this lecture is cancelled

Chapter 6:
Physical Database Design and Performance

Modern Database Management
8th Edition

Jeffrey A. Hoffer, Mary B. Prescott, Fred R. McFadden

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
- Data Volume estimates
- Frequency of using data
- 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)

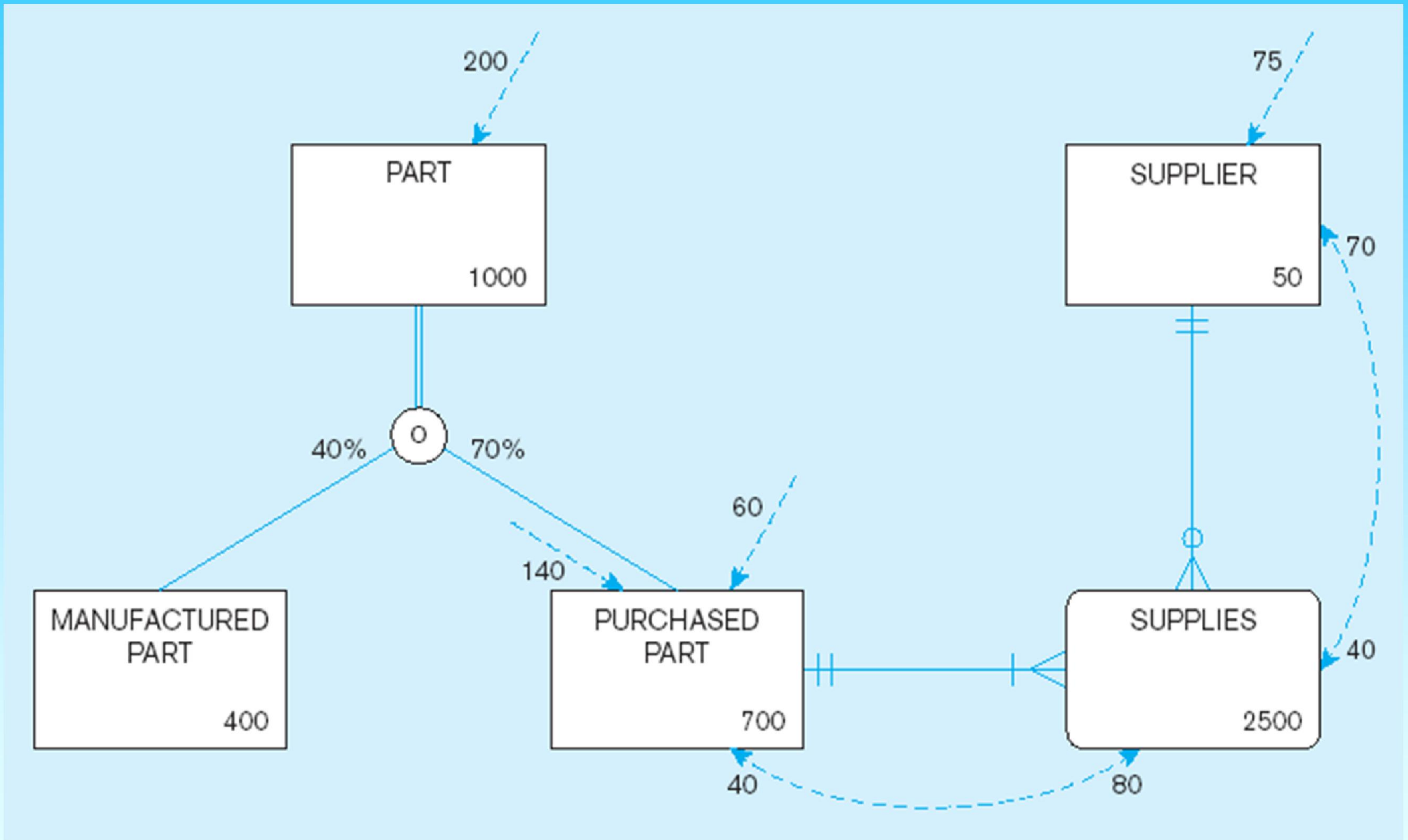


Figure 6-1 Composite usage map (Pine Valley Furniture Company) (cont.)

