第四章 维度建模方法

北京大学信息科学与技术学院

童云海

2005年5月

本章内容

- 数据仓库的最终目标
- 数据仓库建模方法
- 维度建模的相关概念
- 维度建模的基本步骤

数据仓库的最终目标

- 数据仓库必须使组织机构的信息变得容易存取
  - 容易理解，见名知义
  - 存取工具必须简单易用，存取速度快
- 数据仓库必须一致地展示组织机构的信息
  - 数据具有可信性
  - 高质量的数据：一致的、完整的、定义唯一理解的
- 数据仓库必须具有广泛的适应性和便于修改
  - 变化：用户需求、业务情形、数据内容和技术状况
  - 新数据的加入，现有数据和应用不应该发生改变或者崩溃

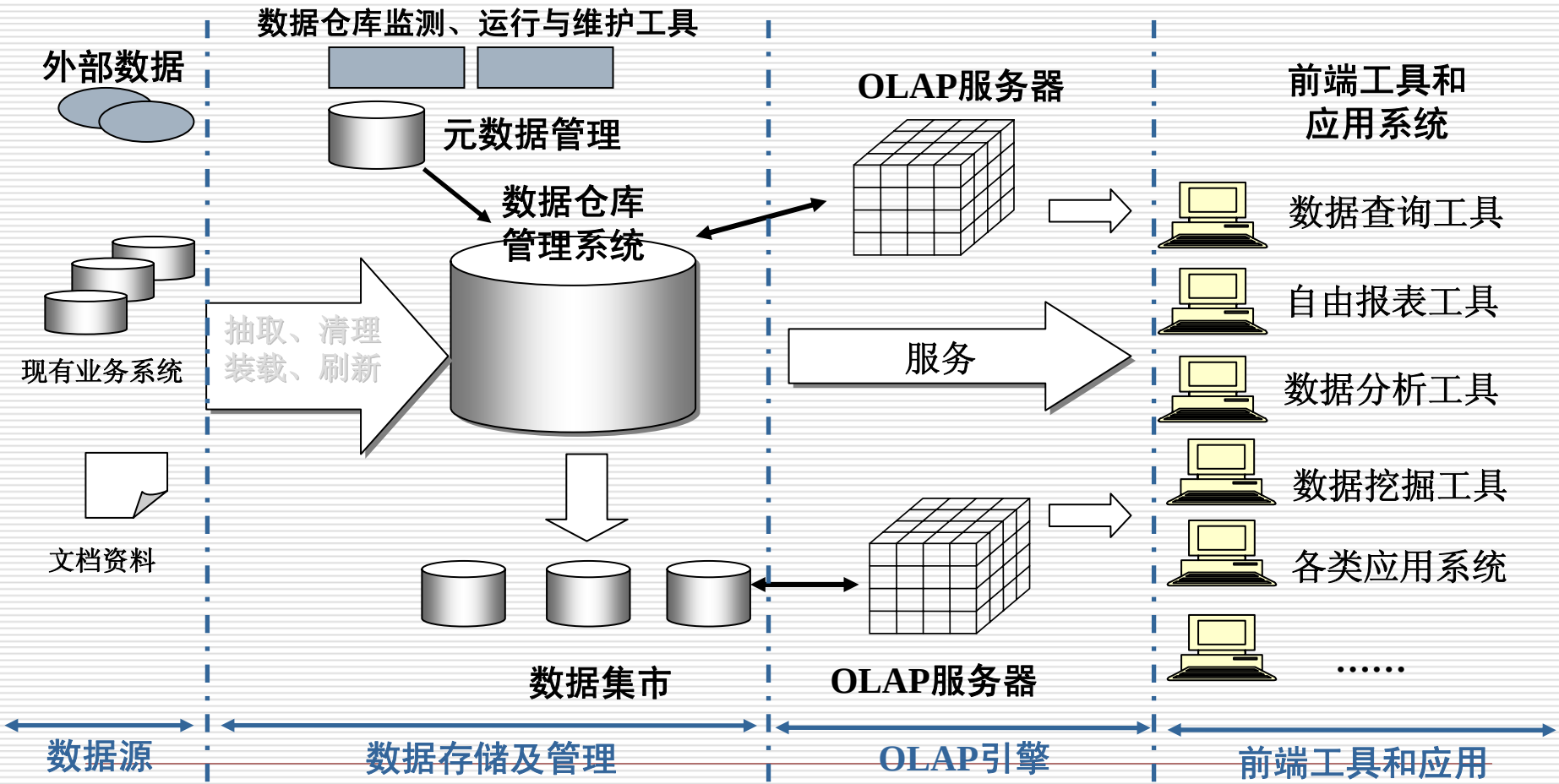
数据仓库的最终目标（续）

- 数据仓库必须发挥安全堡垒作用以保护信息资产
  - 能够有效地控制对机构机密信息和个人隐私信息的访问
- 数据仓库必须在推进有效决策方面承担重要角色
- 数据仓库建设成功的前提是为业务群体所接受

本章内容

- 数据仓库的最终目标
- 数据仓库建模方法
- 维度建模的相关概念
- 维度建模的基本步骤

数据仓库系统的总体架构



逻辑数据模型设计的目标

- 简洁性
  - 用户对设计结果易于理解
  - 逻辑数据模型与用户的概念数据模型相匹配
  - 各类查询非常易于书写，很直观
- 表达能力强
  - 包含各类重要查询所需要的足够的信息
  - 包含所有相关数据
- 性能
  - 为高效的物理设计提供可能

Simplicity, Simplicity, Simplicity

Jeff Byard and Donovan Schneider, Red Brick Systems

数据仓库建设的方法

- 数据集市（Data Mart）：
  - 部门级数据仓库，跟数据仓库相比内容更少，但是更加集中
- 自顶向下的方法
  - 首先建立一个面向企业级别统一的数据仓库
  - 其次建立一些面向部门级别特殊子集的数据，建立数据集市
- 自底向上的方法
  - 首先考虑重要的问题，建立一个数据集市
  - 一个数据集市、另外一个数据集市，……
  - 数据仓库 = 所有数据集市的集合
- 在实际工程中，两者之间的差别并不是很大，通常会穿插进行

