

# 数 据 仓 库

(三)

# 数据仓库

1. 引言
2. 从数据库到数据仓库
3. 数据分析与数据仓库
4. 数据仓库的四大特色
5. 数据仓库的基本结构
6. 数据仓库的设计
7. 联机分析处理 (**OLAP**) ✓
8. 多维建模 (**Dimensional Modeling**)
9. 数据仓库的应用

## 7 联机分析处理（OLAP）

- 60年代，关系数据库之父E. F. Codd提出了关系模型，促进了联机事务处理(OLTP)的发展(数据以表格的形式而非文件方式存储)。
- 1993年，E. F. Codd提出了OLAP概念，认为OLTP已不能满足终端用户对数据库查询分析的需要，SQL对大型数据库进行的简单查询也不能满足终端用户分析的要求。用户的决策分析需要对关系数据库进行大量计算才能得到结果，而查询结果并不能满足决策者提出的需求。因此，Codd提出了多维数据库和多维分析的概念，即OLAP。
- OLAP是目前RDBMS不可缺少的功能，可以作为一个独立的OLAP服务器实现，也可以集成在RDBMS中。

# 7 联机分析处理（OLAP）

- **定义1：**针对特定问题的联机数据访问和分析。通过对信息(维数据)的多种可能的观察形式进行快速、稳定一致和交互性的存取，允许管理决策人员对数据进行深入观察。
- **定义2：**是使分析人员、管理人员或执行人员能够从多种角度对从原始数据中转化出来的、能够真正为用户所理解的、并真实反映企业维特性的信息进行快速、一致、交互地存取，从而获得对数据的更深入了解的一类软件技术。（OLAP委员会的定义）
- **OLAP的目标**是满足决策支持或多维环境特定的查询和报表需求，它的技术核心是“维”这个概念，因此OLAP也可以说是多维数据分析工具的集合。

# 7 联机分析处理（OLAP）

## ➤ 联机事务处理：OLTP

OLAP（On-line Analytical Processing，联机分析处理）是在基于数据仓库多维模型的基础上实现的面向分析的各类操作的集合。与传统的 OLTP（On-line Transaction Processing，联机事务处理）的区别：

	OLTP	OLAP
面向应用	日常交易处理	明细查询，分析决策
访问模式	简单小事务，操作少量数据	复杂聚合查询，可以过大量数据
数据	当前最新数据	历史数据
数据规模	GB	TB ~ PB
数据更新	实时更新	批量更新
数据组织	满足3NF	反范式，星型模型

## 7 联机分析处理（OLAP）

- **OLAP**分析属于验证驱动型发现：用户首先提出自己的假设，然后利用**OLAP**工具检索查询以验证或否定假设。
- 特性
  - 快速性：**用户对**OLAP**的快速反应能力有很高的要求。
  - 可分析性：****OLAP**系统应能处理与应用有关的任何逻辑分析和统计分析。
  - 多维性：**多维性是**OLAP**的关键属性。
  - 信息性：**不论数据量有多大，也不管数据存储在何处，**OLAP**系统应能及时获得信息，并且管理大容量信息。

# 7 联机分析处理（OLAP）

7.1 OLAP中的几个基本概念

7.2 OLAP的基本数据模型

7.3 OLAP中的数据构造方式

7.4 数据立方体（Data Cube）

## 7.1 OLAP中的几个基本概念

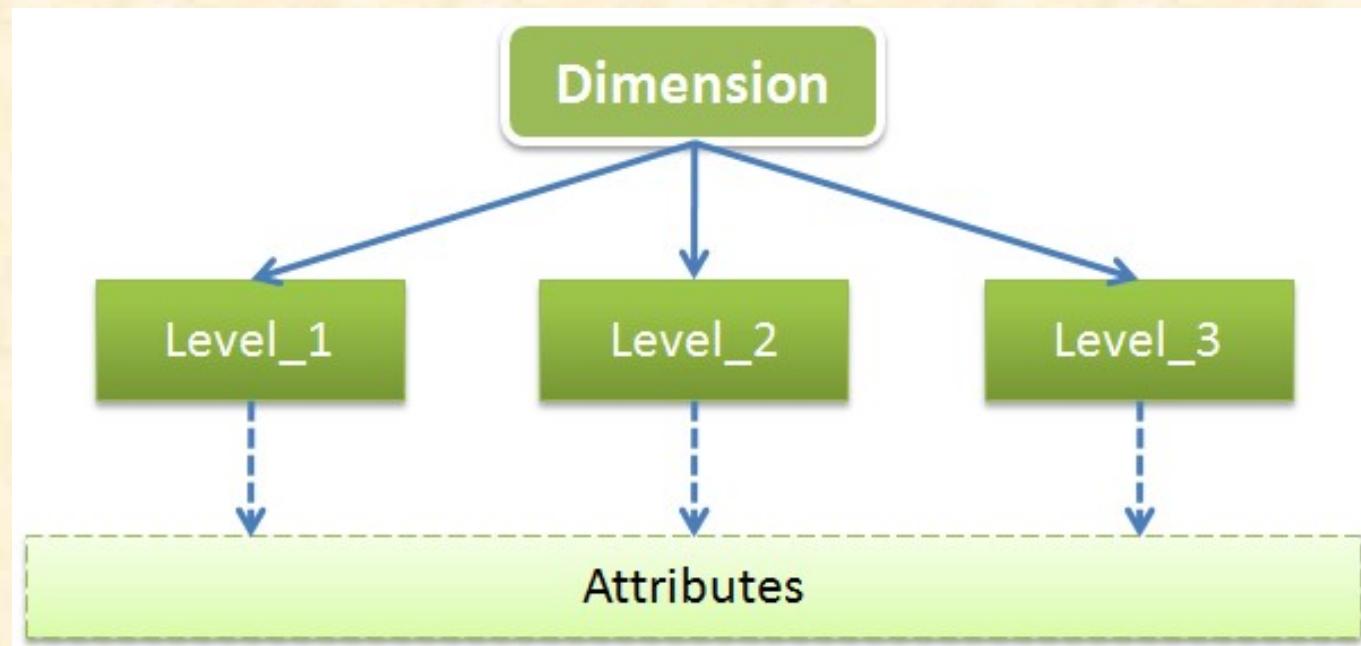
### □ 对象 (Object)

- 在分析型处理中，我们所关心和分析的对象
- 观察对象又被称为 ‘度量值’
  - 在多维数据集中，度量值是一组值，而且通常为数字值。
  - 度量值是所分析的多维数据集的中心值，即：度量值是最终用户浏览多维数据集时重点查看的数值型数据。
  - 度量值的选择取决于最终用户所请求的信息类型。
- 一些常见的度量值有：
  - » 销售金额 **sales**
  - » 成本金额 **cost**
  - » 库存数量 **production count**
  - » 消费金额 **expenditures**

## 7.1 OLAP中的几个基本概念

### 口维 (Dimension)

维是用于从不同角度描述事物特征的，一般维都会有多层（Level），每个 Level 都会包含一些共有的或特有的属性（Attribute），可以用下图来展示下维的结构和组成：



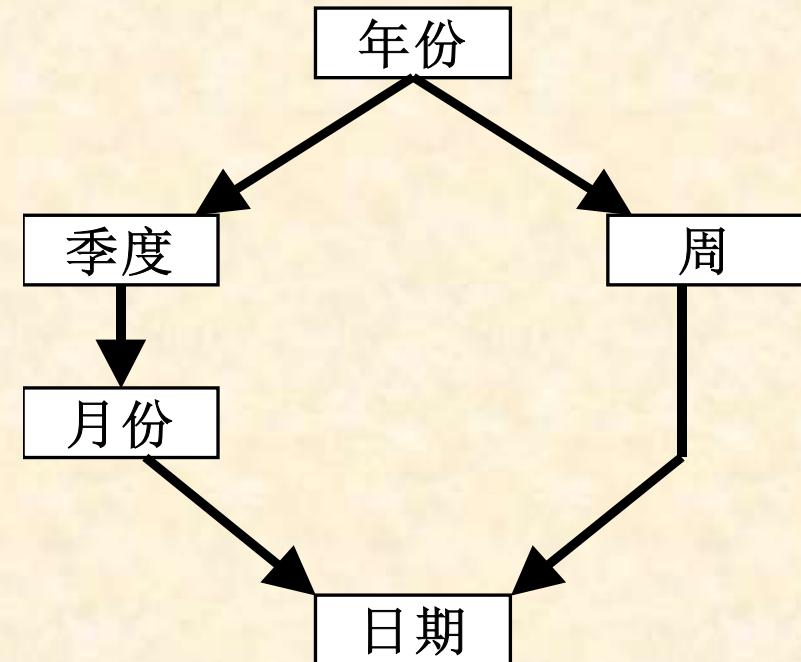
## 7.1 OLAP中的几个基本概念

如：从三个‘维’，观察‘销售金额’对象

➤ 时间维

- 日 — 月 — 季 — 年
- 日 — — 周 — — 年

可从时间角度统计（所有）商品在不同时间段内的销售（总）金额，以便于分析其与时间之间的关系

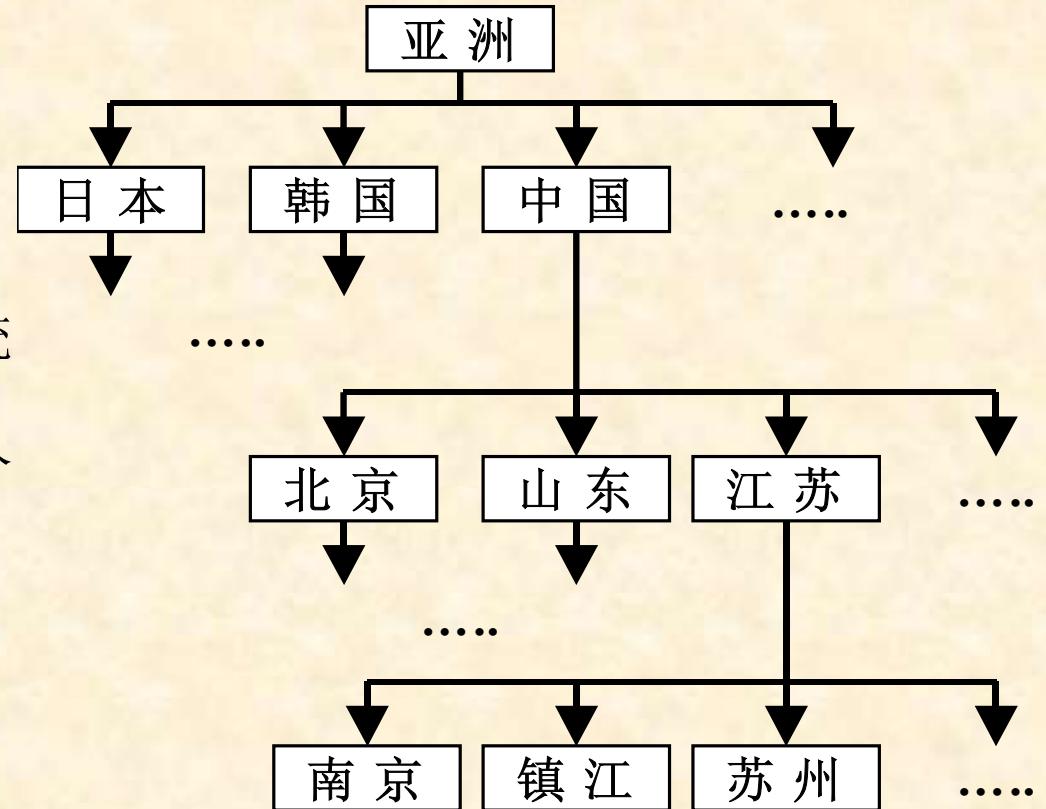


## 7.1 OLAP中的几个基本概念

### ➤ 地域维

▣ 市 — 省 — 国 — 洲

可根据每个连锁店所在的地域统计其销售（总）金额，以便于分析其与地域之间的关系



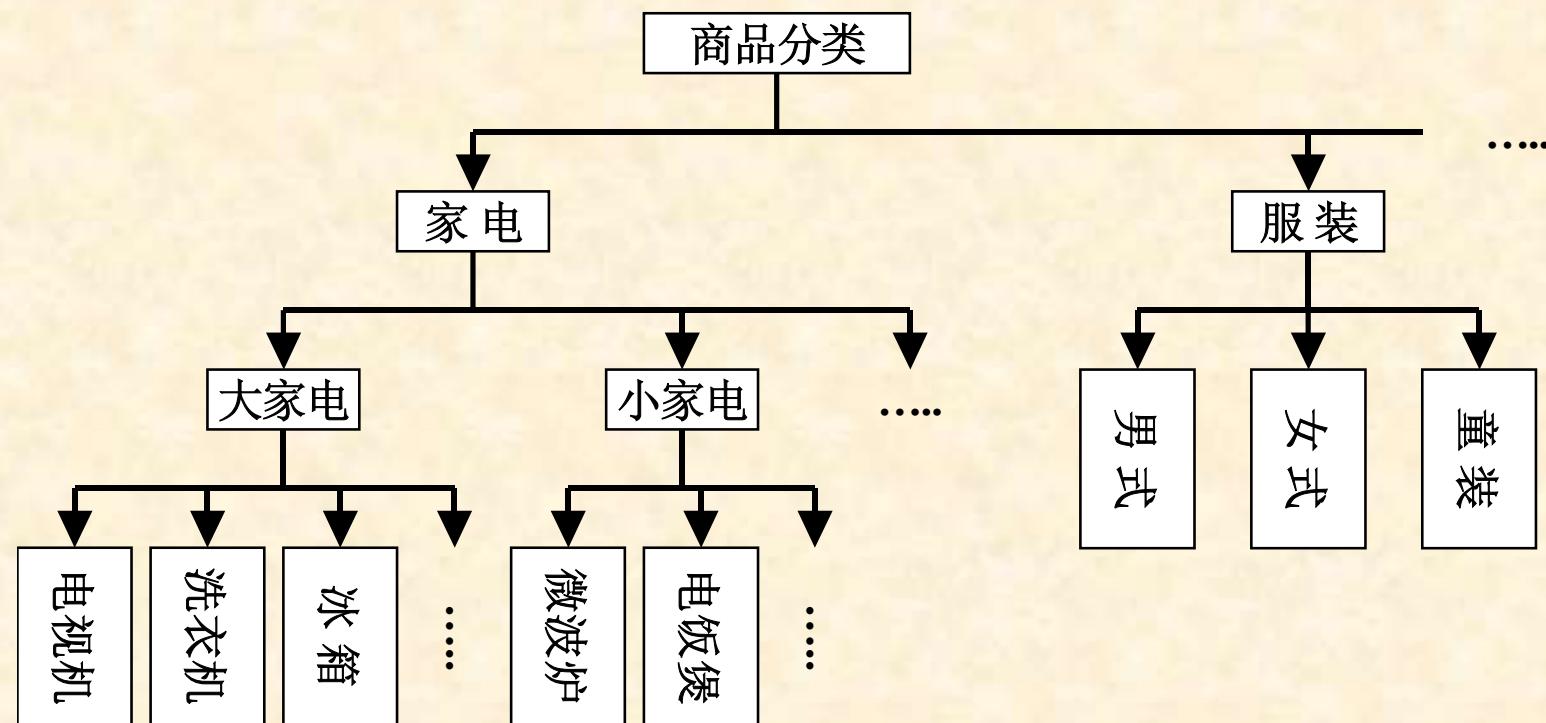
# 7.1 OLAP中的几个基本概念

## ➤ 商品维

按商品的类别可以分为：

家电(...), 小家电(...), 服装(男式, 女式, 童装)

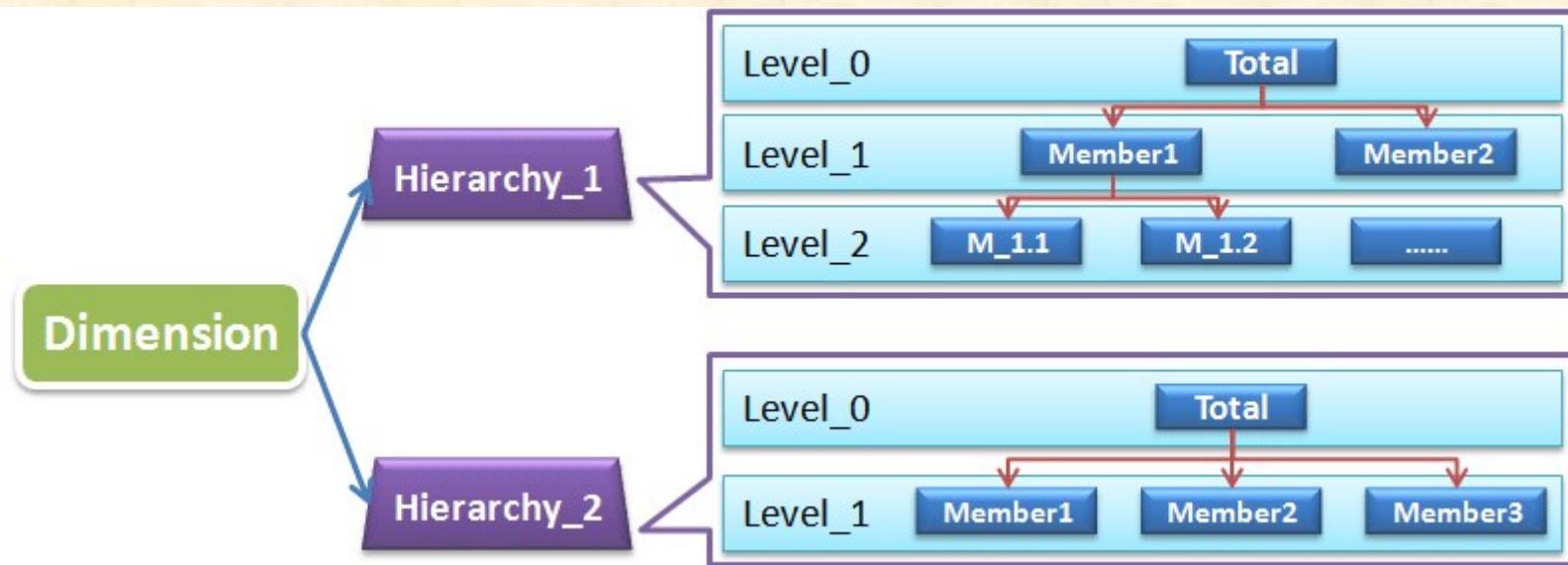
根据商品的分类情况统计每一类商品的销售金额，以便于分析其与商品类型之间的关系



## 7.1 OLAP中的几个基本概念

### □ 层 (Hierarchy)

在分析型应用中，在不同的深度层面上进行分析与观察，可能得到不同的分析结果。因此，‘层’反映了对分析对象的观察深度(时间维：日期、月份、季度、年)。



‘层’是与‘维’相关联的。在一个‘维’中可允许存在若干个‘层’，并且可以采用多种不同的‘层’次划分方法。

## 7.1 OLAP中的几个基本概念

- 例如：‘商品维’也可以按如下的方法进行层次划分
  - 按商品的价格分为
    - 高档，中档，低档
  - 按商品的供应商分为
    - 外资，合资，国营，私营，个体
  - 按购买商品的顾客信息分为
    - 按照年龄层次来划分：老年，中年，青年，少年儿童，婴儿
    - 按照所从事的职业来划分： .....
    - .....

## 7.1 OLAP中的几个基本概念

### □ 维成员

维的一个取值称为该维的一个‘维成员’

- 如果一个维是多层次的，则该维的‘维成员’可以是
  - 1. 在不同维层次上的取值的组合
  - 2. 在某个维层次上的取值
- 对一个数据项（分析‘对象’）来说，维成员是该数据项在某维中位置的描述。

## 7.1 OLAP中的几个基本概念

□ 例如：

### 1. 在不同维层次上的取值的组合

对具有日，月，年三个层次的‘时间维’来说，‘某年某月某日’、‘某年某月’、‘某月某日’、‘某年’都是其维成员，如：**2017年，1月，2017年1月，2017年1月1日，1月1日**

### 2. 在某个维层次上的取值

‘**地域**’维中的‘江苏’，‘南京’，.....

‘**商品**’维中的‘电视机’，‘服装’，.....

## 7.1 OLAP中的几个基本概念

### □多维数组

- 一个多维数组可以表示为  
**(维1, 维2, ..... , 维n, 变量)**
- 其中：
  - 变量**表示我们所观察的数据对象
  - 维1、维2、.....、维n分别表示我们观察该数据对象的角度（维）

## 7.1 OLAP中的几个基本概念

➤ 如：（时间，商品种类，商店，销售额）是一个有关商品销售额的三维数组，其中的数据成员可以表示为：

（**2017年**，家电，南京市，**5000万**）

（**2017年7月**，女式服装，江苏省，**2000万**）

.....

