(ADVANCED) DATABASE SYSTEMS (DATABASE MANAGEMENTS)

PROF. DR. HASAN HÜSEYİN BALIK

(5TH WEEK)

3. OUTLINE

3. Database Design

3.1 Logical Database Design and the Relational Model

3.2 Physical Database Design and Performance

3.2 PHYSICAL DATABASE DESIGN AND PERFORMANCE

OBJECTİVES

- 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

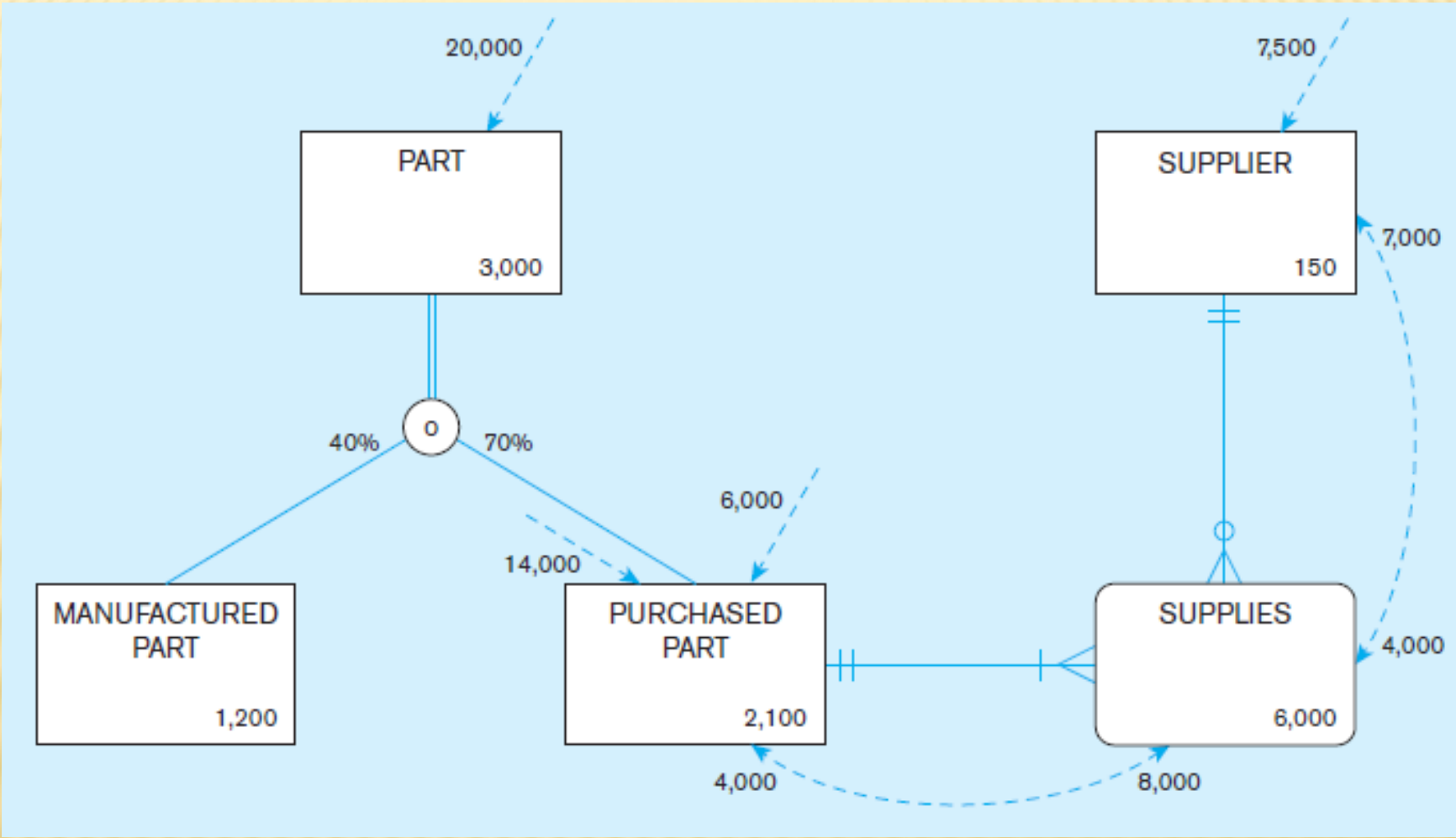
- Attribute data types
- Physical record descriptions (doesn't always match logical design)
- File organizations
- Indexes and database architectures
- Query optimization

PHYSICAL DESIGN FOR REGULATORY COMPLIANCE

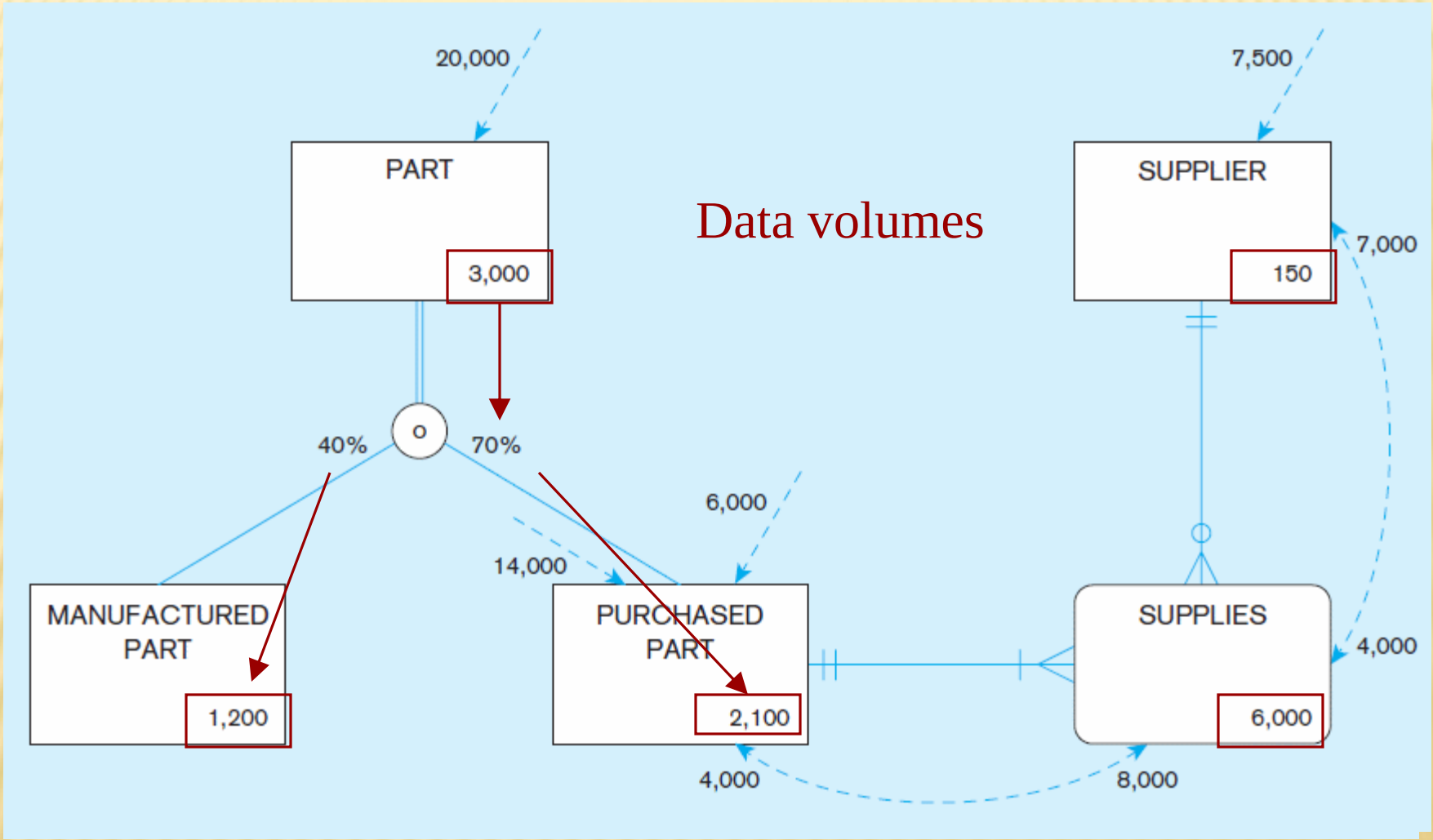
- Sarbanes- Oxley Act (SOX) – protect investors by improving accuracy and reliability
- Committee of Sponsoring Organizations (COSO) of the Treadway Commission
- IT Infrastructure Library (ITIL)
- Control Objectives for Information and Related Technology (COBIT)

