CST3513 Lecture Notes

Database Design

Database Design Phase – Physical DB Design

(Part 1 of 3)

(Lecture Notes 2A)

Prof. Abel Angel Rodriguez

Chapter 5 Physical Database Design

5.1 Database Application Development Lifecycle – Review of Basic Concepts

5.1.1 Overview of the Design Phase

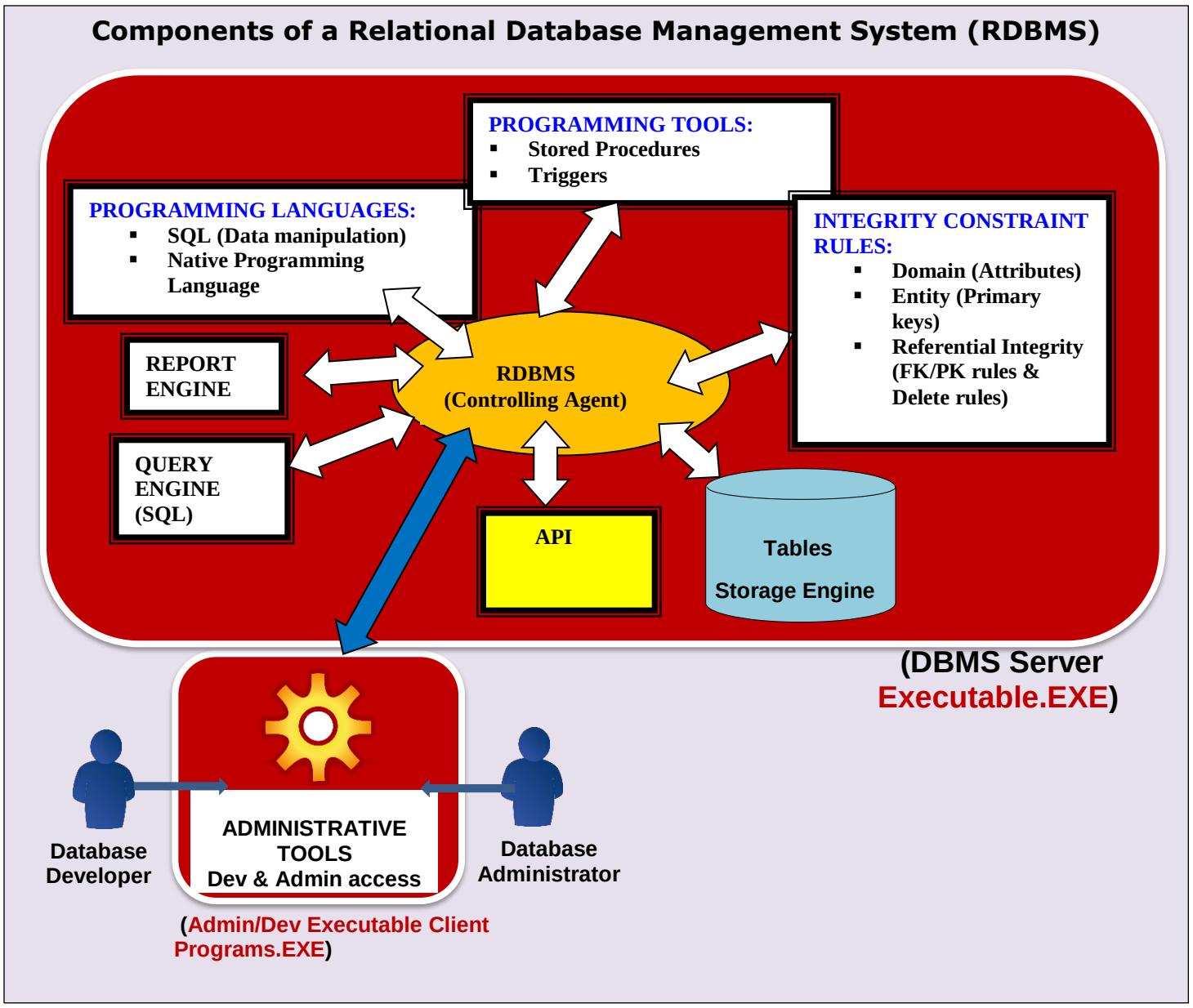
- Design Phase Review:

Phase 3: ***Design*** 	▪ ***Purpose:*** - *Develop a detailed design of database Information System based on all specifications and requirements.* - *Create a **Logical Data Model (Logical Schema)*** - *Create a **Physical Data Model (1. Physical Schema & 2. Technical Specification for Performance/Efficiency, Data integrity, Security, Backup/Disaster Recovery, etc.)*** ▪ ***Database Professional Role:*** - ***Database Analyst** – Logical Data Model* - ***Database Administrator & Analyst** – Physical Data Model* - ***All Roles may be involved*** ▪ ***Activities include:*** 1. *Create detailed **Logical Data Model (Logical Schema)** that defines:* • ***EER Diagram** in terms of **DBMS** (SQL Server, Oracle, etc.) to be used* • ***All entities, attributes and relationships** in terms of **DBMS** to be used.* • ***Business rules** & constraints that dictate data integrity in terms of **DBMS** to be used* • ***NORMALIZATION** – Process of breaking down tables with abnormalities to produce smaller, well-structure tables to reduce redundancy and inconsistencies.* • *Other activities* 2. *Create detailed **Physical Data Model (Physical Schema)**:* • *Specify how Logical Schema is physically stored in secondary memory (Physical storage) by the specified **DBMS** to be used* • *Define the indexes to be use & regulatory compliance* • *Define storage & hardware requirements, etc.* ▪ ***Project Management Activities:*** • *PM Activities – Create Design Document, Update Plan etc.* ▪ ***Deliverables:*** 1. ***Primary Database Deliverable – Logical Data Model (Logical Schema)*** 2. ***Primary Database Deliverable – Physical Data Model (Physical Schema)*** 3. *Other project related documents such as Project Plan etc.* ▪ ***Timeline/duration:*** - *TBD based on timelines & project requirements.*

5.1.2 Review of Database Management System (DBMS)

Database Management System Components

- Illustration of components of a typical DBMS:



- Examples of Database Management System (DBMS) Servers/Programs are:

  - **Oracle**
  - **Microsoft SQL Server**
  - **Sybase SQL Server**
  - **MySQL**
  - **IBM DB2**

    - **Note that Microsoft Access is a Personal or Desktop Database, not true DBMS**

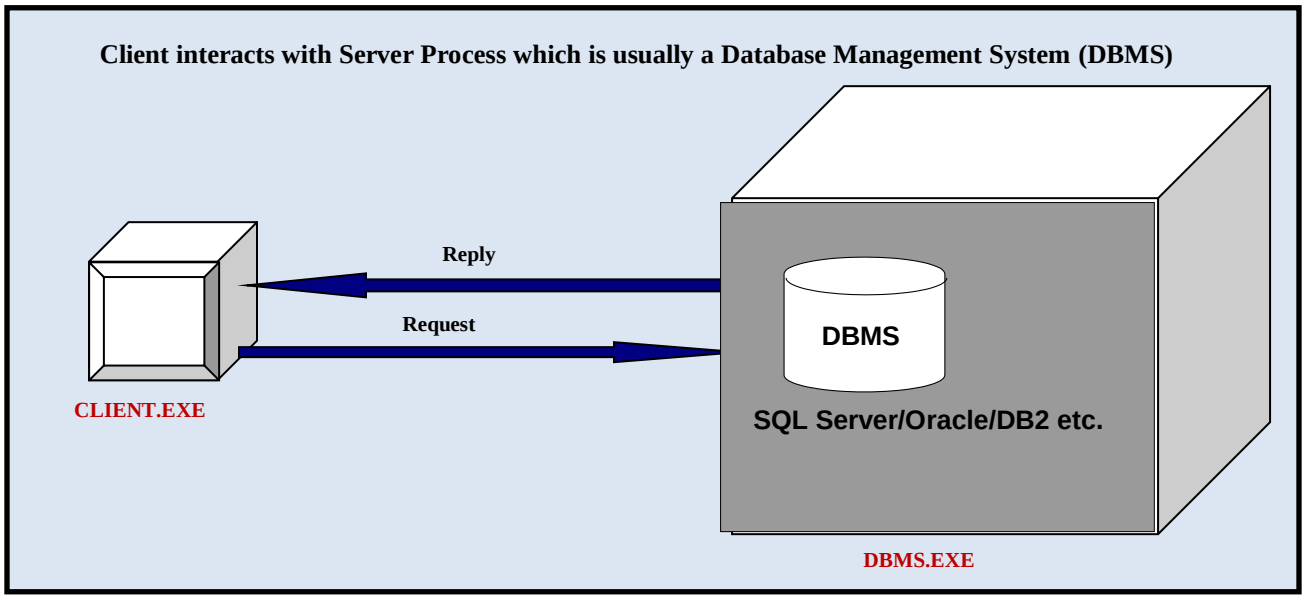
Database Application – Typical Client/Server Architecture (Client process & DBMS)

The Architectures

- Illustration of a Database Application architecture:

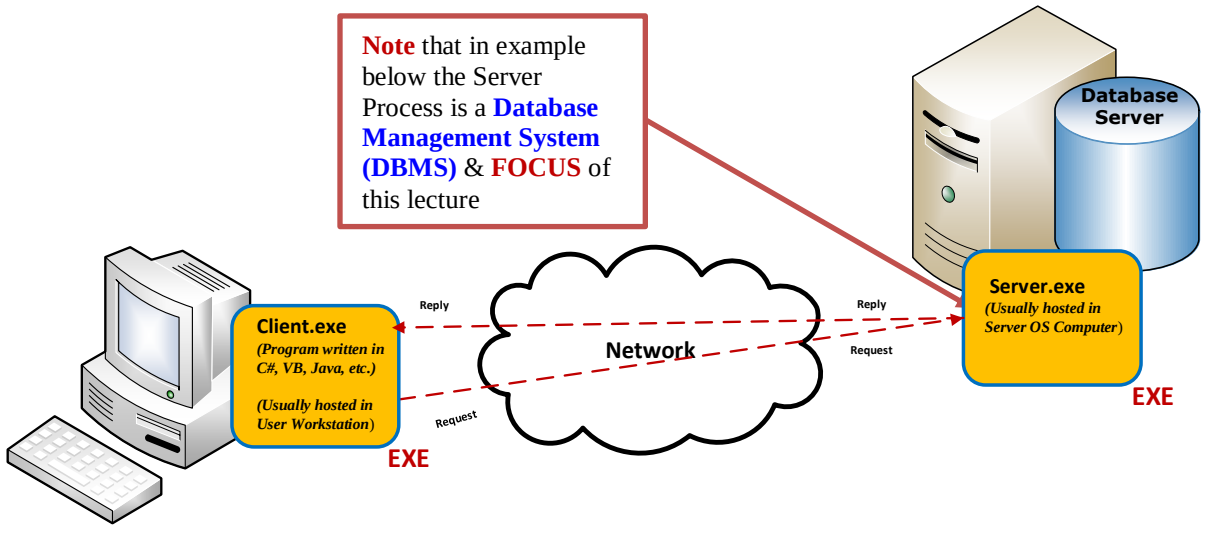
SCENARIO #1 – 2-Tiered-Client/Server: Server is a Database Management System (DBMS)

- Typically, a Client Executable created in Java, C#, other, interacting with a Database Management System (DBMS)

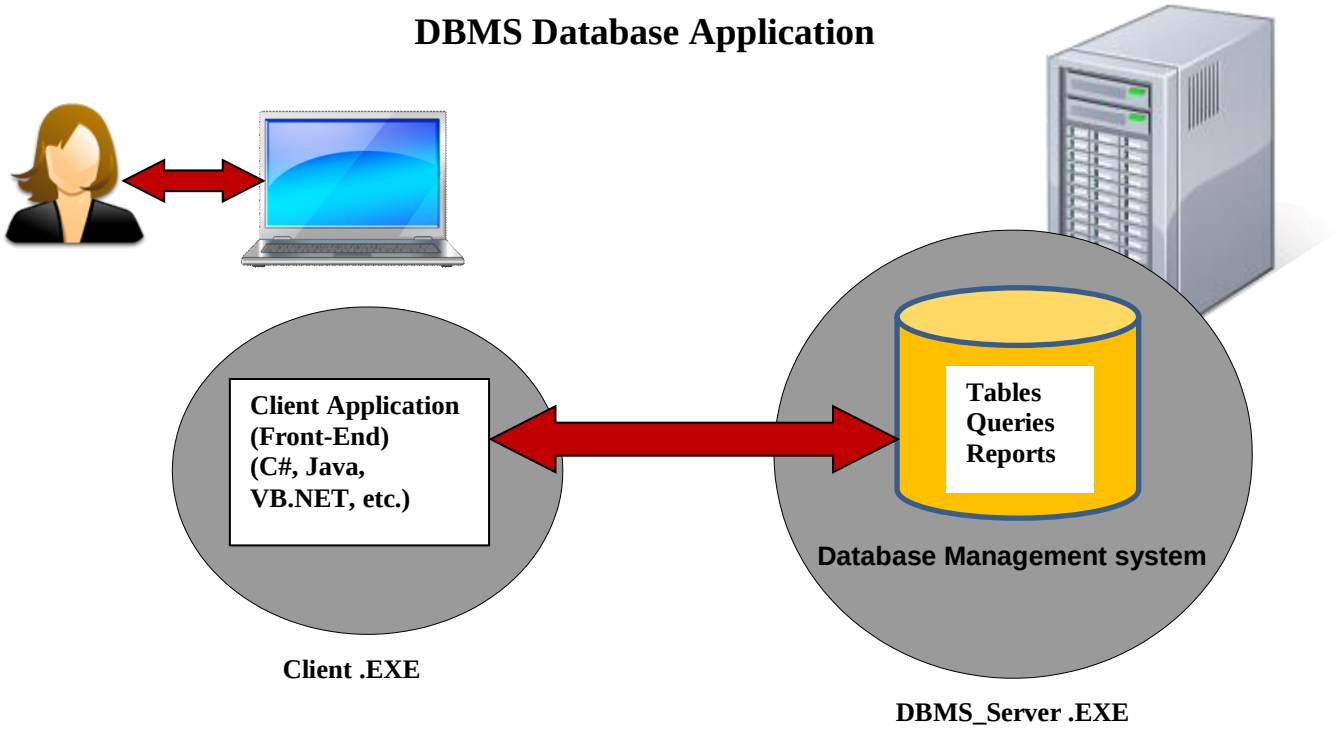


Database Application - Client/Server Application Architecture

- **IMPORTANT!!!** - Note that there are TWO EXECUTABLES involved: **A CLIENT.EXE** and a **DATABASE_SERVER.EXE**.

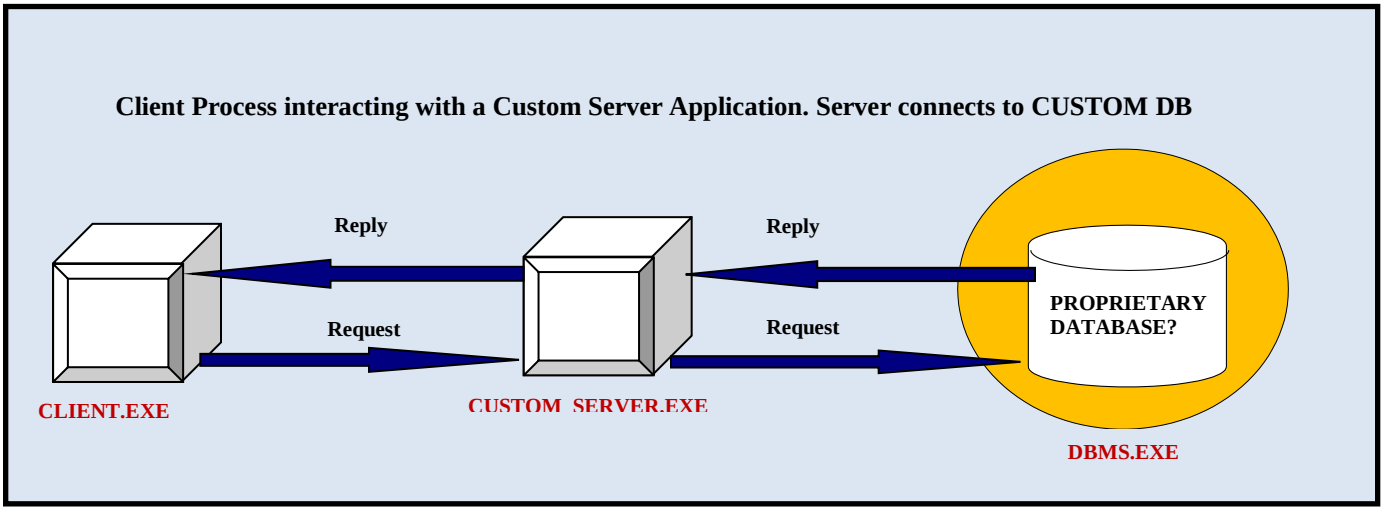


- Therefore when we usually refer to a Client/Server application we really mean a database application with a Client executable written in C#, C++, Java etc., while the Server is a DBMS such as MS SQL Server, Oracle etc. as previously shown:



SCENARIO #2 – 3-Tiered-Client/Server: Server is a Custom Program but CONNECTS to a PROPRIETARY Database

- In the remaining % of Client/Server implementations, the SERVER can be some kind of custom program or server executable but there is still some kind of DATABASE component which is a CUSTOM PROPRIETARY Database created by the developer of the application.

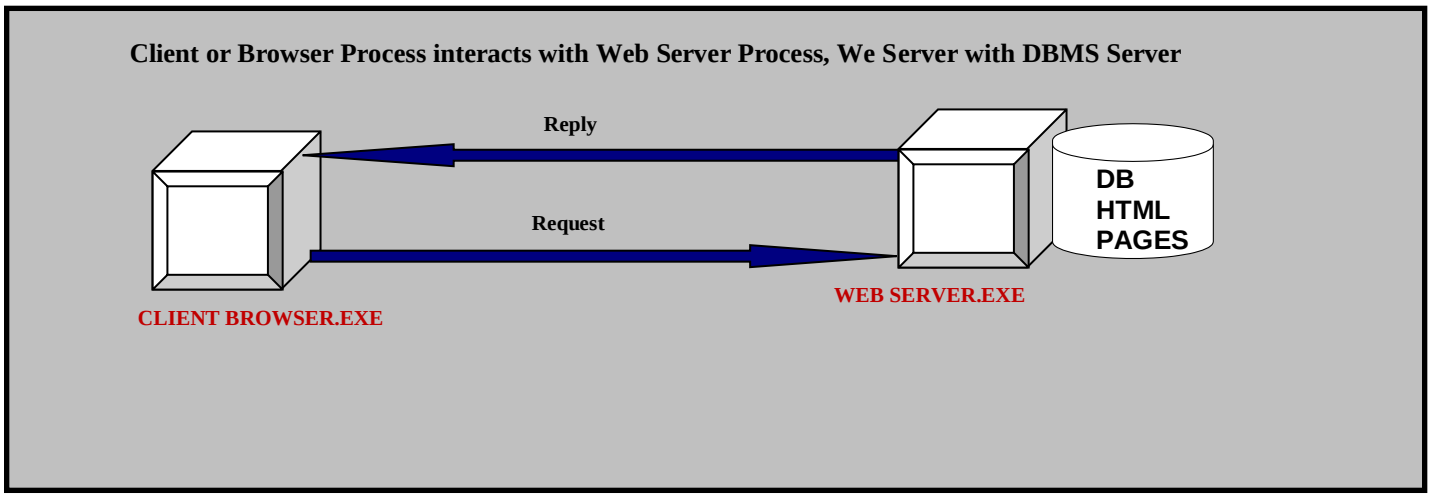


3-Tiered Client/Server - Server executable that connects to a CUSTOMER DBMS

Basic Web-based Client/Server Example of Scenario #2

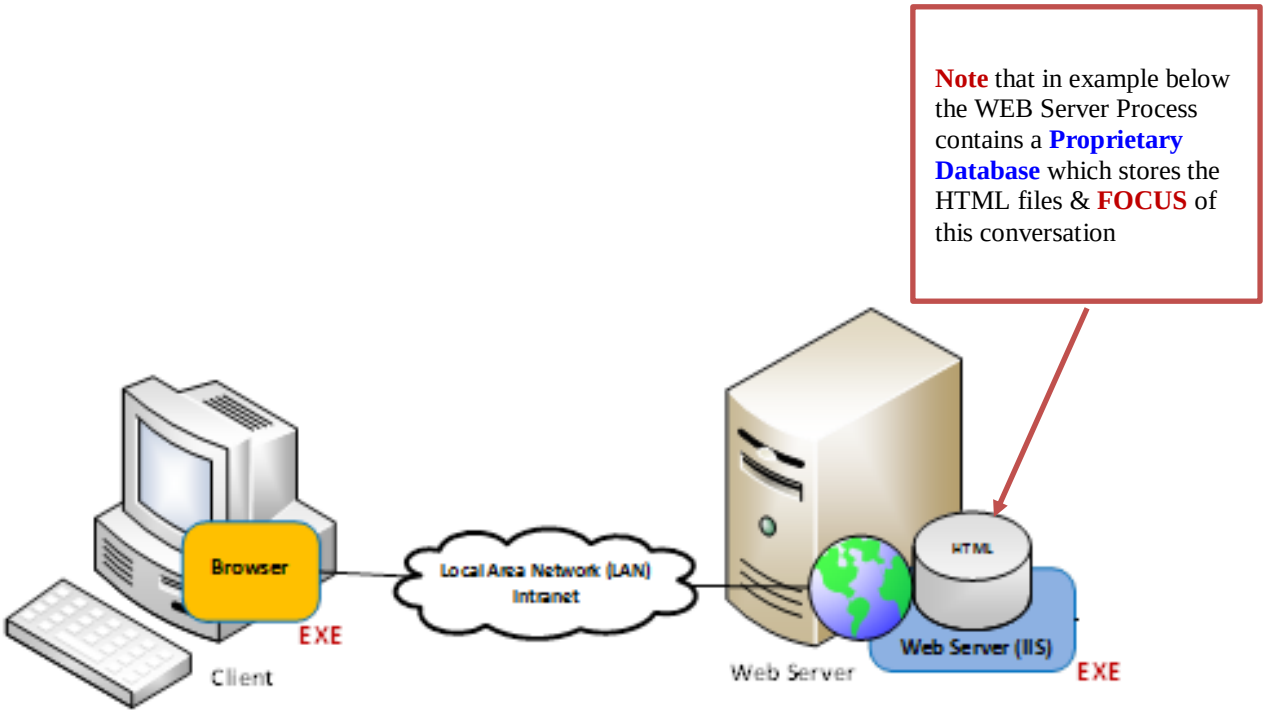
- Take the above **SCENARIO #2** and modify as follows and we have the Web-Based Client/Server Architecture:

  - Instead of a developer **CREATING a CLIENT EXECUTABLE** program, the client is **AREADY CREATED** and is an **HTML BROWSER** Application such as ***Microsoft Internet Explorer***, ***Chrome***, ***Safari***, etc.., which handle presenting the data to the user or ***User-Interface (UI) HTML document***.
  - The Sever is a WEB SERVER which you purchase or download & use it.
  - Finally, the proprietary database is created by the manufacturer of the Web Server and stores the HTML documents.
  - With these requirements, we end up with the following configuration:



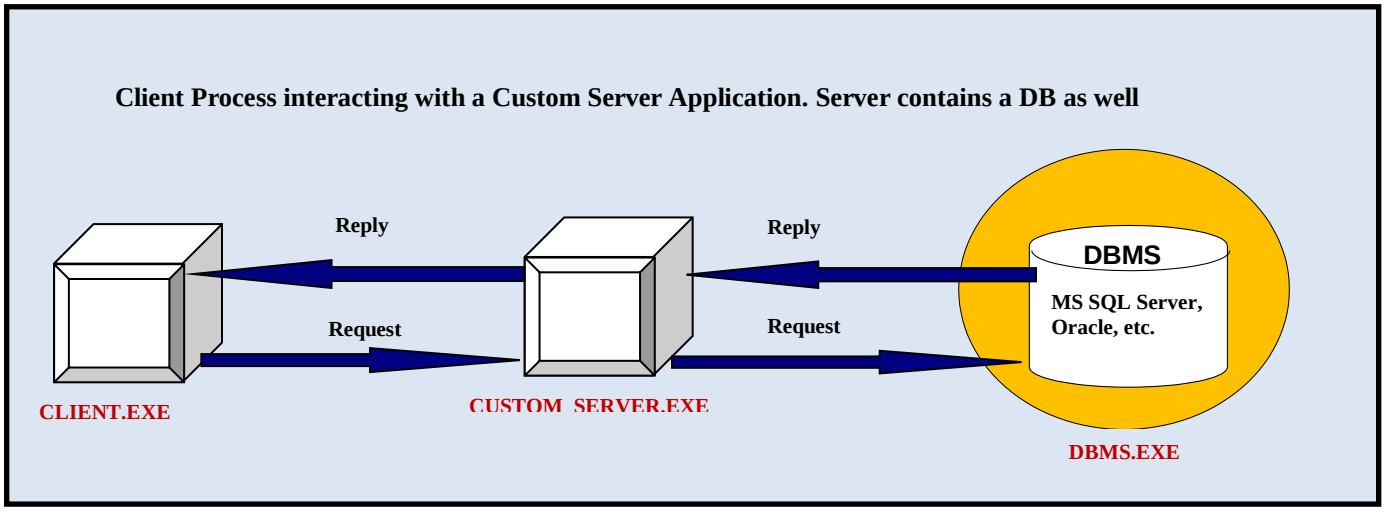
Basic Web Based Client/Server Configuration. Web Server Contains HTML File Database

- Example of a Three-Tier Physical WEB Client/Server Architecture Topology:



SCENARIO #3 – 3-Tiered-Client/Server: Server is a Custom Program but Includes a STANDARD Database Management System

- The remaining Client/Server implementations, the SERVER IS some kind of custom program or server executable but there is still some kind of DATABASE component which is a STANDARD DBMS like MS SQL, Oracle, etc.



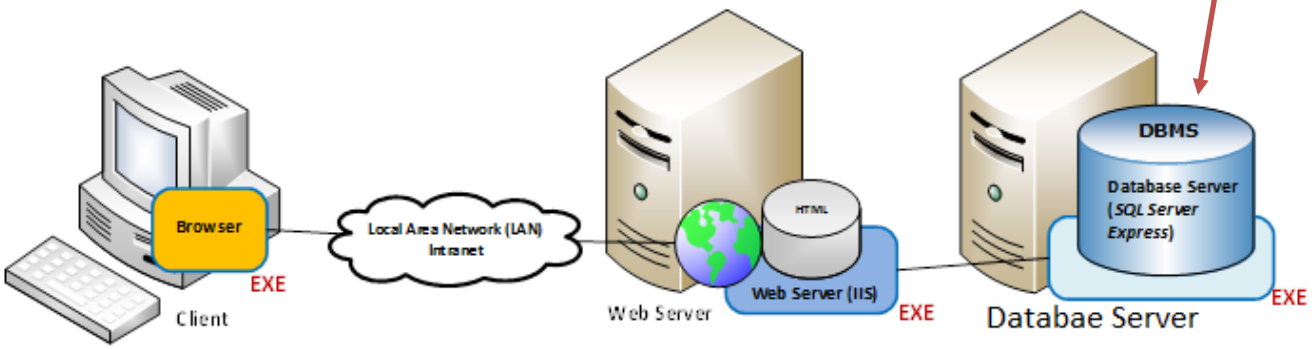
3-Tiered Client/Server - Server executable that connectes to a DBMS

- What we have is 3 EXECUTABLE involved: **CLIENT.EXE**, **CUSTOM_SERVER.EXE** & **DBMS.EXE**.

Three-Tiered Web-Based Database Server Application

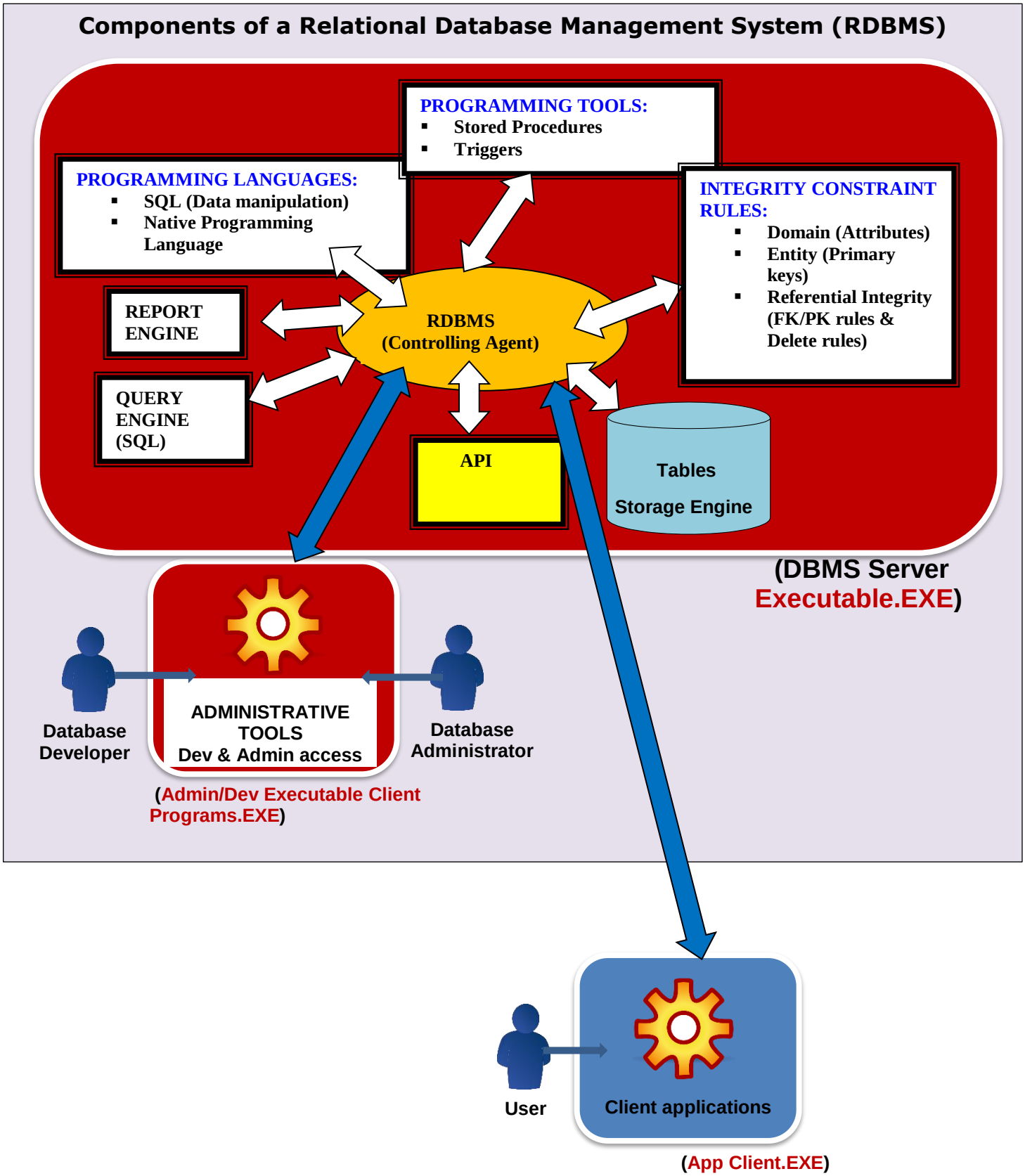
- Example of a Three-Tier Physical WEB/Database Server Client/Server Architecture Topology:

Note that in example below the Server Process is a **Database Management System (DBMS)** & **FOCUS** of this lecture



Relational Database Management System & Database Application (The Big Picture)

- Detailed illustration of a Database Application:



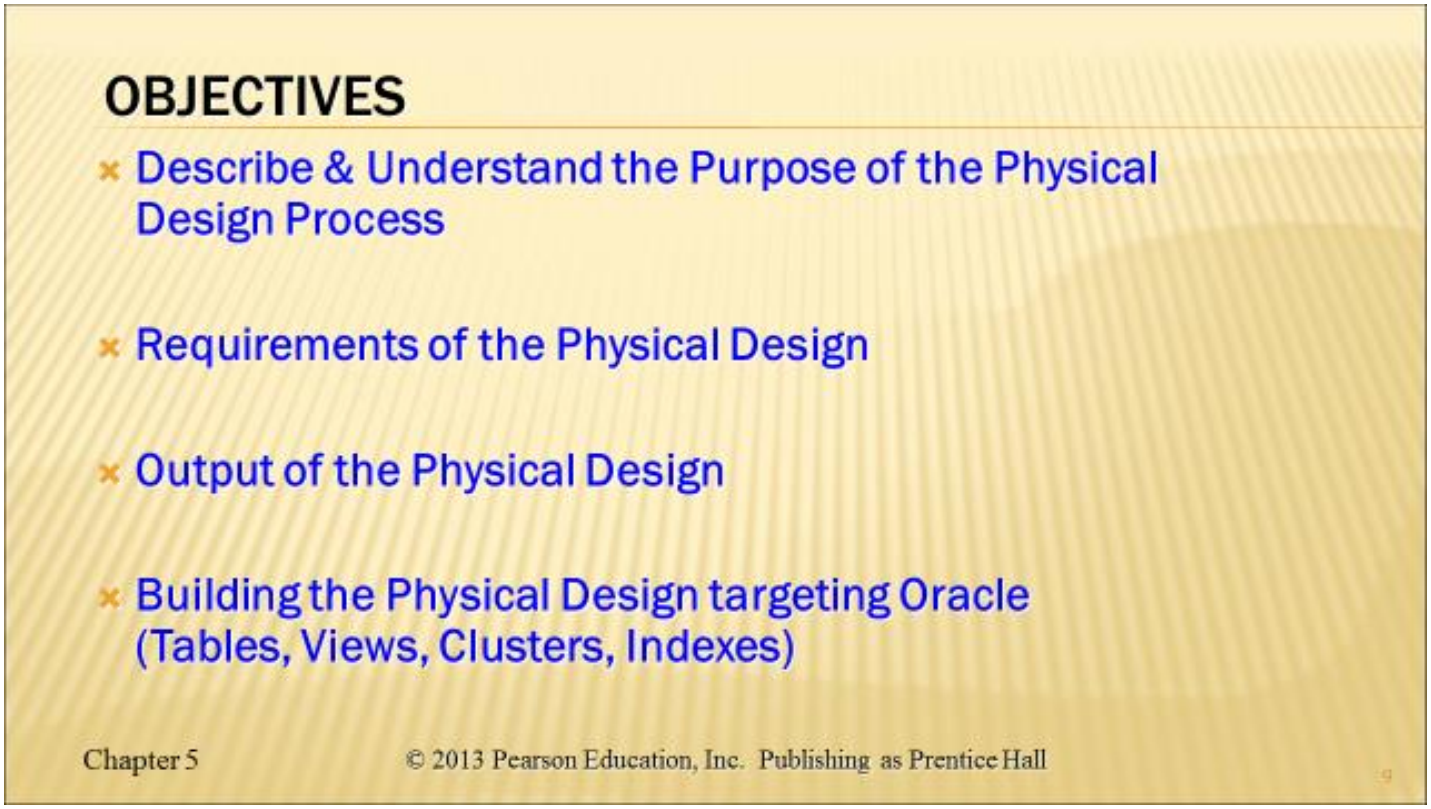
5.2 Physical Database Design – Introduction & Overview

5.2.1 Overview of the Physical Design Phase

- In this section we defined our objectives and goals.