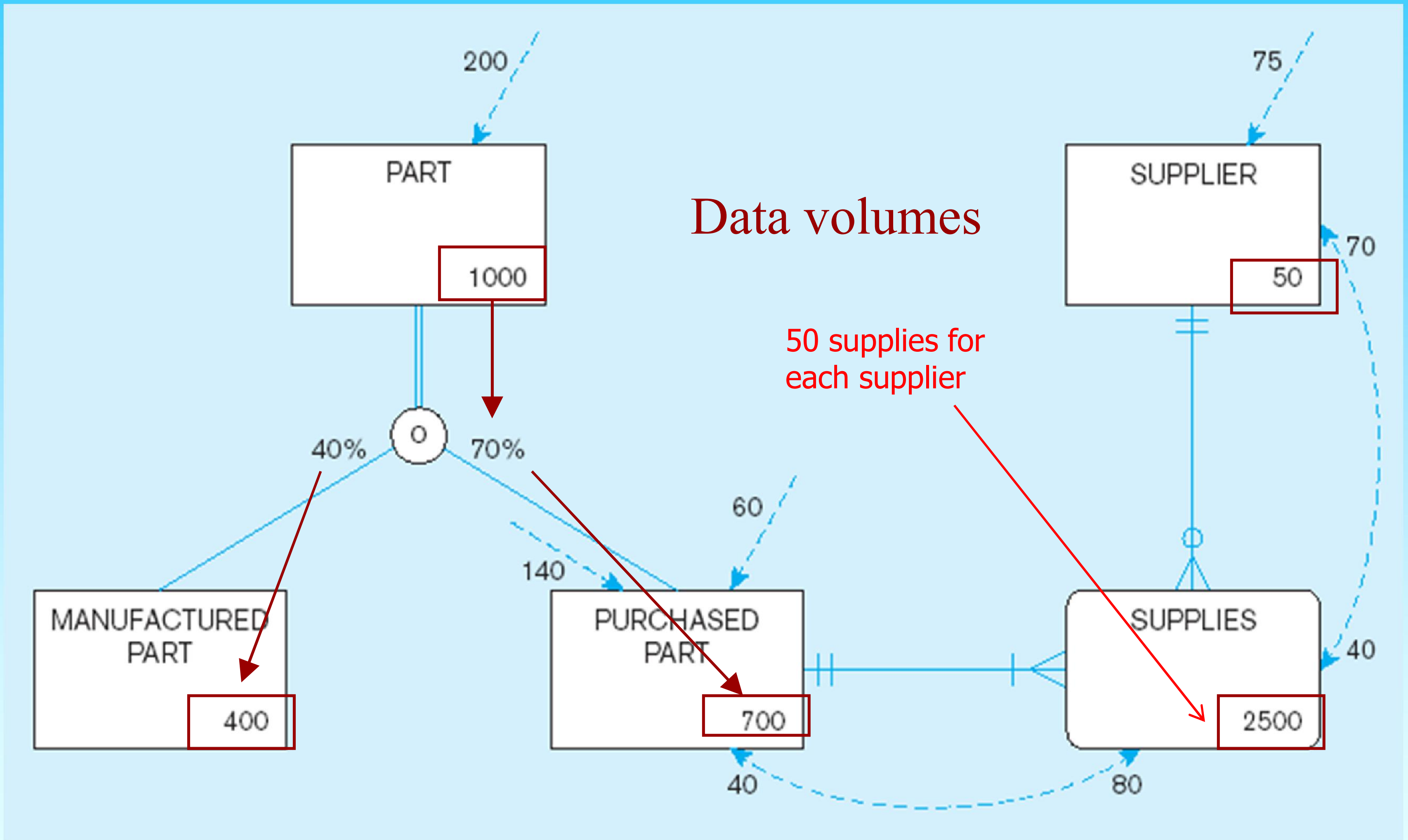


Figure 6-1 Composite usage map (Pine Valley Furniture Company) (cont.)

