Physical Database Design

University of California, Berkeley

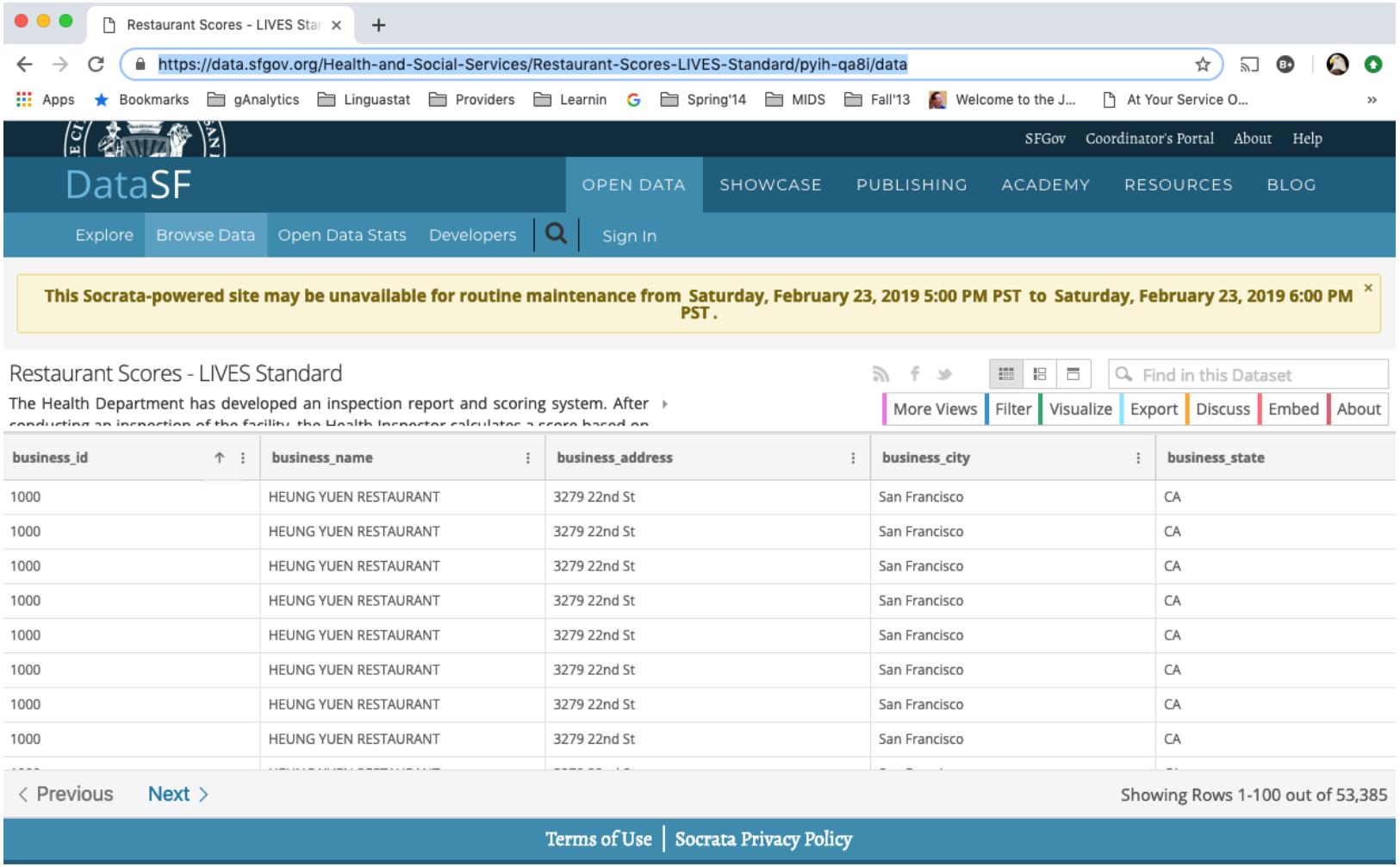
School of Information

I 257: Database Management

Announcements

- Queries/Comments?
- Assignment 1 and 2a Due Next Week (See website for specifics)
  - Anybody having trouble finding a group?
- Lab: Last In-class chance to meet and form groups
- Datasets? Lets look at one.
  - https://data.sfgov.org/Health-and-Social-Services/Restaurant-Scores-LIVES-Standard/pyih-qa8i/data

Datasets


Restaurant Scores - LIVES Star
https://data.sfgov.org/Health-and-Social-Services/Restaurant-Scores-LIVES-Standard/pyih-qa8i/data
Apps
Bookmarks
gAnalytics
Linguastat
Providers
Learnin
Spring'14
MIDS
Fall'13
Welcome to the J...
At Your Service O...
SFGov
Coordinator's Portal
About
Help
DataSF
OPEN DATA
SHOWCASE
PUBLISHING
ACADEMY
RESOURCES
BLOG
Explore
Browse Data
Open Data Stats
Developers
Sign In
This Socrata-powered site may be unavailable for routine maintenance from Saturday, February 23, 2019 5:00 PM PST to Saturday, February 23, 2019 6:00 PM PST .
Restaurant Scores - LIVES Standard
The Health Department has developed an inspection report and scoring system. After
Find in this Dataset
More Views
Filter
Visualize
Export
Discuss
Embed
About
business_id
business_name
business_address
business_city
business_state
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
1000 HEUNG YUEN RESTAURANT 3279 22nd St San Francisco CA
Previous
Next
Showing Rows 1-100 out of 53,385
Terms of Use
Socrata Privacy Policy

Lecture Outline

- Review
  - Normalization (Alternate Deck)
- Physical Database Design
- Access Methods

Lecture Outline

- Review
  - Normalization (Alternate Deck)
- Physical Database Design
- Access Methods

