CHAPTER 5: PHYSICAL DATABASE DESIGN AND PERFORMANCE

Modern Database Management

12th Edition

Jeff Hoffer, Ramesh Venkataraman, Heikki Topi

OBJECTIVES

- Define 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 ensure *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 5-1 Composite usage map (Pine Valley Furniture Company)

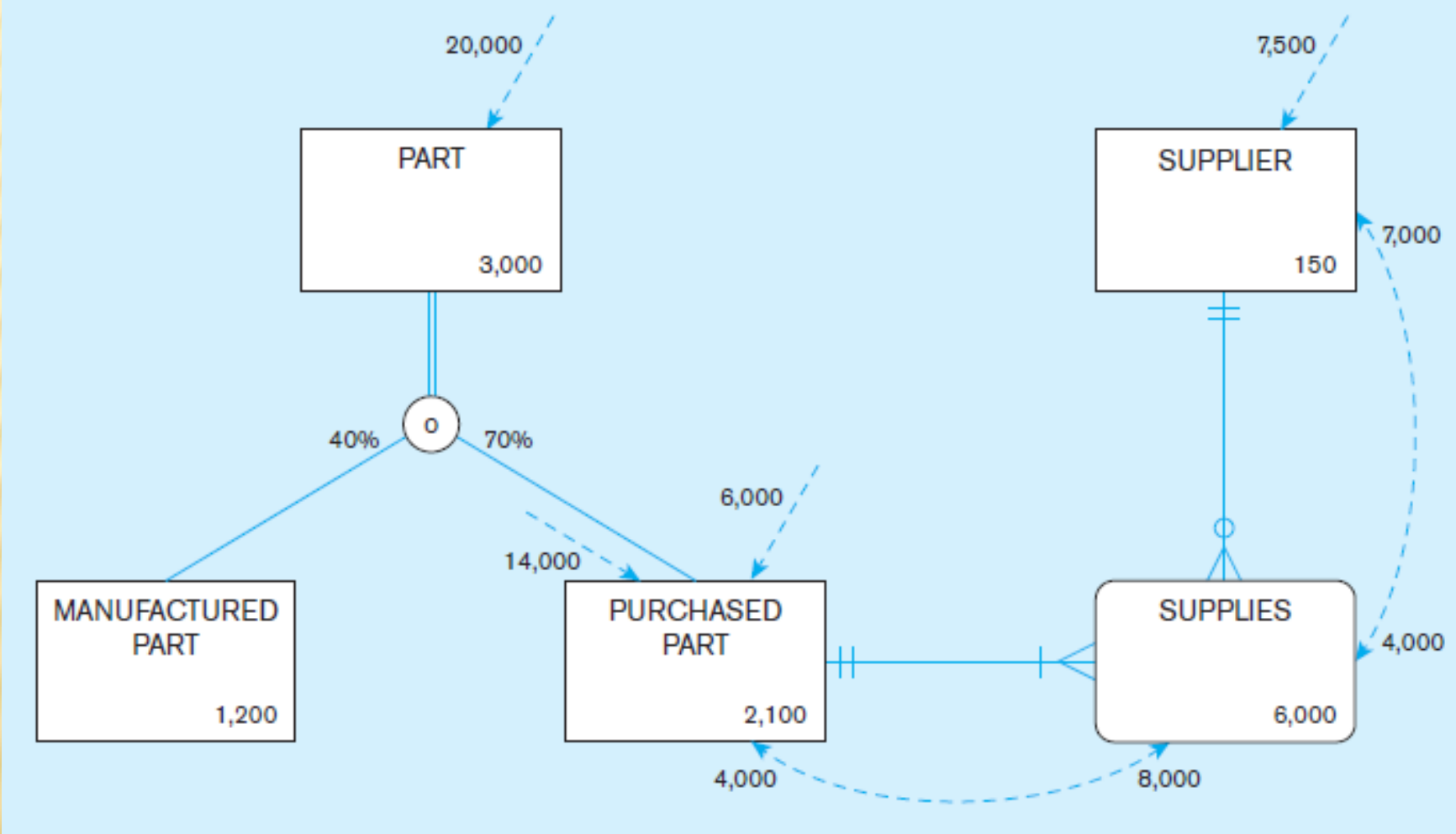


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)

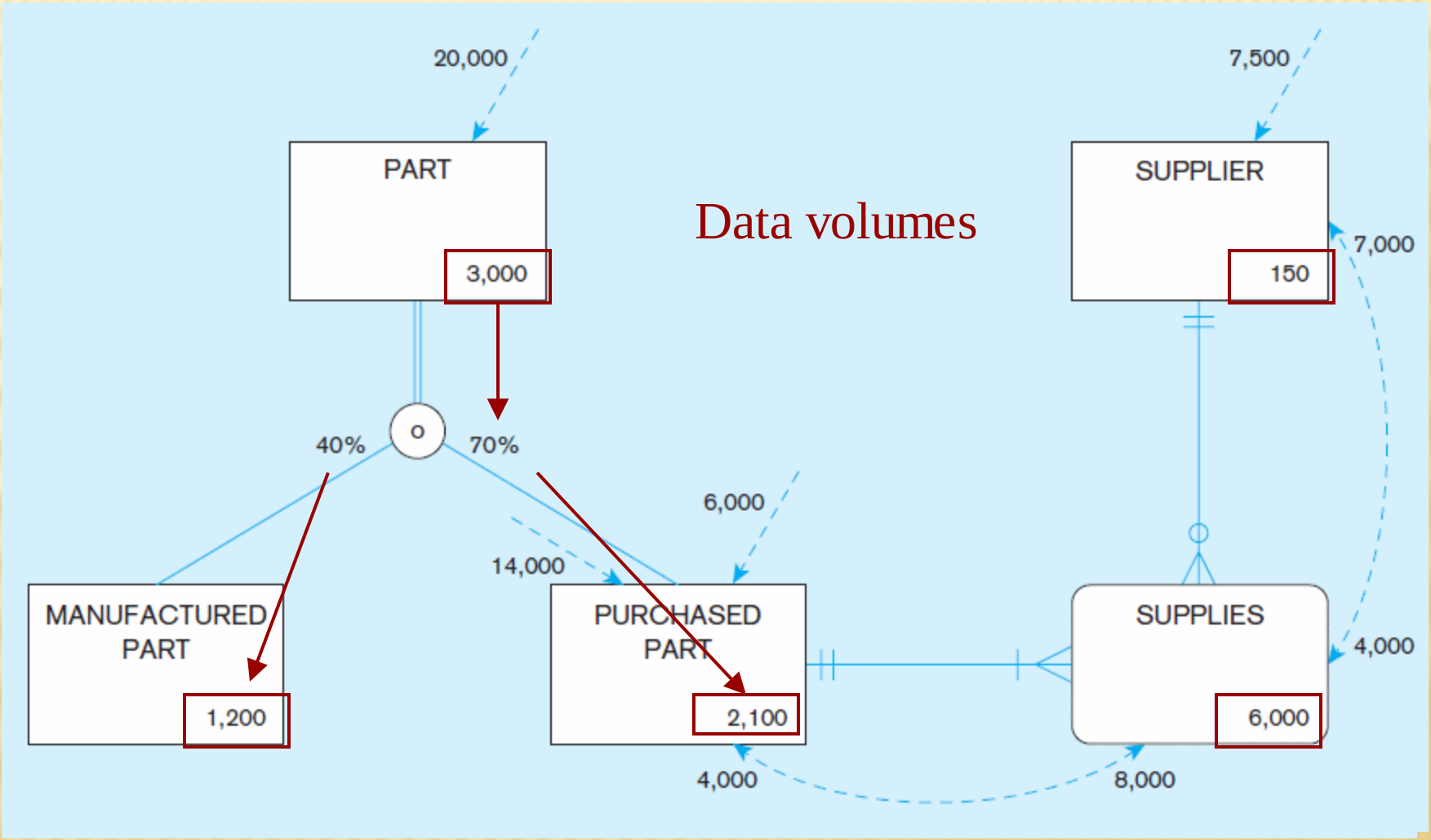


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)

