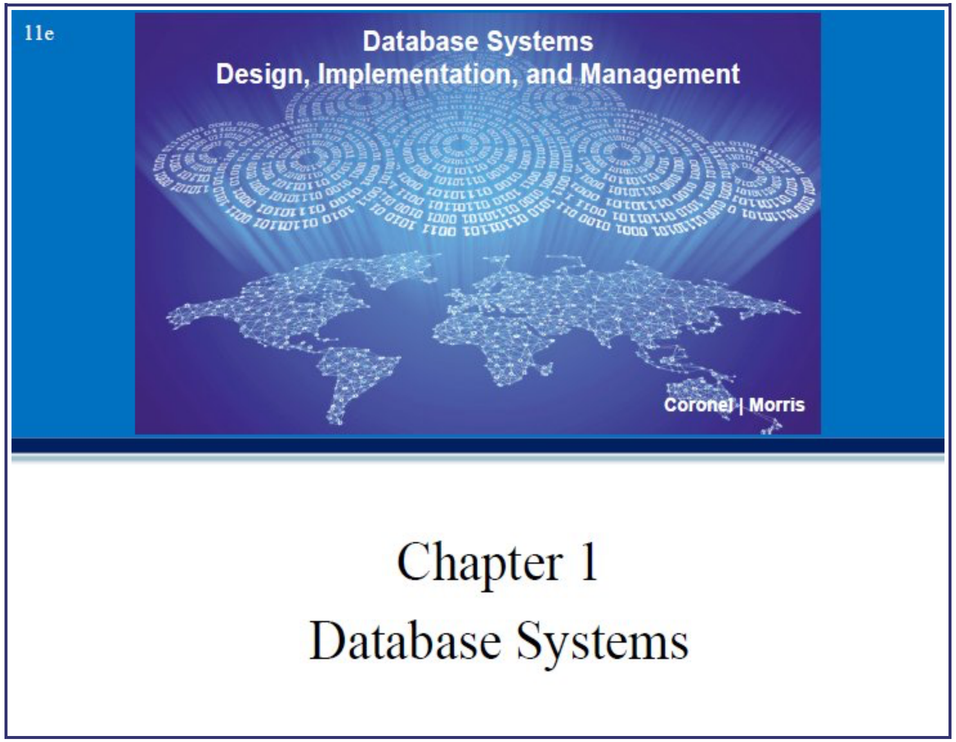
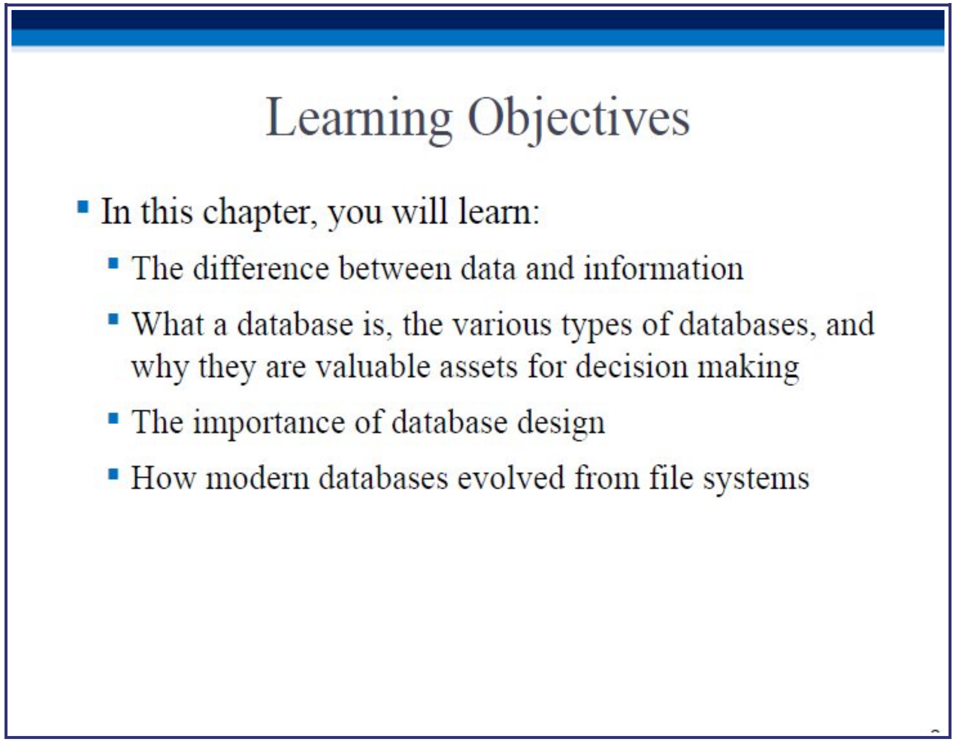
Introduction

Introduction (1) 2023-03-08, 7:18 PM

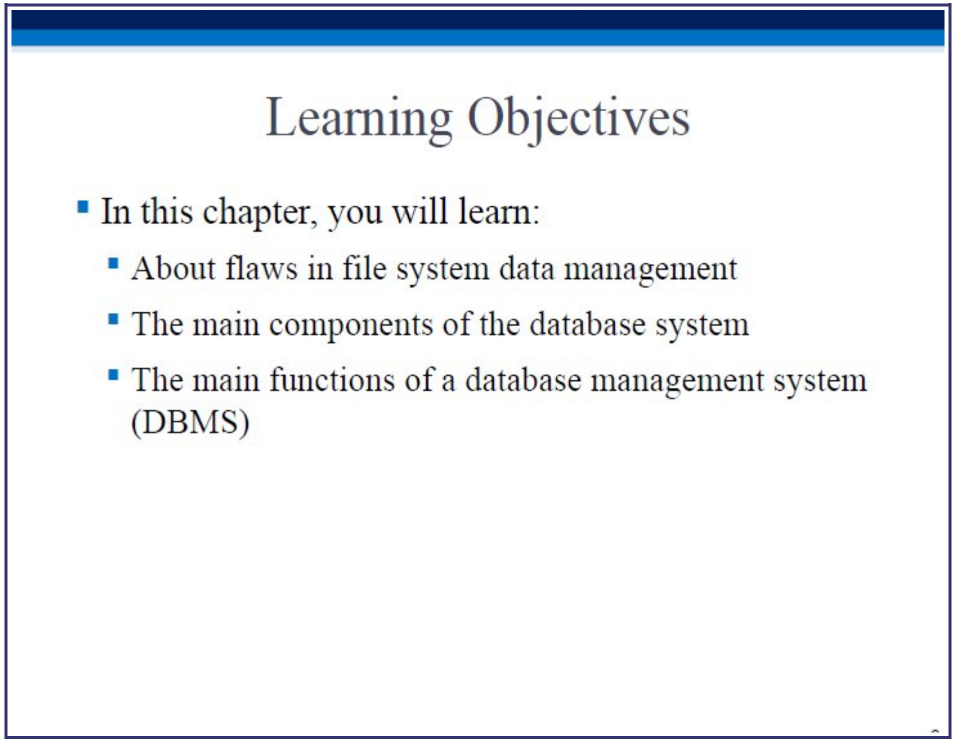
Database systems



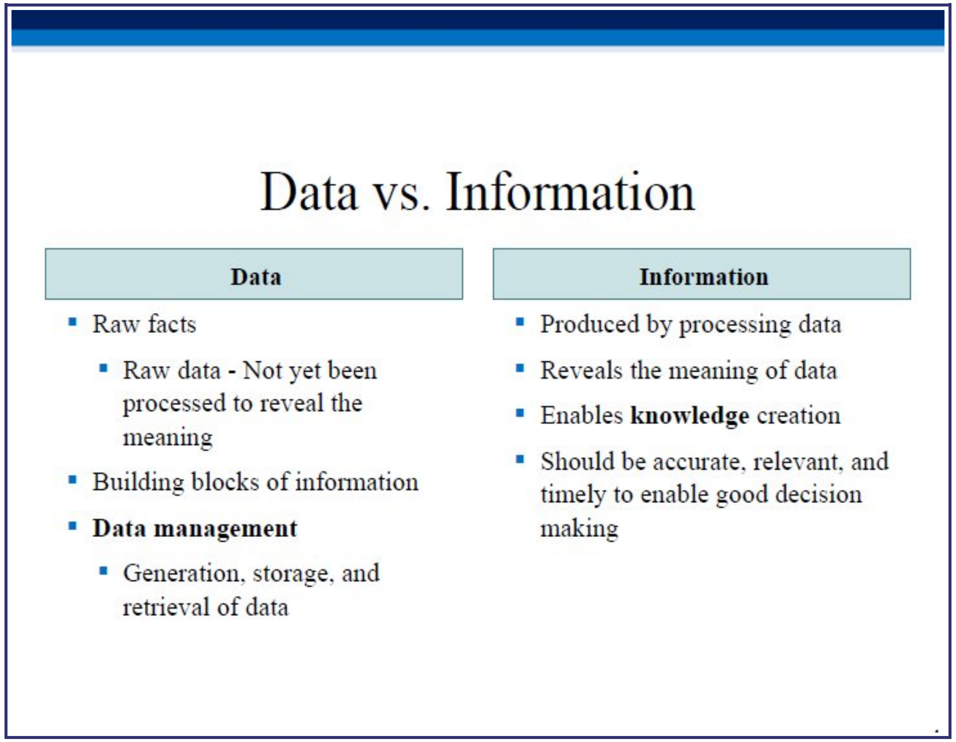
What you will learn:



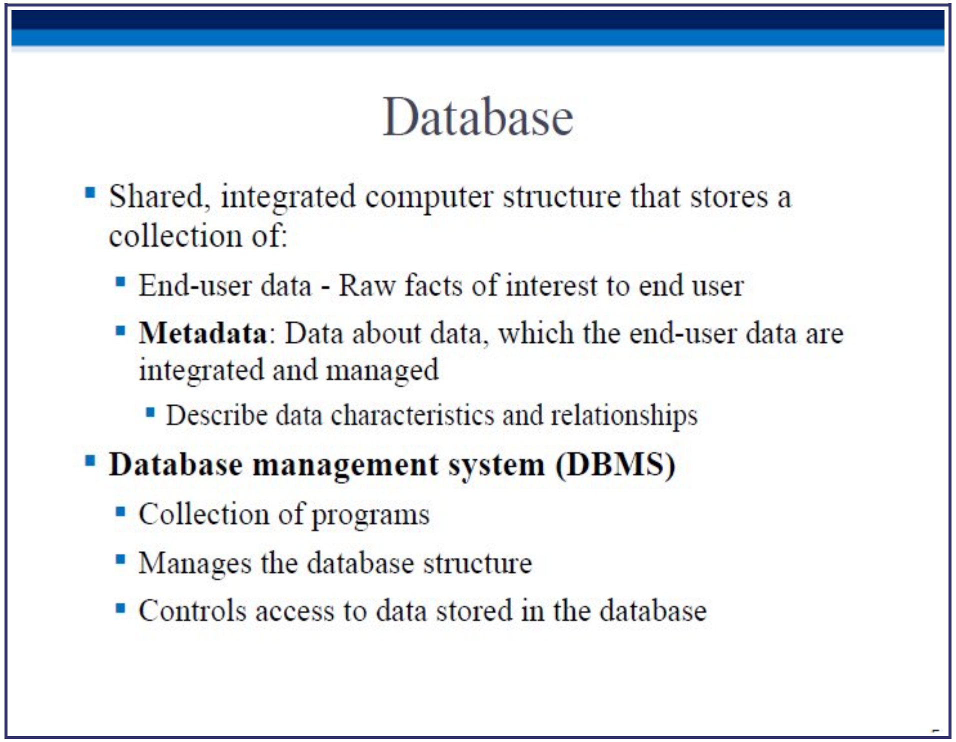
What else you will learn:



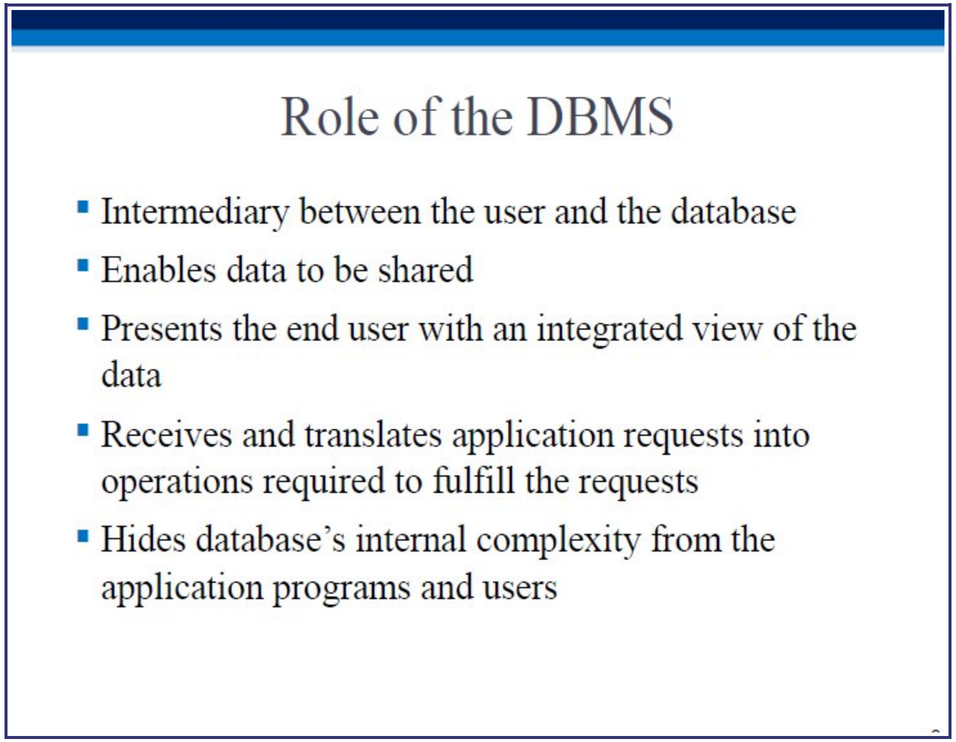
Data != information!



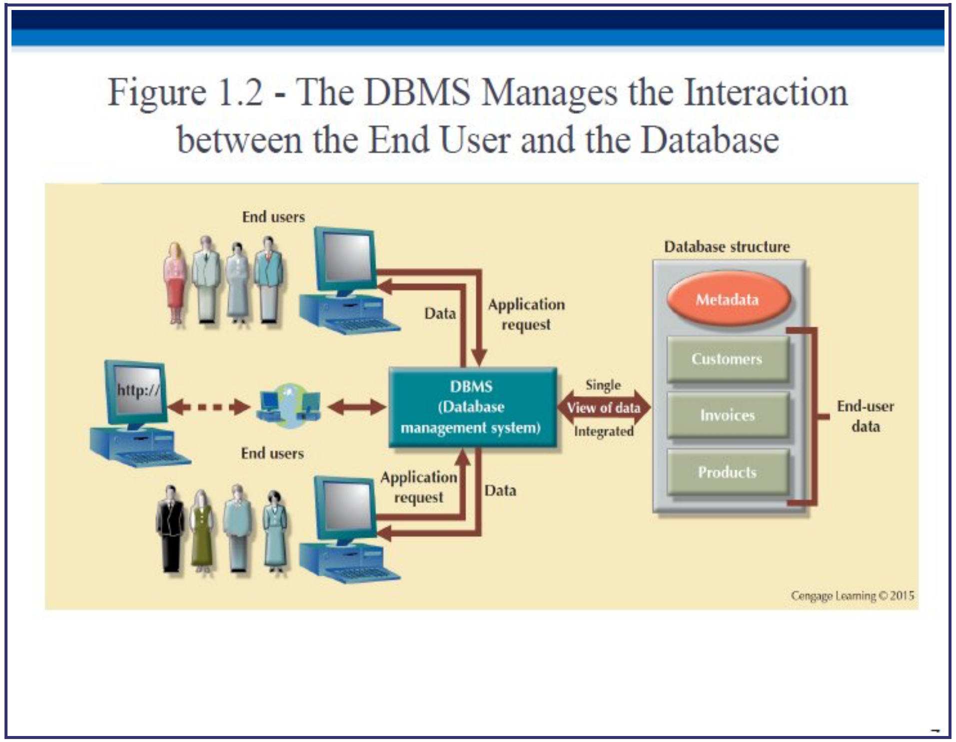
DB, DBMS


Database
Shared, integrated computer structure that stores a collection of:
End-user data - Raw facts of interest to end user
Metadata: Data about data, which the end-user data are integrated and managed
Describe data characteristics and relationships
Database management system (DBMS)
Collection of programs
Manages the database structure
Controls access to data stored in the database

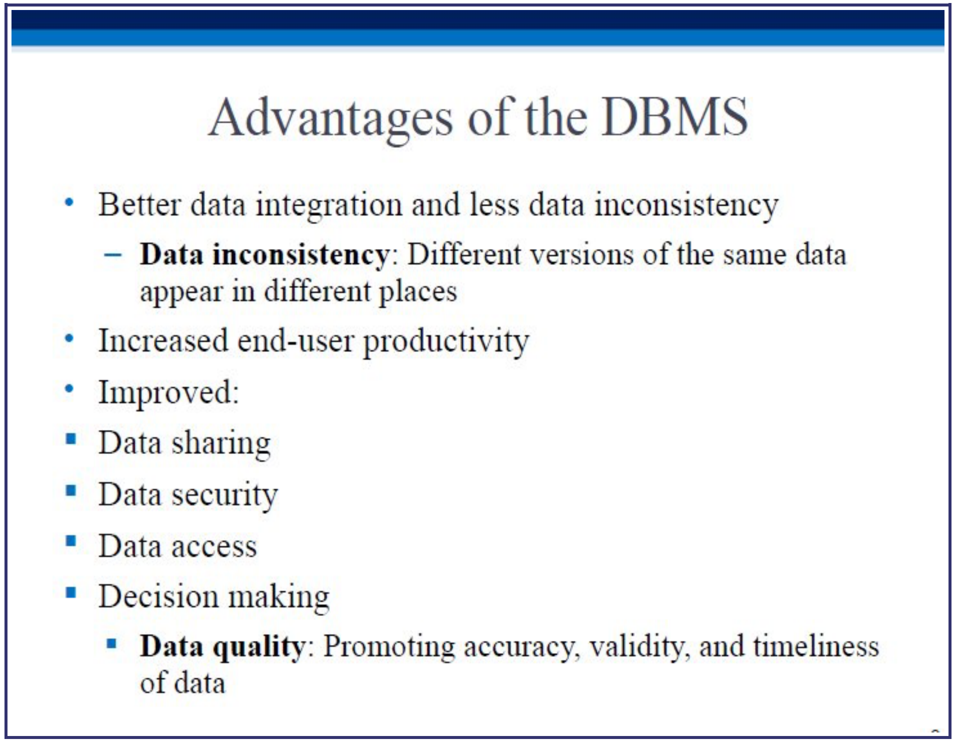
Why DBMS?


Role of the DBMS
Intermediary between the user and the database
Enables data to be shared
Presents the end user with an integrated view of the data
Receives and translates application requests into operations required to fulfill the requests
Hides database's internal complexity from the application programs and users

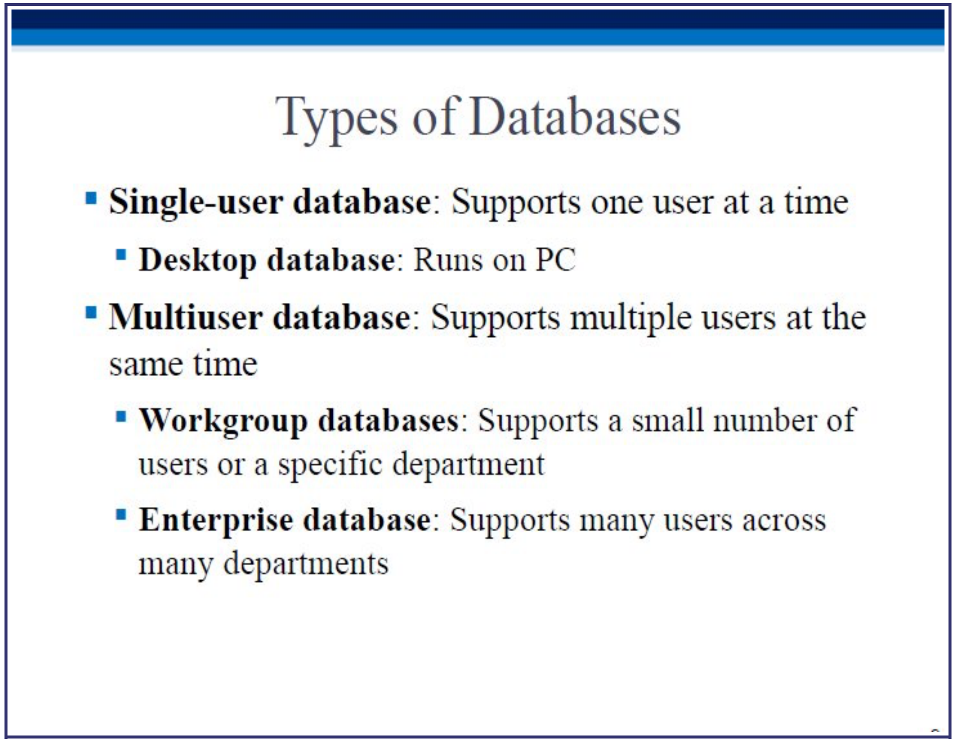
DBMS is a go-between



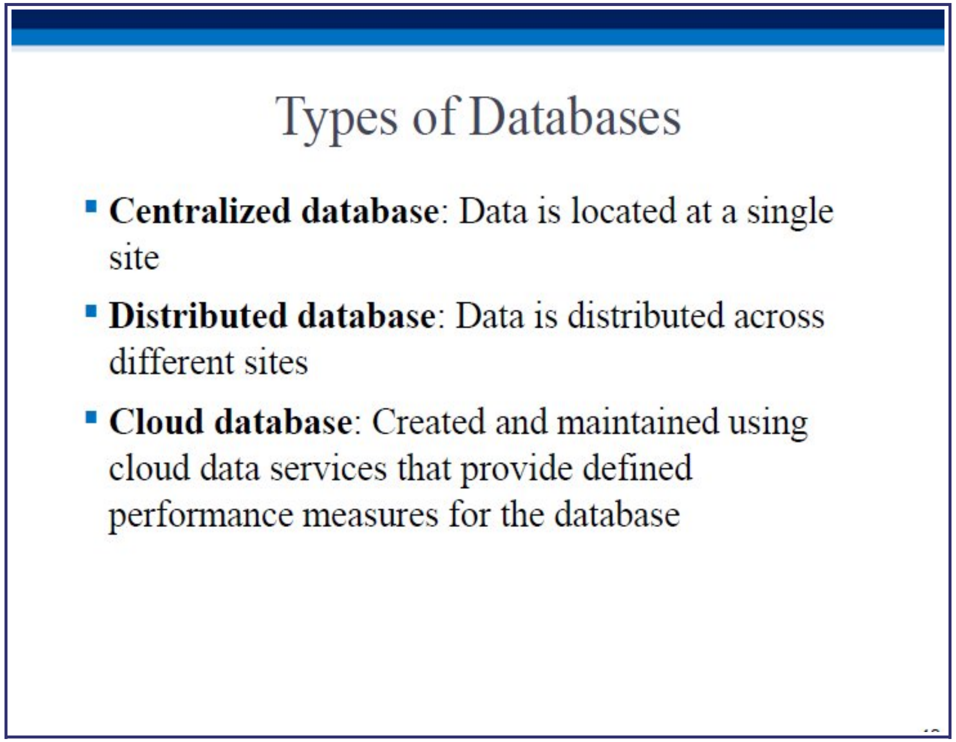
DBMS: advantages



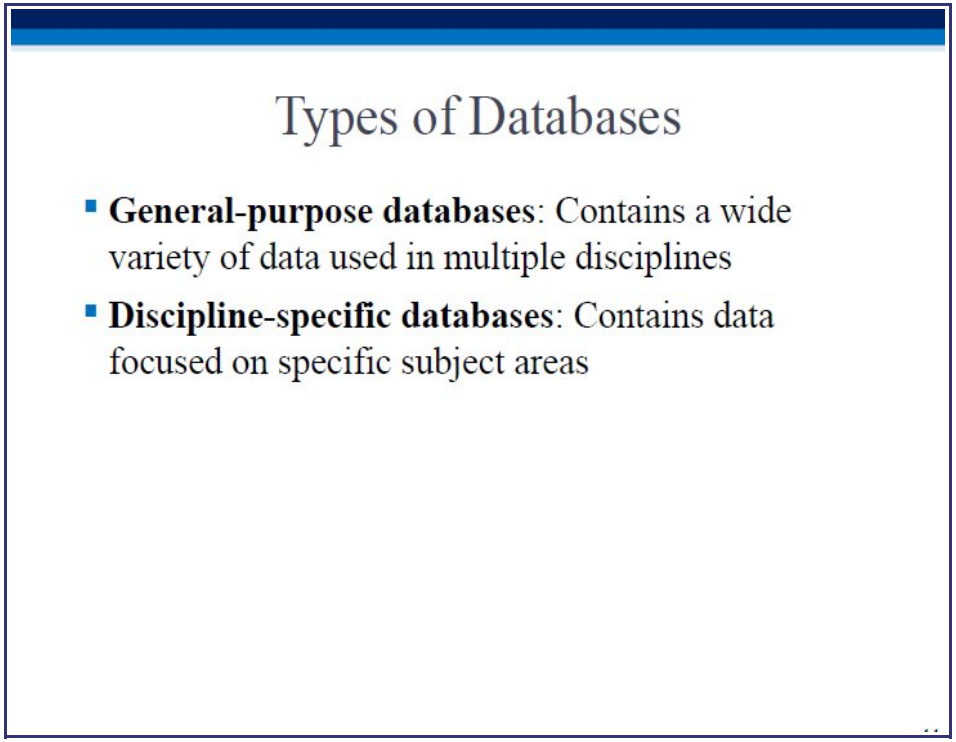
Types of DBs: based on user count



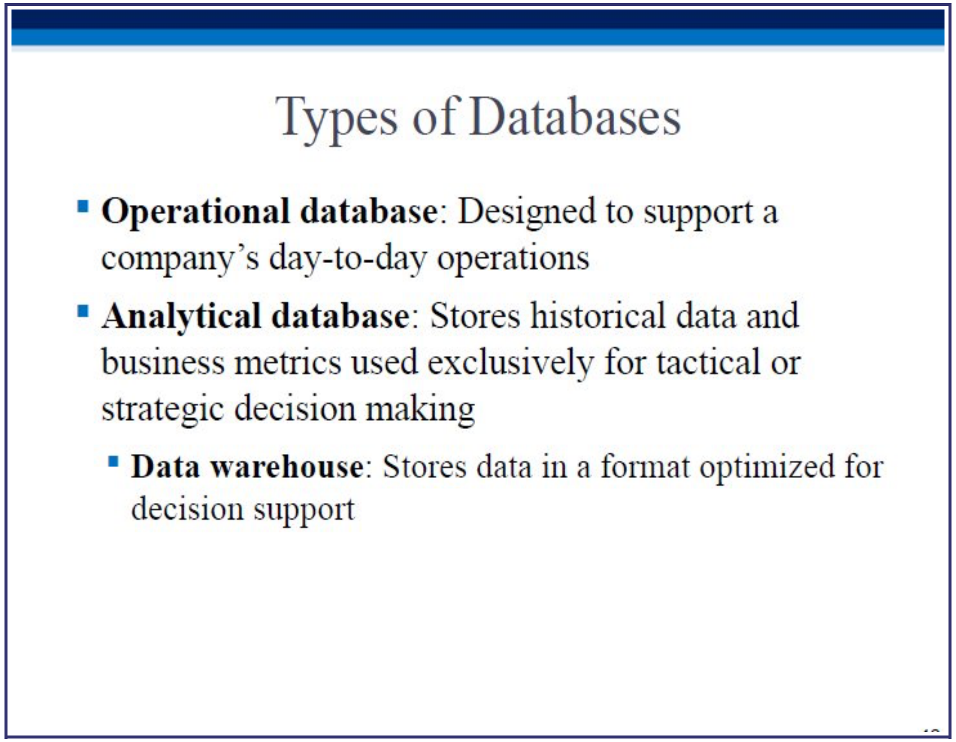
Types of DBs: based on location



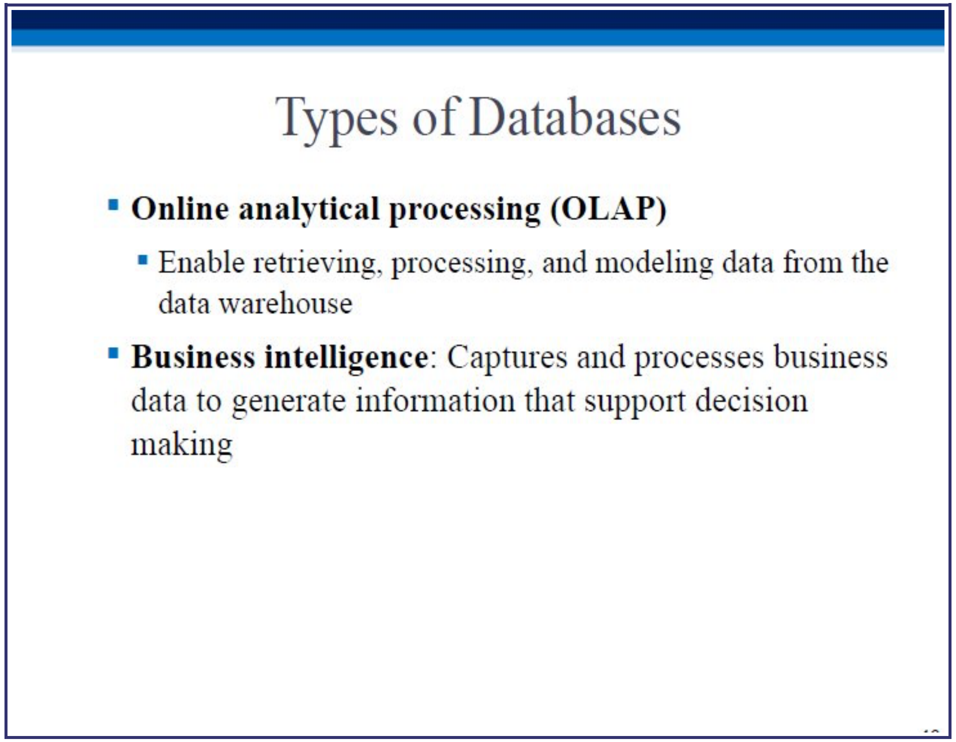
Types of DBs: based on content



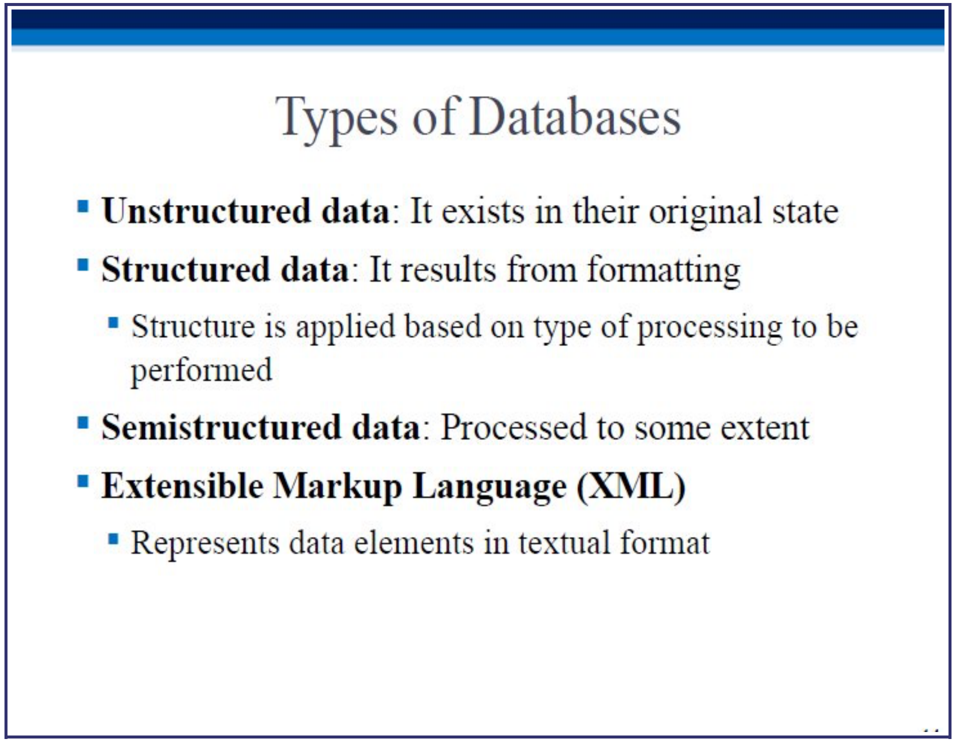
Types of DBs: based on data currency



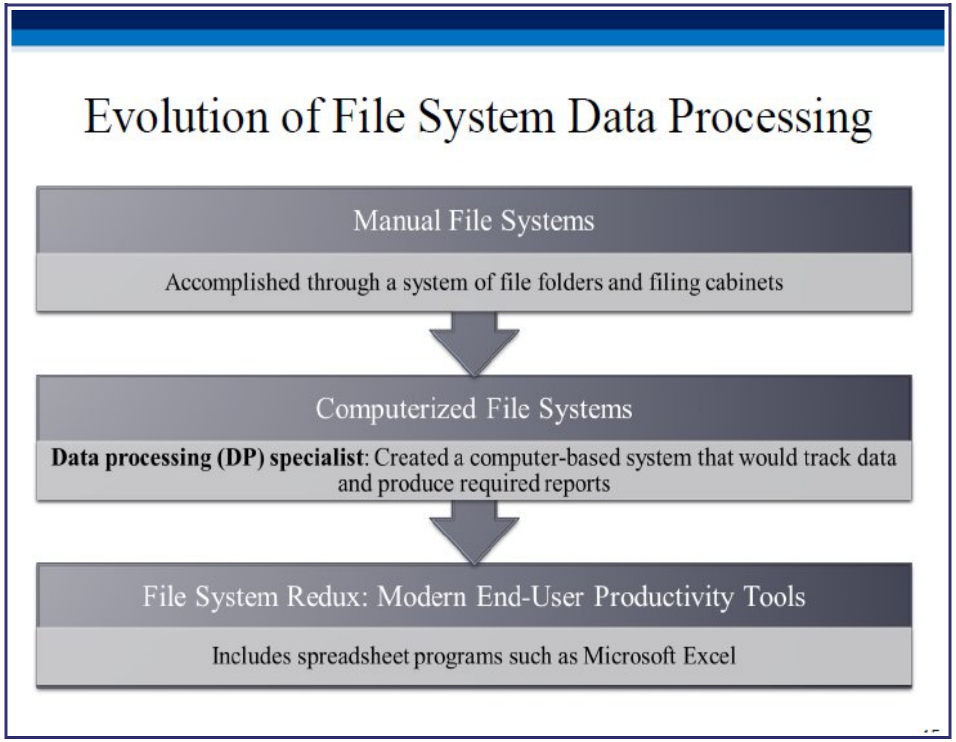
Types of DBs [cont'd]



Types of DBs: based on the structure of contained data



Early DBs: file systems



File systems

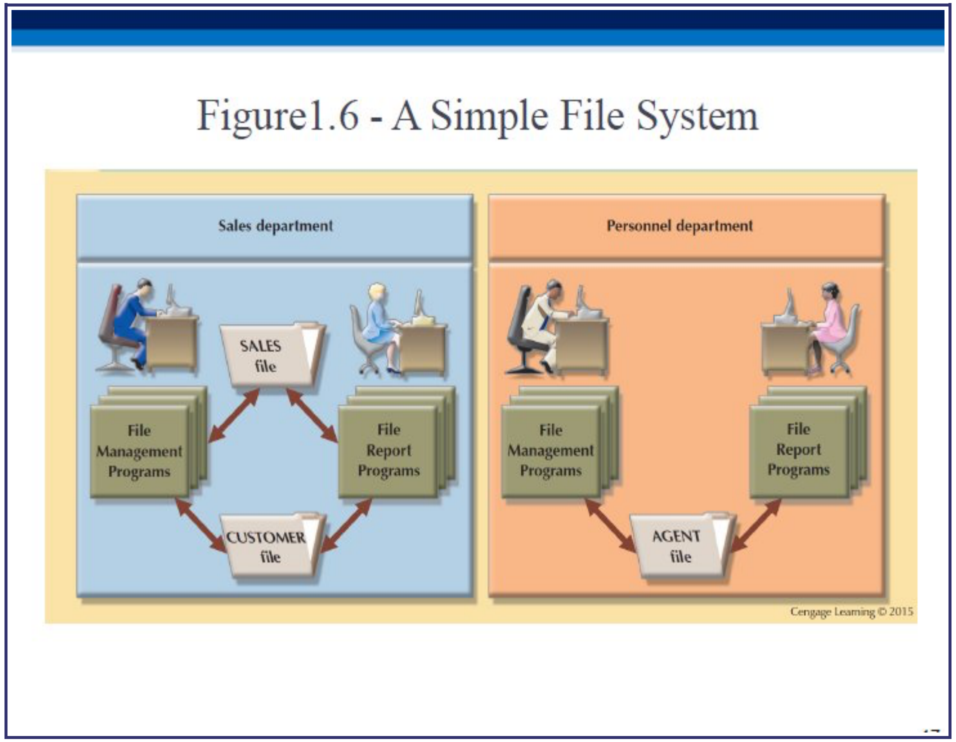
Table 1.2 - Basic File Terminology

TERM	DEFINITION
Data	Raw facts, such as a telephone number, a birth date, a customer name, and a year-to-date (YTD) sales value. Data have little meaning unless they have been organized in some logical manner.
Field	A character or group of characters (alphabetic or numeric) that has a specific meaning. A field is used to define and store data.
Record	A logically connected set of one or more fields that describes a person, place, or thing. For example, the fields that constitute a record for a customer might consist of the customer's name, address, phone number, date of birth, credit limit, and unpaid balance.
File	A collection of related records. For example, a file might contain data about the students currently enrolled at Gigantic University.

Introduction (1) 2023-03-08, 7:18 PM

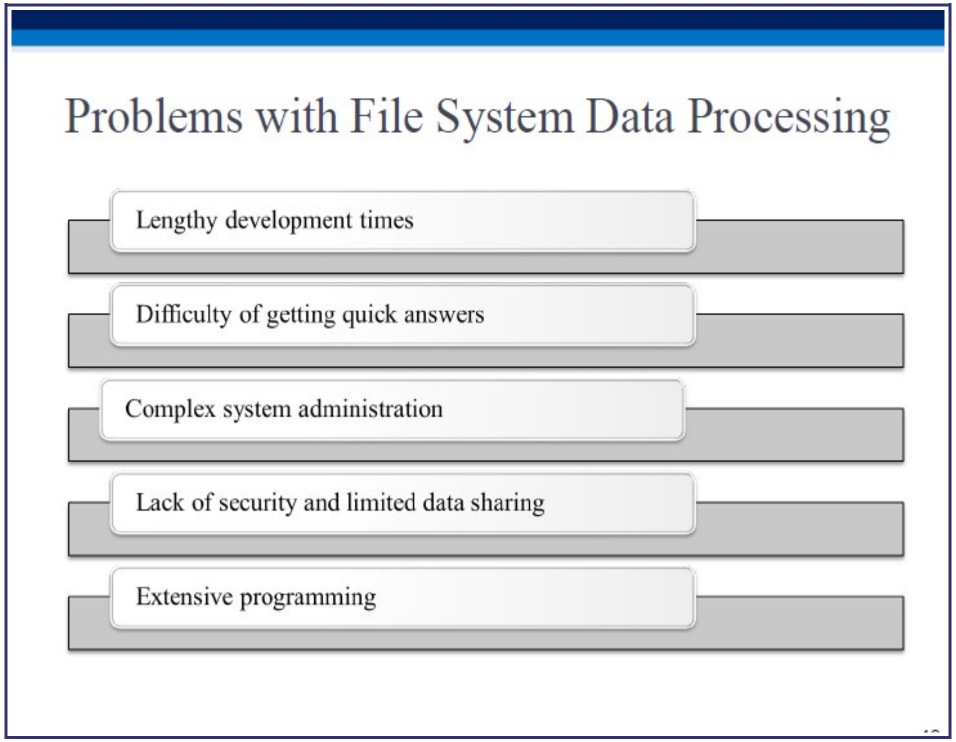
Introduction (1) 2023-03-08, 7:18 PM

File system

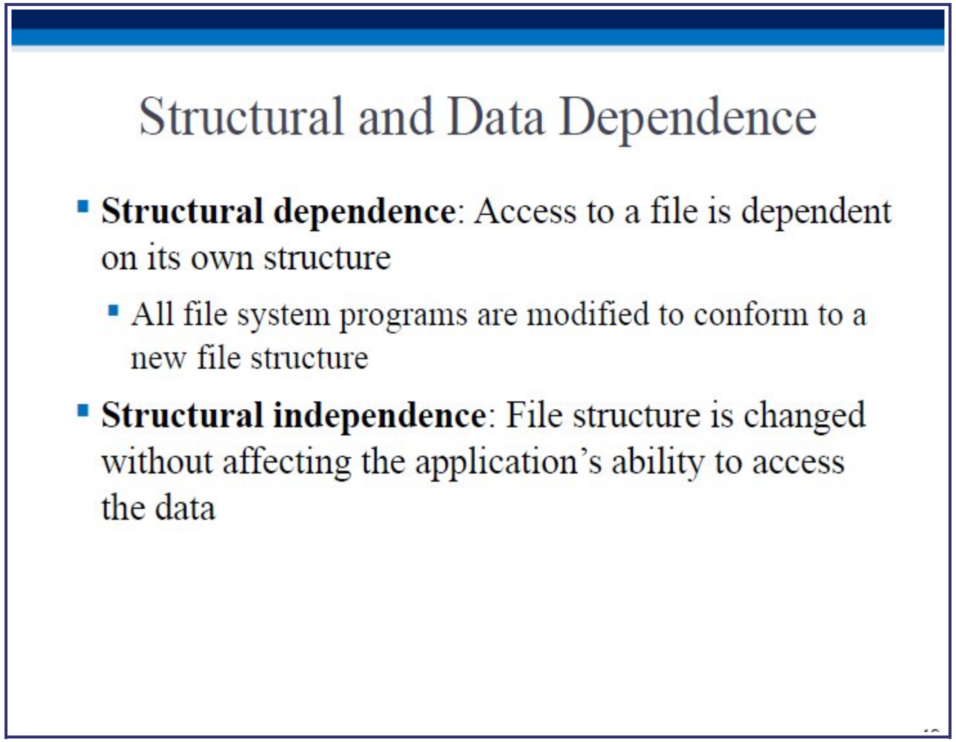


Introduction (1) 2023-03-08, 7:18 PM

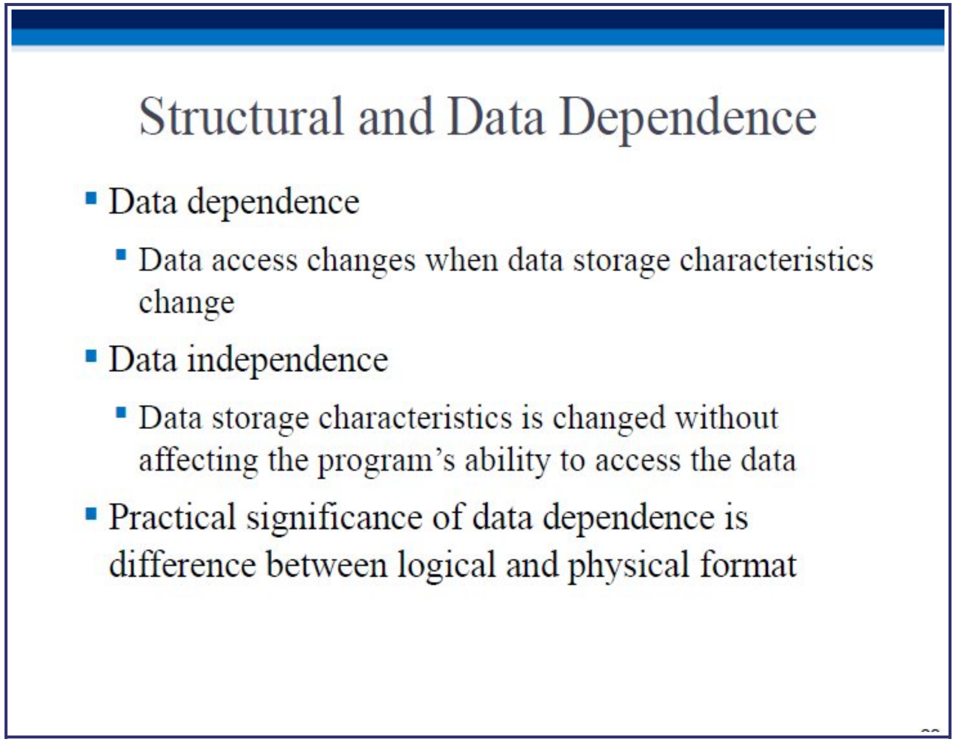
File systems: problems



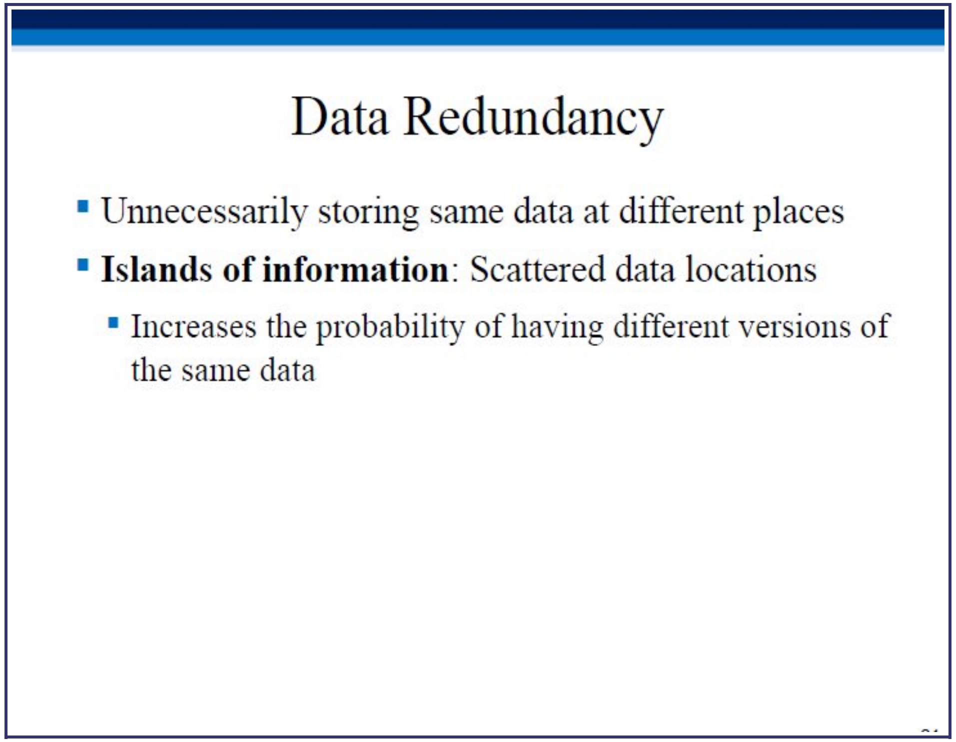
'Structural' dependence (not a good thing!)



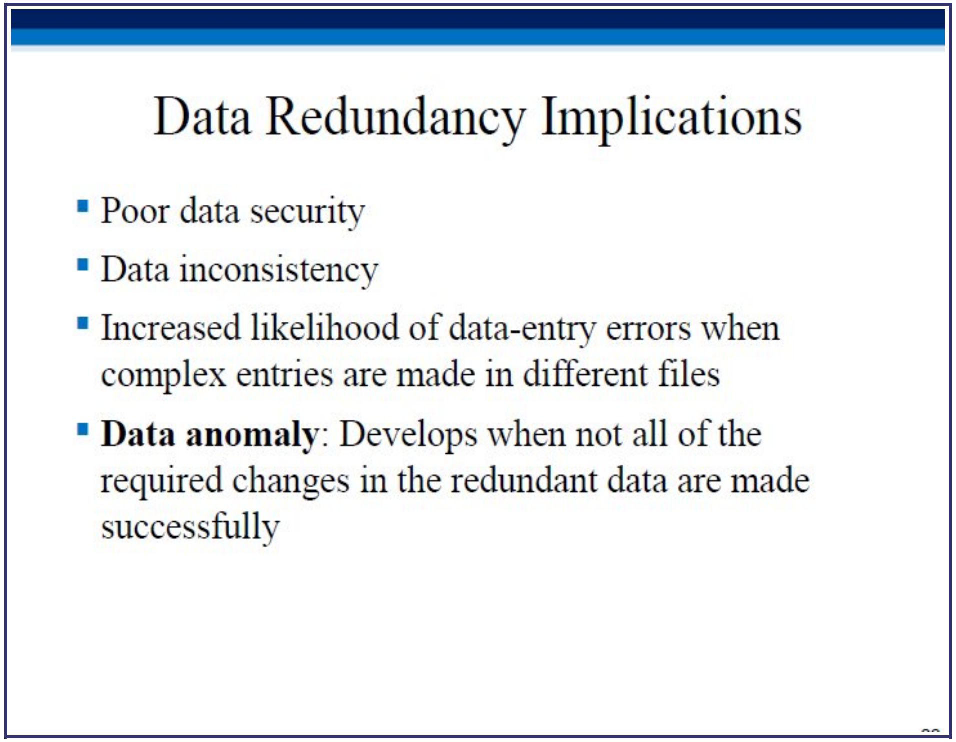
Structural dependence [cont'd]



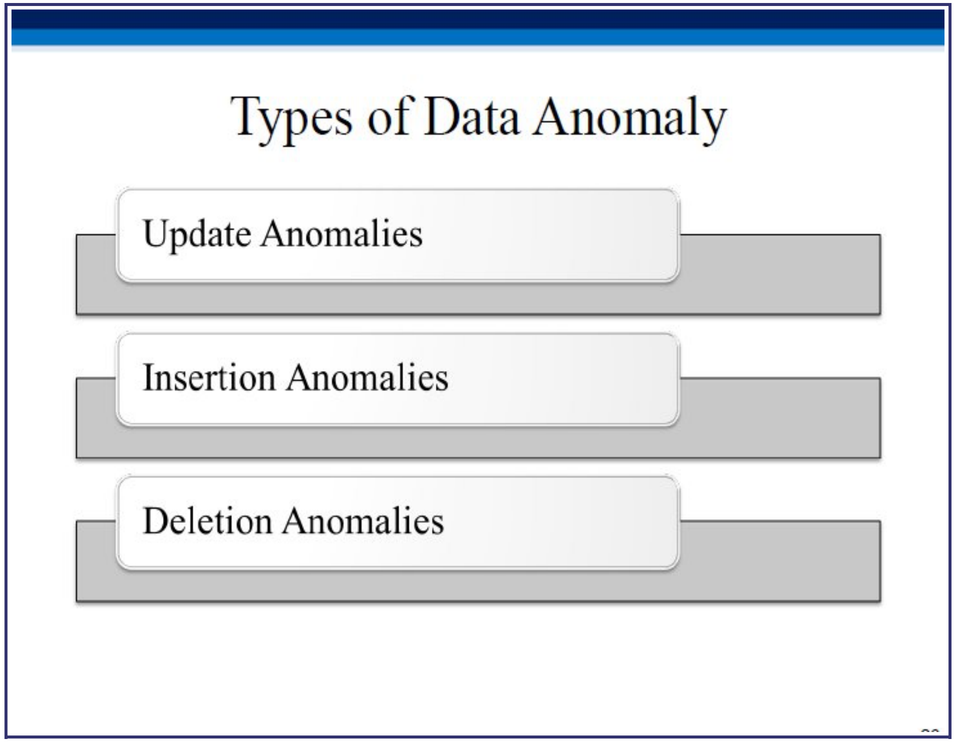
Redundancy of data (again, not a good thing!)



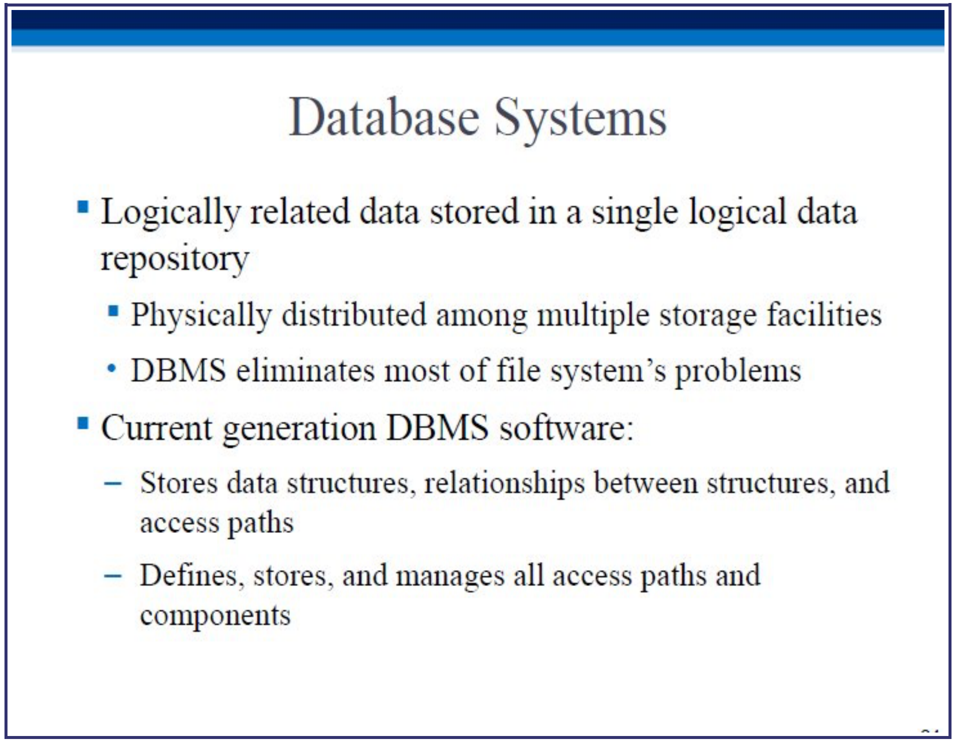
Why is redundancy not a good thing?



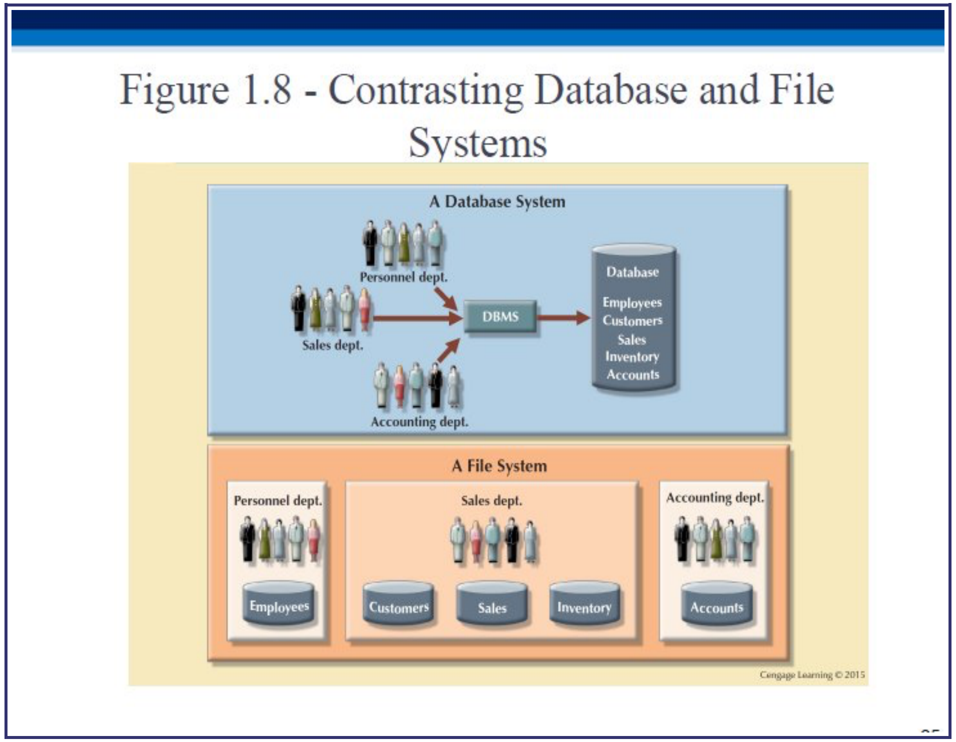
The three types of data anomalies



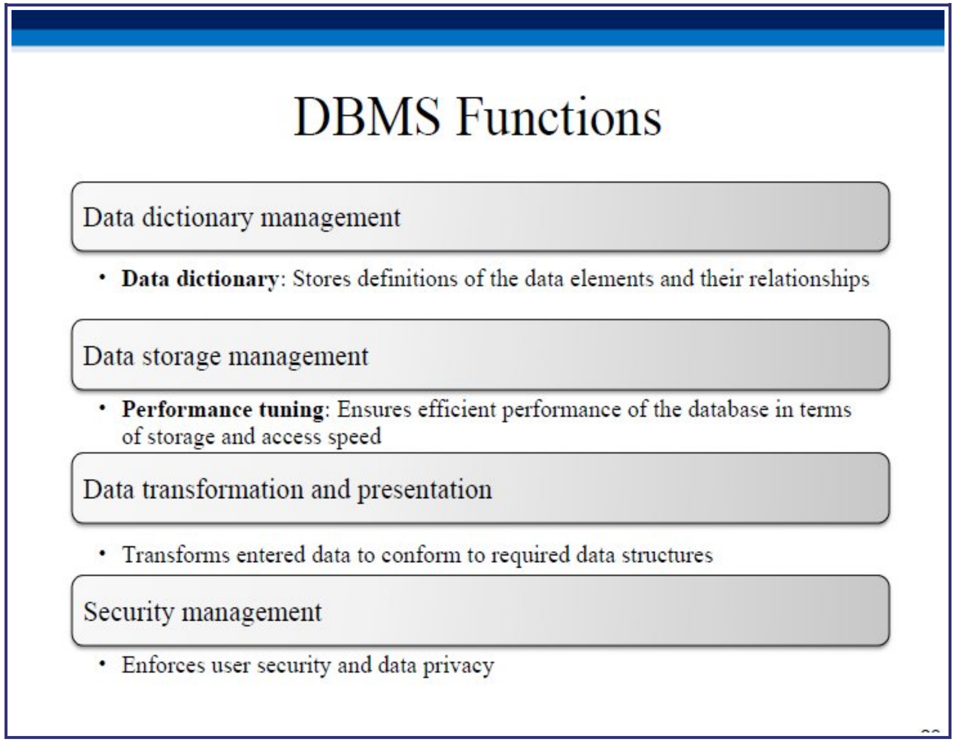
DB systems


Database Systems
Logically related data stored in a single logical data repository
Physically distributed among multiple storage facilities
DBMS eliminates most of file system's problems
Current generation DBMS software:
Stores data structures, relationships between structures, and access paths
Defines, stores, and manages all access paths and components

DB vs file system

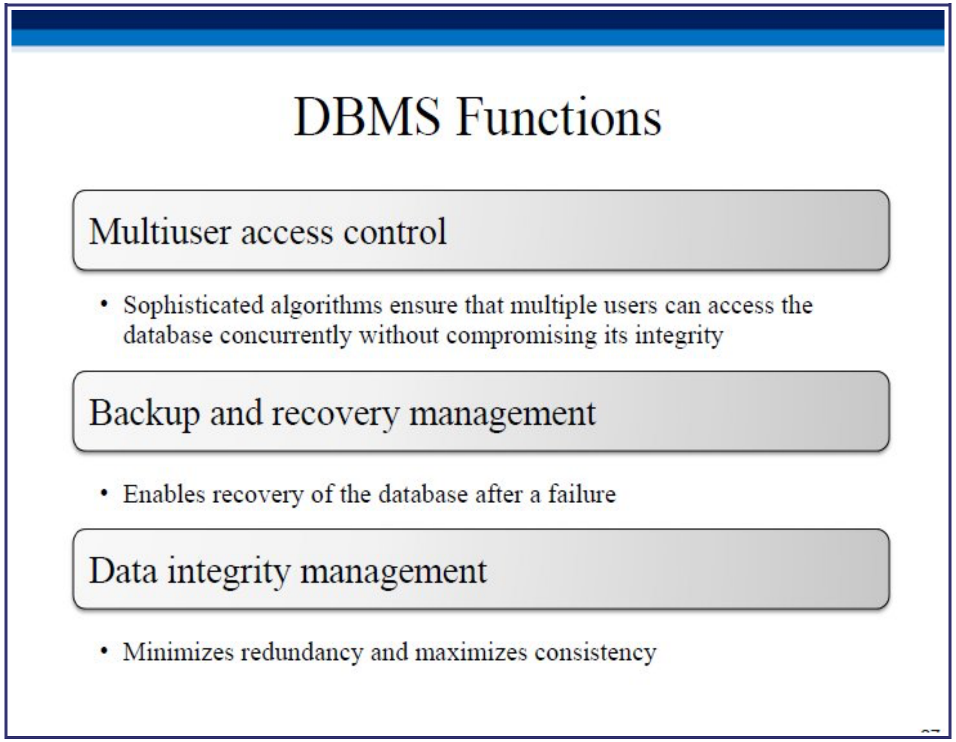


DBMS

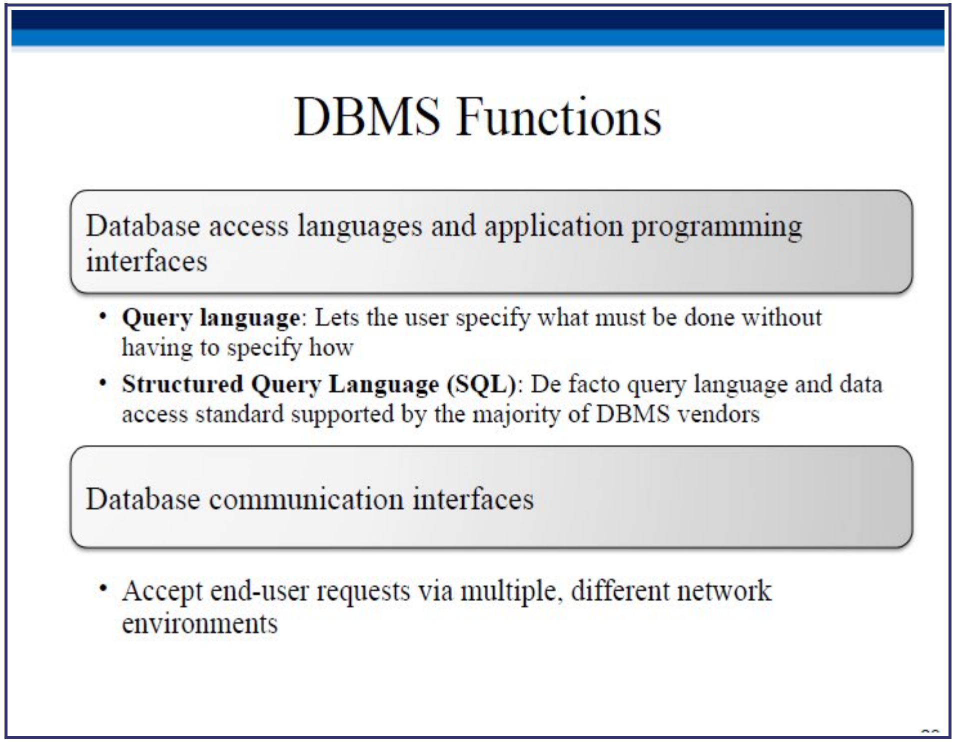


Introduction (1) 2023-03-08, 7:18 PM

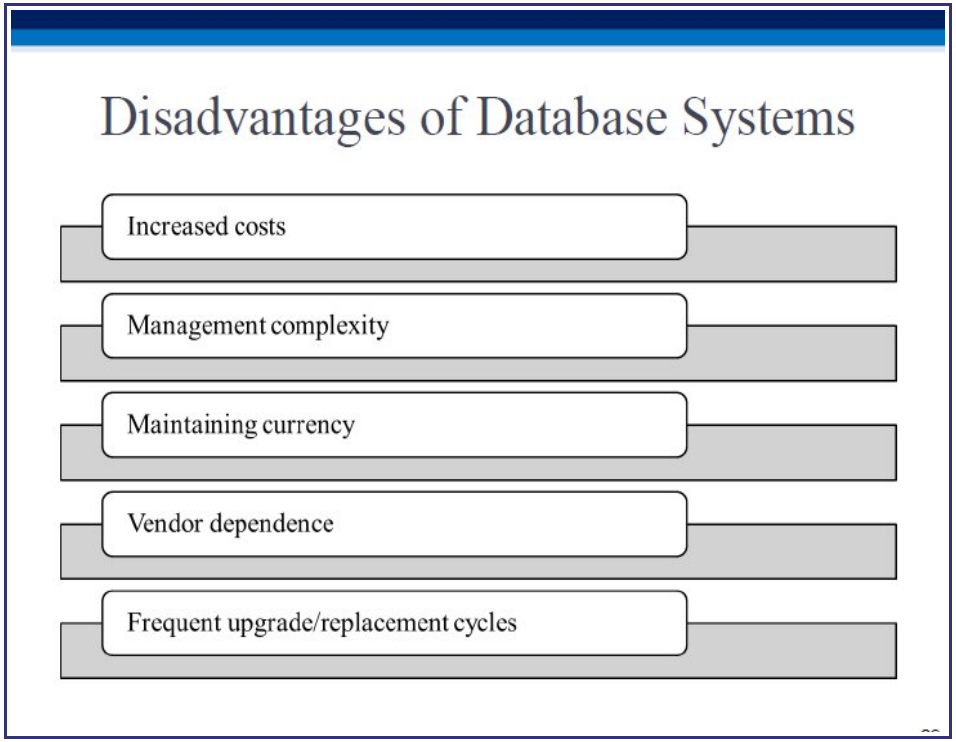
DBMS [cont'd]


DBMS Functions
Multiuser access control
Sophisticated algorithms ensure that multiple users can access the database concurrently without compromising its integrity
Backup and recovery management
Enables recovery of the database after a failure
Data integrity management
Minimizes redundancy and maximizes consistency

DBMS [cont'd]

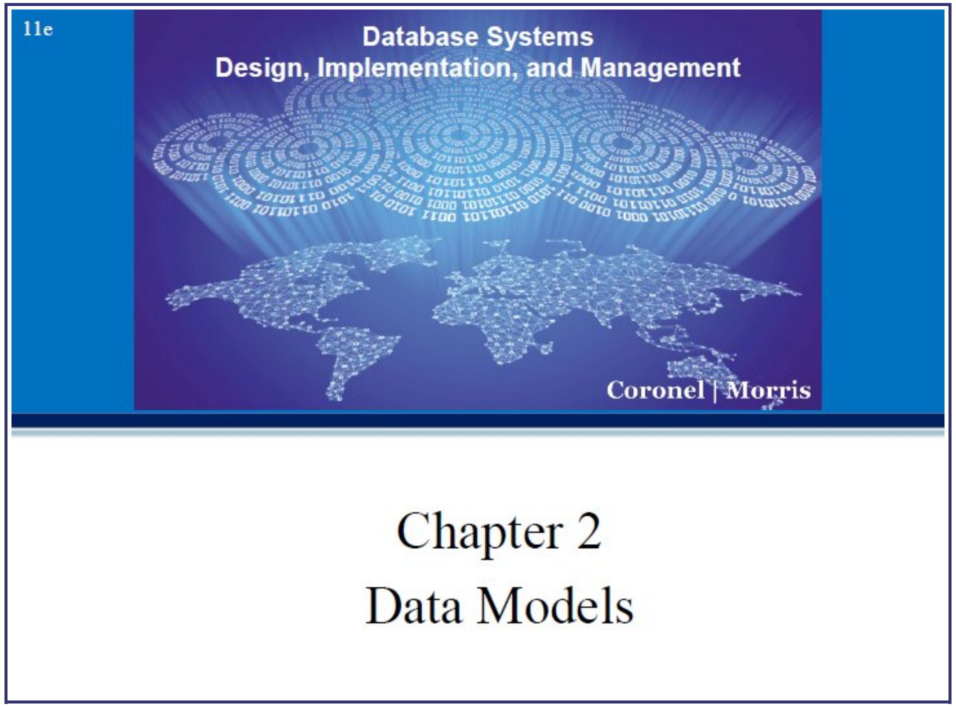

DBMS Functions
Database access languages and application programming interfaces
• **Query language**: Lets the user specify what must be done without having to specify how
• **Structured Query Language (SQL)**: De facto query language and data access standard supported by the majority of DBMS vendors
Database communication interfaces
• Accept end-user requests via multiple, different network environments

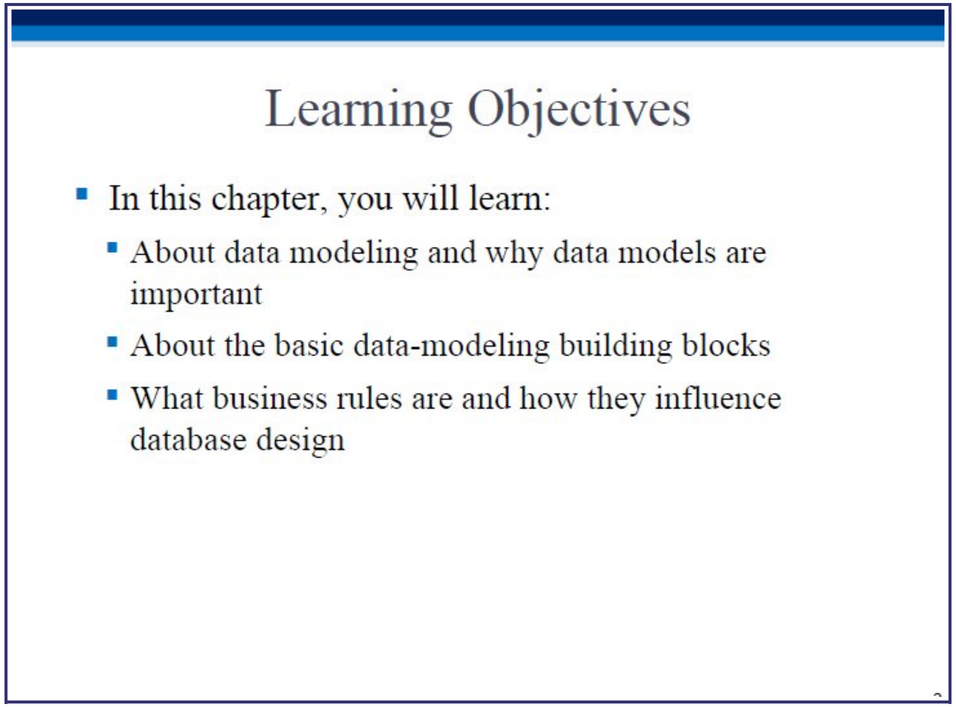
How DBs could be "bad"



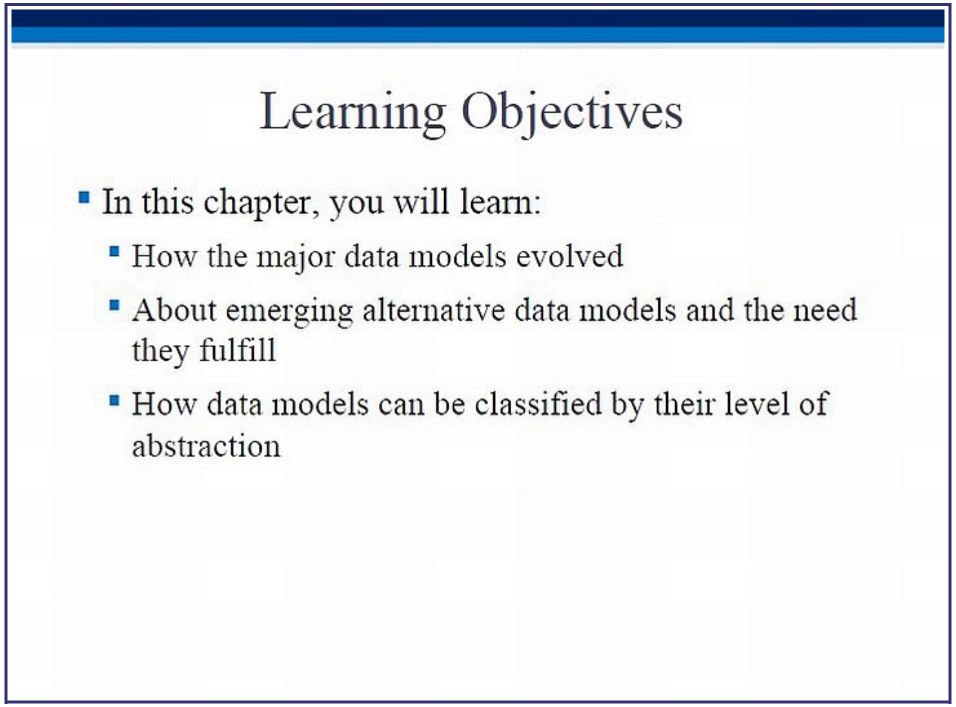
Data Modeling

Data Modeling (1) 2023-03-08, 7:31 PM

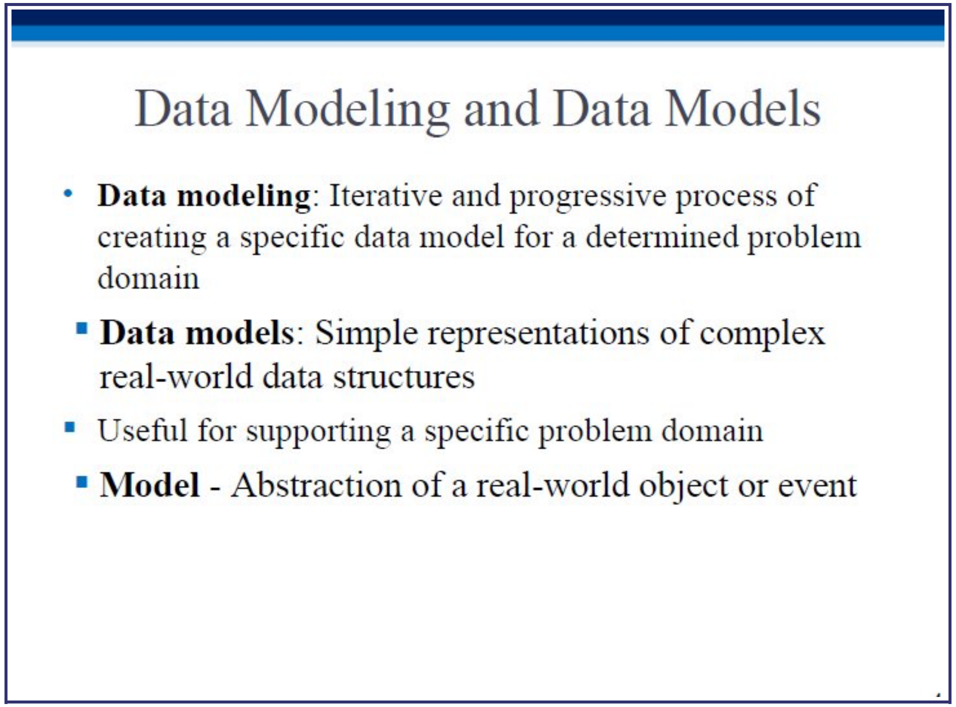
- 
11e
Database Systems
Design, Implementation, and Management
Coronel | Morris
Chapter 2
Data Models

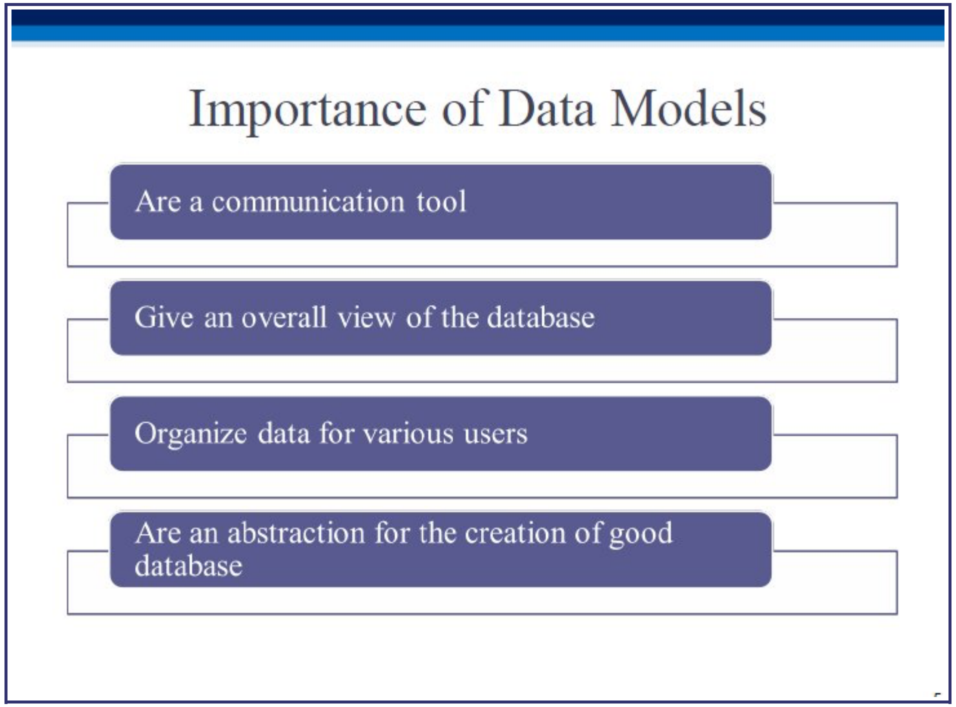
- 
Learning Objectives
In this chapter, you will learn:
About data modeling and why data models are important
About the basic data-modeling building blocks
What business rules are and how they influence database design

