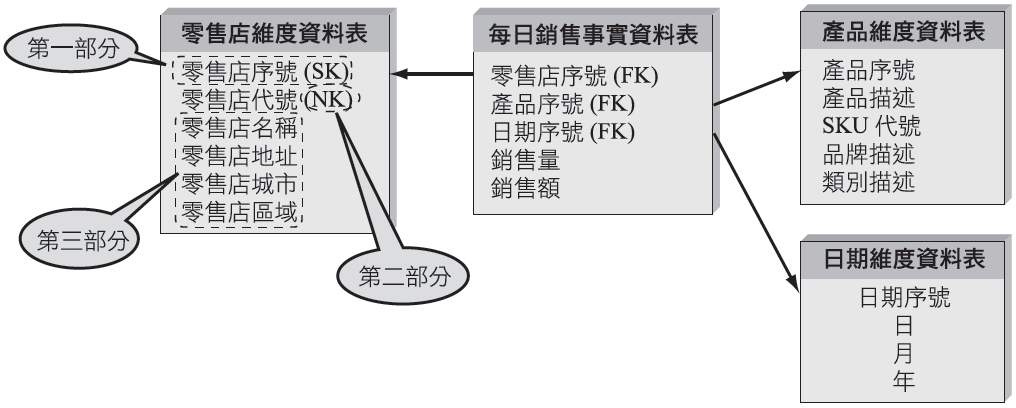


真實商業世界的dimensional model常包含4~15個維度，少於這個數量會讓人懷疑是否太少；若超過20個則不易解釋，宜與予合併。

* **顆粒度 (Granularity) 是描述事實資料表中第二部分欄位的精細度，同一張表中所有衡量欄位都要有相同精細度**
  + **例如以銷售資料來說，常見的欄位有銷售量與銷售金額，**
  + **如果顆粒度是訂單中列項目 (Line item)，則銷售數量與銷售金額都是指某個品項在某張訂單中的銷售數字。**
  + **相對的，如果顆粒度是訂單，則銷售金額與銷售量就是加總一整張訂單上的列項目所得的銷售量與銷售金額。**
  + **如果顆粒度是指一天的營業額則銷售量與銷售金額則是一整天訂單中的銷售量與銷售金額的加總。**
* **Demention基本結構包含三個部分**
  + **第一部分**
    - **系統給定的代理鍵 (Surrogate keys, SK)(如圖 3-12 中的零售店序號)**
  + **第二部分**
    - **自然鍵 (Natural keys, NK) 欄位，這些號碼主要是幫助作業人員容易辨識但因常含有文字類型資料，因此會造成資料查詢速度變慢(如圖 3-12 中的零售店代號)**
  + **第三部分**
    - **一個或多個描述性欄位 (Descriptive fields)，主要以文字型態呈現資料(例如業務員名稱、客戶名稱、產品名稱) ，但有時也有離散型的數值型態的資料(例如訂單編號、出貨單號、採購單號)**

****

Dimension表格沒有包含像Fact表格中那麼多的資料列，且其中的資料是可以修改的，其有以下的特性：

* 其資料少於Fact表格，可能只有上百或上千個資料而已。
* 大都為文字資料。
* 欄位會較多，其是用來管理Dimension的階層（hierarchies）。
* 只有一個主鍵（Primary Key或Dimension Key）。
* 資料可以修改。

缓慢变化维处理系统（Slowly Changing Dimension Processor，SCD）

主要功能是处理维度表的属性随时间变化的情况，处理方式为：类型1（直接覆盖），类型2（生成新行），类型3（添加新列）。

**Accumulating snapshot fact table：累积快照事实表记录的不确定的周期的数据。**

企業流程是由許多串連活動的執行而組成，公司的專案經理人 (Project manager, PM) 需要有一種可以記錄整個企業流程步驟的事實資料表，以便時時可以注意到跨步驟或跨活動資訊來掌握全面性資訊，這樣的事實資料表稱為累積快照事實資料表 (Accumulating snapshot fact tables)。累積 (Accumulating) 一詞的意思就是指這些事情 (活動、步驟、事件) 不會在同一時間完成，因此必須透過許多記錄時間的欄位在事情完成後記錄完成時間與相關的衡量數值。