数据仓库建模方法

- 规范化建模
- 维度建模

本章内容

- 数据仓库的最终目标
- 数据仓库建模方法
- 维度建模的相关概念
- 维度建模的基本步骤

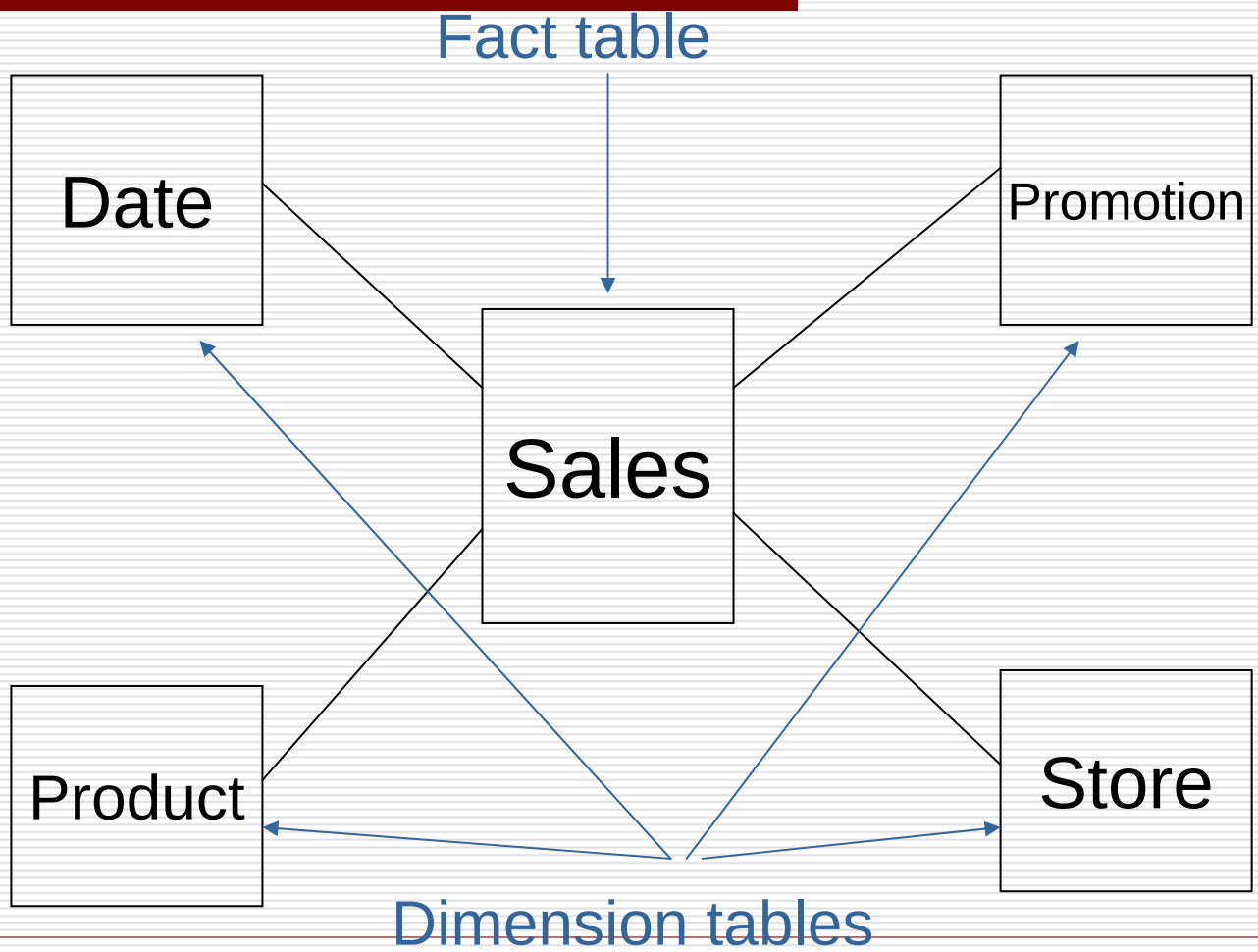
维模型

- 一种非规范化的关系模型
  - 由一组属性构成的表所组成
  - 表跟表之间的关系通过关键字和外键来定义
- 以良好的可理解性和方便的产生报表来进行数据组织，很少考虑修改的性能
- 通过SQL或者相关的工具实现数据的查询和维护

ER Model vs. Dimensional Models

	ER模型	维模型
数据组织	一张表代表一个实体	数据组织以事实表为核心
最求目标	最少的数据冗余	最大的可理解性
优化策略	面向Update操作进行优化	面向检索进行优化
面向系统	面向事务处理的模型	面向数据仓库的模型

星型模式


Fact table
Date
Promotion
Sales
Product
Store
Dimension tables

事实表

- 每一个事实表通常包含了处理所关心的一系列的度量值
- 每一个事实表的行包括
  - 具有可加性的数值型的度量值
    - 文本事实通常具有不可预见的内容，很难进行分析
  - 与维表相连接的外键
    - 通常具有两个和两个以上的外键
    - 外键之间表示维表之间多对多的关系

事实表（续）

- 事实表的特征
  - 非常的大
    - 包含几万、几十万甚至几百几千条的记录
  - 内容相对的窄
    - 列数较少
  - 经常发生变化
    - 现实世界中新事件的发生 → 事实表中增加一条记录
    - 典型情况下，仅仅是数据的追加
- 事实表的使用
  - 各类度量值的聚集计算

