

Converting a Logical Data Model to a Physical Database

By Tom Haughey, President

InfoModel LLC

868 Woodfield Road

Franklin Lakes, NJ 07417

(201) 755-3350 (cell, best number)

(201) 337-9094

email: tom.haughey@InfoModelUSA.com

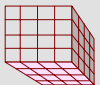
DAMA Philadelphia

January 8, 2008 © InfoModel LLC, 2008



Database Design

- Basic DBMS concepts
- Transition from logical to physical
- First-cut physical
- Safe tradeoffs
- Aggressive tradeoffs
- Technology tradeoffs
- DBMS optimizations
- Implementation of the model





Goal of Database Design

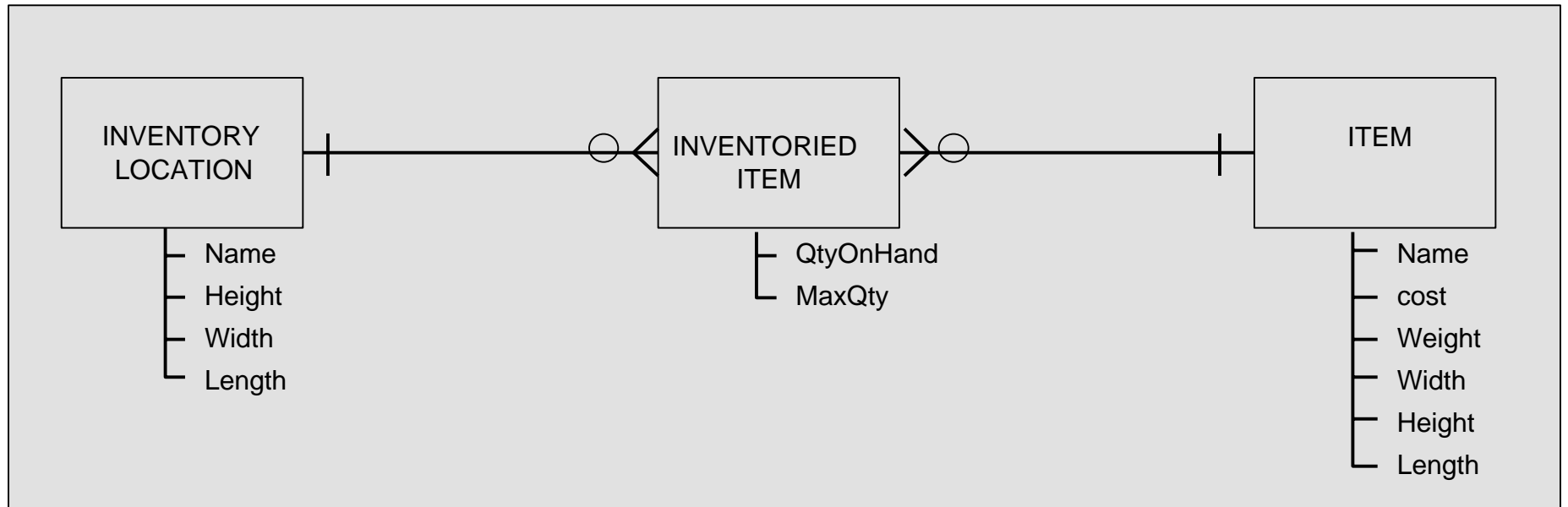
- The goal of Database Design (DBD) is to deliver an optimal database structure (and supporting modules) that are:
 - Efficient
 - Optimized to needs
 - Scalable
 - Modifiable and extendible
 - Supporting flexible user access to data
 - Stable
 - Robust
 - Secure



Logical Data Model Characteristics

Represents
business, not
implementation

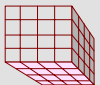
Obeys all
business rules



No
redundancy

No derived
data

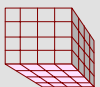
Fully
normalized



Transition Of Objects To Design

During Design, the Data Model provides logical structure of files/databases and the update-delete rules

<i>ANALYSIS OBJECT</i>		<i>DESIGN OBJECT</i>
ENTITIES	—————>	TABLES
ATTRIBUTES	—————>	COLUMNS
RELATIONSHIPS	—————>	FOREIGN KEYS, ACCESS PATHS
DERIVED DATA	—————>	CALCULATED, OR STORED
CARDINALITIES	—————>	INTEGRITY CONSTRAINTS
BUSINESS RULES	—————>	INTEGRITY CONSTRAINTS
DATA VIEWS	—————>	ACCESS PATTERNS, VIEWS
DOMAINS	—————>	DOMAINS

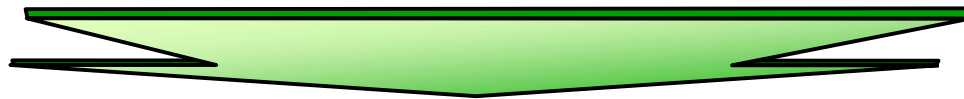


Cartesian Product

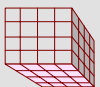
- All possible combinations of rows taken from the respective tables.
- In a search with no predicates, all combinations of rows from these tables will be returned, even though the rows may be completely unrelated.
 - Expensive to execute and often does not produce a meaningful result.
 - Some types of queries will internally yield a Cartesian product

123	Jones	History	JR	21
158	Parks	Math	GR	26
105	Anderson	Management	SN	27
271	Smith	History	JR	19

123	H350	1
105	BA490	3
123	BA490	7



123	Jones	History	JR	21	123	H350	1
123	Jones	History	JR	21	105	BA490	3
123	Jones	History	JR	21	123	BA490	7
158	Parks	Math	GR	26	123	H350	1
158	Parks	Math	GR	26	105	BA490	3
158	Parks	Math	GR	26	123	BA490	7
105	Anderson	Management	SN	27	123	H350	1
105	Anderson	Management	SN	27	105	BA490	3
105	Anderson	Management	SN	27	123	BA490	7
271	Smith	History	JR	19	123	H350	1
271	Smith	History	JR	19	105	BA490	3
271	Smith	History	JR	19	123	BA490	7





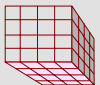
High Level Data Design

-
-
-
-
-
-
-
-



Data Optimization

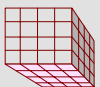
- There are three ways to optimize any system:
 - Get better hardware (or used it better),
 - Get better software (or use it better) or
 - Optimize the application (the data or the queries or both).



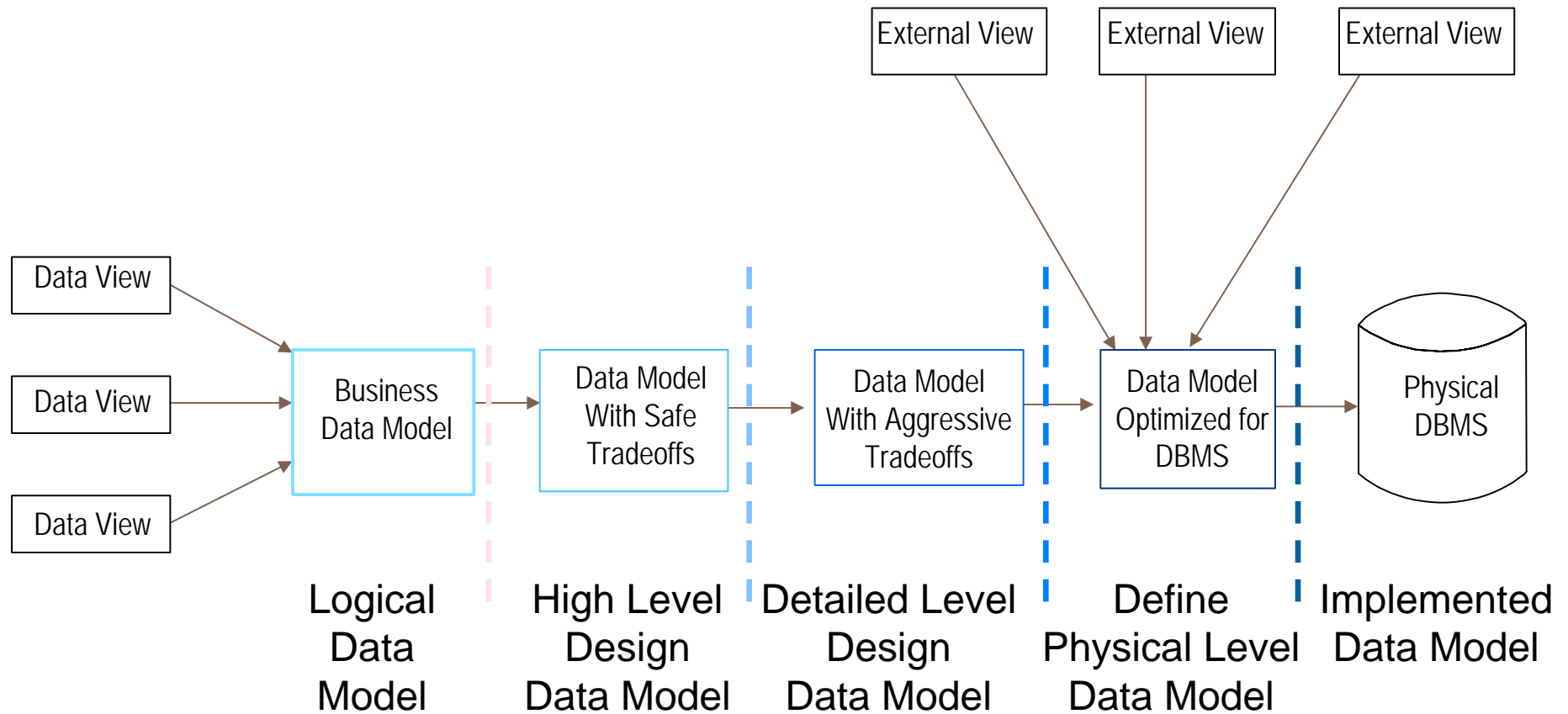


Detailed Database Design Factors

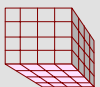
Factor	Description
Number of rows	How many occurrences are in each table?
Number of columns	How many columns are there in each table?
Ratio	How many of one table are there for one occurrence in the other table?
Model complexity	How complex are the relationships?
What data accesses do the queries make? (Query complexity)	What is the access pattern for the queries?
What is the load factor for each query (Concurrency)	How often is each query made and during which period?
What is the required response time?	What kind of performance does the business require?



Transition of Data Model Through Design



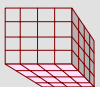
The data model does not usually undergo great change through design.



