5

Modeling Change

ORACLE

Overview

- Date and time
- Modeling change over time
- Prices change
- Journaling

Introduction

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

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

Objectives

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

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

Change and Time

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

ORACLE

Modeling Time

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

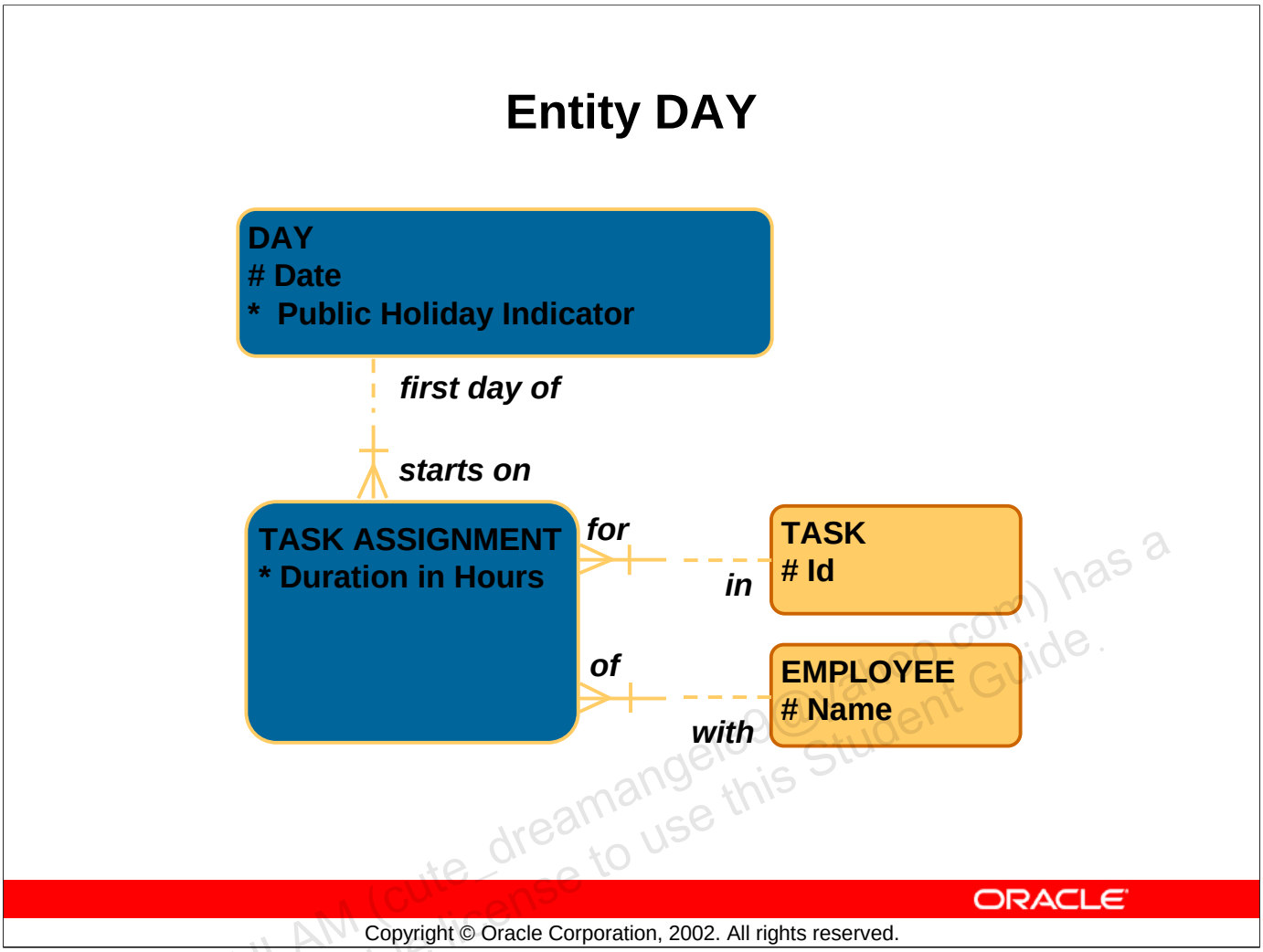
A second time-related issue often helps to increase the usability of a system dramatically. By adding dates like Start, Expiry, End Date, to data in the system, you allow users to work in advance. Suppose a particular value, say the price of gas or diesel, will change as of January 1. It is very useful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

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

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

The following issues arise:

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



Entity DAY

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

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

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

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

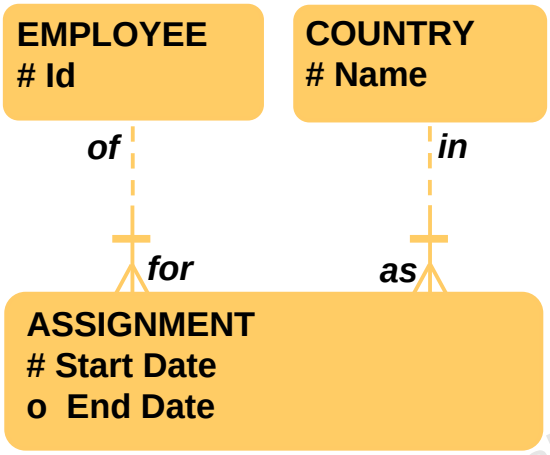
Modeling Days Over Time

Date and Time

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

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

Modeling Change



Modeling Changes Over Time

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

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

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

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

Modeling Changes Over Time

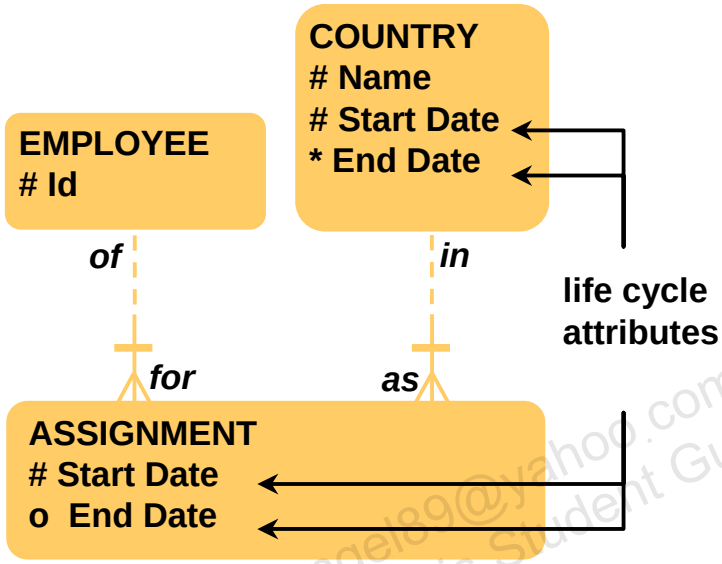
End Date Redundant?

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

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

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

Even a Country Has a Life Cycle



Modeling Changes Over Time

Countries Have a Life Cycle Too

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

This leads to the next model.

Modeling Changes Over Time

Time-related Constraints

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

- An ASSIGNMENT may only refer to a COUNTRY that *is valid* at the Start Date of the ASSIGNMENT.
- The obvious one: End Date must be past Start Date.
- A business rule: ASSIGNMENT periods may not overlap. The Start Date of an ASSIGNMENT for an EMPLOYEE may not be between any Start Date and End Date of an other ASSIGNMENT for the same EMPLOYEE.
- As for the previous constraint, but for End Date.
- You would probably not allow an ASSIGNMENT to be transferred to another COUNTRY, unless the ASSIGNMENT has not yet started, that is, the Start Date of the ASSIGNMENT is still in the future.
- This is an example of ***conditional nontransferability.***

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

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

Similar constraints apply to attribute End Date.

Referential Logic

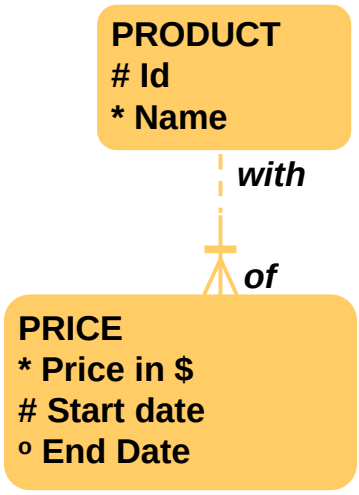
Note that, except for two, these constraints result from *referential logic* only. There may be more additional business constraints.