维表

- 每一张维表对应现实世界中的一个对象或者概念
  - 例如：客户、产品、日期、地区、商场
- 维表的特征
  - 包含了众多描述性的列
    - 维表的范围很宽（具有多个属性）
  - 通常情况下，跟事实表相比，行数相对较小
    - 通常< 10万条
  - 内容相对固定
    - 几乎就是一类查找表

维表（续）

- 维表的应用
  - 基于维属性的过滤（切片、切块等）
  - 基于维属性的各种聚集操作
  - 报表中各类标签的主要来源
  - 事实表通过维表进行引用

事实表与维表的比较

Facts

- 属性个数少（窄）
- 记录行数多（大）
- 数值型度量
- 随着时间的推移，数据增长

Dimensions

- 属性个数多（宽）
- 记录个数少（小）
- 描述型属性
- 静态的，很少发生变化

Facts contain numbers, dimensions contain labels

维模型的优点 (according to Kimball)

- 可预见的、标准化的架构
- 针对用户的需求发生变化，具有良好的适应性，能够自然地进行扩展
- 维度模型的简明性，也为提高性能带来好处
- 容易支持增量数据加载
- 标准的设计方法已经趋于成熟
- 现已有不少的产品支持维度建模

维度模型和规范化模型

- 两者之间存在着一种自然的关系
  - 单个规范化ER图通常可以分解为多个维度方案
- 规范化模型比较复杂，原因在于它将许多从来就不会出现在单个时间点和单个数据集中的多个业务处理放在了单张绘制图中
- 规范化ER图转换到维模型
  - 将ER图中分成若干分散的业务处理过程，然后分别单独建模
  - 选出ER图中含有数字型与可加性非关键字事实的多对多关系，并标记为事实表
  - 将剩下的所有表复合成具有直接连接到事实表的单连关键字的平面表，标记为维表

设计问题

关系模型和多维模型

- 非规范化的、带有索引结构的关系模型具有良好的灵活性
- 利用多维数据模型具有简单、高效的特点

本章内容

- 数据仓库的最终目标
- 数据仓库建模方法
- 维度建模的相关概念
- 维度建模的基本步骤

维度建模四个步骤

- 定义需要建模的业务处理过程
- 定义业务处理所涉及事实的粒度
- 选取用于每个事实表的维度
- 确定用于分析形成每个事实表的数字型事实

零售实例: 连锁超市的销售

- **POS = Point of sale**
  - 数据收集通过对条形码的扫描得到
- 在5个省份范围内的100多家连锁超市
- 大约有60,000种产品放在货架上，SKUs
  - **SKU = stock keeping unit** 库存储藏单位
  - SKU用以表示不同的产品
  - 一些产品来自于外部的生产厂商，并在包装上印有条形码
  - 有些没有(for example, produce, bakery, meat, floral)
- 目标：价格的变动和各项营销活动对产品销售和利润的影响
  - 营销活动 = 临时降价、各类广告、超市的布置

选取业务处理

- 基本原则
  - 建立的第一个维度模型应该是一个最有影响的模型——它应该对最为紧迫的业务问题做出回答，并且对数据的抽取来说，比较容易
- 对于零售实例研究中：
  - 管理方面要做的事情：更好的理解POS系统记录的顾客购买行为
  - 建模所需要提供的业务处理：POS零售业务

零售业务的问题

- What is the lift due to a promotion?
  - Lift = gain in sales in a product because it's being promoted
  - Requires estimated baseline sales value
    - Could be calculated based on historical sales figures
- Detect time shifting
  - Customers stock up on the product that's on sale
  - Then they don't buy more of it for a long time
- Detect cannibalization（调拨情况）
  - Customers buy the promoted product instead of competing products
  - Promoting Brand A reduces sales of Brand B
- Detect cross-sell of complementary products
  - Promoting charcoal increases sales of lighter fluid
  - Promoting hamburger meat increases sales of hamburger buns
- What is the profitability of a promotion?
  - Considering promotional costs, discounts, lift, time shifting, cannibalization, and cross-sell

定义事实表的粒度

- 事实表的粒度 = 事实表每一行的具体含义
- 应优先考虑为业务处理获取最有原子性的信息而开发维度模型。原子数据是所收集的最为详细的信息，该数据不能在做进一步的细分
- 维度模型的细节性数据是安如泰山的，并随时准备接受业务用户各种分析的特殊攻击
- 确定数据仓库最为详细数据的最大层次数目

定义事实表的粒度（续）

- 给业务处理定义较高层次的粒度，这种粒度表示最具有原子性的数据的聚集
  - 选取较高层面的粒度，就意味着将自己限制到更少或者细节性可能更小的维度上
- 好的粒度定义的事实数据表
  - 更好的表达数据的意义: 数据量可能增大
  - 可能存在更多的记录行
- 性能和丰富表达能力之间的权衡
  - 聚集预计算可以大大提高性能
  - 聚集数据作为调整性能的一种手段起着非常重要的作用，但它绝对不能作为用户存取最低层面的细节内容的替代品

选取相关维度

- 根据不同的粒度定义，确定了事实表的基本维度特性，确定候选关键字
  - 例子1：学生选课注册
    - **(Course, Student, Term)是候选关键字**
  - 例子2：超市中的一笔交易
    - **(Transaction ID, Product SKU)是候选关键字**
- 增加相关的维度，这些附加维度在基本维度的每个组合值方面自然的取得唯一的值
  - **例子1: Instructor and Classroom**
  - **例子2: Store, Date, and Promotion**

时间维（日期维）

- 几乎所有的数据仓库中均包含时间维
  - 数据仓库是反映历史变化的
  - 允许针对历史的数据进行分析
- 典型的粒度：**each row = 1 day**
- 日期维的属性
  - 是否节假日标记、是否周末标记、 星期几
  - 销售旺季（情人节、圣诞节、春节等）
  - 财政日期（财政年度、财政季度）
  - 一年中的天数：便于各种计算