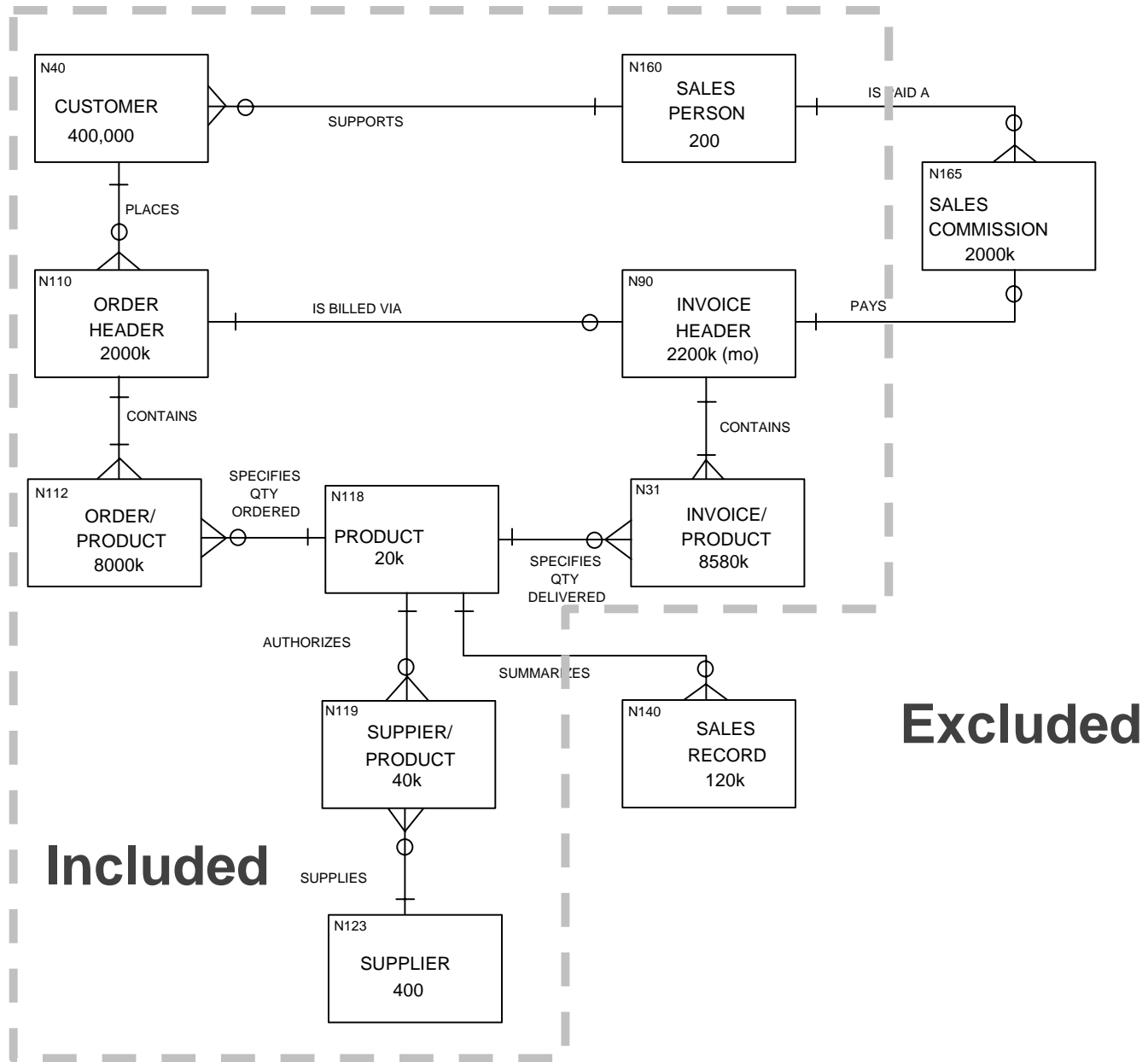


High Level Data Design Objectives

- Evaluate the logical data model
- Identify the major files and data bases to be automated
- Perform first-cut design of the files and data bases
- Optimize the data simply by applying some safe trade-offs
- Retain data integrity and non-redundancy



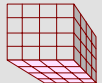
Define the Automation Boundary



— = AUTOMATION BOUNDARY

"Sales Record" done on PC on Lotus spreadsheet.

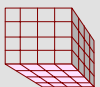
© InfoModel LLC 2008





First Cut Physical Design

- A good practice, especially in Data Warehousing, is to deliver a database 1/3 of the way through the project
- A simple way to create this is to deliver a "first cut physical"
- The following steps represent the conversion of a logical model into a ***first cut physical*** model:
 - Add data types and domains
 - Add constraints as need be
 - Resolve implementation of subtypes
 - Perform any obvious denormalization
 - Adjust keys and add surrogate keys as need be
 - Indexes on primary keys only



Safe Compromises

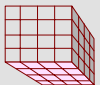
**An organization should adopt a set of standard design compromises
Certain constructs may be removed from the data model with little harm.**

Sample Safe Tradeoffs

- Promote subtypes
- Demote supertypes
- Collapse 1:1 relationships
- Split large or high usage entities
- Collapse repeating groups
- Eliminate non-representative entities
- Collapse similar constructs into one

Safe Compromises

- Do not introduce redundancy
- Do not affect data integrity
- Only combine, split or reuse entities
- Sacrifices clarity and some flexibility in favor of performance
- Simplifies design and introduces a higher degree of uniformity



Elevate Subtypes

... means to take a subtype and merge it into its supertype.

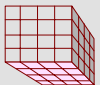
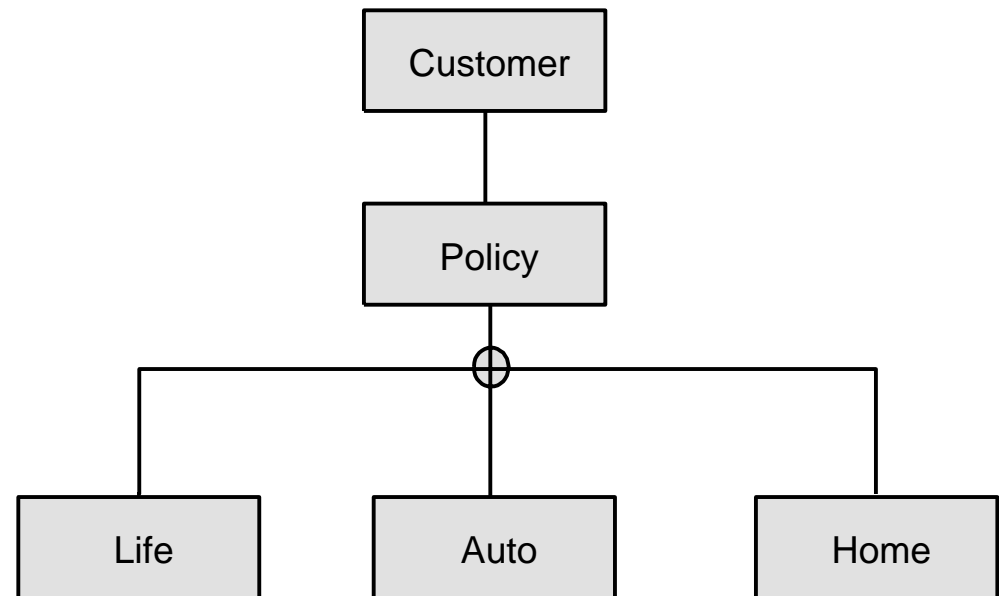
Consider the following factors: number of occurrences, number of attributes and data usage.

Advantages

- Reduces accesses.
- Provides access to the subtype directly when the supertype is retrieved.
- Simplifies the overall structure of the data.

Disadvantages

- May inhibit future growth.
- Subtype attributes can be wasted for supertype records.
- The different importance of the supertype and subtype may be lost.
- The cardinalities are often changed
- Business clarity is reduced.



Collapse Supertypes

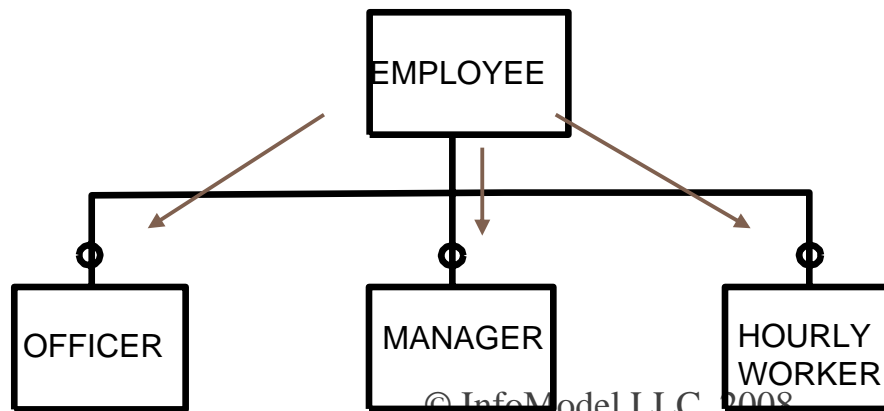
... to collapse a supertype into one or more of its subtypes.

Advantages

- Reducing accesses by eliminating the need to access the supertype for each subtype.

Disadvantages

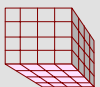
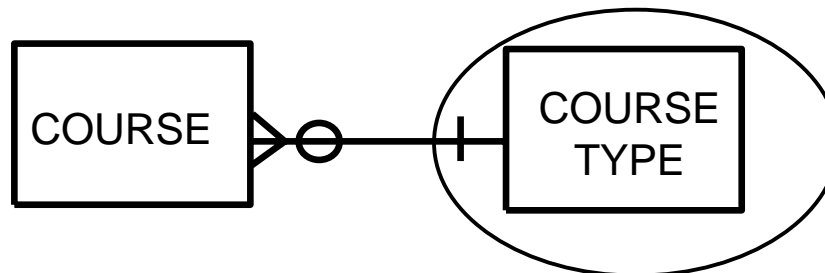
- Reduces the ability to maintain separate supertype data
- The concept represented by the subtype is lost
- And flexibility is diminished
- Essentially, business clarity is reduced.



Eliminate Unimportant Code Tables

... means to eliminate code, status or non-representative entities. These consist of a key and one attribute, such as a name, and have few, or no, relationships. These are usually small reference tables or other descriptive entities.

- They can be collapsed into the entity (or entities) that use them when they do not cause significant redundancy
- Best if the non-representative data is stable (e.g., will not be renamed)



Collapse 1:1 Relationships

... to take the entities involved in a 1:1 relationship (and that are very strongly related to each other) and make a single entity out of them.

For example, merge LOAN and COLLATERAL. This assumes that:

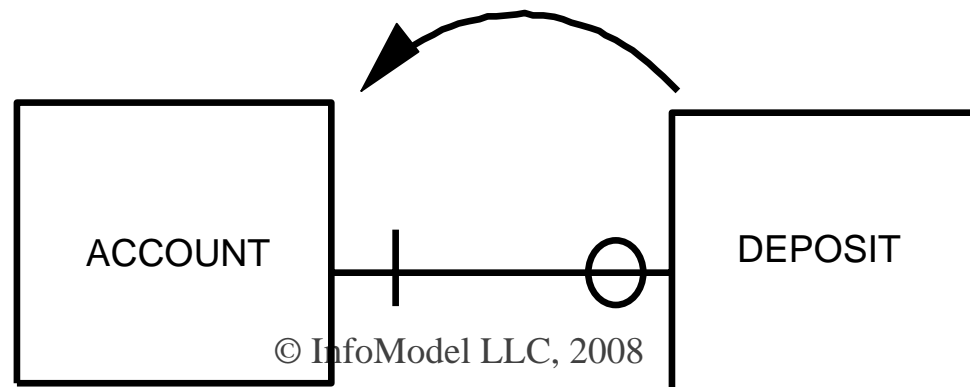
- COLLATERAL is not expected to grow or change significantly in rules
- it is not a major entity of the business and
- it is very strongly related to LOAN

Advantages

- reduced accesses
- reduced files or data bases

Disadvantages

- LOAN and COLLATERAL can not grow
- reduced flexibility
- close binding between LOAN and COLLATERAL