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

Implementation

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

Products and Prices



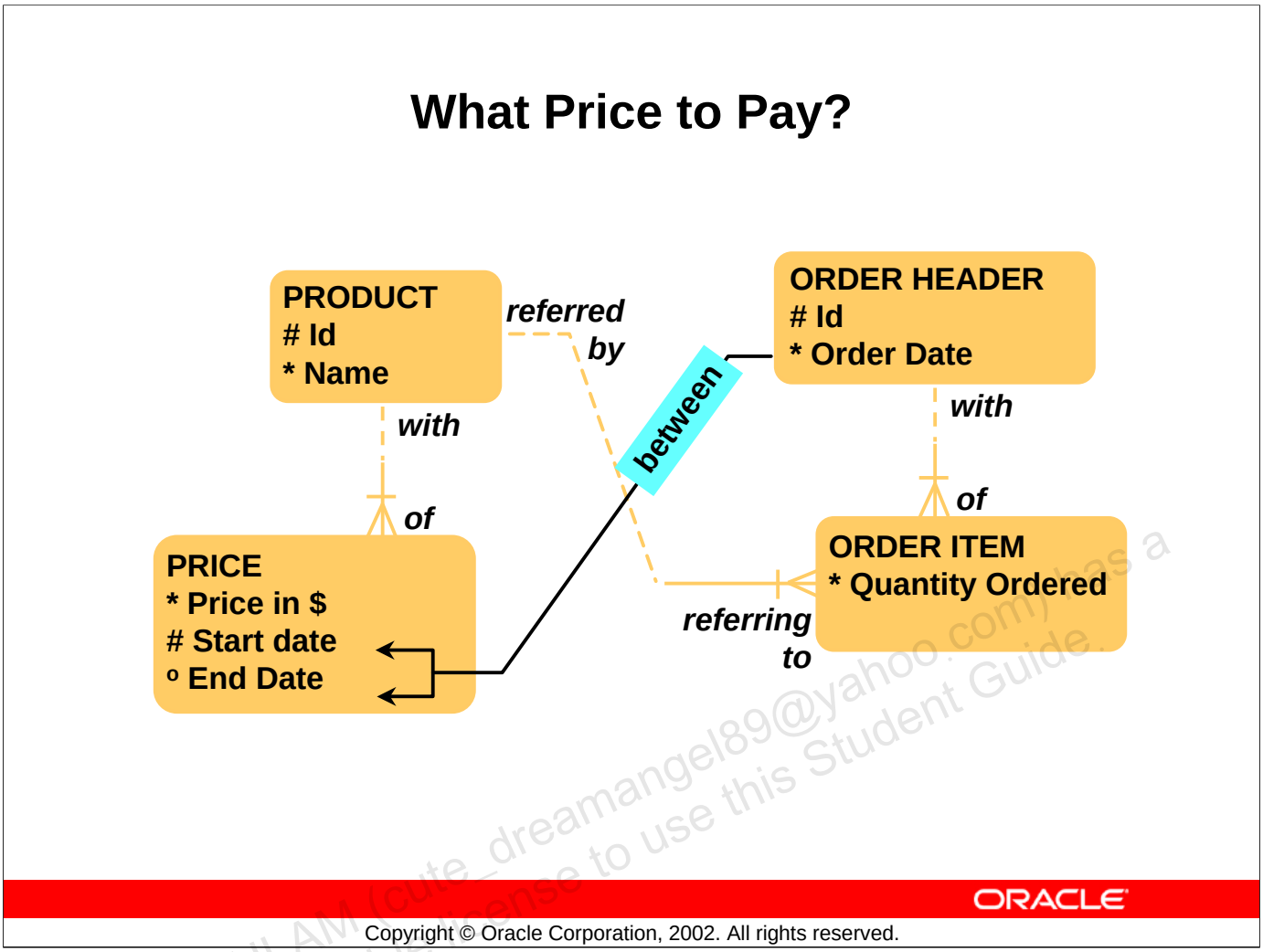
PRICE =
PRICED PRODUCT=
HISTORICAL PRICE