Regulations and standards that impact physical design decisions

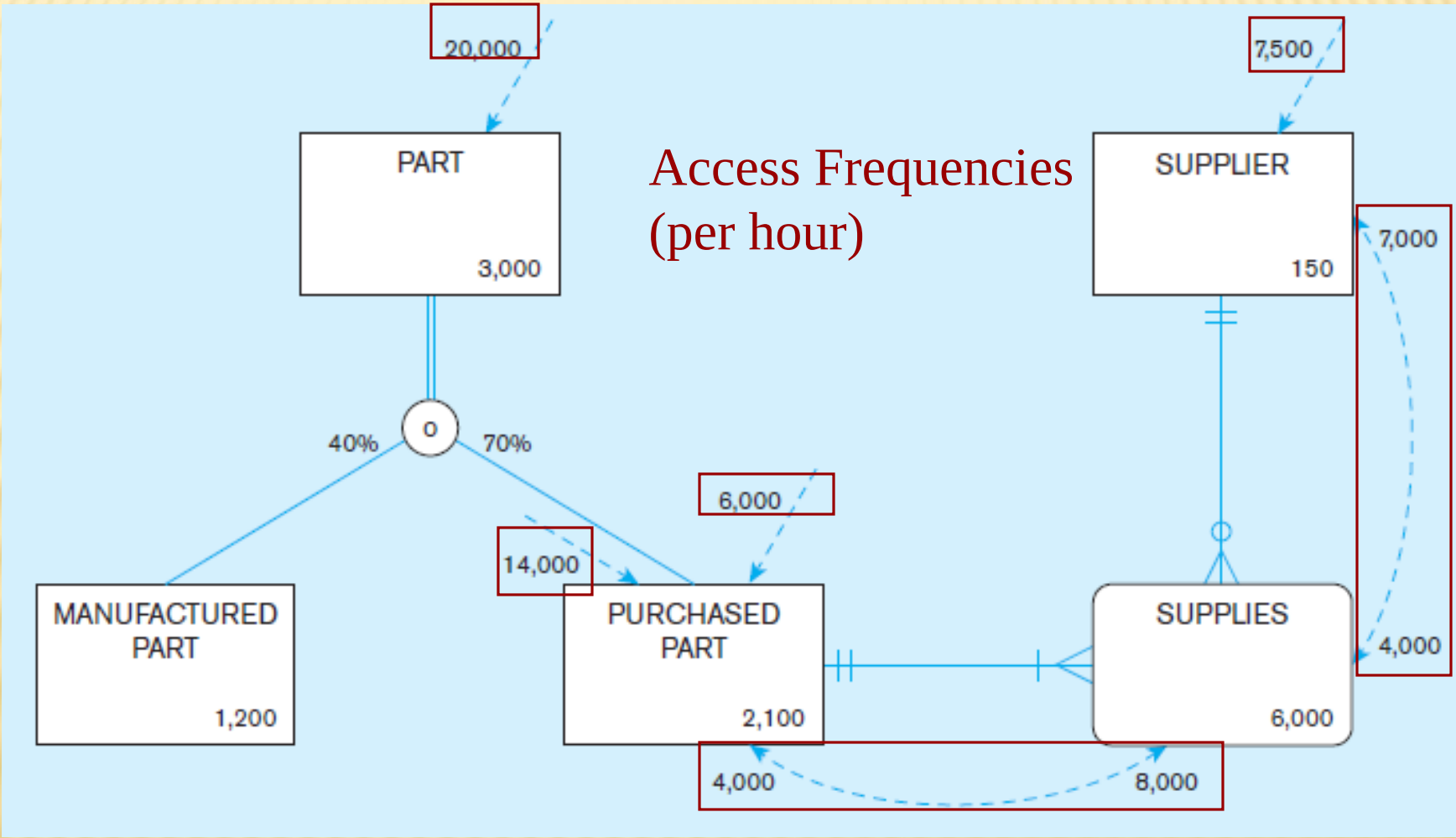
Composite usage map
(Pine Valley Furniture Company)