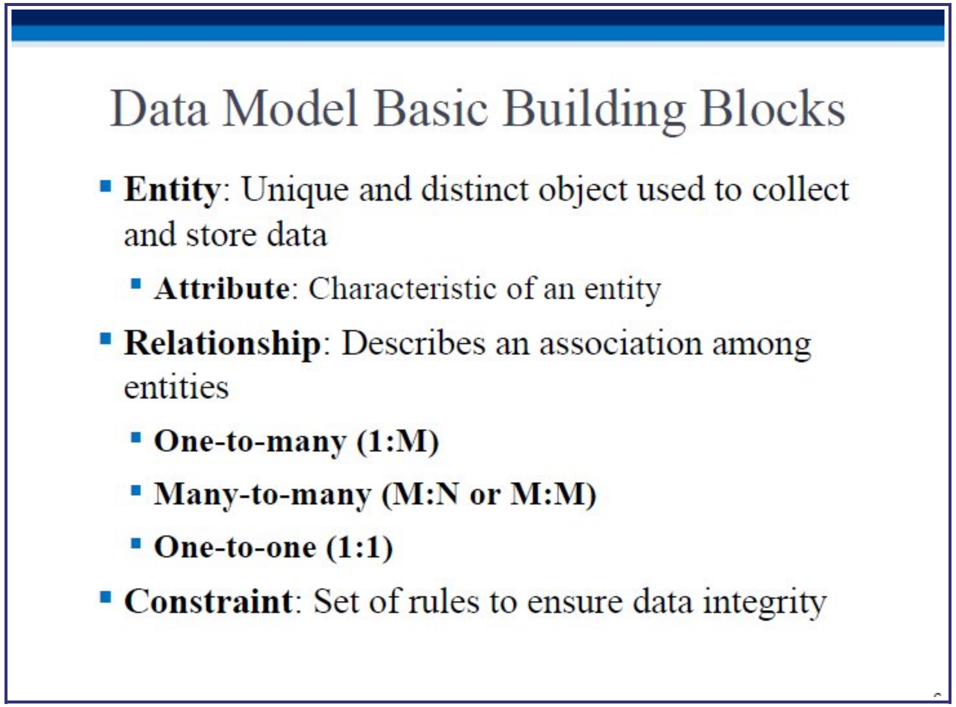
•

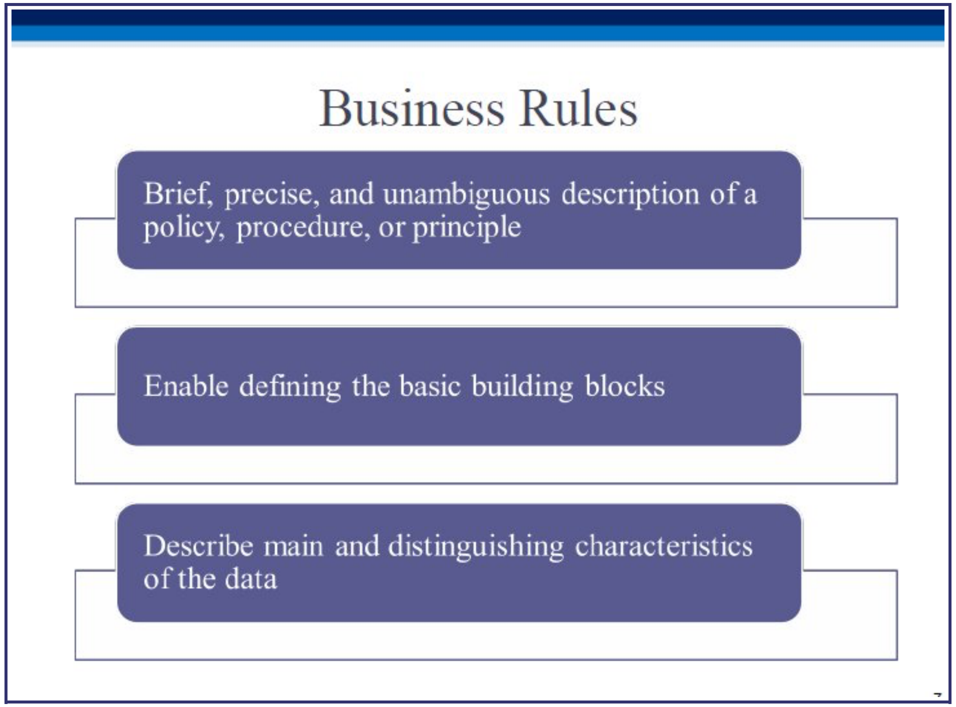


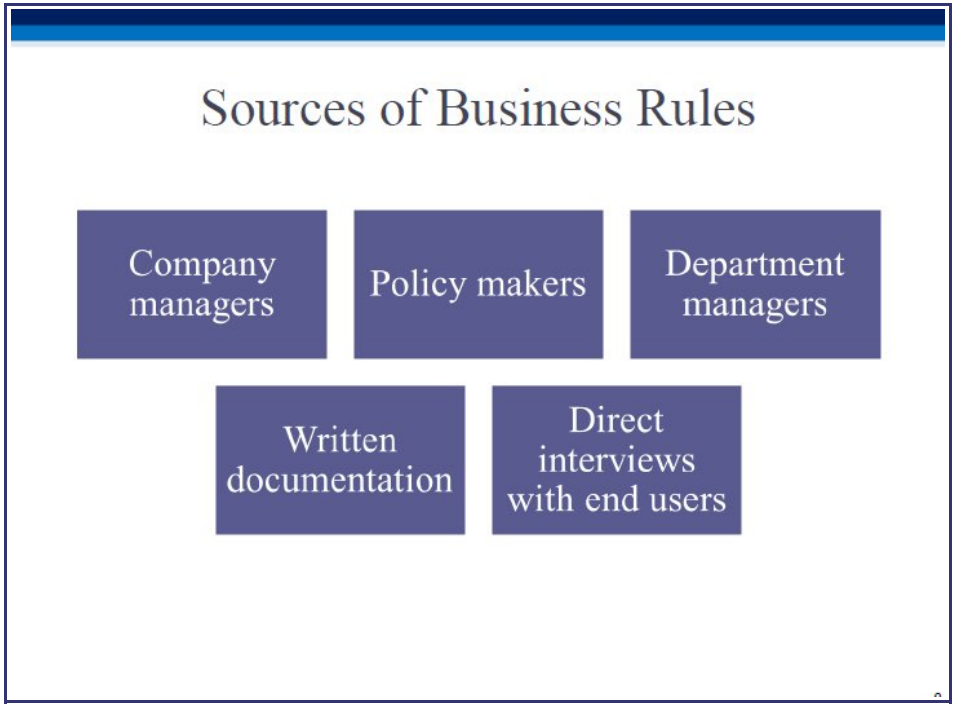
Data Modeling (1) 2023-03-08, 7:31 PM

- 
Data Modeling and Data Models
Data modeling: Iterative and progressive process of creating a specific data model for a determined problem domain
Data models: Simple representations of complex real-world data structures
Useful for supporting a specific problem domain
Model - Abstraction of a real-world object or event

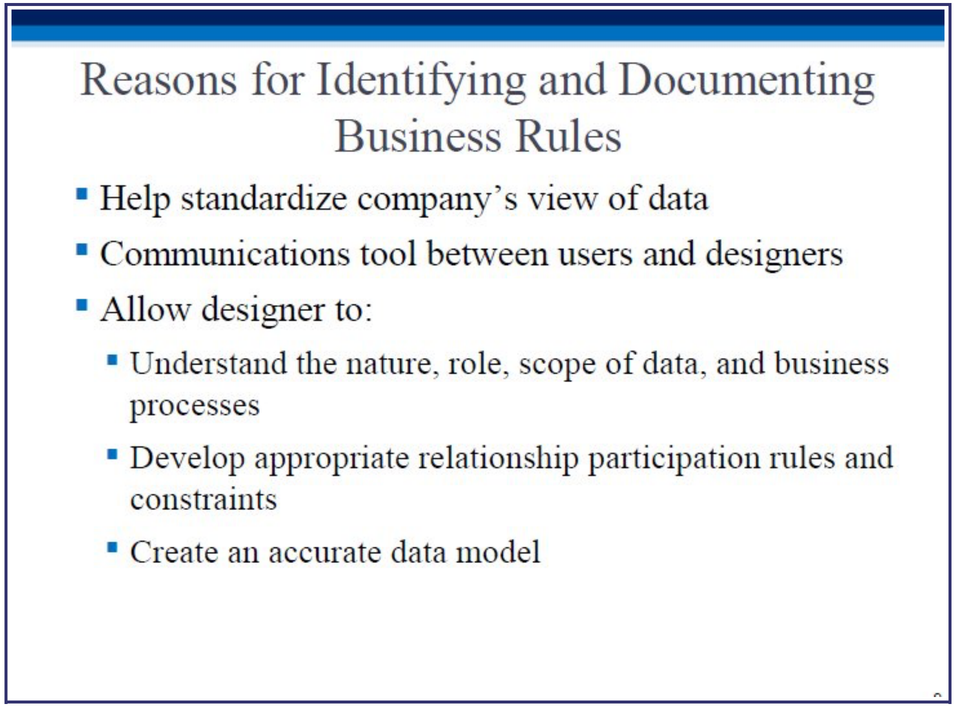
- 
Importance of Data Models
Are a communication tool
Give an overall view of the database
Organize data for various users
Are an abstraction for the creation of good database

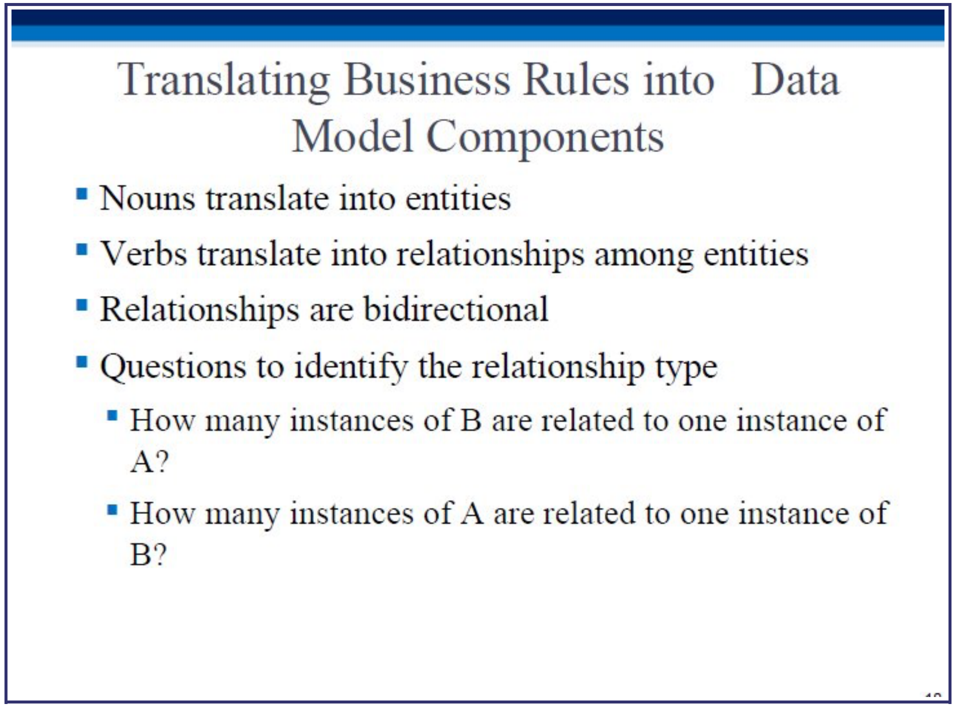
- 
Data Model Basic Building Blocks
Entity: Unique and distinct object used to collect and store data
Attribute: Characteristic of an entity
Relationship: Describes an association among entities
One-to-many (1:M)
Many-to-many (M:N or M:M)
One-to-one (1:1)
Constraint: Set of rules to ensure data integrity

- 
Business Rules
Brief, precise, and unambiguous description of a policy, procedure, or principle
Enable defining the basic building blocks
Describe main and distinguishing characteristics of the data

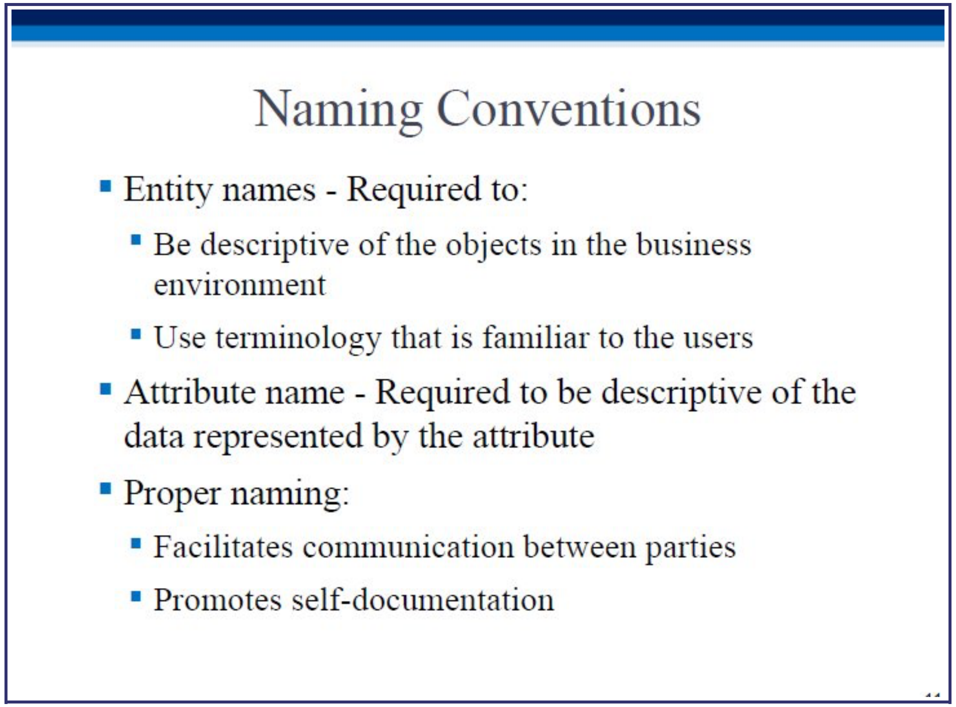
- 

Data Modeling (1) 2023-03-08, 7:31 PM

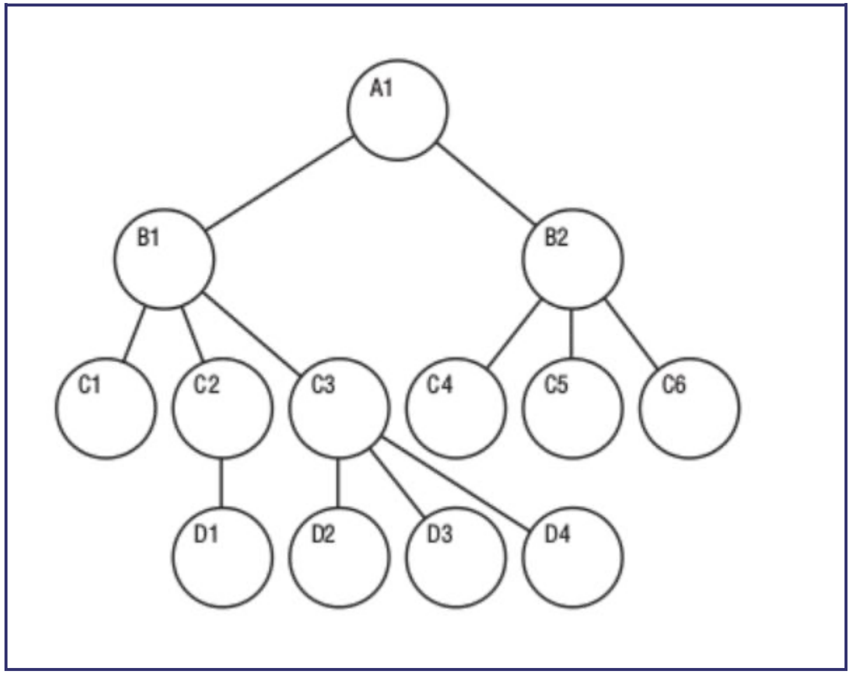
- 
Reasons for Identifying and Documenting Business Rules
Help standardize company's view of data
Communications tool between users and designers
Allow designer to:
Understand the nature, role, scope of data, and business processes
Develop appropriate relationship participation rules and constraints
Create an accurate data model

- 
Translating Business Rules into Data Model Components
Nouns translate into entities
Verbs translate into relationships among entities
Relationships are bidirectional
Questions to identify the relationship type
How many instances of B are related to one instance of A?
How many instances of A are related to one instance of B?

-


Naming Conventions
Entity names - Required to:
Be descriptive of the objects in the business environment
Use terminology that is familiar to the users
Attribute name - Required to be descriptive of the data represented by the attribute
Proper naming:
Facilitates communication between parties
Promotes self-documentation

Hierarchical modeling



At first, data was stored in individual files (transitioned from paper). The next improvement was a 'hierarchical DB model', where data was structured in the form of a tree [similar to a modern filesystem]. Data, in the form of nodes, are linked in a tree-like fashion. To traverse the tree, we need to know the underlying format ('class hierarchy, to make an analogy with classes and objects), and the actual path [eg. to relate A1 and D2, we need to traverse A1->B1->C3>D2].

Hierarchies are good for '1:M' [tree], but not 'M:N' [graph or multiple inheritance].

Data Modeling (1) 2023-03-08, 7:31 PM

Network modeling

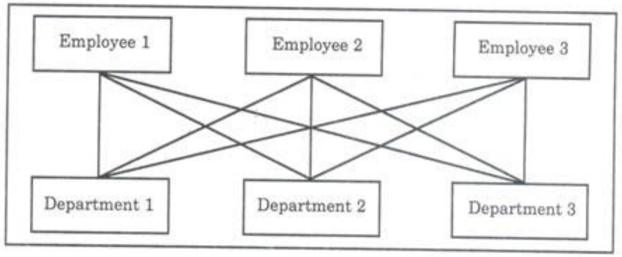
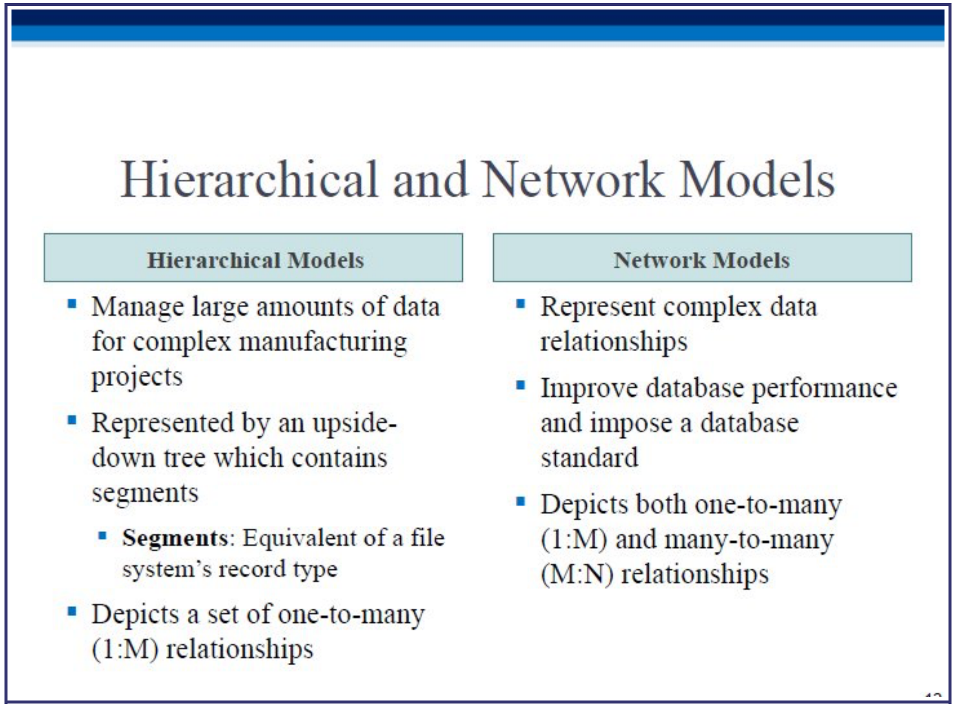
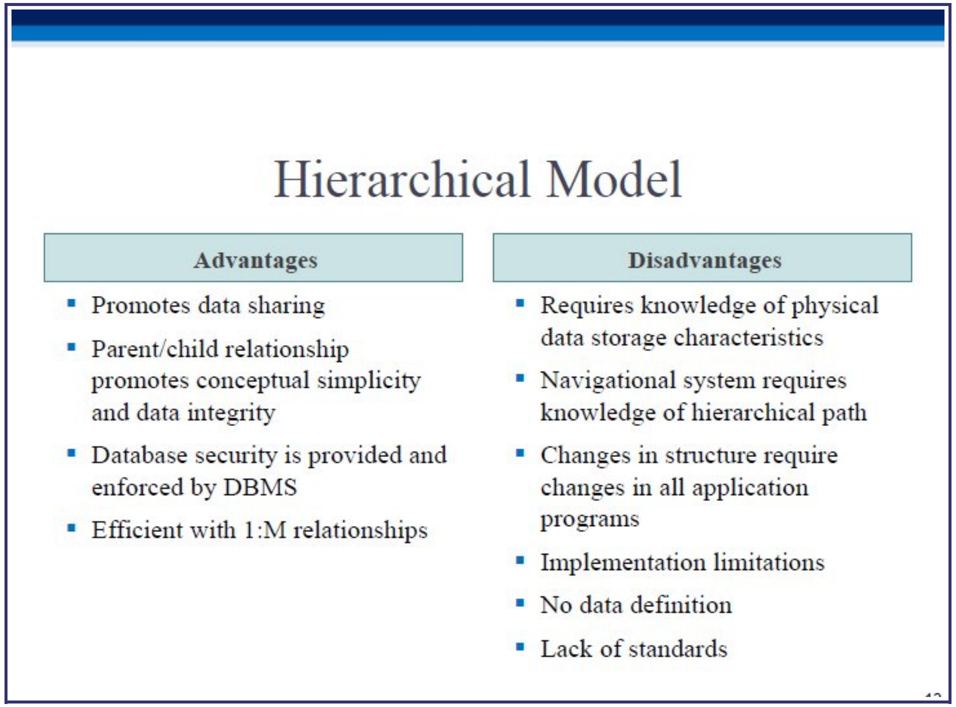


Fig. 9.5 Network model of employee information systems

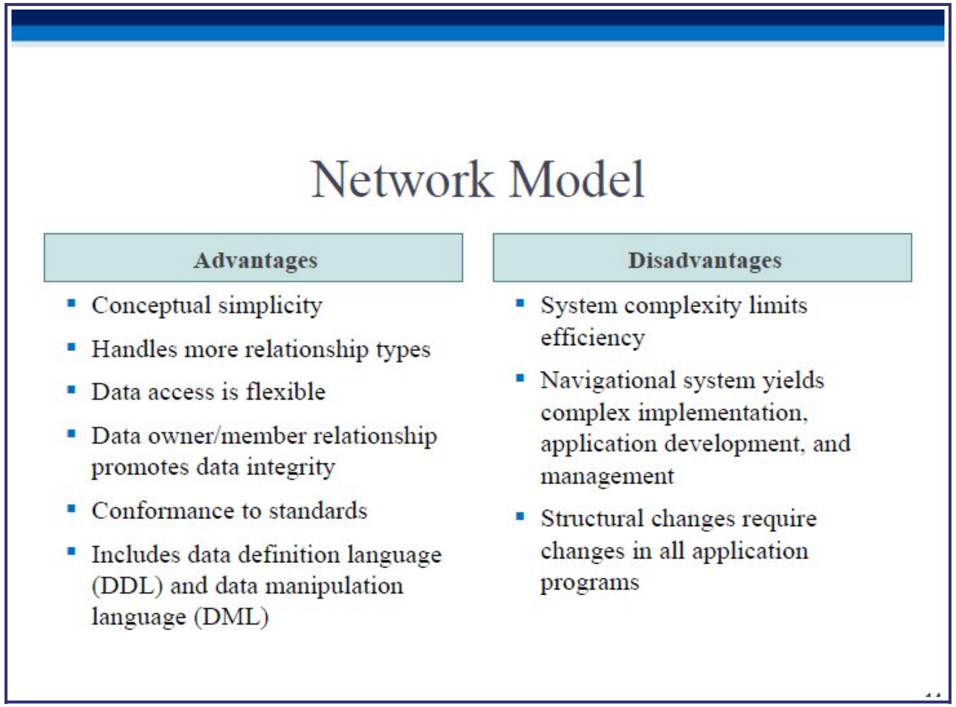
A network model is better than a hierarchical one, because it can capture M:N [in addition to the above, another example is 'products and orders'].

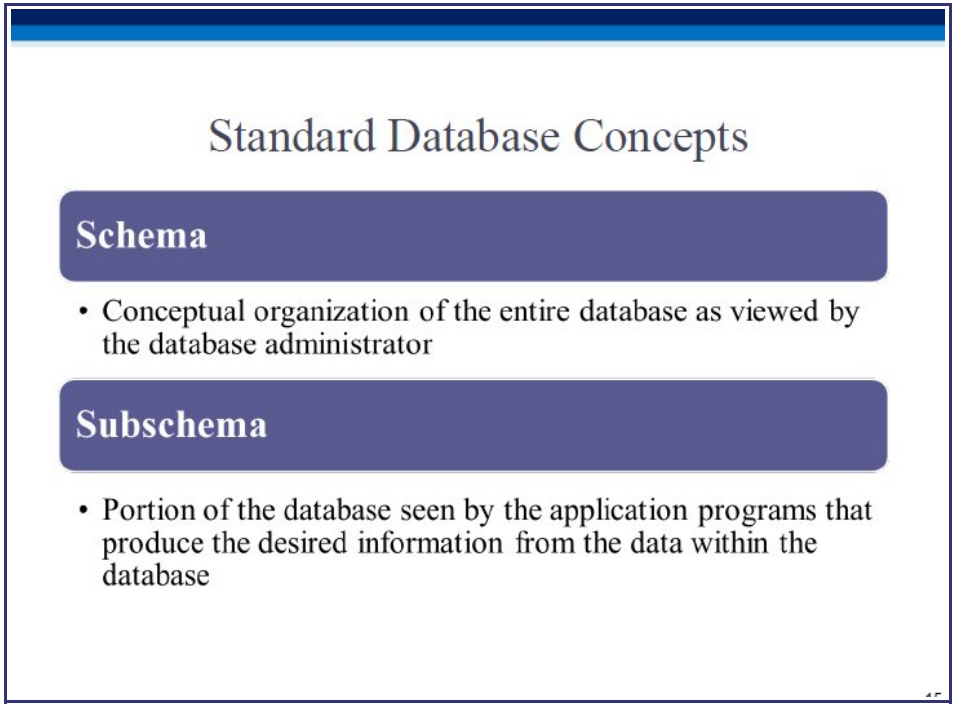
•


Hierarchical and Network Models
Hierarchical Models
Manage large amounts of data for complex manufacturing projects
Represented by an upside-down tree which contains segments
Segments: Equivalent of a file system's record type
Depicts a set of one-to-many (1:M) relationships
Network Models
Represent complex data relationships
Improve database performance and impose a database standard
Depicts both one-to-many (1:M) and many-to-many (M:N) relationships

- 

Hierarchical Model
Advantages
Promotes data sharing
Parent/child relationship promotes conceptual simplicity and data integrity
Database security is provided and enforced by DBMS
Efficient with 1:M relationships
Disadvantages
Requires knowledge of physical data storage characteristics
Navigational system requires knowledge of hierarchical path
Changes in structure require changes in all application programs
Implementation limitations
No data definition
Lack of standards


Network Model
Advantages
Conceptual simplicity
Handles more relationship types
Data access is flexible
Data owner/member relationship promotes data integrity
Conformance to standards
Includes data definition language (DDL) and data manipulation language (DML)
Disadvantages
System complexity limits efficiency
Navigational system yields complex implementation, application development, and management
Structural changes require changes in all application programs

- 
Standard Database Concepts
Schema
• Conceptual organization of the entire database as viewed by the database administrator
Subschema
• Portion of the database seen by the application programs that produce the desired information from the data within the database

Data creation, querying

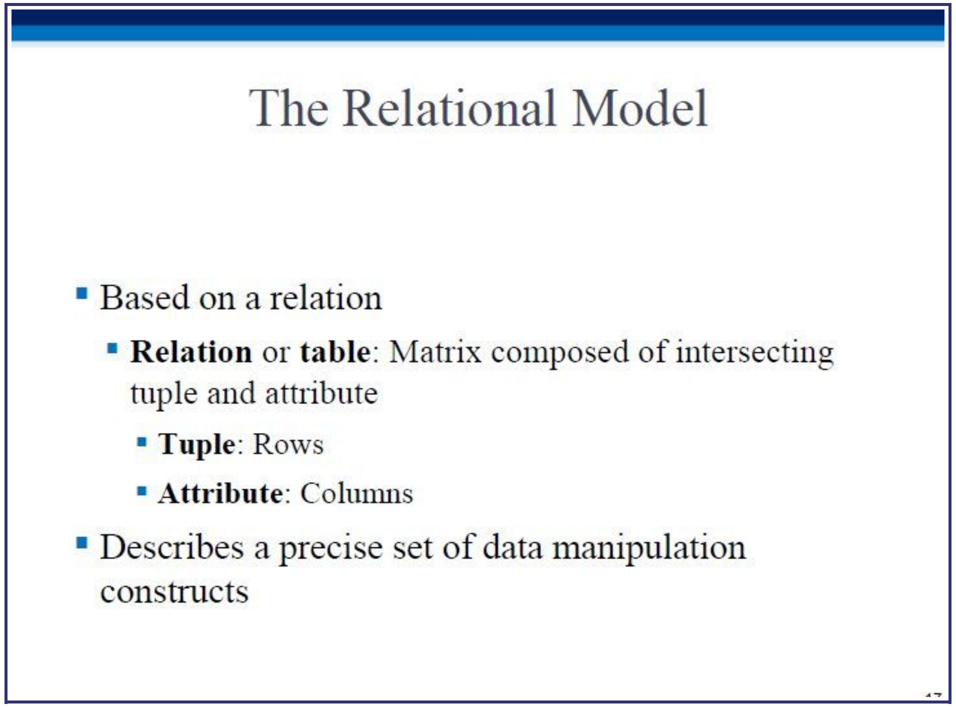
Schema data definition language (DDL)

- Enables the database administrator to define the schema components

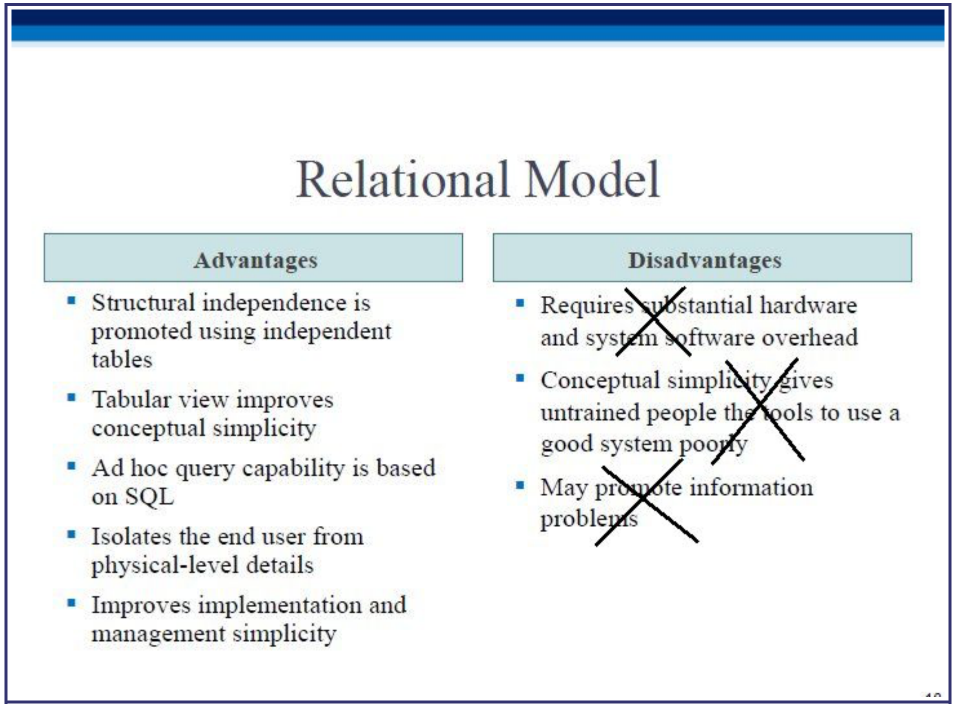
Data manipulation language (DML)

- Environment in which data can be managed and is used to work with the data in the database

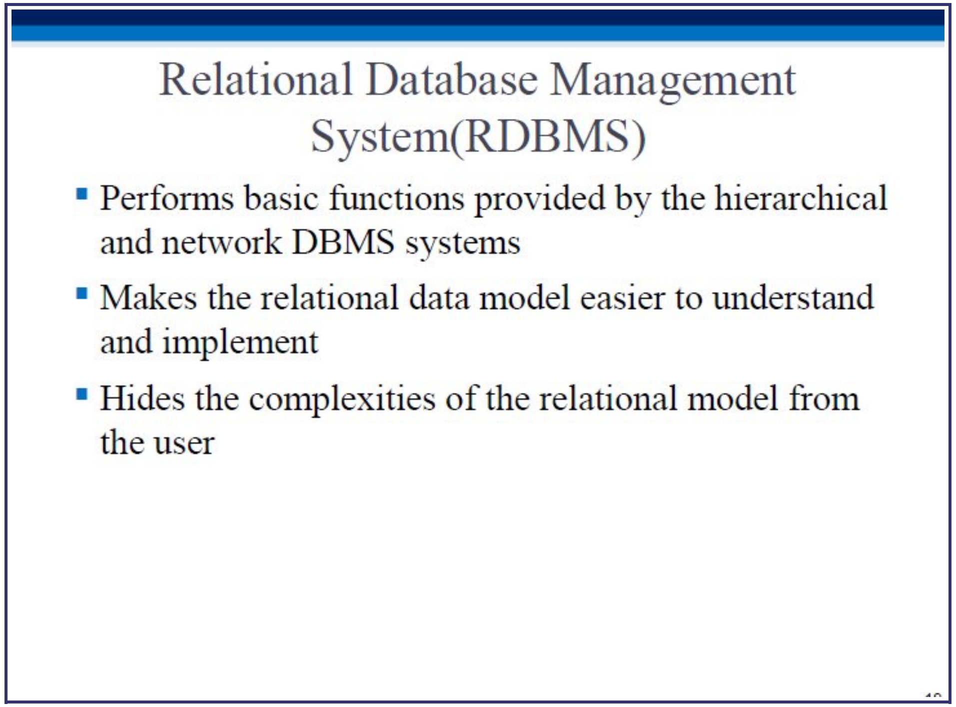
Relational model



Relational model

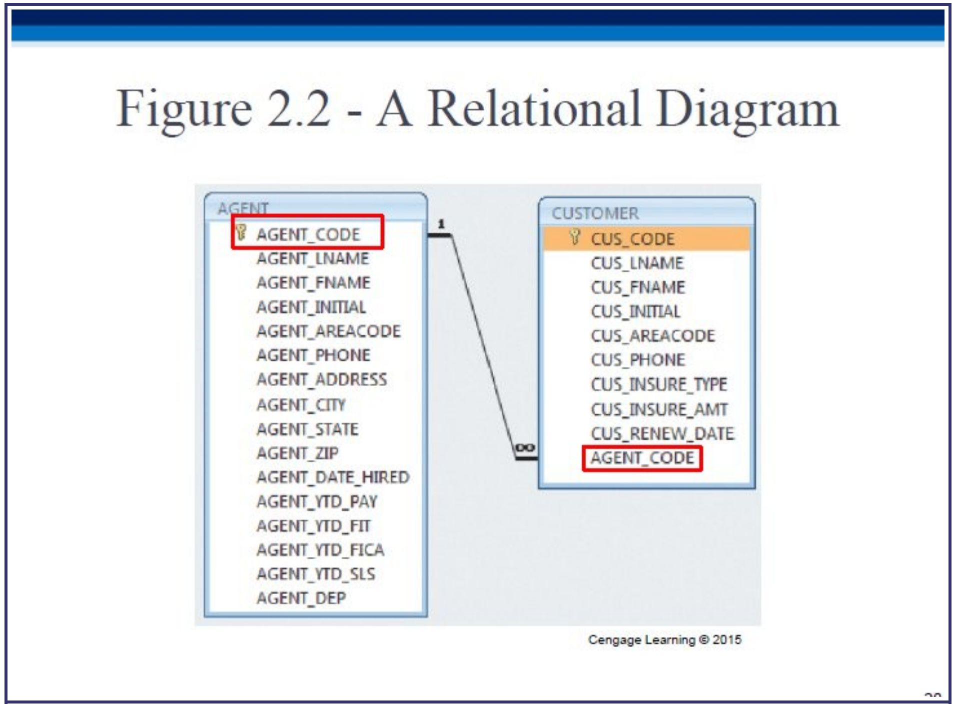


Relational DBMS



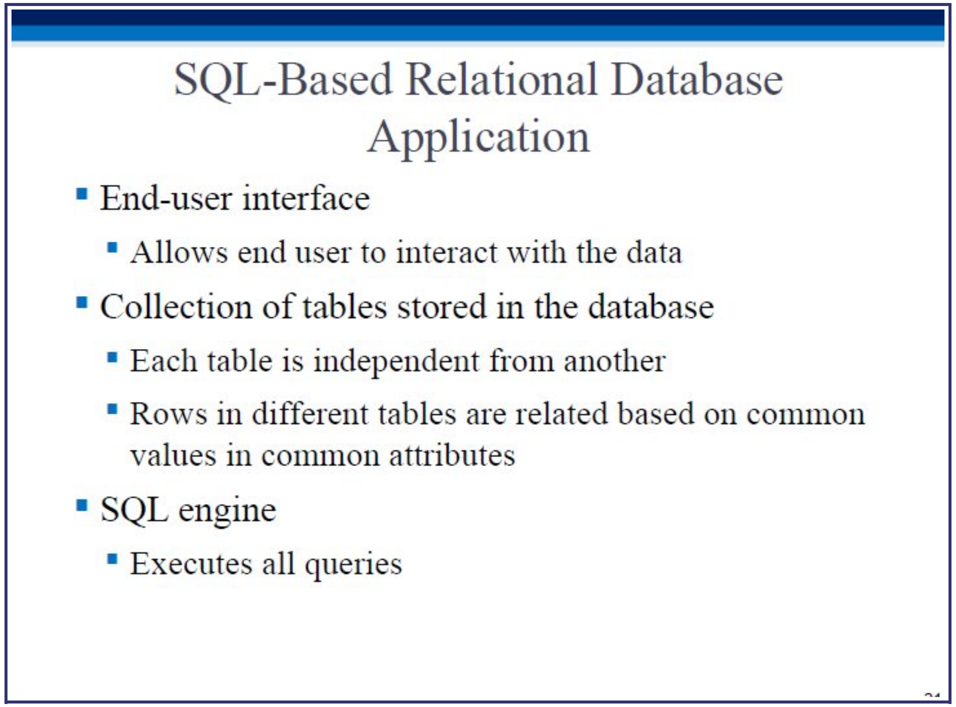
Data Modeling (1) 2023-03-08, 7:31 PM

Relation - BETWEEN entities

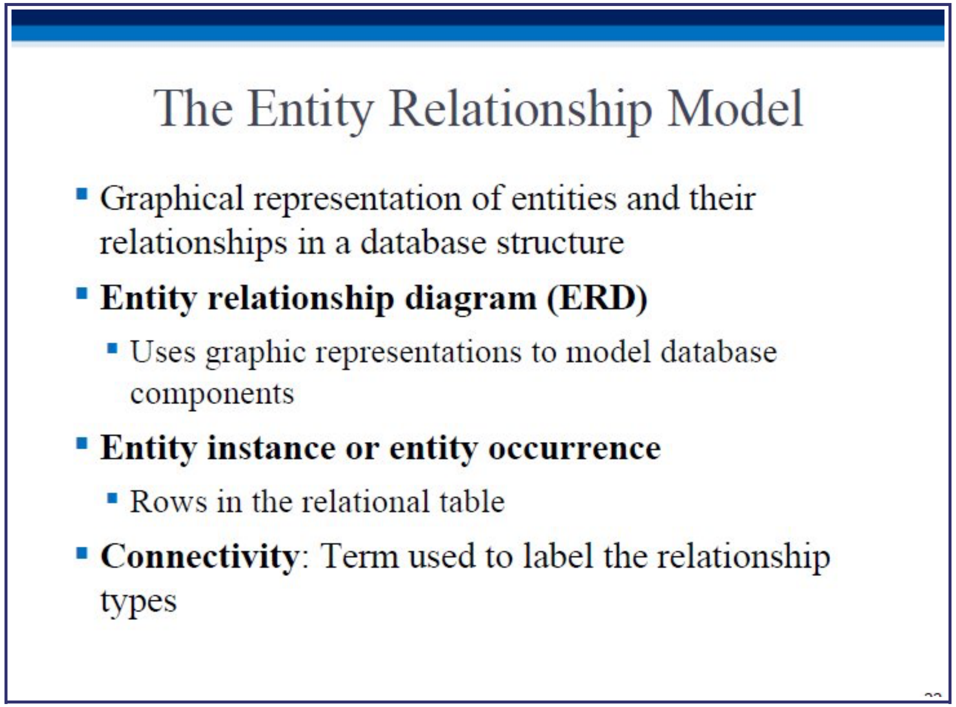


Note: this relation is NOT what relational modeling is about!! Here, we relate two entities, via a common attribute (AGENT_CODE, in our example).

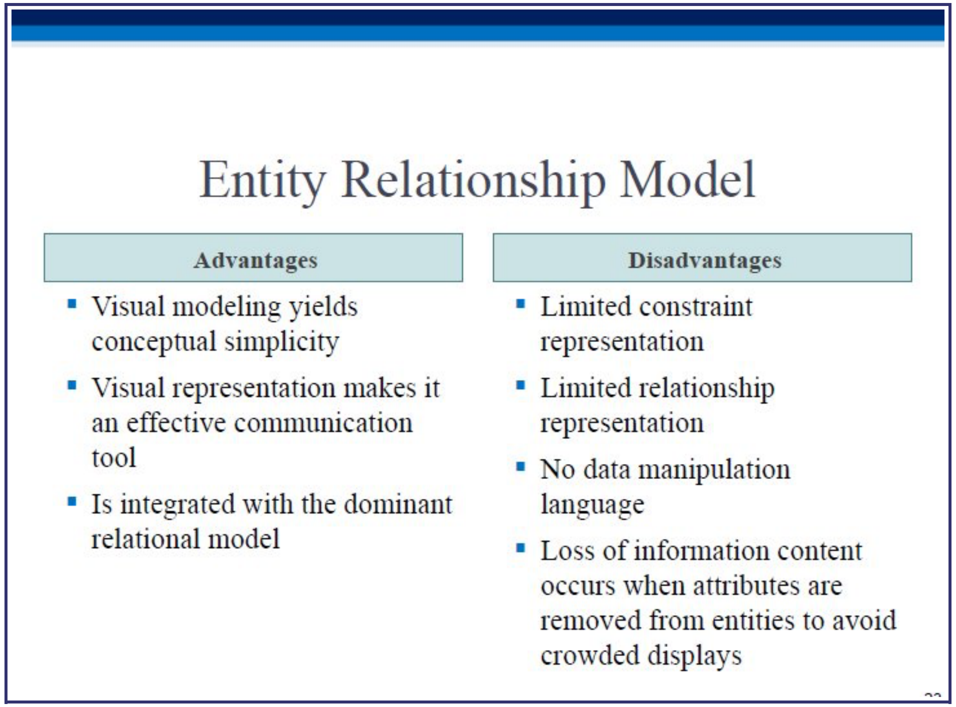
SQL + RDBMS


SQL-Based Relational Database Application
End-user interface
Allows end user to interact with the data
Collection of tables stored in the database
Each table is independent from another
Rows in different tables are related based on common values in common attributes
SQL engine
Executes all queries

E-R


The Entity Relationship Model
Graphical representation of entities and their relationships in a database structure
Entity relationship diagram (ERD)
Uses graphic representations to model database components
Entity instance or entity occurrence
Rows in the relational table
Connectivity: Term used to label the relationship types

E-R


Entity Relationship Model
Advantages
Visual modeling yields conceptual simplicity
Visual representation makes it an effective communication tool
Is integrated with the dominant relational model
Disadvantages
Limited constraint representation
Limited relationship representation
No data manipulation language
Loss of information content occurs when attributes are removed from entities to avoid crowded displays

Notations

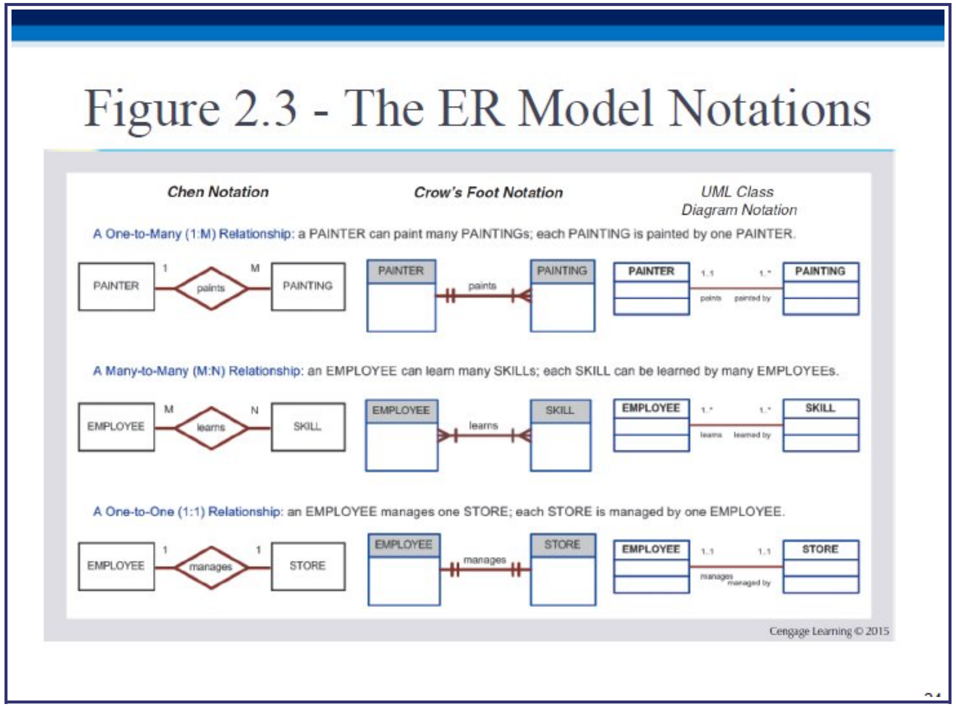


Figure 2.3 - The ER Model Notations
Chen Notation
Crow's Foot Notation
UML Class Diagram Notation
A One-to-Many (1:M) Relationship: a PAINTER can paint many PAINTINGs; each PAINTING is painted by one PAINTER.
PAINTER
paints
PAINTING
A Many-to-Many (M:N) Relationship: an EMPLOYEE can learn many SKILLs; each SKILL can be learned by many EMPLOYEEs.
EMPLOYEE
learns
SKILL
A One-to-One (1:1) Relationship: an EMPLOYEE manages one STORE; each STORE is managed by one EMPLOYEE.
EMPLOYEE
manages
STORE
Cengage Learning © 2015

Notations - more..

Additional reading: here is information on, and comparison between, four ER notations: Chen, Crow, Rein85, IDEFIX.

O-O databases

Also called 'object stores', these dbs offer(ed) a way to store ("persist") objects on disk. The objects (entity instances) are instanced from classes (entities), like with standard OO programming practice.

Advantages:

- 'cleaner' design - objects mimic real-world counterparts
- inheritance and encapsulation possible
- richer datatypes (attributes) available
- good for CAD, multimedia..

Drawbacks:

- harder to query (compared to relational DBs) - no straightforward way to build and traverse relations between objects
- relations are simpler in certain situations

The RDBMS community collectively ignored this development..

O-R databases

These are a compromise between RDBs and OODBs - they feature an O-O front-end over a relational architecture. Interfacing applications do so in an O-O way, and queries/modifications are translated to/from relational form ("ORM").

Benefits:

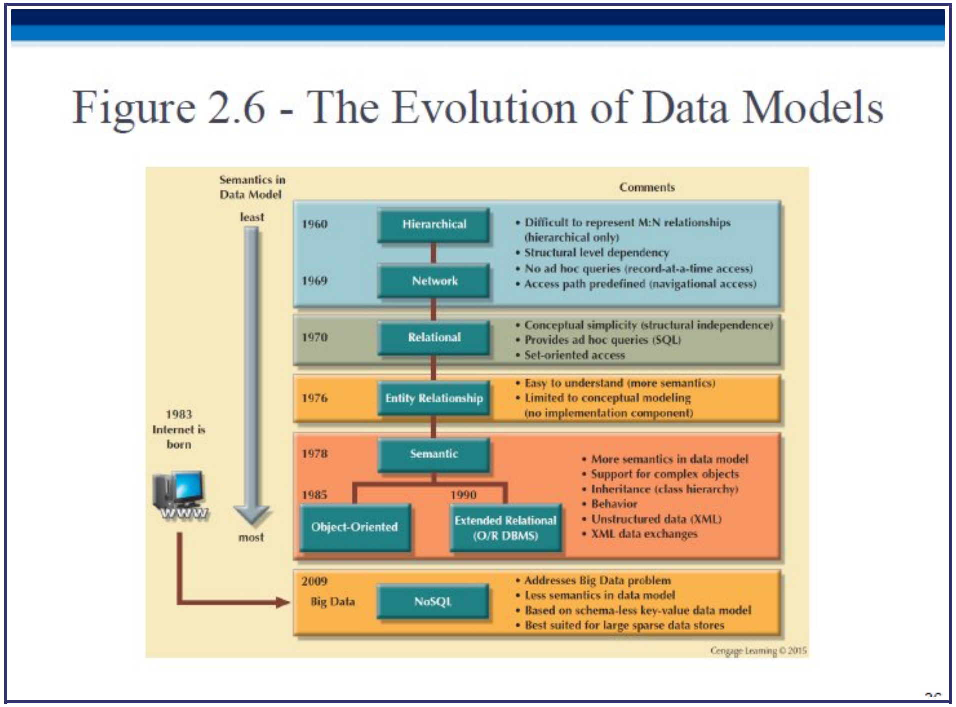
- easy to access the data from an O-O application
- queries can be simpler (can use objects' structure)

Drawback:

- performance can be poor on account of the two-way translation

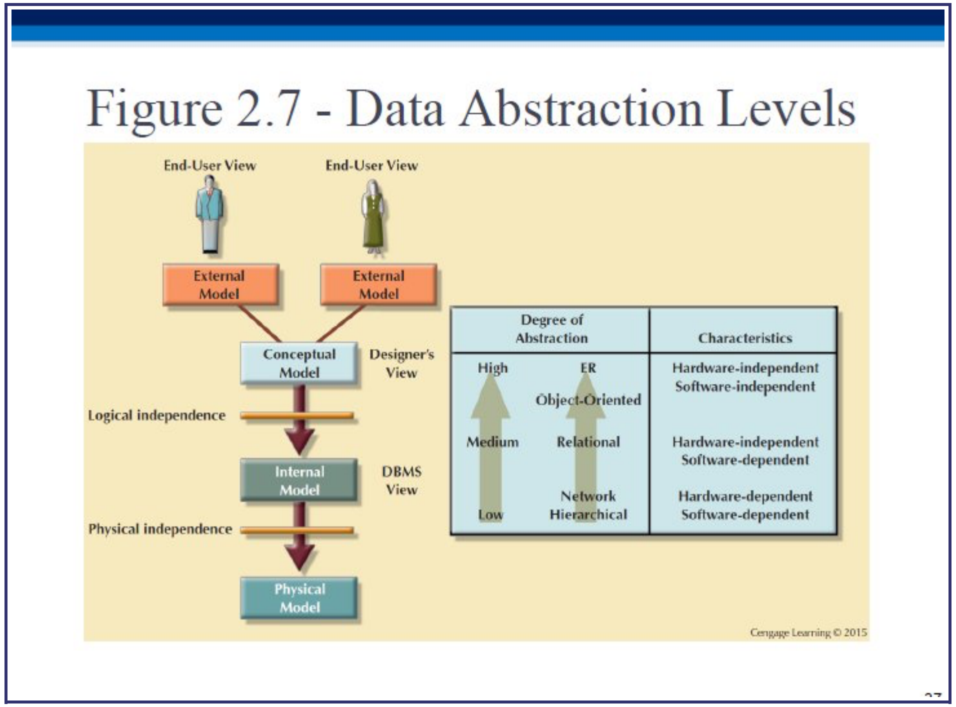
Data Modeling (1) 2023-03-08, 7:31 PM

Data models: hierarchical => => NoSQL

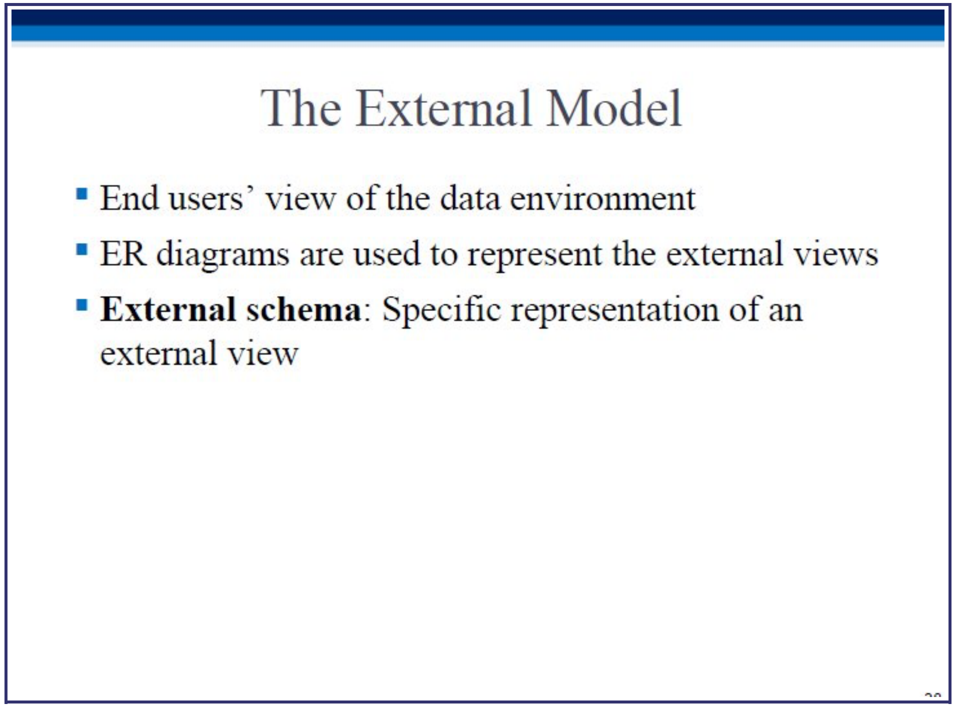


Data models have evolved - from 'hierarchical' (very rigid) to 'NoSQL' (VERY flexible).

Layered data abstraction

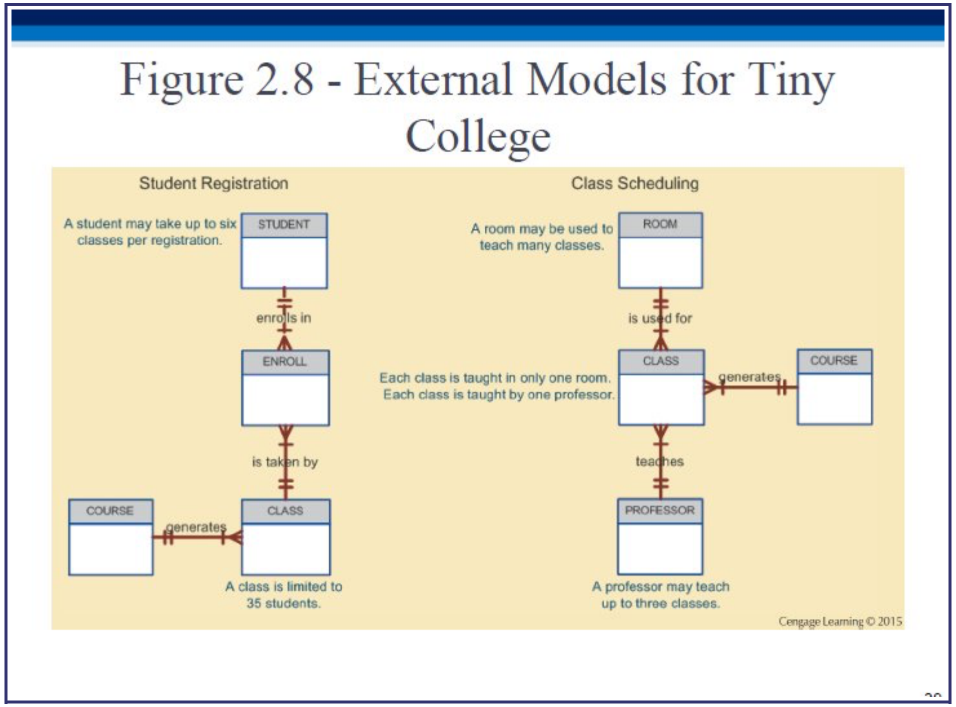


External model

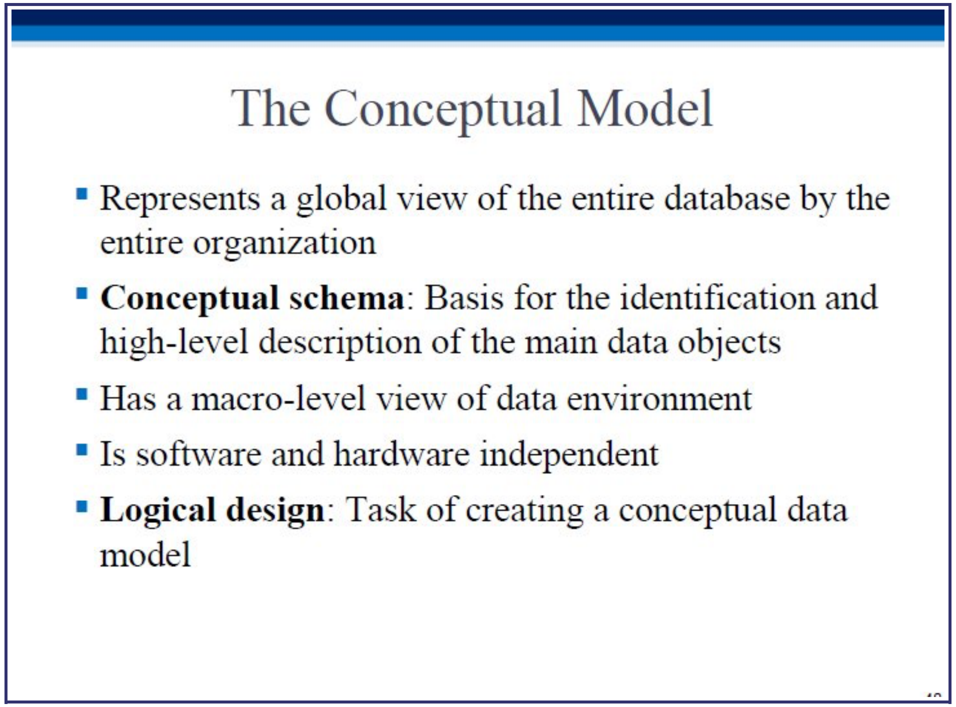


An external model is a collection of 'fragmented', 'from the stakeholders' POV', modeling of a database.

External model

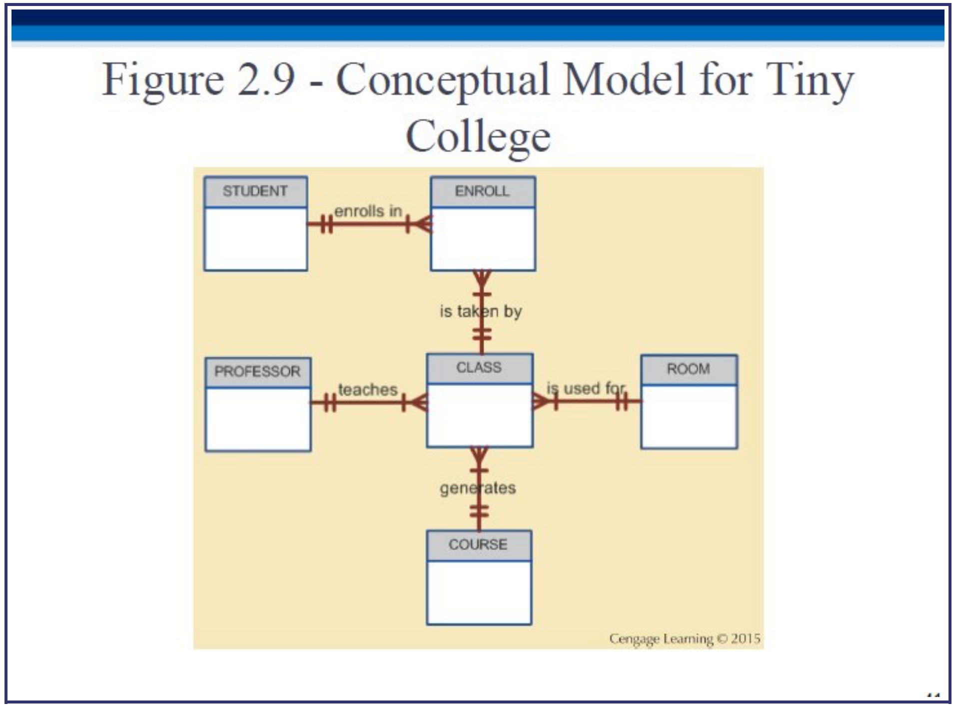


Conceptual model



Data Modeling (1) 2023-03-08, 7:31 PM

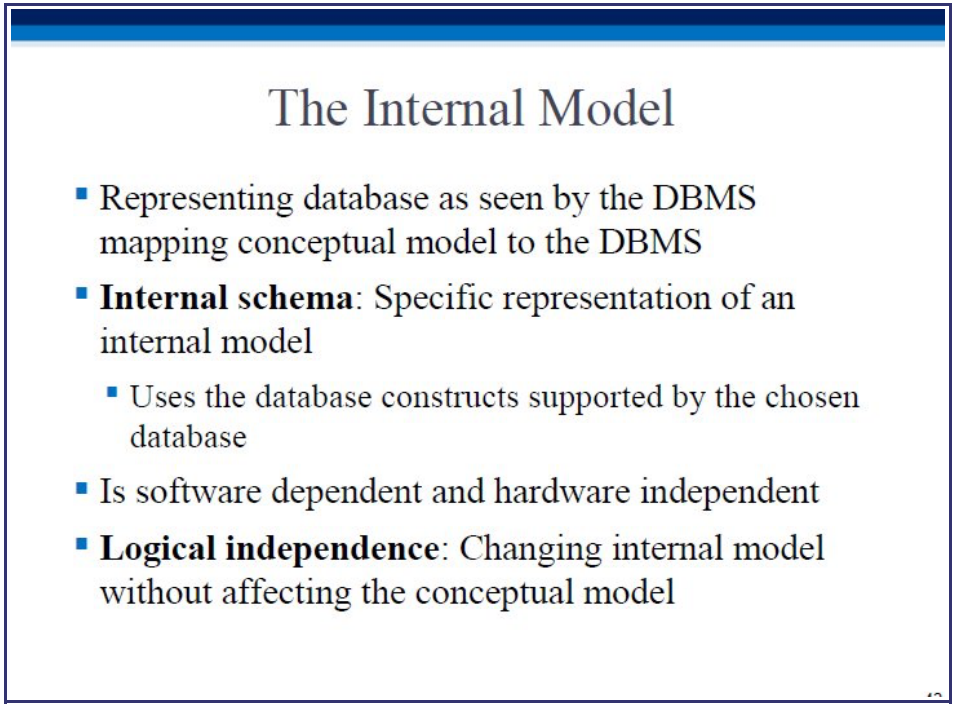
Conceptual model



A conceptual model unifies the external views into a cohesive one.

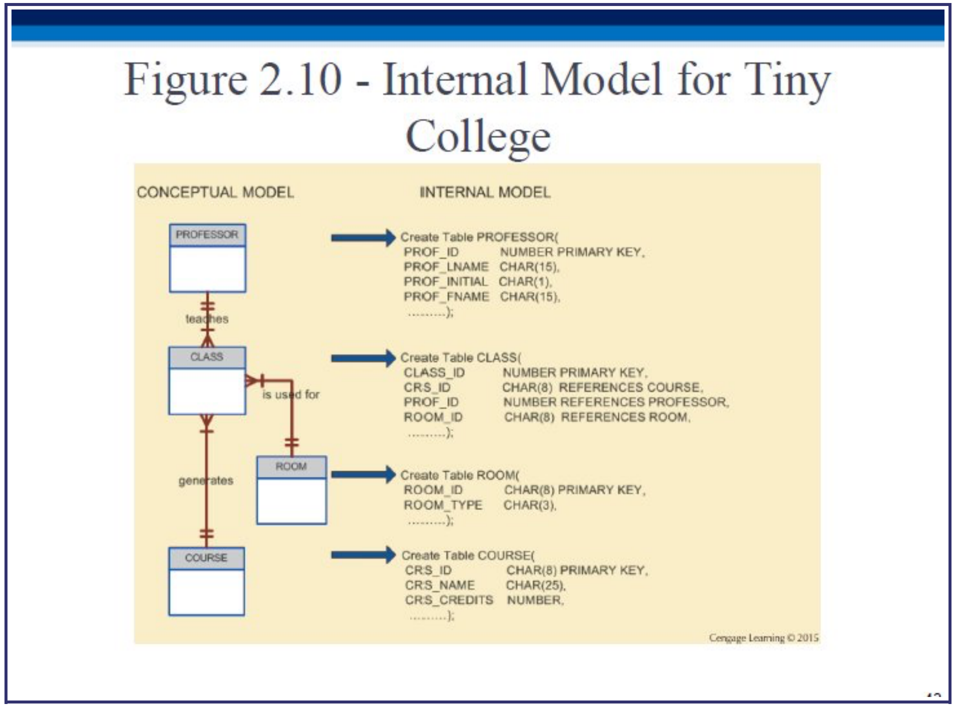
Data Modeling (1) 2023-03-08, 7:31 PM

Internal model



An internal model specifies what type of modeling (eg. relational, NoSQL...) to use for storing the data.

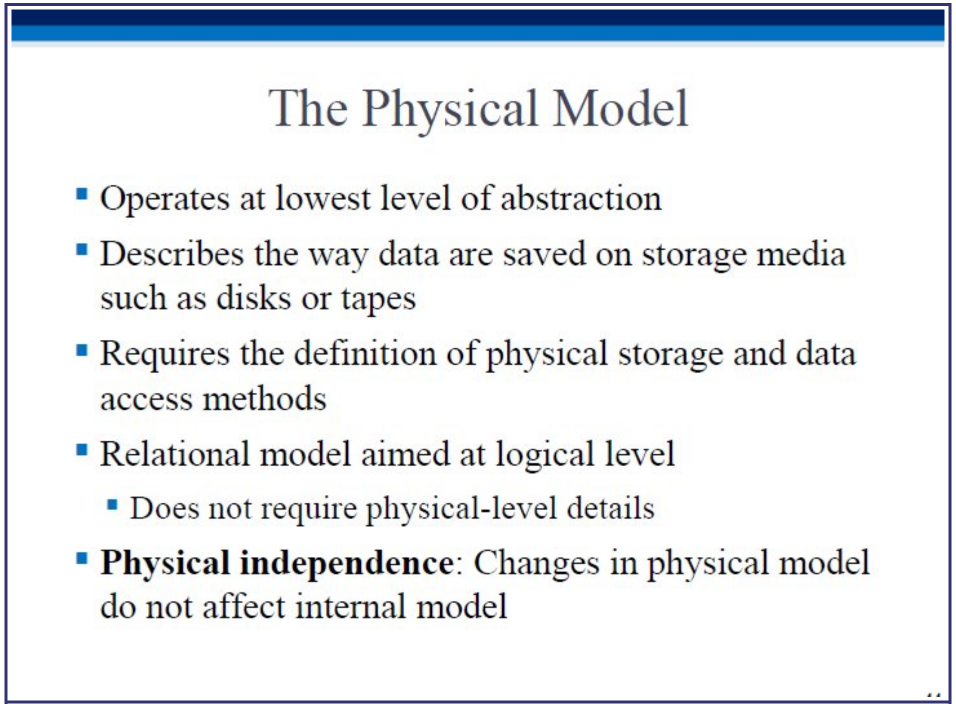
Internal model



Data Modeling (1) 2023-03-08, 7:31 PM

Data Modeling (1) 2023-03-08, 7:31 PM

Physical model



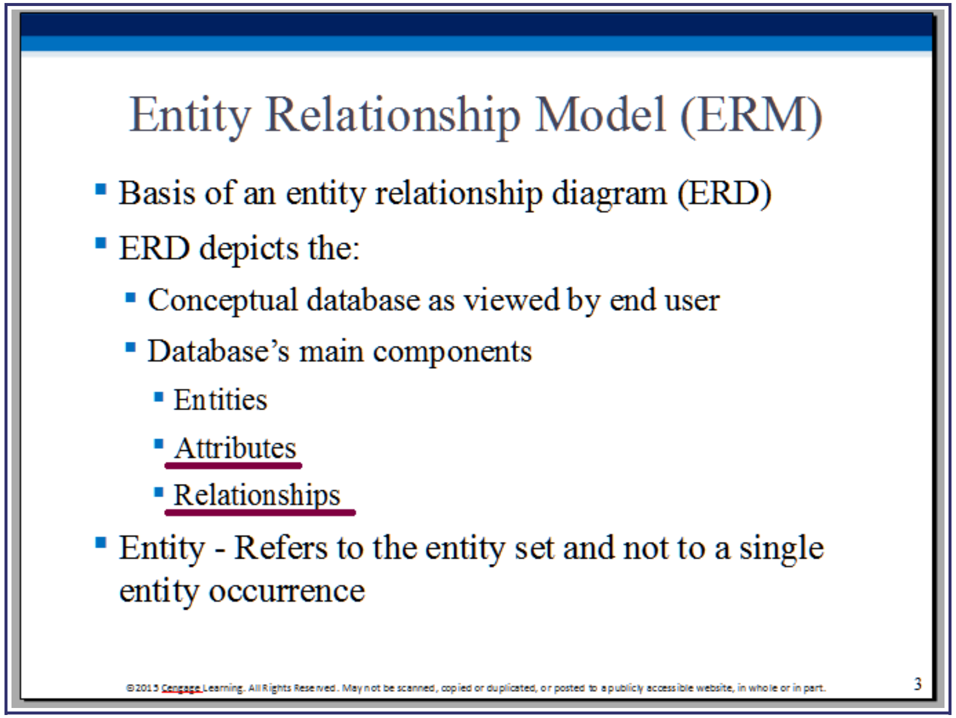
The physical model specifies actual data storage specifics (file format, APIs...).

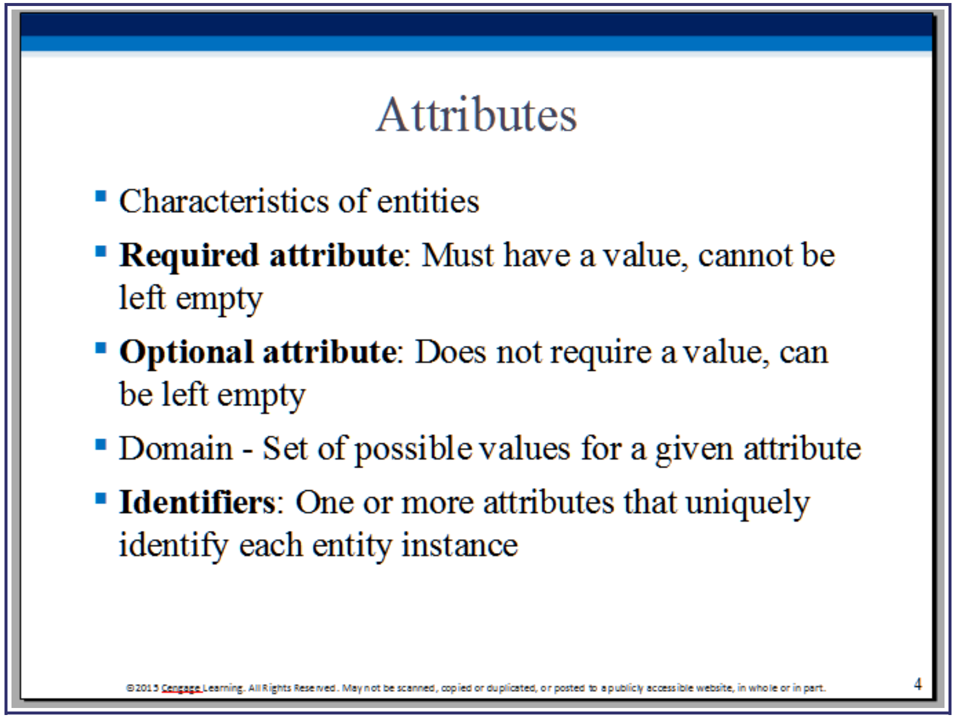
ER diagrams (1) 2023-03-08, 7:41 PM

1/34 7:41:17 * * *

← →

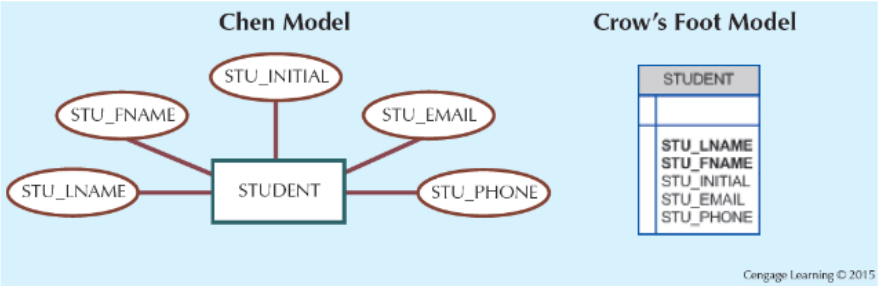
ER


Entity Relationship Model (ERM)
Basis of an entity relationship diagram (ERD)
ERD depicts the:
Conceptual database as viewed by end user
Database’s main components
Entities
Attributes
Relationships
Entity - Refers to the entity set and not to a single entity occurrence
©2013 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
3



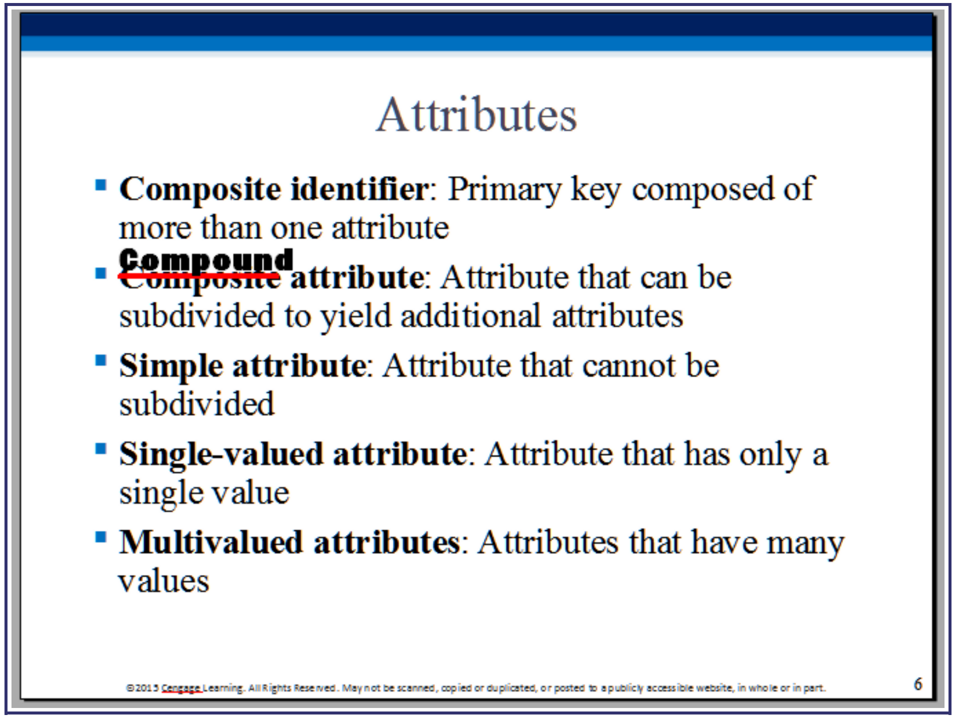
An identifier is also called a KEY, or PRIMARY KEY - **this is one of the 'key' concepts in all of database theory!!** We'll talk much more about keys later.

