MDBM-Chapter 5

Physical Database Design and Performance

Objectives

- 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 to create a design for storing data that will provide *adequate performance (or data processing efficiency)* and insure *database integrity*, *security*, and *recoverability.*

Physical Design Process

Inputs (requirements prior to physical design)

- Normalized relations
- Data volume estimates
- Attribute definitions
- Response time expectations
- Data security needs
- Backup/recovery needs
- Integrity expectations
- **DBMS technology to use**

Leads to

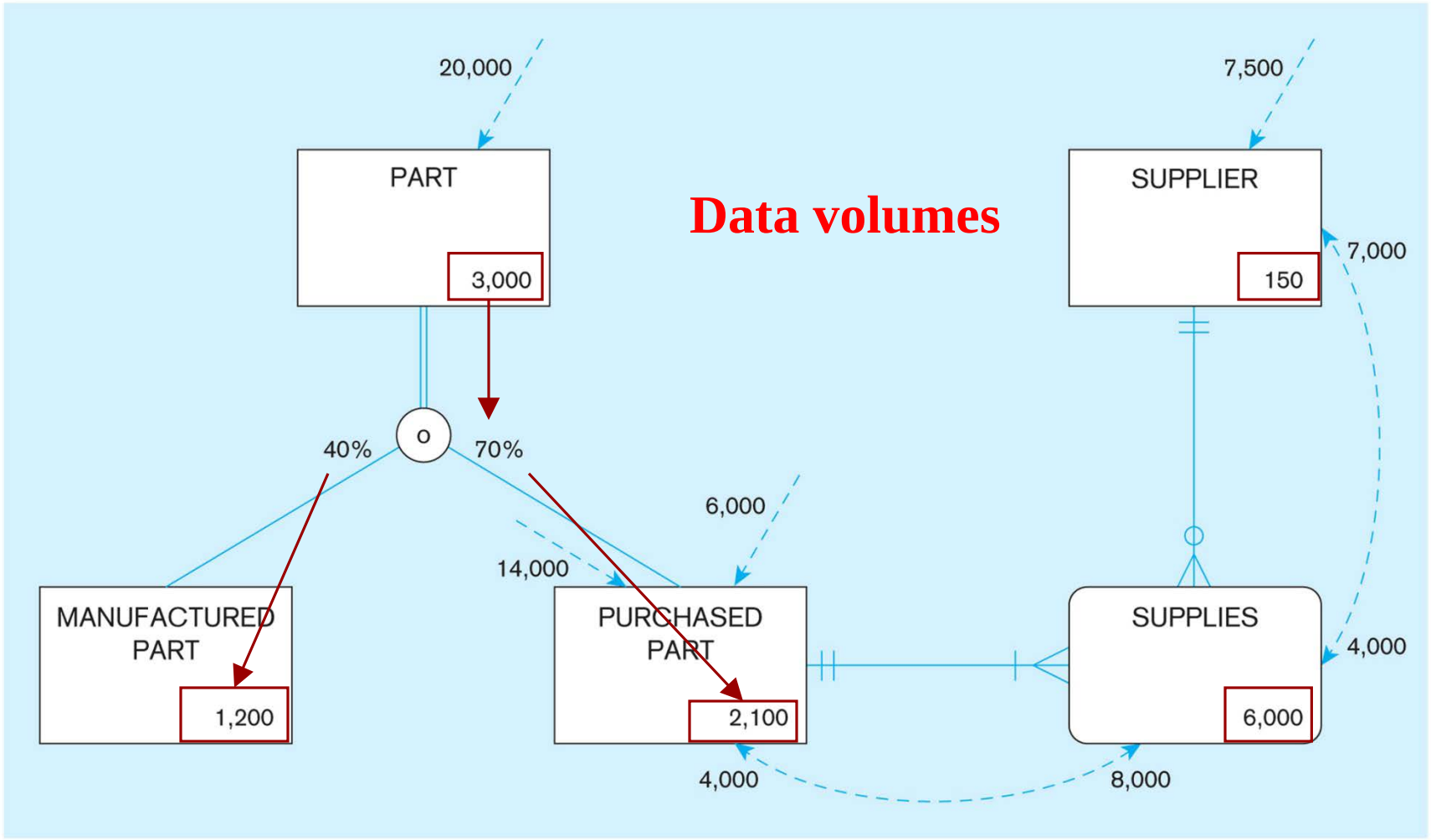
Decisions

- Attribute data types (storage format)
- Physical record descriptions (it doesn't always match the logical design of a DBMS)
- File organizations in secondary memory (hard disks)
- Indexes and database overall architectures for efficient data retrieval
- Strategies for query optimization