## 7.1 OLAP中的几个基本概念

### □ 数据单元（单元格）

- 多维数组可以被看成是一个根据多个下标进行定位的值的集合，其中的每一个取值被称为该多维数组的一个数据单元。
- 当多维数组的每一维都选中一个维成员，这些维成员的组合就唯一确定了一个观察对象的值，即：  
*(维成员1, 维成员2, ……, 维成员n, 对象值)*
- 这样一个值或存放该值的地方我们称其为一个‘*数据单元*’
- “(2000年1月, 上海, 笔记本电脑, \$100000)”

## 7.1 OLAP中的几个基本概念

➤ 假设：

- 在一个分析型应用中有若干个分析对象（设为r个），以它们为聚焦点作不同角度（设每个分析对象有m个维）与深度（设每个维上又分为n个层）的分析
- 那么可以得到多种不同的统计分析结果（共为 $(r \times m \times n)$  种），即可以构造出 $(r \times m \times n)$  个多维数组。

## 7.1 OLAP中的几个基本概念

- 为了方便快速地查到这些统计分析结果，OLAP需要解决以下三个问题：
  - OLAP的基本数据模型
  - OLAP的数据构造方式
    - 数据立方体(Data Cube)
    - 数据超立方体(Data Super Cube/Hypercube)

# 7 联机分析处理（OLAP）

7.1 OLAP中的几个基本概念

7.2 OLAP的基本数据模型

7.3 OLAP中的数据构造方式

7.4 数据立方体（Data Cube）

## 7.2 OLAP的基本数据模型

**OLAP**中使用多维数据模型建立起基于事实和维的数据库模型，从而满足用户多角度多层次进行数据查询和分析的需求。

- 常用的两种**OLAP**数据模型
  - 星形模型
  - 雪花模型

## 7.2 OLAP的基本数据模型

### □ 星型模式(Star Schema)

– 星型模式是一种多维表结构，它一般由两种不同性质的二维表组成：

- 事实表(fact table)

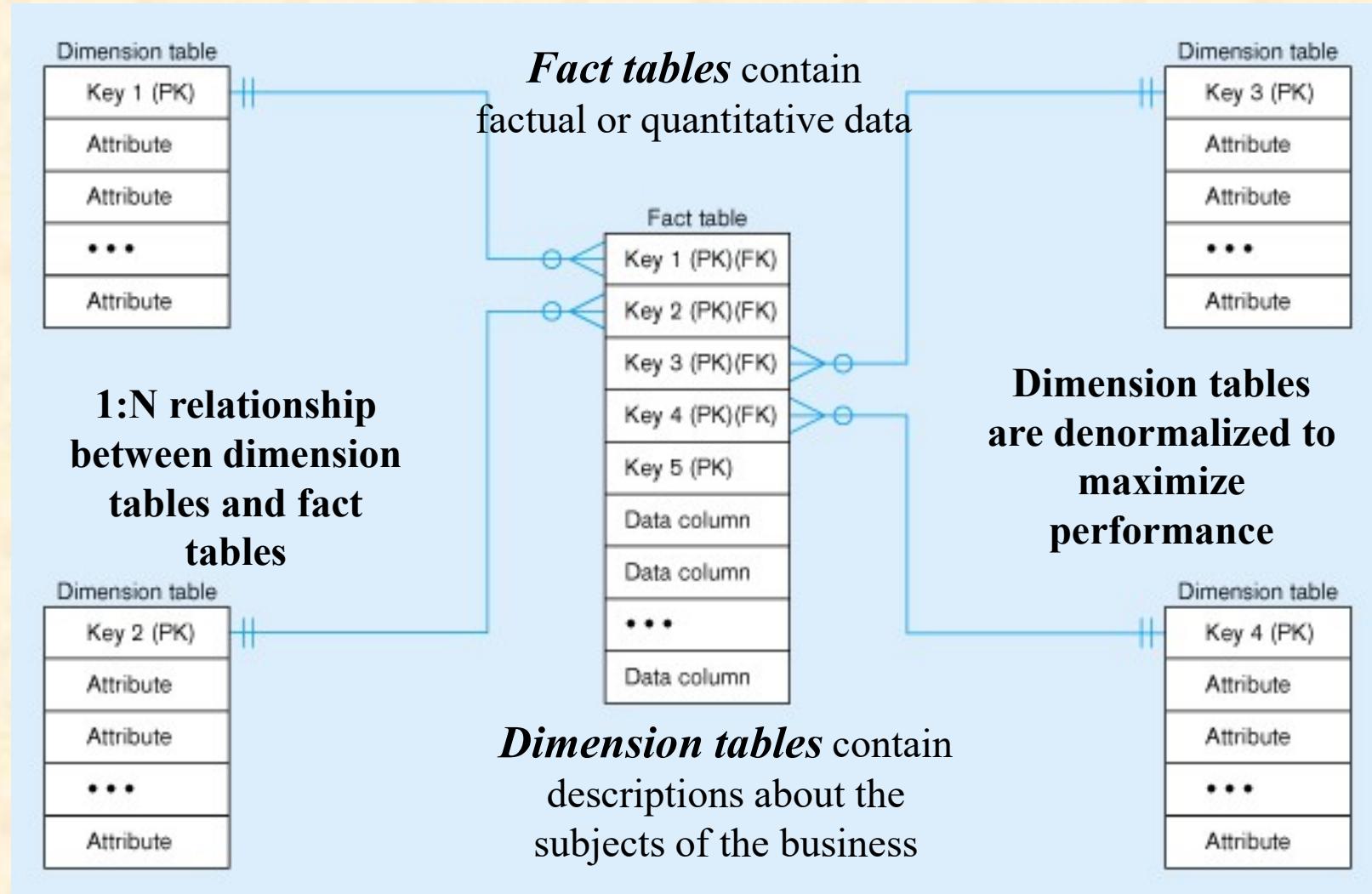
它存放多维表中的主要事实，包含了每个事件的具体要素

- 维表(Dimension Table)

用以存放多维表中的维成员的取值

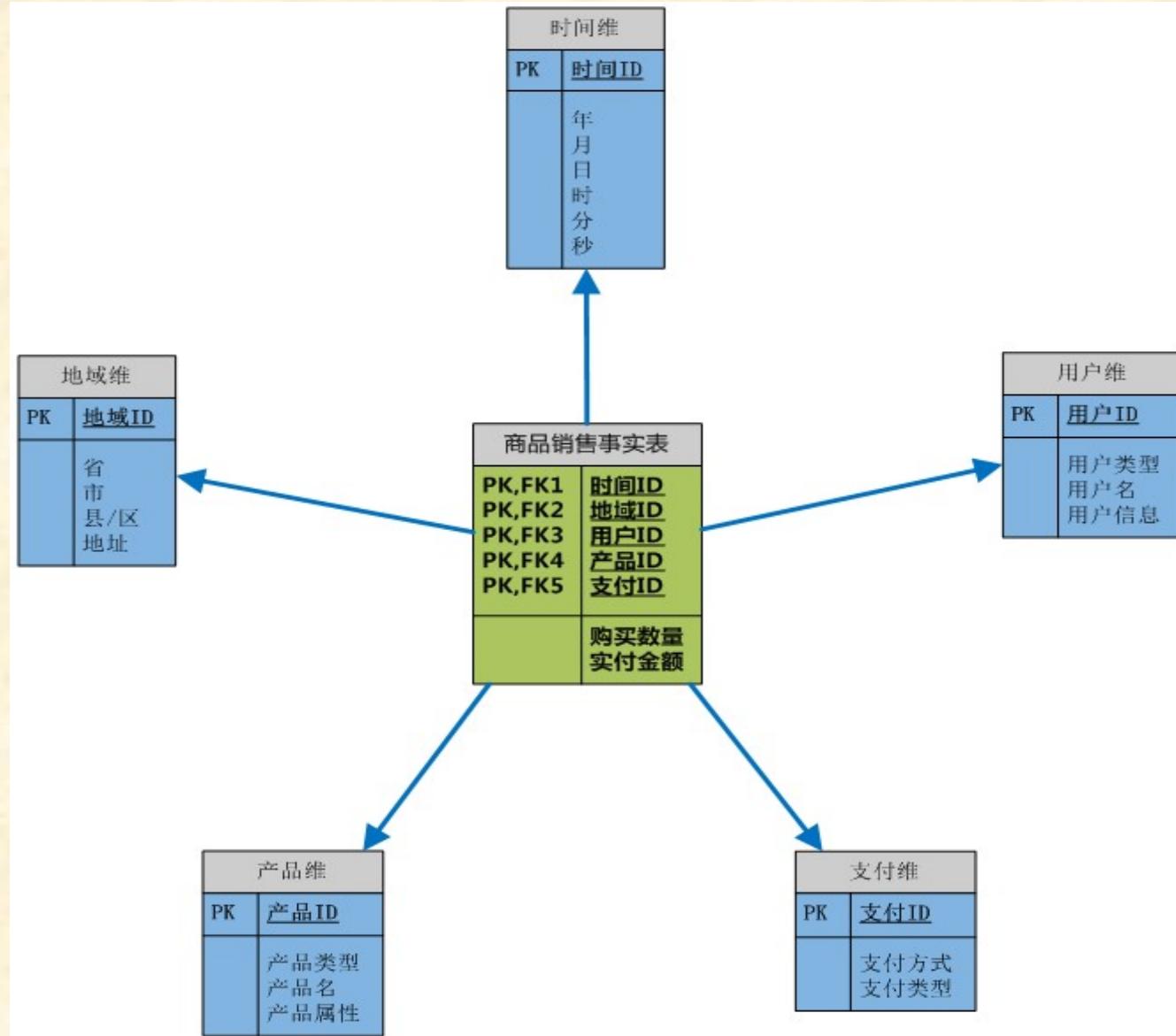
一般一个n维的多维表往往有n个维表和一个事实表，它们构成了一个星形结构，因而称其为‘星型模式’。

# star schema



Excellent for ad-hoc queries,  
but bad for online transaction processing

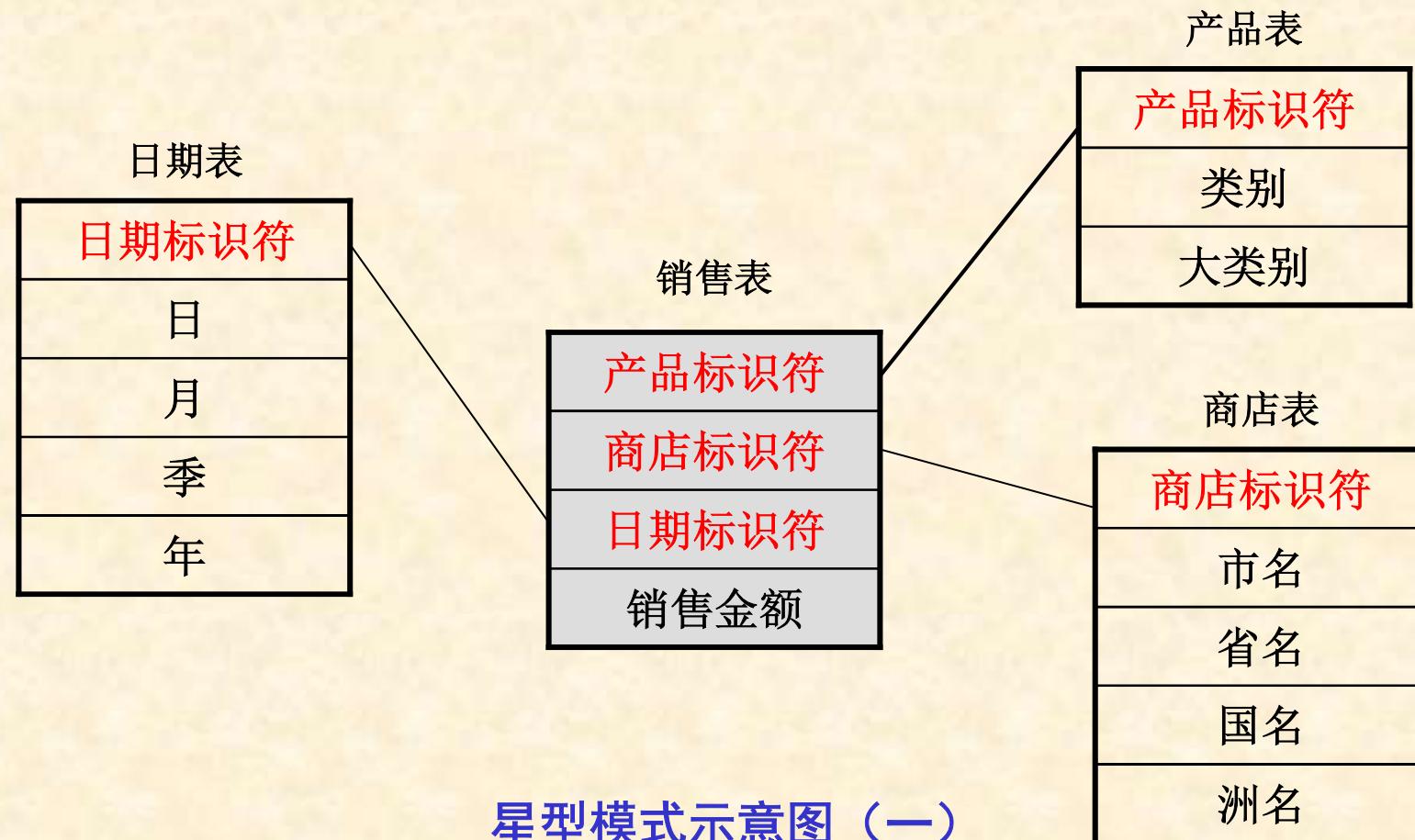
# star schema



在星型模式中，主要数据存储在事实表中，没有冗余，并符合3NF或BCNF。

维值信息存储在维表中。维表一般不需要规范化。主要是原因是维表是静态的，其信息更新频率不高或者保持相对的稳定。是否会产生因更新而导致异常也就不再重要了。

## 7.2 OLAP的基本数据模型



维表中的信息一般是可以分层的，比如时间维的年月日、地域维的省市县等。分层为的是满足事实表中的度量可以在不同的粒度上完成聚合，比如2017年商品的销售额，来自上海市的销售额等。

**Product**

<u>Product</u> <u>_Code</u>	Description	Color	Size
100	Sweater	Blue	40
110	Shoes	Brown	10 1/2
125	Gloves	Tan	M
• • •			

**Period**

<u>Period</u> <u>_Code</u>	Year	Quarter	Month
001	1999	1	4
002	1999	1	5
003	1999	1	6
• • •			

**Sales**

<u>Product</u> <u>_Code</u>	<u>Period</u> <u>_Code</u>	<u>Store</u> <u>_Code</u>	<u>Units</u> <u>_Sold</u>	<u>Dollars</u> <u>_Sold</u>	<u>Dollars</u> <u>_Cost</u>
110	002	S1	30	1500	1200
125	003	S2	50	1000	600
100	001	S1	40	1600	1000
110	002	S3	40	2000	1200
100	003	S2	30	1200	750
• • •					

**Store**

<u>Store</u> <u>_Code</u>	<u>Store</u> <u>_Name</u>	City	Telephone	Manager
S1	Jan's	San Antonio	683-192-1400	Burgess
S2	Bill's	Portland	943-681-2135	Thomas
S3	Ed's	Boulder	417-196-8037	Perry
• • •				