Figure 4.1 - The Attributes of the Student Entity: Chen and Crow's Foot



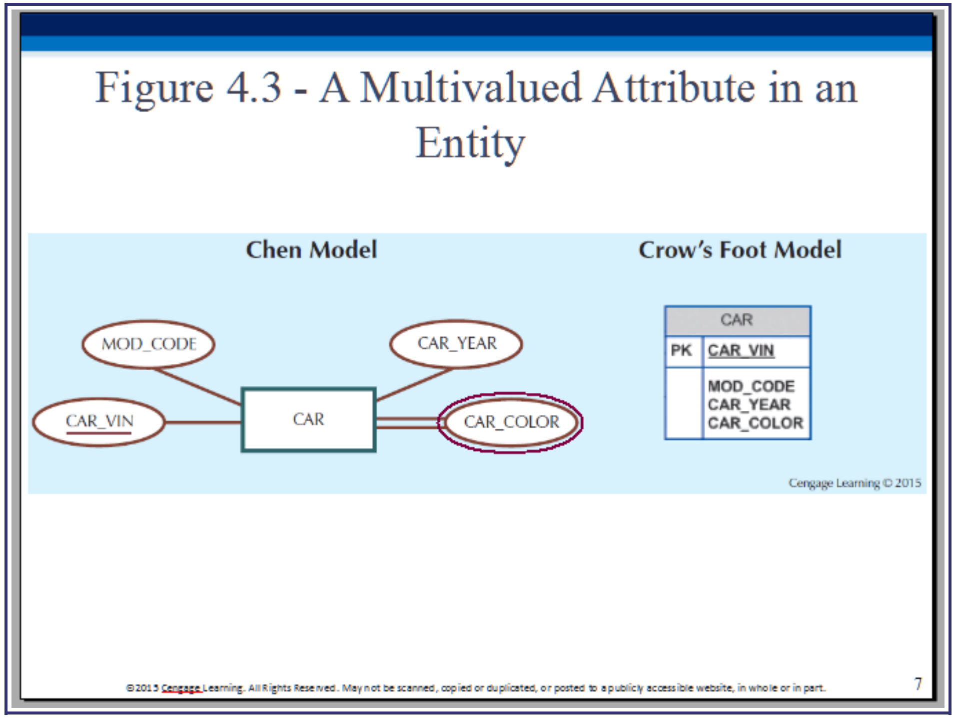
5

ER diagrams (1) 2023-03-08, 7:41 PM



FYI - here is a page on the various types of attributes.

ER diagrams (1) 2023-03-08, 7:41 PM



In Crow's Foot notation, 'bold' attributes are 'required' (can't be null).

ER diagrams (1) 2023-03-08, 7:41 PM

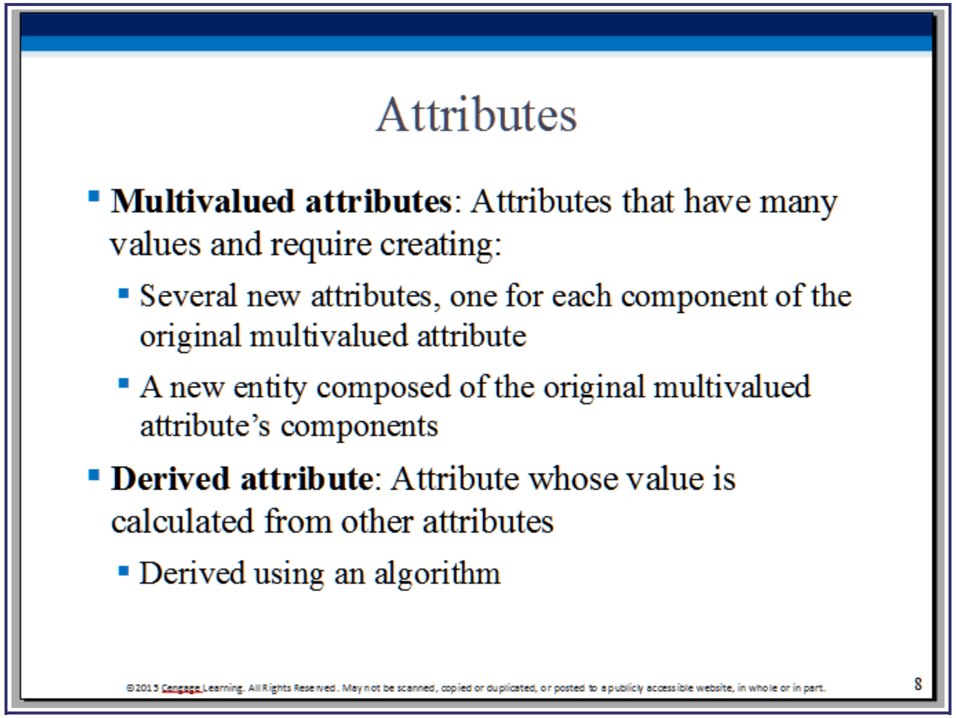
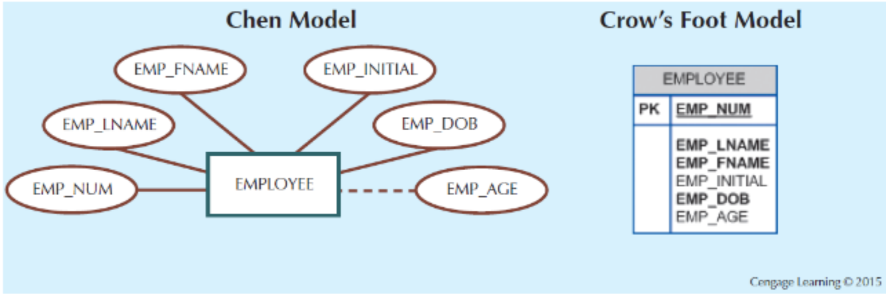


Figure 4.6 - Depiction of a Derived Attribute



9

Table 4.2 - Advantages and Disadvantages of Storing Derived Attributes

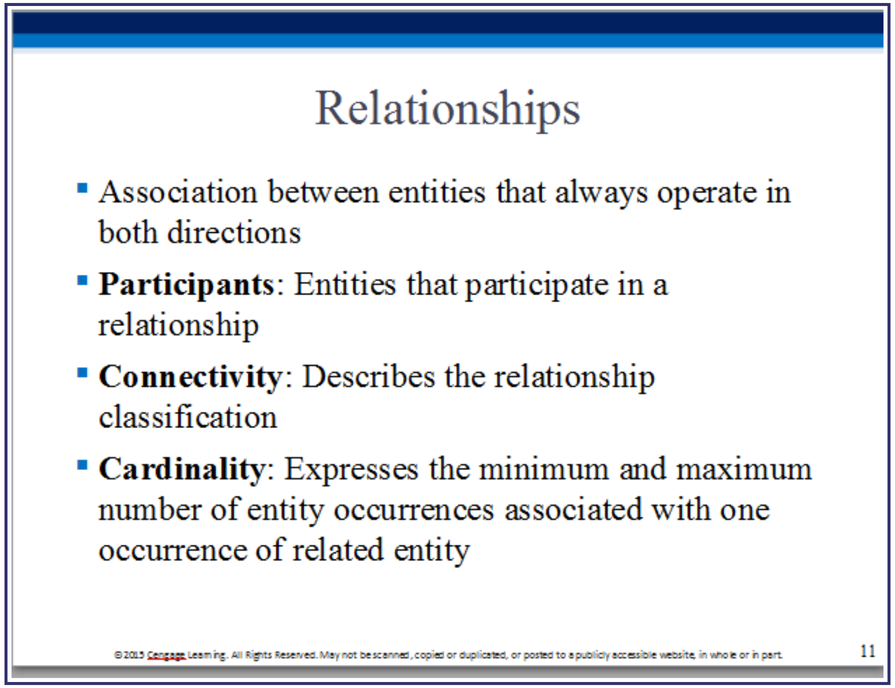
	STORED	NOT STORED
Advantage	Saves CPU processing cycles Saves data access time Data value is readily available Can be used to keep track of historical data	Saves storage space Computation always yields current value
Disadvantage	Requires constant maintenance to ensure derived value is current, especially if any values used in the calculation change	Uses CPU processing cycles Increases data access time Adds coding complexity to queries

Cengage Learning © 2015

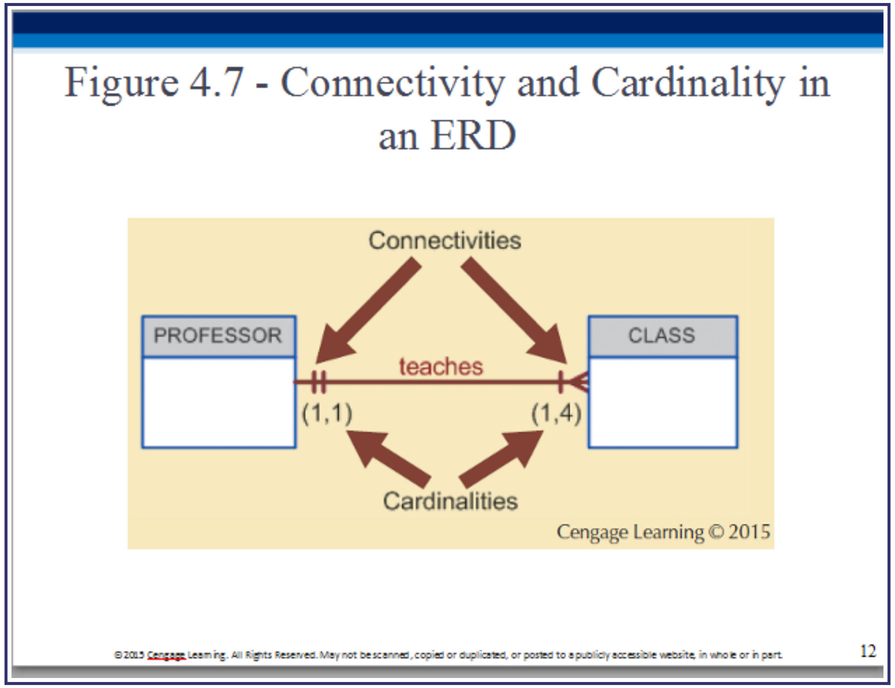
10

ER diagrams (1) 2023-03-08, 7:41 PM

ER diagrams (1) 2023-03-08, 7:41 PM



Connectivity vs cardinality



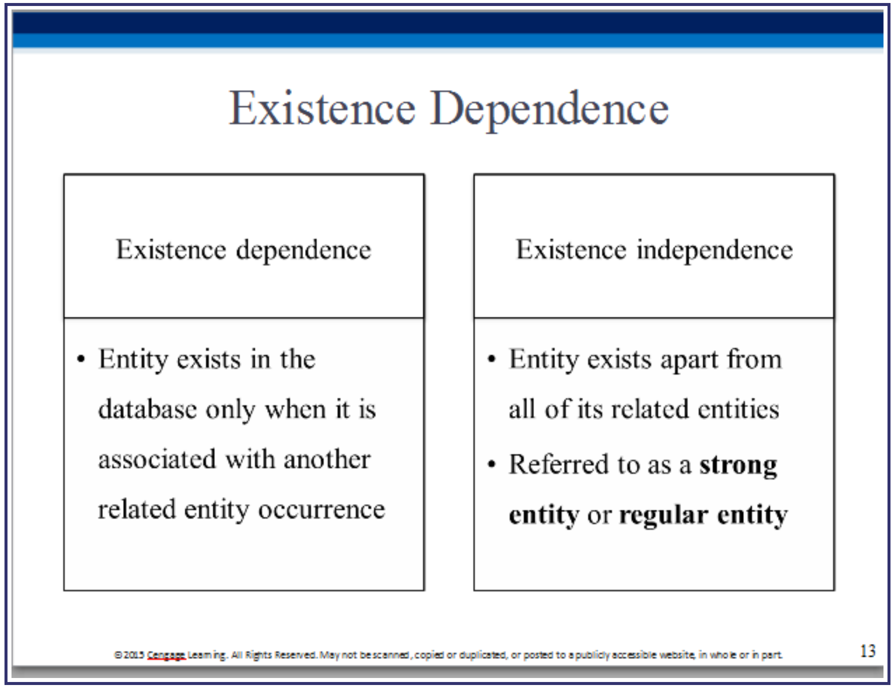
Connectivity: 1:1, 1:M or M:N (three diff ways by which two entities are related).

Cardinality: (min,max) for 1:1, 1:M or M:N (eg. 1:1 can have (1,0) as its cardinality, 1:M can have (0,4) as its cardinality). Sometimes, min is called 'modality' (and max is cardinality). The 'inside' symbols denotes min, and the outside ones, max.

Confusingly, the # rows in a table is ALSO called table's cardinality (and, # of columns is called the table's degree).

Also confusingly, 1:1, 1:M, M:N are called 'cardinality ratios'!

'Can I exist apart from you?'



Existence independence implies a strong entity; but, existence dependence (alone, ie. by itself) does NOT imply a weak entity (there needs to be one more condition, based on 'relationship strength', for it to become 'weak').

In other words, **we need to look at where the FK in the dependent entity is located.**

ER diagrams (1) 2023-03-08, 7:41 PM

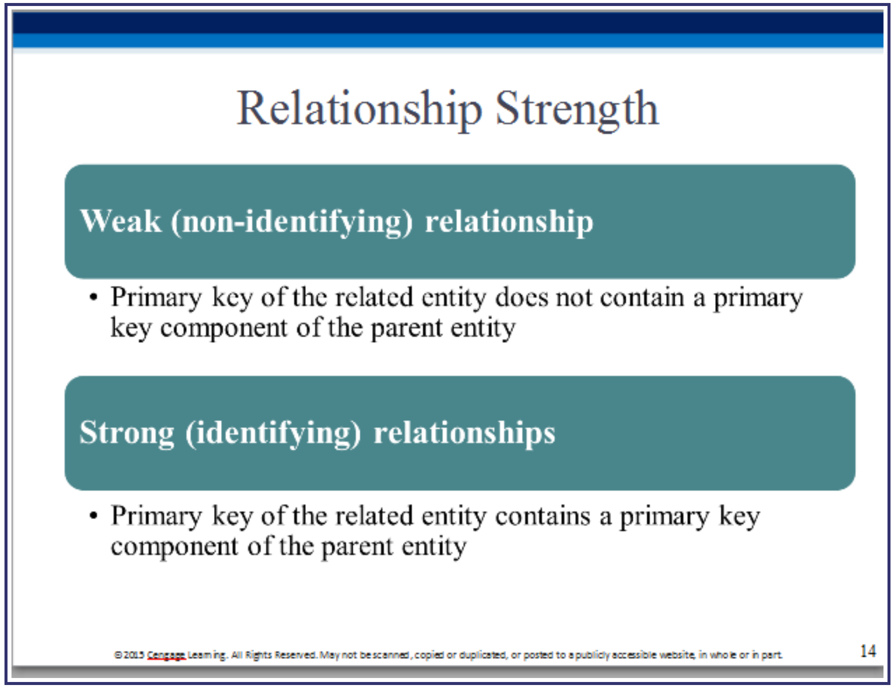
Existence dependence

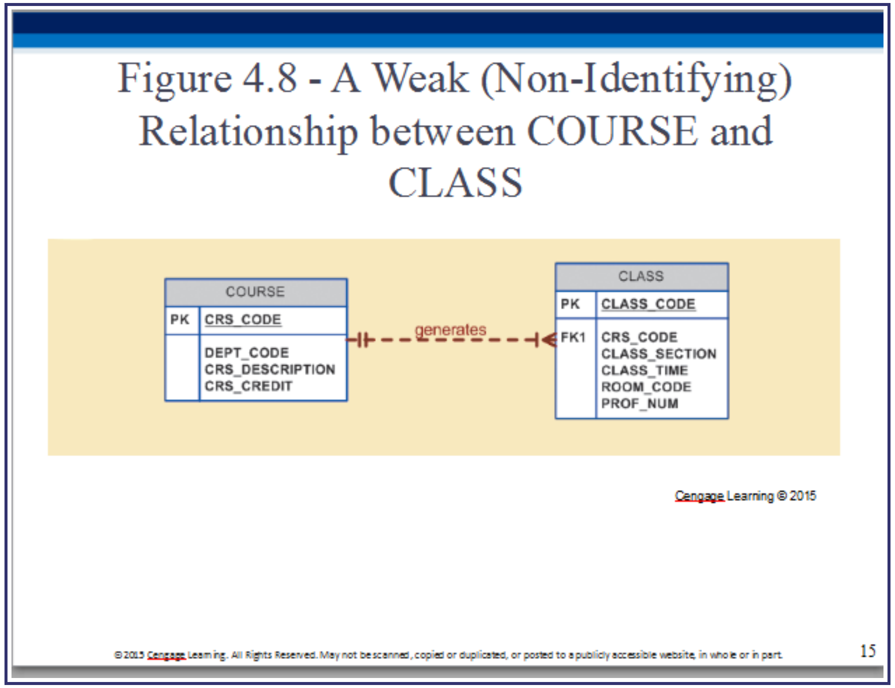
An entity B is "existent dependent" on another entity A, if, a row in B can only exist when its FK is NOT NULL, ie. a corresponding entry exists in A.

Eg. if A is EMPLOYEE and B is DEPENDENT, a dependent (eg. child) in B can only exist if there is a corresponding employee (eg. Dad) in A. THIS ALONE DOES NOT MAKE 'B' A WEAK ENTITY!

Weak vs strong relationship

Again, it's all about the FK [WHERE it goes], in the dependent entity!

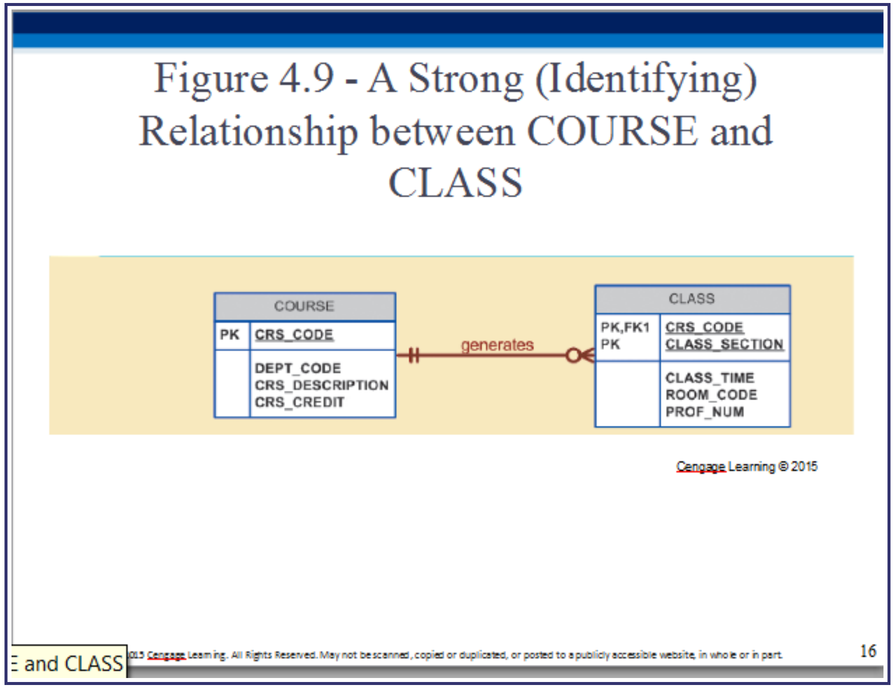




So, here, CLASS is **not** a weak entity.

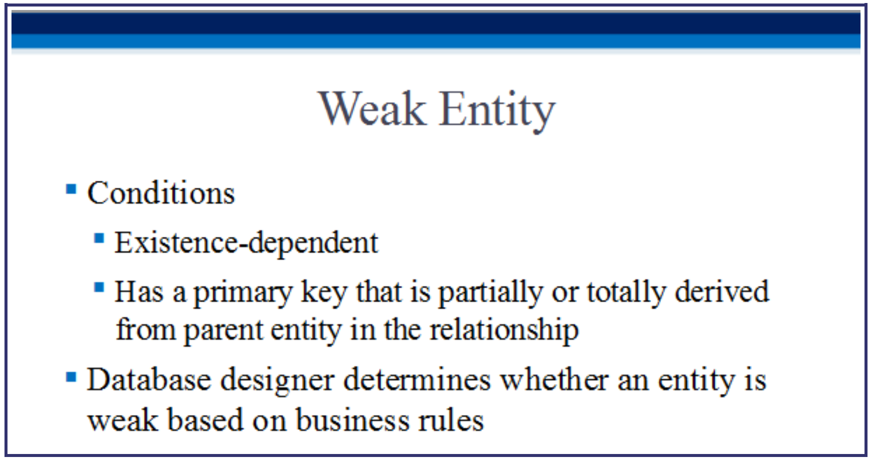
ER diagrams (1) 2023-03-08, 7:41 PM

Strong ("common PK") course-class relation



CLASS is now a weak entity (because: it is existence dependent, AND has a strong relationship).

Weak entity [two conditions]



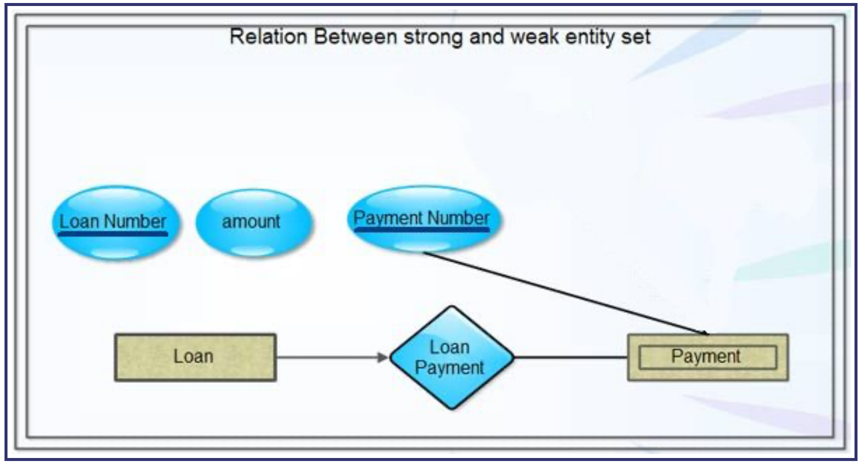
A weak entity needs to satisfy two conditions: existence dependence, strong (identifying/owning) relationship with a parent.

Note that a weak entity implies existence dependence, but existence dependence does not imply a weak entity!

Note too that a weak entity implies a strong ("owning" or "identifying") relationship.

Removing the controlling (owning) entity's key from a weak entity's PK will result in **duplicates** for remaining PK(s) - THAT is what makes it 'weak'.

Weak entity - example



Payment cannot exist independent of Loan, AND needs Loan's key to be part of its own key, so it is a weak entity.

Weak entity

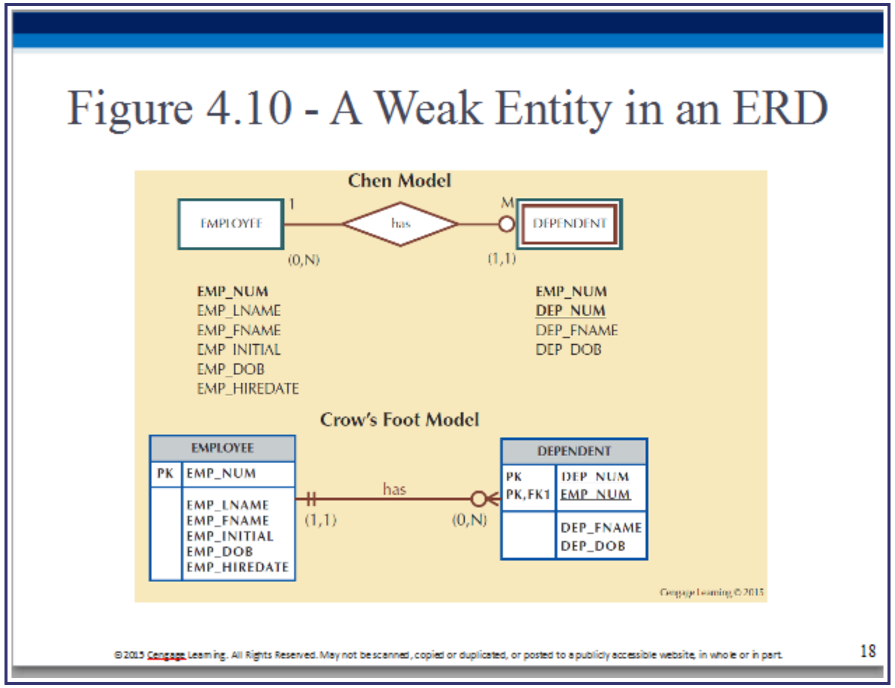


Figure 4.10 - A Weak Entity in an ERD
Chen Model
EMPLOYEE
has
DEPENDENT
1
M
(0,N)
(1,1)
EMP_NUM
EMP_LNAME
EMP_FNAME
EMP_INITIAL
EMP_DOB
EMP_HIREDATE
EMP_NUM
DEP_NUM
DEP_FNAME
DEP_DOB
Crow's Foot Model
EMPLOYEE
PK EMP_NUM
EMP_LNAME
EMP_FNAME
EMP_INITIAL
EMP_DOB
EMP_HIREDATE
has
(1,1)
(0,N)
DEPENDENT
PK DEP_NUM
PK,FK1 EMP_NUM
DEP_FNAME
DEP_DOB
Cengage Learning © 2015
© 2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
18

ER diagrams (1) 2023-03-08, 7:41 PM

Weak entity

Figure 4.11 - A Weak Entity

Table name: EMPLOYEE **Database name: Ch04_ShortCo**

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIREDATE
1001	Callifante	Jeanine	J	12-Mar-64	25-May-97
1002	Smithson	William	K	23-Nov-70	28-May-97
1003	Washington	Herman	H	15-Aug-68	28-May-97
1004	Chen	Lydia	B	23-Mar-74	15-Oct-98
1005	Johnson	Melanie		28-Sep-66	20-Dec-98
1006	Ortega	Jorge	G	12-Jul-79	05-Jan-02
1007	O'Donnell	Peter	D	10-Jun-71	23-Jun-02
1008	Brzenski	Barbara	A	12-Feb-70	01-Nov-03

Table name: DEPENDENT

EMP_NUM	DEP_NUM	DEP_FNAME	DEP_DOB
1001	1	Annelise	05-Dec-97
1001	2	Jorge	30-Sep-02
1003	1	Suzanne	25-Jan-04
1006	1	Carlos	25-May-01
1008	1	Michael	19-Feb-95
1008	2	George	27-Jun-98
1008	3	Katherine	18-Aug-03

 19

ER diagrams (1) 2023-03-08, 7:41 PM

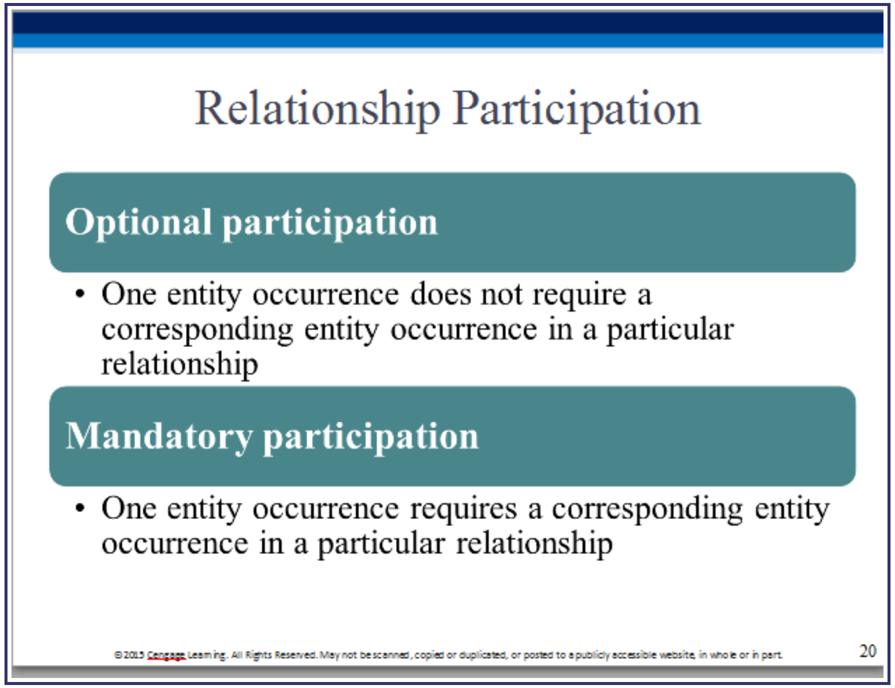


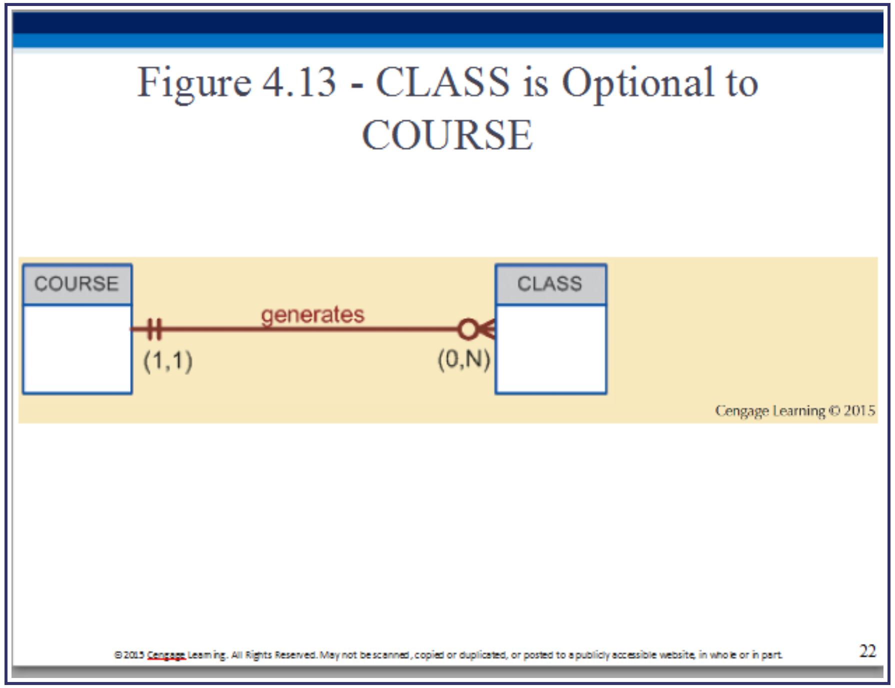
Table 4.3 - Crow's Foot Symbols

CROW'S FOOT SYMBOLS	CARDINALITY	COMMENT
	(0,N)	Zero or many; the "many" side is optional.
	(1,N)	One or many; the "many" side is mandatory.
	(1,1)	One and only one; the "1" side is mandatory.
	(0,1)	Zero or one; the "1" side is optional.

Cengage Learning © 2015

 21

ER diagrams (1) 2023-03-08, 7:41 PM



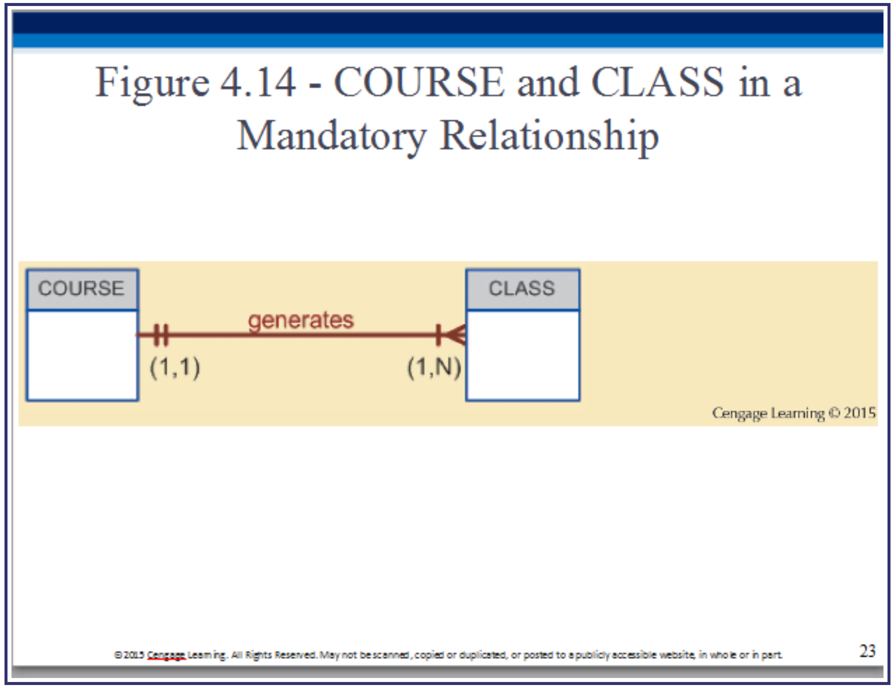
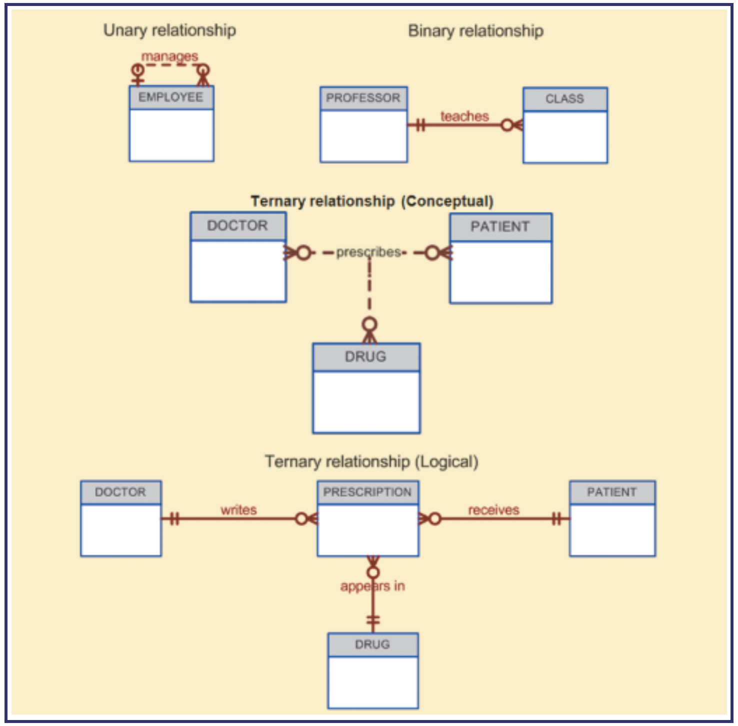


Figure 4.14 - COURSE and CLASS in a Mandatory Relationship
COURSE
CLASS
generates
(1,1)
(1,N)
Cengage Learning © 2015
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
23

ER diagrams (1) 2023-03-08, 7:41 PM

Relationship Degree

- Indicates the number of entities or participants associated with a relationship
- **Unary relationship**: Association is maintained within a single entity
  - **Recursive relationship**: Relationship exists between occurrences of the same entity set
- **Binary relationship**: Two entities are associated
- **Ternary relationship**: Three entities are associated


Unary relationship
Binary relationship
manages
EMPLOYEE
PROFESSOR
teaches
CLASS
Ternary relationship (Conceptual)
DOCTOR
prescribes
PATIENT
DRUG
Ternary relationship (Logical)
DOCTOR
writes
PRESCRIPTION
receives
PATIENT
appears in
DRUG

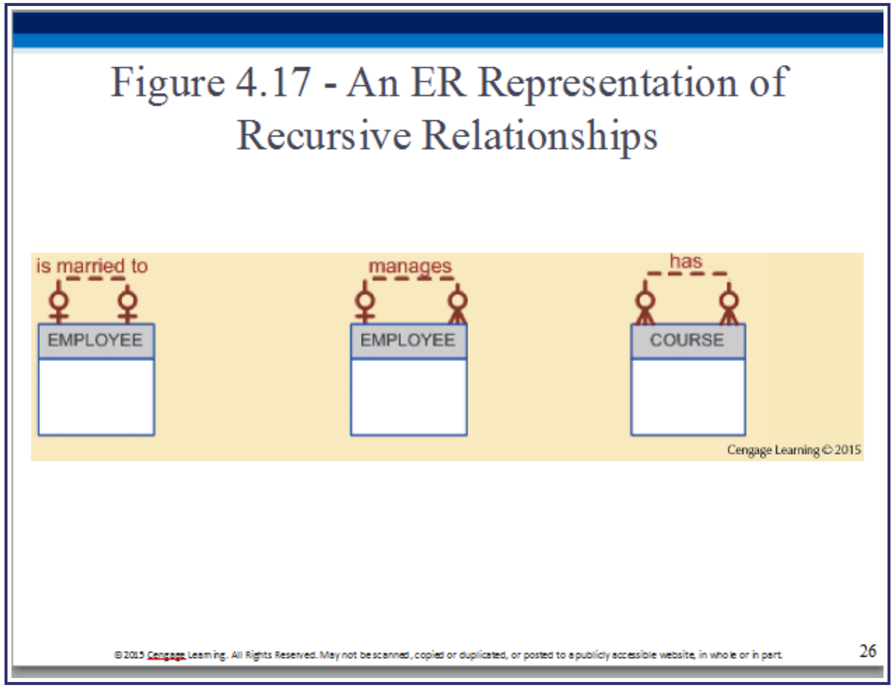


Figure 4.17 - An ER Representation of Recursive Relationships
is married to
EMPLOYEE
manages
EMPLOYEE
has
COURSE
Cengage Learning © 2015
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
26

Bridge entities

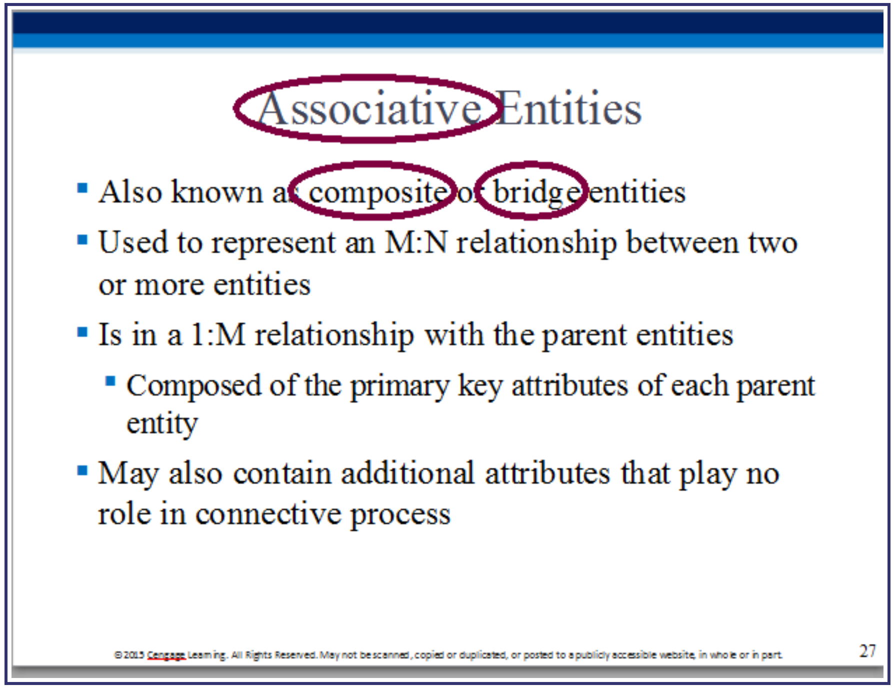


Figure 4.23 - Converting the M:N Relationship into Two 1:M Relationships

Table name: STUDENT **Database name: Ch04_CollegeTry**

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL

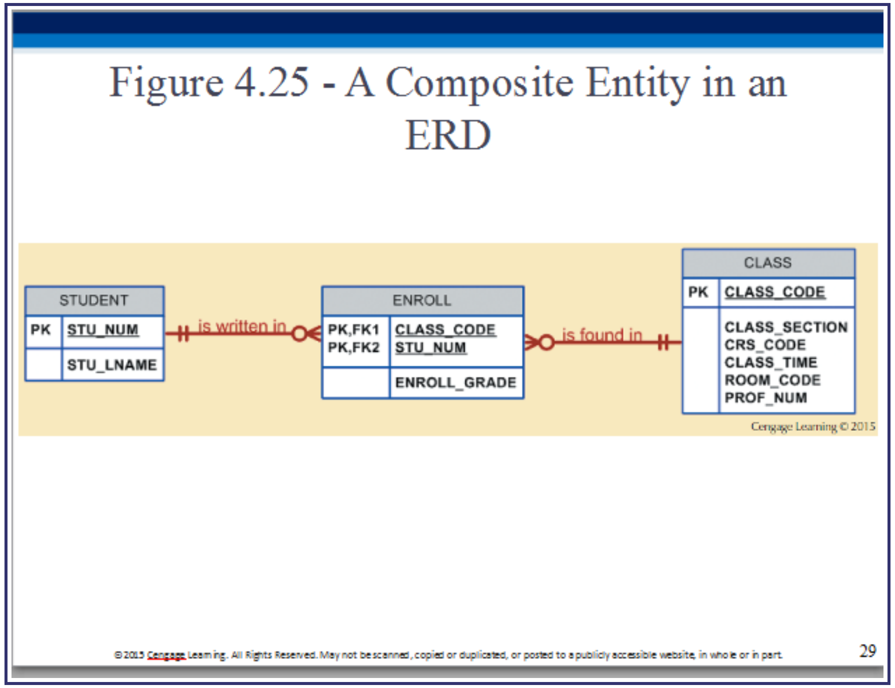
CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

28

ER diagrams (1) 2023-03-08, 7:41 PM



ER diagrams (1) 2023-03-08, 7:41 PM

Putting together an ERD

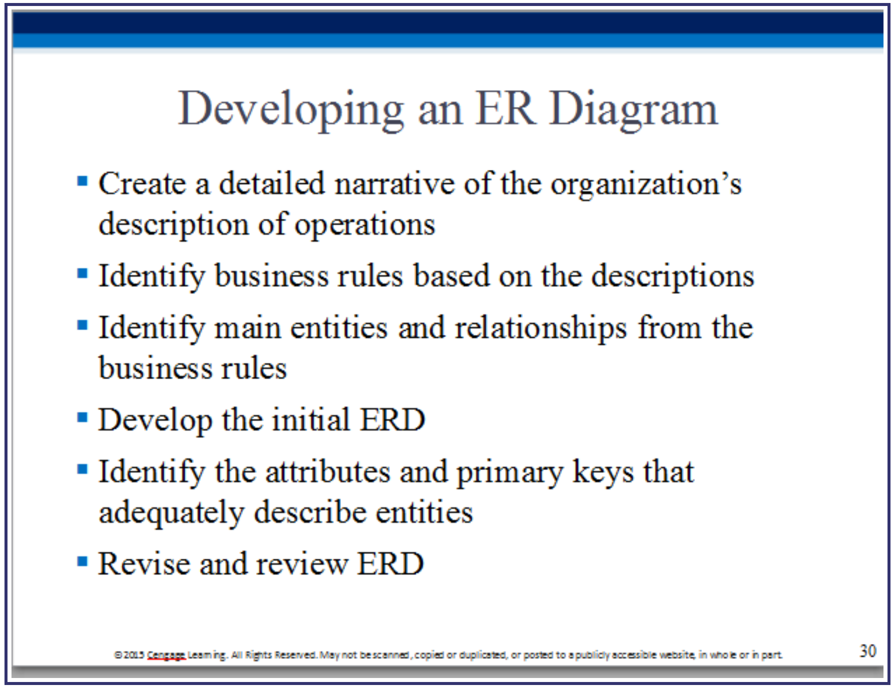
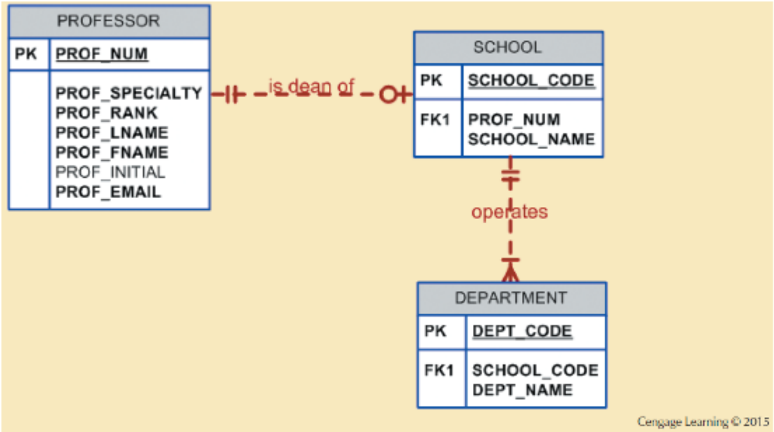
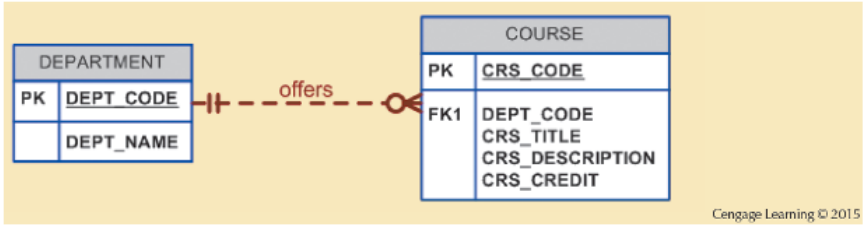


Figure 4.26 - The First Tiny College ERD Segment



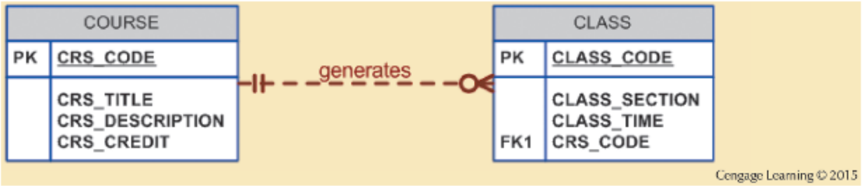
31

Figure 4.27 - The Second Tiny College ERD Segment



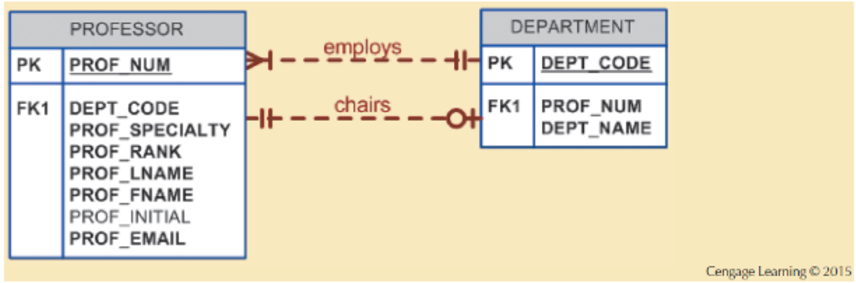
32

Figure 4.28 - The Third Tiny College ERD Segment



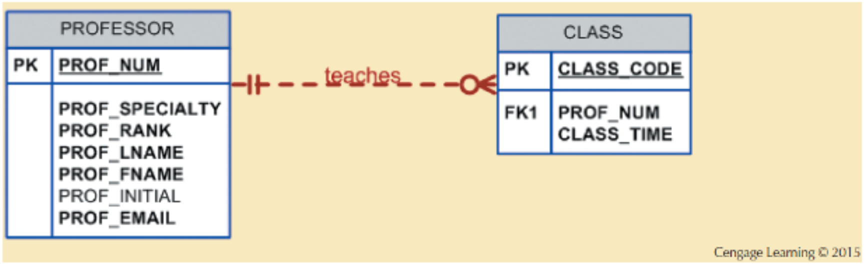
 33

Figure 4.29 - The Fourth Tiny College ERD Segment



 34

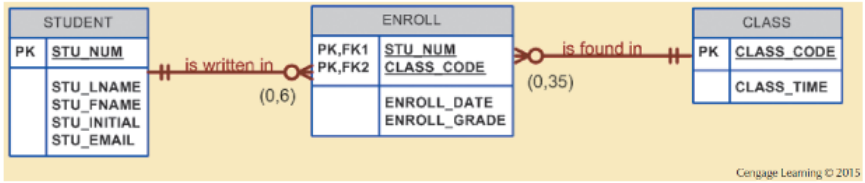
Figure 4.30 - The Fifth Tiny College ERD Segment



35

ER diagrams (1) 2023-03-08, 7:41 PM

Figure 4.31 - The Sixth Tiny College ERD Segment



36

Figure 4.32 - The Seventh Tiny College ERD Segment

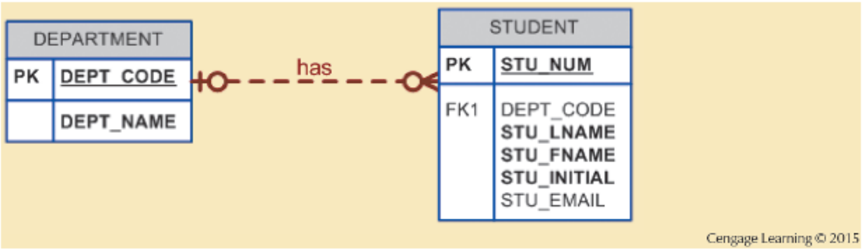


Figure 4.33 - The Eighth Tiny College ERD Segment

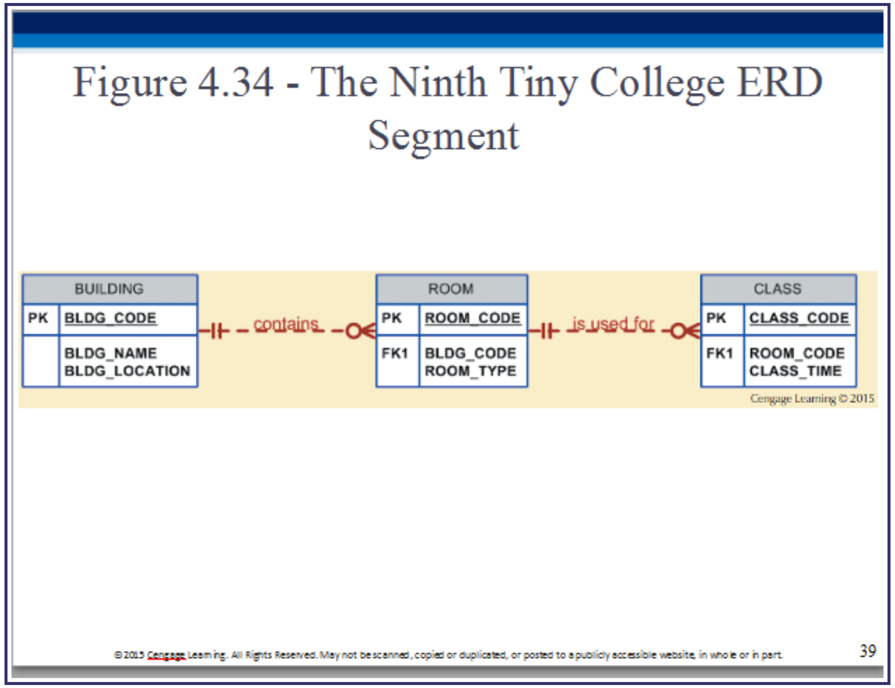


Figure 4.34 - The Ninth Tiny College ERD Segment
BUILDING
PK BLDG_CODE
BLDG_NAME
BLDG_LOCATION
contains
ROOM
PK ROOM_CODE
FK1 BLDG_CODE
ROOM_TYPE
is used for
CLASS
PK CLASS_CODE
FK1 ROOM_CODE
CLASS_TIME
Cengage Learning © 2015
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
39

List of entities, relationships, connectivities

Table 4.4 - Components of the ERM

ENTITY	RELATIONSHIP	CONNECTIVITY	ENTITY
SCHOOL	operates	1:M	DEPARTMENT
DEPARTMENT	has	1:M	STUDENT
DEPARTMENT	employs	1:M	PROFESSOR
DEPARTMENT	offers	1:M	COURSE
COURSE	generates	1:M	CLASS
PROFESSOR	is dean of	1:1	SCHOOL
PROFESSOR	chairs	1:1	DEPARTMENT
PROFESSOR	teaches	1:M	CLASS
PROFESSOR	advises	1:M	STUDENT
STUDENT	enrolls in	M:N	CLASS
BUILDING	contains	1:M	ROOM
ROOM	is used for	1:M	CLASS

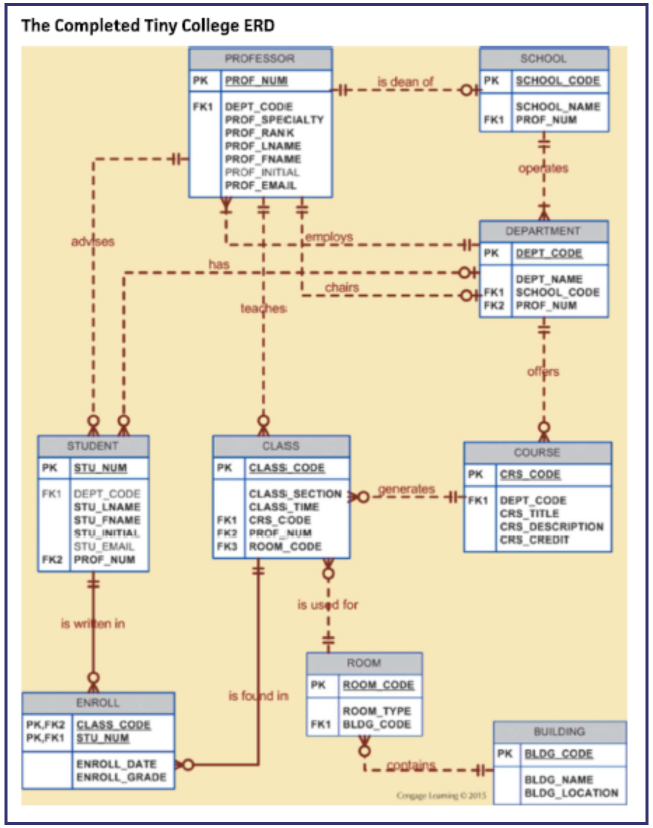
Note: ENROLL is the composite entity that implements the M:N relationship "STUDENT enrolls in CLASS."

Cengage Learning © 2015

ER diagrams (1) 2023-03-08, 7:41 PM

The full schema

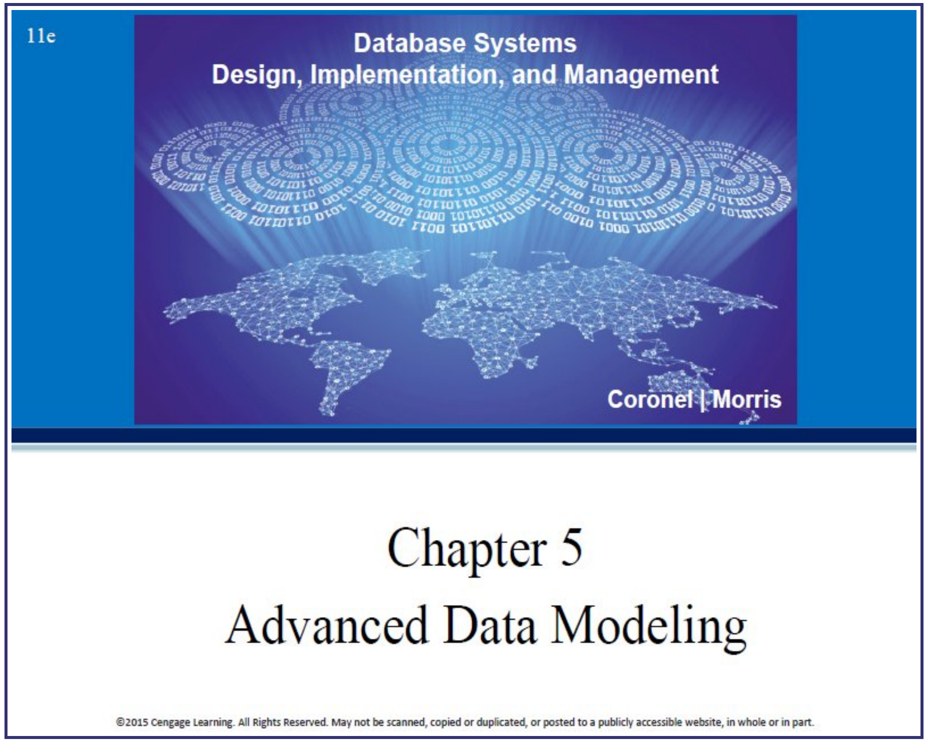
"All together now!"

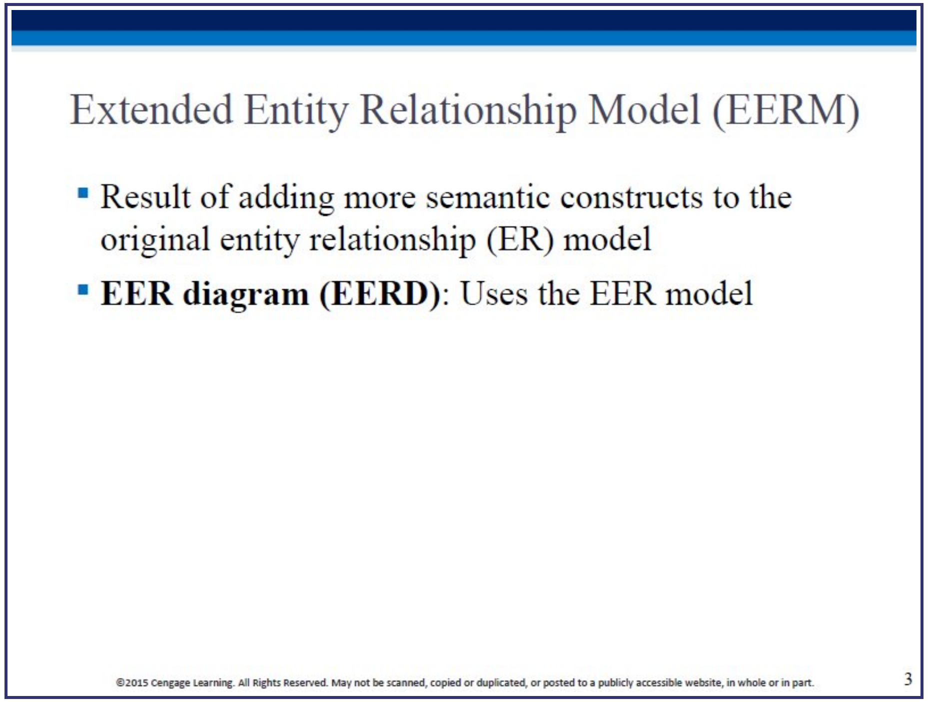


1/13 7:42:56 * * *

← →

Extended ER ("EER")


11e
Database Systems
Design, Implementation, and Management
Coronel | Morris
Chapter 5
Advanced Data Modeling
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.


Extended Entity Relationship Model (EERM)
Result of adding more semantic constructs to the original entity relationship (ER) model
EER diagram (EERD): Uses the EER model
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
3

Entity Supertypes and Subtypes

- **Entity supertype**: Generic entity type related to one or more entity subtypes
  - Contains common characteristics
- **Entity subtype**: Contains unique characteristics of each entity subtype
- Criteria to determine the usage
  - There must be different, identifiable kinds of the entity in the user's environment
  - The different kinds of instances should each have one or more attributes that are unique to that kind of instance

EER [Advanced Data Modeling] (1) 2023-03-08, 7:42 PM

Specialization Hierarchy

- Depicts arrangement of higher-level entity supertypes and lower-level entity subtypes
- Relationships are described in terms of "is-a" relationships
- Subtype exists within the context of a supertype
- Every subtype has one supertype to which it is directly related
- Supertype can have many subtypes

 5

EER [Advanced Data Modeling] (1) 2023-03-08, 7:42 PM

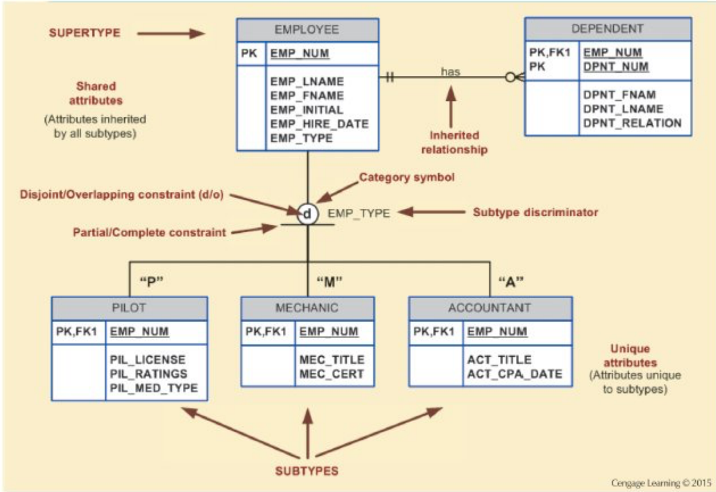
Specialization Hierarchy

- Provides the means to:
  - Support attribute inheritance
  - Define a special supertype attribute known as the subtype discriminator
  - Define disjoint/overlapping constraints and complete/partial constraints

 6

EER [Advanced Data Modeling] (1) 2023-03-08, 7:42 PM

Figure 5.2 - Specialization Hierarchy



7

Inheritance

- Enables an entity subtype to inherit attributes and relationships of the supertype
- All entity subtypes inherit their primary key attribute from their supertype
- At the implementation level, supertype and its subtype(s) maintain a 1:1 relationship
- Entity subtypes inherit all relationships in which supertype entity participates
- Lower-level subtypes inherit all attributes and relationships from its upper-level supertypes

8

EER [Advanced Data Modeling] (1) 2023-03-08, 7:42 PM

Subtype Discriminator

- Attribute in the supertype entity that determines to which entity subtype the supertype occurrence is related
- Default comparison condition is the equality comparison

Disjoint vs overlapping subtypes

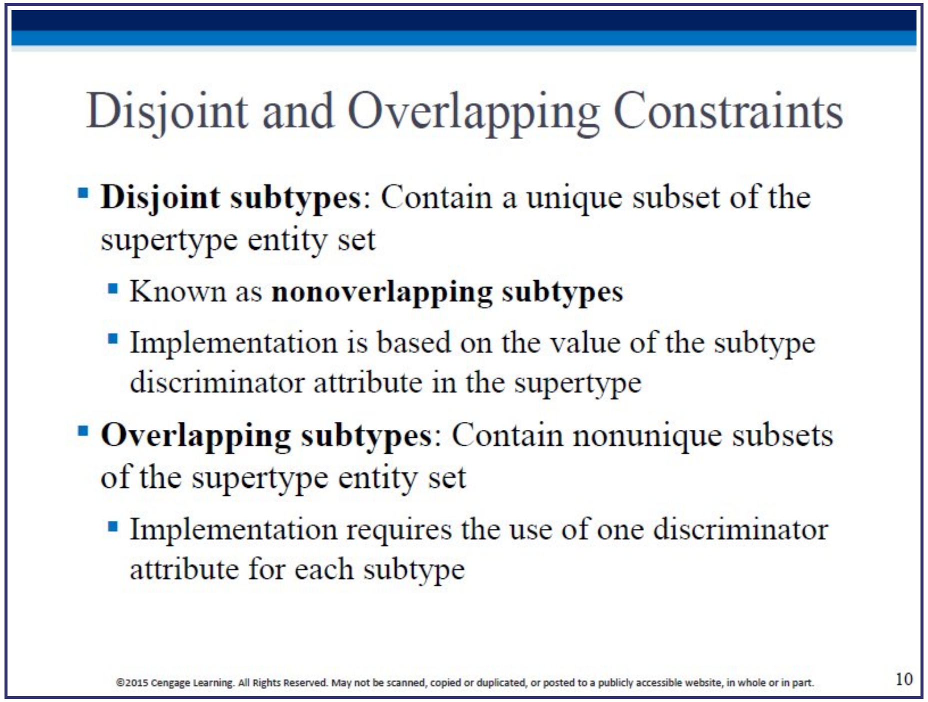
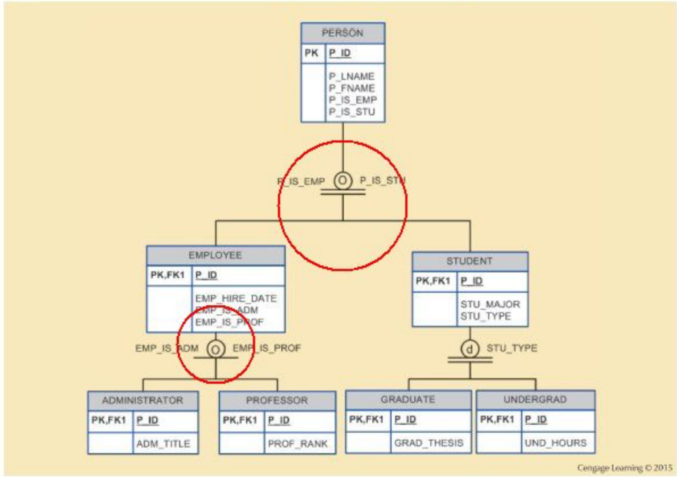


Figure 5.4 - Specialization Hierarchy with Overlapping Subtypes



 11

Completeness Constraint

- Specifies whether each supertype occurrence must also be a member of at least one subtype
- Types
  - **Partial completeness**: Not every supertype occurrence is a member of a subtype
  - **Total completeness**: Every supertype occurrence must be a member of any

 13

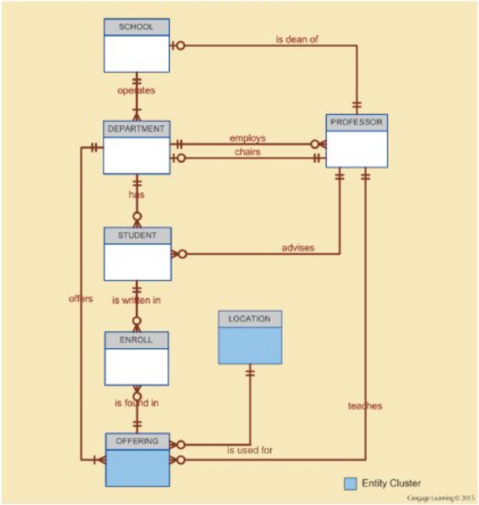
Table 5.2 - Specialization Hierarchy Constraint Scenarios

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial d	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique. o
Total d	Every supertype occurrence is a member of only one subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of at least one subtype. Subtype discriminators cannot be null. Subtype sets are not unique. o

 14

EER [Advanced Data Modeling] (1) 2023-03-08, 7:42 PM

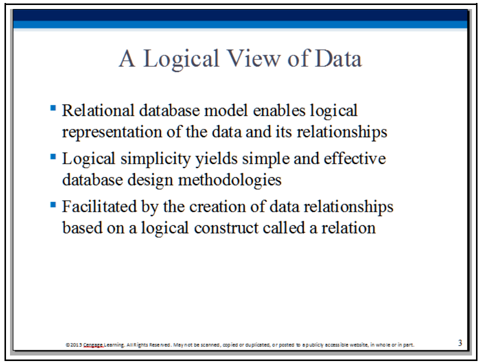
Figure 5.5 - Tiny College ERD Using Entity Clusters



17

Relational Modeling

Logical view: 'relation'



Relational tables

Table 3.1 - Characteristics of a Relational Table

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row (**tuple**) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each intersection of a row and column represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the **attribute domain**.
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or combination of attributes that uniquely identifies each row.

Cengage Learning © 2015

 4

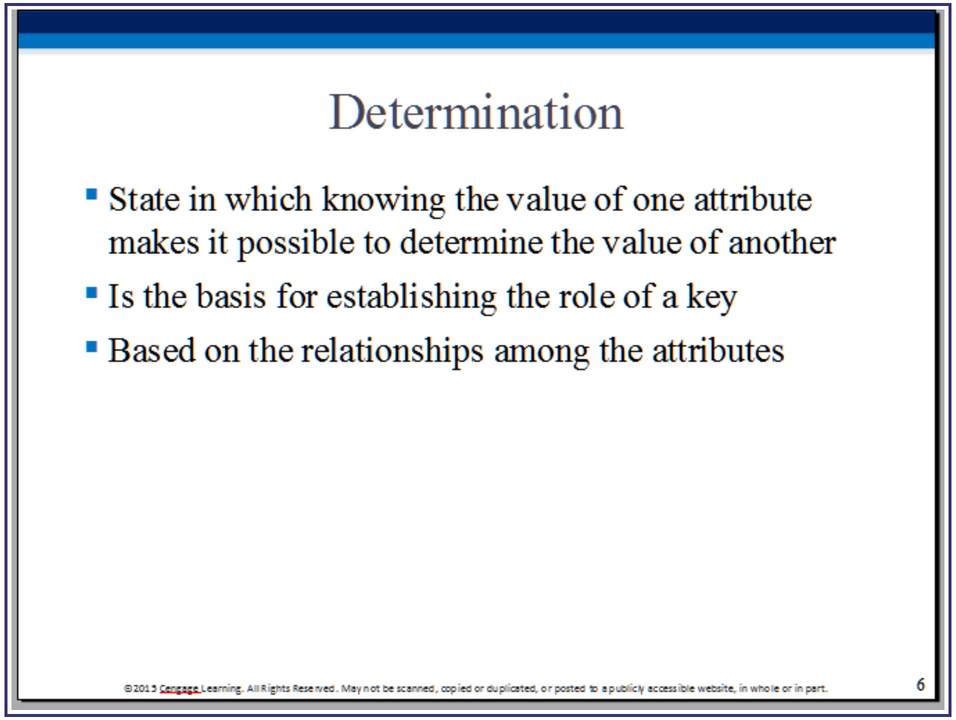
Keys

Keys

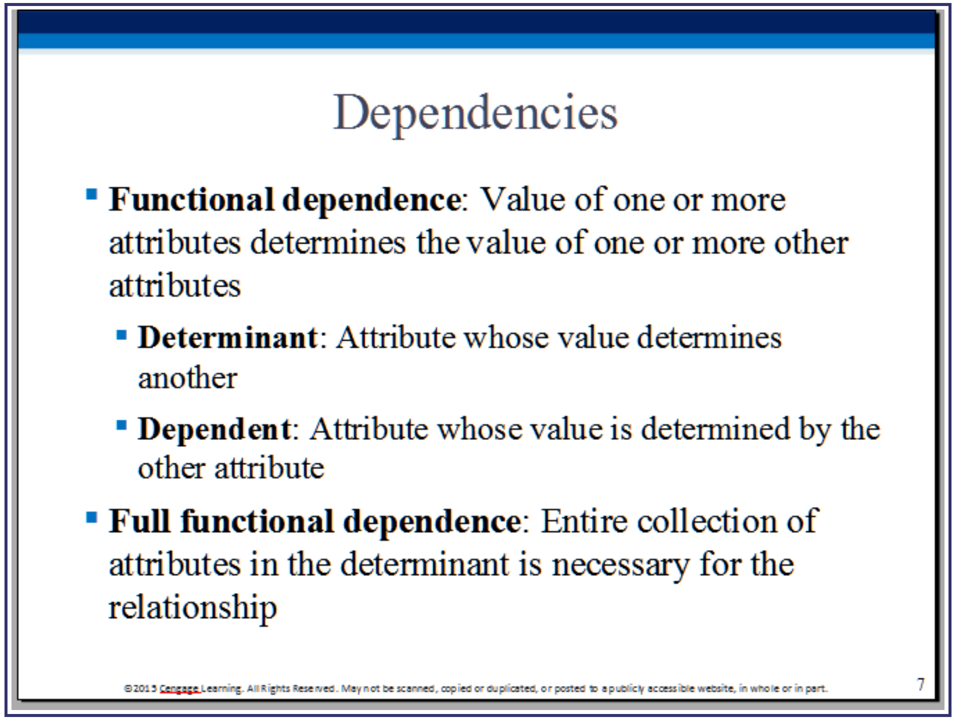
- Consist of one or more attributes that determine other attributes
- Used to:
  - Ensure that each row in a table is uniquely identifiable
  - Establish relationships among tables and to ensure the integrity of the data
- **Primary key (PK)**: Attribute or combination of attributes that uniquely identifies any given row

 5

"determines"



Determinants determine dependents [via] dependencies :)



'Full' functional dependence is a "good" thing.

Functional dependency

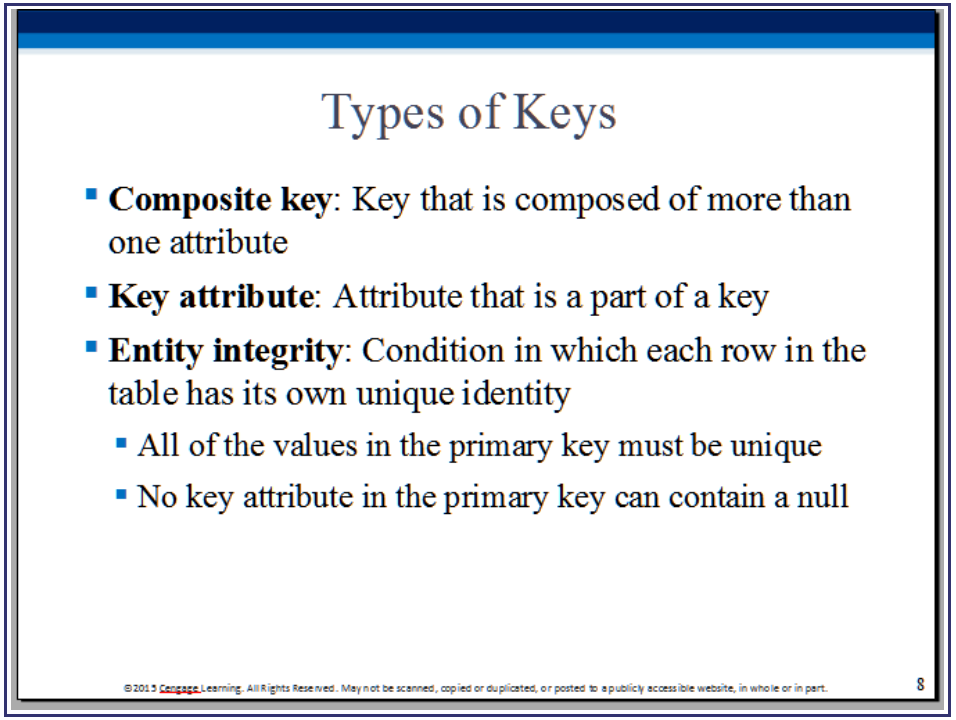
STU_ID[determinant] ->[functionally determines] STU_LNAME[dependent]

STU_ID,STU_LNAME -> GPA is NOT a 'full functional dependency' because the determinant contains an extra (unwanted) attr (STU_LNAME)

STU_LNAME,STU_FNAME -> GPA is a 'full functional dependency' (assuming lastname,firstname is unique)

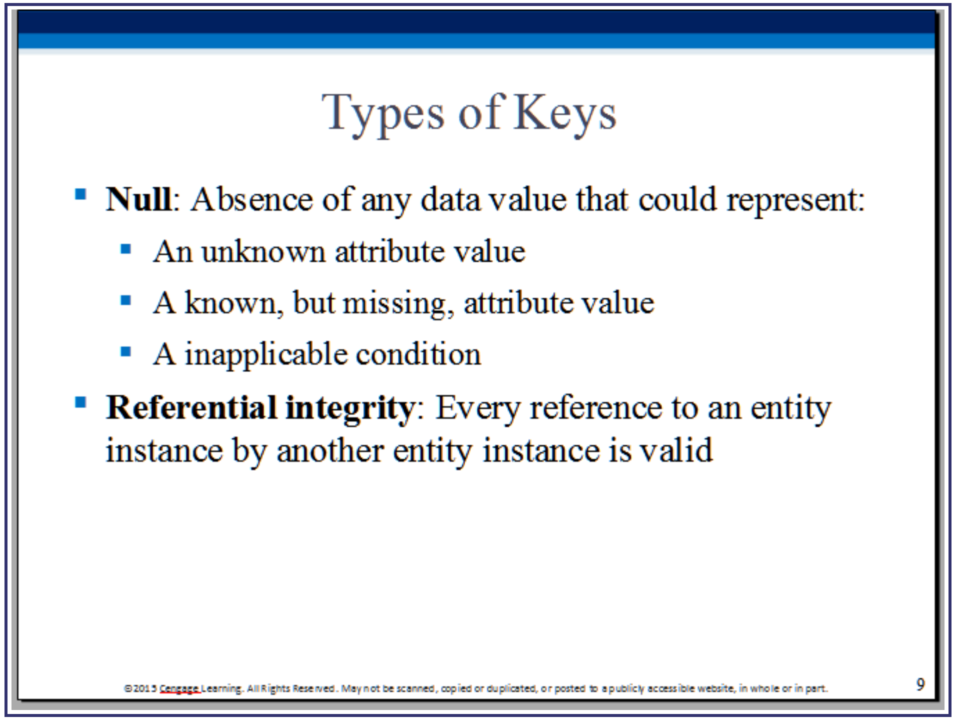
Solemnly swear: "The key, the whole key, and nothing but the key, so help me Codd." :) :)

Composite key; entity integrity



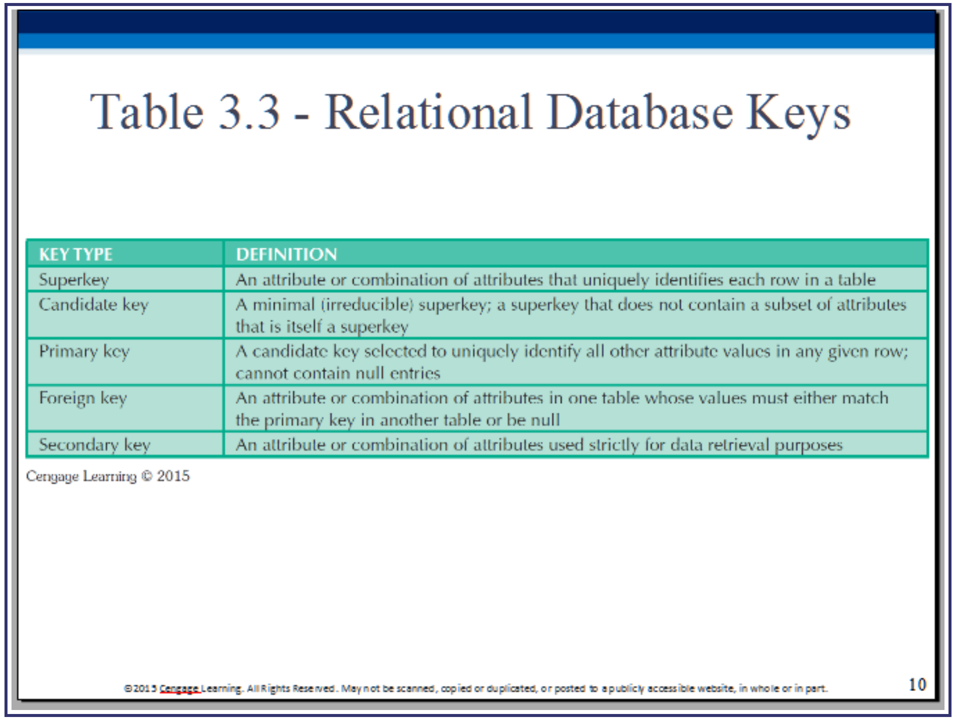
A table 'cannot not' have entity integrity!!