ORACLE

A Time Example: Prices

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

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



A Time Example: Prices

Introducing Order Header and Order Item

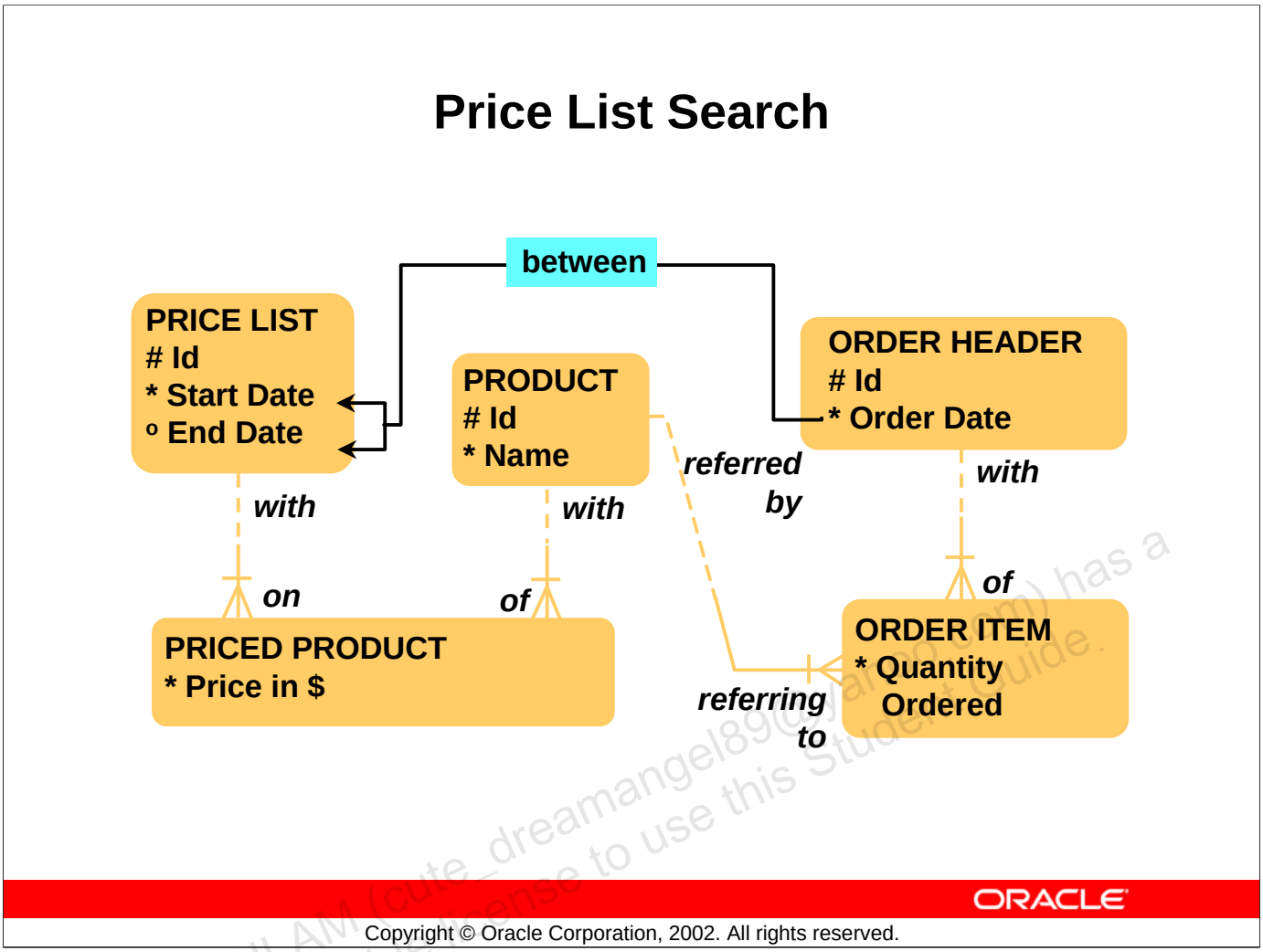
Here, entities ORDER HEADER and ORDER ITEM are introduced. An ORDER HEADER holds the information that applies to all items, like the Order Date and the relationship to the CUSTOMER that placed the order or the EMPLOYEE that handled it. (For clarity, these relationships are not drawn here.) The ORDER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