## 7.2 OLAP的基本数据模型

➤ 上述的星型模式可以转化成下面的四个关系：

- 事实表：

销售表（产品标识符，商店标识符，日期标识符，销售额）

- 维表1：

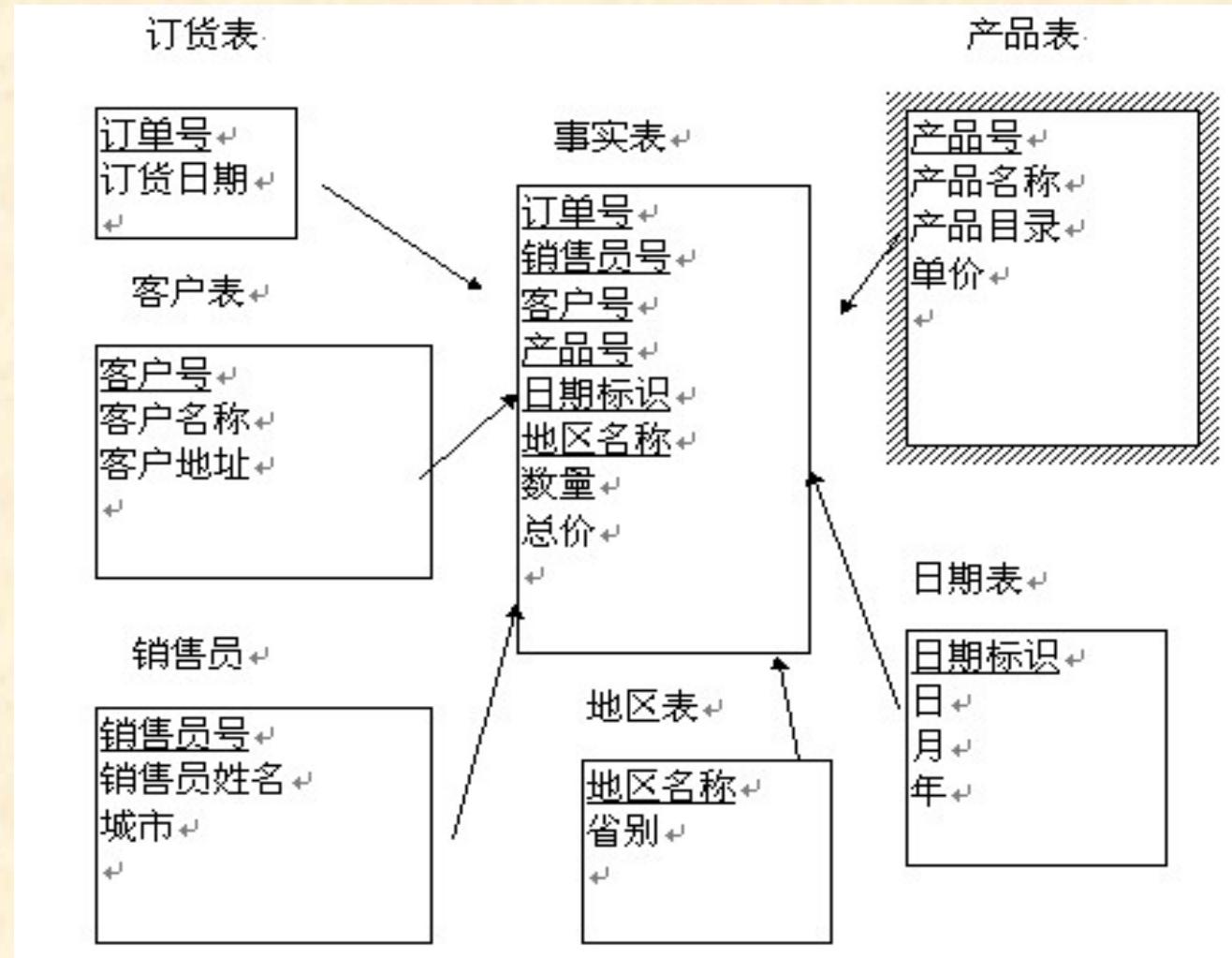
产品表（产品标识符，类别，大类别）

- 维表2：

商店表（商店标识符，市名，省名，国名，洲名）

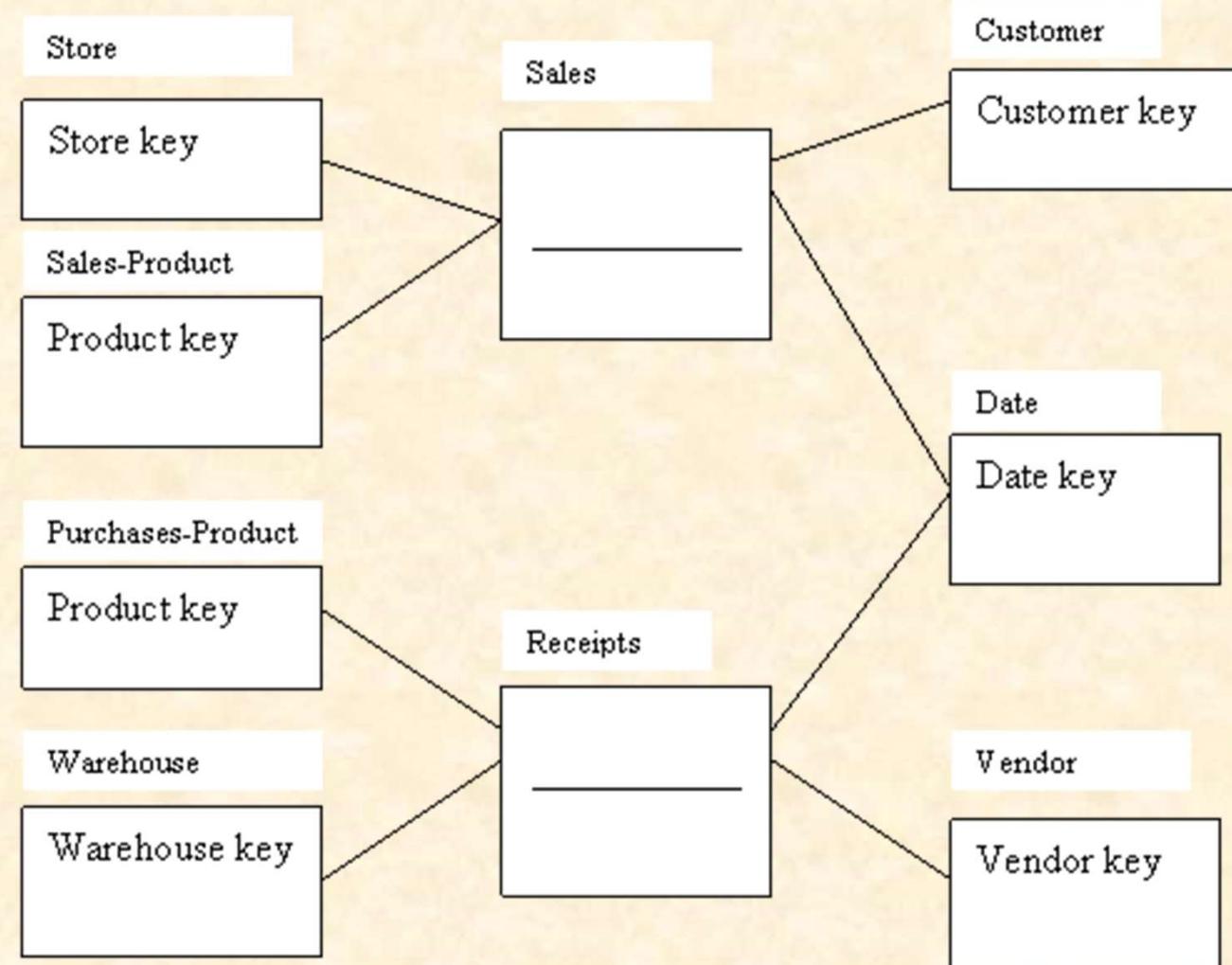
- 维表3：

时间表（时间标识符，日期，月份，季度，年份）

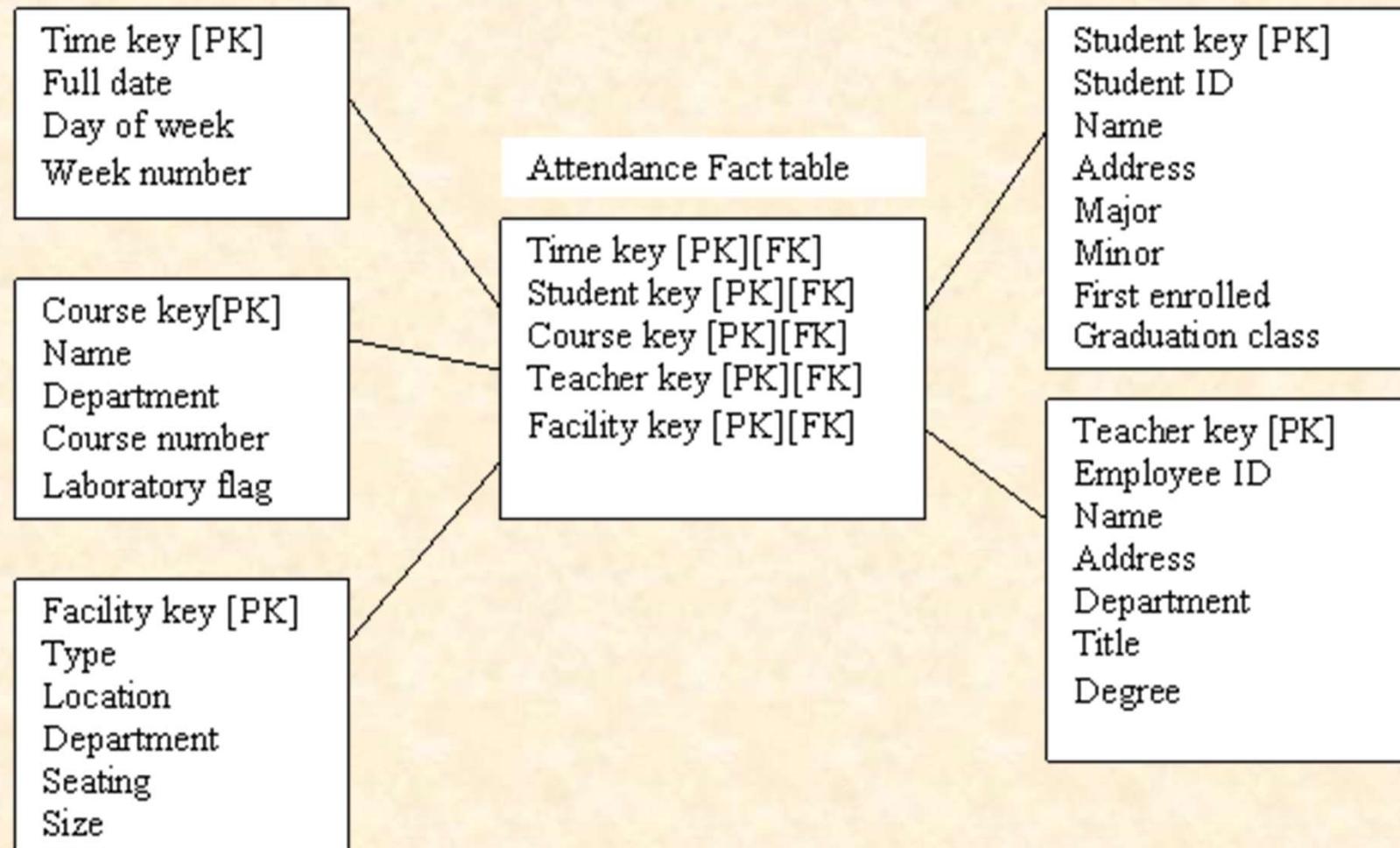


星形模式示意图（二）

# Star Schema扩展：Multiple Fact tables

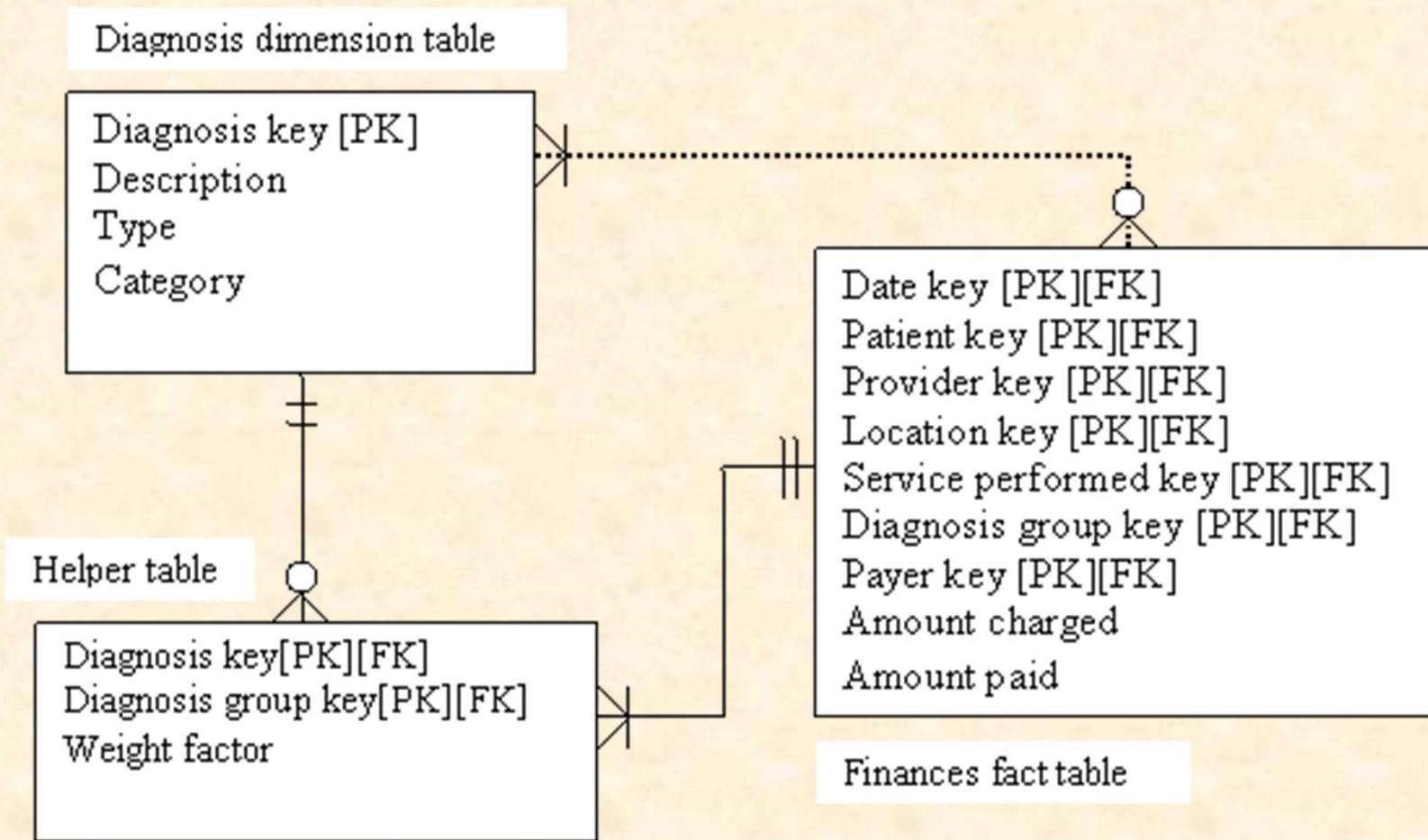


# Star Schema扩展：Factless Fact Tables



用于描述事件的发生情况

# Star Schema 扩展: Normalizing dimension tables



Multivalued dimension: N:M relationship

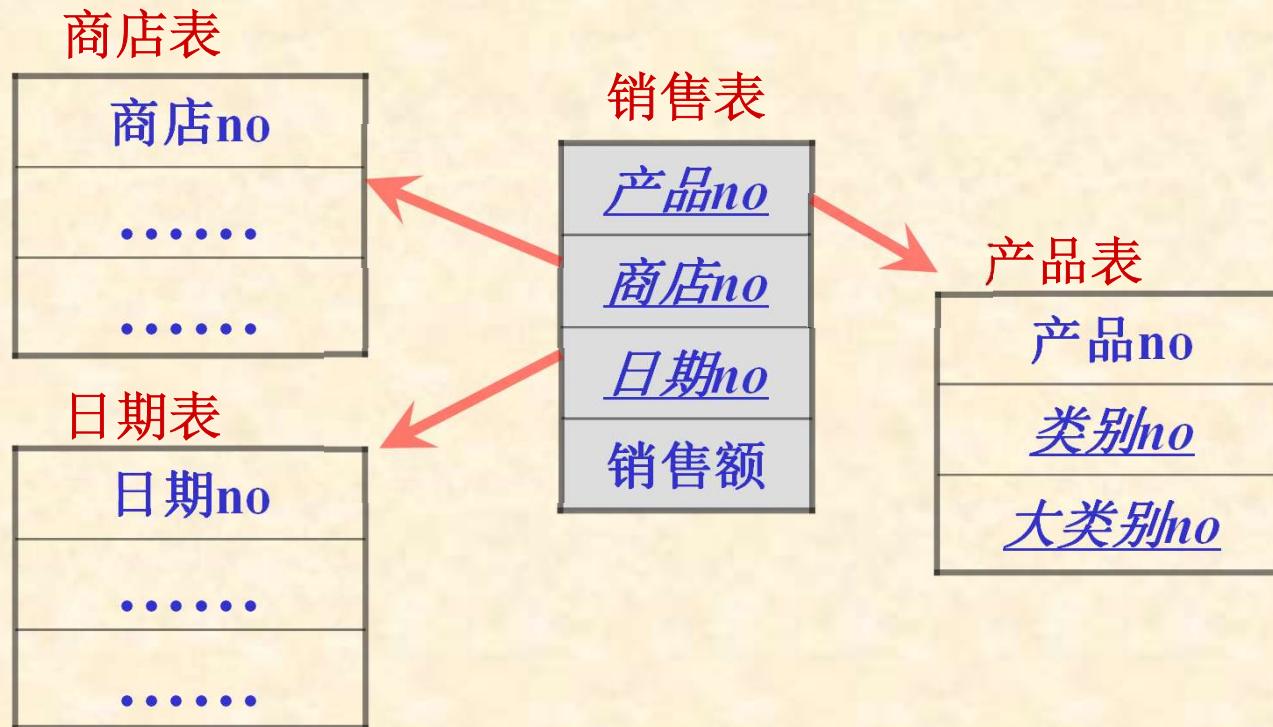
## 7.2 OLAP的基本数据模型

### □ 雪花模式 (Snowflake Schema)

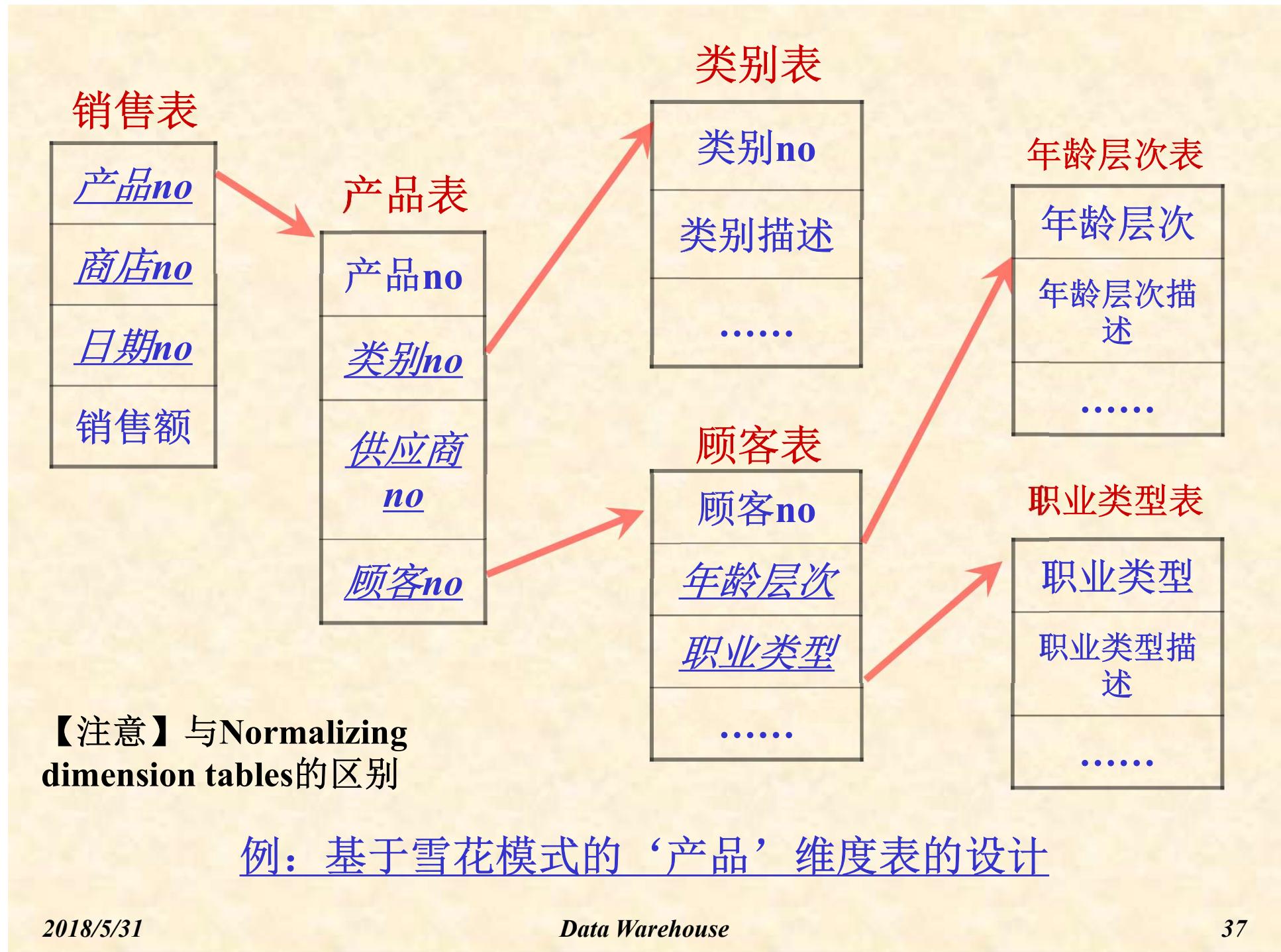
- 雪花模型是对星型模型的扩展

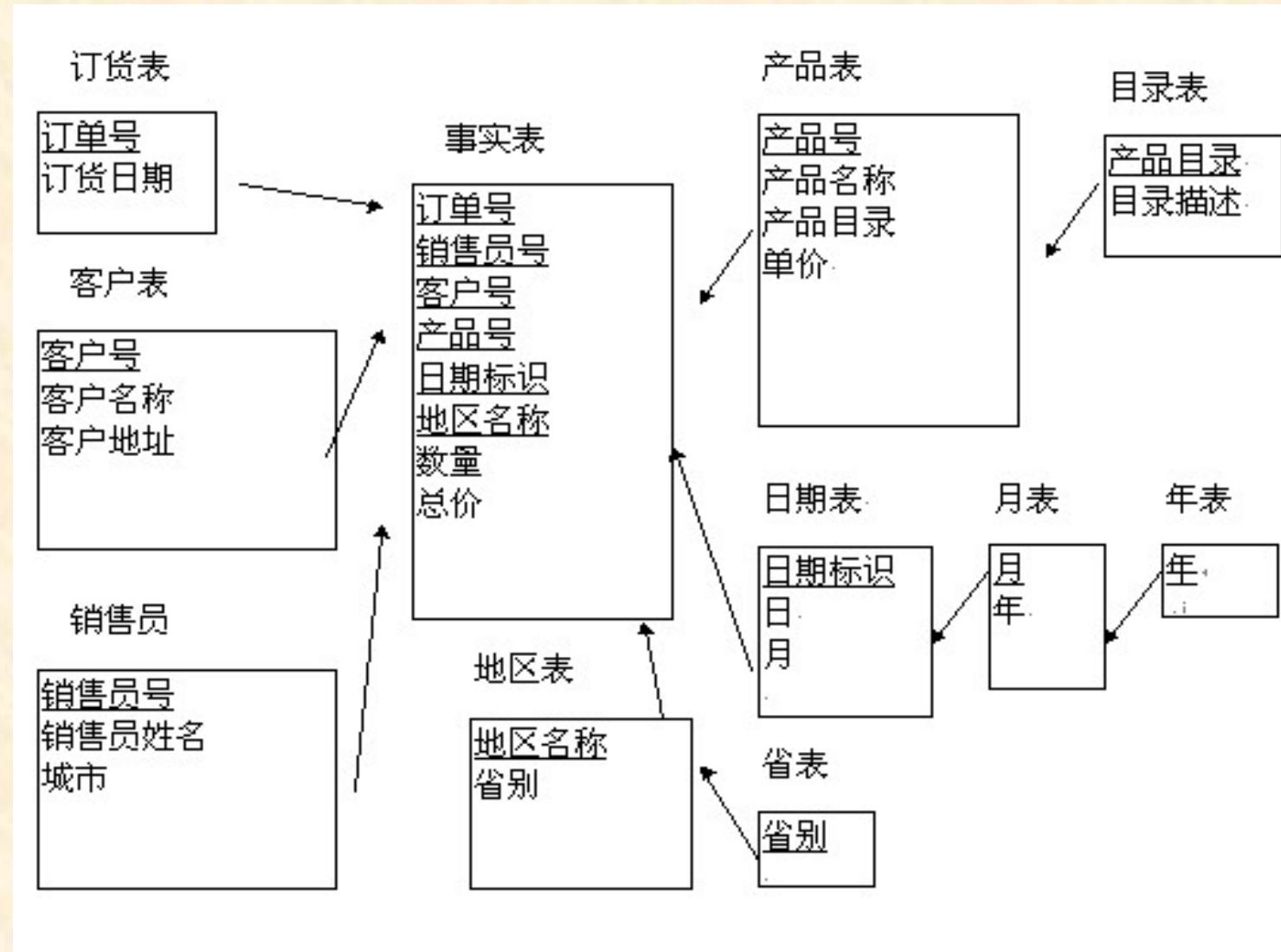
雪花模型对星型模型的维表进一步层次化，原来的维表可能被扩展为小的事实表，形成一些局部的“层次”区域。





- 在上面的‘星型模式’中，在‘产品’维度表中只有产品的分类信息，只能根据产品‘类别’来分析其‘销售额’的变化规律。
- 如果需要从产品的‘供应商’或‘购买顾客’角度来分析产品销售的变化趋势，那么可以在‘产品’维度表中加入与‘供应商’和‘顾客’有关的属性。
  - 如果对扩展后的‘产品’维度表进行规范化设计，那么可以构成一个以‘产品维度表’为事实表、以类别、供应商、顾客为维度的‘星型模型’





## 雪花模式（例二）

# □为什么要对维度表进行规范化设计？

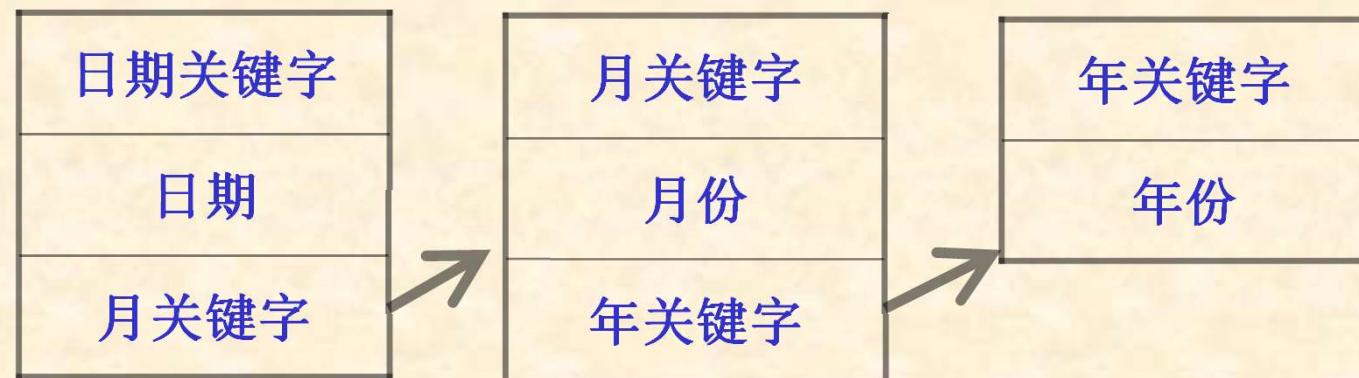
一个简单的  
‘日期’ 维度  
表

日期关键字
日
月
年

1234567
2日
3月
2012年

一个 ‘日期’  
维成员值

规范化后的  
‘日期’ 维度  
表



规范化的 ‘日  
期’ 维成员值



# 同传统的关系模型相比，多维数据模型有着自身的优缺点

## 优点：

基于分析优化的数据组织和存储模式。如：电商网站数据库中记录着某时间，用户购买了某个商品，并寄送到的具体地址。但我们无法马上获取 2017 年的 7 月份到底有多少浙江省的用户购买了商品。但是在基于多维模型的基础上，此类查询将变得简单，只要在时间维上将数据聚合到 2017 年的 7 月份，同时在地域维上将数据聚合到浙江省的粒度就可以实现。

## 缺点：

与关系模型相比其灵活性不够，一旦模型构建就很难进行更改。比如一个订单的事实，其中用户可能购买了多种商品，包括了时间、用户维和商品数量、总价等度量。在关系模型中，如果需要区分订单中包含了哪些商品，只需另外建一张表记录订单号和商品的对应关系即可。但在多维模型中，一旦事实表构建起来，我们无法将事实表中的一条订单记录再进行拆分，于是无法建立以一个新的维度——产品维，只能另外再建个以产品为主题的事实表。

## 7.2 OLAP的基本数据模型

### ➤ 第三范式在数据仓库中的应用

- 大多数人在设计中央数据仓库的逻辑模型时,都按照第三范式来设计;而在进行物理实施时,则由于数据库引擎的限制,不得不对逻辑模型进行不规范处理 (**De-Normalize**), 以提高系统的响应速度。
- 根据数据仓库的测试标准 **TPC-D** 规范,在数据仓库系统中,对数据库引擎最大的挑战主要是这样几种操作:多表连接、表的累计、数据排序、大量数据的扫描。

## 7.2 OLAP的基本数据模型

➤ 在实际系统中针对这些困难采用的折衷处理办法：

- 1、**避免多表连接**：在设计模型时对表进行合并，即所谓的预连接（Pre-Join）。当数据规模小时，也可以采用星型模式，这样能提高系统速度，但增加了数据冗余量。
- 2、**避免表的累计**：在模型中增加有关小计数据（Summarized Data）的项。这样也增加了数据冗余，而且如果某项问题不在预建的累计项内，需临时调整。
- 3、**避免数据排序**：对数据事先排序。但随着数据仓库系统的运行，不断有新的数据加入，数据库管理员的工作将大大增加。大量的时间将用于对系统的整理，系统的可用性随之降低。
- 4、**避免大表扫描**：通过使用大量的索引，可以避免对大量数据进行扫描。但这也将增加系统的复杂程度，降低系统进行动态查询的能力。

## 7.2 OLAP的基本数据模型

### ➤ 不规范处理

- 由于中央数据仓库的数据模型反映了整个企业的业务运行规律，在此进行不规范处理容易影响整个系统，不利于今后的扩展。
- 不规范处理产生的数据冗余将使整个系统的数据量迅速增加，将增加 **DBA**的工作量和系统投资。
- 选择数据集市阶段进行不规范处理

### ➤ 对于部门数据集市，当数据量不大、报表较固定时可以采用星型模式

➤ 对于中央数据仓库，考虑到系统的可扩展能力、投资成本和易于管理等多种因素，最好采用第三范式。

## 7.2 OLAP的基本数据模型

### ➤ OLAP 的优势

OLAP 的优势是基于数据仓库面向主题、集成的、保留历史及不可变更的数据存储，以及多维模型多视角多层次的数据组织形式。

#### 数据展现方式

基于多维模型让数据的展示更加直观，从多个角度多个层面去发现事物的不同特性。

#### 查询效率

多维模型的建立是基于对 OLAP 操作的优化基础上的，这些优化使得对百万千万甚至上亿数量级的运算变得得心应手。

#### 分析的灵活性

用上面介绍的各类 OLAP 操作对数据进行聚合、细分和选取，这样提高了分析的灵活性，可以从不同角度不同层面对数据进行细分和汇总，满足不同分析的需求。

# 7 联机分析处理（OLAP）

7.1 OLAP中的几个基本概念

7.2 OLAP的基本数据模型

7.3 OLAP中的数据构造方式

7.4 数据立方体（Data Cube）

## 7.3 OLAP的数据构造方式

- ROLAP : ( Relational OLAP)
- MOLAP: ( Multi-Dimensional OLAP)
- HOLAP : ( Hybrid OLAP)
- 桌面型OLAP: ( Desktop OLAP)

## 7.3 OLAP的数据构造方式

### ➤ ROLAP: (Relational OLAP)

- 用传统的‘**关系数据库管理系统**’(RDBMS)管理，将星型(雪花型)模式用二维表形式存储，表间用关键字相连，从而构成一个关系模式，它称为**ROLAP**
- 用户在**ROLAP**上的查询操作将被改写成**RDBMS**中的查询操作并执行获得查询结果。
  - **SQL-99**有部分**data warehouse**的扩展
  - 不同数据库厂商对**SQL-99**有不同的支持

## 7.3 OLAP的数据构造方式

### ➤ MOLAP: (Multi-Dimensional OLAP)

- 用‘多维数据库管理系统’管理，多维数据库采用的基本数据模式就是‘多维数组’（多维矩阵）
- 在MOLAP中
  - ‘事实表’被表示成一个‘多维数组’
  - ‘维’的属性值被映射成‘多维数组’的下标
  - 度量值则被作为多维数组的取值存储在数据单元中

## 7.3 OLAP的数据构造方式

### ➤ HOLAP: (Hybrid)

- 介于 MOLAP 和 ROLAP 的类型，细节数据以 ROLAP 的形式存放，更加方便灵活，而高度聚合数据以 MOLAP 的形式展现，更适合于高效的分析处理

### ➤ 桌面型OLAP

- PC环境支持的简单多维分析
- 没有自己的数据存储
- 把用户提交的查询翻译成对数据源的查询，然后从数据源中提取结果数据，并将这些结果数据合成最终的结果返回给客户

产品名称	地区	销售量
冰箱	东北	50
冰箱	西北	60
冰箱	华北	100
电视机	东北	40
电视机	西北	70
电视机	华北	80
空调	东北	90
空调	西北	120
空调	华北	140

图A: ROLAP

	东北	西北	华北
冰箱	50	60	100
电视机	40	70	80
空调	90	120	140

图B: MOLAP

	东北	西北	华北	ALL
冰箱	50	60	100	210
电视机	40	70	80	190
空调	90	120	140	350
ALL	180	250	320	750

图C: MOLAP

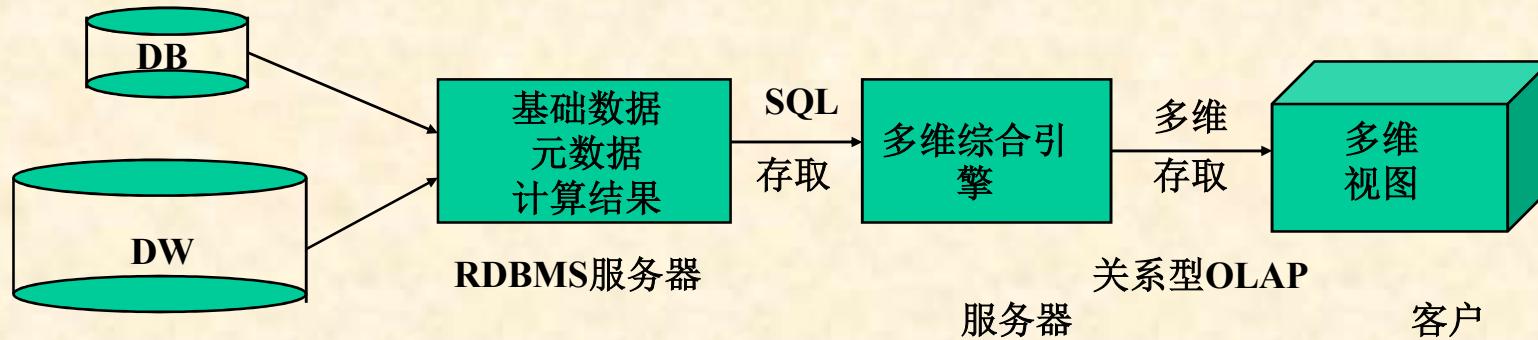
# ROLAP与MOLAP的比较 (1)