Nulls; referential integrity



Whereas entity integrity has to do with a single table, referential integrity relates to two tables (loosely, 'don't allow invalid pointers').

Types (categories) of keys



KEY TYPE	DEFINITION
Superkey	An attribute or combination of attributes that uniquely identifies each row in a table
Candidate key	A minimal (irreducible) superkey; a superkey that does not contain a subset of attributes that is itself a superkey
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row; cannot contain null entries
Foreign key	An attribute or combination of attributes in one table whose values must either match the primary key in another table or be null
Secondary key	An attribute or combination of attributes used strictly for data retrieval purposes

Keys: many types

* primary (foreign) keys are a subset of candidate keys are a subset of superkeys (note - superkeys could be 'wasteful', ie. contain superfluous, non-needed attrs)

* simple keys vs compound keys vs composite keys

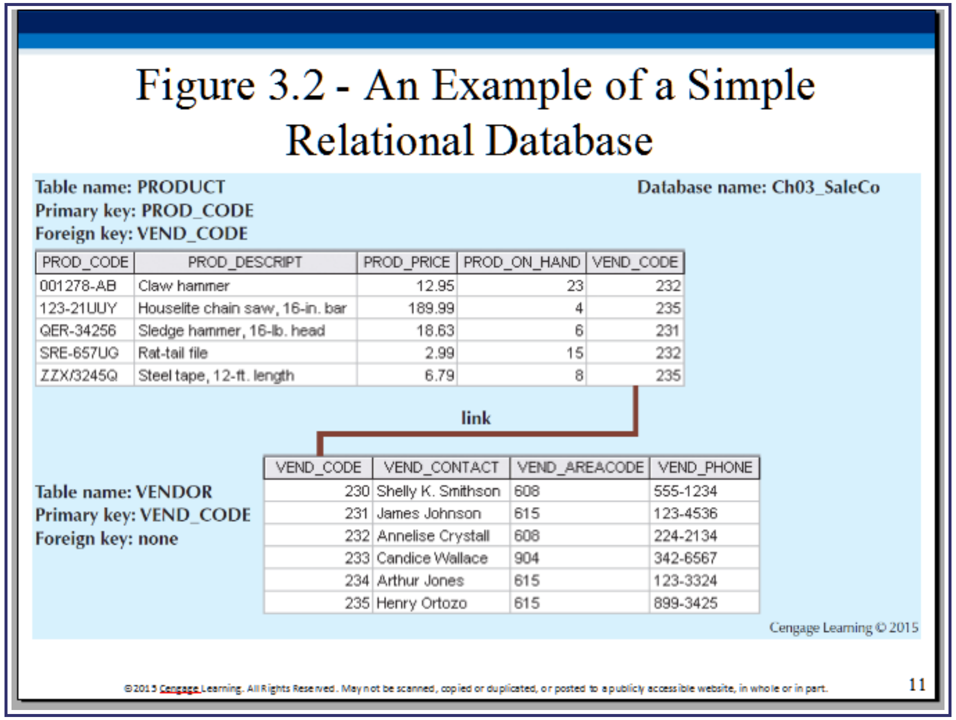
* natural keys - keys that are created from real-world entities (eg. for a US resident, their SSN could be a natural key)

* surrogate keys (just make up brand new unique keys)

* secondary, or 'alternate' keys

You can read a bit more keys here.

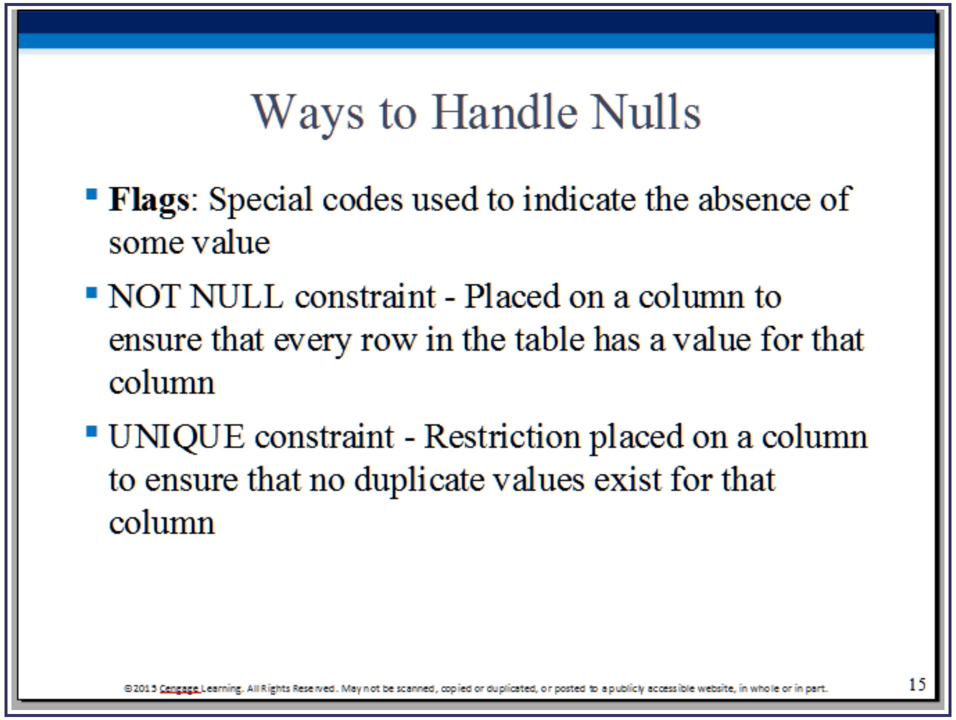
Example relation



PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON_HAND	VEND_CODE
001278-AB	Claw hammer	12.95	23	232
123-21UUY	Houselite chain saw, 16-in. bar	189.99	4	235
QER-34256	Sledge hammer, 16-lb. head	18.63	6	231
SRE-657UG	Rat-tail file	2.99	15	232
ZZX/3245Q	Steel tape, 12-ft. length	6.79	8	235

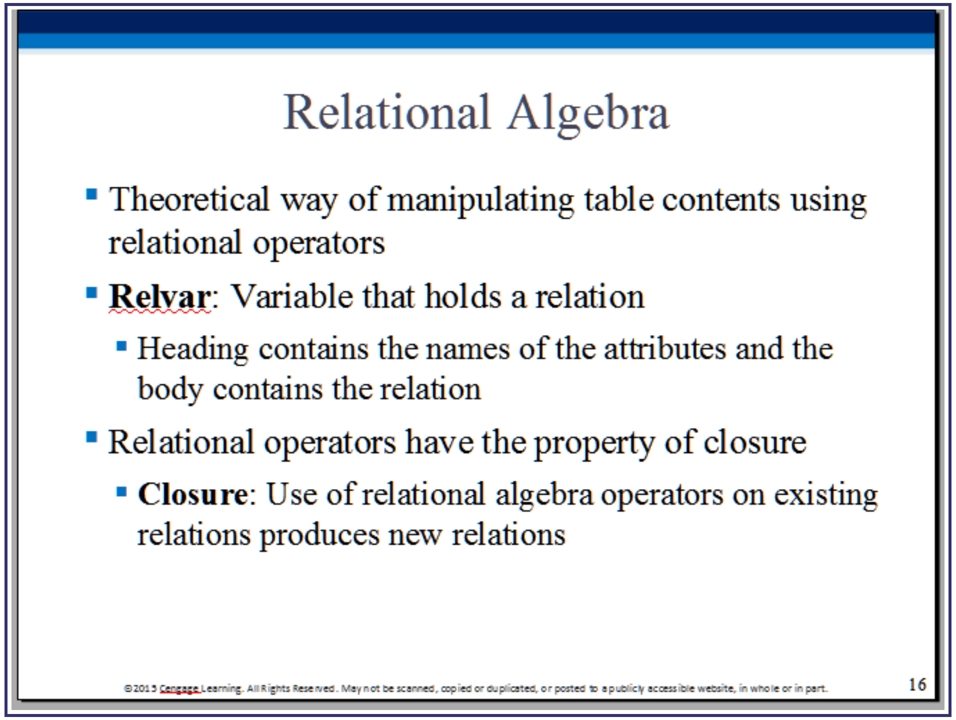
VEND_CODE	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
230	Shelly K. Smithson	608	555-1234
231	James Johnson	615	123-4536
232	Annelise Crystall	608	224-2134
233	Candice Wallace	904	342-6567
234	Arthur Jones	615	123-3324
235	Henry Ortozo	615	899-3425

Nulls - avoid where possible!



In RL, NULLs can't be entirely avoided (look here, for 'interpreted as any of the following').

Relational 'algebra' [fun with one, two or more tables]



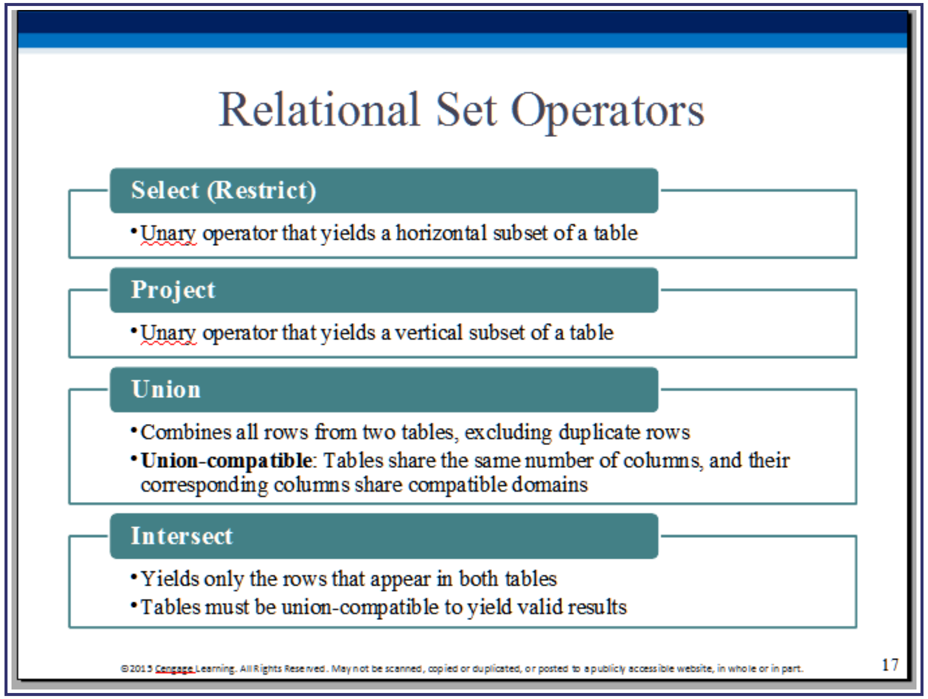
What if a table were a datatype (similar to an int, Vec3D, ComplexNumber, etc)?! Specifically, what operations could be perform on them (eg. similar to addition, square root on doubles)?!

Operations on tables [table(s) in, table out, ie. "closure"]

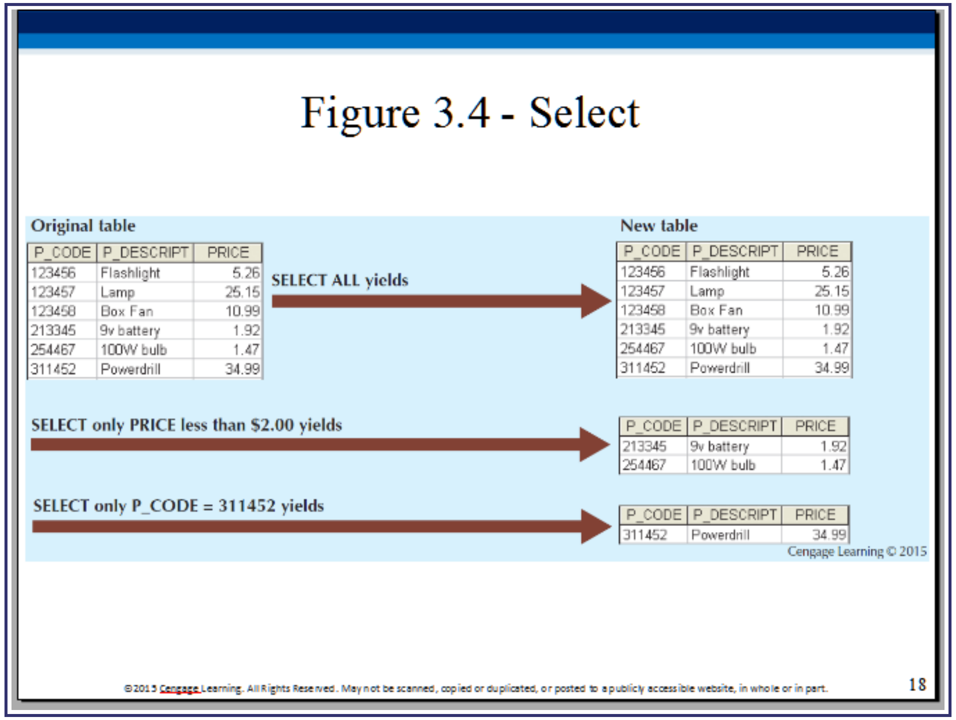
There are (only) EIGHT 'relational set operators' (defined by Ed Codd, at IBM, in 1970), which are all used to operate ("perform relational algebra") on tables: Select, Project, Union, Intersect, Difference, Product, Join, Divide.

This is no exaggeration: these operators are the basis for SQL and the entire relational DB industry!

SELECT; PROJECT; UNION; INTERSECT



SELECT [outputs a subset of rows]



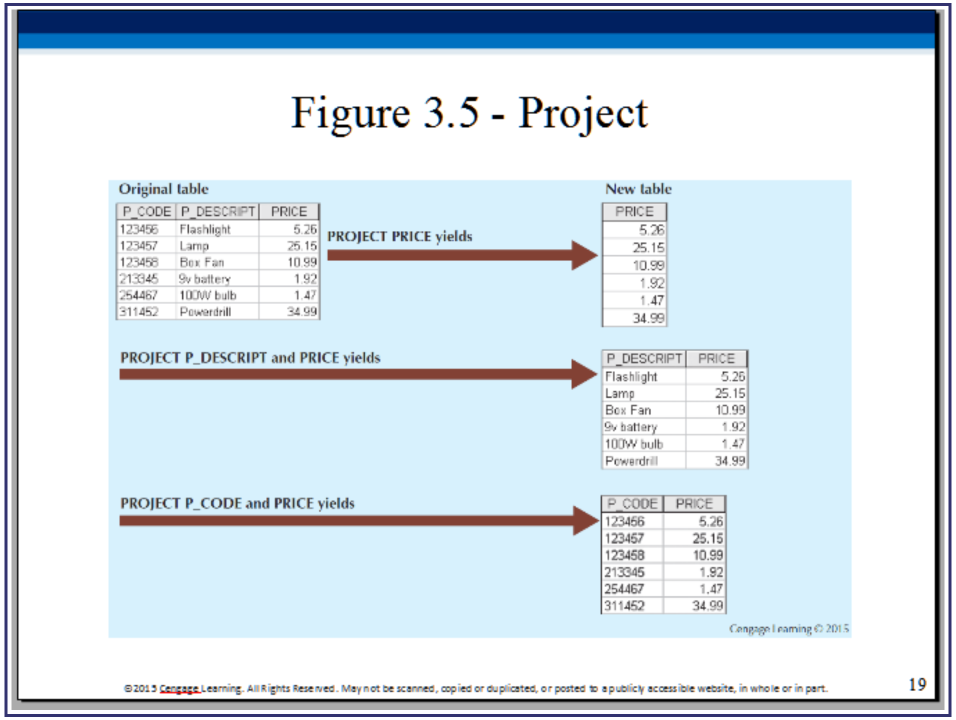
P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

P_CODE	P_DESCRIPT	PRICE
213345	9v battery	1.92
254467	100W bulb	1.47

P_CODE	P_DESCRIPT	PRICE
311452	Powerdrill	34.99

Relational Modeling (1) 2023-03-08, 8:01 PM

PROJECT [outputs a subset of cols]



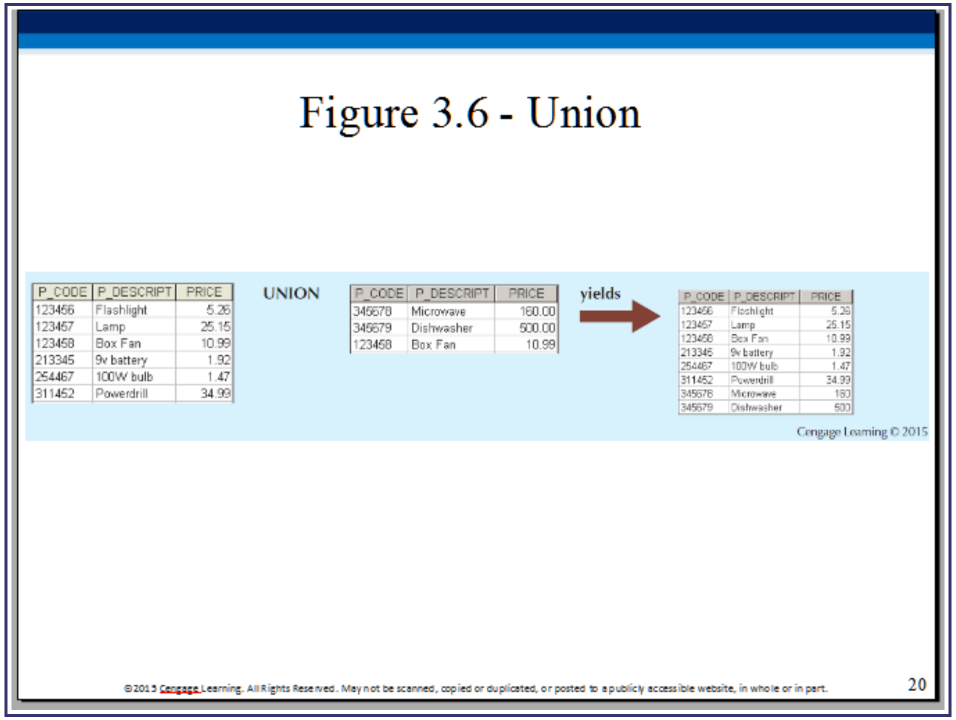
P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

PRICE
5.26
25.15
10.99
1.92
1.47
34.99

P_DESCRIPT	PRICE
Flashlight	5.26
Lamp	25.15
Box Fan	10.99
9v battery	1.92
100W bulb	1.47
Powerdrill	34.99

P_CODE	PRICE
123456	5.26
123457	25.15
123458	10.99
213345	1.92
254467	1.47
311452	34.99

UNION [eqvt to 'cat a b > c']

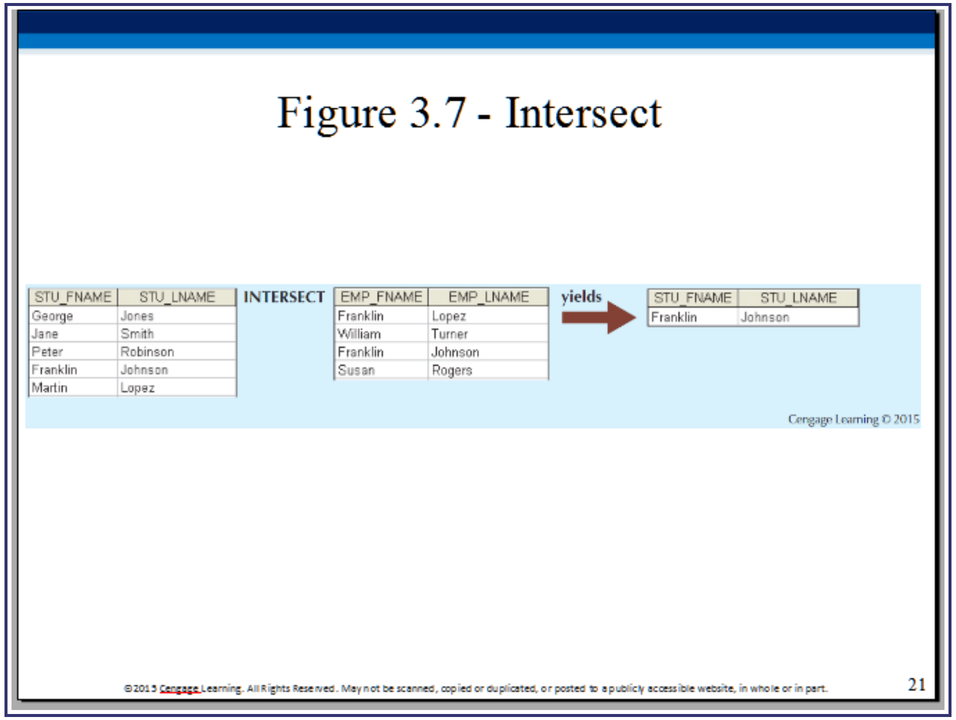


P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

P_CODE	P_DESCRIPT	PRICE
345678	Microwave	160.00
345679	Dishwasher	500.00
123458	Box Fan	10.99

P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99
345678	Microwave	160
345679	Dishwasher	500

INTERSECT [rows common to a and b]

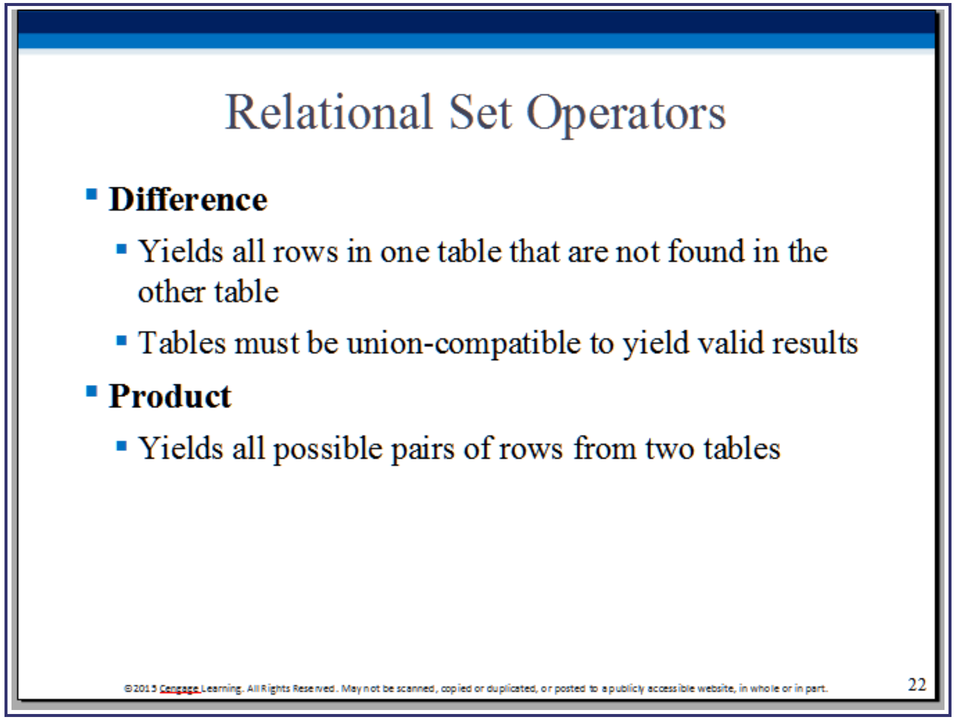


STU_FNAME	STU_LNAME
George	Jones
Jane	Smith
Peter	Robinson
Franklin	Johnson
Martin	Lopez

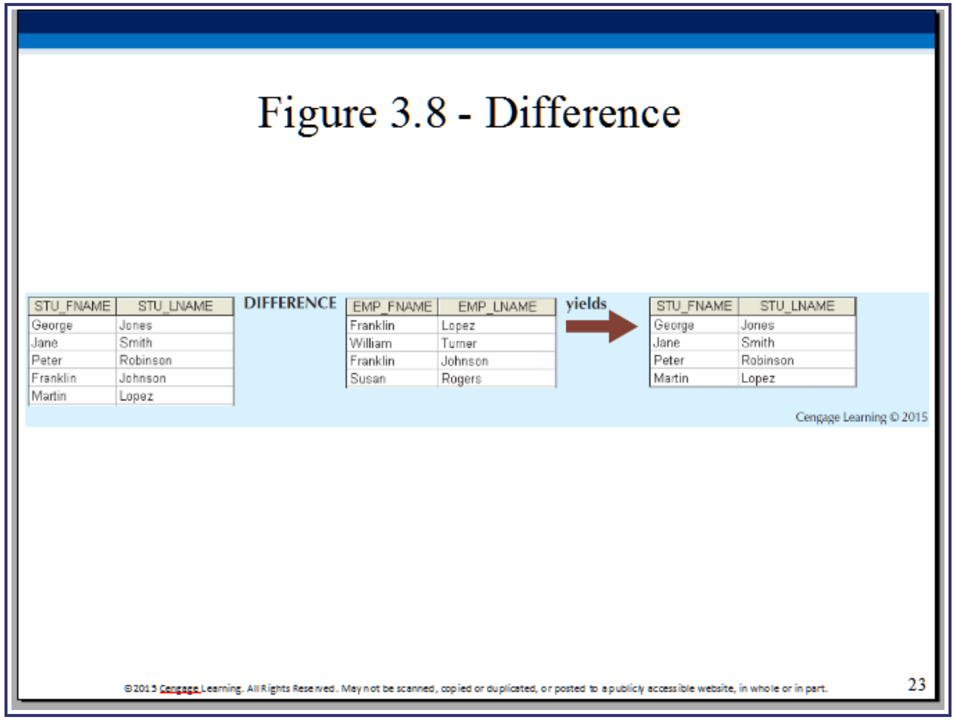
EMP_FNAME	EMP_LNAME
Franklin	Lopez
William	Turner
Franklin	Johnson
Susan	Rogers

STU_FNAME	STU_LNAME
Franklin	Johnson

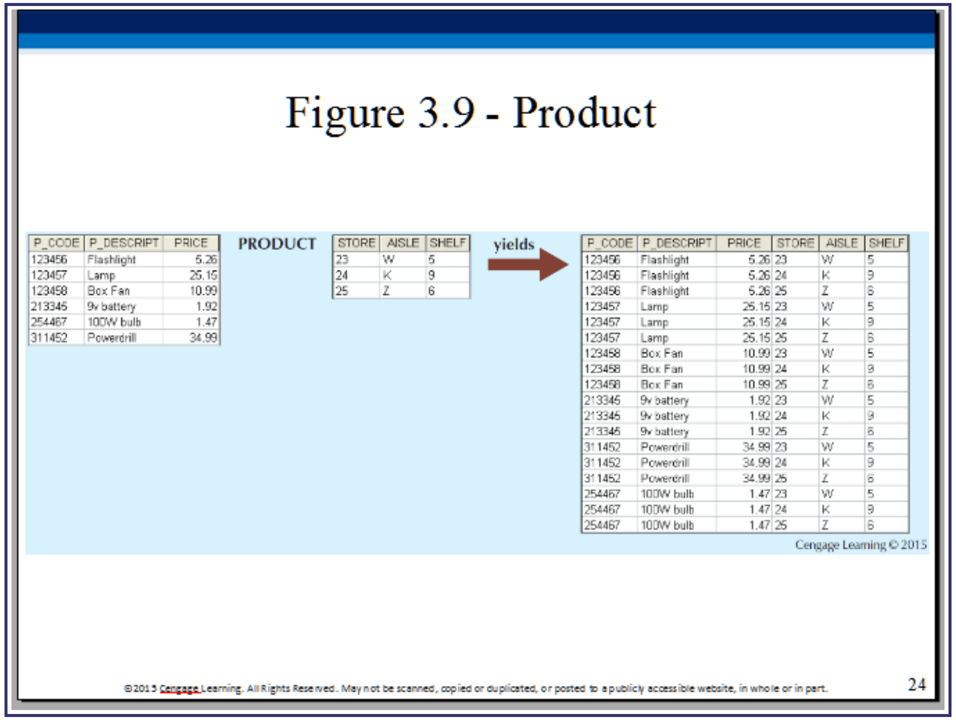
Difference; Product



Difference [a - b]



Product [multiply rows, add columns]

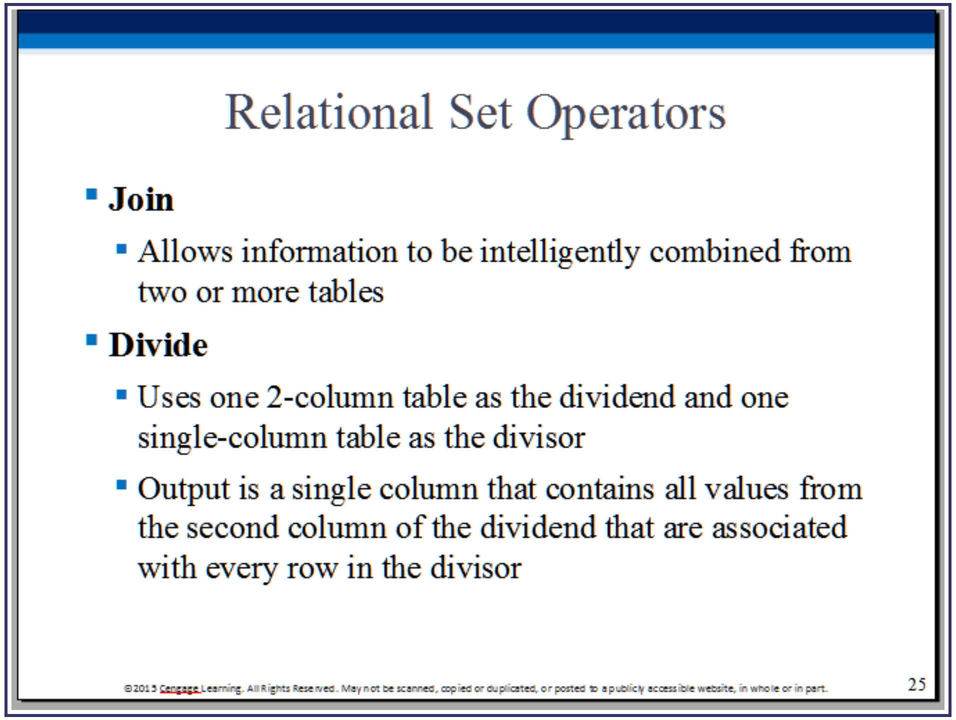


P_CODE	P_DESCRIPT	PRICE
123456	Flashlight	5.26
123457	Lamp	25.15
123458	Box Fan	10.99
213345	9v battery	1.92
254467	100W bulb	1.47
311452	Powerdrill	34.99

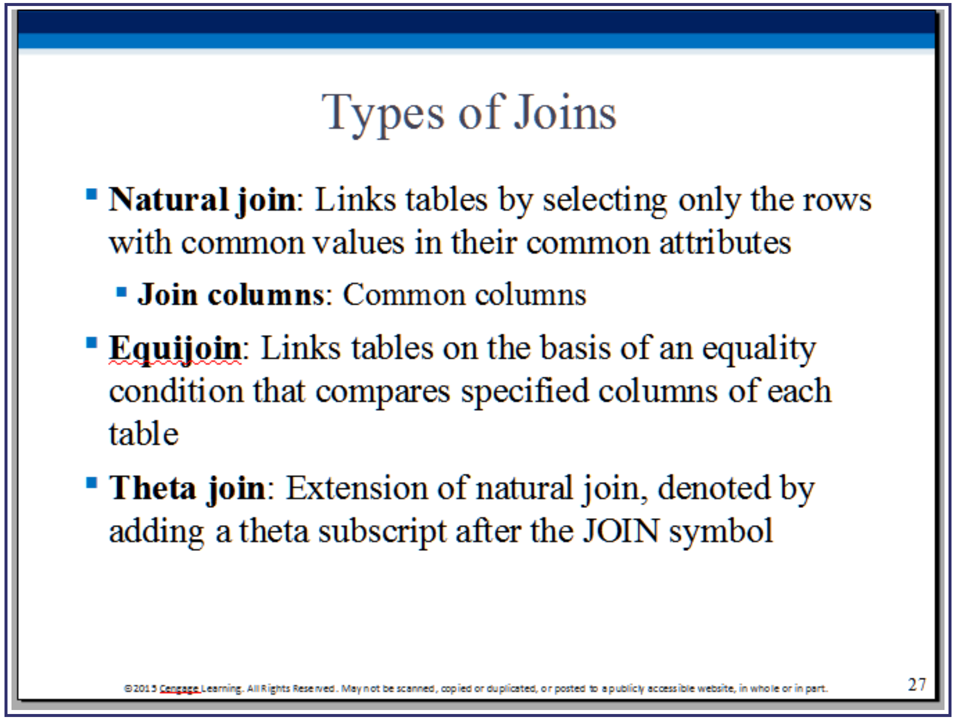
STORE	AISLE	SHELF
23	W	5
24	K	9
25	Z	6

P_CODE	P_DESCRIPT	PRICE	STORE	AISLE	SHELF
123456	Flashlight	5.26	23	W	5
123456	Flashlight	5.26	24	K	9
123456	Flashlight	5.26	25	Z	6
123457	Lamp	25.15	23	W	5
123457	Lamp	25.15	24	K	9
123457	Lamp	25.15	25	Z	6
123458	Box Fan	10.99	23	W	5
123458	Box Fan	10.99	24	K	9
123458	Box Fan	10.99	25	Z	6
213345	9v battery	1.92	23	W	5
213345	9v battery	1.92	24	K	9
213345	9v battery	1.92	25	Z	6
311452	Powerdrill	34.99	23	W	5
311452	Powerdrill	34.99	24	K	9
311452	Powerdrill	34.99	25	Z	6
254467	100W bulb	1.47	23	W	5
254467	100W bulb	1.47	24	K	9
254467	100W bulb	1.47	25	Z	6

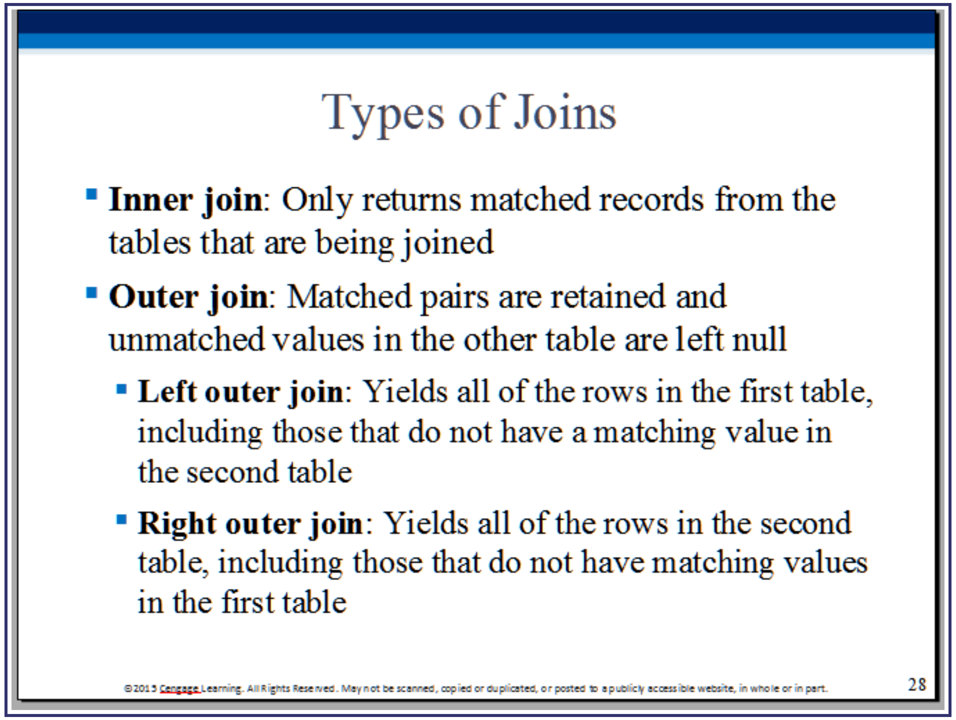
JOIN (several kinds); DIVIDE (?!)



JOIN



JOIN [cont'd]



Tables to illustrate JOIN operations

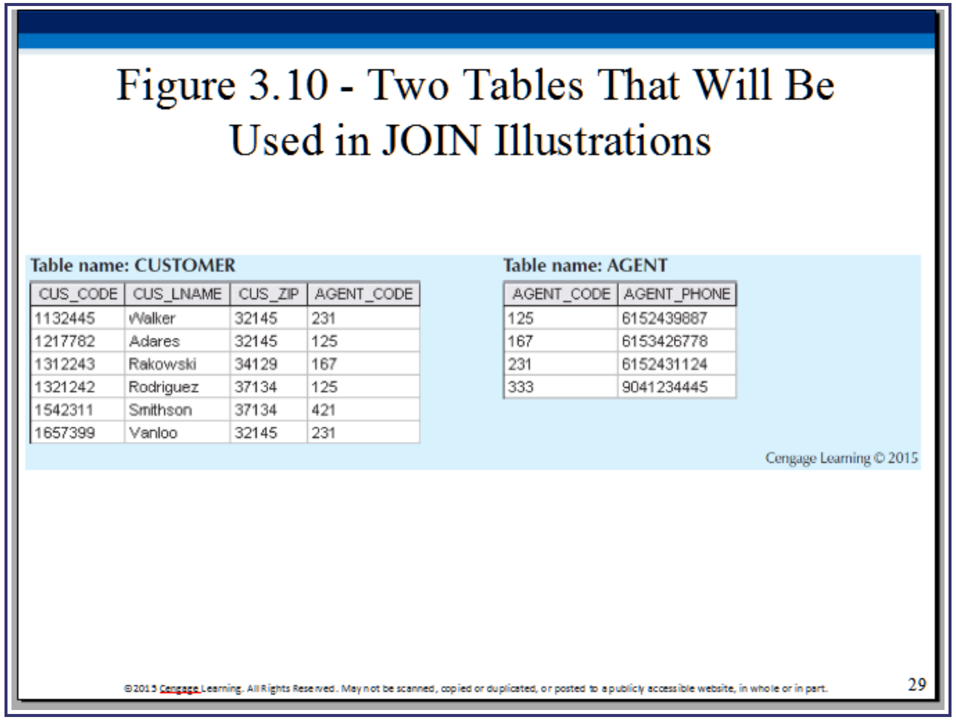


Figure 3.10 - Two Tables That Will Be Used in JOIN Illustrations

Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE
1132445	Walker	32145	231
1217782	Adares	32145	125
1312243	Rakowski	34129	167
1321242	Rodriguez	37134	125
1542311	Smithson	37134	421
1657399	Vanloo	32145	231

Table name: AGENT

AGENT_CODE	AGENT_PHONE
125	6152439887
167	6153426778
231	6152431124
333	9041234445

29

Natural join

A natural join links tables by selecting from two tables, only those rows that have common (identical) values for common attributes.

These three steps result in a natural join: create product, select, project.

Natural join: product

Cartesian product of the two tables (product of rows, juxtaposition of columns):

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1132445	Walker	32145	231	125	6152439887
1132445	Walker	32145	231	167	6153426778
1132445	Walker	32145	231	231	6152431124
1132445	Walker	32145	231	333	9041234445
1217782	Adares	32145	125	125	6152439887
1217782	Adares	32145	125	167	6153426778
1217782	Adares	32145	125	231	6152431124
1217782	Adares	32145	125	333	9041234445
1312243	Rakowski	34129	167	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1312243	Rakowski	34129	167	231	6152431124
1312243	Rakowski	34129	167	333	9041234445
1321242	Rodriguez	37134	125	125	6152439887
1321242	Rodriguez	37134	125	167	6153426778
1321242	Rodriguez	37134	125	231	6152431124
1321242	Rodriguez	37134	125	333	9041234445
1542311	Smithson	37134	421	125	6152439887
1542311	Smithson	37134	421	167	6153426778
1542311	Smithson	37134	421	231	6152431124
1542311	Smithson	37134	421	333	9041234445
1657399	Vanloo	32145	231	125	6152439887
1657399	Vanloo	32145	231	167	6153426778
1657399	Vanloo	32145	231	231	6152431124
1657399	Vanloo	32145	231	333	9041234445

Natural join: select

Select only rows with identical values in the common (joining) columns:

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124

Natural join: project

Project away, ie. remove one of the two duplicate columns:

CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	6152439887
1321242	Rodriguez	37134	125	6152439887
1312243	Rakowski	34129	167	6153426778
1132445	Walker	32145	231	6152431124
1657399	Vanloo	32145	231	6152431124

Result (the table above): natural join.

Left outer join

Output all rows of the left (CUSTOMER) table, including ones for which there are no matching values in the join column in the other (AGENT) table:

CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124
1542311	Smithson	37134	421		

Note that an outer join is an "inner join plus" [it is NOT an opposite of inner join].

Right outer join, full outer join

Output all rows of the right (AGENT) table, including ones for which there are no matching values in the join column in the other (CUSTOMER) table:

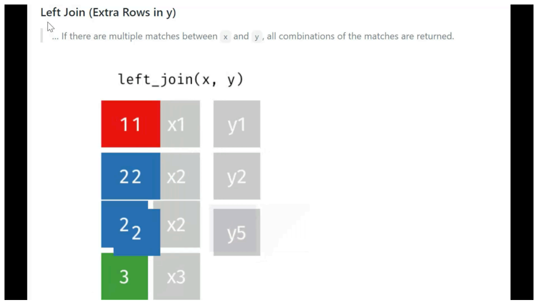
CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
1217782	Adares	32145	125	125	6152439887
1321242	Rodriguez	37134	125	125	6152439887
1312243	Rakowski	34129	167	167	6153426778
1132445	Walker	32145	231	231	6152431124
1657399	Vanloo	32145	231	231	6152431124
				333	9041234445

Cengage Learning © 2015

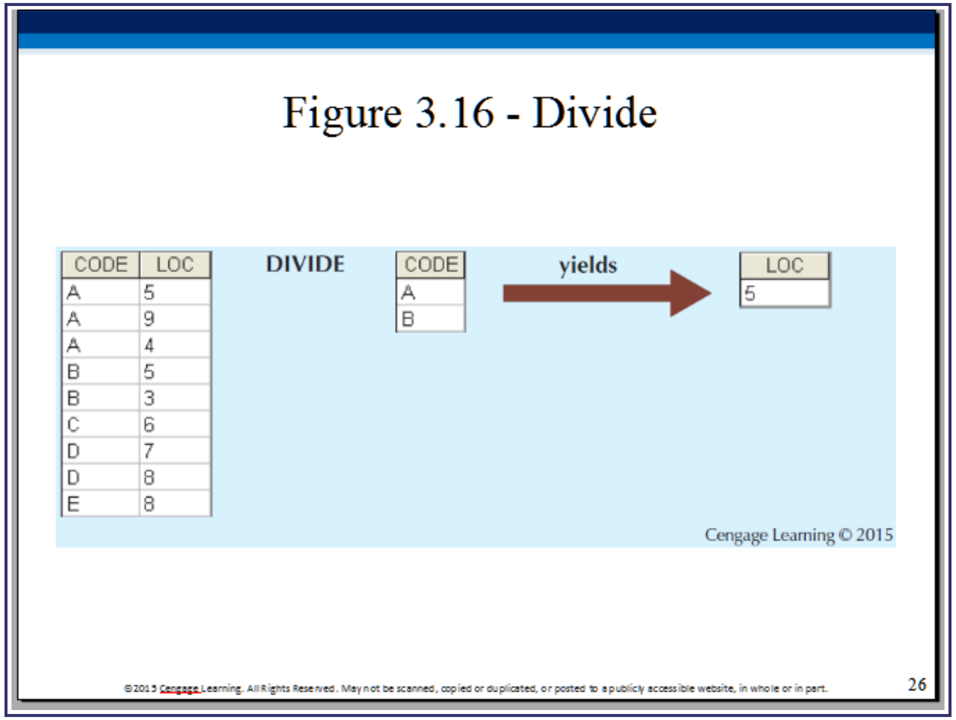
Outer joins are useful in exposing missing information [in our example, customers who don't seem to have an agent, and, agents who don't seem to have customers].

A 'full outer join' is a union of left outer join and right outer join - output all the rows from both tables, including ones for which there are no matches in the other table - this could result in nulls on the left side of some rows, as well as nulls on the right side of others.

This clip shows the various types of joins [thanks, Yash Gupta, for sending this]:

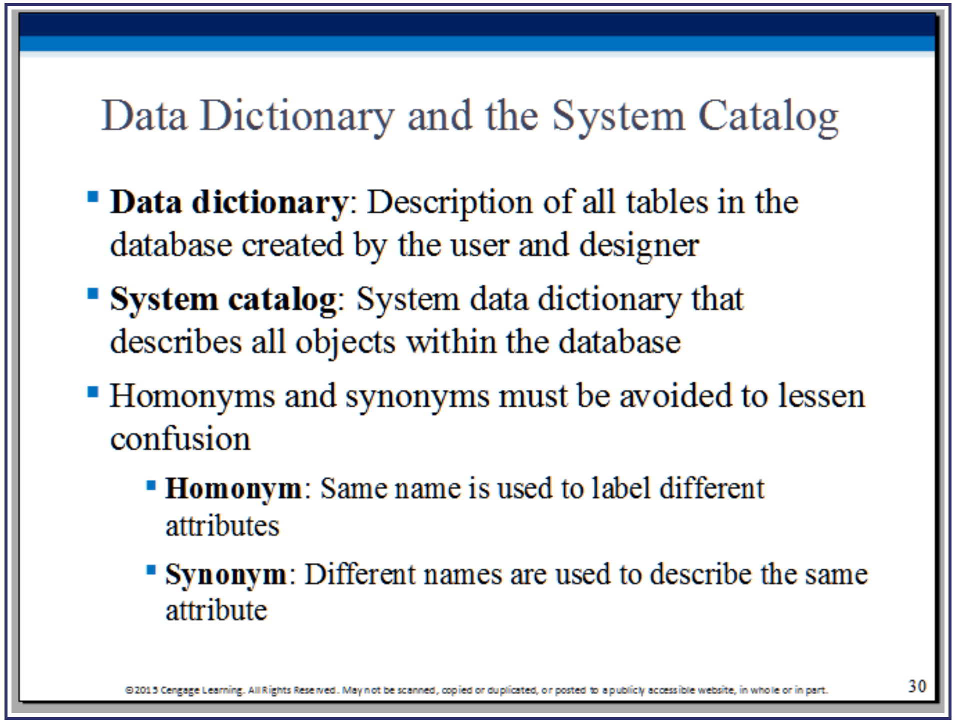


DIVIDE



We're dividing by A and B in the divisor (bottom) table. There's (A,5) and (B,5) in the dividend (top) table, so we output 5 as the result; if the dividend were to contain (A,9) and (B,9) also, then we'd output 5 9 as the result.

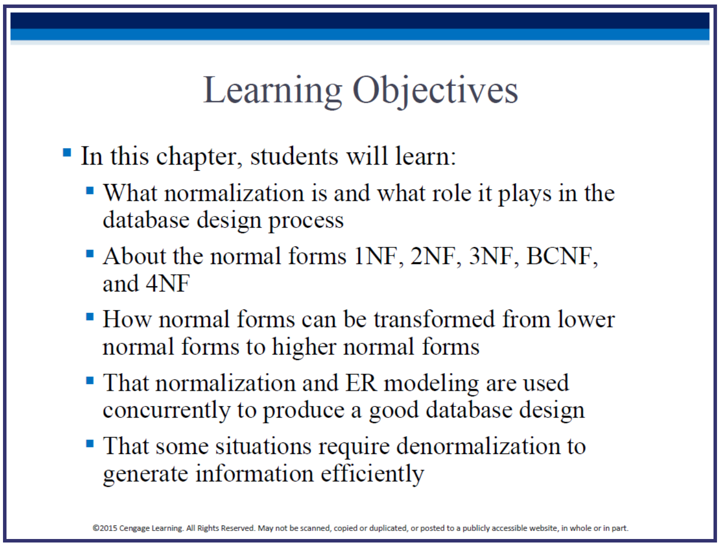
Dictionaries [hold metadata]



A data dictionary is metadata about tables (only); a system catalog, that includes (is a supeset of, although confusingly, the two are conflated in RL) the data dictionary, and more.

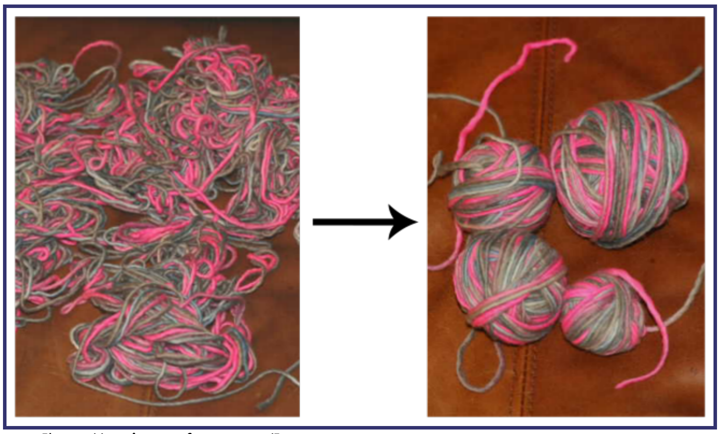
(De/)Normalization

On tap


Learning Objectives
In this chapter, students will learn:
What normalization is and what role it plays in the database design process
About the normal forms 1NF, 2NF, 3NF, BCNF, and 4NF
How normal forms can be transformed from lower normal forms to higher normal forms
That normalization and ER modeling are used concurrently to produce a good database design
That some situations require denormalization to generate information efficiently
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

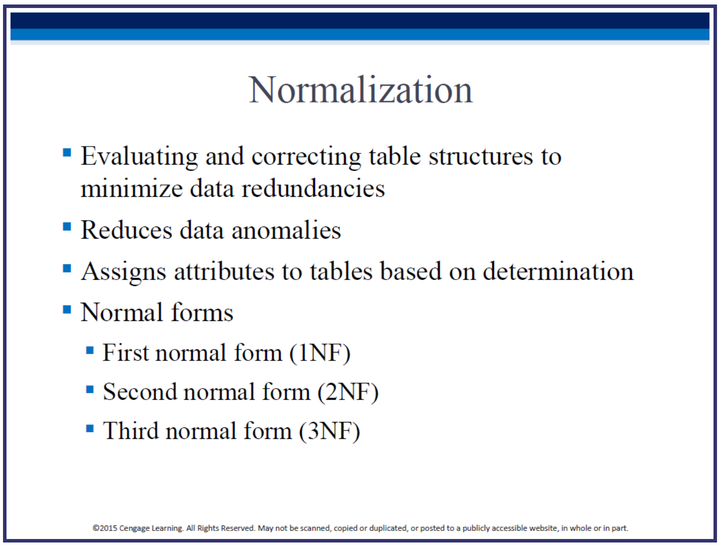
The goal of normalization

Loosely speaking:

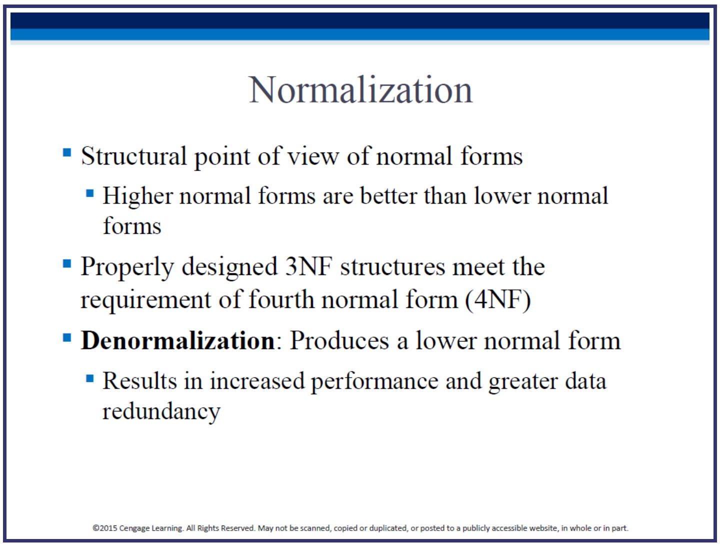


[http://rovingcrafters.com/]

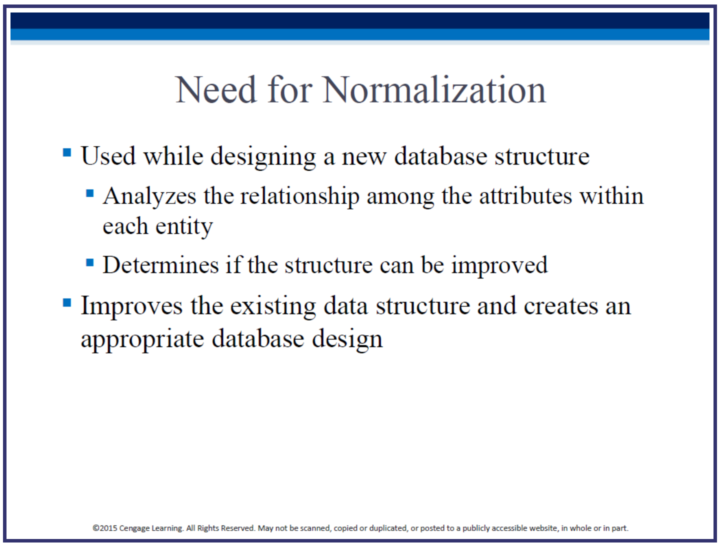
Goal: reduce redundancies, anomalies



Higher normal forms → cleaner designs



Normalization is a design step



A construction company db

Employees of the construction company work on projects. Each employee has an ID, name, job title and corresponding hourly rate.

Each project has a number, name and assigned employees. An employee can be assigned to more than one project.

The company bills clients for projects, based on hours worked by employees.

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.8
		101	John G. News *	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

Report

The construction company periodically generates a report like so:

PROJECT NUMBER	PROJECT NAME	EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CLASS	CHARGE/ HOUR	HOURS BILLED	TOTAL CHARGE
15	Evergreen	103	June E. Arbough	Elec. Engineer	$ 84.50	23.8	$ 2,011.10
		101	John G. News	Database Designer	$105.00	19.4	$ 2,037.00
		105	Alice K. Johnson *	Database Designer	$105.00	35.7	$ 3,748.50
		106	William Smithfield	Programmer	$ 35.75	12.6	$ 450.45
		102	David H. Senior	Systems Analyst	$ 96.75	23.8	$ 2,302.65
				Subtotal			**$10,549.70**
18	Amber Wave	114	Annelise Jones	Applications Designer	$ 48.10	24.6	$ 1,183.26
		118	James J. Frommer	General Support	$ 18.36	45.3	$ 831.71
		104	Anne K. Ramoras *	Systems Analyst	$ 96.75	32.4	$ 3,134.70
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	44.0	$ 2,021.80
				Subtotal			**$ 7,171.47**
22	Rolling Tide	105	Alice K. Johnson	Database Designer	$105.00	64.7	$ 6,793.50
		104	Anne K. Ramoras	Systems Analyst	$ 96.75	48.4	$ 4,682.70
		113	Delbert K. Joenbrood *	Applications Designer	$ 48.10	23.6	$ 1,135.16
		111	Geoff B. Wabash	Clerical Support	$ 26.87	22.0	$ 591.14
		106	William Smithfield	Programmer	$ 35.75	12.8	$ 457.60
				Subtotal			**$13,660.10**
25	Starflight	107	Maria D. Alonzo	Programmer	$ 35.75	24.6	$ 879.45
		115	Travis B. Bawangi	Systems Analyst	$ 96.75	45.8	$ 4,431.15
		101	John G. News *	Database Designer	$105.00	56.3	$ 5,911.50
		114	Annelise Jones	Applications Designer	$ 48.10	33.1	$ 1,592.11
		108	Ralph B. Washington	Systems Analyst	$ 96.75	23.6	$ 2,283.30
		118	James J. Frommer	General Support	$ 18.36	30.5	$ 559.98
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	41.4	$ 1,902.33
				Subtotal			**$17,559.82**
				Total			**$48,941.09**

Note: A * indicates the project leader.

Issues with our db

Here is our table again:

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.8
		101	John G. News *	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

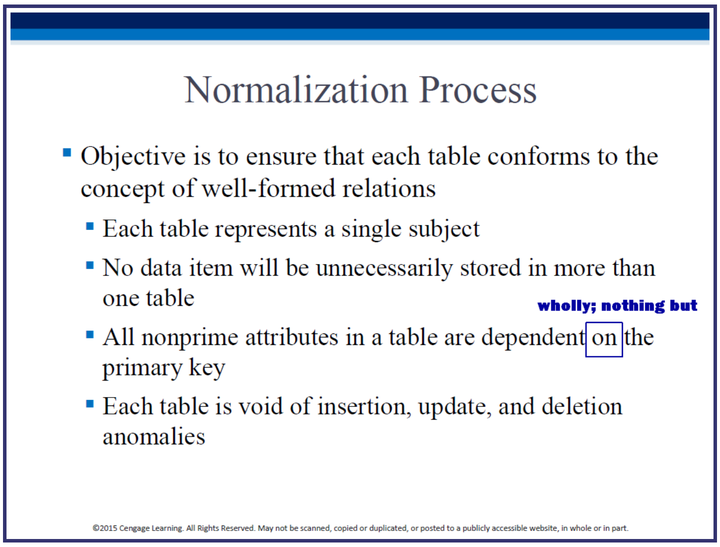
There are numerous issues:

- the PROJ_NUM attr could be used as a PK (or part of a PK, along with PROJ_NAME) but it contains nulls
- possibilities for data inconsistencies exist, eg. if someone's name or title is misspelled
- the redundancies that exist, could lead to insertion anomalies (eg. a new employee needs to be assigned to some project, even a fake one), update anomalies (eg. if an employee's JOB_CLASS changes, it has to be modified multiple times), deletion anomalies (eg. if a project has just one employee and that employee leaves, deleting the lone employee record would lead to the project itself getting deleted!)
- data redundancy leads to wasted storage space

We have 'repeating groups' (for each project, we list all details about each employee) - our table is un-normalized, ie. is in '0NF' :)

So, we need to clean up the design!

Objectives: what do we want?



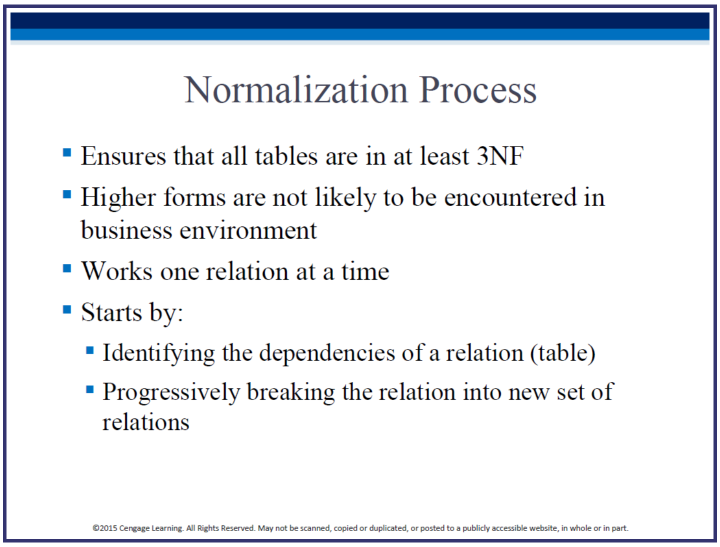
Normal forms

Normalization is a systematic process that yields progressively higher 'normal forms' (NFs) for each entity (table) in our db. We want **at least** 3NF for each table; in RL, we stop **at** 3NF.

Table 6.2 - Normal Forms

NORMAL FORM	CHARACTERISTIC	SECTION
First normal form (1NF)	Table format, no repeating groups, and PK identified	6.3.1
Second normal form (2NF)	1NF and no partial dependencies	6.3.2
Third normal form (3NF)	2NF and no transitive dependencies	6.3.3
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)	6.6.1
Fourth normal form (4NF)	3NF and no independent multivalued dependencies	6.6.2

The process



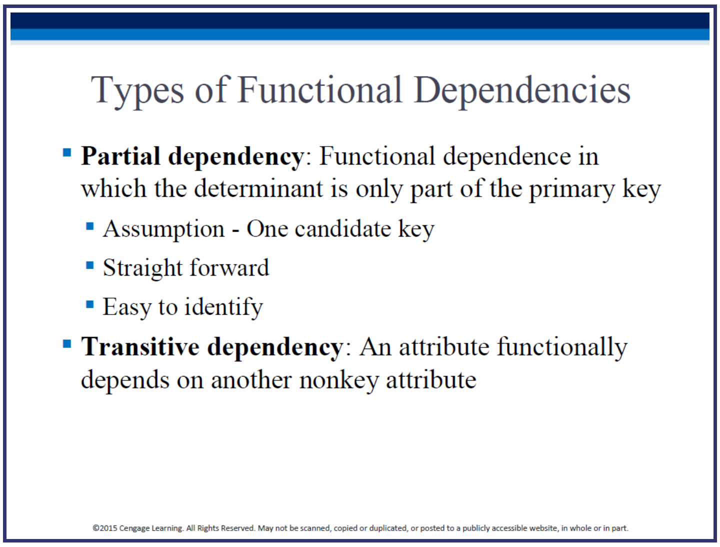
Normalization how-to, in one sentence: **work on one relation (table) at a time: identify dependencies, then 'normalize' - progressively break it down into smaller relations (tables), based on the dependencies we identify in the original relation** so that "only the PK, the whole PK and nothing but the PK" acts as a determinant! But how?? Details follow..

Functional dependence, determination

Functional Dependence Concepts

Concept	Definition
Functional dependence	The attribute B is fully functionally dependent on the attribute A if each value of A determines one and only one value of B.
Functional dependence (Generalized definition)	Attribute A determines attribute B if all of the rows in the table that agree in value for attribute A also agree in value for attribute B.
Fully functional dependence (composite key)	If attribute B is functionally dependent on a composite key A but not on any Subset of that composite key, the attribute B is fully functionally dependent on A.

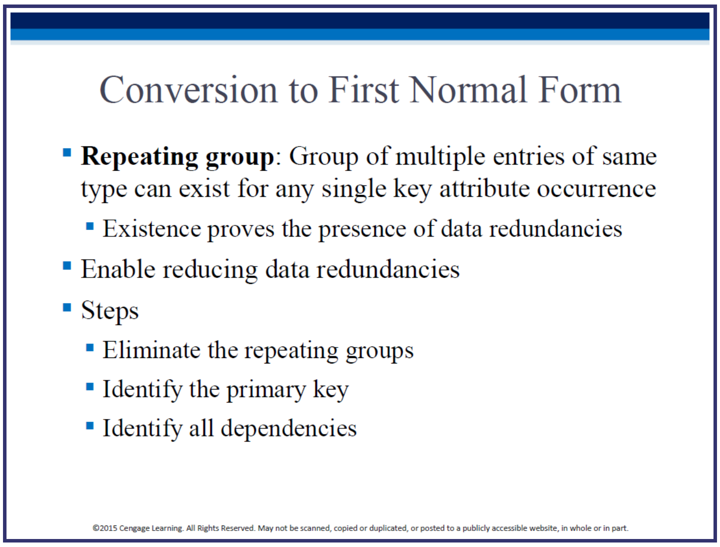
Partial dependency, transitive dependency



If (A,B) is a primary key, we have **partial** dependence if (A,B)->(C,D) and B->C [C is only partially dependent on the PK, ie. we only need B to determine C]. In other words, a part of an existing PK is acting like a PK on its own.

If X is a primary key, we have a **transitive** dependency if X->Y and Y->Z [Z is transitively dependent on X, not directly so]. In other words, a non-PK (regular attr) is acting like a PK.

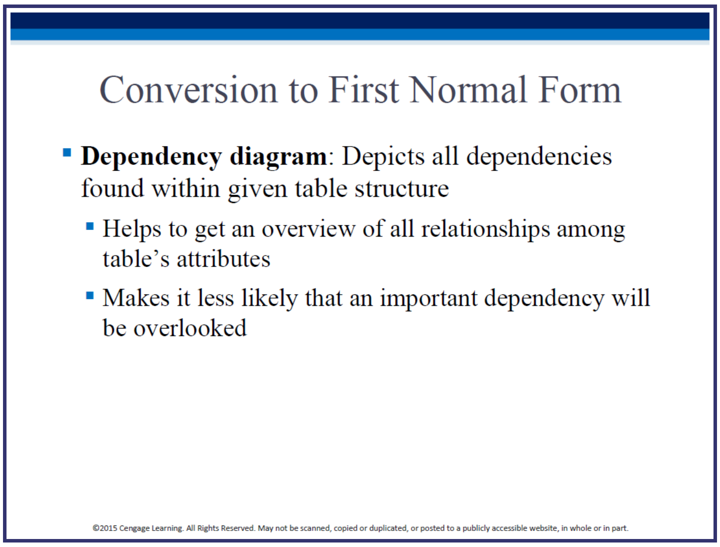
0NF->1NF: eliminate repeating groups



In other words, "fill in the blanks" so that there are no nulls. Now we have a relation (table), with a value in each cell.

Further, identify the PK! In our example, it is (PROJ_NUM,EMP_NUM).

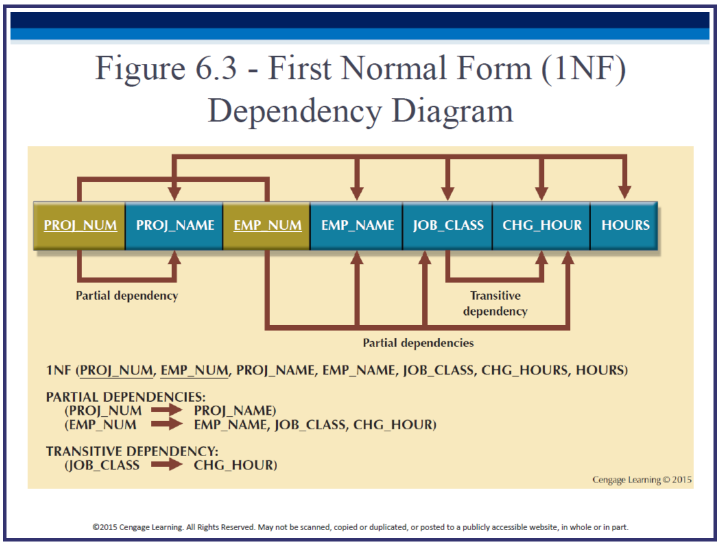
0NF->1NF [cont'd]



Create a **dependency diagram**, showing relationships (dependencies) between the attributes - this will help us systematically normalize the table.

Dependency diagram

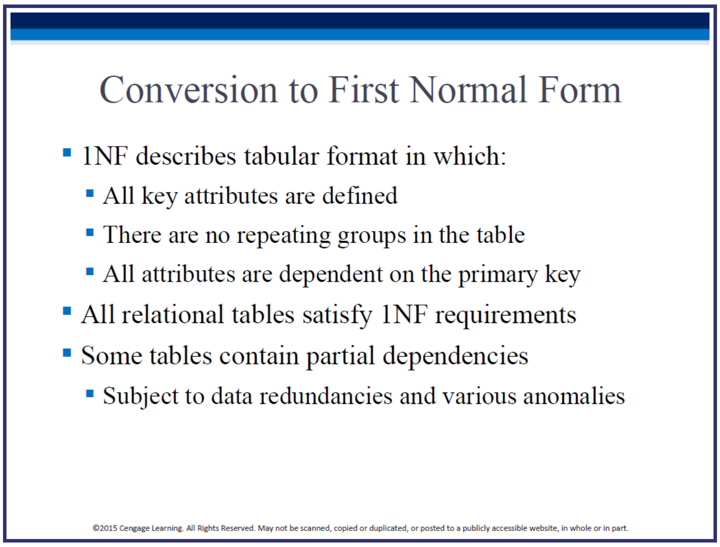
Indicate full dependencies on the top, and partial and transitive dependencies on the bottom. "Top good, bottom bad". Also, color the PK components in a different color (and underline them). Result:



PROJ_NAME has only a partial dependency on the PK (since it is only dependent on PROJ_NUM, which is just a part of the PK).

CHG_HOUR is dependent on JOB_CLASS, which is a non-prime attribute that is itself dependent on EMP_NUM. So JOB_CLASS->CHG_HOUR is a signaling dependency, indicating a EMP_NUM -> CHG_HOUR transitive dependency.

0NF->1NF [cont'd]



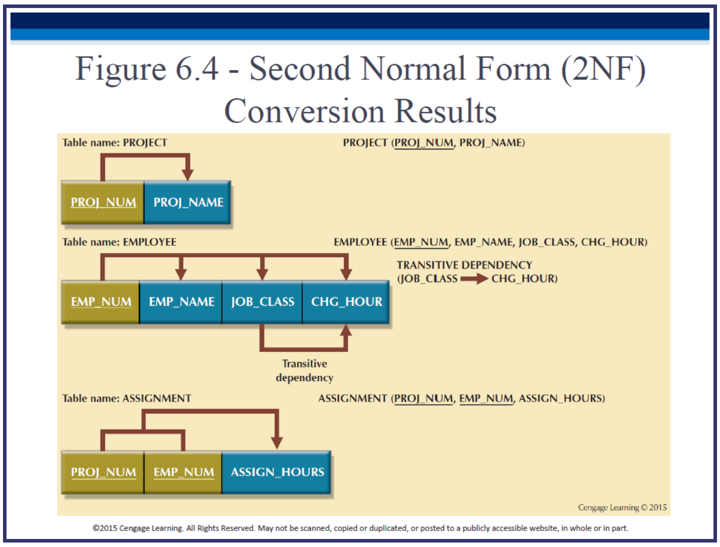
1NF->2NF: remove partial dependencies

Conversion to Second Normal Form

- Steps
  - Make new tables to eliminate partial dependencies
  - Reassign corresponding dependent attributes
- Table is in 2NF when it:
  - Is in 1NF
  - Includes no partial dependencies

1NF->2NF [cont'd]

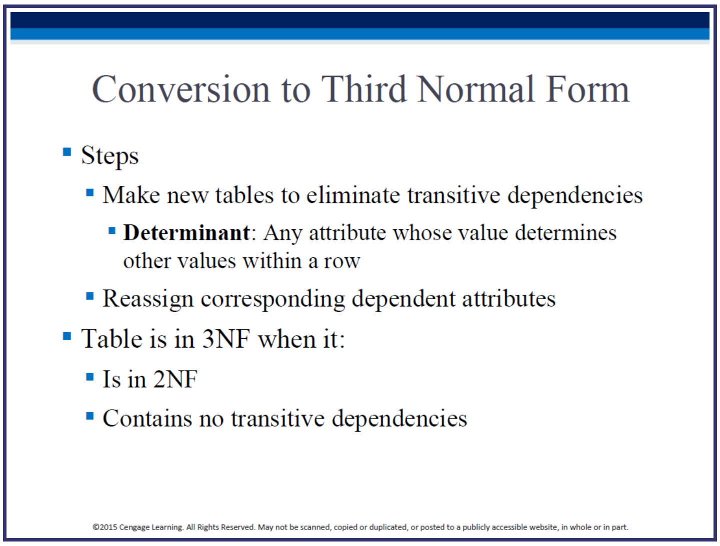
We eliminate partial dependencies by creating separate tables of such dependencies, and removing the dependent attributes from the starter table.



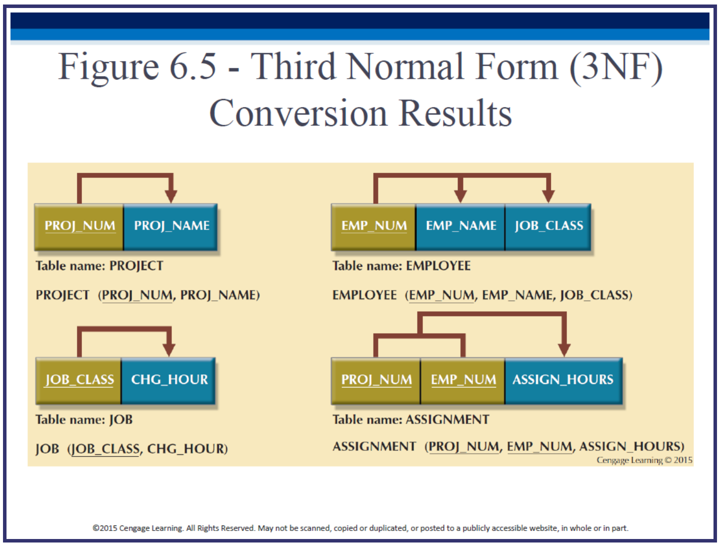
2NF->3NF: remove transitive dependencies

We promote the non-prime keys that masquerade as PKs, into actual PKs (give them their own tables).

Whether we eliminate partial dependencies (to create 2NF) or transitive ones (to create 3NF), we follow the same process: create a new relation for each 'problem' dependency!