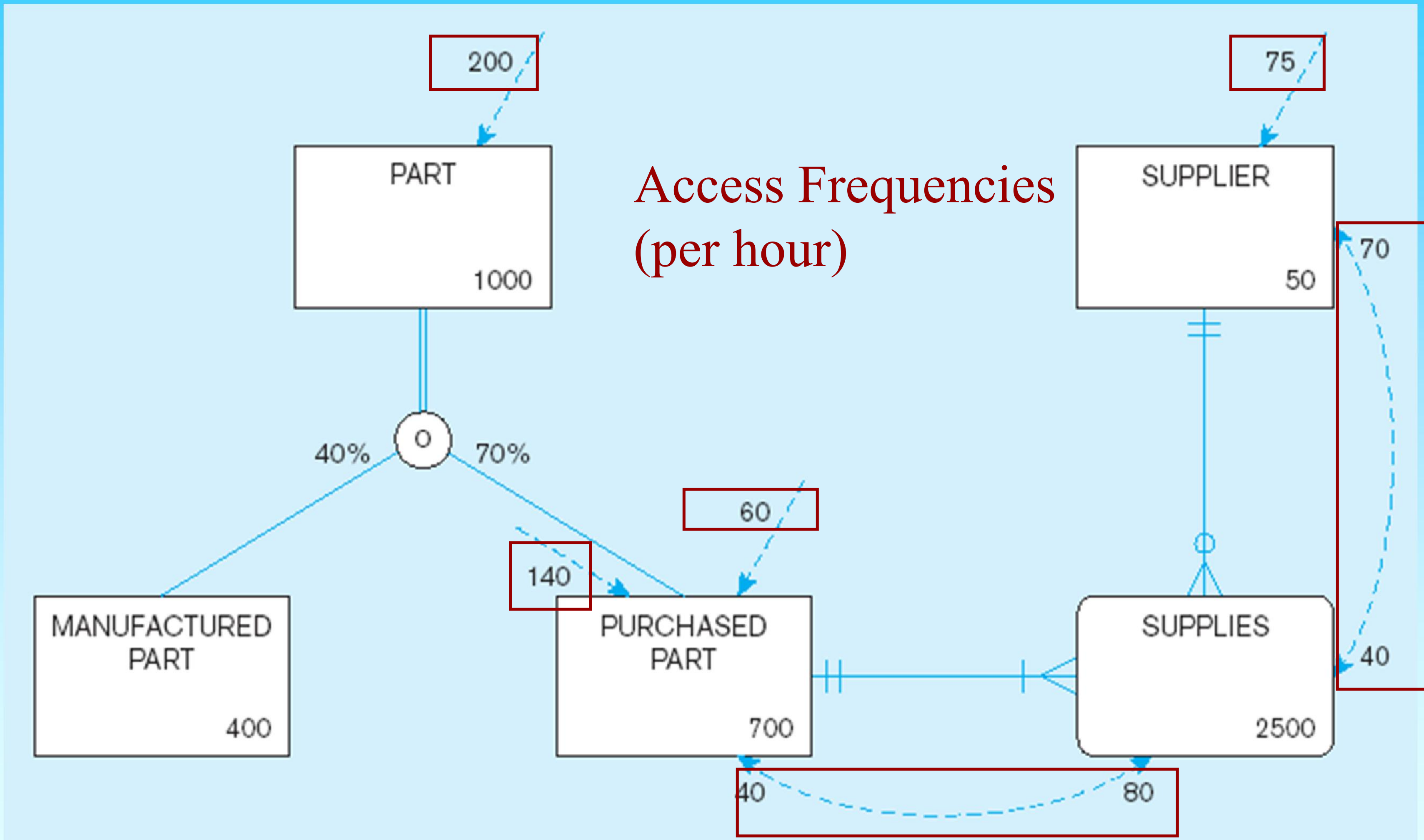
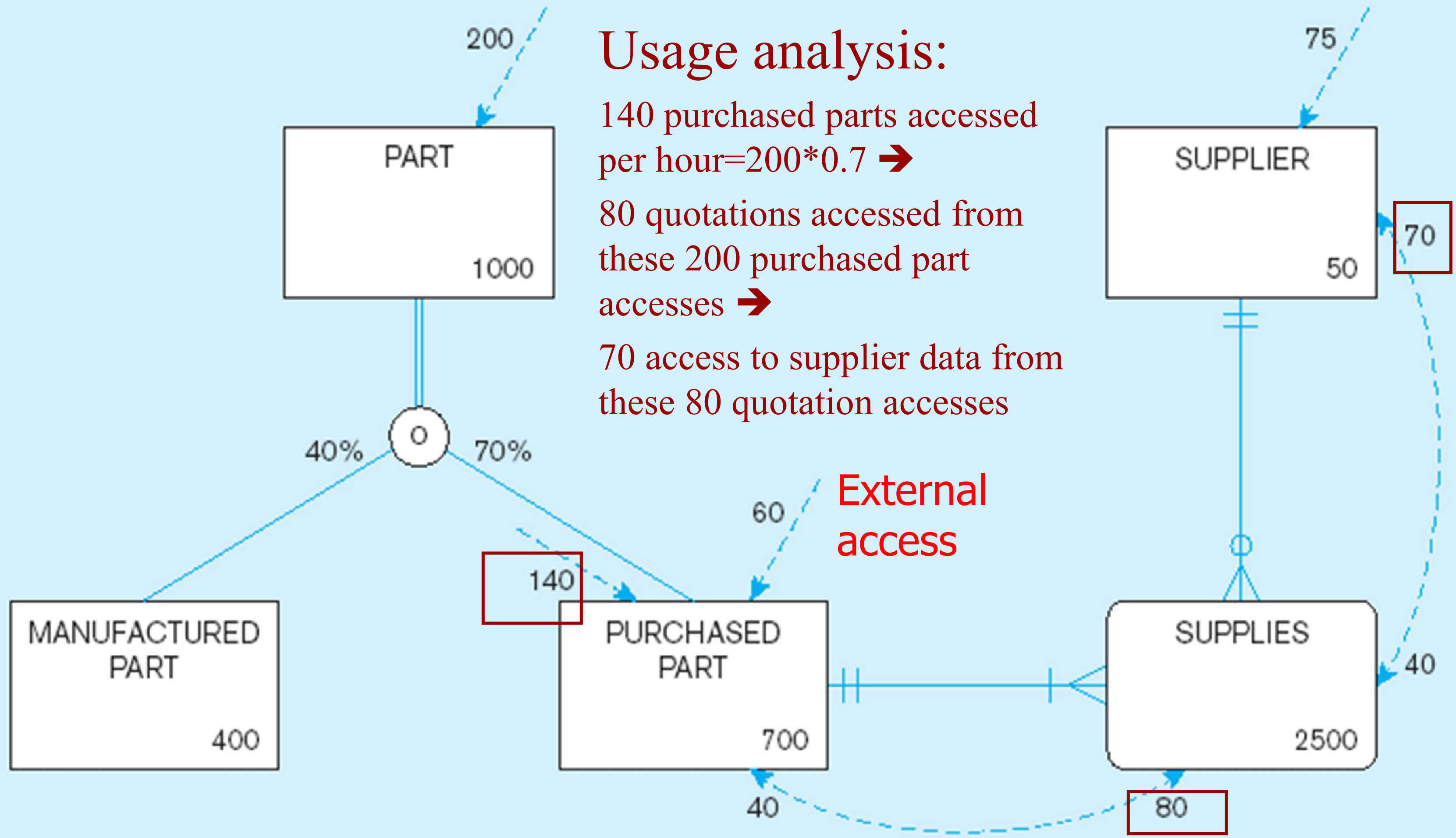


Figure 6-1 Composite usage map (Pine Valley Furniture Company) (cont.)



Usage analysis:

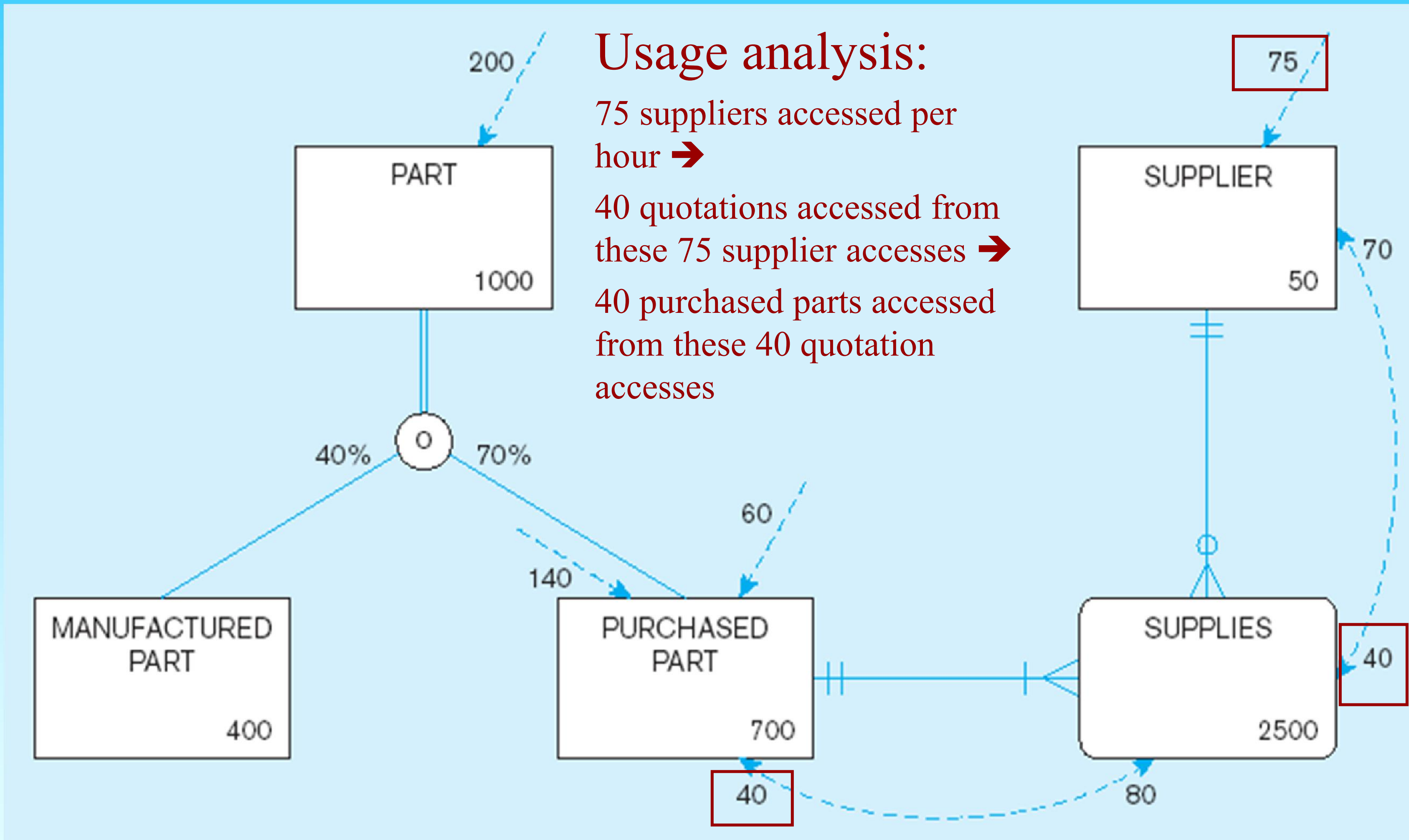
140 purchased parts accessed per hour=200*0.7 ➔

80 quotations accessed from these 200 purchased part accesses ➔

70 access to supplier data from these 80 quotation accesses

External access

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

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 by an Operating System in one I/O operation
- Blocking Factor: The number of physical records per page

Performance Issues

- Usually, not all attributes of a table are used in a query or report.
- Instead, attributes from different tables are accessed in a query or report.
- This makes a DBMS to consume many resources and spend considerable amount of time to execute a query based on multiple tables.

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 (data maintenance anomalies)
- 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

