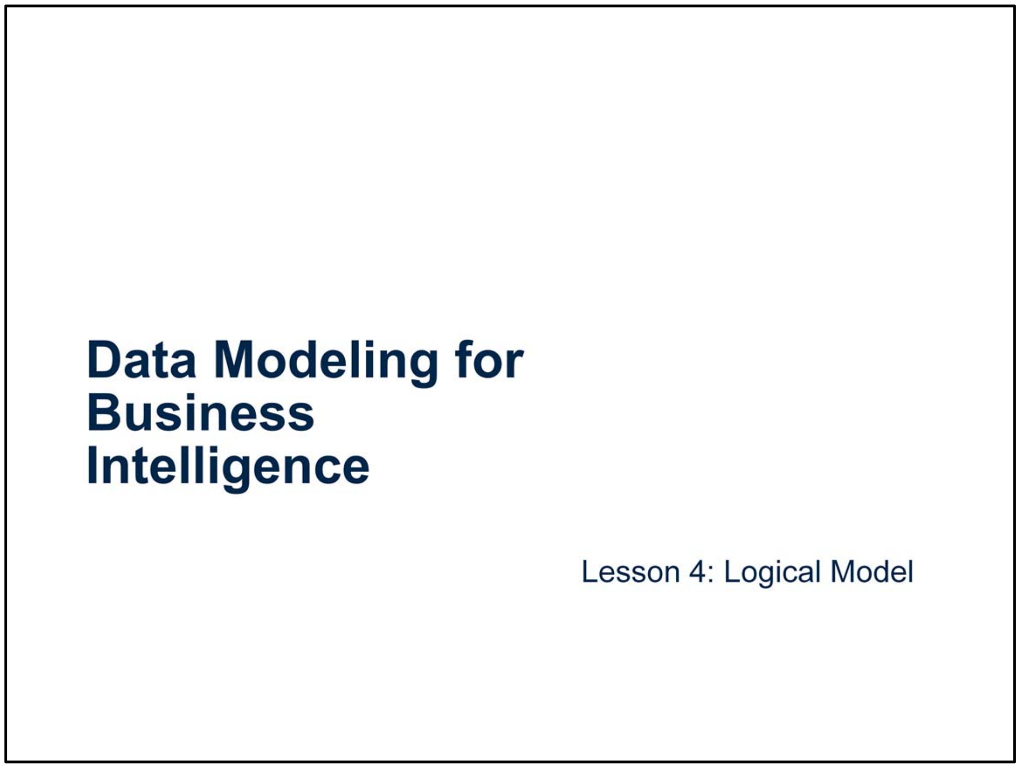
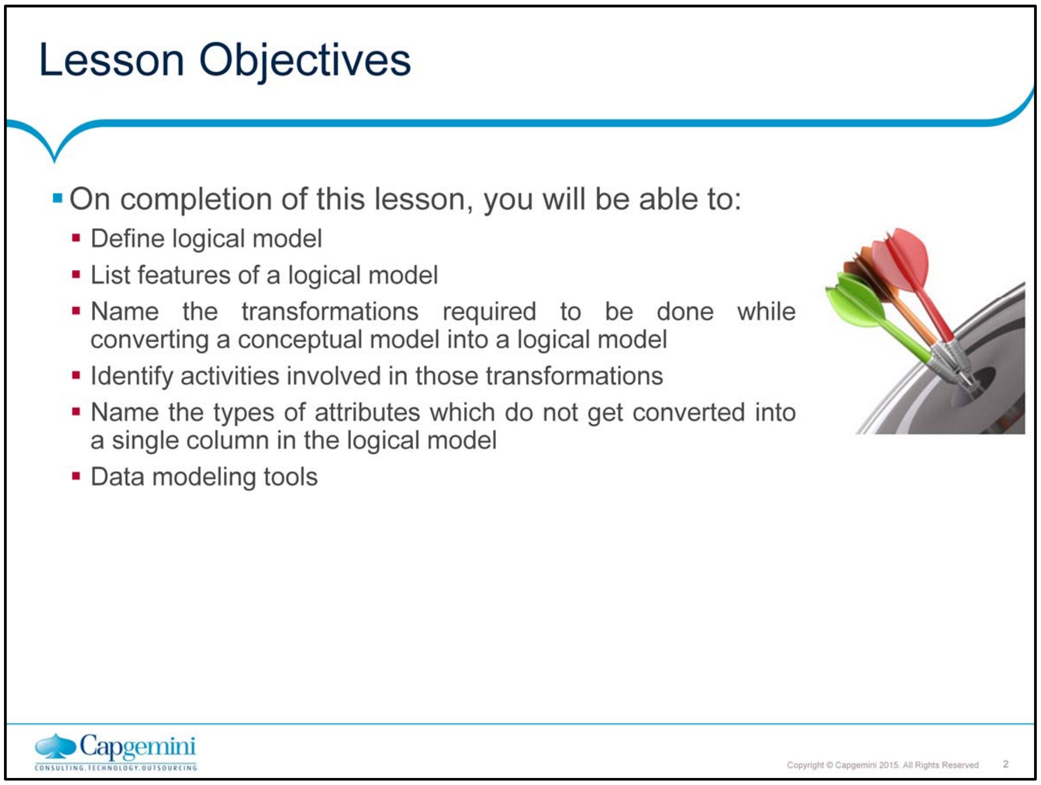


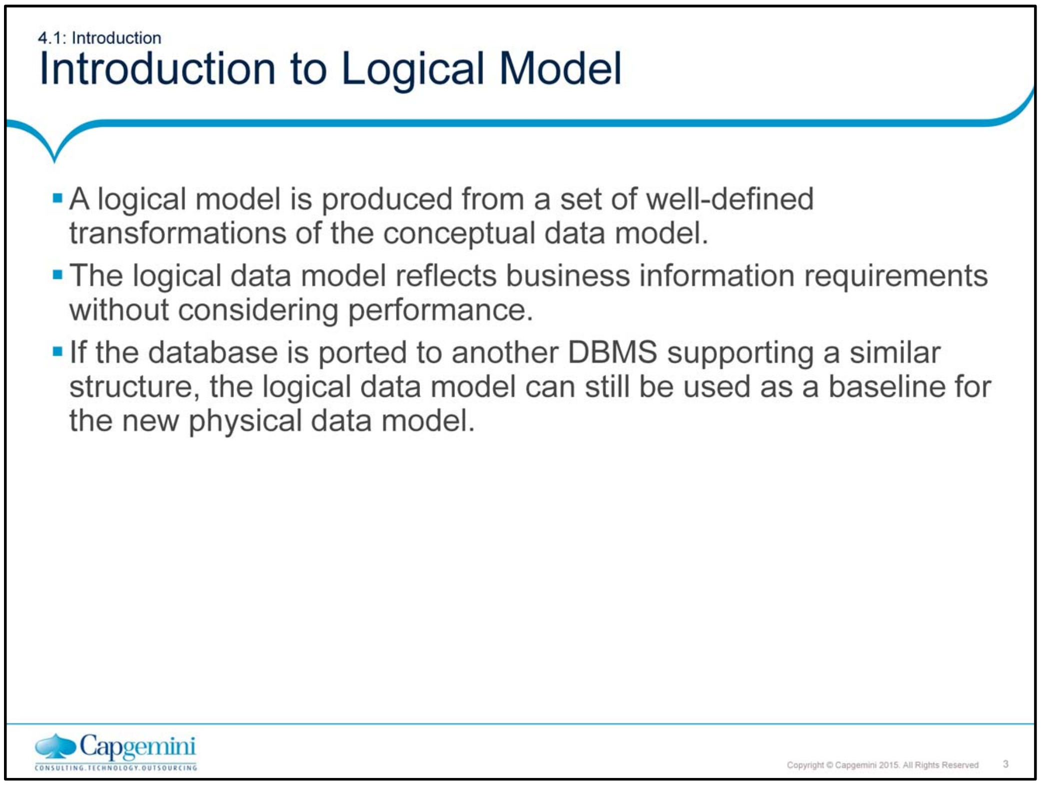
Data Modeling for Business Intelligence

Lesson 4: Logical Model



Lesson Objectives

- On completion of this lesson, you will be able to:
  - Define logical model
  - List features of a logical model
  - Name the transformations required to be done while converting a conceptual model into a logical model
  - Identify activities involved in those transformations
  - Name the types of attributes which do not get converted into a single column in the logical model
  - Data modeling tools


4.1: Introduction

Introduction to Logical Model

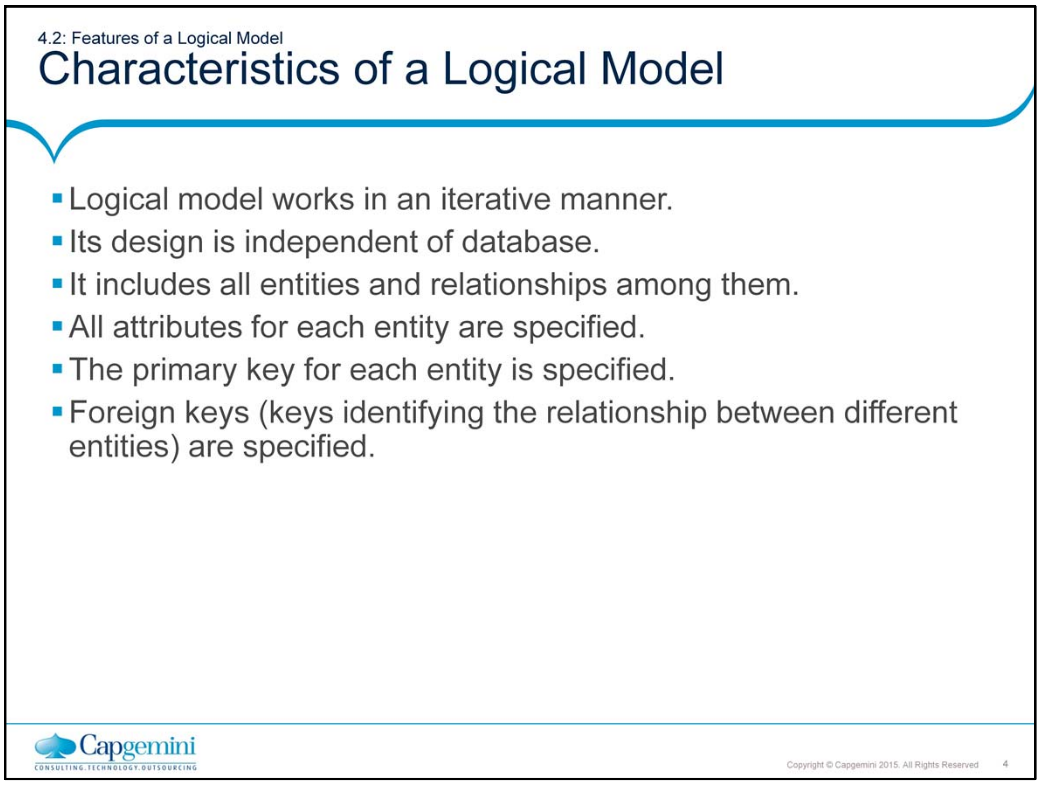
- A logical model is produced from a set of well-defined transformations of the conceptual data model.
- The logical data model reflects business information requirements without considering performance.
- If the database is ported to another DBMS supporting a similar structure, the logical data model can still be used as a baseline for the new physical data model.