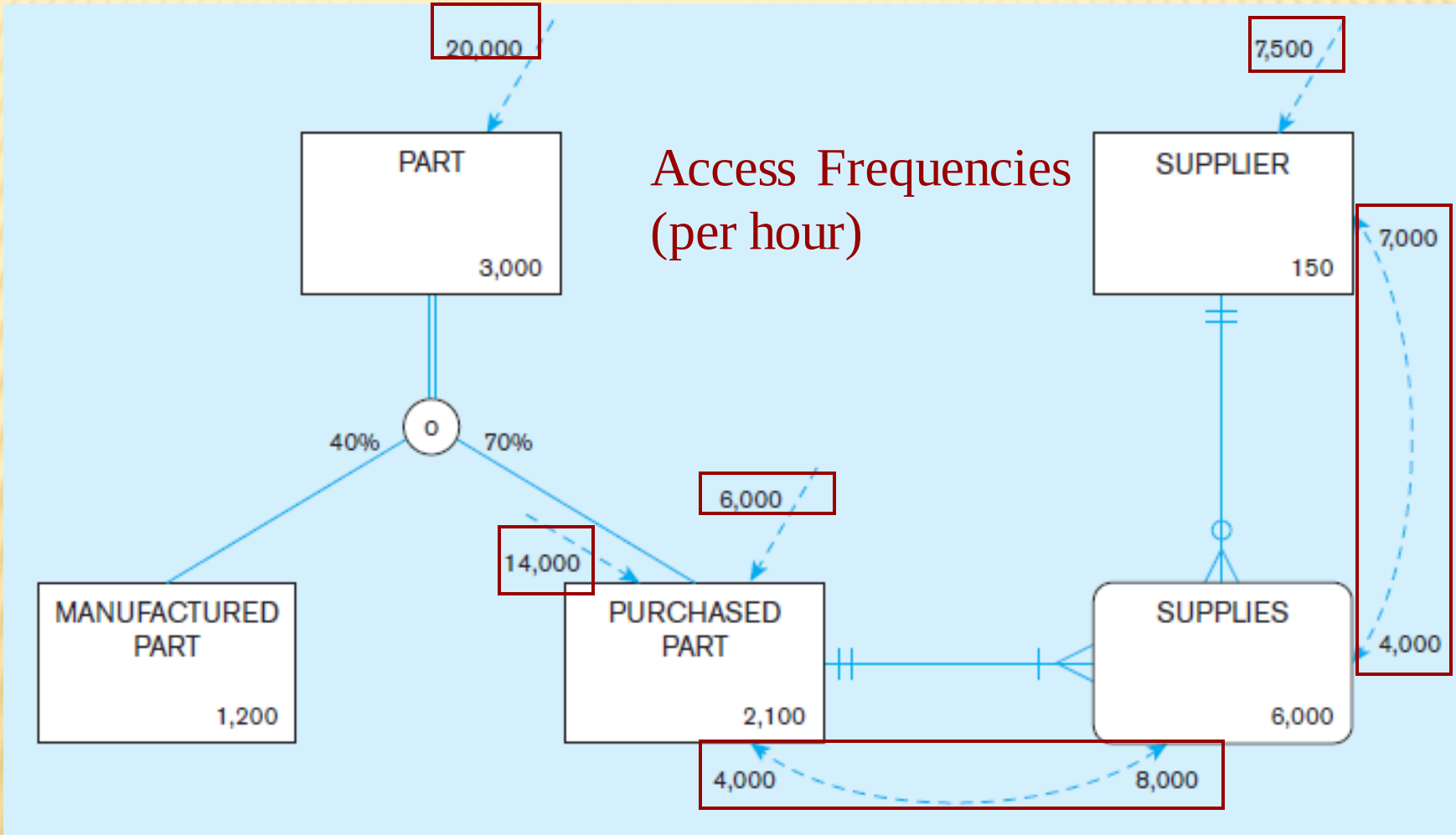


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)

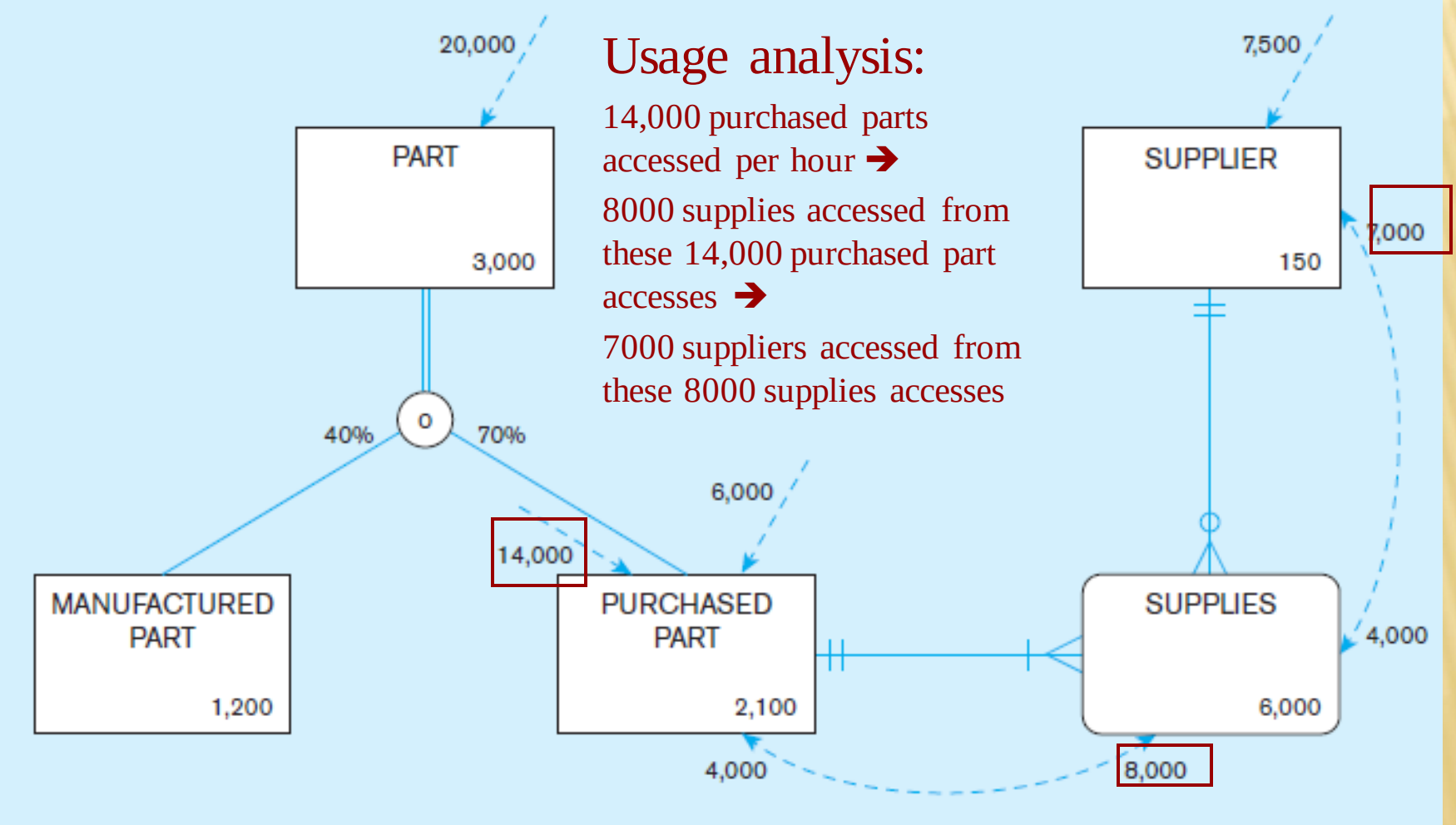
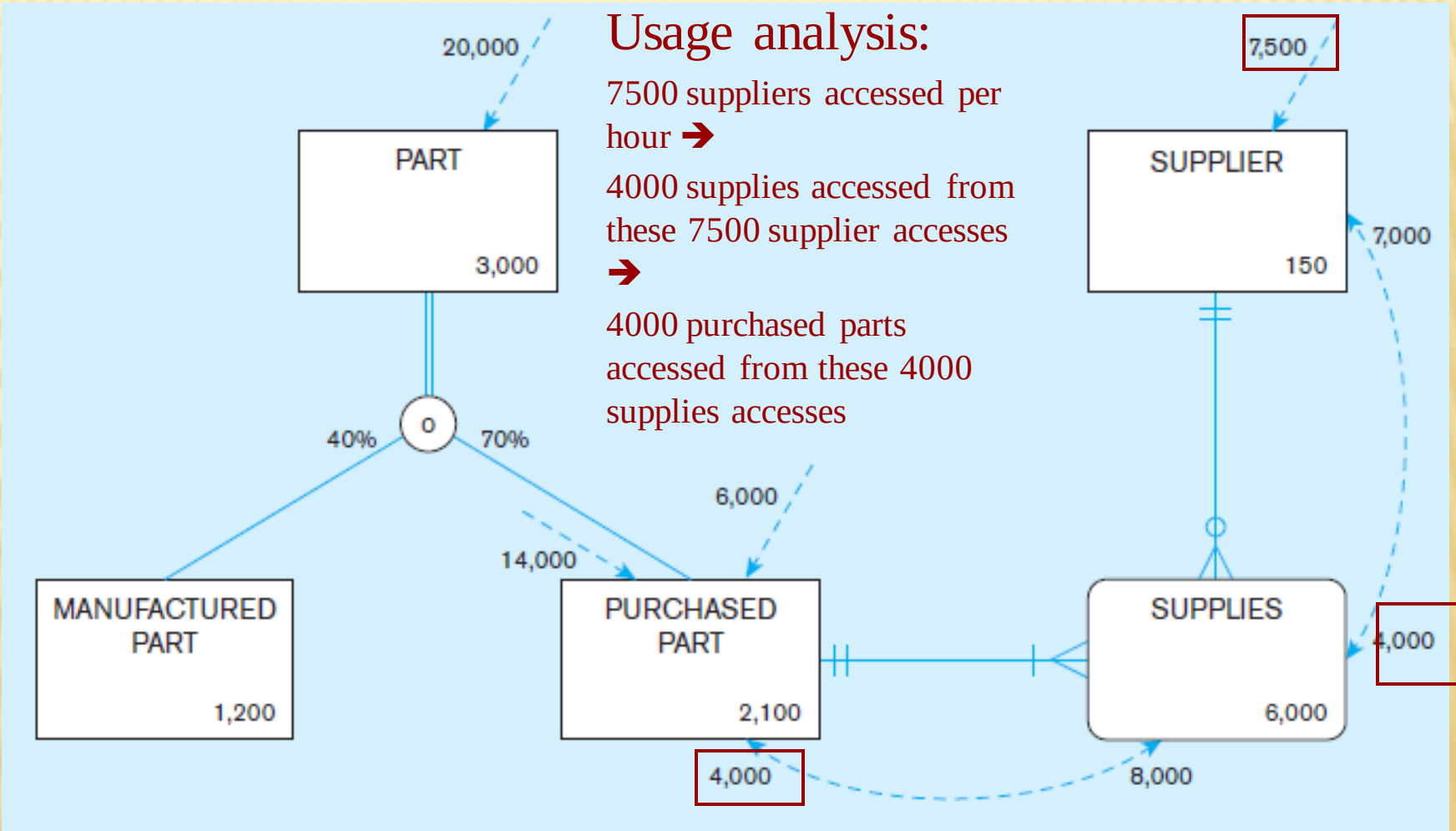