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

Order

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

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

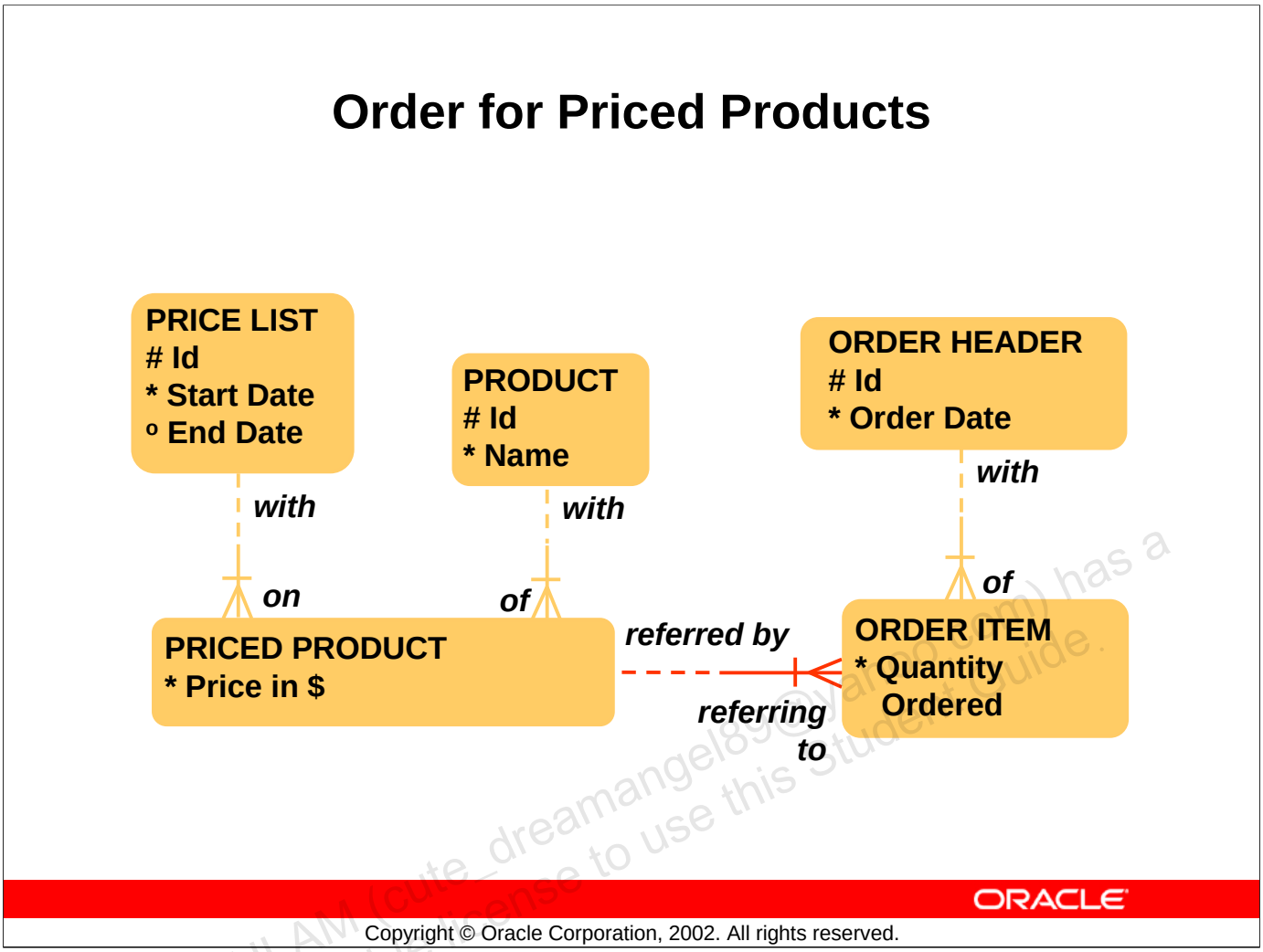


A Time Example: Prices

Price List

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

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

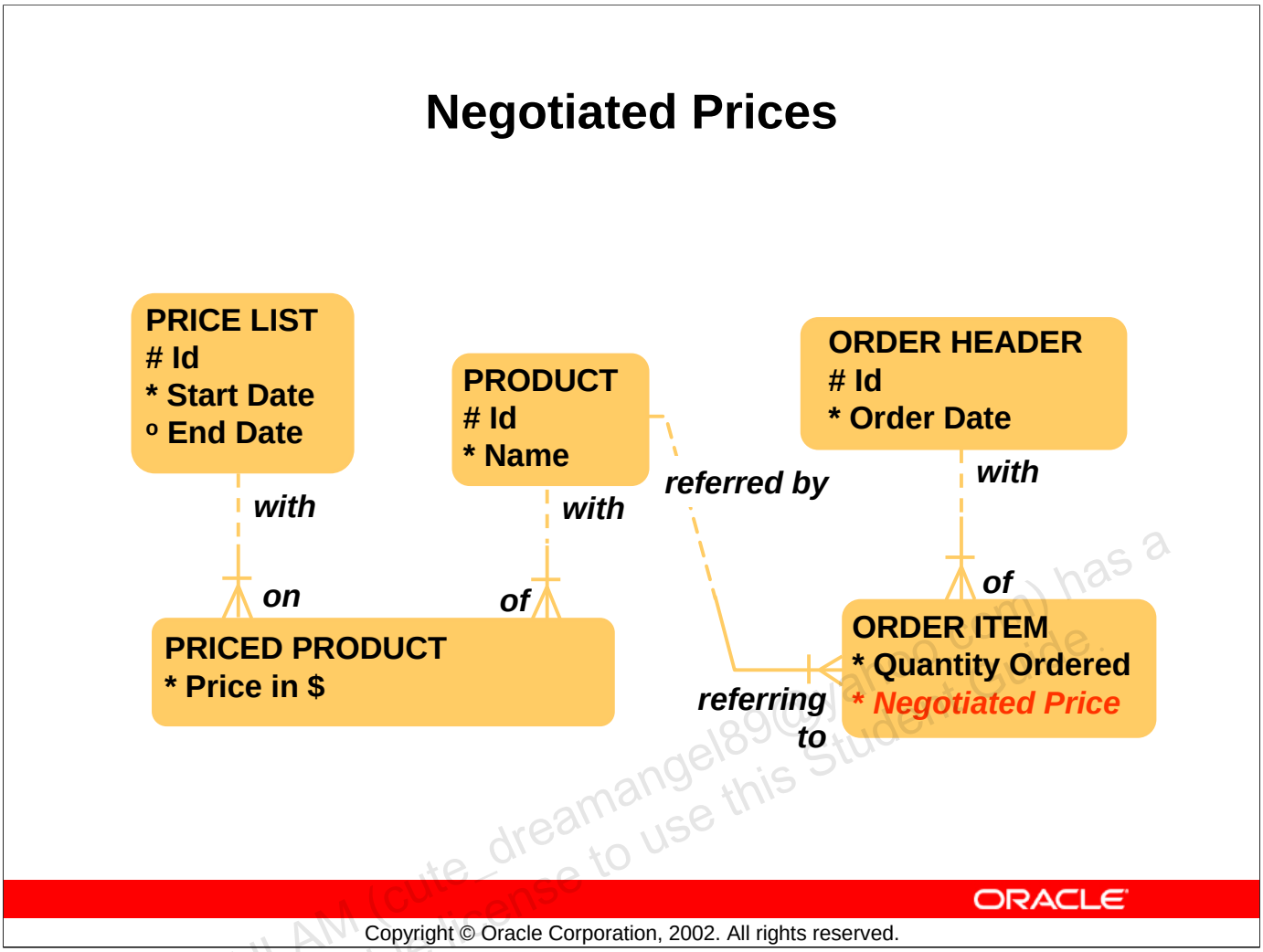


A Time Example: Prices

Buying a PRODUCT or a PRICED PRODUCT?

Another variant of a pricing model is shown here.