The primary purpose of logical data modeling is to document the business information structures, processes, rules, and relationships by a single view.

The logical data model helps to address the following:
1) Validation of the functional application model against business requirements
2) The product and implementation independent requirements for the physical database design (Physical Data Modeling)
3) Clear and unique identification of all business entities in the system along with their relations.

Note:
Without the logical data model, the stored business information is described by
a functional model or conceptual model. Without the logical data model, there is
no single view of all data, and data normalization is impossible.

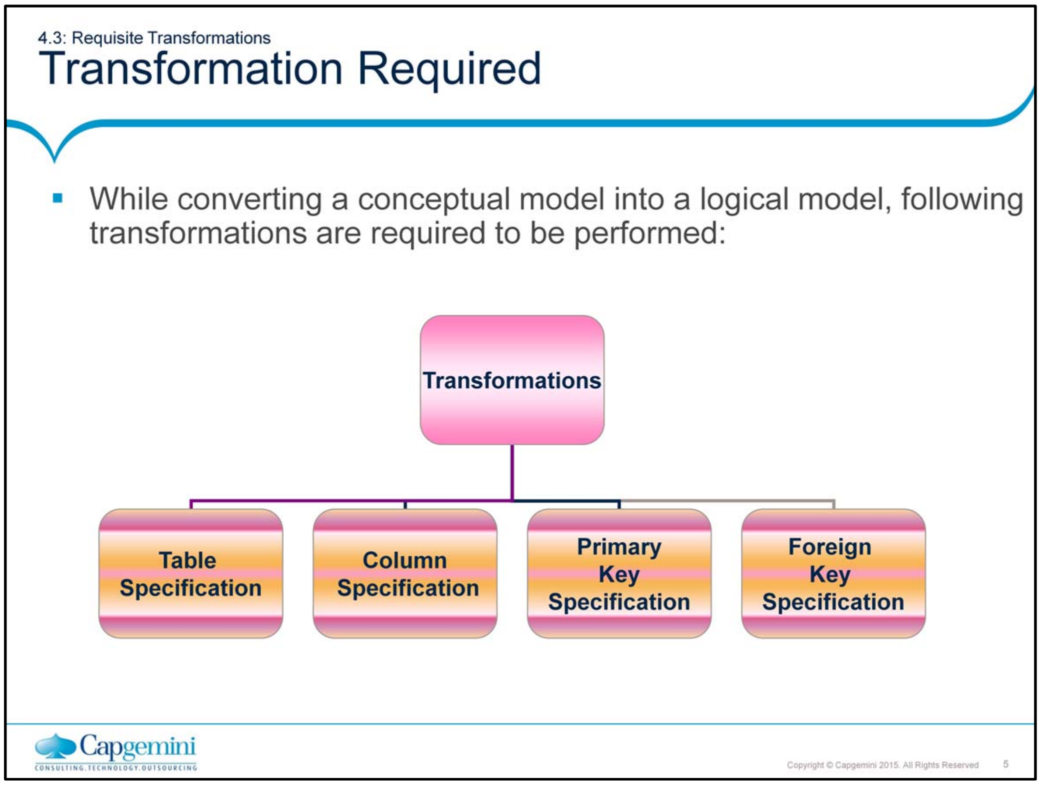
In this case, the physical data model has to be designed from a functional model. This will potentially cause performance problems, and data inconsistency and redundancies, which can result in an inefficient physical design.



4.2: Features of a Logical Model

Characteristics of a Logical Model

- Logical model works in an iterative manner.
- Its design is independent of database.
- It includes all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.

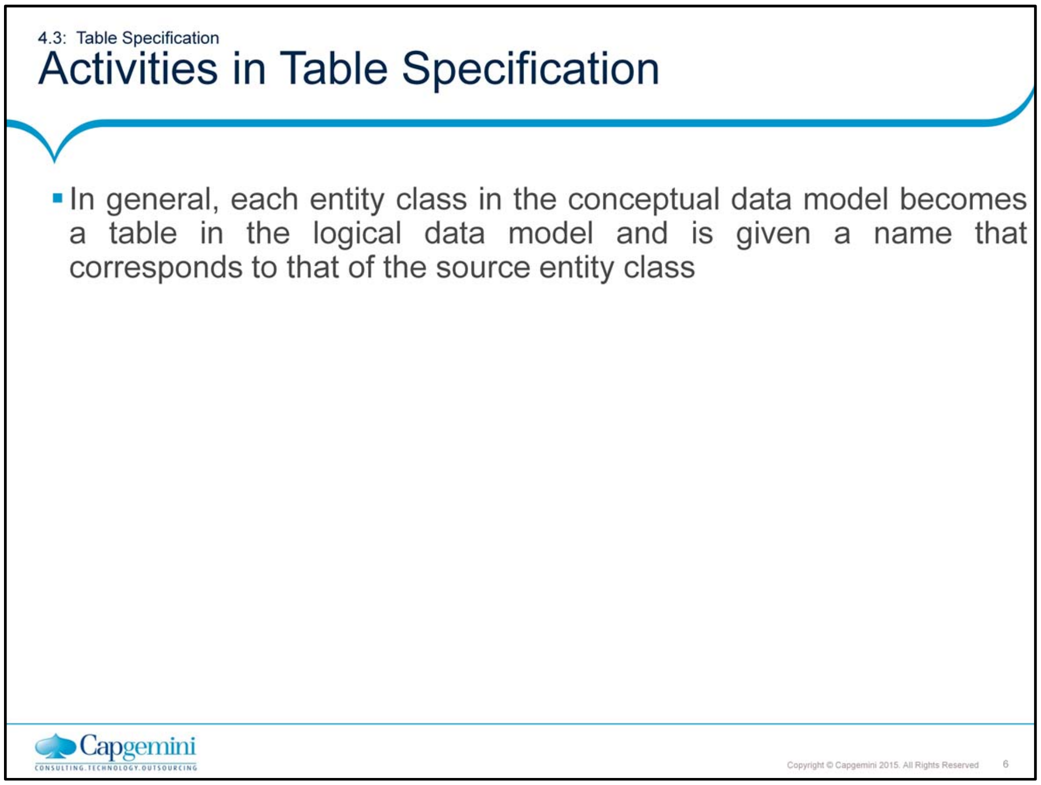


4.3: Requisite Transformations

Transformation Required

- While converting a conceptual model into a logical model, following transformations are required to be performed:

Transformations
Table Specification
Column Specification
Primary Key Specification
Foreign Key Specification



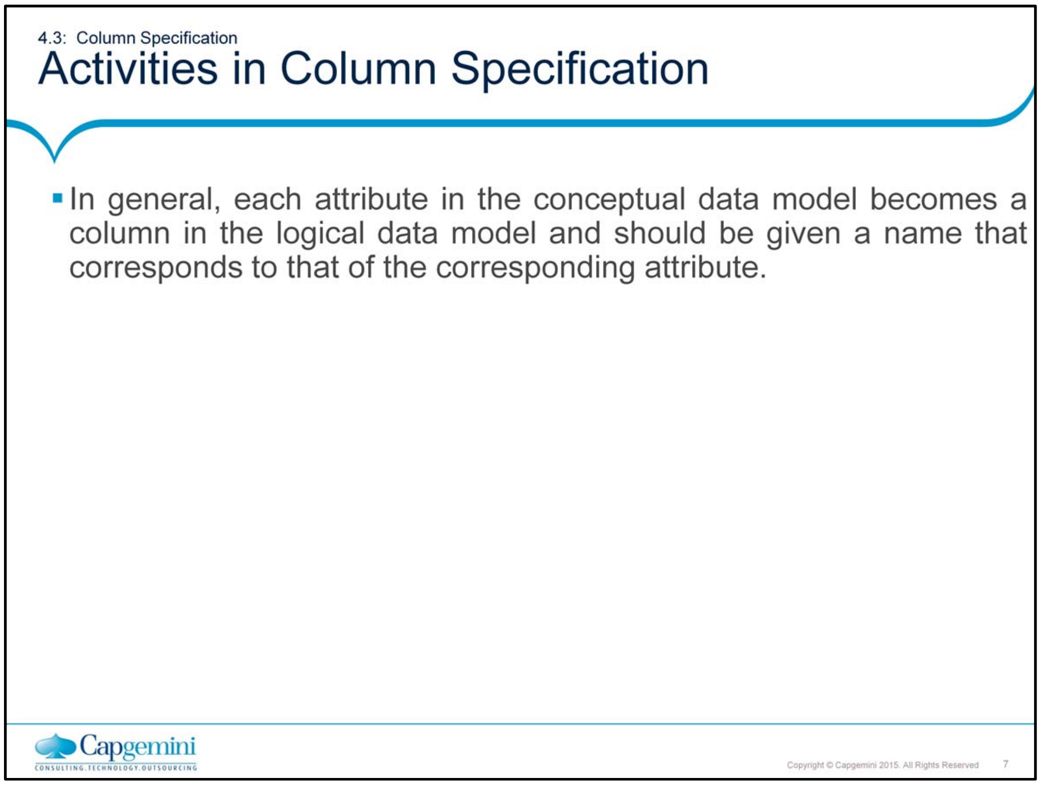
4.3: Table Specification

Activities in Table Specification

- In general, each entity class in the conceptual data model becomes a table in the logical data model and is given a name that corresponds to that of the source entity class

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

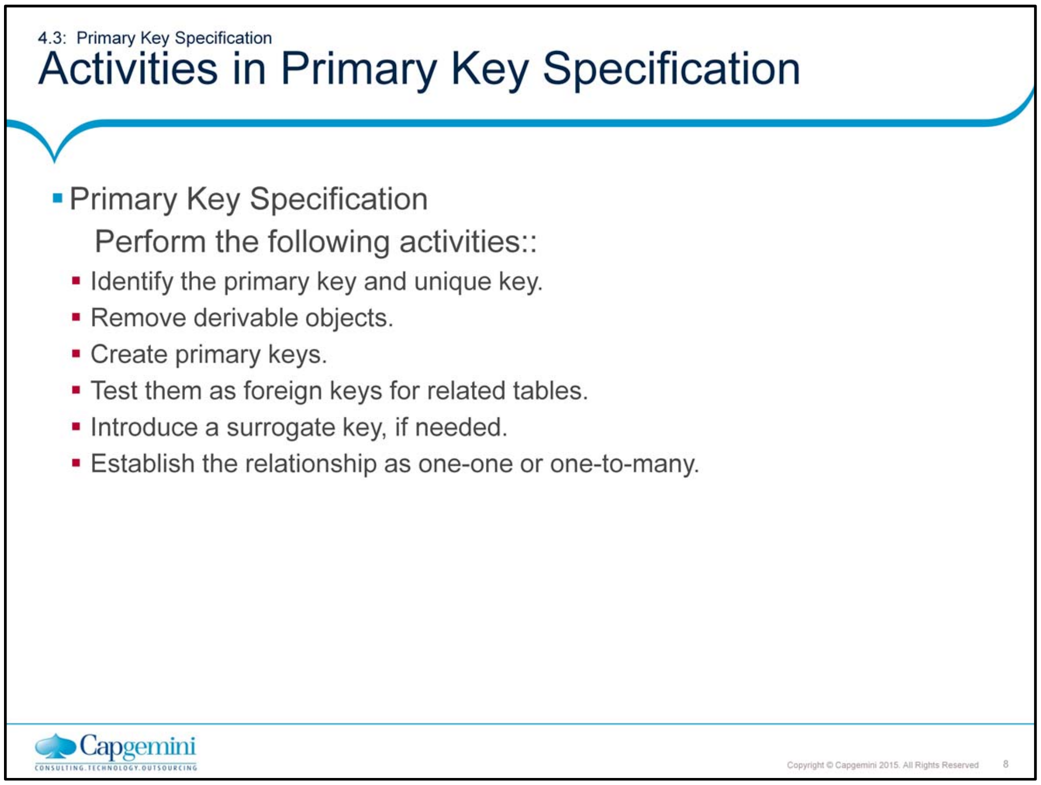
6



4.3: Column Specification

Activities in Column Specification

- In general, each attribute in the conceptual data model becomes a column in the logical data model and should be given a name that corresponds to that of the corresponding attribute.