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)



DESIGNING FIELDS

- Field: smallest unit of application data recognized by system software
- Field design
  - Choosing data type
  - Coding, compression, encryption
  - Controlling data integrity

CHOOSING DATA TYPES

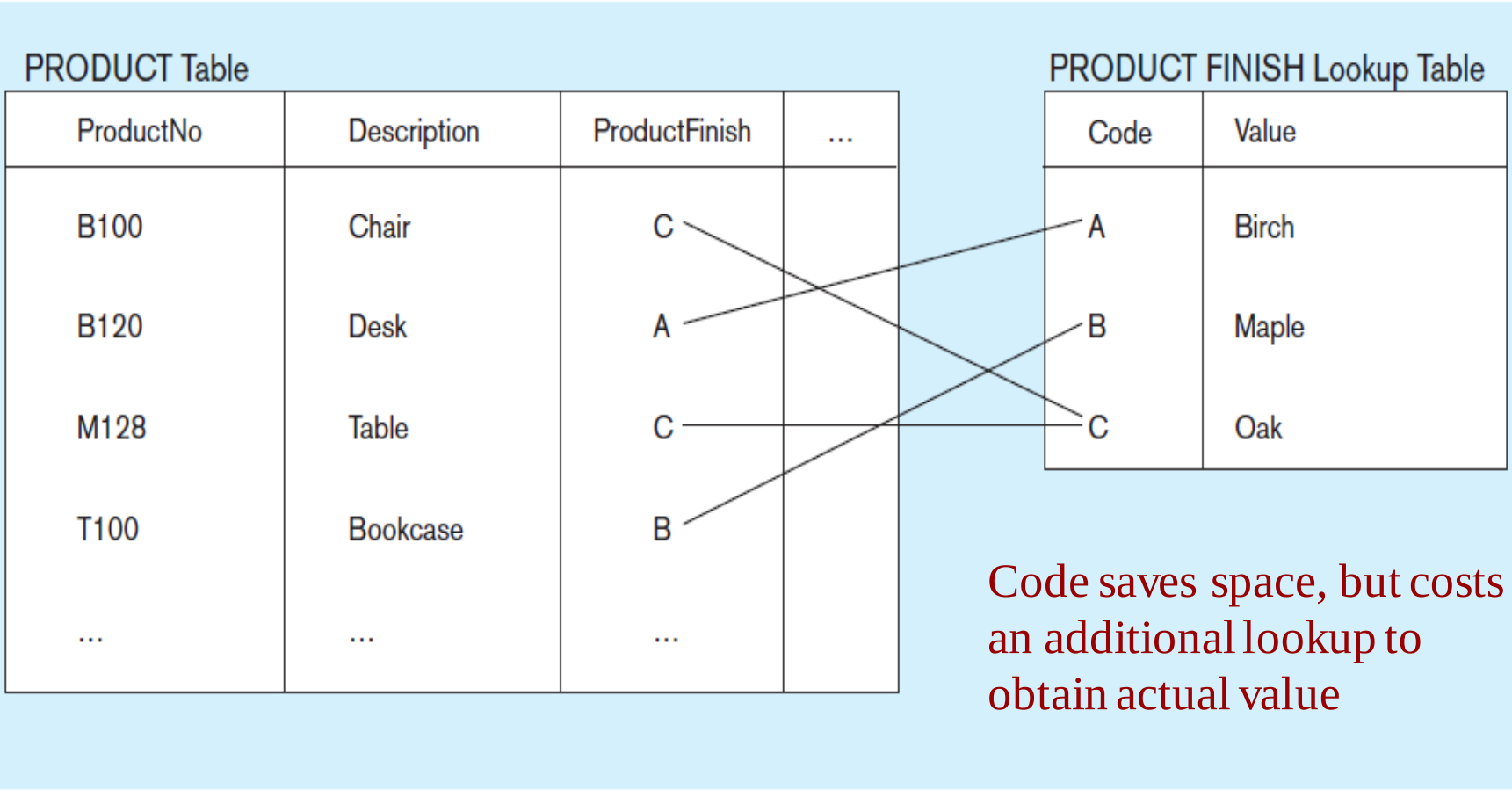
TABLE 5-1 Commonly Used Data Types in Oracle 12c

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. A corresponding data type for Unicode character data allowing for the use of a rich variety of national character sets is NVARCHAR2.
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). There is also a data type called NCHAR, which allows the use of Unicode character data.
CLOB	Character large object, capable of storing up to 4 gigabytes 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 to a maximum of 38) 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 of binary data (e.g., a photograph or sound clip).

CHOOSING DATA TYPES

- Selecting a data type involves four objectives that will have different relative levels of importance for different applications:
  - Represent all possible values.
  - Improve data integrity.
  - Support all data manipulations.
  - Minimize storage space.

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



FIELD DATA INTEGRITY

- Supported field data integrity RDBMS can support
  - 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

HANDLING MISSING DATA

Missing data is inevitable. The following methods can be used to handle 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)

Triggers can be used to perform these operations.