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



A Time Example: Prices

Negotiated Prices

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

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

A Time Example: Prices

Which Variant to Use and When?

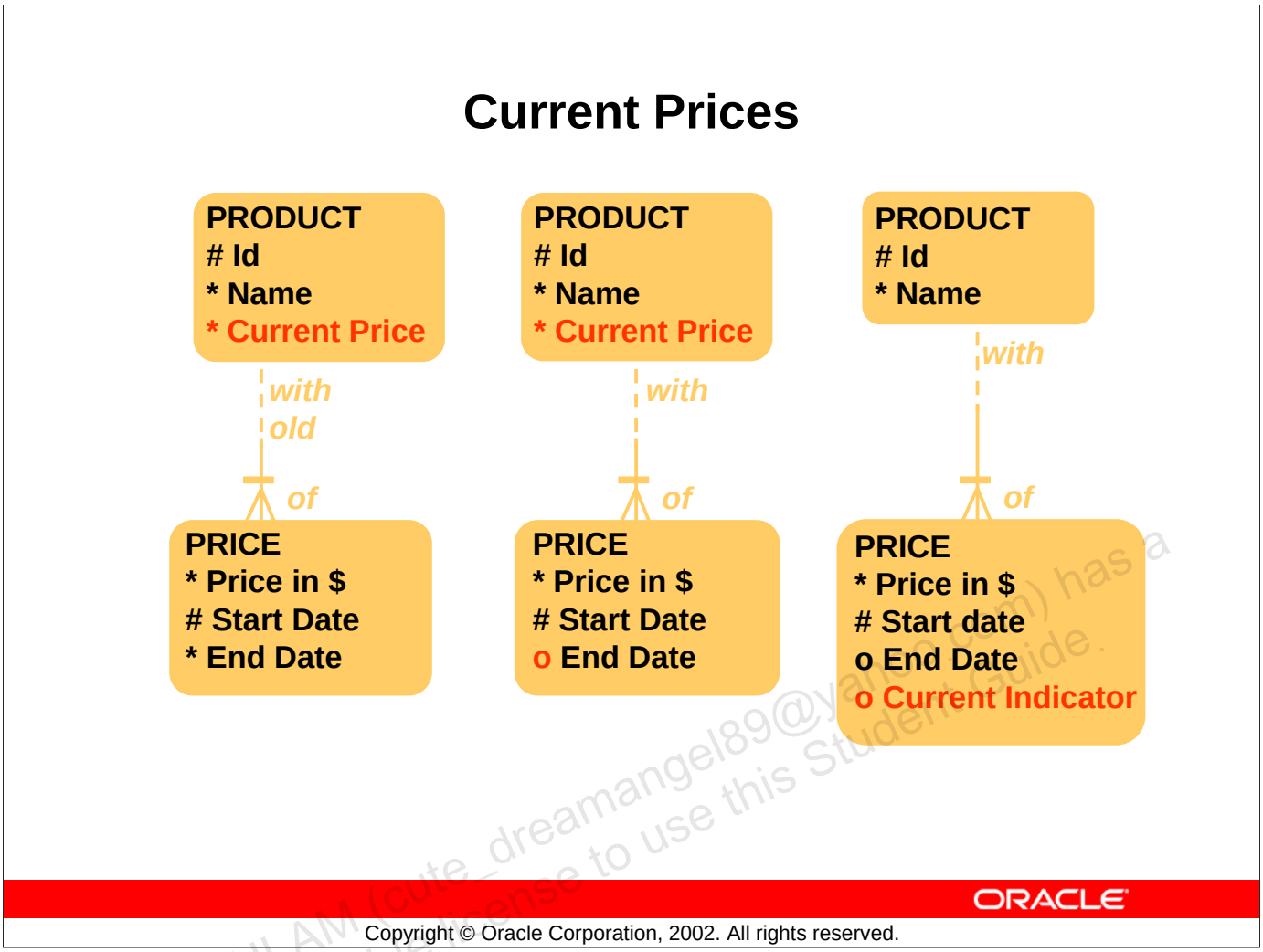
Typically, the model with the negotiated prices will occur where the number of ORDER ITEMS per ORDER HEADER is low, often just a single one, and where the value is high, as, for example, in the context of a used car business.