4.3: Primary Key Specification

Activities in Primary Key Specification

- Primary Key Specification
  Perform the following activities::
  - Identify the primary key and unique key.
  - Remove derivable objects.
  - Create primary keys.
  - Test them as foreign keys for related tables.
  - Introduce a surrogate key, if needed.
  - Establish the relationship as one-one or one-to-many.

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

Activities in Primary Key Specification:

Existing columns are assessed for the primary keys; and if required, surrogate keys are introduced.

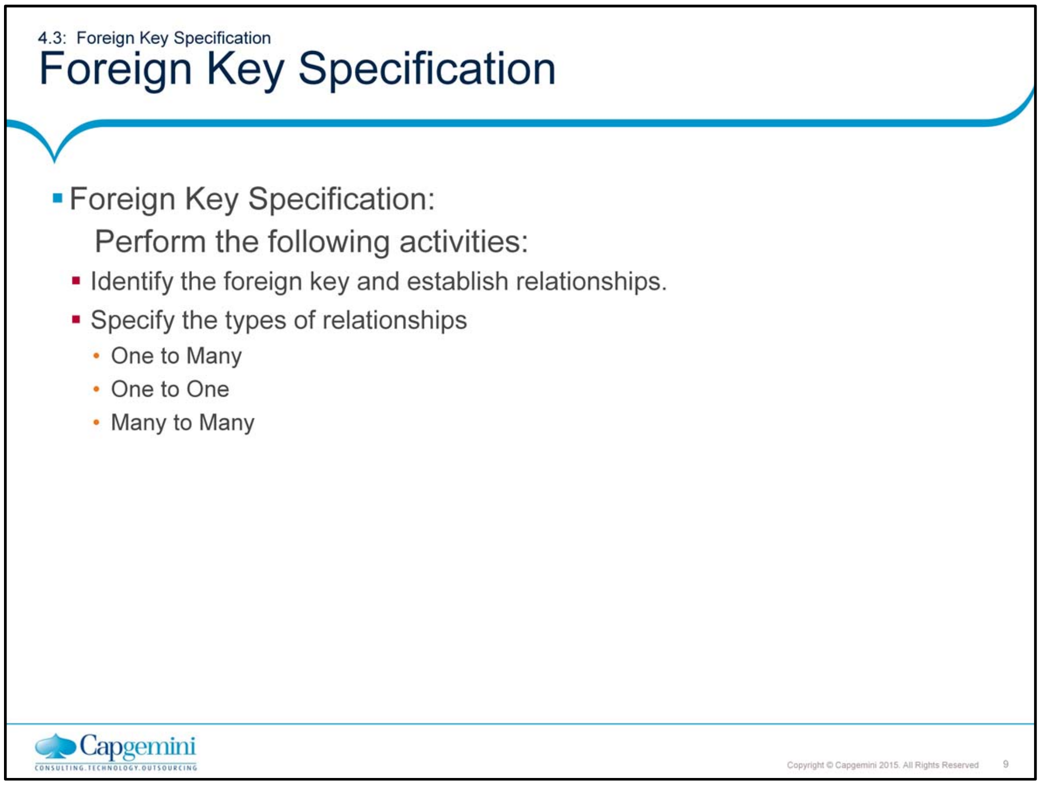
To access data in a relational database, we need to be able to locate specific rows of a table by specifying values for their primary key column or columns.

In particular:

- We must be able to unambiguously specify the row that corresponds to a particular real-world entity instance. When a payment for an account arrives, we need to be able to retrieve the single relevant row in the **Account** table by specifying the Account Number that was supplied with the payment.
- Relationships are implemented using foreign keys, which must each point to one row only. Imagine the problems if we had an insurance policy that referred to customer number “12345” but found two or more rows with that value in the **Customer** table.

So we require that a primary key be *unique*.

- A very simple way of meeting all of the requirements is to invent a new column for each table, specifically to serve as its primary key, and to assign a different system-generated value to each row, and, by extension, to the corresponding entity instance. We refer to such a column as a **surrogate key**, which is typically named by appending “ID” (or, less often, “Number” or “No”) to the table name. Familiar examples are customer IDs, employee IDs, and account numbers allocated by the system.

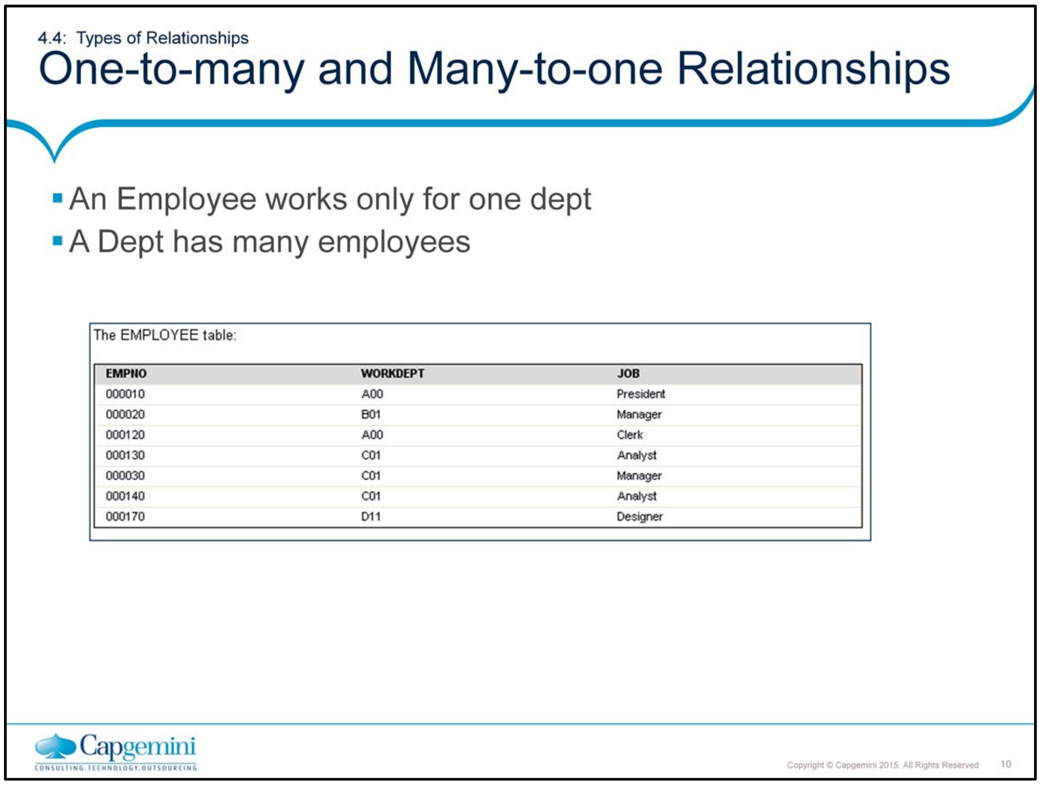


4.3: Foreign Key Specification

Foreign Key Specification

- Foreign Key Specification:
  Perform the following activities:
  - Identify the foreign key and establish relationships.
  - Specify the types of relationships
    - One to Many
    - One to One
    - Many to Many

Foreign keys are our means of implementing one-to-many and occasionally one-to-one relationships.



4.4: Types of Relationships

One-to-many and Many-to-one Relationships

- An Employee works only for one dept
- A Dept has many employees

The EMPLOYEE table:

EMPNO	WORKDEPT	JOB
000010	A00	President
000020	B01	Manager
000120	A00	Clerk
000130	C01	Analyst
000030	C01	Manager
000140	C01	Analyst
000170	D11	Designer

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

An employee can work in only one department; this relationship is single-valued for employees. On the other hand, one department can have many employees; this relationship is multi-valued for departments. The relationship between employees (single-valued) and departments (multi-valued) is a one-to-many relationship.

To define tables for each one-to-many and each many-to-one relationship:

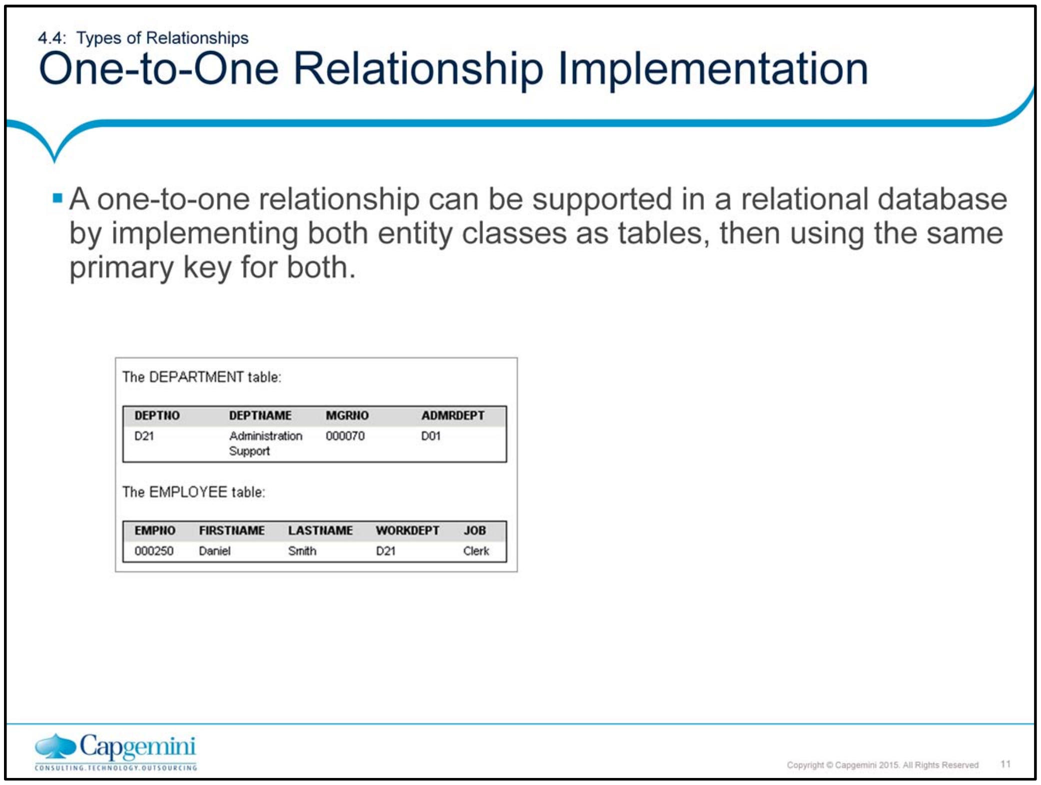
- Group all the relationships for which the "many" side of the relationship is the same entity.
- Define a single table for all the relationships in the group.