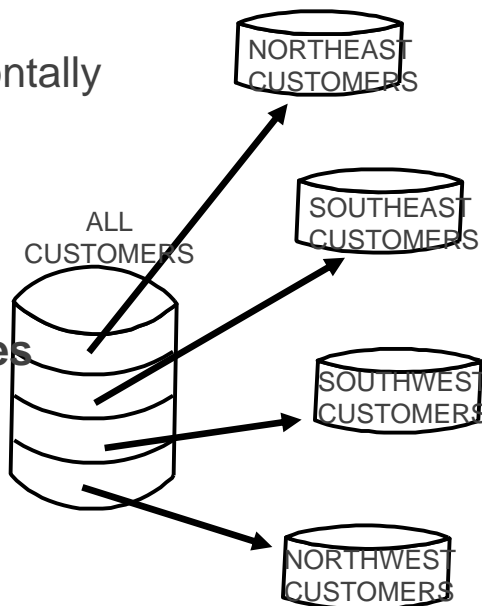


Partition or Distribute High Usage Tables

... refers to entities that have a large number of occurrences or a large number of attributes,

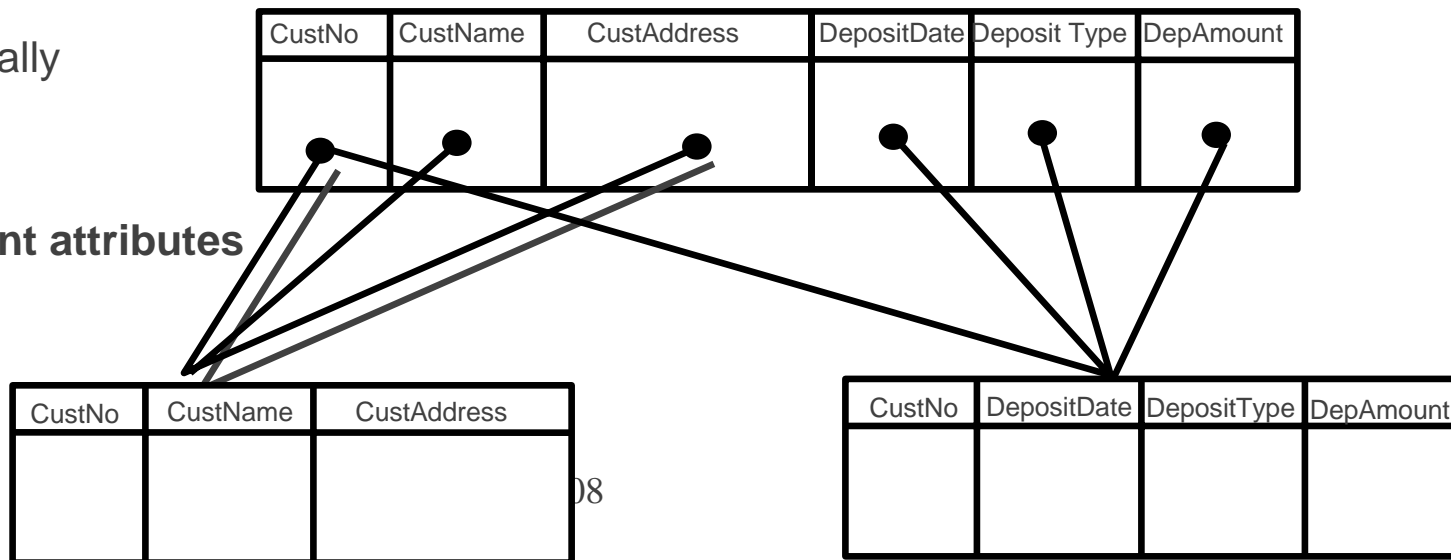
- can be subdivided horizontally

Same attributes, different values



- or vertically

Same values, different attributes

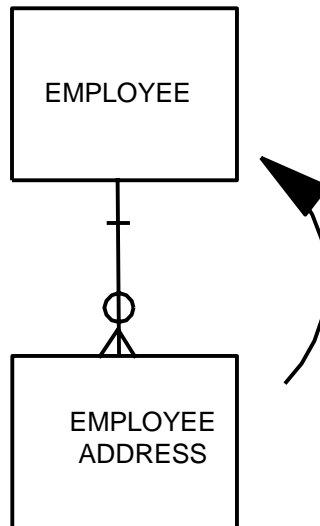


Collapse Repeating Groups of Attributes

If the access to the repeating data is high, it can be advantageous to group it together with the primary entity.

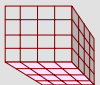
Compromises First Normal Form

For example, say an EMPLOYEE could have several ADDRESSES. We could collapse the ADDRESSES into the EMPLOYEE entity as a repeating attribute within it.



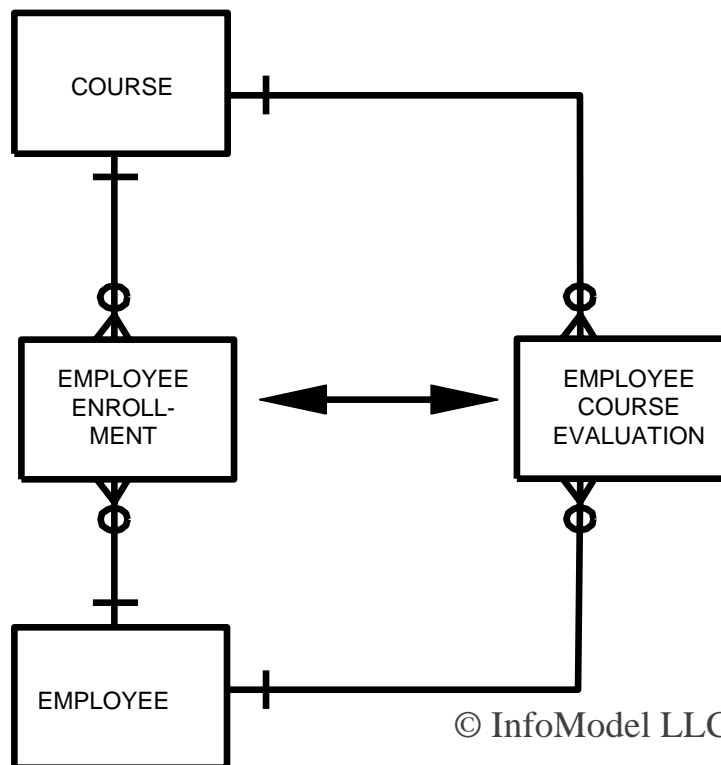
This approach will work in the following situations:

- a row contains few data elements
- data elements accessed frequently in a sequential order
- the pattern of accesses is stable
- the pattern of accesses is via the primary key
- the number of data elements is predictable



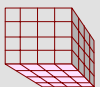
Collapse Similar Constructs Into One

- Often happens when two entities that are very similar in usage are modeled separately.
 - Combined to simplify the structure and reduce accesses.
 - Often these are associative entities, but is not limited to them
 - We could merge EMPLOYEE ENROLLMENT and EMPLOYEE COURSE EVALUATION and put the RATING attribute into the new entity



- Requires entities with:

- Similar keys
- Similar grains
- Compatible data





Detailed Level Data Design

-
-
-
-
-
-
-
-



Aggressive Compromises

Several general types of compromises are possible:

- Store Derived Data
- Store Redundant Data or Relationships
- Alter keys and access sequences

Basic Idea Of These Compromises:

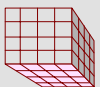
- Modify the record structure to better ensure that the data you need together is kept together.

However, . . .

- They compromise two most important qualities of the data structure: Data Integrity and Data Redundancy.

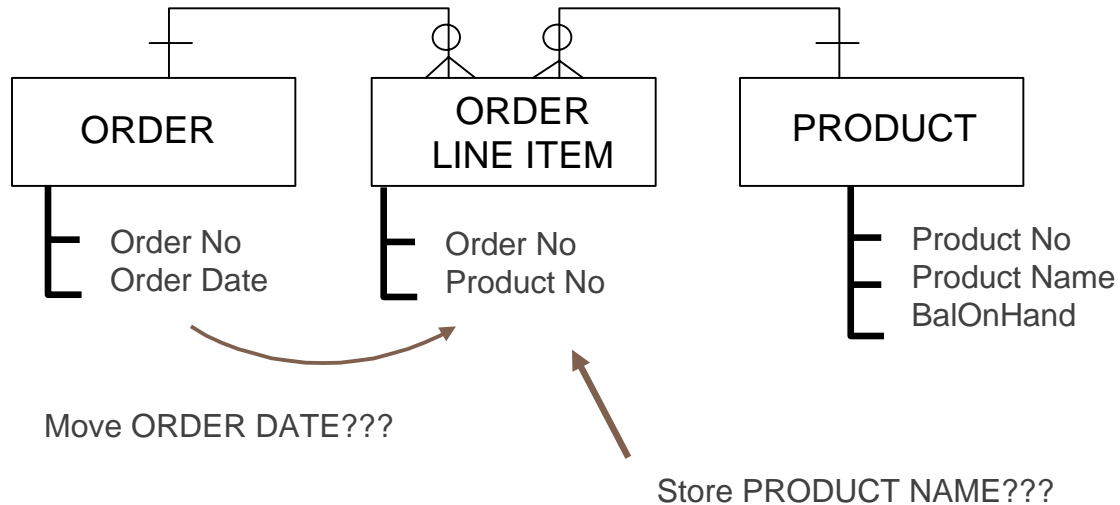
Prerequisites:

- Perform access path analysis
- Prototype the current design
- Test design choices
- Compromise as little as possible
- Iterate until performance is acceptable



Store Redundant Data - 2NF or 3NF Compromise

Data is dependent on only part of the key

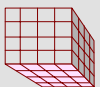


Pros

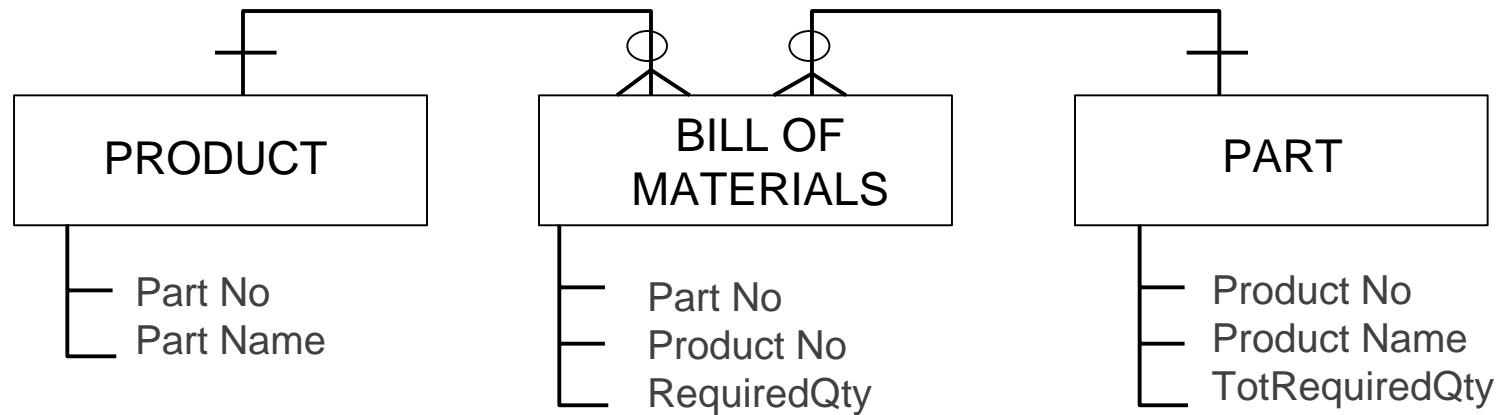
- PRODUCT NAME and ORDER DATE available with LINE ITEM

Cons

- PRODUCT NAME vulnerable to inconsistency; ORDER DATE stable.
- PRODUCT NAME and ORDER DATE repeated in ORDER LINE ITEMS
- Updating PRODUCT NAME expensive because same value tracked down and changed on multiple tables and rows
- If PRODUCT NAME too large, a lot of extra space will be used



Store Derived Data



Store TotRequiredQty???