**Additive, Semi-Additive, and Non-Additive Facts**

事实表中，数值型的度量值一般分为三种类型：  
全加型(Additive)：在任何维度条件下，数据都是有实际意义的可加性的。  
半加型(Semi-Additive)：在部分维度条件下，数据可加；  
非加型(Non-Additive)：在任何维度条件下，数据都是不可加的。如利润率、单价、温度等，不同的值之间相加是没有意义的。

**Allocated Facts**

**Bridge Tables**桥接表（Bridge Table）是维度建模中的一类比较特殊的表。

在数据仓库的建模时，会遇到具有层次结构的维度表，对于这样的表有一种建模方式是建立父子表，即每条记录上包括一个指向其父记录的字段。这种父子表的建立在层级深度可变时尤其有用，是一个紧凑而有效的建模方式。但是这种建模方式也有缺点，就是用标准SQL很难对递归结构进行操作。

与这种递归结构的父子表不同，桥接表采用不同的建模方式也可以表示这种层级结构。桥接表是建立在维度表和事实表中间的一个具有较多冗余信息的表，其中的记录包含层级结构中节点到其下面每个节点的路径。表结构如下所示：

父关键字

子关键字

父层数

层名

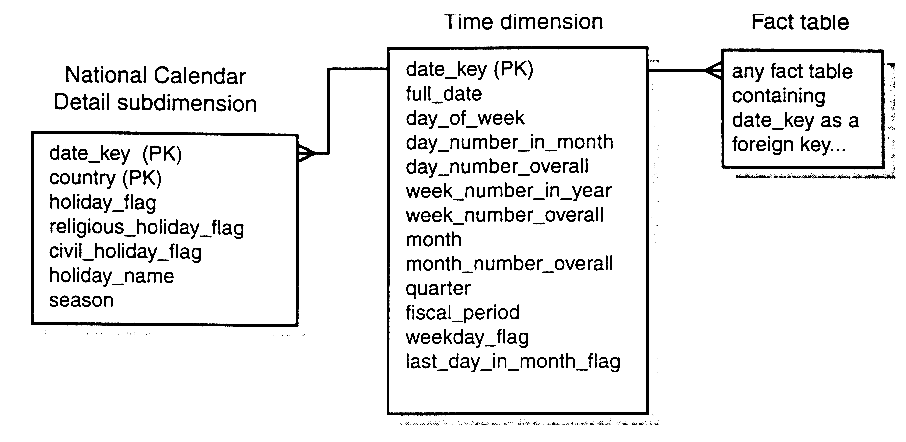
底端标识

顶端标识

在桥接表中，节点与其下面的任意一个节点都建立一个关联记录保存在表中，即父子关系不再局限在相邻层，如第一层与第三层同样有父子关系，通过父层数可以区分相隔了几层。这样，可以通过父层数和父子关系来进行层级结构的查询。

当然，桥接表也不是一个完备的解决方案，它只能是在某些情况下是查询变得容易。

**Calendar Date Dimensions**



**Cardinality**理解为可选项

**Cardinality:**    优化器在计算成本的时候，需要从统计信息中取得数据，然后去估计每一步操作所涉及的行数，叫做Cardinality。  
    比如，一张表T有1000行数据，列COL1上没有直方图，没有空值，并且不重复的值（distinct value）有500个。那么，在使用条件“WHERE COL1=<VALUE>”去访问表的时候，优化器会假设数据均匀分布，它估计出会有1000/500=2行被选出来，2就是这步操作的Cardinality。

**Conformed Dimensions**

Conformed dimension在每一個data mart均相同，例如消費者、產品、位置、交易（促銷）、日期（時間）等。中心資料倉儲設計小組的主要責任是確立、公佈、維護和實施conformed dimensions。確立conformed dimension是相當重要的步驟，例如一個conformed customer dimension是customer經過淨化，包含消費者的key、最佳維護屬性等，用來描述消費者的主要table。它是經過合併而且是去蕪存菁的資料。

Conformed dimensions通常都是最小的單元，因為他們的每一筆資料相當於是單一顧客、產品、天的描述。換句話說，在每個data mart的base level fact tables必須是相當於所有組成dimensions的最低層次。

Conformed dimensions使以下的事情變成可能：

* 在同一個資料庫環境下，單一的dimension table就能提供多個fact tables使用。
* 不管何時使用dimension，使用者界面和資料內容可以一致。
* 屬性可以有一致的解釋。