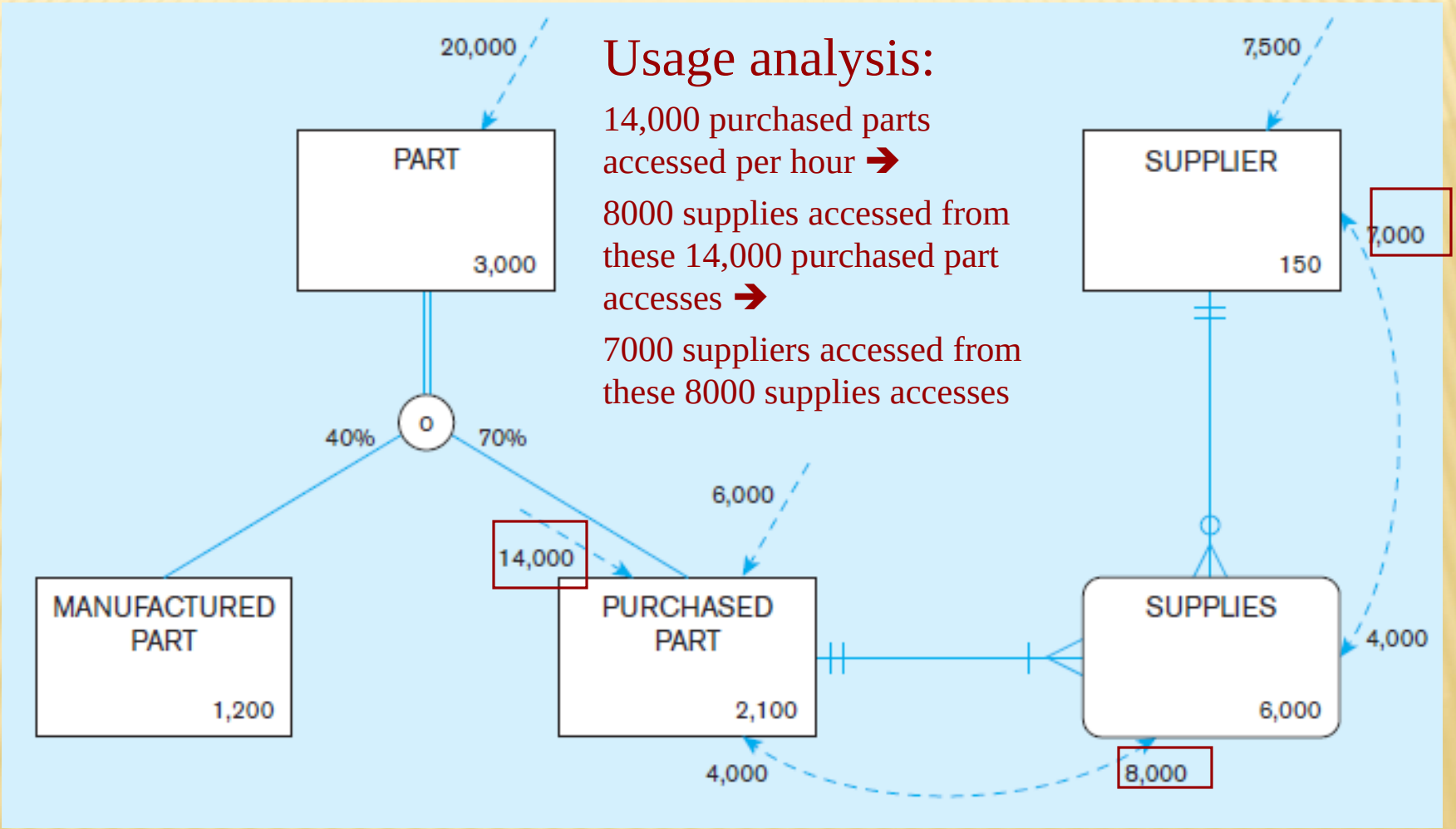
Composite usage map (Pine Valley Furniture Company) (cont.)



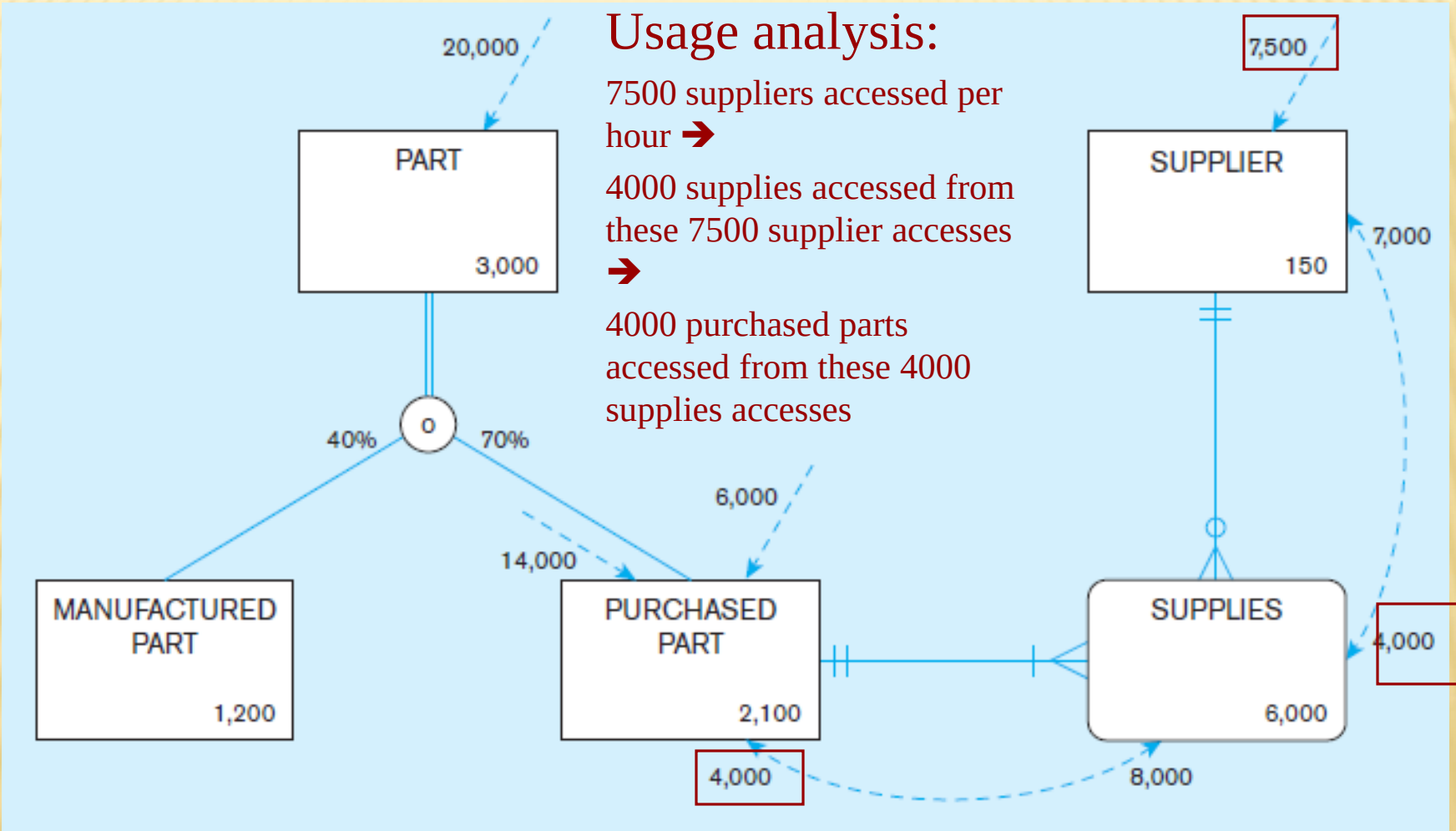
Composite usage map (Pine Valley Furniture Company) (cont.)