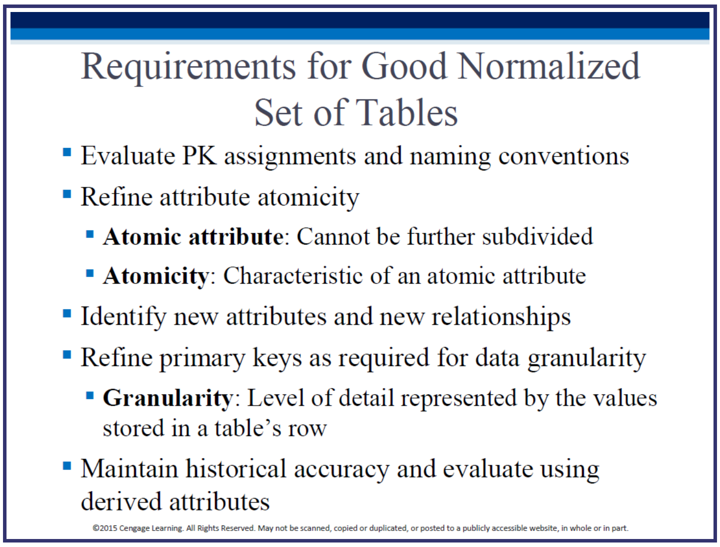
2NF->3NF [cont'd]



'Good' tables

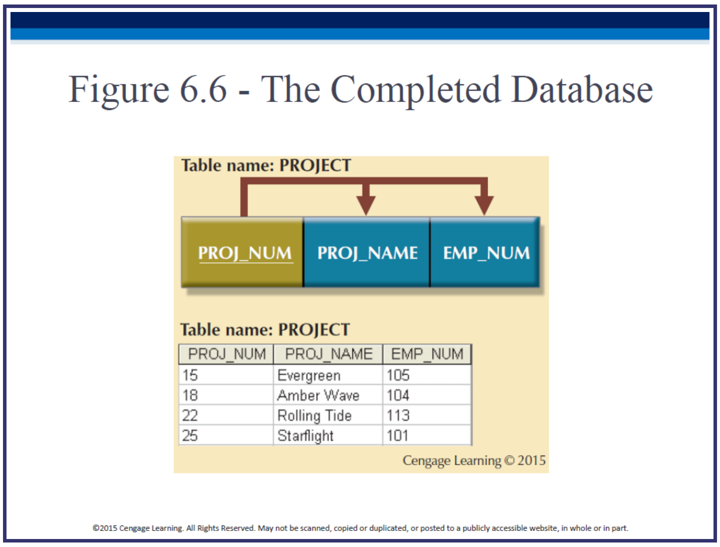
We can create a better DB by doing the following augmentations, to the 3NF model we just created:

- evaluate PKs - create a JOB_CODE
- evaluate naming conventions - eg. JOB_CHG_HOUR
- refine attr atomicity, eg. EMP_NAME
- identify new attrs, eg. EMP_HIREDATE
- identify new relationships, PROJECT can have EMP_NUM as FK [to be able to record a project's (always sole) manager]
- refine PKs for data granularity, eg. ASSIGN_NUM
- maintain historical accuracy [duplicate data], eg. store JOB_CHG_HOUR in ASSIGNMENT
- evaluate derived attrs, eg. ASSIGN_CHARGE



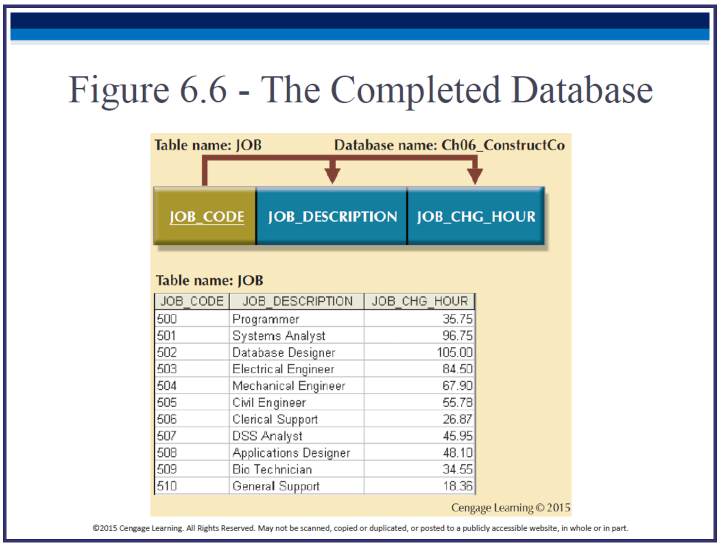
Final result

Here is the result of making the "extra" changes to our 3NF form:



PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

Final result [cont'd]



JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Final result [cont'd]

Figure 6.6 - The Completed Database

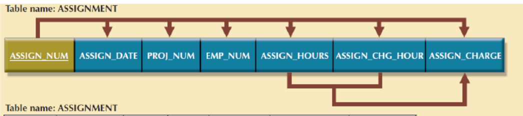


Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-14	15	103	2.6	84.50	219.70
1002	04-Mar-14	18	118	1.4	18.36	25.70
1003	05-Mar-14	15	101	3.6	105.00	378.00
1004	05-Mar-14	22	113	2.5	48.10	120.25
1005	05-Mar-14	15	103	1.9	84.50	160.55
1006	05-Mar-14	25	115	4.2	96.75	406.35
1007	05-Mar-14	22	105	5.2	105.00	546.00
1008	05-Mar-14	25	101	1.7	105.00	178.50
1009	05-Mar-14	15	105	2.0	105.00	210.00
1010	06-Mar-14	15	102	3.8	96.75	367.65
1011	06-Mar-14	22	104	2.6	96.75	251.55
1012	06-Mar-14	15	101	2.3	105.00	241.50
1013	06-Mar-14	25	114	1.8	48.10	86.58
1014	06-Mar-14	22	111	4.0	26.87	107.48
1015	06-Mar-14	25	114	3.4	48.10	163.54
1016	06-Mar-14	18	112	1.2	45.95	55.14
1017	06-Mar-14	18	118	2.0	18.36	36.72
1018	06-Mar-14	18	104	2.6	96.75	251.55
1019	06-Mar-14	15	103	3.0	84.50	253.50
1020	07-Mar-14	22	105	2.7	105.00	283.50
1021	08-Mar-14	25	108	4.2	96.75	406.35
1022	07-Mar-14	25	114	5.8	48.10	278.98
1023	07-Mar-14	22	106	2.4	35.75	85.80

Final result [cont'd]

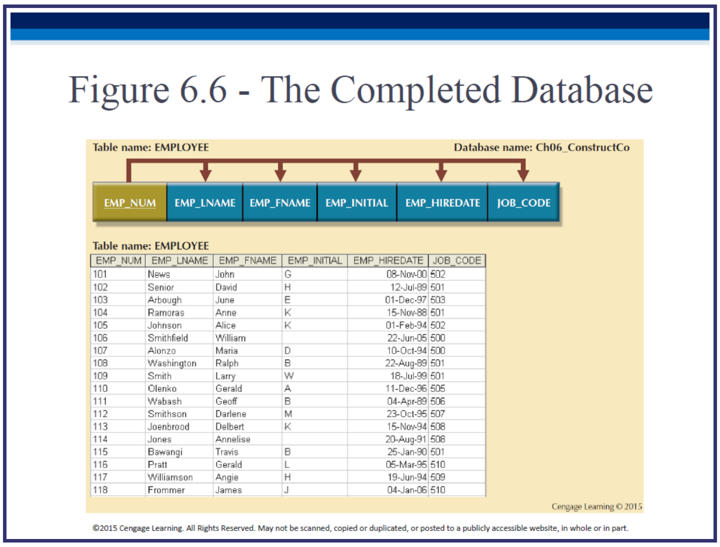


Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Normalization: summary

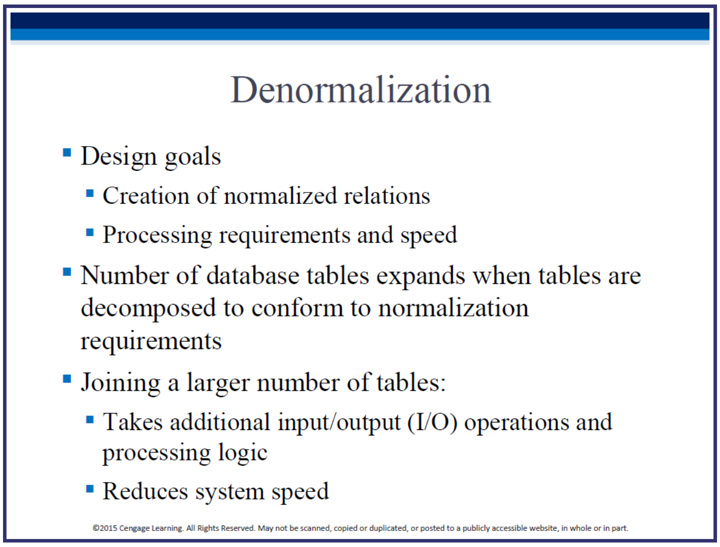
* 1NF: eliminate repeating groups (partial:y, transitive:y)

* 2NF: eliminate redundant data (partial:n, transitive:y)

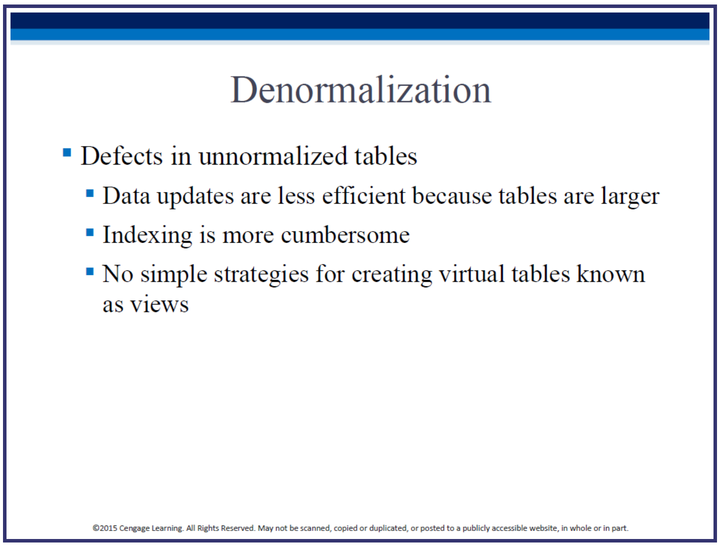
* 3NF: eliminate fields not dependent on key fields (partial:n, transitive:n)

Here is more, on normalization.

Denormalization



Denormalization [cont'd]



SQL

Where did SQL come from?

As mentioned earlier, SQL is an implementation of Ed Codd's relational set operators:

```
1. SELECT [formerly known as RESTRICT]
2. PROJECT
3. JOIN
4. PRODUCT
5. UNION
6. INTERSECT
7. DIFFERENCE
8. DIVIDE
```

Interestingly, Ed's ideas were ignored by IBM, his employer. Only after Oracle debuted (with SQL support right off the bat!) did IBM create DB2, its first relational DB.

SQL (1) 2023-03-08, 8:09 PM

SQL (1) 2023-03-08, 8:09 PM

Structured Query Language (SQL)

- Categories of SQL function
  - Data definition language (DDL)
  - Data manipulation language (DML)
- Nonprocedural language with basic command vocabulary set of less than 100 words
- Differences in SQL dialects are minor

how next page (Right Arrow)

Table 7.1 - SQL Data Definition Command

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

ıow next page (Right Arrow)

Table 7.2 - SQL Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values
Comparison operators	
=, <, >, <=, >=, <>	Used in conditional expressions
Logical operators	
AND/OR/NOT	Used in conditional expressions
Special operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

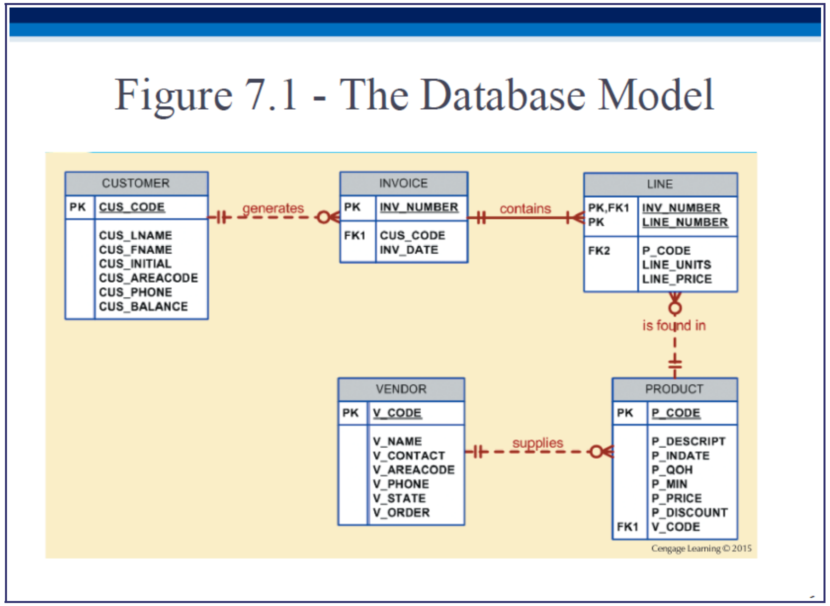
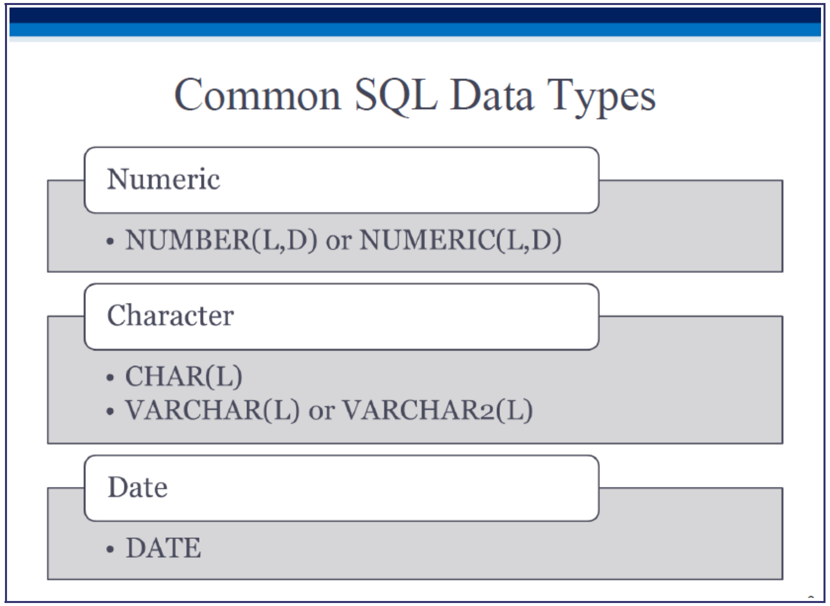


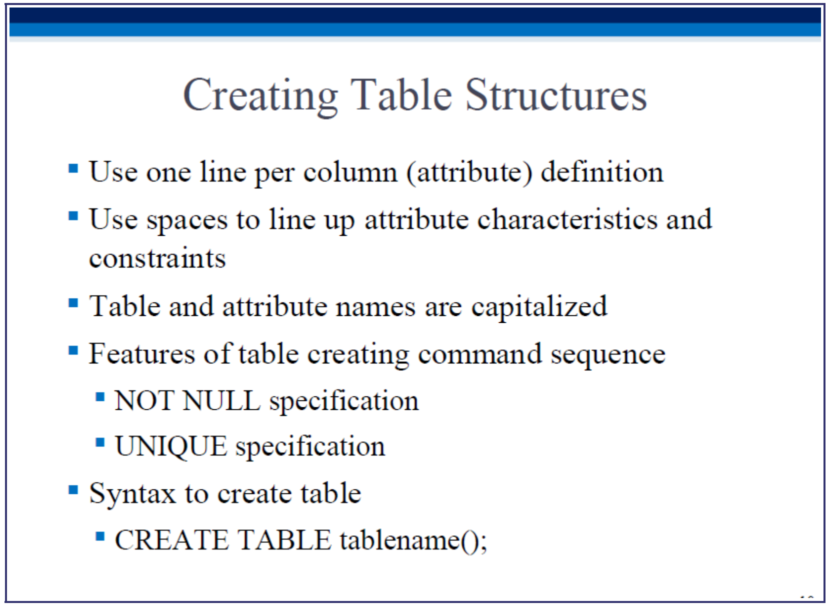
Figure 7.1 - The Database Model
CUSTOMER
PK CUS_CODE
CUS_LNAME
CUS_FNAME
CUS_INITIAL
CUS_AREACODE
CUS_PHONE
CUS_BALANCE
generates
INVOICE
PK INV_NUMBER
FK1 CUS_CODE
INV_DATE
contains
LINE
PK,FK1 INV_NUMBER
PK LINE_NUMBER
FK2 P_CODE
LINE_UNITS
LINE_PRICE
is found in
VENDOR
PK V_CODE
V_NAME
V_CONTACT
V_AREACODE
V_PHONE
V_STATE
V_ORDER
supplies
PRODUCT
PK P_CODE
P_DESCRIPT
P_INDATE
P_QOH
P_MIN
P_PRICE
P_DISCOUNT
FK1 V_CODE
Cengage Learning © 2015



NUMBER(p,s) [or NUMERIC(p,s)] denotes a number (including negative values) with upto 'p' number of total digits, including 's' number of them after the decimal point. 'p' is referred to as precision, 's' denotes scale. Eg. NUMERIC(4,2) would range from -99.99 to 99.99, NUMERIC(3,0) would be -999 to 999.

char vs VARCHAR (reserved for future use by SQL - so don't use!) vs VARCHAR2 [versus NVARCHAR2 (for Unicode)]: http://www.orafaq.com/faq/what_is_the_difference_between_varchar_varchar2_and_char_data_types

SQL (1) 2023-03-08, 8:09 PM



SQL (1) 2023-03-08, 8:09 PM

Primary Key and Foreign Key

- Primary key attributes contain both a NOT NULL and a UNIQUE specification
- RDBMS will automatically enforce referential integrity for foreign keys
- Command sequence ends with semicolon
- ANSI SQL allows use of following clauses to cover CASCADE, SET NULL, or SET DEFAULT
  - ON DELETE and ON UPDATE

```
CREATE TABLE PRODUCT (
P_CODE       VARCHAR(10)   NOT NULL    UNIQUE,
P_DESCRIPT   VARCHAR(35)   NOT NULL,
P_INDATE     DATE          NOT NULL,
P_QOH        SMALLINT      NOT NULL,
P_MIN        SMALLINT      NOT NULL,
P_PRICE      NUMBER(8,2)   NOT NULL,
P_DISCOUNT   NUMBER(5,2)   NOT NULL,
V_CODE       INTEGER,
PRIMARY KEY (P_CODE),
FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE
CASCADE);
```

SQL Constraints

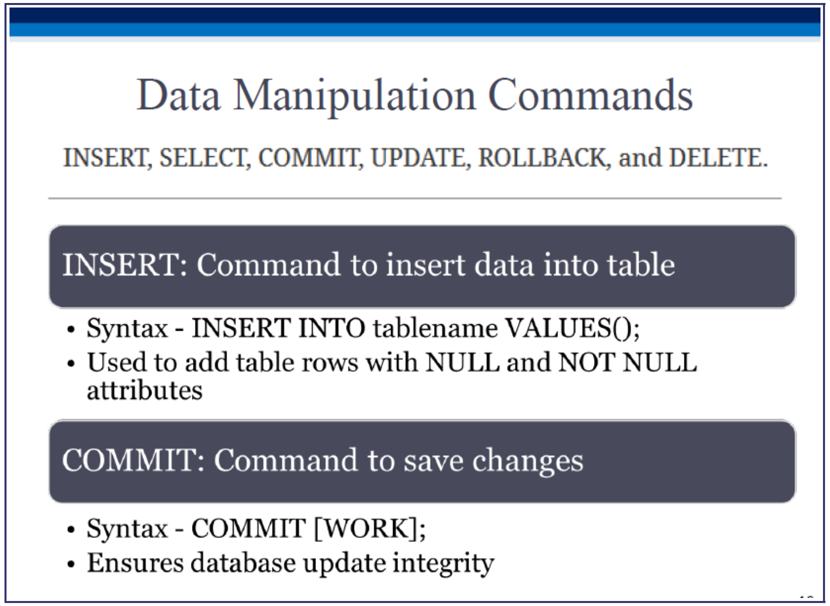
NOT NULL
- Ensures that column does not accept nulls

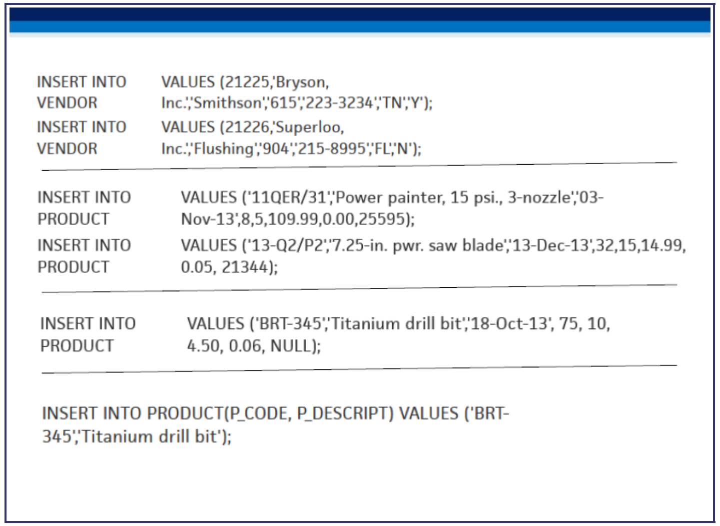
UNIQUE
- Ensures that all values in column are unique

DEFAULT
- Assigns value to attribute when a new row is added to table

CHECK
- Validates data when attribute value is entered

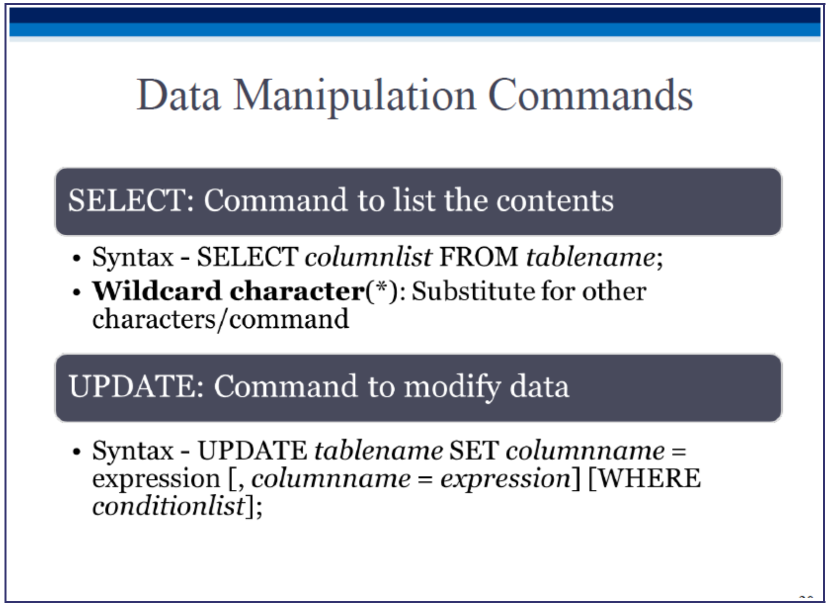
Plus: ON UPDATE CASCADE and ON DELETE CASCADE - both affect [change] a secondary table (that has an FK), when a change is made in the primary table (with the corresponding PK). ON UPDATE will update the values in the secondary table when corresp. values in the primary table are changed; ON DELETE will delete rows in the secondary table, when linked rows are deleted in the primary table.

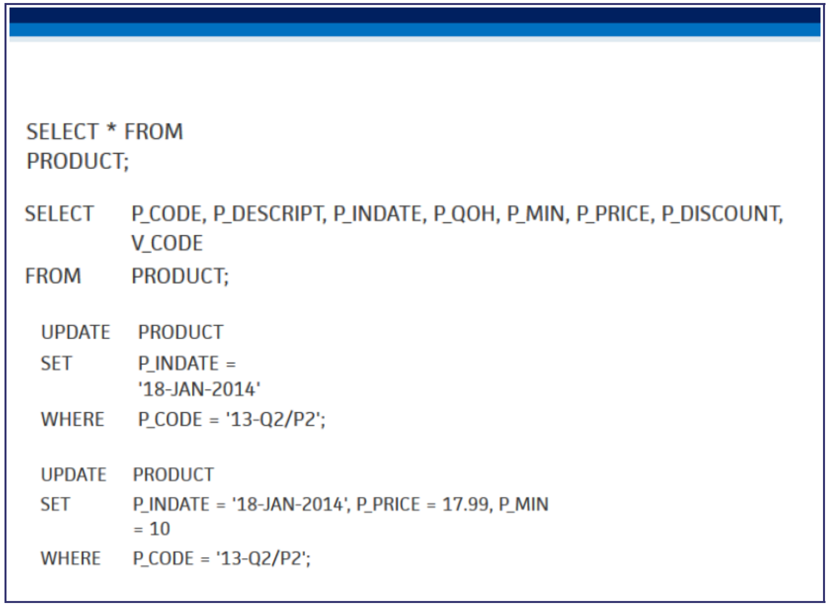

Data Manipulation Commands
INSERT, SELECT, COMMIT, UPDATE, ROLLBACK, and DELETE.
INSERT: Command to insert data into table
• Syntax - INSERT INTO tablename VALUES();
• Used to add table rows with NULL and NOT NULL attributes
COMMIT: Command to save changes
• Syntax - COMMIT [WORK];
• Ensures database update integrity



Vaguely similar to parameter passing/matching during a function call [where arguments need to match in position, type, count].

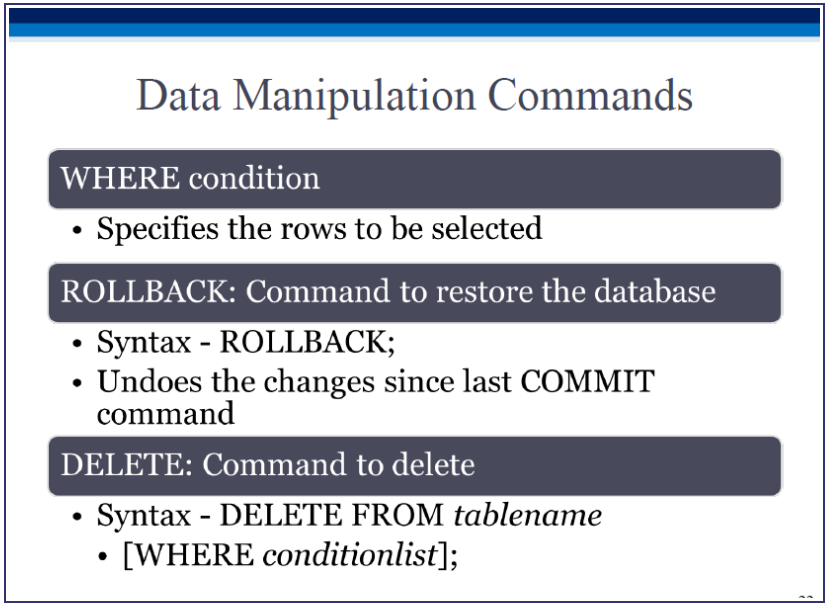
SQL (1) 2023-03-08, 8:09 PM





SELECT operates on 1 or more columns, and 0 or more rows from 1 or more tables - like many SQL commands, it is **set-oriented** and **non procedural.**

UPDATE modifies one or more columns of a table (on all rows, or on specific rows based on a condition specified by WHERE). Here is more.


Data Manipulation Commands
WHERE condition
• Specifies the rows to be selected
ROLLBACK: Command to restore the database
• Syntax - ROLLBACK;
• Undoes the changes since last COMMIT command
DELETE: Command to delete
• Syntax - DELETE FROM *tablename*
• [WHERE *conditionlist*];

Another ex: DELETE FROM PRODUCT WHERE P_MIN=5; (can be any condition on any attribute).

Q: what would DELETE do, if there isn't a WHERE condition?

Inserting Table Rows with a SELECT Subquery

- Syntax
  - INSERT INTO *tablename* SELECT *columnlist* FROM *tablename*
- Used to add multiple rows using another table as source
- SELECT command **-** Acts as a subquery and is executed first
  - **Subquery**: Query embedded/nested inside another query

INSERT INTO *tablename* SELECT *columnlist* FROM *tablename*;

Selecting Rows Using Conditional Restrictions

- Following syntax enables to specify which rows to select
  - SELECT *columnlist*
  - FROM *tablelist*
  - [WHERE *conditionlist*];
- Used to select partial table contents by placing restrictions on the rows
- Optional WHERE clause
  - Adds conditional restrictions to the SELECT statement

```
SELECT    columnlist
FROM      tablelist
[WHERE    conditionlist
          ];
```

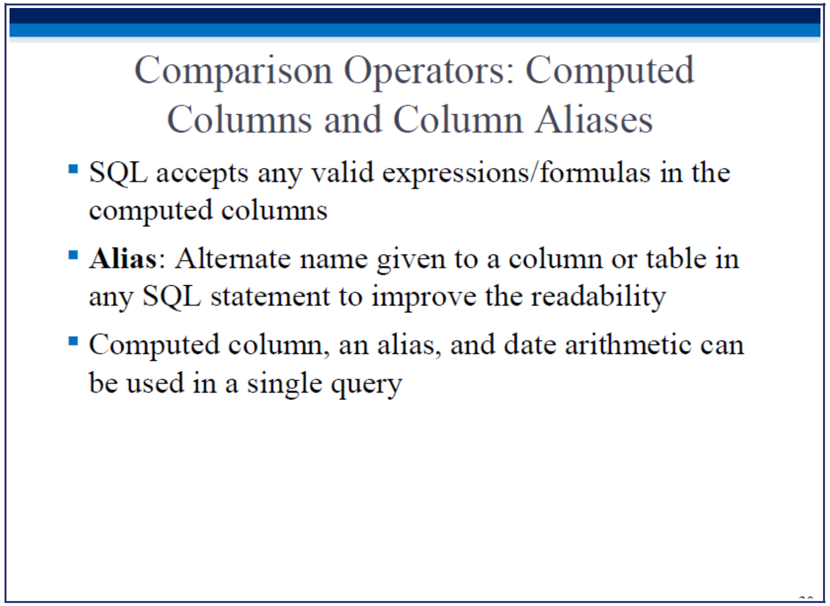
Comparison Operators

- Add conditional restrictions on selected table contents
- Used on:
  - Character attributes
  - Dates

Table 7.6 - Comparison Operators

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Computed columns (an ex of 'feature engineering')




```
SELECT      P_DESCRIPT, P_QOH, P_PRICE, P_QOH *
            P_PRICE
FROM        PRODUCT
```

P_DESCRIPT	P_QOH	P_PRICE	Expr1
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

```
SELECT    P_DESCRIPT, P_QOH, P_PRICE, P_QOH * P_PRICE AS TOTVALUE
FROM      PRODUCT;
```

P_DESCRIPT	P_QOH	P_PRICE	TOTVALUE
Power painter, 15 psi., 3-nozzle	8	109.99	879.92
7.25-in. pwr. saw blade	32	14.99	479.68
9.00-in. pwr. saw blade	18	17.49	314.82
Hrd. cloth, 1/4-in., 2x50	15	39.95	599.25
Hrd. cloth, 1/2-in., 3x50	23	43.99	1011.77
B&D jigsaw, 12-in. blade	8	109.92	879.36
B&D jigsaw, 8-in. blade	6	99.87	599.22
B&D cordless drill, 1/2-in.	12	38.95	467.40
Claw hammer	23	9.95	228.85
Sledge hammer, 12 lb.	8	14.40	115.20
Rat-tail file, 1/8-in. fine	43	4.99	214.57
Hicut chain saw, 16 in.	11	256.99	2826.89
PVC pipe, 3.5-in., 8-ft	188	5.87	1103.56
1.25-in. metal screw, 25	172	6.99	1202.28
2.5-in. wd. screw, 50	237	8.45	2002.65
Steel matting, 4'x8'x1/6", .5" mesh	18	119.95	2159.10

```
SELECT    P_CODE, P_INDATE, P_INDATE + 90 AS EXPDATE
FROM      PRODUCT;
```

Arithmetic operators

Arithmetic operators

- **The Rule of Precedence**: Establish the order in which computations are completed
- Perform:
  - Operations within parentheses
  - Power operations
  - Multiplications and divisions
  - Additions and subtractions

Table 7.7 - The Arithmetic Operators

ARITHMETIC OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of (some applications use ** instead of ^)

Cengage Learning © 2015

Logical (Boolean) operators

Figure 7.12 - Selected PRODUCT Table Attributes: The logical OR

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
7.25-in. pwr. saw blade	13-Dec-13	14.99	21344
9.00-in. pwr. saw blade	13-Nov-13	17.49	21344
B&D jigsaw, 12-in. blade	30-Dec-13	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-13	99.87	24288
Rat-tail file, 1/8-in. fine	15-Dec-13	4.99	21344
Hicut chain saw, 16 in.	07-Feb-14	256.99	24288


```
SELECT P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM   PRODUCT
WHERE  V_CODE = 21344 OR V_CODE = 24288;
```

Figure 7.13 - Selected PRODUCT Table Attributes: The Logical AND

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D cordless drill, 1/2-in.	20-Jan-14	38.95	25595
Claw hammer	20-Jan-14	9.95	21225
PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87	
1.25-in. metal screw, 25	01-Mar-14	6.99	21225
2.5-in. wd. screw, 50	24-Feb-14	8.45	21231

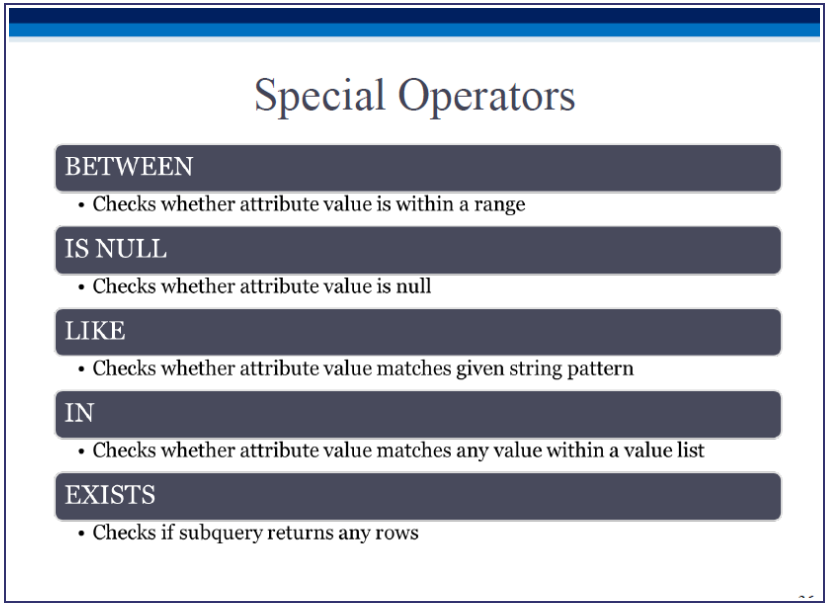

```
SELECT   P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM     PRODUCT
WHERE    P_PRICE < 50
AND      P_INDATE > '15-Jan-2014';
```


Figure 7.14 - Selected PRODUCT Table Attributes: The Logical AND and OR

P_DESCRIPT	P_INDATE	P_PRICE	V_CODE
B&D jigsaw, 12-in. blade	30-Dec-13	109.92	24288
B&D jigsaw, 8-in. blade	24-Dec-13	99.87	24288
B&D cordless drill, 1/2-in.	20-Jan-14	38.95	25595
Claw hammer	20-Jan-14	9.95	21225
Hicut chain saw, 16 in.	07-Feb-14	256.99	24288
PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87	
1.25-in. metal screw, 25	01-Mar-14	6.99	21225
2.5-in. wd. screw, 50	24-Feb-14	8.45	21231


```
SELECT   P_DESCRIPT, P_INDATE, P_PRICE, V_CODE
FROM     PRODUCT
WHERE    (P_PRICE < 50 AND P_INDATE > '15-Jan-2014')
OR       V_CODE = 24288;
```

'Special' operators



BETWEEN

```
SELECT    *
FROM      PRODUCT
WHERE     P_PRICE BETWEEN 50.00 AND 100.00;


SELECT    P_CODE, P_DESCRIPT, V_CODE
FROM      PRODUCT
WHERE     V_CODE IS NULL;
```

LIKE

- % means any and all *following* or *preceding* characters are eligible. For example:

  'J%' includes Johnson, Jones, Jernigan, July, and J-231Q.

  'Jo%' includes Johnson and Jones.

  '%n' includes Johnson and Jernigan.

- _means any *one* character may be substituted for the underscore. For example:

  '_23-456-6789' includes 123-456-6789, 223-456-6789, and 323-456-6789.

  '_23-_56-678_' includes 123-156-6781, 123-256-6782, and 823-956-6788.

  '_o_es' includes Jones, Cones, Cokes, totes, and roles.

```
SELECT   V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM     VENDOR
WHERE    V_CONTACT LIKE 'Smith%';
```

Another example for pattern-matching: load this page, and enter and execute:

```
SELECT * FROM Customers WHERE CustomerName LIKE 'C%';
```

IN

```
SELECT    *
FROM      PRODUCT
WHERE     V_CODE = 21344
OR        V_CODE = 24288;

SELECT    *
FROM      PRODUCT
WHERE     V_CODE IN (21344, 24288);


SELECT    V_CODE, V_NAME
FROM      VENDOR
WHERE     V_CODE IN (SELECT V_CODE FROM PRODUCT);
```

'EXISTS'

The EXISTS clause is used with a query, and returns TRUE if the subquery results in any output (non-zero # of rows being returned), or FALSE if the subquery results in no data. The rest of the query (the 'main' query) will (or will not) run, based on EXIST's output - if EXISTS returns false, the main query will get skipped.

You can loosely think of EXISTS as "ONLY WHEN". We use it to 'defensively' update (insert, modify, delete) parts of a table (after we determine it is updatable).

You can also think of 'WHERE EXISTS' as "such that there exists". Eg. in the first example below (SELECT * FROM VENDOR..), it reads as "Find all vendors such that there exists records for them in the PRODUCT TABLE (via V_CODE) where P_QOH<=P_MIN" [in other words, we are looking for vendors we need to re-order from].

In the second example (INSERT INTO CONTACTS..), read it as "get the supplier ID and name for all suppliers such that there exists order IDs for them, then insert them into the contacts table".

Exercise: what does #3 below (the UPDATE query) do?

```
SELECT    *
FROM      VENDOR
WHERE     EXISTS (SELECT * FROM PRODUCT WHERE P_QOH <= P_MIN);
```

```
INSERT INTO contacts
(contact_id, contact_name)
SELECT supplier_id, supplier_name
FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

```
UPDATE suppliers
SET supplier_name = (SELECT customers.name
                     FROM customers
                     WHERE customers.customer_id = suppliers.supplier_id)
WHERE EXISTS (SELECT customers.name
              FROM customers
              WHERE customers.customer_id = suppliers.supplier_id);
```

```
DELETE FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

A couple more (ex of EXISTS, NOT EXISTS):

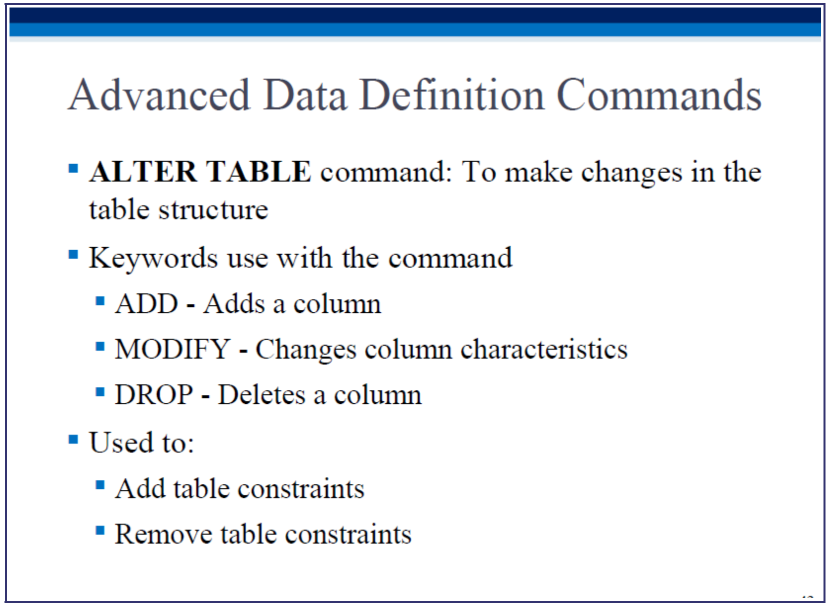
```
SELECT *
FROM suppliers
WHERE EXISTS (SELECT *
              FROM orders
              WHERE suppliers.supplier_id = orders.supplier_id);
```

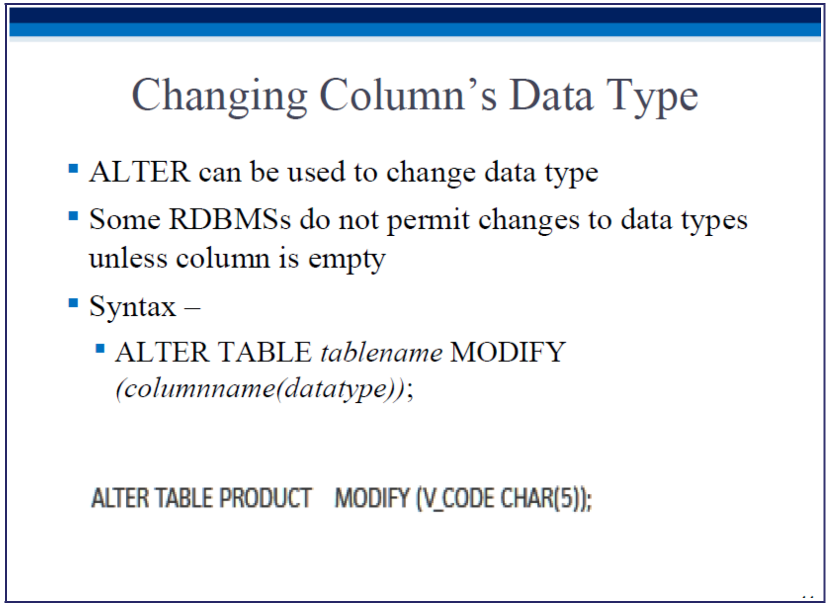
This SQL EXISTS condition example will return all records from the suppliers table where there is at least one record in the orders table with the same supplier_id.

```
SELECT *
FROM suppliers
WHERE NOT EXISTS (SELECT *
                  FROM orders
                  WHERE suppliers.supplier_id = orders.supplier_id);
```

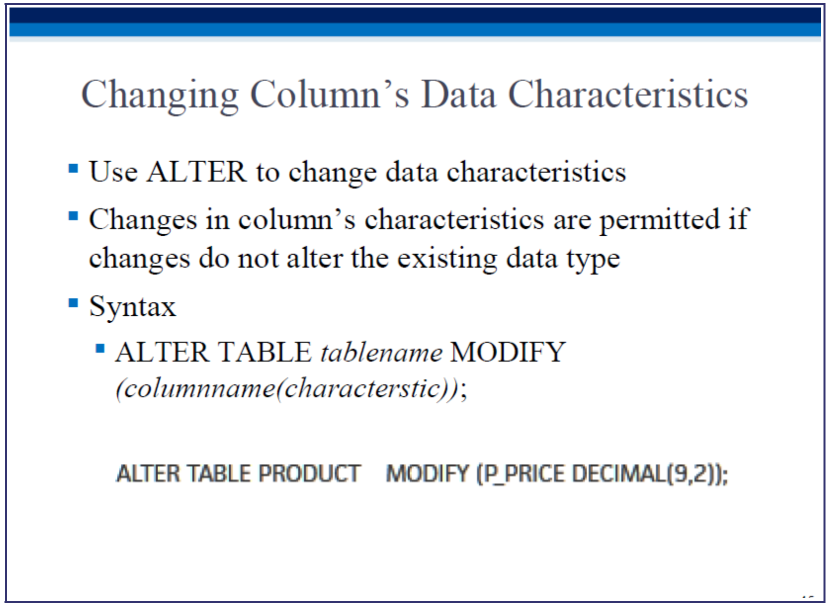
This SQL EXISTS example will return all records from the suppliers table where there are **no** records in the *orders* table for the given supplier_id.

ALTER

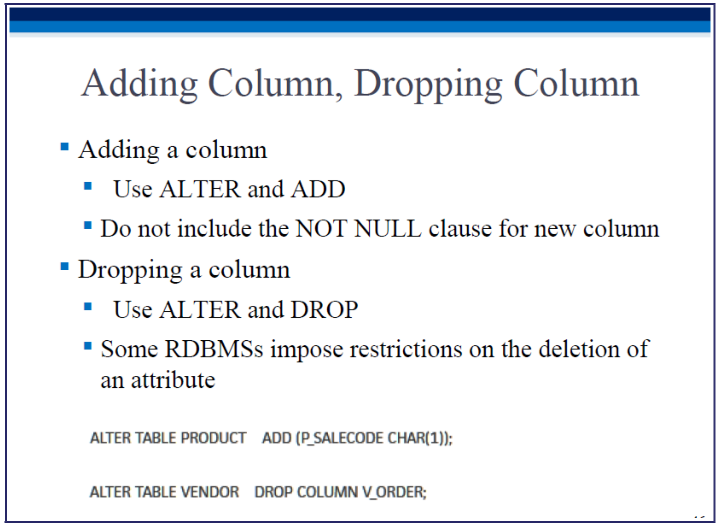

Advanced Data Definition Commands
ALTER TABLE command: To make changes in the table structure
Keywords use with the command
ADD - Adds a column
MODIFY - Changes column characteristics
DROP - Deletes a column
Used to:
Add table constraints
Remove table constraints



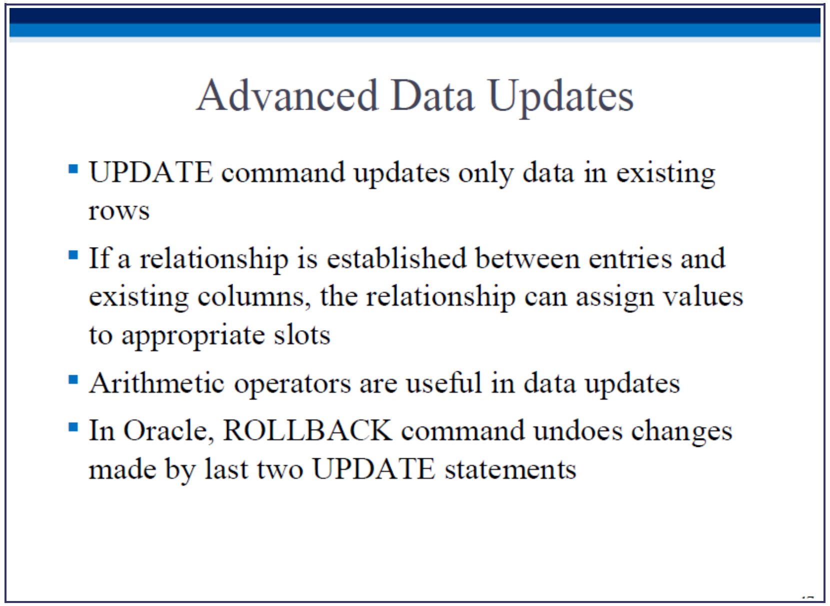
SQL (1) 2023-03-08, 8:09 PM



SQL (1) 2023-03-08, 8:09 PM


Adding Column, Dropping Column
Adding a column
Use ALTER and ADD
Do not include the NOT NULL clause for new column
Dropping a column
Use ALTER and DROP
Some RDBMSs impose restrictions on the deletion of an attribute
ALTER TABLE PRODUCT ADD (P_SALECODE CHAR(1));
ALTER TABLE VENDOR DROP COLUMN V_ORDER;

The UPDATE command



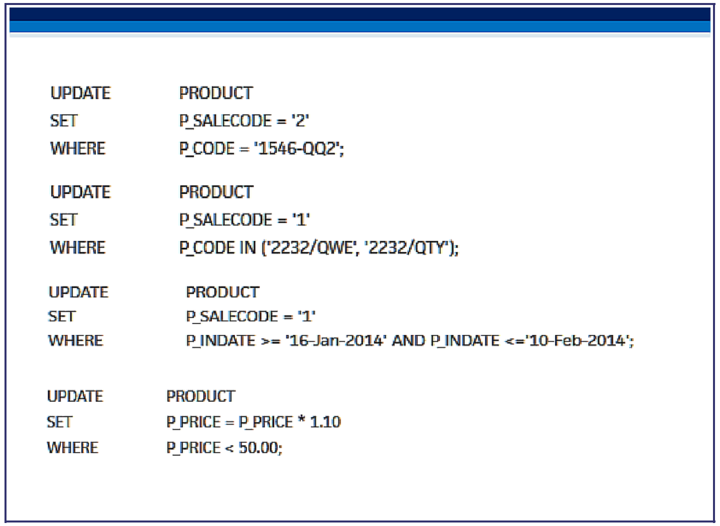
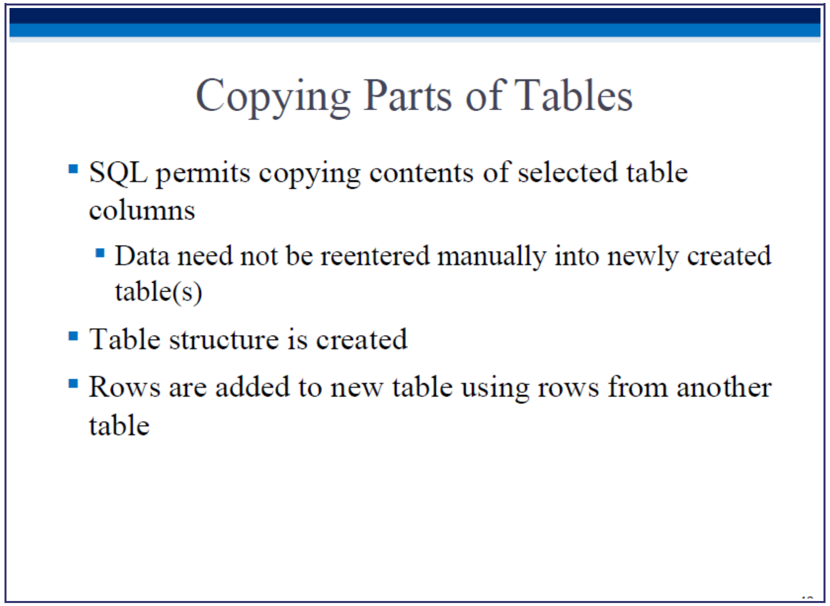
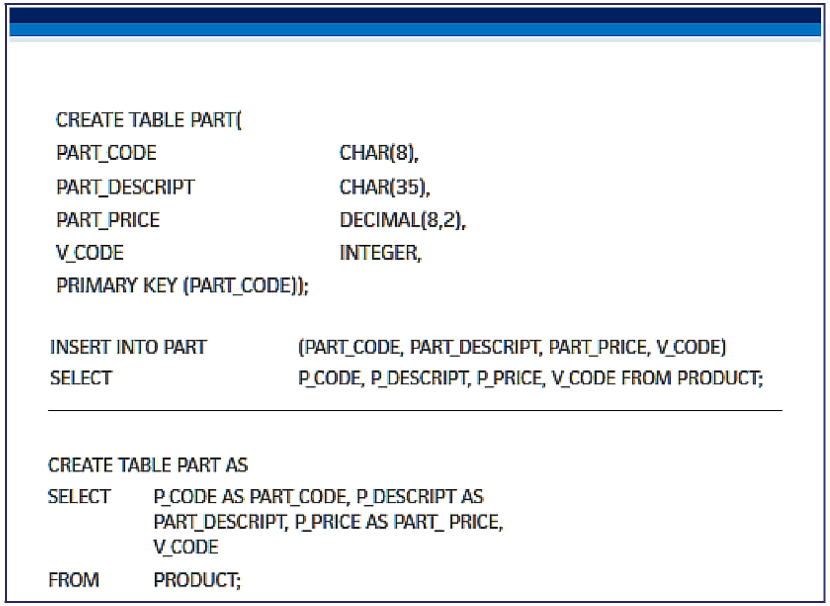


Table update in 'bulk'

Sometimes we can update a table, by filling it with output from a query. The table to be filled in has to exist first (so we need to create it if necessary), and have type-compatible columns that can receive values from our data-fetching query.

The table creation and updating can happen in separate steps, or even be combined for compactness of expression.





SQL (1) 2023-03-08, 8:09 PM

Adding Primary and Foreign Key Designations

- **ALTER TABLE** command
  - Followed by a keyword that produces the specific change one wants to make
  - Options include ADD, MODIFY, and DROP
- Syntax to add or modify columns
  - ALTER TABLE *tablename*
    - {ADD | MODIFY} (*columnname datatype* [{ADD | MODIFY} *columnname datatype*]) ;
  - ALTER TABLE *tablename*
    - ADD *constraint* [ADD *constraint*] ;

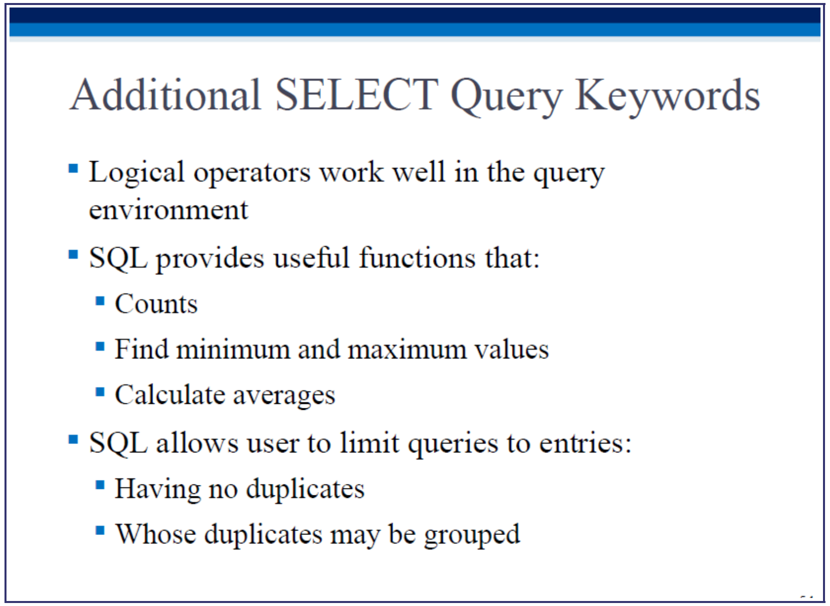
```
ALTER TABLE   PART
    ADD       PRIMARY KEY (PART_CODE);

ALTER TABLE   PART
    ADD       FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Oftentimes the need to do this occurs when a table (eg. PART) is created via bulk-update, from another table (eg. PRODUCT).

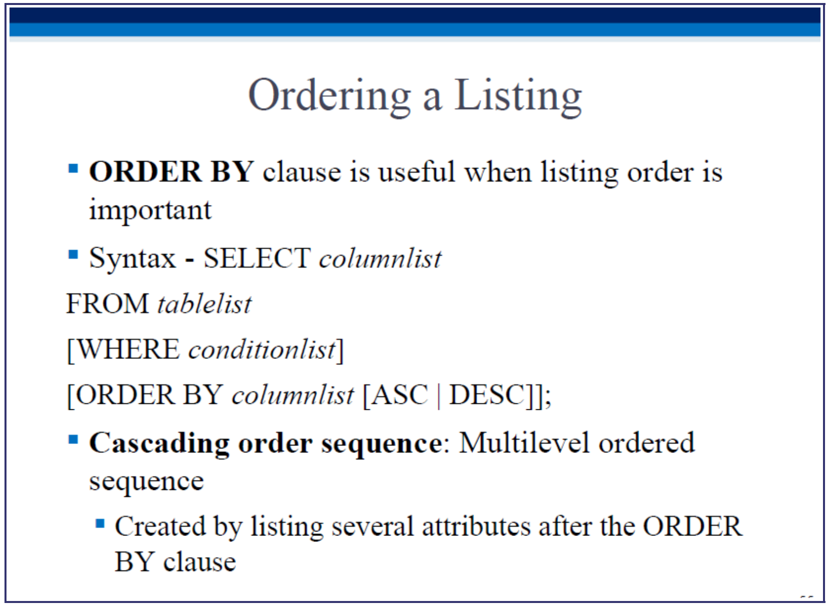
Deleting a Table from the Database

- **DROP TABLE**: Deletes table from database
- Syntax - DROP TABLE tablename;
- Can drop a table only if it is not the one side of any relationship
- RDBMS generates a foreign key integrity violation error message if the table is dropped

Ordering, unique entries, aggregate ops..



ORDER BY



Below is an example of ORDER BY, where by default, values are listed (sorted) in ascending order. To list the values in descending order, we'd do this: ORDER BY P_PRICE DESC;

```
SELECT      P_CODE, P_DESCRIPT, P_INDATE, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE;
```

P_CODE	P_DESCRIPT	P_INDATE	P_PRICE
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-13	4.99
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-14	5.87
SM-18277	1.25-in. metal screw, 25	01-Mar-14	6.99
SW-23116	2.5-in. wd. screw, 50	24-Feb-14	8.45
23109-HB	Claw hammer	20-Jan-14	9.95
23114-AA	Sledge hammer, 12 lb.	02-Jan-14	14.40
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-13	14.99
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-13	17.49
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-14	38.95
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-14	39.95
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-14	43.99
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-13	99.87
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-13	109.92
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-13	109.99
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-14	119.95
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-14	256.99

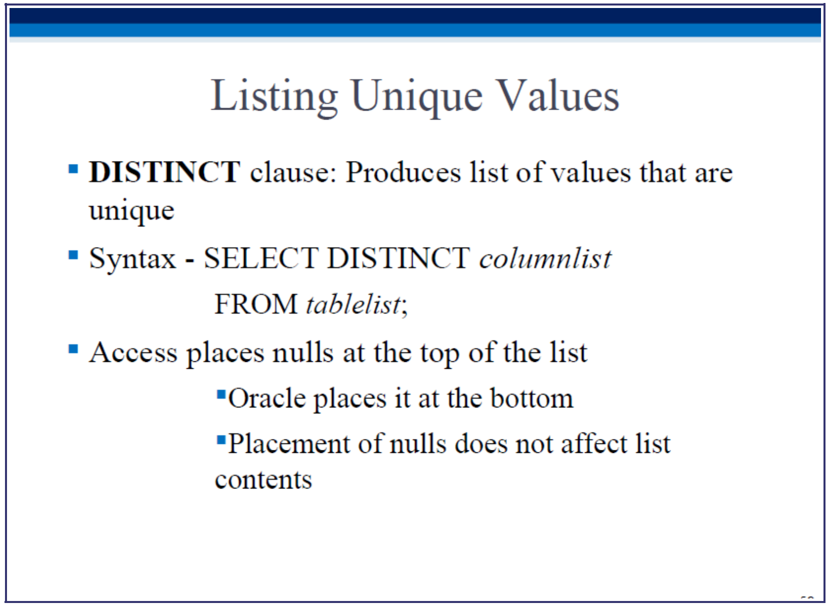
The sequence (listing) obtained by specifying several comma-separated attributes for ORDER BY is called a cascading order sequence.


```
SELECT      EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, EMP_PHONE
FROM        EMPLOYEE
ORDER BY    EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_AREACODE	EMP_PHONE
Brandon	Marie	G	901	882-0845
Diante	Jorge	D	615	890-4567
Genkazi	Leighla	W	901	569-0093
Johnson	Edward	E	615	898-4387
Jones	Anne	M	615	898-3456
Kolmycz	George	D	615	324-5456
Lange	John	P	901	504-4430
Lewis	Rhonda	G	615	324-4472
Saranda	Hermine	R	615	324-5505
Smith	George	A	615	890-2984
Smith	George	K	901	504-3339
Smith	Jeanine	K	615	324-7883
Smythe	Melanie	P	615	324-9006
Vandam	Rhett		901	675-8993
Washington	Rupert	E	615	890-4925
Wiesenbach	Paul	R	615	897-4358
Williams	Robert	D	615	890-3220

DISTINCT

The 'DISTINCT' keyword is used to count unique/distinct occurences of an attribute:



Example of 'DISTINCT' usage

```
SELECT    DISTINCT V_CODE
FROM      PRODUCT;
```

V_CODE
21225
21231
21344
23119
24288
25595

AGGREGATE functions

COUNT, MIN, MAX, SUM, AVG are all functions that operate on a numerical attr/column, and produce a single scalar result (not a table).

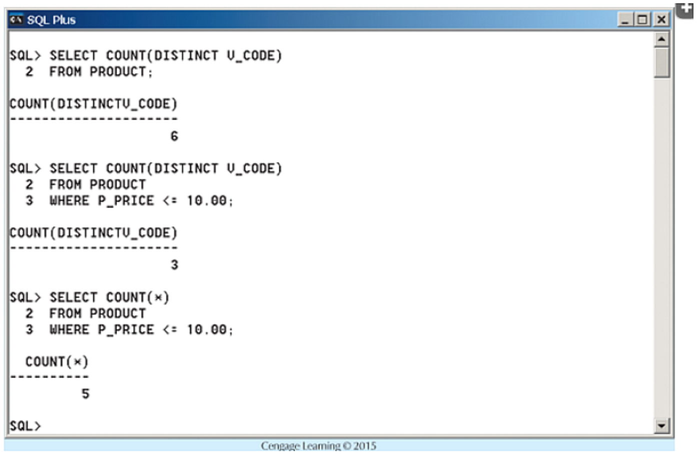
Table 7.8 - Some Basic SQL Aggerate Functions

FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

Aggregate function examples

How many different vendors supply our products? How many supply cheap (PRICE < 10) products?

Note the third query below: COUNT(*) counts the # of rows returned by a query (rows where the product costs < 10 units). In contrast, count() counts the # of non-null values of the column.



MAX, MIN

What is the value for the most expensive item in the product table? Least expensive?

What is the most expensive item (details)? We can't do: WHERE P_PRICE = MAX(P_PRICE). This is because MAX() can only be used in a SELECT statement.

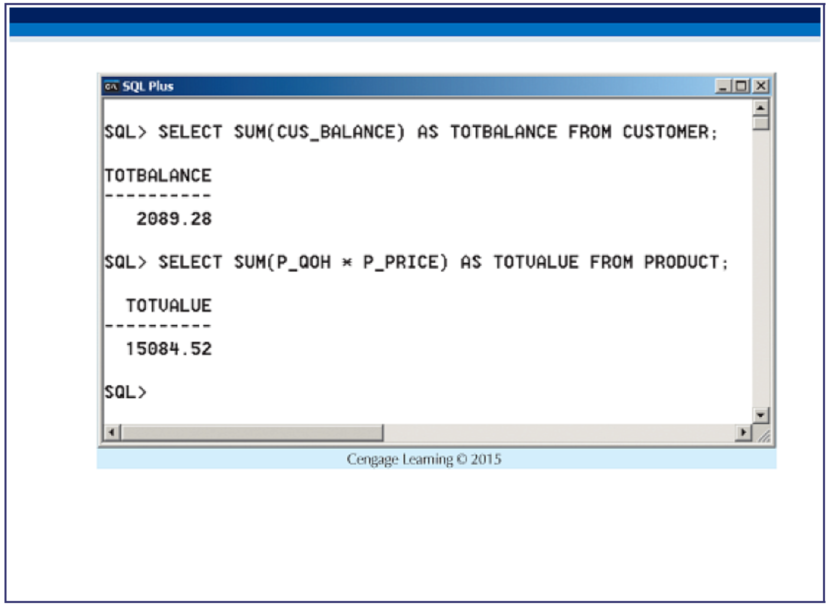
SQL Plus

```
SQL> SELECT MAX(P_PRICE)
  2  FROM PRODUCT;

MAX(P_PRICE)
------------
      256.99

SQL> SELECT MIN(P_PRICE)
  2  FROM PRODUCT;

MIN(P_PRICE)
------------
        4.99

SQL> SELECT P_CODE, P_DESCRIPT, P_PRICE
  2  FROM PRODUCT
  3  WHERE P_PRICE = (SELECT MAX(P_PRICE) FROM PRODUCT);

P_CODE     P_DESCRIPT                              P_PRICE
---------- ----------------------------------- ----------
89-WRE-Q   Hicut chain saw, 16 in.                  256.99

SQL>
```

Cengage Learning © 2015

```
SELECT *
FROM PRODUCT
WHERE P_QOH*P_PRICE = (SELECT MAX(P_QOH*P_PRICE) FROM PRODUCT);
```

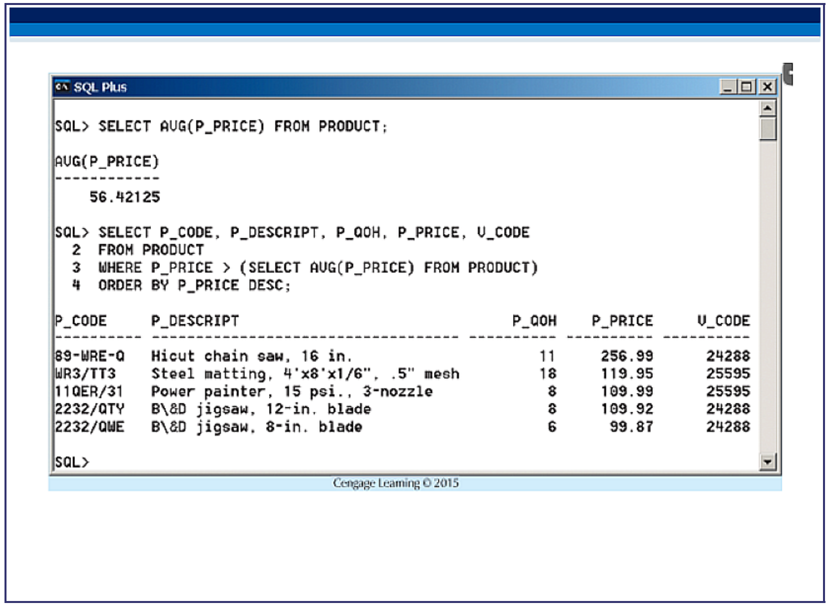
SUM


SQL Plus
SQL> SELECT SUM(CUS_BALANCE) AS TOTBALANCE FROM CUSTOMER;
TOTBALANCE

2089.28
SQL> SELECT SUM(P_QOH * P_PRICE) AS TOTVALUE FROM PRODUCT;
TOTVALUE

15084.52
SQL>
Cengage Learning © 2015

AVG


SQL Plus
SQL> SELECT AVG(P_PRICE) FROM PRODUCT;
AVG(P_PRICE)

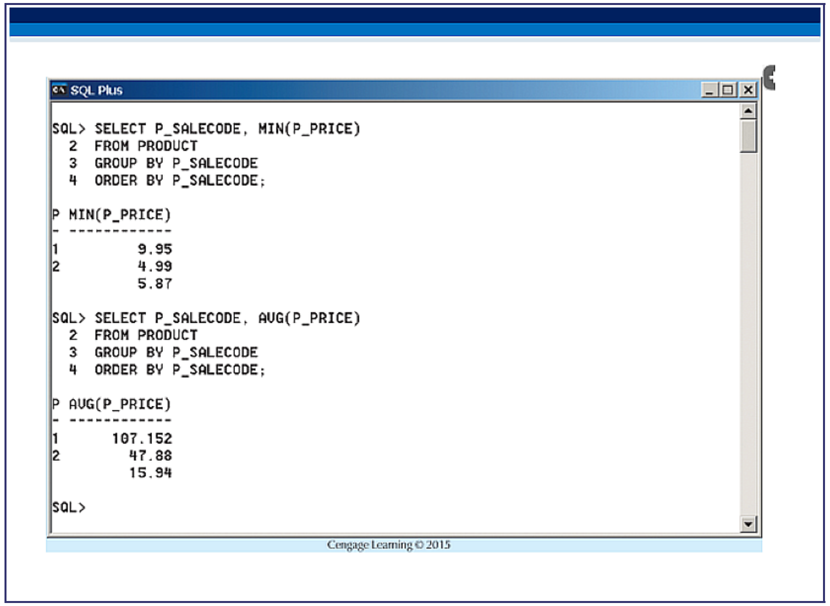
56.42125
SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_PRICE, V_CODE
2 FROM PRODUCT
3 WHERE P_PRICE > (SELECT AVG(P_PRICE) FROM PRODUCT)
4 ORDER BY P_PRICE DESC;
P_CODE P_DESCRIPT P_QOH P_PRICE V_CODE
89-WRE-Q Hicut chain saw, 16 in. 11 256.99 24288
WR3/TT3 Steel matting, 4'x8'x1/6", .5" mesh 18 119.95 25595
11QER/31 Power painter, 15 psi., 3-nozzle 8 109.99 25595
2232/QTY B\&D jigsaw, 12-in. blade 8 109.92 24288
2232/QWE B\&D jigsaw, 8-in. blade 6 99.87 24288
SQL>
Cengage Learning © 2015

GROUP BY (itemizing)

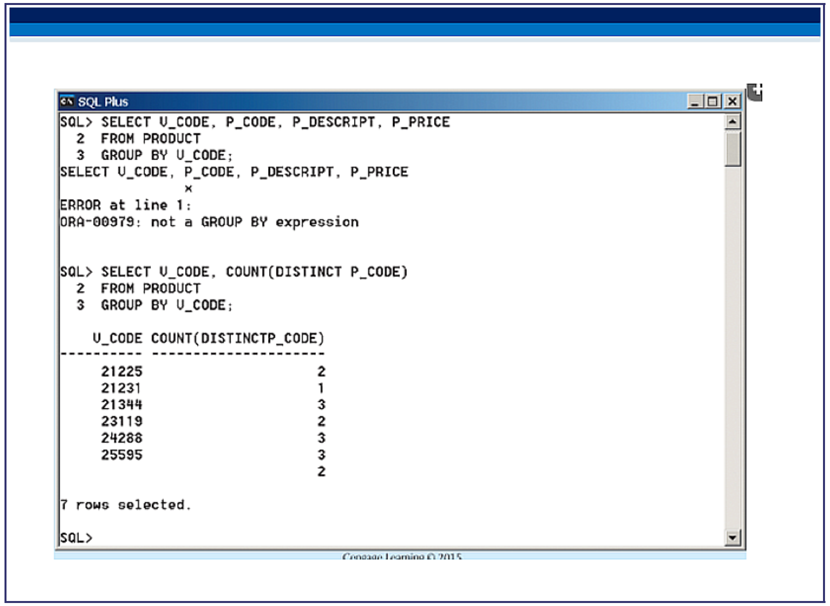
Grouping Data

- Frequency distributions created by **GROUP BY** clause within SELECT statement
- GROUP BY can **only** be used in concert with an aggregate function
- Syntax - SELECT *columnlist*
  FROM *tablelist*
  [WHERE *conditionlist*]
  [GROUP BY *columnlist*]
  [HAVING *conditionlist*]
  [ORDER BY *columnlist* [ASC | DESC]];

'GROUP BY' example



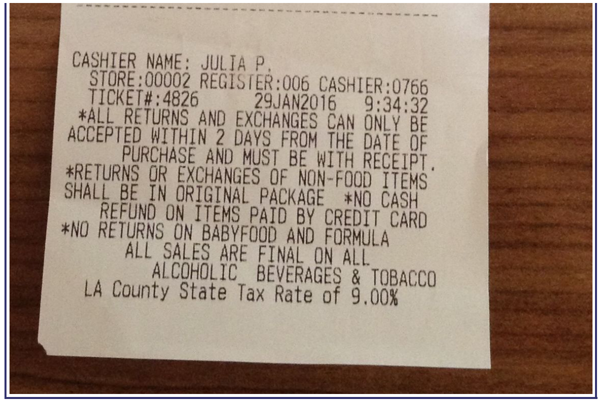
You can think of 'GROUP BY' to mean "CATEGORIZE BY" or "ITEMIZE BY": rather than just a single MIN, MAX, SUM, COUNT or AVG, we're asking for a value PER OCCURRENCE of a GROUP BY value, eg. minimum price PER VENDOR, max GPA PER DEPARTMENT, average earnings PER MAJOR, etc. Specifically, when used with SUM(), GROUP BY is used to request subtotals.



'GROUP BY' used without an aggregate function is meaningless, since there is no aggregate value (MIN, COUNT etc.) to itemize.

Store receipt

Here is a grocery store receipt where subtotals are displayed itemized, aggregated by product types DELI, GROCERY, MIXED NUTS and PRODUCE (presumably using SUM(), together with GROUP BY):

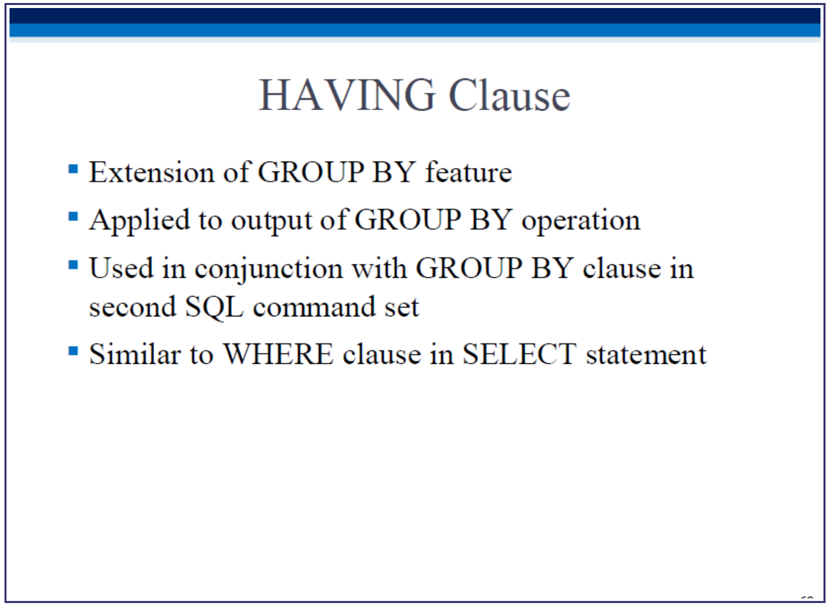


While you're at it, see what other DATA you can spot in the receipt! Even a single trip to the store can generate a LOT of data.

HAVING

The HAVING clause is used to filter rows in a GROUP BY specification (only those rows HAVING met the specified condition will appear in the result).

Note that HAVING can only occur in a query that has a GROUP BY, which in turn can only occur when there is an aggregate function (MIN, MAX, SUM, COUNT or AVG).



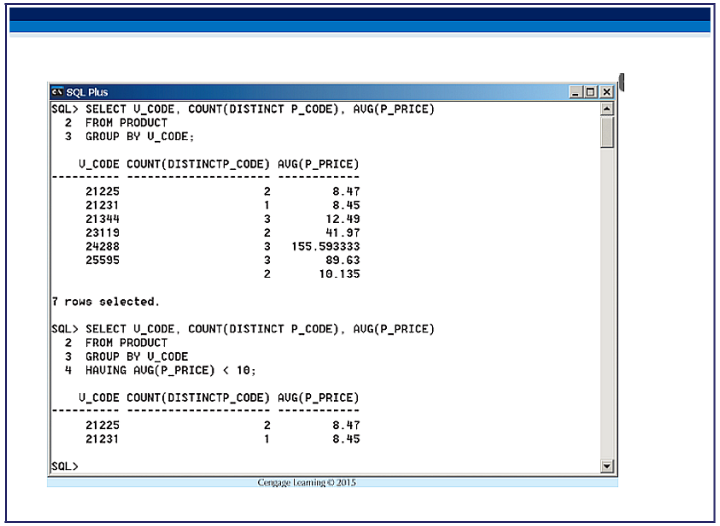
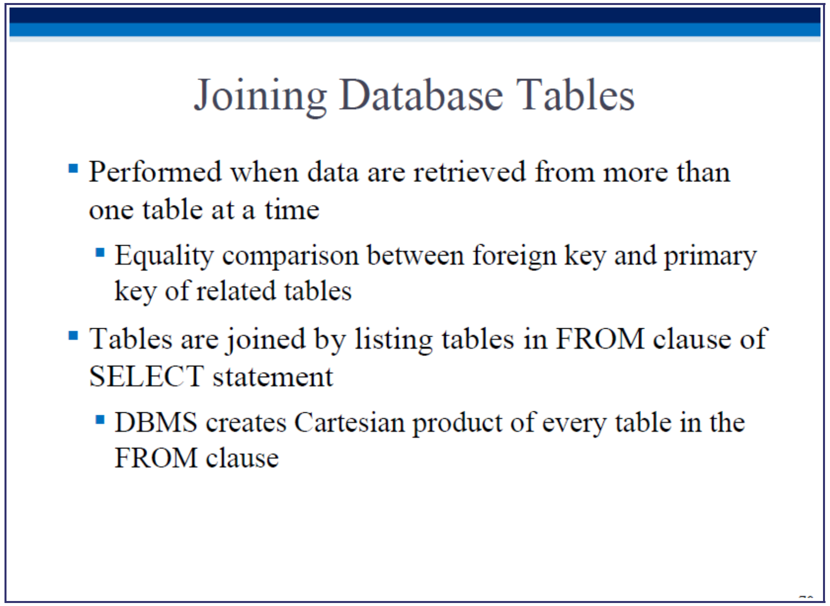


Table joins


Joining Database Tables
Performed when data are retrieved from more than one table at a time
Equality comparison between foreign key and primary key of related tables
Tables are joined by listing tables in FROM clause of SELECT statement
DBMS creates Cartesian product of every table in the FROM clause

Joining Tables With an Alias

- Alias identifies the source table from which data are taken
- Any legal table name can be used as alias
- Add alias after table name in FROM clause
  - FROM tablename alias, eg. 'FROM PRODUCT P'

```
SELECT    P_DESCRIPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM      PRODUCT, VENDOR
WHERE     PRODUCT.V_CODE = VENDOR.V_CODE;
```

P_DESCRIPT	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
Claw hammer	9.95	Bryson, Inc.	Smithson	615	223-3234
1.25-in. metal screw, 25	6.99	Bryson, Inc.	Smithson	615	223-3234
2.5-in. wd. screw, 50	8.45	D&E Supply	Singh	615	228-3245
7.25-in. pwr. saw blade	14.99	Gomez Bros.	Ortega	615	889-2546
9.00-in. pwr. saw blade	17.49	Gomez Bros.	Ortega	615	889-2546
Rat-tail file, 1/8-in. fine	4.99	Gomez Bros.	Ortega	615	889-2546
Hrd. cloth, 1/4-in., 2x50	39.95	Randsets Ltd.	Anderson	901	678-3998
Hrd. cloth, 1/2-in., 3x50	43.99	Randsets Ltd.	Anderson	901	678-3998
B&D jigsaw, 12-in. blade	109.92	ORDVA, Inc.	Hakford	615	898-1234
B&D jigsaw, 8-in. blade	99.87	ORDVA, Inc.	Hakford	615	898-1234
Hicut chain saw, 16 in.	256.99	ORDVA, Inc.	Hakford	615	898-1234
Power painter, 15 psi., 3-nozzle	109.99	Rubicon Systems	Orton	904	456-0092
B&D cordless drill, 1/2-in.	38.95	Rubicon Systems	Orton	904	456-0092
Steel matting, 4'x8'x1/6", .5" mesh	119.95	Rubicon Systems	Orton	904	456-0092

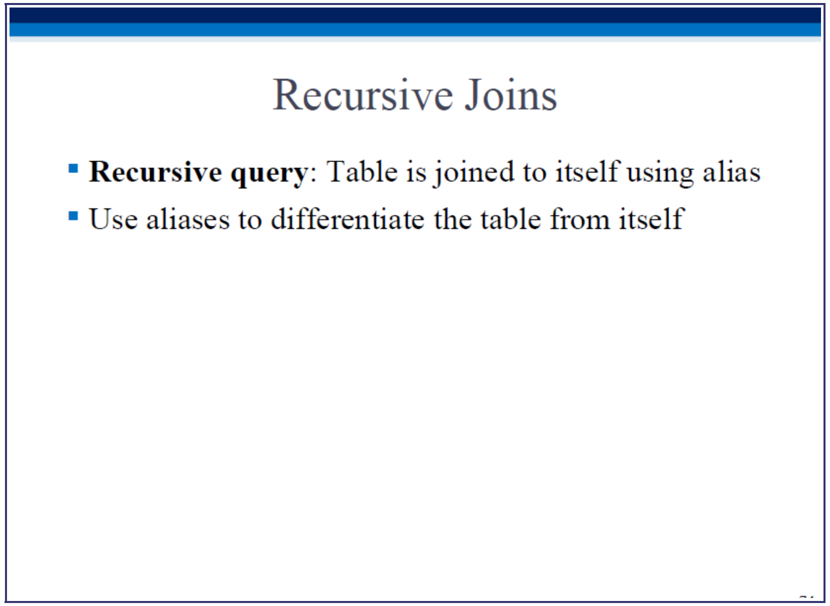

```
ORDER BY    PRODUCT.P_PRICE;
```

```
SELECT      CUS_LNAME, INVOICE.INV_NUMBER, INV_DATE, P_DESCRIPT
FROM        CUSTOMER, INVOICE, LINE, PRODUCT
WHERE       CUSTOMER.CUS_CODE = INVOICE.CUS_CODE
AND         INVOICE.INV_NUMBER = LINE.INV_NUMBER
AND         LINE.P_CODE = PRODUCT.P_CODE
AND         CUSTOMER.CUS_CODE = 10014
ORDER BY    INV_NUMBER;
```

Joining n tables will need (n-1) join conditions.

