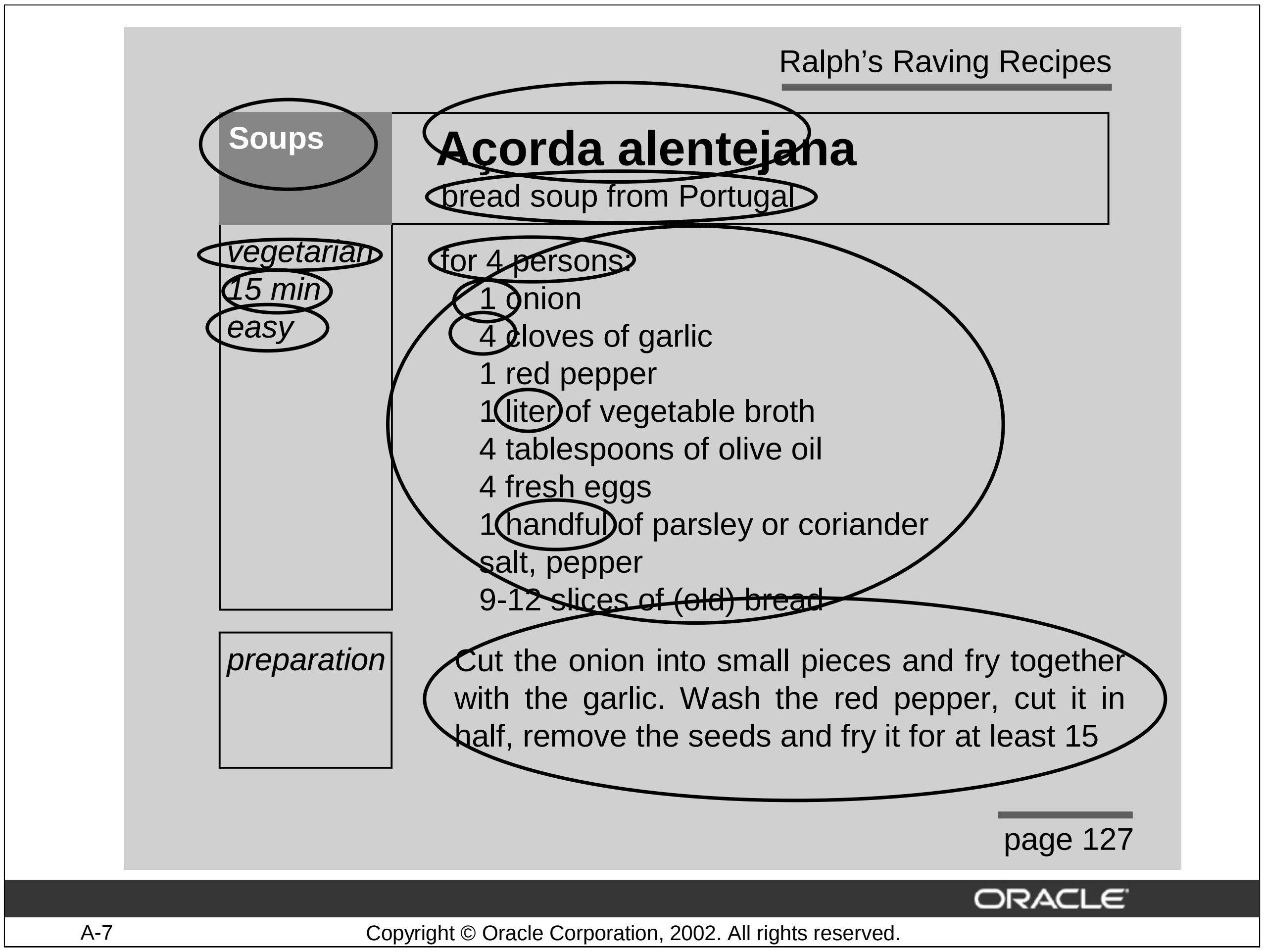
Comment on the relationships of the model presented here.

Possible Comments:

- The relationship the lodging for/ in between entity STAY and HOTEL should be replaced by a relationship between ROOM and HOTEL, such as located in/location for.
- Every PERSON should have the possibility of more than one STAY; a STAY should be for a single person only. When two persons sign in it should lead to two instances of STAY.
- The relationship between PERSON and HOTEL seems to be redundant.

This is a Possible Model:

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


Ralph’s Raving Recipes

Soups

Açorda alentejana

bread soup from Portugal

vegetarian

15 min

easy

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

preparation

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

page 127

A-7

Practice 1-6 Recipe: Solution

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.

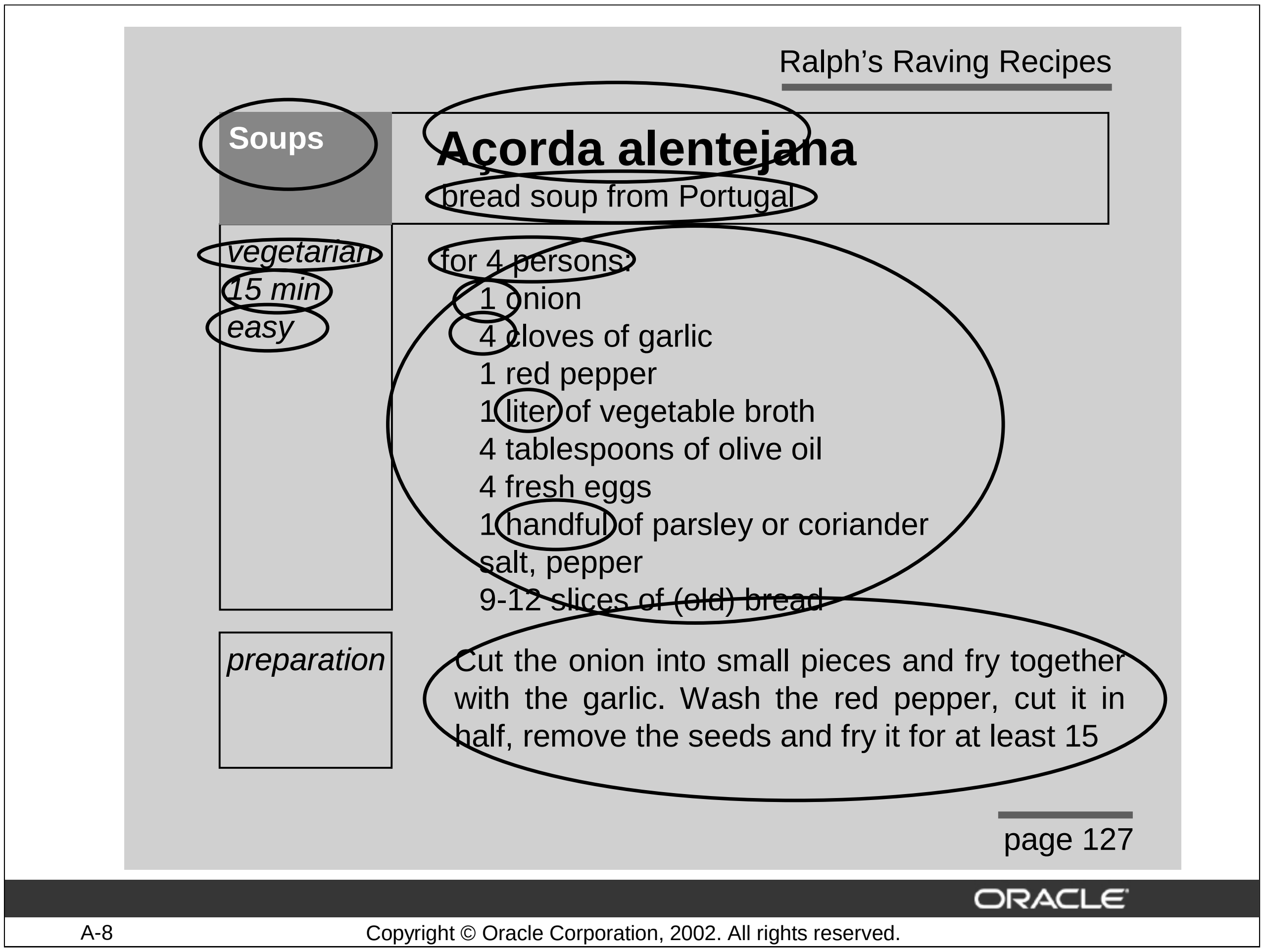
2 Group the various types of information into entities and attributes.

3 Name the relationships you discover and draw a diagram.

A Possible Solution:

Here is a list of types of information. There may be others and you may have named them differently. An example from the text is given in italics.

- **Recipe (Açorda)**
- **Original Name (Açorda alentejana)**
- **Description (Bread Soup from Portugal)**
- **Vegetarian Indicator (Vegetarian)**
- **Number of Persons (4)**
- **Ease of Preparation (Easy)**
- **Preparation Time (15 min)**
- **Preparation Description (“Cut the onion ...”)**
- **Ingredient (Garlic, bread)**
- **Quantity (4, for the cloves of garlic, 1 for the pepper)**
- **Unit (liter, piece, handful)**
- **Recipe Group (Soups)**



Ralph's Raving Recipes

Soups

Açorda alentejana

bread soup from Portugal

vegetarian
15 min
easy

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

preparation

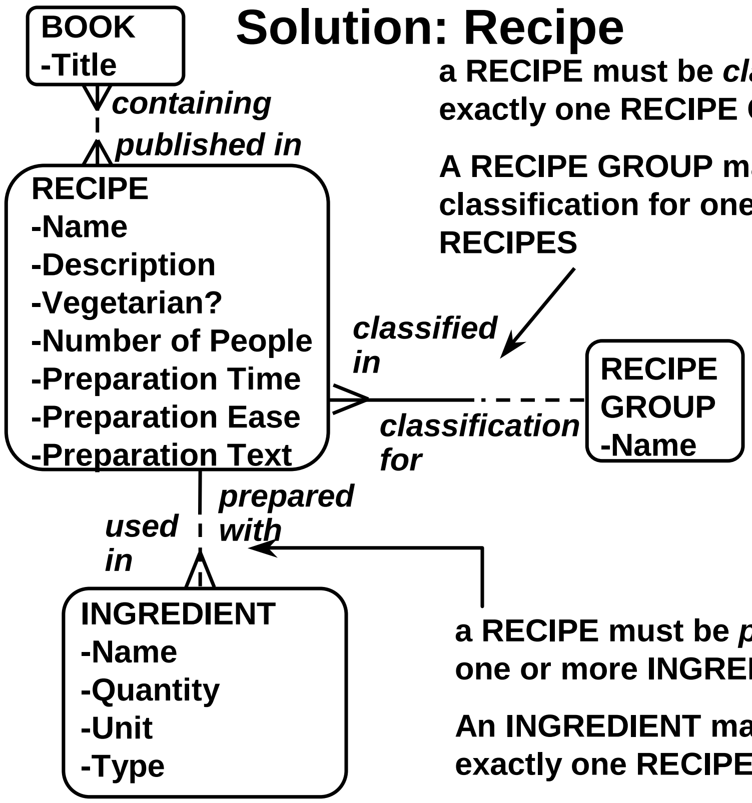
Cut the onion into small pieces and fry together with the garlic. Wash the red pepper, cut it in half, remove the seeds and fry it for at least 15

page 127

ORACLE

A-8

Solution: Recipe



a RECIPE must be *classified in* exactly one RECIPE GROUP

A RECIPE GROUP may be classification for one or more RECIPES

a RECIPE must be *prepared with* one or more INGREDIENTS

An INGREDIENT may be *used in* exactly one RECIPE

Practice 1-6 Recipe: Solution

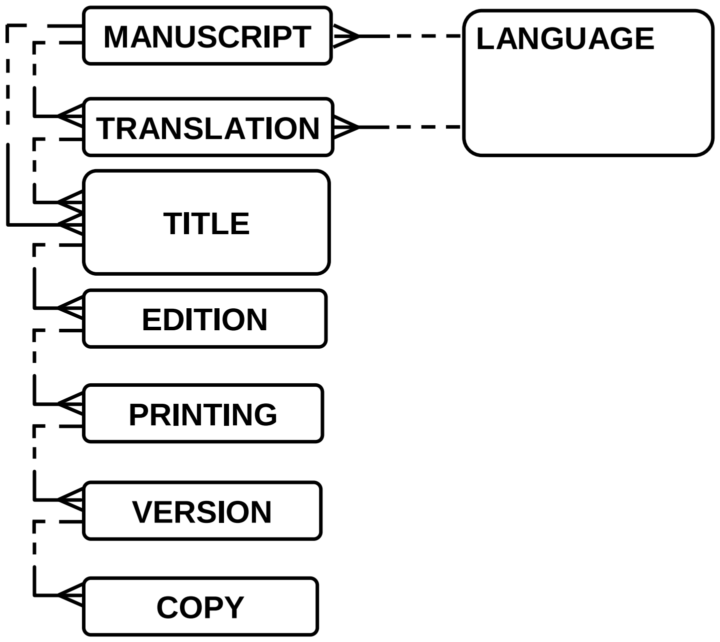
Note that the ingredients are somehow sorted. There must be hidden intelligence to do that.

This might lead to: Ingredient Type (vegetable, spice)

A possible model for the recipes:

Other solutions may include entity AUTHOR, BOOK, BOOK TYPE, RECIPE, RECIPE ITEM, INGREDIENT and INGREDIENT GROUP.

Solution: Books

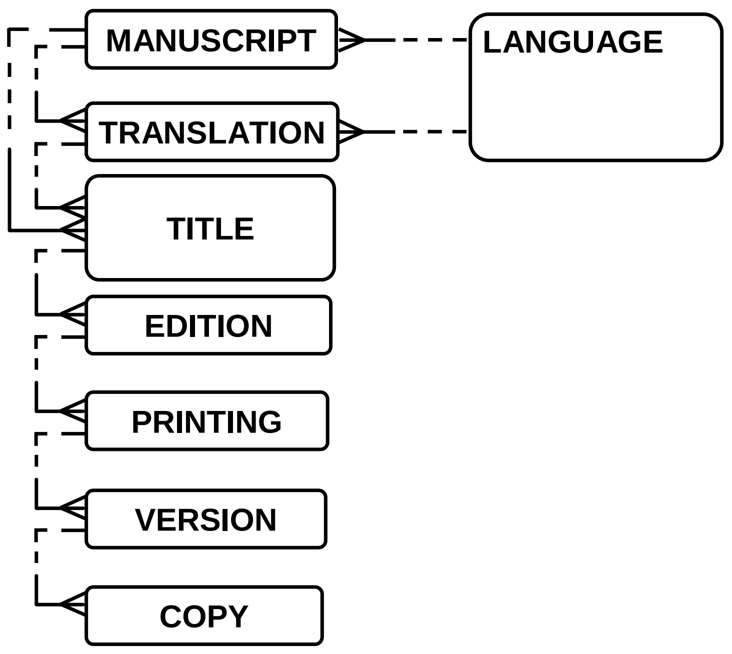


Practice 2-1 Books: Solution

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.

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.

Solution: Books

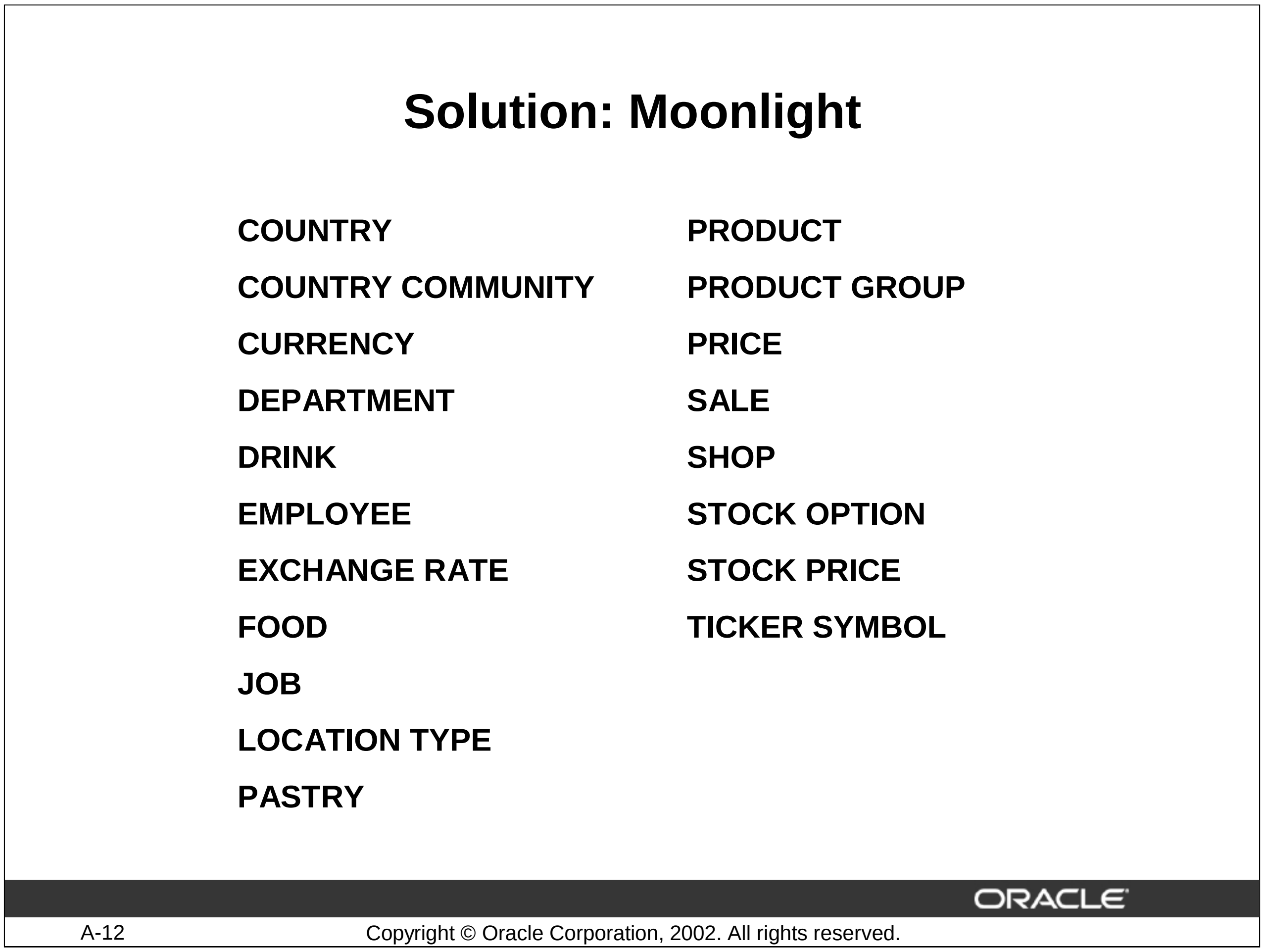


Practice 2-1 Books: Solution

Different meanings of the word Book from the text. Attributes are between brackets. Attributes marked with an asterisk should preferably be modeled as a relationship to another entity.

- **MANUSCRIPT (Title, Year of Completion, Author*)**
- **EDITION (Sequence Number of Printing)**
- **COPY (position on the shelf)**
- **TRANSLATION IN LANGUAGE (Language*, Translator*)**
- **TITLE (Authorized Text)**
- **PRINTING (Font*, Year of Printing)**
- **VERSION (Hardcover Indicator)**

and maybe more.



Solution: Moonlight

COUNTRY

COUNTRY COMMUNITY

CURRENCY

DEPARTMENT

DRINK

EMPLOYEE

EXCHANGE RATE

FOOD

JOB

LOCATION TYPE

PASTRY

PRODUCT

PRODUCT GROUP

PRICE

SALE

SHOP

STOCK OPTION

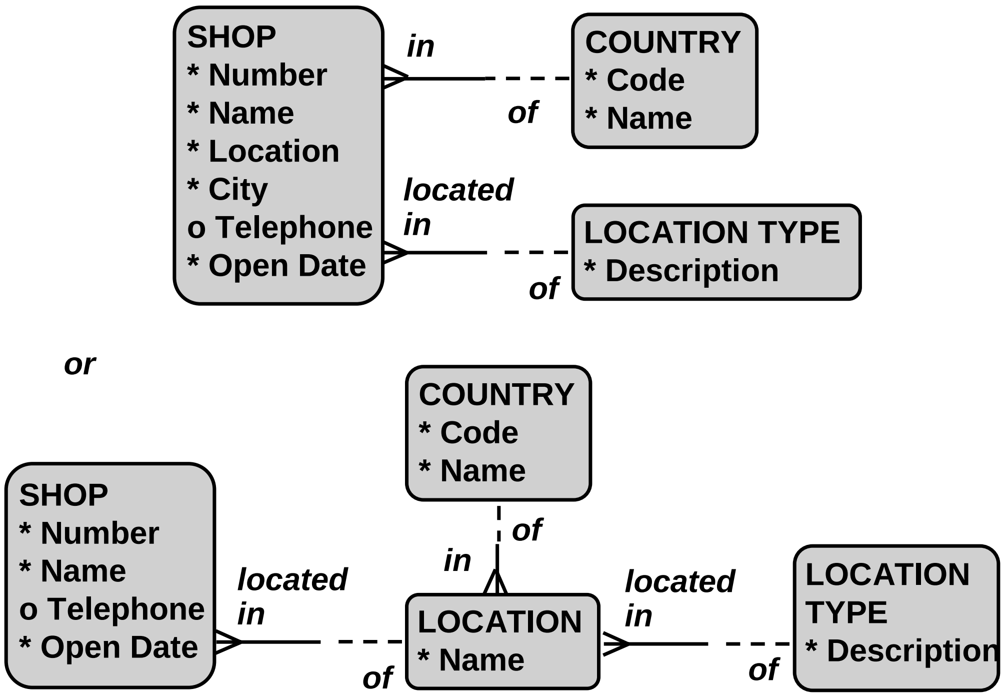
STOCK PRICE

TICKER SYMBOL

Practice 2-2 Moonlight: Solution

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.
   - Possible entities sorted alphabetically:
2. Write a formal definition of the entity that represents: Possible definition:
   - A shop is a place where Moonlight sells, did sell, or will sell products to customers, a retail outlet. An example of a shop is the one in ... that was opened in ... .
   - An employee is a person that works, did work or will work for Moonlight in a named job and is paid by Moonlight. An example of an employee is ... who works as ... in ... .

Solution: Shops



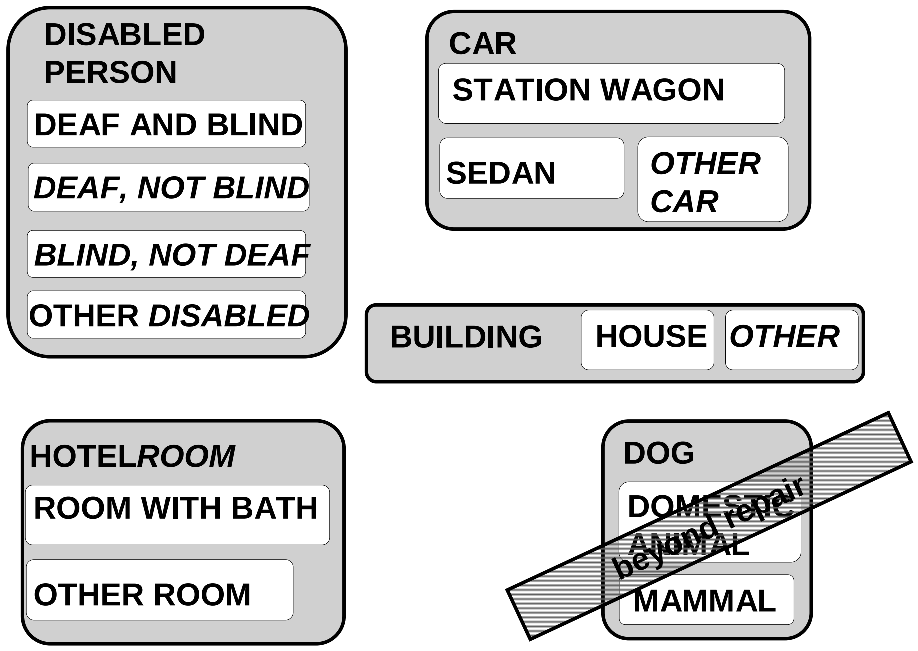
Practice 2-3 Shops: Solution

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

Both solutions are fine. The second sees a location like JFK Airport as a possible location for several shops. Maybe most locations have one shop only.

Note that in the second solution a location does not necessarily have a shop. This allows for future locations. In this respect the second model is somewhat more flexible.