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

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

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

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



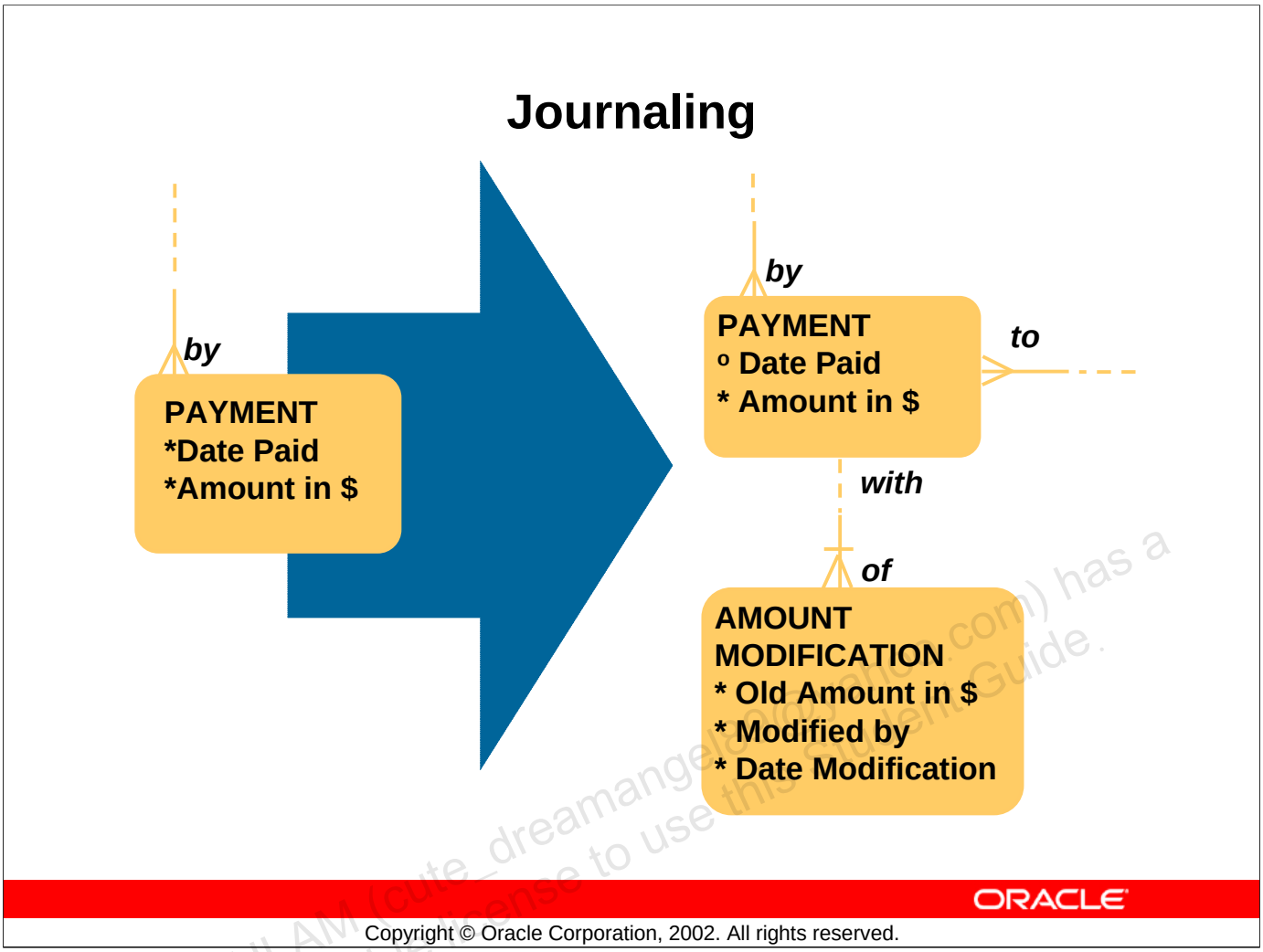
Current Price

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

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

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

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



Journaling

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

Consequences for the Model

A journal usually consists of both the modified value and the information about who did the modification and when it was done. This extra information can, of course, be expanded if you wish. Apart from the consequences for the conceptual data model, the system needs special journaling functionality: any business function that allows an update of Amount In should result in the requested update, plus the creation of an entity instance AMOUNT MODIFICATION with the proper values. Of course, the system would need special functions as well in order to do something with the logged data.