'Recursive join' example

Need different aliases for the table being queried so that we can use such aliases as namespaces for attributes.



Table, for self-join example

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE	EMP_AREACODE	EMP_PHONE	EMP_MGR
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-85	615	324-5456	
101	Ms.	Lewis	Rhonda	G	19-Mar-65	25-Apr-86	615	324-4472	100
102	Mr.	Vandam	Rhett		14-Nov-58	20-Dec-90	901	675-8993	100
103	Ms.	Jones	Anne	M	16-Oct-74	28-Aug-94	615	898-3456	100
104	Mr.	Lange	John	P	08-Nov-71	20-Oct-94	901	504-4430	105
105	Mr.	Williams	Robert	D	14-Mar-75	08-Nov-98	615	890-3220	
106	Mrs.	Smith	Jeanine	K	12-Feb-68	05-Jan-89	615	324-7883	105
107	Mr.	Diante	Jorge	D	21-Aug-74	02-Jul-94	615	890-4567	105
108	Mr.	Wiesenbach	Paul	R	14-Feb-66	18-Nov-92	615	897-4358	
109	Mr.	Smith	George	K	18-Jun-61	14-Apr-89	901	504-3339	108
110	Mrs.	Genkazi	Leighla	W	19-May-70	01-Dec-90	901	569-0093	108
111	Mr.	Washington	Rupert	E	03-Jan-66	21-Jun-93	615	890-4925	105
112	Mr.	Johnson	Edward	E	14-May-61	01-Dec-83	615	898-4387	100
113	Ms.	Smythe	Melanie	P	15-Sep-70	11-May-99	615	324-9006	105
114	Ms.	Brandon	Marie	G	02-Nov-56	15-Nov-79	901	882-0845	108
115	Mrs.	Saranda	Hermine	R	25-Jul-72	23-Apr-93	615	324-5505	105
116	Mr.	Smith	George	A	08-Nov-65	10-Dec-88	615	890-2984	108

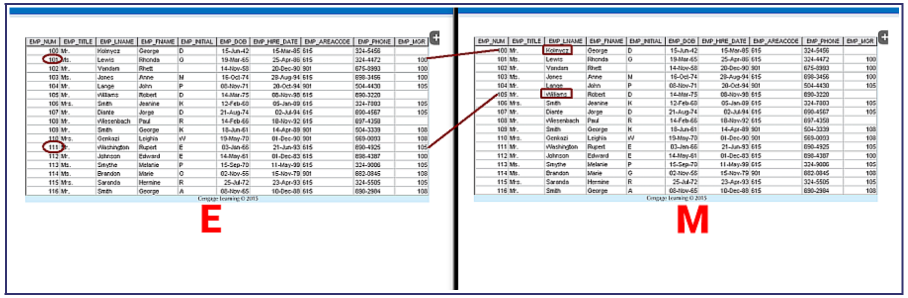
Self-join example

```
SELECT      E.EMP_NUM, E.EMP_LNAME, E.EMP_MGR, M.EMP_LNAME
FROM        EMP E, EMP M
WHERE       E.EMP_MGR=M.EMP_NUM
ORDER BY    E.EMP_MGR;
```

EMP_NUM	E.EMP_LNAME	EMP_MGR	M.EMP_LNAME
112	Johnson	100	Kolmycz
103	Jones	100	Kolmycz
102	Vandam	100	Kolmycz
101	Lewis	100	Kolmycz
115	Saranda	105	Williams
113	Smythe	105	Williams
111	Washington	105	Williams
107	Diante	105	Williams
106	Smith	105	Williams
104	Lange	105	Williams
116	Smith	108	Wiesenbach
114	Brandon	108	Wiesenbach
110	Genkazi	108	Wiesenbach
109	Smith	108	Wiesenbach

Cengage Learning © 2015

Here is how the join works:



Sooo... are joins 'evil'? Not really :)

SQL is declarative, not imperative

Now that you're familiar with SQL syntax, you will appreciate knowing how it compares with a regular programming language such as C++. Shown below is such a comparison, using C# which lets us write code in an imperative (command-oriented) manner as well a declarative (result-oriented) one [this is from a book on functional programming]:

Listing 1.3 Imperative data processing (C#)

```
List<string> res = new List<string>();                  #1
foreach(Product p in Products) {                        #2
   if (p.UnitPrice > 75.0M) {
      res.Add(String.Format("{0} - ${1}",
         p.ProductName, p.UnitPrice));                  #3
   }
}
return res;
```

#1 Create resulting list
#2 Iterate over products
#3 Add information to list of results

You'll probably need to read the code carefully to understand what it does, but that's not the only aspect we want to improve. The code is written as a sequence of some basic imperative commands. For example, the first statement creates new list (#1), the second iterates over all products (#2) and a later one adds element to the list (#3). However, we'd like to be able to describe the problem at a higher level. In more abstract terms, the code just filters a collection and returns some information about every returned product.

In C# 3.0, we can write the same code using query expression syntax. This version is closer to our real goal–it uses the same idea of filtering and transforming the data. You can see the code in listing 1.4.

Listing 1.4 Declarative data processing (C#)

```
var res = from p in Products
          where p.UnitPrice > 75.0M                      #1
          select string.Format("{0} - ${1}",
             p.ProductName, p.UnitPrice);                #2
return res;
```

#1 Filter products using predicate
#2 Return information about product

The expression that calculates the result (`res`) is composed from basic operators such as `where` or `select`. These operators take other expressions as an argument, because they need to know exactly what we want to filter or select as a result.

Ways to 'do' SQL..

SQL is a data creation+manipulation language, so it's best learned HANDS ON (not just by looking at slides and reading about the syntax) - you need access to a relational database where you can **create tables, enter data in them and do queries on the data** (tables ← data ← queries).

There are three ways to get your hands on a DB: use a browser page [a server runs the DB software, you simply access it from a page]; install a DB locally on your laptop/tablet/phone; use a 'cloud-based' DB [this is similar to, but more powerful than, accessing a DB via a web page].

1. **Browser-based SQL IDEs**

An easy way to practice running SQL queries is to use browser-based interfaces (nothing to download, no login needed) to create/query databases. Here are some sites that provide this form of access:
* SQL Tutorial: http://www.w3schools.com/sql/default.asp
* ideone: http://www.ideone.com
* sqlfiddle: http://sqlfiddle.com/
* Code School's Try SQL: http://campus.codeschool.com/courses/try-sql/contents
* SQLZOO: http://sqlzoo.net/ - has extensive tutorials
* another tutorial: http://www.sqltutorial.org/
* w3resource: http://www.w3resource.com/sql-exercises/ - more tutorials
* Khan Academy: https://www.khanacademy.org/computer-programming/new/sql
* 'Online SQL interpreter': https://sql-js.github.io/sql.js/examples/GUI/ - JavaScript-based! [here is the source: https://cdnjs.com/libraries/sql.js ; more info: https://sql.js.org/#/]
* small, simple, fun examples: https://sql-playground.wizardzines.com/ [to go with this delightful booklet]

Below are examples for three of the above links.

Bring up the w3schools Tryit Editor , enter and run the following pair of sql command sets one at a time (these commands run on a set of pre-installed (by w3schools.com) collection of tables called the Northwind database) :

```
SELECT City FROM Customers
WHERE Country="Germany";

SELECT City, CustomerID FROM Customers
WHERE Country="Germany" AND CustomerID>60;
```

In ideone , select SQLite (for the choice of language), then enter and run this:

```
-- warmup
create table tble(str varchar(20));
insert into tble values ('Hello world!'),('SQL is fun!!'
```

```
);
select * from tble;


-- recipes
CREATE TABLE recipes (
recipe_id INT NOT NULL,
recipe_name VARCHAR(30) NOT NULL,
PRIMARY KEY (recipe_id),
UNIQUE (recipe_name)
);

INSERT INTO recipes
(recipe_id, recipe_name)
VALUES
(1,"Tacos"),
(2,"Tomato Soup"),
(3,"Grilled Cheese");


-- quick check
SELECT recipe_name
FROM recipes;


-- ingredients
CREATE TABLE ingredients (
ingredient_id INT NOT NULL,
ingredient_name VARCHAR(30) NOT NULL,
ingredient_price INT NOT NULL,
PRIMARY KEY (ingredient_id),
UNIQUE (ingredient_name)
);

INSERT INTO ingredients
(ingredient_id, ingredient_name, ingredient_price)
```



```
VALUES
(1, "Beef", 5),
(2, "Lettuce", 1),
(3, "Tomatoes", 2),
(4, "Taco Shell", 2),
(5, "Cheese", 3),
(6, "Milk", 1),
(7, "Bread", 2);


--  recipe_ingredients
CREATE TABLE recipe_ingredients (
recipe_id int NOT NULL,
ingredient_id INT NOT NULL,
amount INT NOT NULL,
PRIMARY KEY (recipe_id,ingredient_id)
);

INSERT INTO recipe_ingredients
(recipe_id, ingredient_id, amount)
VALUES
(1,1,1),
(1,2,2),
(1,3,2),
(1,4,3),
(1,5,1),
(2,3,2),
(2,6,1),
(3,5,1),
(3,7,2);


SELECT *
FROM recipes
ORDER BY recipe_id;
```

Here is the recipes db table and its queries again, this time in sqlfiddle. After the page loads (with our SQL commands above, filled in!), click 'Build Schema' on the left, then click 'Run SQL' on the right [and wait for a few seconds for the output].

2. **Locally installed DBs**

You can install Oracle on your machine, or install the light-weight but powerful DB Browser for SQLite, aka SQLite Browser. On Windows, this offers a 'notebook' environment: https://sqlnotebook.com/ [a mix of SQL and non-SQL commands]. It is also OK to use Postgres, or MySQL; or you could use some other DB...

Here is a quickstart guide for SQLite Browser - we create a table called CosineTable, and fill it with rows containing [angle, cos(angle)] pairs. And here is another clip that shows how to create a simple sqlite DB and query it.

The second clip above, also shows that the .db file that is created, contains data in a SQLite-native binary format [which is public and well-documented]. FYI, there are sqlite API calls in multiple languages that let you access the (binary) data using functions/methods that accept SQL commands as regular strings. This is mixed-language programming that lets us accomplish a lot, because we leverage the power of Java, Python etc., as well as that of SQL. Eg. here's how it works in Python:

```
>>> for row in c.execute('SELECT * FROM stocks ORDER BY price'):
        print row # or we could email this, put it up on an LED display, buy/sell the cheap/pricey stocks..

(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)
(u'2006-04-05', u'BUY', u'MSFT', 1000, 72.0)
```

You could also write a binary reader+parser in C++, Java etc. and extract the sqlite-native data that way - in other words, you'd never lose your data if you store it in a sqlite .db file!

As you can see, sqlite is very popular :)

3. **Cloud-based DBs**

You can also work with a relational DB via the cloud, eg. using Amazon's RDS offering. RDS offers a way to create databases such as Amazon's own Aurora, Oracle, SQL Server, MariaDB, MySQL, Postgres.

The QUICKEST/EASIEST cloud solution is https://bit.io - give it a try! Here is a good way to get started: https://innerjoin.bit.io/sql-snow-day-a-holiday-adventure-in-postgres-basics-with-bit-io-77f294bb9bed Also, check out https://bit.io/integrations - cool!

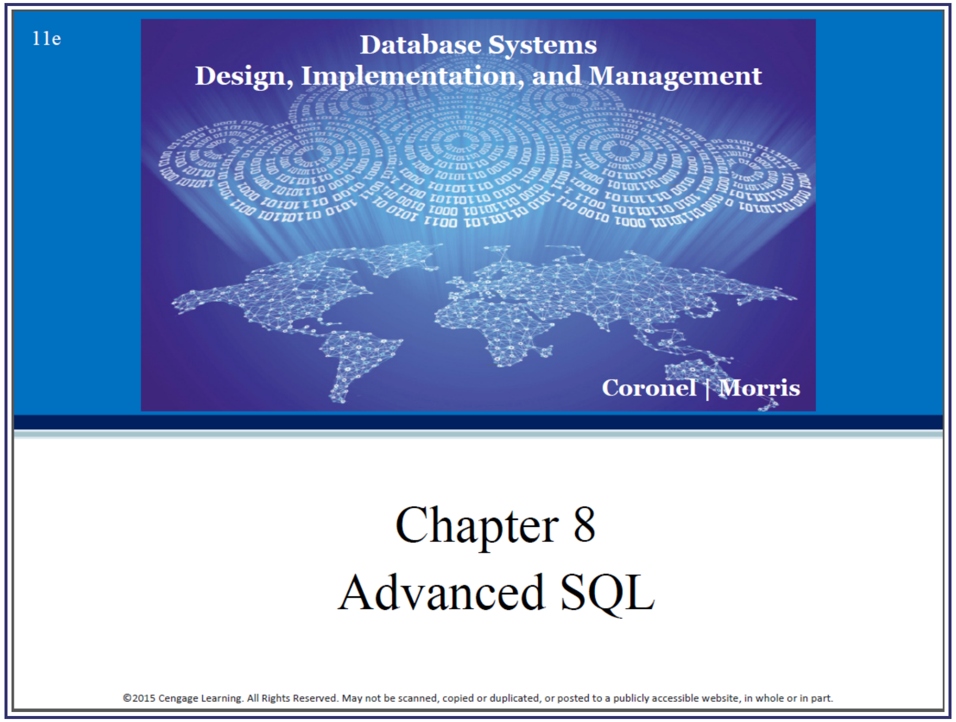
1/50 8:18:11 * * *

← →

More SQL

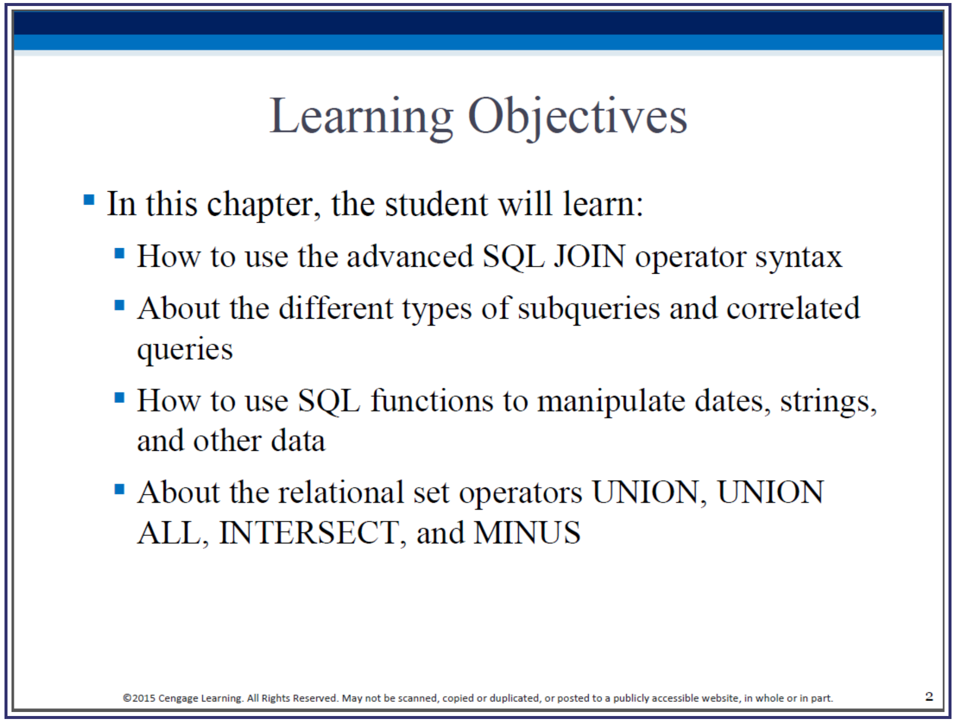
More SQL (1) 2023-03-08, 8:18 PM

Ch.8

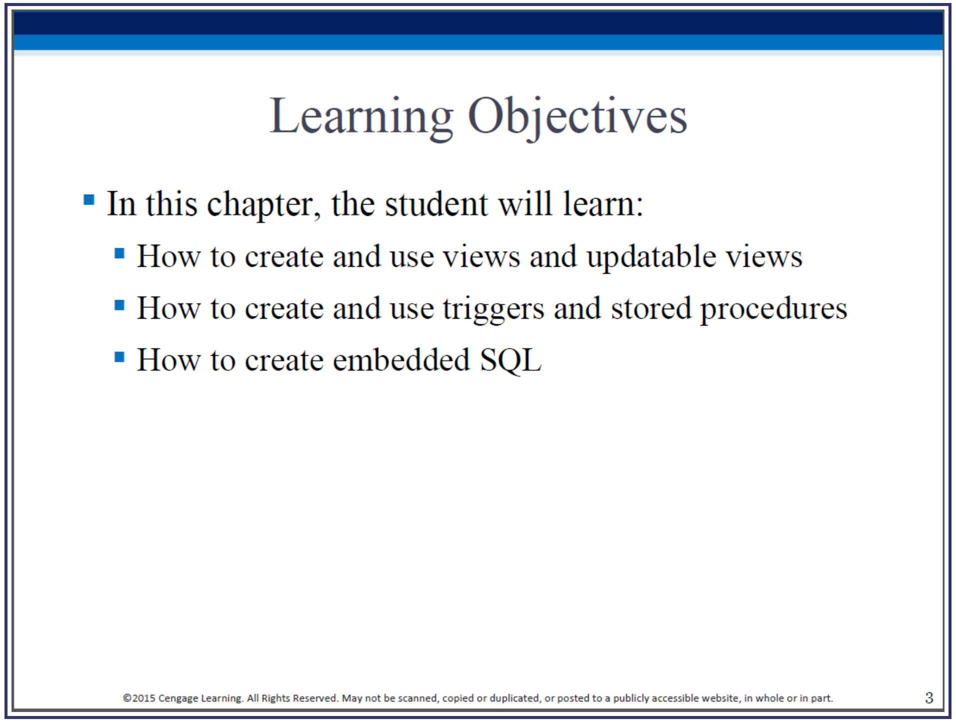

11e
Database Systems
Design, Implementation, and Management
Coronel | Morris
Chapter 8
Advanced SQL
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

More SQL (1) 2023-03-08, 8:18 PM

Objectives



Objectives


Learning Objectives
In this chapter, the student will learn:
How to create and use views and updatable views
How to create and use triggers and stored procedures
How to create embedded SQL
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
3

More SQL (1) 2023-03-08, 8:18 PM

The 'JOIN' operation

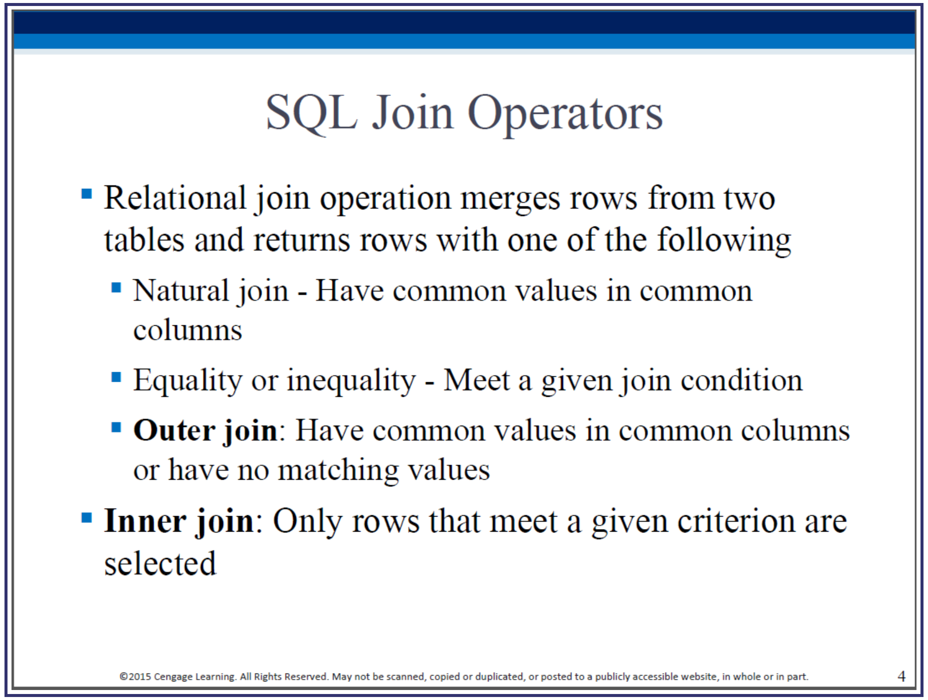
Recall that we looked at examples of joining - entries from two tables, and entries from a single table. These joins were based on 'join conditions'.

It is also possible to join tables using the 'JOIN' keyword..

JOIN conditions

Note that JOINs can be based on != (aka <>), >, <, >= and <= as well, in addition to equality. Eg. to list all students who will be getting a 'A' (uses two inequality comparisons indirectly):

```
// http://www.comp.nus.edu.sg/~ooibc/courses/sql/dml_query_join.htm
SELECT  a.name, a.score
FROM student_scores a, grade_class b
WHERE b.grade = 'A' AND a.score BETWEEN
b.low_end AND b.high_end;
```

Ways to specify JOIN conditions

Table 8.1 - SQL Join Expression Styles

JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
CROSS	CROSS JOIN	SELECT * FROM T1, T2	Returns the Cartesian product of T1 and T2 (old style)
		SELECT * FROM T1 CROSS JOIN T2	Returns the Cartesian product of T1 and T2
INNER	Old-style JOIN	SELECT * FROM T1, T2 WHERE T1.C1=T2.C1	Returns only the rows that meet the join condition in the WHERE clause (old style); only rows with matching values are selected
	NATURAL JOIN	SELECT * FROM T1 NATURAL JOIN T2	Returns only the rows with matching values in the matching columns; the matching columns must have the same names and similar data types
	JOIN USING	SELECT * FROM T1 JOIN T2 USING (C1)	Returns only the rows with matching values in the columns indicated in the USING clause
	JOIN ON	SELECT * FROM T1 JOIN T2 ON T1.C1=T2.C1	Returns only the rows that meet the join condition indicated in the ON clause

Cengage Learning © 2015

 5

Outer vs inner vs full ('both') JOINs

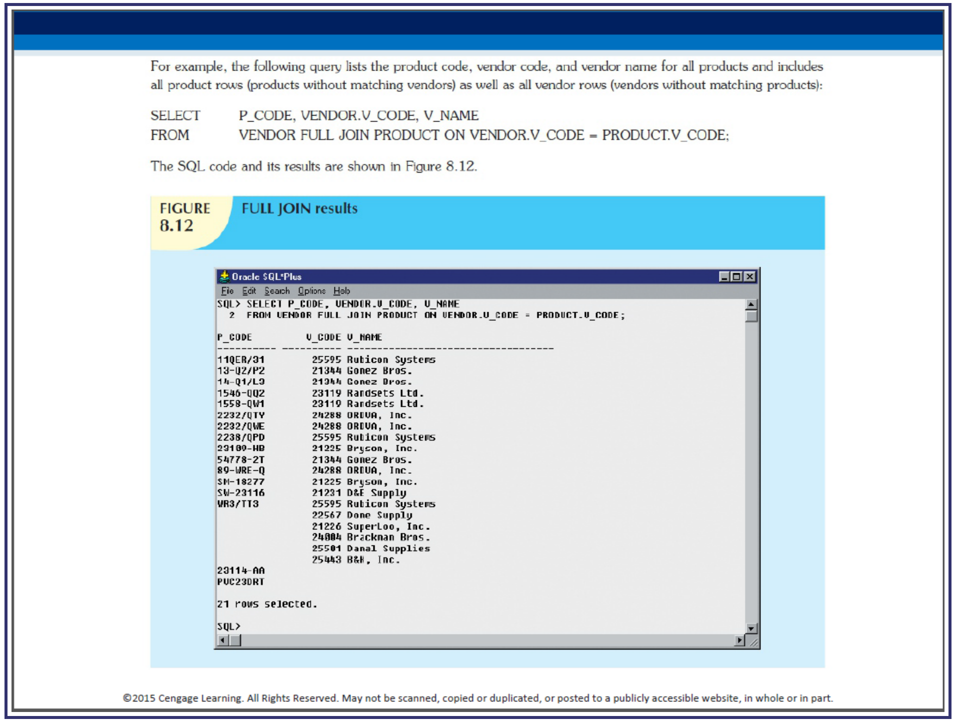
Table 8.1 - SQL Join Expression Styles

JOIN CLASSIFICATION	JOIN TYPE	SQL SYNTAX EXAMPLE	DESCRIPTION
OUTER	LEFT JOIN	SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the left table (T1) with unmatched values
	RIGHT JOIN	SELECT * FROM T1 RIGHT OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from the right table (T2) with unmatched values
	FULL JOIN	SELECT * FROM T1 FULL OUTER JOIN T2 ON T1.C1=T2.C1	Returns rows with matching values and includes all rows from both tables (T1 and T2) with unmatched values

Cengage Learning © 2015

6

Full (left+right outer) JOIN example



For example, the following query lists the product code, vendor code, and vendor name for all products and includes all product rows (products without matching vendors) as well as all vendor rows (vendors without matching products):

```
SELECT    P_CODE, VENDOR.V_CODE, V_NAME
FROM      VENDOR FULL JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;
```

The SQL code and its results are shown in Figure 8.12.

FIGURE 8.12 FULL JOIN results

```
Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT P_CODE, VENDOR.V_CODE, V_NAME
  2  FROM VENDOR FULL JOIN PRODUCT ON VENDOR.V_CODE = PRODUCT.V_CODE;

P_CODE         V_CODE V_NAME
---------- ---------- -----------------------------------
11QER/31        25595 Rubicon Systems
13-Q2/P2        21344 Gomez Bros.
14-Q1/L3        21344 Gomez Bros.
1546-QQ2        23119 Randsets Ltd.
1558-QW1        23119 Randsets Ltd.
2232/QTY        24288 ORDVA, Inc.
2232/QWE        24288 ORDVA, Inc.
2238/QPD        25595 Rubicon Systems
23109-HB        21225 Bryson, Inc.
54778-2T        21344 Gomez Bros.
89-WRE-Q        24288 ORDVA, Inc.
SM-18277        21225 Bryson, Inc.
SW-23116        21231 D&E Supply
WR3/TT3         25595 Rubicon Systems
                22567 Dome Supply
                21226 SuperLoo, Inc.
                24004 Brackman Bros.
                25501 Damal Supplies
                25443 B&K, Inc.
23114-AA
PVC23DRT

21 rows selected.

SQL>
```

'SELECT' subqueries

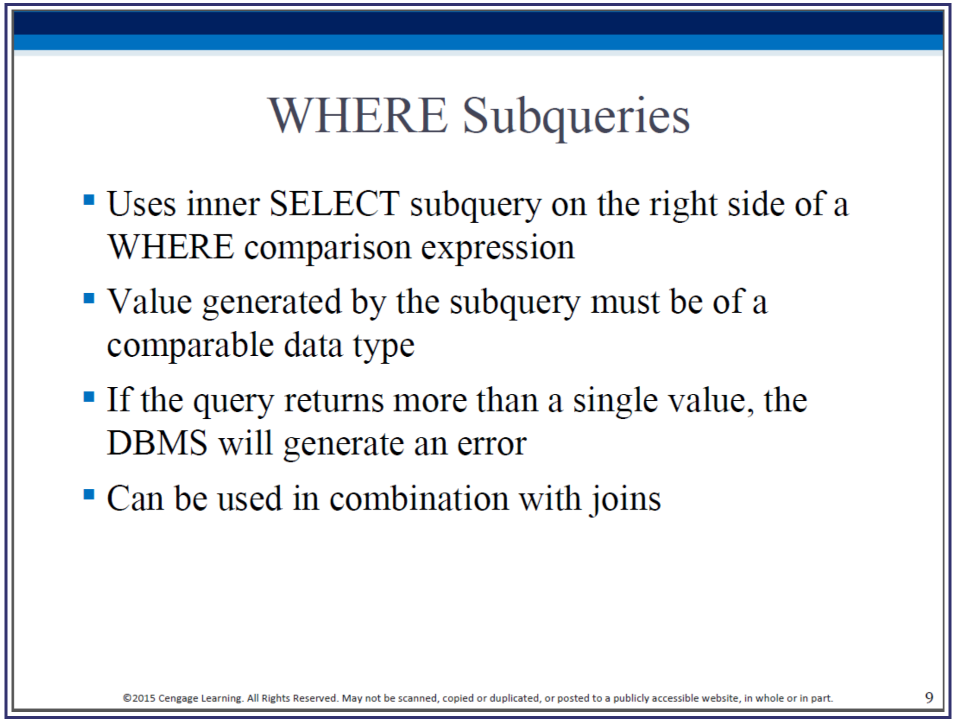
SELECT SUBQUERY EXAMPLES	EXPLANATION
INSERT INTO PRODUCT SELECT * FROM P;	Inserts all rows from Table P into the PRODUCT table. Both tables must have the same attributes. The subquery returns all rows from Table P.
UPDATE PRODUCT SET P_PRICE = (SELECT AVG(P_PRICE) FROM PRODUCT) WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_AREACODE = '615')	Updates the product price to the average product price, but only for products provided by vendors who have an area code equal to 615. The first subquery returns the average price; the second subquery returns the list of vendors with an area code equal to 615.
DELETE FROM PRODUCT WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_AREACODE = '615')	Deletes the PRODUCT table rows provided by vendors with an area code equal to 615. The subquery returns the list of vendor codes with an area code equal to 615.

Subqueries and Correlated Queries

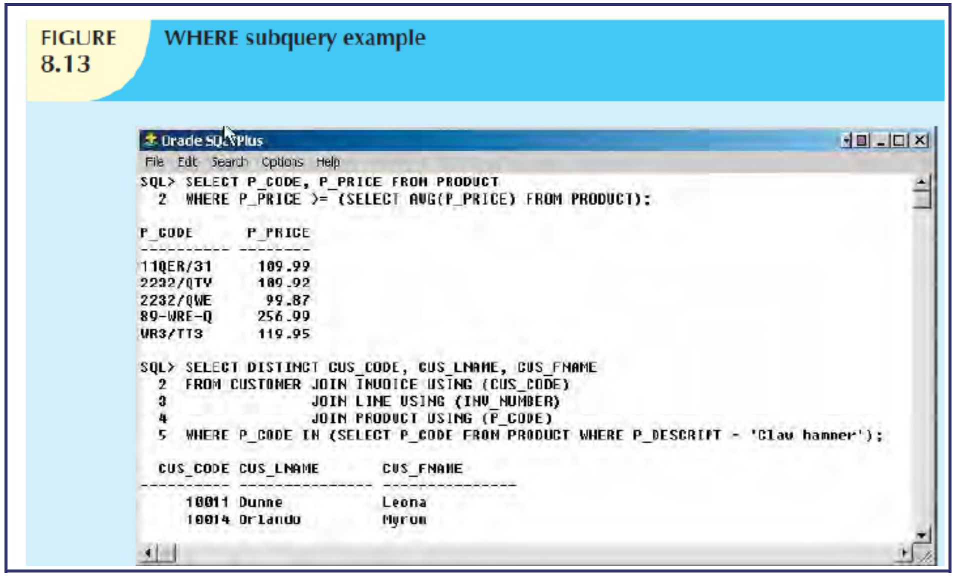
- Subquery is a query inside another query
- Subquery can return:
  - One single value - One column and one row
  - A list of values - One column and multiple rows
  - A virtual table - Multicolumn, multirow set of values
  - No value - Output of the outer query might result in an error or a null empty set

 8

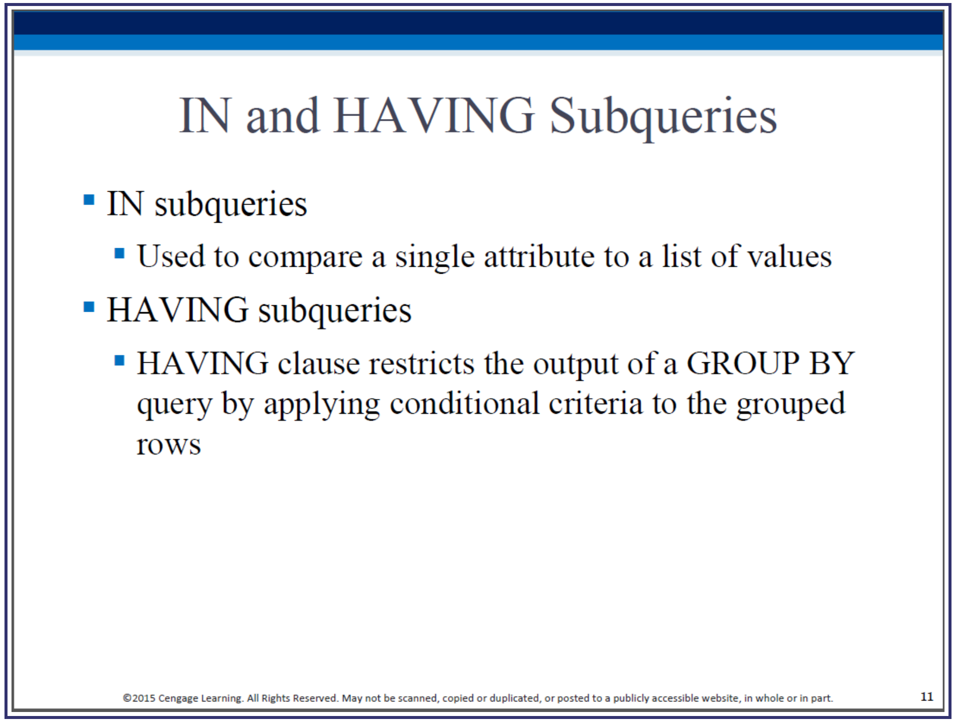
'WHERE' subqueries



WHERE subquery example



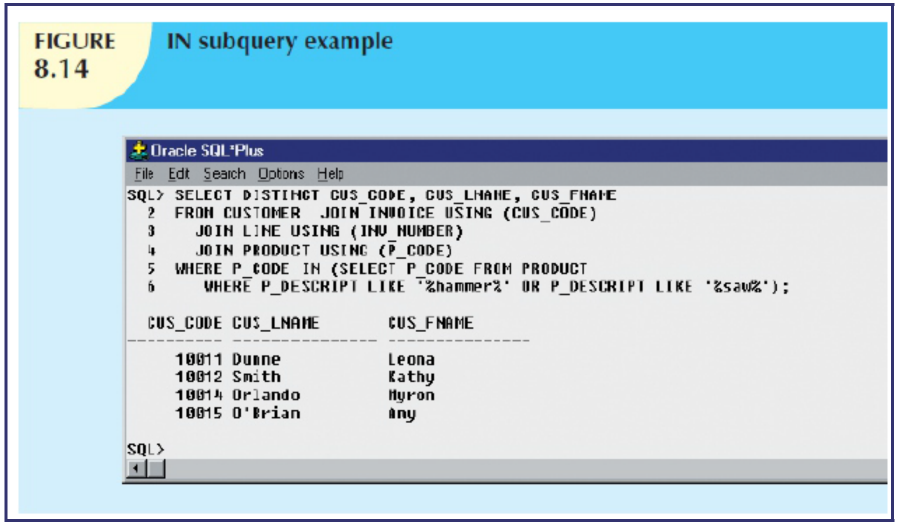
IN, HAVING subqueries



More SQL (1) 2023-03-08, 8:18 PM

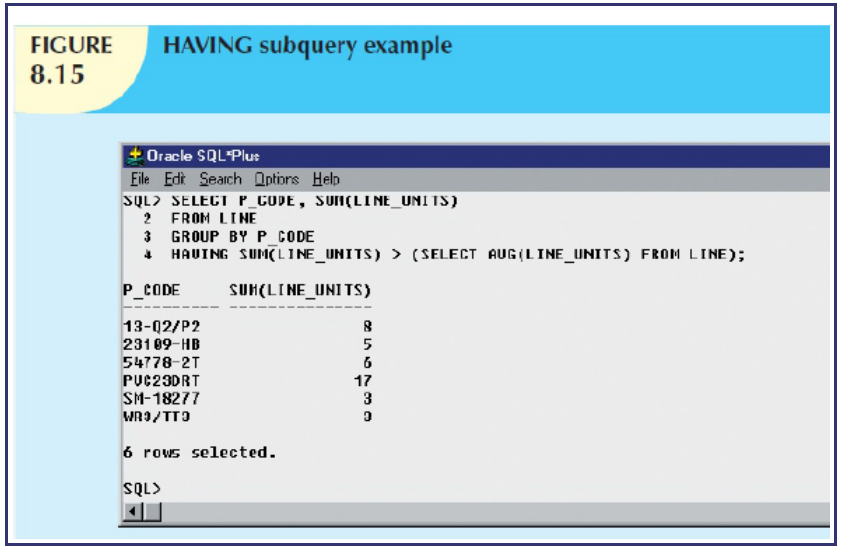
'IN' subqueries

Compare against a LIST of values..



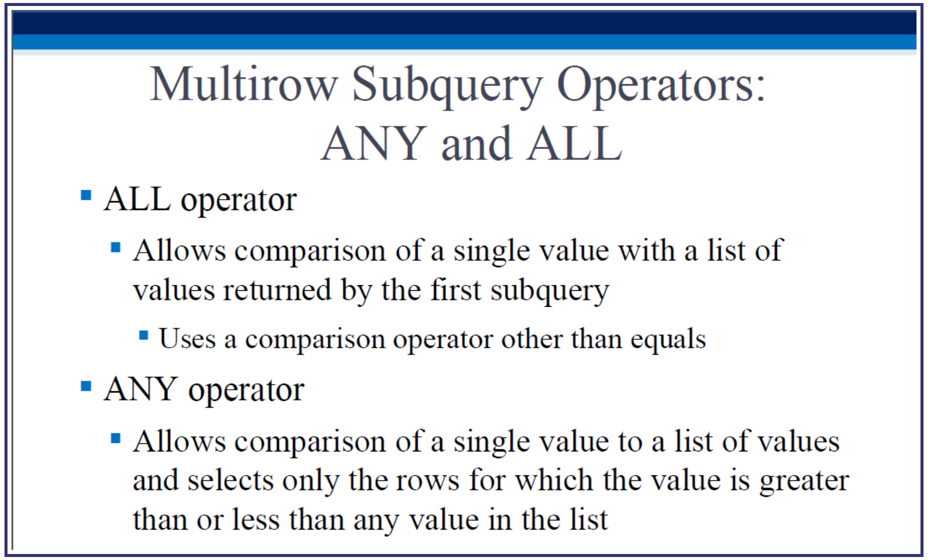
'HAVING' subqueries

As we saw earlier, this restricts the results of a GROUP BY clause. Eg. here's how to list all products sold, whose totals are greater than the average quantity sold:



ALL, ANY (inequality comparisons)

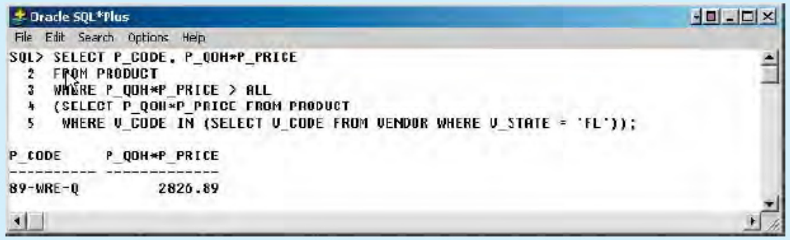
Recall that 'IN' is an equality comparison against a list. To do inequality **comparison of a value against a list of values** (eg. need to be greater than ALL, need to be less than ANY..), use ALL, ANY.



ALL, ANY

Eg. "which products do we own [in our store], whose value is more than ALL other products's values supplied by vendors in Florida?"

FIGURE 8.16 Multirow subquery operator example



Note that 'greater than ALL' is eqvt to 'greater than the largest of'. 'ALL' is used to select rows [plural in general] that comparison-succeed against all values in a list.

Another powerful operator is the ANY multirow operator (the near cousin of the ALL multirow operator). The ANY operator allows you to compare a single value to a list of values, selecting only the rows for which the inventory cost is greater than any value of the list or less than any value of the list. You could use the equal to ANY operator, which would be the equivalent of the IN operator.

'ANY' is used to select rows [plural in general] that comparison-succeed with any value in a list.

Note that '= ANY(list of values)' is equivalent to the 'IN' operator (which is itself equivalent to multiple == conditions joined by ORs). So the following are all equivalent, for a given value of 'M':

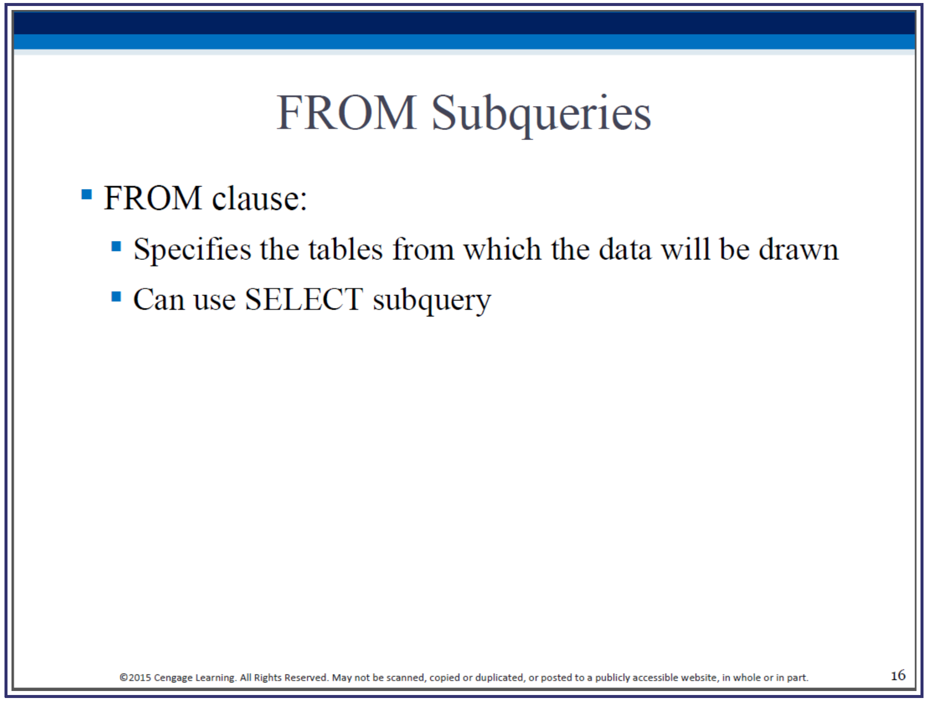
```
(M==6) OR (M==8) OR (M==10)

M IN (6,8,10)

M = ANY (6,8,10)
```

So loosely speaking, ALL is equivalent to AND, and ANY is equivalent to OR.

More SQL (1) 2023-03-08, 8:18 PM

'FROM' subqueries

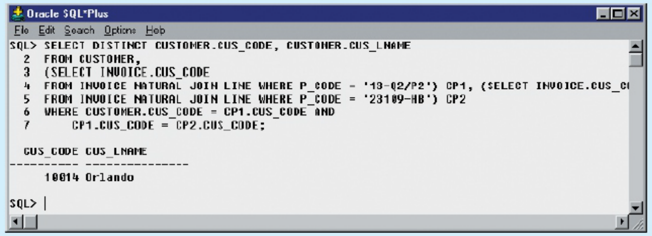
A SELECT query that appears in FROM, creates a **virtual table** against which the main query can run.



FROM subquery example

All customers who bought both specified products

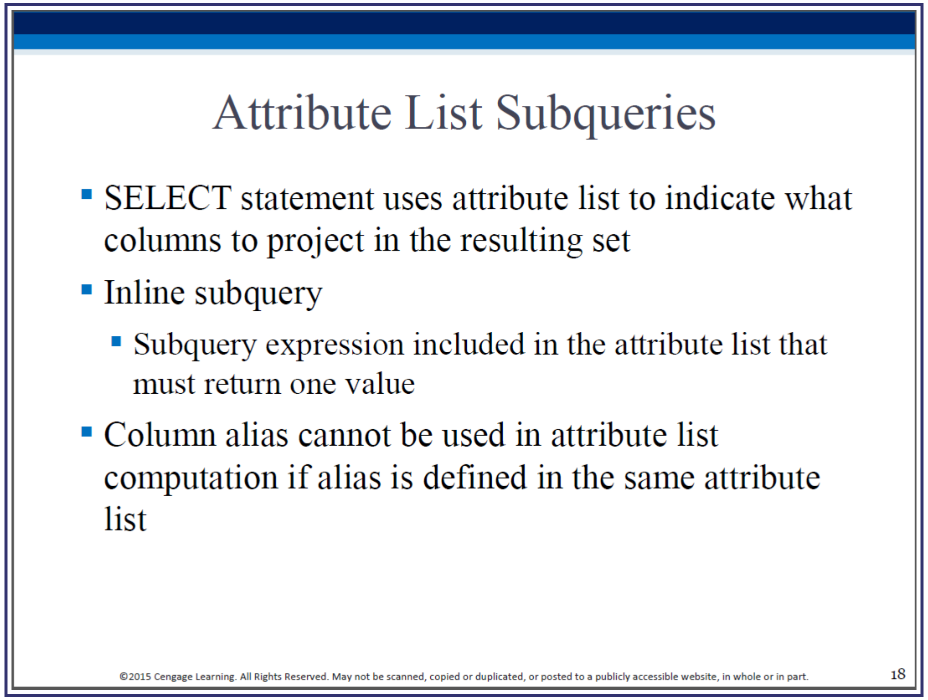
FIGURE 8.17 FROM subquery example



Attribute list subqueries

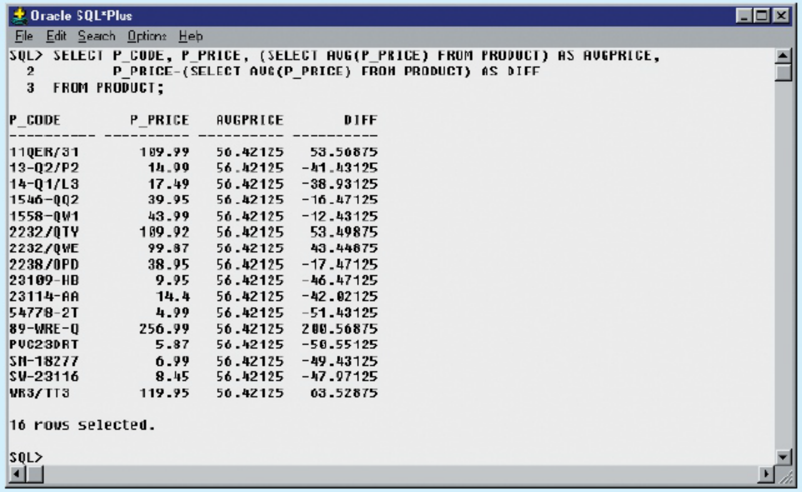
These subqueries determine what columns get output by the main query - they can be actual (existing) columns or computed columns or results of aggregate functions.

These are also known as 'column subqueries' or 'inline subqueries'.



Attribute subquery example

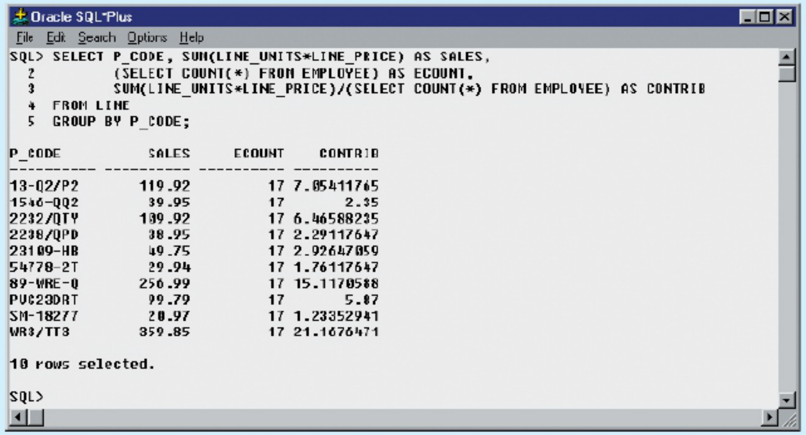
FIGURE 8.18 Inline subquery example



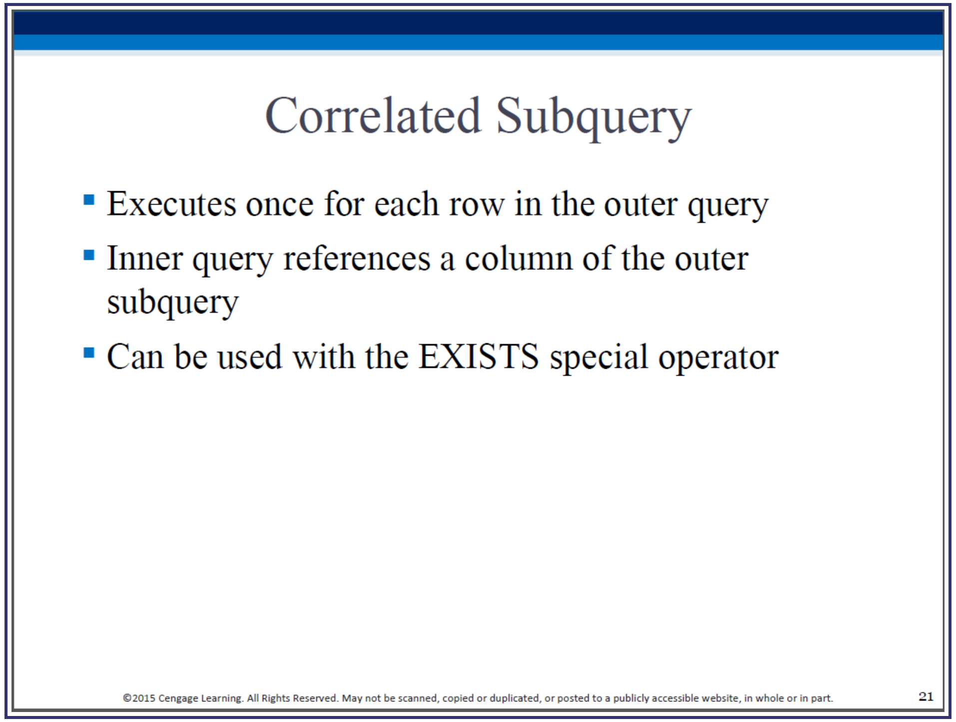
```
Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT P_CODE, P_PRICE, (SELECT AVG(P_PRICE) FROM PRODUCT) AS AVGPRICE,
  2         P_PRICE-(SELECT AVG(P_PRICE) FROM PRODUCT) AS DIFF
  3  FROM PRODUCT;

P_CODE        P_PRICE   AVGPRICE       DIFF
---------- ---------- ---------- ----------
11QER/31       109.99   56.42125   53.56875
13-Q2/P2        14.99   56.42125  -41.43125
14-Q1/L3        17.49   56.42125  -38.93125
1546-QQ2        39.95   56.42125  -16.47125
1558-QW1        43.99   56.42125  -12.43125
2232/QTY       109.92   56.42125   53.49875
2232/QWE        99.87   56.42125   43.44875
2238/QPD        38.95   56.42125  -17.47125
23109-HB         9.95   56.42125  -46.47125
23114-AA         14.4   56.42125  -42.02125
54778-2T         4.99   56.42125  -51.43125
89-WRE-Q       256.99   56.42125  200.56875
PVC23DRT         5.87   56.42125  -50.55125
SM-18277         6.99   56.42125  -49.43125
SW-23116         8.45   56.42125  -47.97125
WR3/TT3        119.95   56.42125   63.52875

16 rows selected.

SQL>
```

Another attribute subquery example

FIGURE 8.19 Another example of an inline subquery



Correlated subqueries



In a correlated subquery, the inner (sub) query is repeatedly run, for each row of the outer query! The inner is said to be (co-)related with the outer query when it references a column in the outer query's table. This is in effect, like a double (nested) 'for' loop..

```
for each row in OUTER table
  run subquery on EACH row in INNER tab
le, gather
  results, use in outer table's query
```


Here is the Wikipedia entry on correlated subqueries. This is the example shown there [select employees who make more than the average salary for their department]:

```
SELECT employee_number, name
  FROM employees AS Bob
  WHERE salary > (
    SELECT AVG(salary)
      FROM employees
      WHERE department = Bob.department)
;
```

In the above, the outer query "passes in", for each employee (each row), the employee's dept. [which the inner query refers to as Bob.department]. The inner query selects all salaries for that dept., computes the average, compares it with the passed-in employee's salary; if the test passes, the outer query selects the employee's # and name.

Correlated subqueries [cont'd]

Until now, all subqueries you have learned execute independently. That is, each subquery in a command sequence executes in a serial fashion, one after another. The inner subquery executes first; its output is used by the outer query, which then executes until the last outer query executes (the first SQL statement in the code).

In contrast, a **correlated subquery** is a subquery that executes once for each row in the outer query. That process is similar to the typical nested loop in a programming language. For example:

```
FOR X = 1 TO 2
    FOR Y = 1 TO 3
        PRINT "X = "X, "Y = "Y
    END
END
```

1. It initiates the outer query.
2. For each row of the outer query result set, it executes the inner query by passing the outer row to the inner query.

That process is the opposite of that of the subqueries as you have already seen. The query is called a *correlated* subquery because the inner query is *related* to the outer query by the fact that the inner query references a column of the outer subquery.

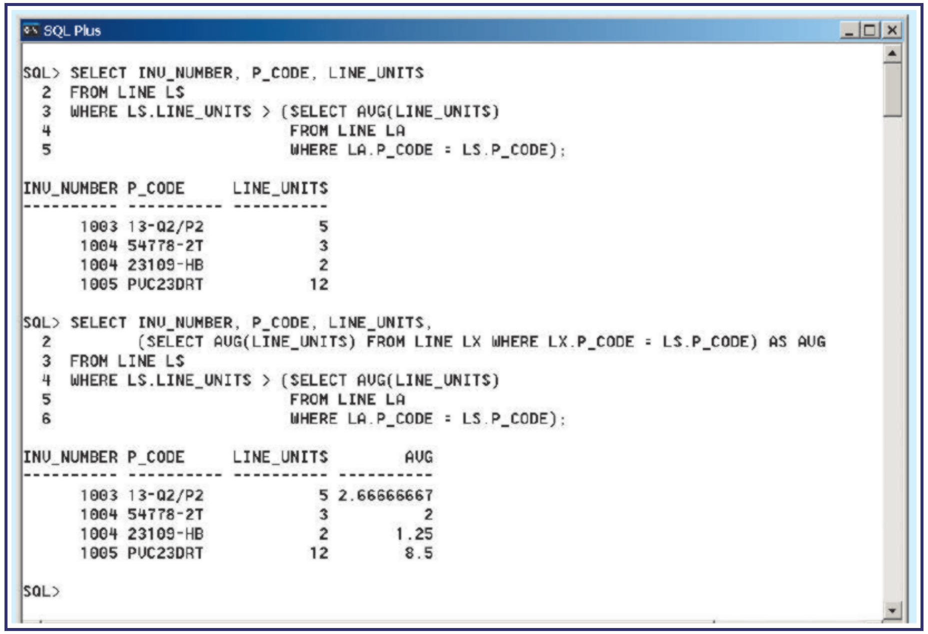
Correlated subquery examples

To see the correlated subquery in action, suppose that you want to know all product sales in which the units sold value is greater than the average units sold value *for that product* (as opposed to the average for *all* products). In that case, the following procedure must be completed:

1. Compute the average units sold for a product.
2. Compare the average computed in Step 1 to the units sold in each sale row, and then select only the rows in which the number of units sold is greater.

The following correlated query completes the preceding two-step process:

```
SELECT   INV_NUMBER, P_CODE, LINE_UNITS
FROM     LINE LS
WHERE    LS.LINE_UNITS > (SELECT AVG(LINE_UNITS)
                          FROM LINE LA
                          WHERE LA.P_CODE = LS.P_CODE);
```

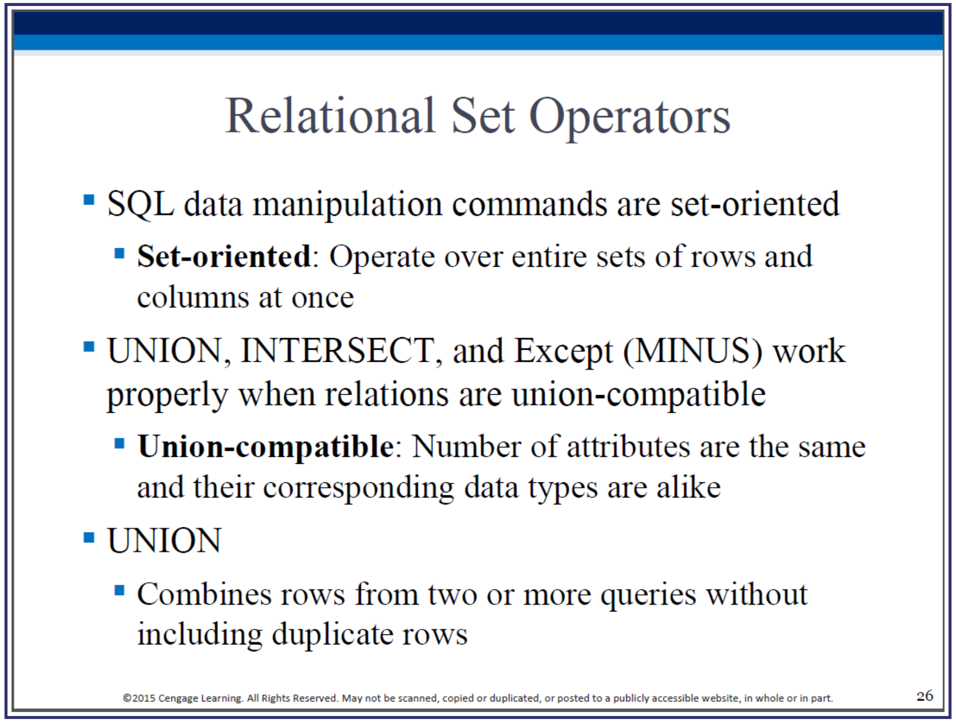


In the top query and its result in Figure 8.14, note that the LINE table is used more than once, so you must use table aliases. In this case, the inner query computes the average units sold of the product that matches the P_CODE of the outer query P_CODE. That is, the inner query runs once, using the first product code found in the outer LINE table, and returns the average sale for that product. When the number of units sold in the outer LINE row is greater than the average computed, the row is added to the output. Then the inner query runs again, this time using the second product code found in the outer LINE table. The process repeats until the inner query has run for all rows in the outer LINE table. In this case, the inner query will be repeated as many times as there are rows in the outer query.

To verify the results and to provide an example of how you can combine subqueries, you can add a correlated inline subquery to the previous query. (See the second query and its results in Figure 8.14.) As you can see, the new query contains a correlated inline subquery that computes the average units sold for each product. You not only get an answer, you can also verify that the answer is correct.

In the second query above, we have TWO correlated subqueries (that are identical), both of which need to run for every row of the main query.

UNION, INTERSECTION, DIFFERENCE



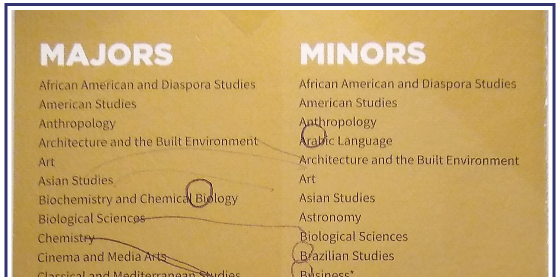
UNION, INTERSECTION, DIFFERENCE [cont'd]

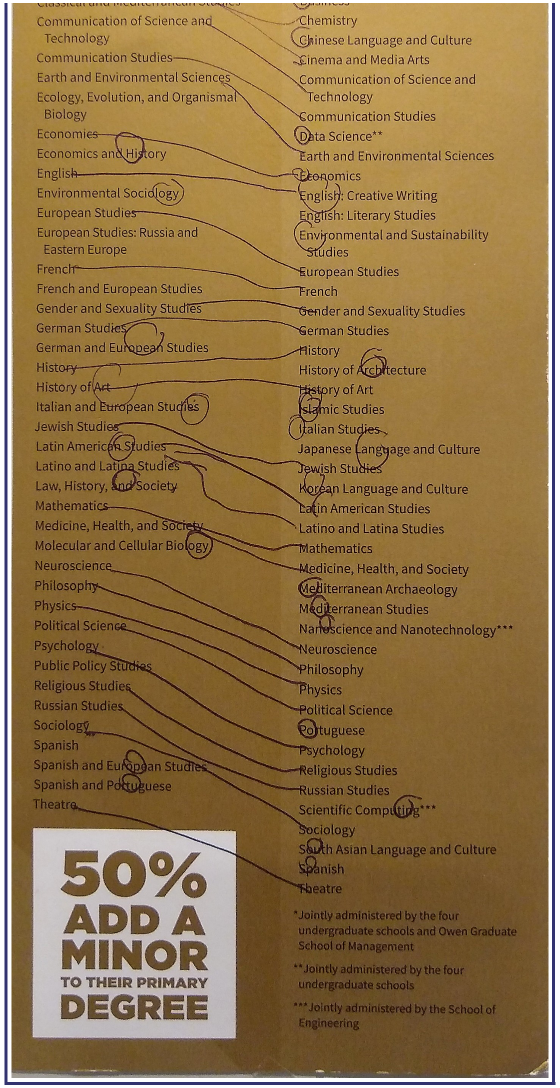
Relational Set Operators

- Syntax - query UNION query
- UNION ALL
  - Produces a relation that retains duplicate rows
  - Can be used to unite more than two queries
- INTERSECT
  - Combines rows from two queries, returning only the rows that appear in both sets
  - Syntax - query INTERSECT query

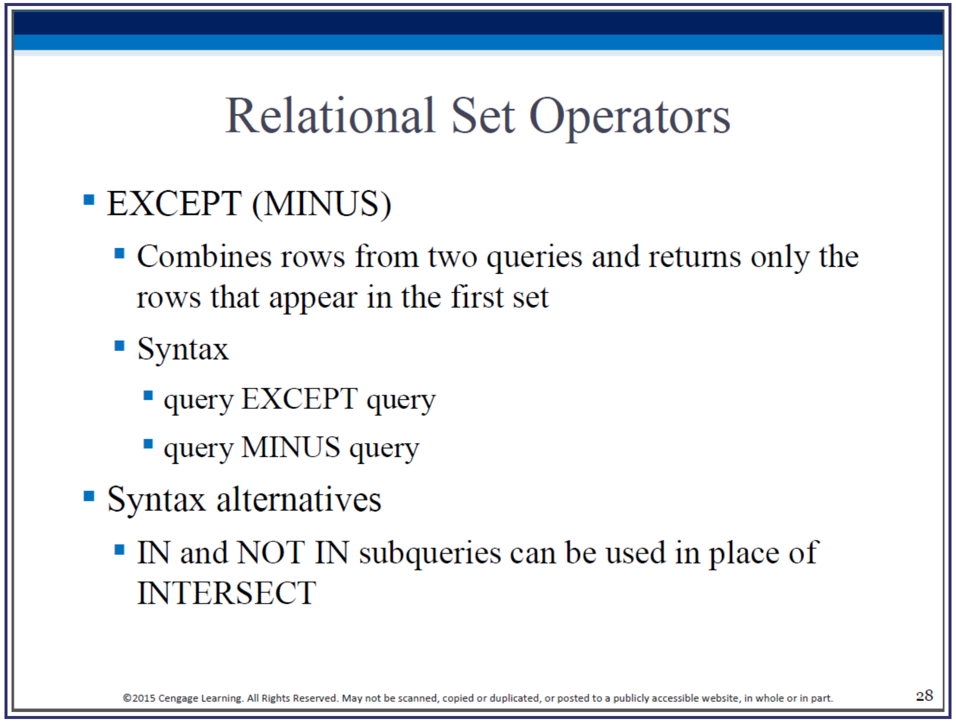
 27

If the columns below are in two different tables, the interection of them would list items in both (eg. History, Physics...):

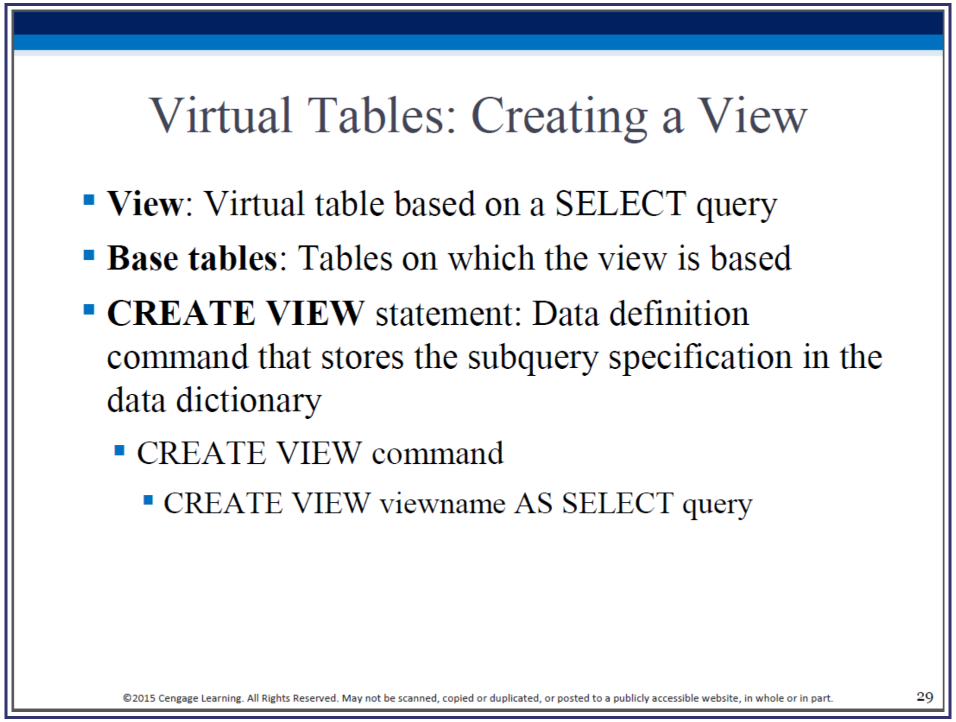




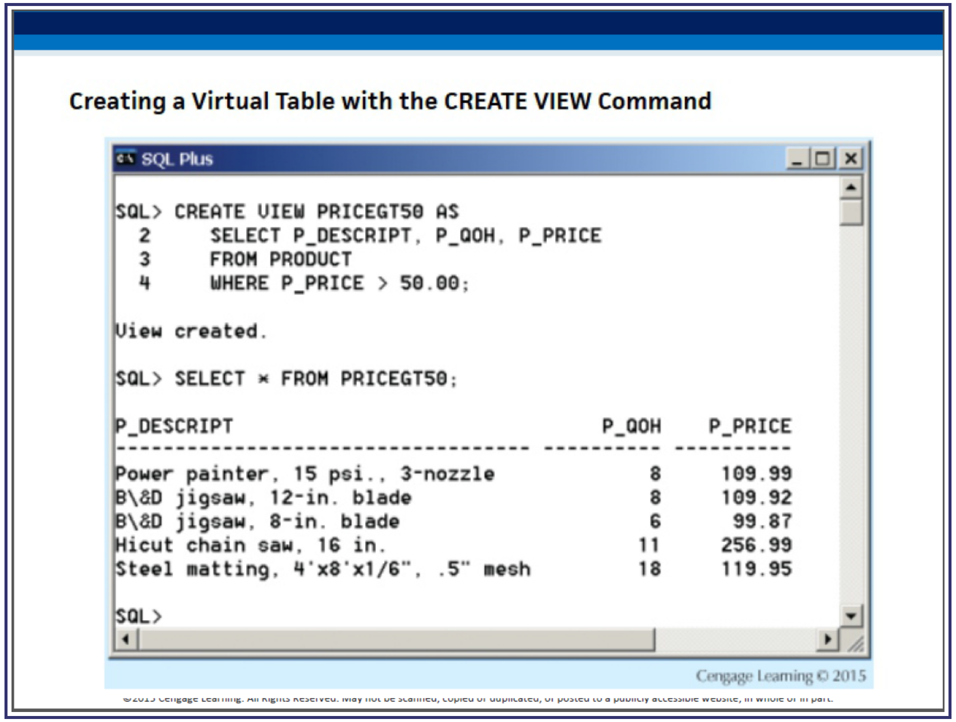
UNION, INTERSECTION, DIFFERENCE [cont'd]



VIEWs


Virtual Tables: Creating a View
View: Virtual table based on a SELECT query
Base tables: Tables on which the view is based
CREATE VIEW statement: Data definition command that stores the subquery specification in the data dictionary
CREATE VIEW command
CREATE VIEW viewname AS SELECT query
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
29

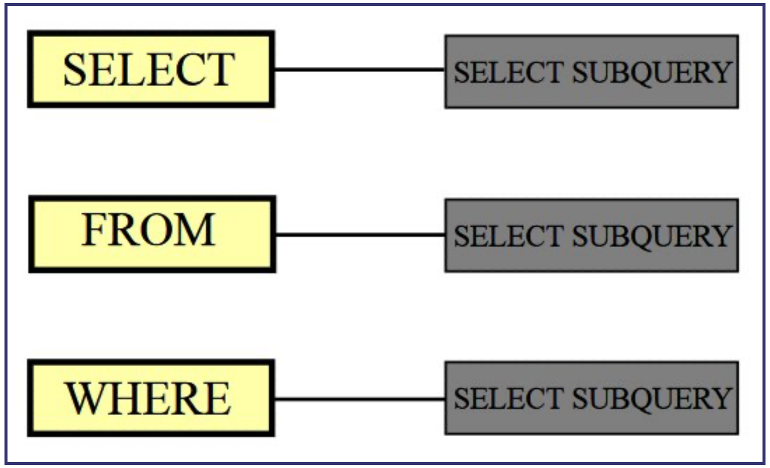
VIEW example



Queries: summary

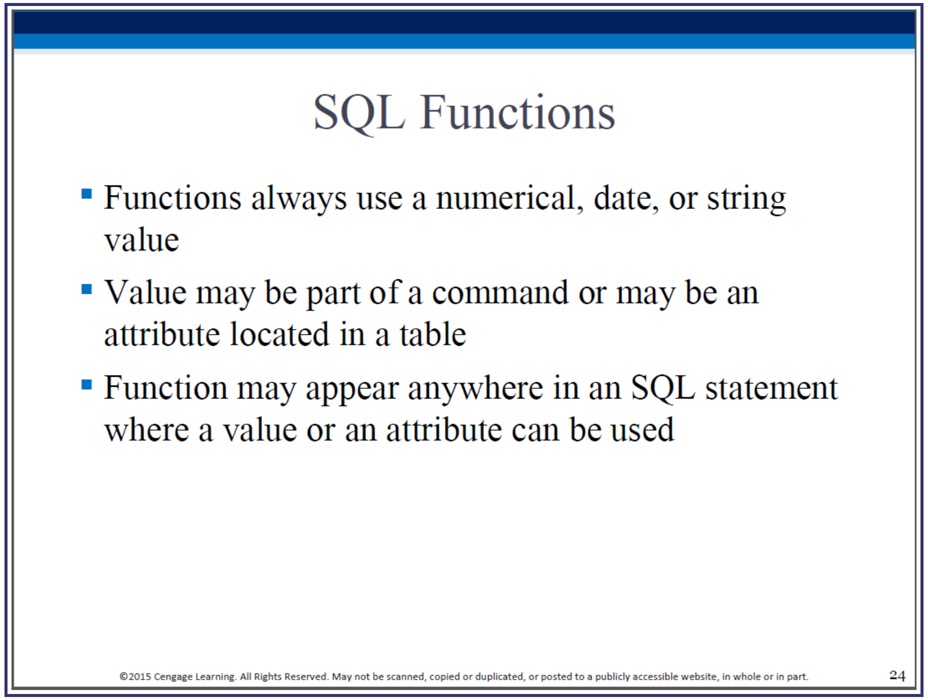
We looked at several variations of queries and subqueries (SELECT, WHERE, HAVING, IN..).

Most interestingly, a SELECT subquery can appear at the top (SELECT), middle (FROM) or bottom (WHERE) of a parent query, which provides a flexible way to express **complex logic** (since such subqueries can be recursively nested):

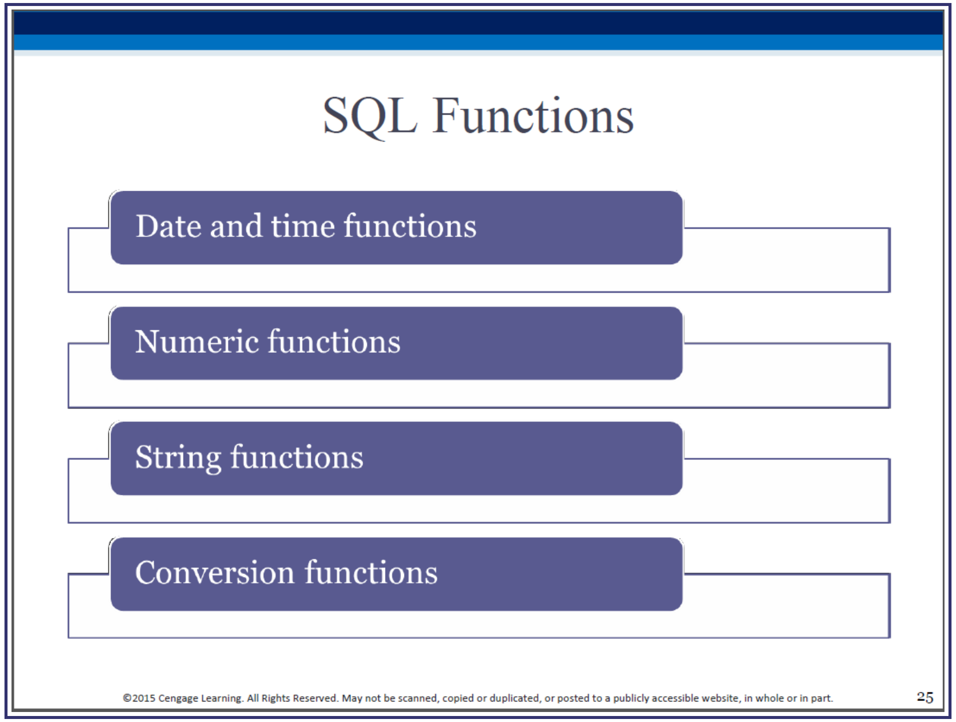


SQL functions (built-ins)

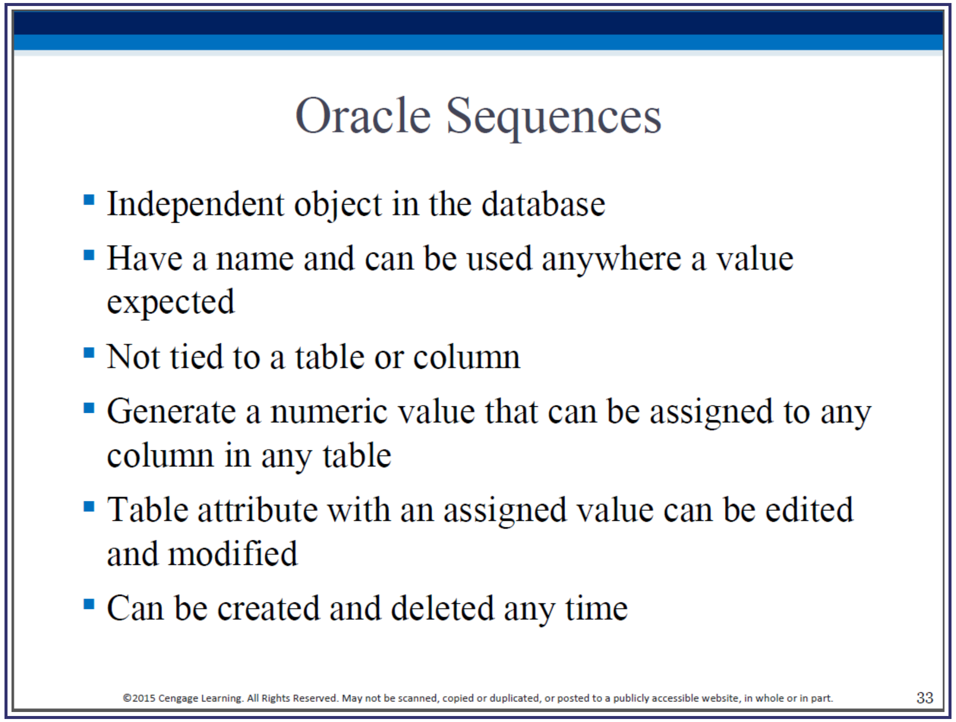
Functions **return values...**



SQL functions [cont'd]



Sequences


Oracle Sequences
Independent object in the database
Have a name and can be used anywhere a value expected
Not tied to a table or column
Generate a numeric value that can be assigned to any column in any table
Table attribute with an assigned value can be edited and modified
Can be created and deleted any time
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
33

Sequence creation example

Figure 8.27 - Oracle Sequence

SQL Plus

```
SQL> CREATE SEQUENCE CUS_CODE_SEQ START WITH 20010 NOCACHE;

Sequence created.

SQL> CREATE SEQUENCE INV_NUMBER_SEQ START WITH 4010 NOCACHE;

Sequence created.

SQL> SELECT * FROM USER_SEQUENCES;

SEQUENCE_NAME                   MIN_VALUE  MAX_VALUE INCREMENT_BY C O CACHE_SIZE LAST_NUMBER
------------------------------ ---------- ---------- ------------ - - ---------- -----------
CUS_CODE_SEQ                            1 1.0000E+27            1 N N          0       20010
INV_NUMBER_SEQ                          1 1.0000E+27            1 N N          0        4010

SQL>
```

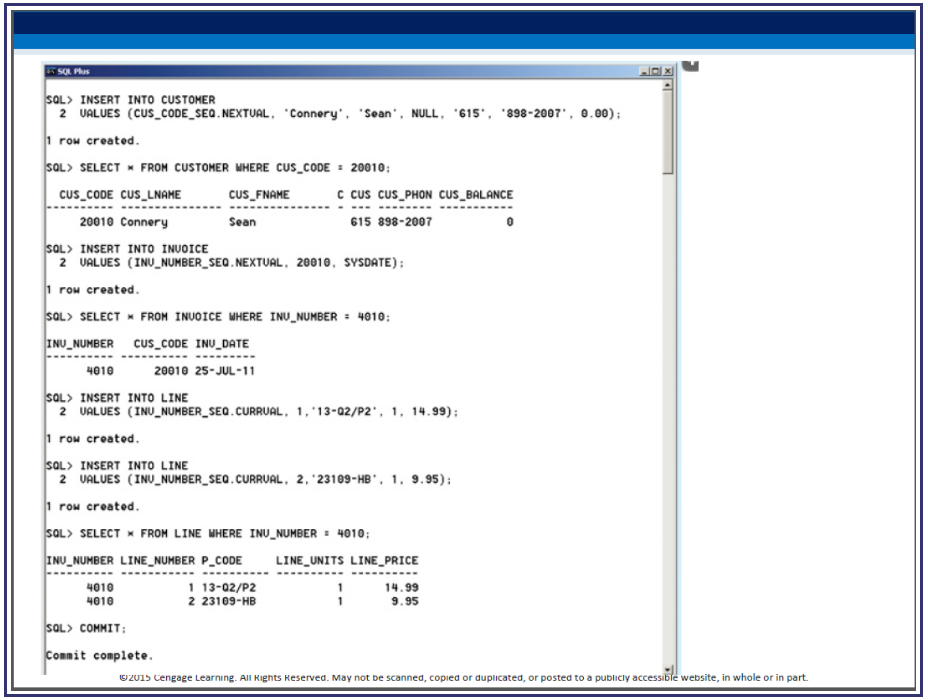
Cengage Learning © 2015

```
INSERT INTO CUSTOMER
VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery', 'Sean', NULL, '615', '898-2007', 0.00);
```

 34

Sequence: NEXTVAL, CURRVAL

NEXTVAL returns the current value, **then** does ++ (ie. it does 'post increment', ie. C++ as opposed to ++C); CURRVAL on the other hand, just fetches the current value (does not ++ it).