Solution: Subtypes



Practice 2-4 Subtypes: Solution

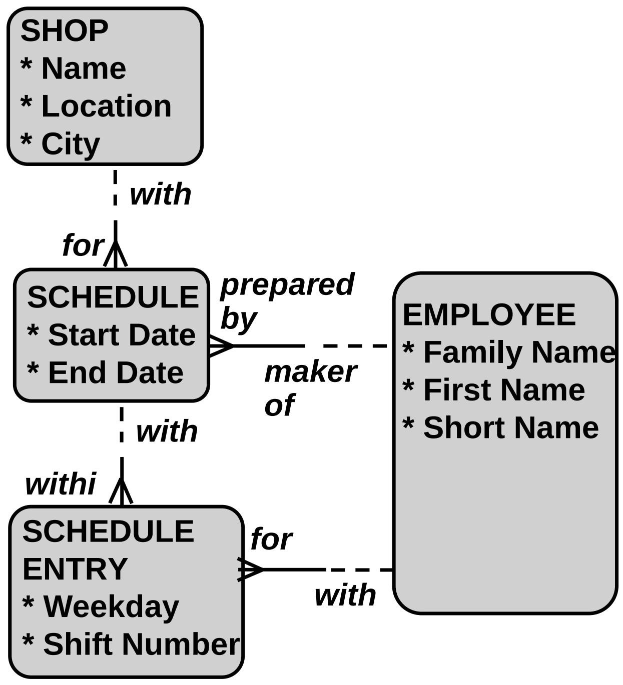
Find all incorrect subtyping in the illustration. Explain why you think the subtyping is incorrect.

Adjust the model to improve it.

All examples are cases of wrong subtyping.

- **The subtypes of DISABLED PERSON are not mutually exclusive: DEAF and BLIND may overlap.**
- **The subtypes of CAR are not exhaustive: there are cars that are neither a station wagon nor a sedan.**
- **Entity BUILDING should have at least two subtypes.**
- **If ROOM WITH BATH is a subtype of HOTEL this would mean that every ROOM WITH BATH is a HOTEL.**
- **DOMESTIC ANIMAL and MAMMAL are not mutually exclusive. In fact, DOG is subtype of both, instead of the other way around.**

Solution: Schedule *(1/2)*



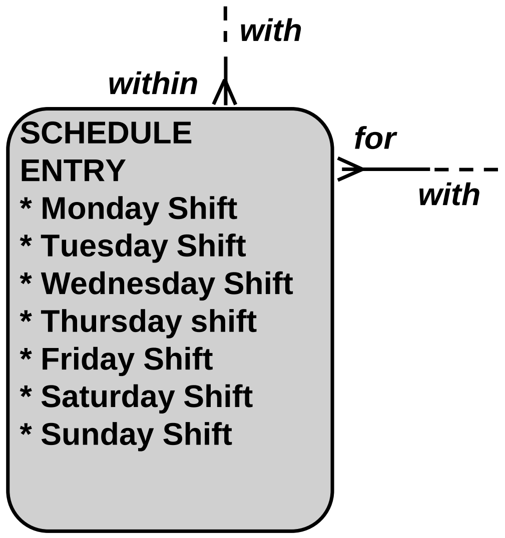
ORACLE

A-15 Copyright © Oracle Corporation, 2002. All rights reserved.

Practice 2-5 Schedule: Solution

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.

Solution: Schedule *(2/2)*



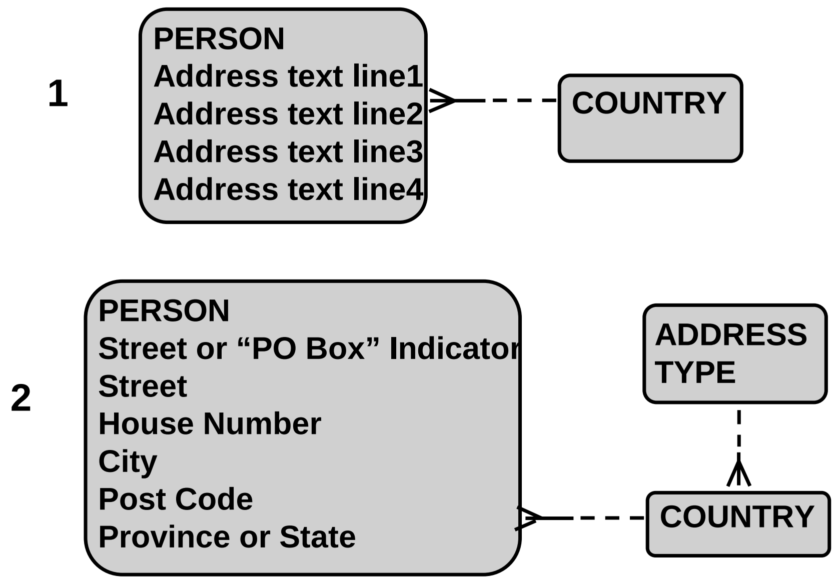
A-16

Practice 2-5 Schedule: Solution

An alternative solution models SCHEDULE ENTRY differently:

The first model does allow for more than one shift per employee per day; the second does not.

Solution: Address



Practice 2-6 Address: Solution

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

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

- **Solution 1 can easily cope with the various address formats. It simply recognizes the fact that an address may consist of 4 lines of text at maximum. This avoids the need to interpret a given address. Is 1045 ST 88 the house numbered 1045 on Street 88 or is it the house numbered 88 at 1045 Street?**
- **Solution 2 cuts the address into individual pieces and accounts for post boxes as well. It assumes the interpretation is no problem. By adding entity ADDRESS TYPE the sequence of display of the various address pieces can be set.**

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

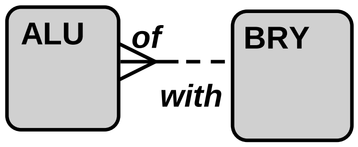
3 Check if your model would be different if the system is also required to have facilities to search addresses in the following categories. Make the necessary changes, if any.

Practice 2-6 Address: Solution

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 RD**
- **In Reading**
- **The second model does allow most of the required queries, although the last one may need some trials. If a system needs to allow quick and easy searches on the address information, often a separate attribute is added that stores the address information in uppercase using a particular set of rules to sort street name elements and to abbreviate.**
- **For example, a street name like "President A. B. Collins Road" would be taken as COLLINS, RD PRES A B. Such an attribute is not derivable from the address line unless the system has a built-in parser for this purpose.**

Solution: Read the Relationship

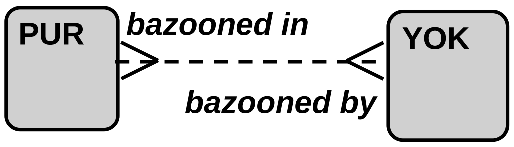


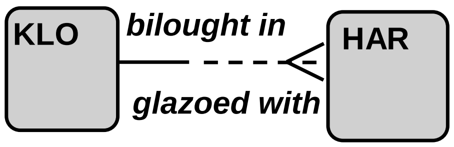
Every ALU must be of exactly one BRY

Every BRY may be with one or more ALUS

Every PUR may be bazooned in one or more YOKS

Every YOK may be bazooned by one or more PURS





Every KLO must be bilought in one or more HARS

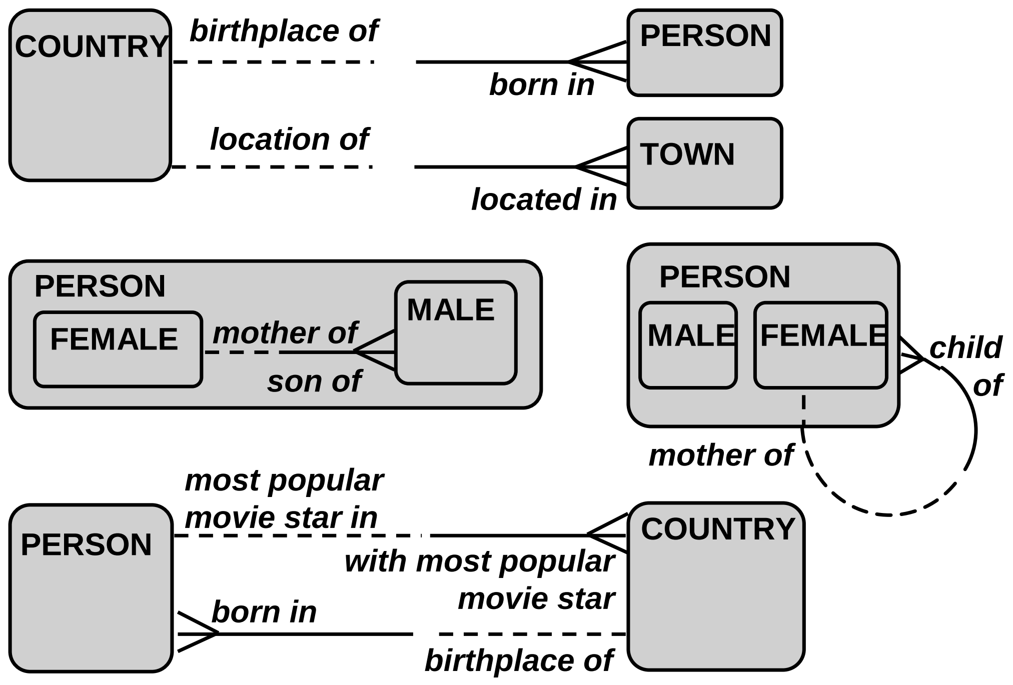
Every HAR may be glazoed with exactly one KLO

Practice 3-1 Read the Relationship: Solution

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.

Even if the phrases sound as if they come from some strange Science Fiction movie, make sure you are fluent in reading them. Only start judging what you have just heard after reading the lines. In a real life context, you sometimes read what you *think* is represented, rather than what is actually modeled. It is really hard to switch off your interpretation machine.

Solution: Find a Context

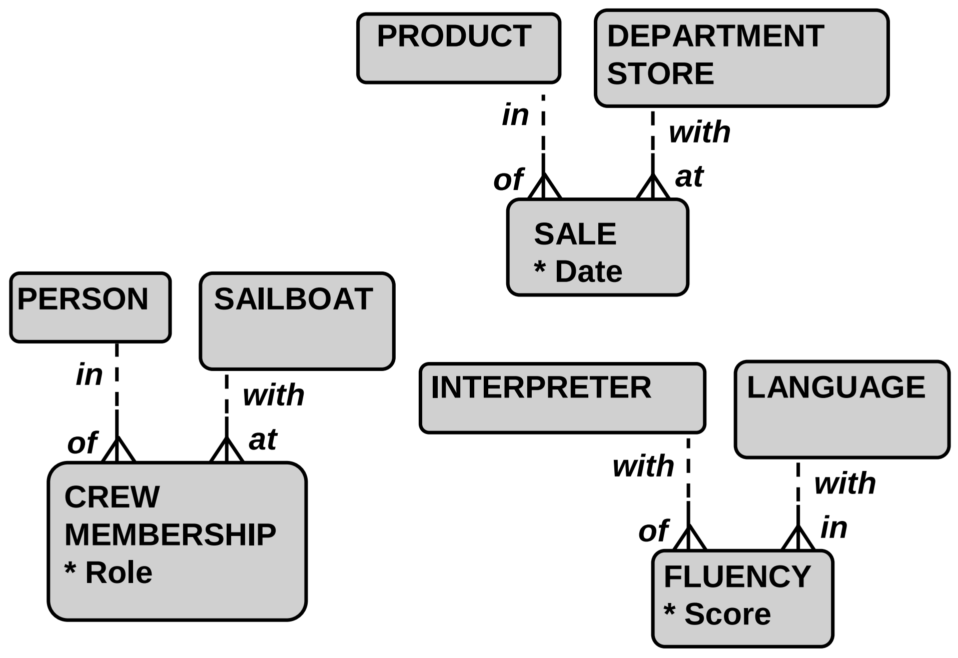


Practice 3-2 Find a Context: Solution

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

Here are just some examples of possible contexts.

Solution: Name the Intersection Entity



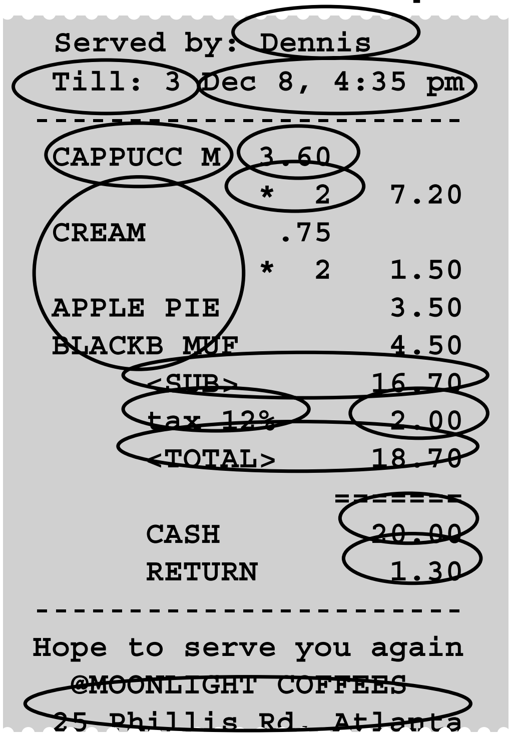
Practice 3-3 Name the Intersection Entity: Solution

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.

These are only possible solutions, based on just one possible context and one assumption on desired functionality.

Solution: Receipt



```
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
```

Practice 3-4 Receipt: Solution

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

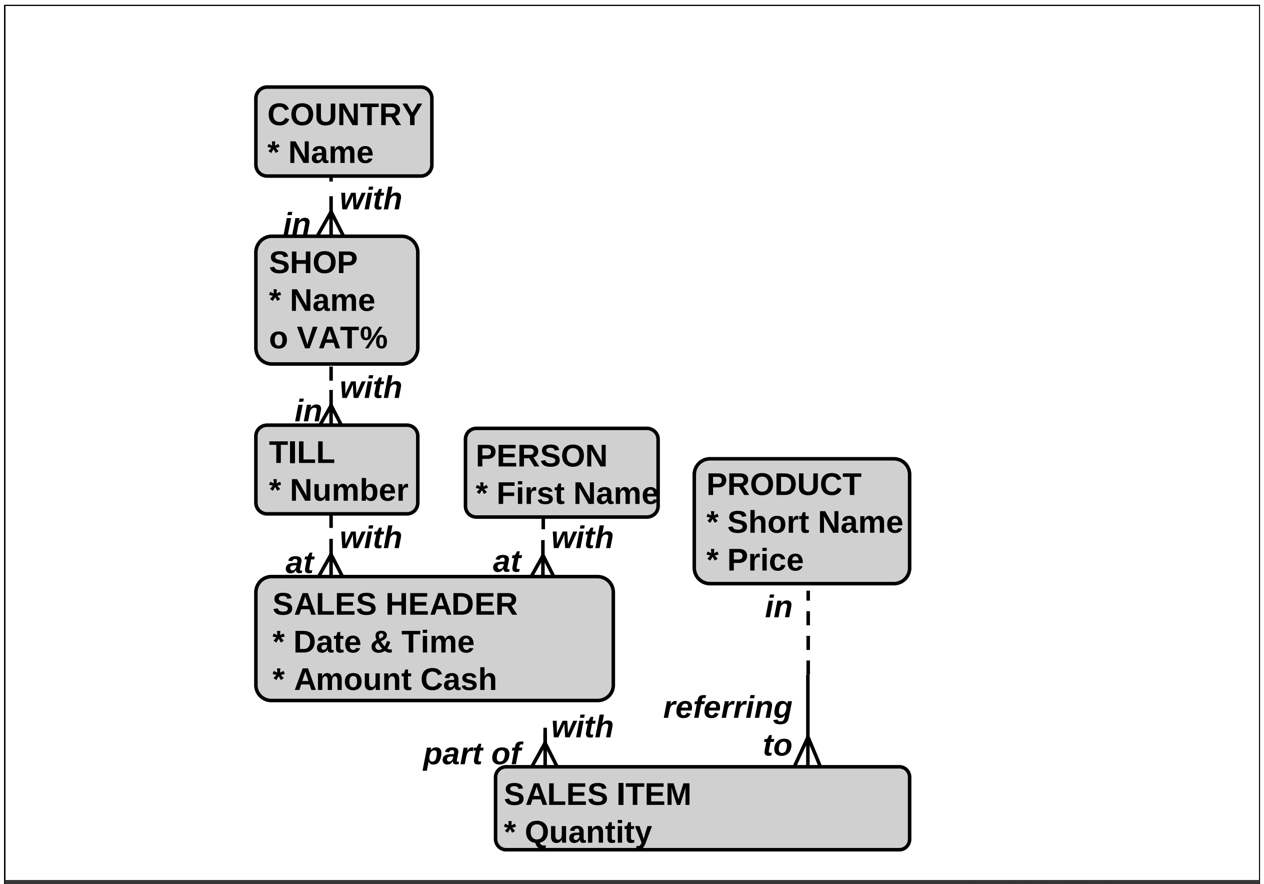
An assumption is that all, non-derivable, information that is displayed on the receipt must be stored for at least some time.

This model is based on several assumptions. One is that all entities may exist in isolation: for example, there may be countries (recorded in the system) without shops, shops without tills, and tills without sales. This explains all 1:m relationships that are optional at the 1 side.

Sales tax or VAT percentage for these kinds of products is a fixed value at a local shop level, such as in, for example, the USA, or at Country level, such as for example in many European countries. In this model the choice is made to model it at shop level.

Many countries know two VAT percentages: a low percentage for normal and a high percentage for luxury goods. Moonlight may have to deal with both as mineral water and coffee possibly fall in different categories.

The Attribute Price of PRODUCT could be modeled as a separate entity as products may have various prices over time.



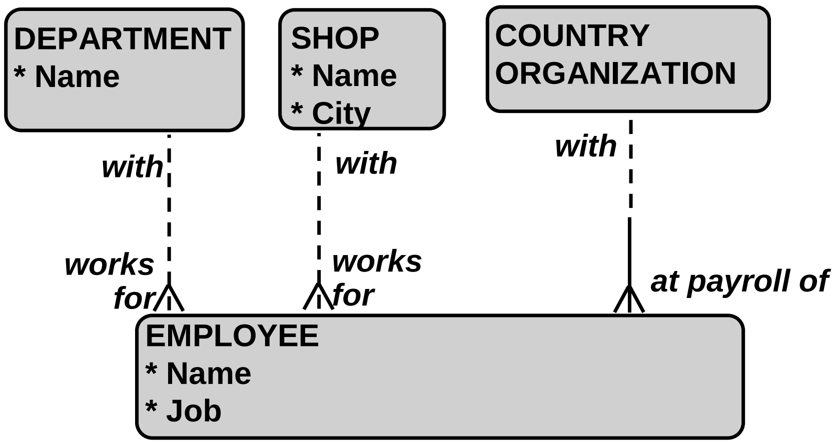
Practice 3-5 Moonlight P&O: Solution

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

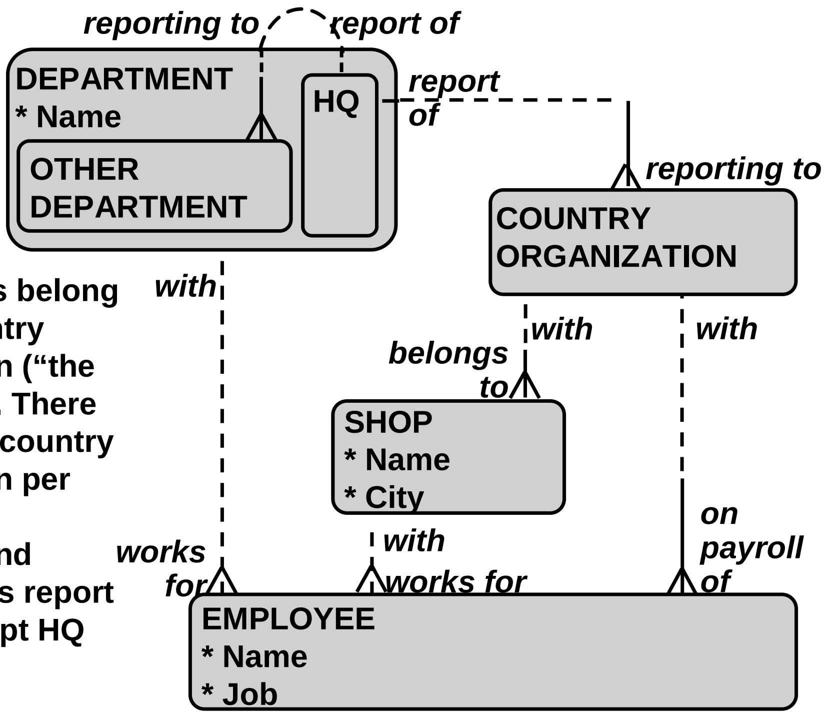
- **Note the optional relationships from EMPLOYEE to DEPARTMENT and SHOP. These result from "... country organizations *or* for a shop."**

Solution: Moonlight P&O (1)

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 …



Solution: Moonlight P&O (2)

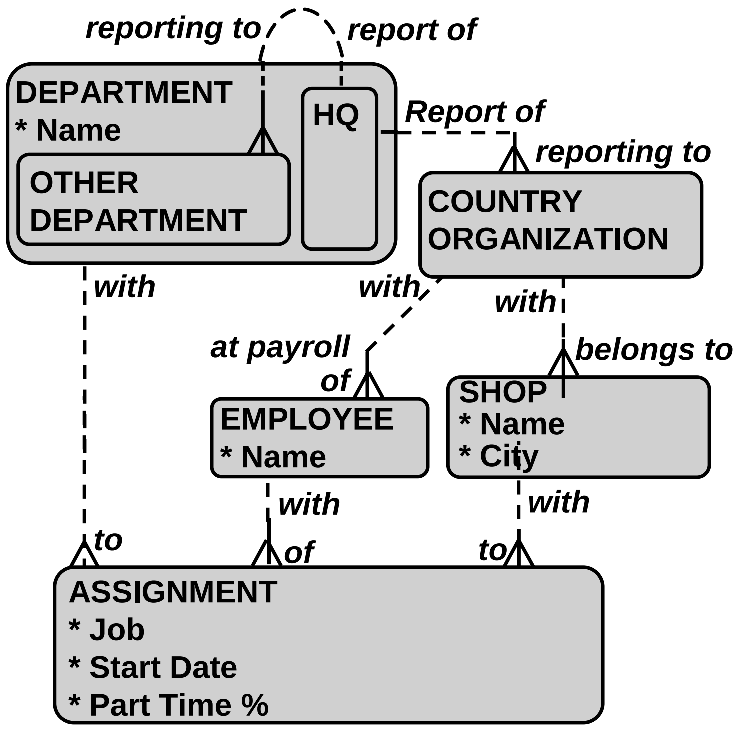


… All shops belong to one country organization (“the countries”). There is only one country organization per country. All countries and departments report to HQ, except HQ itself …

Practice 3-5 Moonlight P&O: Solution

2 Extend or modify the diagram based on this information:

Solution: Moonlight P&O (3)



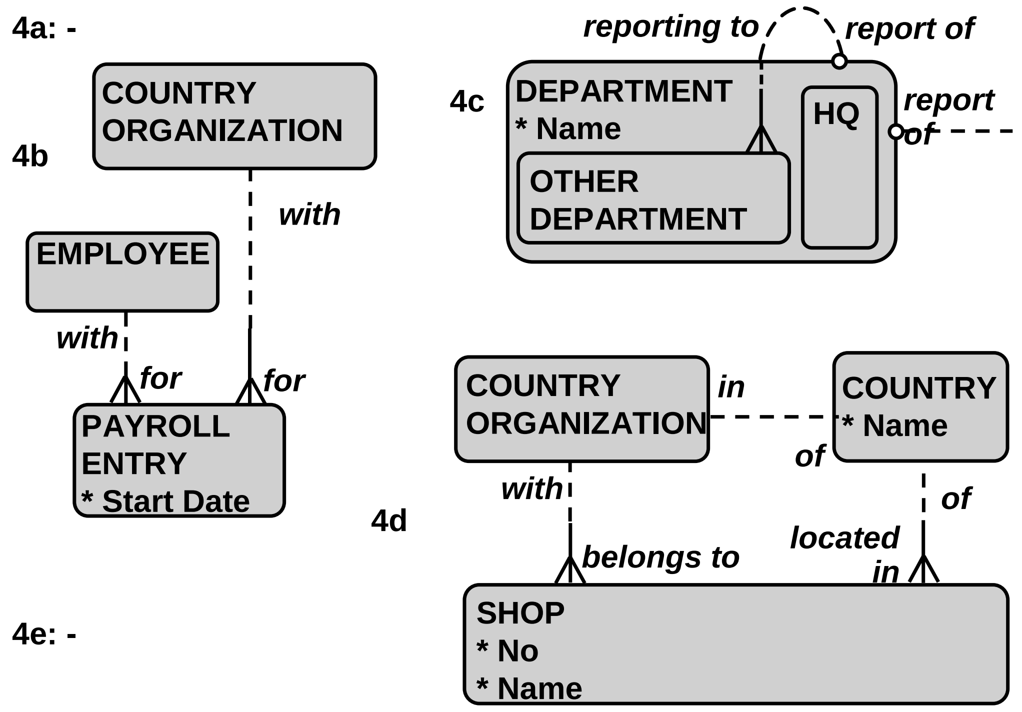
… Employees can work part time. Lynn has had an 80% assignment for Product Development since the 1st September. Before that she had a full-time position.