ROLAP	MOLAP
<ul style="list-style-type: none"><li>沿用现有的关系数据库的技术</li></ul>	<ul style="list-style-type: none"><li>专为<b>OLAP</b>所设计</li></ul>
<ul style="list-style-type: none"><li>响应速度慢</li></ul>	<ul style="list-style-type: none"><li>性能好、响应速度快</li></ul>
<ul style="list-style-type: none"><li>数据装载速度快</li></ul>	<ul style="list-style-type: none"><li>数据装载速度慢</li></ul>
<ul style="list-style-type: none"><li>存储空间耗费小，维数没有限制</li></ul>	<ul style="list-style-type: none"><li>需要进行预算算，可能导致数据爆炸；维数有限；无法支持维的动态变化</li></ul>
<ul style="list-style-type: none"><li>借用<b>RDBMS</b>存储数据，没有文件大小限制</li></ul>	<ul style="list-style-type: none"><li>受操作系统平台中文件大小的限制，难以达到<b>TB</b>级</li></ul>

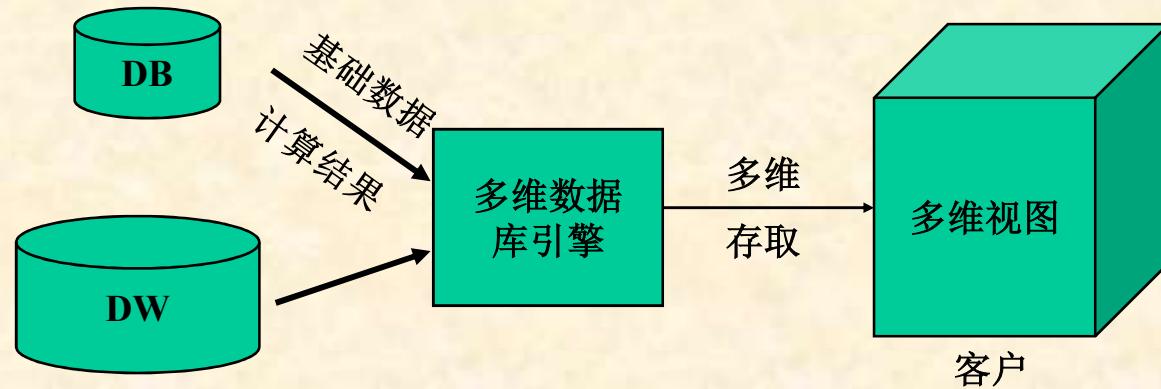
## ROLAP与MOLAP的比较 (2)

ROLAP	MOLAP
<ul style="list-style-type: none"><li>可以通过<b>SQL</b>实现详细数据与概要数据的存储</li></ul>	<ul style="list-style-type: none"><li>缺乏数据模型和数据访问的标准</li></ul>
<ul style="list-style-type: none"><li>不支持有关预算计算的读写操作</li><li><b>SQL</b>无法完成部分计算</li><li>无法完成多行的计算</li><li>无法完成维之间的计算</li></ul>	<ul style="list-style-type: none"><li>支持高性能的决策支持计算</li><li>复杂的跨维计算</li><li>多用户的读写操作</li><li>行级的计算</li></ul>
<ul style="list-style-type: none"><li>维护困难</li></ul>	<ul style="list-style-type: none"><li>管理简便</li></ul>

## ROLAP



## MOLAP



## 7.3 OLAP的数据构造方式

- OLAP服务器必须提高对OLAP数据的访问效率，包括：
  - 数据抽取、转换及加载的效率
  - OLAP数据查询效率
  - OLAP数据更新效率

## 7.3 OLAP的数据构造方式

- 目前可有多种方法以提高**OLAP**中的处理效率：
  - 1) 采用物化视图方式；  
    包括一个查询结果的数据库对象
  - 2) 采用特殊的索引与集簇方式，以加速星型模式内表的连接速度；
  - 3) 尽量采用并行操作方式以提高处理速度；
  - 4) 采用**OLAP**中的查询优化技术，如共享排序技术等；
  - 5) 采用增量技术，在**OLAP**数据更新时保留不变的数据，仅更改变动的数据以加快数据更新速度。

## 7.3 OLAP的数据构造方式

### 3. HOLAP

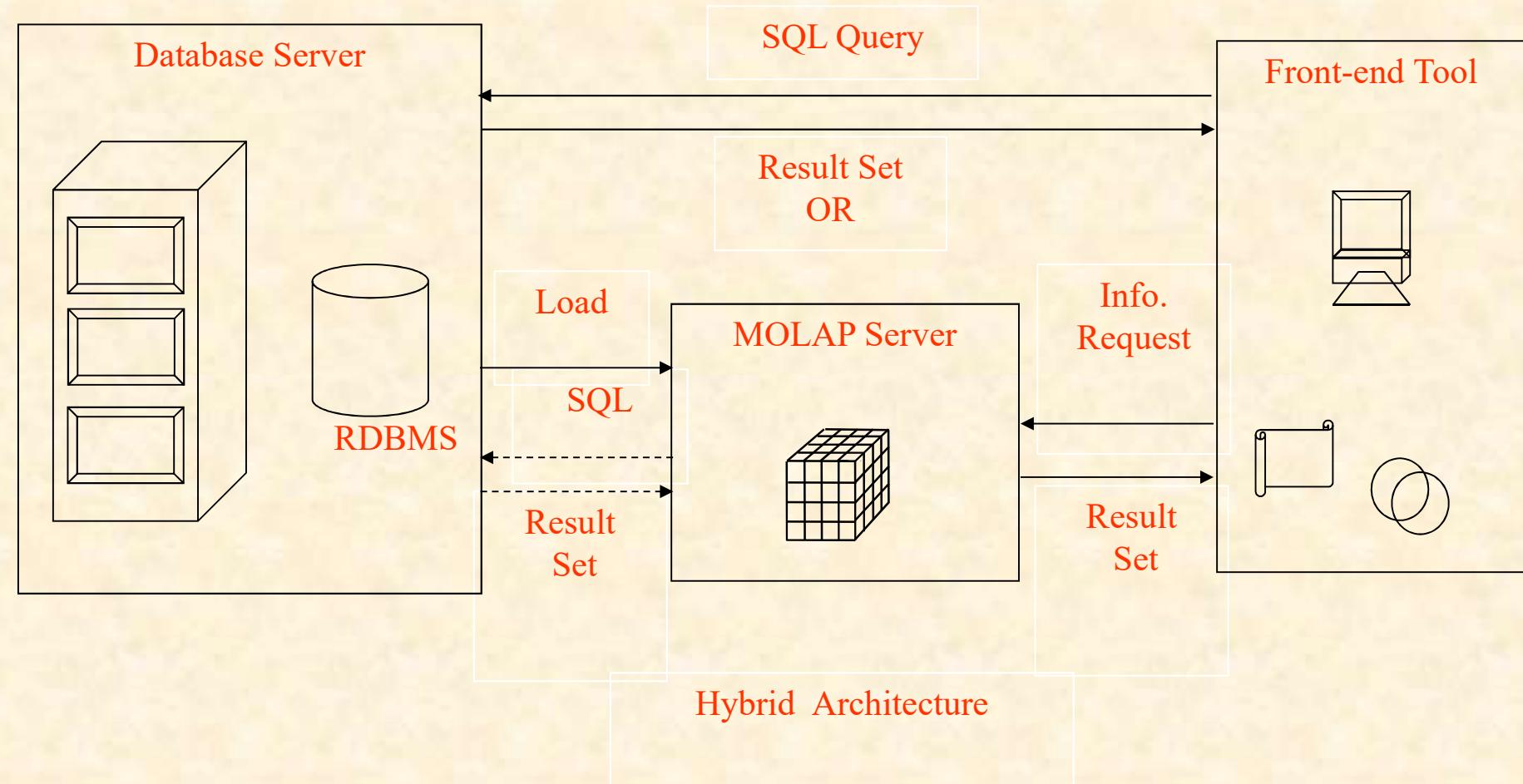
- 由于MOLAP和ROLAP有着各自的优点和缺点,且它们的结构迥然不同, 这给分析人员设计OLAP结构提出了难题。
- 为此一个新的OLAP结构——混合型OLAP (HOLAP) 被提出, 它能把MOLAP和ROLAP两种结构的优点结合起来。
- 迄今为止, 对HOLAP还没有一个正式的定义。但很明显, HOLAP结构不应该是MOLAP与ROLAP结构的简单组合, 而是这两种结构技术优点的有机结合, 能满足用户各种复杂的分析请求。

## 7.3 OLAP的数据构造方式

### 3. HOLAP

- 以HOLAP格式存储的立方体，要比以OLAP格式存储的立方体小，在查询总结数据时，又比ROLAP快。
- HOLAP存储格式一般比较适合于需要总结数据的查询有较快的响应时间，同时基数据的量又比较大的场合。

## ➤ HOLAP



# 7 联机分析处理（OLAP）

## 7.1 OLAP中的几个基本概念

## 7.2 OLAP的基本数据模型

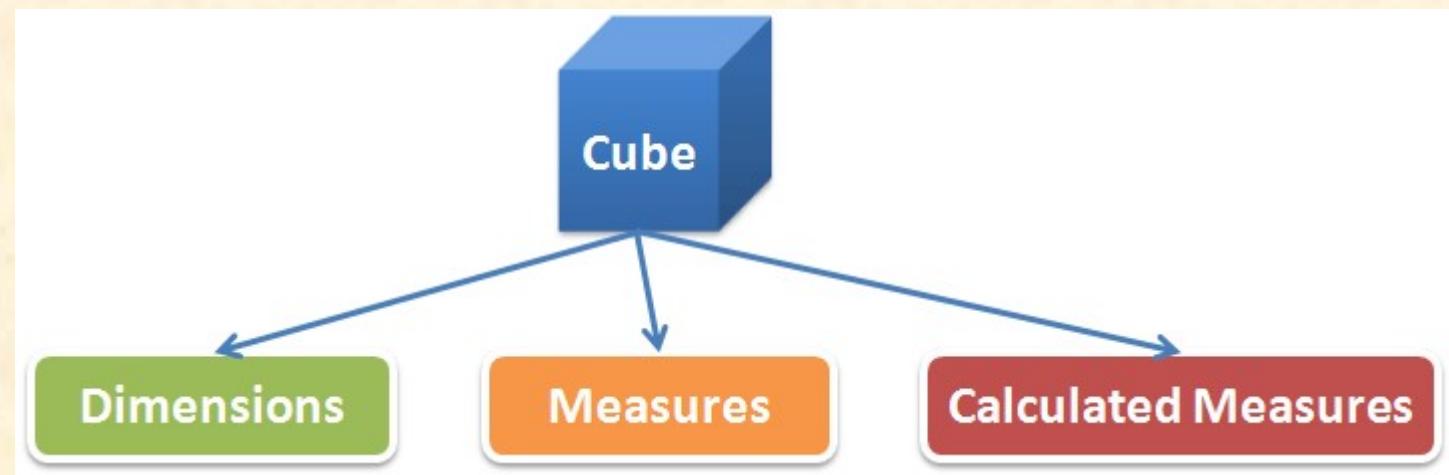
## 7.3 OLAP中的数据构造方式

## 7.4 数据立方体（Data Cube）

## 7.4 数据立方体与超立方体

### □ 数据立方体 (Data Cube)

- 数据仓库的数据模式通常可以看成是定义在多个数据源上的数据视图，其中存储的分析型数据通常是一些经过统计而获得的总结性数据。



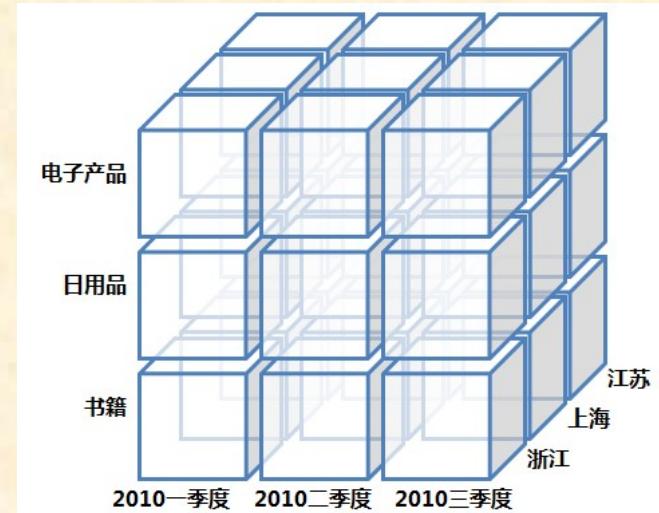
度量 (Measure) : 用于描述事件的数字尺度，比如网站的浏览量 (Pageviews)、访问量 (Visits)，再如电子商务的订单量、销售额等。

计算度量 (Calculated Measure) : 通过度量计算得到，比如同比 (如去年同期的月利润)、环比 (如上个月的利润) 等

## 7.4 数据立方体与超立方体

### 口物化视图

- 为了提高对统计信息的查询速度，我们可以预先计算好数据视图中的统计信息并保存在数据仓库中，这称为‘物化视图’，即将虚的视图转变成实际的视图。
- 存放物化视图的三维数据模型叫‘数据立方体’
- 获取这些总结性数据的常用方法是在视图中用统计函数进行计算，但这种方法的缺点是显见的：时间开销太大



## 7.4 数据立方体与超立方体

以上面的星型模式为例，其事实表共有三维，即产品P（product）、商店S（store）及日期D(Date)，可以为它们定义一系列的物化视图。

### (1) PSD视图

**CREATE VIEW PSD** (产品标识符，商店标识符，  
日期标识符，销售总额)

**AS**

(**SELECT** 产品标识符，商店标识符，  
日期标识符，**SUM** (销售金额))

**FROM** 销售表

**GROUP BY** 产品标识符，商店标识符，  
日期标识符)

## 7.4 数据立方体与超立方体

### (2) PS、SD、PD视图

**CREATE VIEW PS** (产品标识符, 商店标识符,  
销售总额)

**AS (SELECT** 产品标识符, 商店标识符,  
**SUM** (销售金额)

**FROM PSD**

**GROUP BY** 产品标识符, 商店标识符)

采用类似的方法也可以定义出**SD**、**PD**视图。

## 7.4 数据立方体与超立方体

### (3) P、S、D视图

**CREATE VIEW P** (产品标识符, 销售总额)

**AS (SELECT 产品标识符, SUM (销售金额)**

**FROM PS**

**GROUP BY 产品标识符)**

采用类似的方法也可以定义出S、D视图

## 7.4 数据立方体与超立方体

### (4) ALL视图

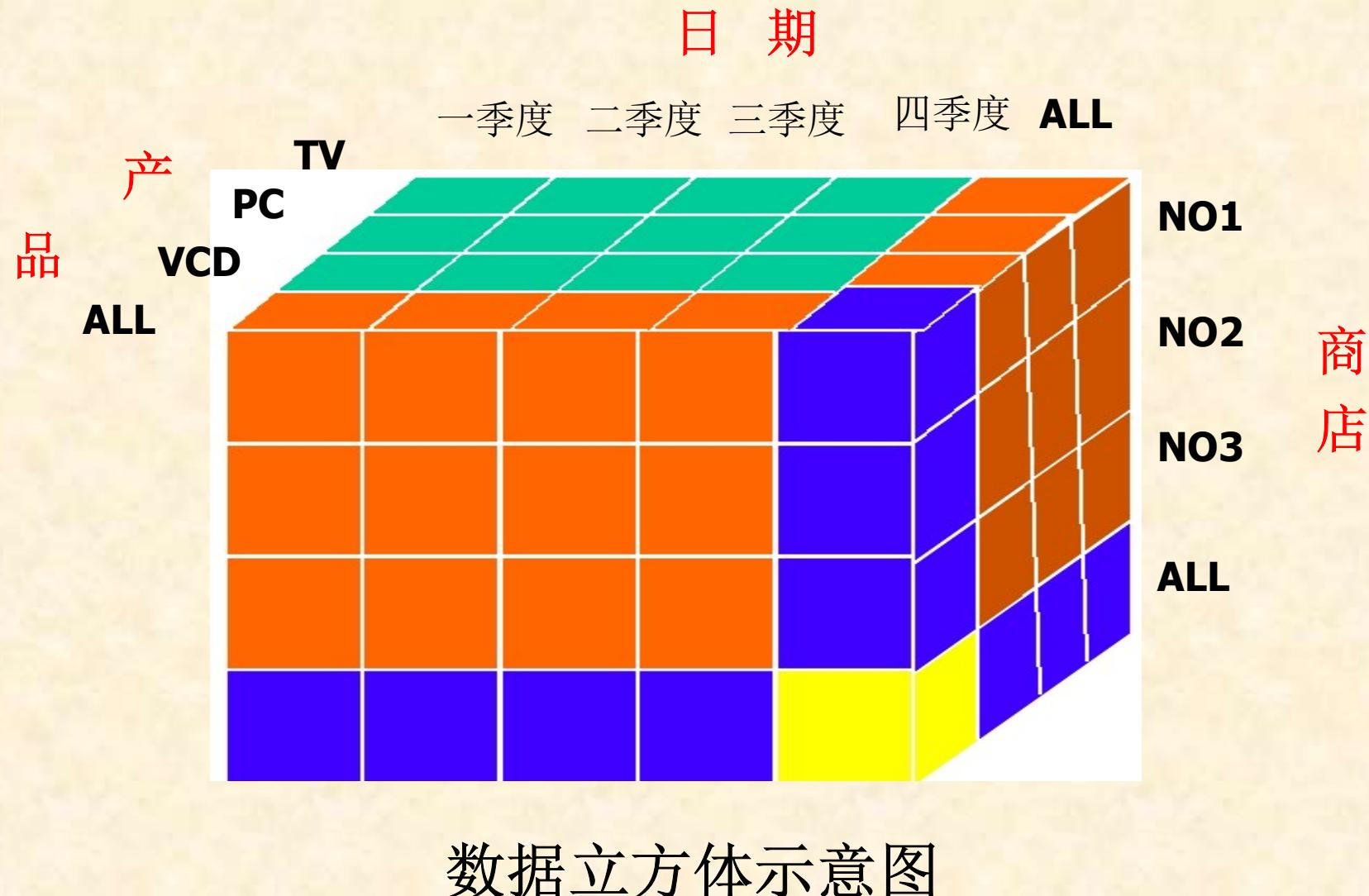
ALL视图表示不分组，该视图中的销售总额表示销售表中所有销售金额之和。其定义如下：

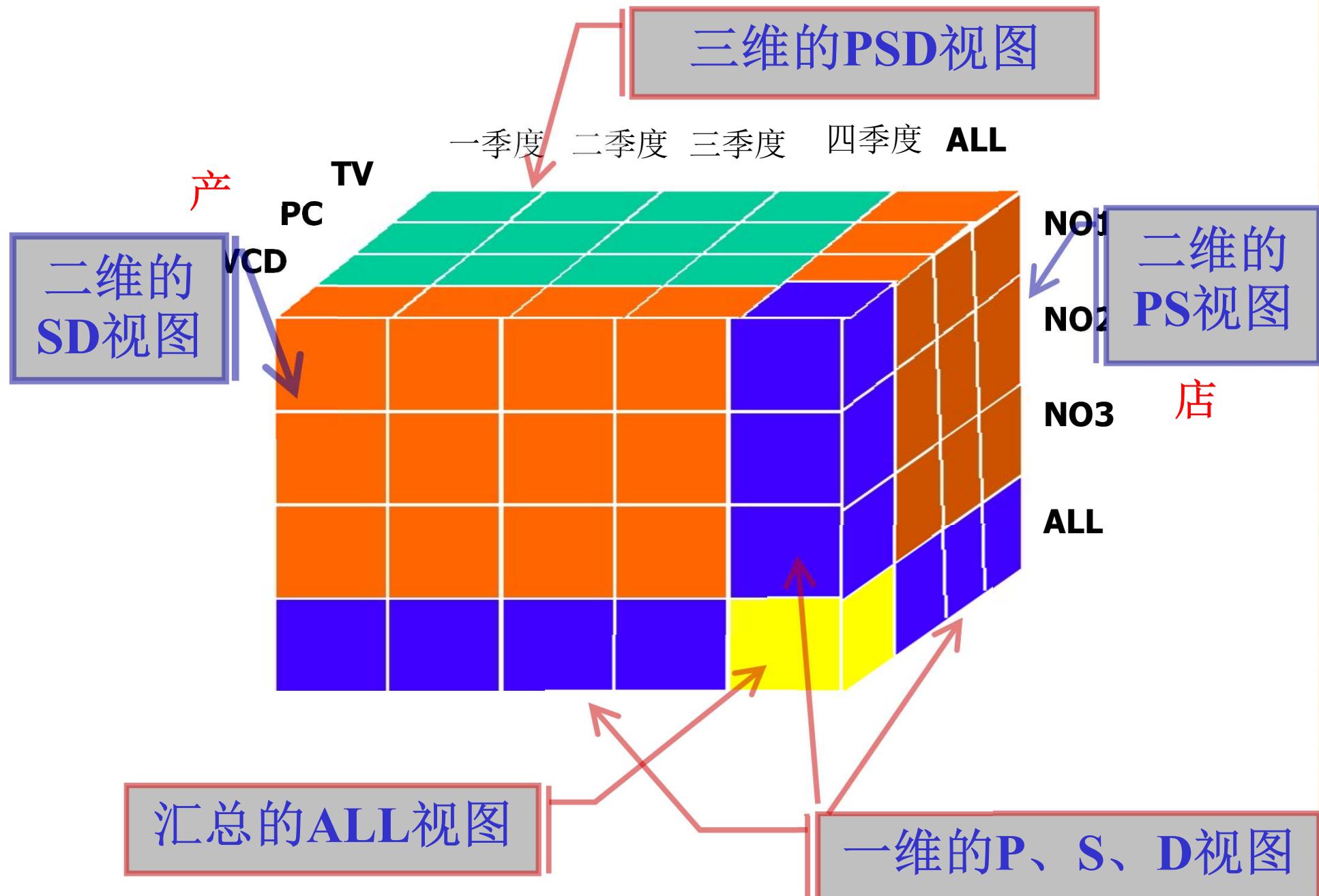
**CREATE VIEW ALL** (销售总额)

**AS (SELECT SUM (销售总额)**

**FROM PSD)**

## 7.4 数据立方体与超立方体





## 7.4 数据立方体与超立方体

### □数据超立方体（Data Super Cube）

- 在数据立方体中进行的是一个三维的分析应用。
- 当应用中分析对象超过三维时，则构成一个多维(或称 $n$ 维,  $n \geq 4$ )应用，此时无法用数据立方体表示其中的数据，而只能通过虚拟的 $n$  ( $n \geq 4$ ) 维空间建立 $n$ 维立方体，它称为‘数据超立方体’

## 7.4 数据立方体与超立方体

### □ 数据立方体的分区存储（物理设计）

- 分区：实现数据的逻辑分段
- 立方体数据结构可以存储到一个或多个分区
- 每个分区可以使用不同的存储模式（**ROLAP**、**MOLAP**、**HOLAP**）
- 分区可以组合
- 优点：有助于数据结构进行管理和协调