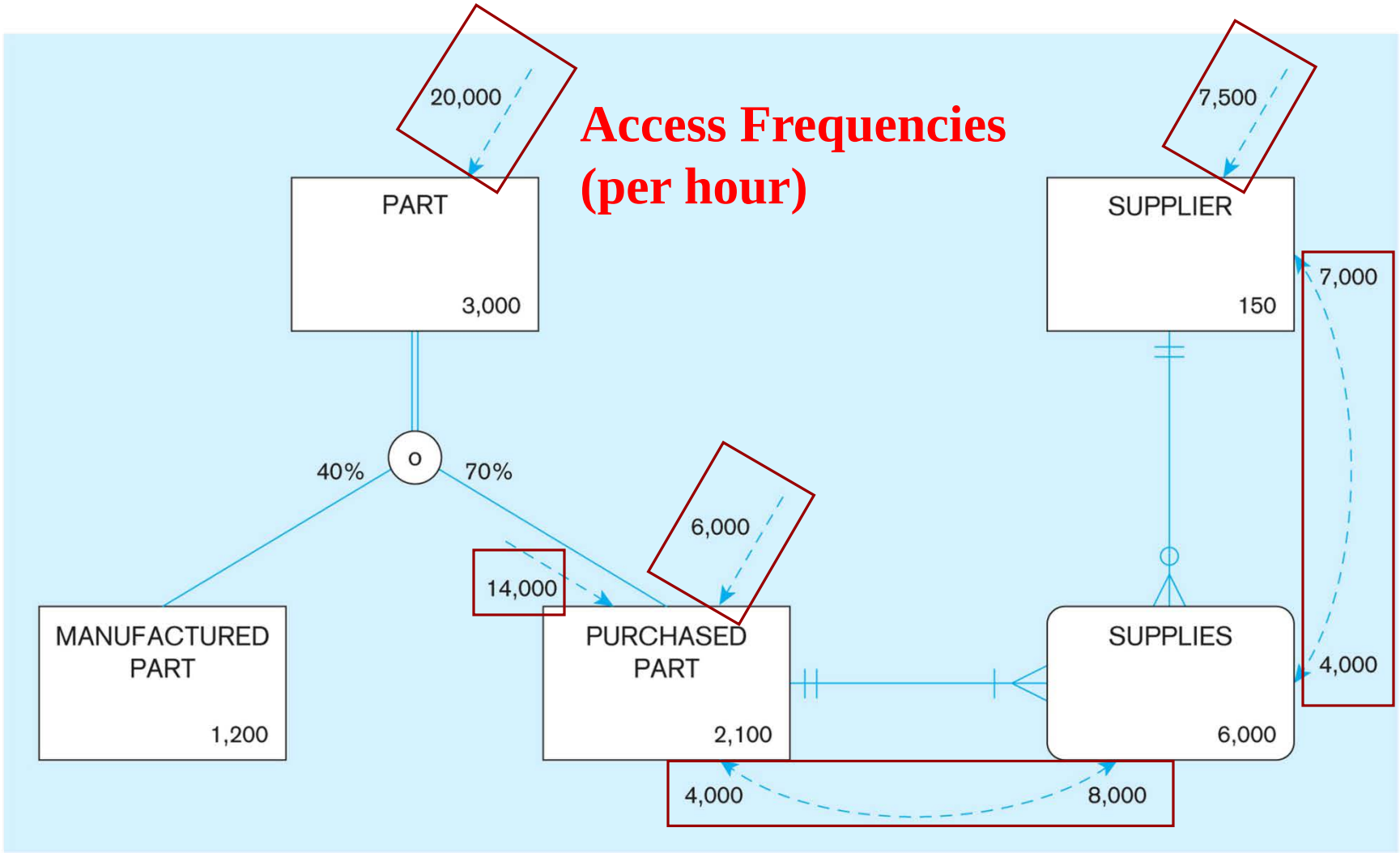
Data Volume and Usage Analysis

- Adding notations to the EER diagram representing normalized relations.
- Writing a number in the lower right corner of an entity as the data volume.
- Using dashed arrows to represent access frequencies.

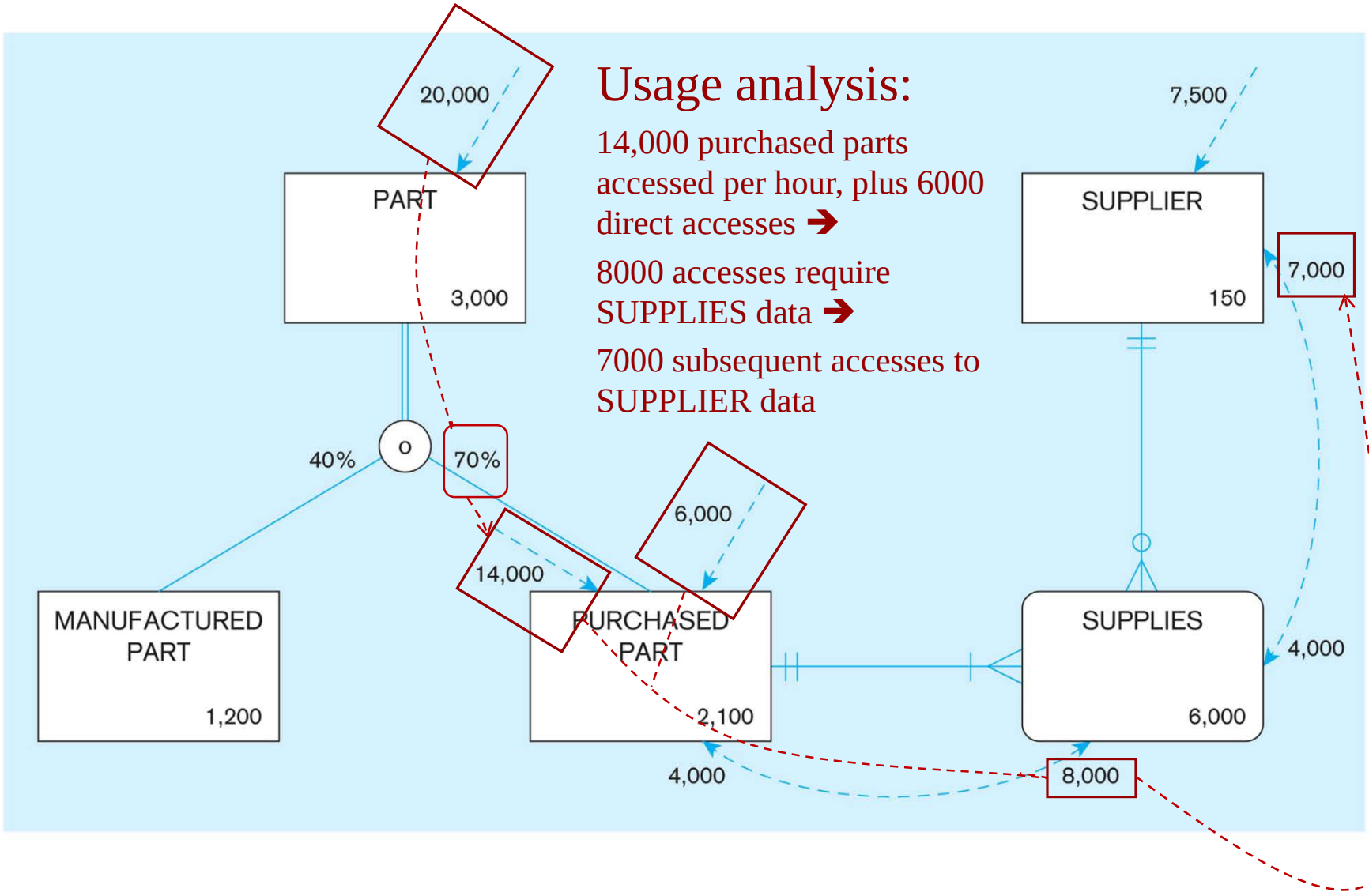
Composite Usage Map



Composite Usage Map (cont.)