A-26

Practice 3-5 Moonlight P&O: Solution

3 And again:

Solution: Moonlight P&O (4)



Practice 3-5 Moonlight P&O: Solution

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.

a. Does not require changes in the model. People can be employed in various places-the model allows that.

e To prevent conflicting responsibilities, employees are not allowed to work for a department and for a shop at the same time.

b. See diagram.

b Last year Tess resigned in Brazil as a shop manager and moved to Toronto. Recently she joined the shop at Toronto Airport.

c. See diagram.

c To reduce the number of direct reports, departments and country organizations may also report to another department instead of Headquarters.

d. See diagram.

d The shops in Luxembourg report to Belgium.

e. Cannot be modeled with what you have learned so far.

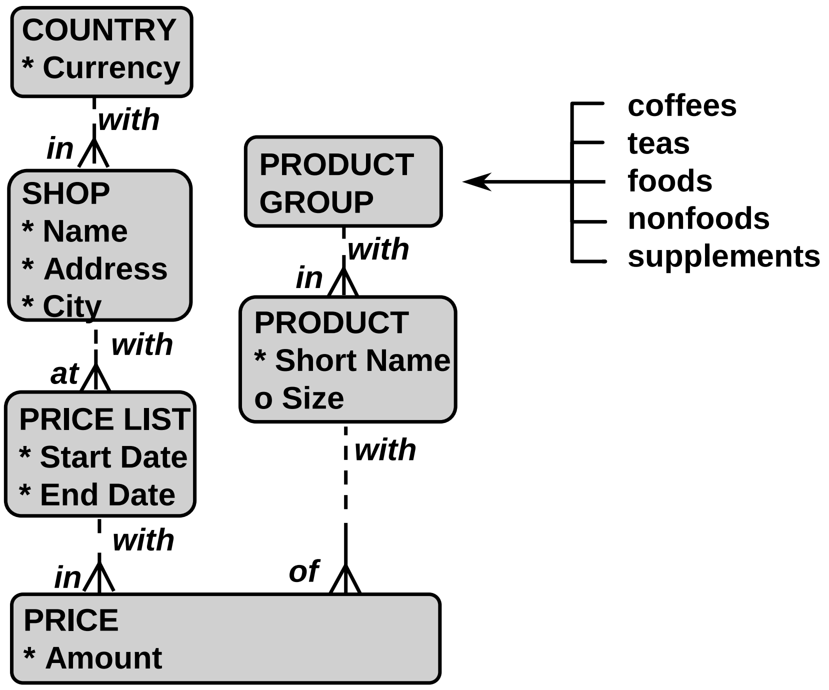
Practice 3-5 Moonlight P&O: Solution

1. 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?
     a, b. Given the previous models, these questions can be answered.
   - c Are there currently any employees working for Marketing in France?
     c. Departments are not related to Countries-so there is no Marketing in France. However, we can find who (if anyone) is working for the Marketing Department who is also on the French payroll.
   - d What is the largest country in terms of number of employees? In terms of managers? In terms of part-timers?
     d. The question can be answered by counting the distinct current assignments per country organization.
   - e When can we celebrate Lynn's fifth year with the company? When can we do the same with Tess' fifth year with Moonlight?
     e The model does not (yet) contain an attribute End Date for ASSIGNMENT and PAYROLL ENTRY. Given Tess' career at Moonlight, there is a need for attribute End Date as the periods are not necessarily contiguous.
   - f What country has the lowest number of resignations?
     f Can be answered.

Note:

- This model is quite complex, as both departments and country organizations are part of the organization. Departments do not have a location, unlike country organizations. Departments are 'virtual' organizations. Think about a department as a collection of tasks rather than a group of people. A department can still exist even if there are no people working for it - the tasks still remain. The location of the department may be defined by 'the office of that country organization which currently has the manager of the department on its payroll.'
- A country organization is not the same as a country. It is wise to distinguish the two as countries are beyond the scope of Moonlight's power, whereas country organizations can be defined by Moonlight. A country organization in Haiti may not (yet) exist, but the country does.

Solution: Price List



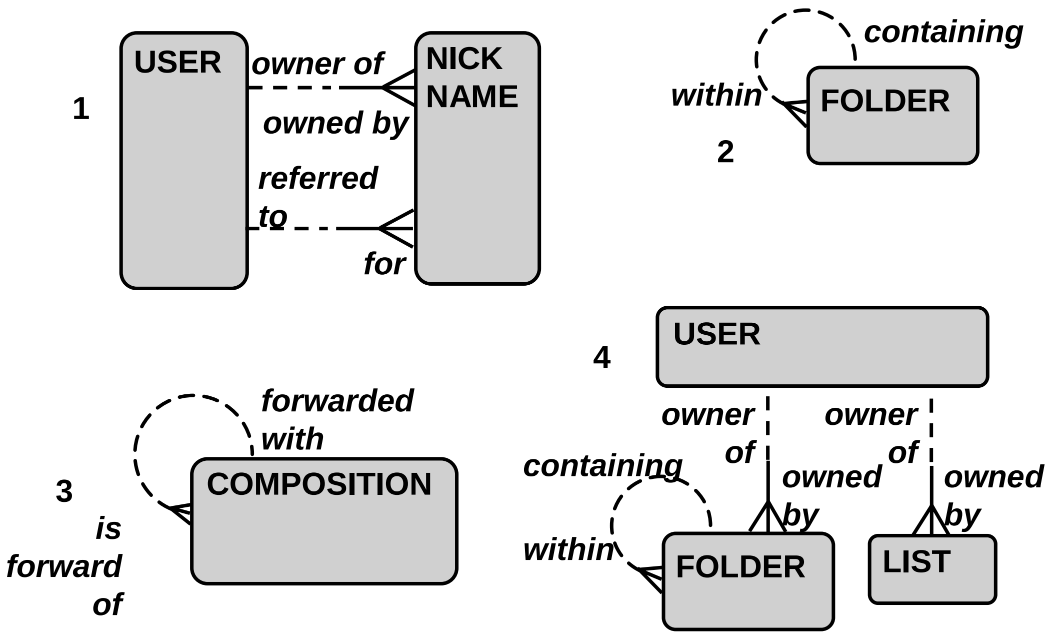
Practice 3-6 Price List: Solution

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

One of the questions that arises here is the following: is *decaffeinated* a PRODUCT? In some respects it behaves differently from other products-you cannot buy it separately, nor in combination with, for example, apple pie. On the other hand it is something you can buy, and it has a price just like the other products.

In this solution, decaffeinated is regarded as a product in a special product group: supplements (maybe you have a suggestion for a better name...). Products in this group need special attention, but otherwise can be treated as products. Note also the entity PRODUCT GROUP. PRODUCT GROUP instances are represented by the lines between the various products on the price list.

Solution: EMail

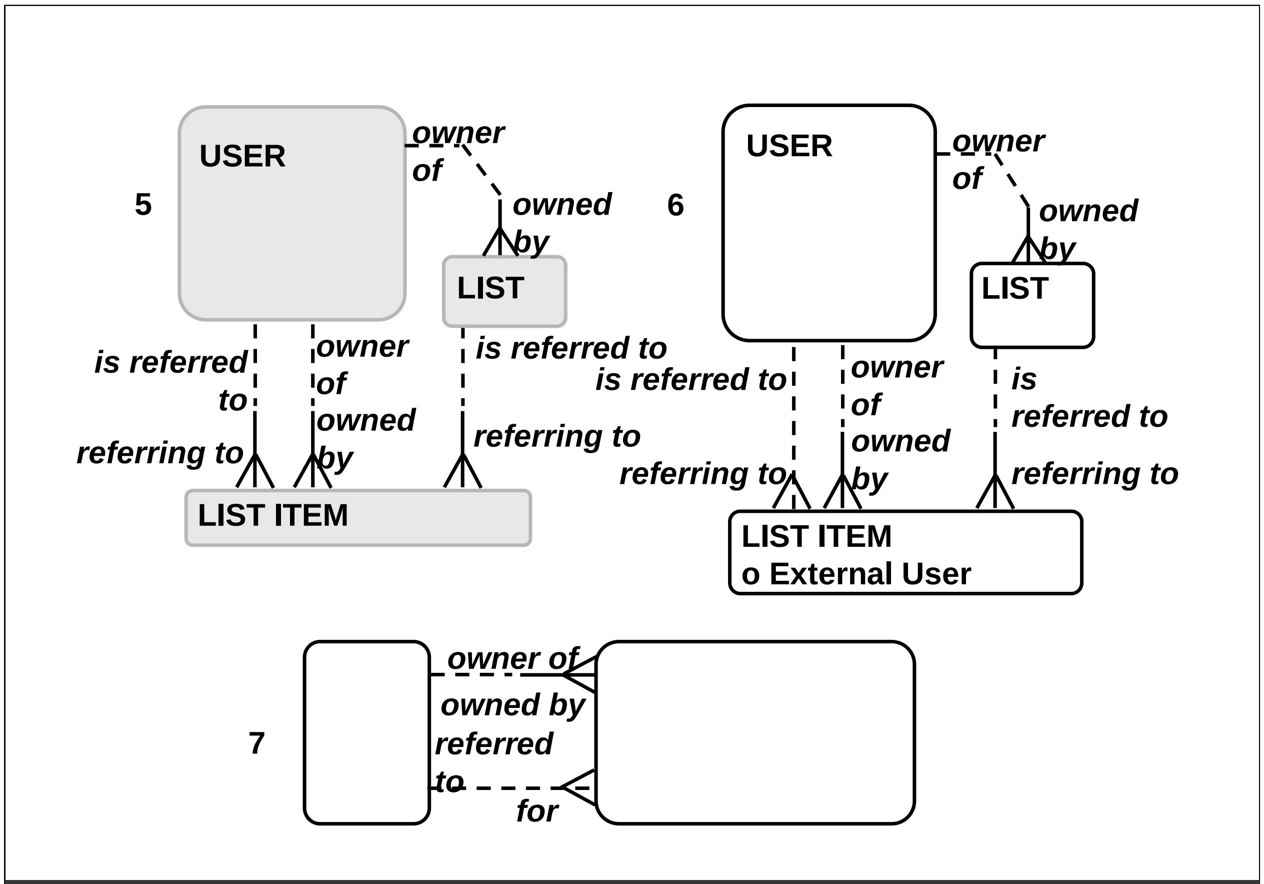


Practice 3-7 E-mail: Solution

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

See the numbers in the illustration for the various assignments.

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.



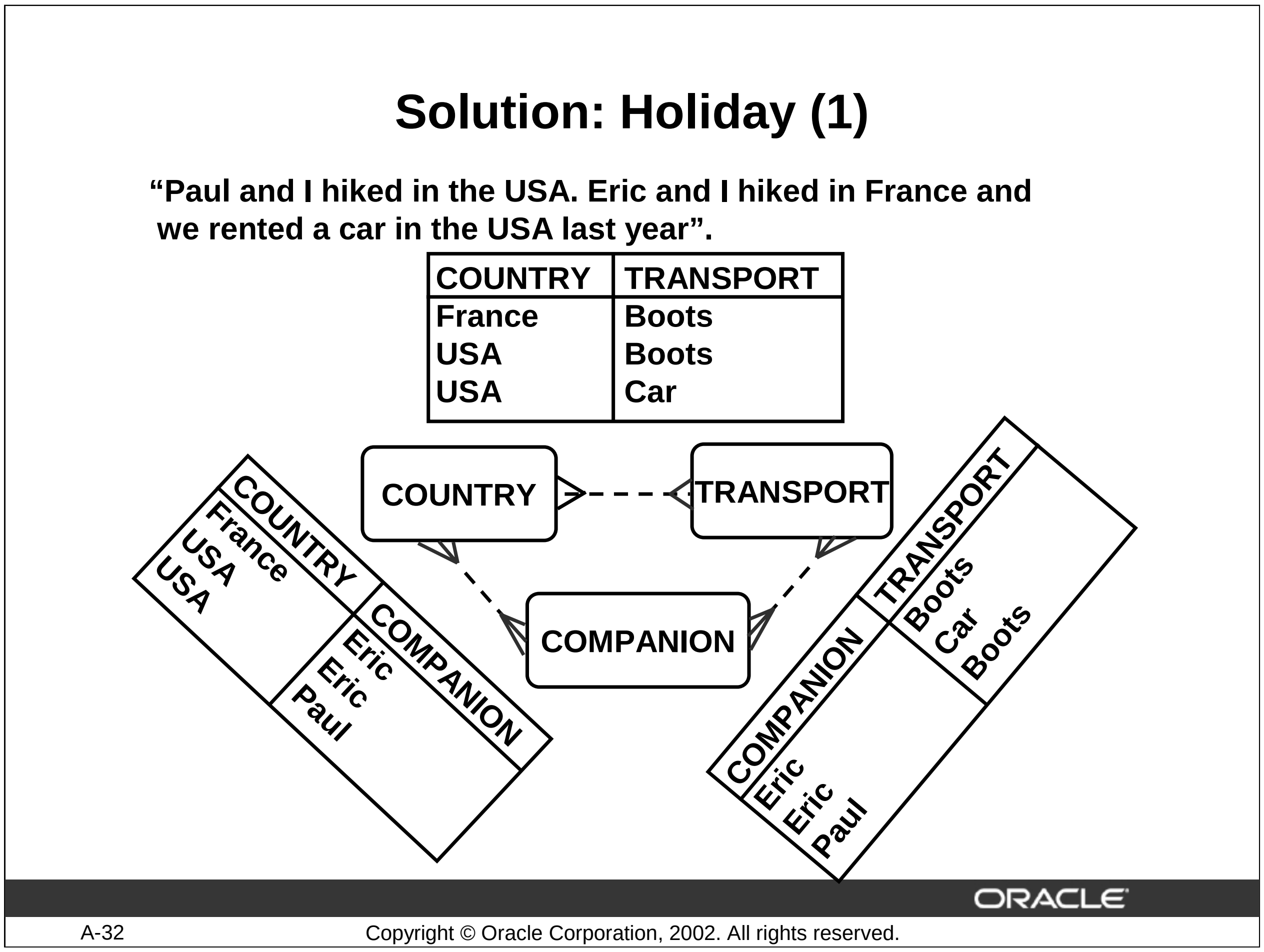
Practice 3-7 E-mail: Solution

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”.

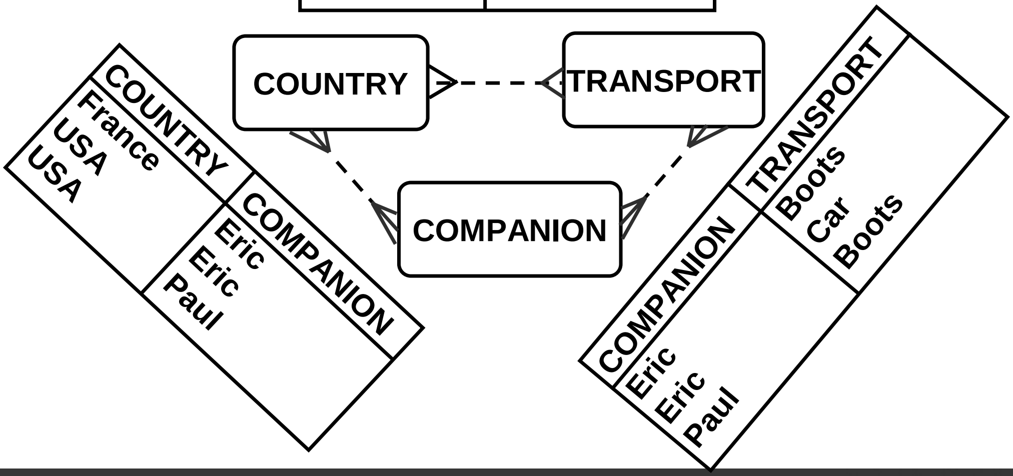
7 A nickname may be an alias for an external address.



Solution: Holiday (1)

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

COUNTRY	TRANSPORT
France	Boots
USA	Boots
USA	Car



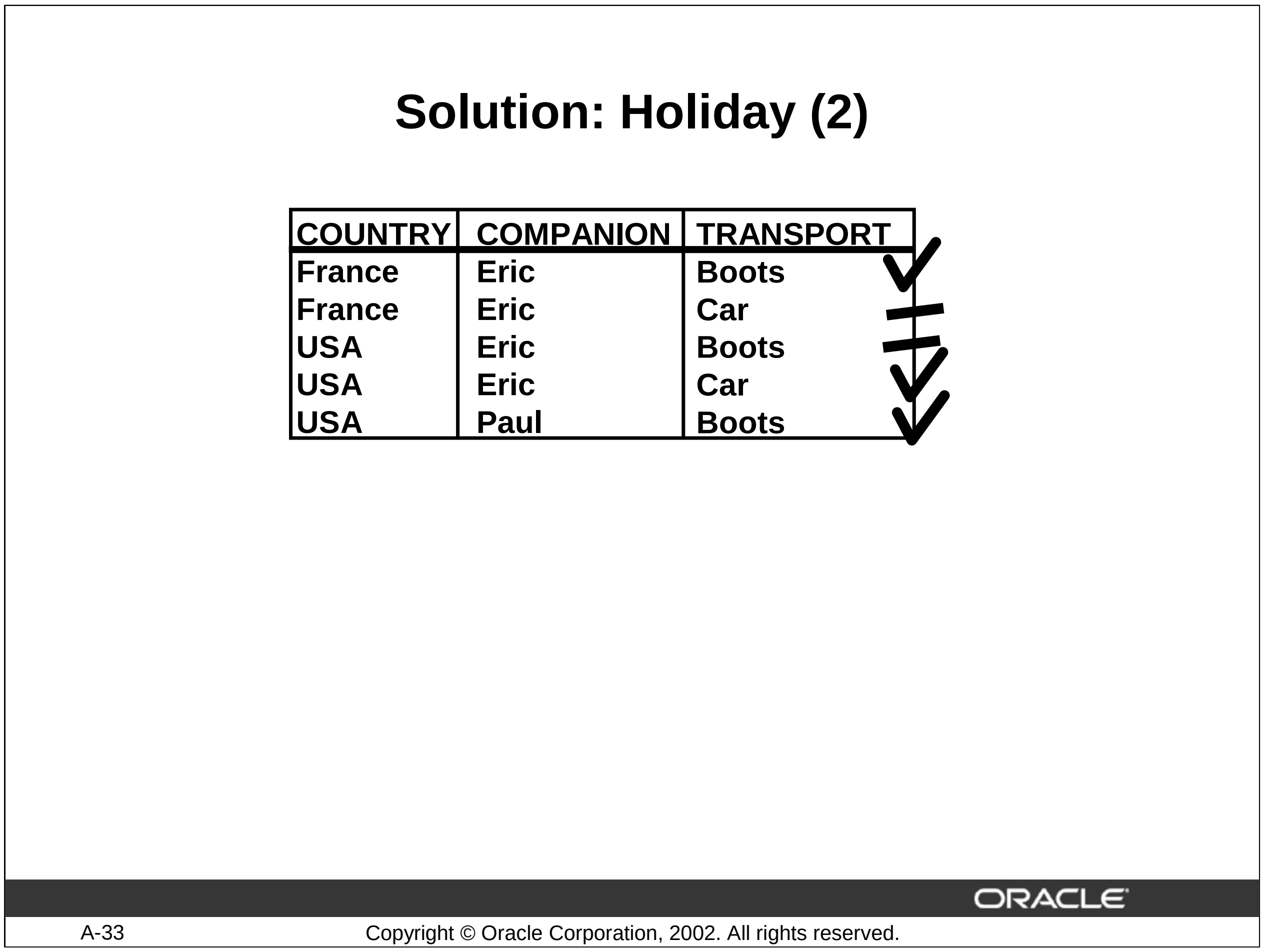
COUNTRY	COMPANION
France	Eric
USA	Eric
USA	Paul

TRANSPORT	COMPANION
Boots	Eric
Car	Eric
Boots	Paul

A-32 Copyright © Oracle Corporation, 2002. All rights reserved.

Practice 3-8 Holiday: Solution

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



Solution: Holiday (2)

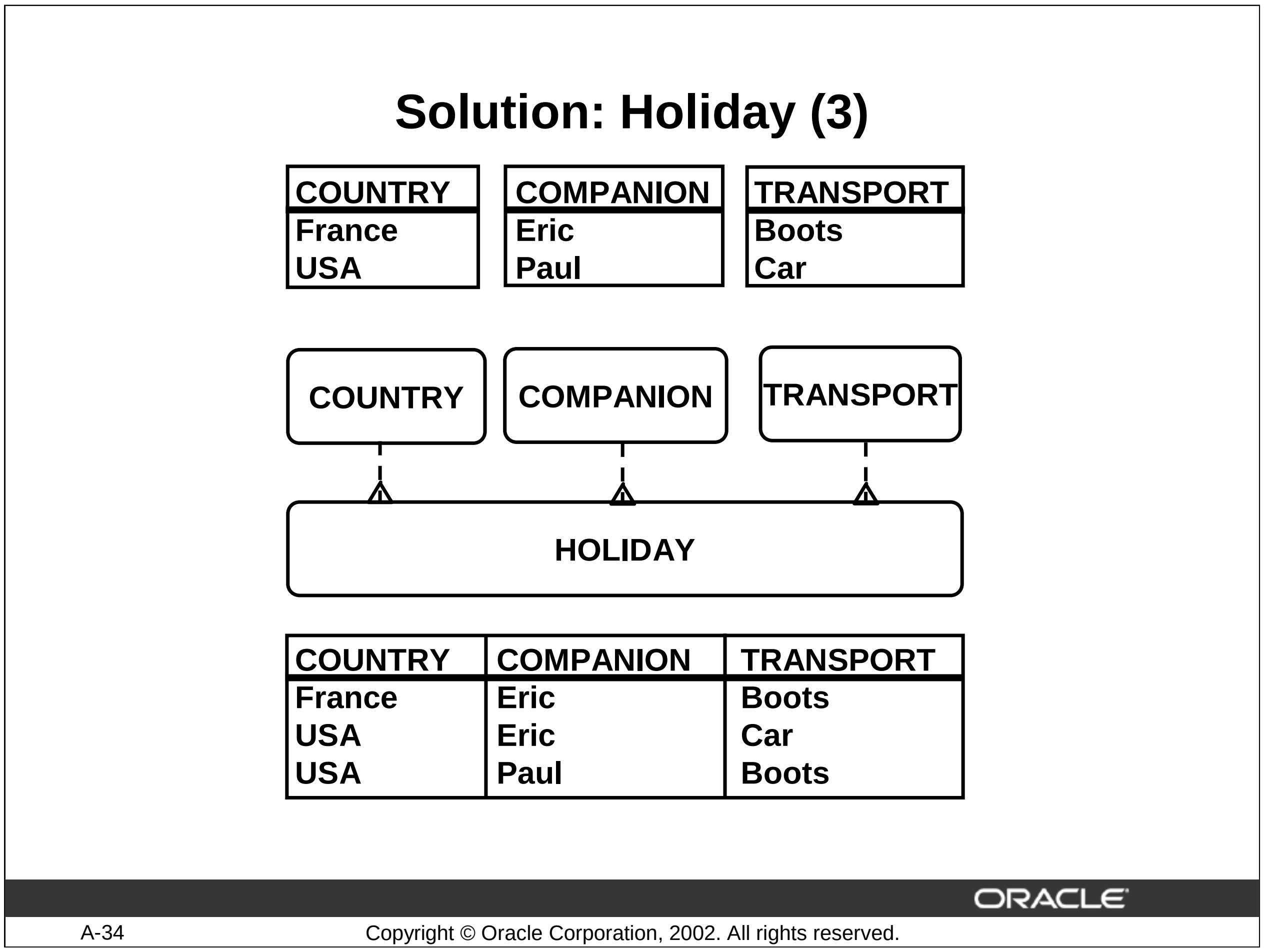
COUNTRY	COMPANION	TRANSPORT	
France	Eric	Boots	✓
France	Eric	Car	✗
USA	Eric	Boots	✗
USA	Eric	Car	✓
USA	Paul	Boots	✓

A-33 Copyright © Oracle Corporation, 2002. All rights reserved.