One-to-Many Relationship Implementation

When moving from a conceptual to a logical data model, however, we work from a diagram to tables and apply the following rule

A one-to-many relationship is supported in a relational database by holding the primary key of the table representing the entity class at the
“one” end of the relationship as a foreign key in the table representing the entity class at the “many” end of the relationship.

In the logical data model, therefore, we create, in the table representing the entity class at the “many” end of the relationship, a copy of the primary
key of the entity class at the “one” end of the relationship.



4.4: Types of Relationships

One-to-One Relationship Implementation

- A one-to-one relationship can be supported in a relational database by implementing both entity classes as tables, then using the same primary key for both.

The DEPARTMENT table:

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
D21	Administration Support	000070	D01

The EMPLOYEE table:

EMPNO	FIRSTNAME	LASTNAME	WORKDEPT	JOB
000250	Daniel	Smith	D21	Clerk

11

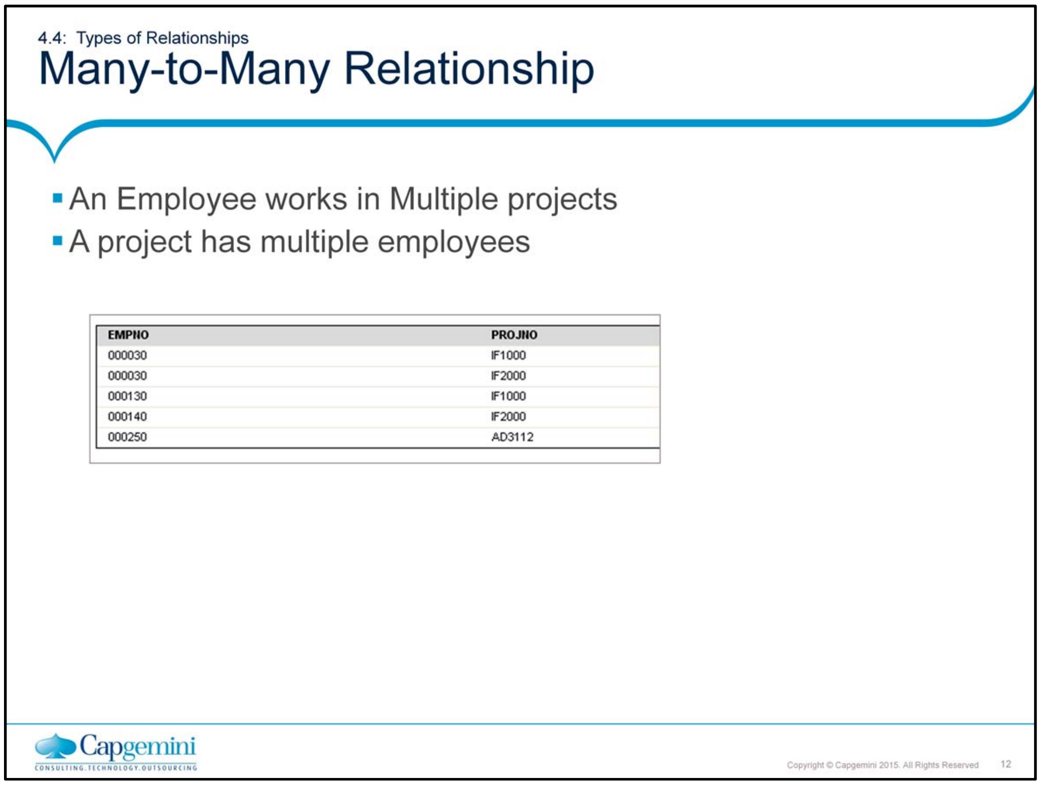
One-to-one relationships are single-valued in both directions. A manager manages one department; a department has only one manager. The questions, "Who is the manager of Department C01?", and "What department does Sally Kwan manage?" both have single answers.

The relationship can be assigned to either the DEPARTMENT table or the EMPLOYEE table. Because all departments have managers, but not all employees are managers, it is most logical to add the manager to the DEPARTMENT table, as shown in the following example.

The above table shows the representation of a one-to-one relationship.

You can have more than one table describing the attributes of the same set of entities. For example, the EMPLOYEE table shows the number of the department to which an employee is assigned, and the DEPARTMENT table shows which manager is assigned to each department number. To retrieve both sets of attributes simultaneously, you can join the two tables on the matching columns, as shown in the following example. The values in WORKDEPT and DEPTNO represent the same entity, and represent a *join path* between the DEPARTMENT and EMPLOYEE tables.

When you retrieve information about an entity from more than one table, ensure that equal values represent the same entity. The connecting columns can have different names (like WORKDEPT and DEPTNO in the previous example), or they can have the same name (like the columns called DEPTNO in the department and project tables).


4.4: Types of Relationships

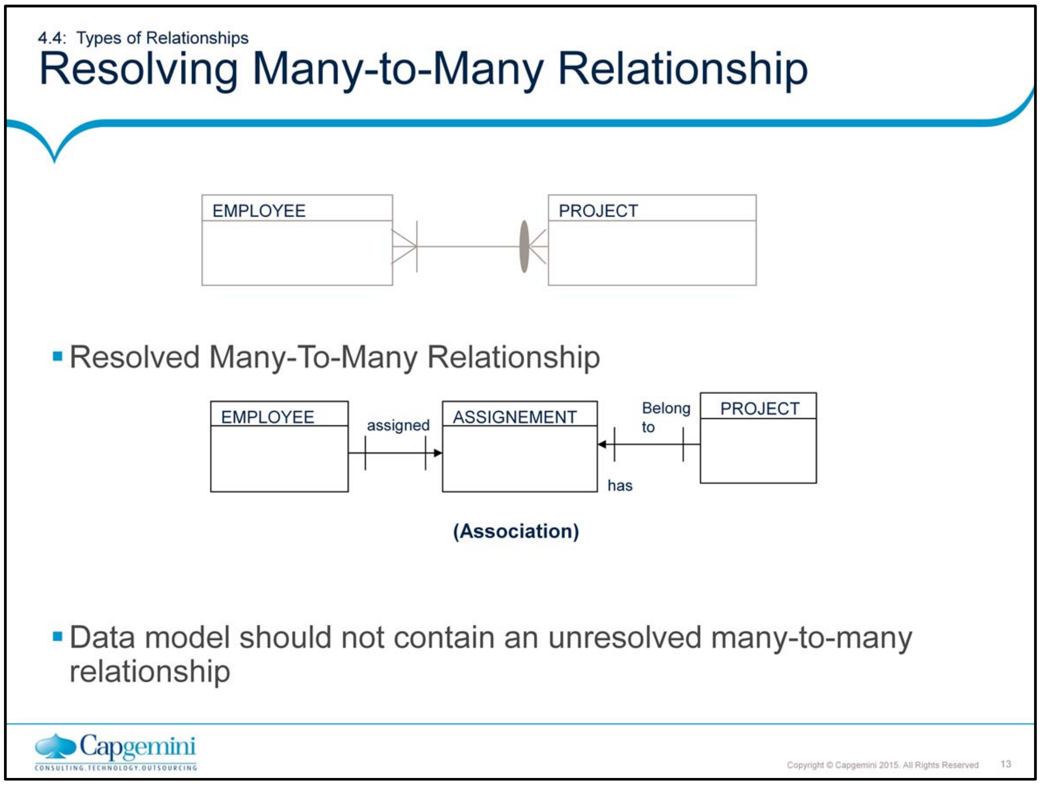
Many-to-Many Relationship

- An Employee works in Multiple projects
- A project has multiple employees

EMPNO	PROJNO
000030	IF1000
000030	IF2000
000130	IF1000
000140	IF2000
000250	AD3112

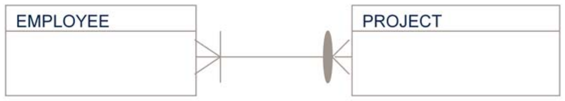
Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

A relationship that is multi-valued in both directions is a many-to-many relationship. An employee can work on more than one project, and a project can have more than one employee. The questions "What does Dolores Quintana work on?", and "Who works on project IF1000?" both yield multiple answers. A many-to-many relationship can be expressed in a table with a column for each entity ("employees" and "projects").

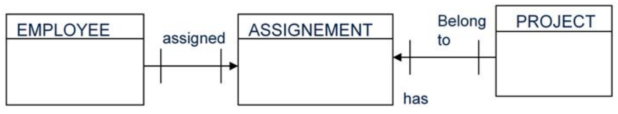


4.4: Types of Relationships

Resolving Many-to-Many Relationship



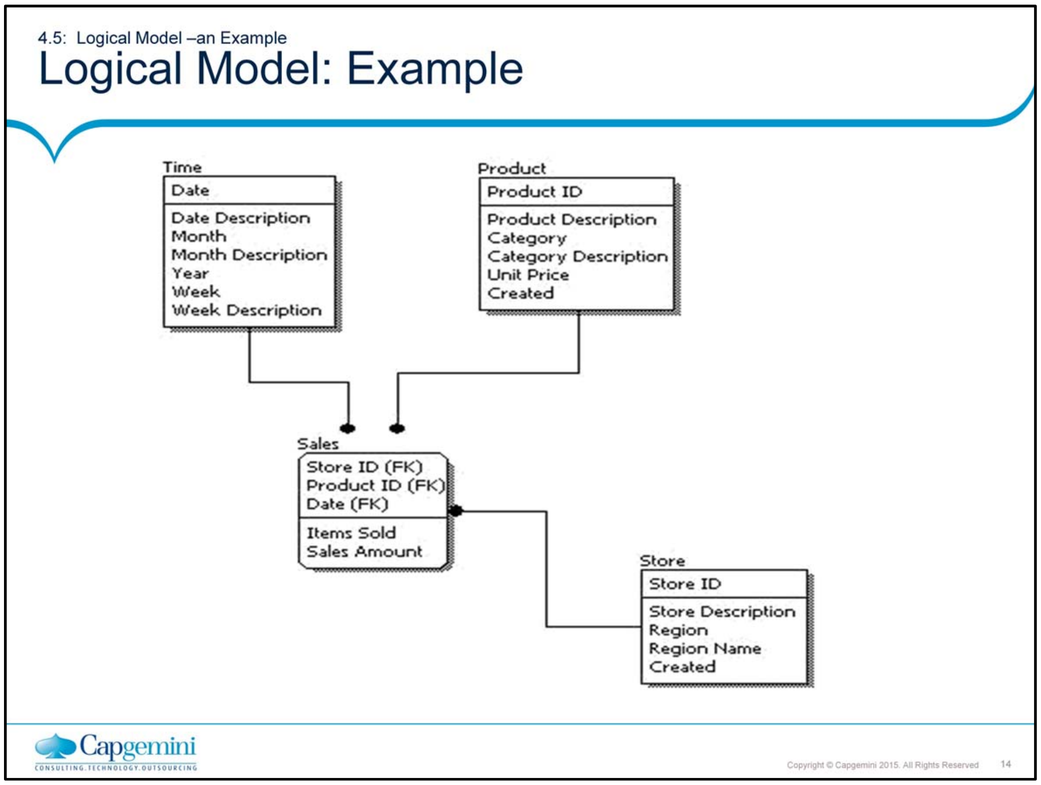
- Resolved Many-To-Many Relationship



(Association)