Objectives and Goals

- Physical Design phase objectives, goals etc.:



CONCEPTUAL & LOGICAL DESIGN GOALS

- Conceptual & Logical Design Purpose
  - Describe, Organize & give meaning to the data via abstract model (EER Diagrams & Logical Diagrams)
- Limitations of Conceptual & Logical Design
  - Does not explain how data is processed & stored (Physical)

PHYSICAL DESIGN GOALS

- Translate Conceptual & Logical Models into TECHNICAL SPECIFICATION for storing and retrieving data
- Technical Specifications for processing & storing data that will ensure:
  - Adequate Efficiency & Performance
  - Integrity (correct, consistent, accessible)
  - Security
  - Recoverability (redundancy & backup)
  - Etc.

PHYSICAL DESIGN GOALS (CONT.)

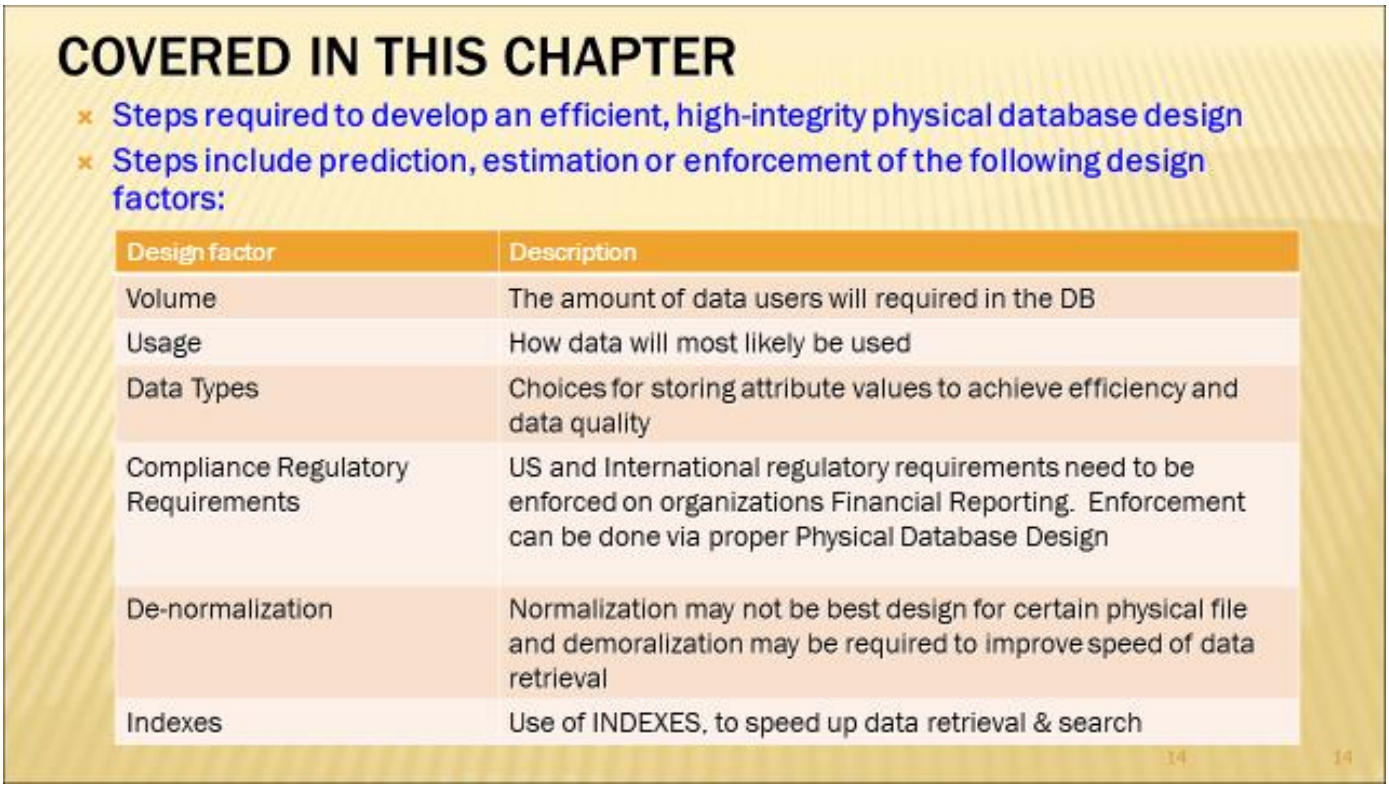
- Derive Technical Specifications for DB Programmers, DB Admins & other Information System Pros involved in the implementation of Information System
- Physical Design DOES NOT include implementing the database. This is done in the Implementation Phase
- Primary goal focus:
  - Efficiency & Performance
  - Integrity, Security & Recoverability important too, but first concern is efficiency

PHYSICAL DESIGN GOALS (CONT.)

- Very Important Step in development of Information System.
- Must be done with care and attention
- Decisions made during this stage have major impact on:
  - Response Times
  - Data quality
  - Security
  - User friendliness
  - Etc.
- Database Roles involved in the Physical Design Process:
  - All roles may be involved (DB Dev, DB Admin, Analyst, etc.
  - Database Administrations (DB Admins) Play a MAJOR ROLE in physical database design

Overview of Chapter 5 in the book

- What we are covering in this chapter:



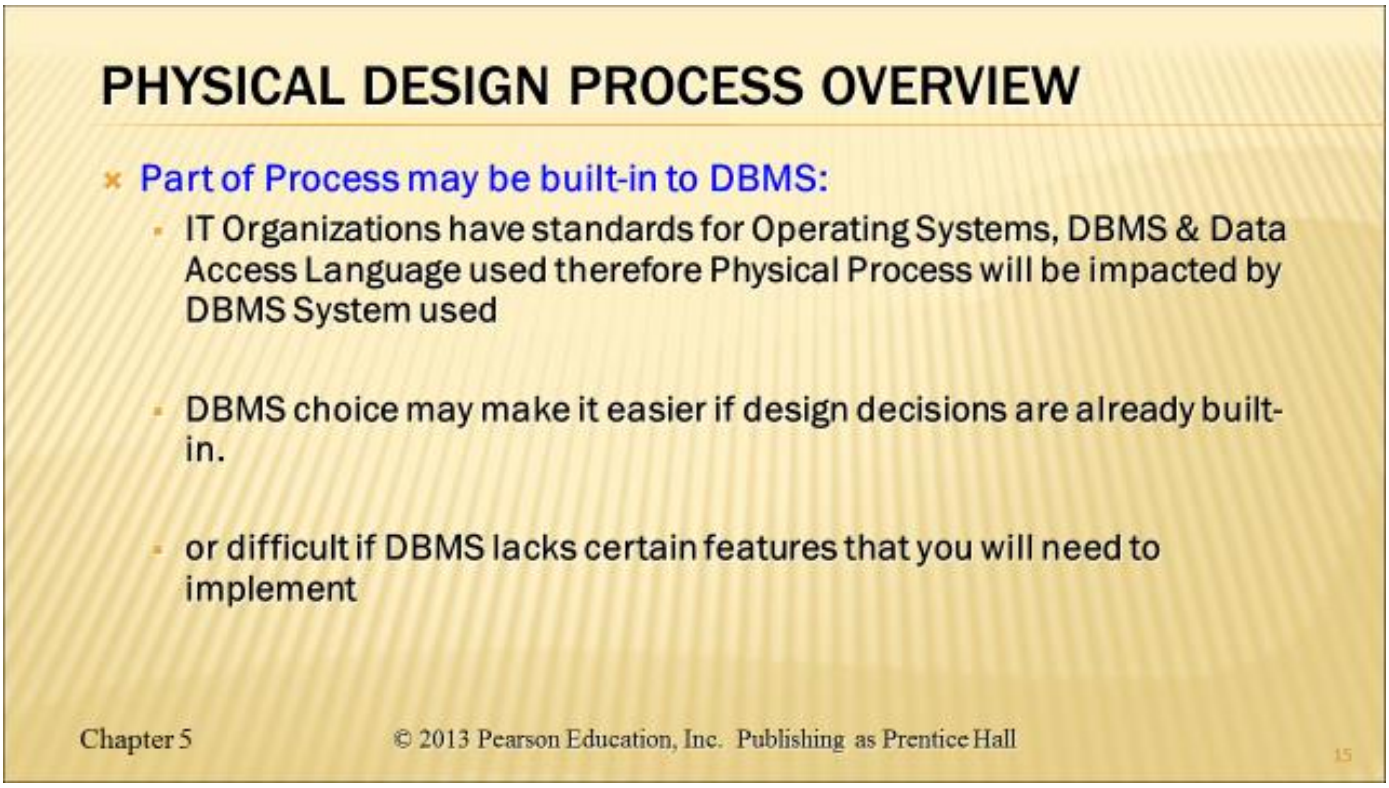
COVERED IN THIS CHAPTER

- Steps required to develop an efficient, high-integrity physical database design
- Steps include prediction, estimation or enforcement of the following design factors:

Design factor	Description
Volume	The amount of data users will required in the DB
Usage	How data will most likely be used
Data Types	Choices for storing attribute values to achieve efficiency and data quality
Compliance Regulatory Requirements	US and International regulatory requirements need to be enforced on organizations Financial Reporting. Enforcement can be done via proper Physical Database Design
De-normalization	Normalization may not be best design for certain physical file and demoralization may be required to improve speed of data retrieval
Indexes	Use of INDEXES, to speed up data retrieval & search

The Physical Design Process Overview – What's involved?

- Now we take a high-level look as what is involved in the Physical Design process:



PHYSICAL DESIGN PROCESS OVERVIEW

- Focus on design requirements NOT dependent on any DBMS:
  - Speed – Minimize time for user to get information:
    - Reduce the time required by user to interact with Information System.
    - How? Focus on how to make *Processing* of Physical Files & databases efficient!!!!
  - Storage Space – reduction of space used by data:
    - Important as well but focus is more on Speed & Efficiency
- Primary goal focus:
  - Efficiency & Performance based on Physical Files & Databases

The Physical Design Process Overview – Designing the Physical Files & Databases for Efficiency

- Within the Physical Design process one key design focus for efficiency & performance is design of the Physical Files & Databases:

PHYSICAL FILES & DATABASES DESIGN PROCESS

- Design of Physical Files & Databases requires the following INFORMATION from Conceptual and Logical Design Phases as INPUT. This includes:

Requirements:	Description
Normalized relations (tables)	The normalized tables from EER/Logical Diagrams
Normalized relations (tables) instances estimates	Estimates of the range or number of rows in each table
Attribute Definition & Specifications	Define each attributes including physical specifications such as maximum possible lengths, etc.
Usage Description	Description of how data is used (entered, retrieved, deleted, updated)
Usage Frequency	Number of times or frequency of data use

Chapter 5 17

PHYSICAL FILES & DATABASES DESIGN PROCESS

- Design of Physical Files & Databases required INFORMATION as INPUT (cont.):

Requirements:
Expectations & Requirements for the following: ▪ Response time ▪ Data Security ▪ Backup ▪ Recovery ▪ Retention – Regulatory compliance ▪ Integrity
Description of Technology to be used: ▪ Which DBMS (MS SQL Server, Oracle, MySQL, etc.)

Chapter 5 18

PHYSICAL FILES & DATABASES DESIGN PROCESS

- Design of Physical Files & Databases requires the following DECISIONS based on the INPUT INFORMATION to achieve integrity and performance:

Decisions:	Description
Attribute Storage Format (Data Type)	▪ What data type to use for each attribute
Attribute Storage Format (Data Type) Specifications	▪ What parameter for each data type attribute to: – Maximize integrity – Minimize storage space
Attribute Grouping	▪ Attribute grouping in Logical design may not be best for storing *physical records*. ▪ DBMS can provide guidance on this decision
Attribute Storage (File Organization)	▪ How to arrange *physical record* structure stored in Secondary Memory (Hard Disks) so records can be rapidly store, retrieve & updated

PHYSICAL FILES & DATABASES DESIGN PROCESS

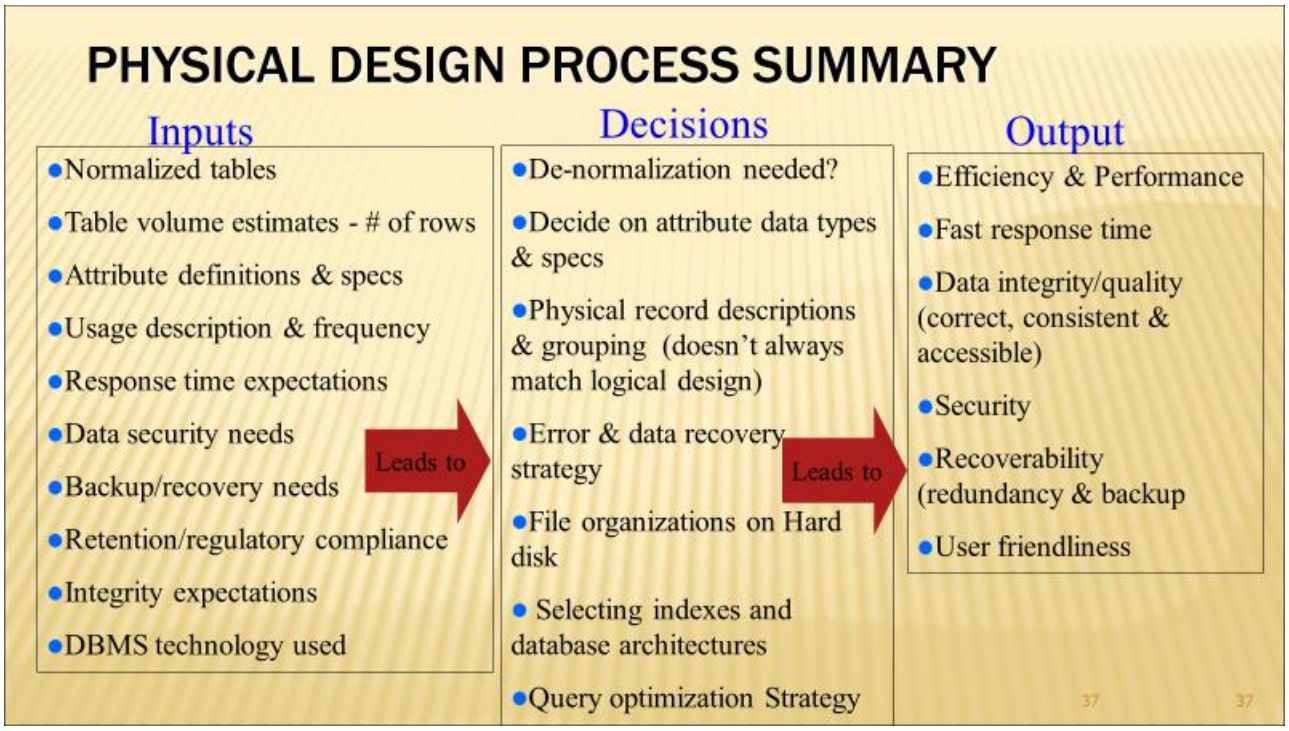
- Design of Physical Files & Databases requires the following DECISIONS based on the INPUT INFORMATION (Cont.):

Decisions:	Description
Error & Data Recovery	▪ How to protect and recover data if error occurs
Selecting Structure for storing & connecting Files	▪ What structures to use such as indexes, database architecture etc., for storing and connecting files for efficient data retrieval
Query Optimization Strategy	▪ Develop strategies for optimizing Queries for better performance by taking advantage of file structures (indexes) specified in previous decisions. ▪ Structures specified in previous decisions will only be efficient work Queries are fine-tuned to use them.

The Physical Design Process Overview – Final Summary

- Here is the entire picture:

  - **INPUT** - What is required from the previous conceptual & logical phases (EER diagram, Logical Diagram etc.)
  - **DESIGN DECISIONS** – What design decisions based on the list of technologies provided by a DBMS and Physical Model theory need to be made based these INPUTS
  - **OUTPUT** - And finally, to generate the OUTPUT or outcome that will result in achieving the required efficiency, data integrity, security, recoverability etc.
  - **So the idea is given these INPUTS and making the right DESIGN DECISIONS on the various technical specifications, we result is certain OUTPUTS that result is required efficiency, data integrity, security, recoverability etc.**



The Physical Design & Regulatory Compliance

- Database design and implementation is not just about the technology and how is implemented, but it also impacted by the business and regulatory compliance imposed by government and other regulatory agencies.
- Regulatory compliance is a very important goal in the design of a database management system as you will experience as you work in the world of IT.

PHYSICAL DESIGN & REGULATORY COMPLIANCE

- Sarbanes-Oxley (SOX) (cont.)
  - Companies must follow SOX:
    - Standards
    - Guidelines
    - Governance rules
    - Risk assessments
    - Data security control guidelines
- How SOX compliance is enforced in database:
  - SOX Compliance must be part of the database design and enforced by the DBMS. For example:
    - Attribute Level data integrity controls
    - Triggers & stored procedures
    - Audit trails (who uses what) & Activity logs
    - Etc.

REGULATORY COMPLIANCE AGENCIES

- Sarbanes-Oxley Act (SOX)
  - Most recent regulations to protect investors by improving accuracy and reliability
- Committee of Sponsoring Organizations (COSO) of the Treadway Commission
  - Improve quality of financial reporting through business ethics, effective internal controls, and corporate governance

25

REGULATORY COMPLIANCE AGENCIES

- Control Objectives for Information and Related Technology (COBIT)
  - Open standard IT control framework built in part upon the COSO framework
- IT Infrastructure Library (ITIL)
  - Focuses on IT services and is often used to complement the COBIT framework
- US Department of Health and Human Services (HIPAA)
  - Standards and regulations on healthcare providers on Personal Health Information (PHI)
  - Focus is on privacy of PHI

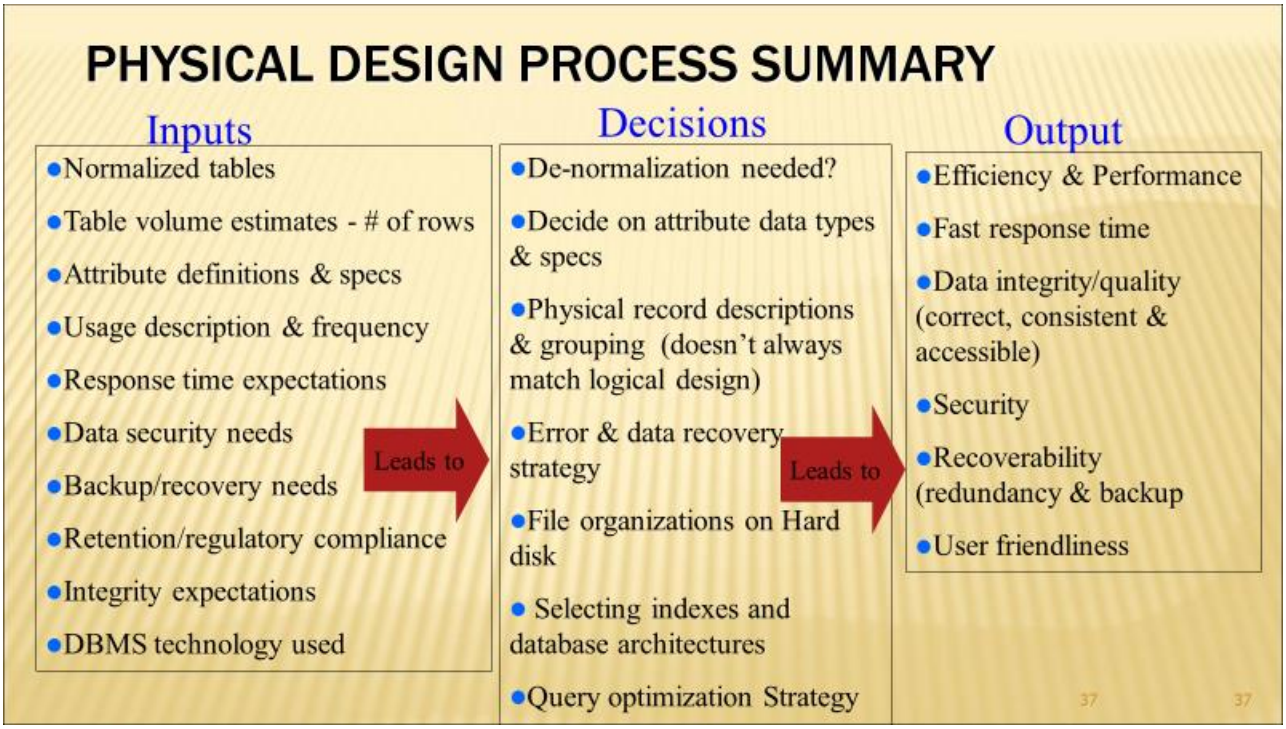
26

5.3 Physical Database Design – Physical Design Process (How is done)

5.3.1 Physical Design Process Summary

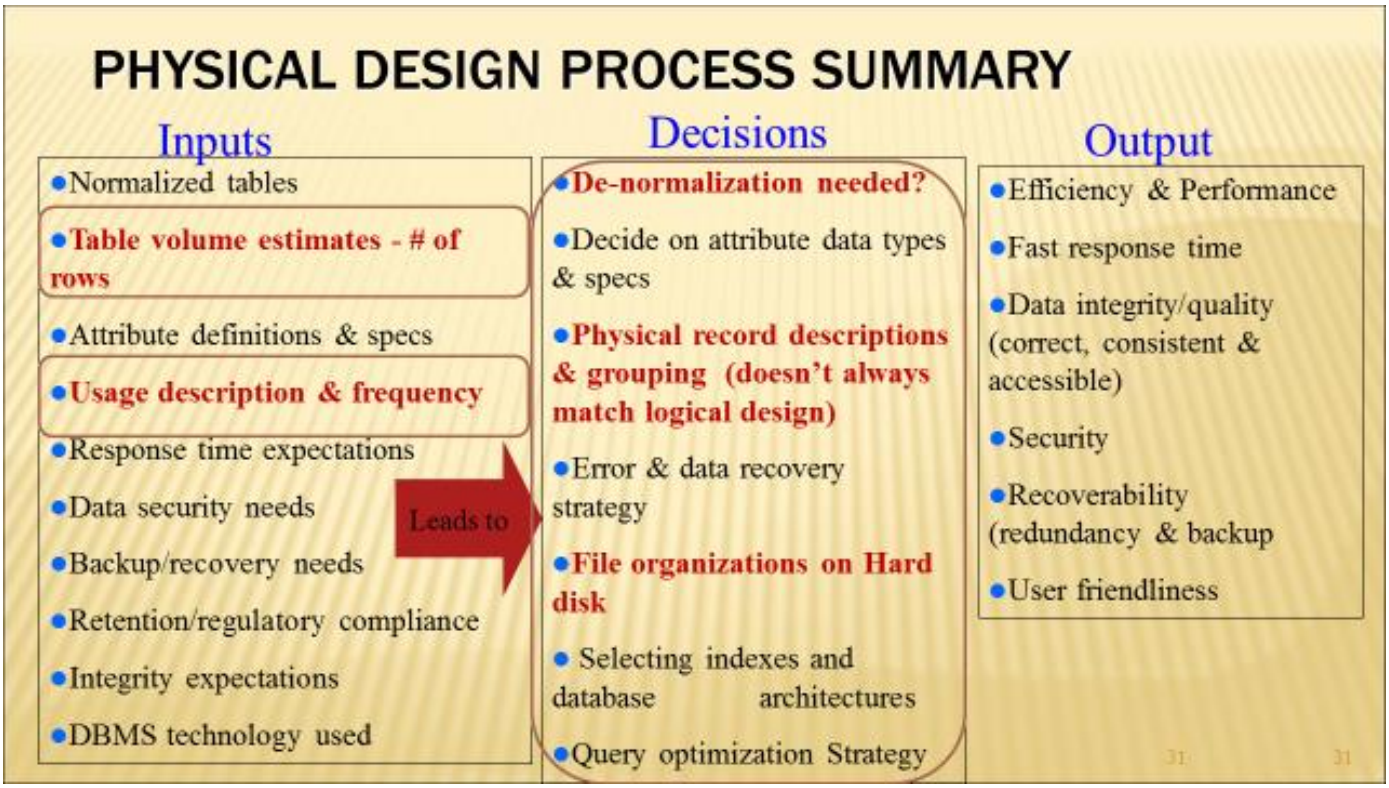
- In this section we go into details on how to implement the Physical Design Process.
- Here, we see how is done from a high-level. Sone activities we won't go into great details since in the CST3604 course we will dive deep into physical design theory.
- In the previous section we looked at an overview of what the process is about.
- Illustration below is once again the summary of the process:

  - What is required from the previous conceptual & logical phases (EER diagram etc.)
  - What decisions need to be made
  - And finally, the outcome or what we want our information system to have, which is efficiency, data integrity, security, etc.



5.3.2 Physical Design Process – Volume and Usage Analysis

- In this section we go into details on the design given the **volume and usage INPUTS** in order to achieve efficiency of our physical design process & the design **DECISIONS** that are required:



DATA VOLUME & USAGE ANALYSIS

- Data volume and frequency-of-use statistics are important inputs to the physical database design process
  - Must maintain an understanding of size and usage patterns of the database
  - When is this done?
    - Throughout the life cycle of the information system
    - Must be part of database development life-cycle or methodology

28

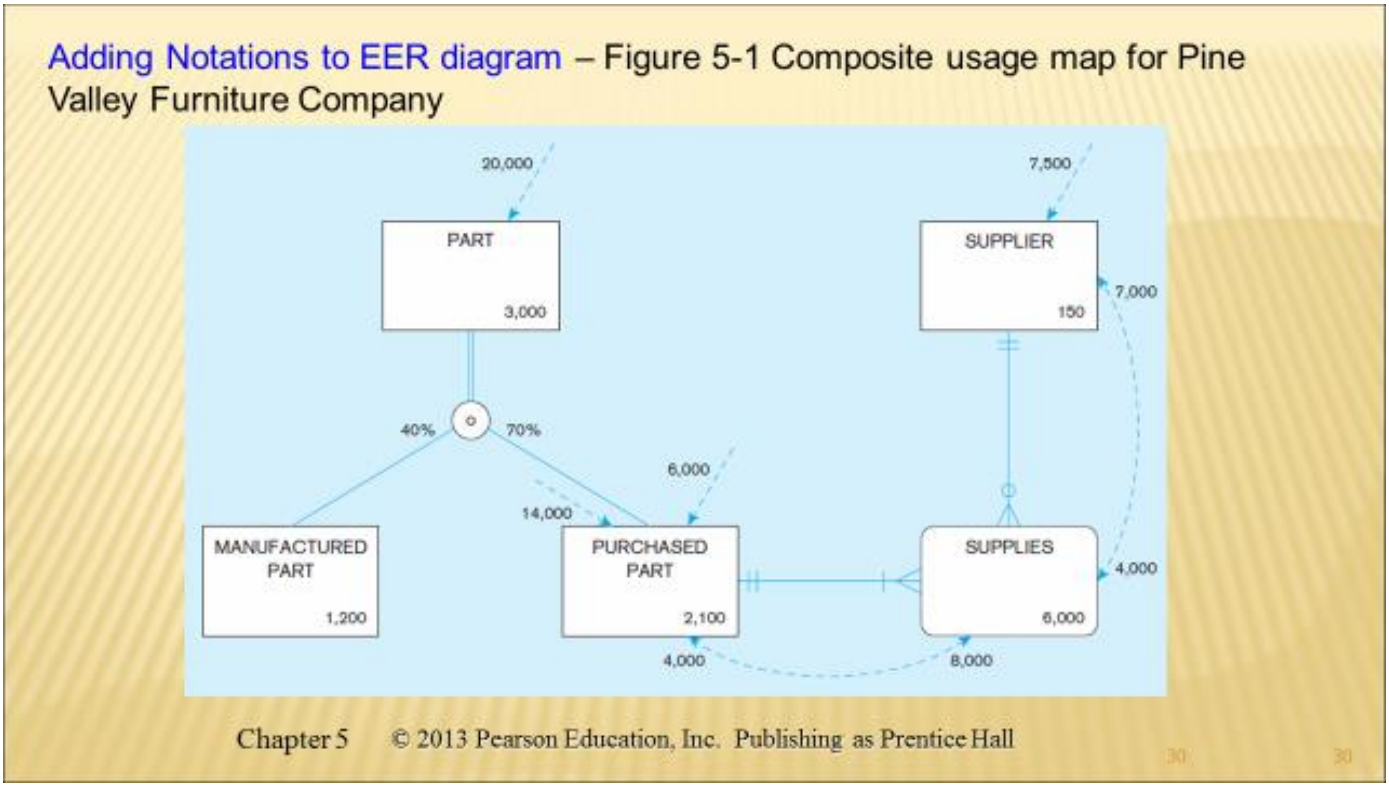
DATA VOLUME & USAGE ANALYSIS APPROACH

- There are several approach to track volume & usage analysis
- We will show one easy approach. Future lectures in CST3604 will show others
- Approach:
  - Usage Map – Adding Notations to EER diagram (normalized tables)
  - Idea is that you add statistic data of volume and usage onto the EER diagram