## 7.4 数据立方体与超立方体

### ➤ 数据立方体的更新

#### ➤ 增量更新

将新数据添加到立方体中的分区并更新聚合。

#### ➤ 刷新

清除并重新加载立方体数据，并重新计算它的聚合。

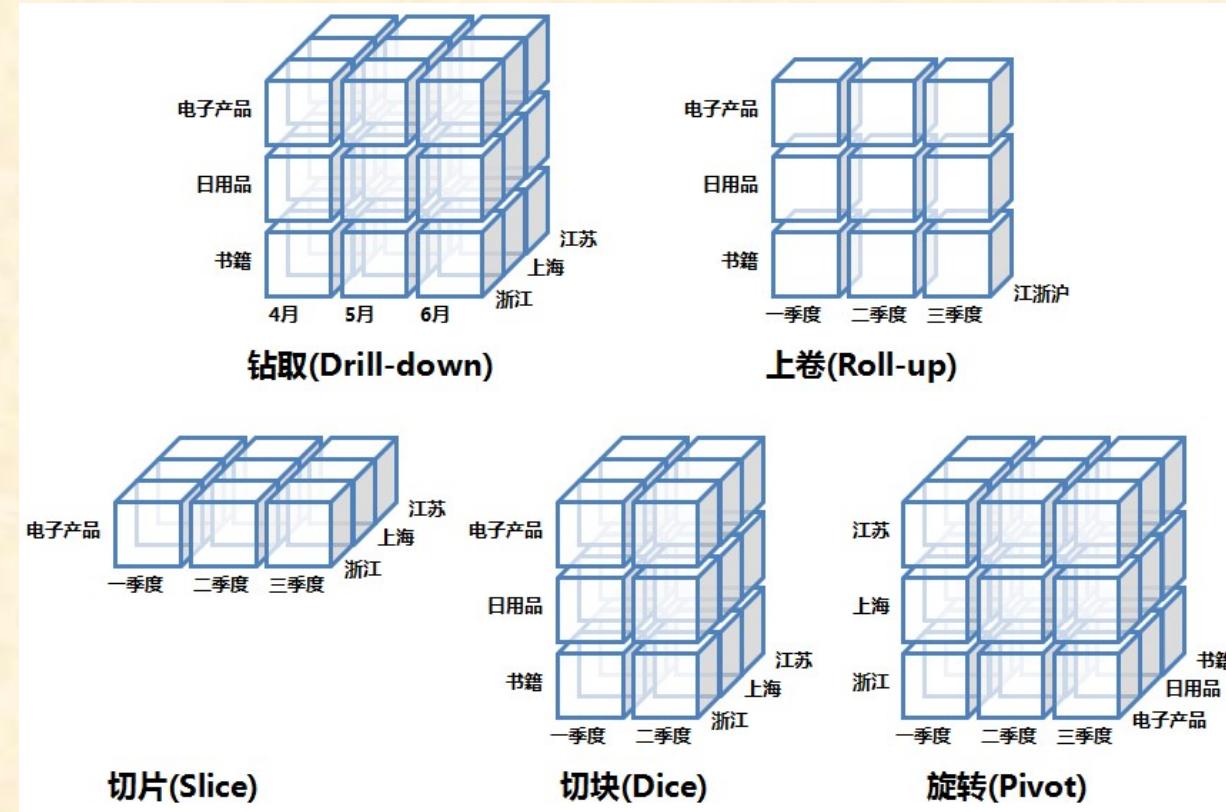
#### ➤ 重新构造

在当前定义基础上完全重新构造立方体，然后重新计算它的数据。

## 7.4 数据立方体与超立方体

### ➤ 多维数据分析

OLAP 的多维分析操作包括：钻取(Drill-down)、上卷( Roll-up )、切片( Slice )、切块( Dice )以及旋转( Pivot )，使最终用户能从多个角度、多个侧面地观察数据，从而深入地了解被包含在数据中的信息、内涵。



## 7.4 数据立方体与超立方体

### ➤ 钻取 (Drill-down)

在维的不同层次间的变化，从上层降到下一层，或者说是将汇总数据拆分到更细节的数据。比如通过对 2017 年第二季度的总销售数据进行钻取来查看 2017 年第二季度 4、5、6 每个月的消费数据

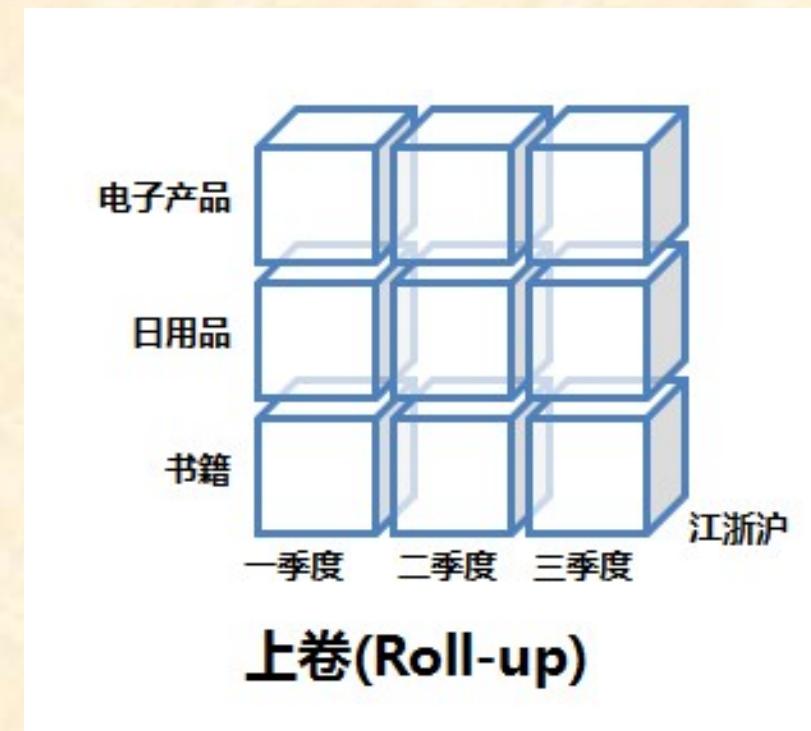


如上图；当然也可以钻取浙江省来查看杭州市、宁波市、温州市…… 这些城市的销售数据。

## 7.4 数据立方体与超立方体

### ➤ 上卷 (Roll-up)

钻取的逆操作，即从细粒度数据向高层的聚合，如将江苏省、上海市和浙江省的销售数据进行汇总来查看江浙沪地区的销售数据



## 7.4 数据立方体与超立方体

### ➤ 切片 (Slice)

选择维中特定的值进行分析，比如只选择电子产品的销售数据，或者 2017 年第二季度的数据。



## 7.4 数据立方体与超立方体

### ➤ 切块 (Dice)

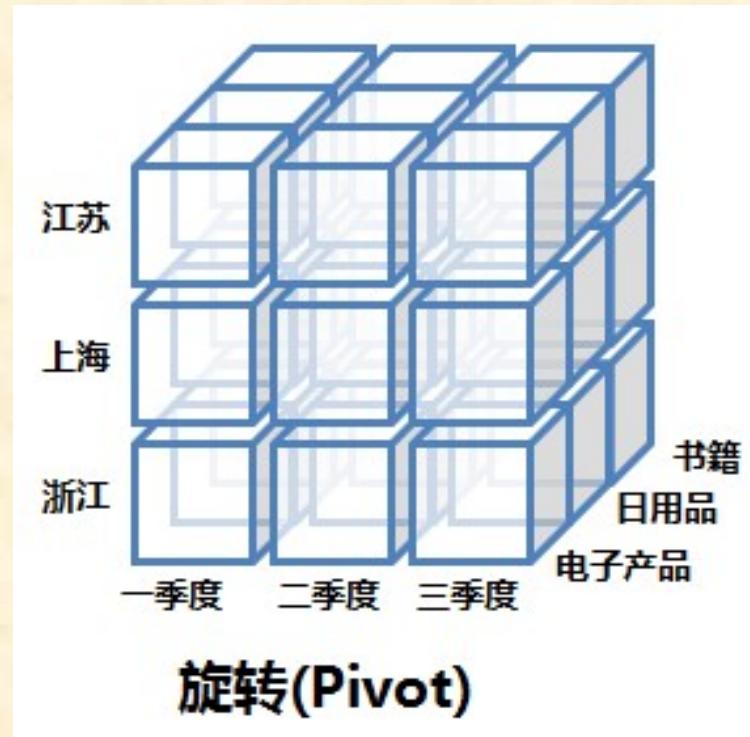
选择维中特定区间的数据或者某批特定值进行分析，比如选择 2010 年第一季度到 2010 年第二季度的销售数据，或者是电子产品和日用品的销售数据。



## 7.4 数据立方体与超立方体

### ➤ 旋转 (Pivot)

即维的位置的互换，就像是二维表的行列转换，如图中通过旋转实现产品维和地域维的互换。



## 7.4 数据立方体与超立方体

**OLAP** 的优势是基于数据仓库面向主题、集成的、保留历史及不可变更的数据存储，以及多维模型多视角多层次的数据组织形式。

### 数据展现方式

基于多维模型让数据的展示更加直观，从多个角度多个层面去发现事物的不同特性。

### 查询效率

多维模型的建立是基于对 OLAP 操作的优化基础上的，这些优化使得对百万千万甚至上亿数量级的运算变得得心应手。

### 分析的灵活性

用上面介绍的各类 OLAP 操作对数据进行聚合、细分和选取，这样提高了分析的灵活性，可以从不同角度不同层面对数据进行细分和汇总，满足不同分析的需求。

# 实例分析

## AllElectronics销售数据数据仓库sales, 记录商店销售

- 涉及维: time, item, branch和location, 记录商品的月销售, 销售商品, 销售的分店和地点.
- 每个维都有一个表与之相关联（维表）, 进一步描述维. 例如, item维可以包含属性item\_name, brand和type.
- 数据仓库的多维数据模型围绕中心主题sales组织.
- sales的事实用数值度量, 包括dollars\_sold, units\_sold和amount\_budgeted.

# 实例分析

□ 观察AllElectronics销售数据中Vancouver每季度销售的商品

- Vancouver的销售用维time(按季度组织)和维item(按所售商品的类型组织)表示.
- 所显示的事实或度量是dollars\_sold(单位: \$1000).

维time, item的2-D视图

---

location = “Vancouver”

---

time(季度)	item(类型)			
	家庭娱乐	计算机	电话机	安全设备
Q1	605	825	14	400
Q2	680	952	31	512
Q3	812	1023	30	501
Q4	927	1038	38	580

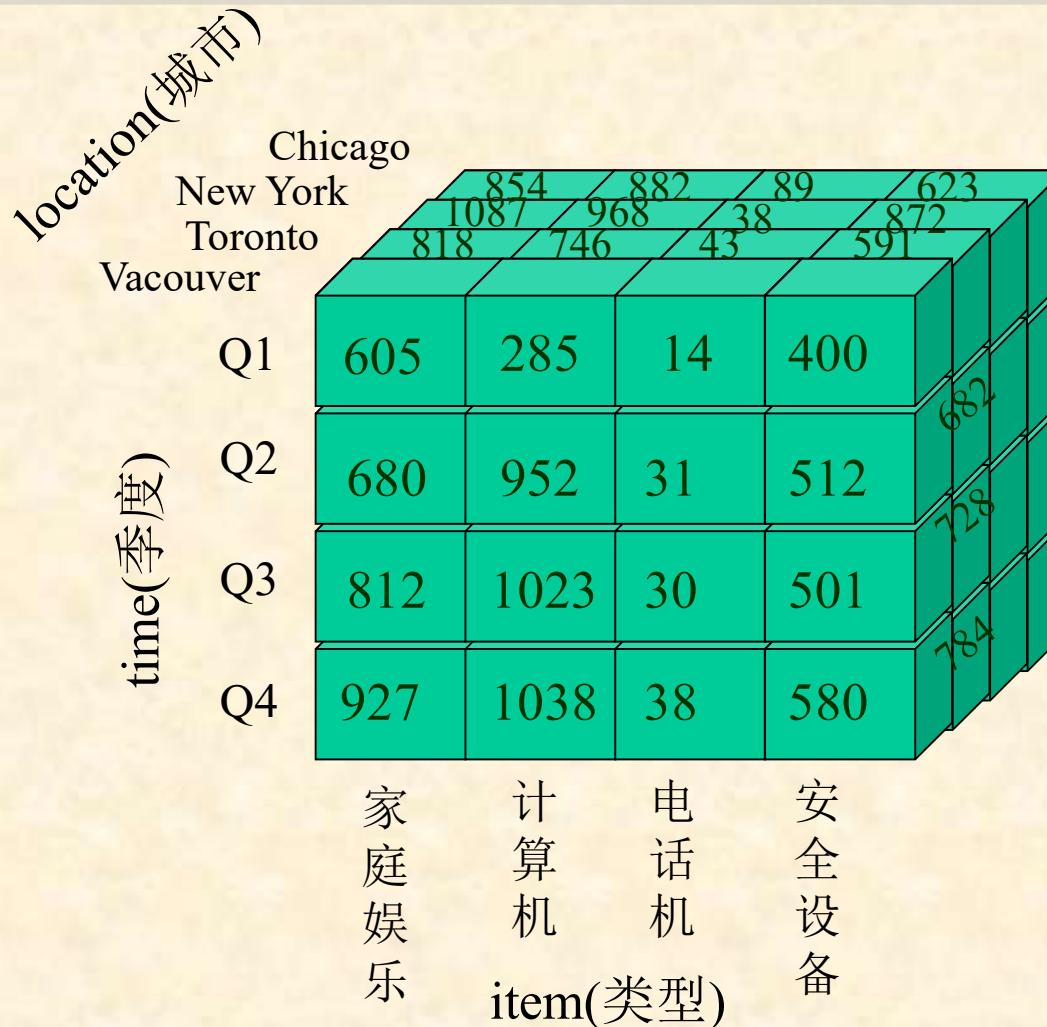
---

# 实例分析

AllElectronics的销售数据按照维time, item,location的3-D视图  
(所显示的事实或度量是dollars\_sold)

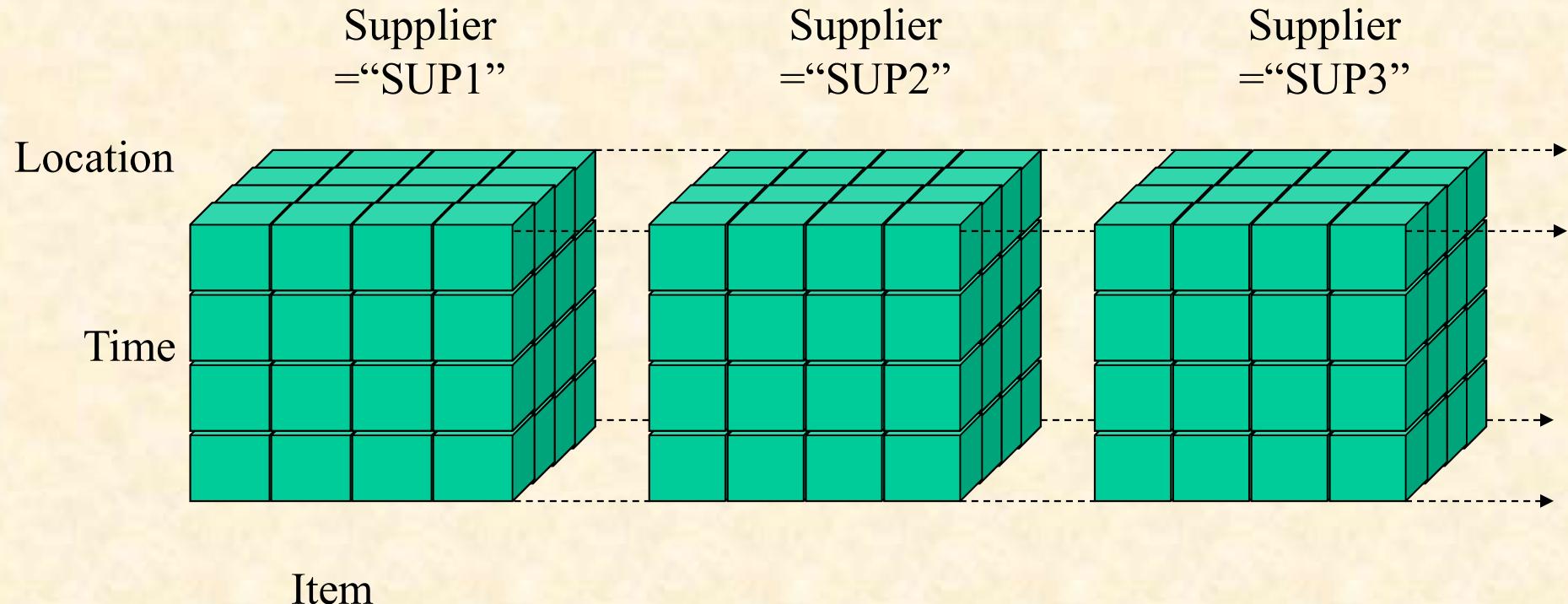
time	Location = “Chicago”				Location = “New York”				Location = “Toronto”			
	家庭娱乐	计算机	电话机	安全设备	家庭娱乐	计算机	电话机	安全设备	家庭娱乐	计算机	电话机	安全设备
Q1	854	882	89	623	1087	968	38	872	818	746	43	591
Q2	943	890	64	689	1130	1024	41	935	894	769	52	682
Q3	1032	924	59	798	1034	1048	45	1020	940	795	58	728
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784

# 实例分析



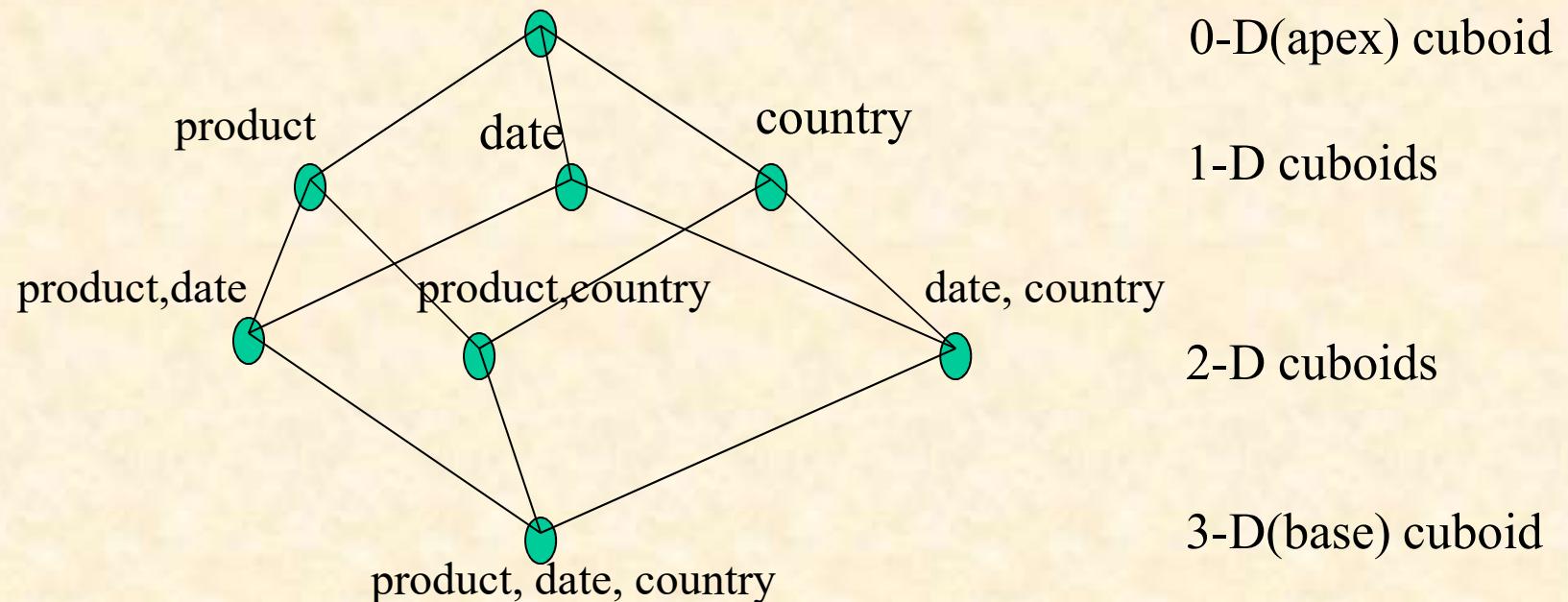
3-D数据立方体, 维是time, item, location, 所显示的度量为  
Dollars\_sold (单位: \$1000)

# 实例分析

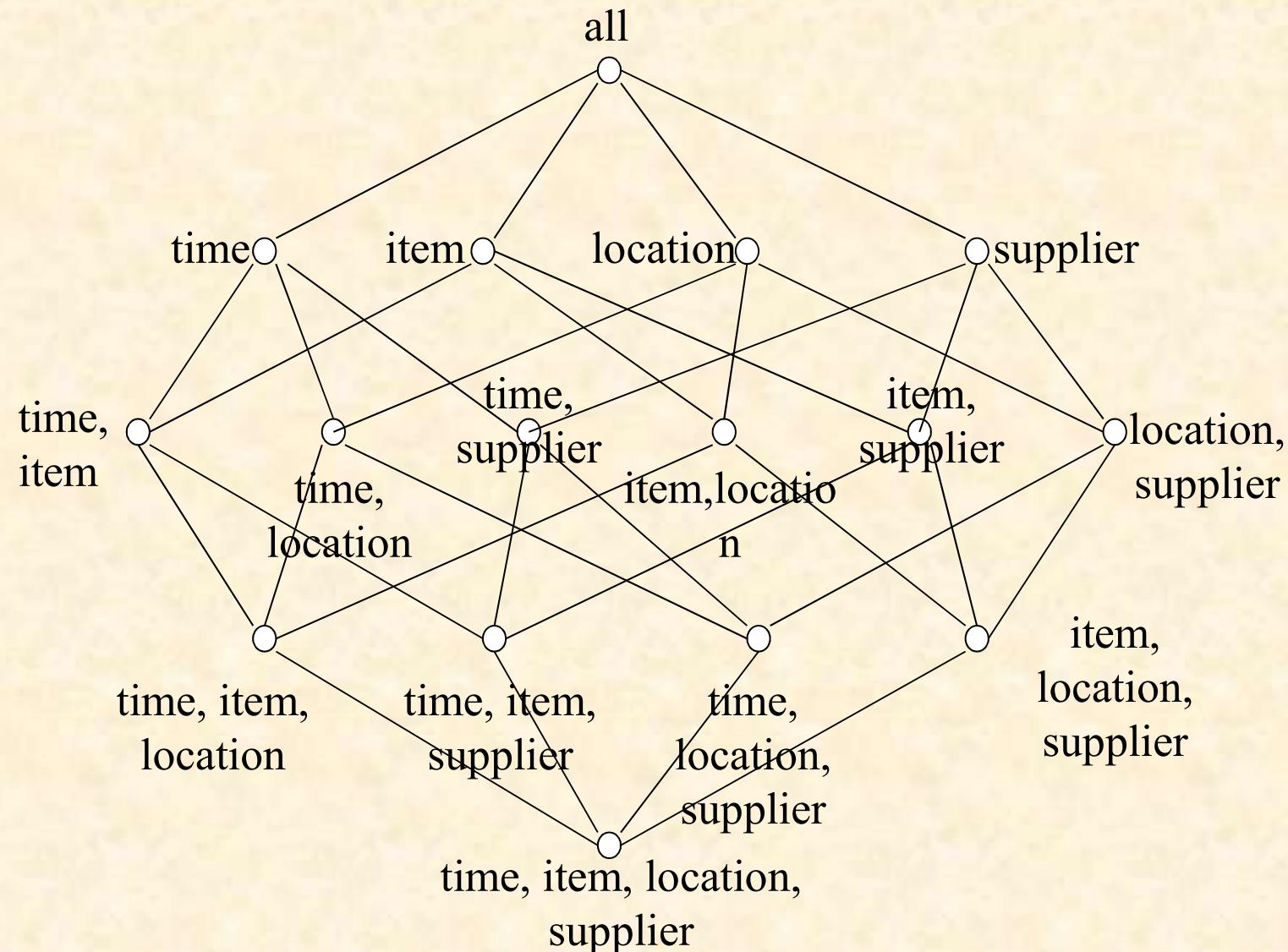


销售数据的4-D数据立方体表示, 维是time, item, location, supplier,  
所显示的度量为dollars\_sold (单位: \$1000)

# Cuboids Corresponding to the Cube



# 实例分析



# 数据立方体如何定义？

- 专门语言，如DMQL

```
define cube sales[item, city, year]: sum(sales_in_dollars)  
compute cube sales
```

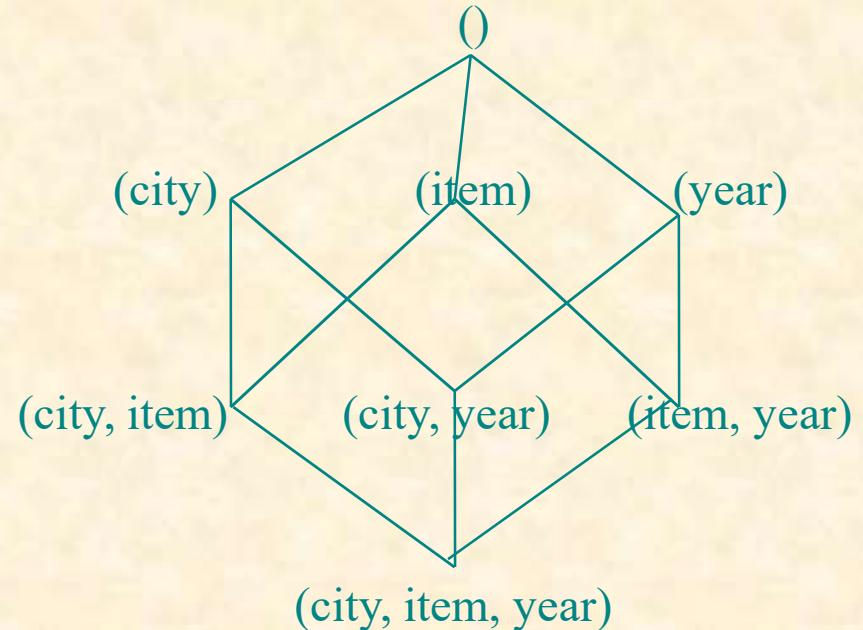
- SQL语言扩展 (例如，引入新operator)

```
SELECT item, city, year, SUM(amount)
```

```
FROM SALES
```

```
CUBE BY item, city, year
```