Composite usage map (Pine Valley Furniture Company) (cont.)



Composite usage map (Pine Valley Furniture Company) (cont.)



DESIGNING FIELDS

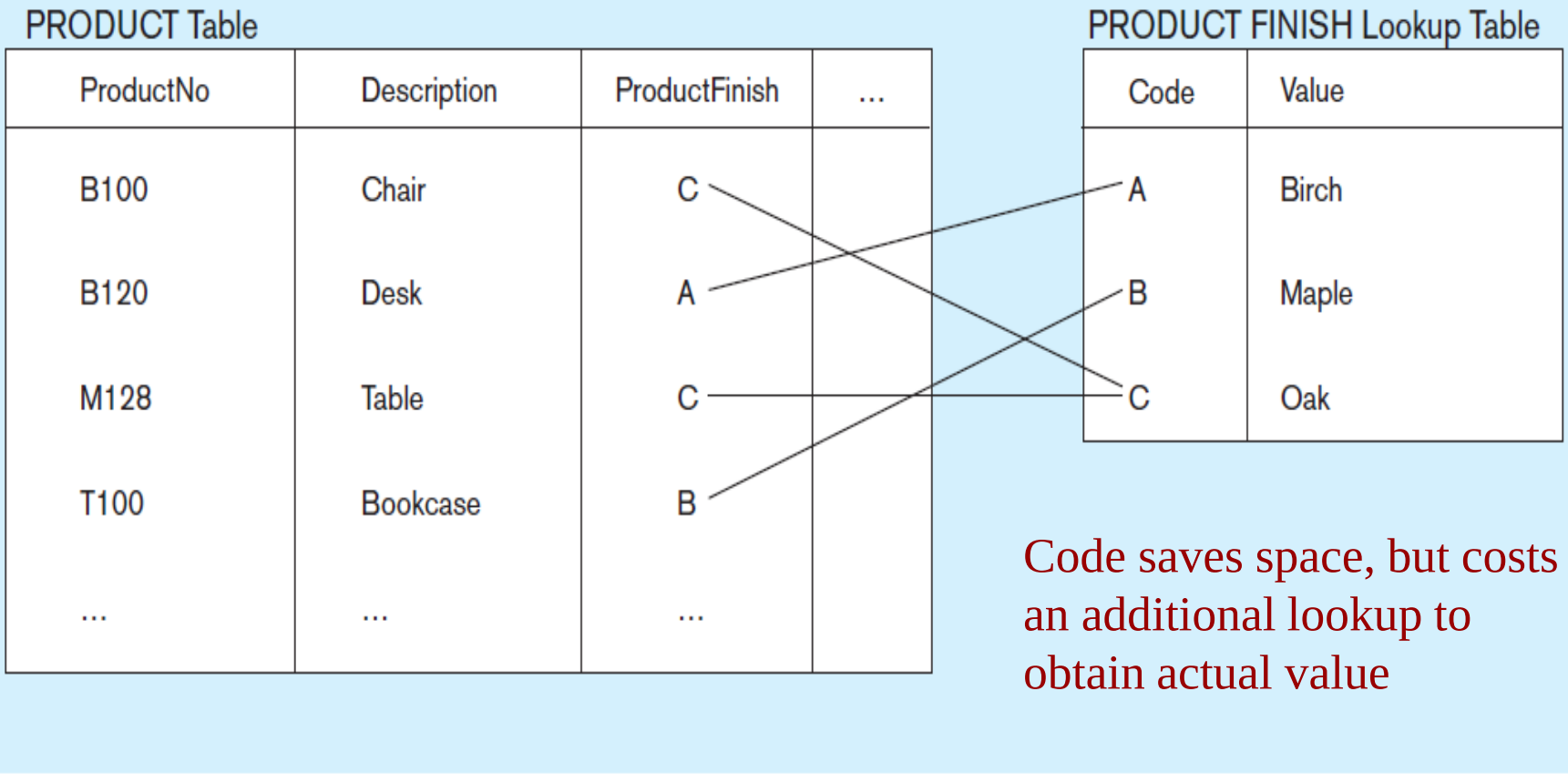
- Field: smallest unit of application data recognized by system software
- Field design
  - Choosing data type
  - Coding, compression, encryption
  - Controlling data integrity

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).