Journaling

No Journal Entity

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

Journaling Registers Only

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

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

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

Summary

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

ORACLE

Summary

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

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

Practices

- Shift
- Strawberry Wafer
- Bundles
- Product Structure

Practice: Shift

Museumplein, Amsterdam, March 21

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

ORACLE

Practice 5-1: Shift

Goal

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

Scenario

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

Your Assignment

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

Practice: Strawberry Wafer

- **Prices are at the same level within a country; prices are determined by the Global Pricing Department. Usually the prices for regular, global products are re-established once a year.**
- **Prices and availability for local specialties are determined by the individual shops. For example, the famous Norwegian Vafler med Jordbær (a delicious wafer with fresh strawberries) is only available in summer. Its price depends on the current local market price of fresh strawberries.**

ORACLE

Practice 5-2: Strawberry Wafer

Scenario

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

Your Assignment

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

Practice: Price list

prijslijst

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

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

inclusief BTW
16 September

ORACLE

Practice: Bundles(1)

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

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

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

A FamilyFeast(tm) consists of two BigBoxes(tm) plus two SweetTreats™ plus a small surprise.

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

ORACLE

Practice 5-3: Bundles

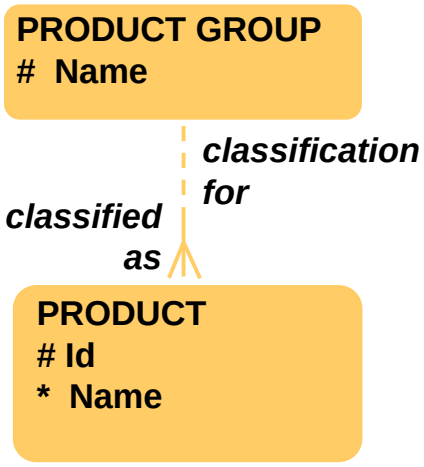
Goal

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

Scenario

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

Practice: Bundles(2)



ORACLE

Practice 5-3: Bundles

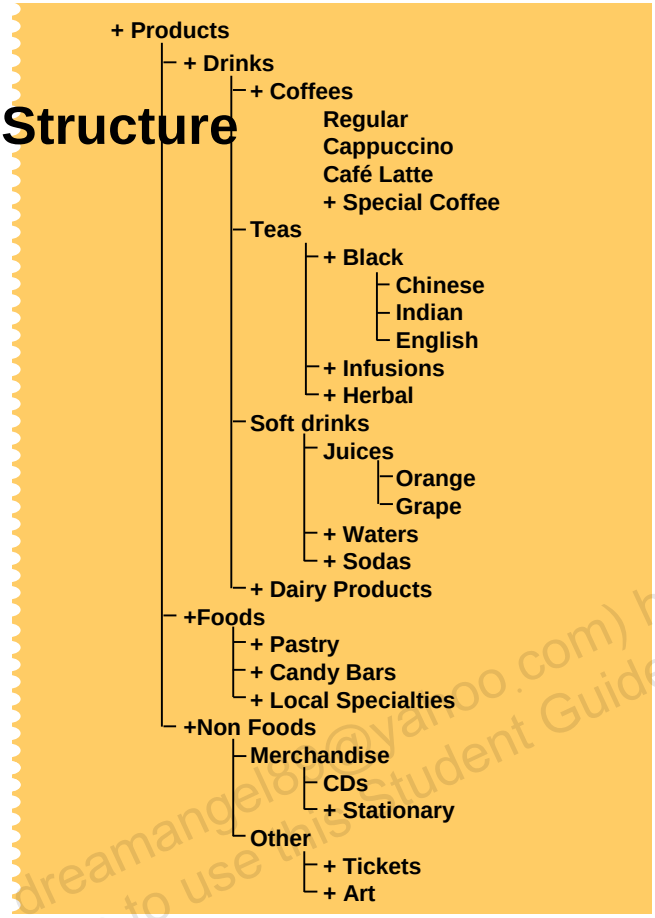
Bundles sell very well; all kinds of new bundles are expected to come.

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

Your Assignment

1. Modify the product part of the model in such a way that the desired calculations can be completed.
2. Change the model in such a way that it allows for:

Practice: Product Structure



Practice 5-4: Product Structure

Goal

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

Scenario

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

Your Assignment

1. Create a model for a product classification structure.
2. (Optional) How would you treat the bundled products?