清洗和规范化

清洗和规范化在 ETL 系统中是最富价值的的步骤。当然，数据抽取和加载步骤绝对必需，只是它们做的仅仅是改变数据位置和改变数据格式。清洗和规范化实际上真正的改变了数据，并对数据能否用于预期的目标起决定作用。

在本章，我们将会跟您一起建立三份重要的可提交内容：数据评估报告(data-profiling report)，错误事件事实表(error event fact table)，审计维(audit dimension)。围绕着这三份提交内容，你可以建立一个功能强大的清洗和规范化系统。

清洗和规范化步骤会生成有效的元数据。回顾原始数据源时，这些元数据可以作为源系统的错误诊断报告。最终，仅仅通过改变源系统采集数据的方式就可以修正这些脏数据。记得我们说过"业务流程再造"么？

在清洗和规范化步骤中产生的元数据，将伴随着真实数据一起摆在用户桌面上，或者，至少它应该是这样的。ETL 小组必须使清洗和规范化的元数据可用，这也是引起审计维的原因。

请仔细阅读本章，这非常重要。本章作了一系列的努力为通常难以归类的问题提供了特定的技术和结构。本章篇幅比较长，也许你应该至少阅读两遍，但我们确信这将为你在 ETL 系统中构建数据清洗和规范化方面提供有用的指导。

如果你是 ETL 系统设计方面的新手，你可能会问"我该从哪里开始呢？"也许最好的答案是：先尽你所能做好数据评估分析。这对于你更清楚地认识你将要处理的脏数据或不可靠数据的风险大有帮助。借助于从数据评估中获得的认识，你就能够对问题进行分解，并且在设计简单的错误事件事实表和简单的审计维时有更多的自信。

流程检查

规划与设计：需求/现状 -> *架构* -> *实现* -> 测试/发布

数据流：抽取 -> *清洗* -> *规格化* -> 提交

本章按照下面四个与数据质量相关的话题逐步展开：

- 第一部分：设计目标
- 第二部分：清洗提交
- 第三部分：快照及其度量
- 第四部分：规范化提交

这是对于数据质量的自上而下的阐释。在目标部分，我们会督促着你实现"全面、快速、准确、清晰"地设计目标。清洗和规范化过程中我们更多关注的是约束关系与控制而不是数据之上的内容。在某种意义上，这是数据仓库小组不太熟悉的一个领域。在提交阶段，我们会介绍清洗子系统的主要的结构：错误事件表和审计维。我们还会推荐你学习附录 B 中关于 Jack Olson 的著作，用以针对事先的数据评估步骤的结果设计一个有机的结构（the results of your up-front data-profiling pass.）

再往下的过滤器部分，我们会在很多地方定义检查点和筛选器，用来对数据质量进行测量。使用过滤器，我们建立了一个统一的方案来捕捉数据质量事件并对这些事件采取相应动作作为响应。

定义：数据质量过滤器

在本章中，我们会反复提到数据质量过滤器。我们使用过滤器这个词既可以用来表示报表，又可以用来表示筛选。这样，数据质量过滤器在物理上可以被 ETL 团队看作是关于数据质量的状态报告，但同时也是一种防止脏数据通过的“墙”。

在第四部分，我们将描述规范化步骤的主要的提交内容：规范化的维和事实，以及它们如何被传递（handed off）。同时我们也建议了一些元数据方法，用以持续追踪组织所做的决策，以此来标准化维和事实。

最后的度量部分是针对特定数据质量问题的实施指南。与第三章中关于不同类型的抽取问题的详细描述一样，这些数据质量度量将是一个在其之上进行建设的合理的基础。

本章的内容大量的借鉴了 Jack Olsen 和 Larry English 这样的数据质量方面的顶尖级专家的成果。他们的简明而直接的数据质量度量技术在本章大量的使用，并且成为创建有效的数据质量 ETL 流程的模版。

定义数据质量

首先我们应该在一些基础词汇上达成共识，集中在“精确性”。精确的数据意味着这样的数据：

- 正确的：数据的值和描述真实的反映了它需要表达的对象，并且这种表达是可信的。例如，某一位作者当前居住的城市叫 New Hope，那么，关于家庭地址的“精确的”数据应该包含正确的城市名称 New Hope。
- 明确的：数据的值和描述应该是只有一个含义。例如，在美国至少有十个城市叫 New Hope，而在宾夕法尼亚州只有一个城市叫 New Hope。那么，在“精确的”数据中，描述这个城市中的地址是应该包含城市名 New Hope 以及州名称宾夕法尼亚，这样的数据才是“明确的”。
- 一致的：数据的值和描述用一种不变的标识约定来表达其含义。例如，美国的宾夕法尼亚洲可能在数据中表达为 PA，Penn 或 Pennsylvania。为了满足一致性的要求，关于当前家庭住址的的“精确”数据应该只使用一种约定方式来表达州名称(比如使用全名 Pennsylvania)，并且坚持这一种用法。
- 完整的：关于完整性有两个方面。
  - 第一点是要确保数据中为每个实例定义了（不能为空）特定的值和描述，例如，要确保所有应该有当前住址信息的记录实际上确实有值。
  - 第二点是要确保记录的总数量是完整的，或者确保在整个信息流中没有在任何地方丢失记录。

与完整性相关的一个问题是数据中的缺失值的多种可能的含义。用“空”来描述的缺失值可能代表真实的数据未知，或者不可用。缺失值可以用空格、空字符串或其他自创的描述（比如，不知道或拒绝回答）来表示。