维度建模的数据仓库中，有一个概念叫Conformed Dimension，中文一般翻译为“一致性维度”。一致性维度是Kimball的多维体系结构（MD）中的三个关键性概念之一，另两个是总线架构（Bus Architecture）和一致性事实（Conformed Fact）。

在多维体系结构中，没有物理上的数据仓库，由物理上的数据集市组合成逻辑上的数据仓库。而且数据集市的建立是可以逐步完成的，最终组合在一起，成为一个数据仓库。如果分步建立数据集市的过程出现了问题，数据集市就会变成孤立的集市，不能组合成数据仓库，而一致性维度的提出正式为了解决这个问题。

一致性维度的范围是总线架构中的维度，即可能会在多个数据集市中都存在的维度，这个范围的选取需要架构师来决定。一致性维度的内容和普通维度并没有本质上区别，都是经过数据清洗和整合后的结果。

一致性维度建立的地点是多维体系结构的后台（Back Room），即数据准备区。在多维体系结构的数据仓库项目组内需要有专门的维度设计师，他的职责就是建立维度和维护维度的一致性。在后台建立好的维度同步复制到各个数据集市。这样所有数据集市的这部分维度都是完全相同的。建立新的数据集市时，需要在后台进行一致性维度处理，根据情况来决定是否新增和修改一致性维度，然后同步复制到各个数据集市。这是不同数据集市维度保持一致的要点。

在同一个集市内，一致性维度的意思是两个维度如果有关系，要么就是完全一样的，要么就是一个维度在数学意义上是另一个维度的子集。例如，如果建立月维度话，月维度的各种描述必须与日期维度中的完全一致，最常用的做法就是在日期维度上建立视图生成月维度。这样月维度就可以是日期维度的子集，在后续钻取等操作时可以保持一致。如果维度表中的数据量较大，出于效率的考虑，应该建立物化视图或者实际的物理表。

这样，维度保持一致后，事实就可以保存在各个数据集市中。虽然在物理上是独立的，但在逻辑上由一致性维度使所有的数据集市是联系在一起，随时可以进行交叉探察等操作，也就组成了数据仓库。

**Conformed Facts**必須被conformed的facts如歲入、收益、標準價格、和標準成本等，只要得自這些facts而且名稱是相同的，就必須均是相同的值。這些conformed fact定義需要被定義在相同的dimension context以及每個data mart中與它相同的units之內。歲入與收益必須在相同的時期和地區進行報表產製。

维度建模的数据仓库中，有一个概念叫Conformed Fact，中文一般翻译为“一致性事实”。一致性事实是Kimball的多维体系结构（MD）中的三个关键性概念之一，另两个是总线架构（Bus Architecture）和一致性维度（Conformed Dimension）。

在建立多个数据集市时，完成一致性维度的工作就已经完成了一致性的80%－90%的工作量。余下的工作就是建立一致性事实。

一致性事实和一致性维度有些不同，一致性维度是由专人维护在后台（Back Room），发生修改时同步复制到每个数据集市，而事实表一般不会在多个数据集市间复制。需要查询多个数据集市中的事实时，一般通过交叉探查（drill across）来实现。

为了能在多个数据集市间进行交叉探查，一致性事实主要需要保证两点。第一个是KPI的定义及计算方法要一致，第二个是事实的单位要一致性。如果业务要求或事实上就不能保持一致的话，建议不同单位的事实分开建立字段保存。

这样，一致性维度将多个数据集市结合在一起，一致性事实保证不同数据集市间的事实数据可以交叉探查，一个分布式的数据仓库就建成了。维度建模的数据仓库中，有一个概念叫Conformed Fact，中文一般翻译为“一致性事实”。一致性事实是Kimball的多维体系结构（MD）中的三个关键性概念之一，另两个是总线架构（Bus Architecture）和一致性维度（Conformed Dimension）。

在建立多个数据集市时，完成一致性维度的工作就已经完成了一致性的80%－90%的工作量。余下的工作就是建立一致性事实。

一致性事实和一致性维度有些不同，一致性维度是由专人维护在后台（Back Room），发生修改时同步复制到每个数据集市，而事实表一般不会在多个数据集市间复制。需要查询多个数据集市中的事实时，一般通过交叉探查（drill across）来实现。

为了能在多个数据集市间进行交叉探查，一致性事实主要需要保证两点。第一个是KPI的定义及计算方法要一致，第二个是事实的单位要一致性。如果业务要求或事实上就不能保持一致的话，建议不同单位的事实分开建立字段保存。

