Book Page # 243

Chapter 6:

Physical Database Design and Performance

Objectives

- Definition of terms
- Describe the physical database design process
- Choose storage formats for attributes
- Select appropriate file organizations
- Describe three types of file organization
- Describe indexes and their appropriate use
- Translate a database model into efficient structures
- Know when and how to use denormalization

Physical Database Design

- Purpose–translate the logical description of data into the *technical specifications* for storing and retrieving data
- Goal–create a design for storing data that will provide *adequate performance* and insure *database integrity*, *security*, and *recoverability*

Physical Design Process

Inputs

- Normalized relations
- Volume estimates
- Attribute definitions
- Response time expectations
- Data security needs
- Backup/recovery needs
- Integrity expectations
- DBMS technology used

Leads to

Decisions

- Attribute data types
- Physical record descriptions (doesn’t always match logical design)
- File organizations
- Indexes and database architectures
- Query optimization

Figure 6-1 Composite usage map

(Pine Valley Furniture Company)

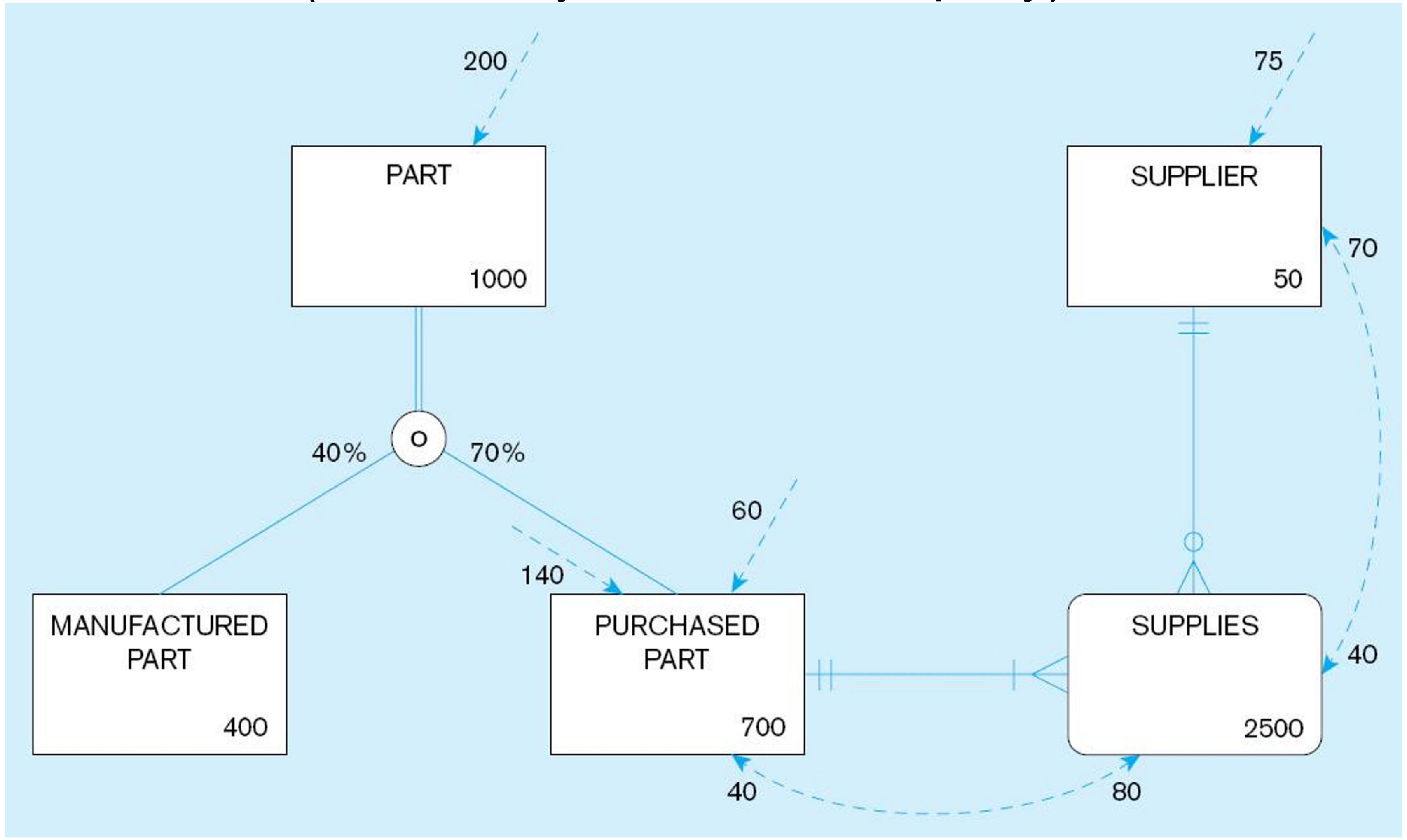


Figure 6-1 Composite usage map

(Pine Valley Furniture Company) (cont.)