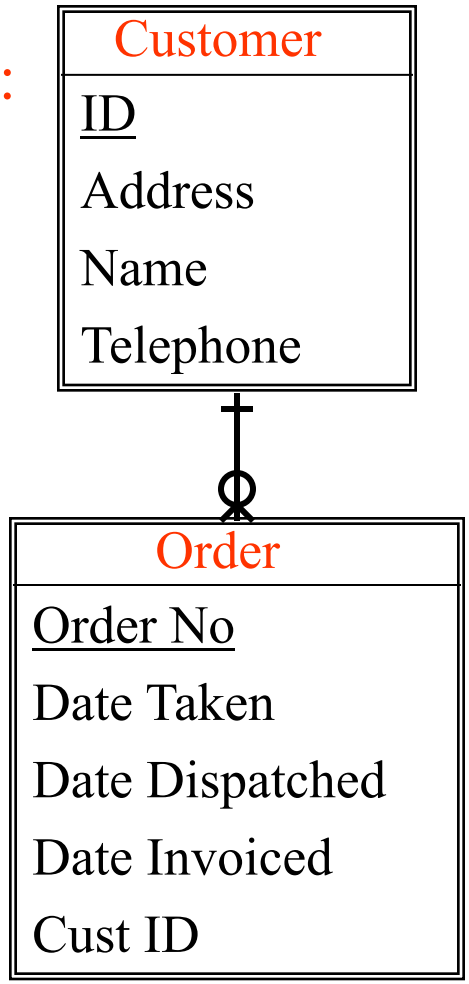
Deck 08A

Normalizing to death

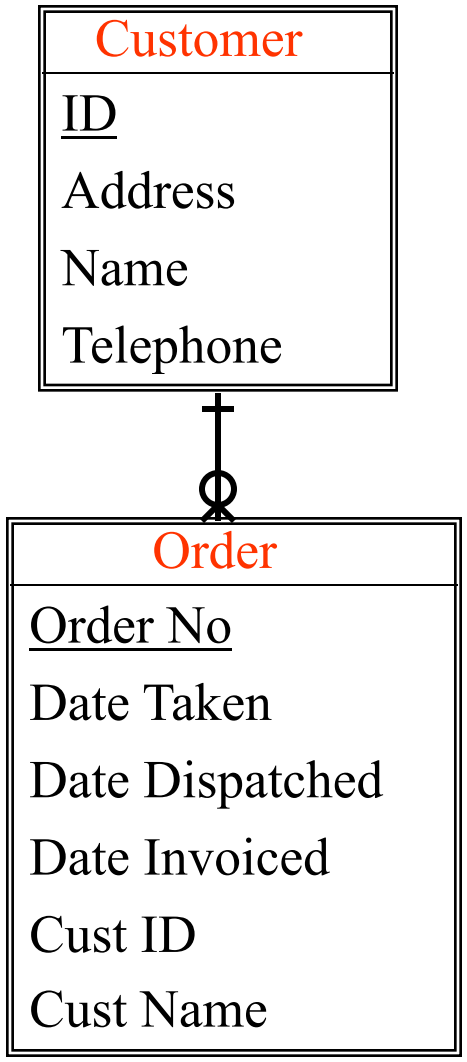
- Normalization splits database information across multiple tables.
- To retrieve complete information from a normalized database, the JOIN operation must be used.
- JOIN tends to be expensive in terms of processing time, and very large joins are very expensive.

Downward Denormalization

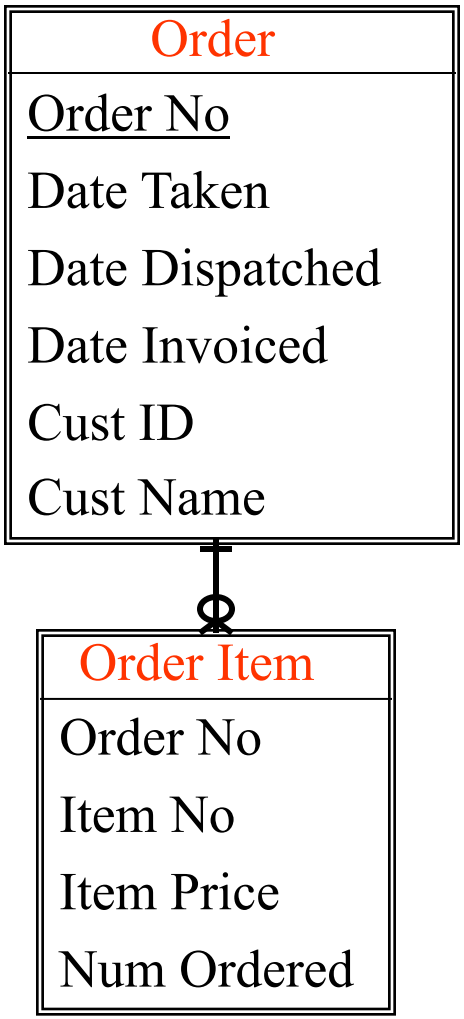
Before:

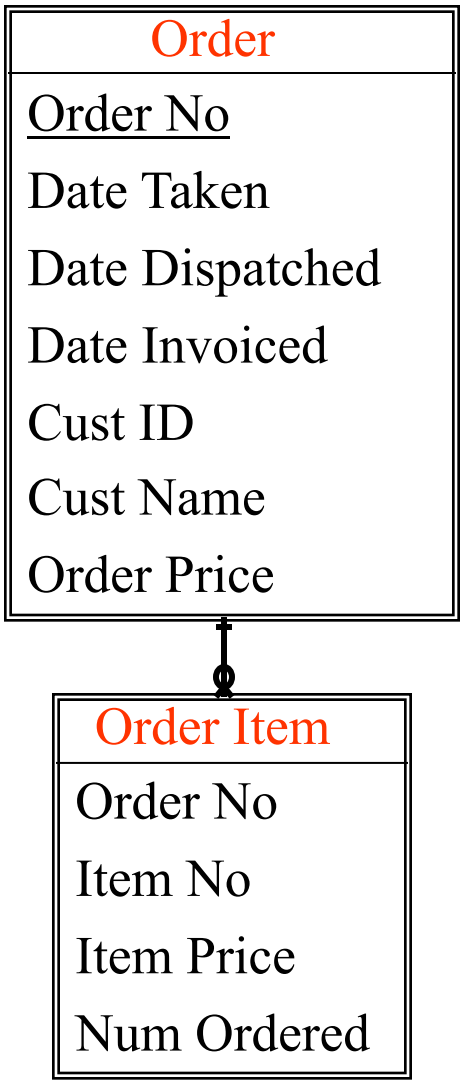


After:



Upward Denormalization

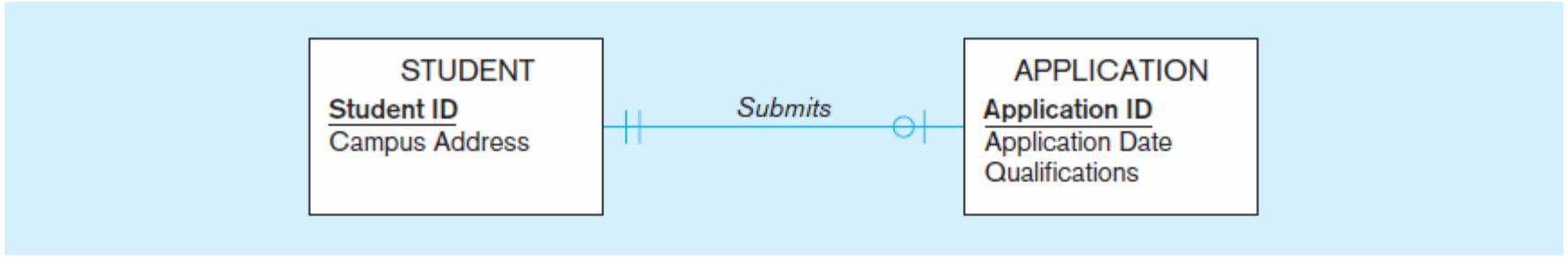




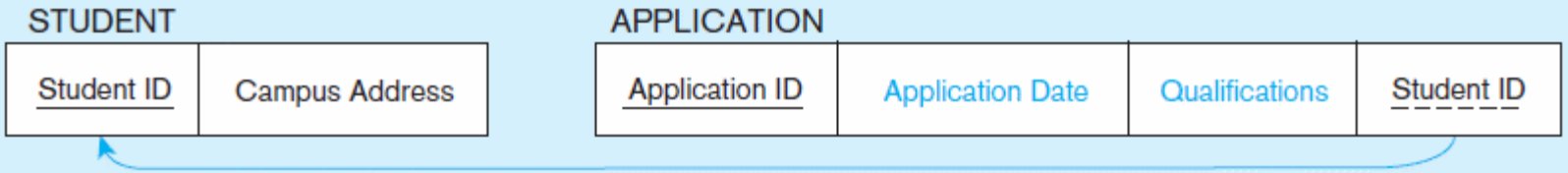
Denormalization

- 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



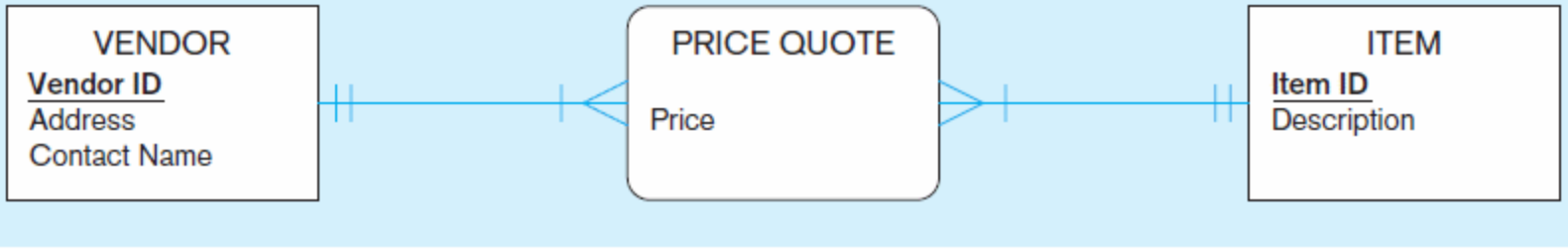
Normalized relations:



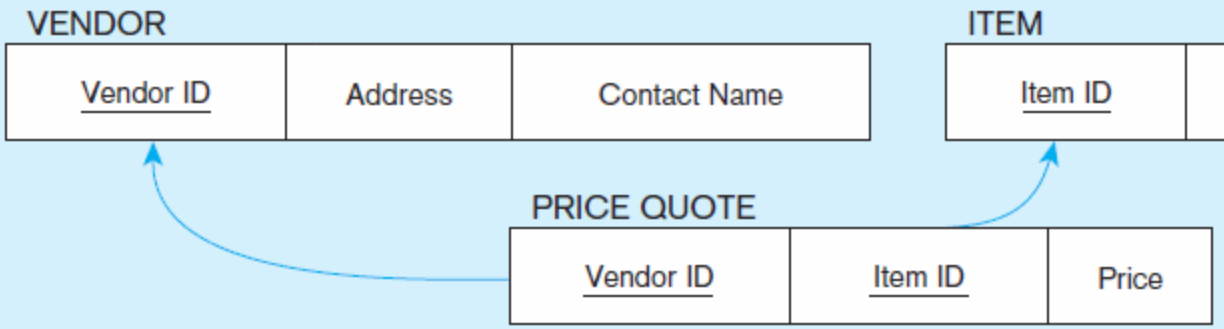
Denormalized relation:

and Application Date and Qualifications may be null

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



Normalized relations:



Extra table access required

Denormalized relations:

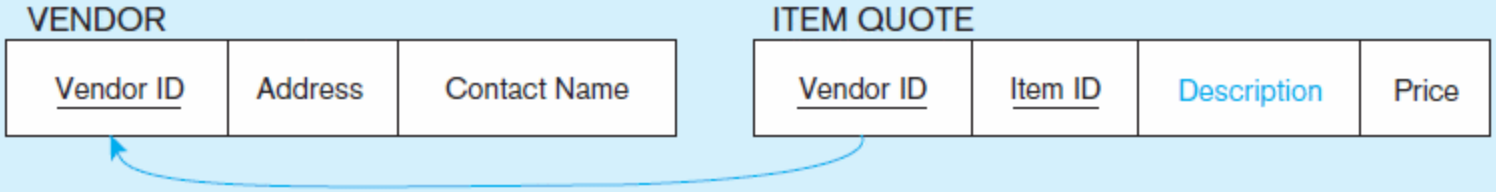
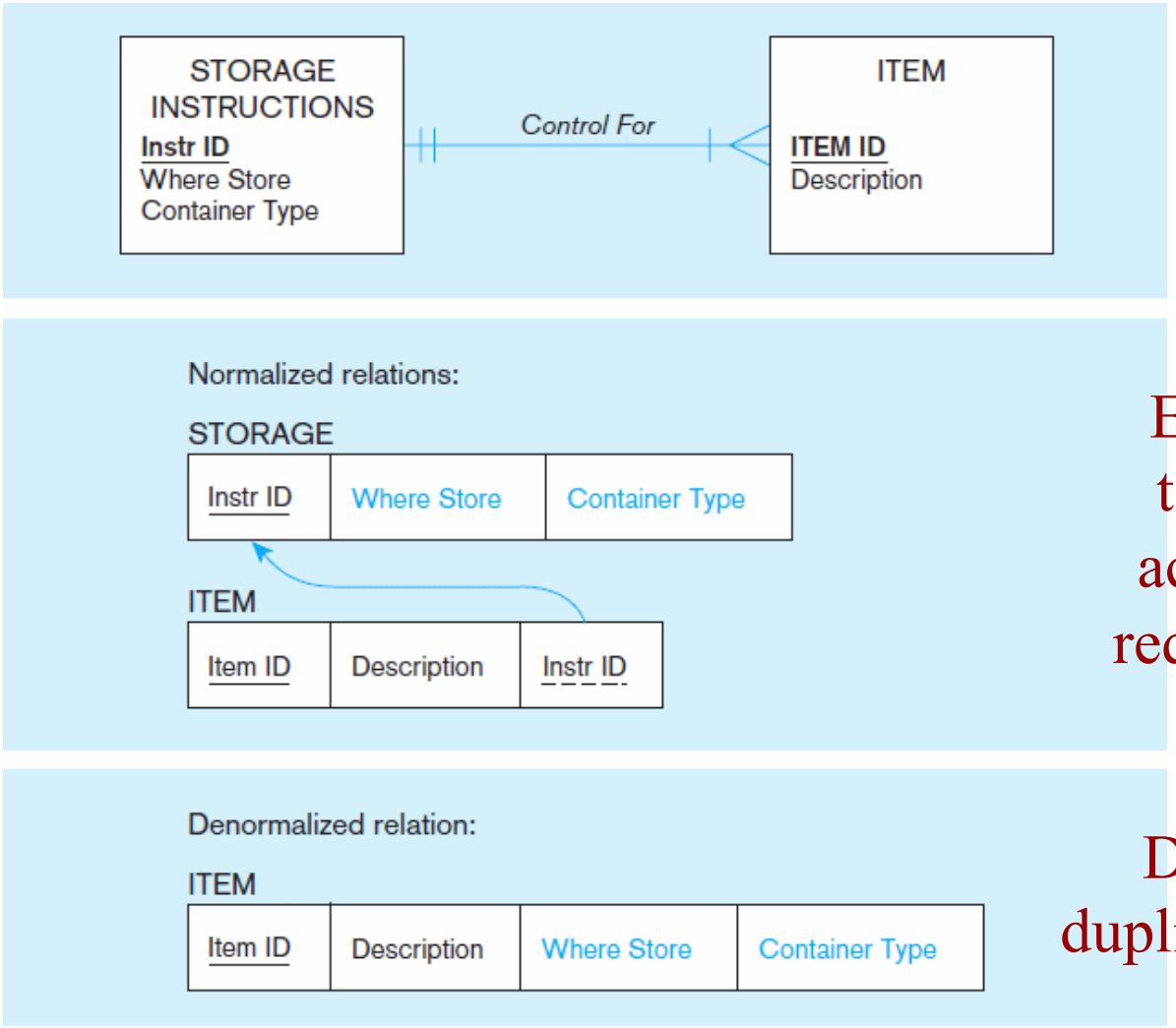