这样，一致性维度将多个数据集市结合在一起，一致性事实保证不同数据集市间的事实数据可以交叉探查，一个分布式的数据仓库就建成了。

**Degenerate Dimensions**

Degenrate diemension是指在业务系统中的旧主键，在转到数据仓库后，并没有相对应的维度表进行绑定，已经退化，但建议对其进行保留。因为利用该维度，还是可以对原始的事务进行还原。

很多維度的設計依循著一些控制文件如訂單、發票、出貨單等，通常這些文件包含著一連串的資料項目，以訂單為例就有時間、客戶、貨品名稱、數量、送貨地址等資料，而這些即是fact table的紀錄。在此例中，我們最適合的儲存顆粒度是以訂單中的每一訂購項目（line item）為基本儲存單位，也就是fact table記錄訂單中的不同訂購項目，而dimension table自然也必須根據不同的訂購項目來設計，如此藉由描述每一項目我們可以很快得到所需的維度（dimension）。

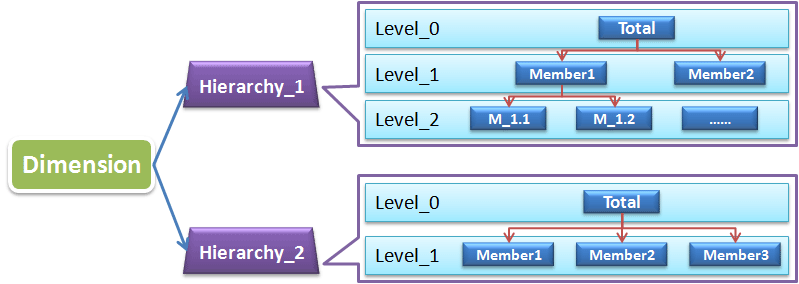
在此例中，我們會把原先order中的其他屬性搬移到其他直接參照line item table的dimension，最後可能會造成order所有的屬性都被搬走了，而在line-item fact table中，我們雖然還有order number屬性，但order number將無任何attribute，稱為degenerate dimension。

在维度建模的数据仓库中，有一种维度叫Degenerate Dimension，中文一般翻译为“退化维度”。这种退化维度一般都是事务的编号，如订单编号、发票编号等。这类编号需要保存到事实表中，但是不需要对应的维度表，所以称为退化维度。

退化维度是维度建模领域中的一个非常重要的概念，它对理解维度建模有着非常重要的作用，尤其是对维度建模的入门者。

退化维度经常会和其他一些维度一起组合成事实表的主键。在Kimball提出的维度建模中，事实表应该保存最细粒度的数据。所以对于象销售单这样的事实表来说，需要销售单编号和产品来共同作为主键，而不能用销售日期、商场、产品等用来分析的维度共同作为主键。

退化维度在分析中可以用来做分组使用。它可以将同一个事务中销售的产品集中在一起。



**Dimension Hierarchies: Fixed Depth Positional Hierarchies**

固定层级结构生成系统（Fixed Hierarchy Dimension Builder）

主要功能是对维度表中各类多对一关系的层级结构进行数据有效性检查和维护。

**Dimension Hierarchies: Multiple Hierarchies in Dimensions 例子：DATA**

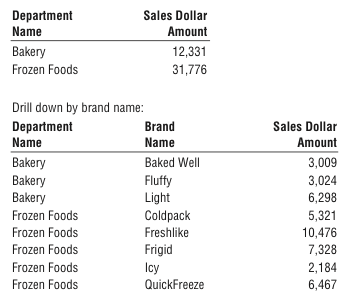
**Dimension Hierarchies: Ragged/Variable Depth Hierarchies bridge table**

可变层级结构生成系统（Variable Hierarchy Dimension Builder）

主要功能是对维度表中所有的层深可变的层级结构的的数据有效性检查和维度，例如组织的层级结构，零件的层级结构等。

**Dimension Hierarchies: Slightly Ragged/Variable Depth Hierarchies**

**Drilling Down**

Drill down 是指从维度表中增加维度，把现有的数据按增加的维度进行细分。 上钻（Drilling up）是指除去维度，并将该维度的数据合并。如下图所示，按照 brand name 下钻：  
[](http://brucenan.qiniudn.com/drilldown.png)下钻

**Enterprise Data Warehouse Bus Architecture**

* **為了解決資料同名異義與異名同義，Kimball 建議需要事先建立資料倉儲匯流排架構 (Data warehouse bus architecture)。**