Composite Usage Map (cont.)



Designing Fields

- **Field**: smallest unit of data in a database, corresponding to a simple attribute (or a single component of a composite attribute) in the logical data model
- Field design
  - Choosing data type
  - Controlling data integrity
  - Handling missing values
  - Dealing with coding, compression and encryption

Data Types

A detailed coding schema recognized by system software (e.g., a DBMS) for representing data.

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 value less than 30 characters 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 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) (e.g., 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).
INTEGER	Positive or negative integer with up to 38 digits (same as SMALL INT).
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).

Four Objectives of Choosing Data Types

- Minimize storage space
  - Just specify the space that is needed.
- Represent all possible values
  - The result of mathematical operations may need greater maximum width than the width of normal field members.
- Support all data manipulations
  - E.g., numeric types for arithmetic operations, and character type for string manipulations.
- Improve data integrity
  - E.g., The data type enforces the type of data and the length of a field value.

Field Data Integrity

- **Default value**: Assumed value if no explicit value is entered, which can reduce data entry time and error.
- **Range control**: Allowable value limitations (e.g., a numerical lower-to-upper bound or a set of specific values).
- **Null value control**: Allowing or prohibiting empty fields (e.g., a PK field does not allow a null value).
- **Referential integrity**: A special form of range control for foreign-key to primary-key match-ups.

Handling Missing Data

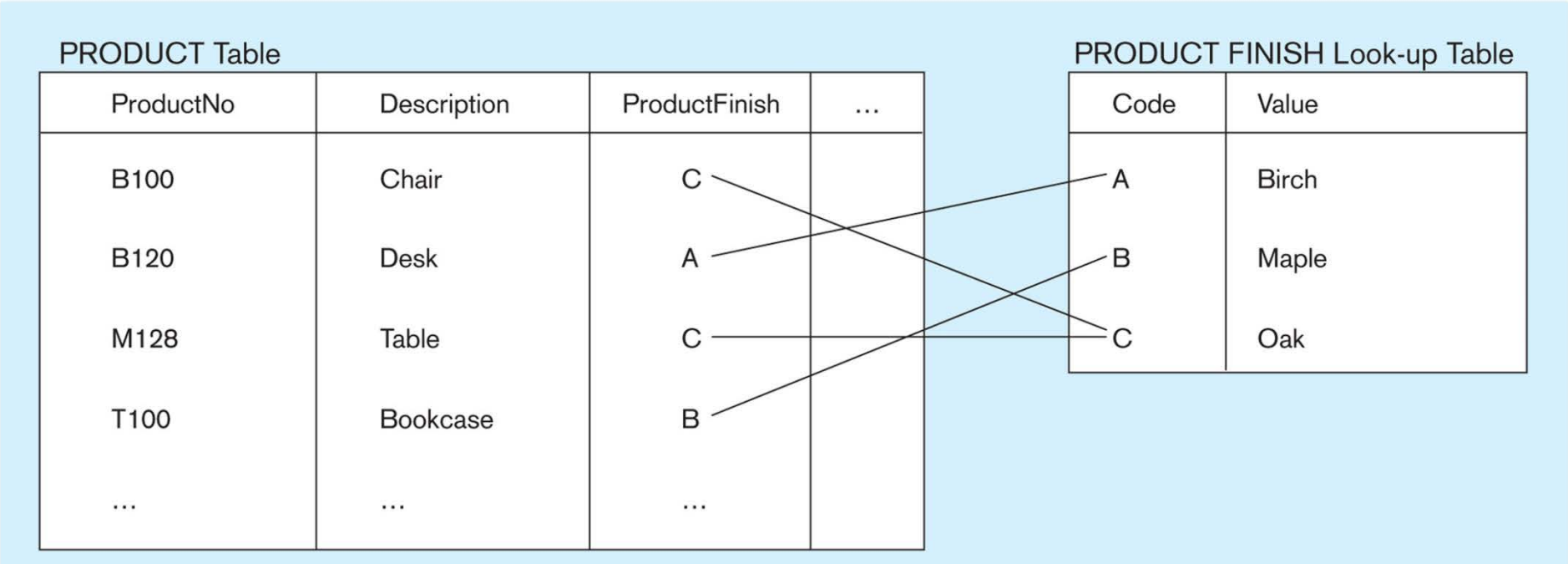
- Use a default value, or do not permit missing values.
- Substitute an estimate of the missing value (e.g., using a formula for an average value) but must be marked for such a substitution.
- In programs, ignore missing-value data unless the value is significant (based on sensitivity testing).
- Construct a report listing missing values so that people are aware of missing values and resolve unknown values quickly.

Triggers can be used to perform these operations (Ch7)

Coding Techniques

- A method of translating an attribute of a sparse set of values or needing considerable storage space into a **code** or **translation table** that requires less space.
- A code is similar to a foreign key.
- The code table does not belong to the conceptual or logical model; it is a physical construct.
- Coding is not suitable for an attribute that it is not used frequently, or the number of distinct values of the field is very large.
- Coding saves space but costs additional space for the lookup table and an extra access to the lookup table (called join) to obtain an actual value.

An Example of a Code Look-up Table (PVFC)



Normalization and Physical DB Design

- Normalization solves data maintenance anomalies and minimizes data redundancy.
- Efficient use of storage space has become less important due to rapid cost drop of storage/unit of data.
- Efficient data processing dominates the physical DB design.
- Queries that need matching up (or joining) multiple relations are time-consuming.
- One-for-one implementation of normalized relations to physical records may not yield most efficient data processing.
- Sometimes a partially normalized DB is more efficient in processing data using a technique called denormalization.