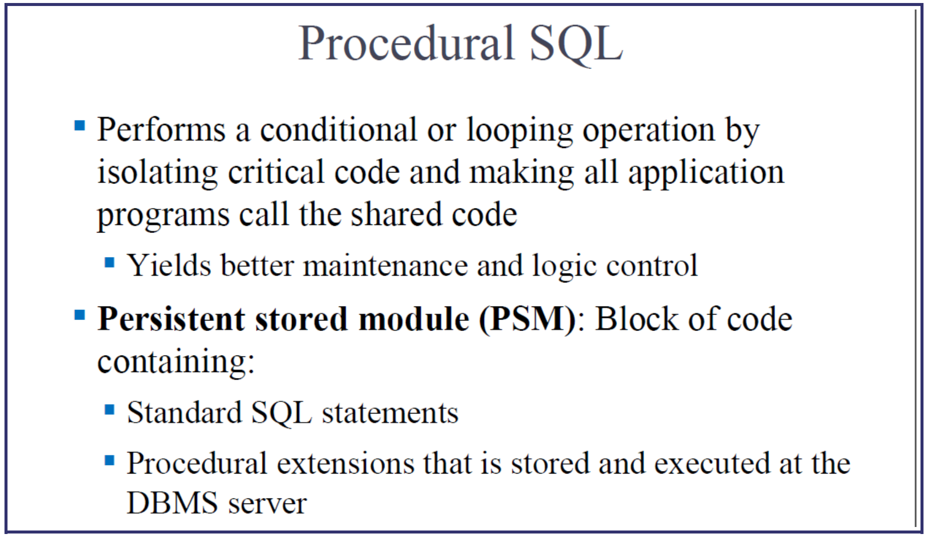

```
SQL> INSERT INTO CUSTOMER
  2  VALUES (CUS_CODE_SEQ.NEXTVAL, 'Connery', 'Sean', NULL, '615', '898-2007', 0.00);

1 row created.

SQL> SELECT * FROM CUSTOMER WHERE CUS_CODE = 20010;

  CUS_CODE CUS_LNAME       CUS_FNAME       C CUS CUS_PHON CUS_BALANCE
---------- --------------- --------------- - --- -------- -----------
     20010 Connery         Sean              615 898-2007           0

SQL> INSERT INTO INVOICE
  2  VALUES (INV_NUMBER_SEQ.NEXTVAL, 20010, SYSDATE);

1 row created.

SQL> SELECT * FROM INVOICE WHERE INV_NUMBER = 4010;

INV_NUMBER   CUS_CODE INV_DATE
---------- ---------- ---------
      4010      20010 25-JUL-11

SQL> INSERT INTO LINE
  2  VALUES (INV_NUMBER_SEQ.CURRVAL, 1,'13-Q2/P2', 1, 14.99);

1 row created.

SQL> INSERT INTO LINE
  2  VALUES (INV_NUMBER_SEQ.CURRVAL, 2,'23109-HB', 1, 9.95);

1 row created.

SQL> SELECT * FROM LINE WHERE INV_NUMBER = 4010;

INV_NUMBER LINE_NUMBER P_CODE     LINE_UNITS LINE_PRICE
---------- ----------- ---------- ---------- ----------
      4010           1 13-Q2/P2            1      14.99
      4010           2 23109-HB            1       9.95

SQL> COMMIT;

Commit complete.
```

Procedural Language SQL (PL/SQL)

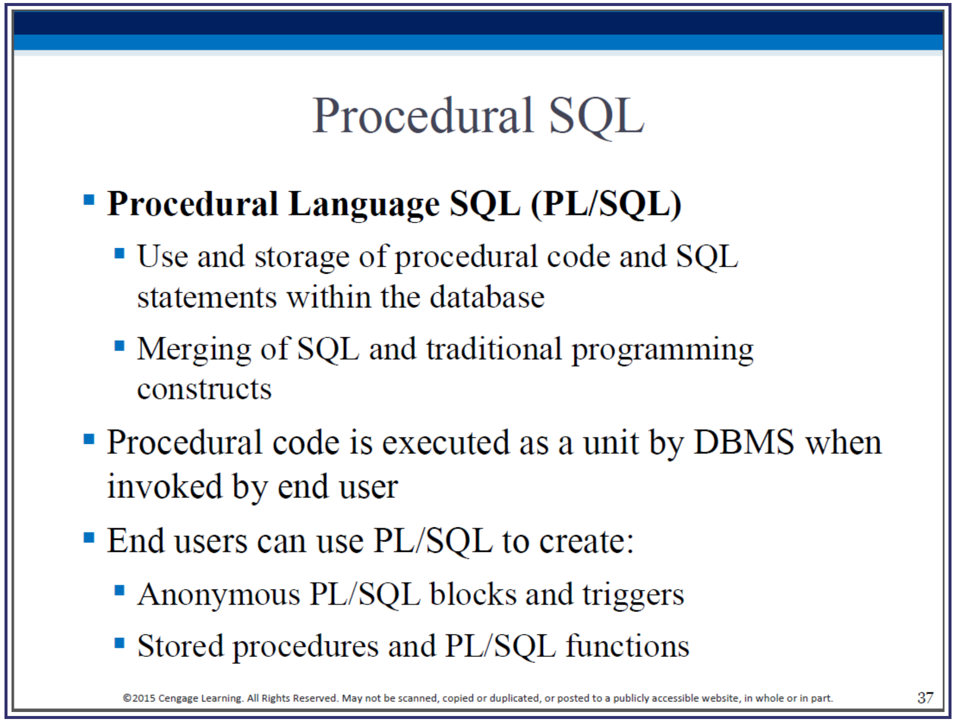
PL/SQL involves extra (augmented) syntax that lets us do looping, branching, variable declaration and function declaration - these are of course not possible using 'plain' SQL.

PL/SQL can be used to create:

- **blocks of code** for one-time execution
- **triggers** - callbacks to invoke
- **stored procedures** - named procedures (no return values) for repeated calling
- **stored functions** - named functions (with return values) for repeated calling



PL/SQL [cont'd]

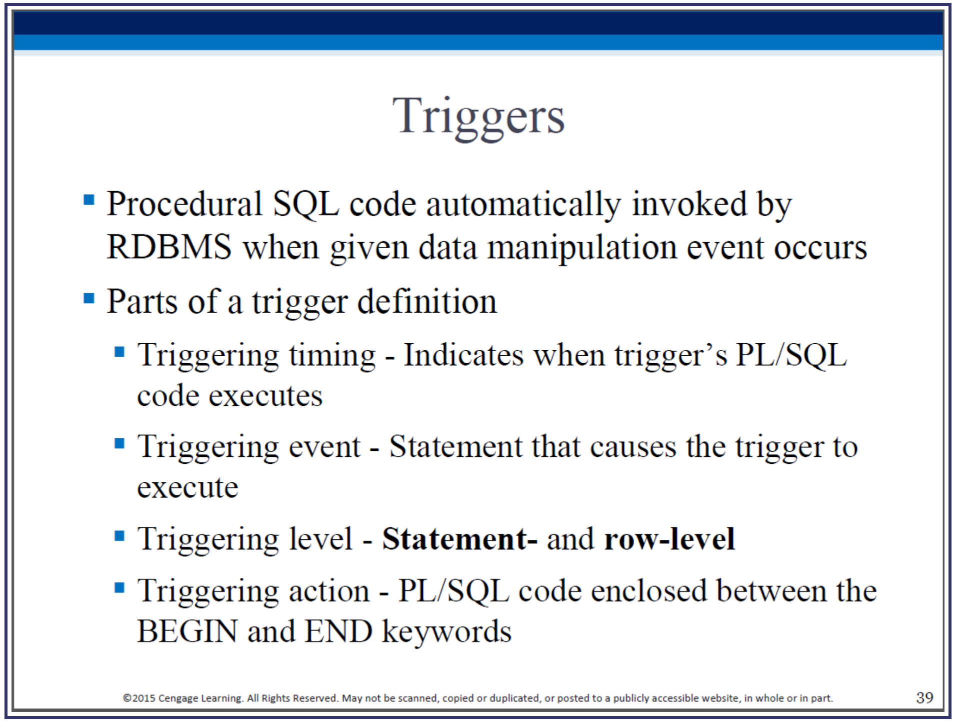

Procedural SQL
Procedural Language SQL (PL/SQL)
Use and storage of procedural code and SQL statements within the database
Merging of SQL and traditional programming constructs
Procedural code is executed as a unit by DBMS when invoked by end user
End users can use PL/SQL to create:
Anonymous PL/SQL blocks and triggers
Stored procedures and PL/SQL functions
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
37

[Unnamed] block creation example

```
SQL> BEGIN
  2  INSERT INTO VENDOR
  3  VALUES (25678, 'Microsoft Corp.', 'Bill Gates', '765', '546-8484', 'WA', 'N');
  4  END;
  5  /

PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON
SQL>
SQL> BEGIN
  2  INSERT INTO VENDOR
  3  VALUES (25772, 'Clue Store', 'Issac Hayes', '456', '323-2009', 'VA', 'N');
  4  DBMS_OUTPUT.PUT_LINE('New Vendor Added!');
  5  END;
  6  /
New Vendor Added!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM VENDOR;

    V_CODE V_NAME                              V_CONTACT      V_A V_PHONE  V_ V
---------- ----------------------------------- -------------- --- -------- -- -
     25678 Microsoft Corp.                     Bill Gates     765 546-8484 WA N
     25772 Clue Store                          Issac Hayes    456 323-2009 VA N
     21225 Bryson, Inc.                        Smithson       615 223-3234 TN Y
     21226 SuperLoo, Inc.                      Flushing       904 215-8995 FL N
     21231 D&E Supply                          Singh          615 228-3245 TN Y
     21344 Gomez Bros.                         Ortega         615 889-2546 KY N
     22567 Dome Supply                         Smith          901 678-1419 GA N
     23119 Randsets Ltd.                       Anderson       901 678-3998 GA Y
     24004 Brackman Bros.                      Browning       615 228-1410 TN N
     24288 ORDVA, Inc.                         Hakford        615 898-1234 TN Y
     25443 B&K, Inc.                           Smith          904 227-0093 FL N
     25501 Damal Supplies                      Smythe         615 890-3529 TN N
     25595 Rubicon Systems                     Orton          904 456-0092 FL Y

13 rows selected.

SQL>
```

More SQL (1) 2023-03-08, 8:18 PM

Triggers



Triggers [cont'd]

- *The triggering timing:* BEFORE or AFTER. This timing indicates when the trigger's PL/SQL code executes—in this case, before or after the triggering statement is completed.
- *The triggering event:* The statement that causes the trigger to execute (INSERT, UPDATE, or DELETE).
- *The triggering level:* The two types of triggers are statement-level triggers and row-level triggers.
  - A **statement-level trigger** is assumed if you omit the FOR EACH ROW keywords. This type of trigger is executed once, before or after the triggering statement is completed. This is the default case.
  - A **row-level trigger** requires use of the FOR EACH ROW keywords. This type of trigger is executed once for each row affected by the triggering statement. (In other words, if you update 10 rows, the trigger executes 10 times.)
- *The triggering action:* The PL/SQL code enclosed between the BEGIN and END keywords. Each statement inside the PL/SQL code must end with a semicolon (;).

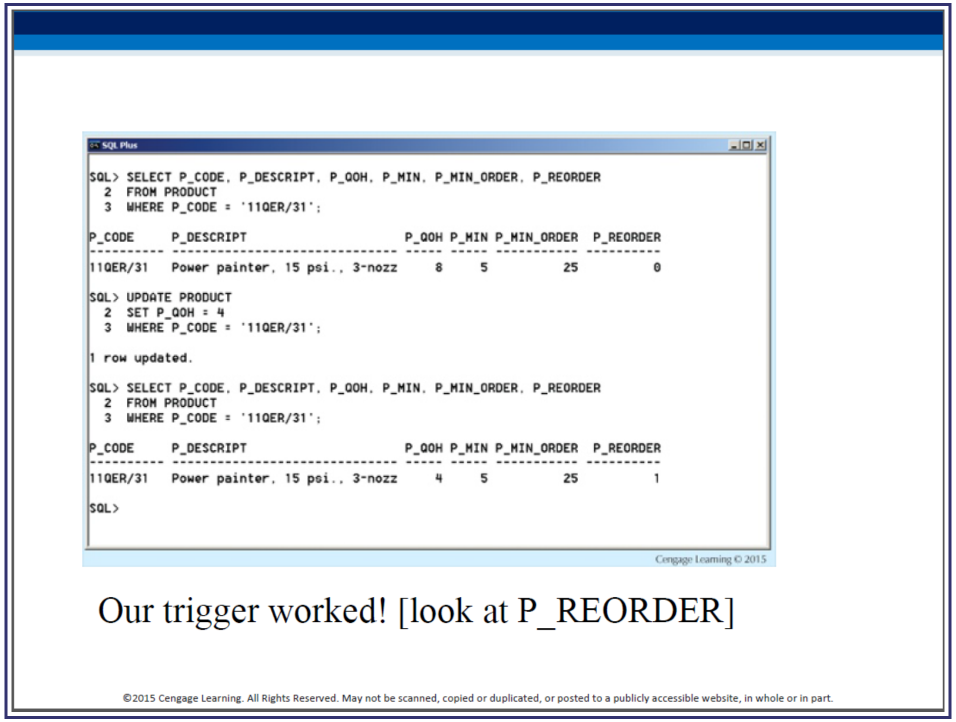
Trigger example

CREATE OR REPLACE TRIGGER *trigger_name*
[BEFORE / AFTER] [DELETE / INSERT / UPDATE OF *column_name*] ON *table_name*
[FOR EACH ROW]
[DECLARE]
[*variable_namedata type*[:=*initial_value*]]
BEGIN
PL/SQL instructions;
....
END;

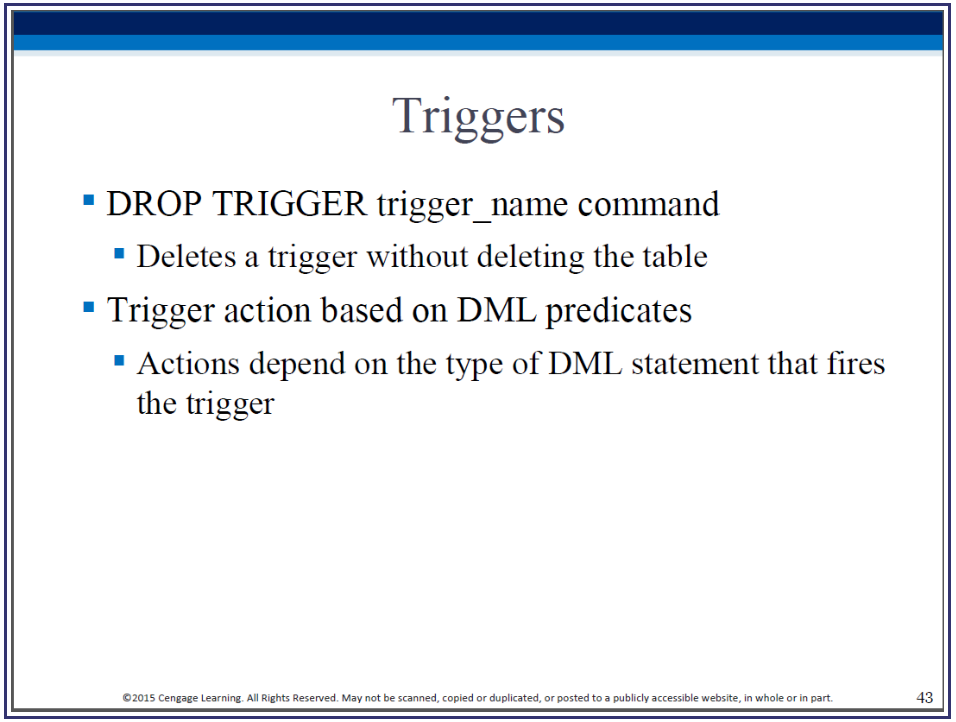
SQL Plus

```
SQL> CREATE OR REPLACE TRIGGER TRG_PRODUCT_REORDER
  2  AFTER INSERT OR UPDATE OF P_QOH ON PRODUCT
  3  BEGIN
  4     UPDATE PRODUCT
  5       SET P_REORDER = 1
  6         WHERE P_QOH <= P_MIN;
  7  END;
  8  /

Trigger created.

SQL>
```

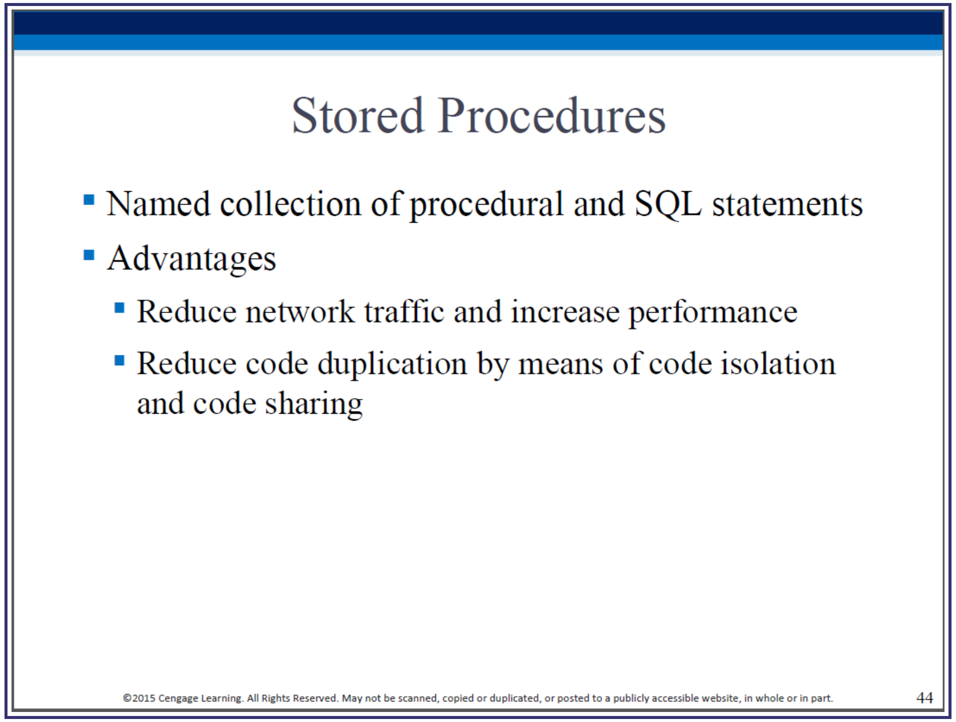
Trigger example



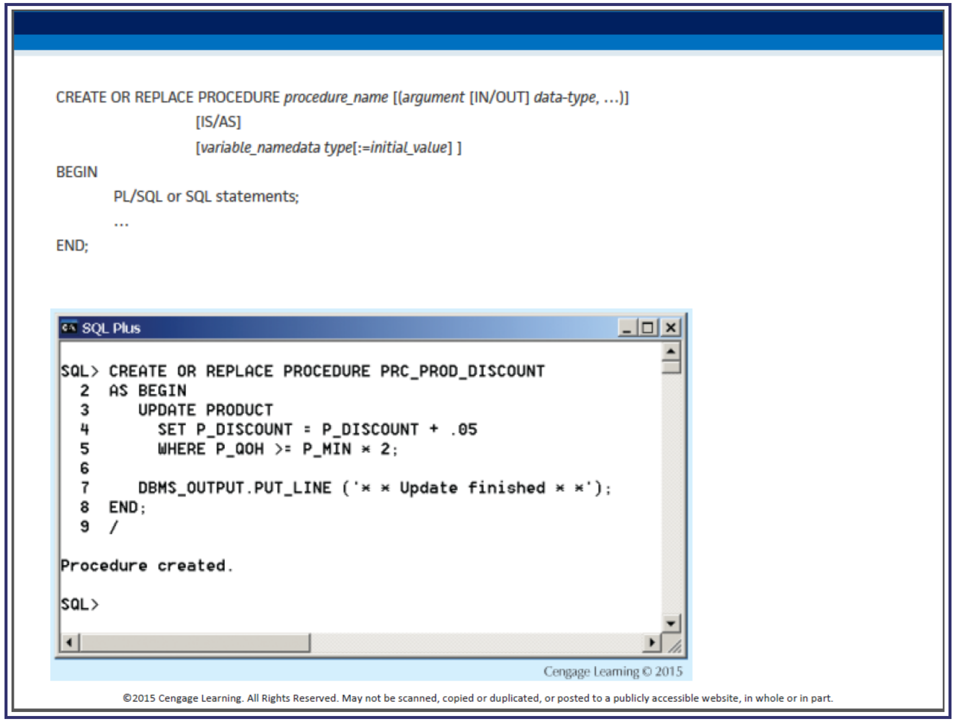
Triggers [cont'd]



Stored procedures



Stored procedure example

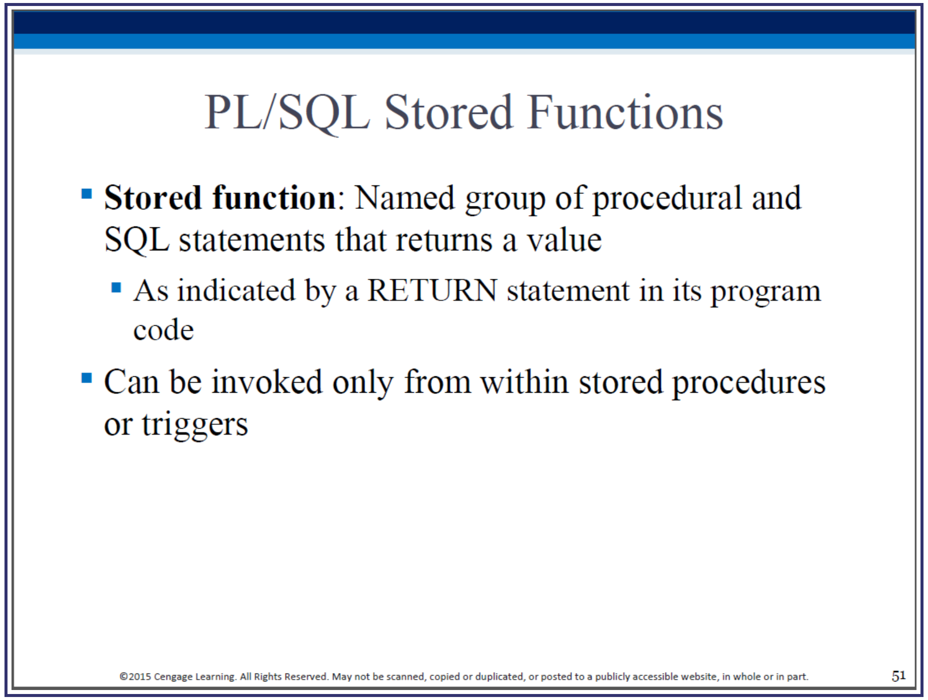


Stored procedure example

```
SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;

P_CODE     P_DESCRIPT                          P_QOH P_MIN P_DISCOUNT
---------- ----------------------------------- ----- ----- ----------
11QER/31   Power painter, 15 psi., 3-nozz         29     5       0.00
13-Q2/P2   7.25-in. pwr. saw blade                32    15       0.05
14-Q1/L3   9.00-in. pwr. saw blade                18    12       0.00
1546-QQ2   Hrd. cloth, 1/4-in., 2x50              15     8       0.00
1558-QW1   Hrd. cloth, 1/2-in., 3x50              23     5       0.00
2232/QTY   B&D jigsaw, 12-in. blade                8     5       0.05
2232/QWE   B&D jigsaw, 8-in. blade                 6     7       0.05
2238/QPD   B&D cordless drill, 1/2-in.            12     5       0.05
23109-HB   Claw hammer                            23    10       0.10
23114-AA   Sledge hammer, 12 lb.                   8    10       0.05
54778-2T   Rat-tail file, 1/8-in. fine            43    20       0.00
89-WRE-Q   Hicut chain saw, 16 in.                11     5       0.05
PVC23DRT   PVC pipe, 3.5-in., 8-ft               188    75       0.00
SM-18277   1.25-in. metal screw, 25              172    75       0.00
SW-23116   2.5-in. wd. screw, 50                 237   100       0.00
WR3/TT3    Steel matting, 4'x8'x1/6", .5"         18     5       0.10

16 rows selected.

SQL> EXEC PRC_PROD_DISCOUNT;
* * Update finished * *

PL/SQL procedure successfully completed.

SQL> SELECT P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_DISCOUNT FROM PRODUCT;

P_CODE     P_DESCRIPT                          P_QOH P_MIN P_DISCOUNT
---------- ----------------------------------- ----- ----- ----------
11QER/31   Power painter, 15 psi., 3-nozz         29     5       0.05
13-Q2/P2   7.25-in. pwr. saw blade                32    15       0.10
14-Q1/L3   9.00-in. pwr. saw blade                18    12       0.00
1546-QQ2   Hrd. cloth, 1/4-in., 2x50              15     8       0.00
1558-QW1   Hrd. cloth, 1/2-in., 3x50              23     5       0.05
2232/QTY   B&D jigsaw, 12-in. blade                8     5       0.05
2232/QWE   B&D jigsaw, 8-in. blade                 6     7       0.05
2238/QPD   B&D cordless drill, 1/2-in.            12     5       0.10
23109-HB   Claw hammer                            23    10       0.15
23114-AA   Sledge hammer, 12 lb.                   8    10       0.05
54778-2T   Rat-tail file, 1/8-in. fine            43    20       0.05
89-WRE-Q   Hicut chain saw, 16 in.                11     5       0.10
PVC23DRT   PVC pipe, 3.5-in., 8-ft               188    75       0.05
SM-18277   1.25-in. metal screw, 25              172    75       0.05
SW-23116   2.5-in. wd. screw, 50                 237   100       0.05
WR3/TT3    Steel matting, 4'x8'x1/6", .5"         18     5       0.15

16 rows selected.

SQL>
```

Stored functions

Reminder - these can RETURN a value.



Stored functions - syntax

```
CREATE FUNCTION function_name (argument IN data-type, …) RETURN data-type [IS]
BEGIN
        PL/SQL statements;
        …
        RETURN (value or expression);
END;
```

Once such a function is defined, it can be CALLED inside triggers or in stored procedures..

Stored functions - example

The following is an example from http://www.tutorialspoint.com/plsql.

Creating/defining a function:

```
FUNCTION findMax(x IN number, y IN numb
er)
RETURN number
IS
    z number;
BEGIN
   IF x > y THEN
      z:= x;
   ELSE
      Z:= y;
   END IF;

   RETURN z;
END;
```

Calling/executing/running the function:

```
DECLARE
   a number;
   b number;
   c number;
BEGIN
   a:= 23;
   b:= 45;

   c := findMax(a, b);
   dbms_output.put_line(' Maximum of (2
3,45): ' || c);
END;
/
```

Result:

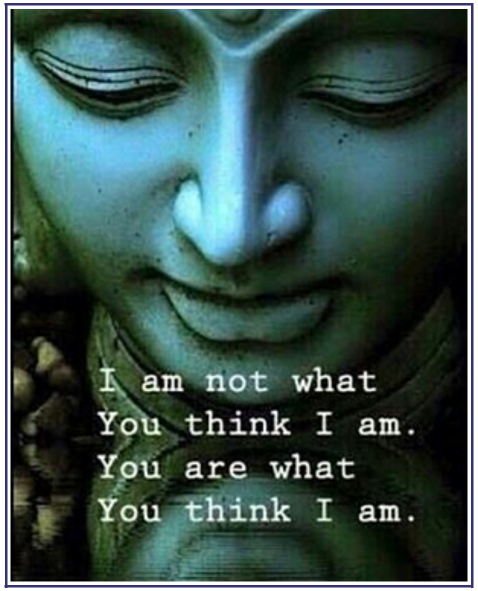
```
Maximum of (23,45): 45
```

1/30 1:11:40 * * *

← →

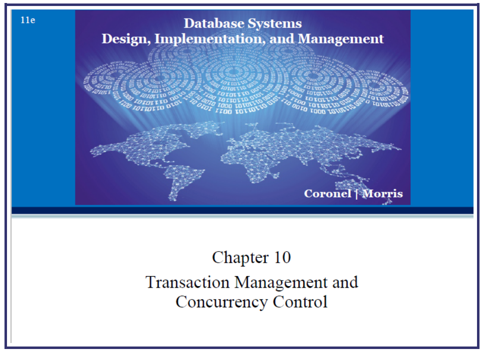
TM

[Transaction Management]

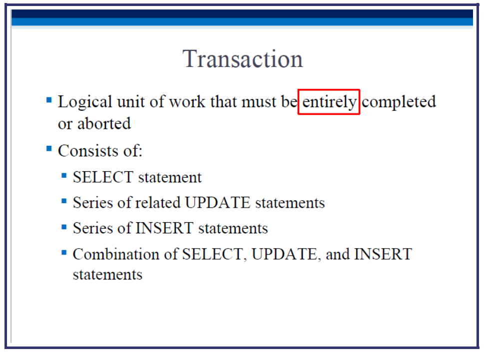


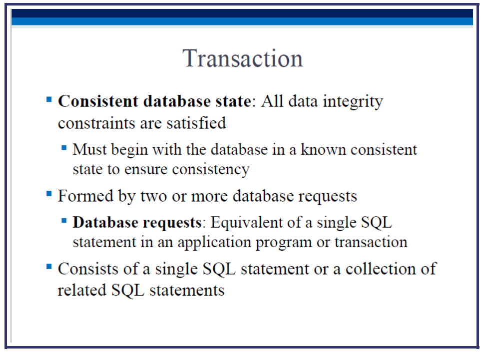
A different kind of TM :)

Ch.10



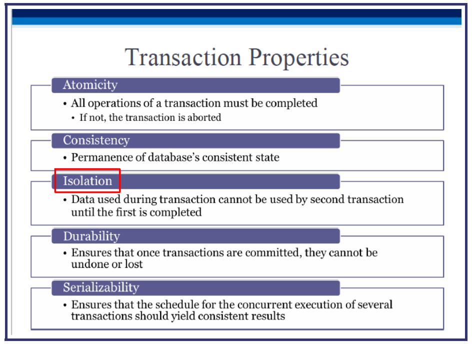
What is a 'transaction'?

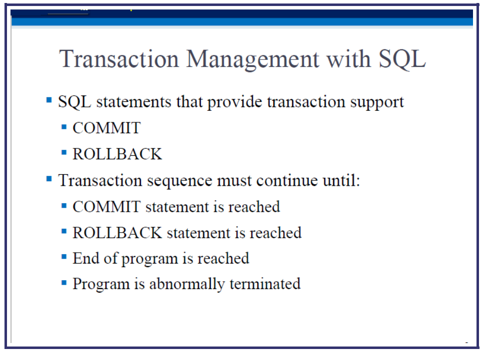




'ACID'

'ACID' (+S) is an acronym to express the desirable properties of a transaction:





Tracking updates

Transaction Log

- Keys track of all transactions that update the database
- DBMS uses the information stored in a log for:
  - Recovery requirement triggered by a ROLLBACK statement
  - A program's abnormal termination
  - A system failure

A Transaction Log

TRL_ ID	TRX_ NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	1558-QW1	PROD_QOH	25	23
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_ BALANCE	525.75	615.73
365	101	363	Null	COMMIT	**** End of Transaction				

TRL_ID = Transaction log record ID
TRX_NUM = Transaction number
PTR = Pointer to a transaction log record ID
(*Note*: The transaction number is automatically assigned by the DBMS.)

Concurrency - 'many at once'

Concurrency Control

- Coordination of the simultaneous transactions execution in a multiuser database system
- Objective - Ensures serializability of transactions in a multiuser database environment

Problems in Concurrency Control

Lost update

- Occurs in two concurrent transactions when:
  - Same data element is $\boxed{\text{updated}}$
  - One of the updates is lost

Uncommitted data

- Occurs when:
  - Two transactions are executed concurrently
  - First transaction is rolled back after the second transaction has already $\boxed{\text{accessed}}$ uncommitted data

Inconsistent retrievals

- Occurs when a transaction $\boxed{\text{accesses}}$ data before and after one or more other transactions finish working with such data

Consider the following pair of transactions, starting with 35 units of an item: purchase 100 units, then sell 30 units:

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 – 30	
6	T2	Write PROD_QOH	105

If the transactions T1 and T2 happen one after another (not concurrently), the PROD_QOH value would/should be 105, as shown above.

Three different kinds of errors are possible, when interleaved transactions are not handled properly.

Error #1: lost updates

'Lost update' problem:

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 – 30	
5	T1	Write PROD_QOH (lost update)	135
6	T2	$\boxed{\text{Write PROD\_QOH}}$	5

Here, instead of 105, our QOH comes out to be 5, which is incorrect.

Consider this rollback situation:

TRANSACTION	COMPUTATION
T1: Purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled back)
T2: Sell 30 units	PROD_QOH = PROD_QOH – 30

Proper rollback (with sequential transactions):

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 – 30	
7	T2	Write PROD_QOH	5

Here the total is 5, which is correct - the QOH goes from 35 to 135, gets rolled back to 35, after which the purchase (of 30 units) happens.

Error #2: reading uncommitted data

'Uncommitted data' problem (improper rollback):

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read uncommitted data)	135
5	T2	PROD_QOH = 135 – 30	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

Here, the total comes out to 105, when it should have been 5.

Consider the following fix (of a typo made earlier - an incorrect order of 10 units was placed for 1558-QW1 by mistake, instead of ordering 1546-QQ2; now we're fixing that error):

TRANSACTION T1	TRANSACTION T2
SELECT SUM(PROD_QOH) FROM PRODUCT	UPDATE PRODUCT SET PROD_QOH = PROD_QOH + 10 WHERE PROD_CODE = 1546-QQ2
	UPDATE PRODUCT SET PROD_QOH = PROD_QOH – 10 WHERE PROD_CODE = 1558-QW1
	COMMIT;

In the above, T1 is a transaction that sums up QOH; T2 is the 'correction' transaction (that fixes the incorrect purchasing).

Proper retrieval of modified data:

	BEFORE	AFTER
PROD_CODE	PROD_QOH	PROD_QOH
11QER/31	8	8
13-Q2/P2	32	32
1546-QQ2	15	(15 + 10) → 25
1558-QW1	23	(23 – 10) → 13
2232-QTY	8	8
2232-QWE	6	6
Total	**92**	**92**

The summing transaction gets proper totals for both affected products, so the total (of 92) is correct.

Error #3: improper [premature] retrieval

'Inconsistent retrievals' problem (improper ("before update") retrieval of modified data):

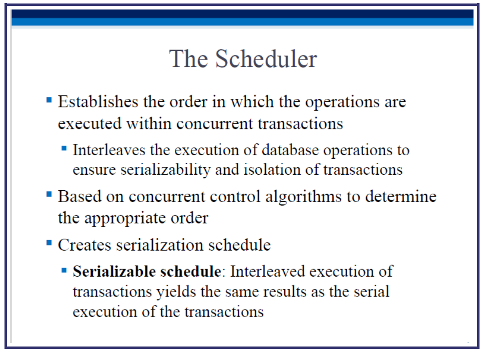
TIME	TRANSACTION	ACTION	VALUE	TOTAL
1	T1	Read PROD_QOH for PROD_CODE = '11QER/31'	8	8
2	T1	Read PROD_QOH for PROD_CODE = '13-Q2/P2'	32	40
3	T2	Read PROD_QOH for PROD_CODE = '1546-QQ2'	15	
4	T2	PROD_QOH = 15 + 10		
5	T2	Write PROD_QOH for PROD_CODE = '1546-QQ2'	25	
6	T1	Read PROD_QOH for PROD_CODE = '1546-QQ2'	25	(After) 65
7	T1	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	(Before) 88
8	T2	Read PROD_QOH for PROD_CODE = '1558-QW1'	23	
9	T2	PROD_QOH = 23 – 10		
10	T2	Write PROD_QOH for PROD_CODE = '1558-QW1'	13	
11	T2	***** COMMIT *****		
12	T1	Read PROD_QOH for PROD_CODE = '2232-QTY'	8	96
13	T1	Read PROD_QOH for PROD_CODE = '2232-QWE'	6	102

T1 should be doing 65+13 (which would be correct), but instead does 65+23 (which makes it incorrect) - in other words, T1 retrieves the correct value (25) for 1546-QQ2, but gets the incorrect value (23) for 1558-QW1.

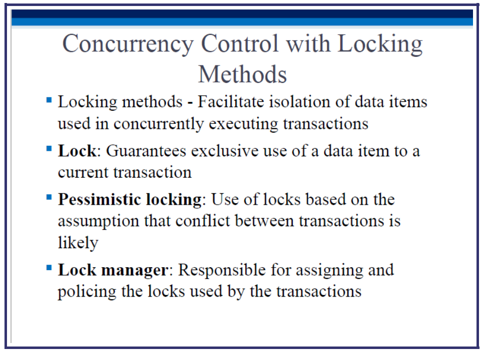
Inconsistent retrieval is also called a 'dirty read'.

Look up: non-repeatable reads, phantom row reads.

Concurrent, serializable schedule



A serializable schedule makes concurrency immaterial (non-issue).



TRL pessimistic 'lock' situations: restrooms, RCS, Robert's Rules Of Order..

Locking: granularity

Lock Granularity

- Indicates the level of lock use
- Levels of locking
  - **Database-level lock**
  - **Table-level lock**
  - **Page-level lock**
    - **Page** or **diskpage**: Directly addressable section of a disk
  - **Row-level lock**
  - **Field-level lock**

Figure 10.3 - Database-Level Locking Sequence

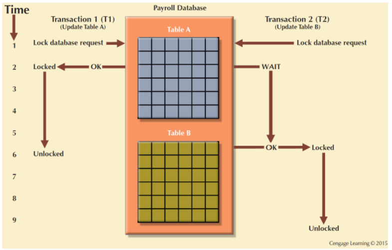


Figure 10.4 - An Example of a Table-Level Lock

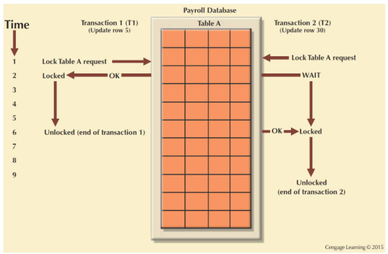


Figure 10.5 - An Example of a Page-Level Lock

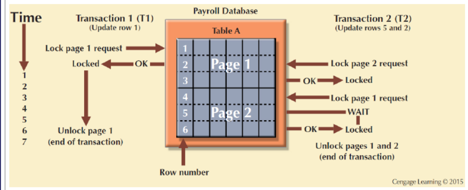
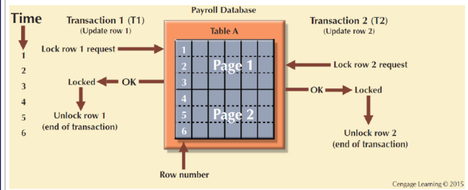
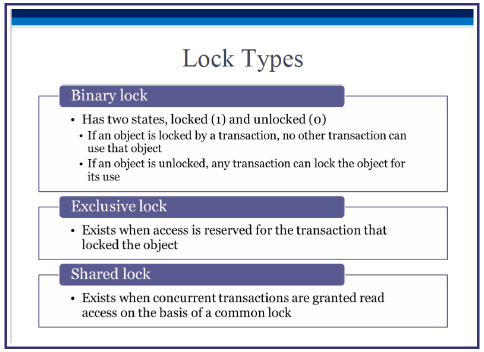
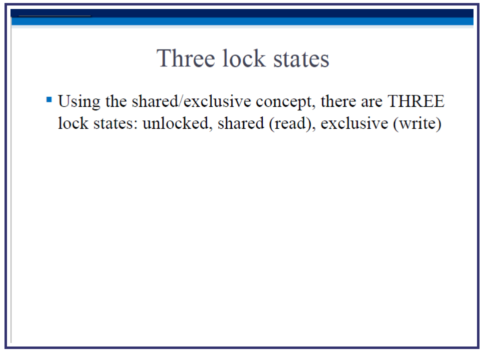


Figure 10.6 - An Example of a Row-Level Lock

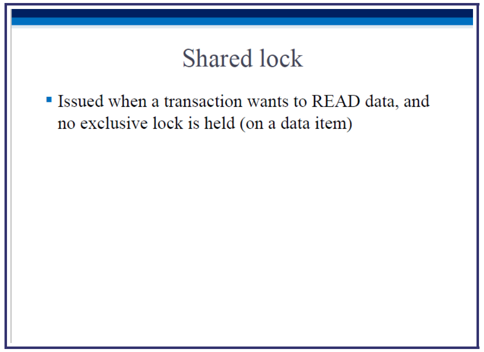


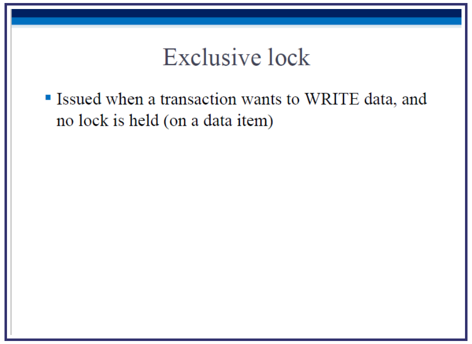
Locking: types [unlocked, locked_read, locked_write]



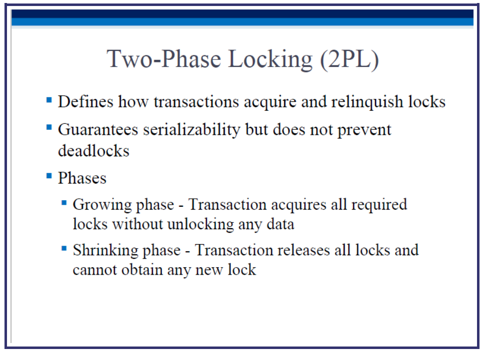


Read(shared), write(exclusive) lock



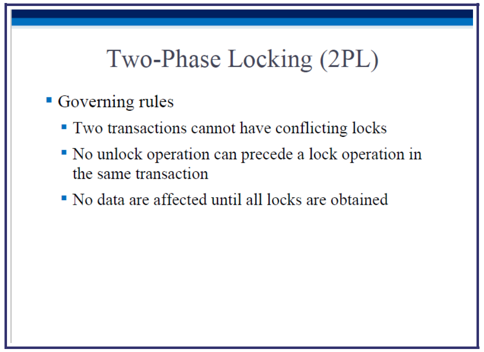


'2PL'



The 2PL type that we are discussing, where a transaction acquires all the locks it needs before processing starts, is called 'Conservative' 2PL (or 'Static' 2PL).

2PL [cont'd]



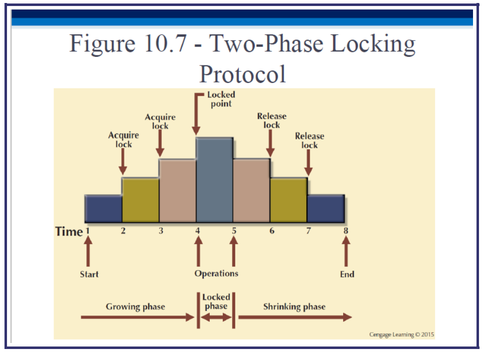
Once all the locks are acquired, a transaction can proceed 'smoothly', ie won't 'hang' or lead to dirty reads etc.

As for the middle point above (unlocking can't precede locking), here is a loose analogy:

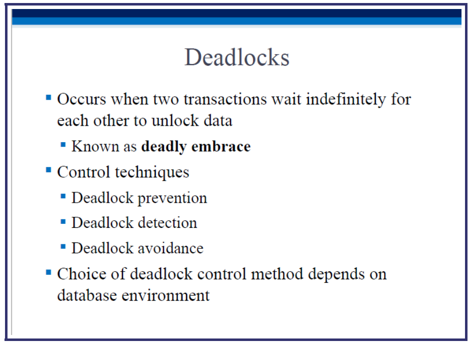
```
OK:
{
   {
      {
         {
            {
               ....
            }
         }
      }
   }
}

Not OK (OK in programming, though!):
{
   {
   }
   {
    {
         {
         }
    }
   }
}
```

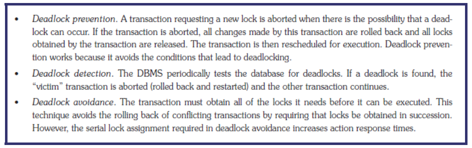

TM (1) 2023-03-10, 1:11 PM

2PL [cont'd]



Deadlocks..





- *Deadlock prevention*. A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. If the transaction is aborted, all changes made by this transaction are rolled back and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.
- *Deadlock detection*. The DBMS periodically tests the database for deadlocks. If a deadlock is found, the "victim" transaction is aborted (rolled back and restarted) and the other transaction continues.
- *Deadlock avoidance*. The transaction must obtain all of the locks it needs before it can be executed. This technique avoids the rolling back of conflicting transactions by requiring that locks be obtained in succession. However, the serial lock assignment required in deadlock avoidance increases action response times.

Timestamping-based schemes prevent deadlocks; 2PL avoids deadlocks; detection is used in both.

How/why a deadlock occurs

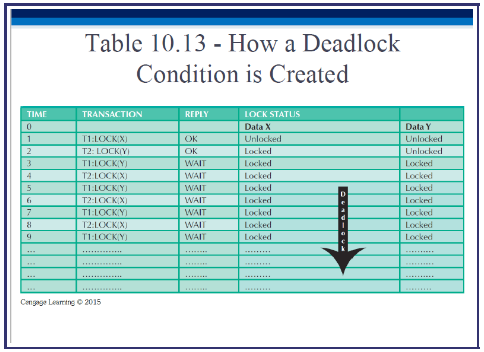


Table 10.13 - How a Deadlock Condition is Created

TIME	TRANSACTION	REPLY	LOCK STATUS	
0			**Data X**	**Data Y**
1	T1:LOCK(X)	OK	Unlocked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Unlocked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....
...	....	....	....	....

Cengage Learning © 2015

In the above, we see that T1 locks X, T2 locks Y, then deadlock occurs because T1 wants Y and T2 wants X. Maybe you are thinking - why can't each of them release what they are holding (since they might be done with it), and grab what they want next (ie. T1 would release X, T2 would release Y)? BECAUSE THEY CAN'T - they NEED to access what the other has, BEFORE they can release what they have. Eg. T1 might need to read Y that is locked by T2, in order to update its X; T2 might need T1's X to use in an expression to compare with its Y. **They need access to each other's resources, **before** they can release their own! That is what causes deadlocking.**

Note that more than two transactions can become deadlocked as well, on account of 'circular' (cyclical) waiting.

Here's a humorous take on deadlocking that can occur in human interactions.

Deadlock occurrence in 2PL

Earlier we noted that the 2PL scheme cannot prevent deadlock creation. Here is an example of how a deadlock could occur (at the end of step 5):

An important and unfortunate property of 2PL schedulers is that they are subject to *deadlocks*. For example, suppose a 2PL scheduler is processing transactions T_1 and T_3

$$T_1: r_1[x] \rightarrow w_1[y] \rightarrow c_1 \qquad T_3: w_3[y] \rightarrow w_3[x] \rightarrow c_3$$

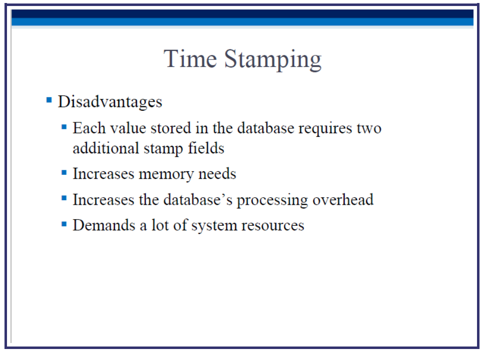
and consider the following sequence of events:

1. Initially, neither transaction holds any locks.
2. The scheduler receives $r_1[x]$ from the TM. It sets $rl_1[x]$ and submits $r_1[x]$ to the DM.
3. The scheduler receives $w_3[y]$ from the TM. It sets $wl_3[y]$ and submits $w_3[y]$ to the DM.
4. The scheduler receives $w_3[x]$ from the TM. The scheduler does not set $wl_3[x]$ because it conflicts with $rl_1[x]$ which is already set. Thus $w_3[x]$ is delayed.
5. The scheduler receives $w_1[y]$ from the TM. As in (4), $w_1[y]$ must be delayed.

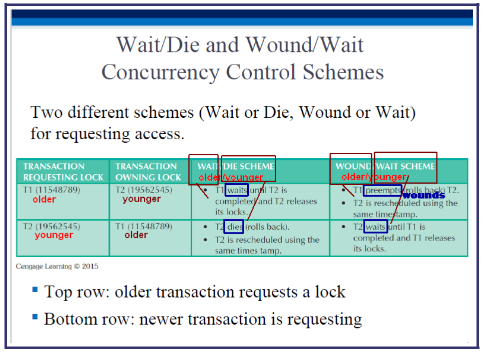
Time Stamping

- Assigns global, unique time stamp to each transaction
  - Produces explicit order in which transactions are submitted to DBMS
- Properties
  - **Uniqueness**: Ensures no equal time stamp values exist
  - **Monotonicity**: Ensures time stamp values always increases

Here is one way to get monotonically increasing GUIDs..



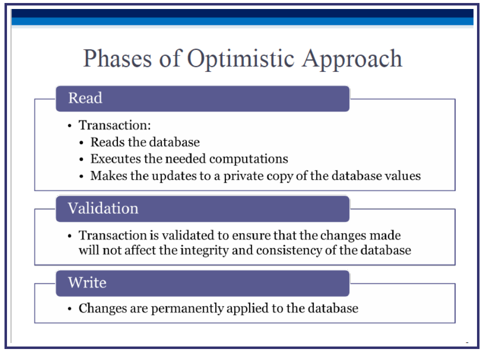
Deadlock prevention



Note that wound-wait is a preemptive deadlock prevention scheme, whereas wait-die is a non-preemptive one..

PS: Deadlock detection is periodically carried out by detecting cycles in waiting transactions.

Deadlock 'indifference' ['fix-it-later']



1/30 1:06:12 ***

← →

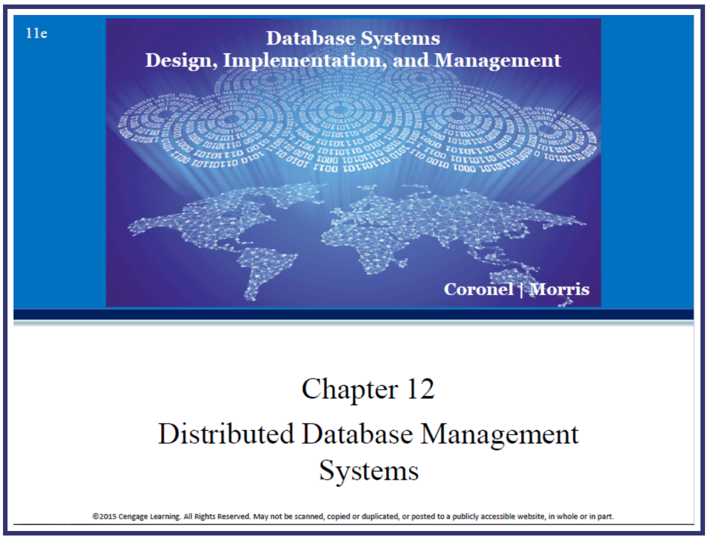
Distributed DBs

what is distributed?

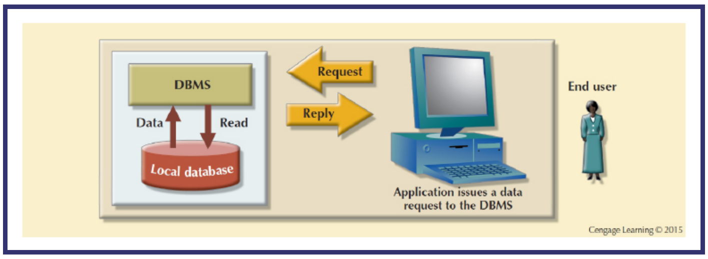
how are transactions managed?

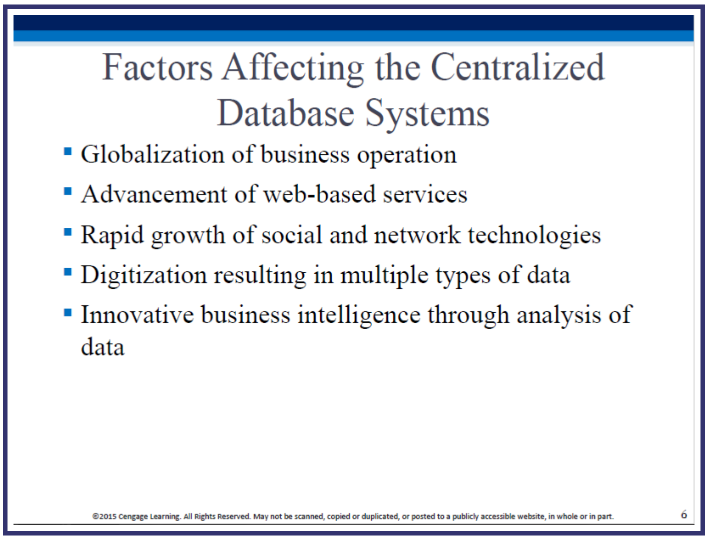
what is the architecture?

Ch.12


11e
Database Systems
Design, Implementation, and Management
Coronel | Morris
Chapter 12
Distributed Database Management Systems
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.

'Centralized' DBs - no longer popular/useful

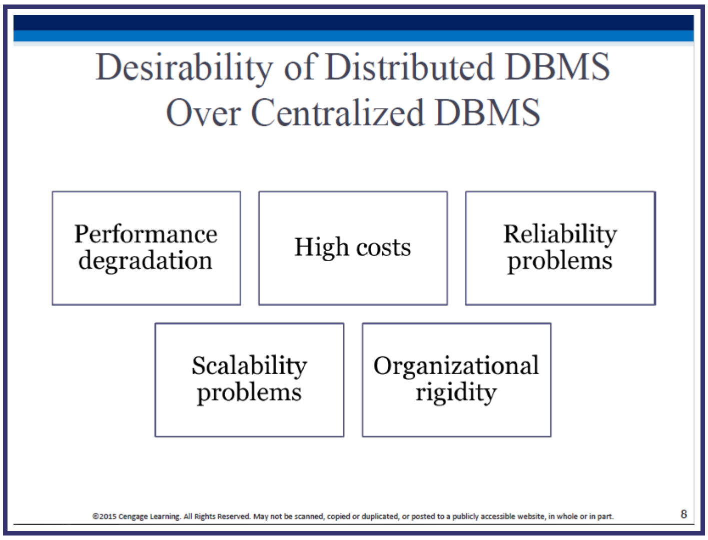




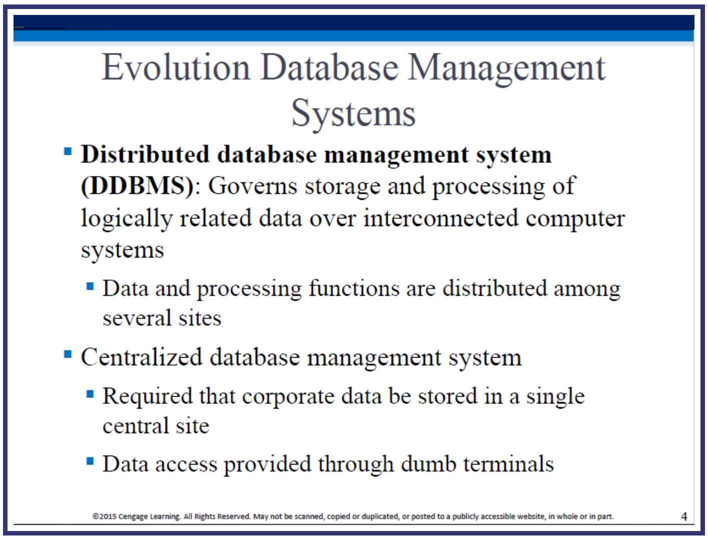
Two factors in particular, necessitated change:

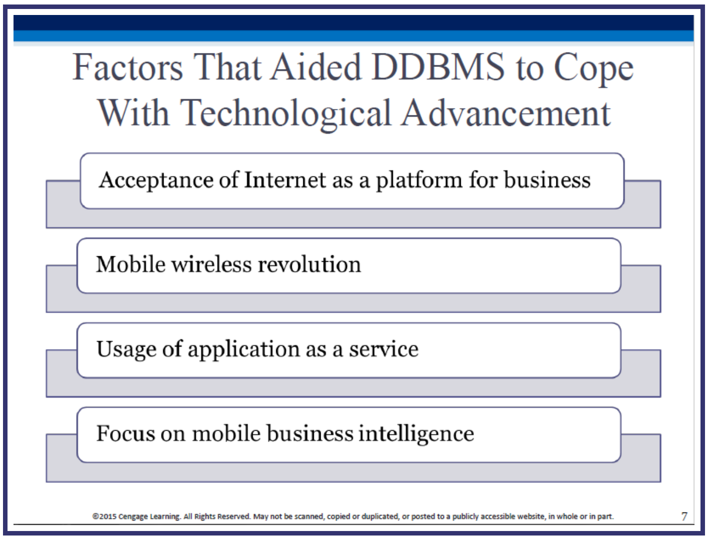
- rapid, ad-hoc access to data was needed ['Internet speed' decision-making]
- distributed data access was needed, to serve dispersed business units [globalization]

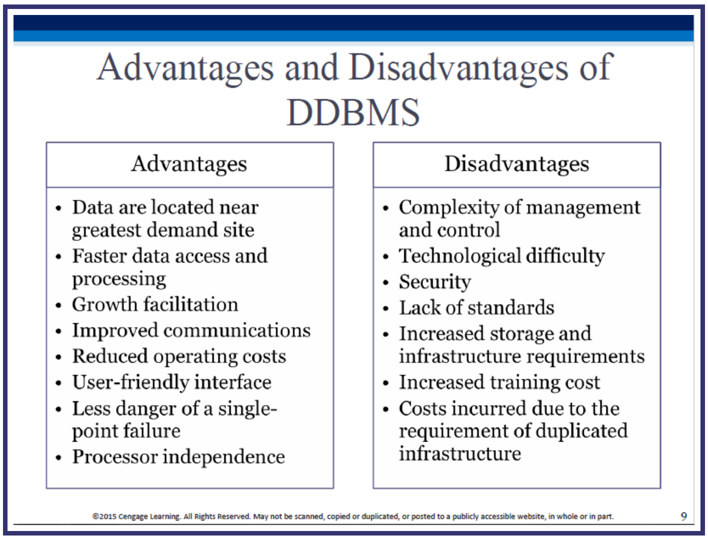
So now we have **D**DBMSs - **distributed** DBMSs.


Desirability of Distributed DBMS Over Centralized DBMS
Performance degradation
High costs
Reliability problems
Scalability problems
Organizational rigidity
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
8

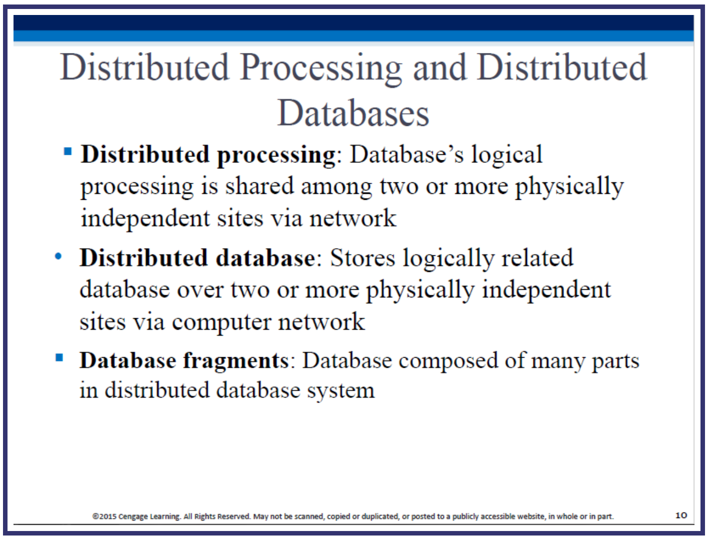
Distributed DBs - almost ALL (web-based)!



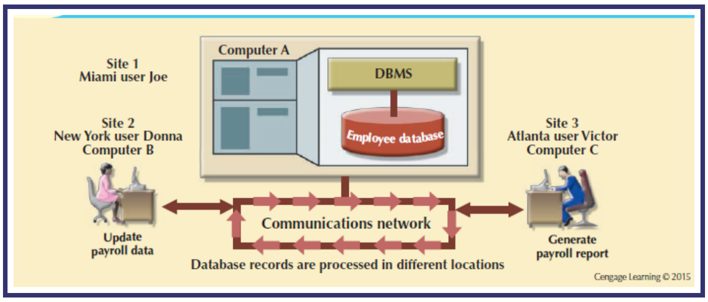



Advantages and Disadvantages of DDBMS
Advantages
Data are located near greatest demand site
Faster data access and processing
Growth facilitation
Improved communications
Reduced operating costs
User-friendly interface
Less danger of a single-point failure
Processor independence
Disadvantages
Complexity of management and control
Technological difficulty
Security
Lack of standards
Increased storage and infrastructure requirements
Increased training cost
Costs incurred due to the requirement of duplicated infrastructure
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
9

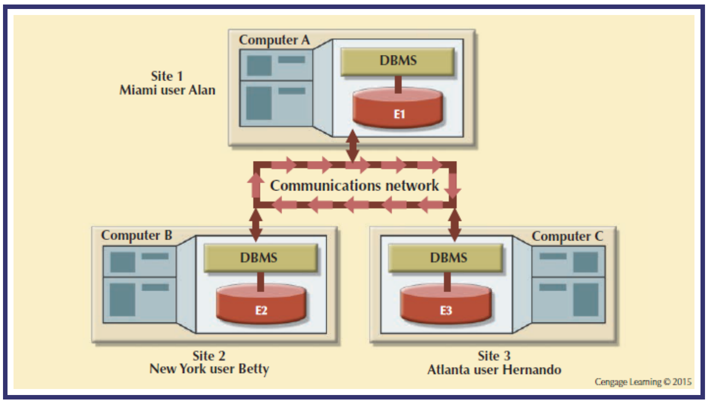
Distributed *processing* vs distributed *databases*



Distributed *processing*

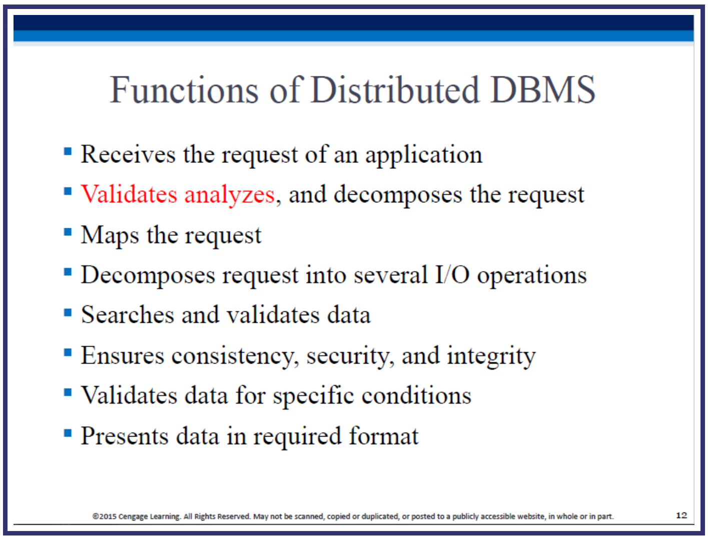


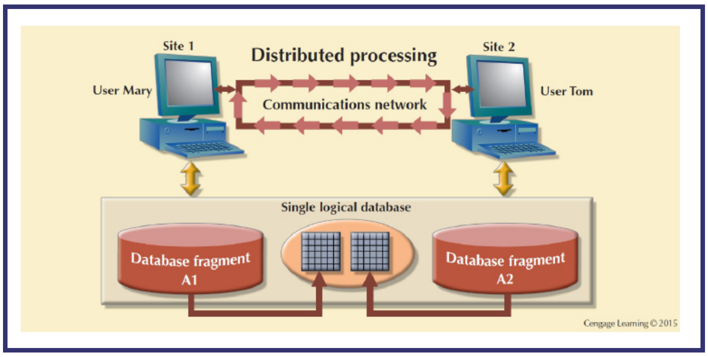
Distributed *databases*



Fully distributed DBMSs: functions

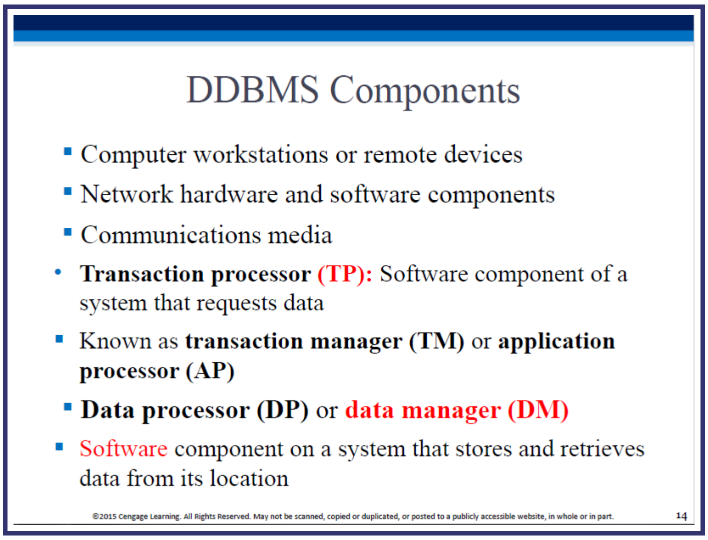
Distributed processing AND data storage -> 'fully' distributed.

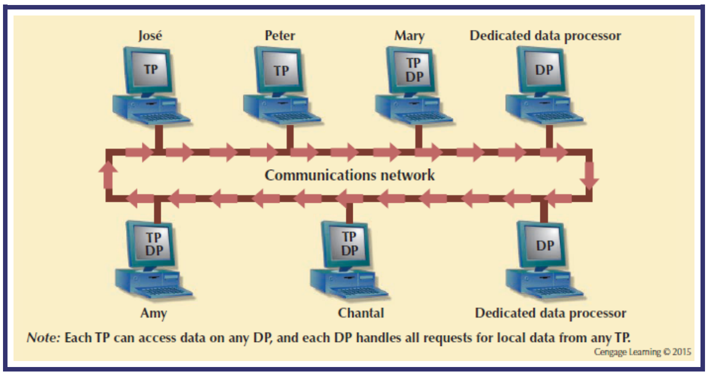




Two fragments, two sites - but each user thinks they have a single (their own) local version (transparent access).

TP(TM) vs DP(DM)





A set of protocols is used by the DDBMS, to enable the TPs and DPs to communicate with each other.

TP: Transaction Processor - receives data requests (from an application), and requests data (from a DP); also known as AP (Application Processor) or TM (Transaction Manager). A TP is what fulfills data requests on behalf of a transaction.

DP: Data Processor - receives data requests from a TP (in general, multiple TPs can request data from a single DP), retrieves and returns the requested data; also known as DM (Data Manager).

Data and processing distribution: 3 variations

12.6 Levels of Data and Process Distribution

Current database systems can be classified on the basis of how process distribution and data distribution are supported. For example, a DBMS may store data in a single site (using a centralized DB) or in multiple sites (using a distributed DB), and may support data processing at one or more sites. Table 12.2 uses a simple matrix to classify database systems according to data and process distribution. These types of processes are discussed in the sections that follow.

TABLE 12.2 Database Systems: Levels of Data and Process Distribution

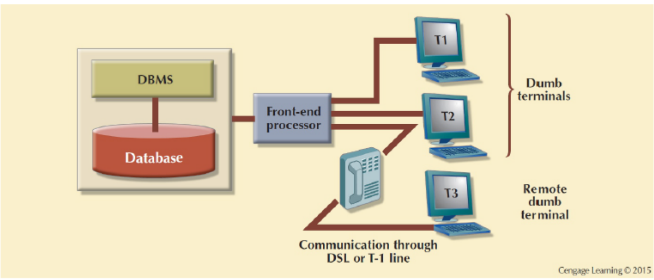
~~ADVANTAGES~~	SINGLE-SITE DATA	MULTIPLE-SITE DATA
Single-site process	Host DBMS	Not applicable (Requires multiple processes)
Multiple-site process	File server Client/server DBMS (LAN DBMS)	Fully distributed Client/server DDBMS

SP/SD

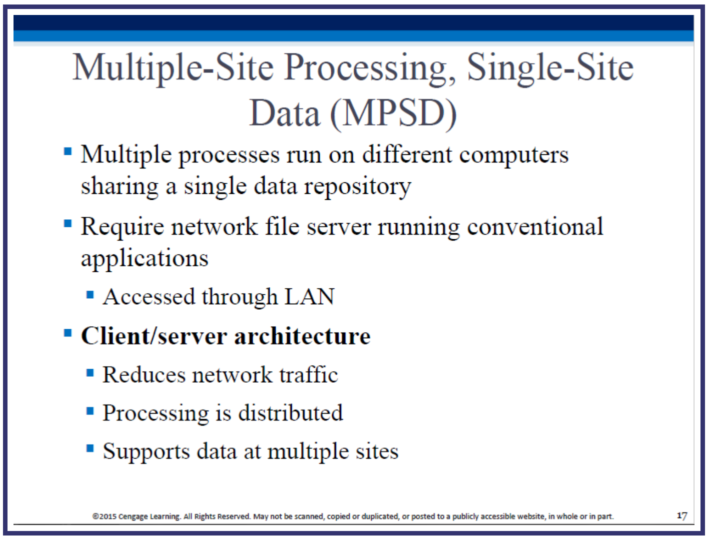
Single-Site Processing, Single-Site Data (SPSD)

- Processing is done on a single host computer
- Data stored on host computer's local disk
- Processing restricted on end user's side
- DBMS is accessed by dumb terminals

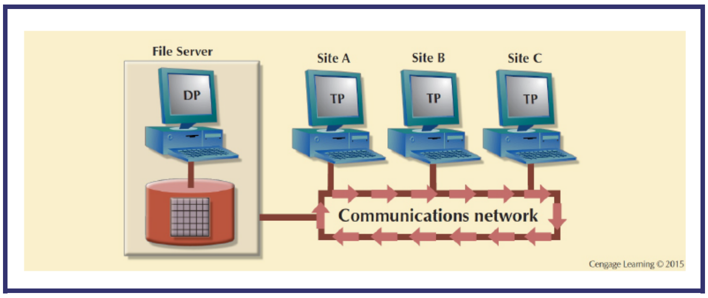
 15



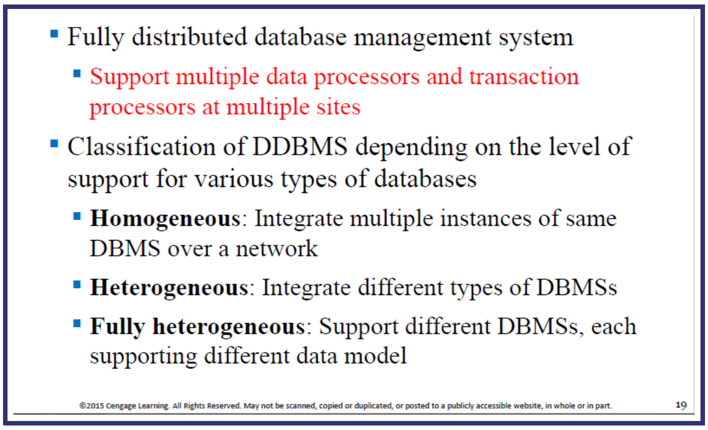
MP/SD [note: no SP/MD!]



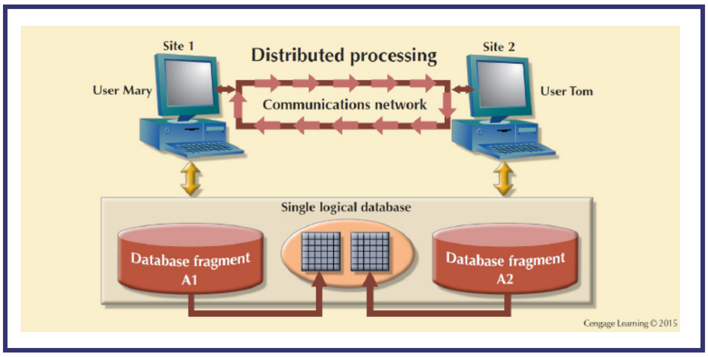
Not truly 'distributed' (data I/O is centralized).



MP/MD ['fully distributed']

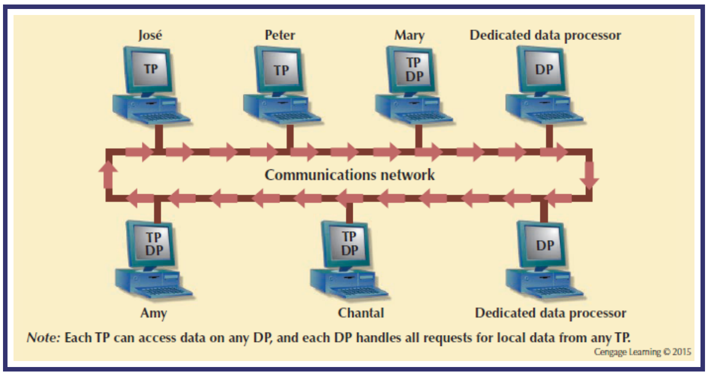


"The whole enchilada!".. Comes in three varieties.



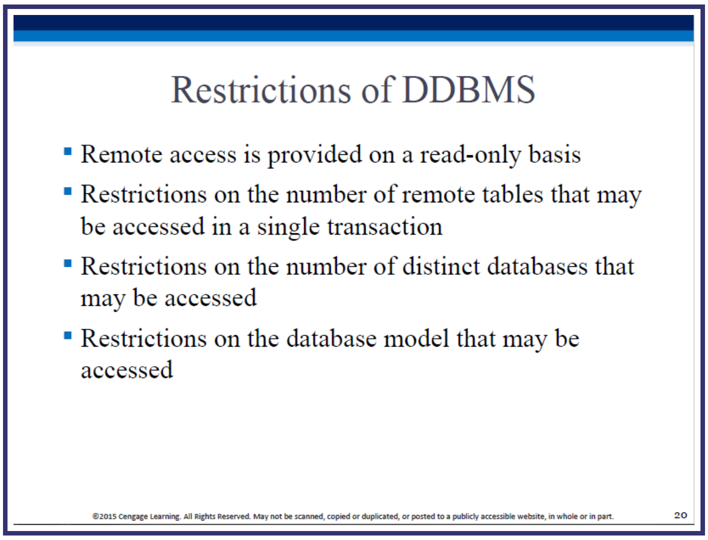
The above schematic is same as the one you saw earlier..

In a fully distributed database, the TPs (that request data) are distributed, as are the DPs (that serve data), like so [you saw this in an earlier slide]:



The DDBMS uses protocols to transport across its network, commands as well as data, from distributed DPs and TPs: it receives data requests from multiple TPs, collects/synchronizes data (needed by multiple TPs) from multiple DPs, and intelligently routes to multiple TPs, the data (collected from multiple DPs) they requested. In other words, since transactions need data, the DDBMS acts as a switch, shuttling and routing data, from multiple DPs, to multiple TPs.