When you join the information in COUNTRY-COMPANION table and COMPANION-TRANSPORT table, this is the result:

This is obviously wrong. The model somehow generated more than was put into it!

The reason? It is the combination of Paul, Hiking, USA that is important, just as the combination of Eric, Hiking, France is important. It is the combination of the *three* values that matter. See the model and table structure in the next illustration.

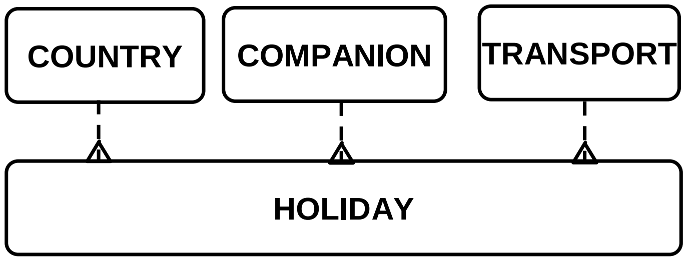


Solution: Holiday (3)

COUNTRY
France
USA

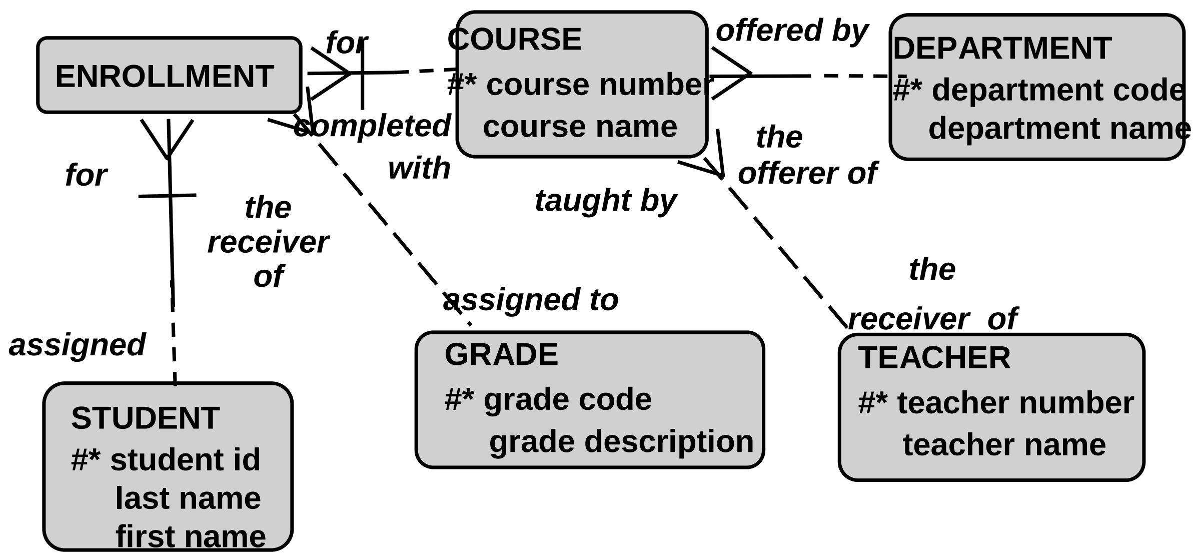
COMPANION
Eric
Paul

TRANSPORT
Boots
Car



COUNTRY	COMPANION	TRANSPORT
France	Eric	Boots
USA	Eric	Car
USA	Paul	Boots

Practice: Normalize an ER Model: Solution



Practice 3-9: Normalize an ER Model : Solution

Goal

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

Your Assignment

- 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.

STUDENT Entity

- This entity is in 3NF with all attributes properly placed.

COURSE Entity

- This entity is only in 2NF. department name, department code, teacher number and teacher name violate 3NF and are misplaced.
- department name and department code depend on each other. Move them to a separate DEPARTMENT entity with a relationship to course. Teacher name and teacher number depend on each other. Move them to a separate TEACHER entity with a relationship to COURSE.

Practice 3-9: Normalize an ER Model : Solution

ENROLLMENT Entity

- This entity is only in 1NF. Teacher number and course name violate 2NF and are misplaced. Grade code and grade description violate 3NF and are misplaced.
- Teacher number and course name are dependent only on the COURSE subset of the UID so move them o the COURSE entity.
- Grade description is dependent on the non-UID attribute grade code and vice-versa. Move both to a GRADE entity with a relationship to ENROLLMENT.
- Optionally, re-draw the ER diagram in third normal form.

Goal

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

Your Assignment

- 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.

STUDENT Entity

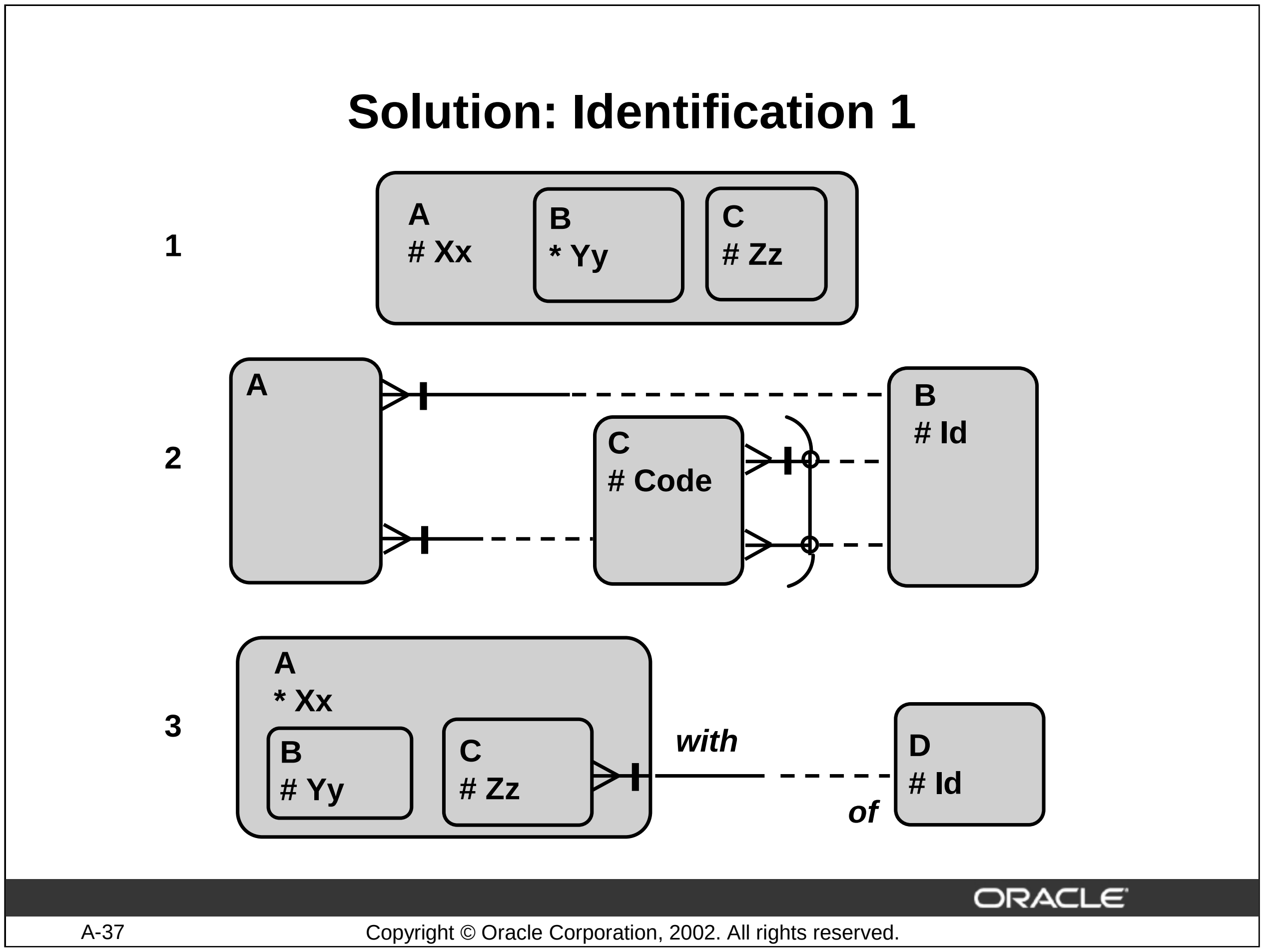
- **This entity is in 3NF with all attributes properly placed.**

COURSE Entity

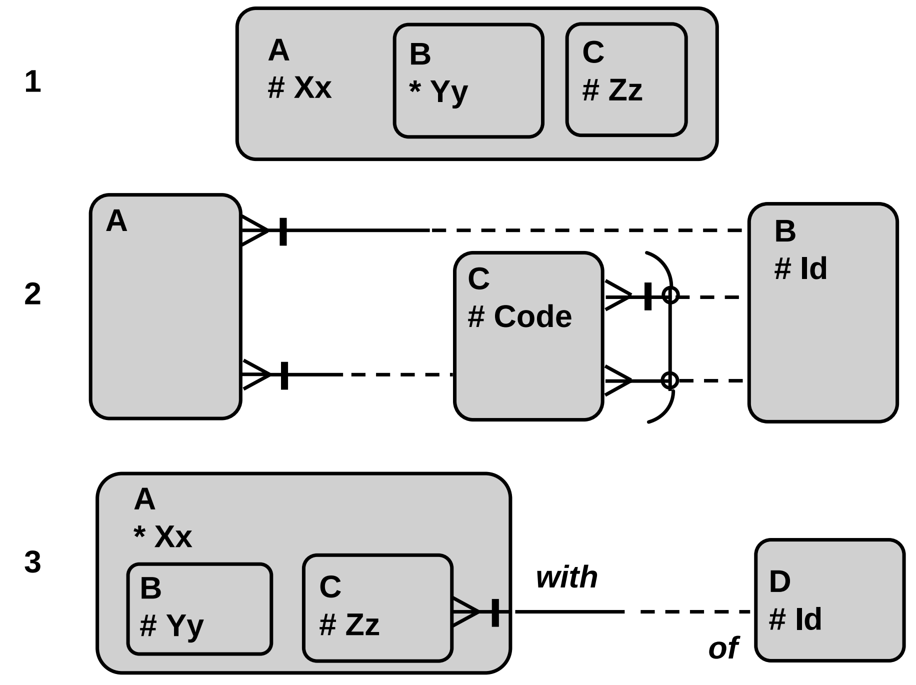
- This entity is only in 2NF. department name, department code, teacher number and teacher name violate 3NF and are misplaced.
- department name and department code depend on each other. Move them to a separate DEPARTMENT entity with a relationship to course. Teacher name and teacher number depend on each other. Move them to a separate TEACHER entity with a **relationship to COURSE.**

ENROLLMENT Entity

- This entity is only in 1NF. Teacher number and course name violate 2NF and are misplaced. Grade code and grade description violate 3NF and are misplaced.
- Teacher number and course name are dependent only on the COURSE subset of the UID so move them o the COURSE entity.
- Grade description is dependent on the non-UID attribute grade code and vice-versa. Move both to a GRADE entity with a relationship to ENROLLMENT.
- Optionally, re-draw the ER diagram in third normal form.



Solution: Identification 1

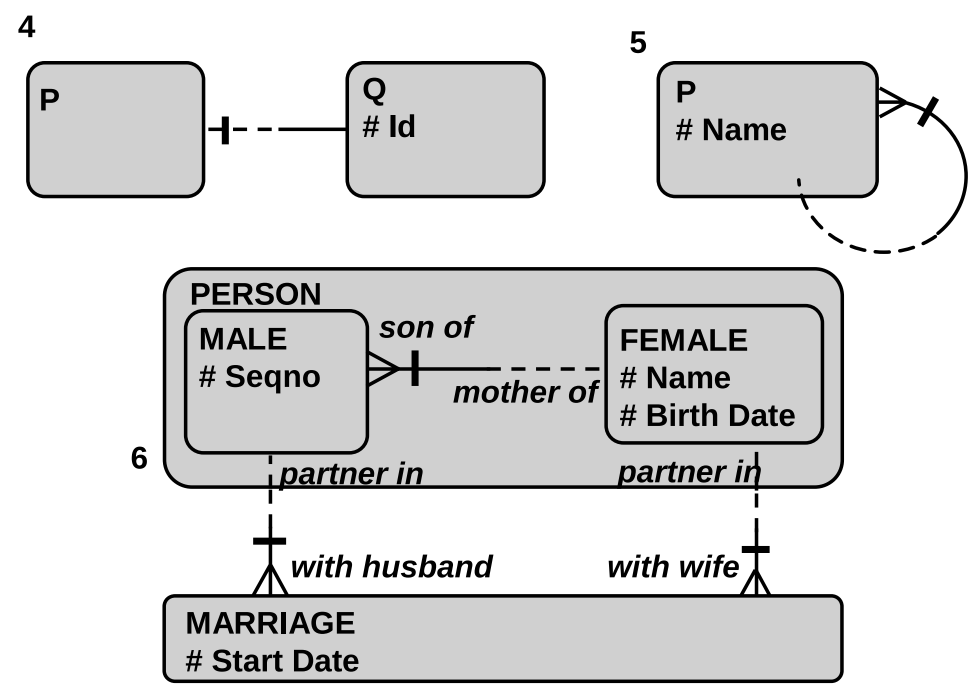


Practice 4-2 Identification: Solution

Are the entities in the next diagrams identifiable?

1. **Because every A is identifiable by attribute Xx, every B and C are as well.**
2. **B is identifiable by Id. A is identifiable, only if B and C are. This leaves C. Because of the arc, not every C will have the barred relationship with B. But that leaves C's unidentified.**
3. **D is identifiable by Id. Every C is identifiable by ZZ and D. Every B is identifiable by Yy. As every B and every C is identifiable, so is every A.**

Solution: Identification 2



Practice 4-2 Identification: Solution

1. **Every Q is identifiable by Id. P's that are related to an instance of Q are identifiable by that instance, but not every P is related.**
2. **Conceptually there is no problem here. Every P is identified by its Name and the reference to its "predecessor". You could argue about how to deal with the first P, but the question in return would be: who told you there is a first P? More of a problem is the fact that the identification of an instance of the nth generation takes n+1 names. This in particular makes this type of identification highly impractical.**

   Note: the next model describes a context that may be different from the world you are familiar with.
3. **Entity FEMALE is identified by Name and Birth Date. This may not be true for the entire population of the world, but may be true for a much smaller set of people a business is interested in.**
4. **Entity MALE is identifiable as well: I am the only second son of my mother.**
5. **Every PERSON is identifiable because every person is female or male. Entity MARRIAGE is identifiable by the combination of both MALE and FEMALE that participates and the Date of the marriage.**

Practice 4-2 Identification: Solution

7. Given the above model, answer the following questions.
   - a Can person A marry twice?

     Yes
   - b Can person A marry twice on the same day?

     Yes
   - c Can person A marry with person B twice?

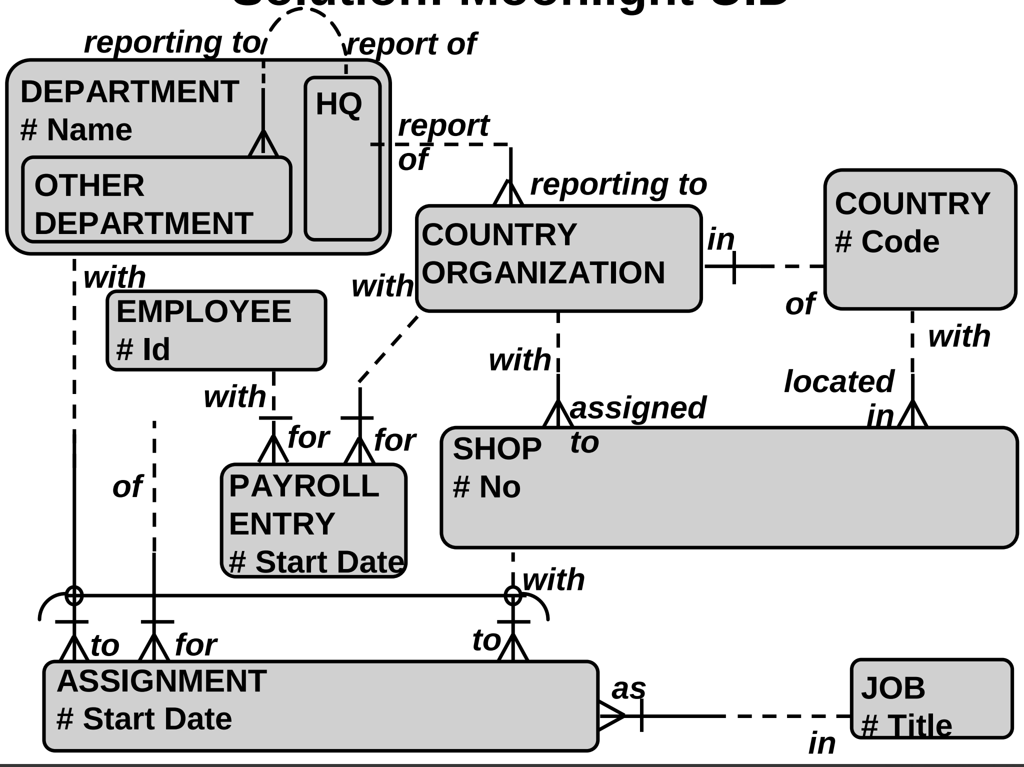
     Yes
   - d Can person A marry with person B twice on the same day?

     No
   - e Can person A be married to person B and person C simultaneously?

     Yes
   - f Can person A be married to person A?

No, because a marriage is always between a male and a female and person. A cannot be both male and female as these are distinct subtypes of person.

Solution: Moonlight UID



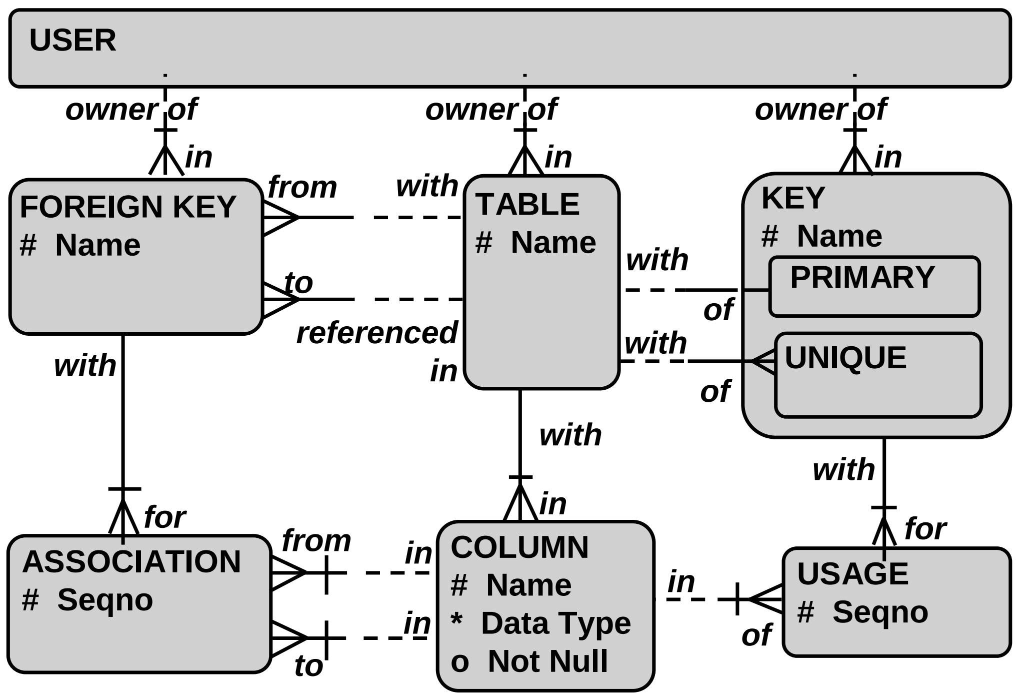
Practice 4-3 Moonlight UID: Solution

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?

For transparency reasons, only the attributes that are needed for identification are added in this model.

Solution: Table



Practice 4-4 Tables: Solution

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.

Erroneous:

- **Both relationships from FOREIGN KEY to TABLE should not be barred.**
- **A Table is not identified by Name only, but also by the owning USER. The same is true for a KEY and a FOREIGN KEY.**
- **Every KEY, PRIMARY, UNIQUE and FOREIGN must be named uniquely per USER.**

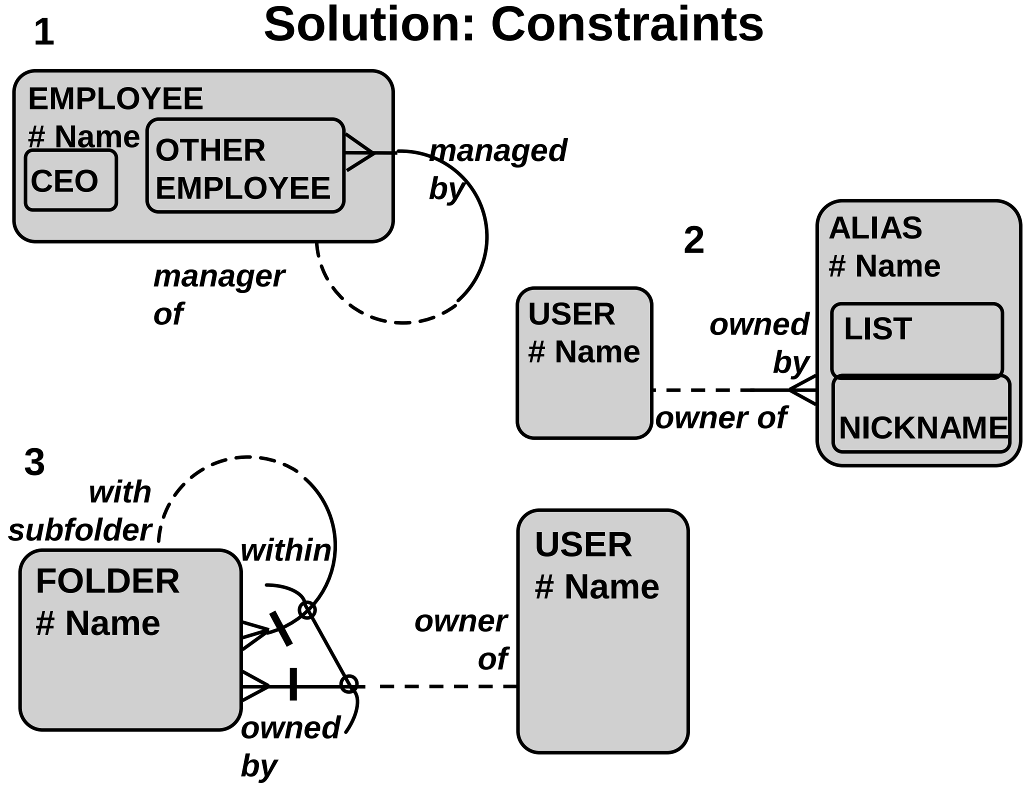
These errors have been removed in the diagram below.

Not shown:

- **COLUMN in a KEY must be a COLUMN of the TABLE the KEY belongs to.**
- **COLUMNS in a FOREIGN KEY must refer to COLUMNS in a KEY of the TABLE the FOREIGN KEY refers to.**

These constraints cannot be shown and must be listed separately.