Figure 5-5 A possible denormalization situation: reference data



Extra table access required

Data duplication

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 join
  - 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

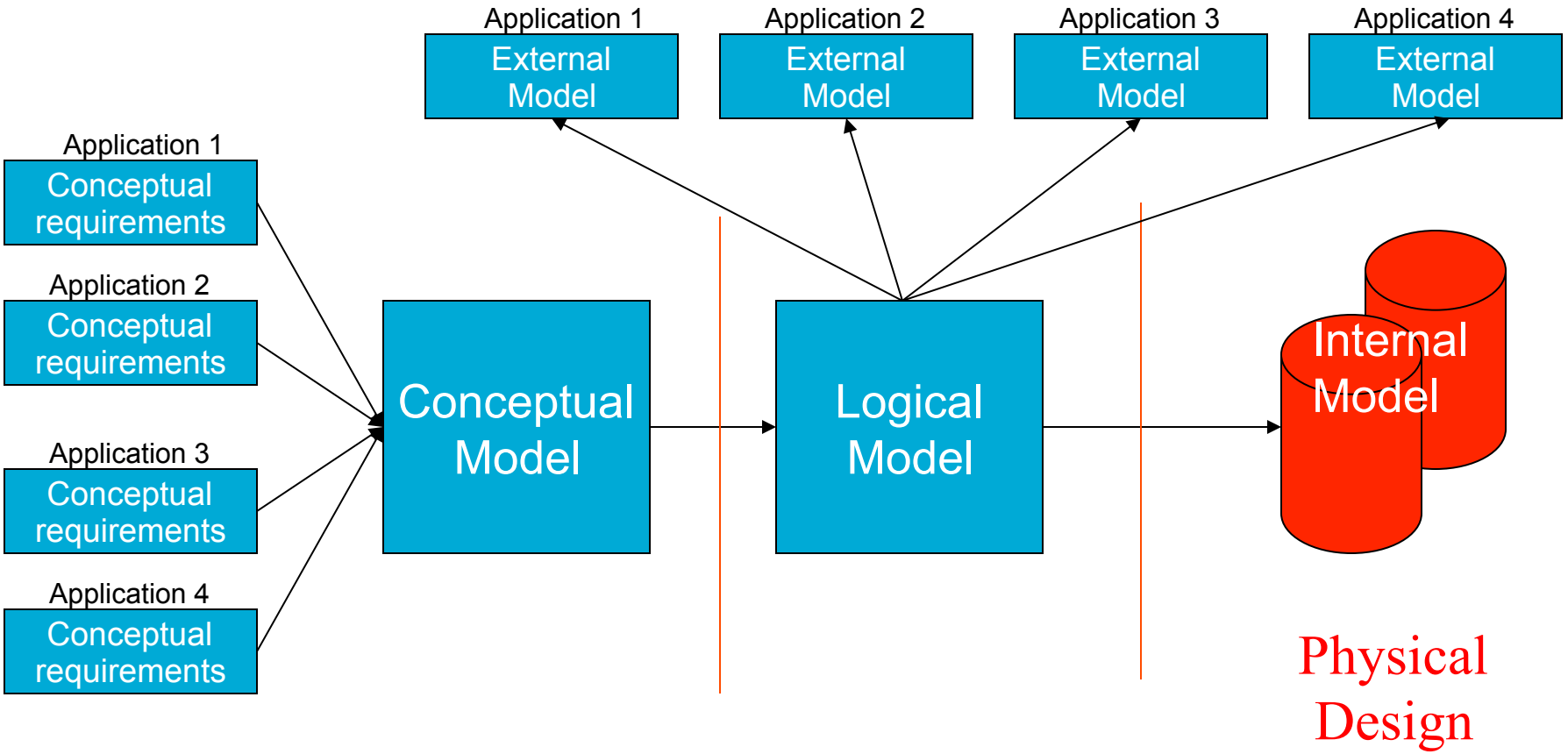
Oracle's Horizontal Partitioning

- 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

Lecture Outline

- Review
  - Normalization
- Physical Database Design
- Access Methods

Database Design Process



Physical Design

- This is where you, as the database designer, are faced with the built-in limitations of whatever DBMS you are using
- You will need to recognize and use, or work around, such limitations

Physical Database Design

- Many physical database design decisions are implicit in the technology adopted
  - Also, organizations may have standards or an "information architecture" that specifies operating systems, DBMS, and data access languages -- thus constraining the range of possible physical implementations.
- We will be concerned with some of the possible physical implementation issues

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)
- HIPAA – Health Insurance Portability and Accountability Act

Regulations and standards that impact physical design decisions

HIPAA - Health Insurance Portability and Accountability Act

- Complete Data Encryption — All health data is encrypted while in the database and during transit.
- Data Stores — If applicable, any subsystems that store encrypted BLOBs should have no 'knowledge' of what is being stored.
- Unique User IDs — HIPAA requires unique user IDs for all users and prohibits the sharing of user login credentials.
- Authentication — A database must securely authenticate users who will have access to PHI.
- Audit Logs — All data usage (user logins, reads, writes and edits) must be logged in a separate infrastructure and archived according to HIPAA requirements.
- Database Backups — Must be created, tested and securely stored. All database backups must themselves be fully encrypted. Note that, under current HIPAA Rules, data that has been properly encrypted does not trigger mandatory Breach Reporting if the data is stolen or compromised.
- Data Disposal — Methods must be in place or available to ensure that data and media are disposed of securely when no longer needed.