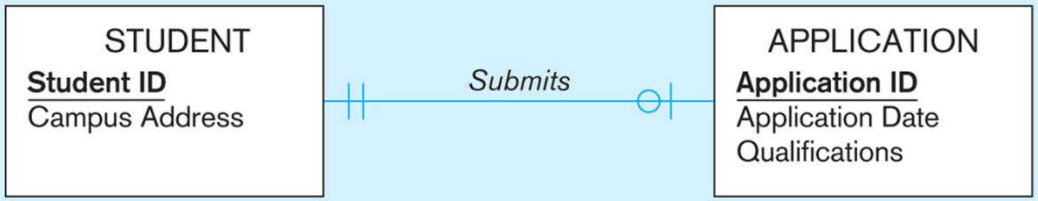
Physical Records

- **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.
- **Blocking Factor**: The number of physical records per page.

Denormalization

- Definition: Transforming ***normalized*** relations into ***non-normalized*** physical record specifications by combining attributes from different relations, or partitioning a relation, or both.
- Benefits:
  - To improve performance (speed) by reducing number of table lookups (i.e. *reduce number of necessary join queries*), especially when the frequency of use is high.
- Costs (due to data duplication and anomalies):
  - Wasted storage space.
  - Data integrity/consistency threats.
  - Must be used with caution.
- Common denormalization opportunities:
  - One-to-one relationship.
  - Many-to-many relationship with non-key attributes (associative entity).
  - Reference data (1:M relationship where 1-side participates in no other relationship).

A Possible Denormalization Situation: Two Entities with One-to-one Relationship



Normalized relations:

STUDENT

StudentID	CampusAddress

APPLICATION

ApplicationID	ApplicationDate	Qualifications	StudentID

Denormalized relation:

STUDENT

StudentID	CampusAddress	ApplicationDate	Qualifications

and ApplicationDate and Qualifications may be null

Application ID is not necessary since this is a one-to-one relationship, but it may also be included.

A Possible Denormalization Situation: a Many-to-many Relationship with Non-key Attributes

Normalized relations:

VENDOR

VendorID	Address	ContactName

ITEM

ItemID	Description

PRICE_QUOTE

VendorID	ItemID	Price

Extra table access required

Combine one of the relations w/ associative relation

Denormalized relations:

VENDOR

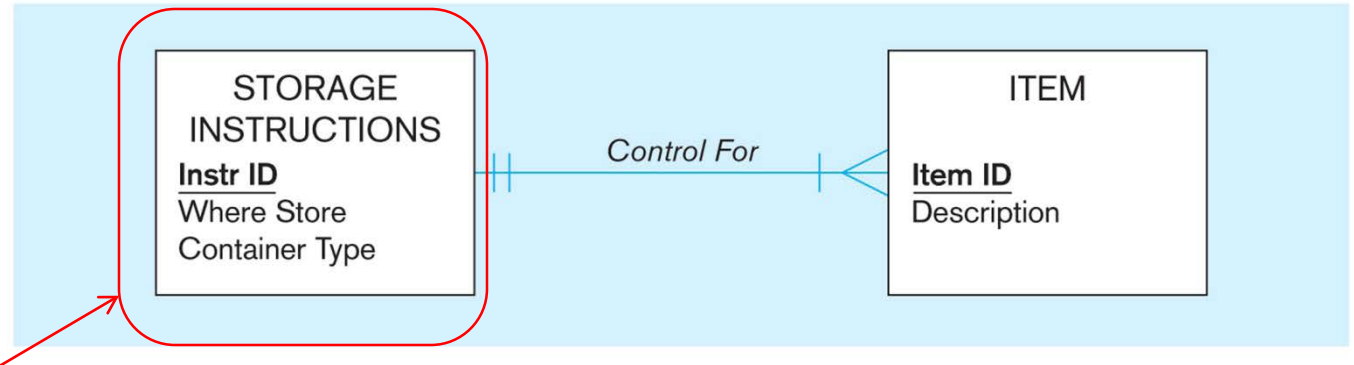
VendorID	Address	ContactName

ITEM_QUOTE

VendorID	ItemID	Description	Price

Null description possible

A possible denormalization situation: reference data



Participating in no other relationship

Normalized relations:

STORAGE

InstrID	WhereStore	ContainerType

ITEM

ItemID	Description	InstrID

Extra table access required

Denormalized relation:

ITEM

ItemID	Description	WhereStore	ContainerType

Data duplication

Partitioning

- **Definition:** Another form of denormalization by often partitioning a relation into multiple physical tables; it often corresponds with user views.
- **It has both advantages and disadvantages.**
- **Horizontal Partitioning.**
- **Vertical Partitioning.**

Partitioning: Advantages vs. Disadvantages

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.

Horizontal Partitioning

- A partitioning method that places different **rows** of a table into several separate tables based on common column values. It is very similar to creating a supertype/subtype relationship.
  - Useful for situations where different users need access to different rows
  - Oracle DBMS uses a method that one table has several partitions rather than separate tables. Oracle 11g has three methods: range partitioning, hash partitioning, and list partitioning.