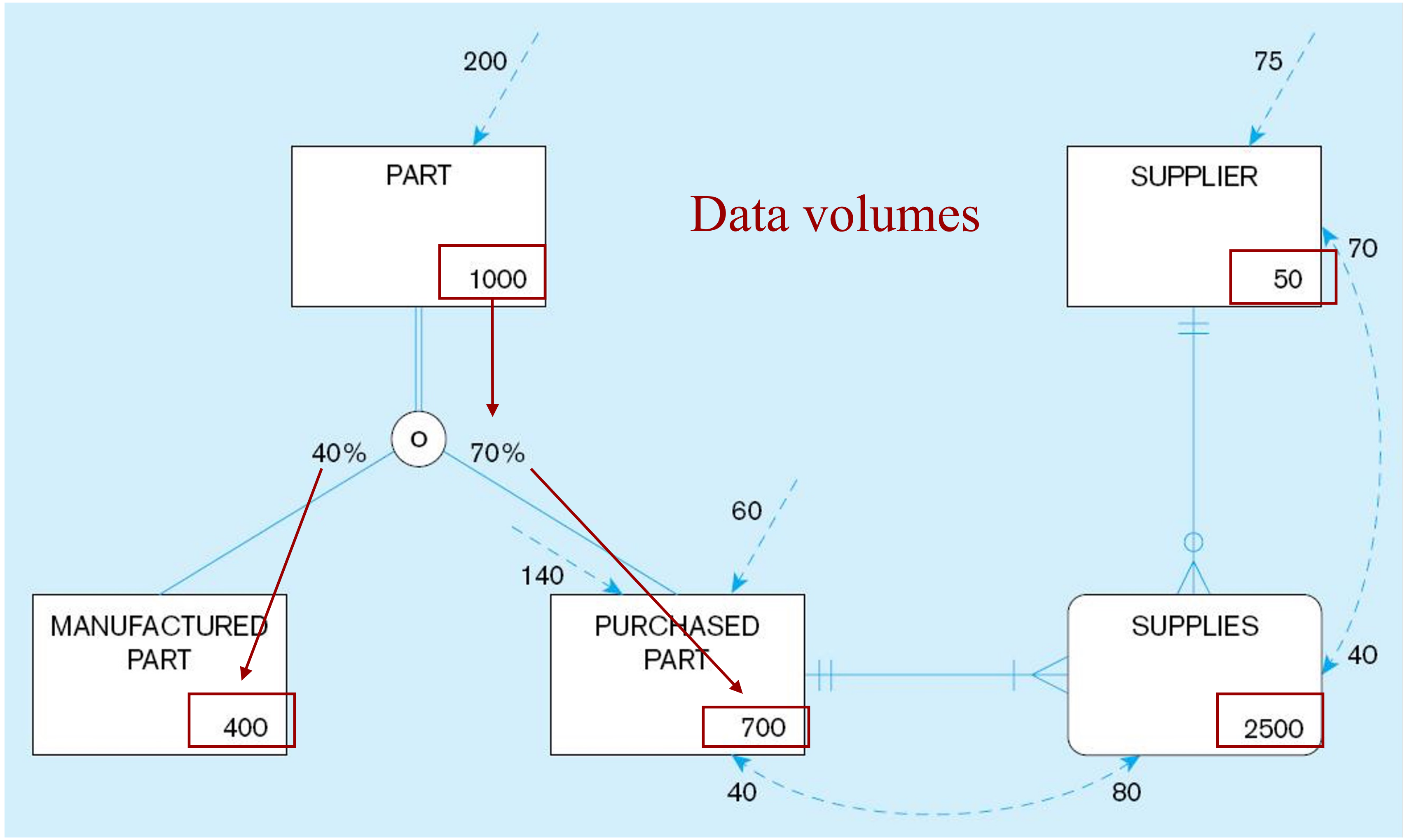


Figure 6-1 Composite usage map

(Pine Valley Furniture Company) (cont.)