29

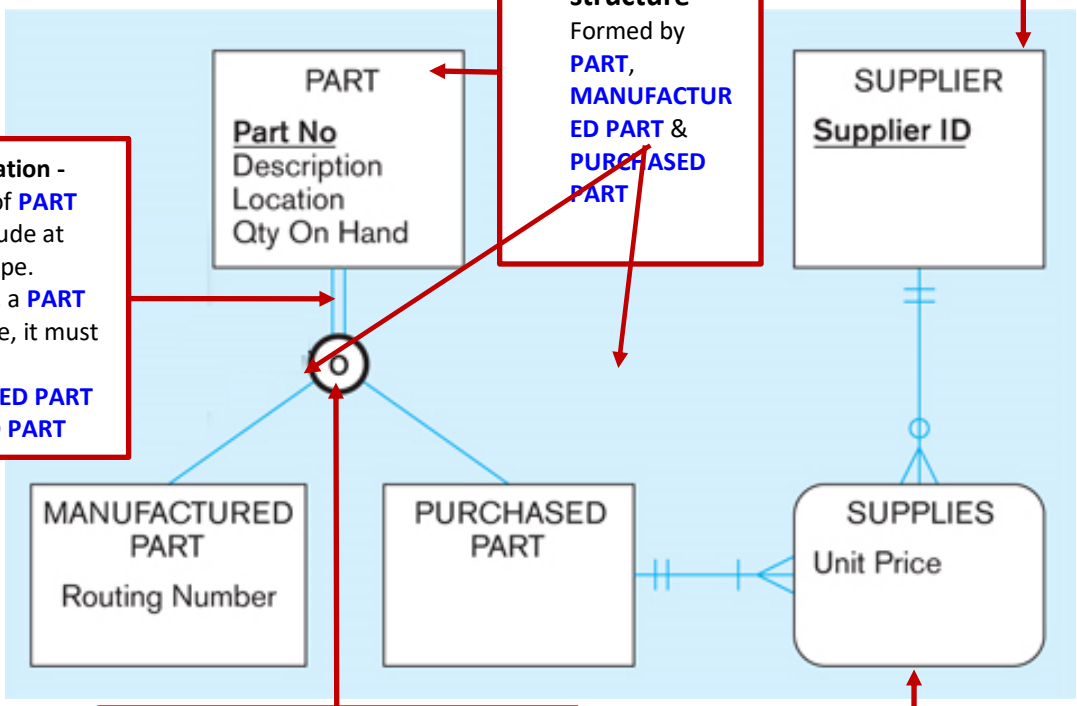
Volume & Usage Map Analysis

- The idea is to add volume and usage statistics to the normalized EER diagram from logical design
  Keep in mind that these are estimates you need to make. See example from your book below:



Pine Valley Furniture EER Diagram – Review of characteristics of this EER Diagram

- Before we dive deep into the volume usage analysis, let's review some of the characteristics of this EER diagram to review the basic concepts:



- ENTITY **SUPPLIER** – Suppliers that supply the **PURCHASED PART**

- **Supertype/ subtype structure** – Formed by **PART**, **MANUFACTURED PART** & **PURCHASED PART**

- **Total Specialization -** Each instance of **PART** **MUST** also include at least one subtype.
- In other words, a **PART** cannot be alone, it must be either a **MANUFACTURED PART** or **PURCHASED PART**

- **Overlap Rule -** A **PART** can be BOTH a **MANUFACTURED PART** and **PURCHASED PART** at the same time
- When you add a record or instance of **PART**, you can also add an instance of **MANUFACTURED PART** and **PURCHASED PART** at the same time

- ASSOCIATIVE ENTITY **SUPPLIES** – associates which **SUPPLIER** **SUPPLIES** a **PURCHASED PART**

Volume & Usage (access frequency) Map Analysis – Data Volume Analysis

- We focus on the data volume:

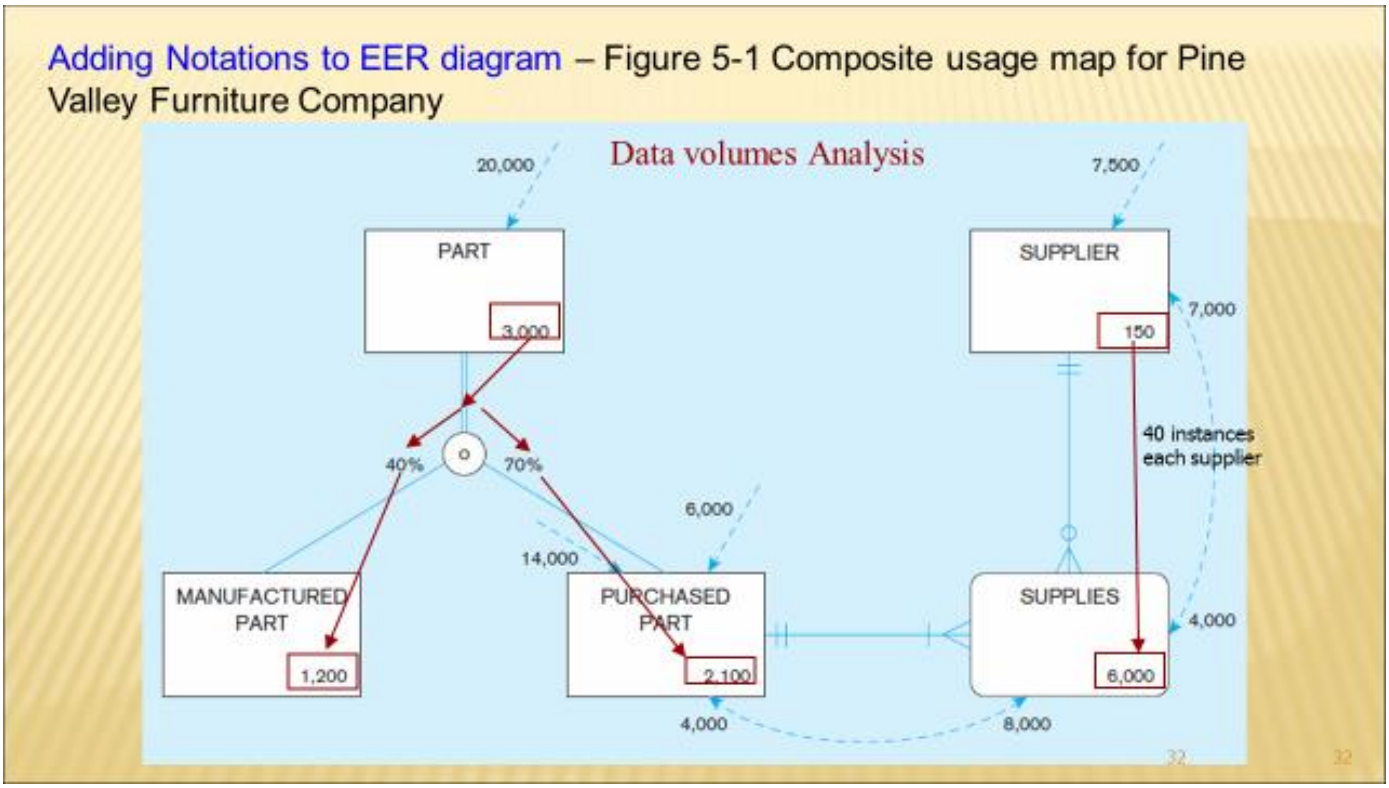
Volume Statistics – Numbers inside Entities & Percentages in supertype subtype structure:

PART, MANUFACTURED PART & PURCHASED PART Volume Analysis:

- Total PART(s) in database = **3000 PART(s)**
- MANUFACTURED PART(s) is 40% of all 3000 PART(s) = 0.4(3000) = **1200 MANUFACTURED PART(s)**
- PURCHASED PART(s) are 70% of all 3000 PART(s) since due to OVERLAP RULE, some PART(s) are of both = 0.7(3000) = **2100 PURCHASED PART(s)**

SUPPLIER, SUPPLIES & PURCHASED PART Volume Analysis:

- An Analysis estimated that there are typically = **150 SUPPLIER**
- Each SUPPLIER on average SUPPLIES 40 instances of PURCHASED PART(s). Total SUPPLIES or instances/records in SUPPLIES table = 40*150 = **6000 SUPPLIES**



Usage Statistics (Access Frequency) – Numbers with dashed arrows on diagram:

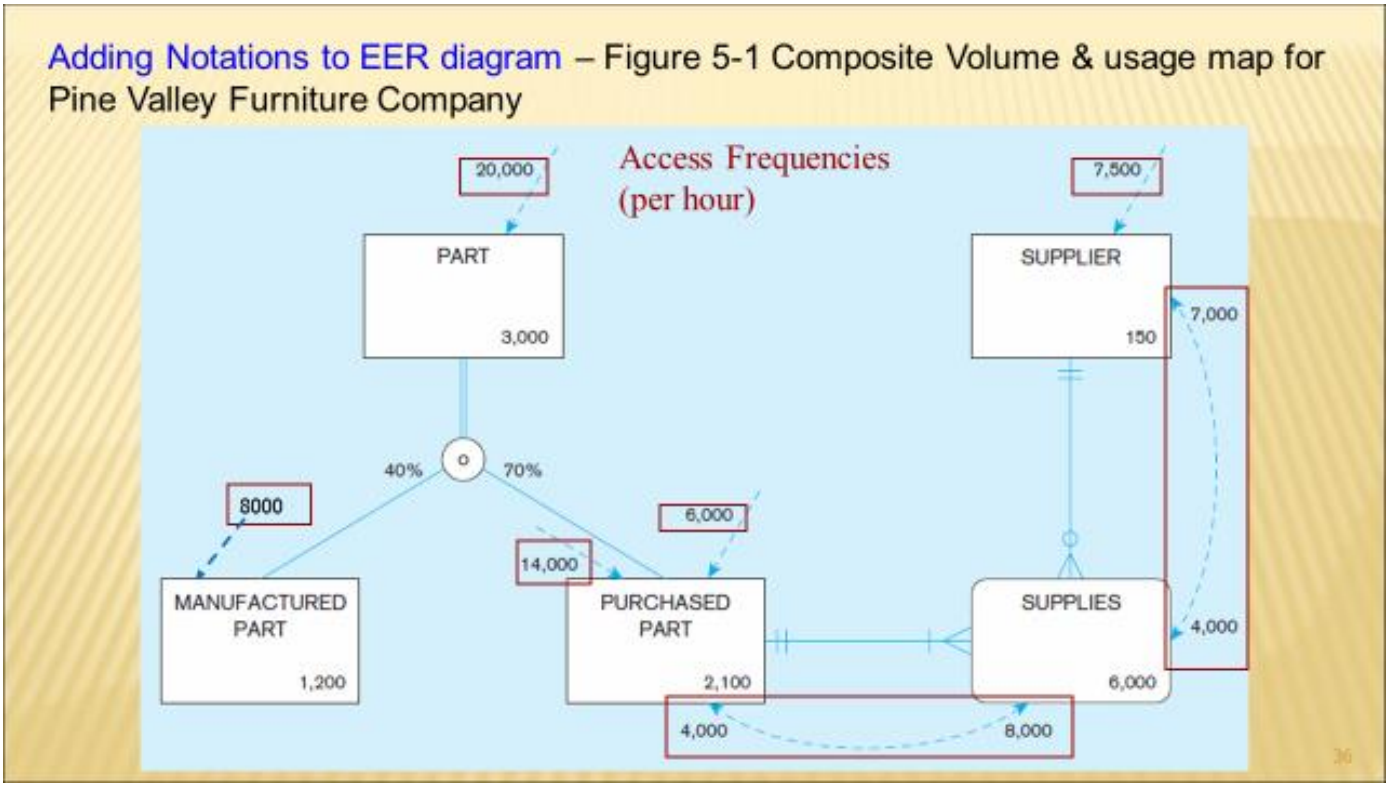
- We focus on the access frequency:

PART, MANUFACTURED PART & PURCHASED PART Usage Analysis:

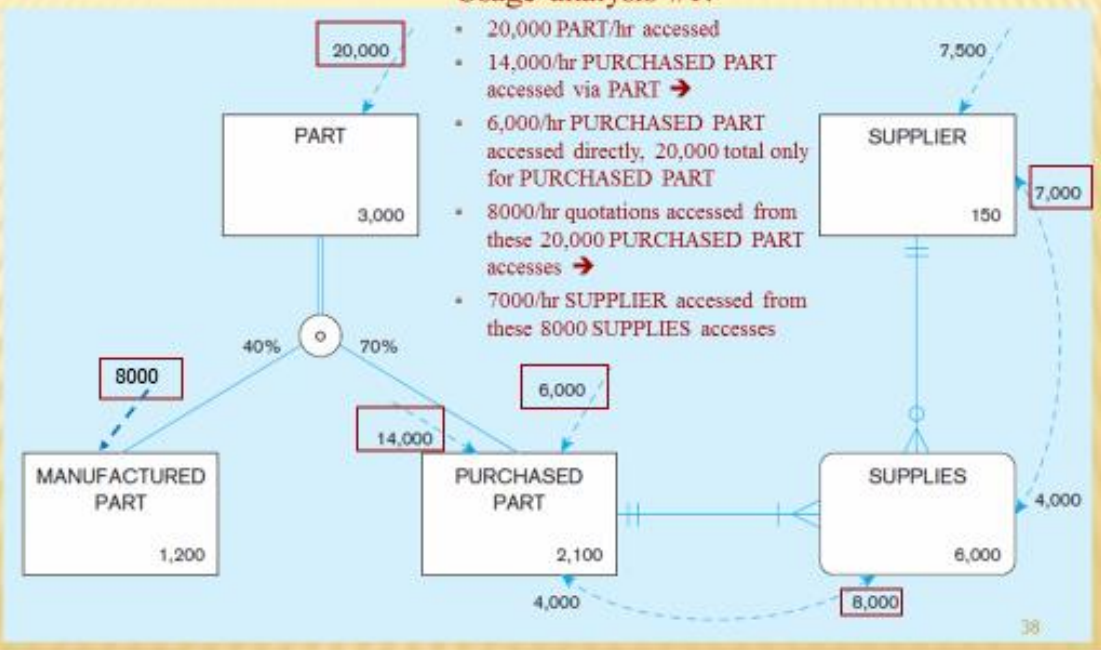
- Estimates show that access per hour of PART data for all applications that use the database = **20,000/hour**
- PURCHASED PART(s) are 70% of all 3000 PART(s), that means 70% of the total PART accesses (20,000) are for PURCHASED PART = .7*20,000 = **14,000 access/hour**
- Analysis also shows that there are 6,000 direct accesses to PURCHASED PART = **6,000 direct accesses**
- PURCHASED PART has a total of 14,000 + 6,000 = **20,000** accesses only to **PURCHASED PART**
- MANUFACTURED PART are 40% of all 3000 PART(s), that means 40% of the total PART accesses (20,000) are for MANUFACTURED PART = .4*20,000 = **8,000 access/hour**

SUPPLIER, SUPPLIES & PURCHASED PART Usage Analysis:

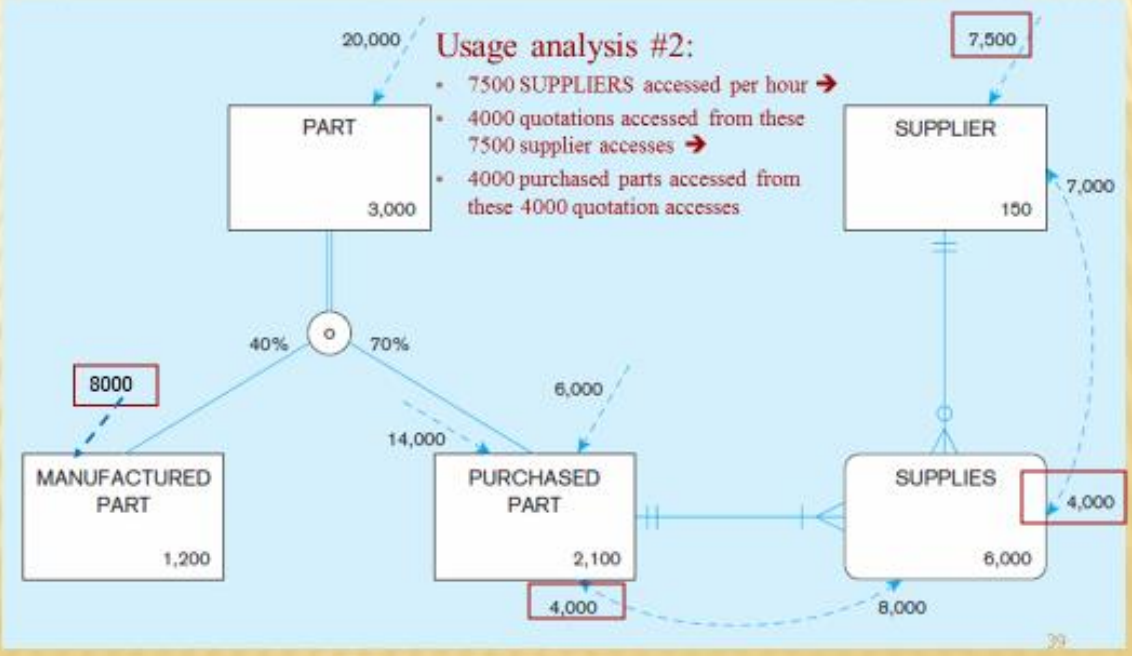
- Analysis shows that SUPPLIER is accessed **7,500/hour**
- SUPPLIES is accessed by **SUPPLIER 4,000/hour,** but each access to **SUPPLIES** means also **4,000/hour** access to **PURCHASED PART**
- Of these 20,000/hour access to **PURCHASED PART**, **8,000/hour** access **SUPPLIES** data and in addition **7,000** to **SUPPLIER** data



Adding Notations to EER diagram – Figure 5-1 Composite Volume & usage map for Pine Valley Furniture Company



Adding Notations to EER diagram – Figure 5-1 Composite Volume & usage map for Pine Valley Furniture Company



Decisions to be made based on results from Volume & Usage Analysis

- Now we analyze the decisions made based on the volume & usage inputs.

DECISIONS TO BE MADE FROM RESULTS FROM VOLUME & USAGE MAP ANALYSIS

- From analysis of the diagram & statistics decisions can be made such as:

Entities & Volume/Usage Results	Decision or Actions
PART – Volume is 3,000 parts	▪ if attributes such as *description* contain a lot of data or too long than correct data type & properties choice for *description* needs to be considered for efficiency
SUPPLIER, SUPPLIES & PURCHASED PART - Access frequency is 4,000/hr	▪ These 3 entities are accessed together every time ▪ De-nomalization opportunity? Combine SUPPLIER & SUPPLIES into one table or file? A possibility to improve performance

40

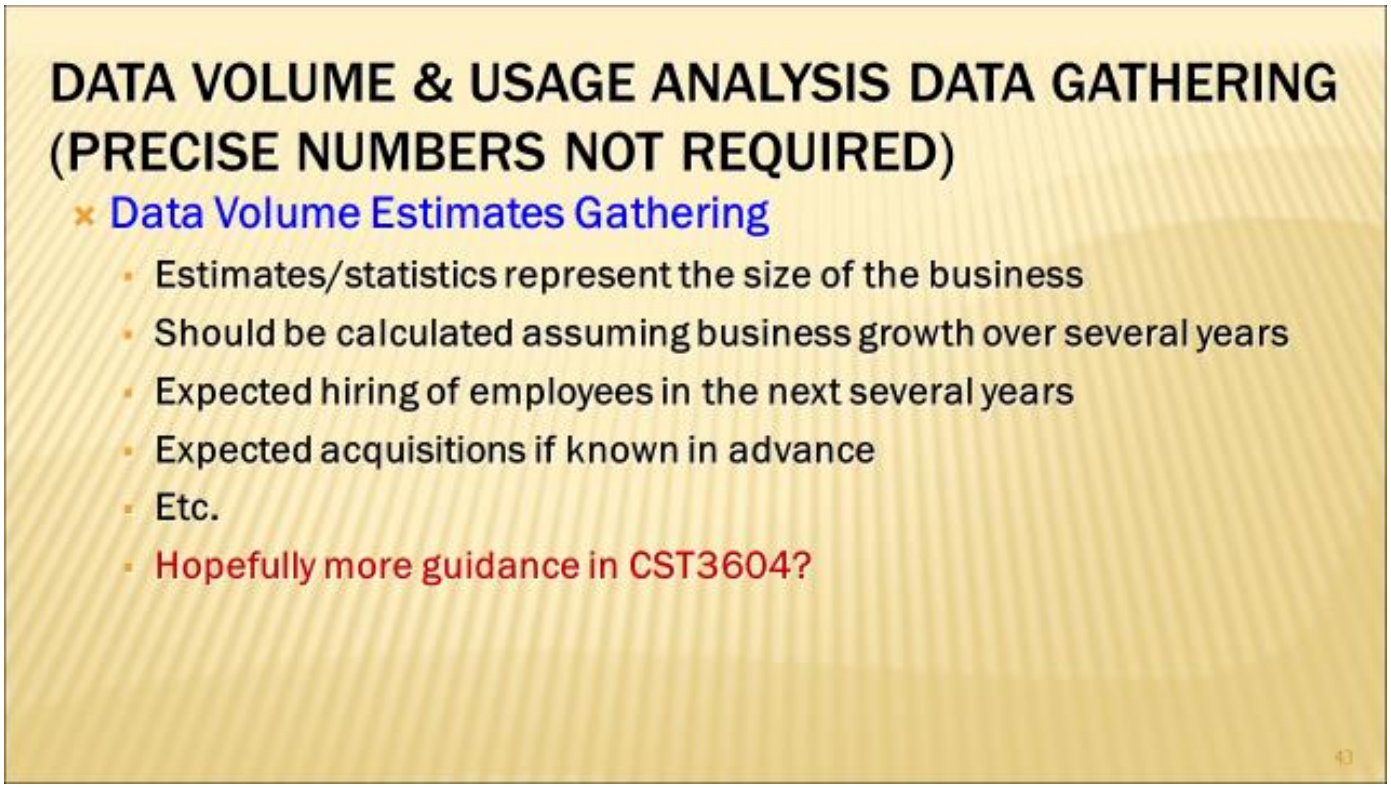
DECISIONS TO BE MADE FROM RESULTS FROM VOLUME & USAGE MAP ANALYSIS

- Decisions (cont.):

Entities & Volume/Usage Results	Decision or Actions
MANUFACTURED PART – Volume is 1,200 parts or 40% of PART PURCHASED PART– Volume is 2,100 parts or 70% of PART. Overlap of MANUFACTURED PART in PURCHASED PART – Volume is 10%	▪ There is redundancy here, but may be acceptable due to only 10% overlap so it may make sense to keep two separate table ▪ But that can change if volume changes and de-normalization required
MANUFACTURED PART – Access frequency is 8,000/hr PURCHASED PART– Access frequency is 20,000/hr (14,000 + 6,000)	▪ Should tables be organized differently due to the significant access volumes? Possibility.

How do we gather this data for Volume & Usage Analysis?

- How do I gather these numbers?

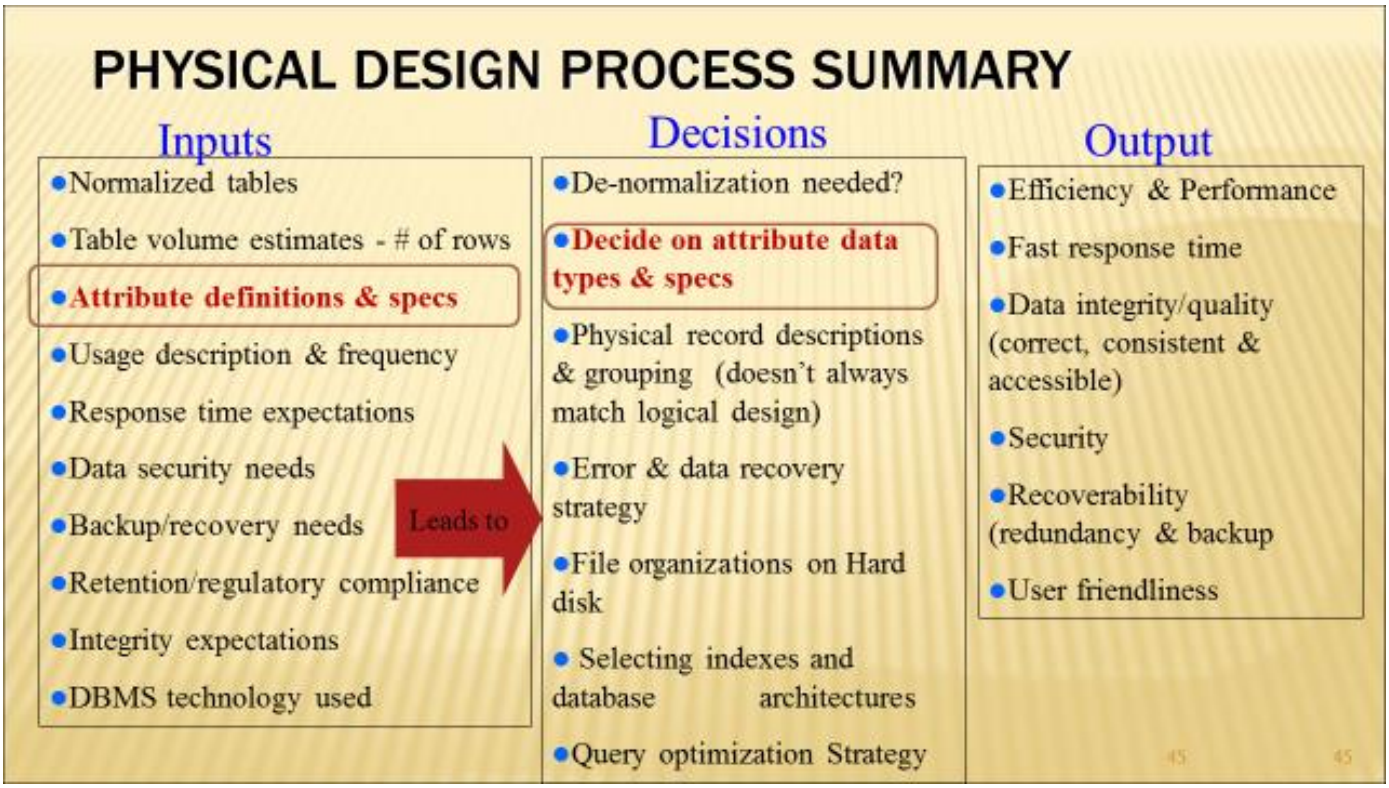


DATA VOLUME & USAGE ANALYSIS DATA GATHERING (PRECISE NUMBERS NOT REQUIRED)

- Usage or Access Frequency Estimates Gathering
  - Challenging to obtained since less certain during analysis phase
  - Estimated from timing of business events, peak times of day, week, months etc.
  - Analysis of how the data is accessed – Primary key, other attributes etc. This requires review of queries etc.
  - Transaction volumes
  - Number of concurrent users (accessing at the same time)
  - Reporting and querying activities
  - Hopefully more guidance in CST3604?

5.3.2 Physical Design Process – Designing Attributes & Data Type Selection

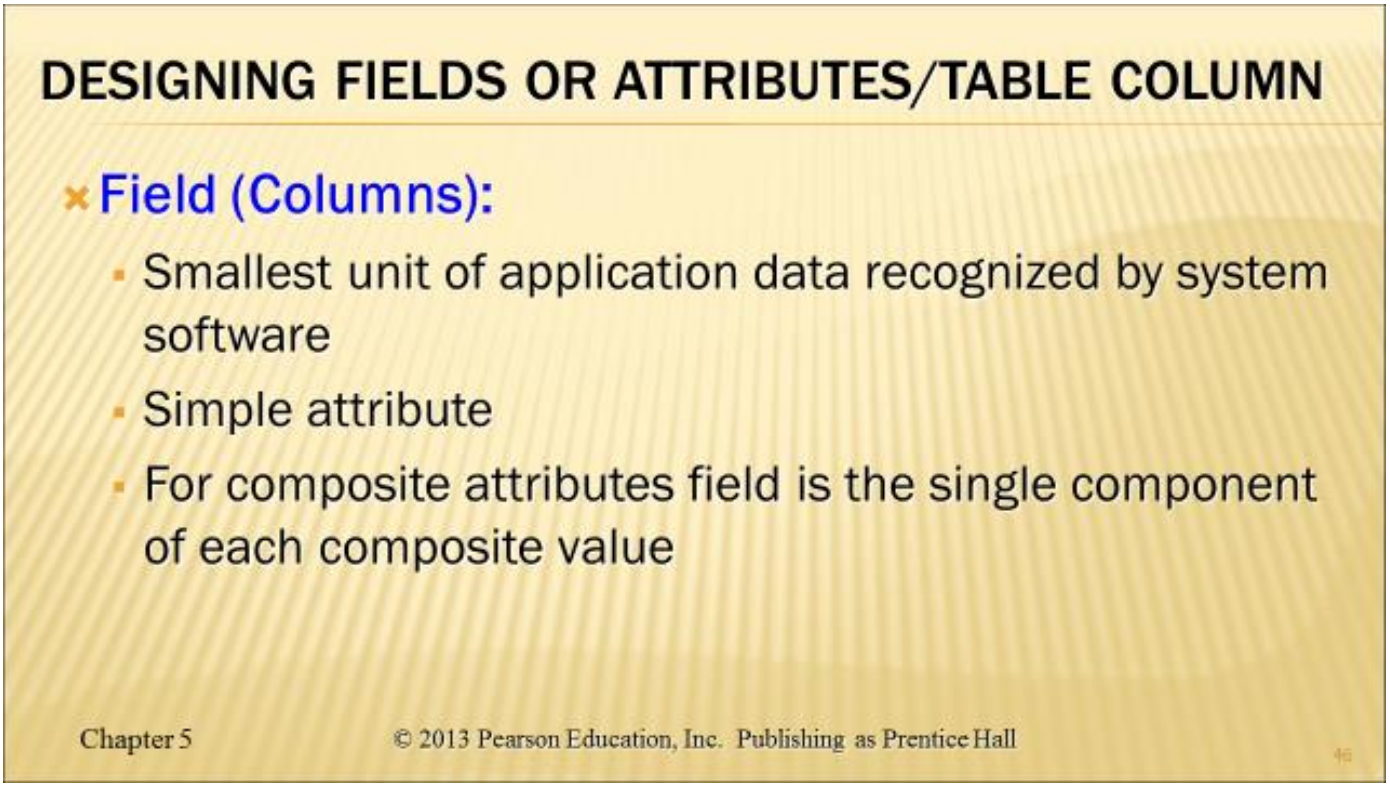
- In this section we go into details on how to design the appropriate data types for the attributes.
- First we review our physical design process and the **INPUT** value we are now targeting which is **Attribute definitions & specifications** and the DECISIONS that need to be made or **Decide on attribute data type & specifications**



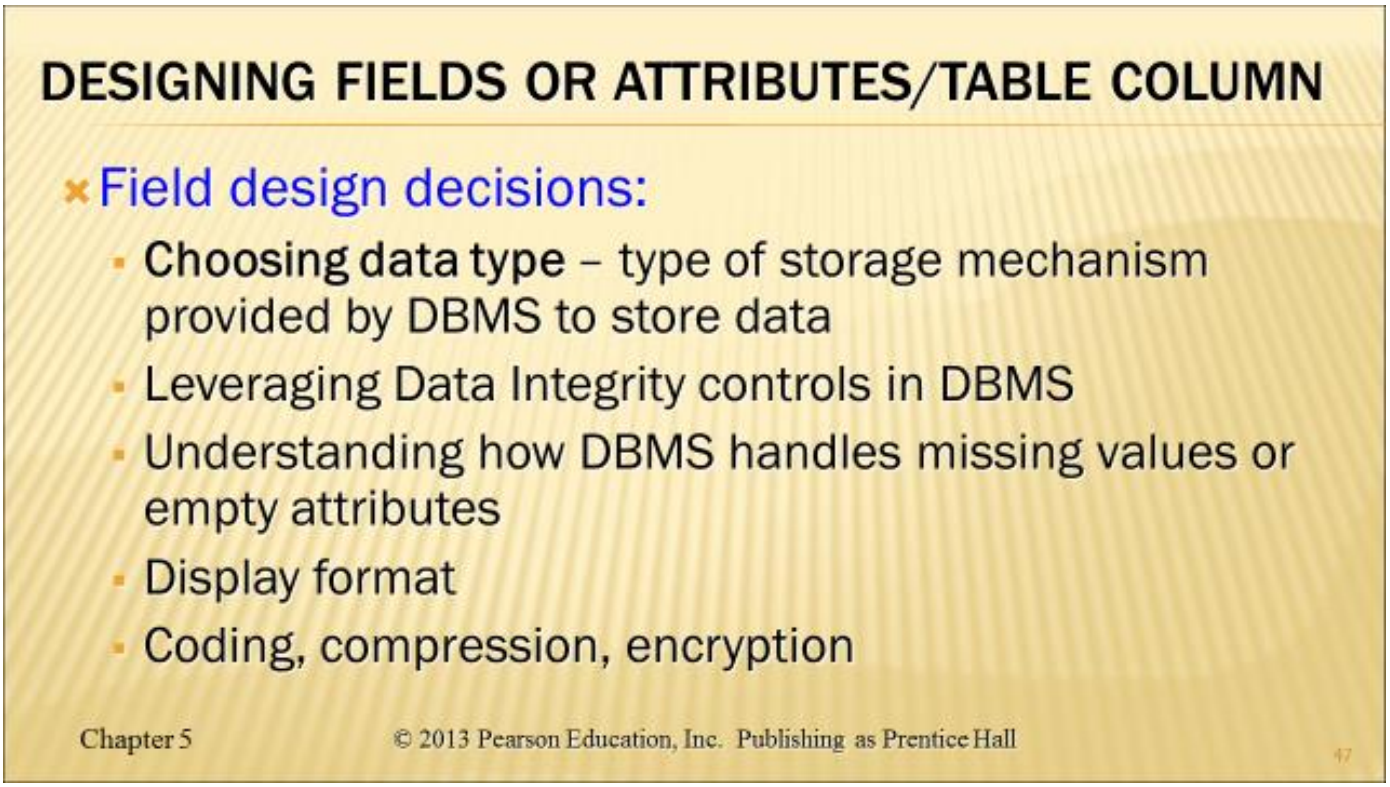
Selecting the appropriate data types & properties for table columns (field)

Definition & Design Decisions

- Definition and overview.