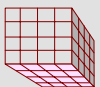
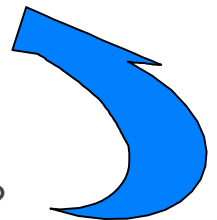
TotRequiredQty = Sum of RequiredQty for this Inventoried Part

Pros

TotRequiredQty immediately available when needed
Eliminates calculation of TotRequiredQty upon retrieval

Cons

TotRequiredQty vulnerable to inconsistency with RequiredQty
TotRequiredQty must be updated whenever RequiredQty changes
TotRequiredQty takes disk space



Types of Derived Data

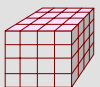
- Derived Data can come in two general forms:
 - A **derived column** added to an existing table, such as an Order Total added to the Order table. This saves having to calculate it on the fly (especially if there are a lot of line items).
 - **Derived tables**, such as a separate table with sales data by Customer by Product by Week. This saves having to roll up individual customer transactions to determine Customer Sales by Product by Week each time it is needed. Such tables are often called:
 - Aggregate tables/aggregates
 - Summary tables
 - Rollups





Usage of Stored Derivations

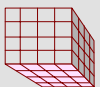
- Derivations are typically stored differently in different types of systems.
 - **Operational** systems, because of their update nature, very often store derived columns, with more limited use of summary tables.
 - **Analytical** systems (like data warehouses), because of their query nature, very often store both derived columns and summary tables.





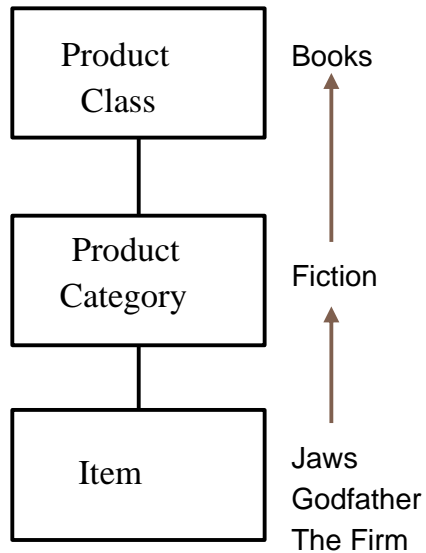
Aggregation Best Practice

- The summarization of data to a higher level than the previous level such as:
 - Order totals roll-up to
 - Customer totals, which roll-up to
 - Salesperson totals
- Aggregate where:
 - Base data contains many possible values.
 - Aggregated data would be dense and abundant
 - Aggregates are used often
- Aggregation is often called roll-up
- ***The 10 : 1 rule***

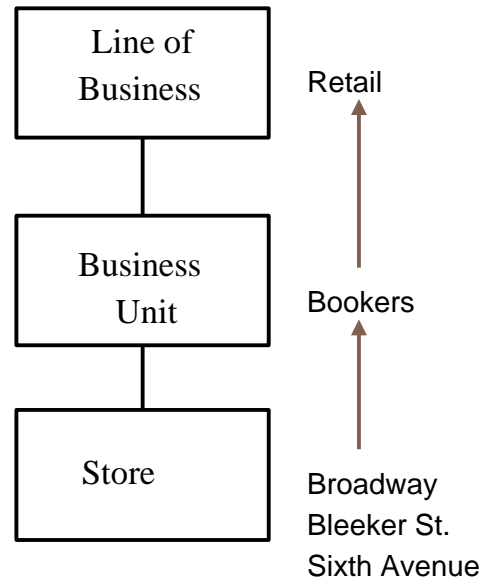


Sample Rollup Hierarchies

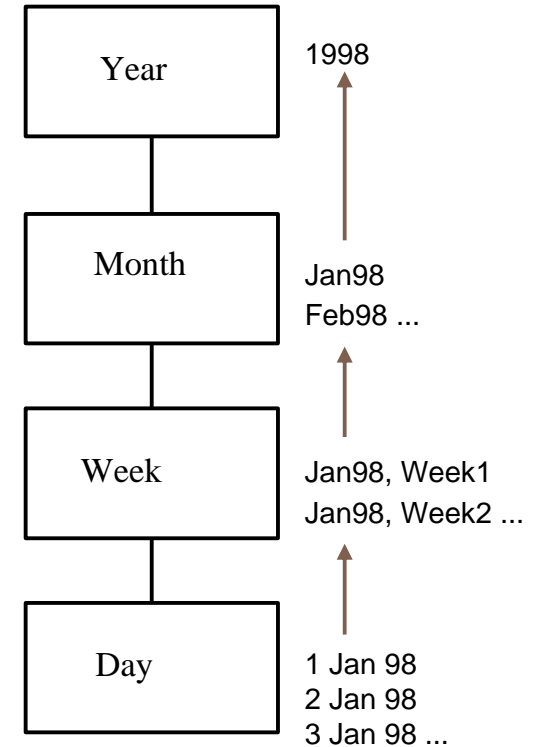
Product Hierarchy



Organization Hierarchy

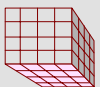


Period Hierarchy

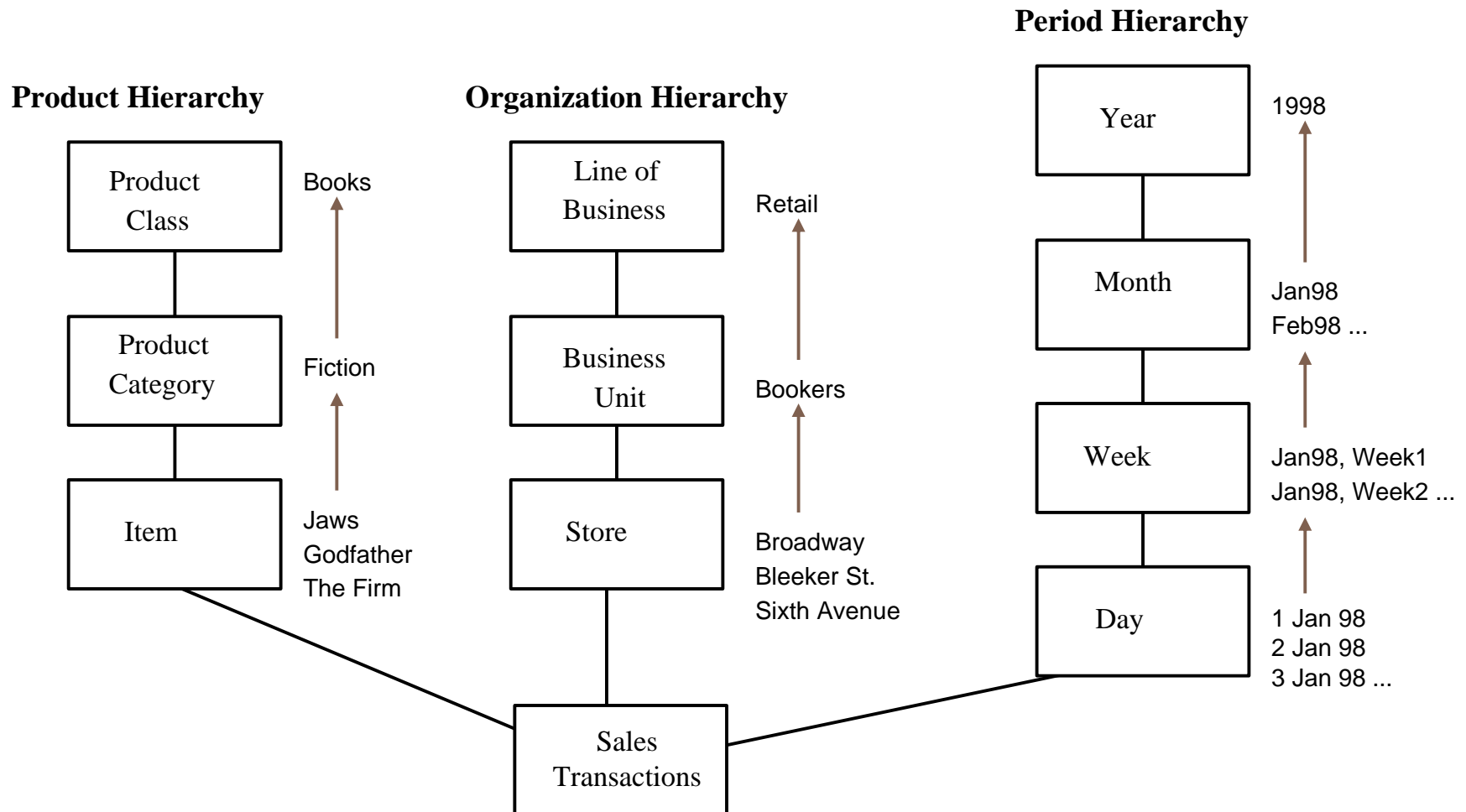


■ The most typical roll-up or summarization dimensions are:

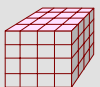
- Organization
- Product
- Time
- Geography
- Business Party



Determining Appropriate Rollup Levels



- Hierarchies like this are often used to roll data up along different lines.
- The trick is to find the optimum combination, such as Item by Store by Week.
- Creating summaries for all permutations of the hierarchies is unthinkable and unnecessary.



Vertical Summarizations

- Summarizations building upon a single dimensional theme:
 - Good for quick, general volumes, indicators and counters
 - Fast response time for recurring questions about popular topics, such as:

Loan Status

Loan ID

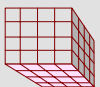
On Time Count
30 Days Late Count
60 Days Late Count
90 Days Late Count
Never Paid Indicator