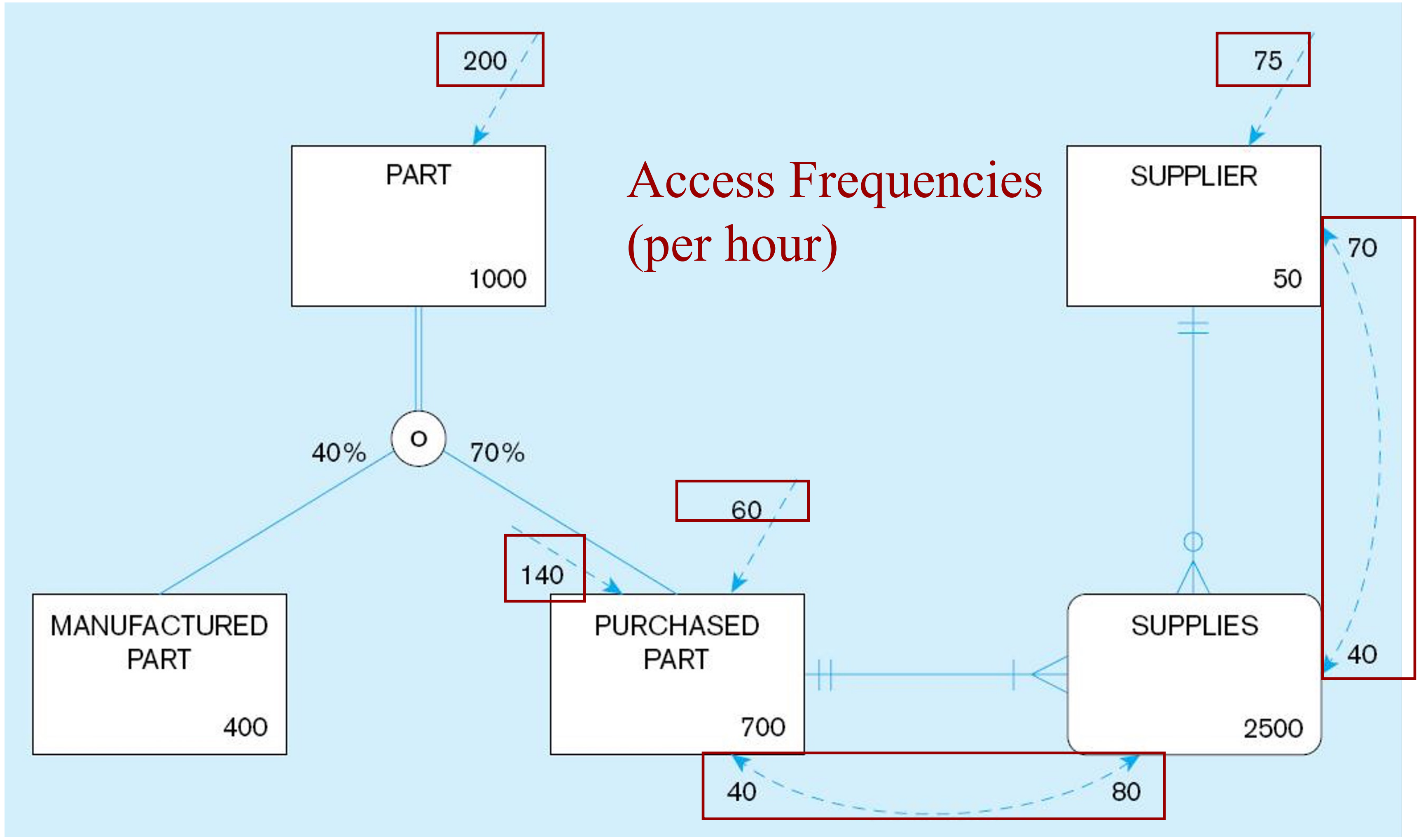
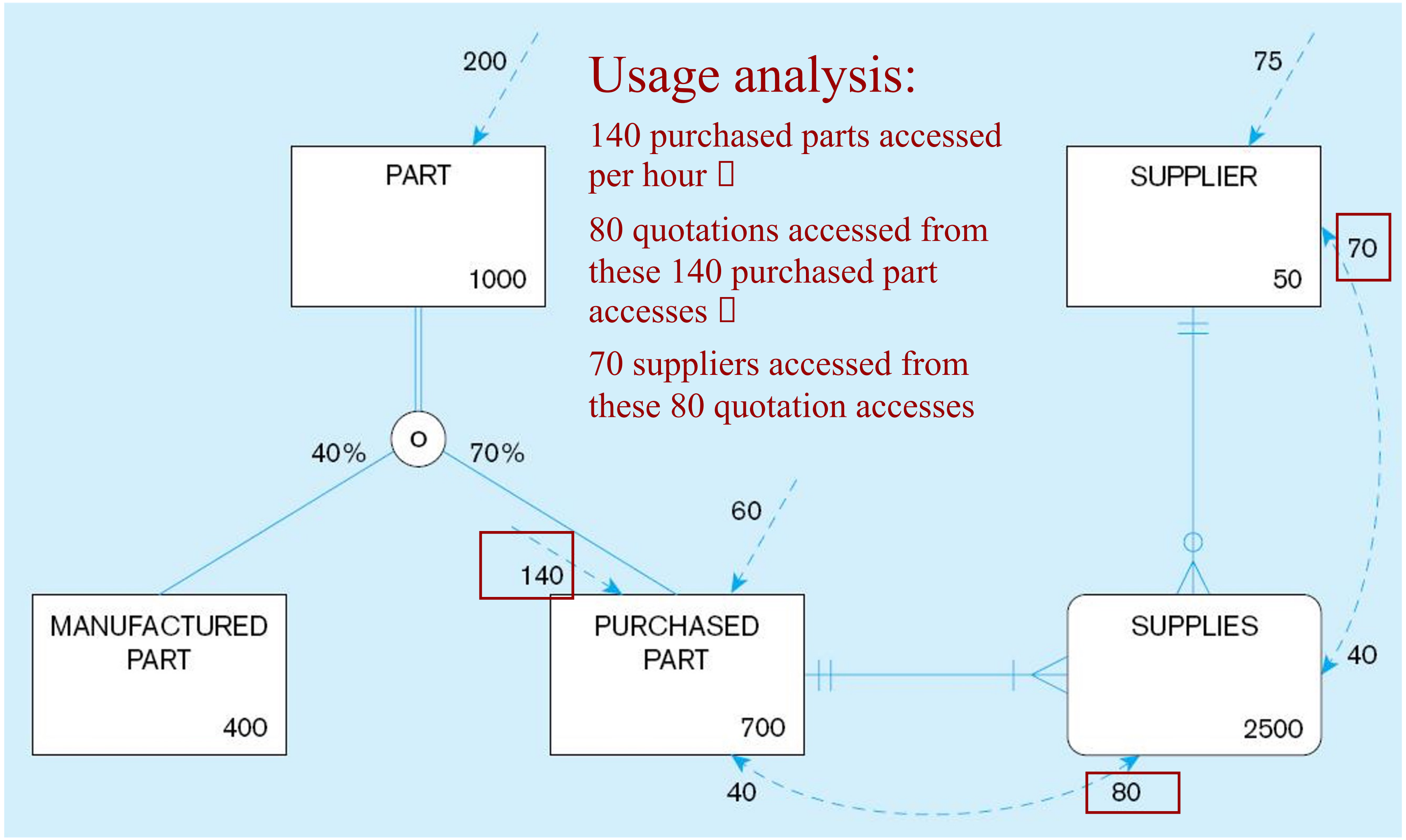