Physical Database Design

- The primary goal of physical database design is *data processing efficiency*
- We will concentrate on choices often available to optimize performance of database services
- Physical Database Design requires information gathered during earlier stages of the design process

Physical Design Information

- Information needed for physical file and database design includes:
  - Normalized relations plus size estimates for them
  - Definitions of each attribute
  - Descriptions of where and when data are used
    - entered, retrieved, deleted, updated, and how often
  - Expectations and requirements for response time, and data security, backup, recovery, retention and integrity
  - Descriptions of the technologies used to implement the database

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)

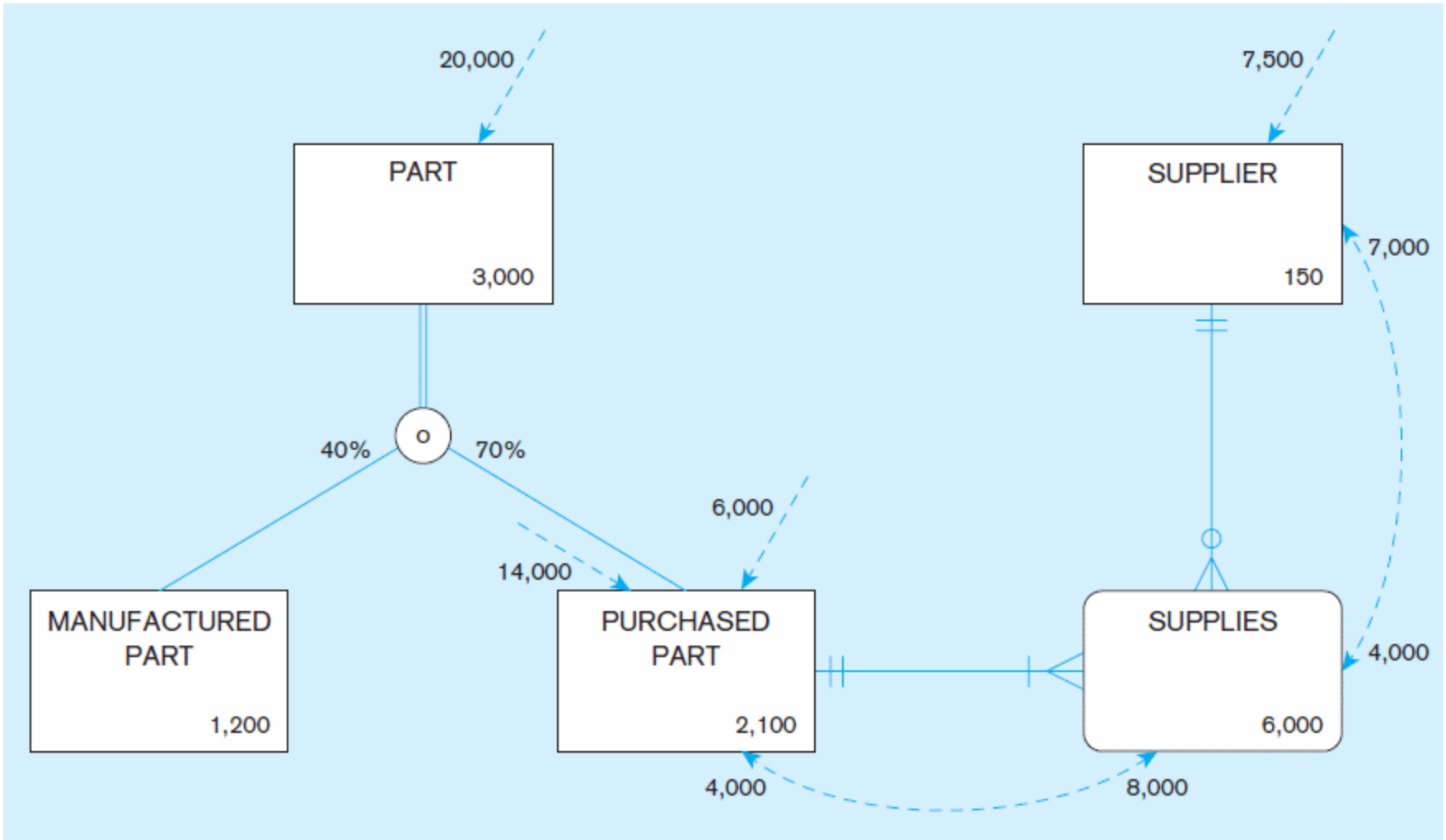


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)

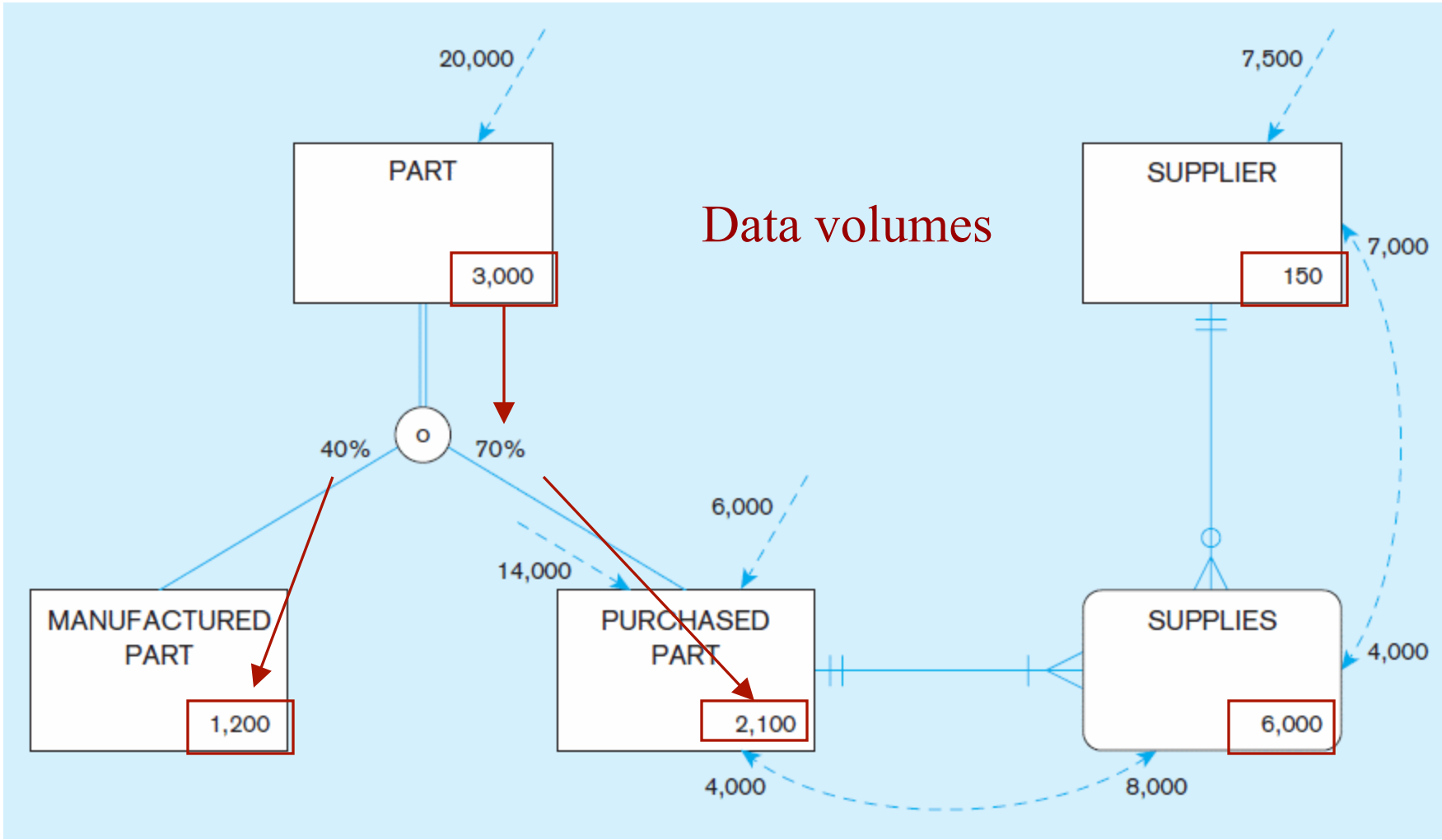


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)