Monthly Customers Sales

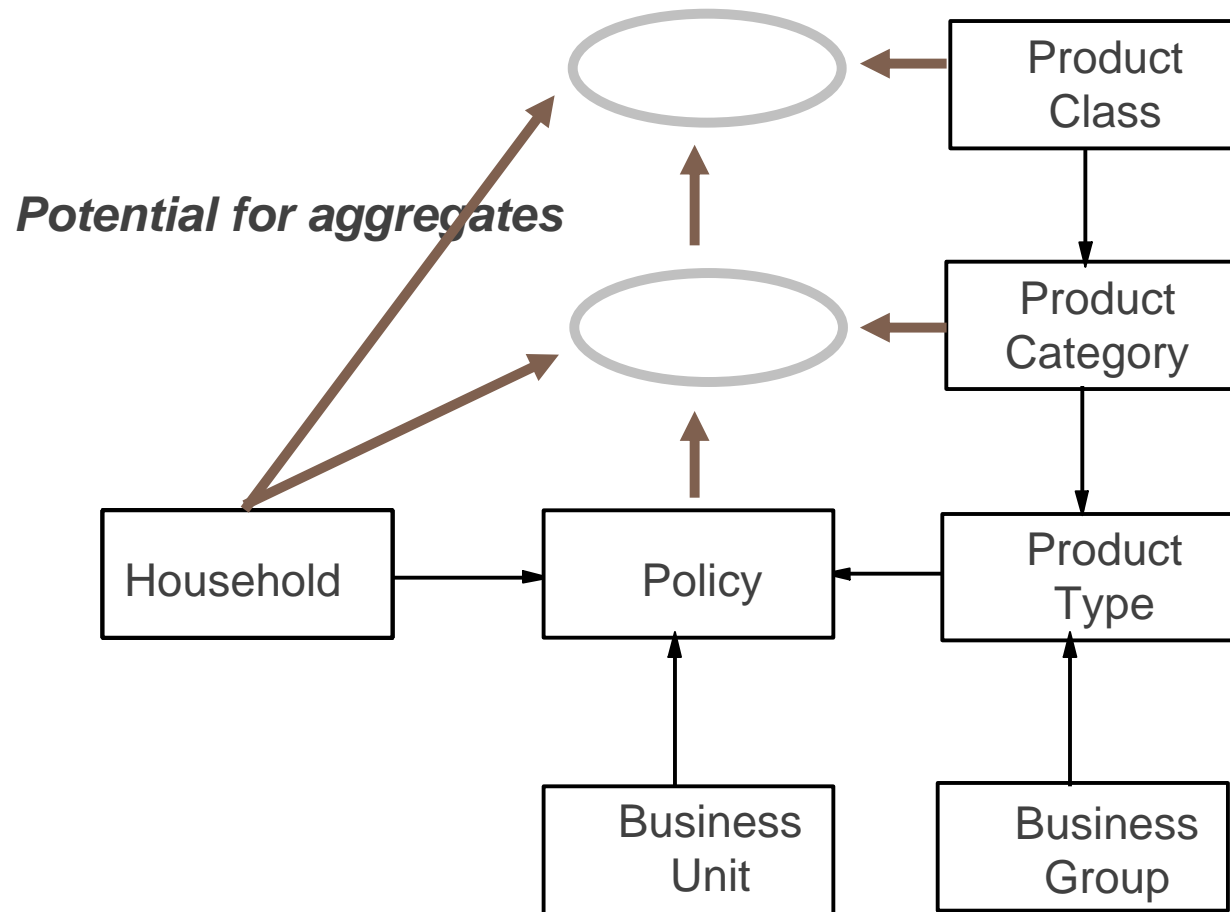
Branch ID

Date

Total # of all customers
Total # of new customers
Total # of inactive customers
Total sales \$
Total sales #

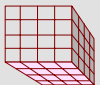
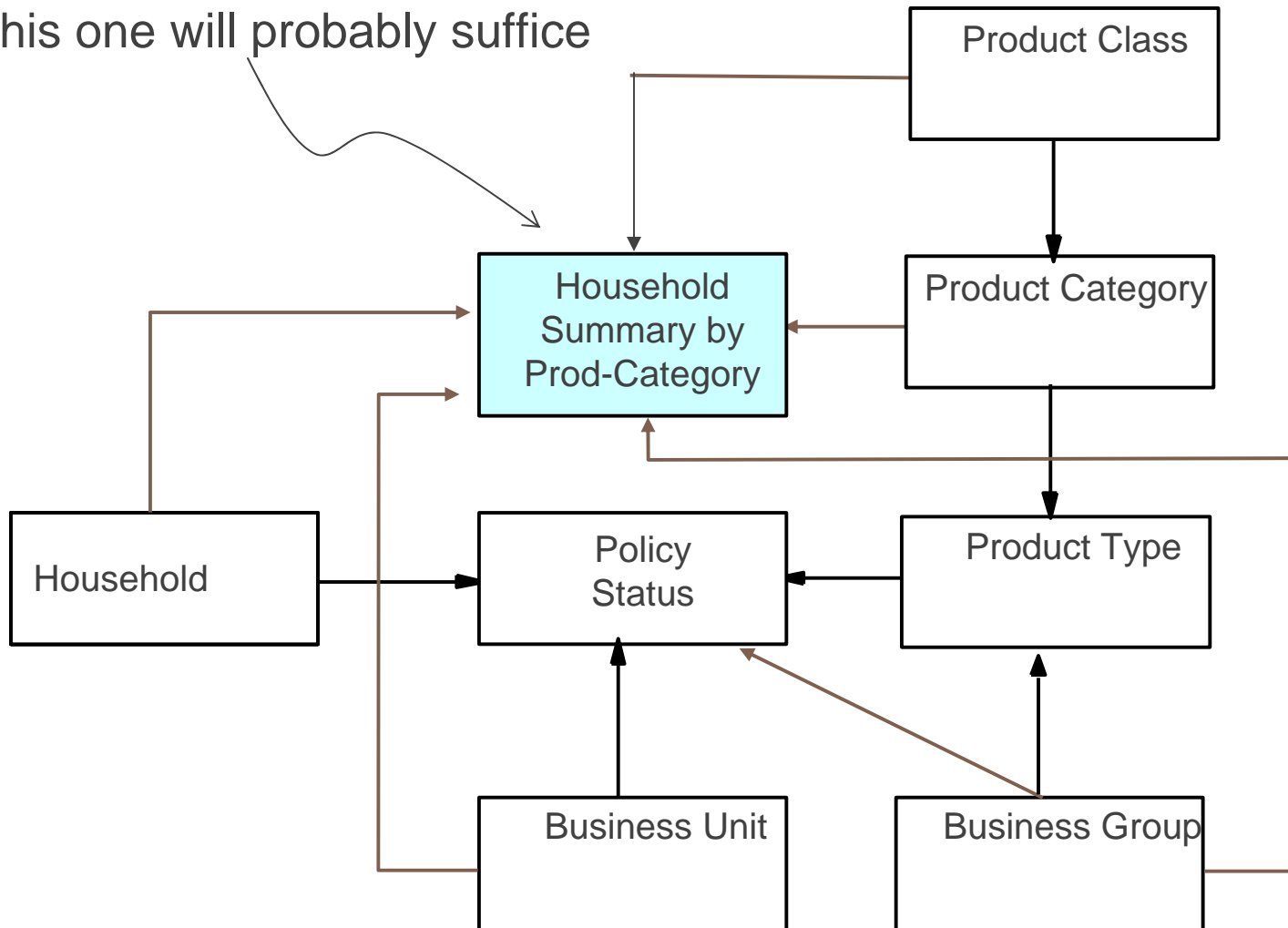


Database Without Aggregates



Database with Aggregates

- This one will probably suffice



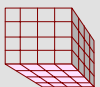


Surrogate vs. Natural Keys

- A **Natural Key** consists of one or more business attributes that serve as the primary identifier of the entity
 - It has three characteristics:
 - Unique
 - Stable
 - Minimal

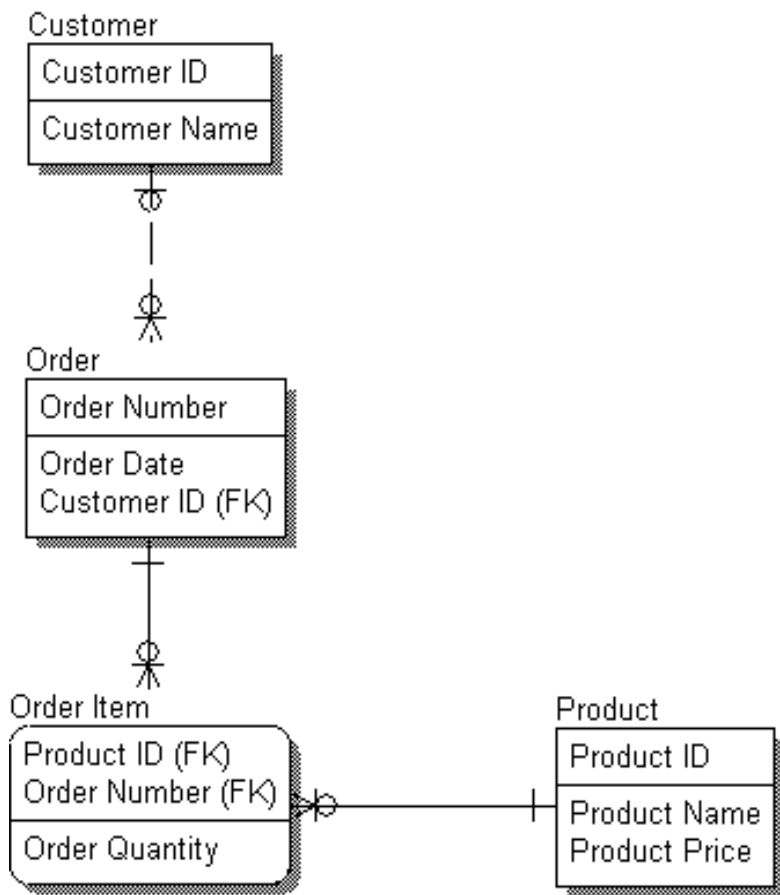
- A **Surrogate Key** is a system generated key that replaces the natural key
 - Surrogate keys have three main purposes:
 - To reduce space requirements
 - To insulate the DW from operational changes
 - To (possibly) improve performance
 - It has three additional characteristics:
 - Immutable
 - Simple
 - Non-intelligent

Surrogate keys are not peculiar to the dimensional model

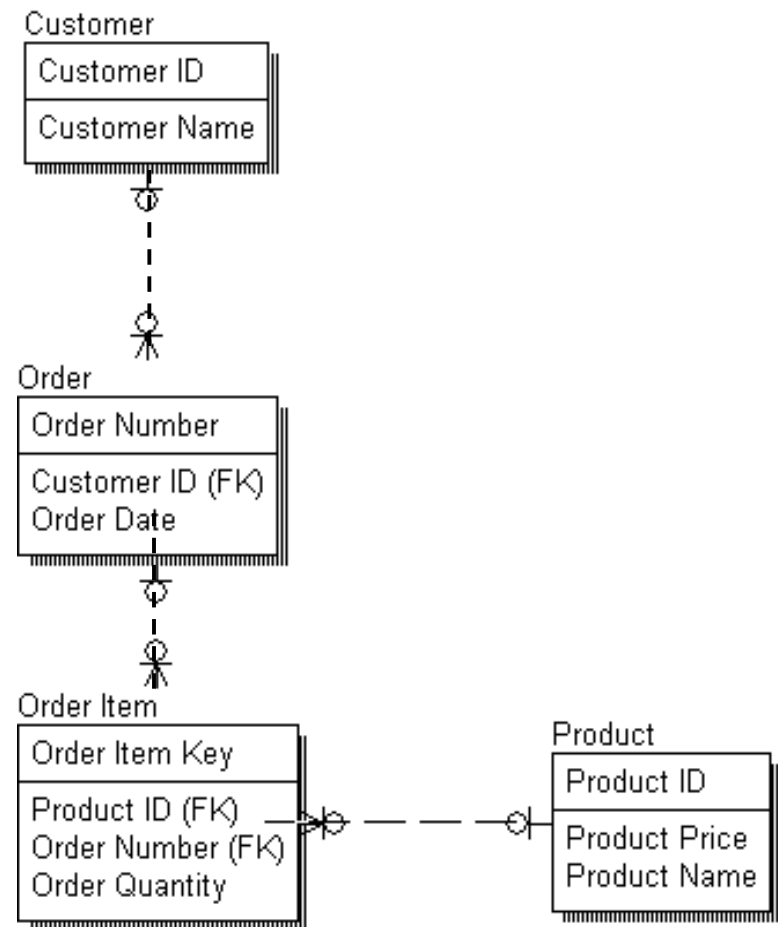


Use of Surrogate Keys

- Logical models, which are business oriented, should always use natural keys; physical data models, or databases, can use surrogate keys.
 - Natural keys enforce identity; surrogate keys disguise identity
 - Chris Date recommends surrogate keys; Joe Celko prefers natural keys.
- Who is right?
- It depends



Natural Order Item Key



Surrogate Order Item Key





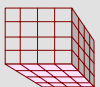
Surrogate Keys vs. Natural Keys

■ Pros of Surrogate Keys

- Reduced space by replacing a large combined primary key, especially when used as a foreign key
- Isolation of the data from operational changes
- Ability to have multiple instances against a given entity
- Speed of access when the optimizer uses a simple, numeric index

■ Cons of Surrogate Keys

- (Possible) Extra storage requirement due to an extra column (if both natural and surrogate are retained)
- Integrity having to be enforced some other way
- Still need access to the data using natural keys
- May have to supplement surrogate keys with a mapping table correlating surrogate key and its equivalent natural key.