THE NEED OF DENORMALIZATION

- A fully normalized database usually creates a large number of tables
- RDBMS often match tables together to form query result (called joining)
- Because joining is time-consuming, performance between total normalized and partial normalized is dramatic.
- Therefore, denormalization is a solution

DENORMALIZATION

- Transforming ***normalized*** relations into ***non-normalized*** 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. 5-3)
  - Many-to-many relationship with non-key attributes (associative entity) (Fig. 5-4)
  - Reference data (1:N relationship where 1-side has data not used in any other relationship) (Fig. 5-5)

Figure 5-3 A possible denormalization situation: two entities with one-to-one relationship

Combine these two relations into one record definition

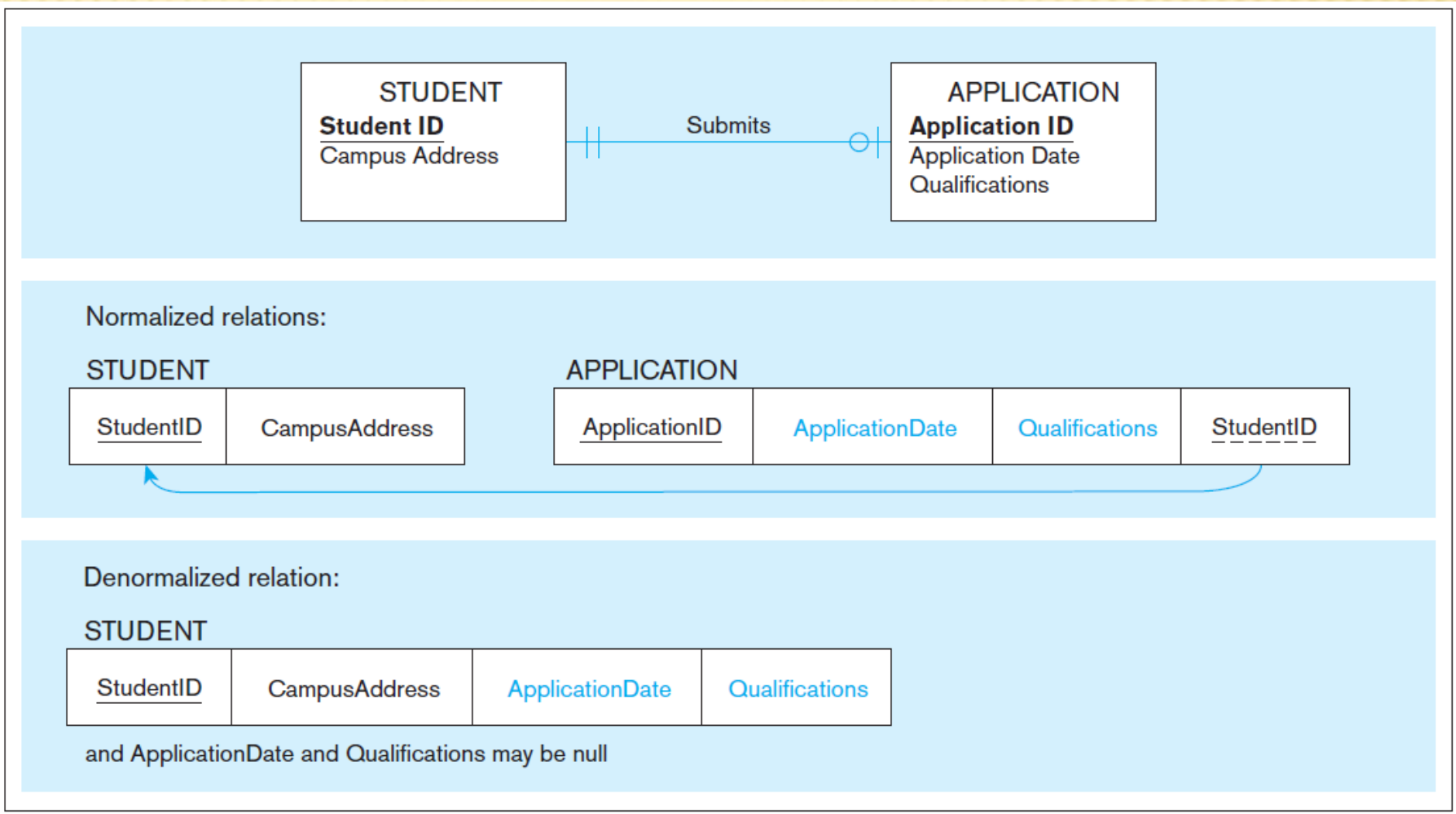
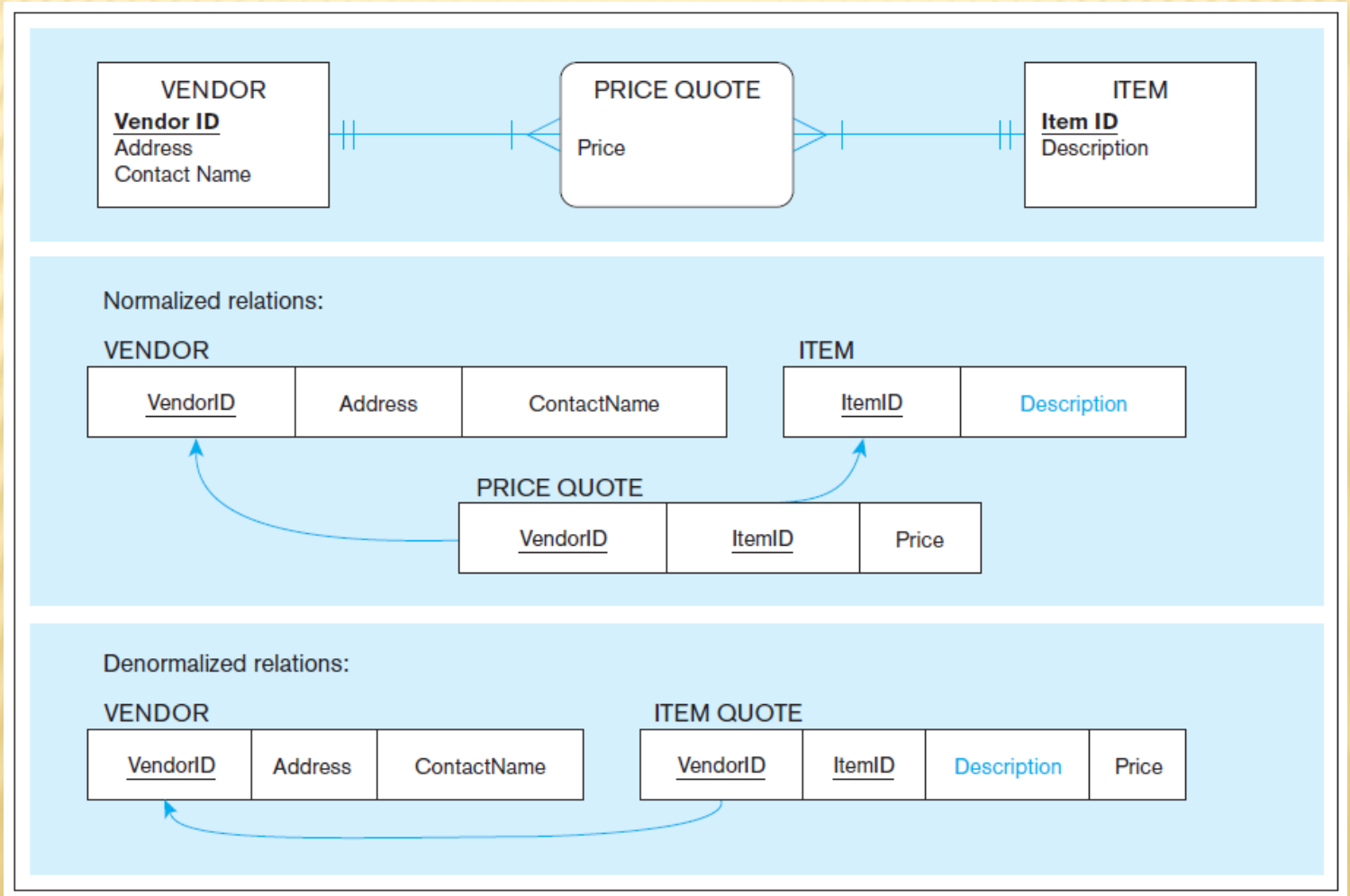


Figure 5-4 A possible denormalization situation: a many-to-many relationship with nonkey attributes

Combine attributes from one of the entities into the record representing the many-to-many relationship, thus avoiding one of the join operations