-



Choosing Data Types – Choosing Correct Data Type

- Guidance for selecting the correct data type.

Tables – Column Data Types

- **Column Data Types:**
  - Storage mechanism provided by DBMS to store the values of the attributes or columns
  - Every DBMS has a list of available data types to choose from the DBMS you choose will determine choices available
  - Data Types:
    1. Define type of values or type of data can be entered to a column
    2. Define the Properties/characteristics (Name, meaning, data type, size and range, etc.) of the data
- **Data Types Properties/Size, Characteristics known as METADATA:**

INTEGRITY CONSTRAINTS – CHOOSING CORRECT DATA TYPES

- Integrity Constraints or rules on attributes (chapter 4) can help in selection
- Domain Constraints
  - Allowable values you can assigned to an attribute
  - Domain definition – Name, meaning, data type, size and range
- Entity Integrity
  - No primary key attribute may be null.
  - All primary key fields MUST have data.
- Action Assertions (enforcement limiting certain actions)
  - Business rules
  - Referential integrity

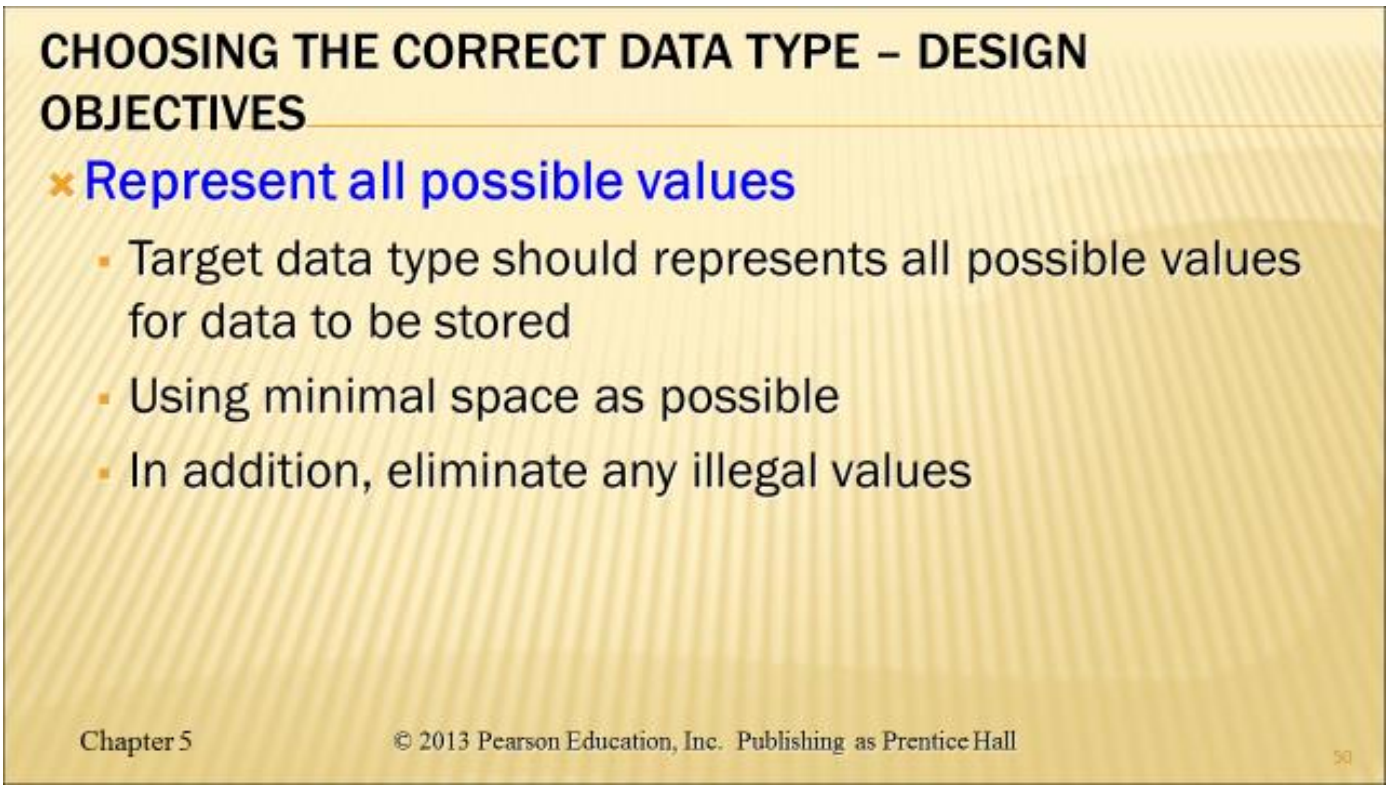
- Example table below lists some of the data types available in Oracle 11g DBMS:

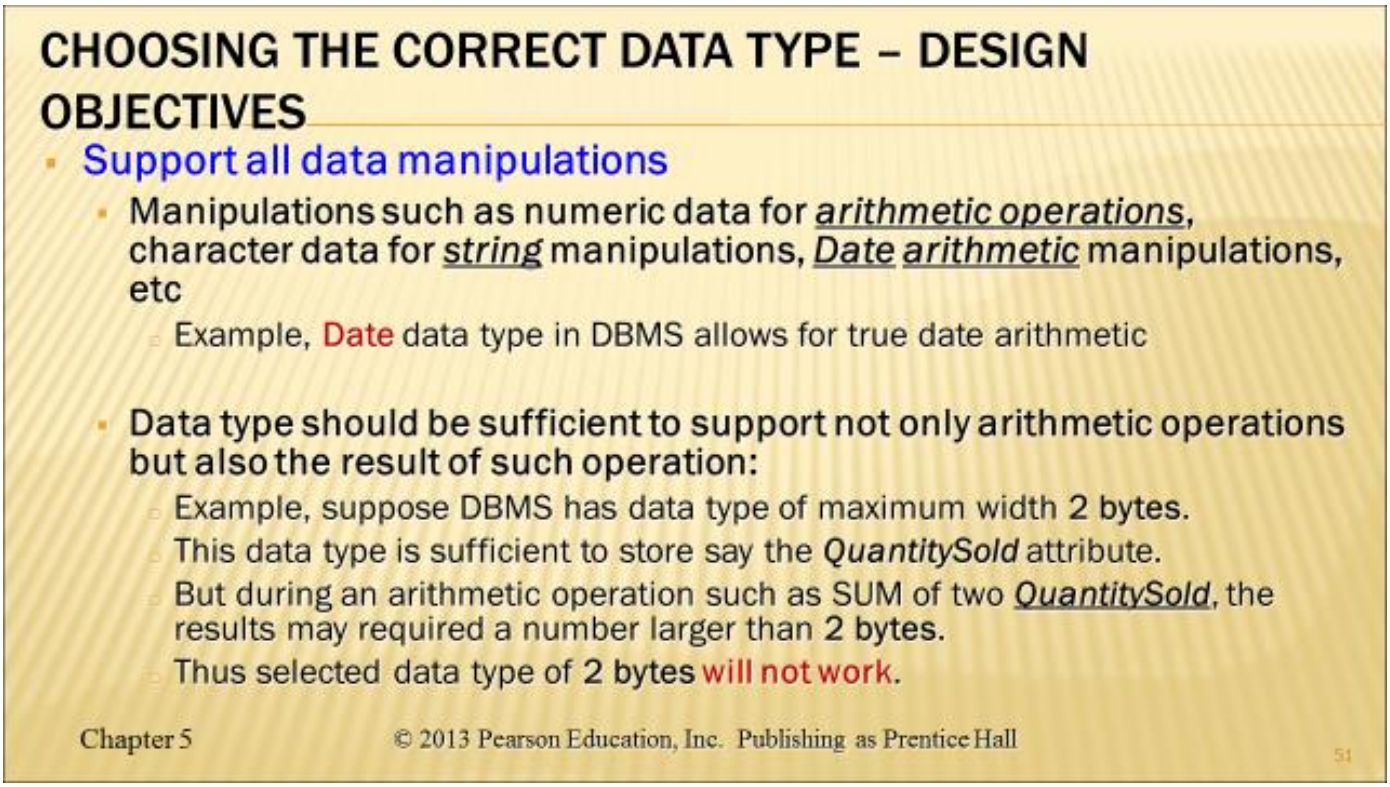
TABLE 5-1 Commonly Used Data Types in Oracle 11g

Data Type	Description
VARCHAR2	Variable-length character data with a maximum length of 4,000 characters; you must enter a maximum field length [e.g., VARCHAR2(30) specifies a field with a maximum length of 30 characters]. A string that is shorter than the maximum will consume only the required space.
CHAR	Fixed-length character data with a maximum length of 2,000 characters; default length is 1 character (e.g., CHAR(5) specifies a field with a fixed length of 5 characters, capable of holding a value from 0 to 5 characters long).
CLOB	Character large object, capable of storing up to (4 gigabytes – 1) * (database block size) of one variable-length character data field (e.g., to hold a medical instruction or a customer comment).
NUMBER	Positive or negative number in the range 10^{-130} to 10^{126}; can specify the precision (total number of digits to the left and right of the decimal point) and the scale (the number of digits to the right of the decimal point). For example, NUMBER(5) specifies an integer field with a maximum of 5 digits, and NUMBER(5,2) specifies a field with no more than 5 digits and exactly 2 digits to the right of the decimal point.
DATE	Any date from January 1, 4712 B.C., to December 31, 9999 A.D.; DATE stores the century, year, month, day, hour, minute, and second.
BLOB	Binary large object, capable of storing up to (4 gigabytes – 1) * (database block size) of binary data (e.g., a photograph or sound clip).

Choosing Data Types – Choosing Correct Data Type Design Objectives

- Here we look at the goals we target in order to choose the right data type.





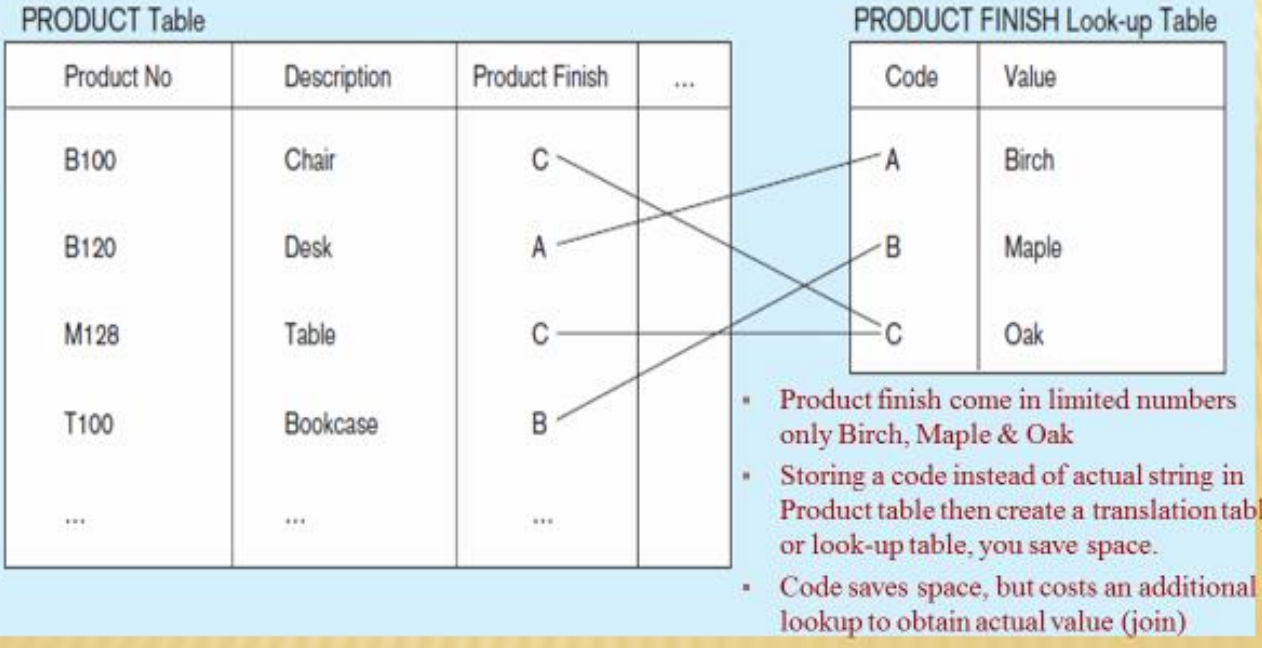
CHOOSING THE CORRECT DATA TYPE – DESIGN OBJECTIVES

- Minimized storage space
  - The goal is also to minimized storage space
  - Some attributes data can be so large, that given their data volume in a table, large storage space will be consumed
  - There are techniques you can use to reduce storage space in this scenario. One technique is known as Coding Technique

CHOOSING THE CORRECT DATA TYPE – DESIGN OBJECTIVES

- Minimize Storage Space – Coding Technique
  - Coding Technique can be used to reduce storage space when attributes data is large, that given their data volume in a table, large storage space will be consumed
  - Techniques works best when the attribute has limited numbers of possible values.
  - In the next slide we look at an example

Figure 5-2 Example of a code look-up table (Pine Valley Furniture Company)



- Product finish come in limited numbers only Birch, Maple & Oak
- Storing a code instead of actual string in Product table then create a translation table or look-up table, you save space.
- Code saves space, but costs an additional lookup to obtain actual value (join)

55 55

CHOOSING THE CORRECT DATA TYPE – DESIGN OBJECTIVES

- Improve data integrity
  - Selecting the right data type can improve data integrity (correctness, consistent, accessibility)
  - Integrity controls – control on the possible values a field or column can support.
  - Many DBMS enforce these data integrity controls in the data types they provide as follows:
    - Limit the type of data (numeric or character or date, other)
    - Length of the field or column
    - Others

Sarbanes-Oxley Act (SOX) legislates importance of financial data integrity

CHOOSING THE CORRECT DATA TYPE – DESIGN OBJECTIVES

- Improve data integrity
  - Examples of Integrity Controls a DBMS may support:

Integrity Control	Description
Default value	▪ assumed value if user does not enter an explicit value ▪ Using default can reduce data entry time since default value is skipped ▪ Can reduce data entry errors since default is automatically selected
Range control	▪ Limits the set of values allowed to be set in a field. ▪ Range may have lower and upper bound limits. ▪ Use with caution since limits of range may change with time (ex. Y2K issue in year 2000. Year column has range 00 to 99 thus no support for year 2000)

CHOOSING THE CORRECT DATA TYPE – DESIGN OBJECTIVES

- Improve data integrity (Cont.) Examples (cont):

Integrity Control	Description
Null value control	▪ A NULL is an EMPTY VALUE. Caution is NOT a BLANK. ▪ This control provides mechanism for allowing or prohibiting empty fields. ▪ Many columns may required that NULLS are not allowed, depending on company policies ▪ Example, a PRIMARY KEY, must have integrity control to prohibit NULL values. A KEY cannot be NULL.
Referential integrity	▪ Rule – Each foreign key in one table, it must match the primary key of the other table in relation or value must be NULL. ▪ Range control (and null value allowances) for foreign-key to primary-key match-ups ▪ A foreign key can be NULL only if relationship is optional otherwise it cannot be NULL. ▪ Maintains consistency between two tables.

Choosing Data Types – Handling missing data

- What do we do when NULLS or missing data is required in a column or table cell? Here we look at techniques we can use to address empty data

CHOOSING CORRECT DATA TYPE – HANDLING MISSING DATA

- Handling Missing Data (NULL or what?)
  - There a cases where a NULL is necessary
  - But entering no value may not be sufficient
  - For example:
    - Supposed a customer zip code can be null or not required.
    - Nevertheless a report is later created that summarizes total sales by month & zip code.
    - What should the report look like for those zip code entries that are null?

Chapter 5 © 2013 Pearson Education, Inc. Publishing as Prentice Hall 50

CHOOSING THE CORRECT DATA TYPE – DESIGN OBJECTIVES

- Support all data manipulations
  - Manipulations such as numeric data for *arithmetic operations*, character data for *string* manipulations, *Date arithmetic* manipulations, etc
    - Example, Date data type in DBMS allows for true date arithmetic
  - Data type should be sufficient to support not only arithmetic operations but also the result of such operation:
    - Example, suppose DBMS has data type of maximum width 2 bytes.
    - This data type is sufficient to store say the *QuantitySold* attribute.
    - But during an arithmetic operation such as SUM of two *QuantitySold*, the results may required a number larger than 2 bytes.
    - Thus selected data type of 2 bytes will not work.

Chapter 5 © 2013 Pearson Education, Inc. Publishing as Prentice Hall 51

CHOOSING CORRECT DATA TYPE – HANDLING MISSING DATA

- Options for handling missing data:

Option	Description
Default value (From Data Integrity Controls)	▪ Use a default value instead of a NULL. Select a value that makes sense to the situation
Null value control (From Data Integrity Controls)	▪ This control provides mechanism for allowing or prohibiting empty fields. ▪ Simply prohibit nulls
Substitute an estimate for the missing value (e.g., using a formula)	▪ Substitute an estimate of the missing value ▪ Use a formula to calculate the missing value if needed ▪ Estimates must be marked so that user know the value is not real but an estimate.

Triggers can be used to perform these operations.

CHOOSING CORRECT DATA TYPE – HANDLING MISSING DATA

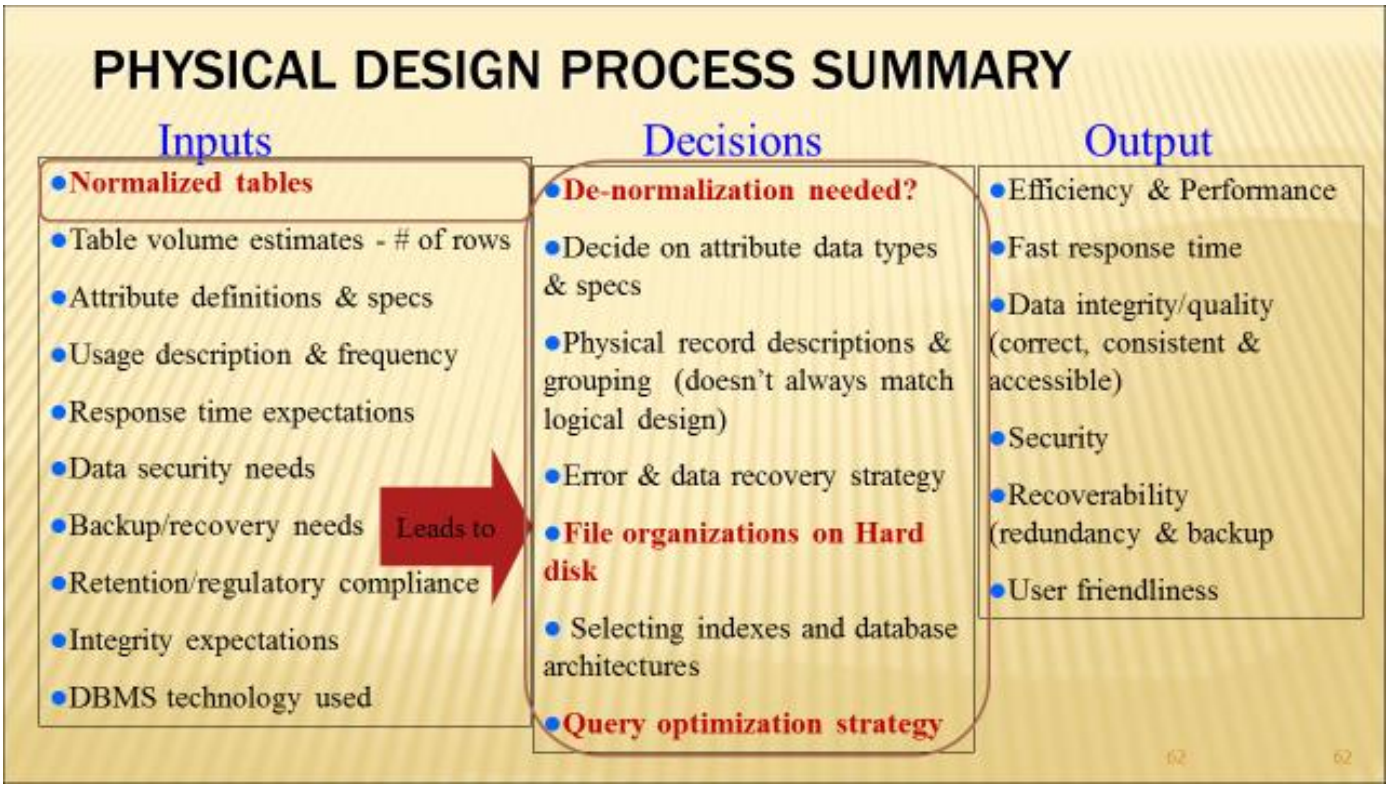
- Options for handling missing data (cont.):

Option	Description
Report of missing values	▪ Construct a report listing missing values ▪ Mechanism must be in place to help users resole unknown values quickly. ▪ Database Triggers can be used to assist. Triggers automatically execute when some event occurs. You can program the triggers to log missing entries in a file & another trigger to create report.
Sensitivity Testing	▪ Perform sensitive testing, or process that will handle decisions on null values ▪ Idea is to ignore missing data unless the value is significant. If so, handle via a process. ▪ Most complex method since requires sophisticated programming in the database or client application ▪ Modern DBMS provide mechanism to create programming logic or processing such as triggers, stored procedures etc.

Triggers & other DBMS capabilities can be used to perform these operations.

5.3.3 Physical Design Process – De-normalization

- DBMS have important role in how data is actually physically stored on the media.
- In this section we discuss how **de-normalization** can be used as a mechanism to **improve efficiency**.
- First we review our physical design process and the **INPUT** value we are now targeting which is **Normalized Tables** and the required **DECISIONS** needed to be made to in order to improve efficiency.



De-normalization Overview & Impact on Performance

- DBMS have important role in how data is actually physically stored on the media.
- The efficiency of the database processing is significantly affected by how the logical tables are physically stored (Physical Records) in the database.

DENORMALIZATION

- DBMS Role, Logical Design Implemenation & Efficiency
  - DBMS determines how data is stored in physical media
  - Efficiency is significantly affected by how logical tables are actually structured as physical tables in database
  - Objectives of this section:
    - Discuss how de-normalization can be used as mechanism to improve efficient processing of data & quick retrieval
    - Discuss de-normalization approach

Chapter 5 © 2013 Pearson Education, Inc. Publishing as Prentice Hall 75

DENORMALIZATION

- Why De-normalize? (Cont.)
  - Problems with Normalization:
    - Often when querying data, attributes in tables are not used together, data/attributes from other tables are needed. *Thus a JOIN of multiple related tables is required for queries and reports*.
    - JOINING tables results in performance issues due to considerable computer resources required to perform the JOIN.
    - JOINS are time-consuming and result in inefficiency
    - The PERFORMANCE difference between a totally Normalize & Partially Normalize (de-normalized) database can be DRAMATIC!!!

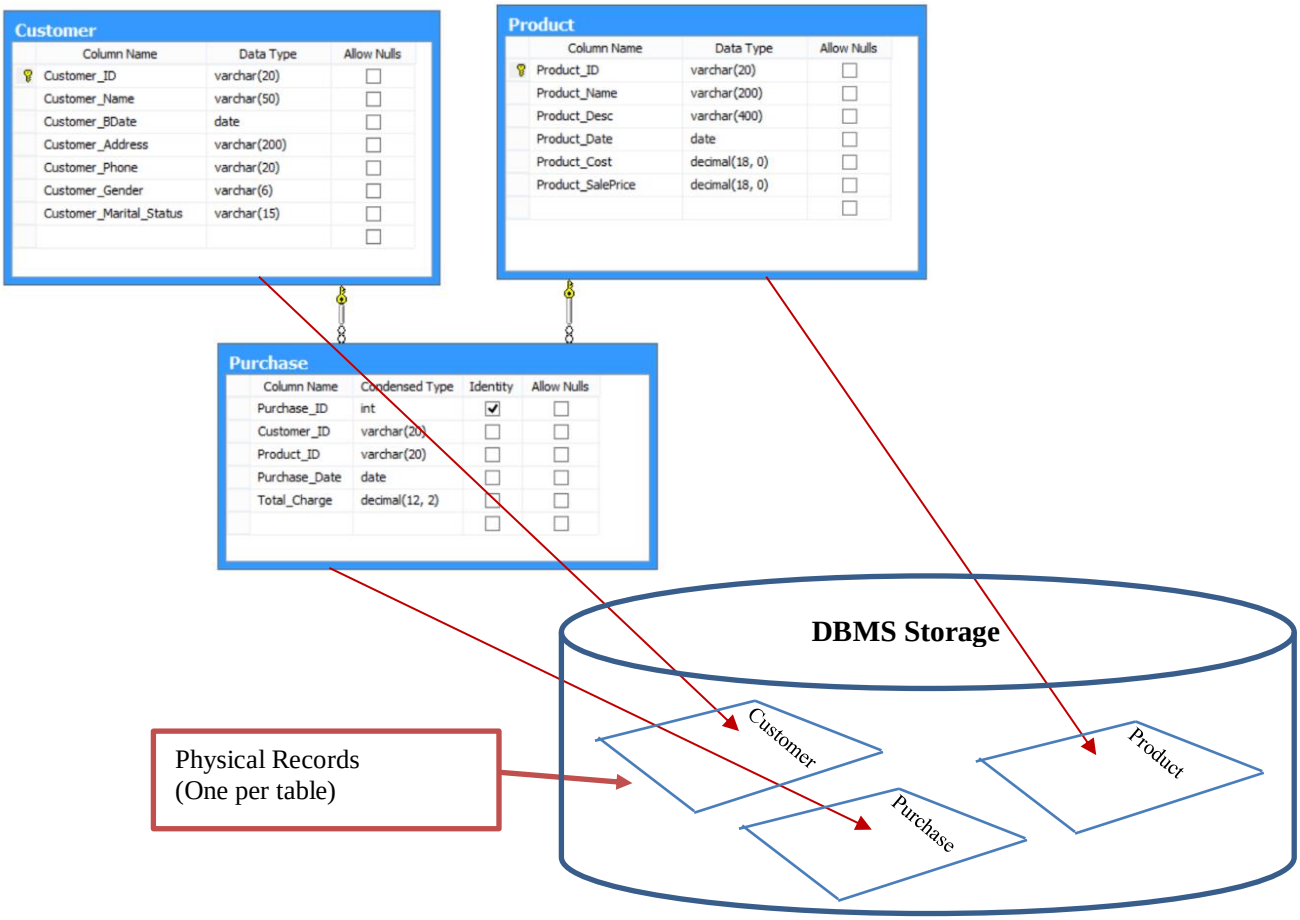
77 77

DENORMALIZATION

- De-normalization Defined
  - Transforming *normalized* relations into *non-normalized* physical record specifications

- Why De-normalize?
  - Normalization solves the redundancy (storage space) problem (not speed)
  - Normalization creates large numbers of tables. Each table is stored in its own Physical Record(one-to-one physical record per table)
  - Our Goals – Efficient data processing (speed) is more important. (Redundancy or saving storage space takes second place to speed)

- Examples of joining tables:



Virtual Table temp Created in DBMS to implement JOIN

Overhead of JOIN Query & one source of inefficiencies

C.CustID	…	P.CustID	P.ProdID	….	Prod.ProdID	…
1		1	101		101	
4		4	201		201	
14		14	100		100	

- Joining two tables (Customer & Purchase) to obtain data contained in the two tables:

```
--Equi Join Query between Customer & Purchase Table
SELECT Customer.Name, Purchase.Purchase_Date, Purchase.Total_Charge
FROM Customer, Purchase
WHERE Purchase.Customer_ID = Customer.Customer_ID
```

- Joining three tables (Customer, Purchase & Product) to obtain data contained in all three tables:

```
--Equi Join Query between Customer, Purchase & Product Table
SELECT Customer.Name, Product.Product_Name, Purchase.Purchase_Date, Purchase.Total_Charge
FROM Customer, Purchase, Product
WHERE Purchase.Customer_ID = Customer.Customer_ID
      AND Purchase.Product_ID = Product.Product_ID
```

DENORMALIZATION

- De-normalization Benefits
  - Improved performance (speed) by reducing number of table lookups (i.e. reduce number of necessary JOINS in queries)

- How is done?
  1. May partition a table into several Physical Records
  2. May combine attributes from several tables together into one Physical Record
  3. Combination of 1 & 2

DENORMALIZATION

- De-normalization Disadvantages & Cost
  - Redundancy or Wasted storage space due to data duplication
  - Data integrity/consistency threats
  - Caution must be taken when de-normalizing

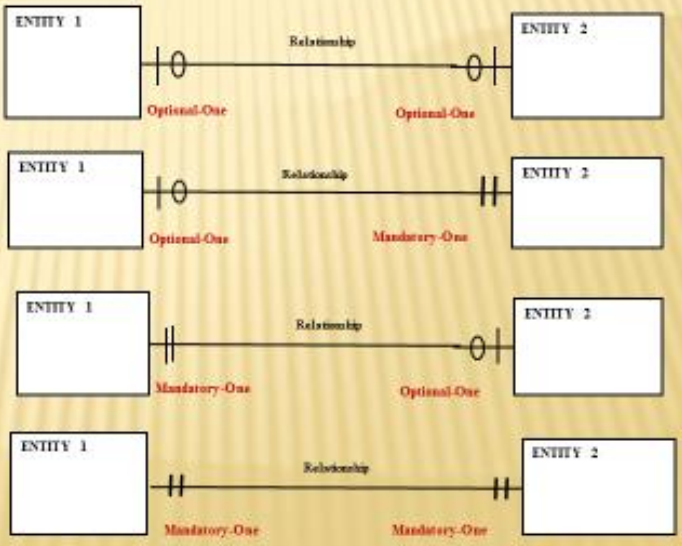
- Common De-normalization Opportunities
  1. Two entities with a One-to-one relationship
  2. A many-to-many relationship (Associate Entity) with non-key attribute (no key attribute included)
  3. Reference data in a two entities with One-to-Many relationship

De-normalization Opportunities to REDUCE JOINS – Two entities with a One-to-one relationship

- Scenario #1.

DENORMALIZATION – SCENARIO #1

- Two entities with a One-to-one relationship
- Scenarios:
  - One-to-One Scenarios



DENORMALIZATION – SCENARIO #1 (CONT.)