Partitioning

- There is a logical division of one fact table into several smaller tables.
 - Remember to partition based on USAGE, NOT JUST CONVENIENCE.

Unpartitioned Table

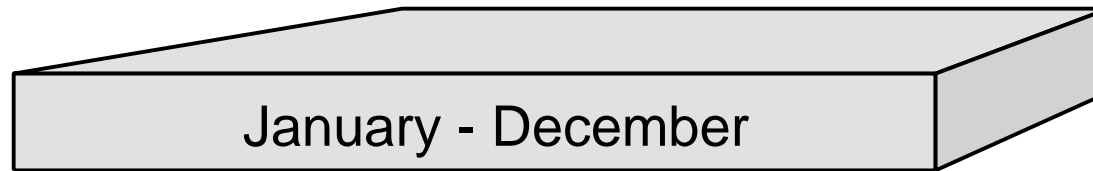
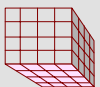
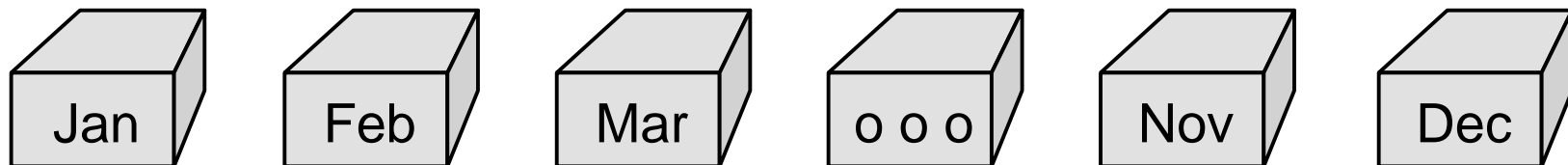


Table Partitioned by Month





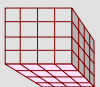
Indexing

-
-
-
-
-
-
-
-



Indices

- The "index" functions like a smaller version of the table.
- Like a list of page references in a book's index, a database index provides direct access to the rows of interest without scanning the whole table.
- Much faster than a full-table scan.
- Automatically created when you define a uniqueness constraint or primary key constraint on a column
 - Pros
 - Help the system access data quickly
 - Can enforce uniqueness of the values in a table
 - Provides logical ordering on the rows of the table
 - Can provide physical ordering of the rows of a table
 - Cons
 - Space: each index takes up room
 - Reduced performance for insert, update and delete operations
- Vendors often have index wizards and other index tools.
 - E.g., IBM UDB Index Advisor





General Index Types

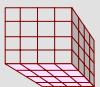
- B-Tree Indexes

- used for high cardinality columns
- designed for few rows returned
- standard in most databases

- Bit-mapped Indexes

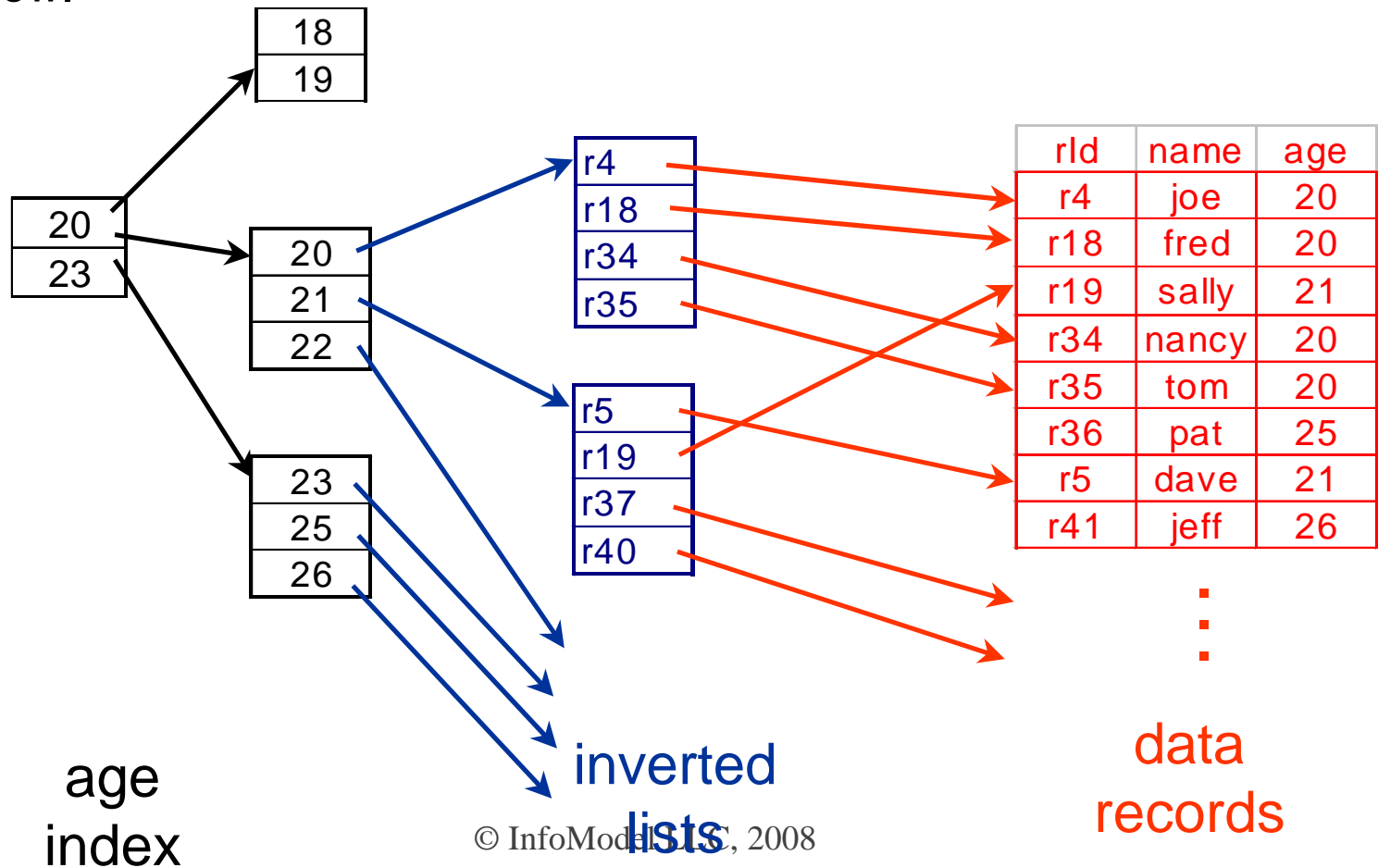
- used for low cardinality columns
- designed for many rows returned
- support varies with database vendors

- Unlike OLTP, data warehouse databases use indexes extensively.



B-Tree Index

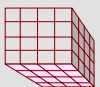
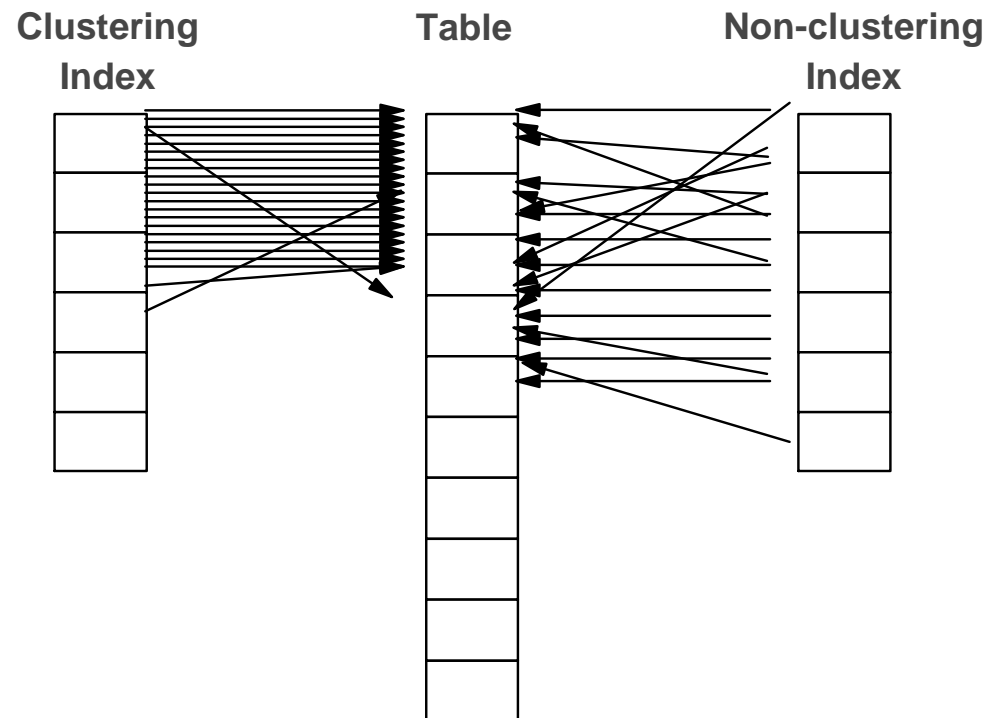
- Traditional indexing technique, used by most RDBMS vendors
- Contains a hierarchy of highest-level and succeeding lower-level index blocks
 - The upper level blocks (called branch blocks) point to the lower-level blocks.
 - The leaf blocks (lower-level blocks) contain the unique ROWID of the actual row.



Clustered Indices

- Clustering an index means storing data rows according to the order of the index.
 - Searching on a clustered index is much faster.
 - You can have only one clustered index per table
 - Usually, the clustering index is the primary key.
 - You don't need to explicitly create indexes on primary key columns

- To qualify for the clustered index, the column should be:
 - 1. Non-volatile
 - 2. Small (preferably an integer)
 - 3. Unique
 - 4. A column frequently searched via a range
 - 5. In the order of preferred retrieval.