Figure 6-1 Composite usage map

(Pine Valley Furniture Company) (cont.)



Usage analysis:

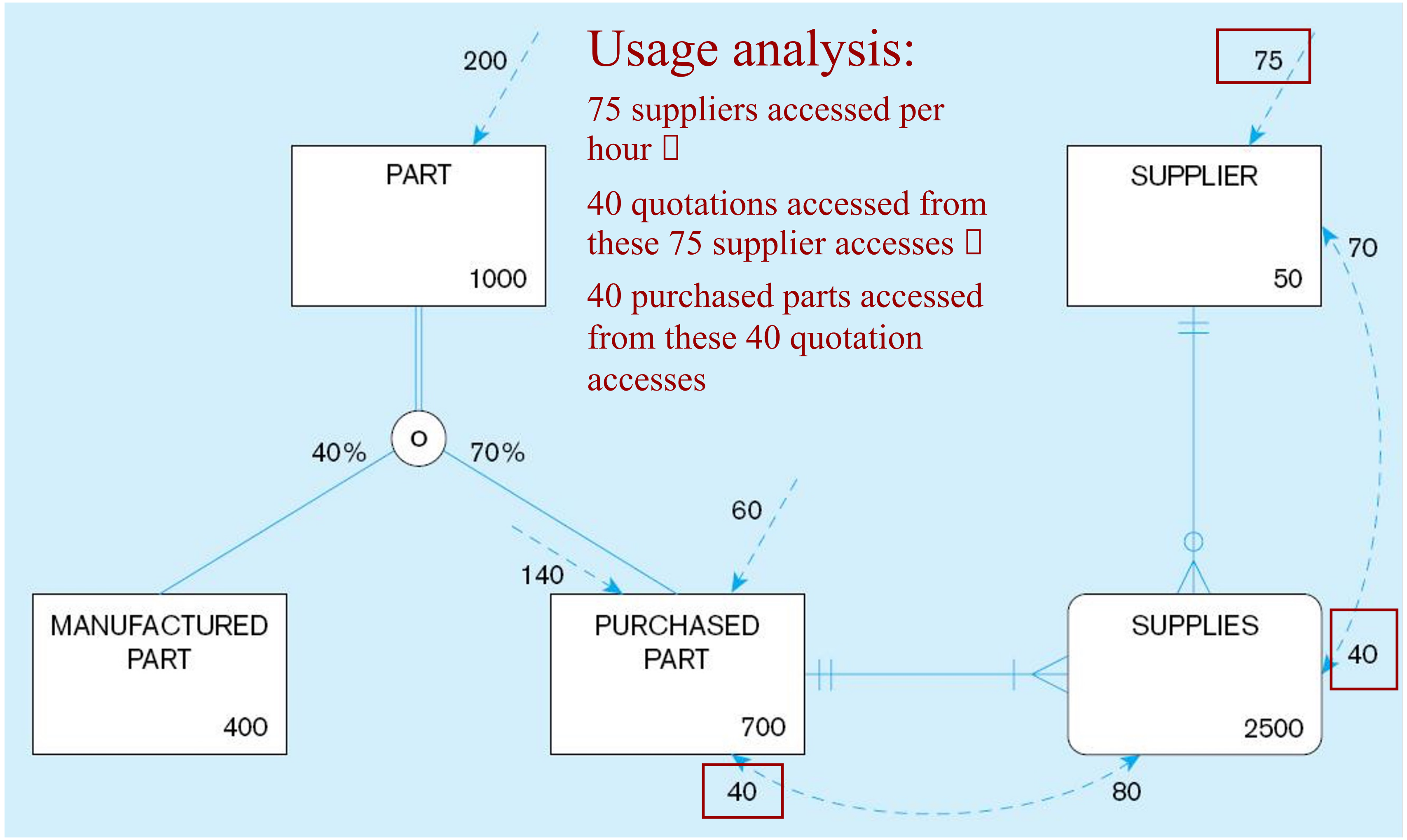
140 purchased parts accessed per hour

80 quotations accessed from these 140 purchased part accesses

70 suppliers accessed from these 80 quotation accesses

Figure 6-1 Composite usage map

(Pine Valley Furniture Company) (cont.)