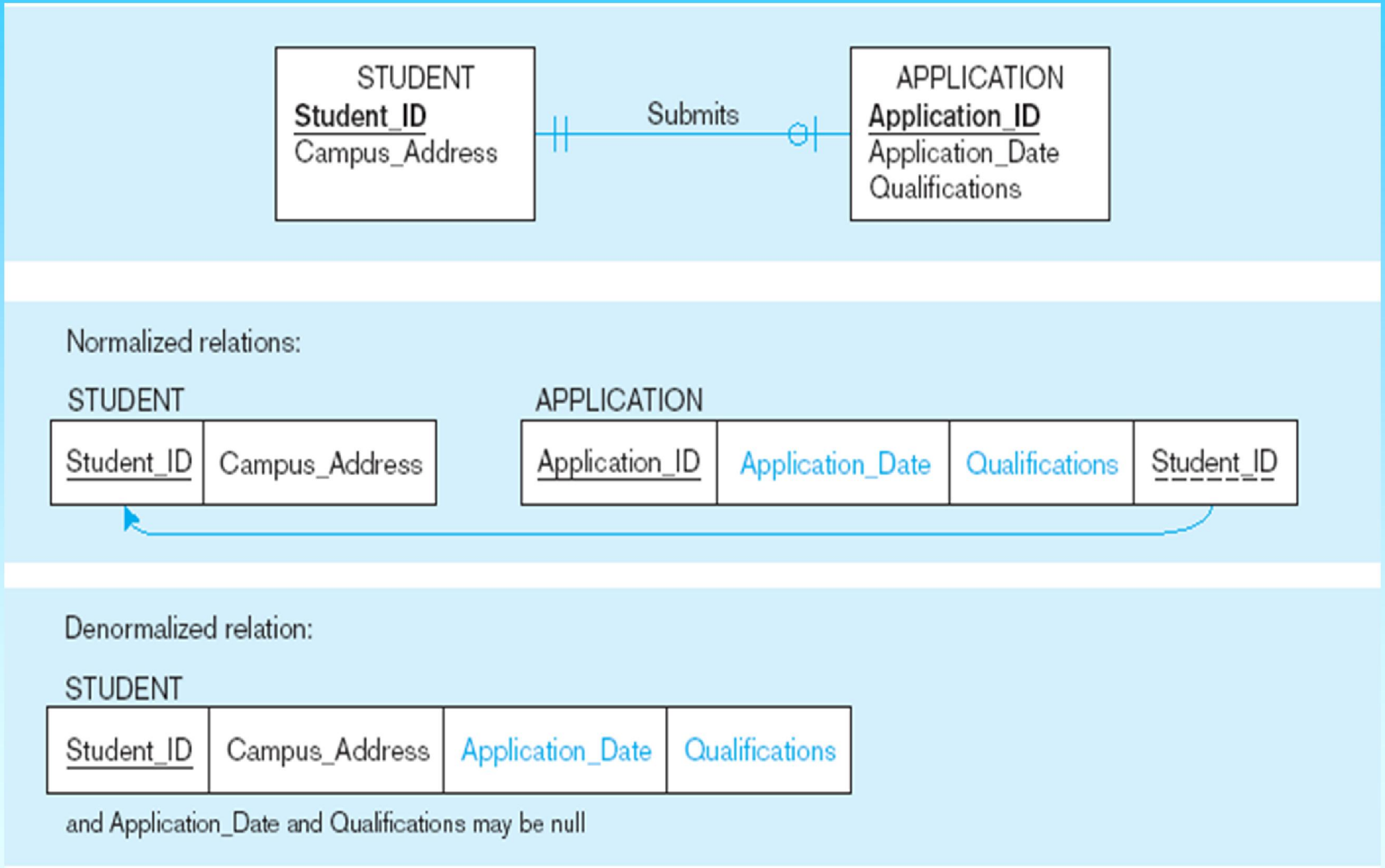


Figure 6-4 A possible denormalization situation: a many-to-many relationship with nonkey attributes

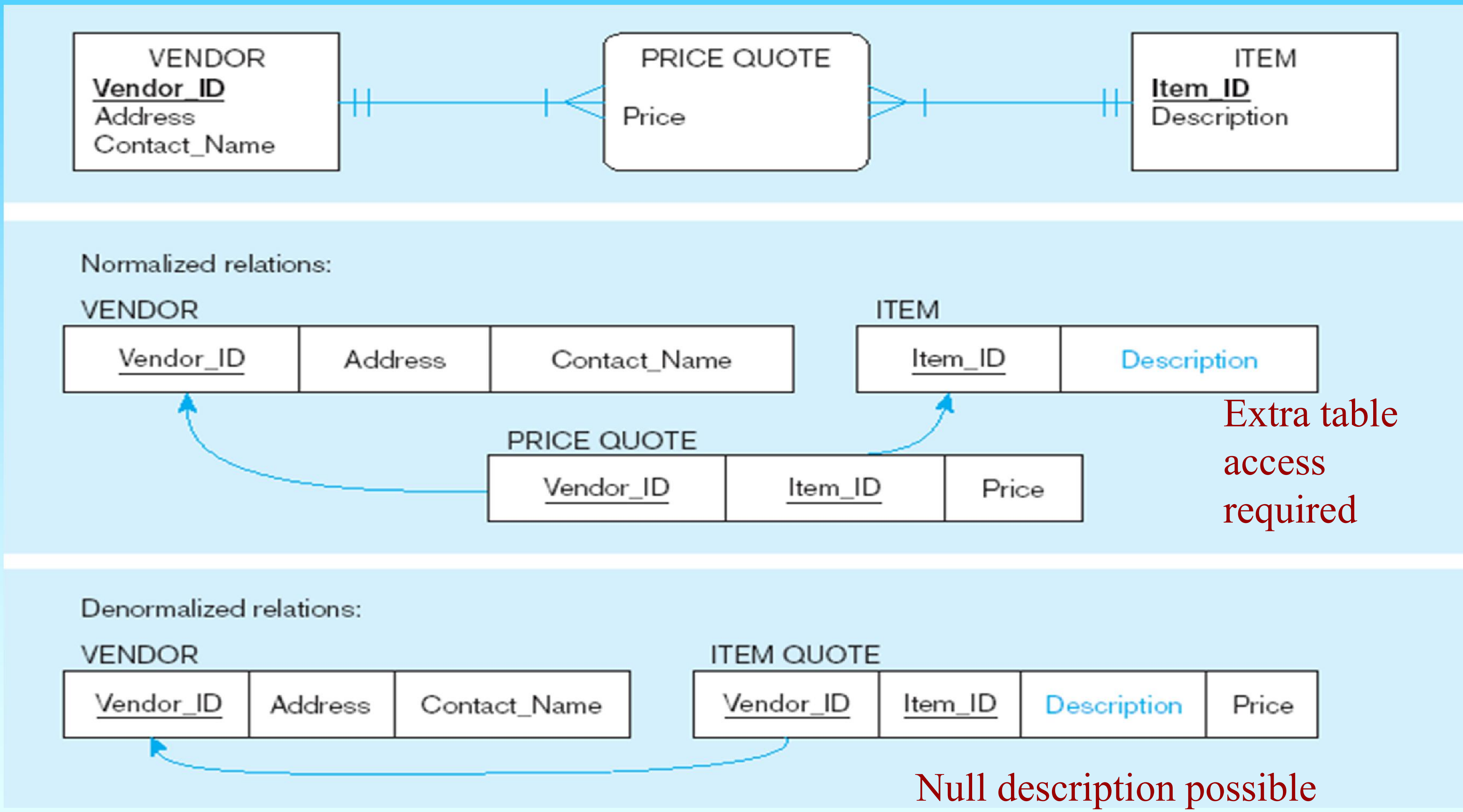


Figure 6-5
A possible denormalization situation: reference data

Normalized relations:

STORAGE

Instr_ID	Where_Store	Container_Type

ITEM

Item_ID	Description	Instr_ID

Extra table access required

Denormalized relation:

Data duplication

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 (grouping of data as 1st, 2nd, 3rd year students or Department)
  - It improves both security and performance.
- 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

Partitions often correspond with User Schemas (user views)