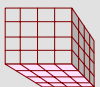
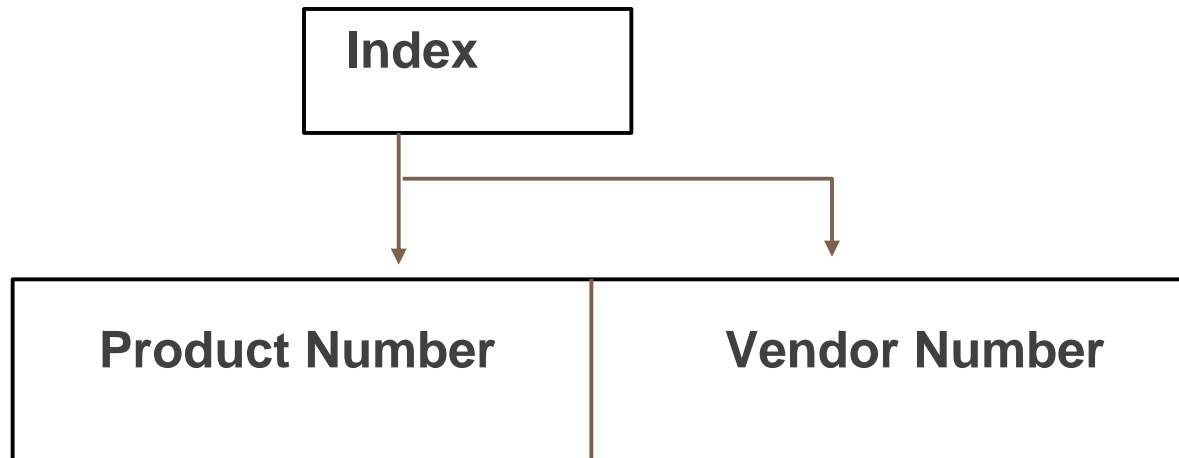


Composite Indices

- Multi-column indices

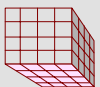
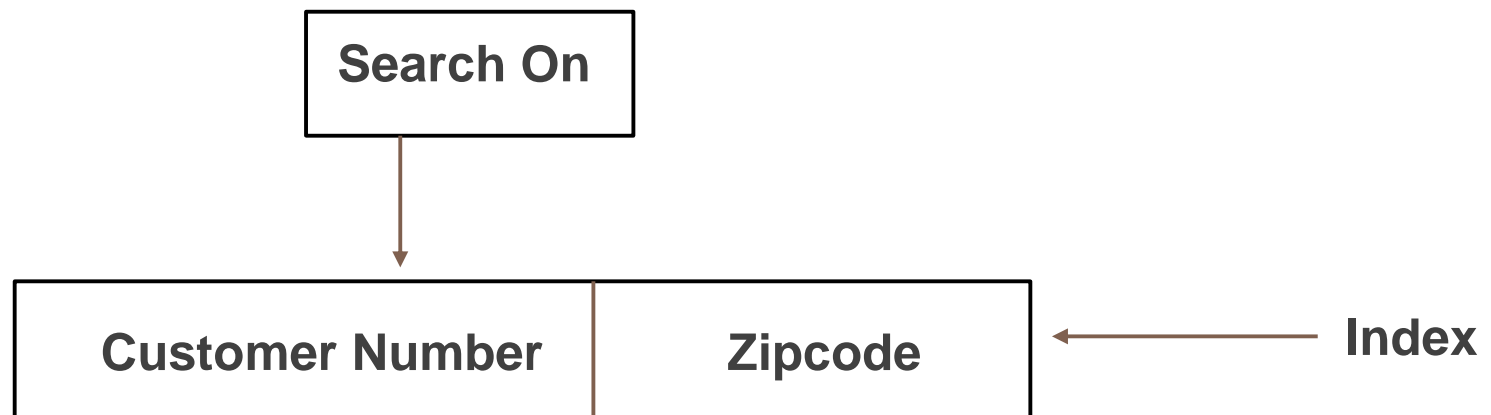
- Leftmost should be the one that occurs most often
- Can use up to five columns in a compound index
- Sometimes a cost-based optimizer will choose to scan a table instead of using indices

- E.g., if a column satisfies a predicate 90% of the time, the optimizer will choose a table scan (using I/O pre-fetch)



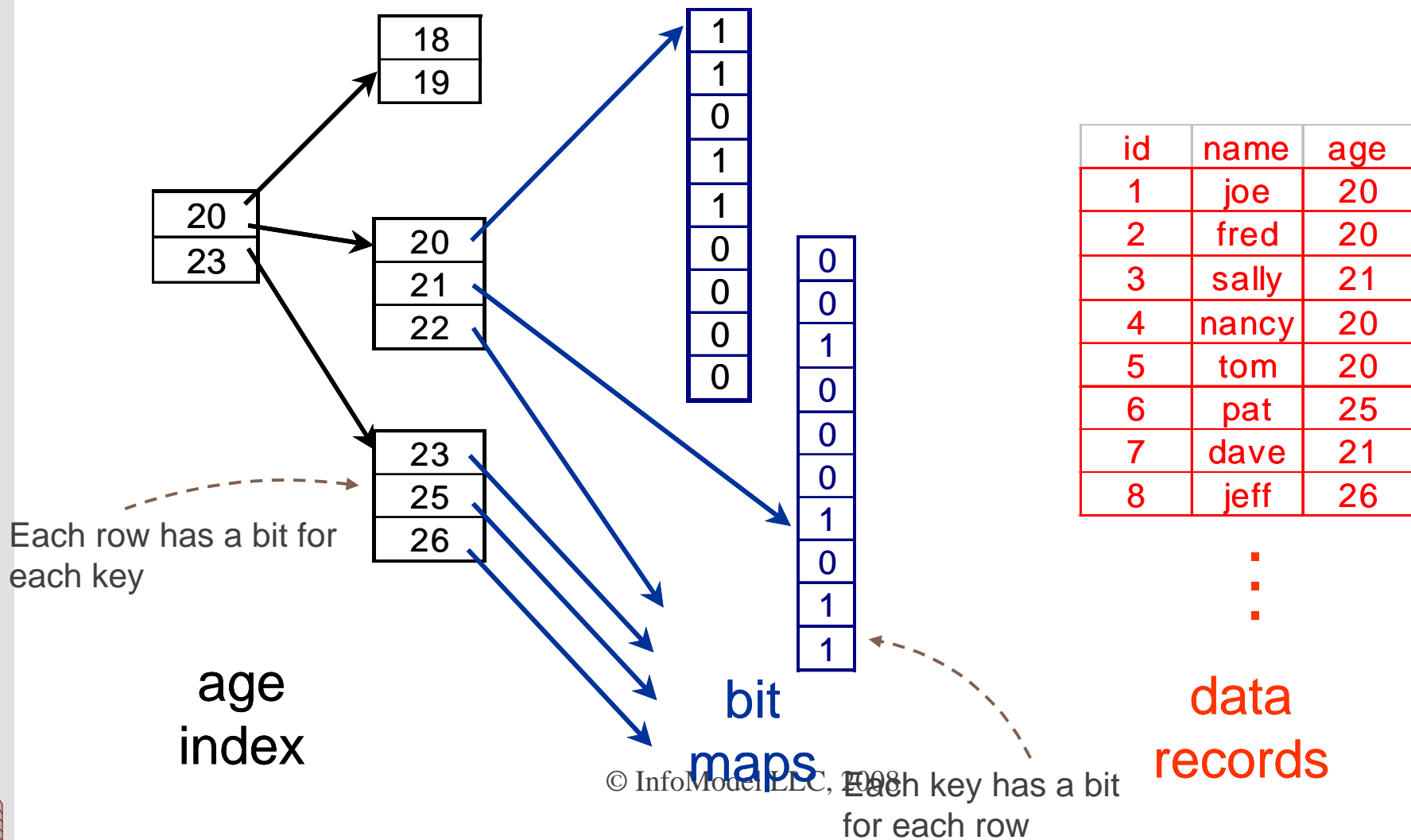
Covering Index

- An index that includes columns other than the index search field and that are often used together with the index field as a predicate
 - Save disk reads (when all SELECT columns are in the index)
 - If table has many rows, a covering index will dramatically improve performance
 - Index acts like a small table
 - Query will read only the index to get the result set



Bit-mapped Indices

- Uses binary encoding to represent presence of data values
- Used for low cardinality columns
- Appropriate for data warehousing environments where ad-hoc queries are more common and data is updated less frequently





Choosing Bit-mapped Indices

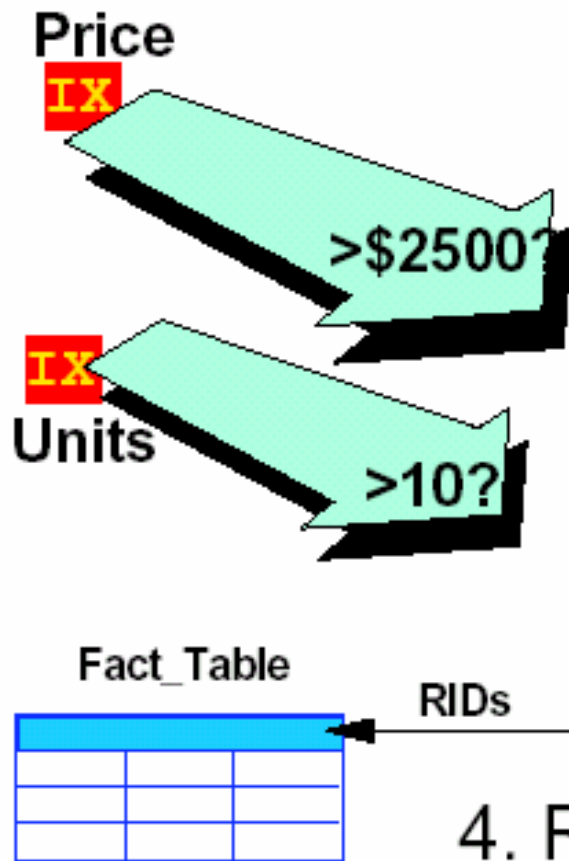
- Bitmap indexes should be chosen when the data and queries have the following characteristics:
 - Low cardinality - small number of distinct values
 - Low selectivity - # of distinct values / total # of values
 - ANDing or ORing of predicates
 - An index is available
 - A dimensional type structure
 - Queries with =, <, >, [NOT] EXISTS, UNIQUE, GROUP BY
 - The optimizer determines it is more cost effective (UDB)

- In some DBMSs, the DBA can build bit-mapped indexes explicitly; in others, the Optimizer can make this choice



Bit-mapped Index Example

Bitmap technology



Select ... From Fact_Table
Where Price > \$2500 and Units > 10

1. Scan Indexes
2. Generate Bitmaps

1 0 0 1 1 0 1 1 1 0 1 0 1 0 1 1 0 0 1

3. AND Bitmaps

1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1

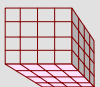
1 0 0 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1

4. Retrieve Qualifying Rows



Indexing Guidelines

- Don't over-index the database. Start with basic indexes, monitor performance and refine your indexing strategy.
- Expand to indexes on fields in query predicate (WHERE clause).
- Increase performance of applications by issuing queries that select from the smaller tables in the schema first and by making efficient use of aggregate data.
- Composite indexes (sometimes called multicolumn indexes) are generally recommended if queries often contain the same columns joined with AND.
- Use different types of indexes, based on data distribution
 - Bitmapped indexes for low cardinality
 - B-Tree indexes for high cardinality.
- The left-most column in a compound index should be the one that occurs most often in the queries.
- It may be faster to do a full table scan than to use an index when you practically hit all records anyway.
- Remember, the more indexes you have, the longer your load or update process will take.