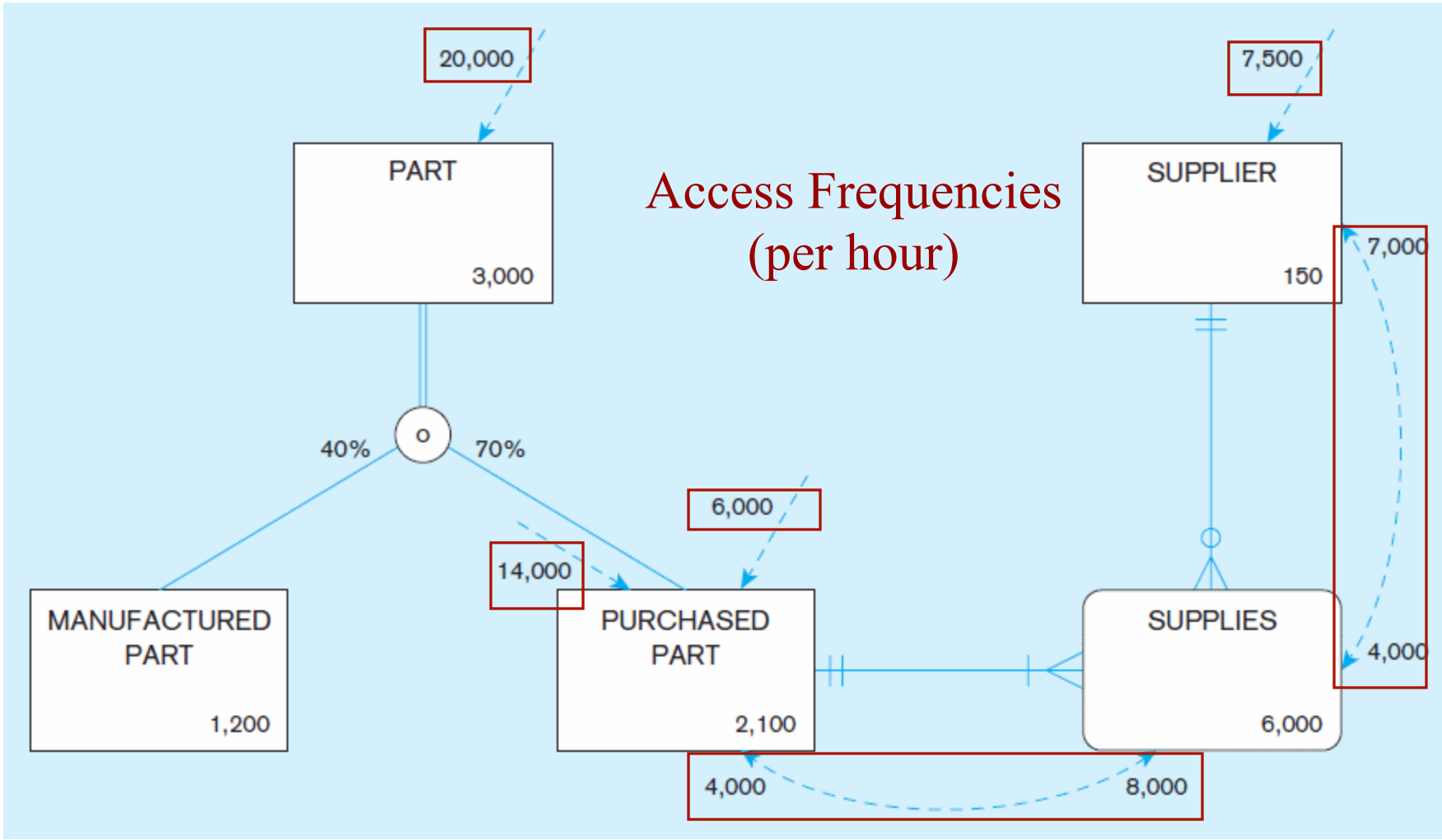


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)