Usage analysis:

75 suppliers accessed per hour

40 quotations accessed from these 75 supplier accesses

40 purchased parts accessed from these 40 quotation accesses

Designing Fields

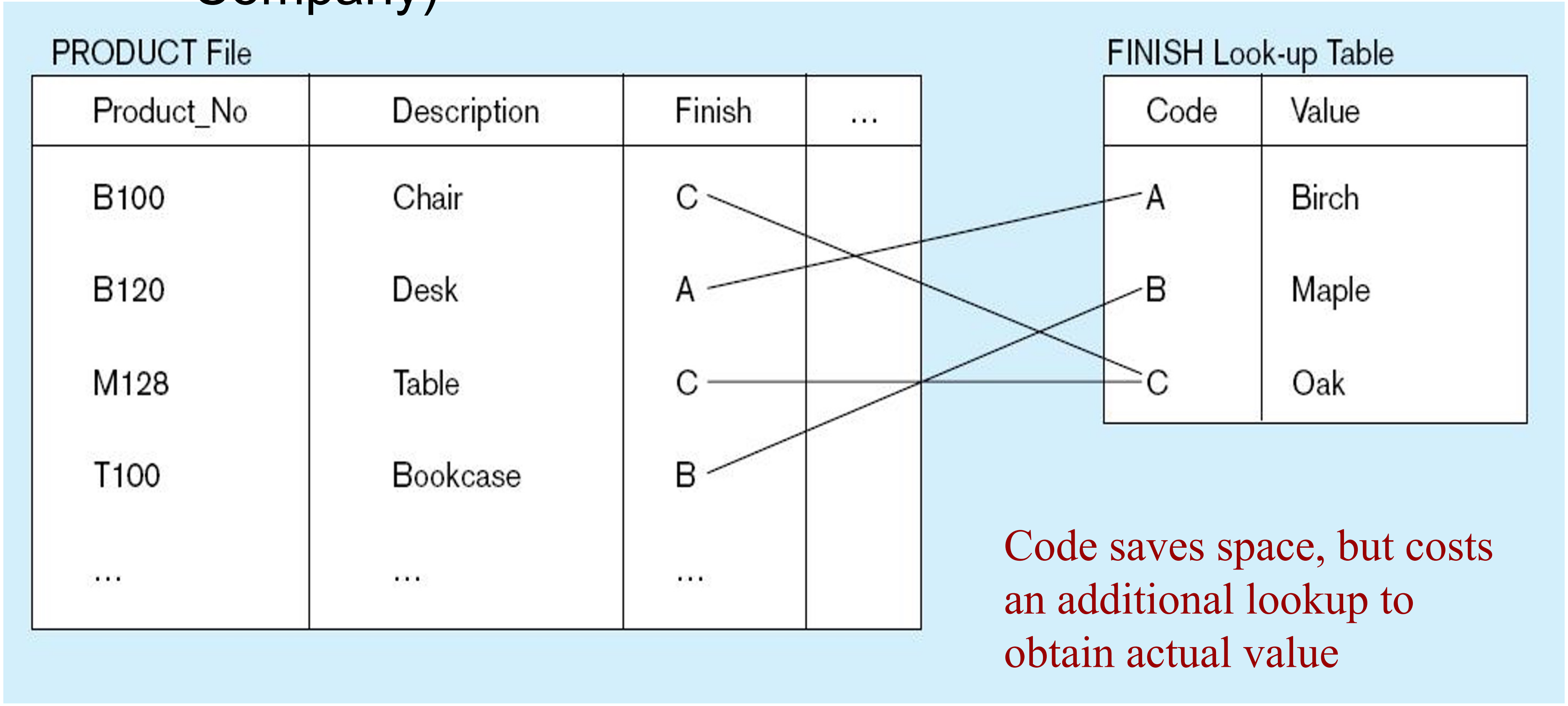
- Field: smallest unit of data in database
- Field design
  - Choosing data type
  - Coding, compression, encryption
  - Controlling data integrity

Choosing Data Types

- CHAR–fixed-length character
- VARCHAR2–variable-length character (memo)
- LONG–large number
- NUMBER–positive/negative number
- INEGER–positive/negative whole number
- DATE–actual date
- BLOB–binary large object (good for graphics, sound clips, etc.)