架構階段是在任何data mart的實行前進行，目的是定義一個facts標準和合適的dimensions，這些標準的結果集合就叫Data Warehouse Bus Architecture

维度建模的数据仓库中，有一个概念叫Bus Architecture，中文一般翻译为“总线架构”。总线架构是Kimball的多维体系结构（MD）中的三个关键性概念之一，另两个是一致性维度（Conformed Dimension）和一致性事实（Conformed Fact）。

在多维体系结构（MD）的数据仓库架构中，主导思想是分步建立数据仓库，由数据集市组合成企业的数据仓库。但是，在建立第一个数据集市前，架构师首先要做的就是设计出在整个企业内具有统一解释的标准化的维度和事实，即一致性维度和一致性事实。而开发团队必须严格的按照这个体系结构来进行数据集市的迭代开发。

一致性维度就好比企业范围内的一组总线，不同数据集市的事实的就好比插在这组总线上的元件。这也是称之为总线架构的原因。

实际设计过程中，我们通常把总线架构列表成矩阵的形式，其中列为一致性维度，行为不同的业务处理过程，即事实，在交叉点上打上标记表示该业务处理过程与该维度相关。这个矩阵也称为总线矩阵（Bus Matrix）。

总线架构和一致性维度、一致性事实共同组成了Kimball的多维体系结构的基础，也建立了一套可以逐步建立数据仓库的方法论。由于总线架构是多维体系结构的核心，所以我们有时就把多维体系结构直接称为总线架构。

**Enterprise Data Warehouse Bus Matrix**

**Factless Fact Tables**

对于没有度量值的事实表，我们称之为无事实的事实表，一般通过利用count，distinct等统计记录个数的方式，得出所需的度量指标。

Factless Fact Table，中文一般翻译为“非事实型事实表”。在事实表中，通常会保存十个左右的维度外键和多个度量事实，度量事实是事实表的关键所在。在非事实型事实表中没有这些度量事实，只有多个维度外键。非事实型事实表通常用来跟踪一些事件或者说明某些活动的范围。下面举例来进行说明。

第一类非事实型事实表是用来跟踪事件的事实表。例如：学生注册事件，学校需要对学生按学期进行跟踪。维度表包括学期维度、课程维度、系维度、学生维度、注册专业维度和取得学分维度，而事实表是由这些维度的主键组成，事实只有注册数，并且恒为1。这样的事实表可以回答大量关于大学开课注册方面的问题，主要是回答各种情况下的注册数。

第二类非事实型事实表是用来说明某些活动范围的事实表。例如：促销范围事实表。通常销售事实表可以回答如促销商品的销售情况，但是对于那些没有销售出去的促销商品没法回答。这时，通过建立促销范围事实表，将商场需要促销的商品单独建立事实表保存。然后，通过这个促销范围事实表和销售事实表即可得出哪些促销商品没有销售出去。这样的促销范围事实表只是用来说明促销活动的范围，其中没有任何事实度量。

**Junk Dimensions**從複雜的資料來源萃取資料後會留下許多難以理解的旗標(flag)及text attribute，設計者有很多解決方案，但需避免：

* + 把flag及text attribute留在fact table紀錄中：因為會造成fact table紀錄的定義驚人的增加。
  + 使每一個flag及attribute都存入各自不同的dimension中：原因同上。
  + 將flag及attribute刪除：假使flag and attribute 和某些交易有關係，則會產生錯誤。

解決方法是仔細研究flag and attribute ，並將flag and attribute 整理並集合存於一個或以上的junk dimensions中。

在维度建模的数据仓库中，有一种维度叫Junk Dimension，中文一般翻译为“杂项维度”。杂项维度是由操作系统中的指示符或者标志字段组合而成，一般不在一致性维度之列。

在操作系统中，我们定义好各种维度后，通常还会剩下一些在小范围内取离散值的指示符或者标志字段。例如：支付类型字段，包括现金和信用卡两种类型，在源系统中它们可能是维护在类型表中，也可能直接保存在交易表中。

一张事实表中可能会存在好几个类似的字段，如果作为事实存放在事实表中，会导致事实表占用空间过大；如果单独建立维度表，外键关联到事实表，会出现维度过多的情况；如果将这些字段删除，会有人不同意。