Oracle 11g Horizontal Partitioning Methods

- **Range partitioning**
  - Partitions defined by a range of field values (lower and upper key value limits)
  - Could result in unbalanced distribution of rows
- **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 (e.g., a partitioning based on the value of the column State, rows having a State value of “NY,” “MA,” or “NJ” are in one partition)

(Composite partitioning: a combination of two of the three approaches)

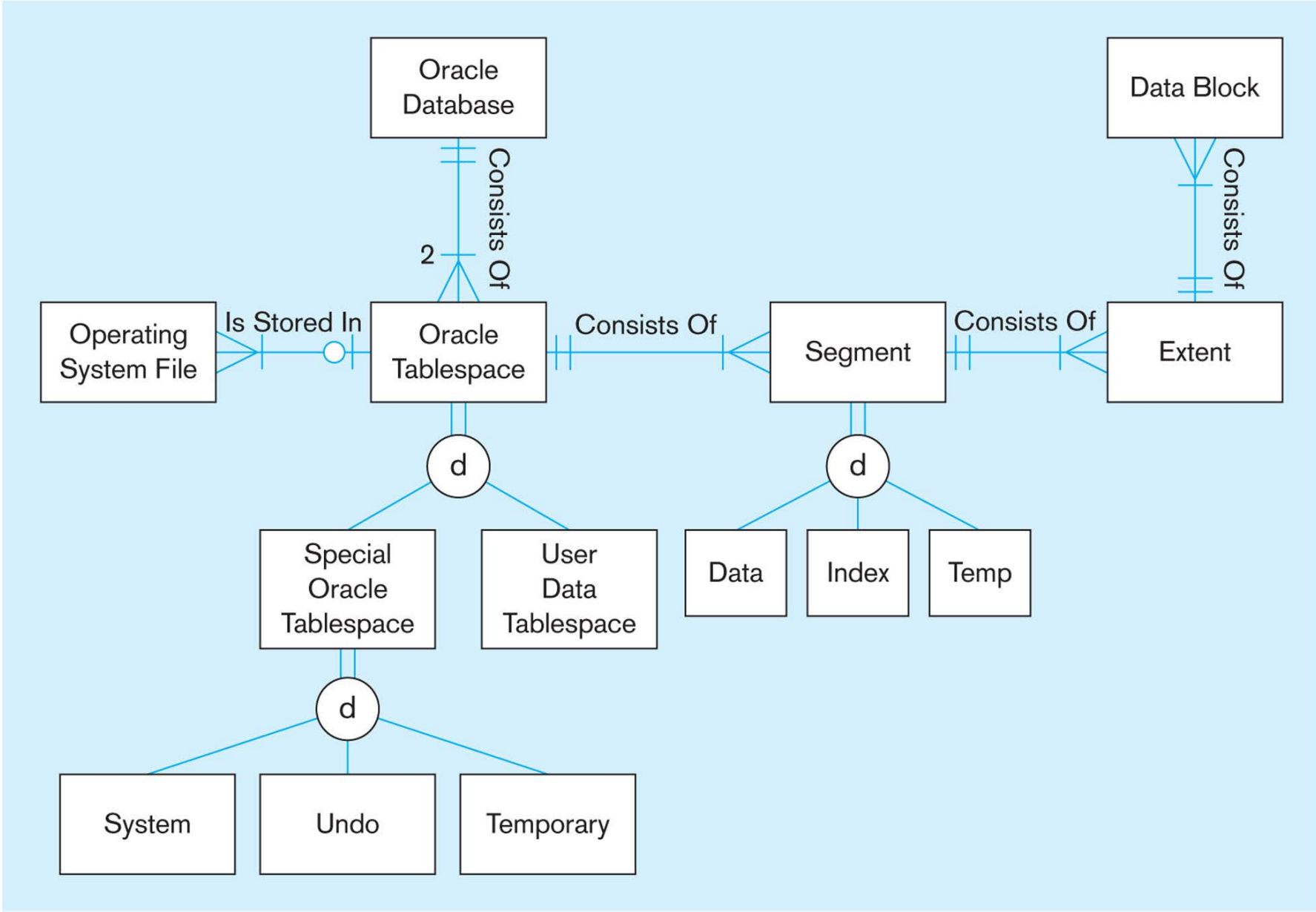
Vertical Partitioning

A partitioning method that distributes the columns of a table into several separate tables; the *primary key* is repeated in each table. It is useful for situations where different users need to access to different columns.

Designing Physical Files

- **Physical File**: A named portion of secondary memory allocated for the purpose of storing physical records
- **Tablespace**: A named logical storage unit in which physical files for database tables, views and other database objects can be stored
- **Extent**: Contiguous section of disk storage space, which consists of contiguous data blocks
- **Data block**: the smallest unit of storage space
- **Constructs** linking two pieces of data:
  - Sequential storage
  - Pointers: A field of data that can be used to locate related fields or records