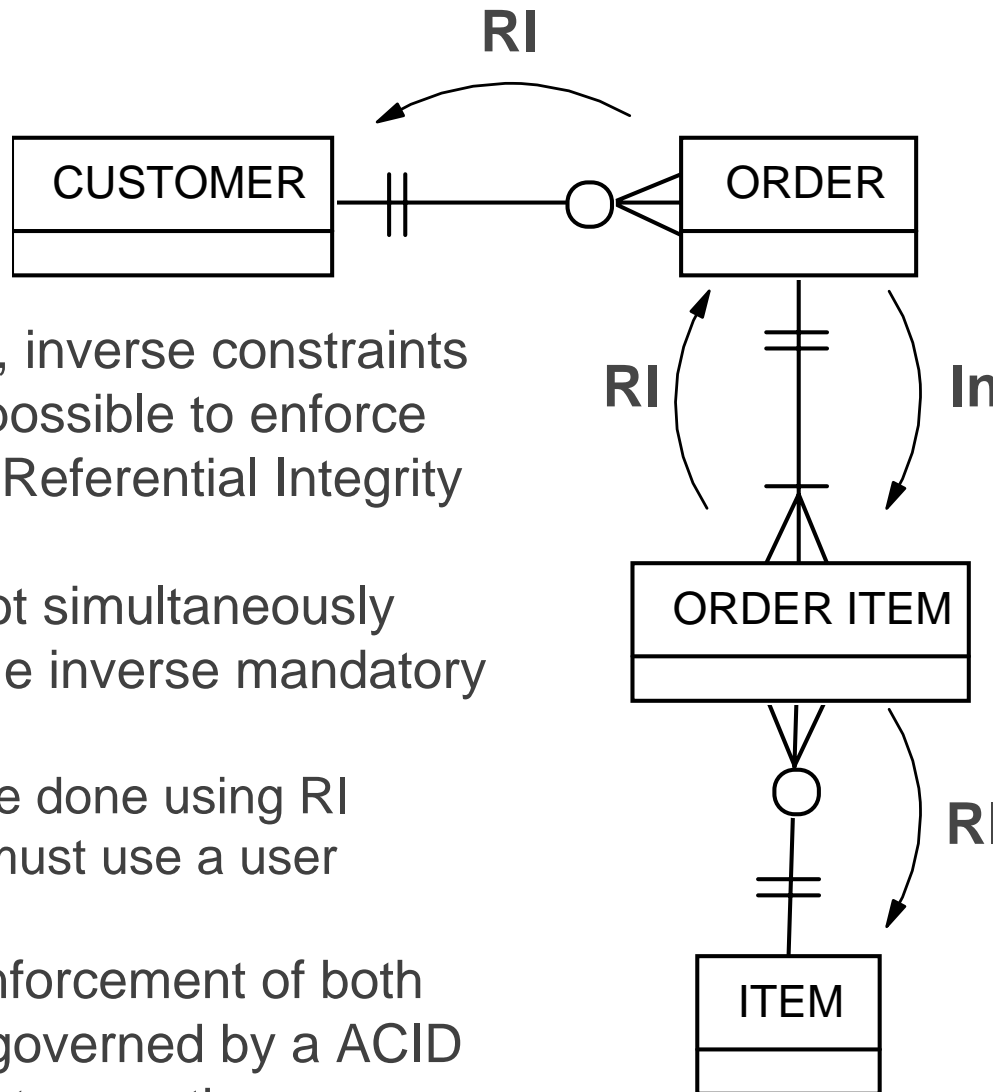




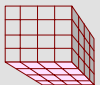
Integrity Modules

-
-
-
-
-
-
-
-

Enforcing Constraints



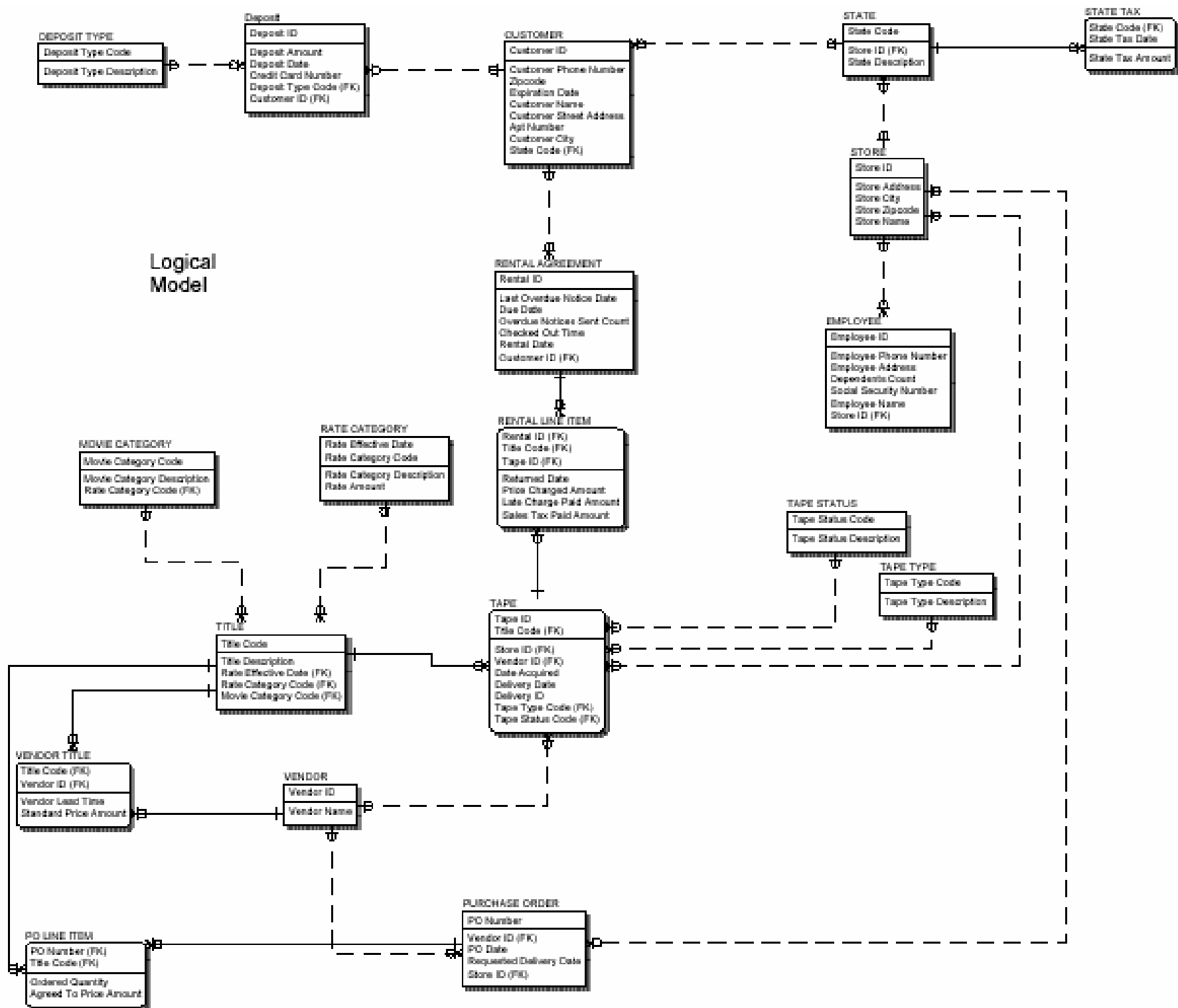
- When multiple, inverse constraints exist, it is not possible to enforce them all using Referential Integrity
- DBMSs can not simultaneously enforce multiple inverse mandatory relationships:
 - One may be done using RI
 - The other must use a user constraint
- Remember, enforcement of both constraints is governed by a ACID properties of a transaction, not by pure physical simultaneity





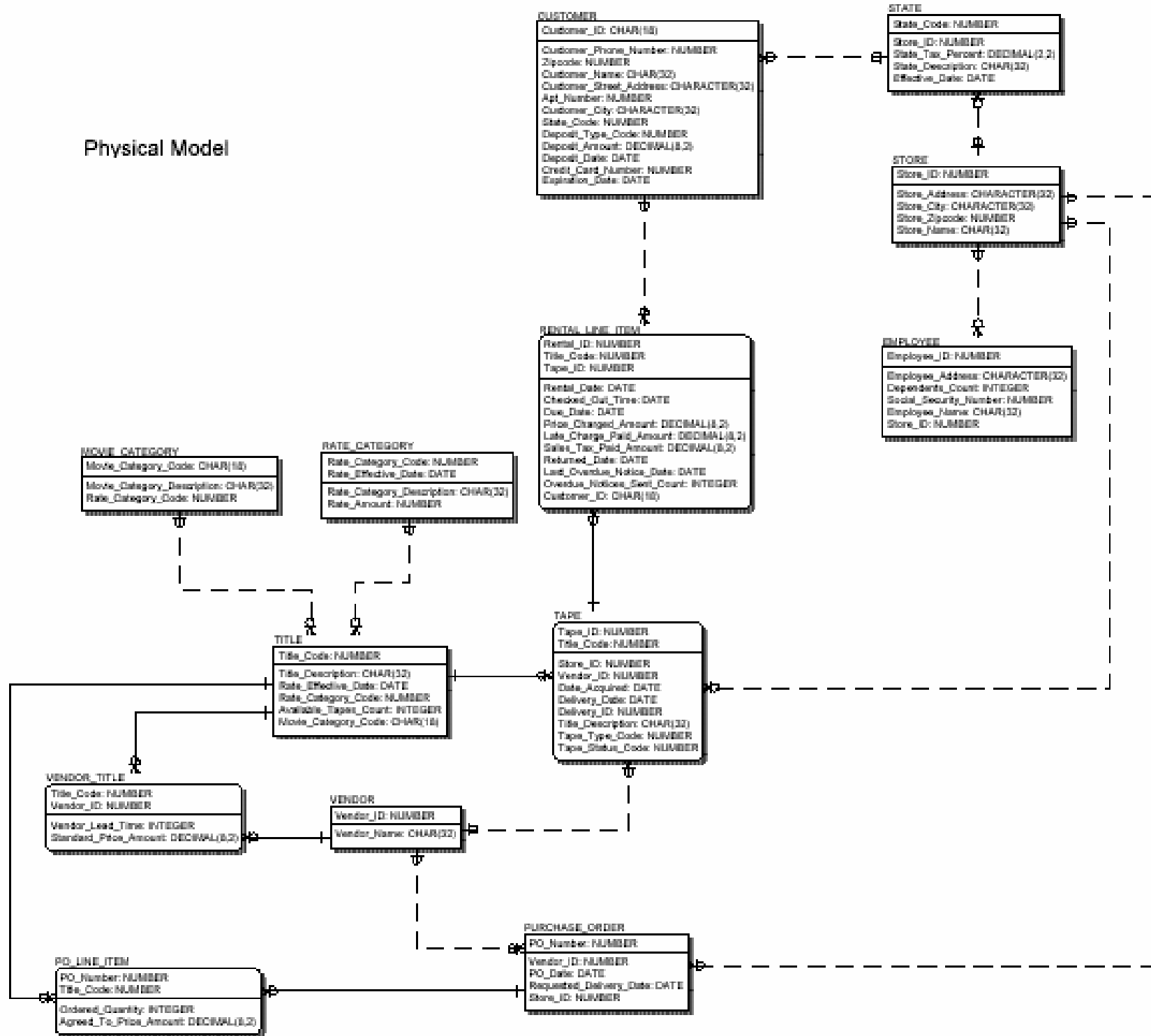
Sample Model





Logical Model

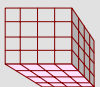
Physical Model





Possible Optimizations

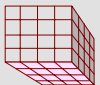
- Deposit collapsed into Customer (1:1)
- Rental Agreement collapse into Line Item because:
 - Tapes due on different dates and returned on different dates;
 - Few attributes in rental agreement;
 - Overdue notices sent by rental line item.
 - Ratio of rental agreement to line item is mostly 1:1.
- Tax calculated and stored in Rental Line Item
- Current Tax Percent moved to State and only current rate kept.
- Title Description stored in Tape to eliminate join.
- Available Tape Count calculated and stored so speed access.
- Tape Status collapsed into Tape.
- Indices on primary keys only.





Summary

- Basic DBMS concepts
- Transition from logical to physical
- First-cut physical
- Safe tradeoffs
- Aggressive tradeoffs
- Technology tradeoffs
- DBMS optimizations
- Implementation of the model





Further Questions?