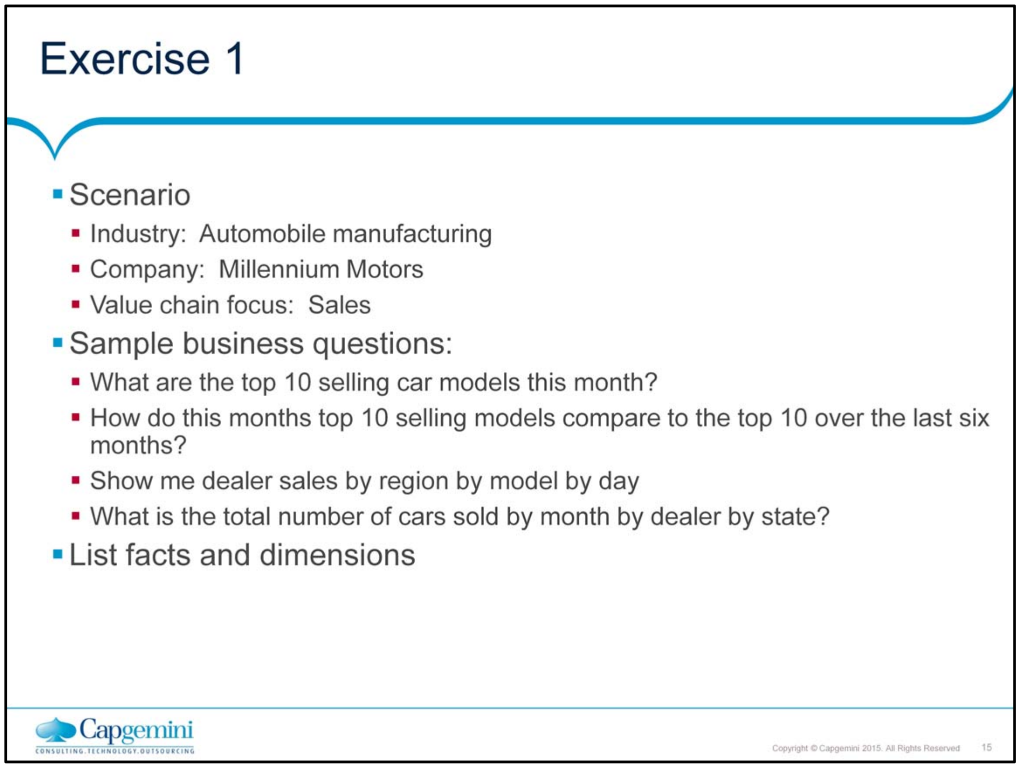
- Data model should not contain an unresolved many-to-many relationship

Many-to-many relationships cannot be used in the data model because they cannot be represented by the relational model. Therefore, many-to-many relationships must be resolved early in the modelling process. The strategy for resolving many-to-many relationship is to replace the relationship with an association entity and then relate the two original entities to the association entity



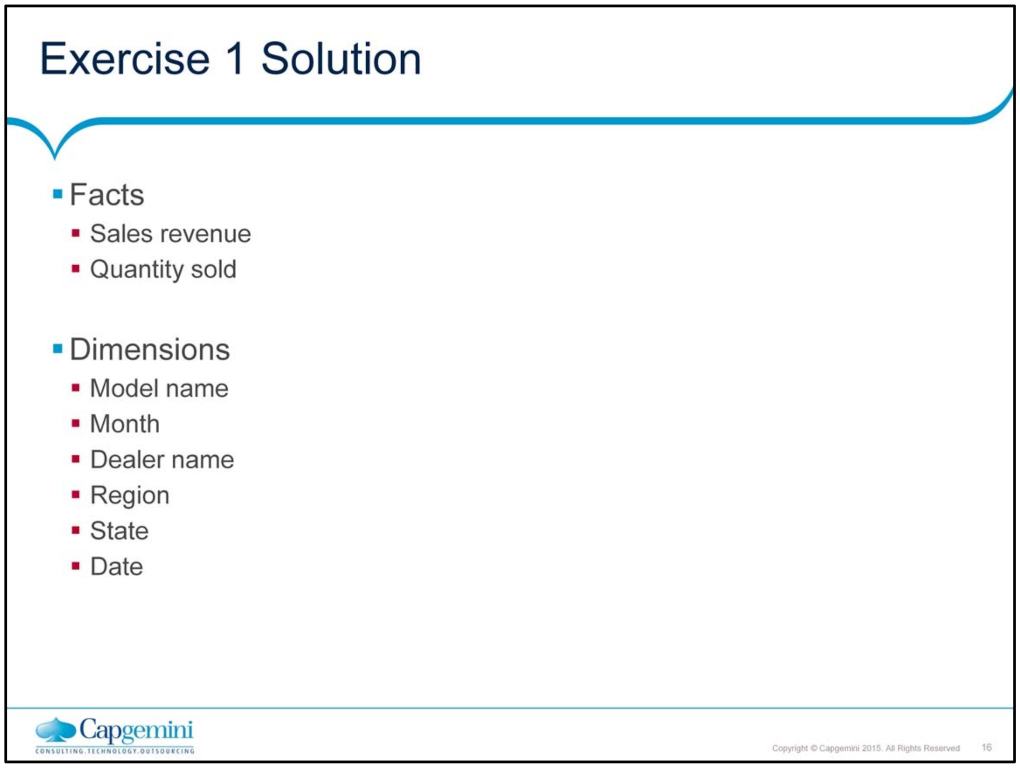
4.5: Logical Model –an Example

Logical Model: Example



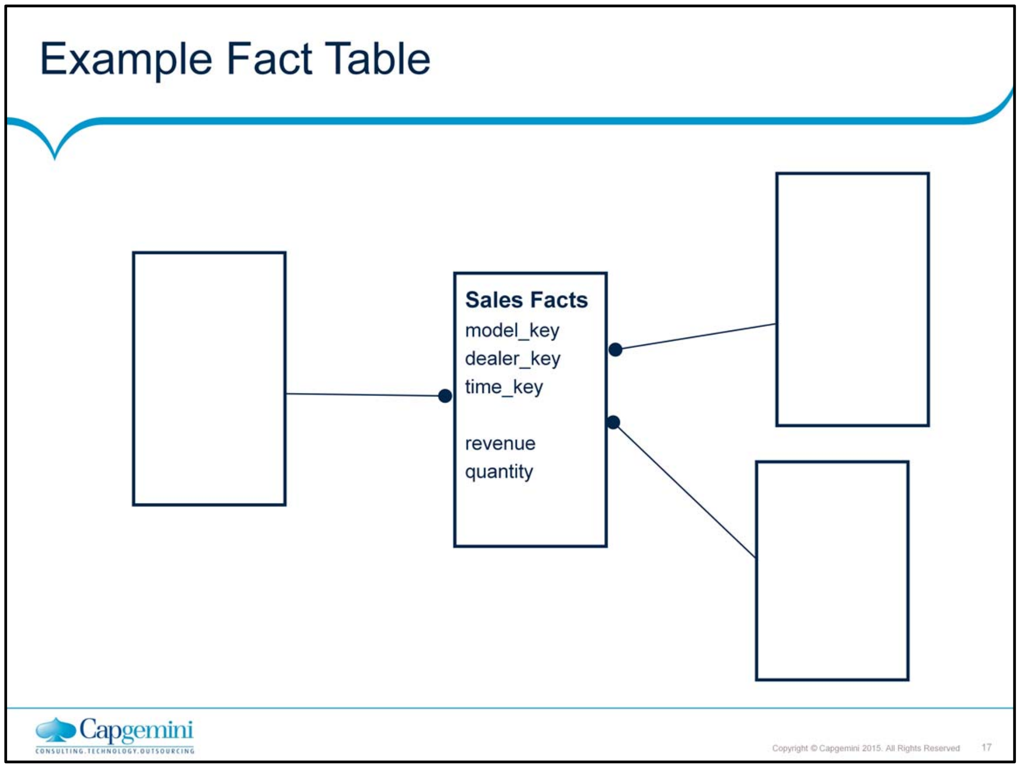
Exercise 1

- Scenario
  - Industry: Automobile manufacturing
  - Company: Millennium Motors
  - Value chain focus: Sales
- Sample business questions:
  - What are the top 10 selling car models this month?
  - How do this months top 10 selling models compare to the top 10 over the last six months?
  - Show me dealer sales by region by model by day
  - What is the total number of cars sold by month by dealer by state?
- List facts and dimensions