DBMS Terminology in an Oracle 11g Environment

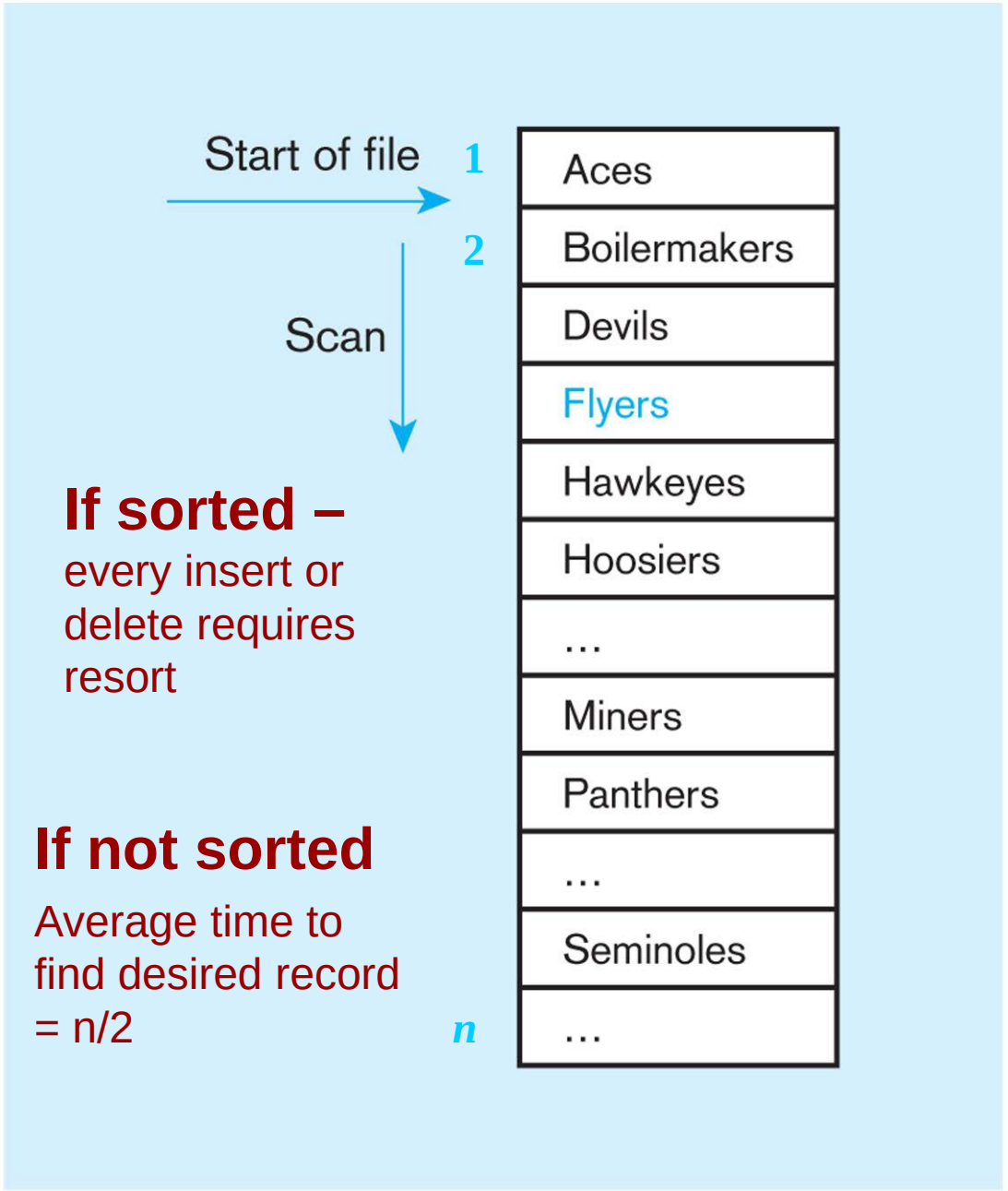


File Organizations

- A technique for physically arranging the records of a file on secondary storage devices
- Seven factors for selecting a file organization:
  - Fast data retrieval
  - High throughput for data input and maintenance transactions
  - Efficient storage space utilization
  - Protection from failure and data loss
  - Minimizing need for reorganization
  - Accommodating growth
  - Security from unauthorized use
- Three types of file organizations
  - Sequential
  - Indexed
  - Hashed

Sequential file organization

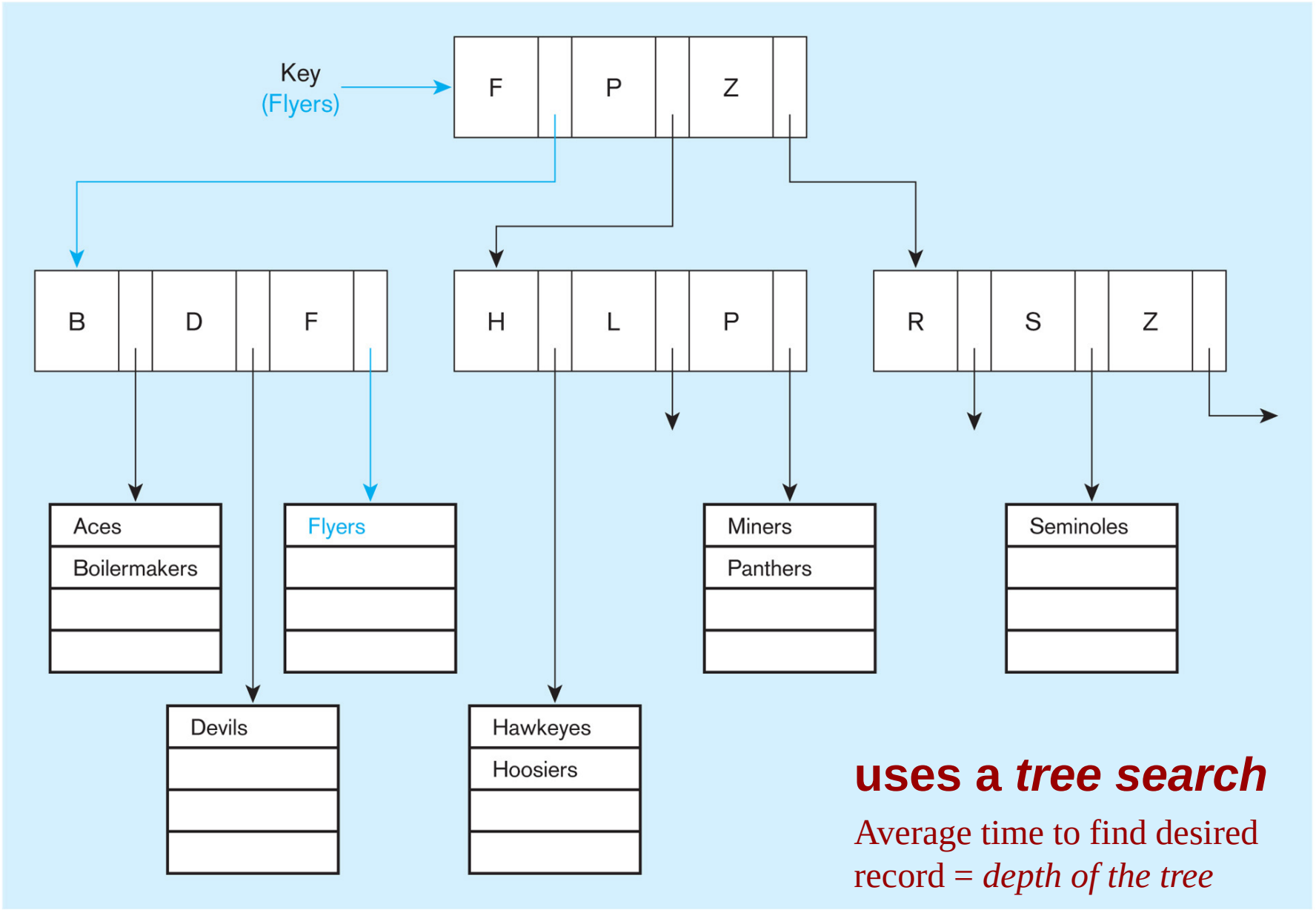
- Records of the file are stored in sequence by the primary key field values.
- Sequential files are not used in an active database but may be used for the files to back up a database.