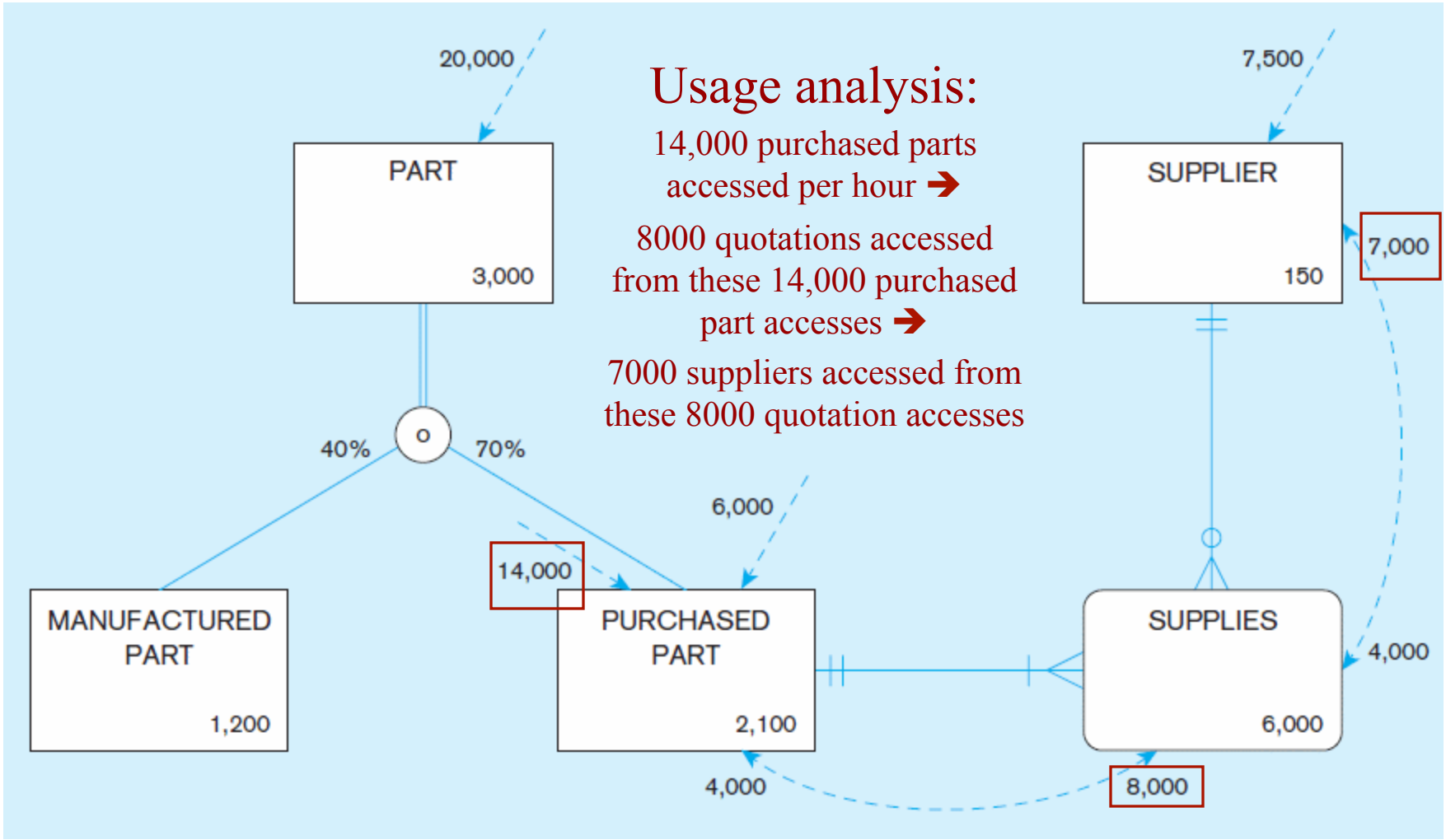
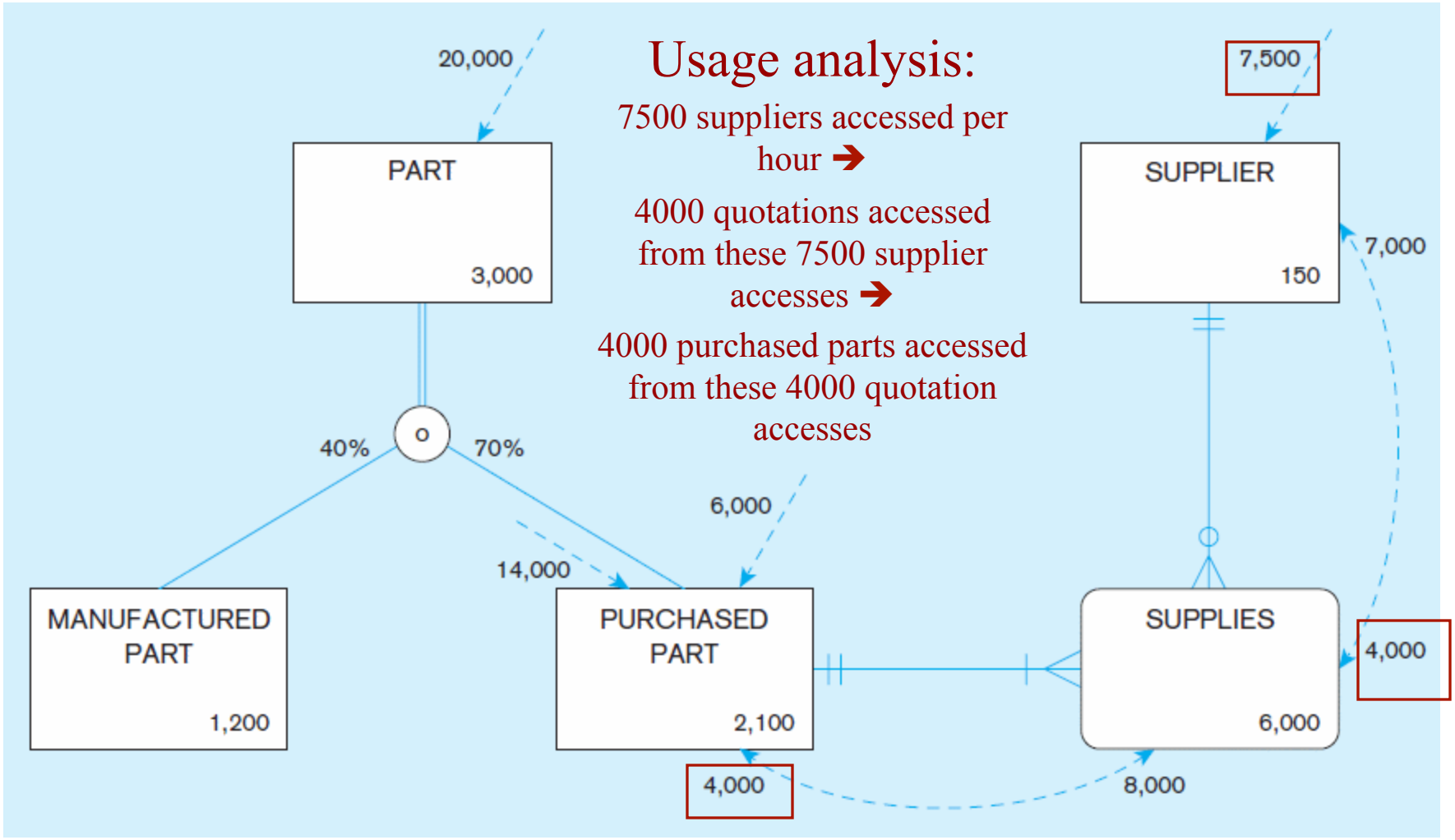


Figure 5-1 Composite usage map (Pine Valley Furniture Company) (cont.)



Physical Design Decisions

- There are several critical decisions that will affect the integrity and performance of the system
  - Storage Format
  - Physical record composition
  - Data arrangement
  - Indexes
  - Query optimization and performance tuning

Storage Format

- Choosing the storage format of each *field* (attribute). The DBMS provides some set of data types that can be used for the physical storage of fields in the database
- Data Type (format) is chosen to minimize storage space and maximize data integrity

Objectives of data type selection

- Minimize storage space
- Represent all possible values
- Improve data integrity
- Support all data manipulations
- The correct data type **should**, in minimal space, represent every possible value (but eliminate illegal values) for the associated attribute *and* can support the required data manipulations (e.g. numerical or string operations)

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

PRODUCT Table

Product No	Description	Product Finish	...
B100	Chair	C	
B120	Desk	A	
M128	Table	C	
T100	Bookcase	B	
...	...	...	