Solution: Constraints

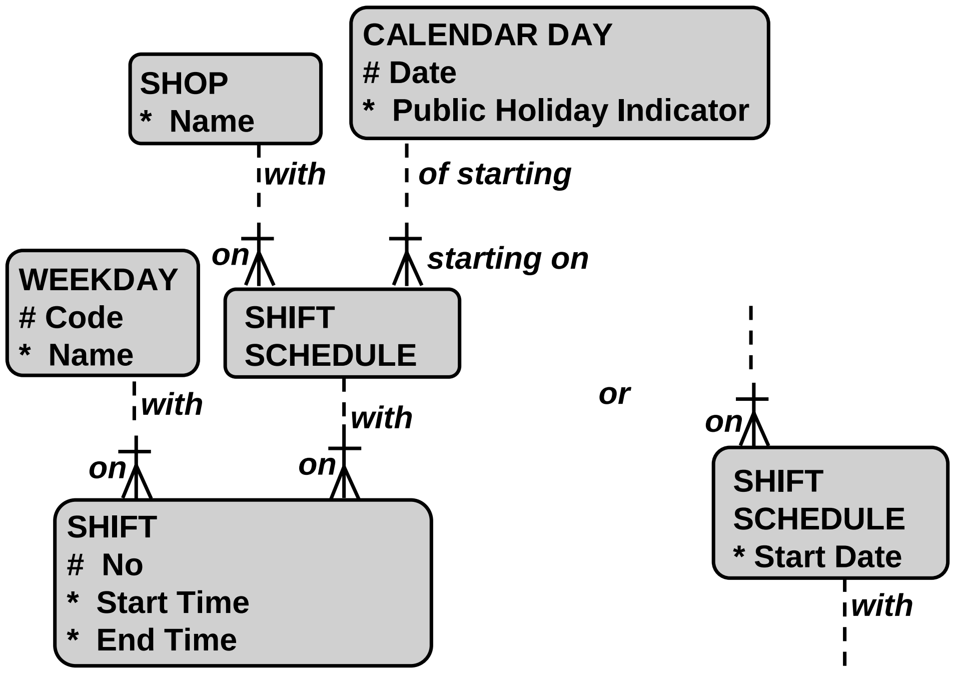


Practice 4-5 Constraints: Solution

Change the diagrams to model the constraint given.

1. Every EMPLOYEE must have a manager, except the Chief Executive Officer.
   - By creating a CEO subtype of EMPLOYEE the constraint is easily modeled. You may argue the use of this CEO subtype if modeling the constraint is the only reason for its existence.
2. A user may not use the same name for both NICKNAME and LIST name.
   - LIST and NICKNAME share the same namespace.This can be modeled with a supertype. Note the repositioning of attribute 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.
   - Adding the arc and barred relationships is enough. Note that the recursive relationship from FOLDER to FOLDER is made mandatory at the many end. See also the remark at Practice 4- Identification 5: Solution about recursive identification.

Solution: Shift



A-43

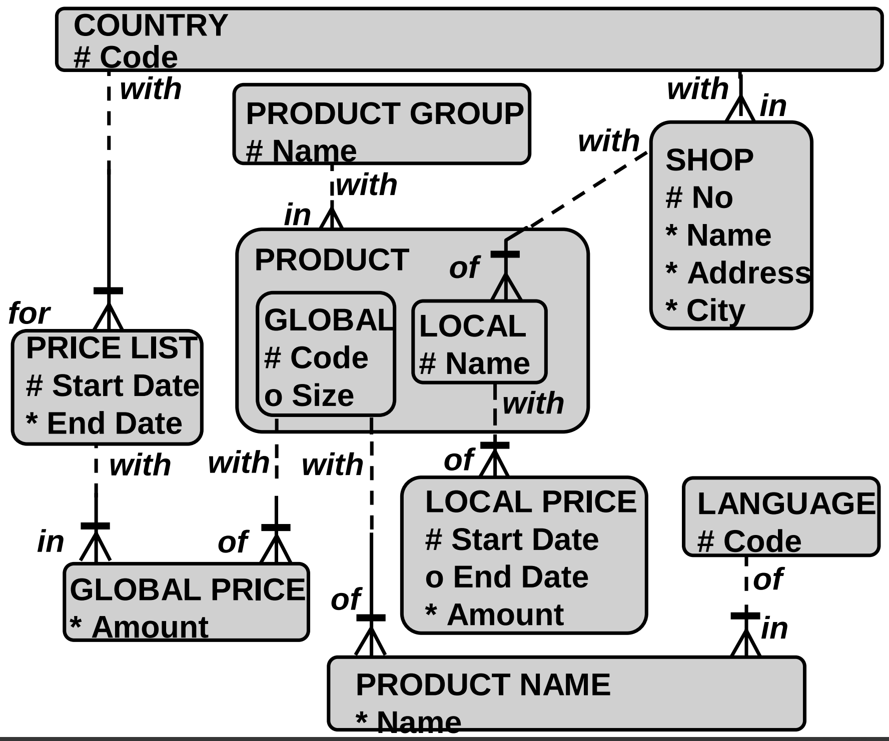
Practice 5-1 Shift: Solution

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

In this solution, entity CALENDAR DAY is used. This might be sensible in the context of a resource-planning environment where public holidays usually play a role. Be aware of the fact that the dates of public holidays are completely different around the world. In other terms, Public Holiday Indicator is not single valued in a international context.

In a global context the issue of daylight saving time (which is not common and certainly does not start at the same date around the world) and TIME ZONE play a role.

Solution: Moonlight Pricing



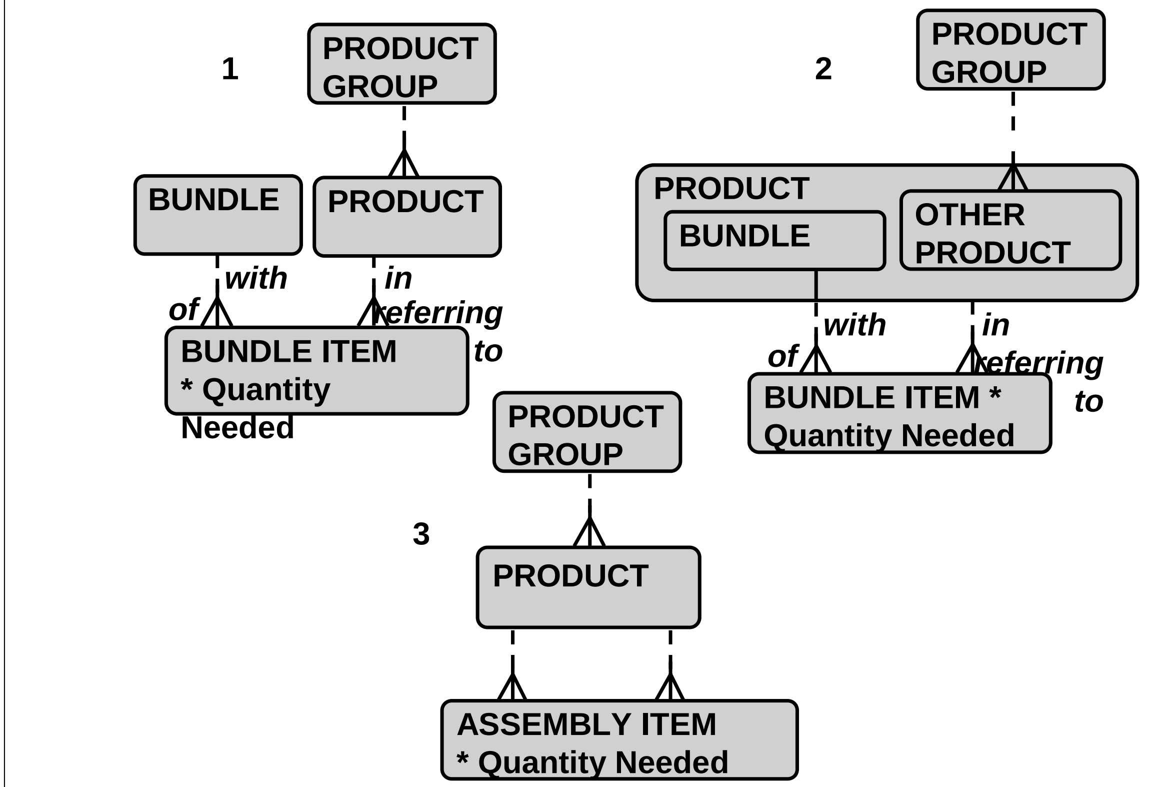
Practice 5-2 Strawberry Wafer: Solution

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

In this model several minor and major changes were made, compared to the model that was presented earlier (Practice 3 - Price List: Solution).

The major change is the issue of the prices that are either determined on national level or at shop level. This has lead to two price entities: GLOBAL and LOCAL PRICE. Note the different attributes and relationships of these two.

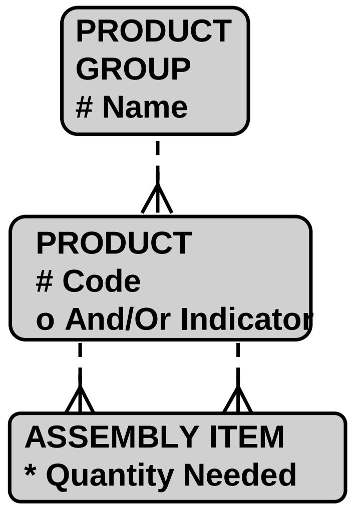
The price list also shows the same product range as the earlier one, but now products can be named in various languages.



Practice 5-3 Bundles: Solution

1. Modify the product part of the model in such a way that the desired calculations can be completed.
   - The first model is probably what you came up with first. A bundle consists of several products. You model this using the BUNDLE ITEM entity. But this solution is not correct as a bundle can consist of another bundle as well, such as SuperSweetTreat. Model 2 takes care of that. A BUNDLE ITEM can now consist of one or more instances of BUNDLE and instances of OTHER PRODUCT.
   - Note that model 2 assumes there is always a clear distinction between a product and a bundle, because they are modeled as subtypes. Model 3 does not assume that. A bundle is treated as any product here. You do not need to worry if you should consider apple pie with whipped cream as a bundle or as a single product.

DecafPunch (DP) = {Coffee (C) *or* Tea (T)} *and* {Blackberry Muffin (BM)}



Consider this as:
AS1 = (C *or* T)
DP = (AS1 *and* BM)

PRODUCTS

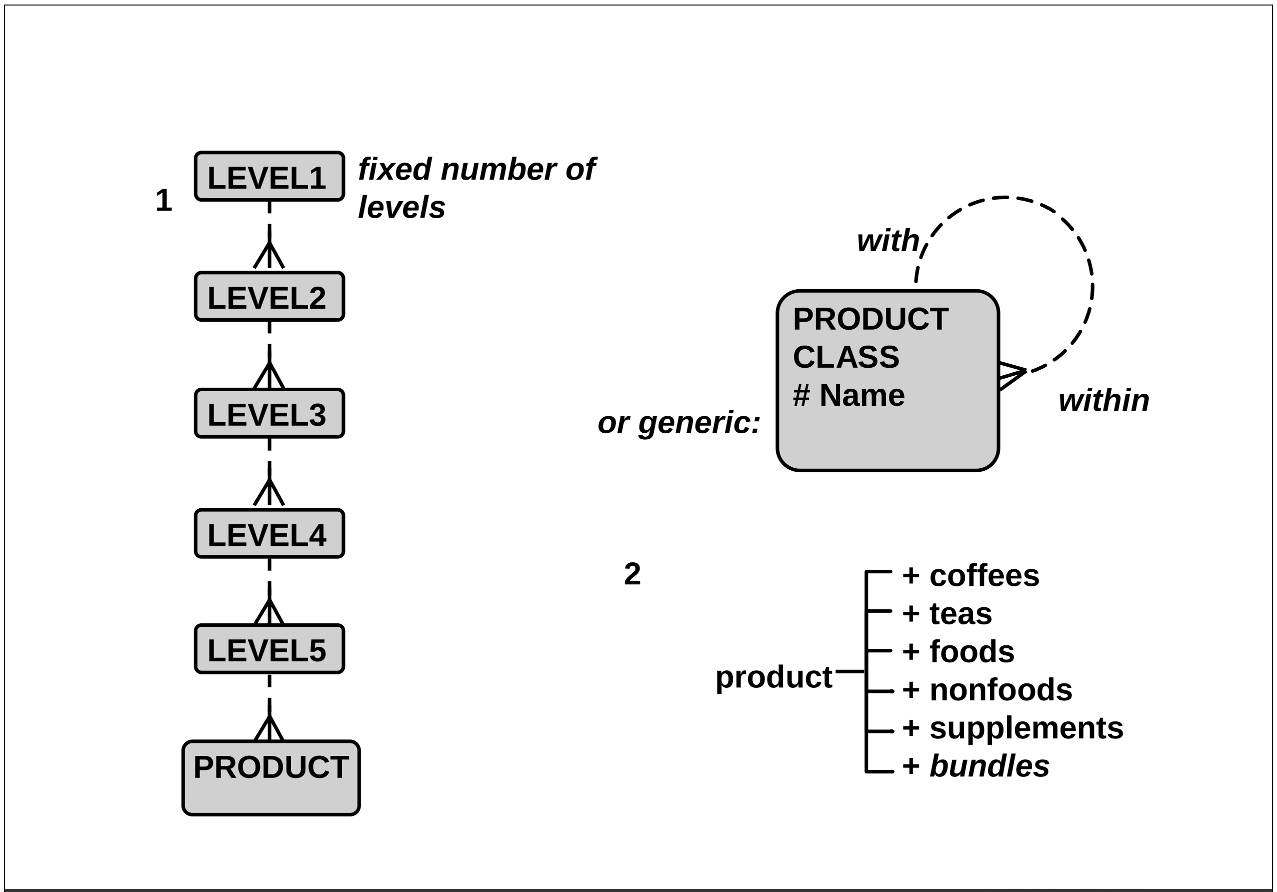
Code	And_or	Pg_name
C		..
T		…
AS1	OR	….
DP	AND	…..

ASSEMBLY ITEMS

Prod_code	Using_code	Quantity
AS1	C	1
AS1	T	1
DP	AS1	1
DP	BM	1

Practice 5-3 Bundles: Solution

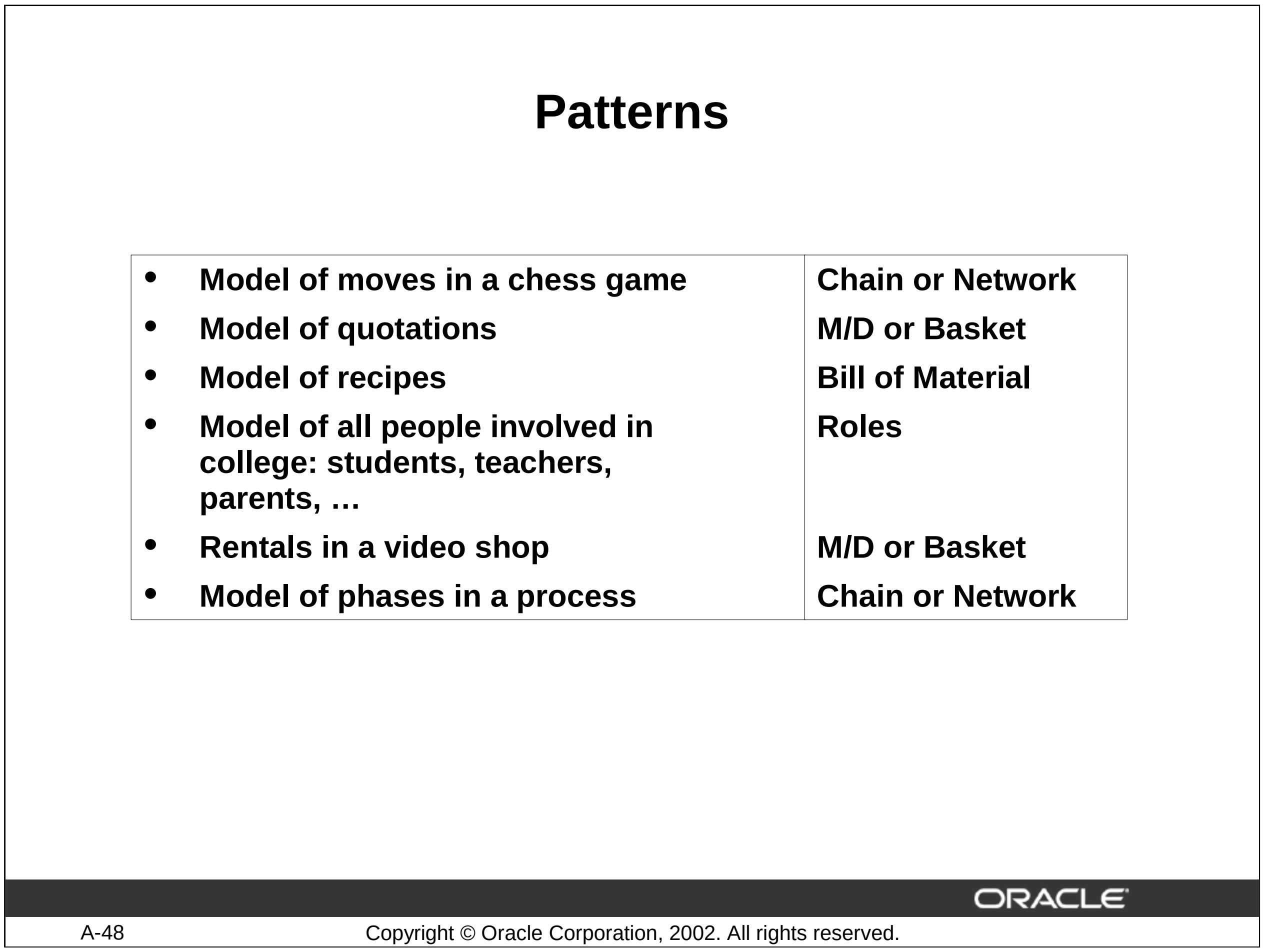
2. Change the model in such a way that it allows for:
   - This is a tricky one. You can regard a DecafPunch as a product group with two products: DecafPunch Coffee and DecafPunch Tea, both with the same price (because of the or). This point of view is already covered in the PRODUCT GROUP / PRODUCT model.
   - In the solution presented, this aspect is ignored. This solution focuses on how you can model the and or or type of constructions.
   - You see that DecafPunch can be regarded as a series of elementary groups that are connected with and or or. An elementary group is seen as a group that consists of either a single product or a product pair that is connected with an and or or. Products can always be described this way (this follows from mathematical logic).
   - See the table structure to check that the model works.
   - Note This model complexity may not make much sense in the context of Moonlight. However, this type of model certainly does make sense for the bill of material structures many production companies use for storing the composition information of their products, with alternative compositions and suchlike.



Practice 5-4 Product Structure: Solution

1. Create a model for a product classification structure.
   - When the number of levels is known and fixed, the left model can be used. Note that the model forces you to use all five levels for every product. To avoid this you could create a relationship from PRODUCT to all LEVEL entities, plus an arc across all.
   - The left model gives a lot of freedom. No limits to the minimum or maximum number of levels. Freedom to change the classification during the lifetime of the system.

2 (Optional) How would you treat the bundled products?
   - The problem with the bundles is that, strictly speaking, they cannot be classified in class Drinks nor Foods as they can consist of both. Probably the easiest way to handle this is to regard a bundle as an instance of PRODUCT CLASS.



Patterns

• Model of moves in a chess game	Chain or Network
• Model of quotations	M/D or Basket
• Model of recipes	Bill of Material
• Model of all people involved in college: students, teachers, parents, …	Roles
• Rentals in a video shop	M/D or Basket
• Model of phases in a process	Chain or Network

Practice 6-1 Patterns: Solution

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.

Moves in a Chess Game

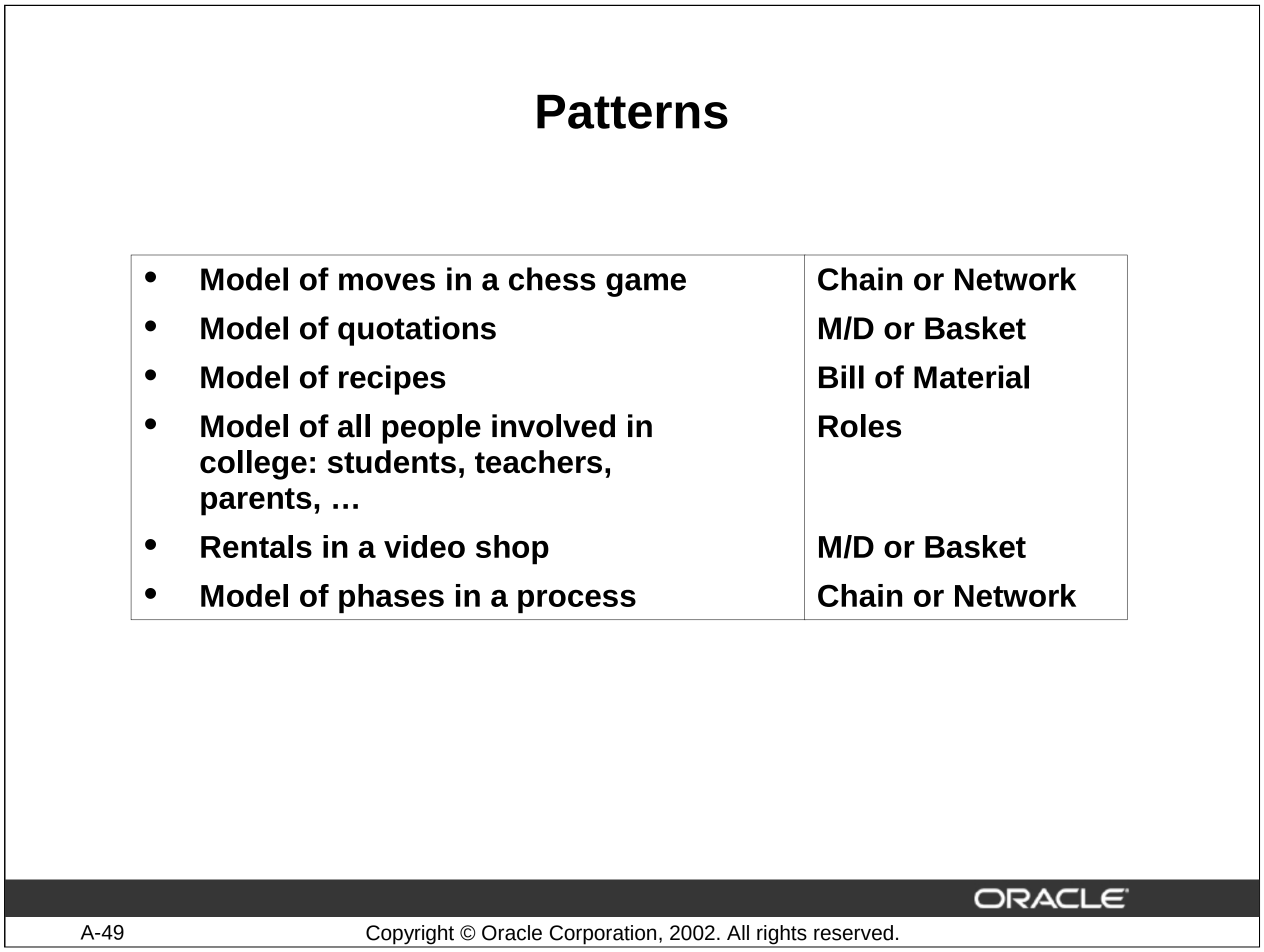
A MOVE moves a chess piece from one FIELD to another. This is a typical *Network* pattern. You can also see the MOVES as a *Chain* of ordered steps in the chess game.

Quotations

A quotation makes an offer to deliver a number of products of a certain quantity. This structure is similar to a ORDER and will have the *Master-Detail* or the *Basket* pattern.

Recipes

A recipe usually describes how to make something: in order to create a ... you need two ..., one ... and so on. This is typical *Bill of Material* pattern.



Patterns

• Model of moves in a chess game	Chain or Network
• Model of quotations	M/D or Basket
• Model of recipes	Bill of Material
• Model of all people involved in college: students, teachers, parents, …	Roles
• Rentals in a video shop	M/D or Basket
• Model of phases in a process	Chain or Network

Practice 6-1 Patterns: Solution

People Involved in a College

Pattern: *Roles.* A teacher can also be parent of a student, a parent can be an former student and so on.

Rentals in Video Shop

A RENTAL is an event such as a sale or a quotation: *Master-Detail* or the *Basket* pattern.