Indexed File Organizations

- **Indexed File Organization**: the records are stored either sequentially or nonsequentially with an index that allows software to locate individual records.
- **Index**: a table or other data structure used to determine more quickly and efficiently in a file the location of records that satisfy some condition.
- The structure of an index: often a table by itself with 2 columns: the key and the address of the record(s) containing that key value.
- Primary keys are automatically indexed.

Indexed File Organization: B-tree Index



uses a *tree search*

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

Creating Index for the Primary Key

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

Creating Index for a Composite Primary Key

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

Creating a Nonunique Index

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

Rules for Using Indexes

1. Use on large 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/or GROUP BY commands.
5. When there are >100 values for an attribute 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 a key to index is used to determine location of record, use surrogate (like a serial #) to allow even spread in storage area.
8. DBMS may have a 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.

Join Index: an Index on Columns from Two or More Tables from the Same Domain of Values

a) Join index for common non-key columns

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
. . .		

b) Join Index for Matching Foreign Key (FK) and Primary Key (PK)

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	C1062	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
. . .				

Join Index

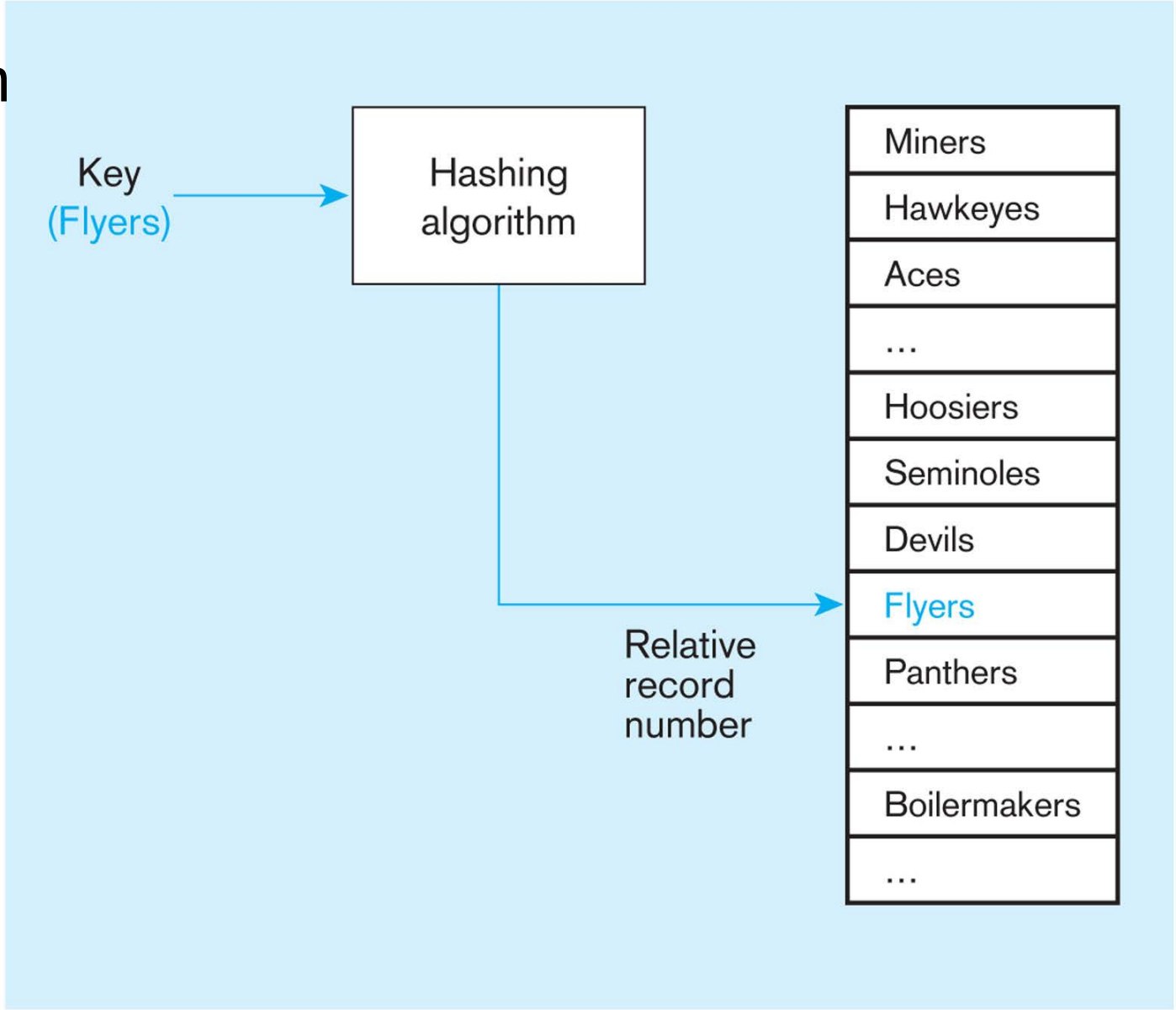
CustRowID	OrderRowID	Cust#
10001	30004	C2027
10002	30002	C1062
10002	30003	C1062
10004	30001	C3861
. . .		

Hashed File Organization

Definition: A data storage system in which the address for each record is determined by a hashing algorithm.

Hashing algorithm: A routine that divides each primary key value by a suitable prime number and then use the remainder of the division as the relative storage location (e.g., the **remainder** of 12396/997 is 432, thus record is stored at location 432 in the file).