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

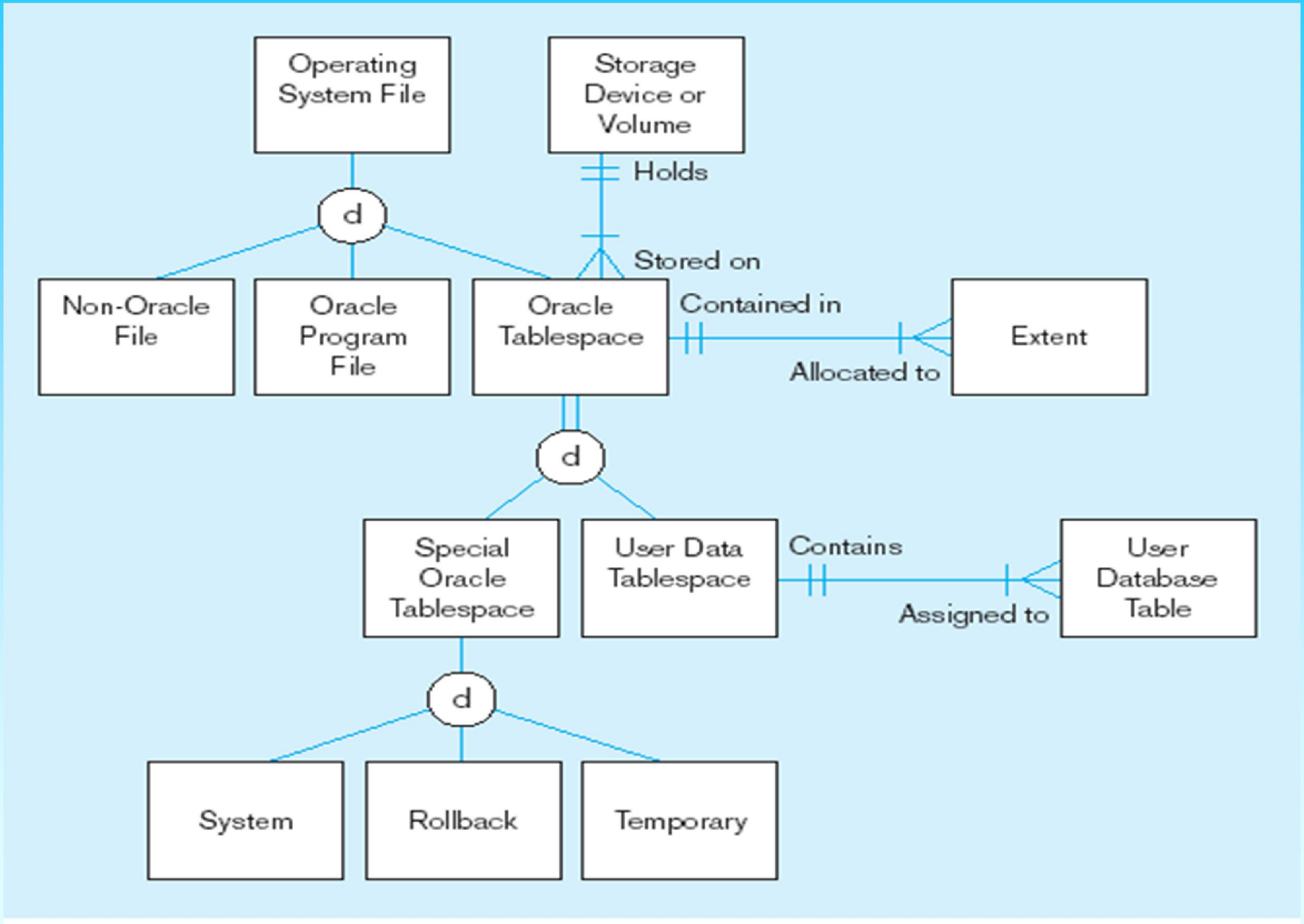
Data Replication

- 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

Designing Physical Files

- Physical File:
  - A named portion of secondary memory allocated for the purpose of storing physical records
  - Tablespace – named set of disk storage elements in which physical files for database tables can be stored
  - Extent – contiguous section of disk space
- Constructs to link two pieces of data:
  - Sequential storage
  - Pointers – field of data that can be used to locate related fields or records