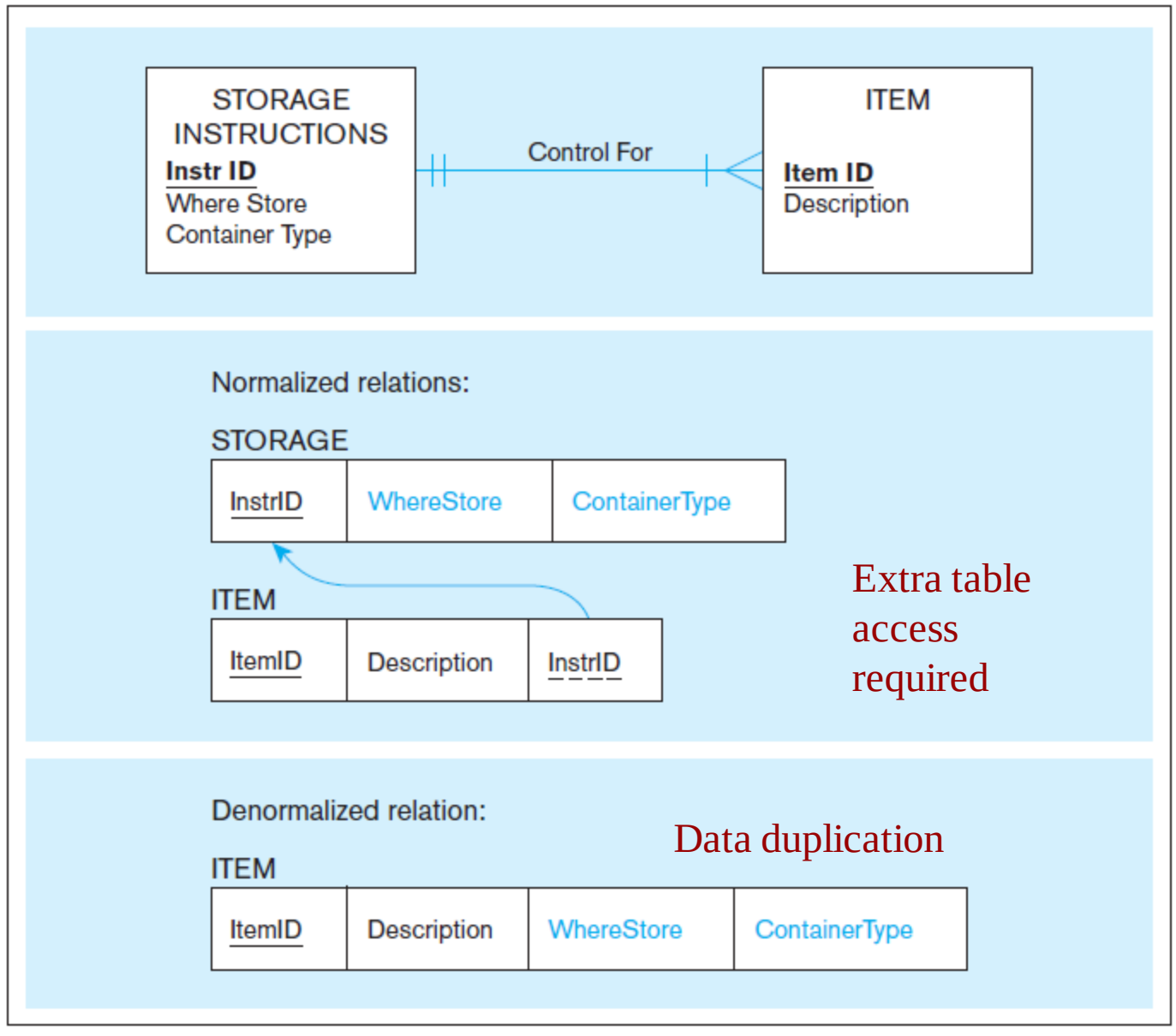


Extra table access required

Duplicate description possible

Figure 5-5
A possible denormalization situation: reference data

Merging the two entities in this situation into one record definition when there are few instances of the entity on the many side for each entity instance on the one side



DENORMALIZE WITH CAUTION

- Denormalization can
  - Increase chance of errors and inconsistencies
  - Reintroduce anomalies
  - Force reprogramming when business rules change
- 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 optimization

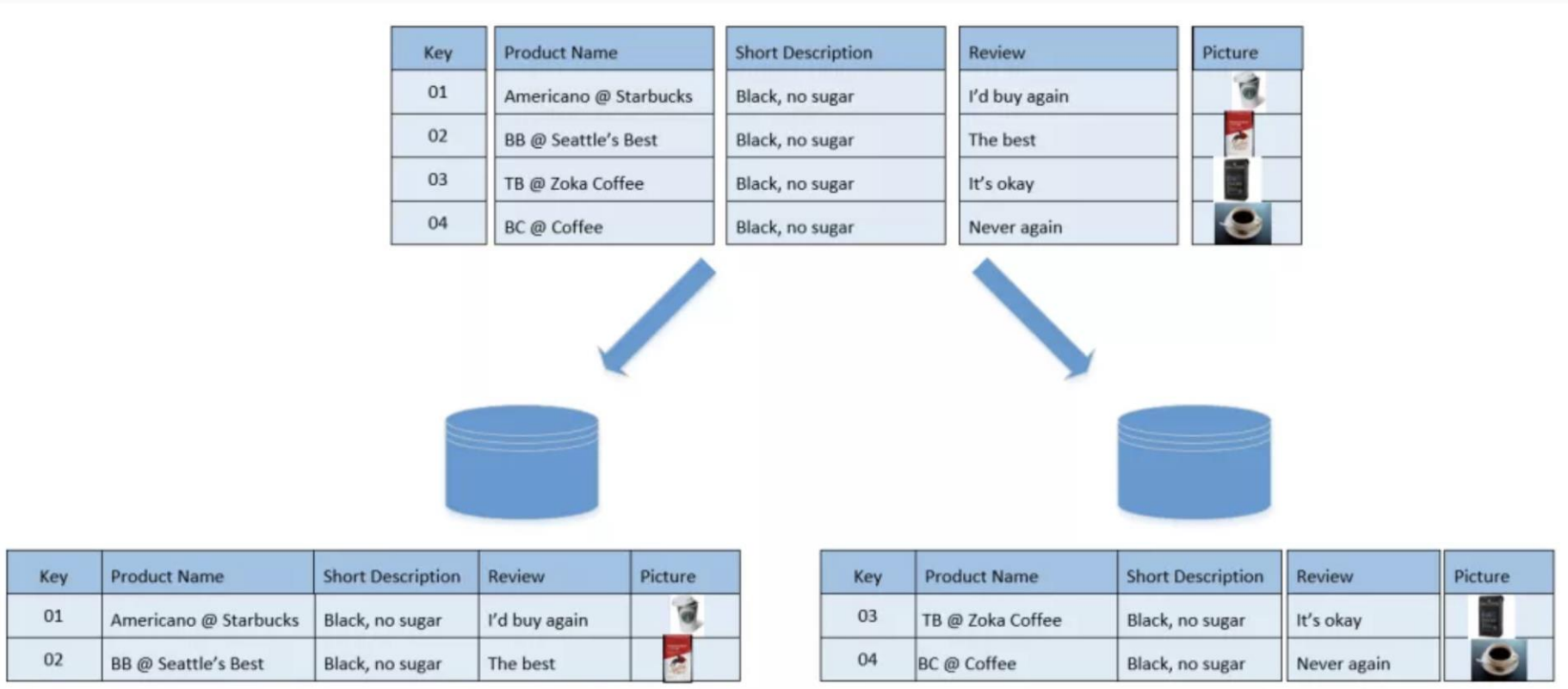
PARTITIONING

- **Horizontal Partitioning:** Distributing the rows of a logical relation into several separate tables
  - Useful for situations where different users need access to different rows
  - Three types: Key Range Partitioning, Hash Partitioning, or Composite Partitioning
- **Vertical Partitioning:** Distributing the columns of a logical relation into several separate physical tables
  - 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

HORIZONTAL PARTITIONING

- Placing different rows into different tables, based on common column values
- Benefits
  - Make maintenance of a table more efficient because fragmenting and rebuilding can be isolated to single partitions as storage space needs to be reorganized
  - More secure because file-level security can be used to prohibit users from seeing certain rows of data

HORZ. PARTITIONING EXAMPLE



Key	Product Name	Short Description	Review	Picture
01	Americano @ Starbucks	Black, no sugar	I'd buy again	
02	BB @ Seattle's Best	Black, no sugar	The best	
03	TB @ Zoka Coffee	Black, no sugar	It's okay	
04	BC @ Coffee	Black, no sugar	Never again	