- Scenario #1 Condition
  - ENTITY #2 contains lots of records (instances)
  - High access frequency (high usage) between these two tables
- Scenario #1 Steps
  - Combine two tables into ONE RECORD
- Example of de-normalization of 1:1 relationship – Fig 5.3
  - Student & Scholarship Application Entities 1:1 relationship, denormalized from 2 tables to one table

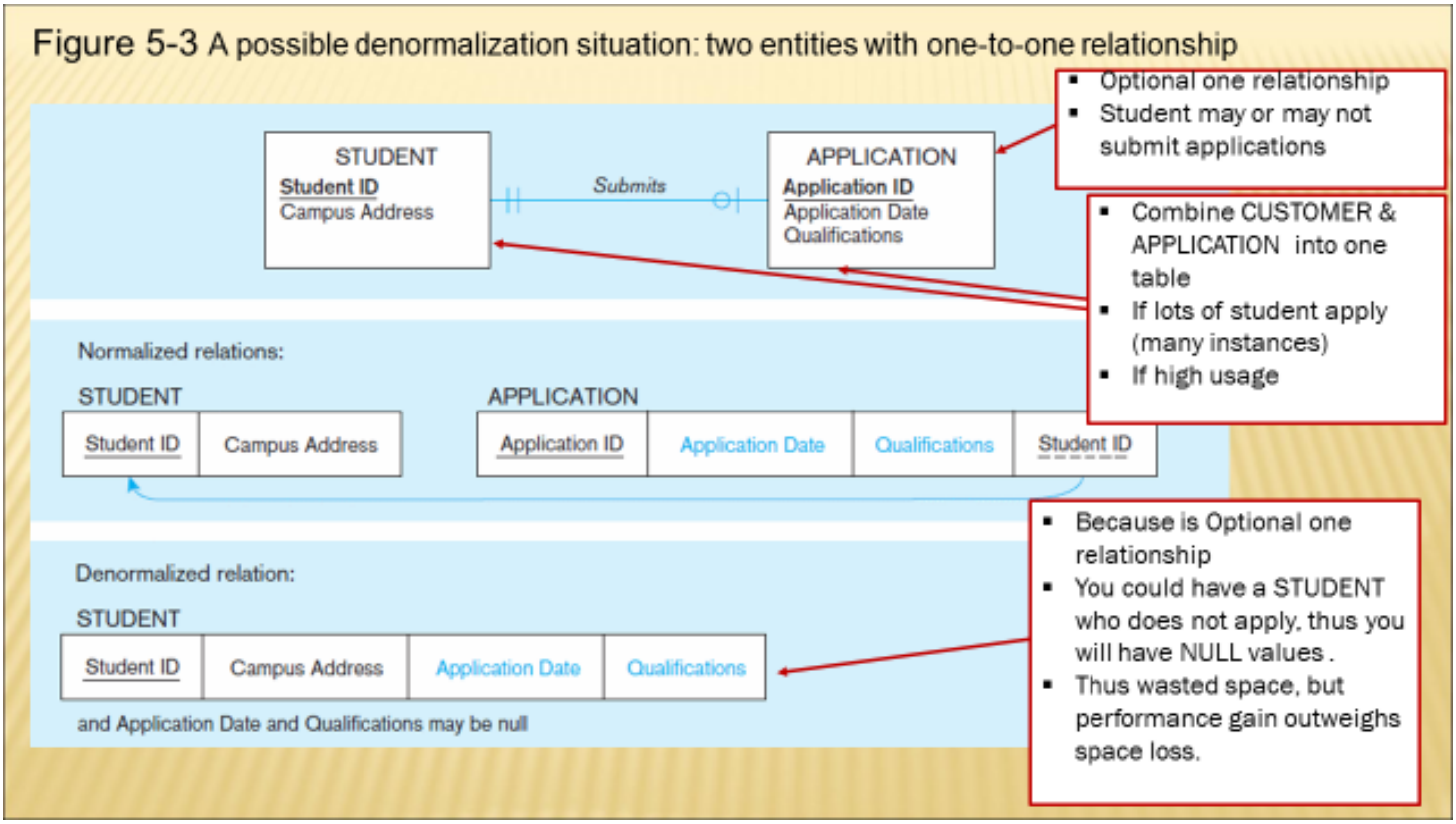


Figure 5-3 A possible denormalization situation: two entities with one-to-one relationship
STUDENT
Student ID
Campus Address
Submits
APPLICATION
Application ID
Application Date
Qualifications
Optional one relationship
Student may or may not submit applications
Combine CUSTOMER & APPLICATION into one table
If lots of student apply (many instances)
If high usage
Normalized relations:
STUDENT
Student ID
Campus Address
APPLICATION
Application ID
Application Date
Qualifications
Student ID
Denormalized relation:
STUDENT
Student ID
Campus Address
Application Date
Qualifications
and Application Date and Qualifications may be null
Because is Optional one relationship
You could have a STUDENT who does not apply, thus you will have NULL values .
Thus wasted space, but performance gain outweighs space loss.

De-normalization Opportunities to REDUCE JOINS – A many-to-many relationship (Associate Entity) with non-key attribute (no key attribute included)

- Scenario #2.

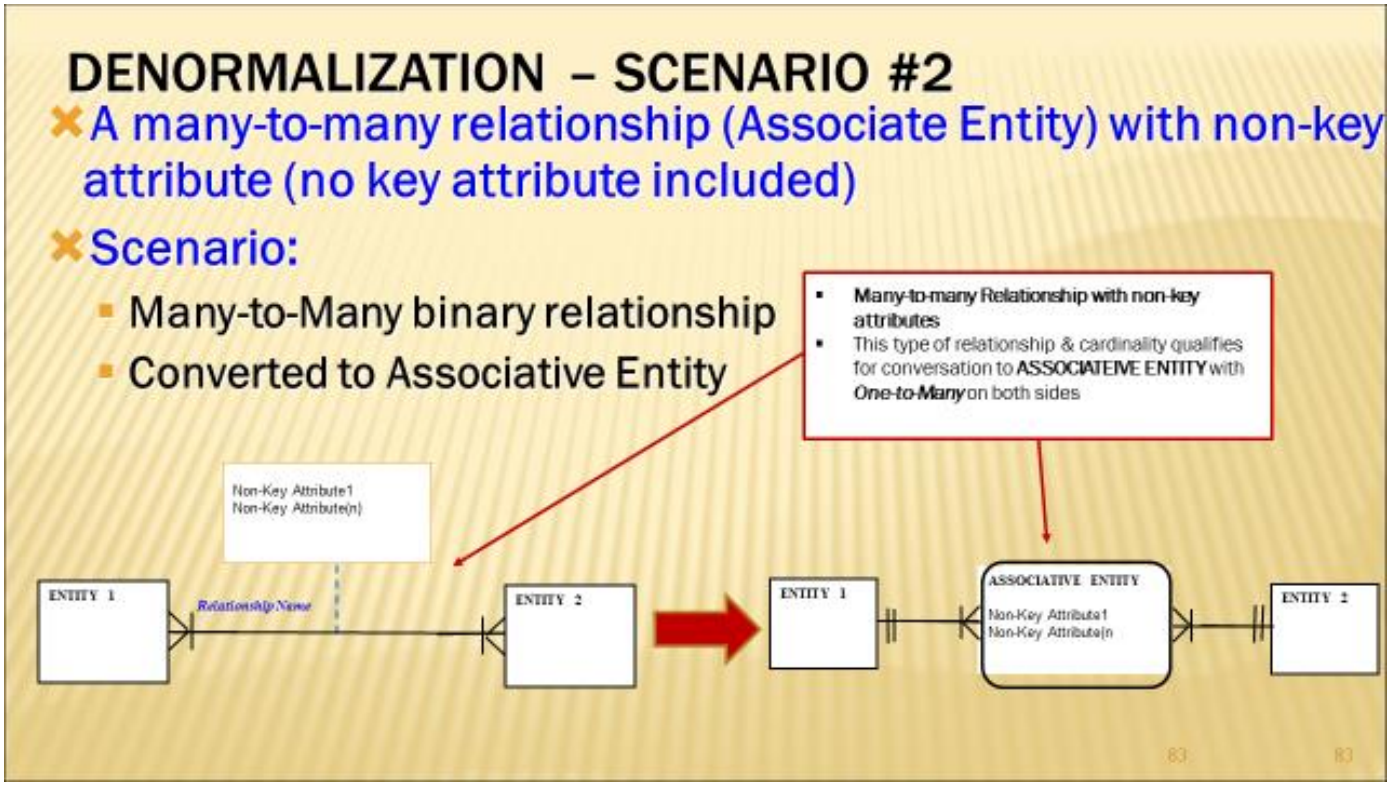
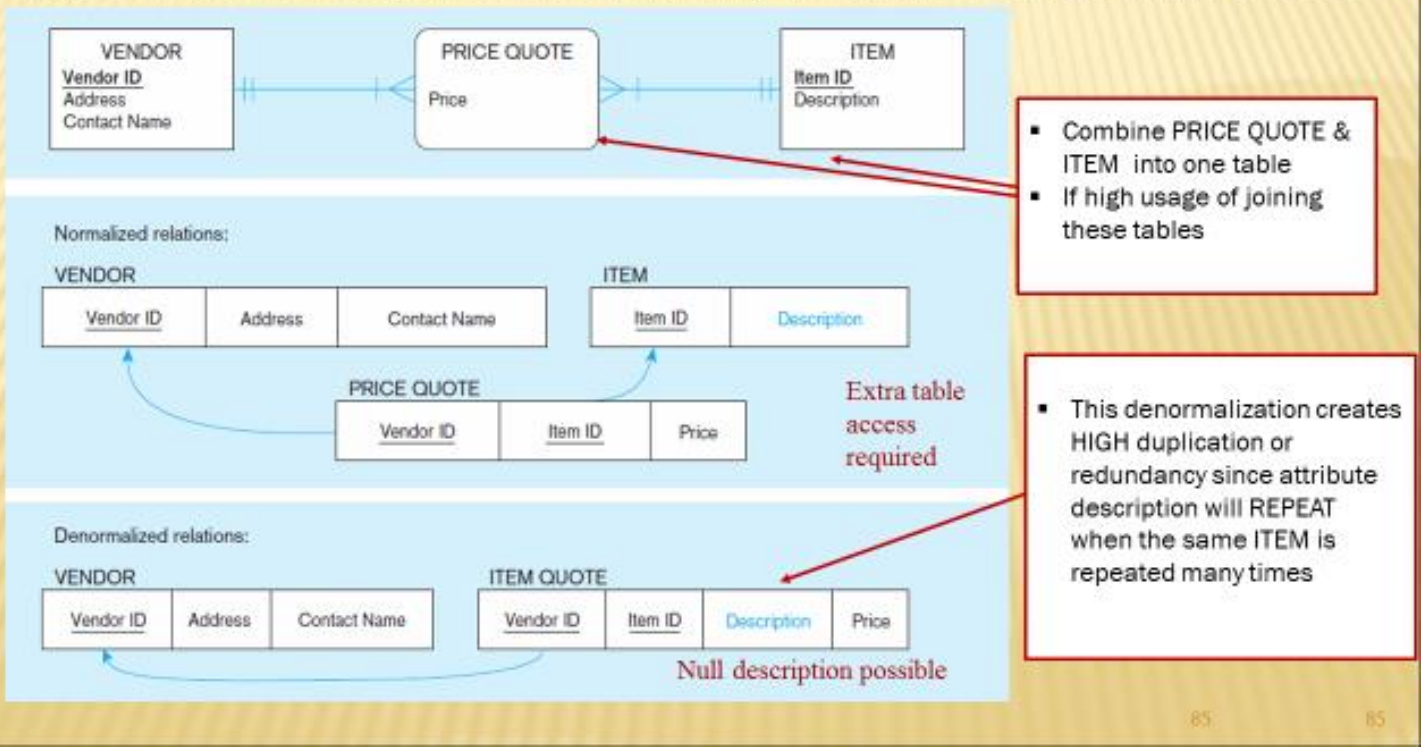


Figure 5-4 A possible denormalization situation: a many-to-many relationship with nonkey attributes



De-normalization Opportunities to REDUCE JOINS – A One-to-many binary relationship where one-side entity has no other relationship

- Scenario #3

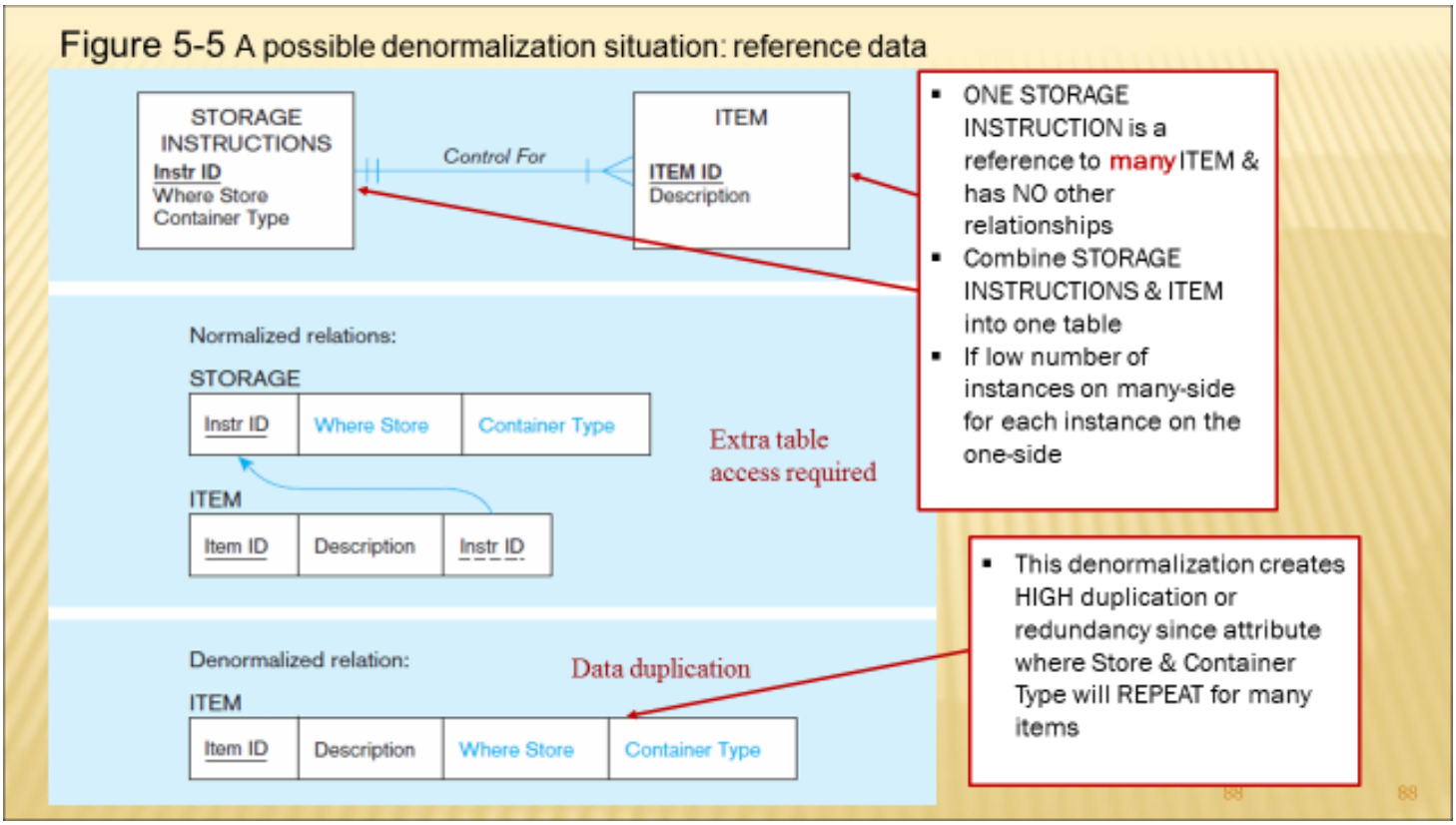
DENORMALIZATION – SCENARIO #3

- Reference Data exists in binary One-to-many relationship on the One side & the One side entity participates in NO other relationship
- Scenario:
  - One-to-Many binary relationship
  - One-side Entity attributes has reference data to the many-side Entity & One-side has no other relationships

DENORMALIZATION – SCENARIO #3 (CONT.)

- Scenario #3 Condition
  - One-to-many relationship where one-sided Entity attributes has reference data to the many-side Entity & One-side has no other relationships
  - If low number of instances on many-side for each instance on the one-side
- Scenario #3 Steps
  - Combine two tables into ONE RECORD by combining attributes from one-sided entity to the one-to-many side
- Example of de-normalization of 1:N relationship – Fig 5.5
  - ITEM & STORAGE INSTRUCTIONS relationship, denormalized from 2 tables to 1 table

Figure 5-5 A possible denormalization situation: reference data



De-normalization Final Words – Use with Caution. Consider other methods first

- Scenario #3

DENORMALIZE WITH CAUTION

- Denormalization can
  - Increase chance of errors and inconsistencies
  - Reintroduce anomalies
  - Force reprogramming when business rules change (Queries have to change)
  - Optimized for speed, but at the expense of redundancy
  - Causes more storage space
- Denormalization Consideration
  - When the gain of processing speed is significant & no other design methods are sufficient to meet expectations

DENORMALIZE WITH CAUTION

- Perhaps other methods could be used to improve performance of joins
  - Organization of tables in the database (file organization and clustering)
  - Proper query design and query optimization capabilities of DBMS
- Approach
  - Try the above methods first to achieve performance
  - If expectations not reached, then consider denormalization but with caution

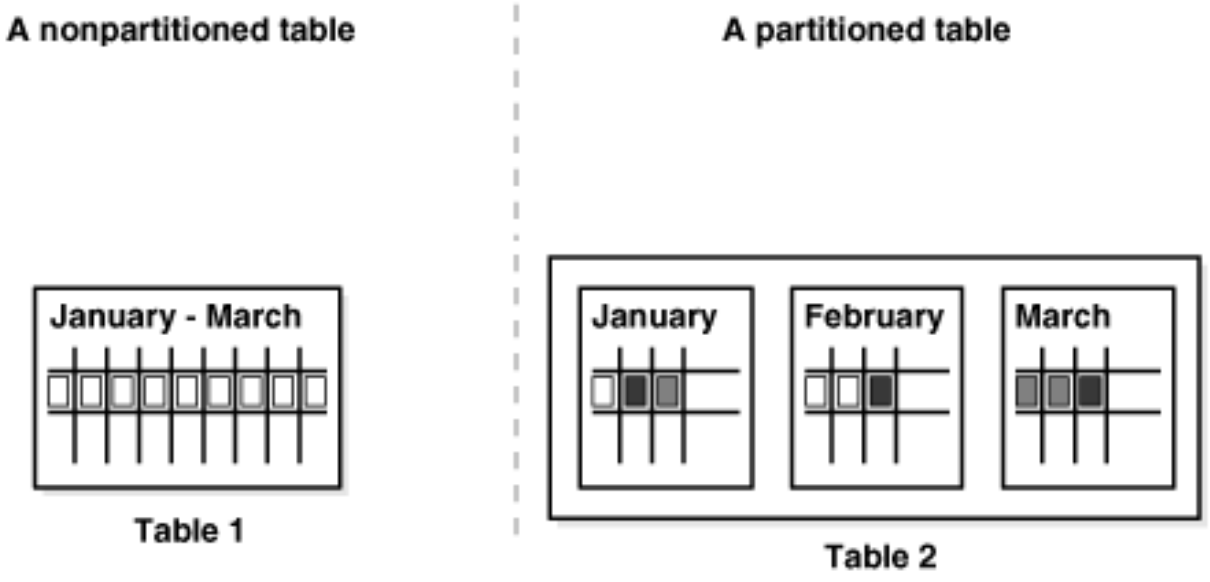
5.3.4 Physical Design Process –Partitioning Data (Another Form of De-Normalization)

- In this section we discuss another form of **de-normalization** called **PARTITIONING**, which involves breaking up or partitioning a table into more tables.
- Two types of partitioning are covered: **Horizontal & Vertical Partitioning**.

PARTITIONING

- Partitioning Defined
  - Breaking up (partitioning) a table into multiple smaller physical tables.
  - Each partitioned table has its own name.
  - A table can be partitioned by ROWS or COLUMN:
    - Partition by Row – Columns stay the same in new partitioned tables, but rows are added to each table based on the category column
    - Partition by Column – Rows stay the same in new partitioned tables, but Columns are partition to separate tables
- Figure below illustrates the concept of Partitioning:

91 91

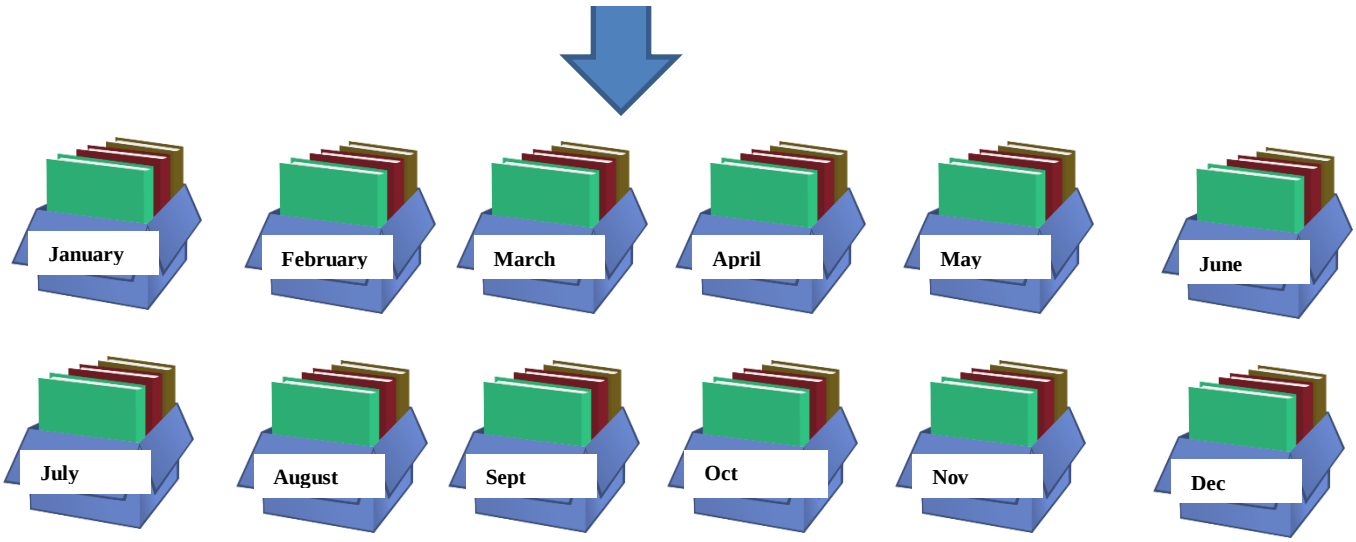


PARTITIONING

Partition Analogy

- Suppose you have an HR Manager that has one big box that contain employee folder. Each employee folder contains the employee hired date.
- Often, requests (queries) are made for list of employees hired a particular month.
- HR Manager can rummage through the box searching for the scattered folders in the one big box. Time consuming and inefficient.
- In contrast, HR Manager can use many/12 smaller boxes, each box containing only employee folders for each month.
- Now manager can go directly for the box for a particular month to respond to queries. More efficient.
- Also if a small box is damaged, the other boxes remain available.
- Finally, moving the office becomes easier since instead of carrying a single heavy box, simply carry several smaller boxes.

BIG BOX PARTITIONED TO LITTLE BOXES



PARTITIONING

- Why Partition?
  - Divide & Conquer approach (Instead of a one big table, break it down)
  - DBA has more flexibility in managing smaller tables.
  - Backup & Recovery operations can be performed on an individual partition & made unavailable, while leaving the other partitions available for use by application.
  - Query performance can improve by accessing smaller or relevant partition.
- Goals in Physical Data Model – Goal is efficient data processing (speed), therefore Partitioning help meet these goals by INPROVING PERFORMANCE & MAINTENANCE/MANEGABILITY.

92 92

PARTITIONING

- Why Partition? (Cont.)
  - Article from DB Expert: If you have a database with over 500 gigabytes of data and is not partitioned you headed for a disaster! Databases become unmanageable & serious problems will occur:
    - SQL Queries May Perform Poorly: Without partitioning, SQL queries with full-table scans take hours to complete. The smaller the partition or table, the faster the performance.
    - Recovery: File recoveries takes days, not minutes. Smaller partition faster recovery!
    - Maintenance: Rebuilding indexes (important to reclaim space and improve performance is affected.
    - Backups: Backup of larger tables will be slow. Smaller tables fast backups.

93 93

PARTITIONING

- When to Partition?

1. Oracle suggests tables greater than 2 GB should be considered.
2. Tables containing HISTORICAL DATA.
3. When contents of a table need to be distributed across different STORAGE DEVICES and managed separately

94 94

PARTITIONING

- Partition is TRANSPARENT to the Client Application
  - Partition is entirely transparent to the client application.
  - No costly or time-consuming application changes are required.
  - You do not need to rewrite the Client Application
  - No changes to SQL command required (unless desired for even more performance). The DBMS Query Optimization system will automatically decide which partition to use.
  - DBA has flexibility. Table that was partitioned can be managed with its individual partitioned tables or as a whole.

95 95

PARTITIONING STRATEGIES

- How is done?
  1. Horizontal Partitioning
  2. Vertical Partitioning
  3. Combination of 1 & 2

HORIZONTAL PARTITIONING

PARTITIONING – HORIZONTAL PARTITIONING

- Horizontal Partitioning:
  - Distributing/moving the rows of a logical table into several separate physical tables based on a ***common*** *column* or *category*
  - Each new table created from partitioning have the same columns or attributes
  - In short, breaking up tables into smaller pieces
  - Example:
    - Customer table has a column or attribute named Region (which represents *North East*, *South East*, *North West*, *South West*, etc.)
    - Region is the column that determines the category.
    - This table could be broken up into a four region customer table: NORTH EAST, SOUTH EAST, NORTH WEST, SOUTH WEST.

93 93

PARTITIONING – HORIZONTAL PARTITIONING

- Horizontal Partitioning Usage (When to partition?)
  - Situations where different users need access to different categories of rows. Each category can be placed into separate tables & processed separately
  - Useful in scenarios where a table has large amount of data (Oracle suggests tables greater than 2 GB)
  - Table with historical data where new data is added to newest partition (older table can be read-only, newest table is writable)
  - When content of table needs to be distributed across different storage devices

131 131

PARTITIONING – HORIZONTAL PARTITIONING

- How is done:
  1. Use SQL CREATE TABLE command to break up the rows of a logical table into several separate physical tables based on a common column or category
  2. Each new table created from partitioning have the same columns or attributes
- Types of Horizontal Partitioning Strategies:
  1. Range Partitioning
  2. Hash Partitioning
  3. List Partitioning

95 95

PARTITIONING – HORIZONTAL PARTITIONING

- Important definitions before defining types of Horizontal Partitioning:
  - Horizontal Partitioning – Break up table into smaller tables based on a category column
  - Partitioning Key – (*The category column*) one or more columns that determine the point of partition where each row will be stored in target partitioned table
  - Partitioned Table – Table that was partitioned

96 96

Horizontal Partitioning – Range Partitioning

HORIZONTAL PARTITIONING – RANGE PARTITIONING

- Range Partition Defined:
  - Partitions a table based on a RANGE of values (lower & upper key limits) of the Partitioning Key
  - Most common of the partitioning methods
- Range Partition Usage Scenarios:
  - Used when you have distinct ranges of data you want to store together
  - Can be used with any category type Partitioning Key or column that has a range (dates, zipcode, ID, etc.)

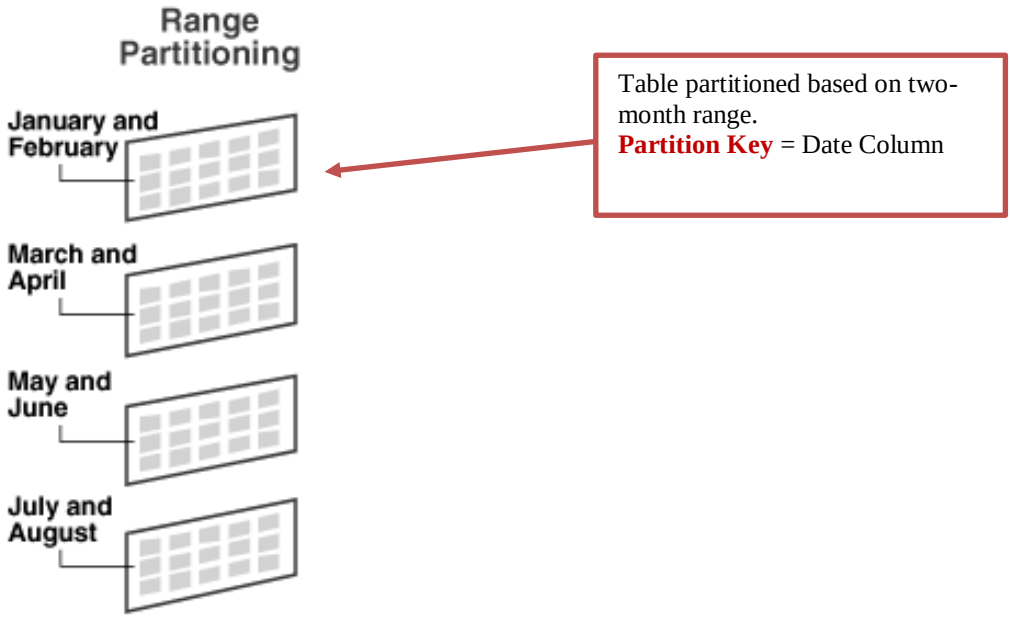
97 97

HORIZONTAL PARTITIONING – RANGE PARTITIONING

- Range Partition Usage Scenarios (Cont.):
  - Often used with dates columns, because date is often a qualifier in queries (determining factor)
    - Ex. For a table with date column as partition key & partitioned by two months range:
      - January to February 2010 partition would contain rows with values from 01/01/2010 to 01/31/2010, March to April 2010 partition would contain rows with values from 03/01/2010 to 03/31/2010, etc. See figure below.

98 98

Figure - Tables partitioned using Range Partitioning by two month periods



HORIZONTAL PARTITIONING – RANGE PARTITIONING

- How is done:
  - Done when table is created using SQL CREATE TABLE Statement.
  - Review: SQL CREATE TABLE Statement is used to create a table
  - A basic syntax to this statement

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  column_n datatype [ NULL | NOT NULL ]
)
```

HORIZONTAL PARTITIONING – RANGE PARTITIONING

How is done (Cont.):

- The SQL CREATE TABLE Statement is extended using the PARTITION BY RANGE & PARTITION Clauses:

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  column_n datatype [ NULL | NOT NULL ]
)
PARTITION BY RANGE(partitionKey) (
PARTITION partition_name VALUES LESS THAN {(expr) | MAXVALUE)}
)
```

PARTITION BY RANGE, PartitionKey & statements

PARTITION, VALUES LESS THAN & MAXVALUE Clauses

HORIZONTAL PARTITIONING – RANGE PARTITIONING

How is done (Cont.):

- The CREATE TABLE statement contains two clauses used to determine the range and maximum value:
  - VALUE LESS THAN (expr | MAXVALUE) – used to compare & determine *non-inclusive* upper-bound of every range targeted for a partition table.
    - Expr = transition point or highest value in the range. Is *non-inclusive so not included in expression.*
    - Example: PARTITION e1 VALUE LESS THAN (1000) will add all rows whose value is less than 1000 (999 or less) to the partition table e1.
  - MAXVALUE – a virtual infinite value provided by DBMS that will identifies the highest partition possible of the range. This value is higher than any of the partition key, including a NULL.
    - Example: PARTITION e3 VALUE LESS THAN (MAXVALUE) will add all rows whose value is less than the maximum value to the partition table e3.