Figure 6-2 Example code look-up table

(Pine Valley Furniture Company)



PRODUCT File

Product_No	Description	Finish	…
B100	Chair	C	
B120	Desk	A	
M128	Table	C	
T100	Bookcase	B	
…	…	…	

FINISH Look-up Table

Code	Value
A	Birch
B	Maple
C	Oak

Code saves space, but costs an additional lookup to obtain actual value

Field Data Integrity

- Default value–assumed value if no explicit value
- Range control–allowable value limitations (constraints or validation rules)
- Null value control–allowing or prohibiting empty fields
- Referential integrity–range control (and null value allowances) for foreign-key to primary-key match-ups

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

Handling Missing Data

- Substitute an estimate of the missing value (e.g., using a formula)
- Construct a report listing missing values
- In programs, ignore missing data unless the value is significant (sensitivity testing)

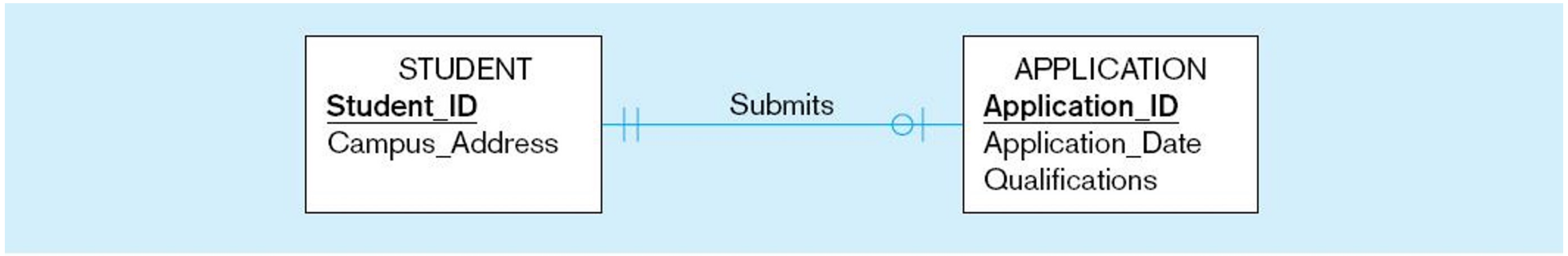
Physical Records

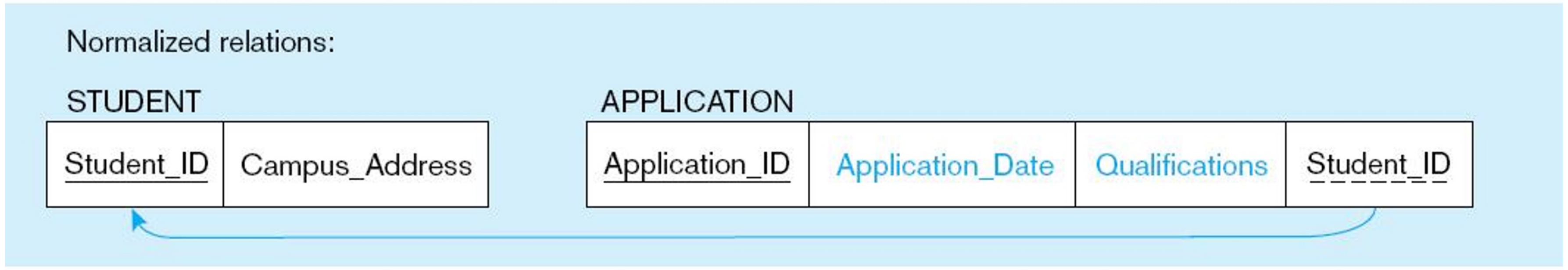
- Physical Record: A group of fields stored in adjacent memory locations and retrieved together as a unit
- Page: The amount of data read or written in one I/O operation

Denormalization

- Transforming ***normalized*** relations into ***unnormalized*** physical record specifications
- Benefits:
  - Can improve performance (speed) by reducing number of table lookups (i.e. *reduce number of necessary join queries*)
- Costs (due to data duplication)
  - Wasted storage space
  - Data integrity/consistency threats
- Common denormalization opportunities
  - One-to-one relationship (Fig. 6-3)
  - Many-to-many relationship with attributes (Fig. 6-4)
  - Reference data (1:N relationship where 1-side has data not used in any other relationship) (Fig. 6-5)

Figure 6-3 A possible denormalization situation: two entities with one-to-one relationship