时间维（日期维）

- 为什么需要一个明确的日期维
  - 关系数据库不能处理日期维度表的高效率连接
  - 在性能方面，绝大多数数据库不能对SQL日期运算进行索引，因此在SQL运算字段进行约束的查询不会用到索引
  - 从可用性来讲，典型的业务用户并不精通SQL日期函数的语法
  - SQL日期函数不能支持诸如周末、节假日、财政日期或者重大事件与平日这样的属性而进行的过滤操作
  - 通常对于日期的关键字指定为整数类型，而不是任何形式的日期数据

日期维与时间维

- 日期和时间通常是完全不相关的
- 不建议将两者进行合并，否则维表会变得相当的大

Surrogate Keys

- **Primary keys of dimension tables should be surrogate keys, not natural keys**
  - **Natural key: A key that is meaningful to users**
  - **Surrogate key: A meaningless integer key that is assigned by the data warehouse**
  - **Keys or codes generated by operational systems = natural keys (avoid using these as keys!)**
    - **E.g. Account number, UPC code, Social Security Number**
  - **Syntactic vs. semantic**

Benefits of Surrogate Keys

- 能够对数据仓库环境的操作型变化进行缓冲
  - 对多个数据源的数据更加容易集成
- 可以获得性能上的优势
  - **Narrow dimension keys → Thinner fact table → Better performance**
  - **This can actually make a big performance difference.**
- 便于处理例外的情况
  - **For example, what if the value is unknown or TBD?**
  - **Using NULL is a poor option**
    - **Three-valued logic is not intuitive to users**
    - **They will get their queries wrong**
    - **Join performance will suffer**
  - **Better: Explicit dimension rows for "Unknown", "TBD", "N/A", etc.**

Benefits of Surrogate Keys

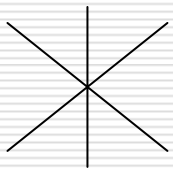
- 避免一些带有隐含语义信息的查询
  - **Example: WHERE date_key < '01/01/2004'**
  - **Will facts with unknown date be included?**
- 更好的支持处理维度表属性进行修改的一项基本技术

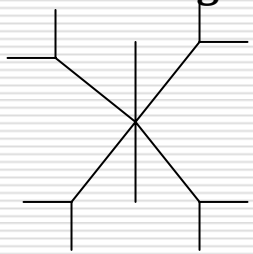
More Dimension Tables

- **Product**
  - **Merchandise hierarchy**
    - **SKU → Brand → Category → Department**
  - **Other attributes**
    - **Product name, Size, Weight, Package Type, etc.**
- **Store**
  - **Geography hierarchy**
    - **Store → ZIP Code → County → State**
  - **Administrative hierarchy**
    - **Store → District → Region**
  - **Other attributes**
    - **Address, Store name, Store Manager, Square Footage, etc.**
- **Hierarchies**
  - **Common in dimension tables**
  - **Multiple hierarchies can appear in the same dimension**
  - **Don't need to be strict hierarchies**
    - **e.g. ZIP code that spans 2 counties**

Snowflake Schema

- Dimension tables are not in normal form
  - Redundant information about hierarchies
- Normalizing dimension tables leads to snowflake schema
  - Avoid redundancy → some storage savings





- Snowflaking not recommended in most cases
  - More tables = more complex design
  - More tables → more joins → slower queries
  - Space consumed by dimensions is small compared to facts
  - Exception: Really big dimension tables
    - In some warehouses, customer dimension is really large
    - We'll return to this next week.

Degenerate Dimensions

- Occasionally a dimension is merely an identifier, without any interesting attributes
  - (Transaction ID, Product) was our candidate key
  - But "Transaction ID" is just a unique identifier
  - Serves to group together products that were bought in the same shopping cart
- Two options:
  - Discard the dimension
    - Fact table will lack primary key, but that's OK
    - A good option if the dimension isn't needed for analysis
  - Use a "degenerate dimension"
    - Store the dimension identifier directly in the fact table
    - Don't create a separate dimension table
    - Used for transaction ID, invoice number, etc.

How many dimensions?

- Should two concepts be modeled as separate dimensions or two aspects of the same dimension?
- Example: Different types of promotions
  - Ads, discounts, coupons, end-of-aisle displays
  - Option A: 4 dimensions
    - Separate dimension for each type of promotion
  - Option B: 1 dimension
    - Each dimension row captures a combination of ad, discount, coupon, and end-of-aisle display
- Factors to consider
  - How do the users think about the data?
    - Are an ad and a coupon separate promotions or two aspects of the same promotion?
  - Fewer tables = good
    - Generally fewer tables = simpler design
  - Performance implications

How many dimensions? Performance Implications

- **Most OLAP queries are "I/O bound"**
  - **Data-intensive not compute-intensive**
  - **Reading the data from disk is the bottleneck**
  - **For "typical" queries, on "typical" hardware**
- **Size of data on disk ≈ query performance**
  - **Keeping storage requirements small is important**
- **Dimensional modeling impacts storage requirements**

Performance Implications

- Let's consider the extremes
- Assumptions:
  - 100 million fact rows
  - 3 four-byte measurement columns in the fact table
  - 100 dimensional attributes, average size = 20 bytes
- Three modeling options
  - One "Everything" dimension
  - Each attribute gets its own dimension table
  - 5 dimensions (Date, Product, Store, Promotion, Transaction ID)

Option A

- **Option A: One "Everything" dimension**
  - **Fact table is very thin (16 bytes per row)**
    - **3 four-byte fact columns**
    - **1 four-byte foreign key to the Everything dimension**
  - **Dimension table is very wide (2000 bytes per row)**
    - **100 attributes * 20 bytes each**
  - **Dimension table has as many rows as fact table!**
    - **Each fact row has a different combination of attributes**
  - **Total space = 1.6 GB fact + 200 TB dimension**
    - **16 bytes * 100 million rows = 1.6 GB**
    - **2000 bytes * 100 million rows = 200 TB**