(introduced by Gray et al.'96)



# DMQL

- DMQL: A Data Mining Query Language for Relational Databases (Han et al, Simon Fraser University)

- CUBE + DIMENSION

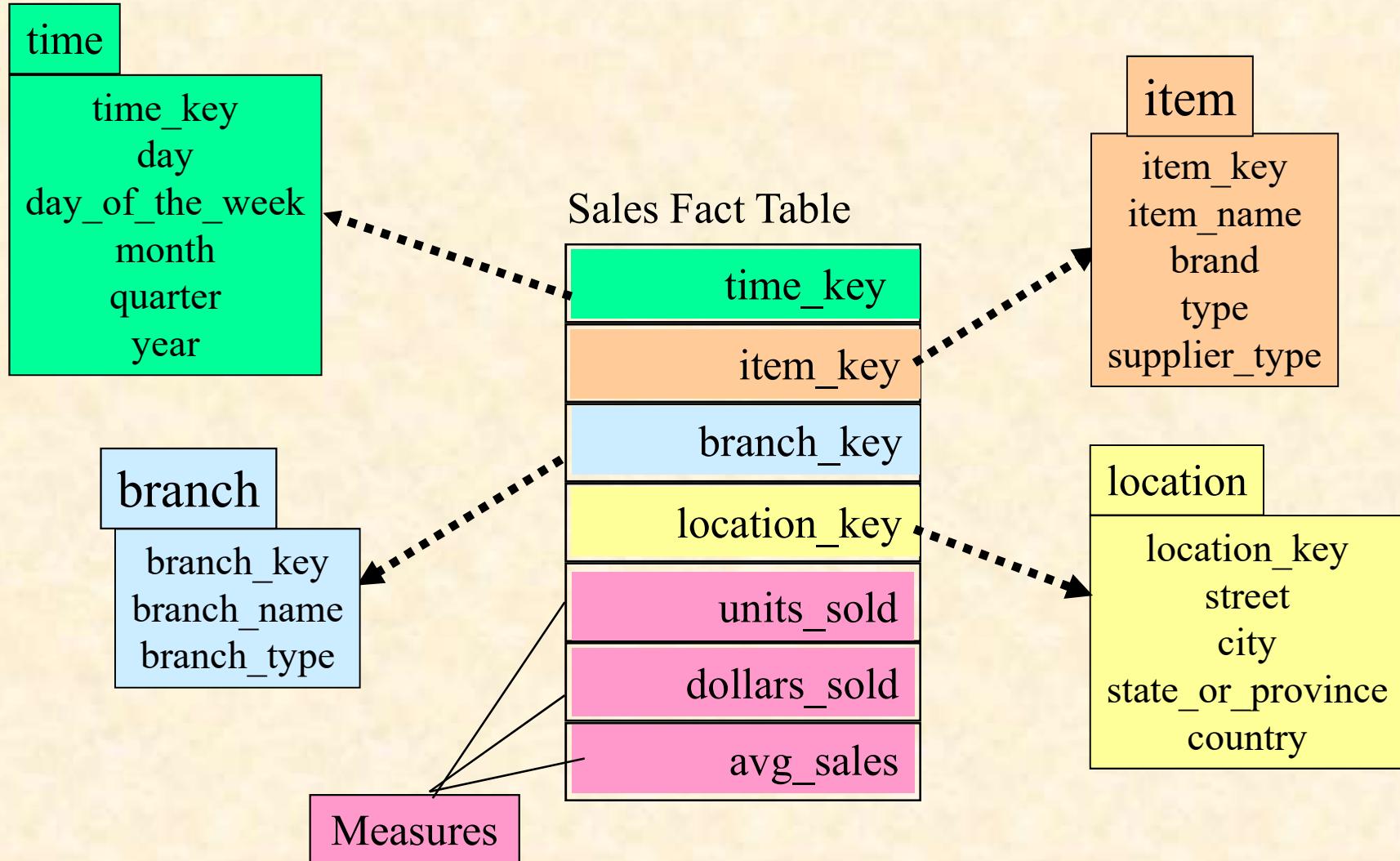
- Cube Definition (Fact Table)

```
define cube <cube_name> [<dimension_list>]:  
    <measure_list>
```

- Dimension Definition (Dimension Table)

```
define dimension <dimension_name> as  
    (<attribute_or_subdimension_list>)
```

# 星型模式



# DMQL 定义

```
define cube sales_star [time, item, branch, location]:  
    dollars_sold = sum(sales_in_dollars), units_sold = count(*)
```

```
define dimension time as (time_key, day, day_of_week,  
month, quarter, year)
```

```
define dimension item as (item_key, item_name, brand,  
type, supplier_type)
```

```
define dimension branch as (branch_key, branch_name,  
branch_type)
```

```
define dimension location as (location_key, street, city,  
province_or_state, country)
```

## 执行方式： 创建关系表

**time(time\_key,day\_of\_week,month,quater,year)**

**item(item\_key,item\_name,brand,type,supplier\_type)**

**branch(branch\_key,branch\_name,branch\_type)**

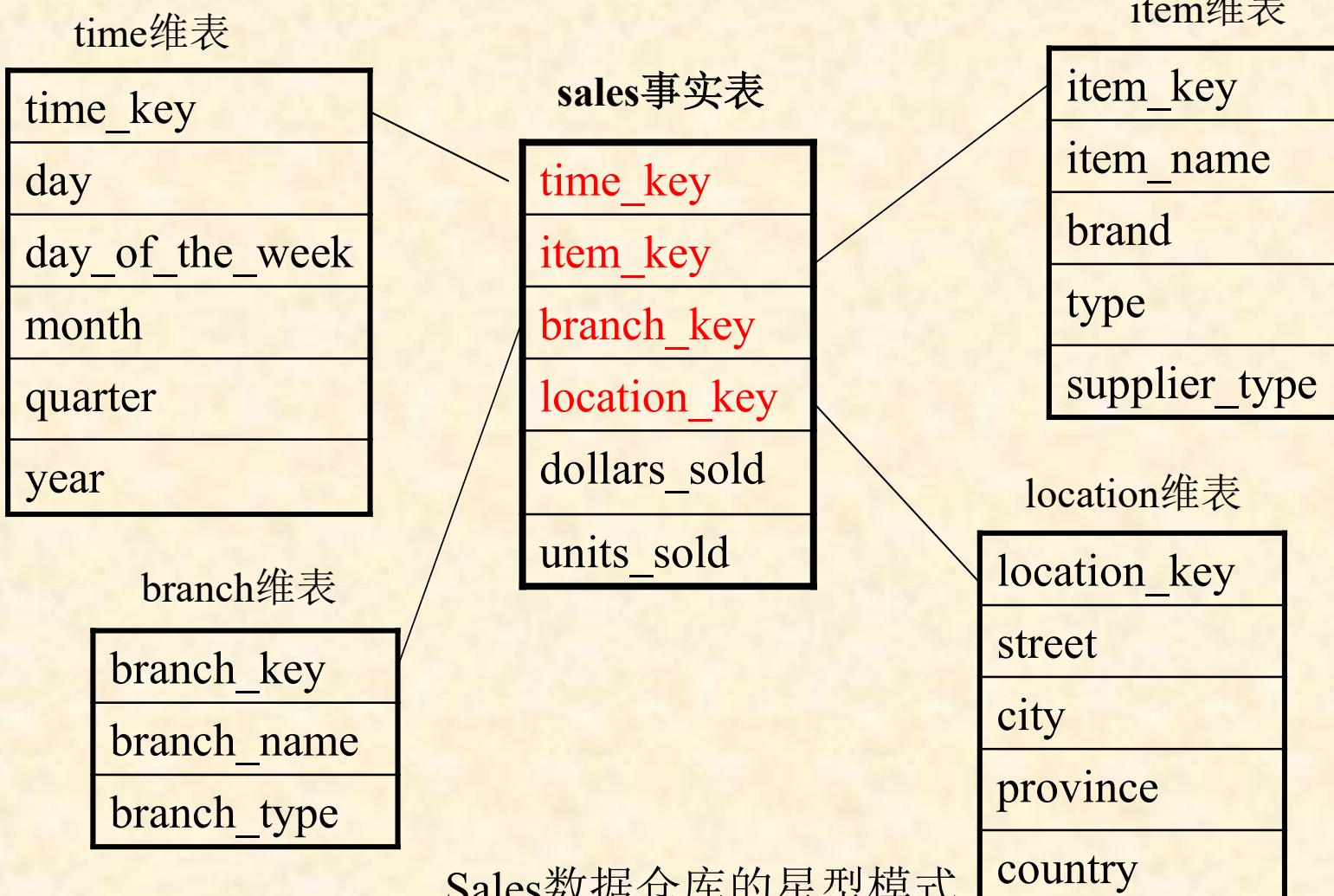
**location(location\_key,street,city,province\_or\_state,country)**

**sales(time\_key,item\_key,branch\_key,location\_key,number\_of\_units\_sold, price)**

## 执行方式：解释为SQL语句执行

```
SELECT s.time_key,s.item_key,s.branch_key,s.location_key,  
      SUM(s.number_of_units_sold*s.price),  
      SUM(s.number_of_units_sold)  
FROM time t, item i, branch b, location l, sales s,  
WHERE s.time_key=t.time_key AND s.item_key=i.item_key  
AND s.branch_key=b.branch_key AND  
s.location_key=l.location_key  
GROUP BY (s.time_key,s.item_key,s.branch_key,s.location_key);
```

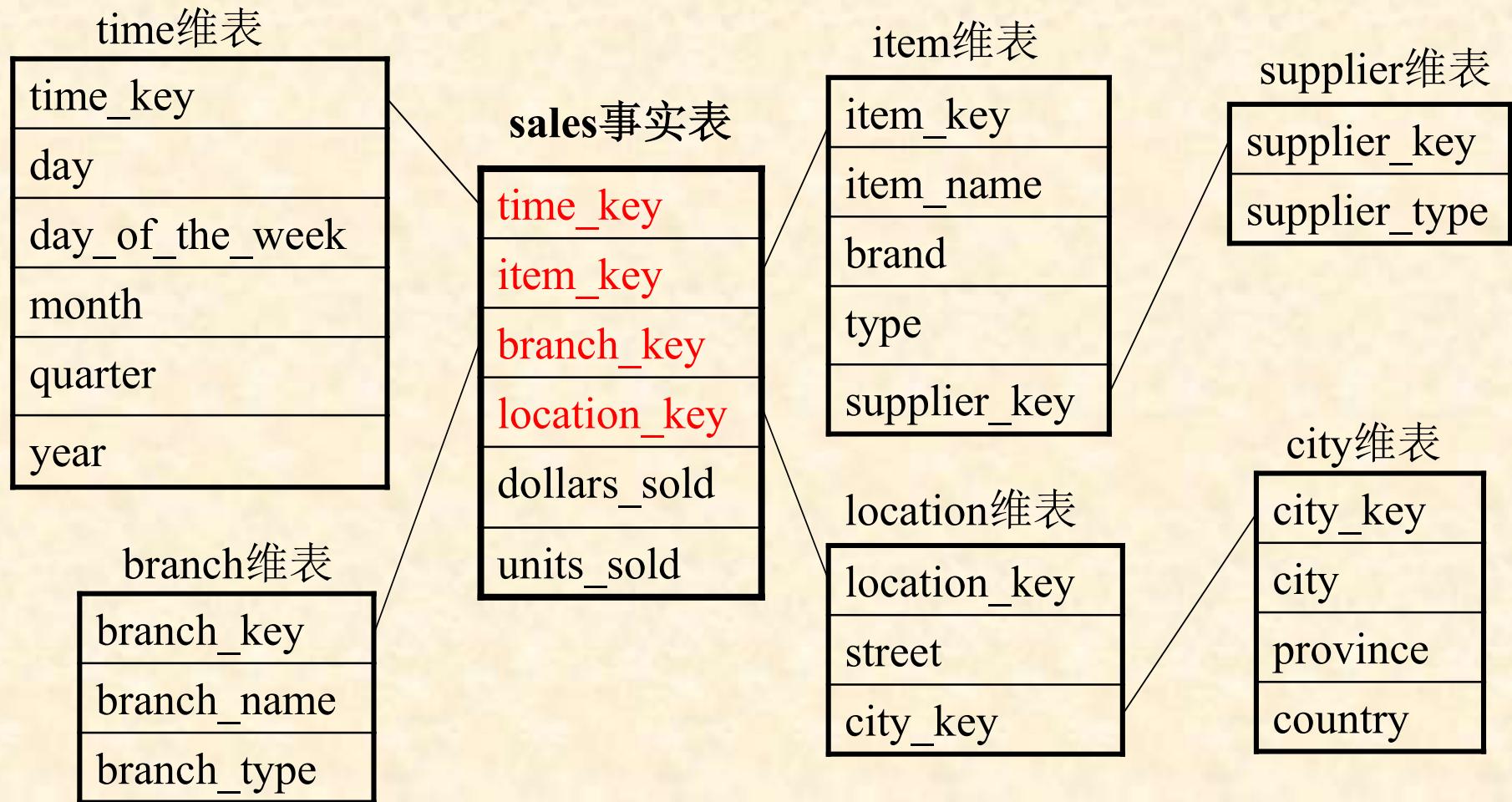
# 实例分析



Sales数据仓库的星型模式

其中包括一个大的包含大批数据和不含冗余的中心表(事实表)和一组小的附属表(维表),每维一个.

# 实例分析



Sales数据仓库的雪花模式(snowflake schema)

与星型模式相比: 雪花模式的维表可能是规范化形式, 以便减少冗余.

# DMQL表示

**define cube** sales\_snowflake [time, item, branch, location]:

**dollars\_sold** = sum(sales\_in\_dollars), **avg\_sales** =  
    **avg(sales\_in\_dollars)**, **units\_sold** = count(\*)

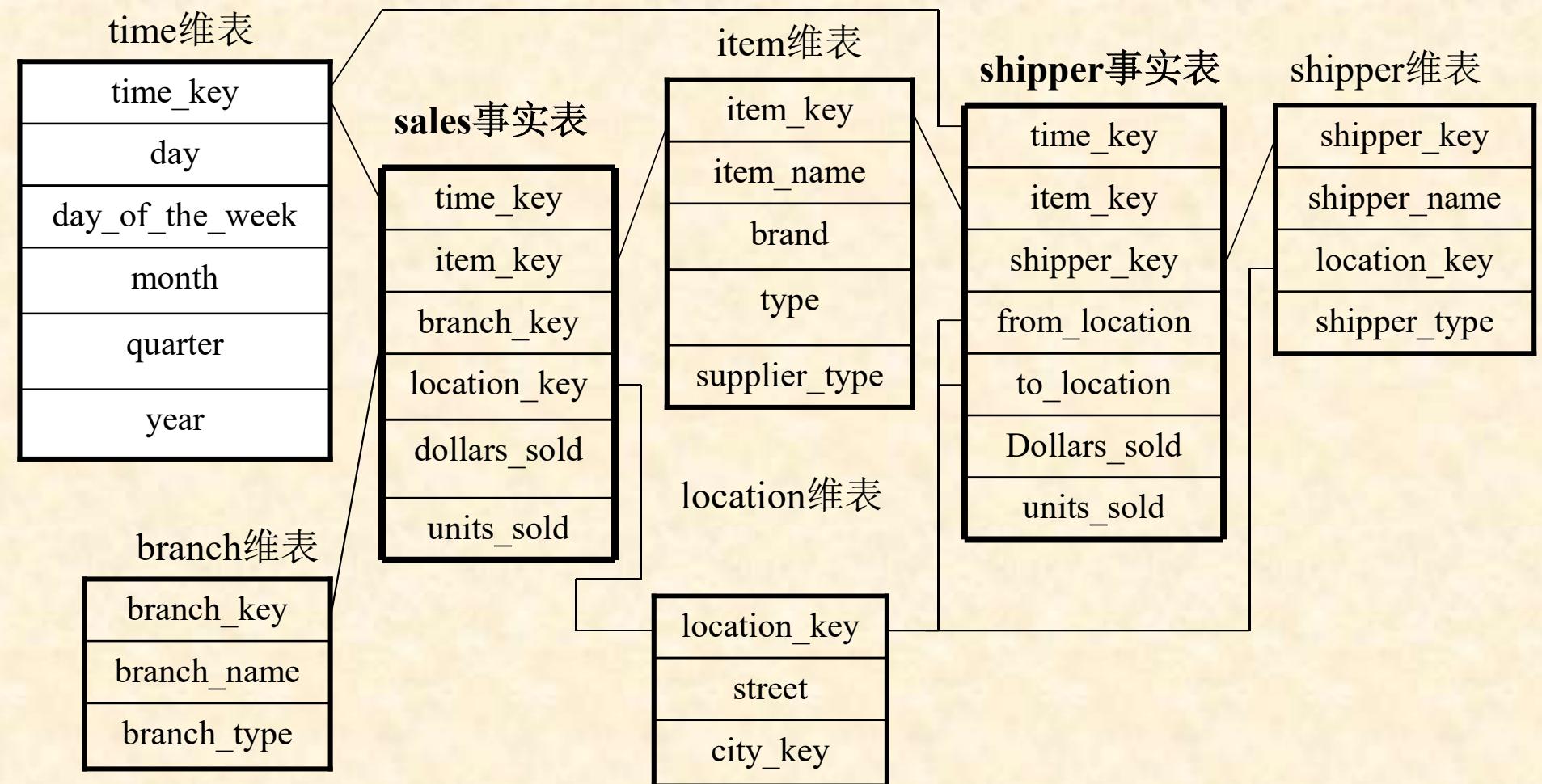
**define dimension** time **as** (time\_key, day, day\_of\_week, month,  
quarter, year)

**define dimension** item **as** (item\_key, item\_name, brand, type,  
supplier(supplier\_key, supplier\_type))

**define dimension** branch **as** (branch\_key, branch\_name,  
branch\_type)

**define dimension** location **as** (location\_key, street, city(city\_key,  
province\_or\_state, country))

# 实例分析



Sales数据仓库的事实星座(fact constellation)

复杂的应用可能需要多个事实表共享维表. 可以被看作是星型模式集.

# DMQL表示

**define cube sales [time, item, branch, location]:**

dollars\_sold = sum(sales\_in\_dollars), avg\_sales = avg(sales\_in\_dollars),  
units\_sold = count(\*)

**define dimension time as (time\_key, day, day\_of\_week, month, quarter, year)**

**define dimension item as (item\_key, item\_name, brand, type, supplier\_type)**

**define dimension branch as (branch\_key, branch\_name, branch\_type)**

**define dimension location as (location\_key, street, city, province\_or\_state,  
country)**

**define cube shipping [time, item, shipper, from\_location, to\_location]:**

dollar\_cost = sum(cost\_in\_dollars), unit\_shipped = count(\*)

**define dimension time as time in cube sales**

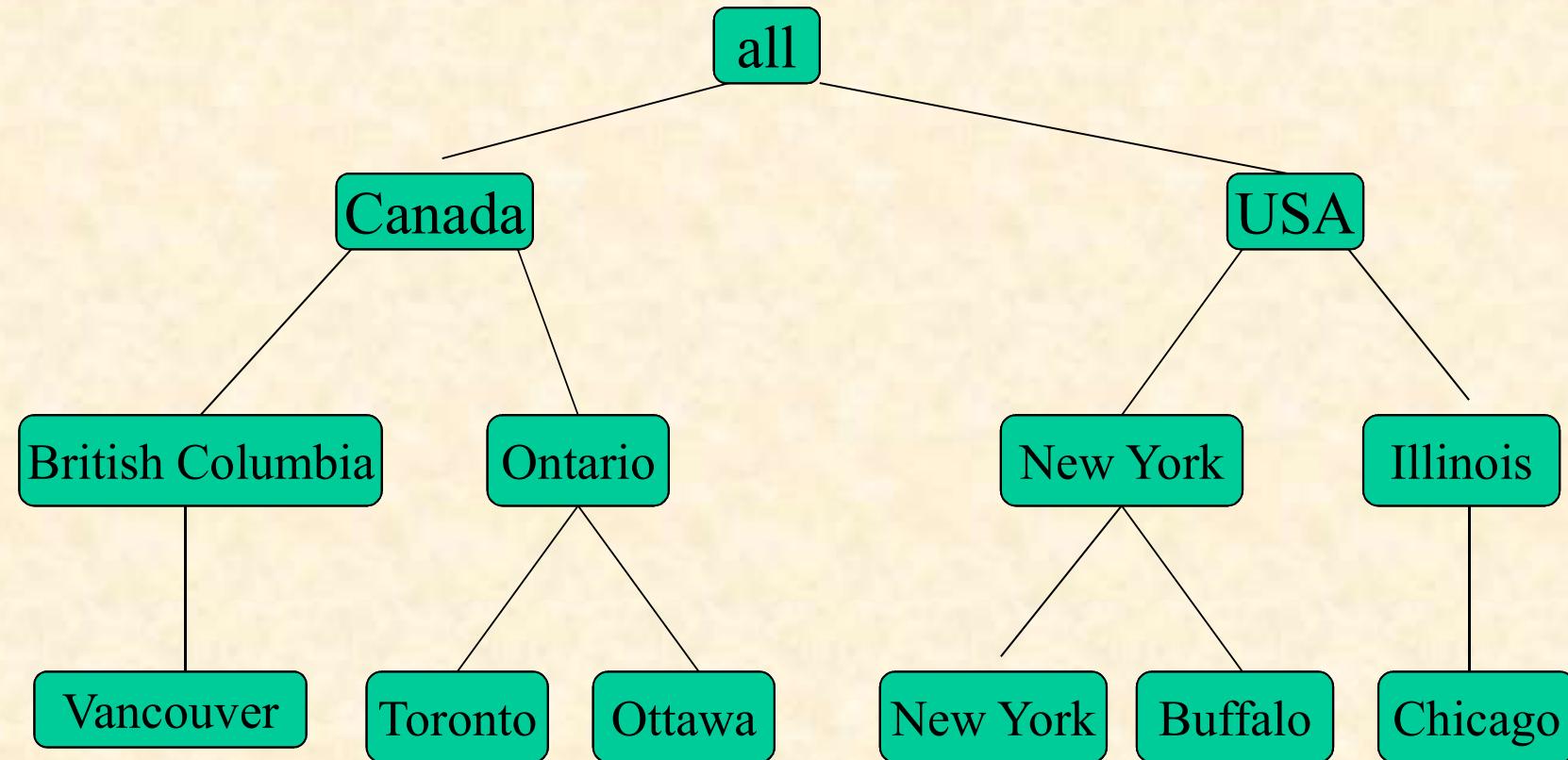
**define dimension item as item in cube sales**

**define dimension shipper as (shipper\_key, shipper\_name, location as location  
in cube sales, shipper\_type)**

**define dimension from\_location as location in cube sales**

**define dimension to\_location as location in cube sales**

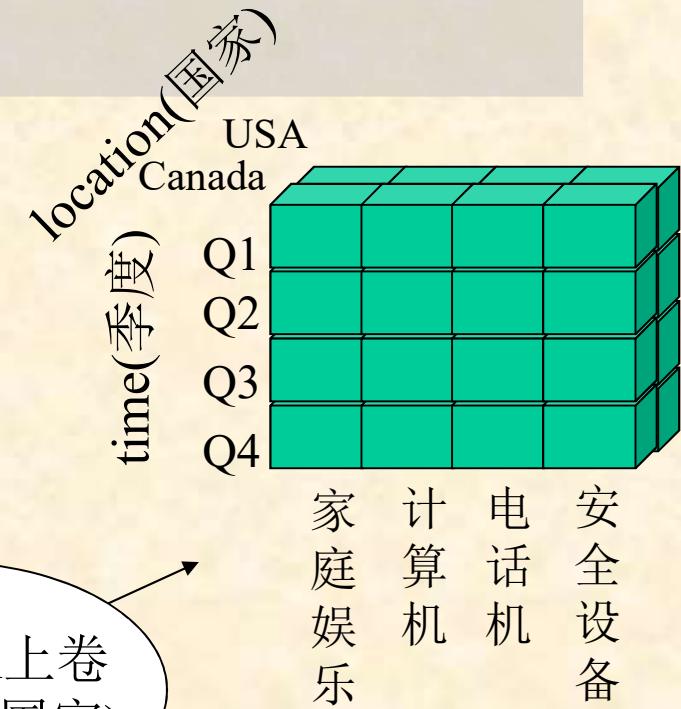
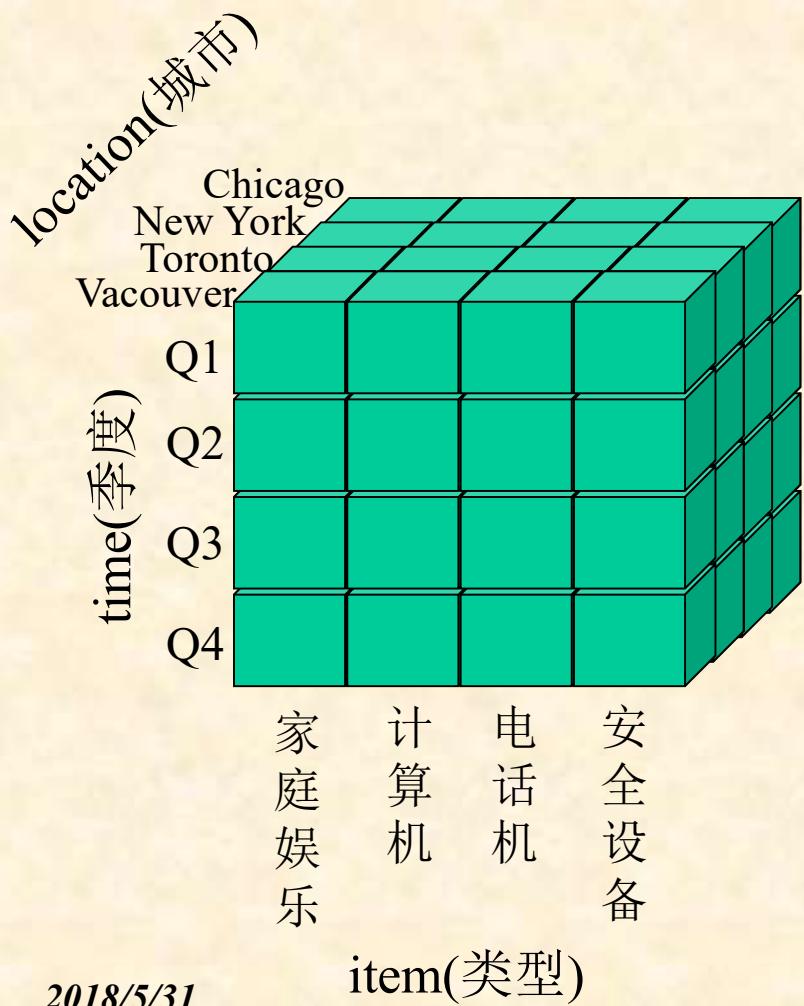
# 实例分析



概念分层(concept hierarchy)

定义一个映射序列, 将低层概念映射到更一般的高层概念.