这时，我们通常的解决方案就是建立杂项维度，将这些字段建立到一个维度表中，在事实表中只需保存一个外键。几个字段的不同取值组成一条记录，生成代理键，存入维度表，并将该代理键保存入相应的事实表字段。建议不要直接使用所有的组合生成完整的杂项维度表，在抽取时遇到新的组合时生成相应记录即可。杂项维度的ETL过程比一般的维度略为复杂。

**Multivalued Dimensions = Bridge Tables**

在维度建模的数据仓库中，有一种维度表叫multivalue dimension，中文一般翻译为“多值维度”。

多值维度有两种情况，第一种情况是指维度表中的某个属性字段同时有多个值。举例来说，一个帐户维度表中，帐户持有人姓名，可能会有多个顾客。这样，一个帐户对应多个顾客姓名，一个顾客也可以有多个帐户，它们之间是多对多的关系。正因为一个帐户可能会有多个对应的顾客，所以不能直接将顾客ID放入帐户维度表中。而帐户维度表中的这种情况就叫做多值维度。

多值维度的第二种情况是事实表在某个维度表中有多条对应记录。举例来说，对于一个健康护理单分列项事实表来说，它的粒度是一个健康护理单，但是该护理单却有可能有多次诊断，即该事实表与诊断维度的是一对多的关系。这个与事实表粒度不匹配的诊断维度也称之为多值维度。

处理多值维度最好的办法是降低事实表的粒度。如第二种情况中，将健康护理单分列项事实表的粒度降低到具体的诊断粒度上，这样就避免了多值维度的出现。这种处理方式也是维度建模的一个原则，即事实表应该建立在最细粒度上。这样的处理，需要对事实表的事实进行分摊。

但是有些时候，事实表的粒度是不能降低的，多值维度的出现是无法避免的。如第一种情况中，事实表是月帐户快照事实表，这张事实表与顾客维度没有直接的关系，不能将数据粒度进行细分，即使细分的话帐户余额也很难分摊。这时，可以采用桥接表技术进行处理。在帐户维度表和顾客维度表之间建立个帐户-顾客桥接表。这个桥接表可以解决掉帐户维度和顾客维度之间的多对多关系，也解决掉的帐户维度表的多值维度问题。

总之，多值维度是应该尽量避免的，它给数据处理带来了很大的麻烦。如果多值维度不能避免的话，应该建立桥接表来进行处理。

**Natural, Durable, and Supernatural Keys**

**Null Attributes in Dimensions**

**Opportunity/Stakeholder Matrix**

**Outrigger Dimensions**支架维度－－outrigger dimension

**Periodic Snapshot Fact Tables**

周期快照事实表以具有规律性的、可预见的时间间隔来记录事实，时间间隔如每天、每月、每年等等。典型的例子如销售日快照表、库存日快照表等。   
  
       周期快照事实表的粒度是每个时间段一条记录，通常比事务事实表的粒度要粗，是在事务事实表之上建立的聚集表。周期快照事实表的维度个数比事务事实表要少，但是记录的事实要比事务事实表多。   
  
       周期快照事实表的日期维度通常是记录时间段的终止日，记录的事实是这个时间段内一些聚集事实值。事实表的数据一旦插入即不能更改，其更新方式为增量更新。

**Role-Playing Dimensions** 在数据仓库领域有一个概念叫Role-playing dimensions，中文一般翻译为“角色模仿维度”。角色模仿维度是为了处理一个维度在一个事实表中同时出现多次而使用的一种技术处理手段。

在建立了角色模仿维度以后，在底层只有一个物理表存在，但是针对这个物理表会建立多个角色提供给数据访问工具，而且对数据访问工具来说这多个角色是不同的。例如对与累计快照事实表中会出现多个日期字段联接到日期维度。这时就可以针对日期维度建立多个角色模仿维度。

角色模仿维度的建立方法通常是使用视图来完成。例如订单日期维度表如下所示：

CREATE VIEW order\_date(order\_date\_key, order\_day\_of\_week, order\_month, … )

AS SELECT data\_key, day\_of\_week, month, … FROM DATA

使用同样的方式还可以建立多个不同日期的角色模仿维度。

需要补充的一点是，目前市场上的大部分展现工具，都提供了对一个表选择多次的功能。也就是说，角色模仿维度的功能展现工具自己就可以实现。这样，就不需要我们在数据库中建立角色模仿维度的视图了，而直接使用展现工具完成即可。