Option B

- Option B: Each attribute gets its own dimension table
  - Store Manager First Name dimension, Store Manager Last Name dimension, etc.
  - Fact table is wide (212 bytes per row)
    - Assume 2-byte keys for all dimension tables
    - This is a generous assumption
  - Dimension tables are very thin, have few rows
  - Space for fact table = 21.2 GB
  - Space for dimension tables = negligible
    - < 132 MB total for all dimensions
    - No dimension table has more than 60,000 rows
    - Each dimension row is 22 bytes
    - 100 dimension tables

Option C

- **Option C: Four dimensions (Date, Product, Store, Promotion)**
  - **Fact table is quite thin (28 bytes)**
    - **2-byte keys for Date and Store**
    - **4-byte keys for Product, Promotion, Transaction ID**
    - **3 4-byte fact columns**
  - **Dimension tables are wide, have few rows**
    - **No dimension table has more than 60,000 rows**
  - **Space for fact table = 2.8 GB**
    - **28 bytes * 100 million rows**
  - **Space for dimension tables = negligible**
    - **< 130 MB for all dimensions**

Why is Option C the best?

- Attributes that pertain to the same logical object have a high degree of correlation
  - Correlated attributes
    - (product name, brand)
    - Number of distinct combinations = number of distinct products
    - Product name and brand are completely correlated
  - Uncorrelated attributes
    - (product name, date)
    - Number of distinct combinations = number of products * number of dates
    - No correlation between date and product
    - Most possible combinations of values will appear in the fact table
  - Combining non-correlated attributes in the same dimension leads to blow-up in size of dimension table
- When attributes are semi-correlated, designer has a choice
  - Frequently, multiple types of promotion occur together
  - E.g. product being promoted has ad, coupon, and in-store display
  - Number of (ad, coupon, discount, display) combinations is small
  - Combining them in a single Promotion dimension is reasonable

Additivity

- **Additive facts are easy to work with**
  - **Summing the fact value gives meaningful results**
  - **Additive facts:**
    - **Quantity sold**
    - **Total dollar sales**
  - **Non-additive facts:**
    - **Averages (average sales price, unit price)**
    - **Percentages (% discount)**
    - **Ratios (gross margin)**
    - **Count of distinct products sold**

Month	Quantity Sold
June	12
July	10
August	14
OVERALL	36

Month	Avg. Sales Price
June	$35
July	$28
August	$30
OVERALL	$93 ← Wrong!

Handling Non-Additive Facts

- Taxonomy of aggregation functions
  - From Data Cube paper by Jim Gray et al.
  - How hard is it to compute the aggregate function from sub-aggregates?
  - Three classes of aggregates:
    - Distributive
      - Compute aggregate directly from sub-aggregates
      - Examples: COUNT, SUM, MAX, MIN
    - Algebraic
      - Compute aggregate from constant-sized summary of subgroup
      - Examples: AVERAGE, STDDEV
      - For AVERAGE, summary data for each group is SUM, COUNT
    - Holistic
      - Require unbounded amount of information about each subgroup
      - Examples: COUNT DISTINCT, MEDIAN
      - Usually impractical in data warehouses!

Additivity and the Fact Table

- Store additive quantities in the fact table
- Example:
  - Don't store "unit price"
  - Store "quantity sold" and "total price" instead
- Additive summaries used for distributive aggregates are OK
  - Numerator and denominator for averages, percentages, ratios
- Big disadvantage of non-additive quantities:
  - Cannot pre-compute aggregates!

Transactional vs. Snapshot Facts

- **Transactional**
  - **Each fact row represents a discrete event**
  - **Provides the most granular, detailed information**
- **Snapshot**
  - **Each fact row represents a point-in-time snapshot**
  - **Snapshots are taken at predefined time intervals**
    - **Examples: Hourly, daily, or weekly snapshots**
  - **Provides a cumulative view**
  - **Used for continuous processes / measures of intensity**
  - **Examples:**
    - **Account balance**
    - **Inventory level**
    - **Room temperature**

Transactional vs. Snapshot Facts

Transactional

Brian	Oct. 1	CREDIT	+40
Rajeev	Oct. 1	CREDIT	+10
Brian	Oct. 3	DEBIT	-10
Rajeev	Oct. 3	CREDIT	+20
Brian	Oct. 4	DEBIT	-5
Brian	Oct. 4	CREDIT	+15
Rajeev	Oct. 4	CREDIT	+50
Brian	Oct. 5	DEBIT	-20
Rajeev	Oct. 5	DEBIT	-10
Rajeev	Oct. 5	DEBIT	-15

Snapshot

Brian	Oct. 1	40
Rajeev	Oct. 1	10
Brian	Oct. 2	40
Rajeev	Oct. 2	10
Brian	Oct. 3	30
Rajeev	Oct. 3	30
Brian	Oct. 4	40
Rajeev	Oct. 4	80
Brian	Oct. 5	40
Rajeev	Oct. 5	55

Transactional vs. Snapshot Facts

- Two complementary organizations
- Information content is similar
  - Snapshot view can be always derived from transactional fact
- Why use snapshot facts?
  - Sampling is the only option for continuous processes
    - E.g. sensor readings
  - Data compression
    - Recording all transactional activity may be too much data!
    - Stock price at each trade vs. opening / closing price
  - Query expressiveness
    - Some queries are much easier to ask/answer with snapshot fact
    - Example: Average daily balance

A Difficult SQL Exercise

How to generate snapshot fact from transactional fact?