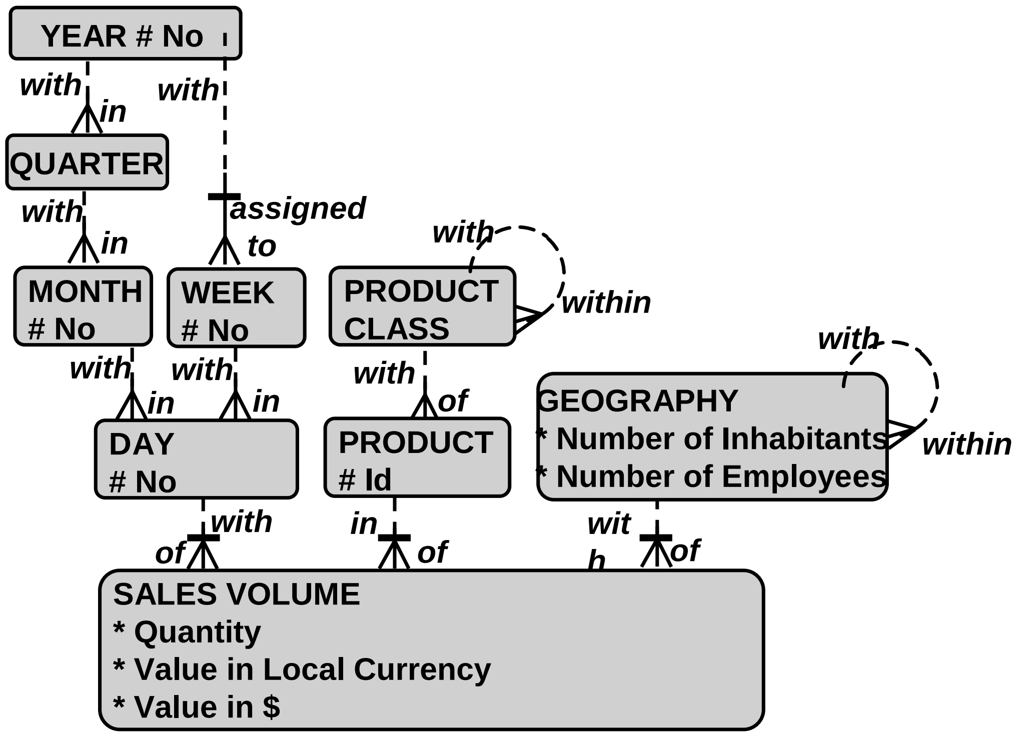
Phases in a Process

PROCESS and PHASE have a *Master-Detail* pattern. Be careful at this point. The crucial thing about phases in a process is that the phases are usually ordered.

The phases may form a *Chain* if there is always one phase at most that precedes and follows a particular phase.

If, on the other hand, the phases can be preceded and followed by any number of phases (maybe even including itself) than the structure would be a *Network*.

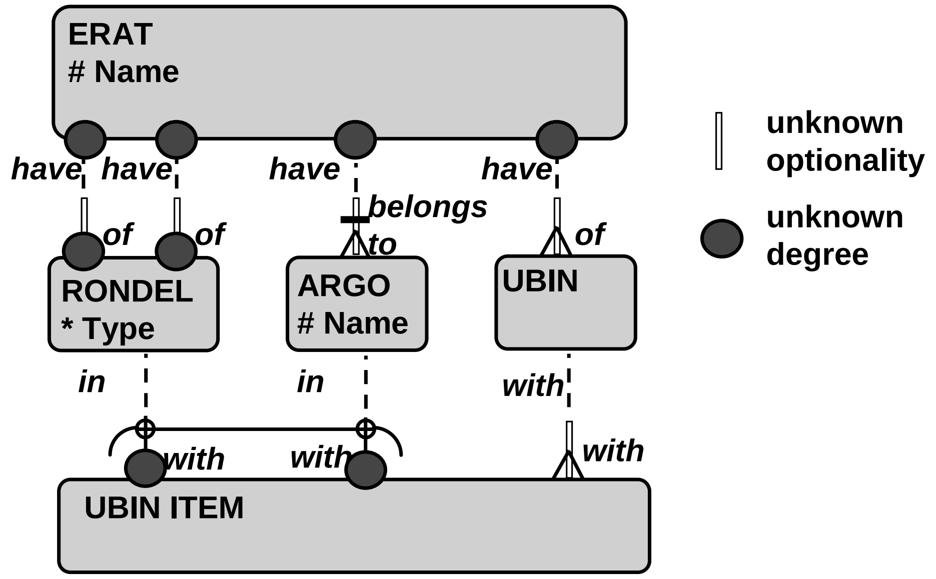
Moonlight Data Warehouse



Practice 6-2 Data Warehouse: Solution

1. Check the Moonlight models you created so far. Do they cater for answering the listed questions. If not, make the appropriate changes.
   - **No formal solution.**
2 For a data warehouse data model, suggest the central "facts" entity.
   - **The diagram answers more than the practice asked for. The central entity is SALES VOLUME.**
   - **This is a typical snowflake model, rather than a star model. All three dimensions, time, geography, and products, have their own internal structure.**
   - **Note that a week is not assigned to a month. Weeks are assigned to a year, even though not all weeks fit exactly into a calendar year. Weeks have a fixed length, like day and year, but behave differently as a week is not based on an astronomical fact.**
   - **The product structure we defined earlier can be used here. The distinction between local and global products has gone.**
   - **Geography is the generic word for an area (such as a part of country or a country or a continent) that is of interest.**

Solution: Argos and Erats



Constraint not shown:
A ubin always consists of one or more argos *of the erat*, one or more...

Practice 6-3 Argos and Erats: Solution

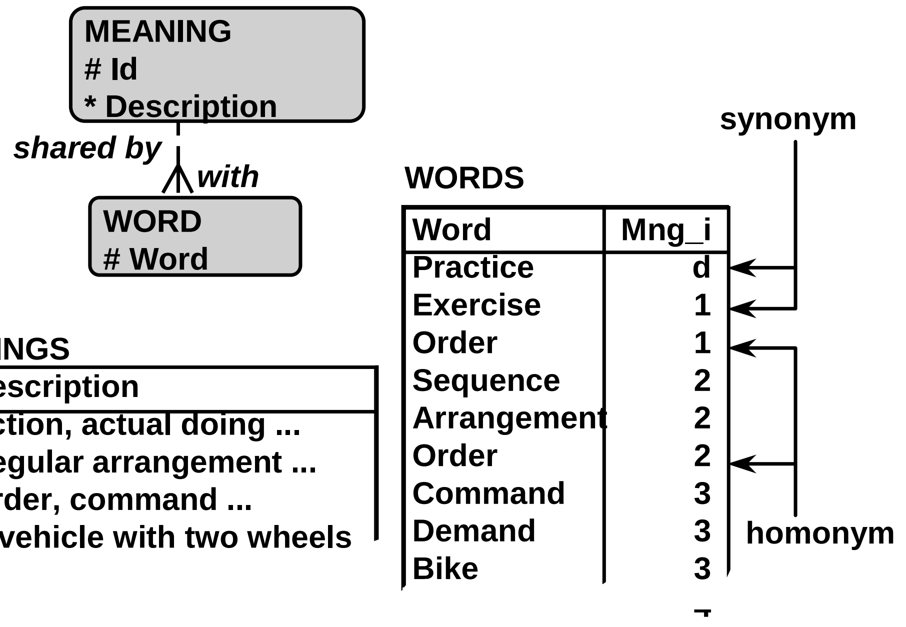
Make a conceptual data model based on the information in the text. Mark all the pieces in the diagram that can be confirmed from the text.

Not all information you need is present in the given text. You may be surprised how much actually is still left to check.

The only part you can be sure about is the UBIN - UBIN ITEM to model the given "A ubin always consist of one or more ...". This is a typical basket construction.

Note: this text was created by using some find/replace commands in a normal text. Ask your instructor about the original text.

Solution: Synonym



MEANINGS

Id	Description
1	Action, actual doing ...
2	Regular arrangement ...
3	Order, command ...
4	A vehicle with two wheels

WORDS

Word	Mng_i
Practice	d
Exercise	1
Order	1
Sequence	2
Arrangement	2
Order	2
Command	3
Demand	3
Bike	3

Practice 6-4 Synonym: Solution

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

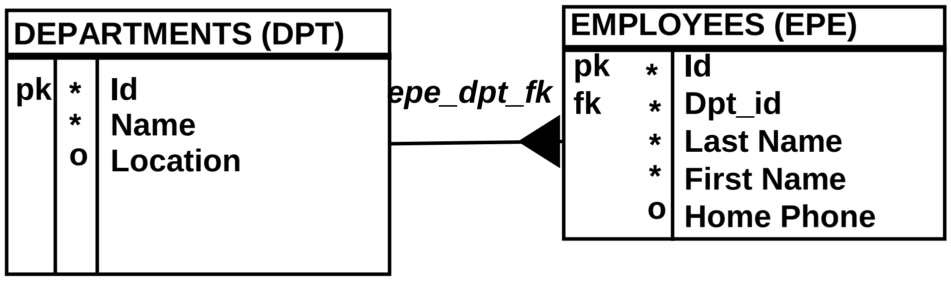
The solution is, probably, surprisingly and frustratingly easy.

Think of synonyms as words that share the same meaning. Do not confuse the concept *meaning* with the concept *word*. Although you express a meaning using a word or words, it is not the same as a word. Words are representations of meaning.

Note that there is no entity SYNONYM. A synonym is something circumstantial: if there is a MEANING shared by two or more words, then there is a synonym.

Conversely, if there is a WORD with two or more MEANINGS, there is a homonym.

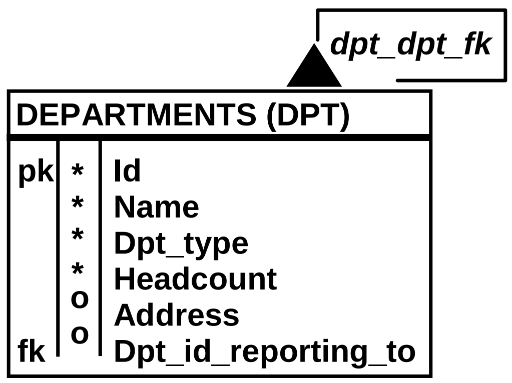
Solution: Mapping basic Entities, Attributes and Relationships



Practice 7-1 Mapping basic Entities, Attributes and Relationships: Solution

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.

Solution: Mapping Supertype



Practice 7-2 Mapping Supertype: Solution

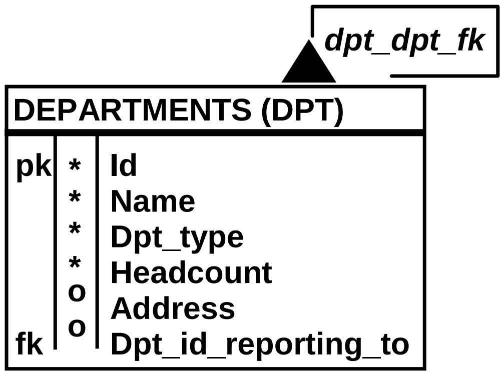
1 What would have been the rationale of this choice?

Possible considerations:

- **The number of departments is probably not large, say 100 or less.**
- **There are only few specific attributes for the subtypes.**
- **One subtype, HQ may have only one instance.**
- **HQ seems to be modeled only to show the fact that all departments except HQ must report to a department.**

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.

Solution: Mapping Supertype



Practice 7-2 Mapping Supertype: Solution

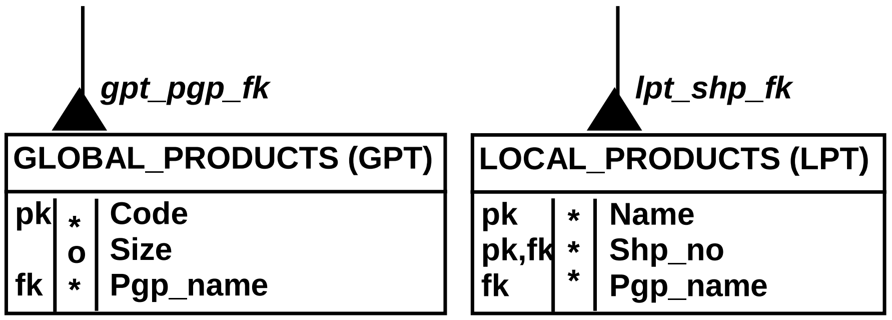
Note the optional Address column. Note also the optional foreign key column and its name. Following the rules, the name could have been Dpt_id. As many people would assume that to be a primary key column name, the name is followed by the name of the relationship that the foreign key represents.

Two constraints are needed:

To guard the subtype:

```
(       Dpt_type = ’HQ’
  and Address is not null
  and Dpt_id_reporting_to is null)
or
(      Dpt_type <> ’HQ’
  and Address is null
  and Dpt_id_reporting_to is not null)
```

Solution: Quality Check Subtype Implementation



Practice 7-3 Quality Check Subtype Implementation: Solution

Perform a quality check on the proposed subtype implementation of entity PRODUCT.

Errors:

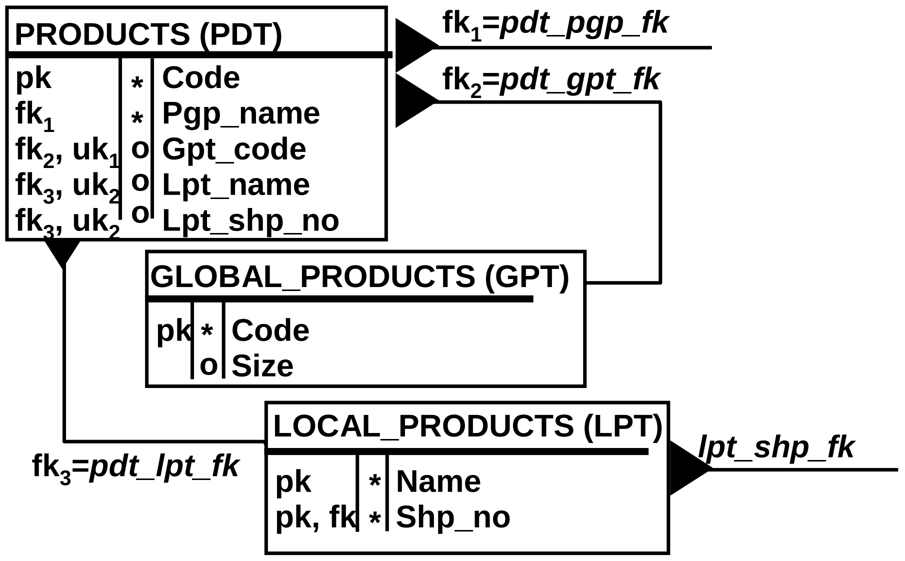
GLOBAL_PRODUCTS

- **The foreign key column Pgp_name is not marked with fk.**

LOCAL_PRODUCTS

- **Column Shop_no should be named Shp_no**
- **Column Shp_no is also part of the primary key.**
- **The foreign key to table SHOPS should be named *lpt_shp_fk***

Solution: Quality Check both Supertype and Subtype Implementation



Practice 7-4 Quality Check Arc Implementation: Solution

Perform a quality check on the proposed supertype and subtype implementation of the entity PRODUCT and its subtypes. Also, check the selected names.

Errors:

PRODUCTS

- **Gpt_code and Lpt_name are both part of a unique key uk_1**
- **The last characters of the names of the foreign keys should all read _*fk***
- ***A fk column is missing as the primary key of LOCAL_PRODUCTS consists of two columns. Column Lpt_shp_no must be added. This column is part of fk_3* and part of uk_2.**

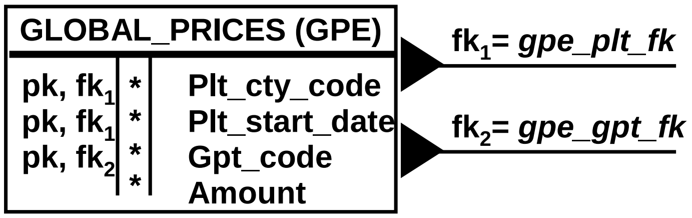
GLOBAL_PRODUCTS

- **This is an awkward one:**
- **The foreign key line in the diagram must be *removed* as GLOBAL_PRODUCTS does not have its own foreign key to PRODUCT_GROUPS.**

LOCAL_PRODUCTS

- **Column Pgp_name must be removed. Shp_n**

Solution: Primary Keys and Columns



Practice 7-5 Primary Keys and Columns: Solution

Identify the Primary key columns and names resulting from the transformation of the GLOBAL PRICE entity. Give the short name.

Note that GLOBAL_PRICES has three columns that are part of its primary key. All these columns are also part of foreign keys. The first two come from PRICE_LISTS and have prefix Pct_. The third comes from GLOBAL_PRODUCTS and has prefix Gpt_.

Note the double prefix in column name Pct_cty_code as this column refers to Cty_code of PRICELISTS, which is a foreign key column itself.

Practice 8-1 Name that Denormalization: Solution

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:

- Pre-Joining Tables
- Hard-Coded Values
- Keeping Details with Master
- Repeating Single Detail with Master
- Short-Circuit Keys

Type - Prejoin tables

Why: The Name column from the WEEKDAYS table has been added to the SHIFTS table, as the Wdy_name column.

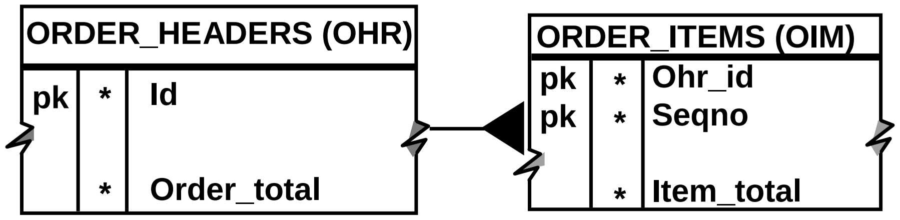
Type - Short-Circuit Key

Why: A new foreign key constraint and column has been added to the PROD_NAMES table, to join to the PROD_GRPS table.

Type - Invalid (could have been either End Date Column or Current Indicator)

Why: When you add a column to a table to facilitate issues concerning date retrieval or insert, add either an end-date column or current indicator column, not both.

Practice: Triggers *(1/6)*



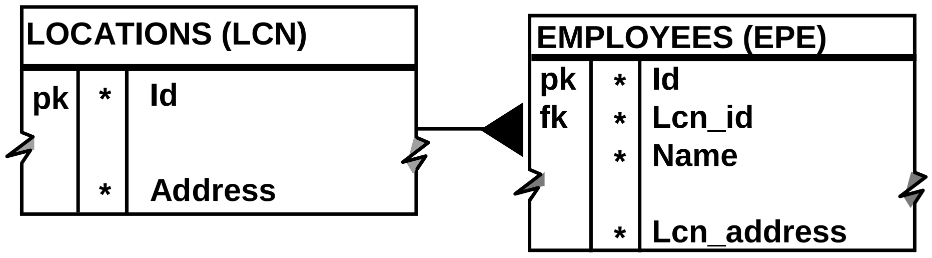
Practice 8-2 Triggers: Solution

1. Indicate which triggers are needed and what they should do to handle the denormalized column Order_total of ORDER_HEADERS.
   - How to handle Order_total for ORDER_HEADERS

Solution: Triggers *(2/6)*

Table	Trg Type	Column	Needed?	What should it do?
OHR	Insert		Y	Order_total := 0
	Delete		N	
	Update	Id	N	
		Order_total	Y	prevent update
OIM	Insert		Y	recalculate Order_total
	Delete		Y	recalculate Order_total
	Update	Ohr_id	Y	recalculate Order_total
		Item_total	Y	recalculate Order_total

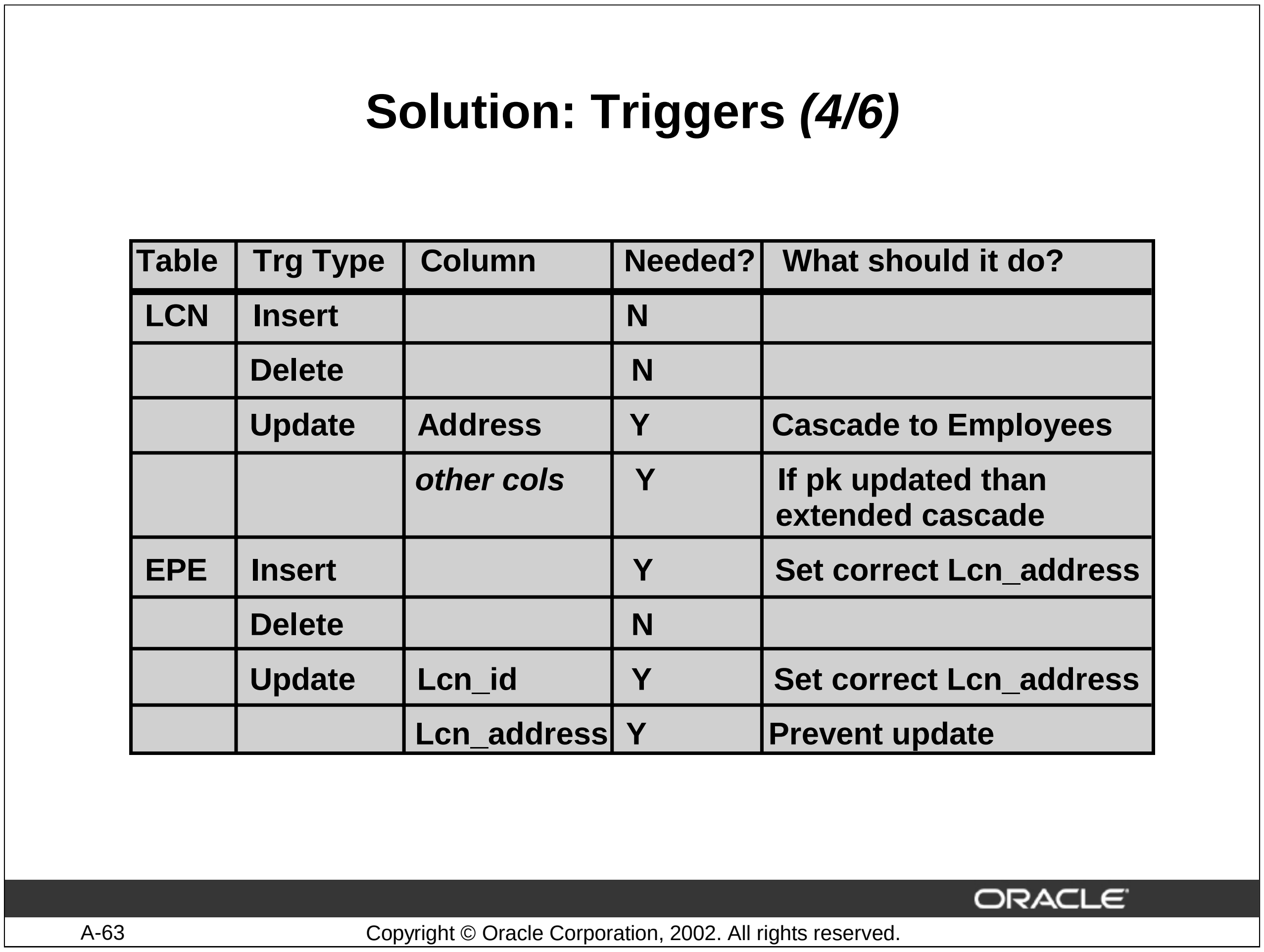
Practice: Triggers *(3/6)*



Practice 8-2 Triggers: Solution

2 Indicate which triggers are needed and what they should do to handle the denormalized column Lcn_address of EMPLOYEES.

- How to handle Lcn_address for EMPLOYEES



Solution: Triggers *(4/6)*

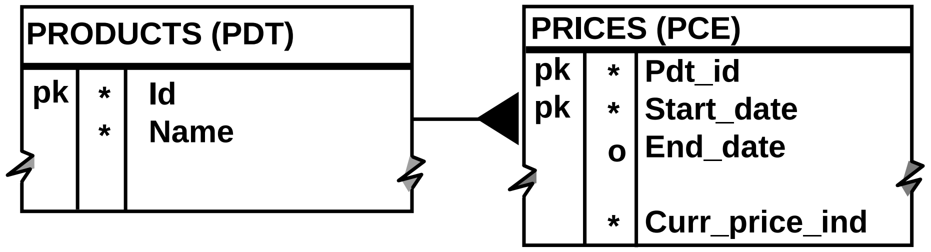
Table	Trg Type	Column	Needed?	What should it do?
LCN	Insert		N	
	Delete		N	
	Update	Address	Y	Cascade to Employees
		other cols	Y	If pk updated than extended cascade
EPE	Insert		Y	Set correct Lcn_address
	Delete		N	
	Update	Lcn_id	Y	Set correct Lcn_address
		Lcn_address	Y	Prevent update

A-63

Practice 8-2 Triggers: Solution

3. Indicate which triggers are needed and what they should do to handle the denormalized column Curr_price_ind of table PRICES.
   - How to handle Curr_price_ind for PRICES

Practice: Triggers *(5/6)*

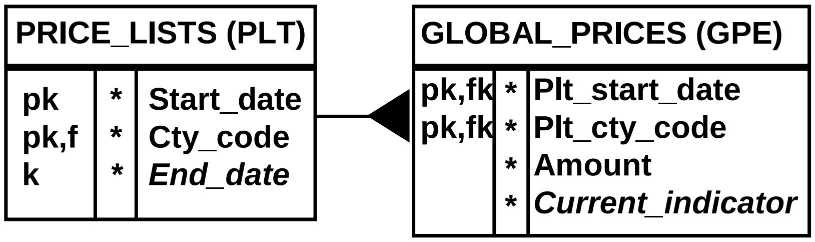


Solution: Triggers *(6/6)*

Table	Trg Type	Column	Needed?	What should it do?
PDT	Insert		N	
	Delete		N	
PCE	Insert		Y	Prevent overlap in price periods
	Delete		N	
	Update	Pdt_id	Y	Set Curr_price_ind to NULL
		Start_date	Y	Re-evaluate Curr_price_ind
		End_date	Y	Re-evaluate Curr_price_ind
		Curr_price_Ind	Y	Prevent update by user

Note that this solution is incomplete. There is no guarantee that the Curr_price_ind is indeed set for a price with Start_date ≤ Sysdate ≤ End_date. To achieve that you need to set a timer that checks, say once a day, if the End_date of a price that is marked as current has expired.