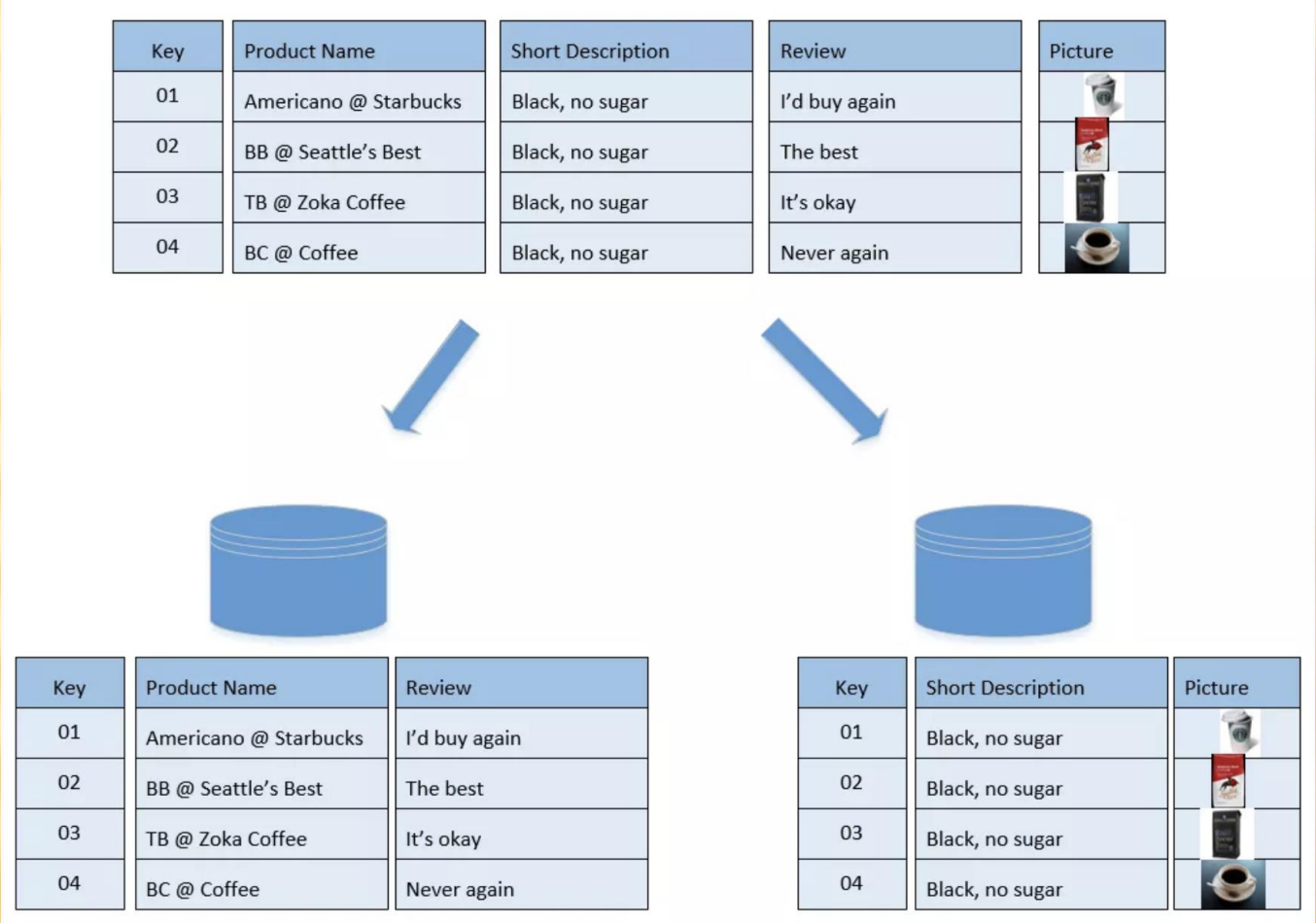
Key	Product Name	Short Description	Review	Picture
01	Americano @ Starbucks	Black, no sugar	I'd buy again	
02	BB @ Seattle's Best	Black, no sugar	The best	

Key	Product Name	Short Description	Review	Picture
03	TB @ Zoka Coffee	Black, no sugar	It's okay	
04	BC @ Coffee	Black, no sugar	Never again	

VERTICAL PARTITIONING

- Distribution of the columns of a logical relation into several separate physical tables.
- Example:
  - One PART table involving accounting, engineering, and sales attributes.
  - Split into three, each with the same Product ID, one for each user group.
  - This reduces demand on individual relations.
  - When combinations of data are required, perform join queries for all needed relations.

VERTICAL PARTITIONING EXAMPLE



PARTITIONING PROS AND CONS

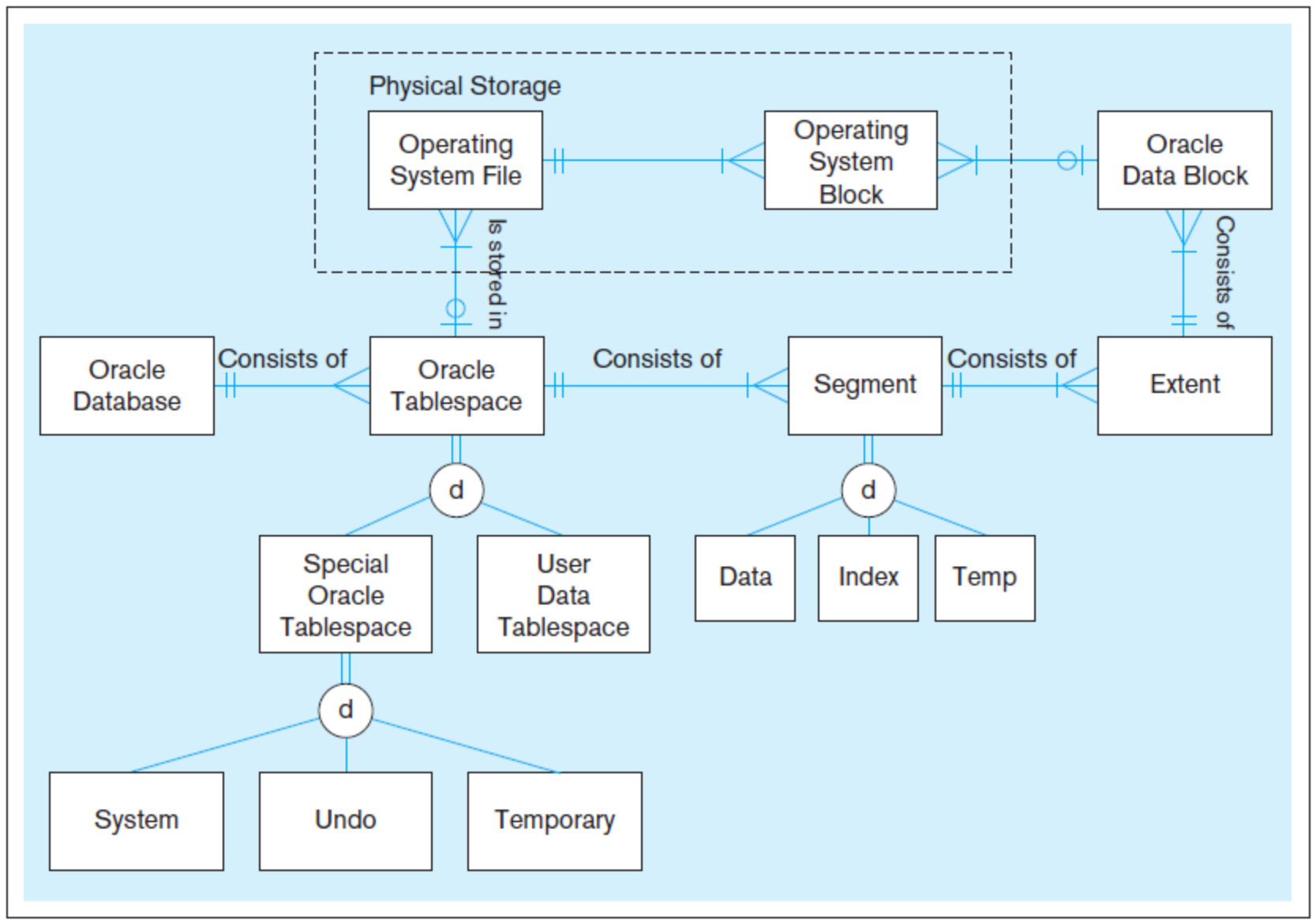
TABLE 5-2 Advantages and Disadvantages of Data Partitioning

Advantages of Partitioning
1. *Efficiency*: Data queried together are stored close to one another and separate from data not used together. Data maintenance is isolated in smaller partitions.
2. *Local optimization*: Each partition of data can be stored to optimize performance for its own use.
3. *Security*: Data not relevant to one group of users can be segregated from data those users are allowed to use.
4. *Recovery and uptime*: Smaller files take less time to back up and recover, and other files are still accessible if one file is damaged, so the effects of damage are isolated.
5. *Load balancing*: Files can be allocated to different storage areas (disks or other media), which minimizes contention for access to the same storage area or even allows for parallel access to the different areas.
Disadvantages of Partitioning
1. *Inconsistent access speed*: Different partitions may have different access speeds, thus confusing users. Also, when data must be combined across partitions, users may have to deal with significantly slower response times than in a non-partitioned approach.
2. *Complexity*: Partitioning is usually not transparent to programmers, who will have to write more complex programs when combining data across partitions.
3. *Extra space and update time*: Data may be duplicated across the partitions, taking extra storage space compared to storing all the data in normalized files. Updates that affect data in multiple partitions can take more time than if one file were used.