Brian	Oct. 1	CREDIT	+40		Oct. 1	40
Rajeev	Oct. 1	CREDIT	+		Oct. 1	10
Brian	Oct. 3	DEBIT		B	Oct. 2	40
Rajeev	Oct. 3	CREDIT	+20		Oct. 2	10
Brian	Oct. 4	DEBIT	-5		Oct. 3	30
Brian	Oct. 4	CREDIT	+15	ajeev	Oct. 3	30
Rajeev	Oct. 4	CREDIT	+50	Brian	Oct. 4	40
Brian	Oct. 5	DEBIT	-20	ajeev	Oct. 4	80
Rajeev	Oct. 5	DEBIT	-10	rian	Oct. 5	40
Rajeev	Oct. 5	DEBIT	-15	Rajeev	Oct. 5	55

Semi-Additive Facts

- Snapshot facts are semi-additive
- Additive across non-date dimensions
- Not additive across date dimension
- Example:
  - Total account balance on Oct 1 = OK
  - Total account balance for Brian = NOT OK
- Time averages
  - Example: Average daily balance
  - Can be computed from snapshot fact
    - First compute sum across all time periods
    - Then divide by the number of time periods
    - Can't just use the SQL AVG() operator

关于事实表

- 事务型数据库中不包含没有发生的任何事件
  - 例如：如果产品没有销售出去，就不会包含任何关于该产品的销售信息
- 优点与缺点
  - 优点：利用数据的“稀疏”
    - 数据的存储量减少
  - 缺点：不包含任何没有发生事件的相关实体
    - 例如：哪些产品在促销活动中没有销售出去？

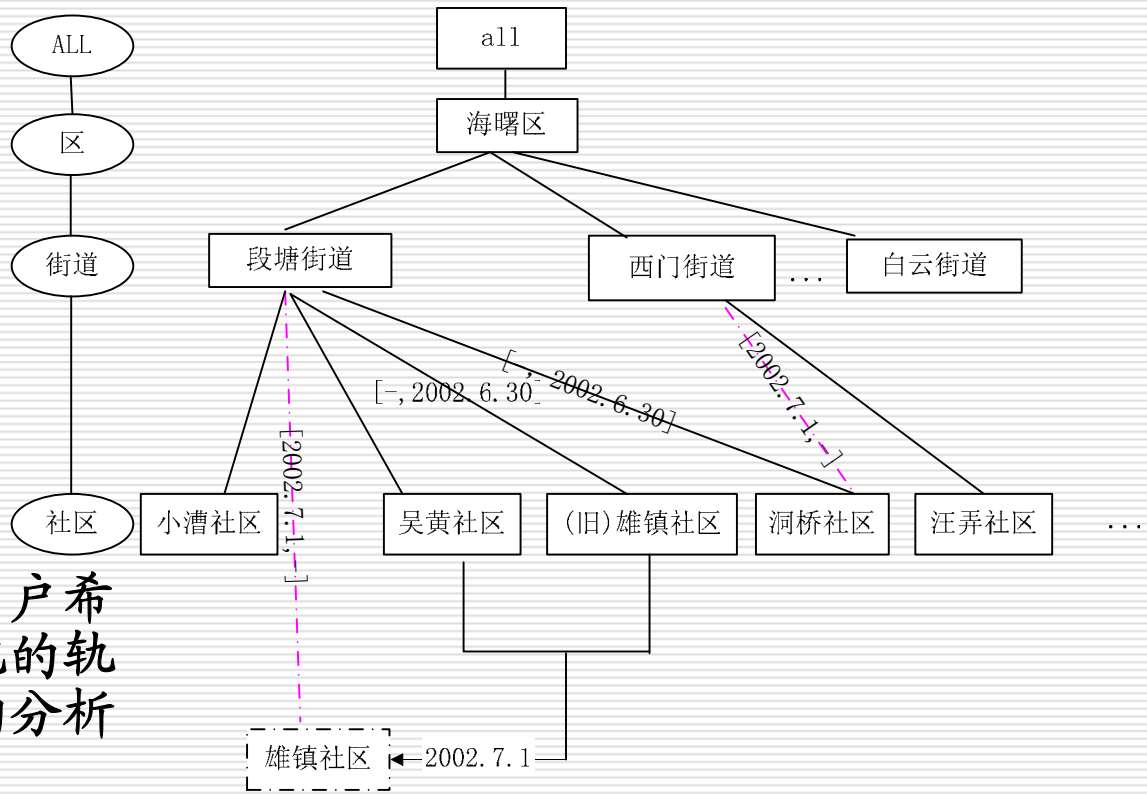
非事实型事实表

- 事件通常作为含有一系列关键字的事实表进行建模，其中每个关键字表示参与事件的一个维度。事件事实表通常不具有相联系的显式数字型事实，并因此被称为非事实型事实表
- 事务型事实表中不包含任何的度量事实
  - 一个事实表中不包含任何的数字型的度量
  - 仅仅表示维度之间的关系
  - 包含一个虚构的事实度量，其值恒为1
- 例如：
  - 学生选课：学期、学号、课程关键字、学分、（注册数）

渐变维度（Slowly Changing Dimensions）

- 先前的所有讨论，基于以下假设:
  - 每个维度在逻辑上独立于所有其他维度（正交关系）
  - 维度被看成与时间无关
- 实际情况:
  - 跟事实表相比，维表的内容相对稳定
    - 新的事务或者交易不断产生
    - 新产品的加入却相对较少
    - 新商场的开张更少
  - 有些维度随着尽管变化相当缓慢，但维度属性可能随着时间发生变化
    - 客户地址发生变化
    - 商场根据地域进行分组，但是由于企业重组，地域的划分也随之改变

渐变维度举例



- 维发生变化，用户希望能够看到变化的轨迹，并且在查询分析中加以体现

渐变维度举例

- 关于行政区划动态变化的例子
  - 2002年7月1日，“洞桥社区”从“段塘街道”划归“西门街道”
  - “吴黄社区”和“旧雄镇社区”合并，➔“雄镇社区”