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



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

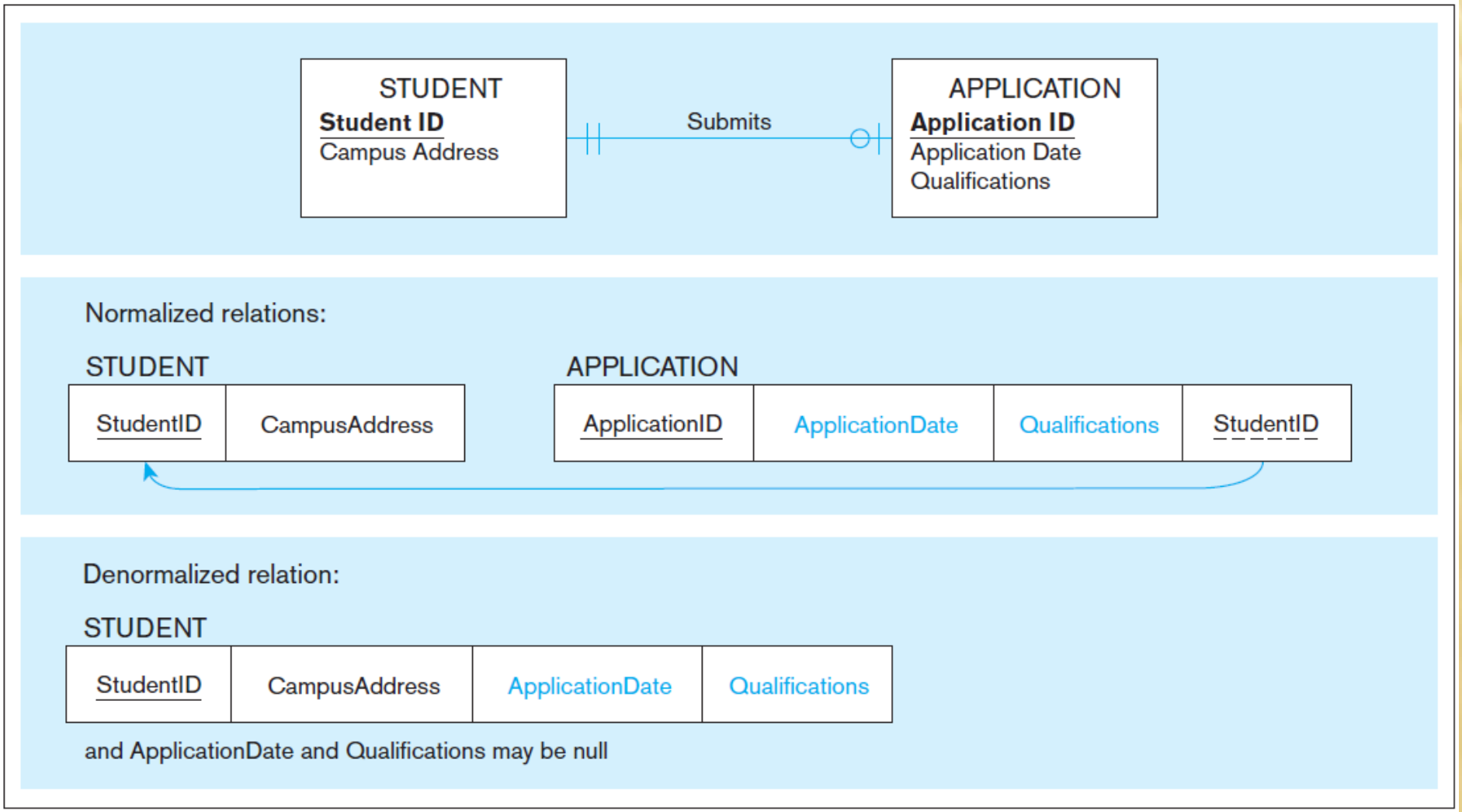
- Substitute an estimate of the missing value (e.g., using a formula)
- Construct a report listing missing values
- In programs, ignore missing data unless the value is significant (sensitivity testing)

Triggers can be used to perform these operations.

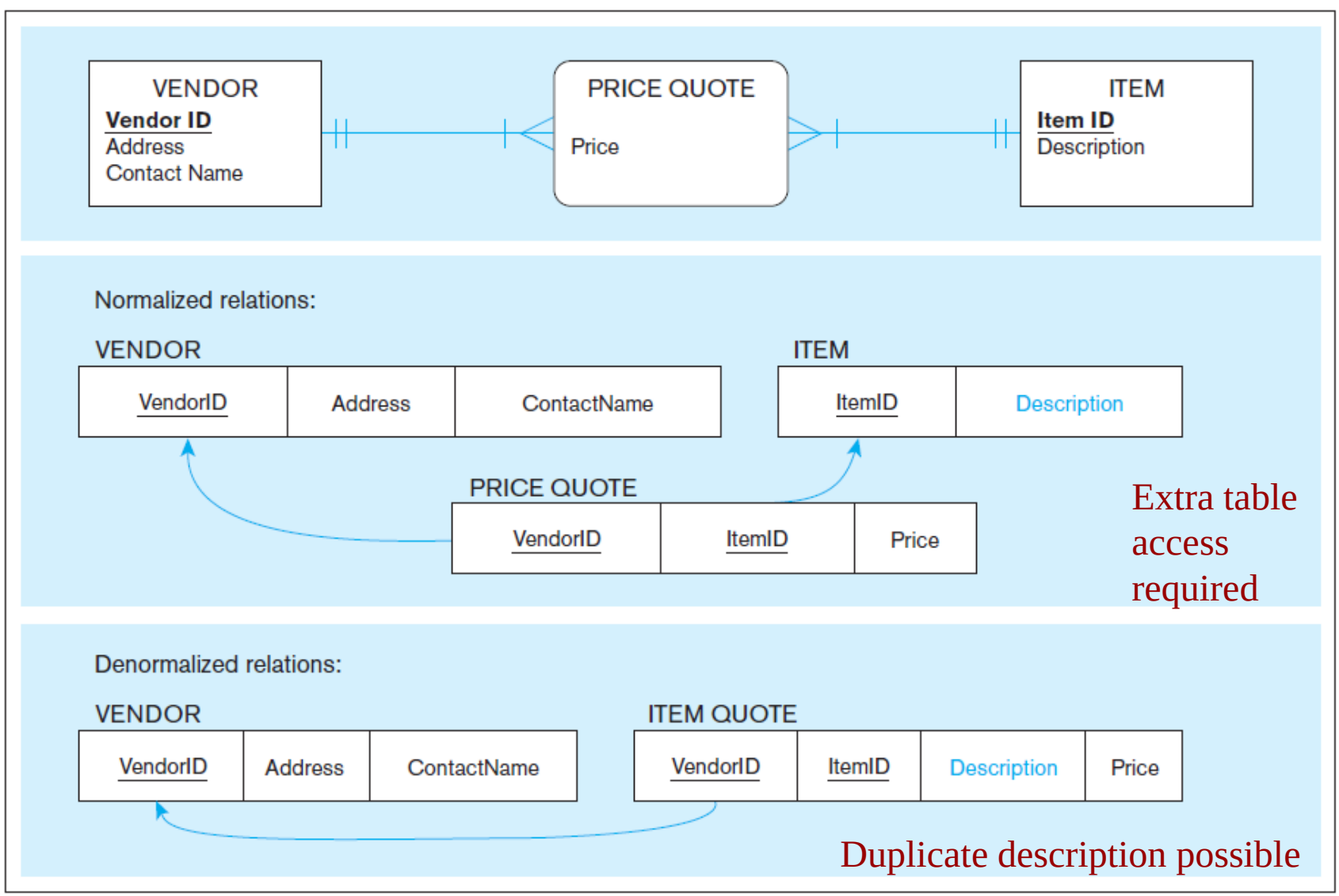
DENORMALIZATION

- 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)