# 实例分析

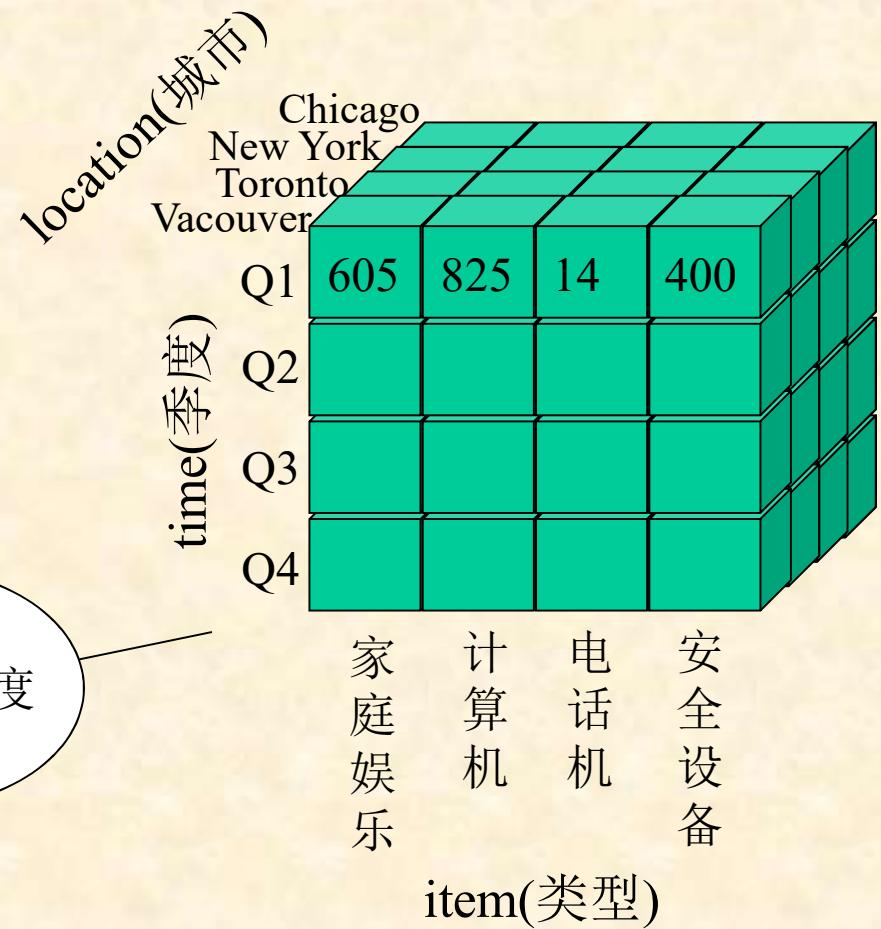


按location上卷  
(从城市到国家)

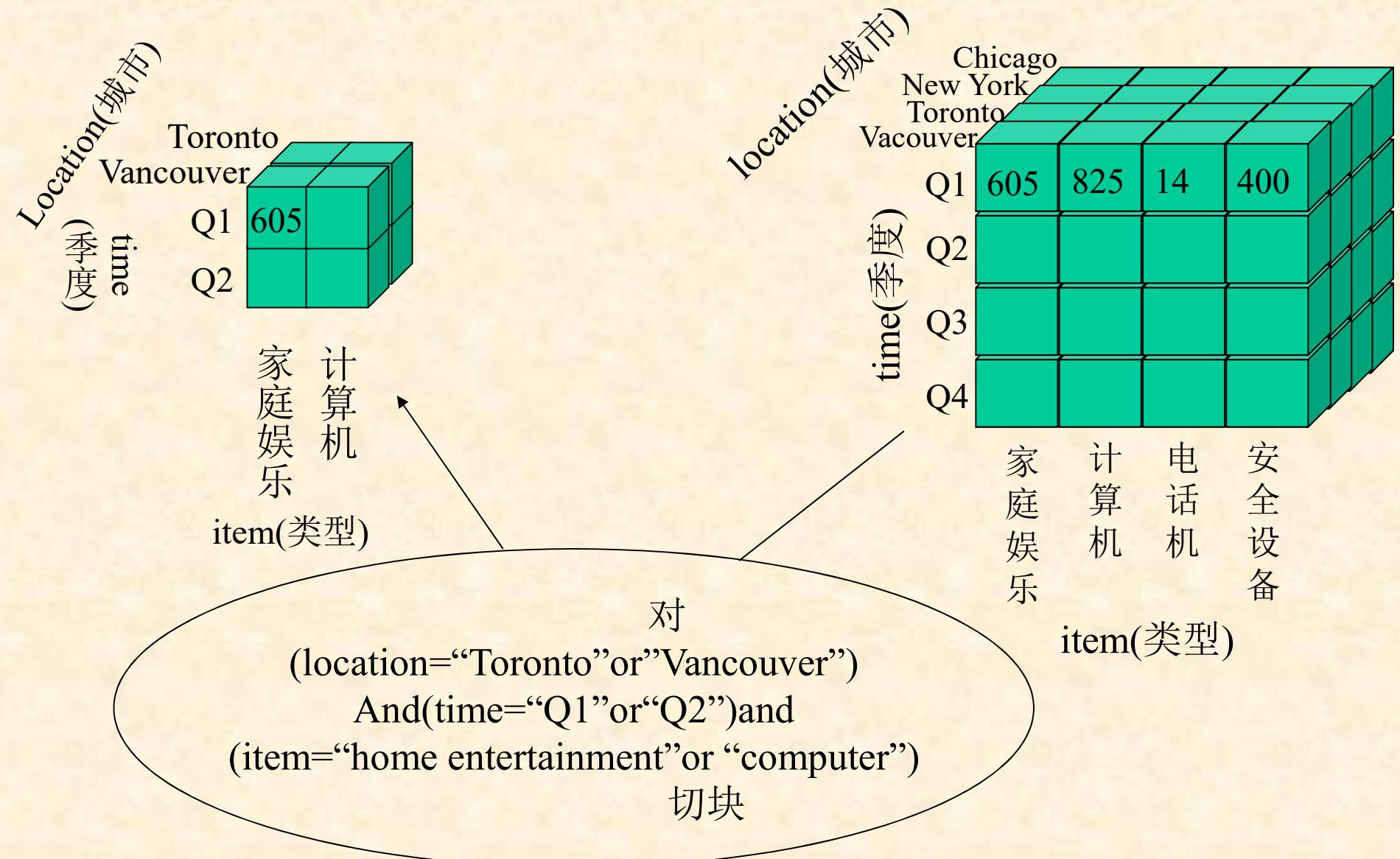
# 实例分析



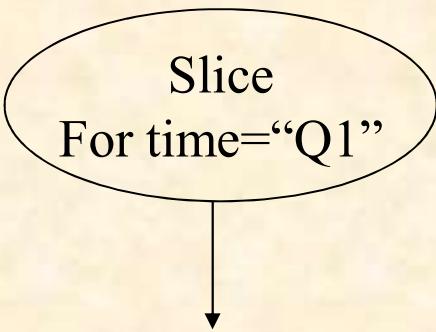
按time  
下钻(从季度  
到月份)



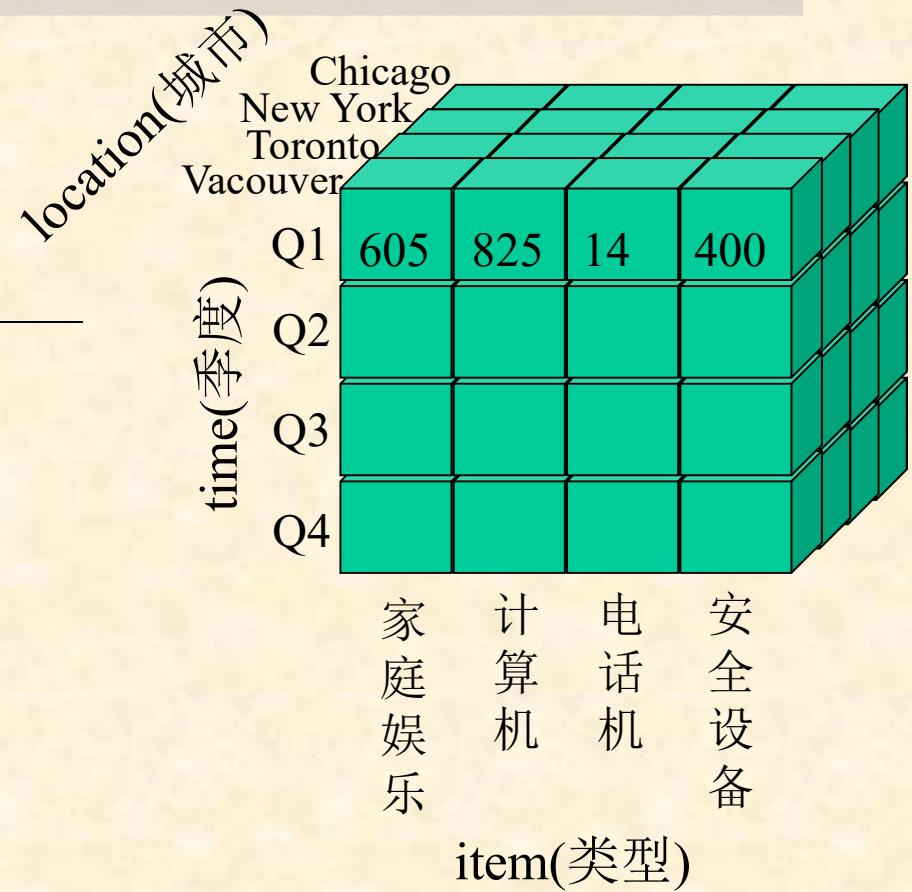
# 实例分析



# 实例分析



		Item(类型)			
		家庭娱乐	计算机	电话机	安全设备
location(城市)	Chicago	605	825	14	440
	New York				
location(城市)	Toronto				
	Vancouver				



# MDX

- Multidimensional Expressions (MDX) as a Language
- 1998年提出，用于OLAP查询

SELECT

{ [Time] . [1997] , [Time] . [1998] } ON

COLUMNS ,

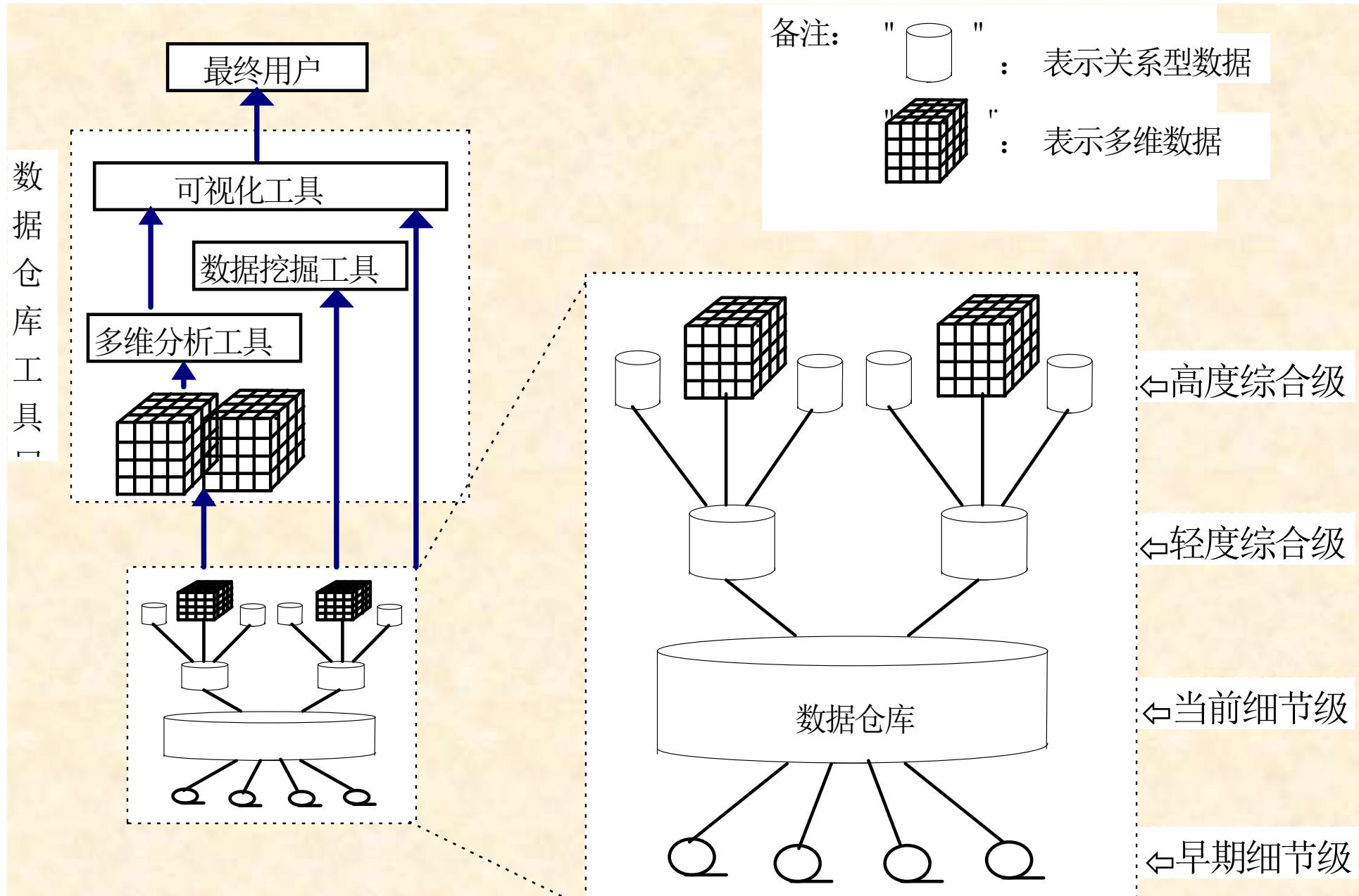
{ [Measures] . [WarehouseSales] ,

[Measures] . [Warehouse Cost] } ON

ROWS

FROM Warehouse

WHERE ([Store]<sup>Data Warehouse</sup> . [All Stores] . [USA] )



# 流行的OLAP工具介绍

## ➤ OLAP产品

- Hyperion Essbase
- Oracle Express
- IBM DB2 OLAP Server
- Sybase Power dimension
- Informix Metacube
- Microsoft Plato
- Brio
- Cognos
- Business Object
- MicroStrategy

## ➤ OLAP产品涉及的业务操作

- 由外部或内部数据源批量装入数据
- 由业务系统增量装入数据
- 沿数据层次汇总数据
- 对基于业务模型的新数据进行计算
- 时间序列分析
- 高复杂的查询
- 沿数据层次细化分析
- 随机查询
- 多个联机会话(多用户同时访问)

# 流行的OLAP工具介绍(续)

## •Hyperion Essbase

- 以服务器为中心的分布式体系结构
- 有超过100个的应用程序
- 有300多个用Essbase作为平台的开发商
- 具有几个计算公式，支持多种计算
- 用户可以自己构件复杂的查询。
- 快速的响应时间，支持多用户同时读写
- 有30多个前端工具可供选择
- 支持多种财务标准
- 能与ERP或其他数据源集成
- 全球用户超过1500家

## ➤ Oracle Express

- Oracle DW支持GB~TB数量级
- 采用类似数组的结构，避免了连接操作，提高分析性能
- 提供一组存储过程语言来支持对数据的抽取
- 用户可通过Web和电子表格使用
- 灵活的数据组织方式，数据可以存放在Express Server内，也可直接在RDB上使用
- 有内建的分析函数和4GL用户自己定制查询
- 全球超过3000家



# 流行的OLAP工具介绍(续)

## ➤ IBM DB2 OLAP Server

- 把Hyperion Essbase的OLAP引擎和DB2的关系数据库集成在一起
- 与Essbase API完全兼容
- 数据用星型模型存放在关系数据库DB2中

## ➤ Informix Metacube

- 采用metacube技术，通过OLE和ODBC对外开放，
- 采用中间表技术实现多维分析引擎，提高响应时间和分析能力
- 开放的体系结构可以方便地与其他数据库及前台工具进行集成

## ➤ Sybase Power dimension

- 数据垂直分割（按“列”存储）
- 采用了突破性的数据存取方法-----bit-wise索引技术
- 在数据压缩和并行处理方面有多到之处
- 提供有效的预连接（Projection）技术

# OLAP常见框架

## ➤ Presto:

Facebook 开源的基于Java的分布式数据查询框架，原生集成了 Hive、Hbase 和关系型数据库。Presto 背后所使用的执行模式与 Hive 有根本的不同，它没有使用 MapReduce，大部分场景下比 hive 快一个数量级，其中的关键是所有的处理都在内存中完成。

## ➤ Druid:

是一个实时处理时序数据的 OLAP 数据库，因为它的索引首先按照时间分片，查询的时候也是按照时间线去路由索引。

## ➤ Spark SQL:

基于 Spark 平台上的一个 OLAP 框架，本质上也是基于 DAG 的 MPP，基本思路是增加机器来并行计算，从而提高查询速度。

## ➤ Kylin:

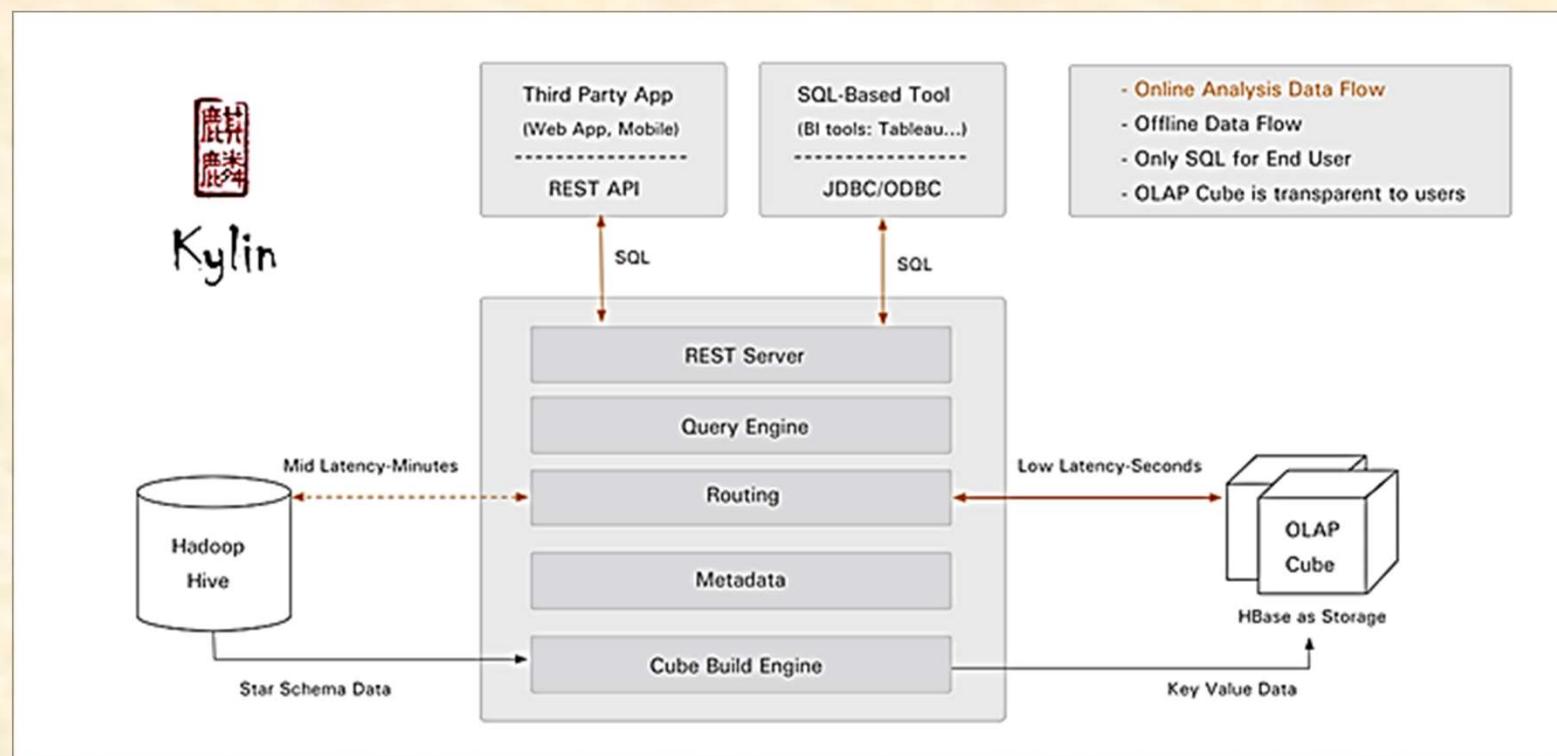
核心是 Cube，cube 是一种预计算技术，基本思路是预先对数据作多维索引，查询时只扫描索引而不访问原始数据从而提速。