DESIGNING PHYSICAL DATABASE FILES

- Physical File:
  - A named portion of secondary memory allocated for the purpose of storing physical records
  - Tablespace–named logical storage unit in which data from multiple tables/views/objects can be stored
- Tablespace components
  - Segment – a table, index, or partition
  - Extent–contiguous section of disk space
  - Data block – smallest unit of storage

Figure 5-6 DBMS terminology in an Oracle 12c environment

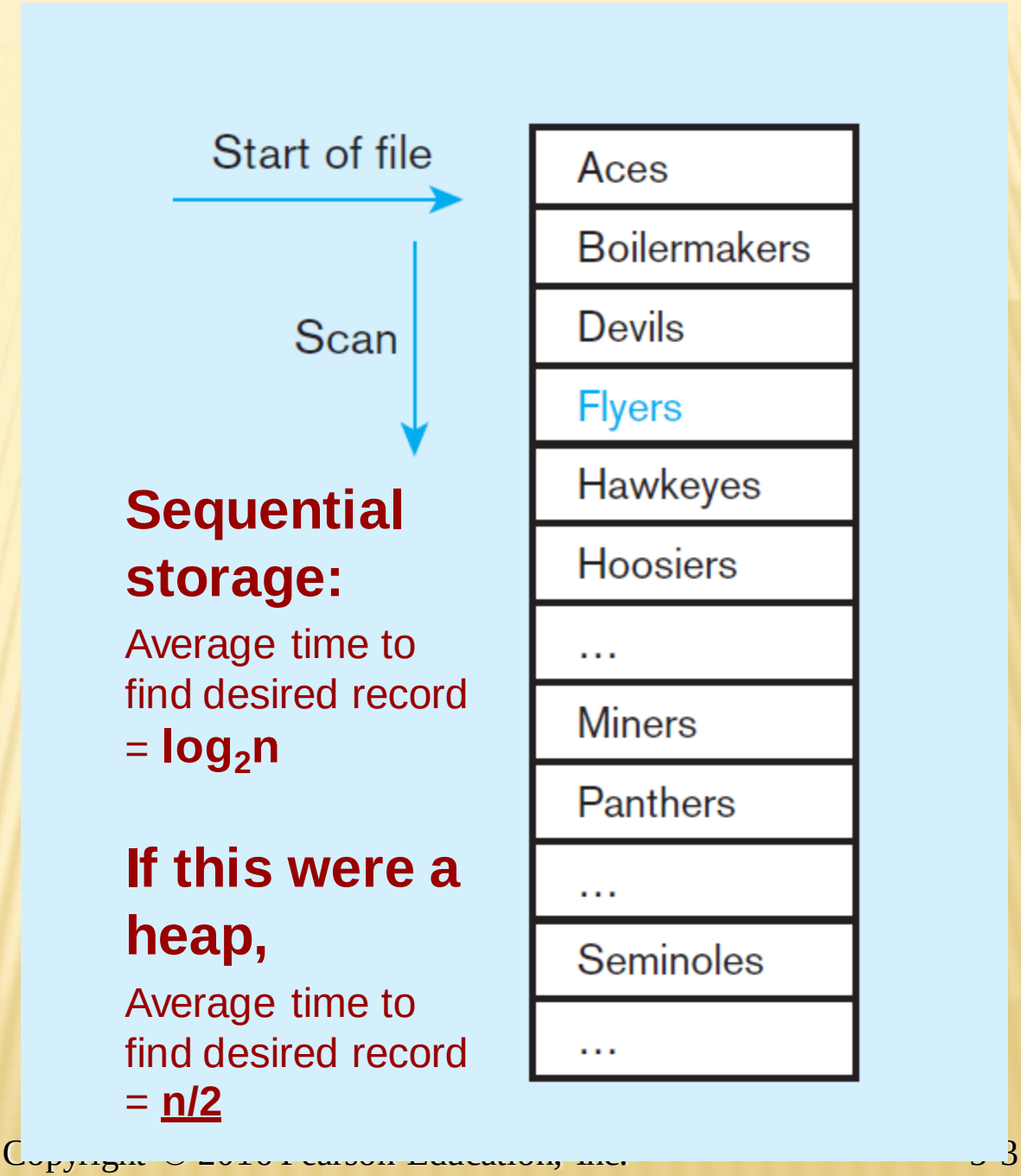


FILE ORGANIZATIONS

- Technique for physically arranging records of a file on secondary storage
- Factors for selecting file organization:
  - Fast data retrieval and throughput: performance time
  - Efficient storage space utilization
  - Protection from failure and data loss: backup/restore.
  - Minimizing need for reorganization
  - Accommodating growth
  - Security from unauthorized use: password
- Types of file organizations
  - Heap – no particular order
  - Sequential
  - Indexed
  - Hashed

Figure 5-7a
Sequential file organization

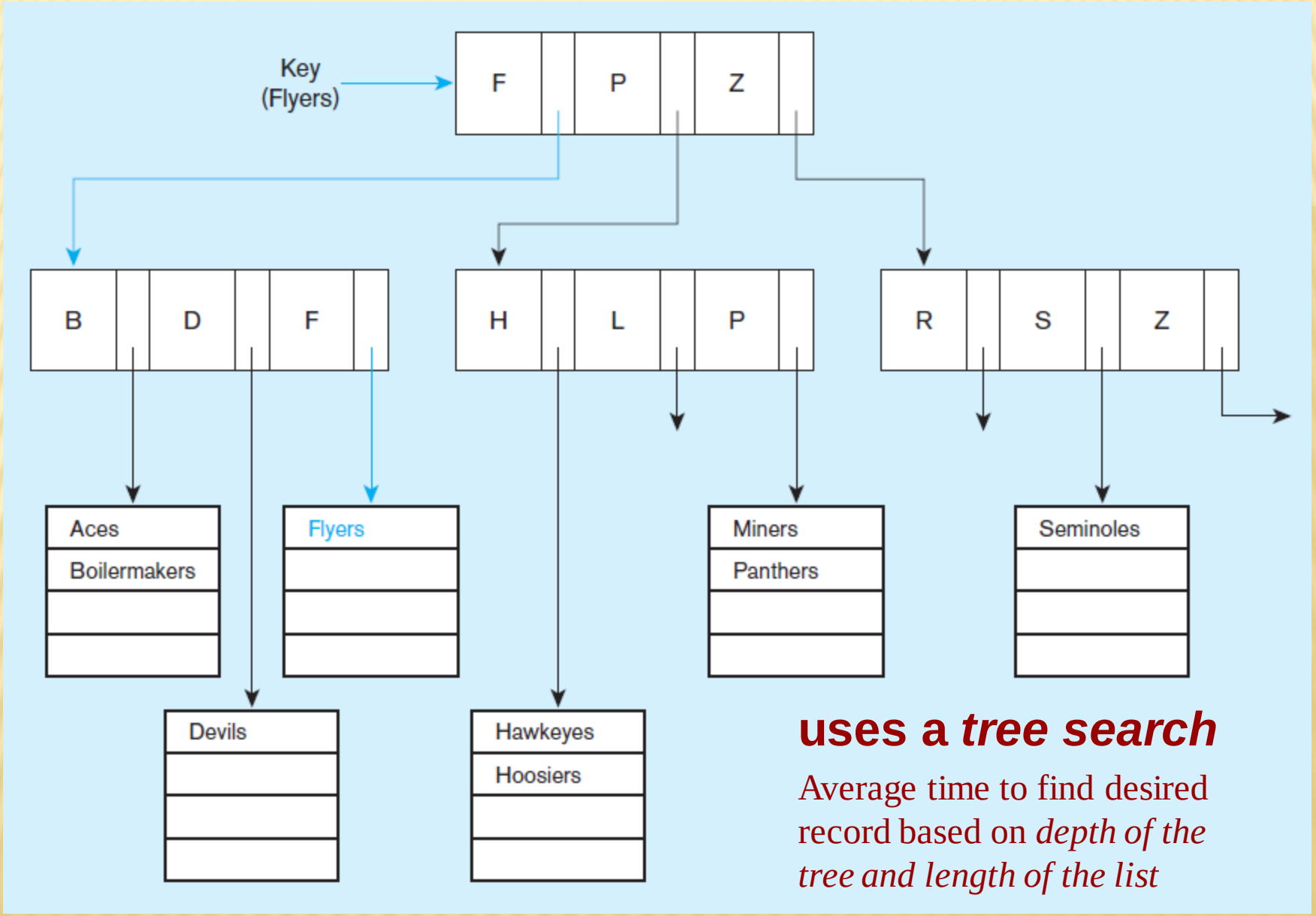
Records of the file are stored in sequence by the primary key field values.