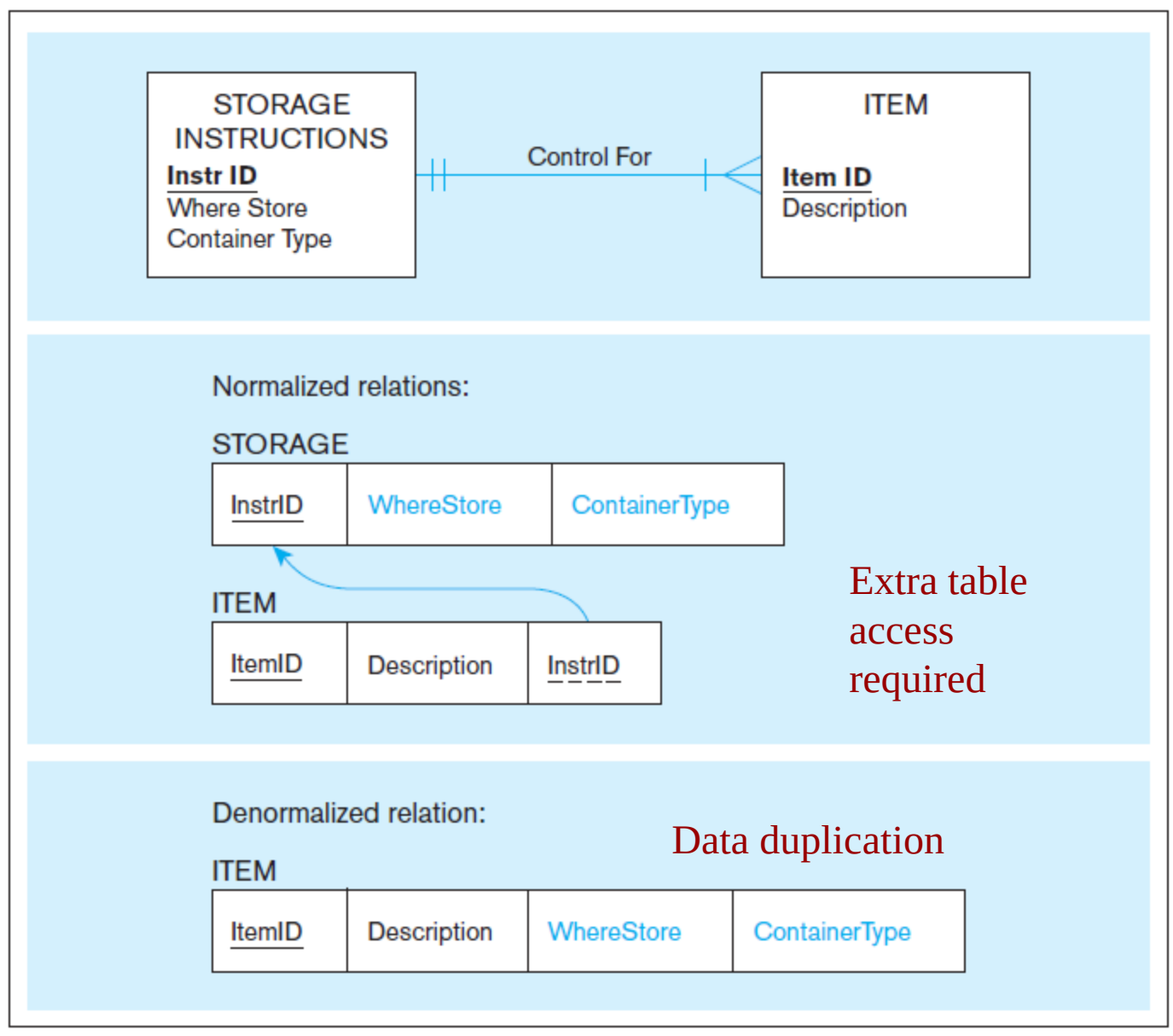
A possible denormalization situation: two entities with one-to-one relationship



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



A possible denormalization situation: reference data



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

PARTITIONING PROS AND CONS

- Advantages of Partitioning:
  - Efficiency: Records used together are grouped together
  - Local optimization: Each partition can be optimized for performance
  - Security: data not relevant to users are segregated
  - Recovery and uptime: smaller files take less time to back up
  - Load balancing: Partitions stored on different disks, reduces contention
- 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

ORACLE'S HORİZONTAL PARTITIONING

- Range partitioning
  - Partitions defined by range of field values
  - Could result in unbalanced distribution of rows
  - Like-valued fields share partitions
- Hash partitioning
  - Partitions defined via hash functions
  - Will guarantee balanced distribution of rows
  - Partition could contain widely varying valued fields
- List partitioning
  - Based on predefined lists of values for the partitioning key
- Composite partitioning
  - Combination of the other approaches