HORIZONTAL PARTITIONING – RANGE PARTITIONING

- Range Partitioning Example 1 – Create sales Table that contains information on product sales & partition based on time_id:
  - Creates a table & uses range portioning on the *time_id* column and divides into 4 partitions or tables based on a provided time ranges.

```
CREATE TABLE time_range_sales
   ( prod_id        NUMBER(6),
     cust_id        NUMBER,
     time_id        DATE,
     channel_id     CHAR(1),
     promo_id       NUMBER(6),
     quantity_sold  NUMBER(3),
     amount_sold    NUMBER(10,2)
   )
PARTITION BY RANGE (time_id)
 (PARTITION SALES_1998 VALUES LESS THAN (TO_DATE('01-JAN-1999','DD-MON-YYYY')),
  PARTITION SALES_1999 VALUES LESS THAN (TO_DATE('01-JAN-2000','DD-MON-YYYY')),
  PARTITION SALES_2000 VALUES LESS THAN (TO_DATE('01-JAN-2001','DD-MON-YYYY')),
  PARTITION SALES_2001 VALUES LESS THAN (MAXVALUE)
 );
```


HORIZONTAL PARTITIONING – RANGE PARTITIONING

- Range Partitioning Example 1 (Cont.):
  - We assume the table currently has the following data:

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
35	2606	17-FEB-00	3	999	1	54.94
45	9491	28-AUG-98	4	350	1	47.45

HORIZONTAL PARTITIONING – RANGE PARTITIONING

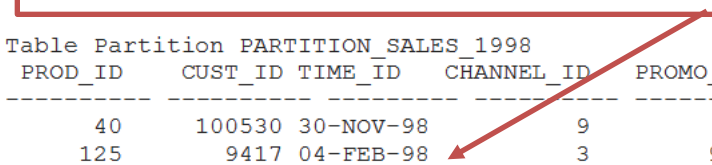
Range Partitioning Example 1 (Cont.):

- The resultant tables are shown in figure below:
- Note the following results & process:
  - Names of the Partitions came from the PARTITION clause in the CREATE TABLE statement
  - The database chooses the rows for each partition based on the time_id value specified in the PARTITION BY RANGE clause & the transition point or maximum value
  - Example:
    - In the results below, the SALES_1998 partition table contains rows with partition key (time_id) VALUE LESS THAN 01-JAN-1999

104 104

All rows less than 01-JAN-1999 are placed in this table SALES_1998 based on STATEMENT:

```
PARTITION SALES_1998 VALUES LESS THAN (TO_DATE('01-JAN-1999','DD-MON-YYYY'))
```



```
Table Partition PARTITION_SALES_1998
 PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
      40     100530 30-NOV-98          9         33             1       44.99
     125       9417 04-FEB-98          3        999             1       16.86
      45       9491 28-AUG-98          4        350             1       47.45


Table Partition PARTITION_SALES_1999
 PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
     116      11393 05-JUN-99          2        999             1       12.18
      36       4523 27-JAN-99          3        999             1       53.89
      24      11899 26-JUN-99          4        999             1       43.04


Table Partition PARTITION_SALES_2000
 PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
     133       9450 01-DEC-00          2        999             1       31.28
      35       2606 17-FEB-00          3        999             1       54.94


Table Partition PARTITION_SALES_2001
 PROD_ID    CUST_ID TIME_ID   CHANNEL_ID   PROMO_ID QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- --------- ---------- ---------- ------------- -----------
     118        133 06-JUN-01          2        999             1       17.12
      30        170 23-FEB-01          2        999             1         8.8
```

HORIZONTAL PARTITIONING – RANGE PARTITIONING

- Range Partitioning Example 2 – Creating Employee Table & partitioning based on employee ID number:
  - Creates a table & use VALUE LESS THAN to determine the upper-bound of ranges of idnumber for each partitioned table & MAXVALUE to determine the upper-bound range to the last partition

```
CREATE TABLE Employee (
   idnumber NUMBER(4),
   name     VARCHAR2(30),
   salary   NUMBER
)
PARTITION BY RANGE(idnumber) (
  PARTITION e1 VALUE LESS THAN (1000),
  PARTITION e2 VALUE LESS THAN (2000),
  PARTITION e3 VALUE LESS THAN (MAXVALUE)
)
```

105 105

HORIZONTAL PARTITIONING – RANGE PARTITIONING

- Range Partitioning Example 3 – Creating Sales Range Table & partitioning based on employee sales date:
  - Creates a table & use VALUE LESS THAN to determine the upper-bound of ranges of salesDate for each partitioned table

```
CREATE TABLE sales range (
   salesmanID   NUMBER(5),
   salesmanName VARCHAR2(30),
   salesAmount  NUMBER(10),
   salesDate    DATE
)
PARTITION BY RANGE(sales_date)
(
PARTITION sales_jan2000 VALUES LESS THAN(TO_DATE('02/01/2000','DD/MM/YYYY')),
PARTITION sales_feb2000 VALUES LESS THAN(TO_DATE('03/01/2000','DD/MM/YYYY')),
PARTITION sales_mar2000 VALUES LESS THAN(TO_DATE('04/01/2000','DD/MM/YYYY')),
PARTITION sales_apr2000 VALUES LESS THAN(TO_DATE('05/01/2000','DD/MM/YYYY'))
)
```

106 106

Horizontal Partitioning – Hash Partitioning

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Important Definitions:
  - Hash – A *number* generated from a string of text.
    - Generated by an hashing algorithm or hash function.
    - Unique & extremely unlikely hash number is duplicated
- Hash Partition Defined:
  - A HASHING algorithm by the DBMS is applied to the Partitioning Key you identify.
  - A HASH value is created for each Partitioned Table.
  - The tables are partitioned based on the HASH values.
  - Hash value determines membership (which rows) to a partition.

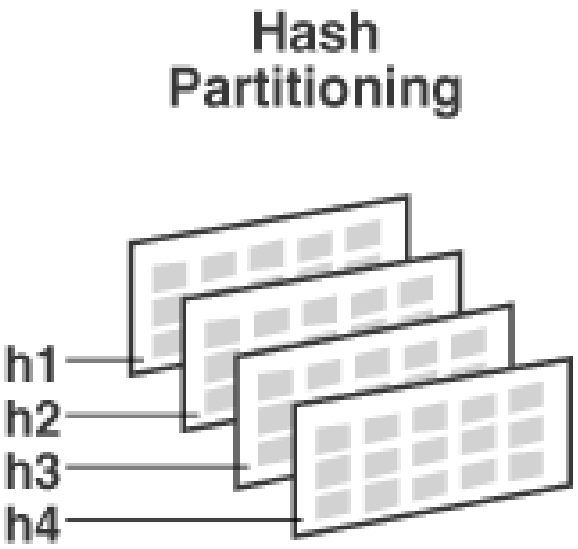
102 102

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partition Defined (Cont.):
  - Hash function distributes rows evenly among partitions, giving each partition approximately the same size.
  - Example - Assuming you desire 4 partitions, the hash function could return a value from hash1 to hash4 & automatically partition the tables for you (See figure below)

105 105

Figure - Tables partitioned by hash values (h1, h2, h3 &h4)



HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partition Usage:
  - Ideal method for situation where you want to distributing data evenly across partitioned tables.
  - Easy to use alternative to Range Partitioning in which you must specify explicitly into *which partition* a column value or set of columns is to be stored.
  - With Hash Partition the DBMS takes care of this for you.
  - Simply provide the column or Partitioning Key and the # of desired Partition and DBMS will do the work

106 106

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partition Usage (Cont.):
  - Good in situations where Partitioning Key is not obvious
  - Important – Useful in situations with LARGE tables:
    - You can divide large tables into smaller tables.
    - Now you can increase manageability & performance by working (Querying) with smaller tables.
    - *Manageability* – Working with smaller tables thus easier
    - *Performance benefits* – Faster response time since small tables

107 107

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partition Usage (Cont):
  - Important – Useful in Online Transaction Processing Systems (OLTP):
    - OLTP – Software applications capable of supporting transaction-oriented applications on the Internet.
    - In other words, large number of users conducting short transactions, such as order entry, financial transactions etc.
    - *Requirements of OLTP*– Fast response time required! Queries must be FAST & return few records (Update Queries are common in this scenario).
    - *Hash Performance benefits* – Fast response time since querying smaller tables.

HORIZONTAL PARTITIONING – HASH PARTITIONING

- How is done:
  - The DBMS (ex. Oracle) takes care of everything for you:
    - Just specify the Partition Key (column) to be hashed & number of partitions to divide.
  - Done when table is created using SQL CREATE TABLE Statement.
  - You add a PARTITION BY HASH (expr) clause to the CREATE TABLE statement

104 104

HORIZONTAL PARTITIONING – HASH PARTITIONING

- How is done (Cont.):
  - The SQL CREATE TABLE Statement extended to use Hash Partition:

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  column_n datatype [ NULL | NOT NULL ]
)
PARTITION BY HASH(expr/partitionKey)
PARTITIONS num;
```

109

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partitioning Example 1 – Create sales Table that contains information on product sales & partition based on product ID:
  - Creates a table & uses hashing portioning on the *prod_id* column and divides into 2 partitions or tables

```
CREATE TABLE hash_sales
   ( prod_id          NUMBER(6),
     cust_id          NUMBER,
     time_id          DATE,
     channel_id       CHAR(1),
     promo_id         NUMBER(6),
     quantity_sold    NUMBER(3),
     amount_sold      NUMBER(10,2)
   )
PARTITION BY HASH (prod_id)
PARTITIONS 2;
```

PARTITION KEY

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partitioning Example 1 (Cont.):
  - We assume the table currently has the following data:

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
35	2606	17-FEB-00	3	999	1	54.94
45	9491	28-AUG-98	4	350	1	47.45

HORIZONTAL PARTITIONING – HASH PARTITIONING

Hash Partitioning Example 1 (Cont.):

- The resultant tables are shown in figure below:
- Note the following results & process:
  - Names of the Partitions are System-generated (I believe you can also assign table name. Need to research)
  - You CANNOT specify the partition in which rows are placed. Is done by DBMS HASH FUNCTION
  - If you change the number of partitions, by changing the CREATE STATEMENT, the DBMS Hash Function redistributes the data over all the partitions.

Name of partitioned table is system generated

Table Partition SYS_P33

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
36	4523	27-JAN-99	3	999	1	53.89
30	170	23-FEB-01	2	999	1	8.8
35	2606	17-FEB-00	3	999	1	54.94

Table Partition SYS_P34

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
133	9450	01-DEC-00	2	999	1	31.28
125	9417	04-FEB-98	3	999	1	16.86
24	11899	26-JUN-99	4	999	1	43.04
45	9491	28-AUG-98	4	350	1	47.45

Target table for rows determined by HASH FUNCTION.

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partitioning Example 2 – Create Employee Table & partition based on store employee works in:
  - Creates a table & uses hashing portioning on the *store_id* column and divides into 4 partitions or tables

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store id)
PARTITIONS 4;
```

115 115

HORIZONTAL PARTITIONING – HASH PARTITIONING

- Hash Partitioning Example 3 – Create Employee Table & partition based on date employee was hired:
  - Creates a table & uses hashing portioning on the *hired* column or year employee was hired and divides into 4 partitions or tables

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;
```

116 116

HORIZONTAL PARTITIONING – HASH PARTITIONING

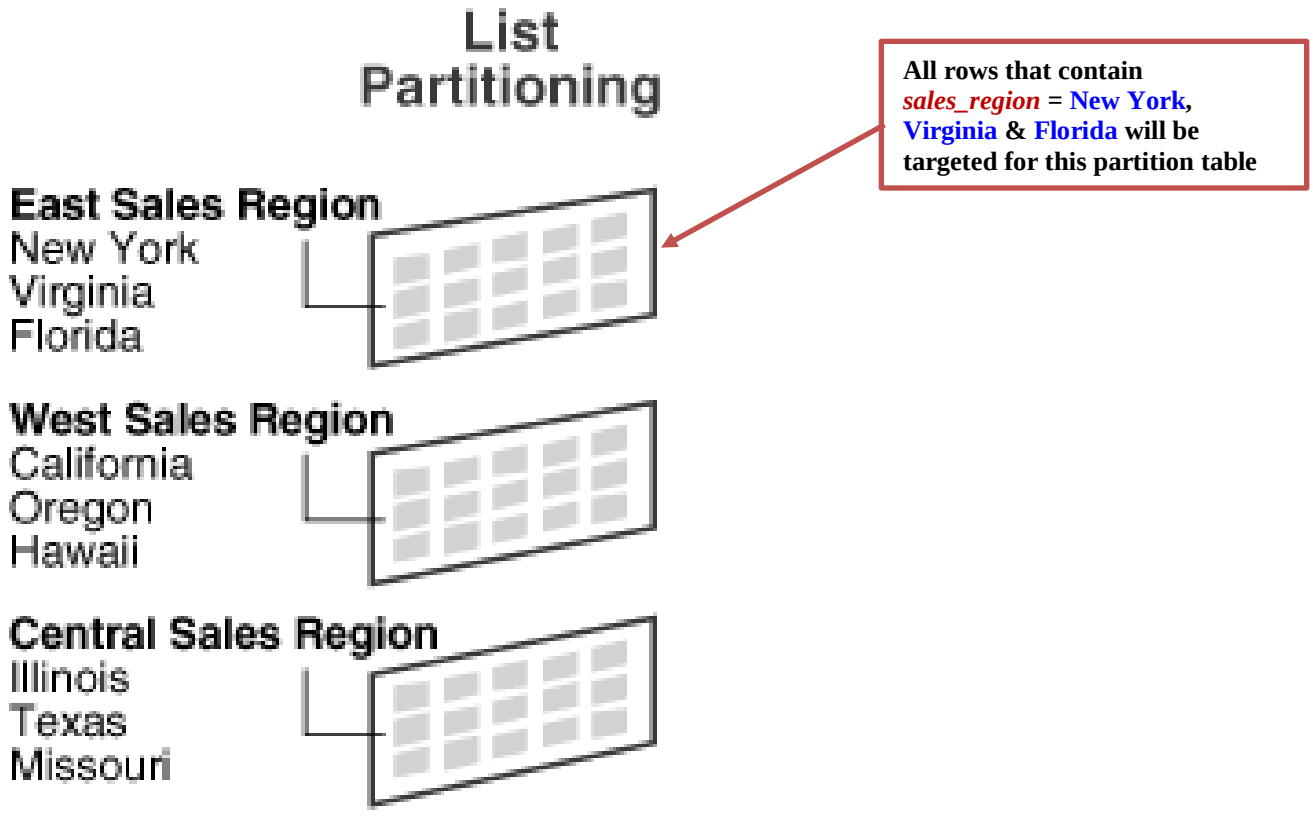
- Hash Partitioning Example 4 – Create Sales Table & partition based on Salesman ID:
  - Creates a table & uses hashing portioning on the *hired* column or year employee was hired and divides into 4 partitions or tables

```
CREATE TABLE sales (
   salesmanID   NUMBER(5),
   salesmanName VARCHAR2(30),
   salesAmount  NUMBER(10),
   weekNo       NUMBER(2)
)
PARTITION BY HASH(salesmanID)
PARTITIONS 4
)
```

Horizontal Partitioning – List Partitioning

HORIZONTAL PARTITIONING – LIST PARTITIONING

- List Partition Defined:
  - List Partition enables you to explicit control which rows map to partition tables by specifying a list of discrete VALUES of the Partitioning Key you identify.
  - In other words, you give it a list of value and rows will be targeted to a specific partition if they match the values on the list.
  - Example:
    - Assuming you have a table with a *sales_region* Column as Partition Key and you desire 3 partitions.
    - But you want then to be grouped or organized by state. Ex, East Sales Region partition you specify list of values such as New York, Virginia & Florida. The West Sales Region partition you specify list of values such as California, Oregon & Hawaii, etc.
    - See figure below



HORIZONTAL PARTITIONING – LIST PARTITIONING

- LIST Partition Usage:
  - Ideal method for situation where you want to group or organize unordered and unrelated sets of data into partitions.
  - List partitioning allows for partitions to reflect real-world groupings (e.g.. business units and territory regions).
  - List partitioning differs from range partition in that the groupings in list partitioning are not side-by-side or in a logical range.
  - List partitioning gives the DBA the ability to group together seemingly unrelated data into a specific partition

122 122

HORIZONTAL PARTITIONING – LIST PARTITIONING

How is done:

- The SQL CREATE TABLE Statement is extended using the PARTITION BY LIST, PARTITION & VALUE Clauses:

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  column_n datatype [ NULL | NOT NULL ]
)
PARTITION BY LIST(expr/partitionKey)
( PARTITION partition_name VALUES (value_list),
   ...
  PARTITION partition_name VALUES (value_list)
);
```

PARTITION BY LIST, PartitionKey

PARTITION, VALUES Clauses

HORIZONTAL PARTITIONING – LIST PARTITIONING

How is done (Cont.):

- The PARTITION BY LIST line is where the *partition key* is identified
- The PARTITION line is where each partition is explicitly named,
- The VALUES (*value_list*) – Contains the specific LIST or grouping of VALUES targeted for that partition table.
- Example:

```
CREATE TABLE customers
(
 name           VARCHAR2(50),
 email          VARCHAR2(100),
 state_code     CHAR(2)

)
PARTITION BY LIST (state_code)
( PARTITION region_south VALUES ('FL, AL, GA'),
  PARTITION region_north VALUES ('NY', MA, 'ND')
);
```

PARTITION BY LIST & PartitionKey = **state_code**

PARTITION & VALUES range FL, AL, GA, etc. Each partition will be grouped by each range.

HORIZONTAL PARTITIONING – LIST PARTITIONING

- List Partitioning Example 1 – Create sales Table that contains information on product sales & partition/group based on the list of channel_ID values indicated:
  - Creates a table & uses list portioning on the *channel_id* column and divides into 2 partitions or tables based on the discrite values listed

```
CREATE TABLE list_sales
   ( prod_id         NUMBER(6)
   , cust_id         NUMBER
   , time_id         DATE
   , channel_id      CHAR(1)
   , promo_id        NUMBER(6)
   , quantity_sold   NUMBER(3)
   , amount_sold     NUMBER(10,2)
   )
PARTITION BY LIST (channel_id)
 ( PARTITION even_channels VALUES ('2','4'),
   PARTITION odd_channels VALUES ('3','9')
 );
```


HORIZONTAL PARTITIONING – LIST PARTITIONING

- List Partitioning Example 1 (Cont.):
  - We assume the table currently has the following data:

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
35	2606	17-FEB-00	3	999	1	54.94
45	9491	28-AUG-98	4	350	1	47.45

HORIZONTAL PARTITIONING – LIST PARTITIONING

List Partitioning Example 1 (Cont.):

- The resultant tables are shown in figure below:
- Note the following results & process:
  - Names of the Partitions came from the PARTITION clause in the CREATE TABLE statement
  - The database chooses the rows for each partition based on the channel_id value specified in the PARTITION BY LIST clause & the values listed in the VALUES clause
  - Example:
    - Rows with a *channel_id* value of 2 or 4 are stored in the EVEN_CHANNELS partition table, while rows with a *channel_id* value of 3 or 9 are stored in the ODD_CHANNELS partition table

- **Rows are partitioned based on VALUES list of partition key *channel_id*.**
- **Rows with *channel_id* values of 2 & 4 only are placed here!**

Table Partition EVEN_CHANNELS

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
45	9491	28-AUG-98	4	350	1	47.45

Table Partition ODD_CHANNELS

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
35	2606	17-FEB-00	3	999	1	54.94

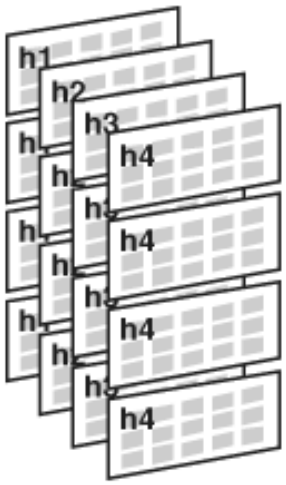
- **Rows with *channel_id* values of 3 & 9 only are placed here!**

Horizontal Partitioning (Partition by ROWS) – Composite Partitioning

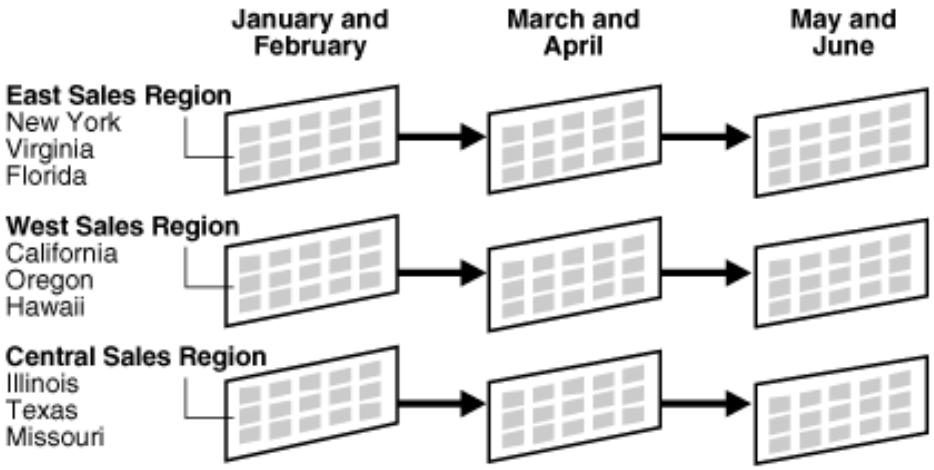
HORIZONTAL PARTITIONING – COMPOSITE PARTITIONING (ORACLE 11.G ONLY?)

- **Composite Partition Defined:**
  - Composite Partition is a combination of the other partition methods (*Range*, *Hash* & *List*)
  - First a table is partitioned by the partition methods *Range*, or *List only* and then each partition is further subdivided into sub-partitions using a second partition methods (*Range*, *Hash* or *List*) method
  - Figure below illustrates a Range-Hash and a Range-List composite Partitioning examples:

Composite Partitioning
Range-Hash



Composite Partitioning
Range - List



HORIZONTAL PARTITIONING – COMPOSITE PARTITIONING

- Composite Partition Usage:
  - Provides higher degree of finer granularity of partitioning approach.
  - Provides more degree of parallelism for Queries (Not one large table but several tables can be queried)
  - Good for historical operations (More clarity on this statement required. TBD)

VERTICAL PARTITIONING

PARTITIONING – VERTICAL PARTITIONING

- Vertical Partitioning Defined:
  - Distributing/moving the COLUMNS of a logical table into several separate physical tables
  - Each new partitioned table created have the same Primary Key but different columns.
  - In short, breaking up tables into smaller pieces by columns.
  - Example:
    - A table contains Stock information such as Stock Name, Description Stock Amount, Price & when it was Last Ordered.
    - Supposed usage of this table shows that 98% of users are searching for Stock Name, Description and Price. Nevertheless, 2% are searching for Stock Amount and Last Ordered.
    - Vertical Partition can be used to INCREASE PERFORMANCE by dividing the tables based on the COLUMNS as shown in figure below:

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Key	Name	Description	Price
ARC1	Arc welder	250 Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

PARTITIONING – VERTICAL PARTITIONING

- Vertical Partitioning Usage (When to partition?)
  - Situations where different users need access to different columns.
  - Very large tables where some columns are accessed more than others. Partition table by column to enhance performance.
  - Security - You want to hide certain data from certain users

137 137

PARTITIONING – VERTICAL PARTITIONING

- How is done:
  1. Use SQL CREATE TABLE command to break up the COLUMNS of a logical table into several separate physical tables
  2. Each new table created have the same PRIMARY KEY

  - We will NOT show any Examples of Vertical Partitioning.

138 138

FINAL SUMMARY – PROS/CONS OF PARTITIONING (HORIZONTAL & VERTICAL)

5.3.5 Physical Design Process – Designing Physical Database Files

- In this section we discuss another Physical Design concept: How a DBMS Manages the Physical Storage.

DBMS PHYSICAL STORAGE MANAGEMENT

- How does a DBMS Manages the Physical Storage Space in Which the Database Objects are Stored is a very important part of the Physical Design
  - In order to OPTIMIZE the PERFORMANCE of the database processing, the DB Admin needs to have thorough knowledge of how the DBMS manages the Physical Storage Space.
  - Physical Storage Management is very specific to a DBMS.
  - Nevertheless, we will cover the basic FOUNDATION for Physical Data Structures used by most DBMS

145 145

DESIGNING PHYSICAL DATABASE FILES

- Most DBMS store different kinds of data in an Operating System Physical File
- Physical File:
  - A part of secondary memory (Hard Disk), given a name & allocated for the purpose of storing physical records (tables, objects, etc.)
  - Tablespace (Oracle) – Logical storage unit in secondary memory (Hard Disk), given a name, and used to store data from multiple tables/views/objects
  - File Group (MS SQL) – Same definition as tablespace but for MS SQL Server

Chapter 5 146

PHYSICAL DATABASE FILES - TABLESPACE

- Tablespace (Oracle):
  - Tablespace – Logical storage unit in secondary memory (Hard Disk), given a name, and used to store data from multiple tables/views/objects
  - Oracle DBMS uses many tablespaces:
    - System Data
    - Temporary workspace
    - Undo operations
    - User business data
    - Etc.

PHYSICAL DATABASE FILES - TABLESPACE

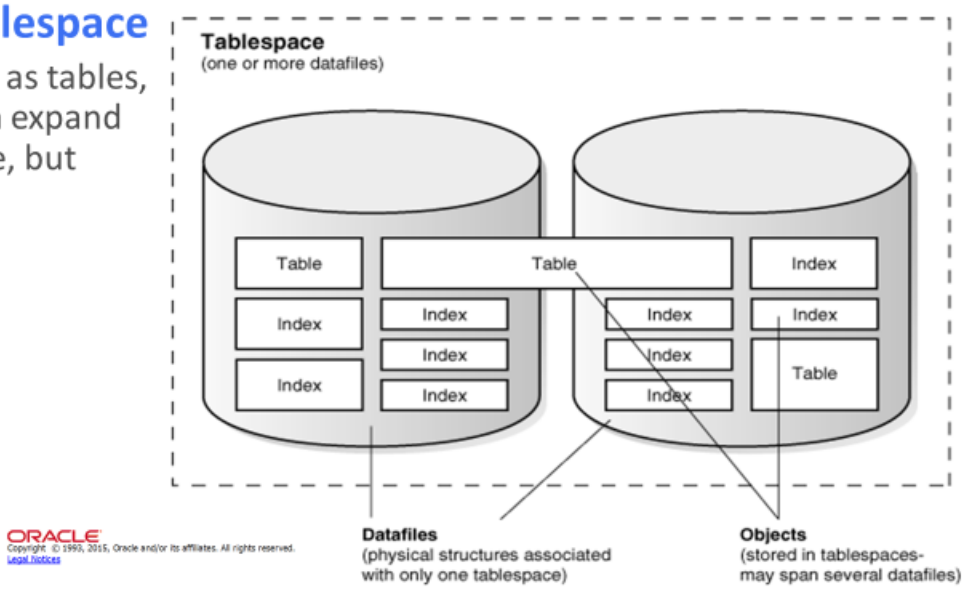
- Tablespace (Cont.):
  - A Tablespace Consists of ONE or MORE Physical Operating System Files.
  - Oracle DBMS is responsible for managing the storage of data inside the tablespace.
  - The Operating System manages the tablespace from the point of view of managing operating system files.

PHYSICAL DATABASE FILES - TABLESPACE

- Tablespace components:
  - Segment – a table, index, or partition
  - Extent – contiguous (together in a sequence) section of disk space
  - Data block – smallest unit of storage

Data Storage – Tablespace

- Illustration of a tablespace
  - Note data objects such as tables, indexes, views, etc. can expand more than one data file, but ONLY one tablespace



PHYSICAL DATABASE FILES - TABLESPACE

- Storing Objects in tablespace:
  - Each tablespace contains ONE or MORE Schema Object (tables, index, etc.)
  - Each Schema Object (tables, index, etc.) belongs to a SINGLE tablespace.
  - A schema object (tables, index, etc.) CANNOT belong to a TWO OR MORE tablespace, only ONE.

150 150

Data Storage – Segment & Extent

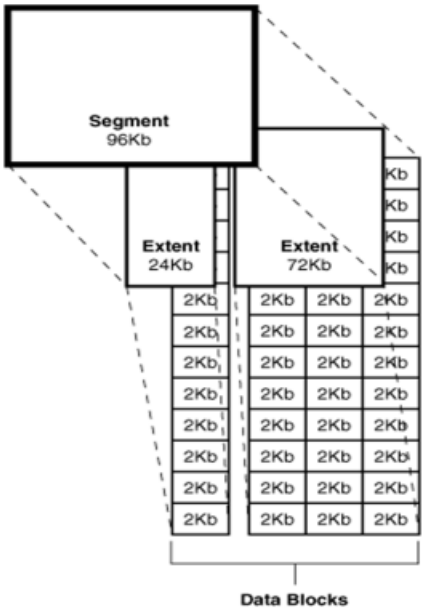
- Tablespace components
  - Segment
    - Logical units that make up a tablespace. A **tablespace** is divided into segments. Segments are divided into extents.
    - Segments are allocated to **specific type of information** stored in the same **tablespace**. For example, a table's data is store in one segment
    - Segments can expand multiple files
  - Extent
    - Segments are divided into contiguous (together in a sequence) section of disk space called extents.
    - Extents store **specific type of information**.
    - Extents belong to specific files and do not extend multiple files.
    - Extents are composed of data blocks

Data Storage – Tablespace

- **Tablespace components**
  - **Data block**
    - Smallest unit of storage within an extent (called logical blocks, oracle blocks or pages)
    - One data block corresponds to specific number of bytes of physical space on disk

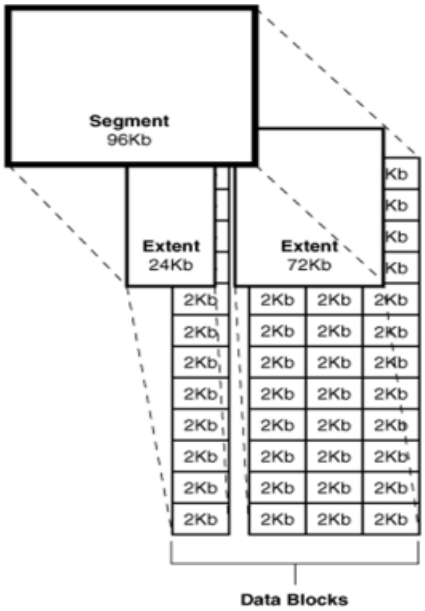
Data Storage – Tablespace

- **Illustration of a Segments, Extents & Data Blocks**
  - Figure shows one **segment** of 96kb
  - Segments can expand multiple files
  - The **segment** is divided into 2 **extents**. One of 24kb & one of 72kb.
  - Each extent is divided into **data blocks** or smallest unit that stores the data. Each data block is of 2kb size.



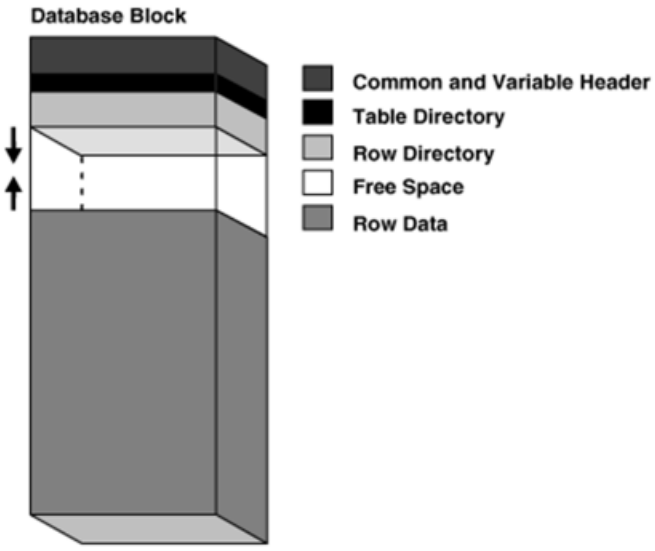
Data Storage – Tablespace

- **Space Allocation**
  - Oracle allocates space for segment in units of extents.
  - When extent is full, oracle allocates another extent.



Data Storage – Blocks

- **Illustration a Data Blocks**
  - Figure shows a **data blocks** is divided into several sections.
  - Some of header and directory information used by the system.
  - Some for free space to accommodate UPDATES or expansion of data
  - Section that contains the table's row data.



Data Storage – Blocks & Performance

- **Block Free Space Allocation**
  - DBMS allow DB Admin to specify Size of block & percent of free space in block:
    - Target is to have blocks full to store more rows, but must also have enough free-space for updates as needed.
    - Compromise is required between having block full and risk of having no free-space for updates.
    - Choice of datatype can help: ex. VARCHAR (variable length) results in more rows per block than CHAR
  - Oracle allows different block size for different tablespaces:
    - Selecting the right size of a block can help performance.