**Shrunken Rollup Dimensions**

**Surrogate Keys (Dimension Surrogate Keys)** 业务系统中表的主键，一般称为自然键natural key。维度表的主键一般不使用自然键，而是使用无意义的数字作为主键。维度表中记录的主键，称为代理键（Surrogate Key）。一般从1开始，2，3这样顺序排下去，它没有实际的意义，主要是用来与事实表进行联接。

代理关键字的称呼有surrogate keys，meaningless keys，integer keys，nonnatural keys，artificial keys，synthetic keys等。与之相对的自然关键字的称呼有natural keys，samat keys等。

在Kimball的维度建模领域里，是强烈推荐使用代理关键字的。在维度表和事实表的每一个联接中都应该使用代理关键字，而不应该使用自然关键字或者智能关键字（Smart Keys）。数据仓库中的主键不应该是智能的，也就是说，要避免通过主键的值就可以了解一些业务信息。当然，退化维度作为事实表的复合主键之一时例外。

使用代理关键字，有很多优点。

1．使用代理关键字能够使数据仓库环境对操作型环境的变化进行缓冲。也就是说，当数据仓库需要对来在多个操作型系统的数据进行整合时，这些系统中的数据有可能缺乏一致的关键字编码，即有可能出现重复，这时代理关键字可以解决这个问题。

2．使用代理关键字可以带来性能上的优势。和自然关键字相比，代理关键字很小，是整型的，可以减小事实表中记录的长度。这样，同样的IO就可以读取更多的事实表记录。另外，整型字段作为外键联接的效率也很高。

3．使用代理关键字可以建立一些不存在的维度记录，例如“不在促销之列”，“日期待定”，“日期不可用”等维度记录。

4．使用代理关键字可以用来处理缓慢变化维。维度表数据的历史变化信息的保存是数据仓库设计的实施中非常重要的一部分。Kimball的缓慢变化维处理策略的核心就是使用代理关键字。

当然，使用代理关键字也有它的缺点，代理关键字的使用使数据加载变得非常复杂。有关使用代理关键字的维度表和事实表的加载方法在ETL Toolkit中有详细的描述。使用代理关键字是一个从长远考虑的策略。

**Transaction Fact Tables**

事务事实表是维度建模的数据仓库中三种基本类型事实表中的一种，另外两种分别是周期快照事实表和累积快照事实表。

       事务事实表与周期快照事实表、累积快照事实表使用相同的一致性维度，但是它们在描述业务事实方面是有着非常大的差异的。    
  
       事务事实表记录的事务层面的事实，保存的是最原子的数据，也称“原子事实表”。事务事实表中的数据在事务事件发生后产生，数据的粒度通常是每个事务一条记录。一旦事务被提交，事实表数据被插入，数据就不再进行更改，其更新方式为增量更新。    
  
       事务事实表的日期维度记录的是事务发生的日期，它记录的事实是事务活动的内容。用户可以通过事务事实表对事务行为进行特别详细的分析。    
  
       通过事务事实表，还可以建立聚集事实表，为用户提供高性能的分析。

**Type 4: Add Mini-Dimension**

维度建模的数据仓库中，有一种维度叫minidimension，中文一般翻译成“微型维度”。微型维度的提出主要是为了解决快变超大维度（rapidly changing monster dimension）。

以客户维度举例来说，如果维度表中有数百万行记录或者还要多，而且这些记录中的字段又经常变化，这样的维度表一般称之为快变超大维度。对于快变超大维度，设计人员一般不会使用TYPE 2的缓慢变化维处理方法，因为大家都不愿意向本来就有几百万行的维度表中添加更多的行。

这时，有一项技术可以解决这个问题。解决的方法是，将分析频率比较高或者变化频率比较大的字段提取出来，建立一个单独的维度表。这个单独的维度表就是微型维度表。

微型维度表有自己的关键字，这个关键字和原客户维度表的关键字一起进入事实表。有时为了分析的方便，可以把微型维度的关键字的最新值作为外关键字进入客户维度表。这时一定要注意，这个外关键字必须做TYPE 1型处理。

在微型维度表中如果有像收入这样分布范围较广的属性时，应该将它分段处理。比如，存储￥31257.98这样过于分散的数值就不如存储￥30000－￥34999这样的范围。这样可以极大的减少微型维度中的记录数目，也给分析带来方便。

**Value Chain**