Solution: Denormalize Price Lists



Practice 8-3 Denormalize Price Lists: Solution

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.

You could handle the design using two denormalization techniques. You could use the *Current Column Indicator* technique for the performance issue, and the *End Date Column* to allow managers to pre-enter price lists.

Slow Performance

You can increase performance on queries, by adding a denormalized column to the GLOBAL_PRICES table. Since this table will store a large volume of records of which only very few will be current, you could consider adding a Current_indicator column to the table.

Some additional code is required to populate the column correctly.

Practice 8-3 Denormalize Price Lists: Solution

Pre-entering Price Lists

The code we need to add should read the Start_date column in the PRICE_LISTS table and set the current record indicator to reflect the current amount. This code will populate the current record indicator, but when should it fire? Let us assume the price of a product can only change once a day. With this scenario, we can simply set a timer for twenty-four hours, and when it expires, fire the code to update current record indicators, if there have been any changes in the price lists.

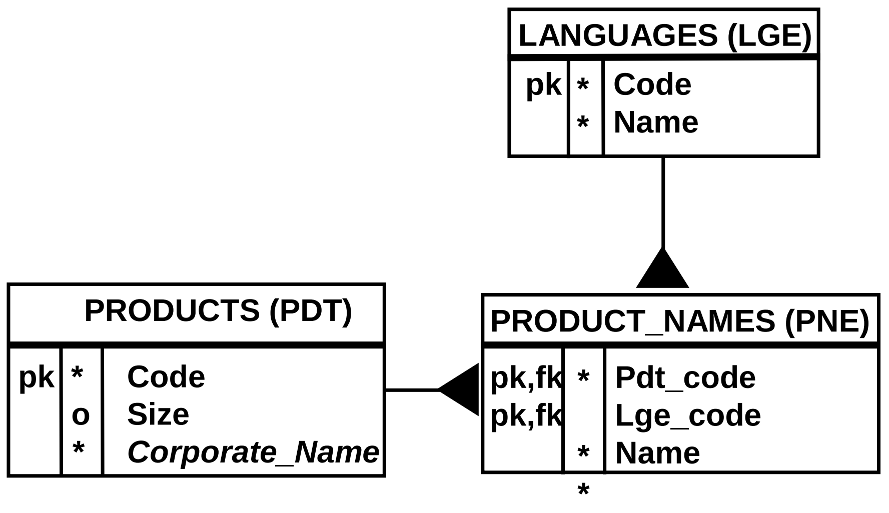
The issue of pre-entering price list information, before its effective date will require another type of denormalization.

Implementing the corporate policy of not waiting to make changes to the Price list requires a similar, and different, type of denormalization. We need to add an *End Date Column* to the PRICE_LISTS table. If we add this new column, then we can use it to show a price's expiration date, before that date is reached. So, rather than calculating the expiration date for a price to be equal to the start date of the next sequential price, we can enter the start date, price, and end date for the price when we insert the original record. In some cases the end date may not be known at the time the record is inserted, so we will make that column optional.

The code we need to add should fire any time a price list is inserted or updated. The code will read the start date of the current price list, then update the end date of the prior price list record. There is a small cost during DML, but it is minimal.

The diagram below shows the table diagram after both denormalization techniques have been implemented.

Solution: Denormalize Global Naming



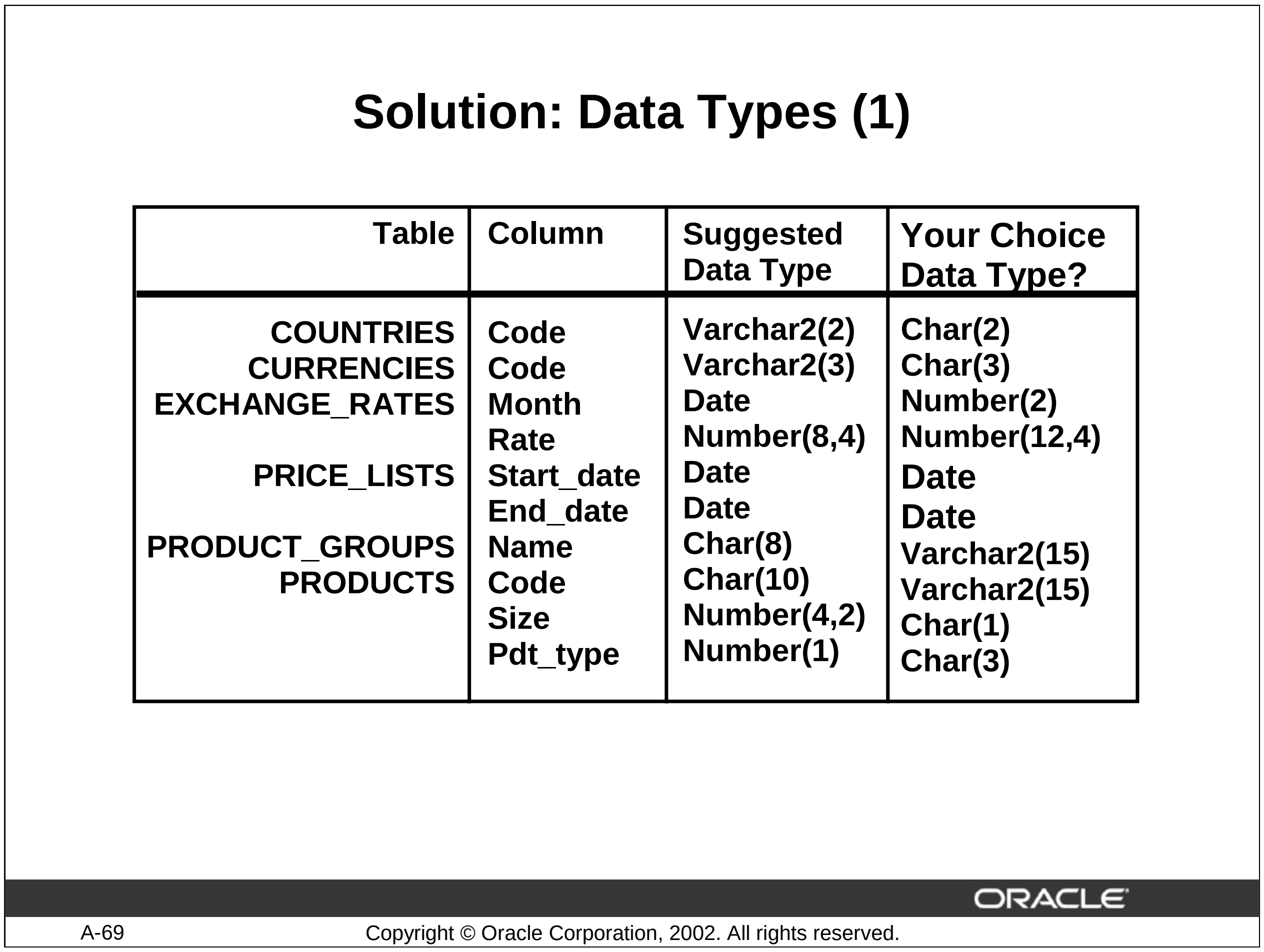
Practice 8-4 Global Naming: Solution

Using the design below, denormalize the table design and describe the additional code that will allow this requirement to be implemented.

One solution to this requirement could be to store the Name for a PRODUCT with English as language in a new denormalized column in the PRODUCTS table. This Corporate_name column would then contain the corporate names for every global product.

You would also need to add some application code to update the Corporate_name column in the PRODUCTS table, if there are any inserts in the PRODUCT_NAMES table and if there are changes made to the Name column from the PRODUCT_NAMES table where the language is English.

The diagram below show the way the tables' design could be structured.



Solution: Data Types (1)

Table	Column	Suggested Data Type	Your Choice Data Type?
COUNTRIES	Code	Varchar2(2)	Char(2)
CURRENCIES	Code	Varchar2(3)	Char(3)
EXCHANGE_RATES	Month	Date	Number(2)
	Rate	Number(8,4)	Number(12,4)
PRICE_LISTS	Start_date	Date	Date
	End_date	Date	Date
PRODUCT_GROUPS	Name	Char(8)	Varchar2(15)
PRODUCTS	Code	Char(10)	Varchar2(15)
	Size	Number(4,2)	Char(1)
	Pdt_type	Number(1)	Char(3)

A-69

Practice 9-1 Data Types: Solution

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.

Solution: Data Types (2)

Table	Column	Your Choice Data Type
GLOBAL_PRICES	Amount	Number(15,3)
LOCAL_PRICES	Start_date	Date
	End_date	Date
	Amount	Number(15,3)
SHOPS	Name	Varchar2(50)
	Address	Varchar2(50)
	City	Varchar2(50)

Practice 9-1 Data Types: Solution

Fixed length values, like the codes for countries and currencies, can best be stored using a Char data type. “Month” is not a date and can best be stored using a number as this allows sorting. In general: do not be hesitant to allow long character names and other values. You will regret it if you are a little too economical!

1. Suggest data types for the following columns. They are all based on previous practices.
2. What data type would you use for a column that contains times only?
   - A time (like attribute Start_time and End_time in entity SHIFT) can be implemented in several ways. One is to use a CHAR(5) to record times like 08:55. If calculations are needed, it is often implemented as two Number(2,0) columns, one for the hours and the second one for the minutes. A third possibility is to store it as a DATE column with a fixed date (like 01/01/01 but perhaps on second thoughts, ...!).
   - Most countries around the world use the two-decimal format for money amounts in the local currency. The other countries use no decimals. There is, as far as is known, only one country that uses three decimal places.

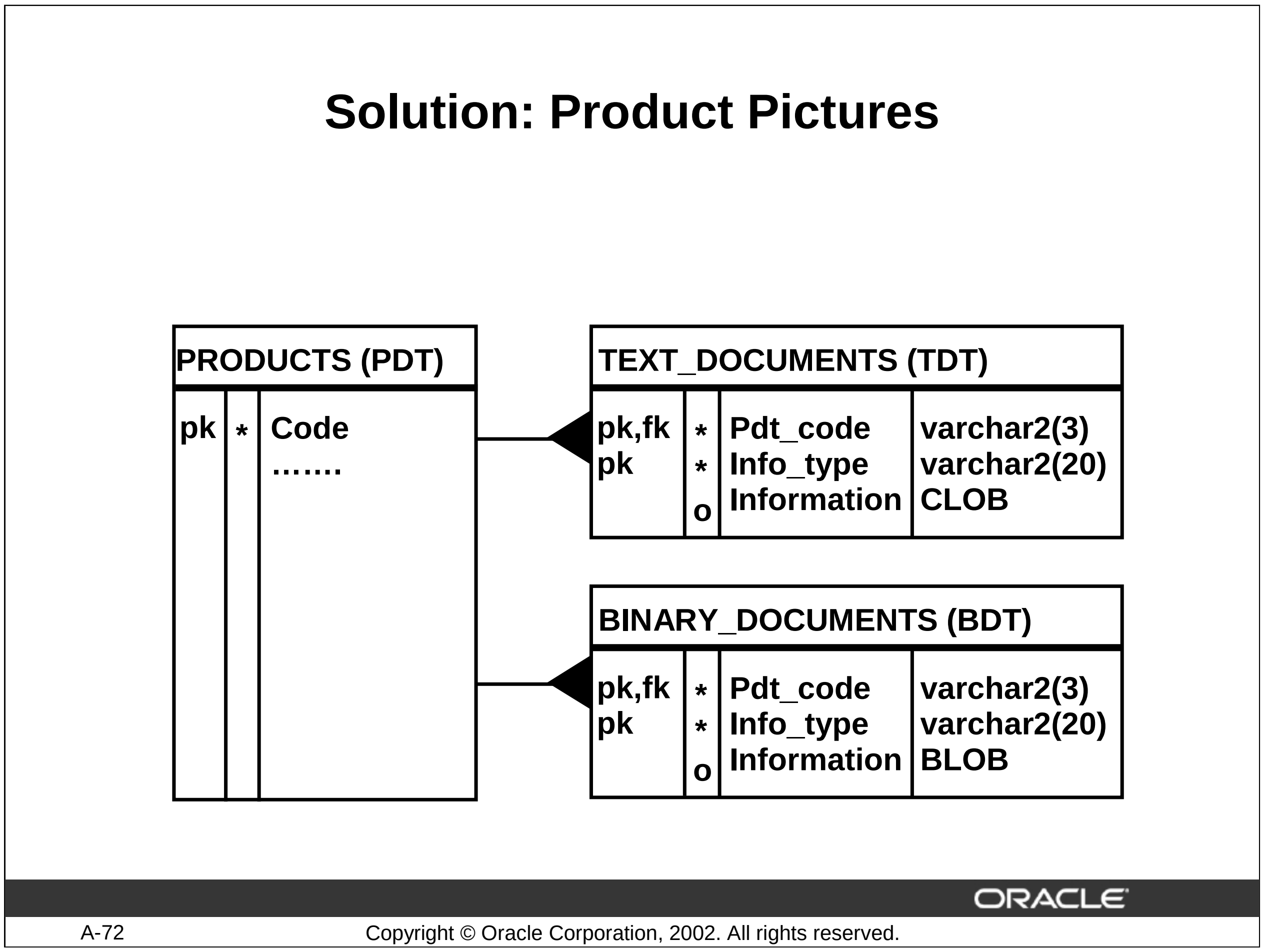
Practice 9-2 Artificial Keys: Solution

Indicate for each table if you see benefits of creating an artificial key and why.

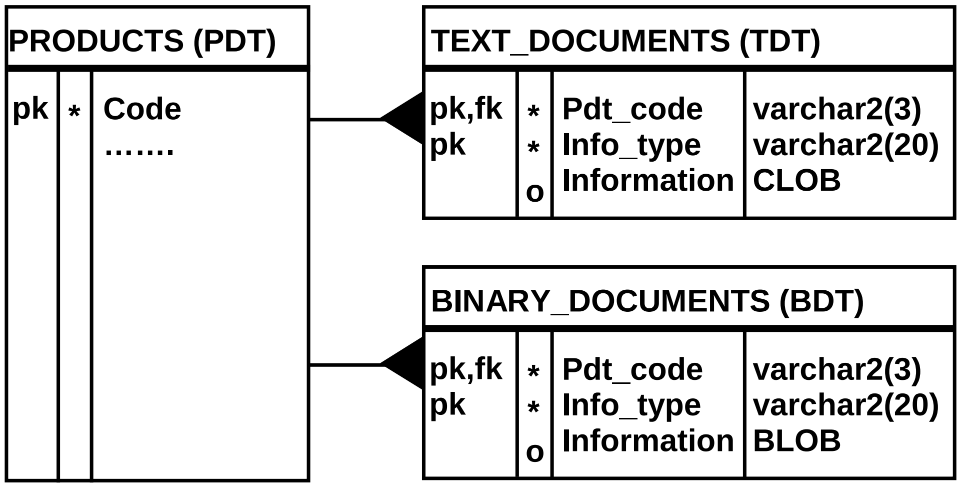
- COUNTRIES
- GLOBAL_PRICES
- PRICE_LISTS
- COUNTRIES have an three-character internationally-used code, which can be used as a primary key. This code is like an abbreviation and is quite easy to memorize. I am not sure what happens when a country renames itself. This may result in a new code. Not the most convincing candidate for an artificial key.
- GLOBAL_PRICES has no need for an artificial key as there are currently no tables referring to it. If, however, there is a chance there will be references in the future, an artificial key would make sense.
- PRICE_LISTS seems a good candidate for an artificial key, as the UID consists of two components, one of which is a date.

• For which tables (if any) based on the Moonlight model does it not make any sense at all to create artificial keys?

• An artificial key on tables EXCHANGE_RATES, GLOBAL_PRICES (see above), LOCAL_PRICES, PRODUCT_NAMES would only be an interesting alternative if the primary key for these tables was referenced in other (future) tables.



Solution: Product Pictures


PRODUCTS (PDT)
pk * Code
…….
TEXT_DOCUMENTS (TDT)
pk,fk * Pdt_code varchar2(3)
pk * Info_type varchar2(20)
o Information CLOB
BINARY_DOCUMENTS (BDT)
pk,fk * Pdt_code varchar2(3)
pk * Info_type varchar2(20)
o Information BLOB

Practice 9-3 Product Pictures: Solution

1 Decide what data type you would advise to be used for each column.

Which data type would you use for each column?

- **Picture LONG RAW**
- **Html_doc LONG**

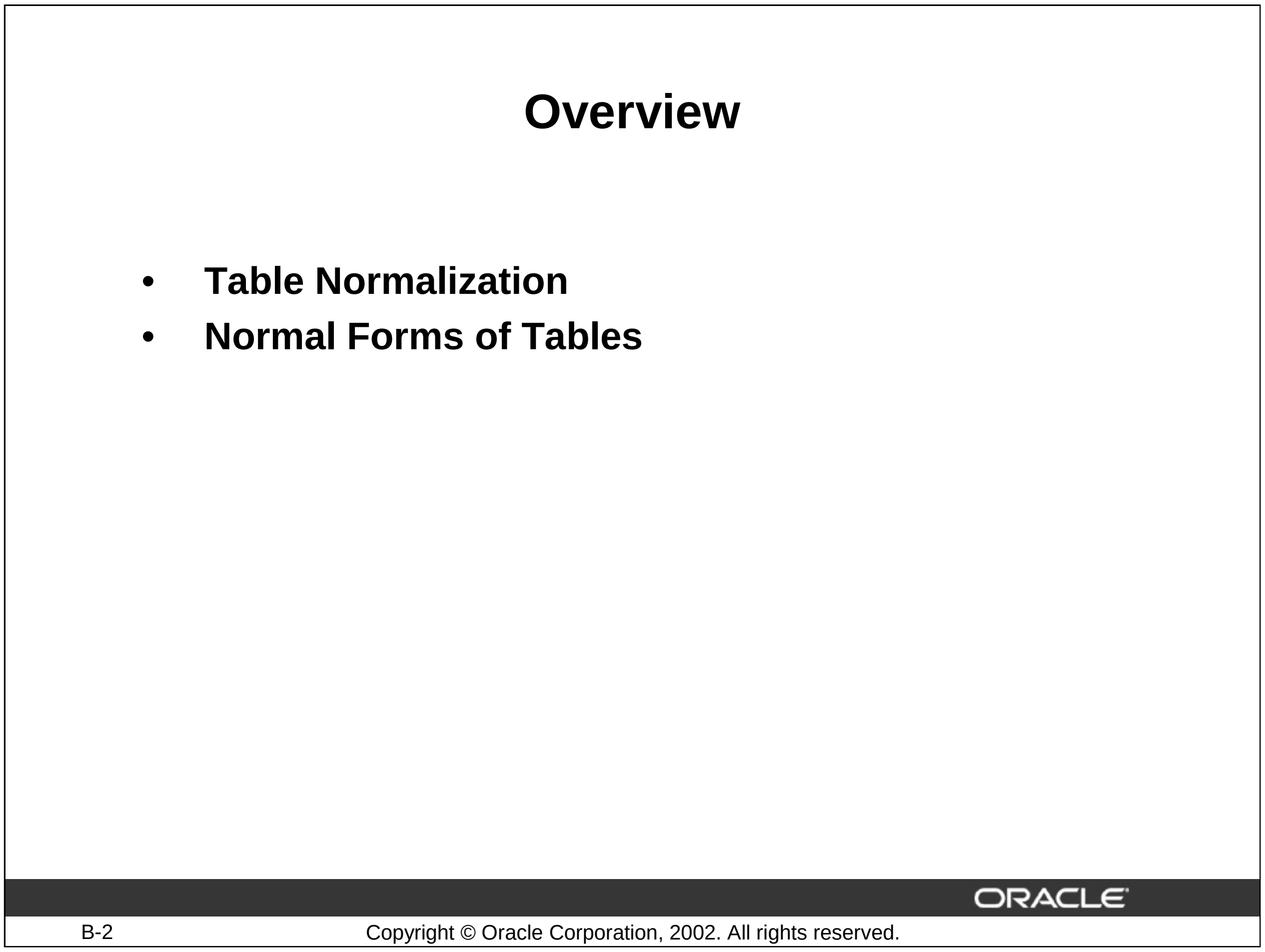
2 You have heard that an old Oracle version would not accept more than one long type column per table. You are not sure if this is still a limitation. Advise about the implementation.

Advise about the implementation. See the table structure diagram. There can be one table for multiple LONG columns for a product. There must be a second table for the LONG RAW columns. This model is, in fact, a vertical partitioning of a product record. It would also be a good structure when tables allow more than one long type column. Separating the huge columns from the small ones increases the speed of a read considerably.

B

Normalization

ORACLE



Overview

- **Table Normalization**
- **Normal Forms of Tables**

B-2 Copyright © Oracle Corporation, 2002. All rights reserved.

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.

TABLE

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.

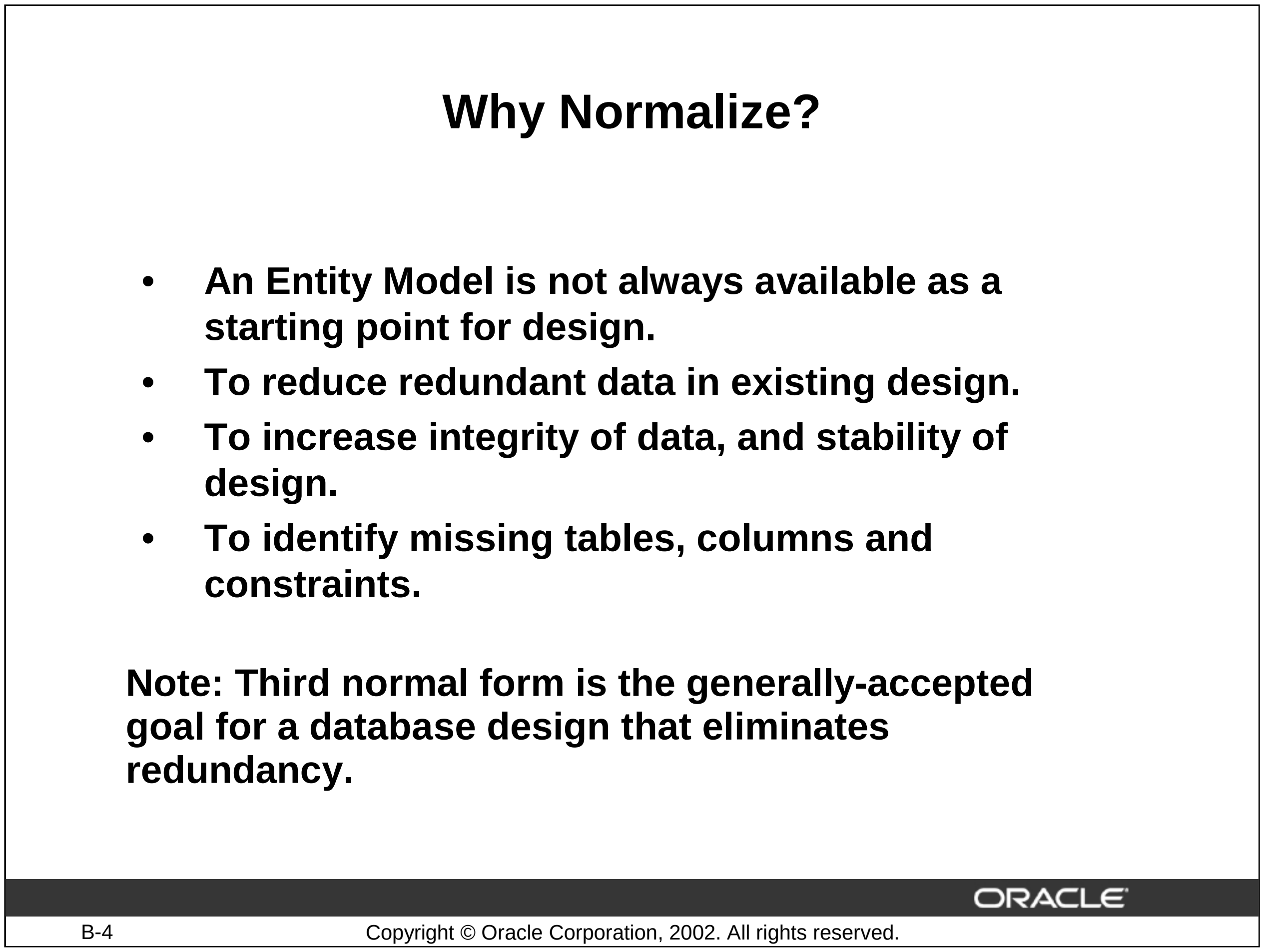
Two terms that have their origins in the normalization technique are still widely in use. One is *normalized data*, the other is *denormalization*.