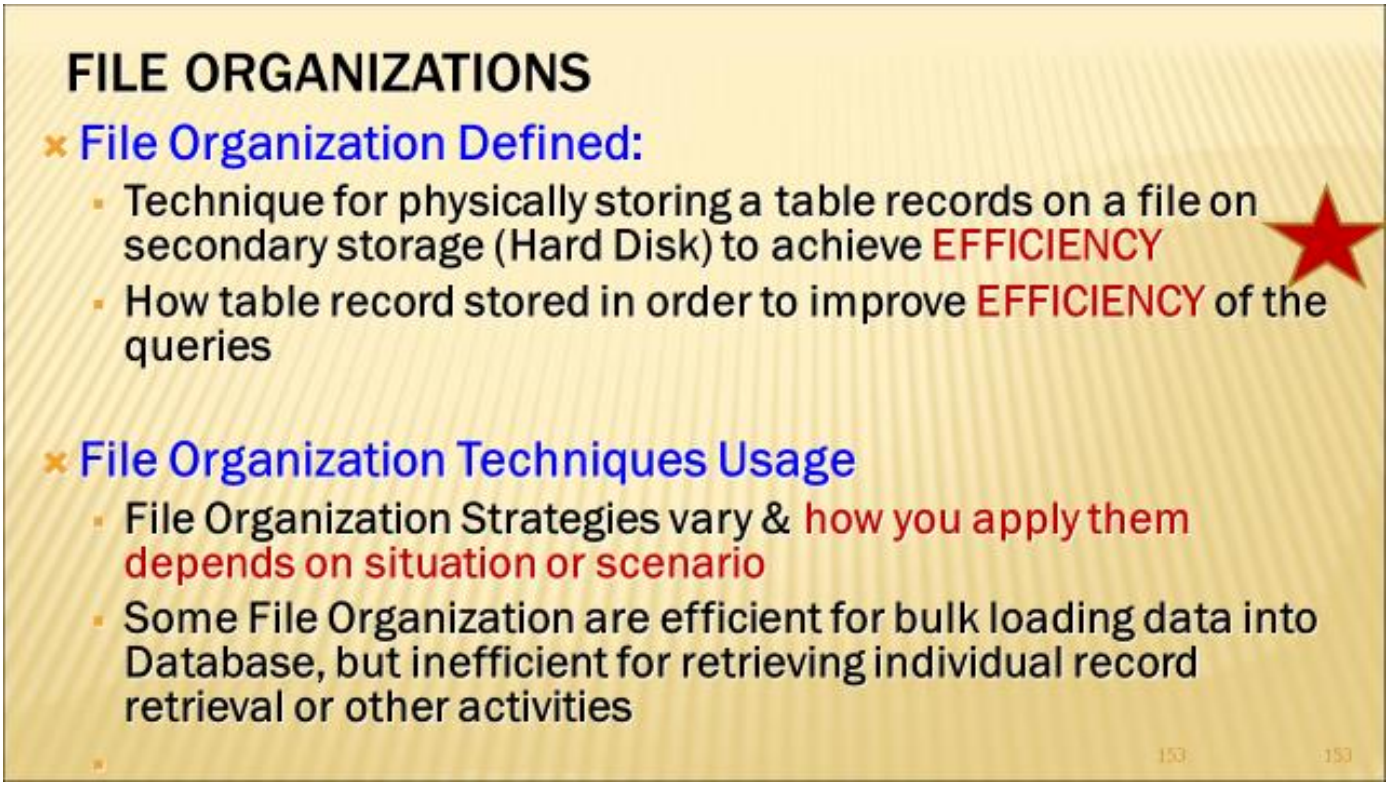
PHYSICAL DATABASE FILES - TABLESPACE

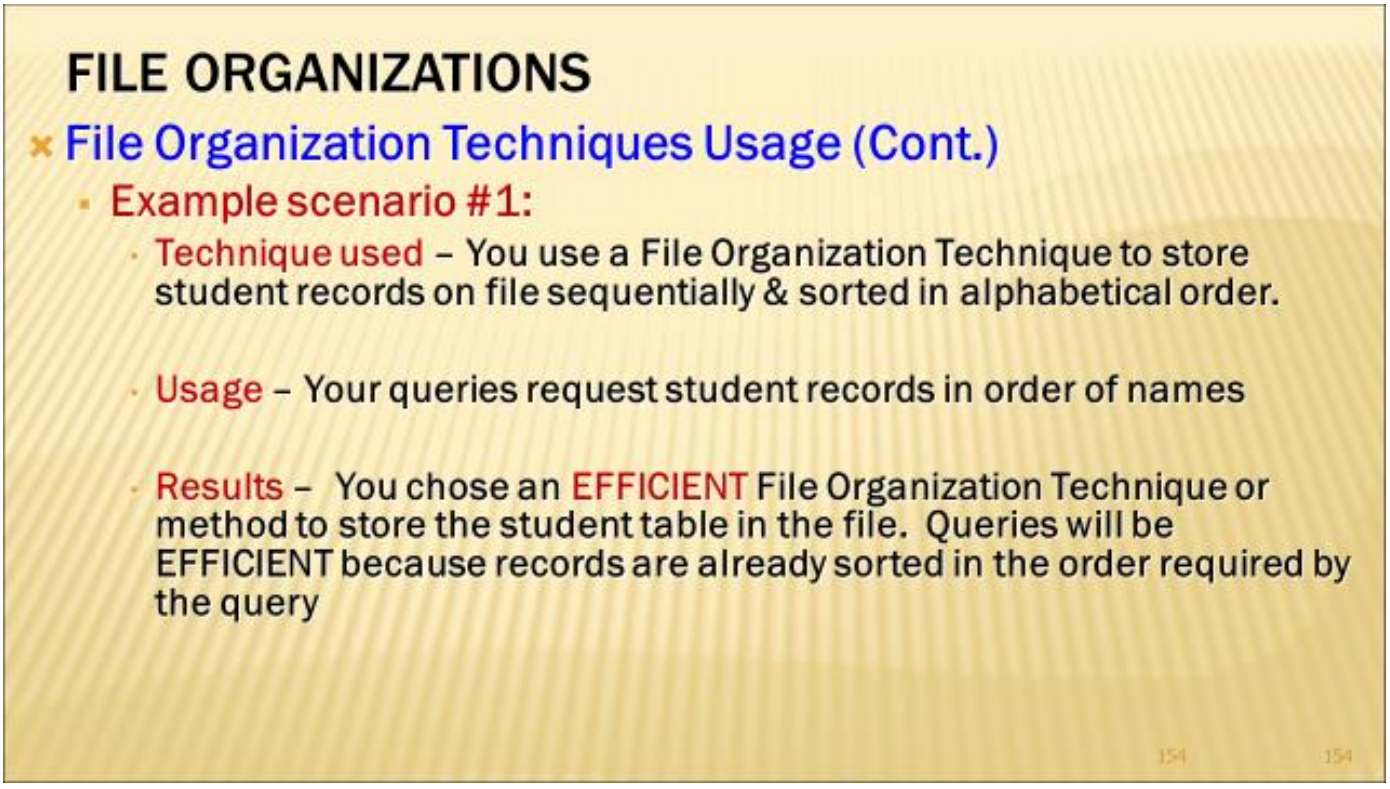
- Responsibility of Managing Tablespaces:
  - Responsibility matrix shown in table below:

Role	Responsibility
Oracle DBMS	▪ The use of the physical devices & files ▪ Manages the allocation of schema objects (tables, index, etc.)
DB Administrator	▪ Controls the size of the disk space allocated to tablespace ▪ Controls how free space is used

File Organization

File Organization Definition, Objectives, Usage & Approach





FILE ORGANIZATIONS

- File Organization Techniques Usage (Cont.)
  - Example scenario #2:
    - Technique used – You use a File Organization Technique to store student records on file sequentially & sorted in alphabetical order
    - Usage – Your queries request all students whose grades are within a certain range
    - Results – You chose an INEFFICIENT File Organization Technique or method to store the student table in the file. Queries will be INEFFICIENT because records are sorted and the grades are NOT stored in a convenient way for the query to be EFFICIENT

155

FILE ORGANIZATIONS – ROLE OF DBMS & DB ADMIN

- DBMS:
  - Manages the physical file
  - Manages how data is stored on the file
- DB Admin Objectives:
  - Selects how data is organized in the file.
  - Choose an OPTIMAL/EFFICIENT File Organization Technique for each table
  - Configure DBMS parameters for File Organization Technique or organizing the file

FILE ORGANIZATIONS – DB ADMIN & APPROACH

- DB Admin needs to choose a File Organization for particular files in a database.
- Factors DB Admin should consider for selecting a file organization scheme:
  - Fast data retrieval and throughput (Fast response and fast processing)
  - Efficient storage space utilization
  - Protection from failure and data loss
  - Minimizing need for reorganization (Do it right the first time)
  - Accommodating growth (*Make is scalable*)
  - Security from unauthorized use (Keep it out of the wrong hands)

157 157

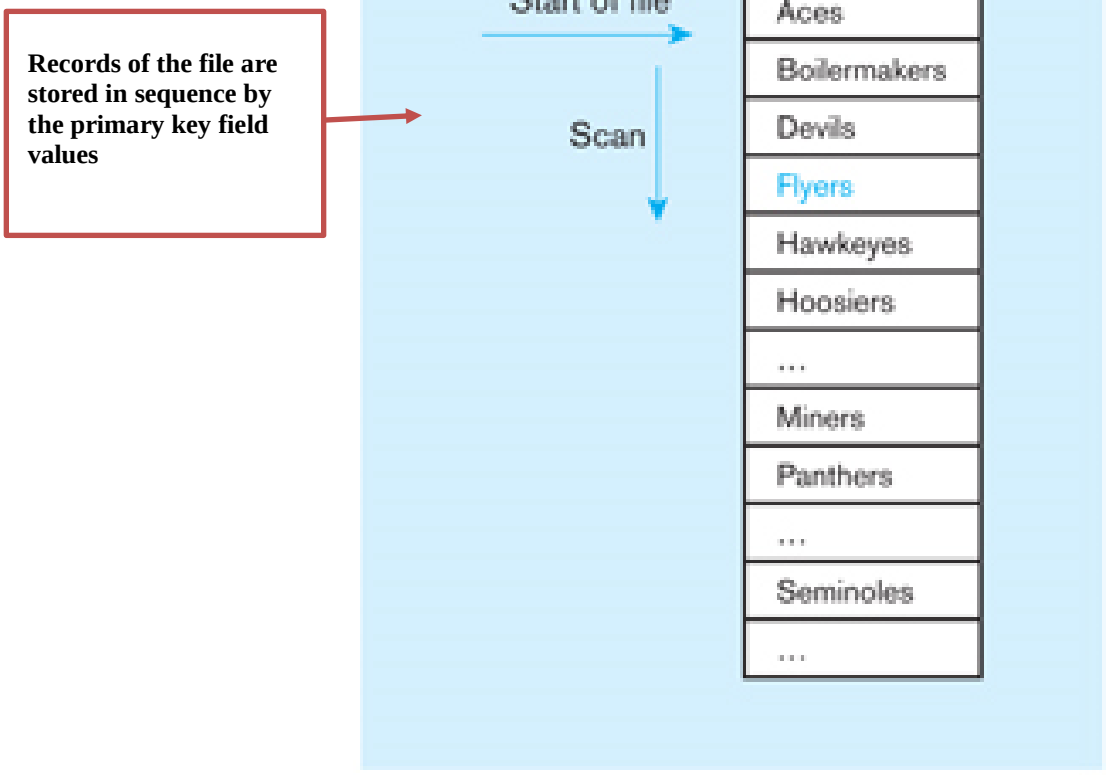
FILE ORGANIZATIONS

- Types of file organizations:
  - Sequential
  - Indexed
  - Hashed
  - Cluster
  - Others

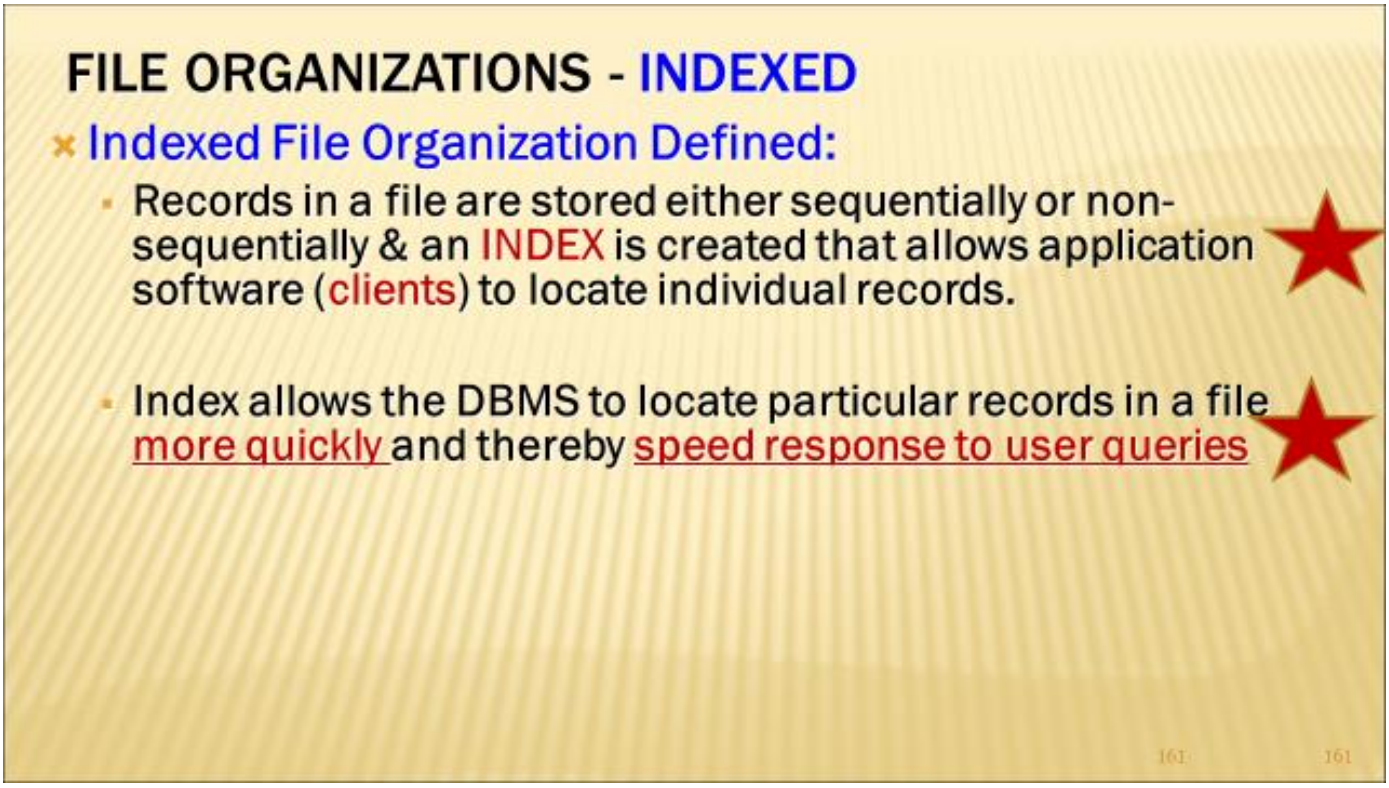
Sequential File Organization

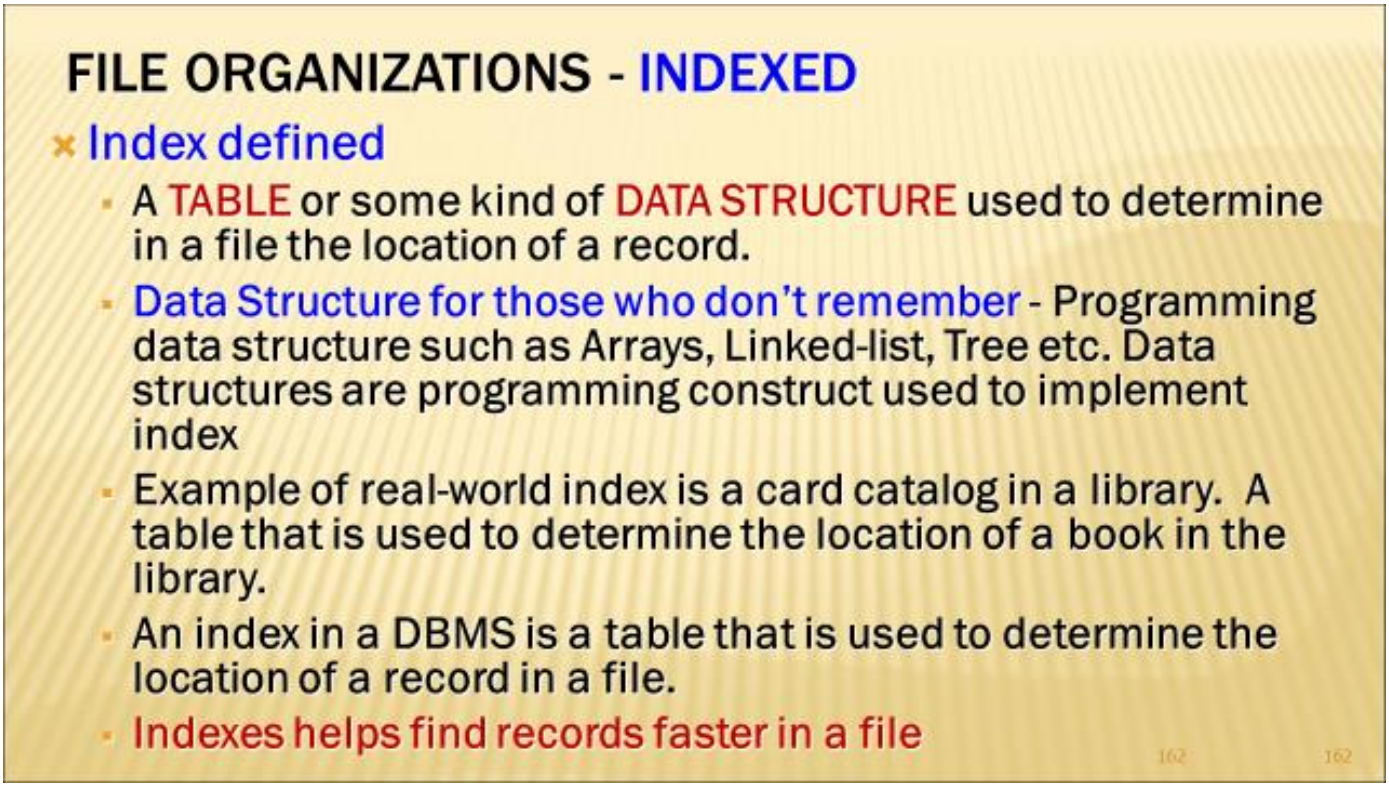
FILE ORGANIZATIONS - SEQUENTIAL

- Sequential File Organization Defined:
  - Records in a file are stored in a sequence according to Primary Key value.
- Characteristics:
  - Searching (scanning) & Updating – Need to scan or search from beginning until desired record is located.
  - Inserting & Deleting:
    - If list is sorted – every INSERT or DELETE requires a resorting of entire list
    - If list is NOT sorted – every INSERT or DELETE requires reorganization of file
  - See figure below:



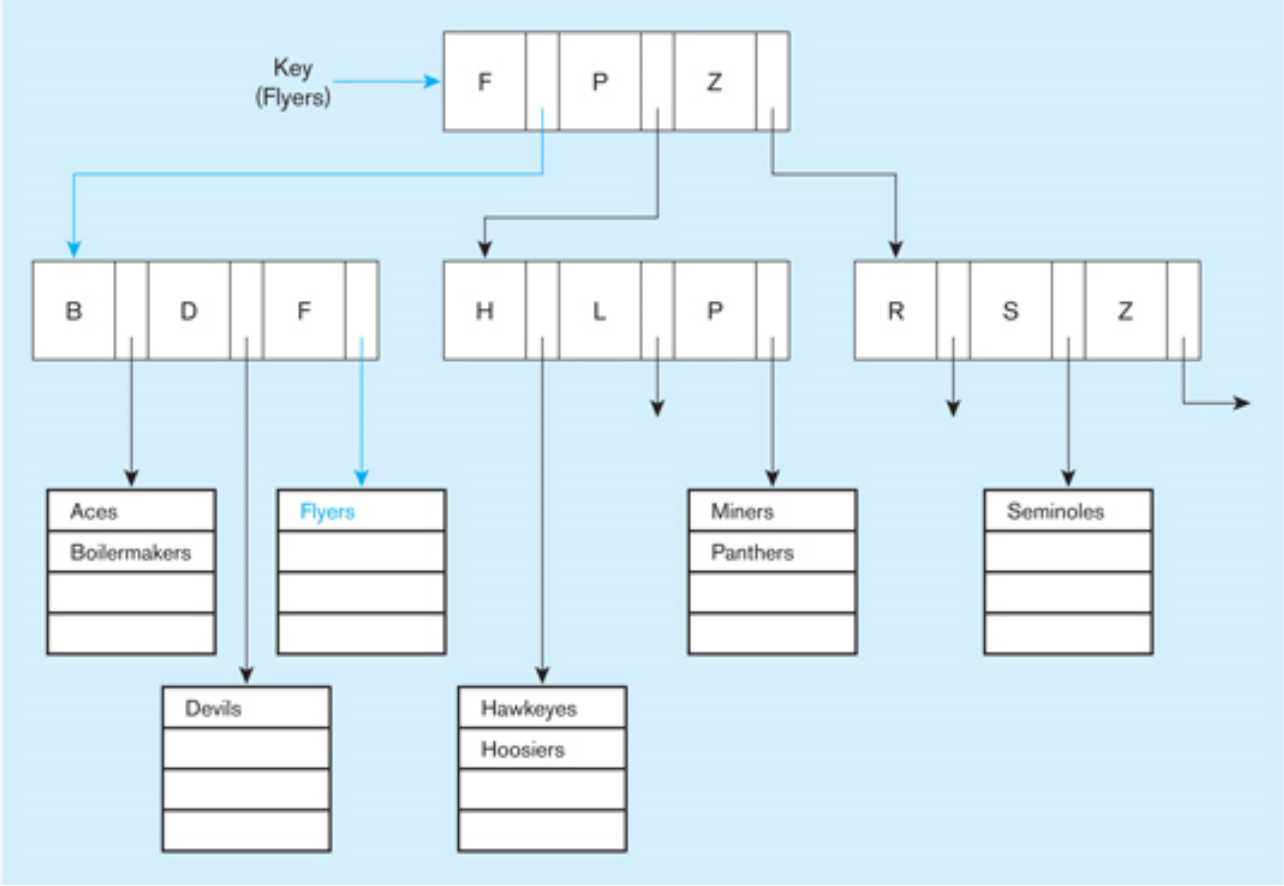
Indexed File Organization





FILE ORGANIZATIONS - INDEXED

- How it works
  - Index are static, created when the file is created
  - Each index entry matches a KEY VALUE with one or more records.
  - Primary Key Index – Index *points* to ONE UNIQUE record (ex. ProductID column of a Product record)
  - Secondary Key Index – Index *points* to MORE THAN ONE record
- See figure below:



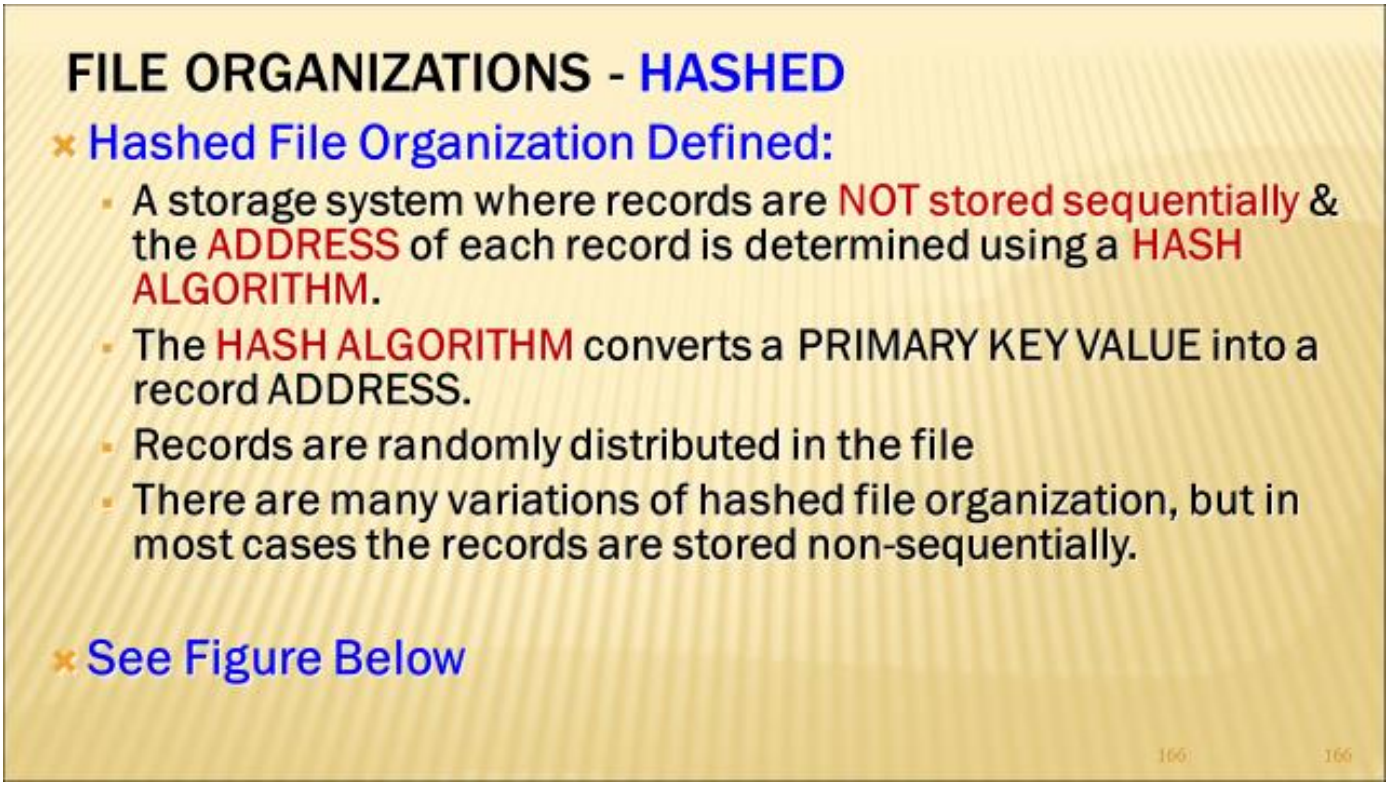
FILE ORGANIZATIONS - INDEXED

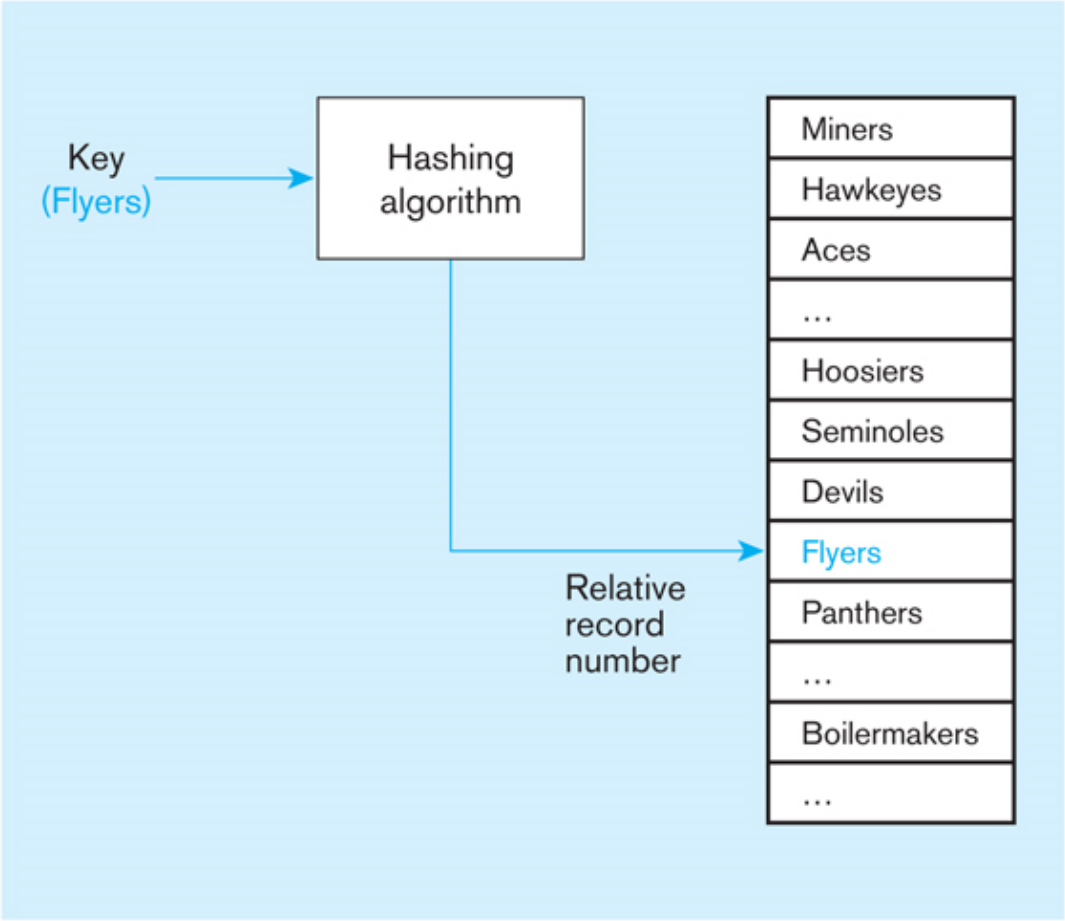
- When to use Index (Usage)
  - Searching/Retrievals on exact key match
  - Pattern Matching
  - Range of values
  - Etc.
- When NOT to use Index (Usage)
  - When performing lots of UPDATES to tables:
    - Since index are static & created when the file is created, many updates will cause the file to lose the key sequences of the indices, thus deteriorating the performance.

FILE ORGANIZATIONS - INDEXED

- Indexes are very important and extensively used in DBMSs
  - The choice of what type of index & how to store the index entries matter greatly database PROCESSING PERFORMANCE.
- While indexes are not strictly necessary to use the DBMS, they can have a significant impact on performance

Hashed File Organization




Key
(Flyers)
Hashing algorithm
Relative record number
Miners
Hawkeyes
Aces
...
Hoosiers
Seminoles
Devils
Flyers
Panthers
...
Boilermakers
...

FILE ORGANIZATIONS - HASHED

- When to use Hash (Usage)
  - Searching/Retrievals on exact match of Hash column if access is random:
    - Example if Student table is hashed on the Name column, then retrieval queries on name such as "Joe Smith" will be EFFICIENT.
- When NOT to use Hash (Usage)
  - Searching/Retrievals are based on range of values for the Hash column. Ex. if Student table is hashed on the Name column & retrieval is on students whose name being with "R"
  - Searching/Retrievals are based on other column other than the Hash column. Ex. if Student table is hashed on the Name column & retrieval is on student Address column.

Comparison of Different File Organization Schemes covered up to this point

TABLE 5-3 Comparative Features of Different File Organizations

	File Organization		
Factor	**Sequential**	**Indexed**	**Hashed**
Storage space	No wasted space	No wasted space for data but extra space for index	Extra space may be needed to allow for addition and deletion of records after the initial set of records is loaded
Sequential retrieval on primary key	Very fast	Moderately fast	Impractical, unless using a hash index
Random retrieval on primary key	Impractical	Moderately fast	Very fast
Multiple-key retrieval	Possible but requires scanning whole file	Very fast with multiple indexes	Not possible unless using a hash index
Deleting records	Can create wasted space or require reorganizing	If space can be dynamically allocated, this is easy but requires maintenance of indexes	Very easy
Adding new records	Requires rewriting a file	If space can be dynamically allocated, this is easy but requires maintenance of indexes	Very easy, but multiple keys with the same address require extra work
Updating records	Usually requires rewriting a file	Easy but requires maintenance of indexes	Very easy

FILE ORGANIZATIONS - CLUSTERING

- Some relational DBMSs such as Oracle support Clustered Tables
- Clustering Defined
  - A group of one or more tables physically stored together because they share common columns and are often used together
  - Related records (rows) from different tables can be stored together in the same disk area
  - With related records being physically stored together, disk access time is improved
- Useful for improving performance of join operations

174 174

FILE ORGANIZATIONS - CLUSTERING

- How it Works
  - A group of tables are related based on typical Primary Key in one table is a Foreign Key in the other table.
  - Both table share the same column in common (Primary Key & Foreign Key)
  - These tables can be clustered or stored together in same disk area for fast access & performance
- See Example Below

175 175

- Tables **EMP TABLE** & **DEPT TABLE** are ***RELATED*** by the Primary Key & Foreign Key **DEPTNO**.
- The **DEPTNO** column is ***COMMON*** to both tables.