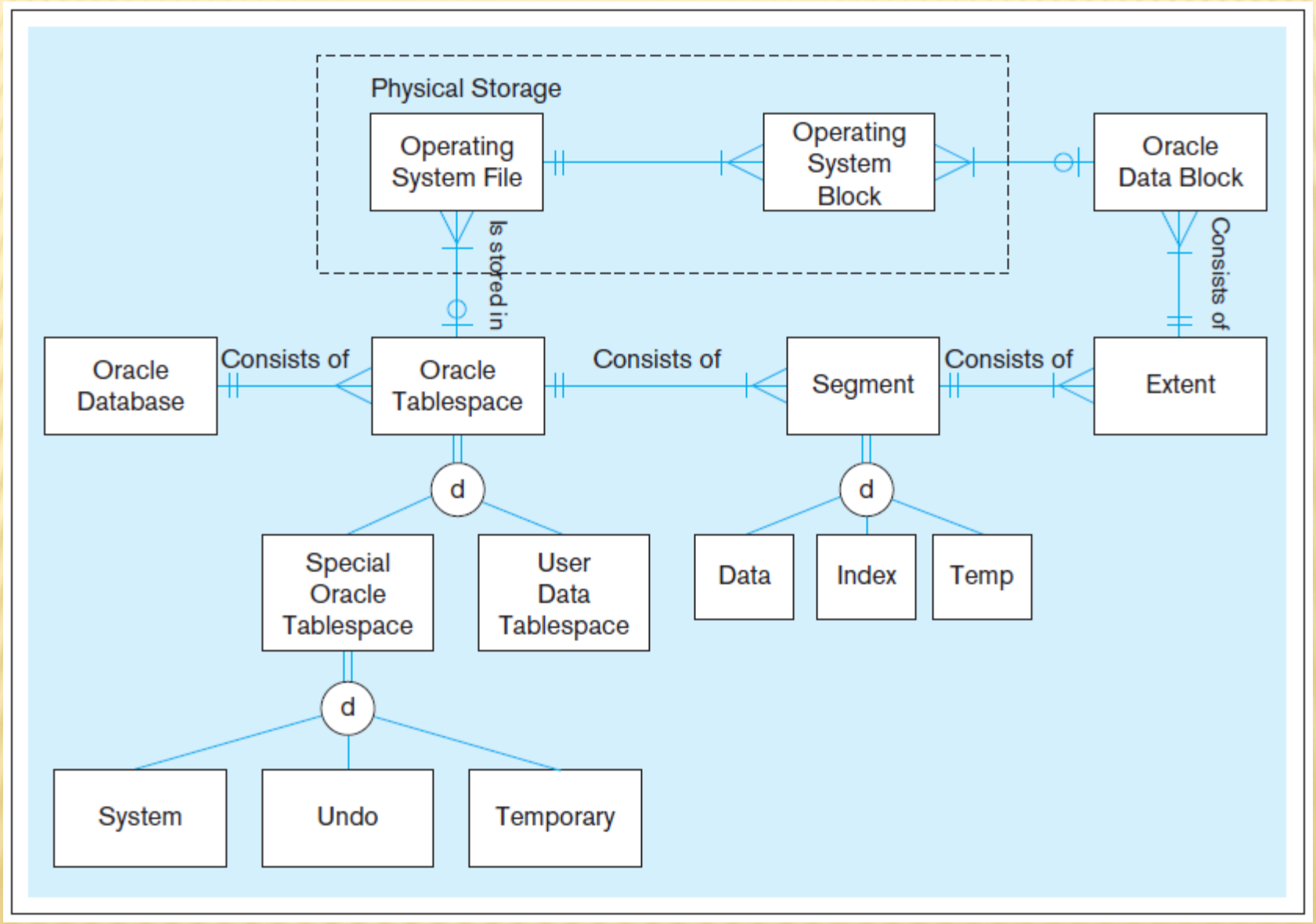
VERTICAL PARTITIONING

- 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.

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

DBMS terminology in an Oracle 12c environment

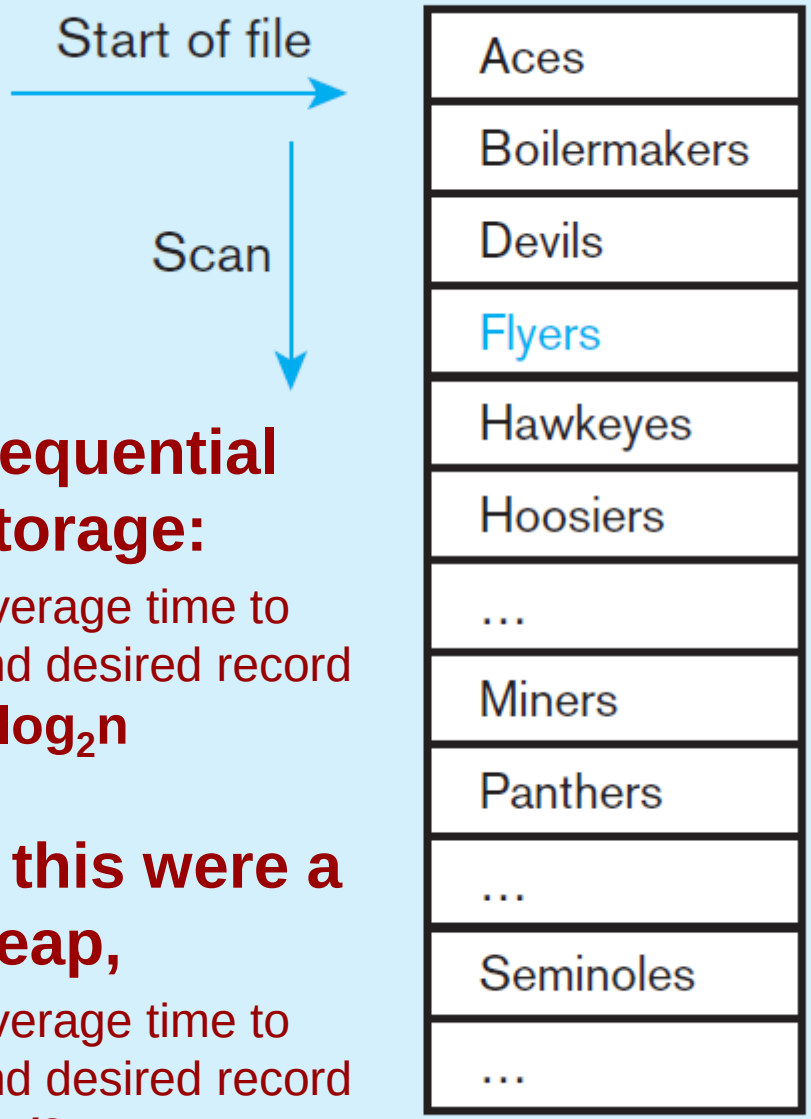


FILE ORGANIZATIONS

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

Sequential file organization

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



Sequential storage:

Average time to find desired record = $\log_2 n$

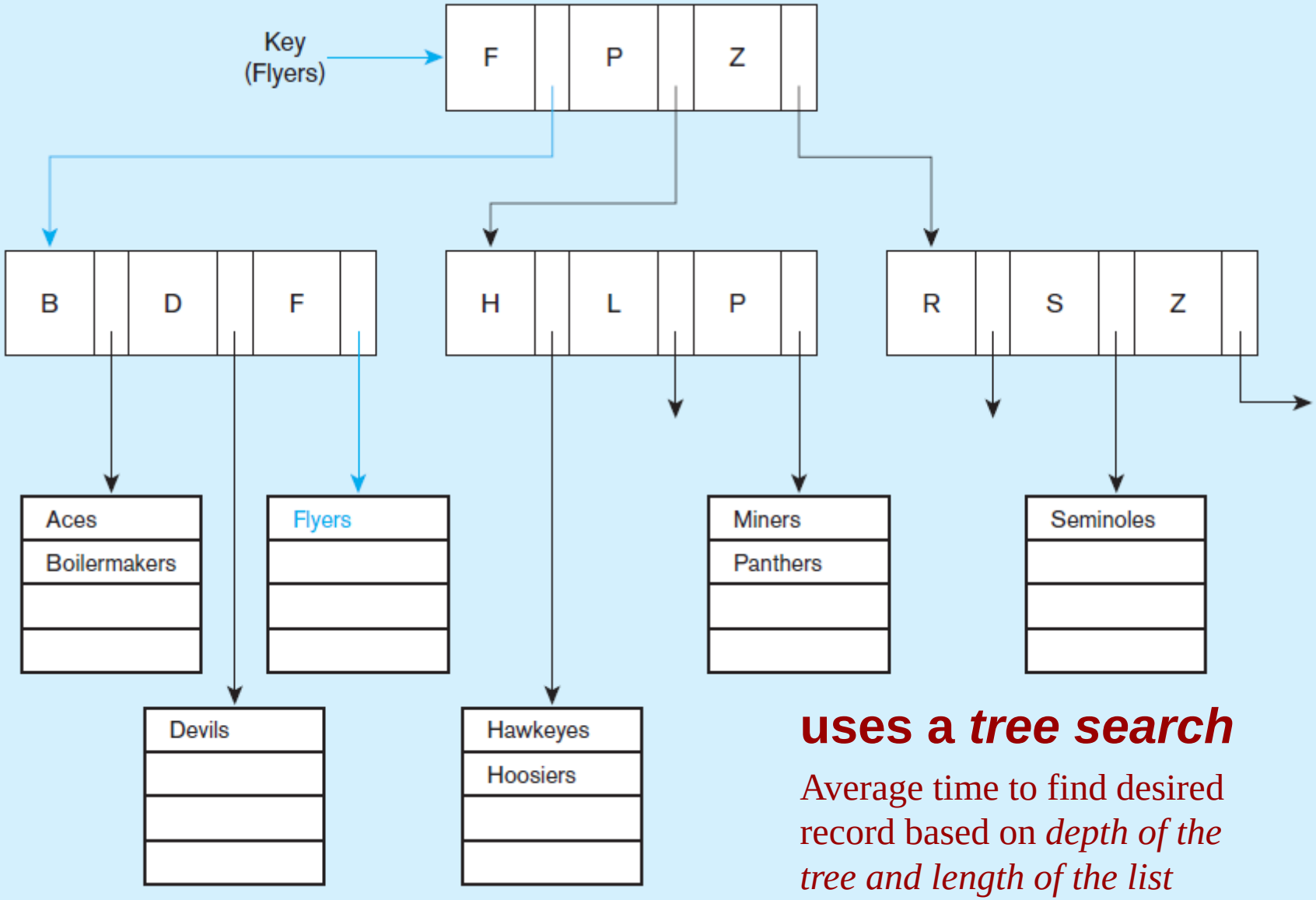
If this were a heap,

Average time to find desired record = **n/2**

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)

Indexed file organization



uses a *tree search*

Average time to find desired record based on *depth of the tree and length of the list*

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
. . .			

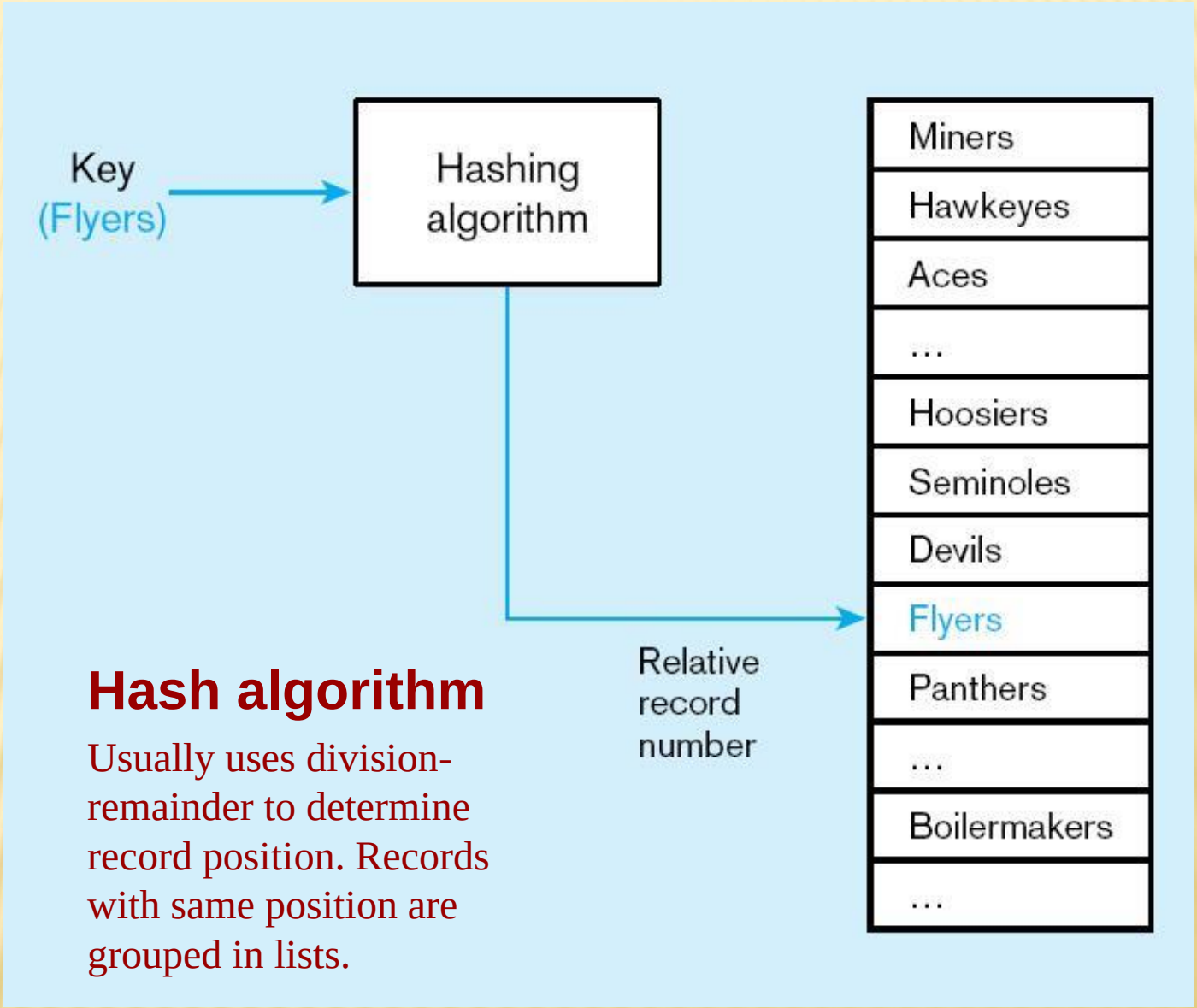
Customer

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
. . .		

Hashed file organization



Hash algorithm

Usually uses division-remainder to determine record position. Records with same position are grouped in lists.

TABLE 5-3 Comparative Features of Different File Organizations

	File Organization			
Factor	**Heap**	**Sequential**	**Indexed**	**Hashed**
Storage space	No wasted 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	Requires sorting	Very fast	Moderately fast	Impractical, unless using a hash index
Random retrieval on primary key	Impractical	Impractical	Moderately fast	Very fast
Multiple-key retrieval	Possible but requires scanning whole file	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 requires reorganization	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	Very easy	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.	Usually requires rewriting a file	Easy but requires maintenance of indexes	Very easy

CLUSTERING FILES

- In some relational DBMSs, related records from different tables can be stored together in the same disk area
- Useful for improving performance of join operations
- Primary key records of the main table are stored adjacent to associated foreign key records of the dependent table
- e.g. Oracle has a CREATE CLUSTER command

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)
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
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

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