现在用户希望察看2002年第二季度和第三季度段塘街道所有社区的人口出生情况

段塘街道	出生人数
合计	20
小漕社区	2
洞桥社区	2
吴黄社区	3
（旧）雄镇社区	1

段塘街道	出生人数
合计	15
小漕社区	2
雄镇社区	3
…	

2002年2季度人口出生情况　　2002年3季度人口出生情况

渐变维度举例

- 事实上，这种变化是很常见的，例如：
  - 1997年，重庆从四川省中分离出来，成为一个独立的直辖市
  - 高校的合并重组
- 渐变维度：
  - 允许维的成员甚至维结构随时间发生一定的变化，相对于事实表而言，这种变化是缓慢的。这种变化主要体现在：
    - 维层次结构上：增加、删除维层次
    - 维成员上：增加、删除、修改、修改父指向、合并、分裂

处理渐变维度的方法

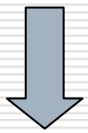
- 必须为维度表的每个属性指定处理变化的策略。也就说，如果属性值在操作型数据源中发生变化，则维度模型该如何应对？
- 具体方法
  - 改写属性值
  - 添加维度行
  - 添加维度列

改写维度值

- 最为简单的方法和选项：修改维表中的记录
  - 用户需要用当前值取代维度行的旧属性值
- 例如：
  - 产品维中，产品包装大小，由于操作人员的失误，将“80 × 80”误写为“8 × 8”
  - 错误在业务系统中被发现
  - 相应的错误必须在数据仓库中予以修改
    - 修改维表
    - 修改预先综合的数据

改写维度值（续）

产品关键字	产品描述	部门	SKU编号（自然关键字）
12345	Microsoft Windows 2000	教育	ABC922-Z



产品关键字	产品描述	部门	SKU编号（自然关键字）
12345	Microsoft Windows 2000	策略	ABC922-Z

- 优点：快速和方便。适合于对于属性值的更正，或者是对于缺失值的处理
- 缺点：所有属性变化的历史数据都会丢失

改写属性值存在的问题

- 无法对旧属性值的任何历史数据进行维护
  - 张三1993年生活在上海
  - 张三1994年搬到北京
  - 假设修改客户信息维
  - 查询：1993年上海客户的销售量是多少？
  - 对于张三1993年的消费情况被忽略了

添加维度行

- 精确的历史信息对于数据仓库来说非常重要
- 如何来保存相应的一些历史变化情况呢？
- 解决方法：添加一行新记录
  - 旧的事实信息指向旧的维度行
  - 新的事实信息指向新的维度行

Cust_key	Name	Sex	State	YOB
457	张三	Male	WI	1976
…	…	…	…	…
784	张三	Male	CA	1976

Old dimension row

New dimension row

添加维度行（续）

Customer Dimension

Cust_key	Name	Sex	State	YOB
457	**张三**	**Male**	**WI**	**1976**
…	…	…	…	…
784	**张三**	**Male**	**CA**	**1976**

Existing fact rows use old dimension row

New fact rows use new dimension row

Sales Fact

Cust_key	…	Quantity
…	…	…
457	…	**5**
…	…	…
784	…	**4**

添加维度行（续）

产品关键字	产品描述	部门	SKU编号（自然关键字）
12345	Microsoft Windows 2000	教育	ABC922-Z
24578	Microsoft Windows 2000	策略	ABC922-Z

- 注意代理关键字的优势。自然关键字字段成为一种维系单一产品的要纽带
- 能够准确跟踪渐变维度属性的主要方法。这种方法由于新维度行能自动区分事实表的历史而显得特别有效
- 如果能够包括一个生效日期（或者失效日期）标志，则更为自然。加日期标志时，为了得到正确的答案，没有必要在维度表生效日期施加约束，但是在ETL过程中是有效的
- 不要将生效日期作为另外的主关键字
- 维度的生效日期或者截至日期标记对于比较高级分析可能是有用的
- 添加维度行的方法是支持进行准确历史属性分析的主导技术
- 不用修改聚集表

添加维度行（续）

- 缺点
  - 由于生成新的行，加快了维度表的膨胀
  - 如果已经超过百万行的维度表来说是不合适的
  - 添加维度行可以区分历史，但不能将新属性值同旧事实历史联系起来

添加维度列

产品关键字	产品描述	部门	前部门	SKU编号（自然关键字）
12345	Microsoft Windows 2000	策略	教育	ABC922-Z

- 适合于同时强烈需要为两个方面的视图提供支持的应用情形
- 虽然已经出现变化，但是在逻辑上仍然可以按照没有变化的情形进行处理
- 与添加维度行的区别在于，当前与以前的描述内容在同一时间都可以看成是成立的
- 在实际中用的相对较少。不适宜跟踪大量的中间属性值带来的影响

Type 1 vs. Type 2的选择

- 两种方法经常交叉使用
  - 对部分属性保持其历史信息
  - 对部分属性直接修改属性值
- 选择的策略:
  - 当查询需要应用源属性或者新属性直接查询时，需要保存历史记录
  - 对于那些附加的维，例如电话号码等，通常不采用保存历史，主要原因在于很少出现在报表列，同时也很少涉及分析

混合渐变维度处理方法

- 假设我们同时需要知道旧属性值与新属性值交叉的情况
  - 对于公司机构重组的情况非常需要!
  - 例如：销售区域每年改变一次
- 两种情况
  - 可预见的多重变化
  - 不可预见的单重变化

可预见的多重情况

- 例如，5年的时间内，营销机构进行5次重组
  - 按当年的区域划分报告每年的销售情况
  - 按任选不同的一年的区域划分报告每年的销售情况
  - 按任选一年的单个区域划分报告跨任意年度的销售情况

营销代表维度
营销代表关键字
营销代表姓名
营销代表地址
当前地区
2004年地区
2003年地区
2002年地区
2001年地区
……

不可预见的单重变化

- 适合于在根据当前值对历史数据的报告提供支持的同时，围绕不可预见的属性变化准确地保留历史内容的应用要求
- 任何一种标准的渐变维度处理方法不足以独立应付这种应用需求