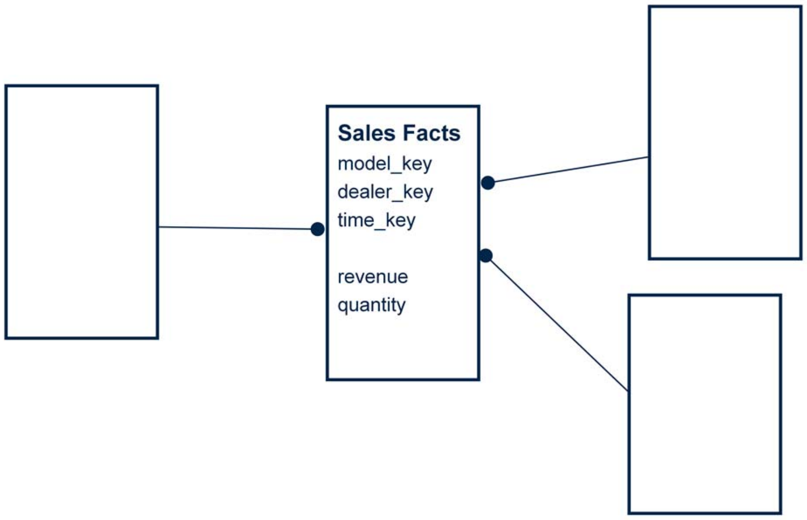


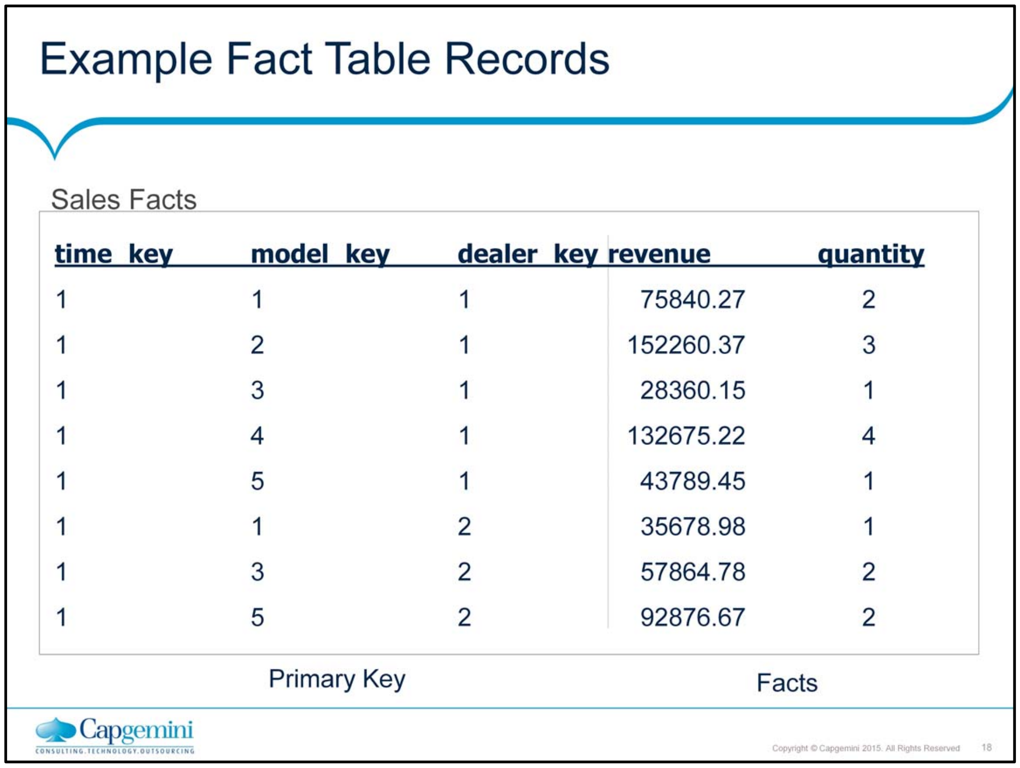
Exercise 1 Solution

- Facts
  - Sales revenue
  - Quantity sold
- Dimensions
  - Model name
  - Month
  - Dealer name
  - Region
  - State
  - Date



Example Fact Table



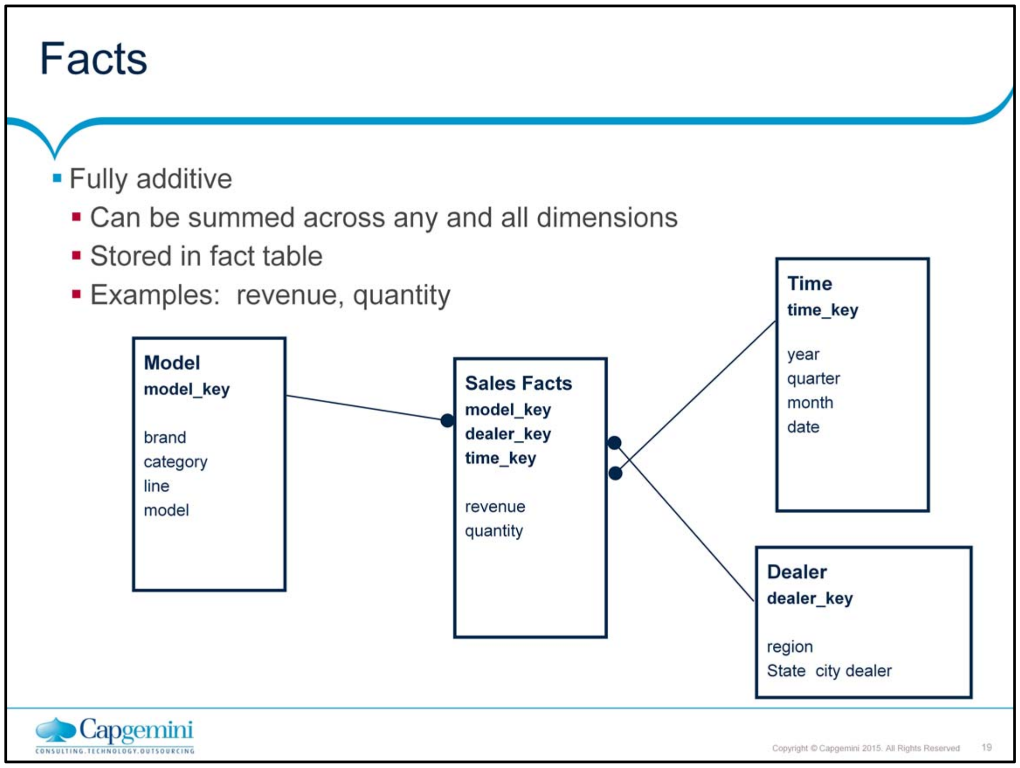


Example Fact Table Records

Sales Facts

time_key	model_key	dealer_key	revenue	quantity
1	1	1	75840.27	2
1	2	1	152260.37	3
1	3	1	28360.15	1
1	4	1	132675.22	4
1	5	1	43789.45	1
1	1	2	35678.98	1
1	3	2	57864.78	2
1	5	2	92876.67	2

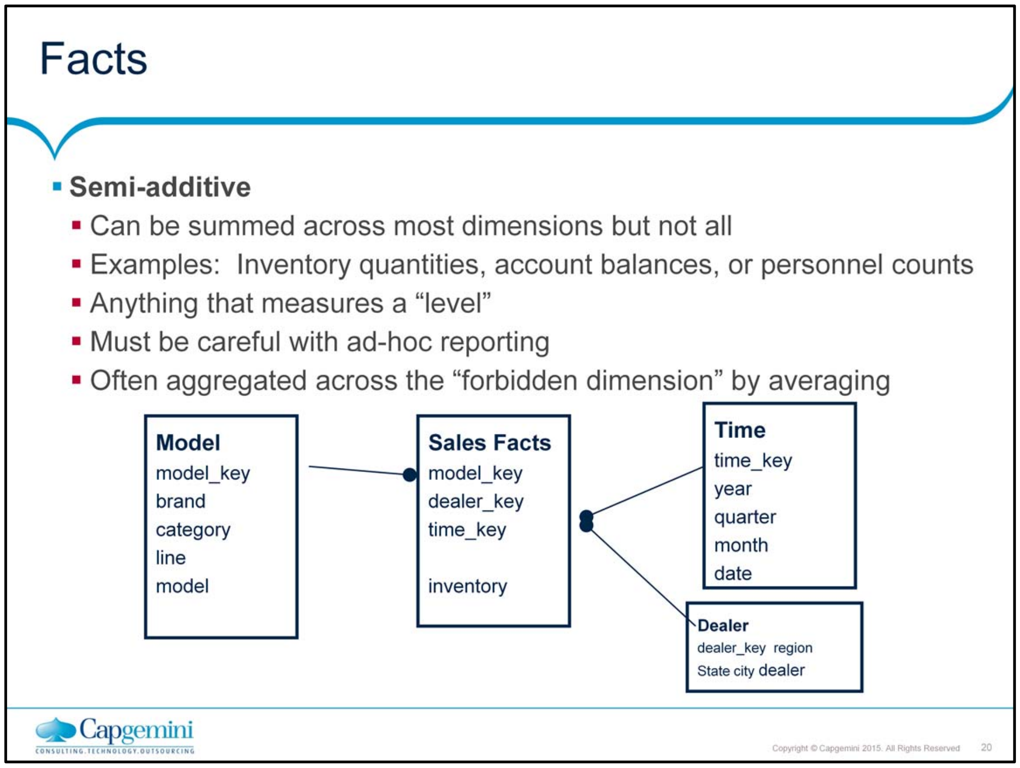
Primary Key Facts



Facts

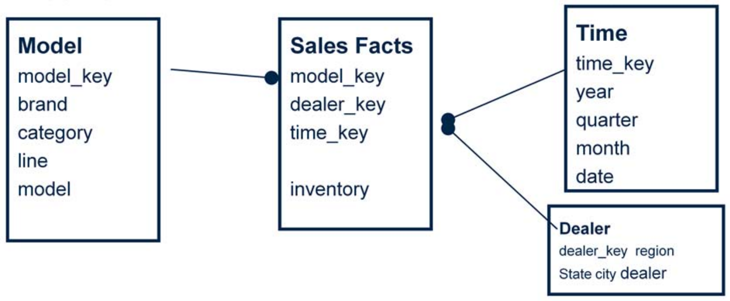
- Fully additive
  - Can be summed across any and all dimensions
  - Stored in fact table
  - Examples: revenue, quantity

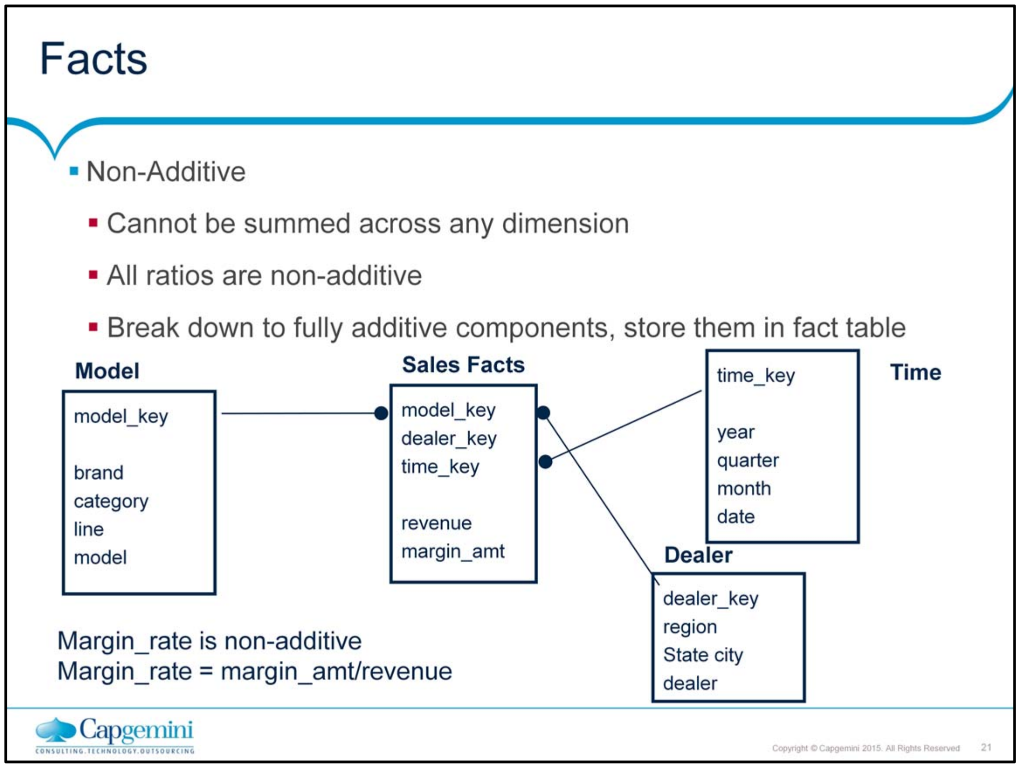
Model
model_key
brand
category
line
model
Sales Facts
model_key
dealer_key
time_key
revenue
quantity
Time
time_key
year
quarter
month
date
Dealer
dealer_key
region
State city dealer