PRODUCT FINISH Look-up Table

Code	Value
A	Birch
B	Maple
C	Oak

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

Access Data Types (*Not MySQL*)

- **Numeric** (1, 2, 4, 8 bytes, fixed or float)
- **Text** (255 max)
- **Memo** (64000 max)
- **Date/Time** (8 bytes)
- **Currency** (8 bytes, 15 digits + 4 digits decimal)
- **Autonumber** (4 bytes)
- **Yes/No** (1 bit)
- **OLE** (limited only by disk space)
- **Hyperlinks** (up to 64000 chars)

Access Numeric types

- **Byte**
  - Stores numbers from 0 to 255 (no fractions). 1 byte
- **Integer**
  - Stores numbers from –32,768 to 32,767 (no fractions) 2 bytes
- **Long Integer** *(Default)*
  - Stores numbers from –2,147,483,648 to 2,147,483,647 (no fractions). 4 bytes
- **Single**
  - Stores numbers from -3.402823E38 to –1.401298E–45 for negative values and from 1.401298E–45 to 3.402823E38 for positive values. 4 bytes
- **Double**
  - Stores numbers from –1.797693134862 31E308 to –4.940656458412 47E–324 for negative values and from 1.797693134862 31E308 to 4.940656458412 47E–324 for positive values. 15 8 bytes
- **Replication ID**
  - Globally unique identifier (GUID) N/A 16 bytes

Oracle Data Types

- CHAR (size) -- max 2000
- VARCHAR2(size) -- up to 4000
- DATE
- DECIMAL, FLOAT, INTEGER, INTEGER(s), SMALLINT, NUMBER, NUMBER(size,d)
  - All numbers internally in same format…
- LONG, LONG RAW, LONG VARCHAR
  - up to 2 Gb -- *only one per table*
- BLOB, CLOB, NCLOB -- up to 4 Gb
- BFILE -- file pointer to binary OS file

MySQL Data Types

- MySQL supports all of the standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION). The keyword INT is a synonym for INTEGER, and the keyword DEC is a synonym for DECIMAL
- Numeric (can also be declared as UNSIGNED)
  - TINYINT (1 byte)
  - SMALLINT (2 bytes)
  - MEDIUMINT (3 bytes)
  - INT (4 bytes)
  - BIGINT (8 bytes)
  - NUMERIC or DECIMAL
  - FLOAT
  - DOUBLE (or DOUBLE PRECISION)

MySQL Data Types

- The date and time types for representing temporal values are DATETIME, DATE, TIMESTAMP, TIME, and YEAR. Each temporal type has a range of legal values, as well as a "zero" value that is used when you specify an illegal value that MySQL cannot represent
  - DATETIME '0000-00-00 00:00:00'
  - DATE '0000-00-00'
  - TIMESTAMP (4.1 and up) '0000-00-00 00:00:00'
  - TIMESTAMP (before 4.1) 00000000000000
  - TIME '00:00:00'
  - YEAR 0000

MySQL Data Types

- The string types are CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET
- Maximum length for CHAR is 255 and for VARCHAR is **65,535**

Value	CHAR(4)	Storage	VARCHAR(4)	Storage
""	" "	4	""	1
"ab"	"ab "	4	"ab"	3
"abcd"	"abcd"	4	"abcd"	5
"abcdefg"	"abcd"	4	"abcd"	5

- VARCHAR uses 1 or 2 bytes for the length
- For longer things there is BLOB and TEXT

MySQL Data Types

- A **BLOB** is a binary large object that can hold a variable amount of data.
- The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the values they can hold
- The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. These correspond to the four BLOB types and have the same maximum lengths and storage requirements
- TINY=1byte, BLOB and TEXT=2bytes, MEDIUM=3bytes, LONG=4bytes

MySQL Data Types

- BINARY and VARBINARY are like CHAR and VARCHAR but are intended for binary data of 255 bytes or less
- ENUM is a list of values that are stored as their addresses in the list
  - For example, a column specified as ENUM('one', 'two', 'three') can have any of the values shown here. The index of each value is also shown:
    - **Value = Index**
    - NULL = NULL
    - ‘’ = 0
    - 'one’ = 1
    - ‘two’ = 2
    - ‘three’ = 3
  - An enumeration can have a maximum of 65,535 elements.

MySQL Data Types

- The final string type (for this version) is a SET
- A SET is a string object that can have zero or more values, each of which must be chosen from a list of allowed values specified when the table is created.
- SET column values that consist of multiple set members are specified with members separated by commas (‘,’)
- For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:
  - ''
  - 'one'
  - 'two'
  - 'one,two ‘
- A set can have up to 64 member values and is stored as an 8byte number

Controlling Data Integrity

- Default values
- Range control
- Null value control
- Referential integrity (next time)
- Handling missing data

Designing Physical Records

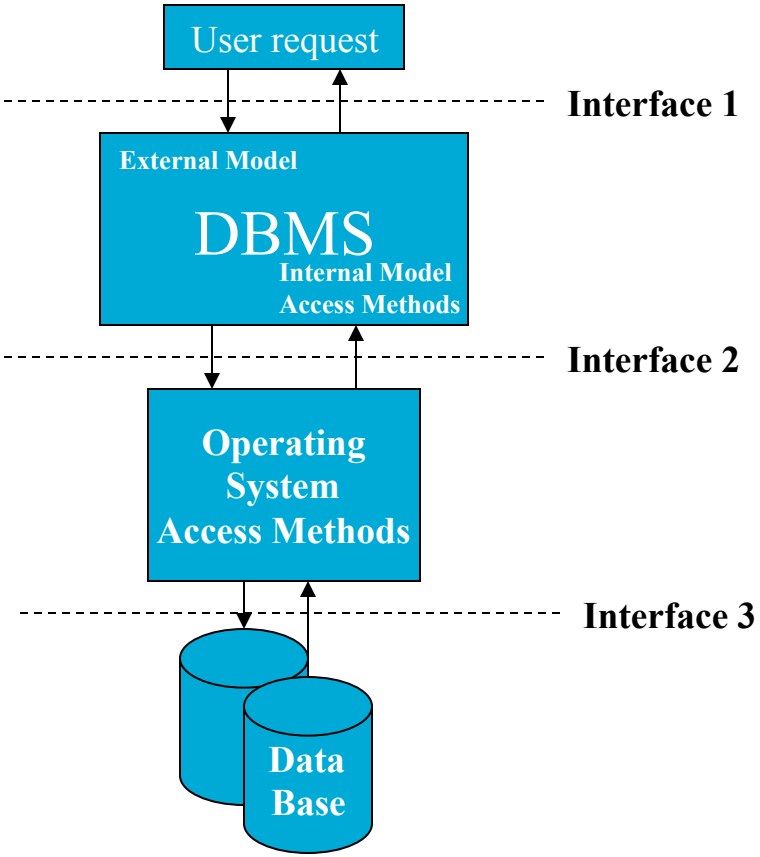
- A physical record is a group of fields stored in adjacent memory locations and retrieved together as a unit
- Fixed Length and variable fields

Designing Physical/Internal Model

- Overview
- terminology
- Access methods

Physical Design

- Internal Model/Physical Model



Physical Design

- Interface 1: User request to the DBMS. The user presents a query, the DBMS determines which physical DBs are needed to resolve the query
- Interface 2: The DBMS uses an internal model access method to access the data stored in a logical database.
- Interface 3: The internal model access methods and OS access methods access the physical records of the database.