Hashed file organization



Hashing Limitation and Hash Index Table

- **Hashing limitation**: Only one key can be used for data storage and retrieval.
- **Solution**: Combining hashing and indexing into a hash index table.
- **Hash index table**: A file organization that uses hashing to map a key into a location in an index, where there is a pointer to the actual data record matching the hashing key.
- **Result**: Several primary and secondary keys, each with its own hashing algorithm and index table, share one data table. This adds flexibility and speed for data retrieval and is useful in parallel data processing.

Comparison of Three File Organizations

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

Clustering Files

- In some relational DBMSs (e.g., Oracle), related records from different tables can be stored together in the adjacent secondary memory space.
- It is 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.
- It is best used for fairly static records.
- The CREATE CLUSTER command in Oracle is for this purpose.

First Create a Cluster (Adjacent Disk Space) in Oracle

```
CREATE CLUSTER Ordering (CustomerID
VARCHAR2(25));
```

Then assign tables to the cluster

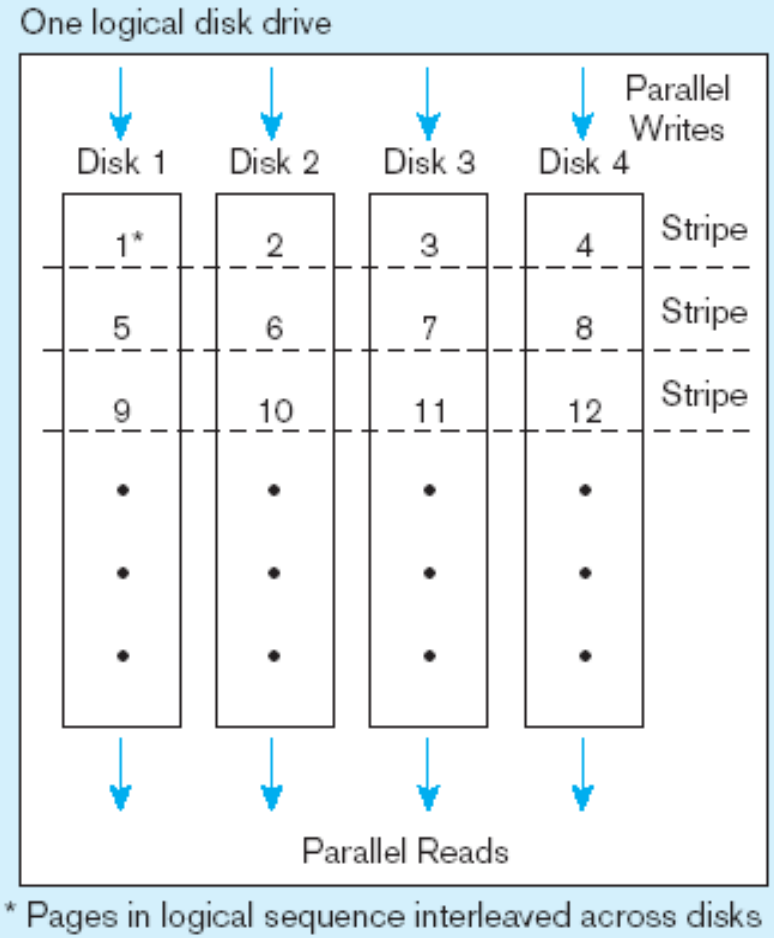
```
CREATE TABLE Customer_T (
        CustomerID          VARCHAR2(25) NOT NULL,
        CustomerAddress   VARCHAR2(15))
        CLUSTER Ordering (CustomerID);
CREATE TABLE Order_T (
        OrderID               VARCHAR2(20) NOT NULL,
        CustomerID          VARCHAR2(25) NOT NULL,
        OrderDate           DATE)
        CLUSTER Ordering (CustomerID);
```

RAID

- Redundant Array of Inexpensive/Independent Disks.
- A set of disk drives that appear to the user to be a single large logical disk drive.
- Stripes cut across all disk drives; workload is balanced.
- Allows parallel access to data (improves access speed).
- Pages are arranged in stripes.
- Technologies that are tolerant to the likelihood of disk drive failure and fault by storing redundant data (variations of RAID).

RAID with Four Disks and Striping

Pages 1-4 can be read/written simultaneously



RAID Types

- RAID 0
  - Maximized parallelism
  - No redundancy
  - No error correction
  - no fault-tolerance
- RAID 1
  - Fully redundant, disk mirror
  - Write operation must be done twice
  - Most common form
- RAID 2
  - No redundancy
  - One record spans across data disks
  - Error correction in multiple disks– reconstruct damaged data
- RAID 3
  - Error correction in one disk
  - Record spans multiple data disks (more than RAID2)
  - Not good for multi-user environments,
- RAID 4
  - Error correction in one disk
  - Multiple records per stripe
  - Parallelism, but slow updates due to error correction contention
- RAID 5
  - Rotating parity array
  - Error correction takes place in same disks as data storage
  - Parallelism, better performance than Raid4

Query Optimization

- Parallel query processing (possible when working in multiprocessor systems).
  - Symmetric multiprocessor (SMP) technology.
  - Breaking apart a query into modules that can be processed in parallel by related processors.
  - Example: Each processor run a copy of query on a horizontal partition (e.g., ALTER TABLE Order_T PARALLEL 3;).
- Overriding automatic query optimization.
  - In a DBMS, query optimizer choose the "best" plan to execute the query based on statistics about each table.
  - The optimizer's plan for processing a query can be learned by command 'EXPLAIN' or 'EXPLAIN PLAN' (steps like access indexes, use parallel servers, join tables...).
  - If you know a better way, you can force the DBMS to do the steps differently.

Query Optimization (cont.)

- Picking a data block size.
  - Too small size results in too many physical I/O operations.
  - Too large size results in extra data being transferred.
  - Normally 2K to 32K.
- Balance I/O across disk controllers.
  - Disks are attached to controllers– the more controllers, the better parallel access.
  - Collect statistics on disk and controller utilization on table accessing, and balance the workload by moving tables between disk drives and controllers.