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-4

Normalization and its Benefits

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.

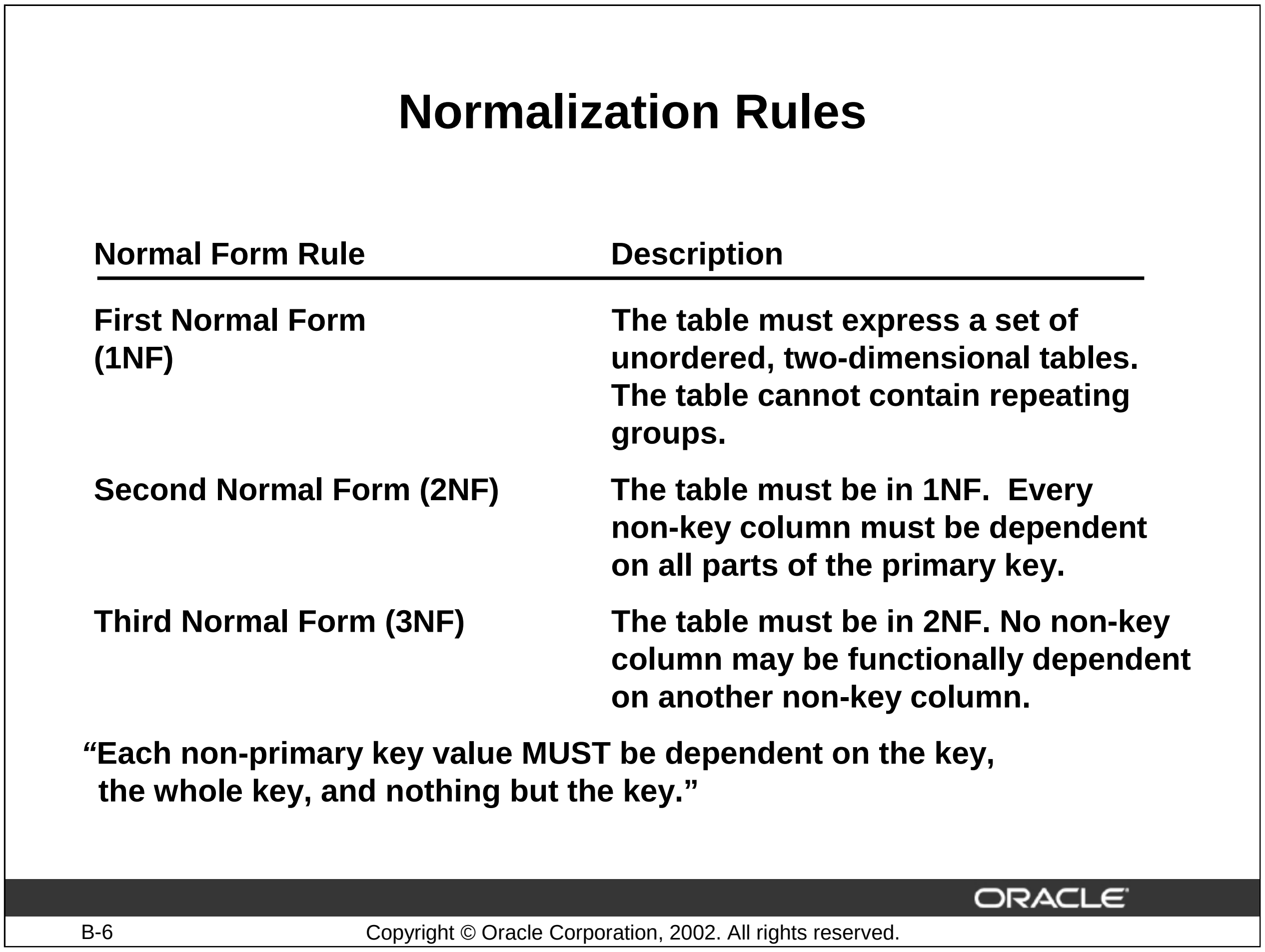
Recognize Unnormalized Data

```
USER USER    MSE REC_                                SRVR SERVER
 _ID _NAME   _ID DATE  SUBJECT         TEXT       _ID  _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-5

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 Rules

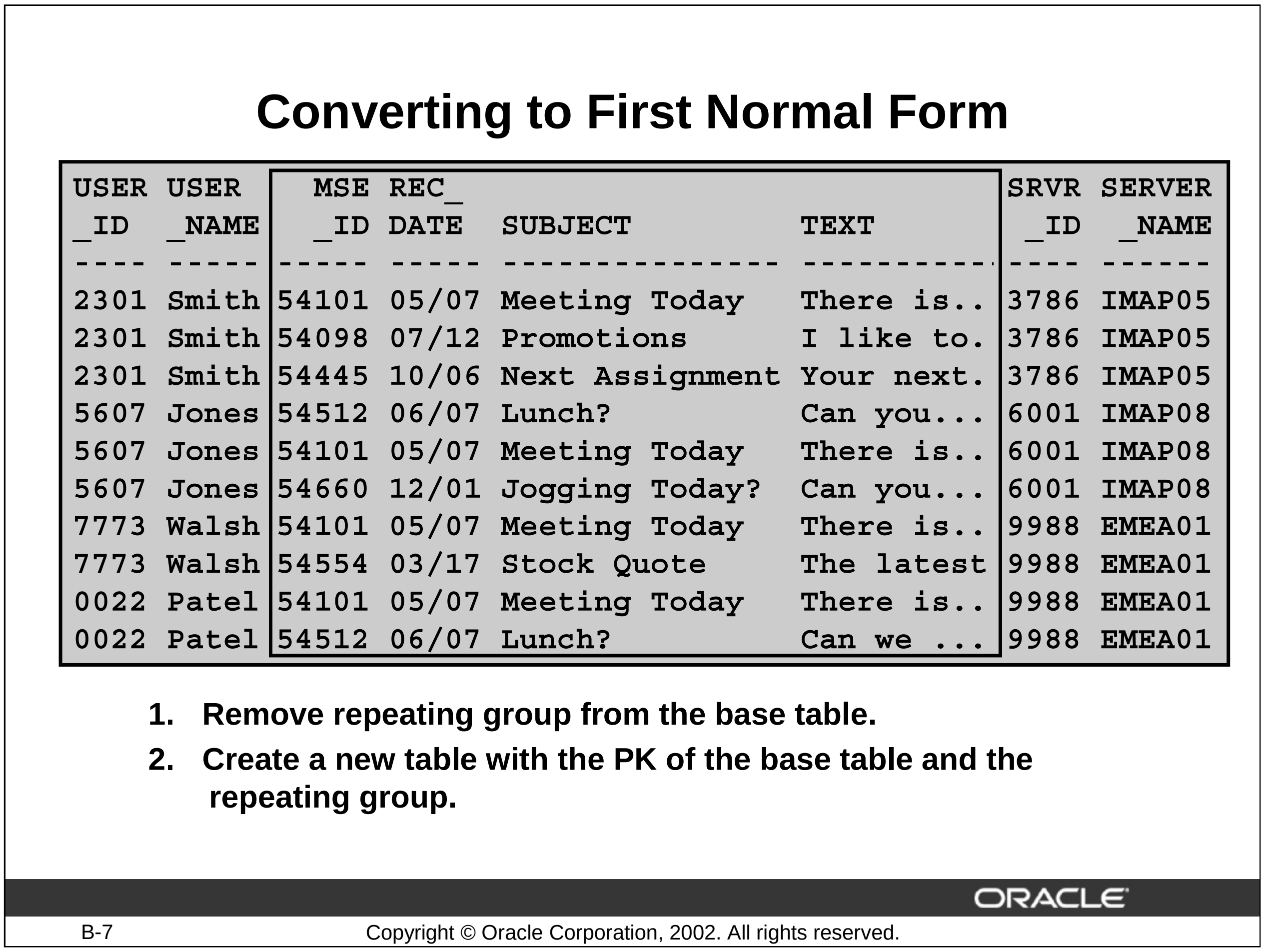
Normal Form Rule	Description
First Normal Form (1NF)	The table must express a set of unordered, two-dimensional tables. The table cannot contain repeating groups.
Second Normal Form (2NF)	The table must be in 1NF. Every non-key column must be dependent on all parts of the primary key.
Third Normal Form (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-6

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.



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.**

ORACLE

B-7

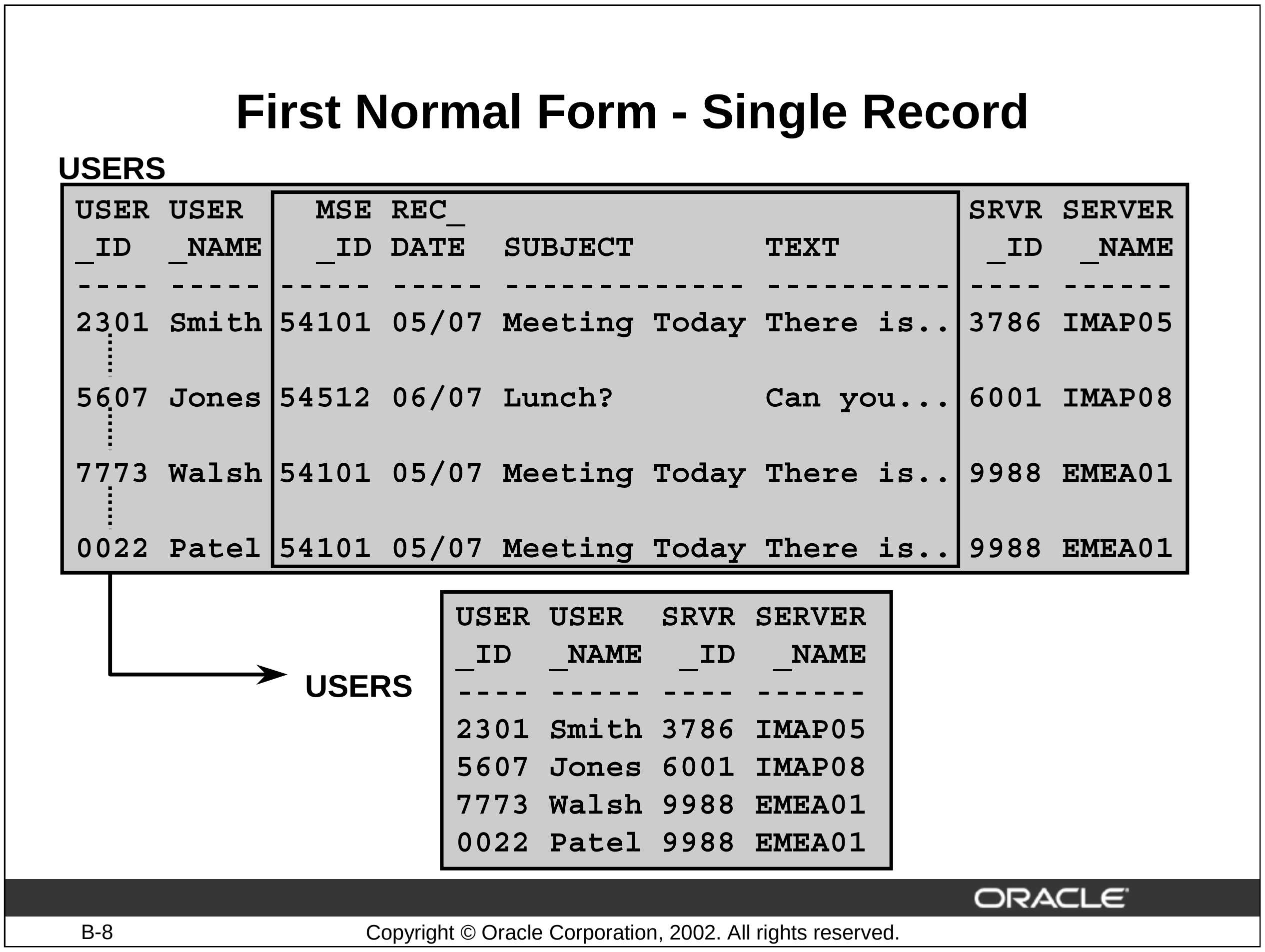
First Normal Form

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

- Remove the repeating columns from the original unnormalized table.
- Create a new table with the primary key of the base table and the repeating columns.
- Add another appropriate column to the primary key, which ensures uniqueness.
- Create a foreign key in the new table to link back to the original unnormalized table.



First Normal Form - Single Record

USERS

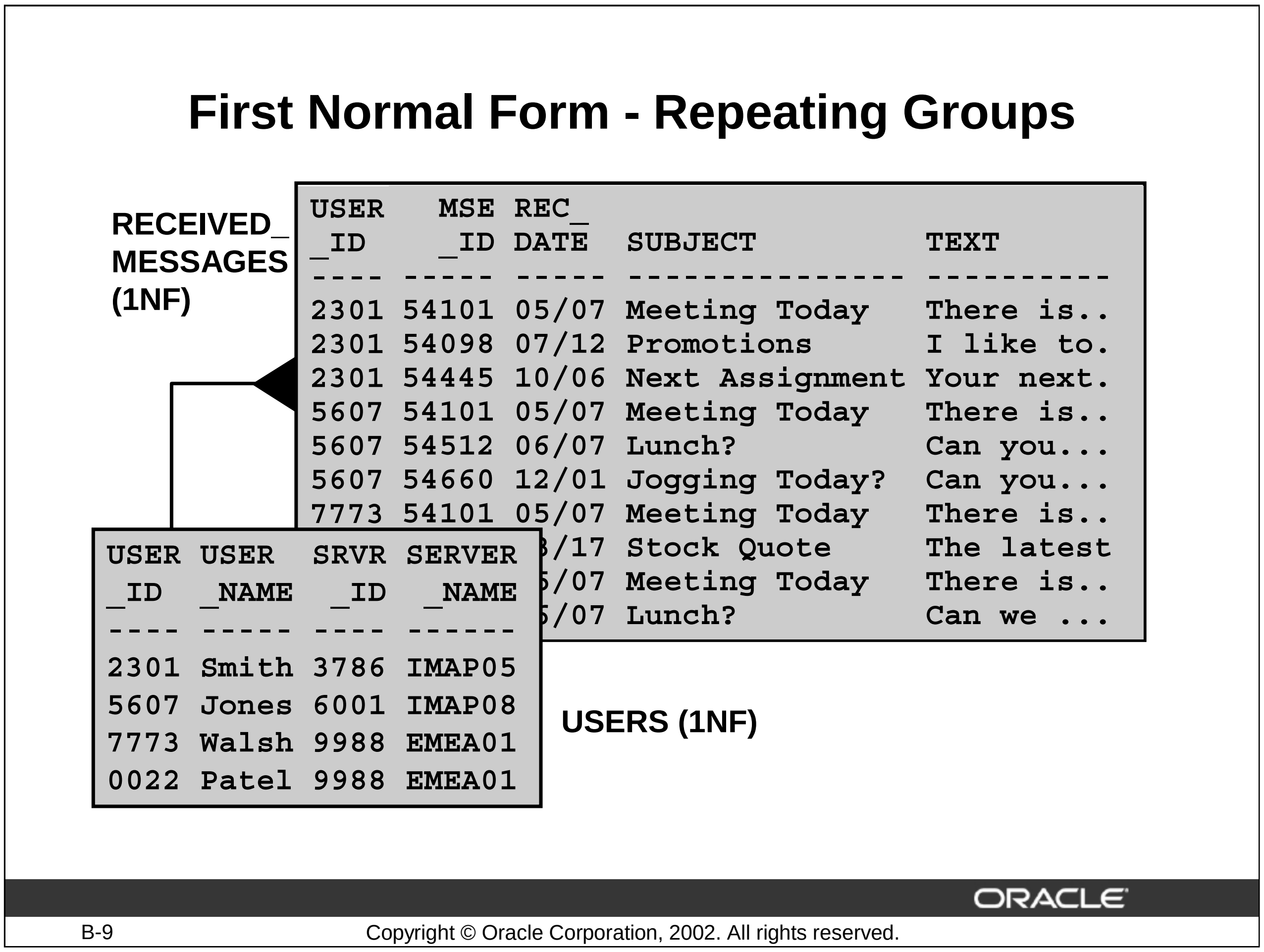
USER_ID	USER_NAME	MSE_ID	REC_DATE	SUBJECT	TEXT	SRVR_ID	SERVER_NAME
2301	Smith	54101	05/07	Meeting Today	There is..	3786	IMAP05
5607	Jones	54512	06/07	Lunch?	Can you...	6001	IMAP08
7773	Walsh	54101	05/07	Meeting Today	There is..	9988	EMEA01
0022	Patel	54101	05/07	Meeting Today	There is..	9988	EMEA01

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

First Normal Form

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

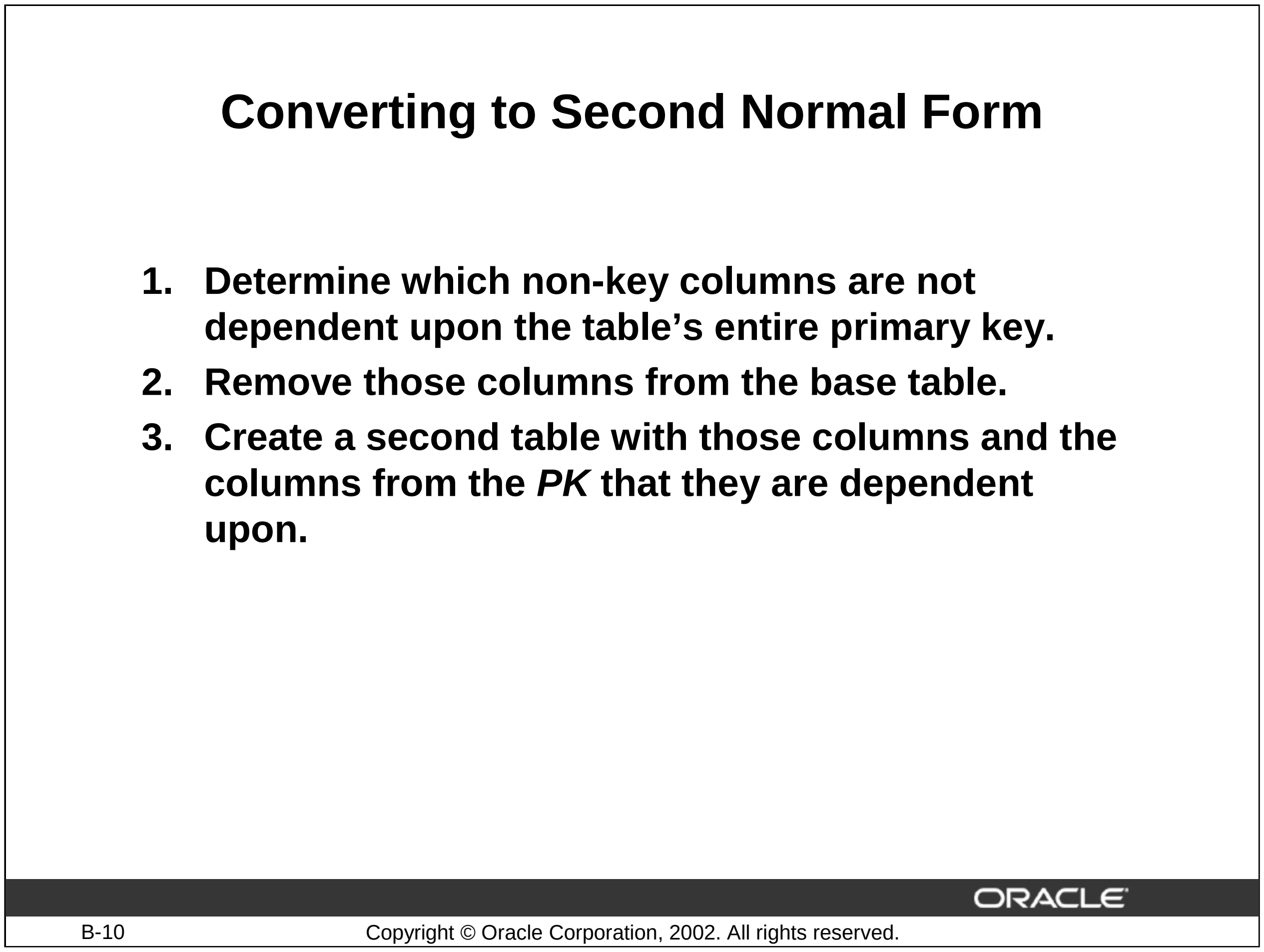
RECEIVED_ MESSAGES (1NF)

USER _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 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..
		3/17	Stock Quote	The latest
		5/07	Meeting Today	There is..
		5/07	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)

ORACLE



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.**

ORACLE

B-10

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.

Steps to Remove Partial Dependencies

- Determine which nonkey columns are dependent upon the table’s entire primary key.
- 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.
- 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 USER  SRVR SERVER
_ID  _NAME  _ID  _NAME
---- ----- ---- ------
2301 Smith 3786 IMAP05
5607 Jones 6001 IMAP08
7773 Walsh 9988 EMEA01
0022 Patel 9988 EMEA01
```

Is the USERS table already in 2NF?

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 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..
73	54101	05/07	Meeting Today	There is.
73	54554	03/17	Stock Quote	The lates
22	54101	05/07	Meeting Today	There is.
22	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

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.**

ORACLE

B-13

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”.

Steps to Remove Transitive Dependencies

- Determine which columns are dependent on another non-keyed column.
- Remove those columns from the base table.
- Create a second table with those columns and the non-key columns that they are dependent upon.
- 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 _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

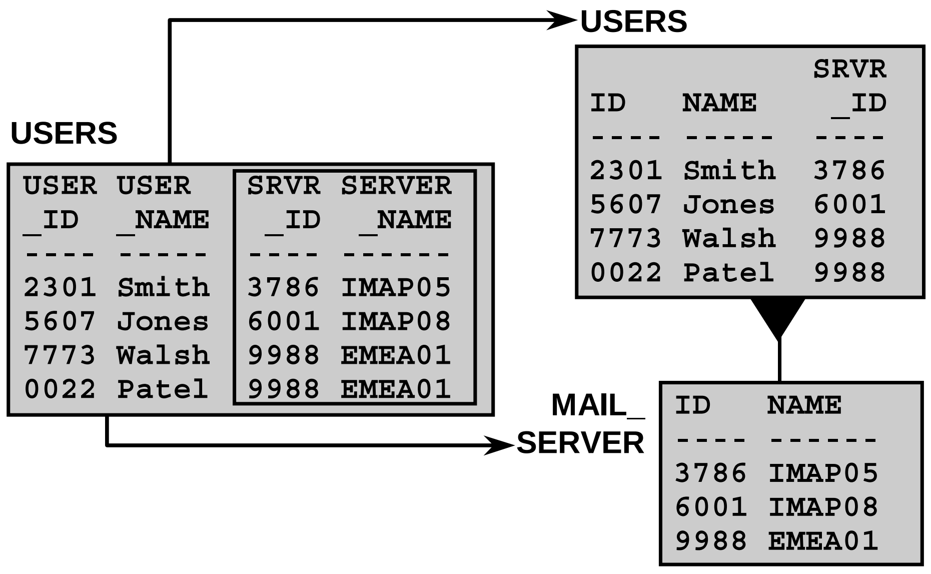
MESSAGES (2NF)

ID	SUBJECT	TEXT
54101	Meeting Today	There is.
54098	Promotions	I like to
54445	Next Assignment	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?

ORACLE

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

Third Normal Form

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

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.