DDBMS restrictions



DDBMs span the spectrum from homegenous to fully heterogenous, and tend to come with restrictions.

Local/distributed requests/transactions

Remote/localized requests and transactions:

Remote request

- Single SQL statement accesses data processed by a single remote database processor

Remote transaction

- Accesses data at single remote site composed of several requests

Distributed (not localized) requests and transactions:

Distributed request

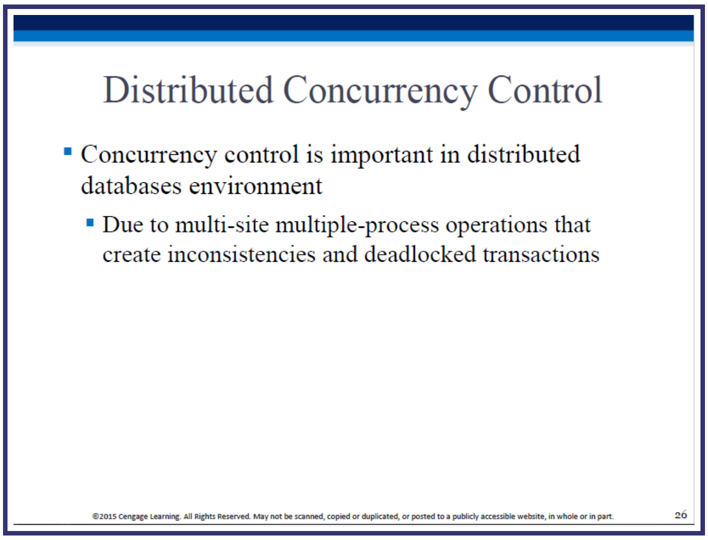
- Single SQL statement references data at several DP sites

Distributed transaction

- Requests data from several different remote sites on network

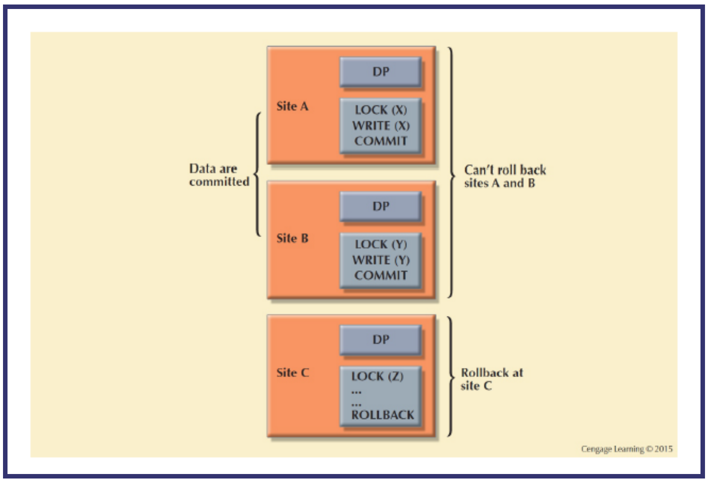
Distributed transactions are what need to be carefully executed so as to maintain distribution transparency - the transactions have to execute 'as if' they all ran on the same machine/location. This is achieved using '2PC' [explained in upcoming slides].

Distributed concurrency control



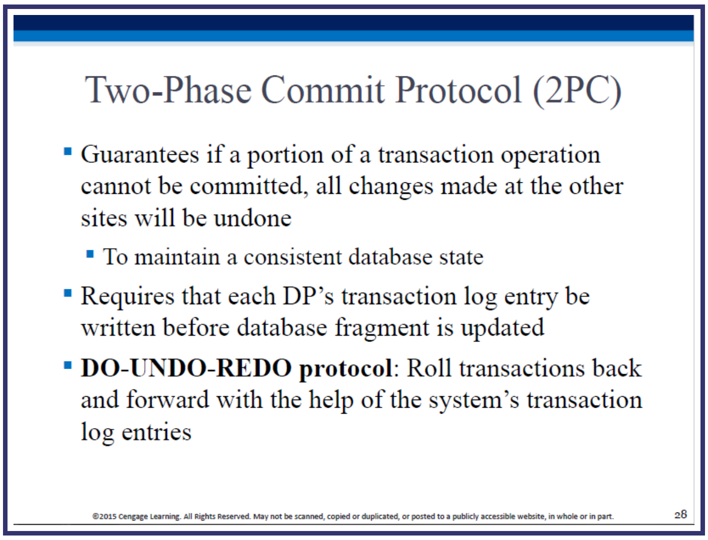
TP must ensure that all parts of a transaction (across sites) are completed, before doing a COMMIT.

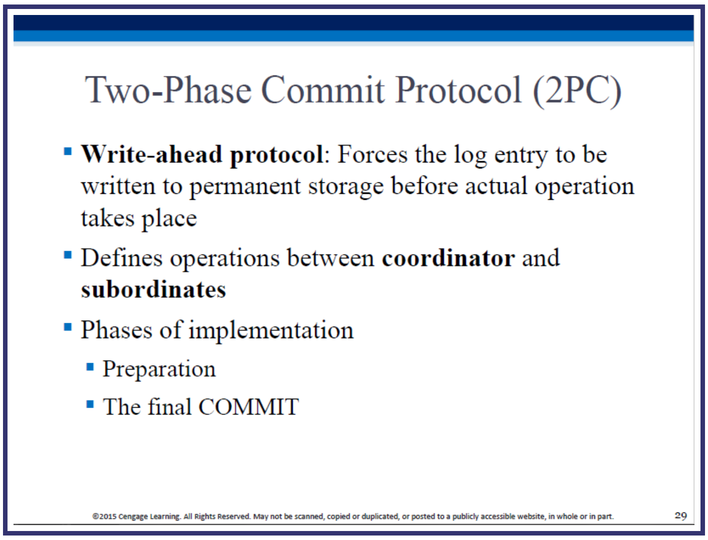
Problem!



Can't COMMIT the entire transaction, can't ROLLBACK it either!
Solution: 2PC.

Two-phase commit ("2PC") protocol





2PC: steps

The following is the sequence of steps (phase 1, phase2) that are used to effect a complete distributed transaction (keep clicking on the >| button):

And here's how a transaction is aborted, when one of the participating nodes is unable to commit a sub-transaction:

Click anywhere to show/hide controls:
* <<, >>: first/last frame
* <*1, >*1: play once from curr, rev/fwd
* <, >: play (loop) rev/fwd
* <|, >|: prev/next frame
* fps −/+: decr/incr fps
* ??: HUD toggle
* ||: halt

Note – fps steps: 0.125,0.25,0.5,1,5,7,10,12,15,18,20,24,28,30

<< <*1 < <| >| > >*1 >>

fps- || fps+

2PC: another description

In the two phase commit protocol, as the name implies, there are two phases:

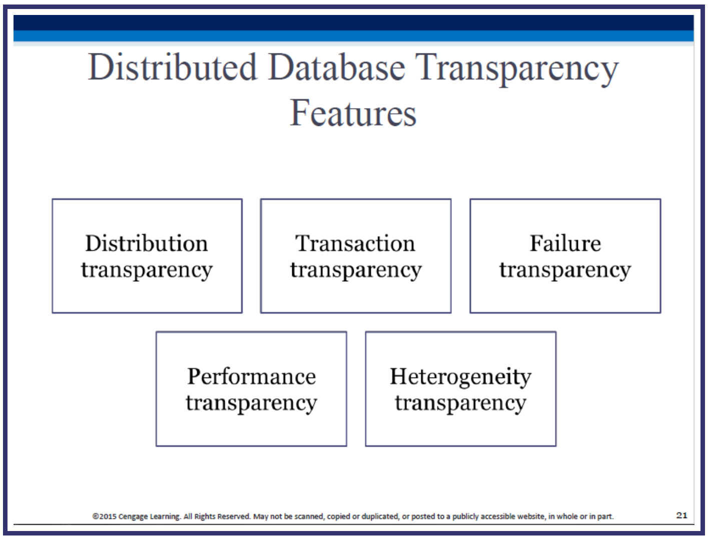
- phase 1: commit request phase, aka 'voting' phase: coordinator sends a 'commit or abort?' (aka 'query to commit' or 'prepare to commit') message to each participating transaction node; each node responds (votes), with a 'can commit' (aka 'ready') or 'need to abort' (aka 'abort') message
- phase 2: commit phase, aka 'completion' phase:
    - if in phase 1, all nodes responded with 'can commit', the coordinator sends a 'commit' phase to each node; each node commits, and sends an acknowledgment to the coordinator
    - or, if in phase 1, any node responded with 'need to abort', the coordinator sends an 'abort' message to each node; each node aborts, and sends an acknowledgment to the coordinator

The devil's in the details - all sorts of variations/refinements exist in implementations, but the above is the overall (simple, robust) idea.

It's an all-or-nothing scheme - either all nodes of a distributed transaction locally commit (in which case the overall transaction is successful), or all of them locally abort so that the DB is not left in an inconsistent state unlike in the 'Problem!' slide (in which case the overall transaction fails and needs to be redone).

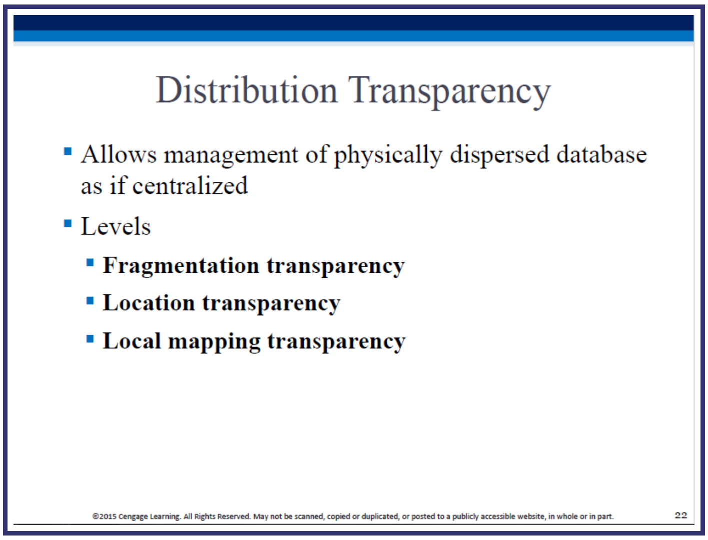
FYI, this is yet another description :)

"What distribution? We have NO distribution!"



DDBMSs are designed to HIDE distribution specifics - this feature is called 'transparency', and can be classified into different kinds.

Distribution transparency



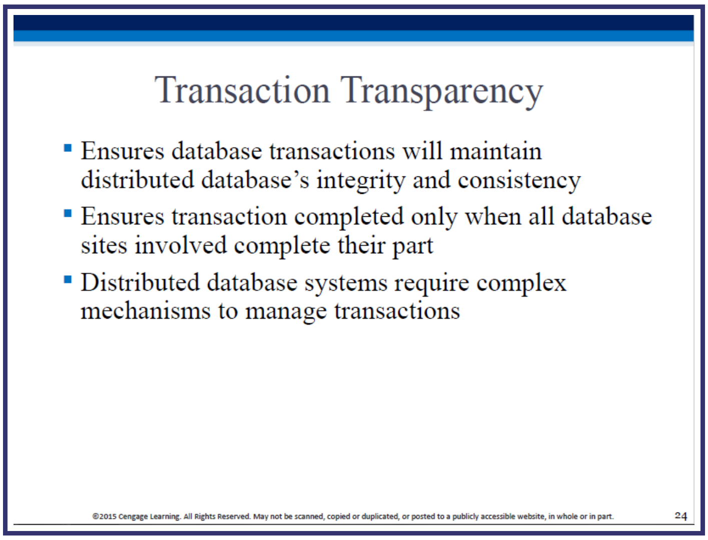
Fragmentation transparency: end user does not know that the data is fragmented.

Location transparency: end user does not know where fragments are located.

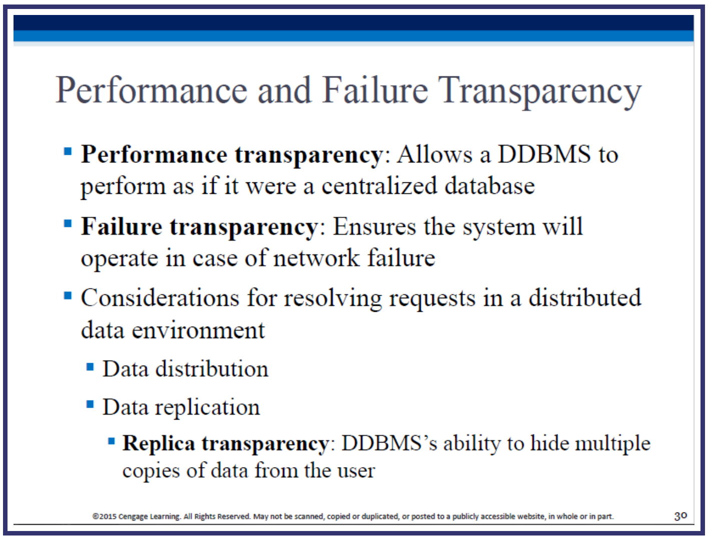
Location mapping transparency: end user does not know how fragments are mapped.

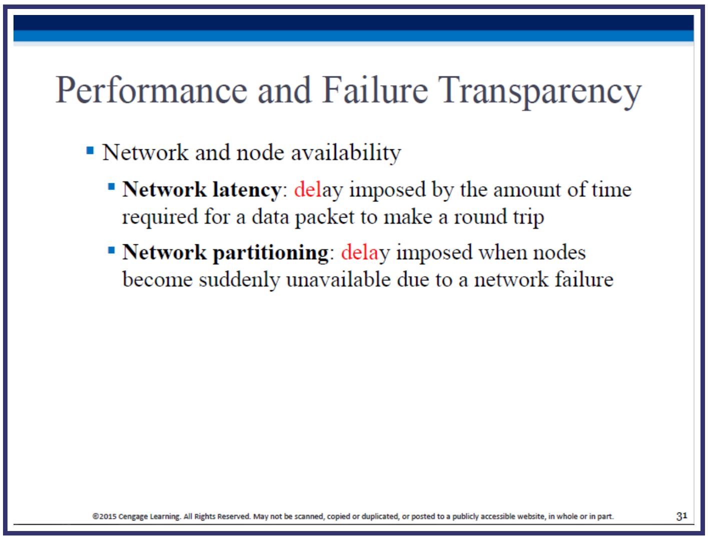
Distrib. transparency is supported via a distributed data dictionary (DDD) [aka DDC], which contains the distributed global schema which local TPs use to translate user requests for processing by DPs.

Transaction transparency



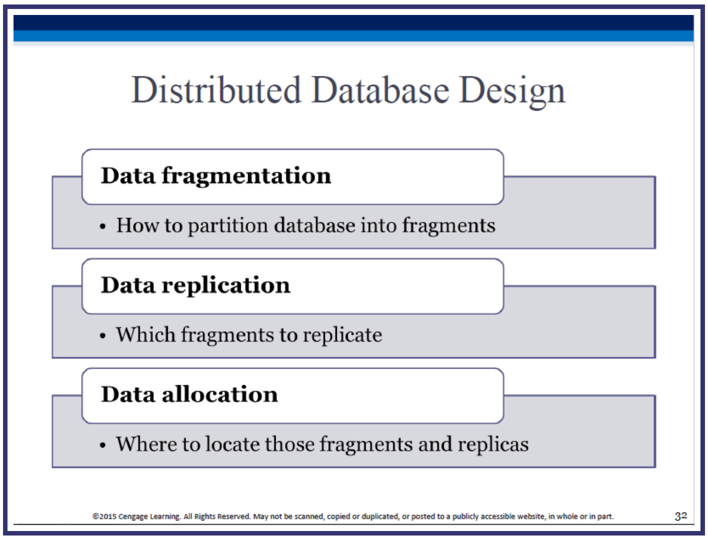
Performance transparency, failure transparency





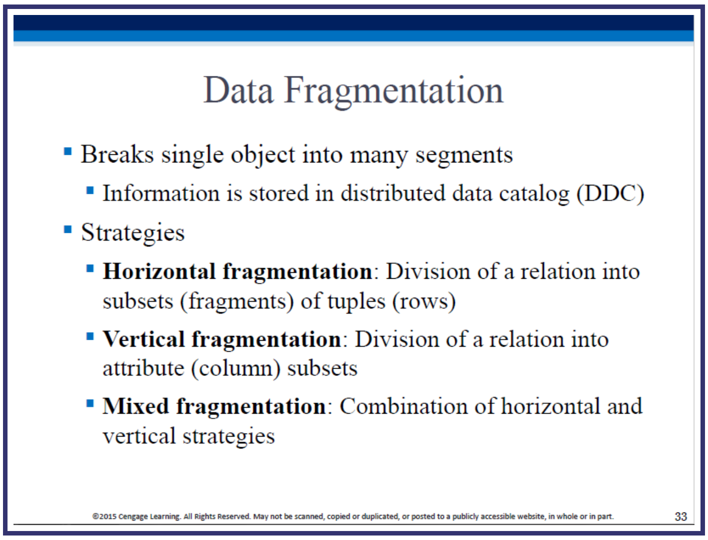
We also need to take into account, network delay and network failure ("partitioning") when planning for or evaluating transparency.

Distributed DB design

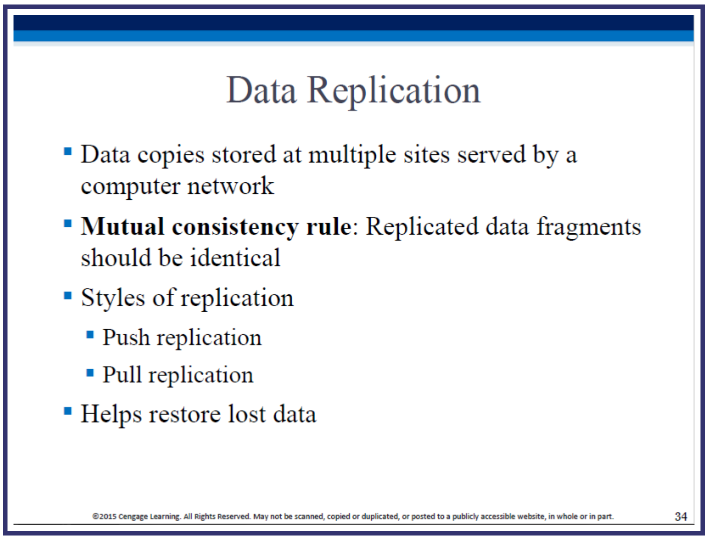


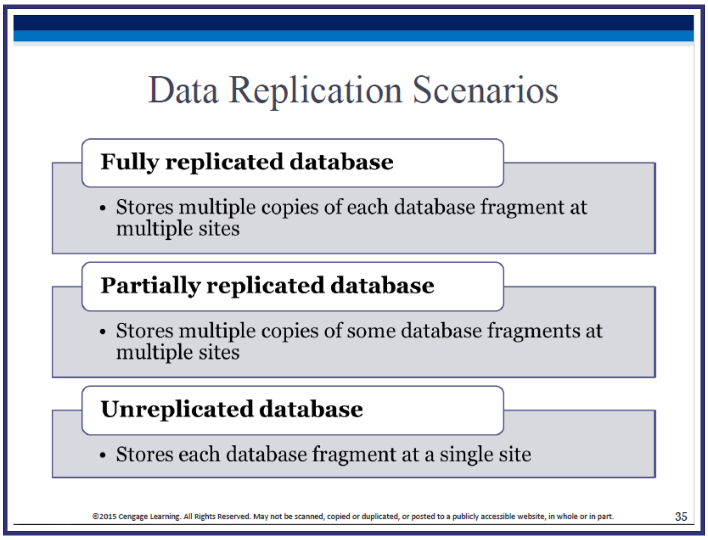
Partition (divide), replicate (copy)? Where to store the partitions/copies?

Fragmentation (partitioning)



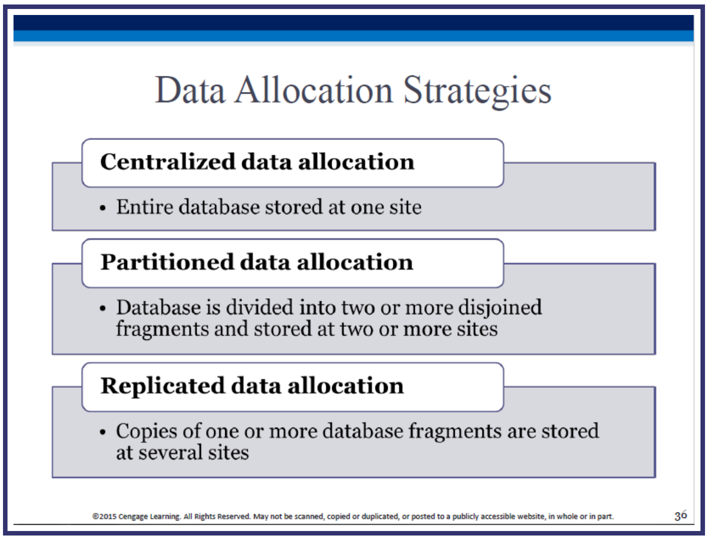
Replication



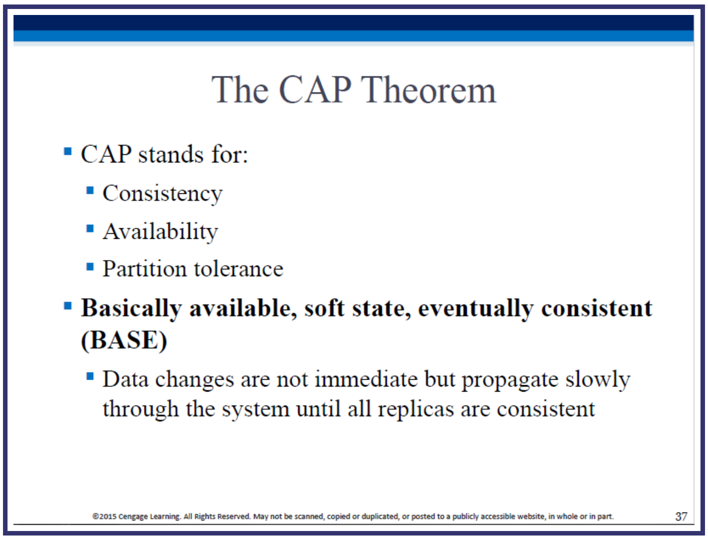


You can read more about replication, here.

Storage of fragments/replicas



The CAP 'theorem'



Consistency: always correct data.

Availability: requests are always filled.

Partition ("outage") tolerance: continue to operate even if (some/most) nodes fail.

The CAP theorem 'used to say' that in a networked (distributed) DB system, at most 2 out of 3 of the above are achievable [PC, CA, or PA]. But CA means low P, that means that we can't even operate, which makes CA a moot point! In other words, CA doesn't exist, ie. a low P is not an option! So now we think of it differently: in the event of a network partition (P has occurred), only one of availability or consistency is achievable.

In the 'ACID' (Atomicity, Consistency, Isolation, Durability) world of older relational DBs, the CAP Theorem was a reminder that it is usually difficult to achieve C,A,P all at once, and that consistency is more important than availability.

In today's 'BASE' (Basically Available, Soft_state, Eventually_consistent) model of non-relational (eg. NoSQL) DBs, we prefer to sacrifice consistency in favor of availability.

DBMS TYPE	CONSISTENCY	AVAILABILITY	PARTITION TOLERANCE	TRANSACTION MODEL	TRADE-OFF
Centralized DBMS	High	High	N/A	ACID	No distributed data processing
Relational DDBMS	High	Relaxed	High	ACID	
(2PC)	Sacrifices availability to ensure con-sistency and isolation				
NoSQL DDBMS	Relaxed	High	High	BASE	Sacrifices consistency to ensure availability

Date's '12 commandments' for distributed DBMSs

RULE NUMBER	RULE NAME	RULE EXPLANATION
1	*Local-site independence*	Each local site can act as an independent, autonomous, centralized DBMS. Each site is responsible for security, concurrency control, backup, and recovery.
2	*Central-site independence*	No site in the network relies on a central site or any other site. All sites have the same capabilities.
3	*Failure independence*	The system is not affected by node failures. The system is in continuous operation even in the case of a node failure or an expansion of the network.
4	*Location transparency*	The user does not need to know the location of data to retrieve those data.
5	*Fragmentation transparency*	Data fragmentation is transparent to the user, who sees only one logical database. The user does not need to know the name of the database fragments to retrieve them.
6	*Replication transparency*	The user sees only one logical database. The DDBMS transparently selects the database fragment to access. To the user, the DDBMS manages all fragments transparently.
7	*Distributed query processing*	A distributed query may be executed at several different DP sites. Query optimization is performed transparently by the DDBMS.
8	*Distributed transaction processing*	A transaction may update data at several different sites, and the transaction is executed transparently.
9	*Hardware independence*	The system must run on any hardware platform.
10	*Operating system independence*	The system must run on any operating system platform.
11	*Network independence*	The system must run on any network platform.
12	*Database independence*	The system must support any vendor's database product.

Think of these more as checklist items, rather than commandments.

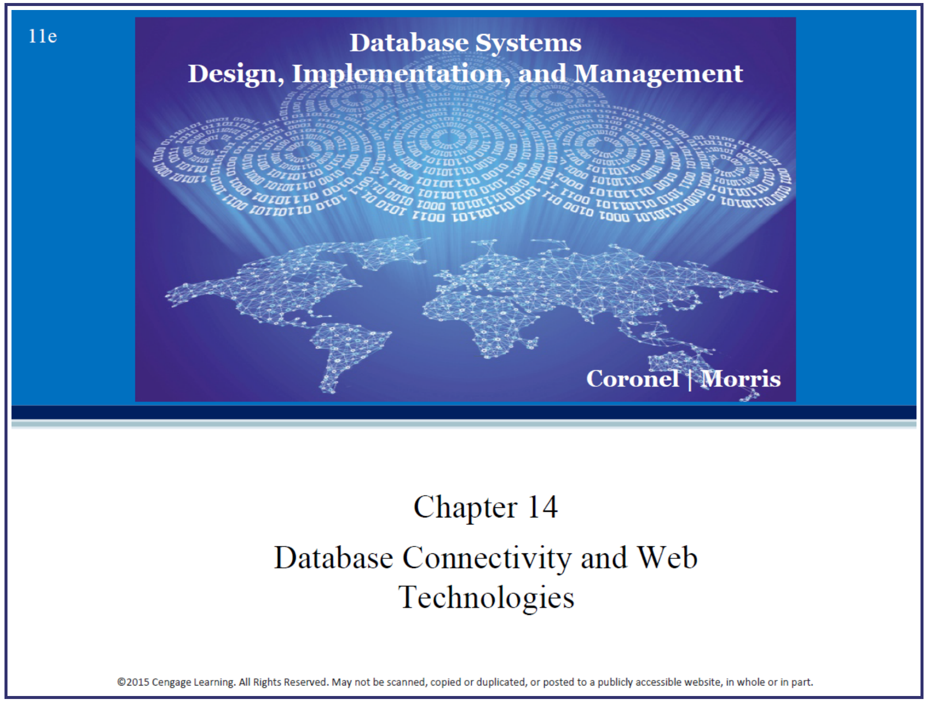
1/30 1:12:17 * * *

← →

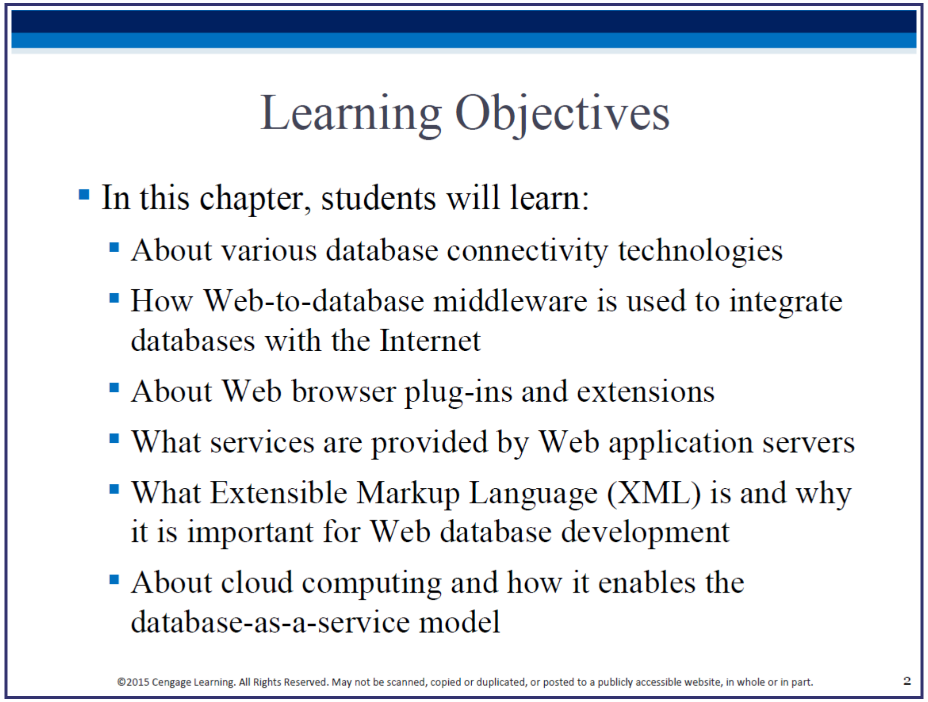
Database connectivity

Database connectivity (1) 2023-03-10, 1:12 PM

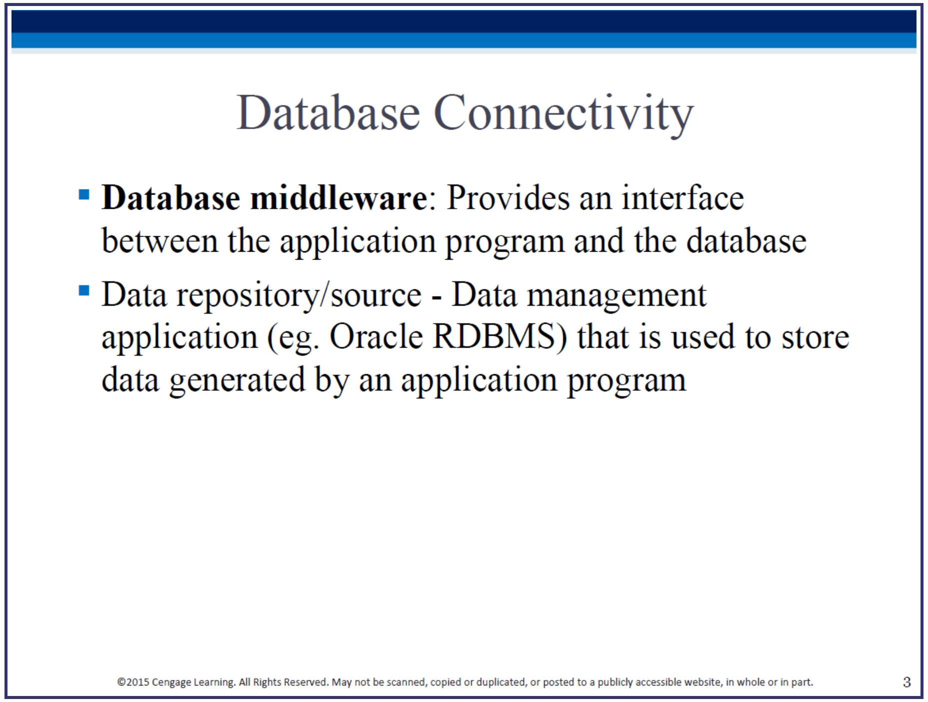
Ch.14



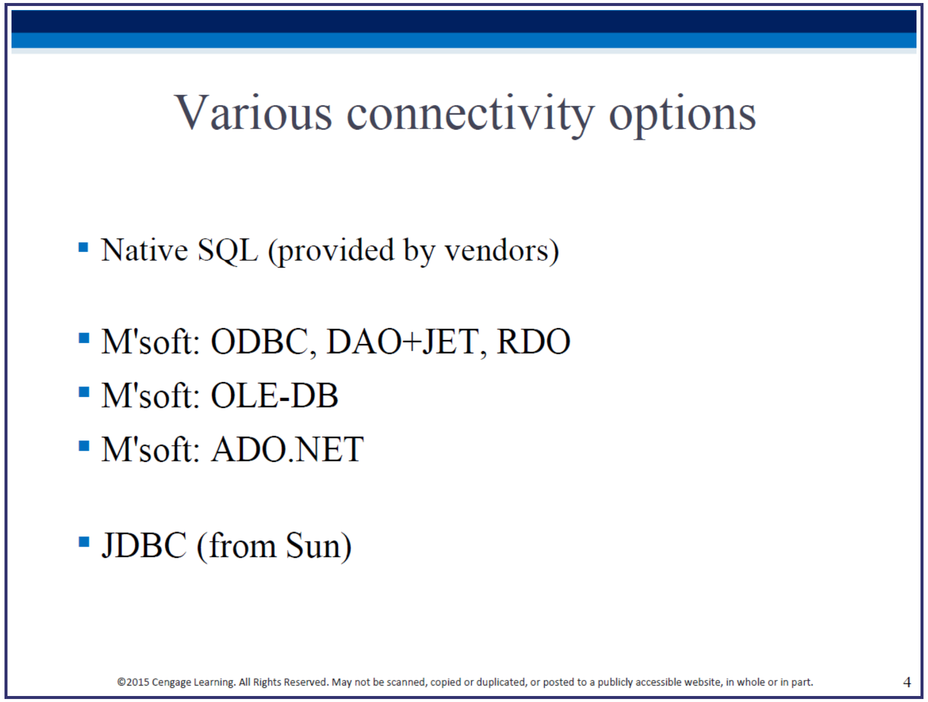
Database connectivity (1) 2023-03-10, 1:12 PM

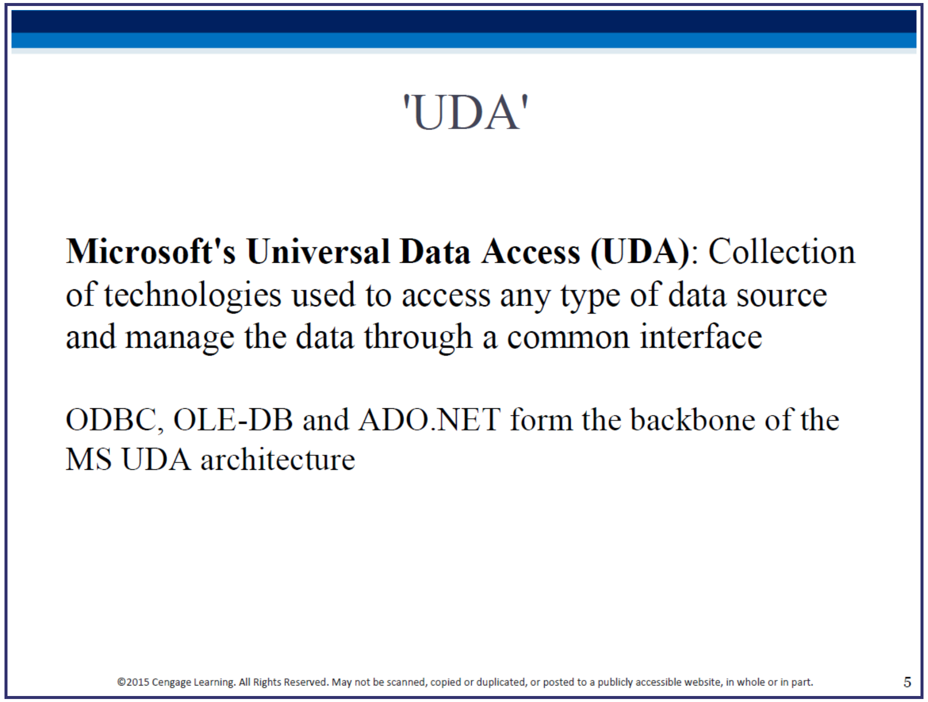

Learning Objectives
In this chapter, students will learn:
About various database connectivity technologies
How Web-to-database middleware is used to integrate databases with the Internet
About Web browser plug-ins and extensions
What services are provided by Web application servers
What Extensible Markup Language (XML) is and why it is important for Web database development
About cloud computing and how it enables the database-as-a-service model
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
2

Database connectivity (1) 2023-03-10, 1:12 PM



Database connectivity (1) 2023-03-10, 1:12 PM



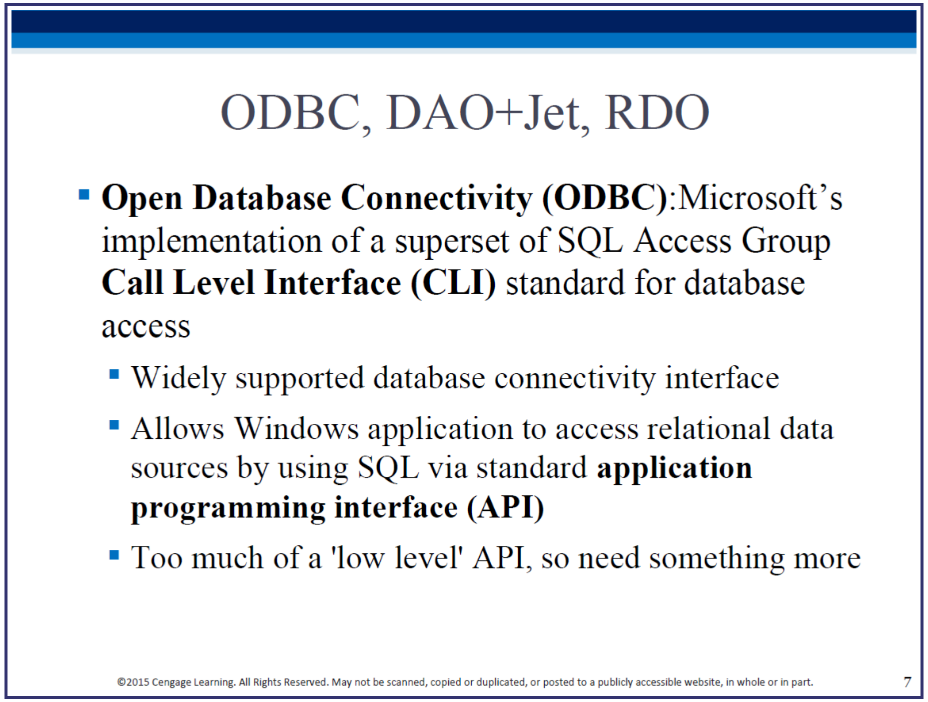

'UDA'
Microsoft's Universal Data Access (UDA): Collection of technologies used to access any type of data source and manage the data through a common interface
ODBC, OLE-DB and ADO.NET form the backbone of the MS UDA architecture
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
5

Database connectivity (1) 2023-03-10, 1:12 PM

Native SQL Connectivity

- Connection interface provided by database vendors, which is unique to each vendor
- Interfaces are optimized for particular vendor's DBMS
- Maintenance is a burden for the programmer

 6



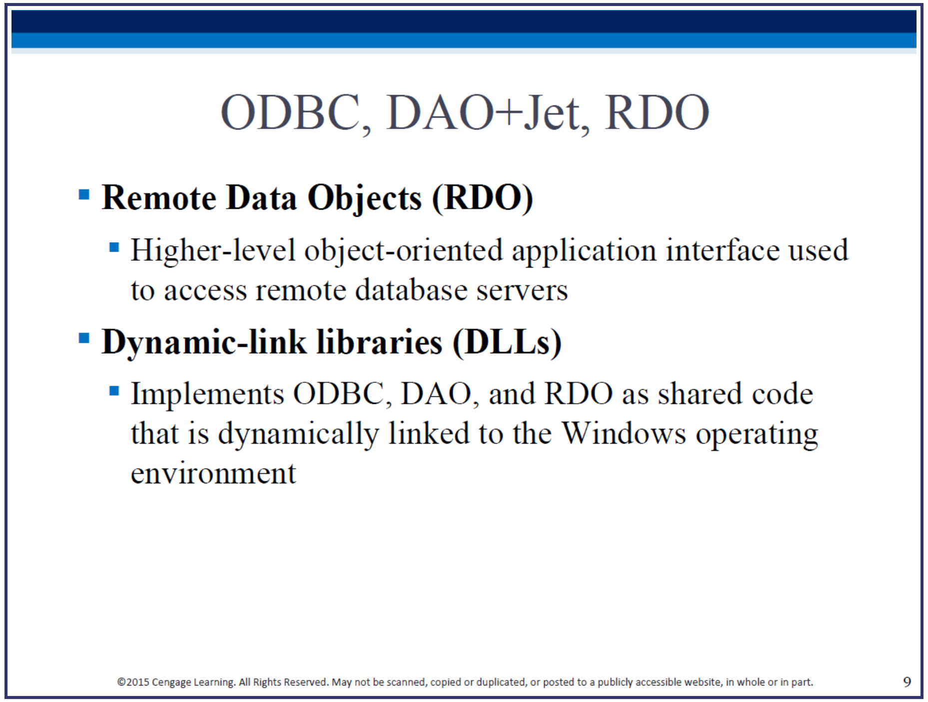
JET: 'Joint Engine Technology'.

Database connectivity (1) 2023-03-10, 1:12 PM

Database connectivity (1) 2023-03-10, 1:12 PM

ODBC, DAO+Jet, RDO

- **Data Access Objects (DAO)**: Object-oriented API used to access MS Access, MS FoxPro, and dBase databases from Visual Basic programs
  - Provides an optimized interface that expose functionality of **Jet** data engine to programmers
  - DAO interface can be used to access other relational style data sources as well



Database connectivity (1) 2023-03-10, 1:12 PM

Database connectivity (1) 2023-03-10, 1:12 PM

Components of ODBC Architecture

- High-level ODBC API through which application programs access ODBC functionality
- Driver manager that is in charge of managing all database connections
- ODBC driver that communicates directly to DBMS

 10

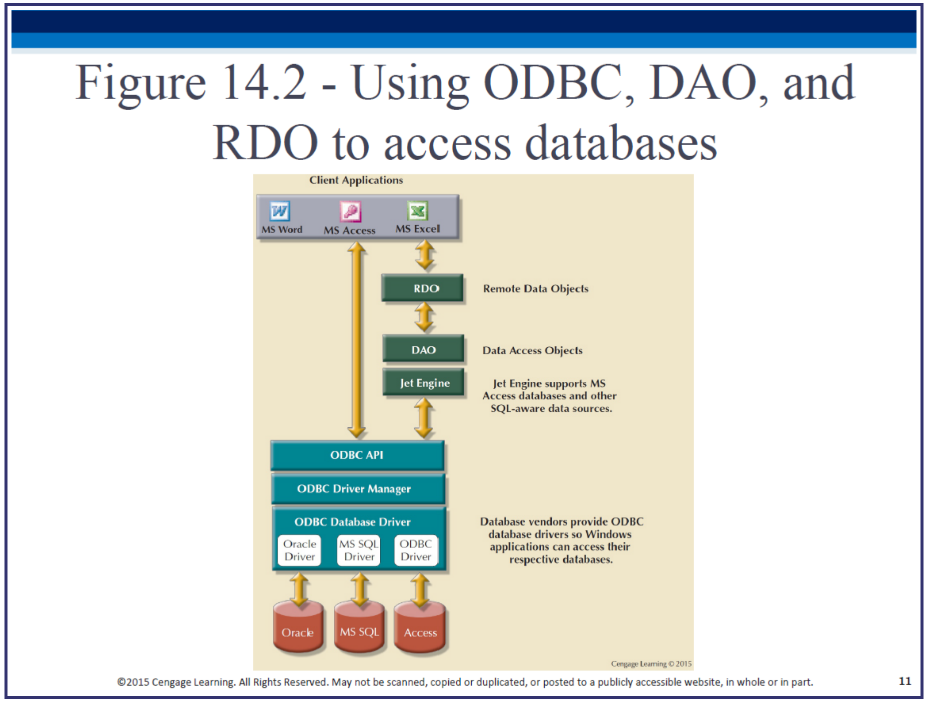
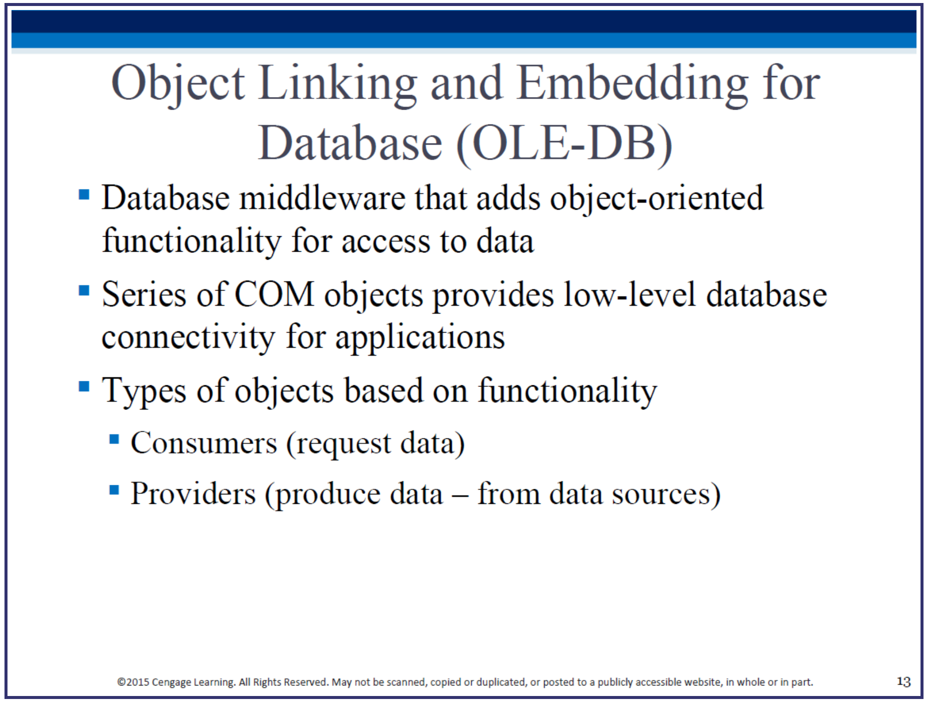


Figure 14.2 - Using ODBC, DAO, and RDO to access databases
Client Applications
MS Word
MS Access
MS Excel
RDO
Remote Data Objects
DAO
Data Access Objects
Jet Engine
Jet Engine supports MS Access databases and other SQL-aware data sources.
ODBC API
ODBC Driver Manager
ODBC Database Driver
Oracle Driver
MS SQL Driver
ODBC Driver
Database vendors provide ODBC database drivers so Windows applications can access their respective databases.
Oracle
MS SQL
Access
Cengage Learning © 2015
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
11

Database connectivity (1) 2023-03-10, 1:12 PM



COM was modeled after OMG's CORBA... Here is more...

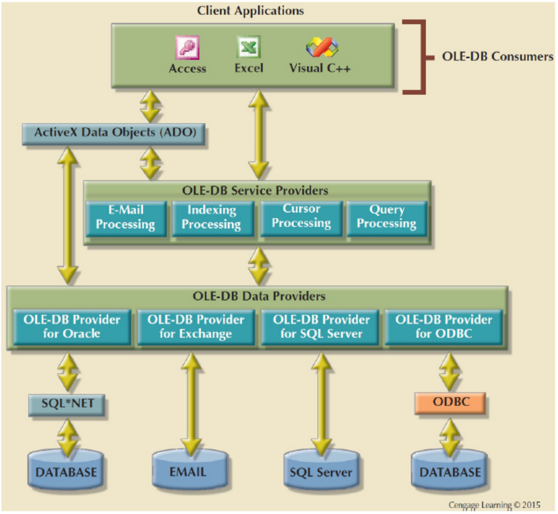
Database connectivity (1) 2023-03-10, 1:12 PM

Object Linking and Embedding for Database (OLE-DB)

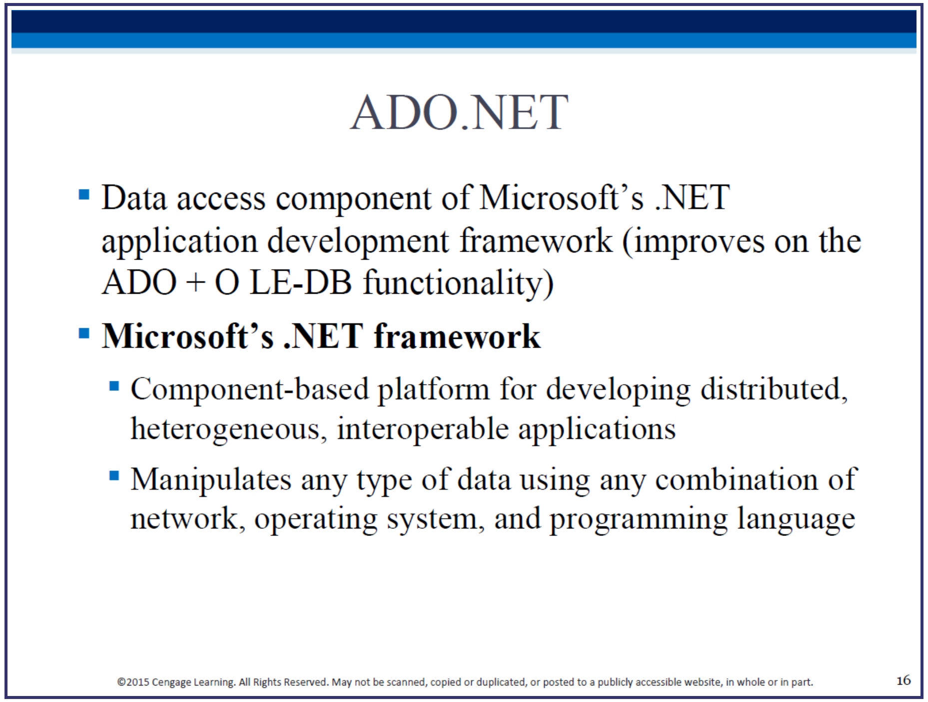
- Does not provide support for scripting languages
- **ActiveX Data Objects (ADO)**: Provides:
  - High-level application-oriented interface to interact with OLE-DB, DAO, and RDO
  - Unified interface to access data from any programming language that uses the underlying OLE-DB objects

 14

Figure 14.5 - OLE-DB Architecture



 15


ADO.NET
Data access component of Microsoft's .NET application development framework (improves on the ADO + O LE-DB functionality)
Microsoft's .NET framework
Component-based platform for developing distributed, heterogeneous, interoperable applications
Manipulates any type of data using any combination of network, operating system, and programming language
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
16

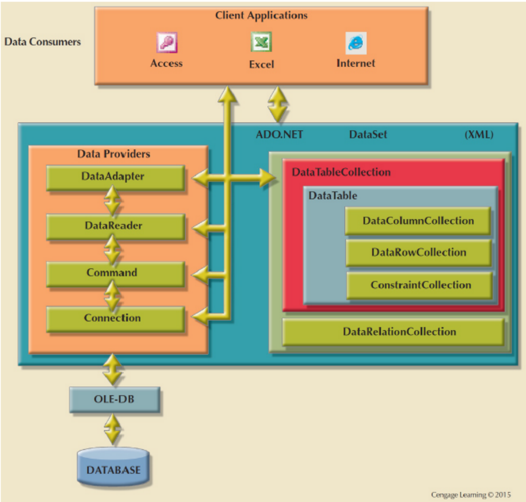
Database connectivity (1) 2023-03-10, 1:12 PM

ADO.NET

- Features critical for the development of distributed applications
  - **DataSet**: Disconnected memory-resident representation of database
  - **XML support**
    - DataSet is internally stored in XML format
    - Data in DataSet could be made persistent as XML documents

Database connectivity (1) 2023-03-10, 1:12 PM

Figure 14.6 - ADO.NET Framework

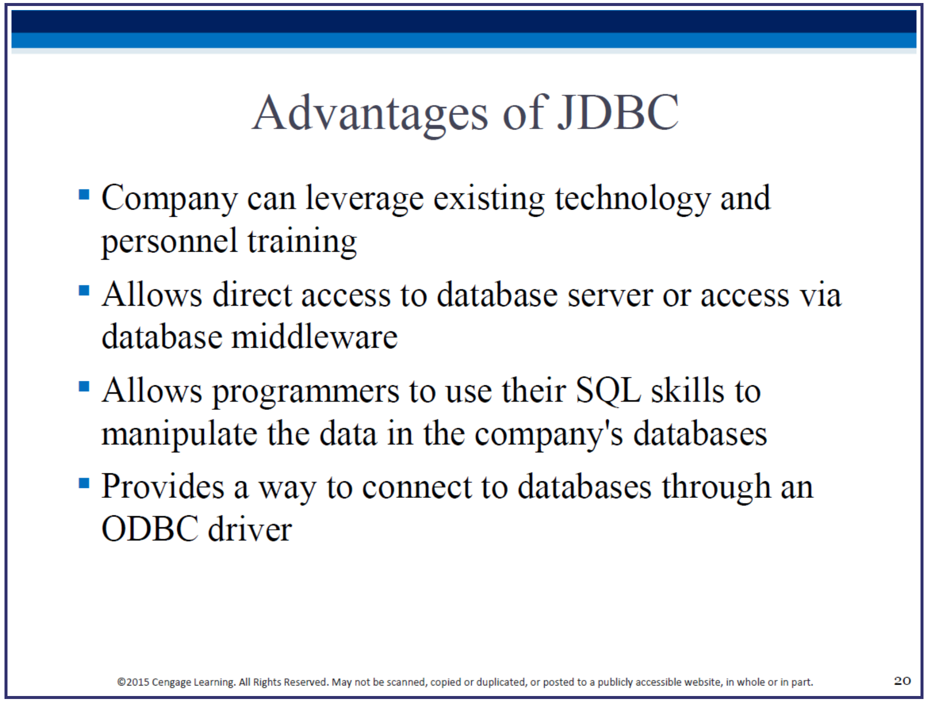


 18

Java Database Connectivity (JDBC)

- **Java**: Object-oriented programming language that runs on top of web browser software, on smartphones, on the desktop and on servers
- JDBC: Application programming interface that allows a Java program to interact with a wide range of data sources – a simple URL is all is needed to connect to a db

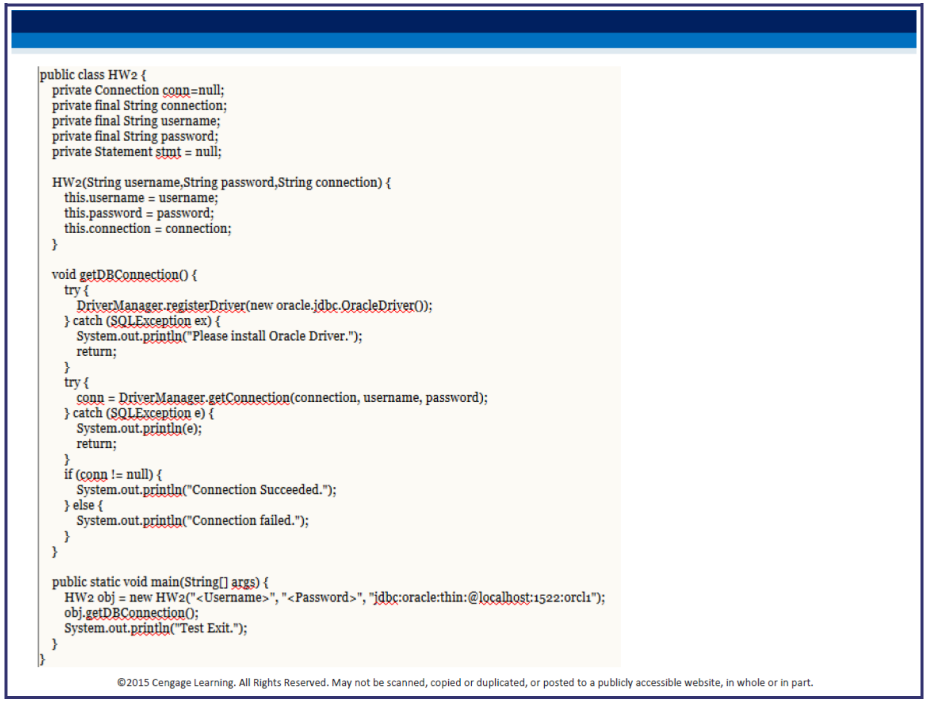
Database connectivity (1) 2023-03-10, 1:12 PM



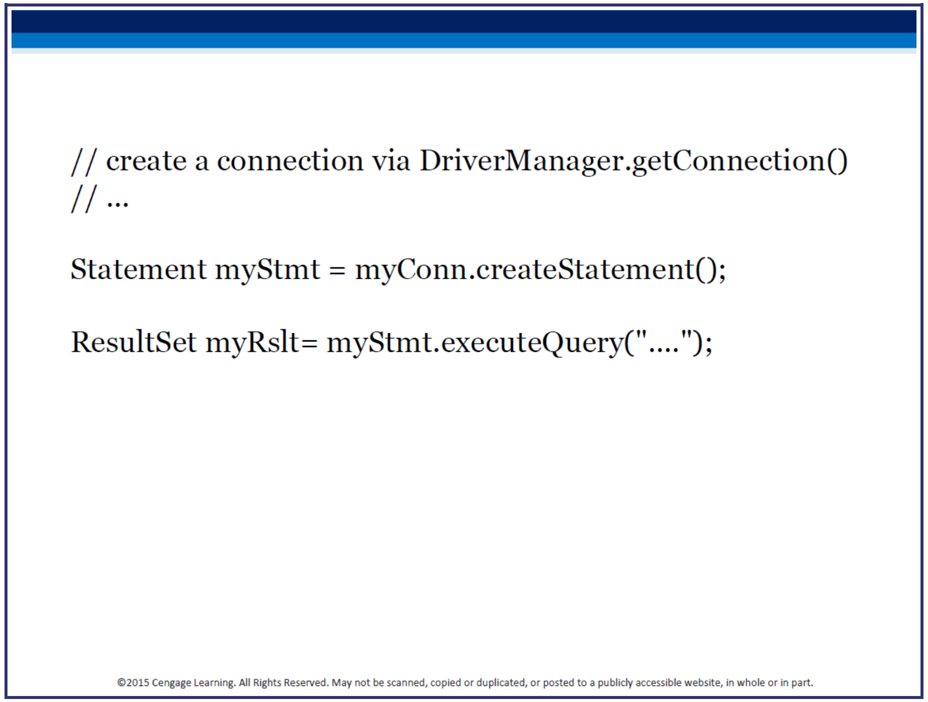
Database connectivity (1) 2023-03-10, 1:12 PM

JDBC usage example

Here is some sample code that uses the JDBC API:

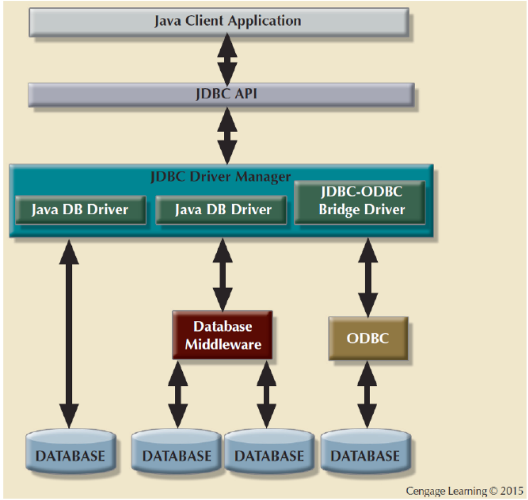


After we create a connection, we can create a statement, execute a query, get back a result set and iterate through it:

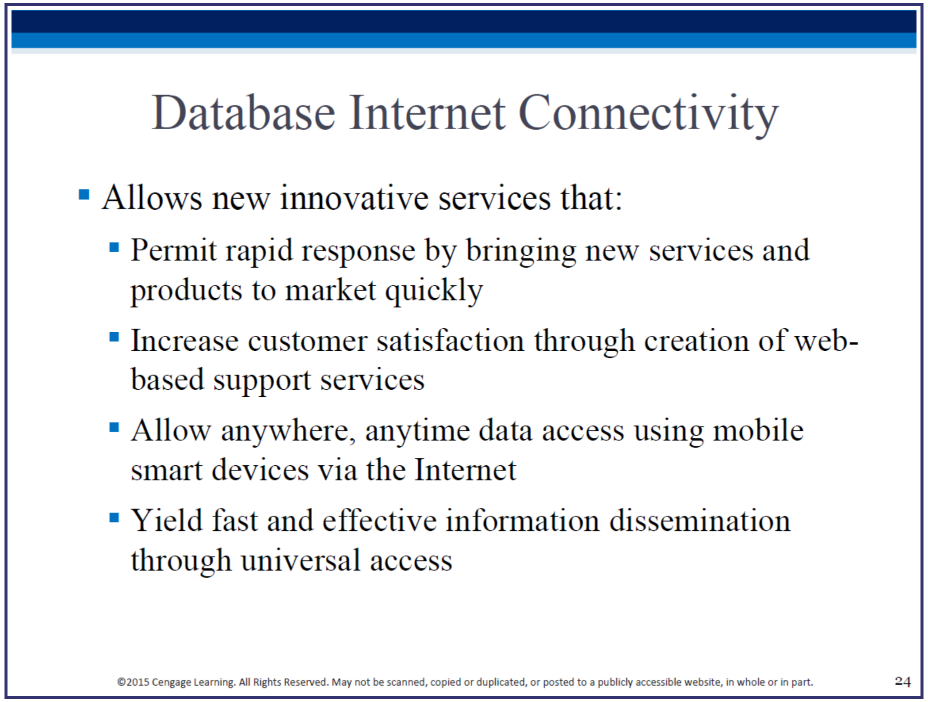


Here is a complete example.

Figure 14.7 - JDBC Architecture



 23

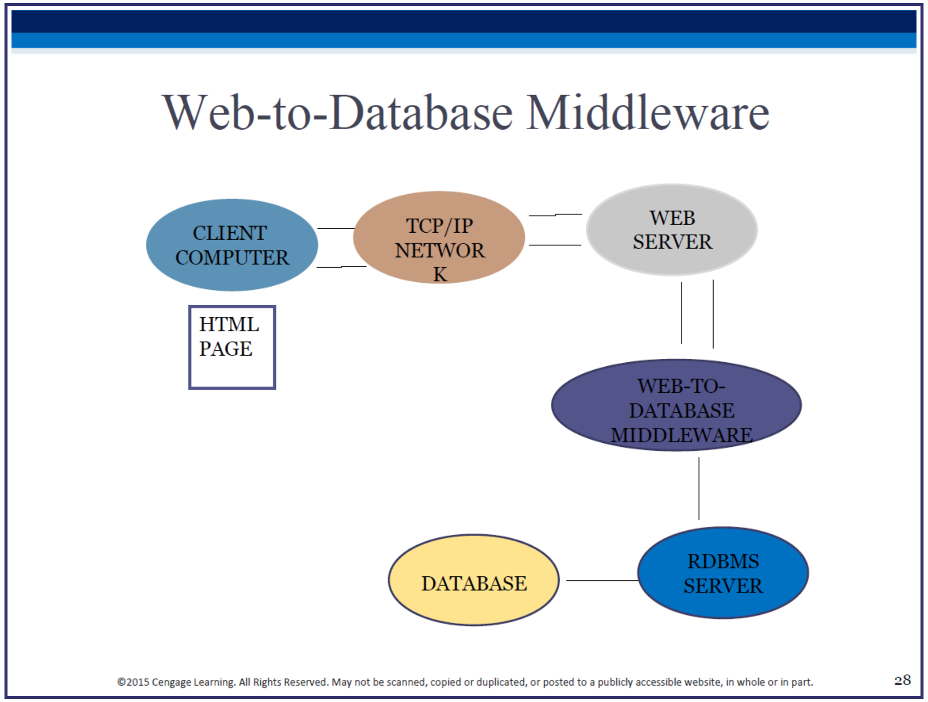


Why? One word: "e-Commerce" :)

Web-to-Database Middleware

- Web server is the main hub through which Internet services are accessed
- **Server-side extension**: Program that interacts directly with the web server
  - Also known as **web-to-database middleware**
  - Provides its services to the web server in a way that is totally transparent to the client browser

27


Web-to-Database Middleware
CLIENT COMPUTER
TCP/IP NETWOR K
WEB SERVER
HTML PAGE
WEB-TO-DATABASE MIDDLEWARE
DATABASE
RDBMS SERVER
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
28

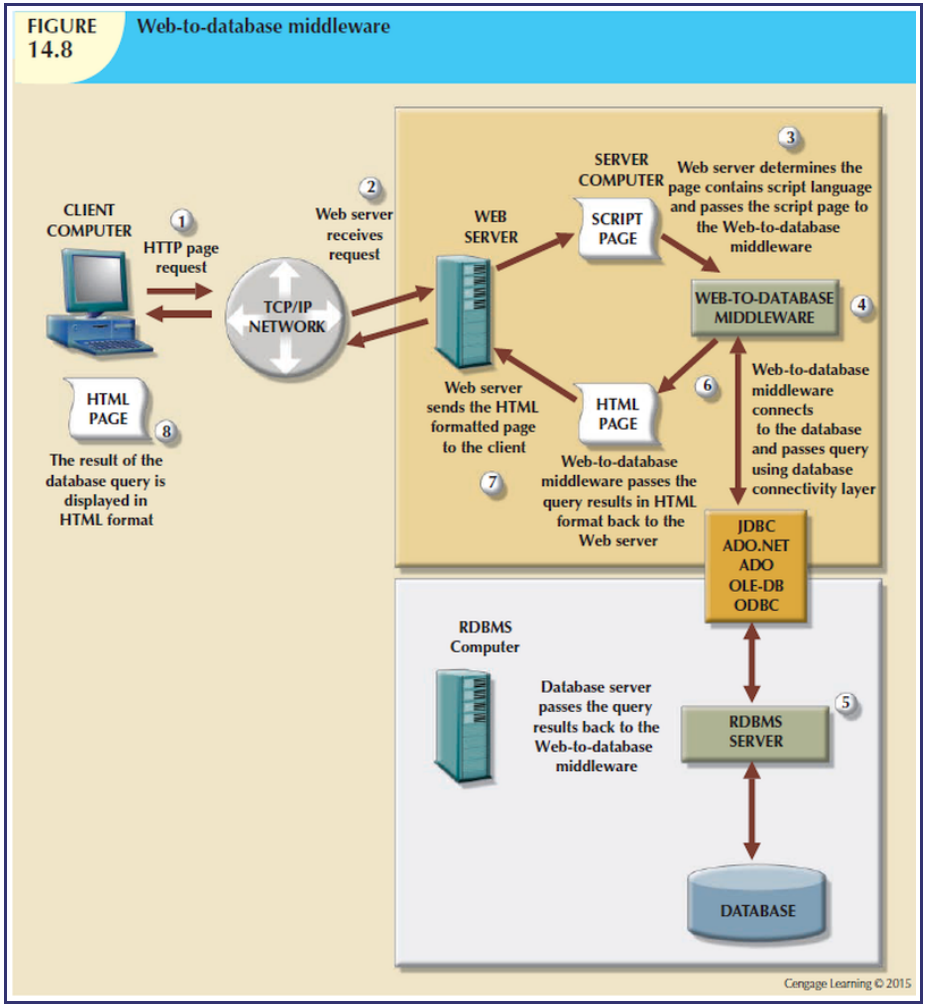
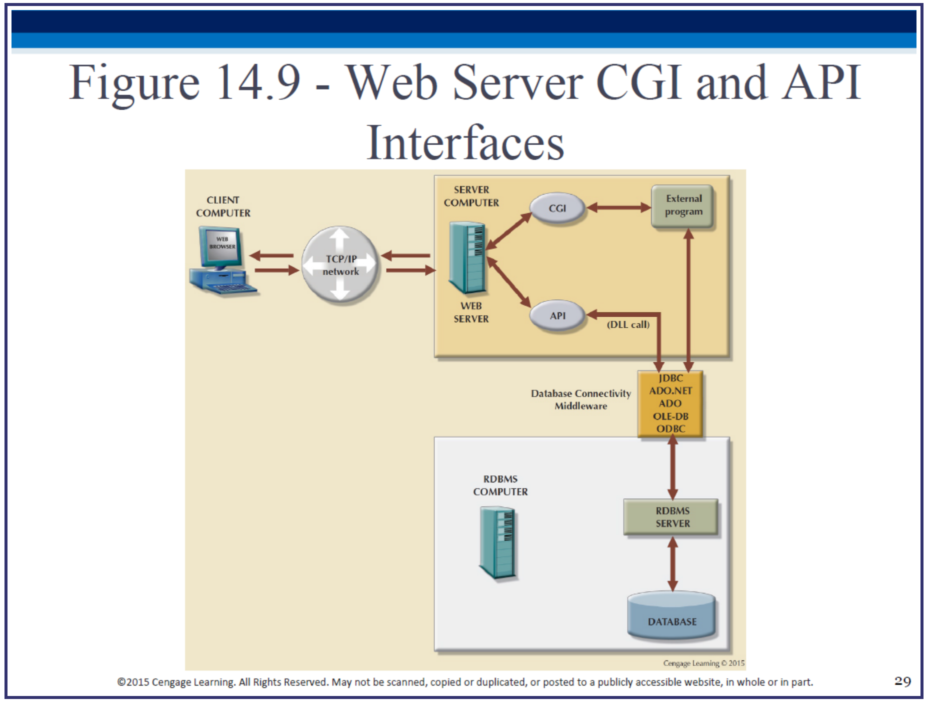


FIGURE 14.8
Web-to-database middleware
CLIENT COMPUTER
1 HTTP page request
TCP/IP NETWORK
2 Web server receives request
WEB SERVER
SERVER COMPUTER
SCRIPT PAGE
3 Web server determines the page contains script language and passes the script page to the Web-to-database middleware
WEB-TO-DATABASE MIDDLEWARE 4
Web-to-database middleware connects to the database and passes query using database connectivity layer
JDBC
ADO.NET
ADO
OLE-DB
ODBC
RDBMS Computer
Database server passes the query results back to the Web-to-database middleware
RDBMS SERVER 5
DATABASE
6 HTML PAGE
Web-to-database middleware passes the query results in HTML format back to the Web server
Web server sends the HTML formatted page to the client 7
HTML PAGE 8
The result of the database query is displayed in HTML format
Cengage Learning © 2015

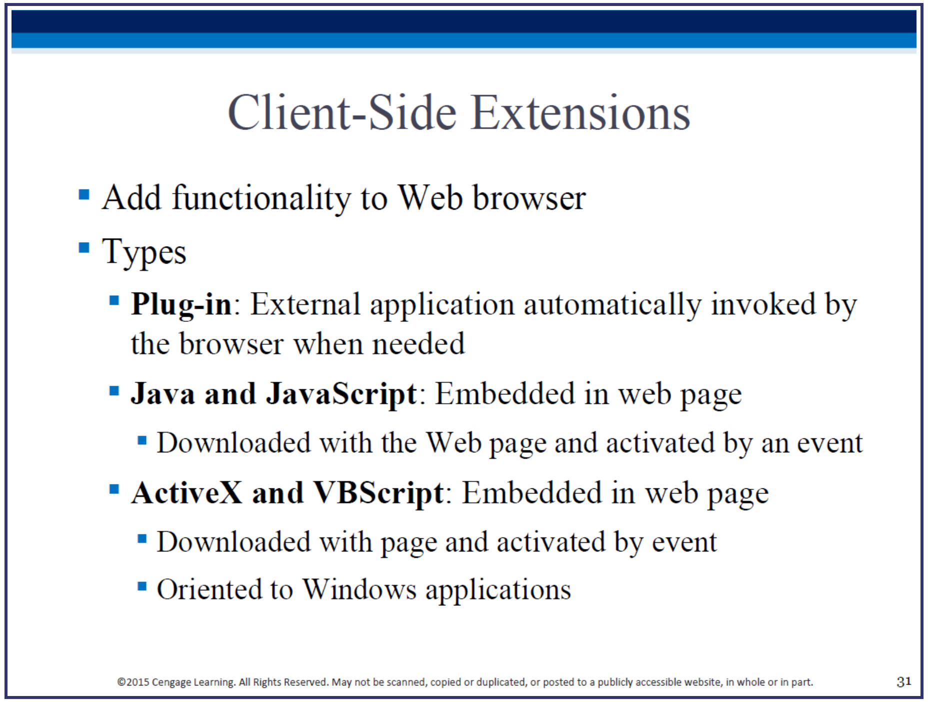


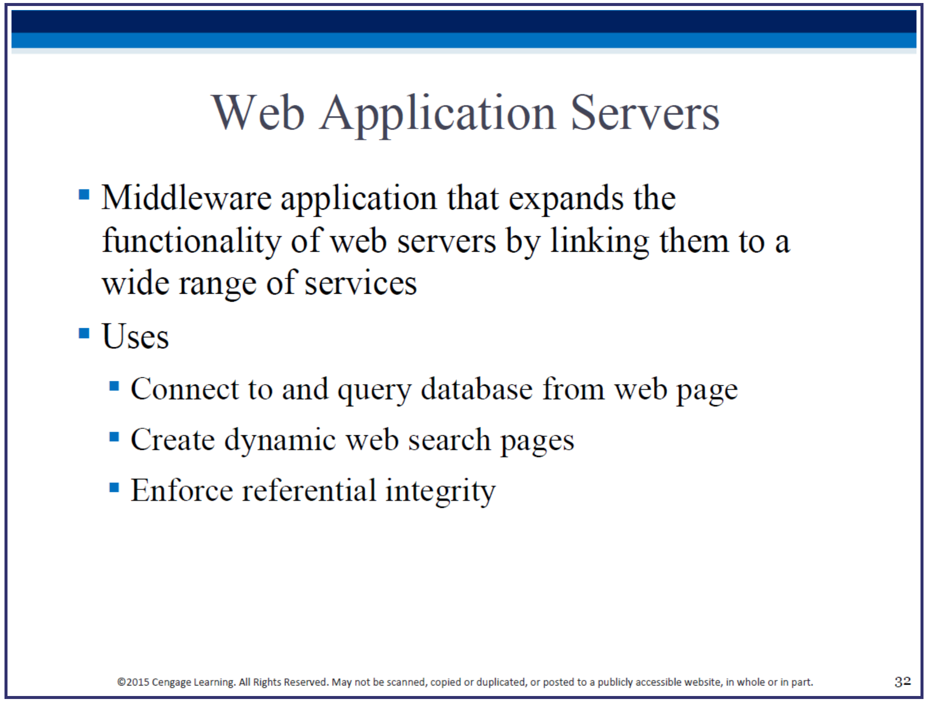
Here is Perl 'CGI' script to interface w/ a DB.

Here are notes on using CORBA to interface a webserver with a DB. These can also be used: JSP, ASP.

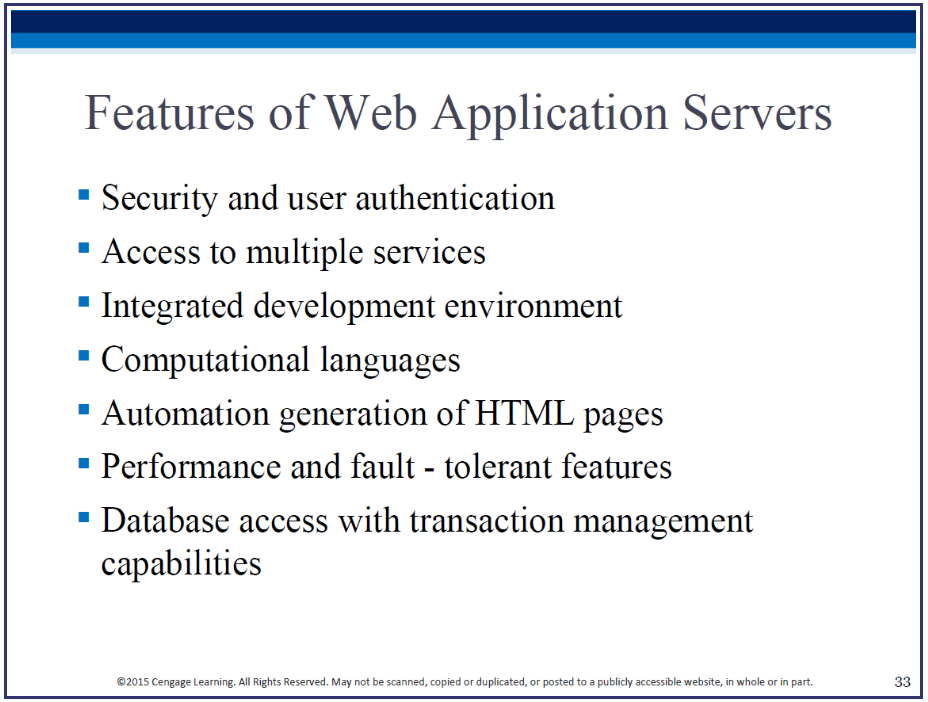
In addition to CGI scripts and APIs, server side includes (SSIs) can also be used for DB connectivity.

As an FYI note, Java provides very many ways to connect a client to a DB via a webserver: applets, sockets, servlets, RMI, CORBA, agents..


Client-Side Extensions
Add functionality to Web browser
Types
Plug-in: External application automatically invoked by the browser when needed
Java and JavaScript: Embedded in web page
Downloaded with the Web page and activated by an event
ActiveX and VBScript: Embedded in web page
Downloaded with page and activated by event
Oriented to Windows applications
©2015 Cengage Learning. All Rights Reserved. May not be scanned, copied or duplicated, or posted to a publicly accessible website, in whole or in part.
31



A 'web application server' is a specialized server that interfaces with web services such as databases, search engines. The client (eg browser) can query these data sources and have results generated dynamically.



Examples of web application servers: WebLogic, ColdFusion/JRun, WebSphere Application Server, WebObjects, IIS, WildFly (JBoss), Tomcat, Jetty..

Each web application server (WAS) offers its own programming environment. Eg. CFML can be used to consume web services and present results for the end user (likewise for DB result sets).