Facts

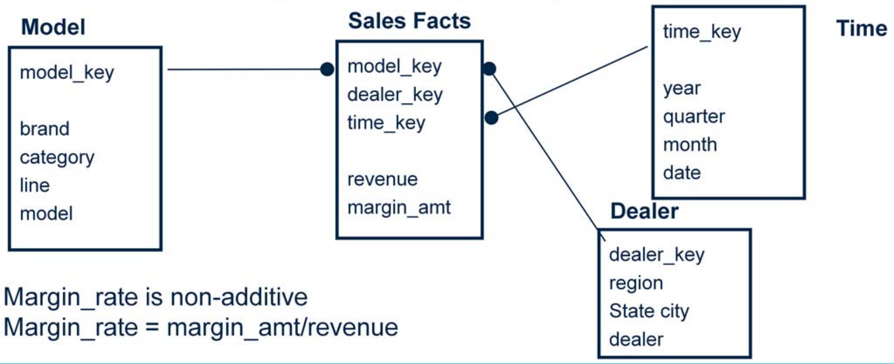
- **Semi-additive**
  - Can be summed across most dimensions but not all
  - Examples: Inventory quantities, account balances, or personnel counts
  - Anything that measures a “level”
  - Must be careful with ad-hoc reporting
  - Often aggregated across the “forbidden dimension” by averaging



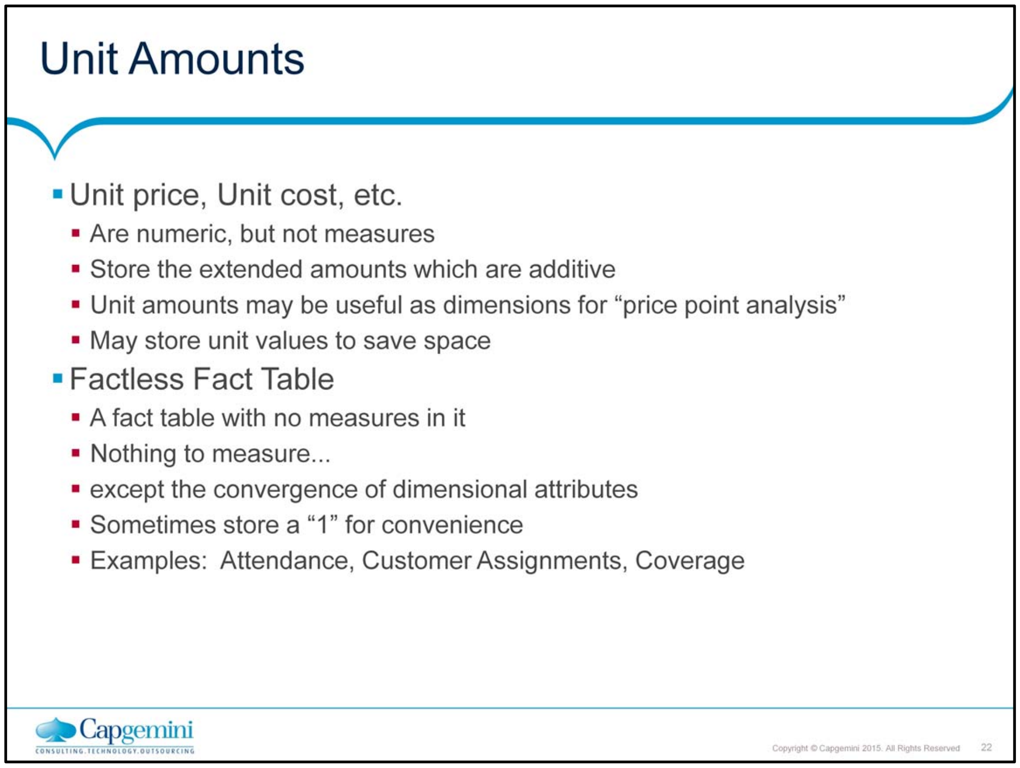


Facts

- Non-Additive
  - Cannot be summed across any dimension
  - All ratios are non-additive
  - Break down to fully additive components, store them in fact table



Margin_rate is non-additive
Margin_rate = margin_amt/revenue

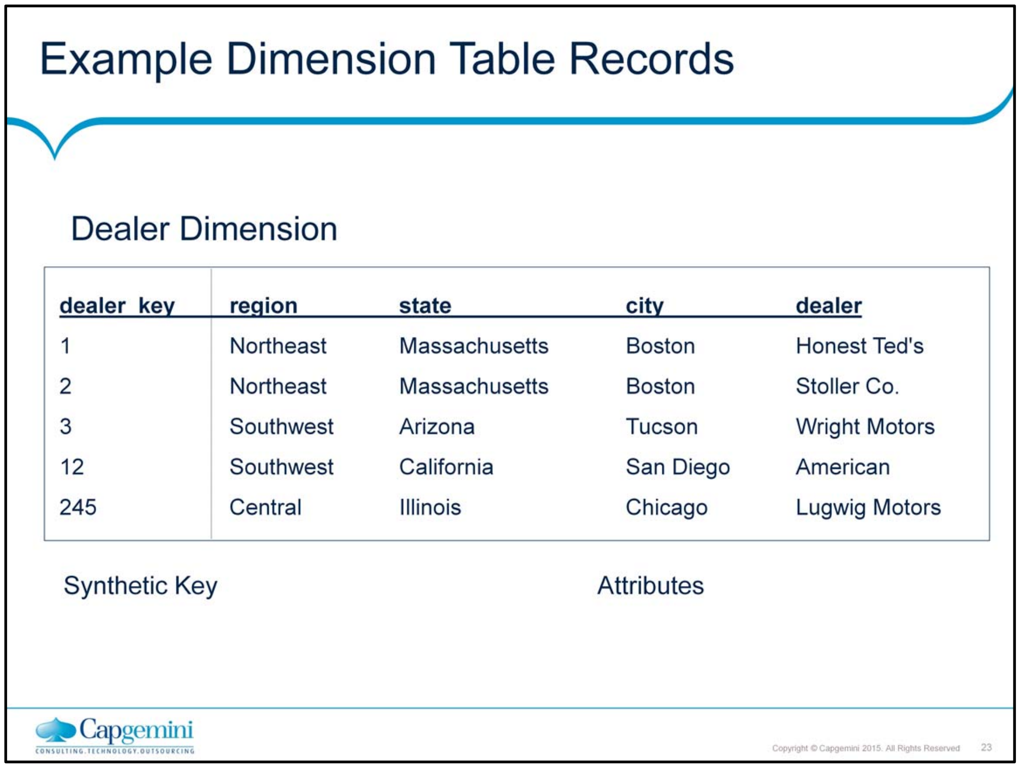


Unit Amounts

- Unit price, Unit cost, etc.
  - Are numeric, but not measures
  - Store the extended amounts which are additive
  - Unit amounts may be useful as dimensions for “price point analysis”
  - May store unit values to save space
- Factless Fact Table
  - A fact table with no measures in it
  - Nothing to measure...
  - except the convergence of dimensional attributes
  - Sometimes store a “1” for convenience
  - Examples: Attendance, Customer Assignments, Coverage

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

22

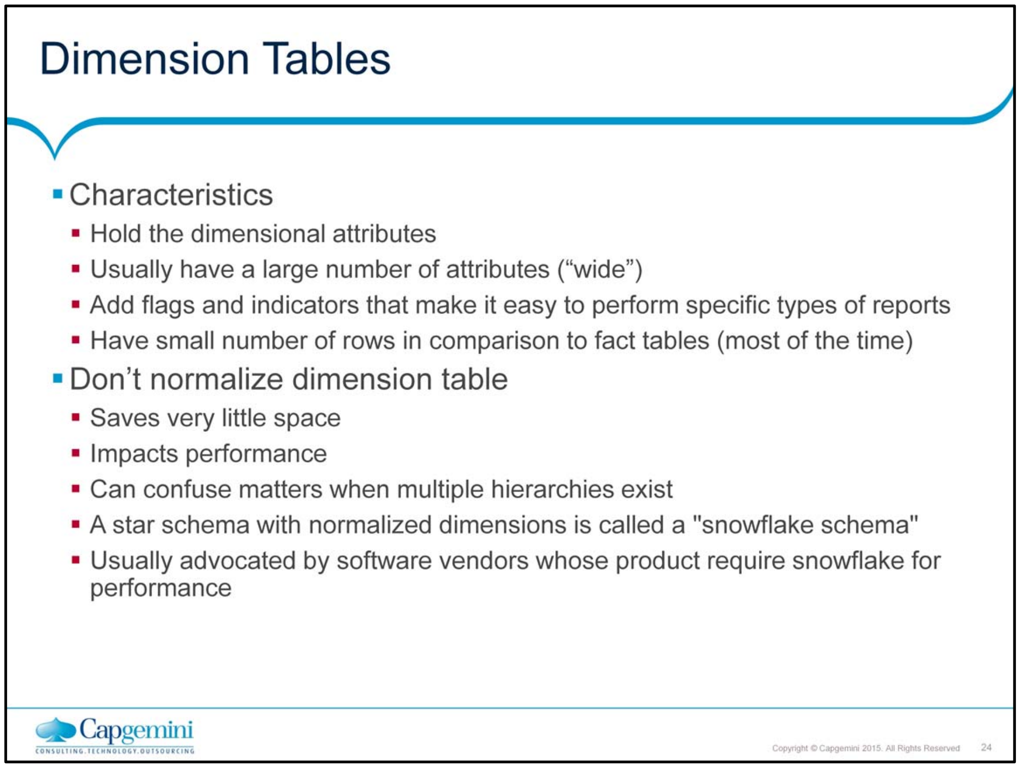


Example Dimension Table Records

Dealer Dimension

dealer_key	region	state	city	dealer
1	Northeast	Massachusetts	Boston	Honest Ted's
2	Northeast	Massachusetts	Boston	Stoller Co.
3	Southwest	Arizona	Tucson	Wright Motors
12	Southwest	California	San Diego	American
245	Central	Illinois	Chicago	Lugwig Motors