EMP TABLE

EMPNO	ENAME	JOB	SAL	CDMM	MGRNO	DEPTNO
1000	Ajay	L	10000	1200	1002	10
1001	Rahat	M	12000	1500		10
1009	Amit	S	9000	1000	1001	10
1002	Raja	L	11000	1200	1004	20
1003	Rahil	M	13000	1300		20
1004	Deep	L	13000	500	1007	30
1007	Ram	S	14000	1500		30

DEPT TABLE

DEPTNO	DNAME	LOCATION
10	ACC	QADIAN
20	SALE	PATILA
30	PER	DELHI

- The **EMP TABLE** & **DEPT TABLE** tables can be both stored CLUSTERED or next to each other on disk based on the column **DEPTNO**.
- When these two tables are clustered, each unique **DEPTNO** value is stored only once, in the cluster key
- Each **DEPTNO** value are attached the column from both these tables

Dname	Location	Deptno	Empno	Ename	Job	Sal	Comm	MgrNo
ACC	QADIAN	10	1000	AJAY	L	10000	1200	1002
			1001	RAHAT	M	12000	1500	
			1009	AMIT	S	9000	1000	1001
SALE	PATIALA	20	1002	RAJA	L	11000	1200	1004
			1003	RAHIL	M	13000	1300	
PER	DELHI	30	1004	DEEP	L	13000	500	1007
			1007	RAM	S	14000	1500	

Emp Table

Dept Table

Cluster key

Emp and Dept table would be stored clustered on Deptno

FILE ORGANIZATIONS - CLUSTERING

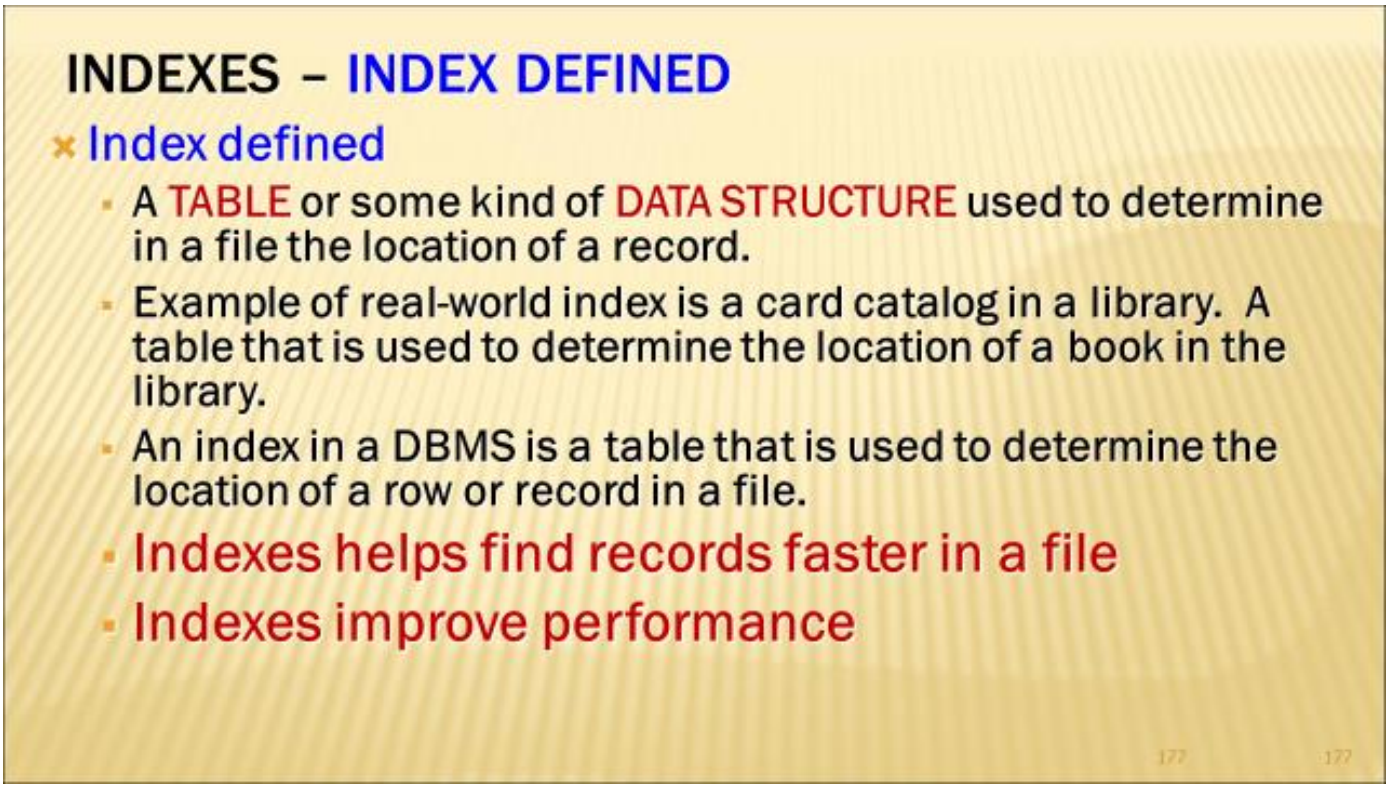
- When to use Clustering
  - When tables are *OFTEN* accessed in JOIN statements
  - When tables have a One-to-Many (1:M) relationship (Often access one row from parent table & corresponding rows from child table)
- When NOT to use Clustering
  - When tables are accessed in JOIN statements but *NOT OFTEN*
  - When common column (Cluster Key) is *OFTEN UPDATED* (Modifying a Clustered Column or common key can takes longer than un-clustered tables)
  - When one of the table *OFTEN* requires a full search (A full search of a clustered table can take longer than a full search of an un-clustered table)

175 175

5.3.6 Physical Design Process – Using and Selecting Indexes (Index Revisited)

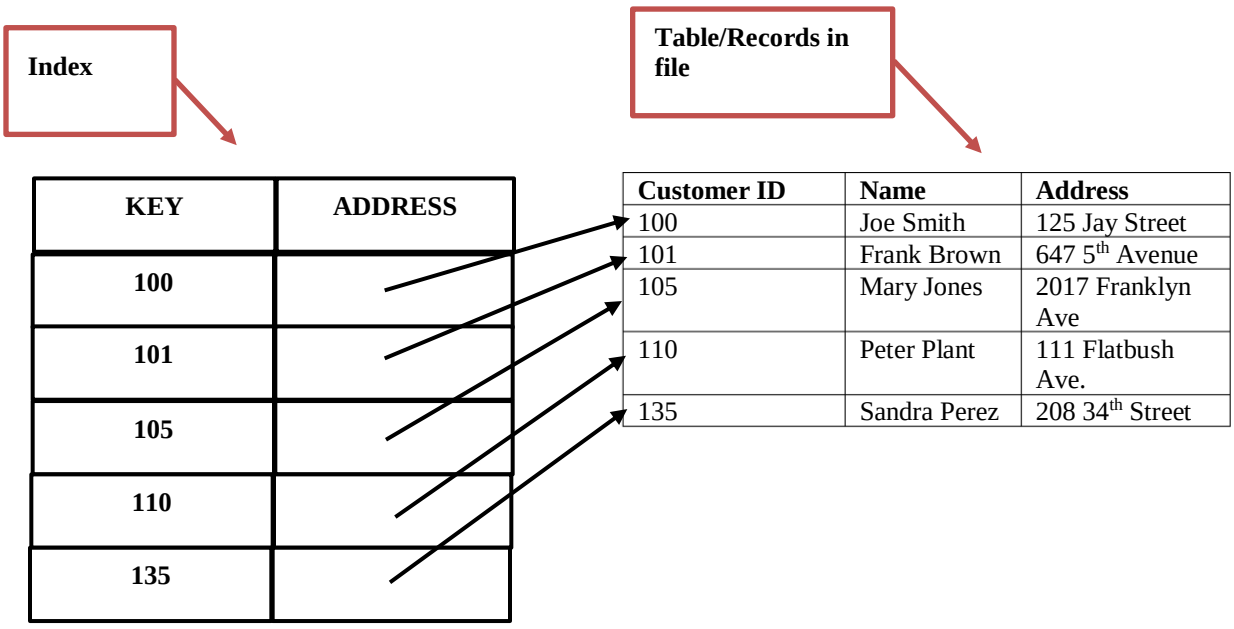
- In this section we discuss in details what indexes are and how they improve performance.

Indexes Defined



INDEXES – INDEX HOW IT WORKS

- How it works
  - Index is a table that has two columns:
    - Column 1 = Key
    - Column 2 = Address or POINTER to one or more records in the file. This is the VALUE or record (row) associated with the key
  - Index are static. They are created when the file that stores the table is created
  - Important – we logically use a table to represent an index, but how this index is actually physically implemented in a DBMS varies. Various data structures are used such as B-Tree etc. This is out of scope of this course.
  - See figures below for example of logical index table :



INDEXES – INDEX DEFINED

- Index Type – There are two types:
  - Primary Key Index:
    - Index created on the Primary Key of a table.
    - Index table has two columns, Primary Key & Address or POINTER to the record.
    - Index *points* to ONE UNIQUE record (ex. CustomerID column of a Customer record)
  - Secondary Key Index:
    - Index on column other than Primary Key
    - Index can *point* to MORE THAN ONE record

INDEXES – INDEX USAGE

- How do you decide which index to create (Primary or Secondary)?
  - Depends on the Queries:
    - DBA needs to analyze the set of queries being applied the tables.
    - Depending on queries, either Primary or Secondary indexes need to be created.
    - WHERE clause & JOIN portion of a queries are indicators
  - Guidelines on identifying which index to create (primary or secondary) by analyzing the Queries:
    - Queries with Primary Key in WHERE clause – If the search is based on Primary key in where clause use PRIMARY KEY INDEX
    - Queries with non-key attributes in WHERE clause – use SECONDARY INDEX.
    - Other criteria such as JOIN, GROUP BY etc.

180 180

INDEXES – INDEX USAGE

- Index creation DOES NOT affect the queries applied to a database by the client application
  - Index creation DOES NOT affect the queries applied to a database
  - Indexes improve performance when you create them, but they are transparent to the queries being applied to the database.
  - The client applications that are submitting queries to the DBMS are not affected.
  - This is an advantages to using indexes

181 181

INDEXES – INDEX USAGE

- Search on a condition is a common database operation
  - Most database operations or queries (search), target one or more records (rows) based on a condition.
  - Examples:

```
/* query that returns
   one row(record)*/
SELECT *
FROM Customer
WHERE Customer_ID = '111';
```

```
/* query that returns one
   or more rows(records)*/
SELECT *
FROM Student
WHERE Major = 'Math';
```

INDEXES – INDEX USAGE

- Customer query example
  - Without index you would have to scan the entire Customer table until you find the record whose Customer_ID = '111'.
  - If the Customer table was sorted, this may help get a faster response, otherwise there is a performance hit.
  - With an index on the primary key or Customer_ID the row being searched can be found in the file much faster.
- Student query example
  - Without index you would have to scan the entire Student table until you find the records whose Major match 'Math'.
  - With a secondary index on the Major column, the search can be greatly improved.

INDEXES – WHY USE INDEXES

- Why use indexes?
  - Improve performance by reducing full table scans.
  - Reduces disk I/O
  - Queries completion times can go from hours to minutes or from minutes to milliseconds.
  - Allows database to search a smaller index table rather than a large table
  - A FASTER PATH TO DATA!!!!
  - Also, helps enforce Referential Integrity between tables that are related by primary key & foreign. By placing an index on primary key of parent table, the database can use the index to validate the child during inserts, updates and delete operations.

Syntax to Creating Unique Primary Key Index

- **PRIMARY KEY INDEX** Syntax:

```
CREATE UNIQUE INDEX IndexName ON Table(Column);
```

  - Where:
    - **CREATE UNIQUE INDEX** – keyword indicating the creation of a primary unique index
    - ***IndexName* ON –** Name given to index
    - ***Table(Column)* –** table & PRIMARY KEY column index is being applied to

- Example:

```
CREATE UNIQUE INDEX CustIndex_PK ON Customer_T(CustomerID);
```

  - Where:
    - **CREATE UNIQUE INDEX** – keyword indicating the creation of a primary unique index
    - ***CustIndex_PK* ON –** Name given to index
    - ***Customer_T(CustomerID)* –** Index being applied to **Customer_T** table and **CustomerID** column

- **COMPOSITE PRIMARY KEY INDEX** Syntax:

```
CREATE UNIQUE INDEX IndexName ON Table(Column1, Column2, etc.);
```

  - Where:
    - **CREATE UNIQUE INDEX** – keyword indicating the creation of a primary unique index
    - ***IndexName* ON –** Name given to index
    - ***Table(Column1, Column2, etc.)* –** table & list of composite PRIMARY KEY column index is being applied to

- Example:

```
CREATE UNIQUE INDEX LineIndex_PK ON OrderLine_T(OrderID, ProductID);
```

  - Where:
    - **CREATE UNIQUE INDEX** – keyword indicating the creation of a primary unique index
    - ***LineIndex_PK* ON –** Name given to index
    - ***OrderLine_T(OrderID, ProductID)* –** Index being applied to **OrderLine_T** table and composite key made out of two columns **OrderID** & **ProdcutID**.

Creating Secondary Index

- **SECONDARY INDEX** Syntax – Same syntax just use a COLUMN other than PRIMARY KEY:

CREATE INDEX *IndexName* **ON** *Table(Non-key Column);*

  - Where:
    - **CREATE INDEX** – keyword indicating the creation of a primary unique index
    - ***IndexName* ON –** Name given to index
    - ***Table(Non-key Column)* –** table & NON-PRIMARY KEY column index is being applied to

- Example:

Database users often want to retrieve rows of a relation based on values for various attributes other than the primary key. For example, in a Product table, users might want to retrieve records that satisfy any combination of the following conditions:

- All table products (Description = "Table")
- All oak furniture (ProductFinish = "Oak")
- All dining room furniture (Room = "DR")
- All furniture priced below $500 (Price < 500)

To speed up such retrievals, we can define an index on each attribute that we use to qualify a retrieval. For example, we could create a nonunique index on the Description field of the Product table with the following SQL command:

  - Index created:

```
CREATE INDEX DescIndex_FK ON Product_T(Description);
```

  - Where:
    - **CREATE UNIQUE INDEX** – keyword indicating the creation of a primary unique index
    - ***DescIndex_FK* ON –** Name given to index
    - ***Product_T(Description)* –** Index being applied to **Product_T** table and **Description** column

Rules for Using and Selecting Indexes

INDEXES – GUIDELINES FOR USING INDEXES

- General guidelines for creating indexes
  - In Physical Design you will need to decide on tables & attributes to create index on.
  - Indexes are good for retrieval queries (SELECT), but not UPDATE, INSERT & DELETE queries.
  - Thus generously targeted for databases where queries are focused on retrieval such as DECISION SUPPORT (reporting etc.) & DATAWAREHOUSE.
  - Should be used cautiously on databases that that support transaction processing and other applications with heavy UPDATING requirements, because the indexes impose additional overhead on updates, inserts etc.

INDEXES – RULES FOR USING INDEXES

1. Use on larger tables
2. Create index for the primary key of each table
3. Create index for search columns frequently in WHERE clauses of SQL commands
   - WHERE clause to qualify the rows to select (e.g., WHERE ProductFinish = "Oak," Creating a Secondary Index on ProductFinish would speed retrieval
   - WHERE clause to link (join) tables (e.g., WHERE Product_T.ProductiD = OrderLine_T.ProduclID (Note that in this JOIN, Product_T.ProductID column is primary key to Product_T table & OrderLine_T.ProductID column is a foreign key in the OrderLine table). Thus creating a Primary Key Index on Product_T table & Secondary Index on OrderLine_T table would improve retrieval performance in this join.

INDEXES – RULES FOR USING INDEXES (CONT.)

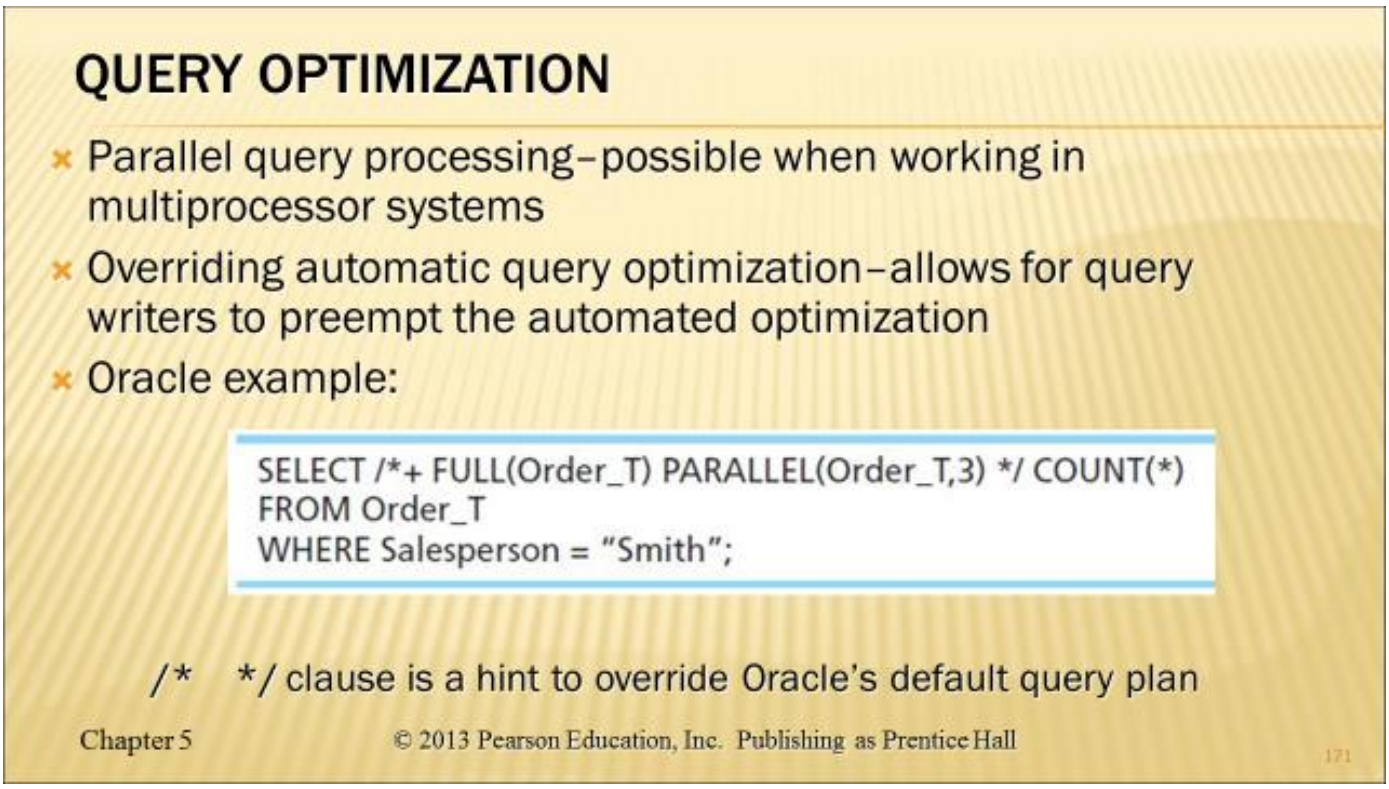
4. Create index in SQL commands with columns in ORDER BY (sorting) and GROUP BY (categorizing) clauses
   - Need to be careful since not all DBMS support these clauses, e.g., Oracle supports indexes on ORDER BY clauses BUT NOT GROUP BY
5. Once other criteria have been applied, create index if the column contains a significant variety of values
   - Oracle suggests that index is NOT useful when there are fewer than 30 different values for an attribute & useful when there are 100 or more different values for an attribute
   - Rule: When different values are >100 but not when there are <30 values

INDEXES – RULES FOR USING INDEXES (CONT.)

6. Avoid use of indexes for columns with long values; perhaps compress values first
   - Indexes created from long values can be slower to process.
   - If columns contains long values, compress by using coding technique (use a code instead of long values & create look-up table).
   - Apply index on code column.
7. If key to index is used to determine location of record, use surrogate (like sequence number/auto-number) to allow even spread in storage area
   - My understanding here is that if key is composite, you can use a surrogate key instead.
   - And use a sequence or auto-number for surrogate key so data is spread evenly in storage area.

INDEXES – RULES FOR USING INDEXES (CONT.)

8. DBMS may have limit on number of indexes you can create per table and the length of the value (number of bytes per indexed columns)
   - Some DBMS permit no more than 16 indexes per table.
   - Some DBMS limit the size of an index key value to no more than 2000 bytes.
   - If such a limit exists in your DBMS, then you may have to choose a Secondary index instead.
9. Be careful of indexing attributes with null values; many DBMSs will not recognize null values in an index search
   - If search requires searching columns with NULL values, then you cannot use index and full scan of table will be required.


QUERY OPTIMIZATION
Parallel query processing–possible when working in multiprocessor systems
Overriding automatic query optimization–allows for query writers to preempt the automated optimization
Oracle example:
SELECT /*+ FULL(Order_T) PARALLEL(Order_T,3) */ COUNT(*)
FROM Order_T
WHERE Salesperson = "Smith";
/* */ clause is a hint to override Oracle's default query plan
Chapter 5
© 2013 Pearson Education, Inc. Publishing as Prentice Hall
171

5.3.7 Book's Examples & Exercises

- Some examples from the book.

Exercise #1 (Page 233 & 234)

1. Consider the following two relations for Millennium College:

STUDENT(StudentID, StudentName, CampusAddress, GPA)
REGISTRATION(StudentID, CourseID, Grade)

Following is a typical query against these relations:

```
SELECT Student_T.StudentID, StudentName,
   CourseID, Grade
FROM Student_T, Registration_T
     WHERE Student_T.StudentID =
        Registration_T.StudentID
     AND GPA > 3.0
ORDER BY StudentName;
```

a. On what attributes should indexes be defined to speed up this query? Give the reasons for each attribute selected.
b. Write SQL commands to create indexes for each attribute you identified in part a.

Books Answers & my comments/modifications:

- **Question A – Attributes targeted for indexes:**
  - **STUDENT TABLE:**

1. **StudentID** in STUDENT because it is a primary key and the index would enforce uniqueness of the key StudentID in STUDENT

2. **StudentID** in REGISTRATION since used in a WHERE clause for joining the STUDENT and REGISTRATION tables, so it likely makes sense to create an index on **StudentID** in REGISTRATION as well.

3. **GPA** in STUDENT because it is a nonkey cluster attribute used to qualify record retrieval

   - **NOTE** - Note that I am not sure what author means by cluster attribute. I could not find any reference to this term, nevertheless, I assume he means is clustered as part of the STUDENT table

4. **StudentName** in STUDENT because it is a nonkey attribute used to sort records

- **REGISTRATION TABLE:**

5. **StudentID**, **CourseID** in REGISTRATION because it is a concatenated **composite** primary key and the index would enforce uniqueness of the key

- **Question B – SQL commands for indexes:**

  - **STUDENT TABLE:**

1. **StudentID** in STUDENT because it is a primary key and the index would enforce uniqueness of the key StudentID in STUDENT

CREATE UNIQUE INDEX *STUPKINDX* ON STUDENT (StudentID);

2. **StudentID** in REGISTRATION since used in a WHERE clause for joining the STUDENT and REGISTRATION tables, so it likely makes sense to create an index on **StudentID** in REGISTRATION as well.

CREATE INDEX *STUDREGINDX* ON REGISTRATION (StudentID);

3. **GPA** in STUDENT because it is a nonkey cluster attribute used to qualify record retrieval

   - **NOTE** - Note that I am not sure what author means by cluster attribute. I could not find any reference to this term, nevertheless, I assume he means is clustered as part of the STUDENT table

CREATE INDEX *CLUST_INDX* ON STUDENT (GPA) CLUSTER;

   - **INPORTANT NOTE** – I am not sure where the author got this syntax from. This syntax implies a special syntax for cluster attributes with this keyword **CLUSTER**. But no reference is made on cluster attributes or this syntax in the chapter. Could this be a type-o? We will keep this in mind and look for any explanation in future notes or CST3604.

4. **StudentName** in STUDENT because it is a nonkey attribute used to sort records

CREATE INDEX *NAMEINDX* ON STUDENT (StudentName);

- **REGISTRATION TABLE:**

5. **StudentID**, **CourseID** in REGISTRATION because it is a concatenated **composite** primary key and the index would enforce uniqueness of the key

CREATE UNIQUE INDEX *REGSPKINDX* ON REGISTRATION (StudentID, CourseID);

Exercise #2 (Page 233 & 234)

2. Choose Oracle data types for the attributes in the normalized relations in Figure 5-4b.

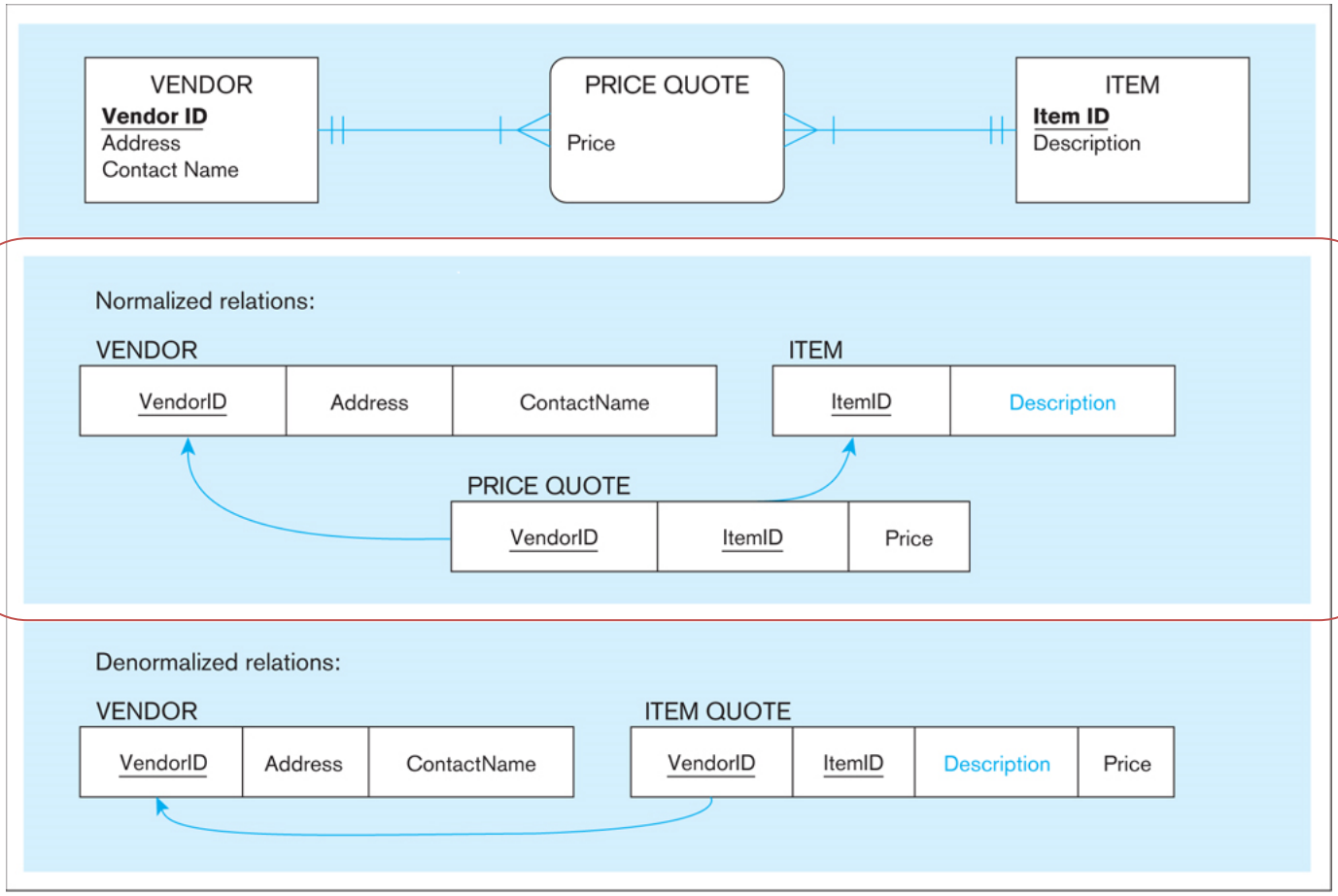


TABLE 5-1 Commonly Used Data Types in Oracle 11g

Data Type	Description
VARCHAR2	Variable-length character data with a maximum length of 4,000 characters; you must enter a maximum field length [e.g., VARCHAR2(30) specifies a field with a maximum length of 30 characters]. A string that is shorter than the maximum will consume only the required space.
CHAR	Fixed-length character data with a maximum length of 2,000 characters; default length is 1 character (e.g., CHAR(5) specifies a field with a fixed length of 5 characters, capable of holding a value from 0 to 5 characters long).
CLOB	Character large object, capable of storing up to (4 gigabytes – 1) * (database block size) of one variable-length character data field (e.g., to hold a medical instruction or a customer comment).
NUMBER	Positive or negative number in the range 10^{-130} to 10^{126}; can specify the precision (total number of digits to the left and right of the decimal point) and the scale (the number of digits to the right of the decimal point). For example, NUMBER(5) specifies an integer field with a maximum of 5 digits, and NUMBER(5,2) specifies a field with no more than 5 digits and exactly 2 digits to the right of the decimal point.
DATE	Any date from January 1, 4712 B.C., to December 31, 9999 A.D.; DATE stores the century, year, month, day, hour, minute, and second.
BLOB	Binary large object, capable of storing up to (4 gigabytes – 1) * (database block size) of binary data (e.g., a photograph or sound clip).

Books Answers & my comments/modifications:

Suggested Oracle data types for Figure 4b:

VendorID	INTEGER
Address	VARCHAR2(40)
ContactName	VARCHAR2(25)
ItemID	INTEGER
Description	VARCHAR2(35)
Price	NUMBER(5,2)

INTEGER is just an alias for NUMBER(38).

Appendix

Oracle/PLSQL: Data Types

The following is a list of datatypes available in Oracle/PLSQL, which includes character, numeric, date/time, LOB and rowid datatypes.

Character Datatypes

The following are the **Character Datatypes** in Oracle/PLSQL:

Data Type Syntax	Oracle 11g	Explanation
char(size)	Maximum size of 2000 bytes.	Where ***size*** is the number of characters to store. Fixed-length strings. Space padded.
nchar(size)	Maximum size of 2000 bytes.	Where ***size*** is the number of characters to store. Fixed-length NLS string Space padded.
nvarchar2(size)	Maximum size of 4000 bytes.	Where ***size*** is the number of characters to store. Variable-length NLS string.
varchar2(size)	Maximum size of 4000 bytes. Maximum size of 32KB in PLSQL.	Where ***size*** is the number of characters to store. Variable-length string.
long	Maximum size of 2GB.	Variable-length strings. (backward compatible)
raw	Maximum size of 2000 bytes.	Variable-length binary strings
long raw	Maximum size of 2GB.	Variable-length binary strings. (backward compatible)

Numeric Datatypes

The following are the **Numeric Datatypes** in Oracle/PLSQL:

Data Type Syntax	Oracle 11g	Explanation
number(p,s)	Precision can range from 1 to 38. Scale can range from -84 to 127.	Where ***p*** is the precision and ***s*** is the scale. For example, number(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.
numeric(p,s)	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, numeric(7,2) is a number that has 5 digits before the decimal and 2 digits after the decimal.
float		
dec(p,s)	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, dec(3,1) is a number that has 2 digits before the decimal and 1 digit after the decimal.
decimal(p,s)	Precision can range from 1 to 38.	Where ***p*** is the precision and ***s*** is the scale. For example, decimal(3,1) is a number that has 2 digits before the decimal and 1 digit after the decimal.
integer		
int		
smallint		
real		
double precision		

Date/Time Datatypes

The following are the **Date/Time Datatypes** in Oracle/PLSQL:

Data Type Syntax	Oracle 11g	Explanation
date	A date between Jan 1, 4712 BC and Dec 31, 9999 AD.	
timestamp (*fractional seconds precision*)	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	Includes year, month, day, hour, minute, and seconds. For example: timestamp(6)
timestamp (*fractional seconds precision*) with time zone	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	Includes year, month, day, hour, minute, and seconds; with a time zone displacement value. For example: timestamp(5) with time zone
timestamp (*fractional seconds precision*) with local time zone	***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	Includes year, month, day, hour, minute, and seconds; with a time zone expressed as the session time zone. For example: timestamp(4) with local time zone
interval year (*year precision*) to month	***year precision*** is the number of digits in the year. (default is 2)	Time period stored in years and months. For example: interval year(4) to month
interval day (*day precision*) to second (*fractional seconds precision*)	***day precision*** must be a number between 0 and 9. (default is 2) ***fractional seconds precision*** must be a number between 0 and 9. (default is 6)	Time period stored in days, hours, minutes, and seconds. For example: interval day(2) to second(6)

Large Object (LOB) Datatypes

The following are the **LOB Datatypes** in Oracle/PLSQL:

Data Type Syntax	Oracle 11g	Explanation
bfile	Maximum file size of 2^{64}-1 bytes.	File locators that point to a binary file on the server file system (outside the database).
blob	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage).	Stores unstructured binary large objects.
clob	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage) of character data.	Stores single-byte and multi-byte character data.
nclob	Store up to (4 gigabytes -1) * (the value of the CHUNK parameter of LOB storage) of character text data.	Stores unicode data.

Rowid Datatypes

The following are the **Rowid Datatypes** in Oracle/PLSQL:

Data Type Syntax	Oracle 11g	Explanation
rowid	The format of the rowid is: BBBBBBB.RRRR.FFFFF Where BBBBBBB is the block in the database file; RRRR is the row in the block; FFFFF is the database file.	Fixed-length binary data. Every record in the database has a physical address or **rowid**.
urowid(size)		Universal rowid. Where ***size*** is optional.