产品关键字	产品描述	部门	前部门	SKU编号（自然关键字）
12345	Microsoft Windows 2000	教育	教育	ABC922-Z

产品关键字	产品描述	部门	前部门	SKU编号（自然关键字）
12345	Microsoft Windows 2000	策略	教育	ABC922-Z
12345	Microsoft Windows 2000	策略	策略	ABC922-Z

产品关键字	产品描述	部门	前部门	SKU编号（自然关键字）
12345	Microsoft Windows 2000	电子	教育	ABC922-Z
12345	Microsoft Windows 2000	电子	策略	ABC922-Z
12345	Microsoft Windows 2000	电子	电子	ABC922-Z

快变维度

- 如果一个维度属性每月都在变化，怎么办？
- 将这些快速变化的属性分裂成一个或者多个单独的维度，然后在事实表中放置两个外关键字，一个用于主表，另外一个用于快变的维表
- 核心：根据数据的稳定情况来判断
- 客户维度表
  - 客户维表：客户关键字，客户ID，客户姓名、客户生日，…
  - 客户人口统计特征：客户人口统计特征关键字、客户年龄段、客户收入分段、客户婚姻状况

国际性问题

- 国际化的组织通常会碰到不同地域使用的一些数据规范和标准
  - 某些交易是按照美元进行的，有些地区是人民币，有些是欧元等
- 不同地域的报表要求可能不一样
  - 标准货币与本地货币
  - 历史汇率与当前汇率
- 不同地域的时间表示形式不同
  - 有些时候使用本地时间具有现实意义
    - 例如发现不同时间段的销售模式
  - 有些时候使用标准时间比较合理（例如：GMT）
    - 很好的表示事件之间的关系

处理多种货币

- 在事实表增加不同货币形式的字段
  - 例如美元、欧元、人民币等
  - 由于货币在理论上是不受限制的，不合理
- 每一个事实表中包含两种货币形式的事实
  - 一列应用本地货币来表示一笔交易
  - 另一列定义一种标准货币形式来表示
  - 货币维来指示本地货币列的单元
  - 历史汇率表来影响对应日期的货币之间的转换
- 创建一张专用的货币转换表
  - 存储当前的两种货币之间的转换系数
  - 用以支持以各种货币形式来展现报表

多种流通货币举例

Sales Fact

Product	Date	Currency	AmtLocal	AmtUSD
443	**87**	**1**	**400**	**400**
1287	**87**	**4**	**1250**	**1447**
34	**88**	**2**	**3500**	**380**

Currency Dimension

Key	Name	Abrv	Country
1	**US Dollar**	**USD**	**USA**
2	**Japanese Yen**	**JPY**	**Japan**
3	**RenminBI**	**RMB**	**China**
4	**Euro**	**EUR**	**Europe**

Conversion Table

Date	From	To	Factor
	1	**2**	**111.3**
	1	**3**	**8.73**
	1	**4**	**.814**
	2	**1**	**.0089**
	…	**…**	**…**

主—次（核心—详细）事实

- 考虑超市的小票
- 每一个小票中包含了一系列的数据项目
- 每一个数据项目表示购买了一种产品
- 度量的计算通常是分层次的
  - 每一个数据项目都有数量和价格
  - 每一个小票中包含了总价、打折
- 非常自然的涉及，不同粒度的两张事实表
  - 小票事实表仅包含一行
  - 一个数据项一行

小票和数据项目

Orders Fact

- **Dimensions**
  - **Date**
  - **Customer**
  - **OrderID (degenerate)**
- **Fact Columns**
  - **Tax**
  - **Discount**
  - **ShippingFee**
  - **TotalPrice**

Lineitem Fact

- **Dimensions**
  - **Date**
  - **Customer**
  - **Product**
  - **OrderID (degenerate)**
- **Fact Columns**
  - **Quantity**
  - **Price**

设计中存在的问题

- 难以按照产品给出收入报表
- 小票事实表中缺少产品维度
  - 增加产品维存在粒度的冲突
- 数据项目事实表中缺少收入信息
  - 打折信息、税收信息非常重要
  - 未能在数据项目中予以体现
- 解决方案：重新设计，变成详细级别的事实表
  - 在数据项目表中增加税收、打折信息

维度建模应该避免的常见疏误

- 过多地将心思放在技术和数据上，而不关注业务地要求和目标
- 未能实现或者再现像数据仓库业务主管所具备地看起来有影响、易访问而又合乎情理的管理功能梦想而耿耿于怀
- 盯住制定一个庞大的多年工程计划不放，而不是去追求一个更容易处理的可能也是更急迫的可以进行迭代开发的方案
- 将精力全部投入到构造规范化数据结构中去了，而在建造一个基于维度模型的可行的展示环节时，却已经用光了给定的投入
- 把注意力放在了后台的作业性能和容易开发上，而不是放在前台的查询性能与容易使用上

维度建模应该避免的常见疏误

- 把展示环节中假定为可查询的数据做得过于复杂
- 在孤立应用的基础上建立维度模型，而没有考虑采用共享的一致的维度将这些模型捆绑在一起的数据体系结构
- 仅仅将总结性数据加入到展示环节的维度中
- 把业务、业务需求与分析内容以及基本数据与支持技术等看成是静态的
- 忽视了数据仓库的成功直接系于用户的接受程度这样的认识

对维度模型的误解

- 维度模型与数据中心都只是应用于概要性的数据方面的
- 维度模型与数据中心是针对部门而不是针对企业的解决方案
- 维度模型与数据中心是不可升级的
- 维度模型与数据中心仅当存在可预见的使用模式时才适合
- 维度模型与数据中心是不能集成的，从而只能形成直通的解决方案

下一次……

联系信息:

E-mail: yhtong@db.pku.edu.cn

Telephone: 62757756 (O)

13701205200 (M)