Synthetic Key Attributes

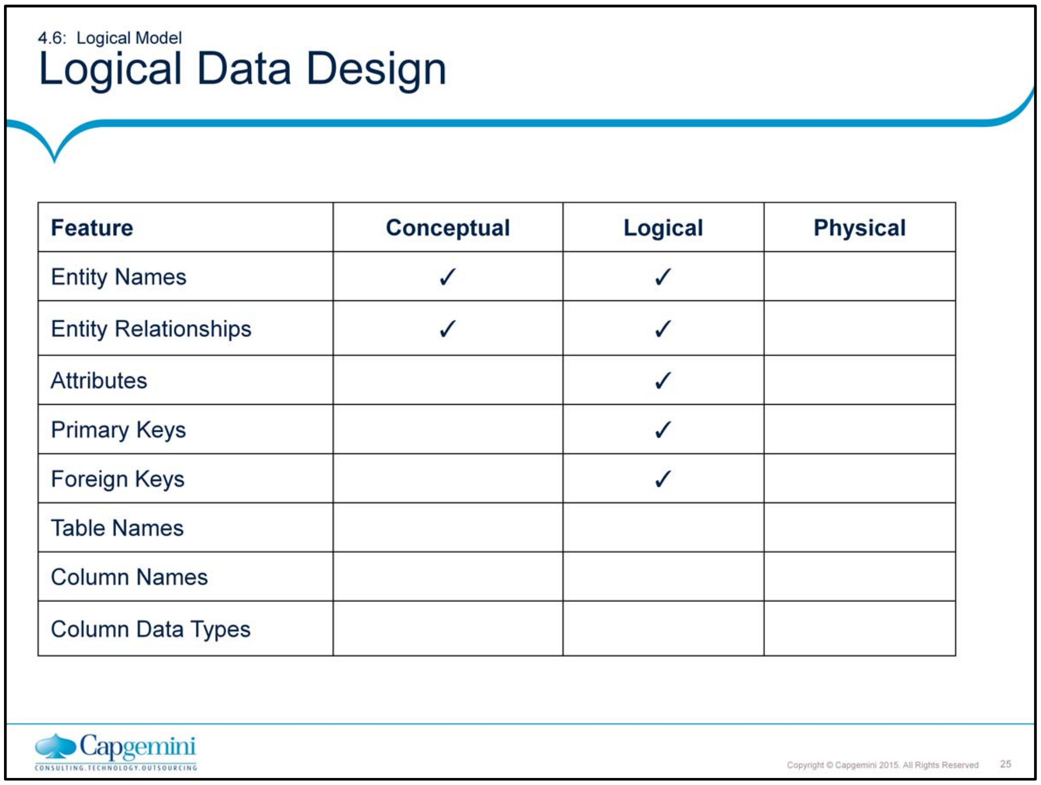


Dimension Tables

- Characteristics
  - Hold the dimensional attributes
  - Usually have a large number of attributes (“wide”)
  - Add flags and indicators that make it easy to perform specific types of reports
  - Have small number of rows in comparison to fact tables (most of the time)
- Don’t normalize dimension table
  - Saves very little space
  - Impacts performance
  - Can confuse matters when multiple hierarchies exist
  - A star schema with normalized dimensions is called a "snowflake schema"
  - Usually advocated by software vendors whose product require snowflake for performance

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

24



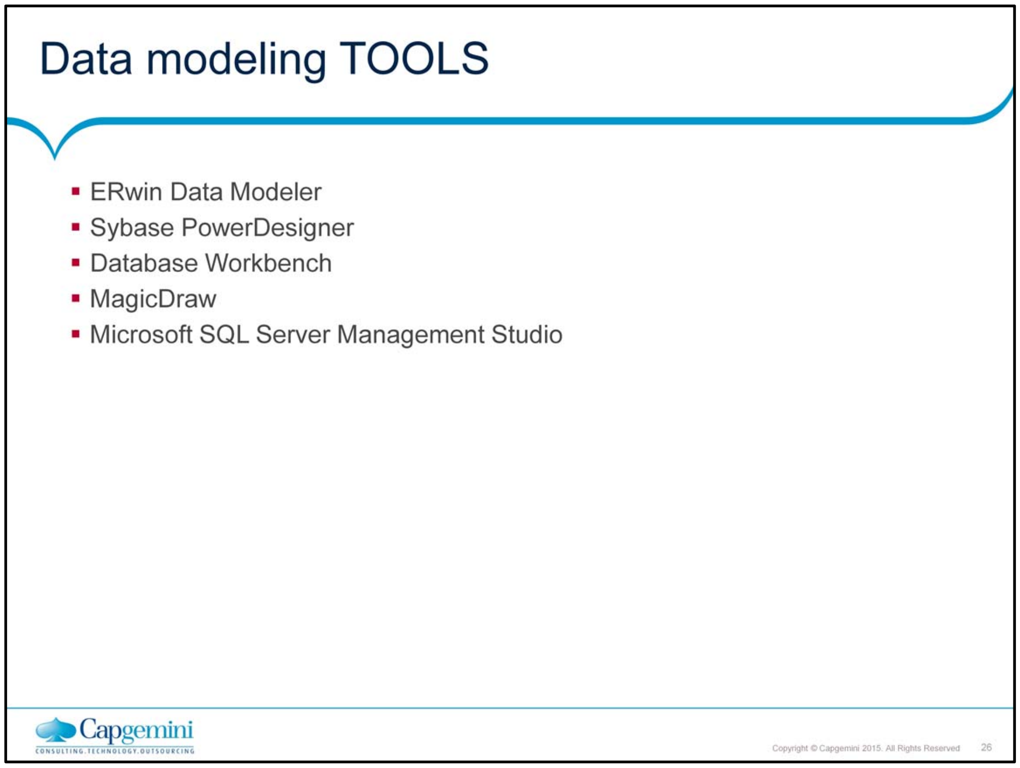
4.6: Logical Model

Logical Data Design

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	
Foreign Keys		✓	
Table Names			
Column Names			
Column Data Types			

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

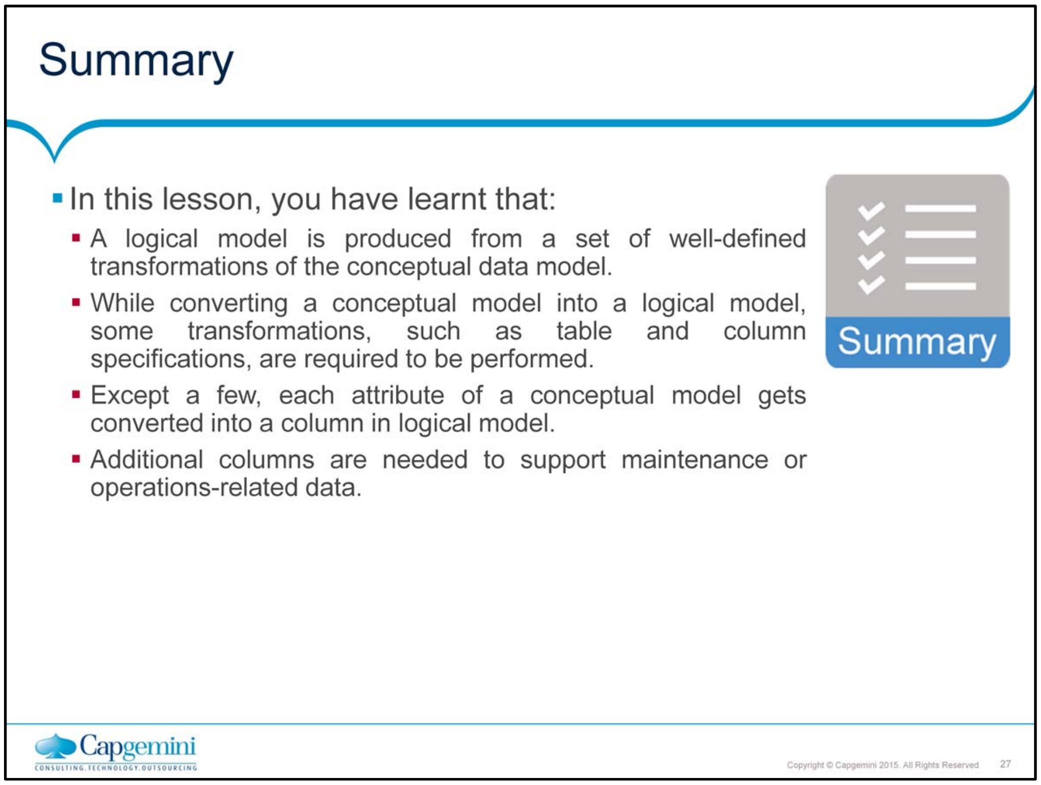
25



Data modeling TOOLS

- ERwin Data Modeler
- Sybase PowerDesigner
- Database Workbench
- MagicDraw
- Microsoft SQL Server Management Studio

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Summary

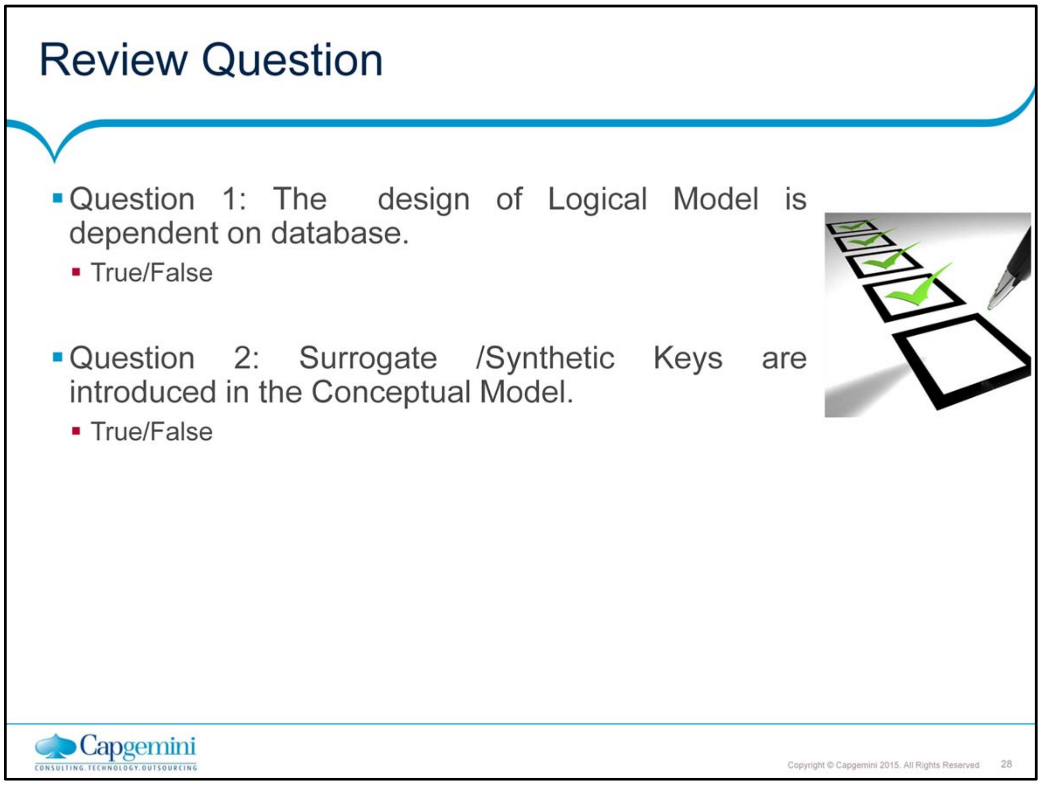
- In this lesson, you have learnt that:
  - A logical model is produced from a set of well-defined transformations of the conceptual data model.
  - While converting a conceptual model into a logical model, some transformations, such as table and column specifications, are required to be performed.
  - Except a few, each attribute of a conceptual model gets converted into a column in logical model.
  - Additional columns are needed to support maintenance or operations-related data.

Summary

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

27

Add the notes here.



Review Question

- Question 1: The design of Logical Model is dependent on database.
  - True/False

- Question 2: Surrogate /Synthetic Keys are introduced in the Conceptual Model.
  - True/False