Figure 6-4 Physical file terminology in an Oracle environment

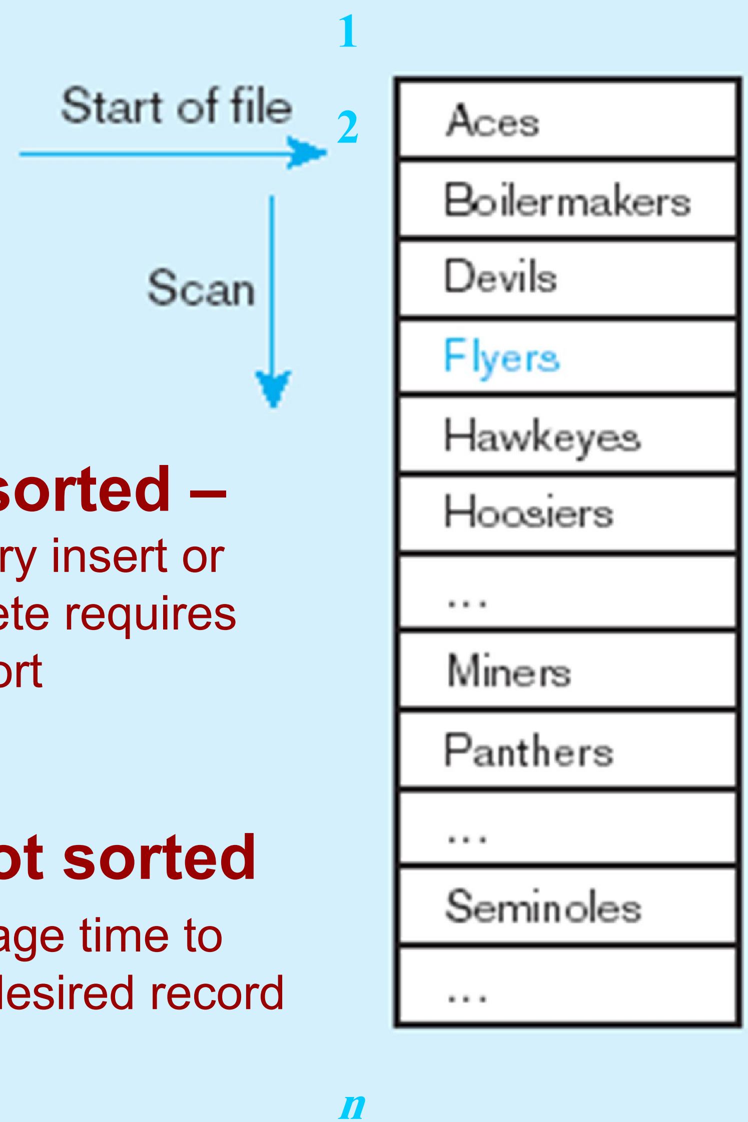


File Organizations

- Technique for physically arranging records of a file on secondary storage devices.
- Factors for selecting file organization:
  - Fast data retrieval
  - High throughput انتاجية for processing data input
  - Efficient storage space utilization
  - Protection from failure and data loss
  - Minimizing need for reorganization
  - Accommodating growth
  - Security from unauthorized use
- Types of file organizations
  - Sequential
  - Indexed
  - Hashed

Figure 6-7a
Sequential file organization

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



If sorted –
every insert or delete requires resort

If not sorted
Average time to find desired record = n/2

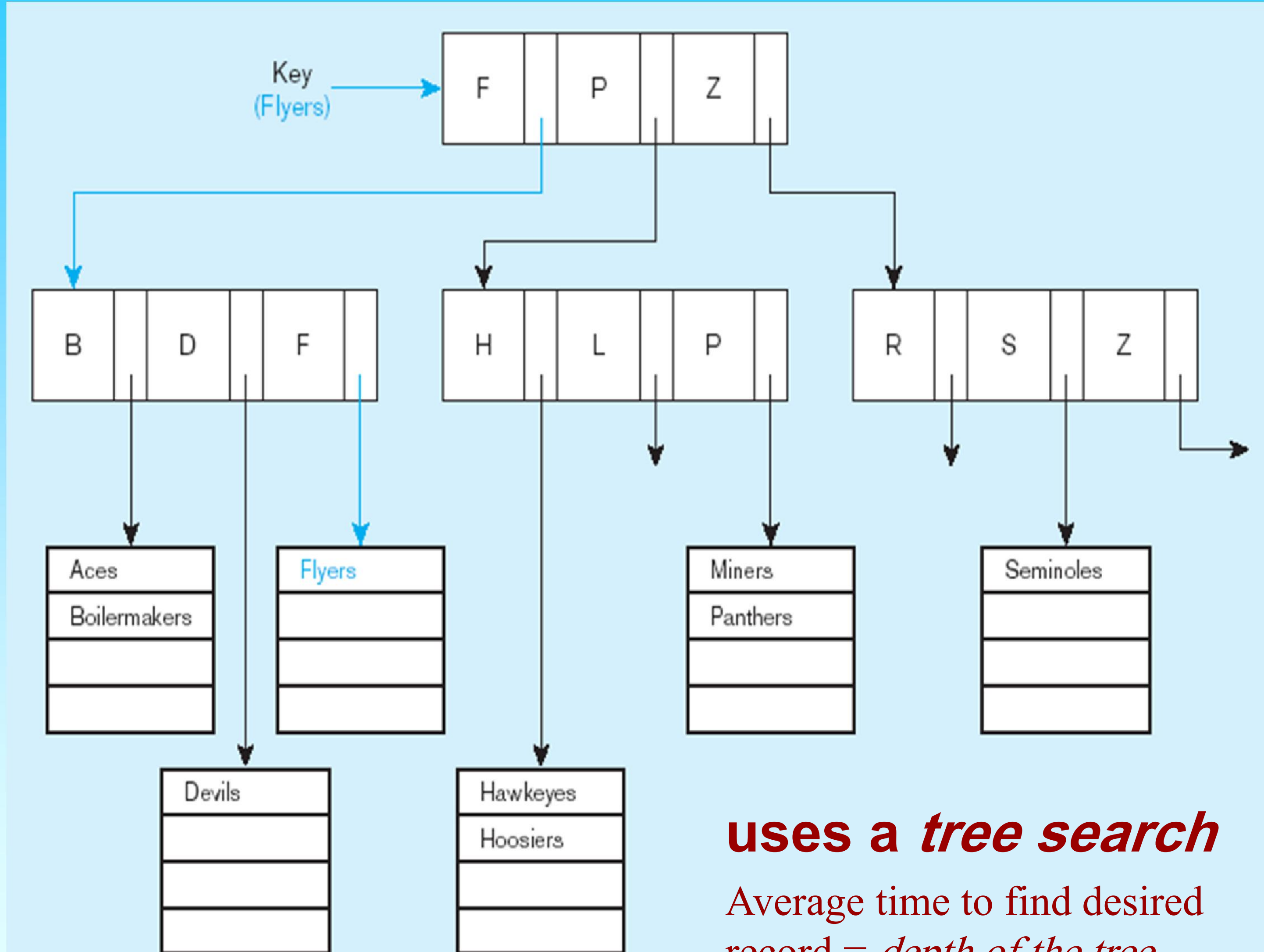
Indexed File Organizations

- Index – a separate table that used to determine the location of records in a file for quick retrieval
- Primary keys are automatically indexed
- Oracle has a CREATE INDEX operation, and MS ACCESS allows indexes to be created for most field types
- Indexing approaches:
  - B-tree , Balanced Tree index, Fig. 6-7b
  - Bitmap index, Fig. 6-8
  - Hash Index, Fig. 6-7c
  - Join Index, Fig 6-9