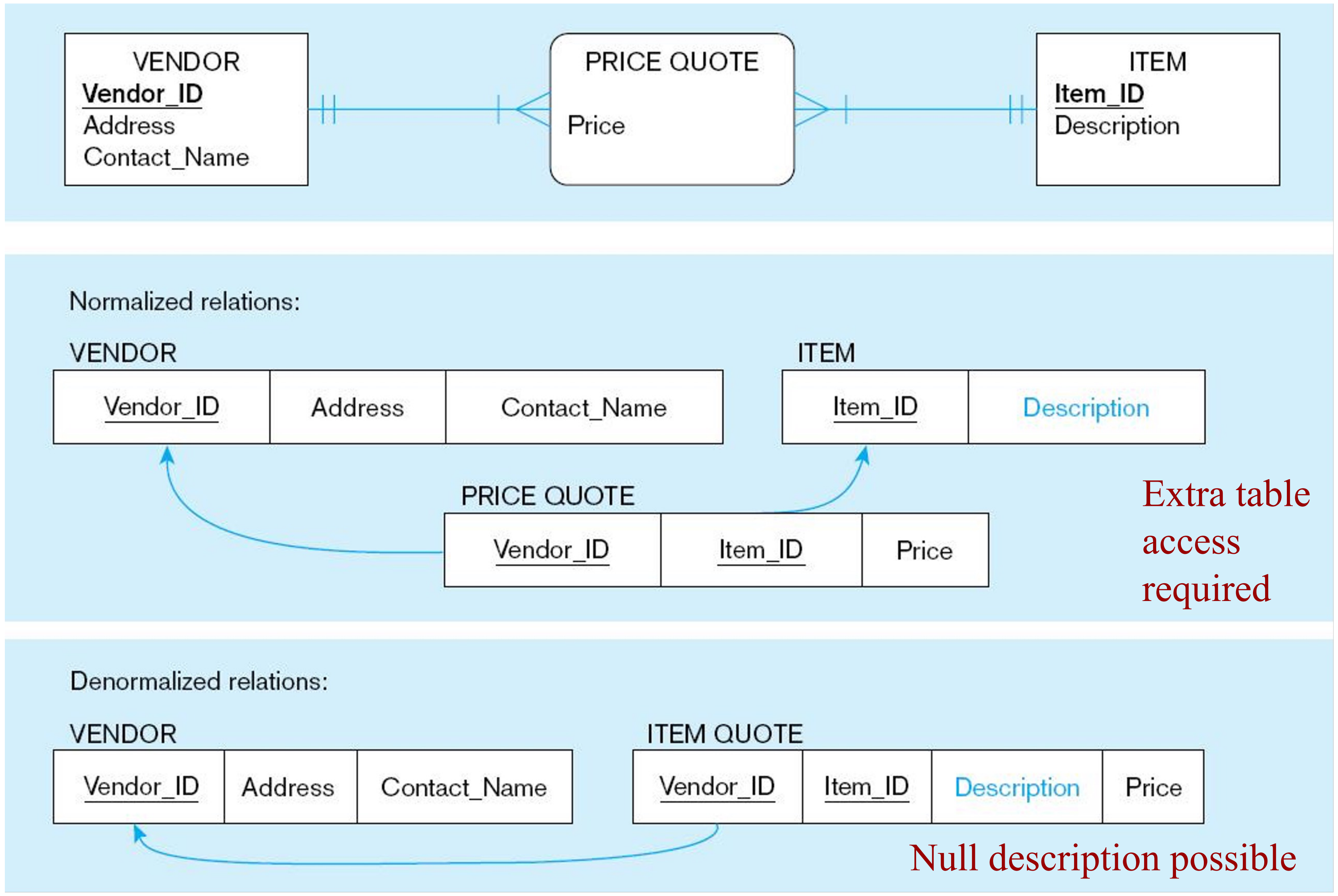
Denormalized relation:

STUDENT

Student_ID	Campus_Address	Application_Date	Qualifications

and Application_Date and Qualifications may be null

Figure 6-4 A possible denormalization situation: a many-to-many relationship with nonkey attributes



Normalized relations:

VENDOR

Vendor_ID	Address	Contact_Name

ITEM

Item_ID	Description

PRICE QUOTE

Vendor_ID	Item_ID	Price

Extra table access required

Denormalized relations:

VENDOR

Vendor_ID	Address	Contact_Name

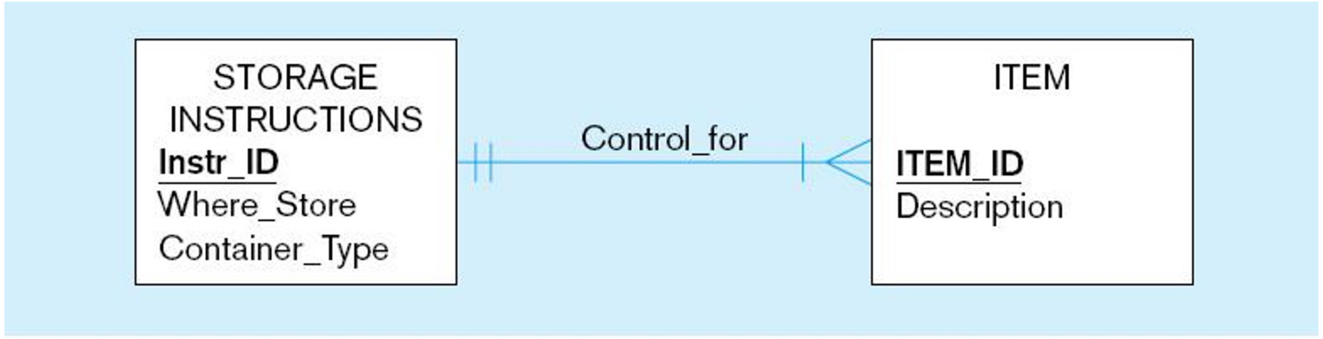
ITEM QUOTE

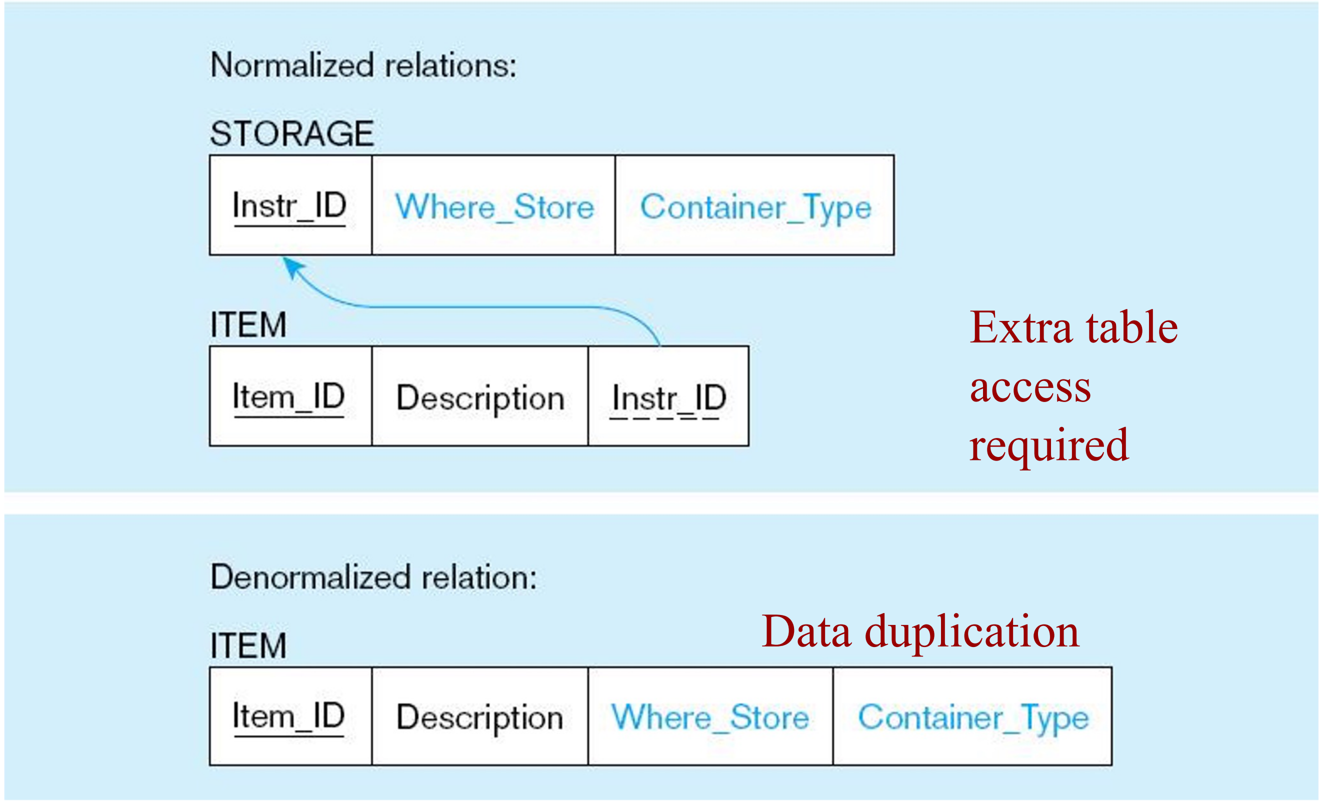
Vendor_ID	Item_ID	Description	Price

Null description possible

Figure 6-5

A possible denormalization situation: reference data





Normalized relations:

STORAGE

Instr_ID	Where_Store	Container_Type

ITEM

Item_ID	Description	Instr_ID

Denormalized relation:

ITEM

Item_ID	Description	Where_Store	Container_Type

Partitioning

- Horizontal Partitioning: Distributing the rows of a table into several separate files
  - Useful for situations where different users need access to different rows
- Vertical Partitioning: Distributing the columns of a table into several separate relations
  - Useful for situations where different users need access to different columns
  - The primary key must be repeated in each file
- Combinations of Horizontal and Vertical

Partitioning (cont.)

- Advantages of Partitioning:
  - Efficiency: Records used together are grouped together
  - Local optimization: Each partition can be optimized for performance
  - Security, recovery
  - Load balancing: Partitions stored on different disks, reduces contention
  - Take advantage of parallel processing capability
- Disadvantages of Partitioning:
  - Inconsistent access speed: Slow retrievals across partitions
  - Complexity: Non-transparent partitioning
  - Extra space or update time: Duplicate data; access from multiple partitions

Data Replication

- Purposely storing the same data in multiple locations of the database
- Improves performance by allowing multiple users to access the same data at the same time with minimum contention
- Sacrifices data integrity due to data duplication
- Best for data that is not updated often