INDEXED FILE ORGANIZATIONS

- Storage of records sequentially or nonsequentially with an index that allows software to locate individual records
- **Index:** a table or other data structure used to determine in a file the location of records that satisfy some condition
- Primary keys are automatically indexed
- Other fields or combinations of fields can also be indexed; these are called secondary keys (or nonunique keys)

Figure 5-7b Indexed file organization



(c) Hashed

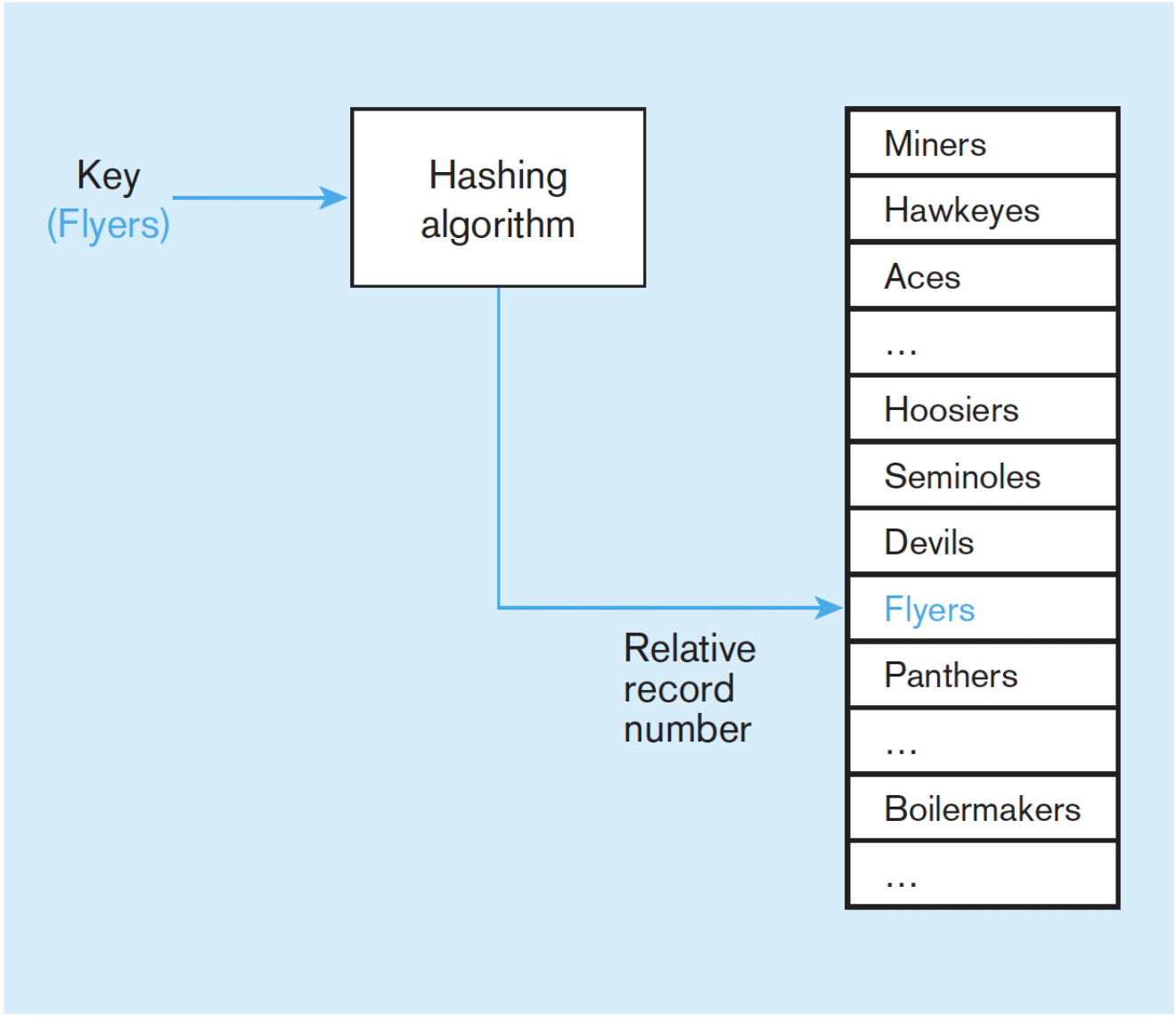

Key
(Flyers)
Hashing algorithm
Relative record number
Miners
Hawkeyes
Aces
…
Hoosiers
Seminoles
Devils
Flyers
Panthers
…
Boilermakers
…

Figure 5-8 **Join** Indexes – to speed up join operations

a) Join index for common non-key columns

Customer

RowID	Cust#	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
. . .				

Store

RowID	Store#	City	Size	Manager
20001	S4266	Dayton	K2	E2166
20002	S2654	Columbus	K3	E0245
20003	S3789	Dayton	K4	E3330
20004	S1941	Toledo	K1	E0874
. . .				

Join Index

CustRowID	StoreRowID	Common Value*
10001	20001	Dayton
10001	20003	Dayton
10002	20002	Columbus
10003	20002	Columbus
10004	20004	Toledo
. . .		

*This column may or may not be included, as needed. Join index could be sorted on any of the three columns. Sometimes two join indexes are created, one as above and one with the two RowID columns reversed.

b) Join index for matching foreign key (FK) and primary key (PK)

Order

RowID	Order#	Order Date	Cust#(FK)
30001	O5532	10/01/2015	C3861
30002	O3478	10/01/2015	C1062
30003	O8734	10/02/2015	C1062
30004	O9845	10/02/2015	C2027
. . .			

Customer

RowID	Cust#(PK)	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1062	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
. . .				

Join Index

CustRowID	OrderRowID	Cust#
10001	30004	C2027
10002	30002	C1062
10002	30003	C1062
10004	30001	C3861
. . .		

COMPARATIVE FEATURES OF FILE ORG.

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

UNIQUE AND NONUNIQUE INDEXES

- Unique (primary) Index
  - Typically done for primary keys, but could also apply to other unique fields

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

- Nonunique (secondary) index
  - Done for fields that are often used to group individual entities (e.g. zip code, product category)

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

RULES FOR USING INDEXES

1. Use on larger tables
2. Index the primary key of each table
3. Index search fields (fields frequently in WHERE clause)
   - Eg: WHERE ProductFinish =“Oak,” for which an index on ProductFinish would speed retrieval
4. Fields in SQL ORDER BY and GROUP BY commands
5. When there are >100 values but not when there are <30 values

RULES FOR USING INDEXES (CONT.)

6. Avoid use of indexes for fields with long values; perhaps compress values first
7. If key to index is used to determine location of record, use surrogate (like sequence number) to allow even spread in storage area. Many DBMSs create a sequence number so that when each row is inserted, sequence increases.
8. DBMS may have limit on number of indexes per table and number of bytes per indexed field(s)
9. Be careful of indexing attributes with null values; many DBMSs will not recognize null values in an index search