# Apache Kylin 框架

作为一个开源的分布式分析引擎，是 Hadoop 生态圈具备海量数据超高查询性能的 OLAP 组件。



Apache Kylin™

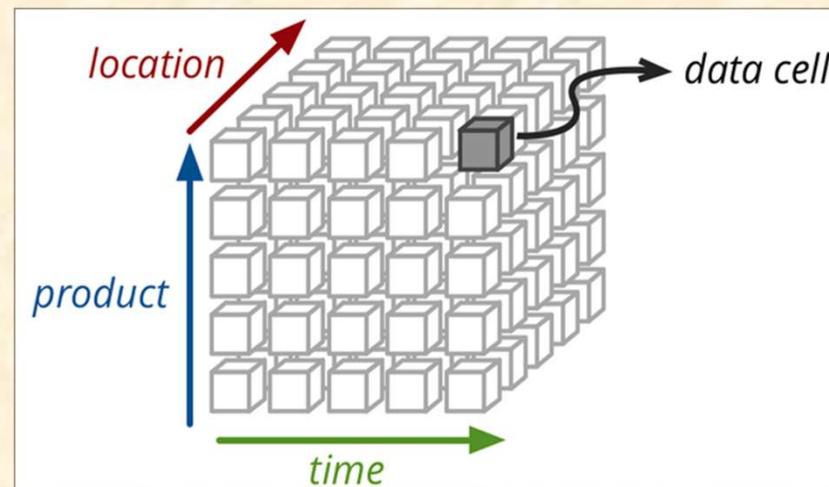


如上图所示，Kylin 从 Hadoop Hive 中获取数据，然后经过 Cube Build Engine，将 Hive 中的数据 Build 成一个 OLAP Cube 保存在 HBase 中。用户执行 SQL 查询时，通过 Query 引擎，将 SQL 语句解析成 OLAP Cube 查询，然后将结果返回给用户。

# Apache Kylin Cube 的构建过程

## ➤ Cube 的物理模型

Apache Kylin 将所有（时间、地点、产品）的各种组合实现算出来，data cell 中存放度量，其中每一种组合都称为 cuboid。给 n 维的数据最多有 $2^n$ 个 cuboid，不过 Kylin 通过设定维度的种类，可以减少 cuboid 的数目。

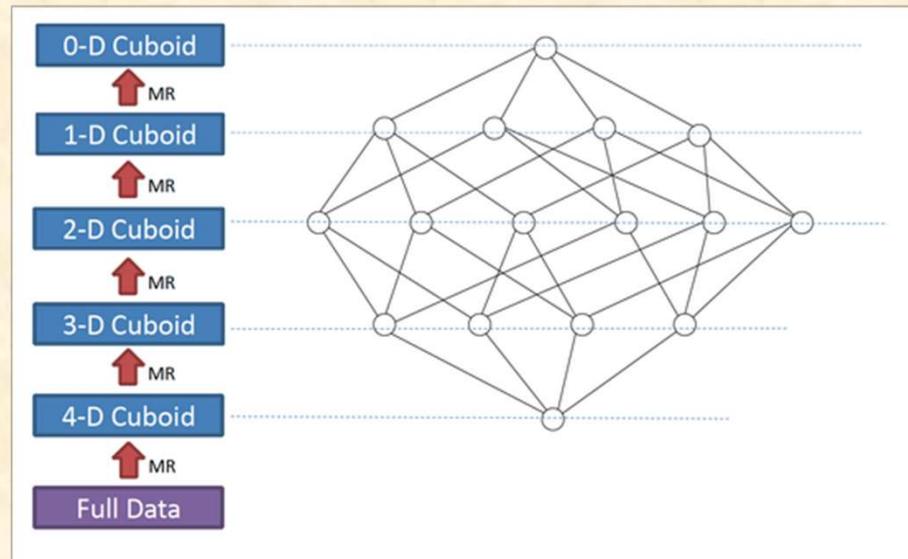


如上图所示，一个常用的 3 维立方体，包含：时间、地点、产品。假如 data cell 中存放的是产量，则我们可以根据时间、地点、产品来确定产量，同时也可根据时间、地点来确定所有产品的总产量等。

# Apache Kylin Cube 的构建过程

## ➤ Cube 构建算法：逐层算法

一个  $N$  维的 Cube，是由 1 个  $N$  维子立方体、 $N$  个  $(N - 1)$  维子立方体、 $N(N - 1)/2$  个  $(N - 2)$  维子立方体、……、 $N$  个 1 维子立方体和 1 个 0 维子立方体构成，总共有  $2^N$  个子立方体组成，在逐层算法中，按维度数逐层减少来计算，每个层级的计算（除了第一层，它是从原始数据聚合而来），是基于它上一层级的结果来计算的。

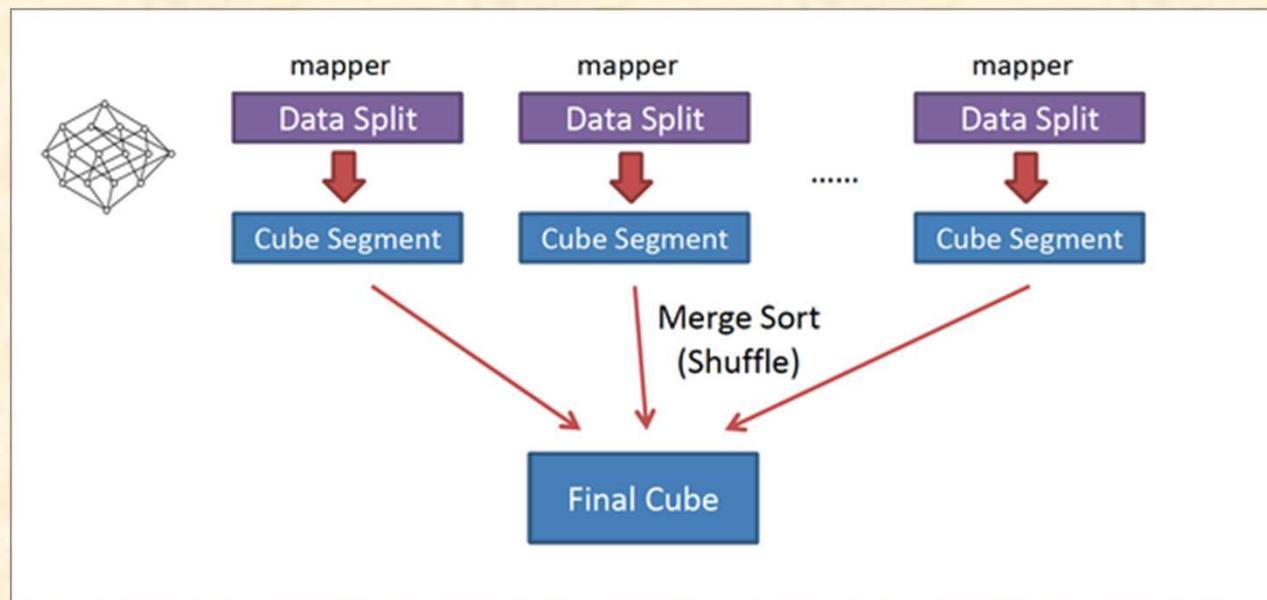


比如，[Group by A, B] 的结果，可以基于 [Group by A, B, C] 的结果，通过去掉 C 后聚合得来的；这样可以减少重复计算；当 0 维度 Cuboid 计算出来的时候，整个 Cube 的计算也就完成了。

# Apache Kylin Cube 的构建过程

## ➤ Cube 构建算法：快速 Cube 算法

该算法的主要思想是，对 Mapper 所分配的数据块，将它计算成一个完整的小 Cube 段（包含所有 Cuboid）；每个 Mapper 将计算完的 Cube 段输出给 Reducer 做合并，生成大 Cube，也就是最终结果；

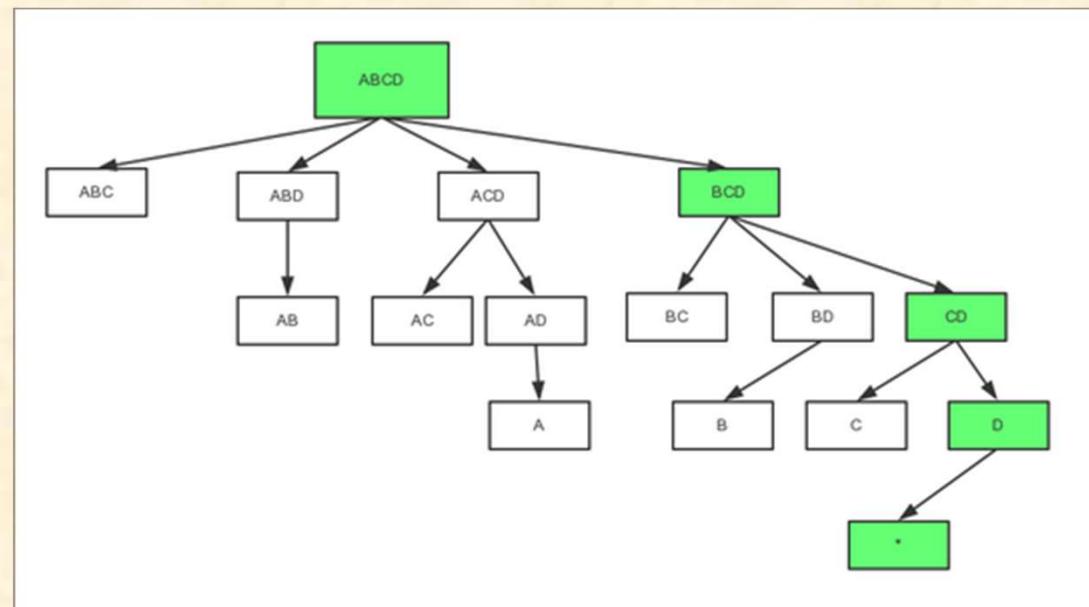


在快速 Cube 算法中，Mapper 会按深度优先遍历（Depth First Search）来计算各个 Cuboid。将父 Cuboid 压栈以计算子 Cuboid，直到没有子 Cuboid 需要计算时才出栈并输出给 Hadoop。

# Apache Kylin Cube 的构建过程

## ➤ Cube 构建算法：快速 Cube 算法

下图是一个四维 Cube 的完整生成树；按照 DFS 的次序，在 0 维 Cuboid 输出前的计算次序是 ABCD -> BCD -> CD -> D ->，ABCD, BCD, CD 和 D 需要被暂存；在被输出后，D 可被输出，内存得到释放；在 C 被计算并输出后，CD 就可以被输出；ABCD 最后被输出。



采用深度优先遍历，兼顾了 CPU 和内存：  
从父 Cuboid 计算子 Cuboid，避免重复计算；  
只压栈当前计算的 Cuboid 的父 Cuboid，减少内存占用。

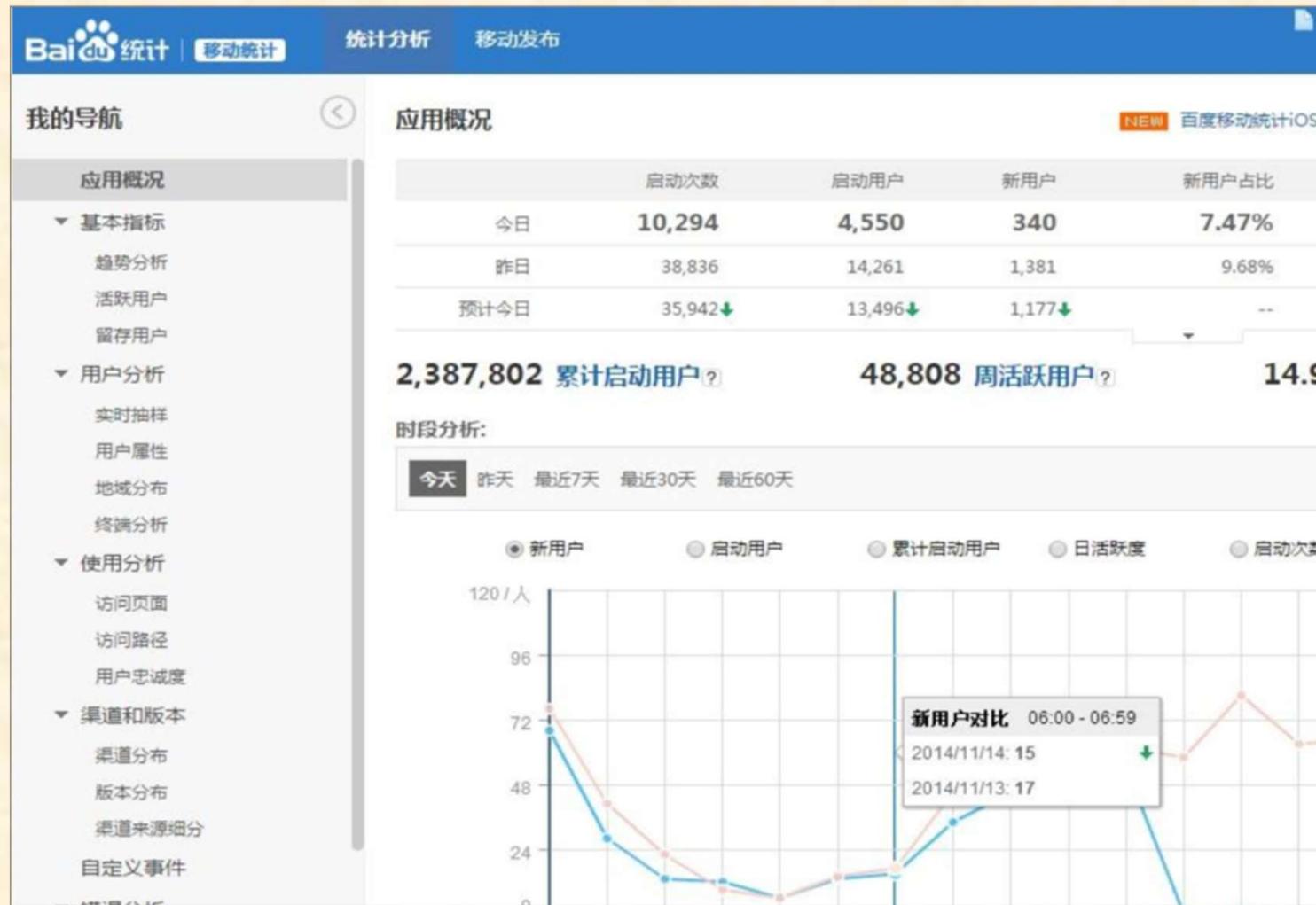
# OLAP发展

## ➤应用领域

- 市场和销售分析(**Marketing and Sales analysis**)
- 电子商务分析(**Clickstream analysis**)
- 基于历史数据的营销(**Database marketing**)
- 预算(**Budgeting**)
- 财务报告与整合(**Financial reporting and consolidation**)
- 管理报告(**Management reporting**)
- 利益率分析(**Profitability analysis**)
- 质量分析(**Quality analysis**)

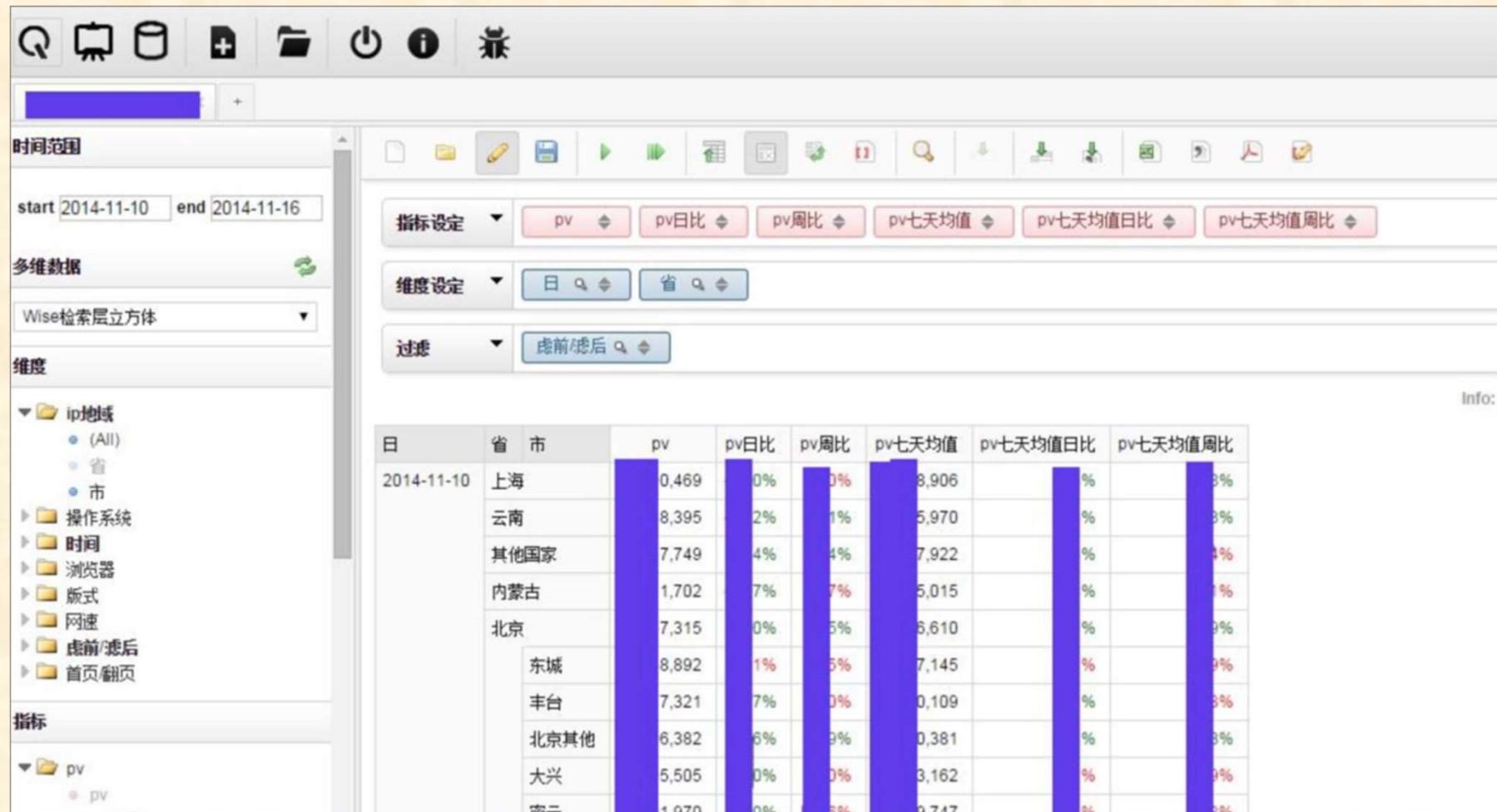
# OLAP发展

## ➤ OLAP应用-在线报表



# OLAP发展

## ➤ OLAP应用-多维分析



# OLAP发展

## ➤ OLAP商用产品

产品	简介	技术特点	收购情况
Netezza	2000年在美国成立 Netezza TwinFin	✓ 软硬一体机 ✓ 采用FPGA数据过滤代替索引	2010年9月20日，IBM出资17.8亿美元收购
Greenplum	2003年在美国成立 Greenplum Database	✓ 行存 + 列存 ✓ Shared-Nothing集群	2010年7月6日，EMC出资3亿美元收购
Vertica	2005年在美国成立 Vertica Analytic Database	✓ 列存 ✓ Shared-Nothing集群	2011年2月，HP出资3.5亿美元收购
Aster Data	2005年在美国成立 nCluster	✓ SQL-MapReduce ✓ Shared-Nothing集群	2011年7月6日，Teradata出资2.63亿美元收购
ParAccel	2005年在美国成立 PADB	✓ 列存 + 自适应压缩 ✓ Shared-Nothing集群	2013年Actian出资1.5亿美元收购， Redshift宣称使用ParAccel

Vendor and Appliance	Memory (GB)	Total Cores	Compression	User Storage (TB, Compressed)	List Price
EMC Greenplum Data Computing Appliance	768	48	4 to 1	144	\$2,000,000
IBM PureData System for Analytics N1001-010	n/a	112	4 to 1	128	\$1,599,000
Microsoft SQL Server 2012 Parallel Data Warehouse <sup>1</sup>	2,304	144	5 to 1	340	\$1,569,970
Oracle Exadata Database Machine X3-2	2,048	128	10 to 1	450	\$13,580,000
Teradata Data Warehouse Appliance 2690	768	96	4 to 1	146	\$1,168,000

# OLAP发展

## ➤ 开源社区及企业

**Hortonworks**

The Stinger Initiative: Making Apache Hive 100 Times Faster

February 20th, 2013 | Alan Gates

**cloudera** Ask Bigger Questions

WHY CLOUDERA PRODUCTS SOLUTIONS PARTNERS RESOURCES SUPPORT ABOUT

Hadoop & Big Data

Cloudera Impala: Real-Time Queries in Apache Hadoop, For Real

by Marcel Komacić & Justin Erickson | October 24, 2012 | 14 comments | Tweet

**Apache Drill** Distributed system for interactive analysis.

Apache Drill (*incubating*) is a distributed system for interactive analysis of large-scale datasets, based on Google's Dremel. Its goal is to efficiently process nested data. It is a design goal to scale to 10,000 servers or more and to be able to process petabytes of data and trillions of records in seconds.

**MemSQL, The Real-Time Analytics Platform.**

MemSQL's real-time analytics platform is built on the world's fastest, most scalable in-memory database, capable of simultaneously handling real-time transactions and analytic workloads. MemSQL unleashes the full potential of Big Data by consuming and returning data instantly.

**Spark SQL**

Download Libraries Documentation Examples Community FAQ

Spark SQL is Spark's module for working with structured data.

2018/5/31

Search Images Maps Play YouTube News Gmail Documents More · Andrew Brust ·

**Google bigquery**

Compose Query

Query History Job History

BigQuery Sandbox

- MyDataset
- NYBabyNames
- NameData
- WordCounts
- publicdata:samples
- github\_timeline
- gsod
- natality
- shakespeare
- trigrams
- wikipedia

SELECT corpus.word\_count FROM publicdata.samples.shakespeare ORDER BY word\_count DESC LIMIT 200;

FROM GCODE | Query running (1.9s)

Recent Queries

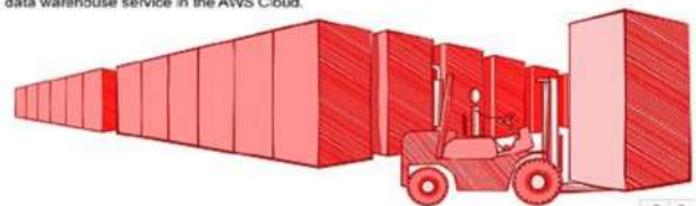
SELECT corpus.word\_count FROM publicdata.samples.shakespeare ORDER BY word\_count DESC LIMIT 200; 3:17pm

SELECT word.COUNT(word) AS wordcount FROM publicdata.samples.shakespeare WHERE word == "A" AND word == "B" GROUP BY word HAVING COUNT(word) > 10 ORDER BY word LIMIT 10; 12:53pm

SELECT word.COUNT(word) AS wordcount FROM publicdata.samples.shakespeare WHERE word == "A" AND word == "B" GROUP BY word HAVING COUNT(word) > 100 ORDER BY word LIMIT 10; 12:53pm

**Introducing Amazon Redshift**

A fast and powerful, fully managed petabyte-scale data warehouse service in the AWS Cloud.



< >

**Mesa: Geo-Replicated, Near Real-Time, Scalable Data Warehousing**

Ashish Gupta, Fan Yang, Jason Govig, Adam Kirsch, Kelvin Chan  
Kevin Lai, Shuo Wu, Sandeep Govind Dhoot, Abhilash Rajesh Kumar, Ankur Agiwal  
Sanjay Bhansali, Mingsheng Hong, Jamie Cameron, Masood Siddiqi, David Jones  
Jeff Shute, Andrey Gubarev, Shivakumar Venkataraman, Divyakant Agrawal  
Google, Inc.

**ABSTRACT**  
Mesa is a highly scalable analytic data warehousing system

ness critical nature of this data result in unique technical and operational challenges for processing, storing, and querying.

116

# OLAP展望

- 面向对象的联机分析处理
  - O3LAP(**Object-Oriented OLAP**)
- 对象关系的联机分析处理
  - OROLAP (**Object Relational OLAP**)
- 分布式联机分析处理
  - DOLAP (**Distributed OLAP**)
- 时态联机分析处理
  - TOLAP (**Temporal OLAP**)