Figure 6-7b B-tree index (Balanced Tree) Hierarchical Index

When the index file is also too large, it can be indexed, (index of an index)

Leaves of the tree are all at same level → consistent access time

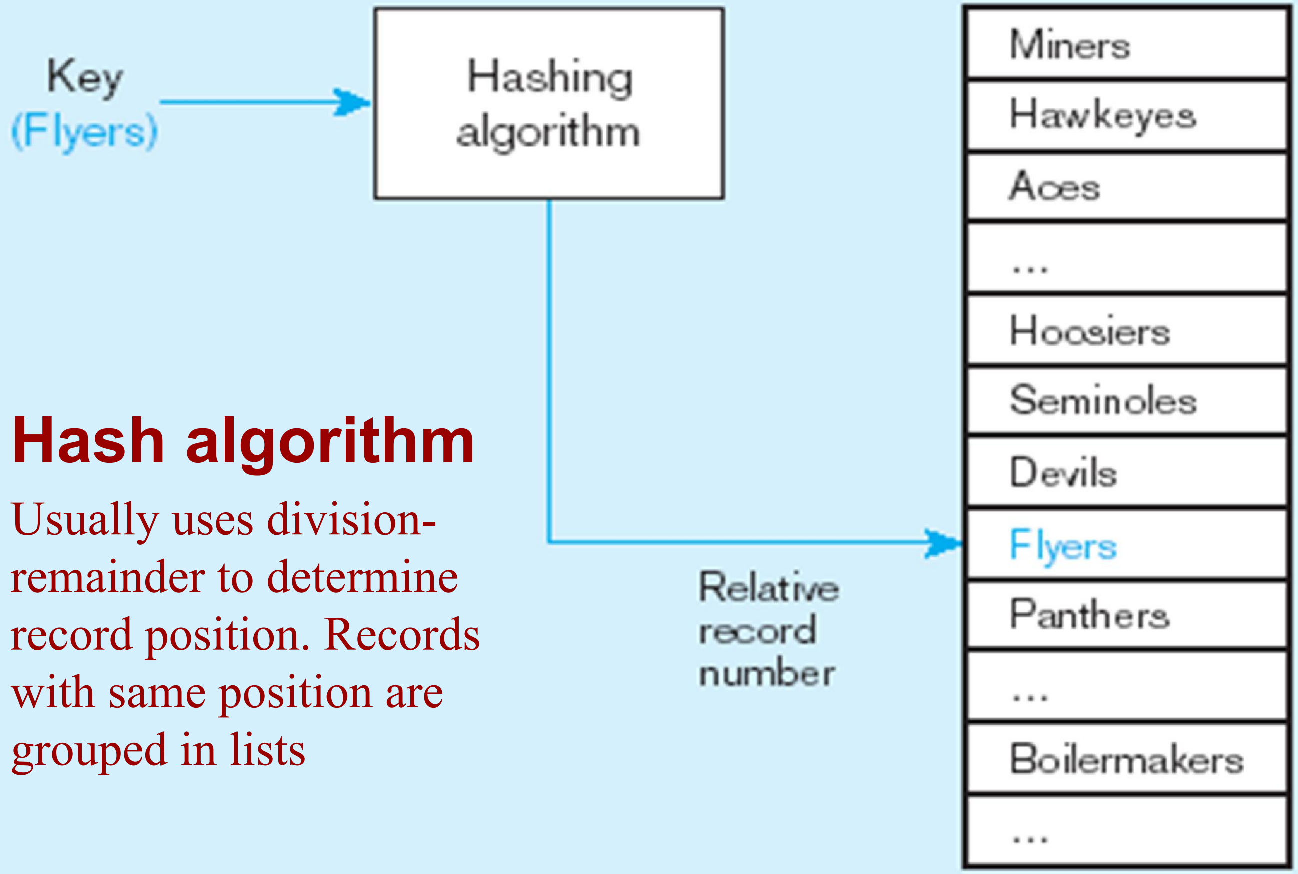


uses a *tree search*

Average time to find desired record = *depth of the tree*

Figure 6-7c

Hashed file or index organization



Hash algorithm

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

Figure 6-8
Bitmap index index organization

Bitmap saves on space requirements

Rows - possible values of the attribute

Columns - table rows

Bit indicates whether the attribute of a row has the values

	Product Table Row Numbers									
Price	1	2	3	4	5	6	7	8	9	10
100	0	0	1	0	1	0	0	0	0	0
200	1	0	0	0	0	0	0	0	0	0
300	0	1	0	0	0	0	1	0	0	1
400	0	0	0	1	0	1	0	1	1	0

Products 3 and 5 have Price $100
Product 1 has Price $200
Products 2, 7, and 10 have Price $300
Products 4, 6, 8, and 9 have Price $400

Figure 6-9 **Join** Indexes–speeds up join operations

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
. . .		

Order

RowID	Order#	Order Date	Cust#(FK)
30001	O5532	10/01/2001	C3861
30002	O3478	10/01/2001	C1062
30003	O8734	10/02/2001	C1062
30004	O9845	10/02/2001	C2027
. . .			

Customer

RowID	Cust#(PK)	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	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
. . .		

Table 6-3 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 initial set of records is loaded
Sequential Retrieval on Primary Key	Very fast	Moderately fast	Impractical, unless use 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 use 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 file	If space can be dynamically allocated, this is easy, but requires maintenance of indexes	Very easy, except multiple keys with same address require extra work
Updating Records	Usually requires rewriting 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

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. DBMS may have limit on number of indexes per table and number of bytes per indexed field(s)
8. Null values will not be referenced from an index
9. Use indexes heavily for non-volatile databases; limit the use of indexes for volatile databases

   Why? Because modifications (e.g. inserts, deletes) require updates to occur in index files