假设

在本章我们会对清洗和规范化发生的环境做一些基本的假设。首先，ETL 作业流中有一些不同的断点，在这些断点的位置可以插入数据质量处理过程。在我们整体的 ETL 数据流模型中，有两个这样的点是显而易见的：

- 第一个里程碑是在数据抽取完毕之后——这意味着数据已经从多个数据源抽取出

来并被放置在一个按主题组织的物理的或者逻辑的结构中。例如，来自各个数据源的客户信息被放入某个表、数据文件或内存结构中，无论原始的数据源情况如何不同，最终的这个目标结构的配置是相同的。这样的结构可用于把来自多个数据源的输入数据沉积并排列等待 ETL 处理。在这里几乎没有做任何数据清洗或整合；来自多处的数据只是简单的改变结构，等待下一步的处理。当然这并不意味着来自多个数据源的数据必须并行地进行处理，而只是说我们将使用某个独立于数据源的结构作为中间存储。当然，即使使用了针对每种数据源特定的中间结构和处理流程，我们所描述的数据质量技术也依然有效，只是在本章末尾描述的那个元数据的例子可能要作稍微的调整。即便如此，我们依然建议在这个阶段多运行一些数据质量处理流程，以此来获得组织中的数据质量状态的精确的视图(虽然数据依然是原来的样子)；同时使用这些数据质量处理流程，也可以清除那些毫无希望的缺陷数据，以免这些数据破坏你的数据清洗和整合处理。

- 第二个里程碑是在数据清理和规范化之后——这意味着数据已经成功地通过了 ETL 流程中所有的数据准备和整合步骤，为提交步骤的最终打包做好了准备。我们建议在这个阶段执行更多的数据质量处理，作为数据清洗和整合软件的安全过滤网。实质上，就是在正式提交它们之前，通过一系列的质量保证检查确保其提交内容的真实性。

为了简化起见，本章假定使用的是批量 ETL 处理，而不是实时或准实时处理。我们假设批量处理能够代表了大多数真实的 ETL 环境中所用到的技术，这样我们就可以只关注数据质量问题，而不用考虑复杂的实时 ETL。在第 11 章中会专门论述相关的话题。

第 1 部分：设计目标

流程检查

规划与设计：需求/现状 -> 架构 -> 实现 -> 测试/发布
数据流：抽取 -> *清洗* -> 规格化 -> 提交

这一部分描述了实现数据质量初始目标的相应压力以及 ETL 队伍必须要平衡的优先级冲突问题。我们建议了一些方法来实现这种平衡，同时明确地叙述了一种数据质量策略来满足重要用户参于者的需求。

了解关键参于者

数据质量子系统必须对数据仓库管理员，信息管理员，以及信息质量负责人等这些角色进行支持。虽然这些角色有可能在实际的人员之间以不同的方式分配，但这种角色的划分依然很有意义。

数据仓库管理员

数据仓库管理员负责数据仓库运行期间需要制定的日常决策，确保其是内/外部的数据源的精确反映，并确保这些数据是按照适当的业务规则和原则进行处理的。

清洗和规范化子系统应该支持数据仓库管理员和周边的业务环境，方法是通过提供一个应用到数据上（把数据装载到数据仓库中）的转换的历史，其中包括所有异常情况的详细的审计。

信息驾驶员

信息驾驶员的主要责任是定义信息策略。职责包括：对分析目标进行正规化定义，选择适当的数据源，设置信息生成原则，组织和发布元数据，并且为适当的应用进行限制。

清洗和规范化子系统应该支持信息驾驶员，方法是提供如下一些度量指标，包括操作性数据仓库对已确定的业务规则的日常满足情况，可能用于测试这些规则边界的数据源问题，以及可能造成针对特定应用来说源数据是否合适的数据质量问题。

信息质量负责人

信息质量负责人负责检测、修整和分析数据质量问题，他与信息驾驶员一起，制定脏数据的处理规则，设定可发布的质量门限值，并平衡完备性与快速性，以及在下一部分描述的纠正和透明之间的矛盾。

清洗和规范化子系统应该支持数据质量负责人，通过提供一系列的度量指标，用以描述数据仓库 ETL 过程中的数据质量问题的发生频率和严重程度。这些记录应该是一个完整的历史审计，通过这些信息，数据质量负责人可以清楚地评估随着时间的推移数据质量的改进的成功与否。

维表管理员

维表管理员负责创建和发布整个组织使用的一个或多个规范化的维表。可能会有多个维表管理员，分别负责不同的维表。维表管理员实现了对通用的描述标签达成一致，这些标签由整个数据仓库不同的使用者来访问。维表管理员还要创建和分配代理键并为每一次发布的维表分配版本号，以便在目标事实表环境中使用。当一个新的维表发布到数据仓库环境中的时候，必须同时复制到所有的目标位置，这样可以确保同步装载新版本的维表。维表管理员的工作非常明确：一个规范化的维表必须只能具有唯一的、一致的源。在本章的后半部分我们还会赋予维表管理员更多的职责。

事实表提供者

事实表提供者一般由本地的 DBA 担任，它拥有给定事实表的唯一实例。事实表提供者负责从不同的维表管理员处接收维表，将本地原始键转化为规范化维表中的代理键，并更新事实表使其对用户环境可用。如果有些维表记录的接收延时，事实表提供者可能将不得不在已有的事实表上作复杂的修正。最后。事实表提供者还要负责创建和管理聚合表，这些聚合表用于存储汇总记录以提高特定查询的性能。在本章的后半部分我们还会赋予事实表提供者更多的职责。

竞争因素

如图 4.1 所示，数据质量系统的目标可以用这四个相关的优先级来描述。

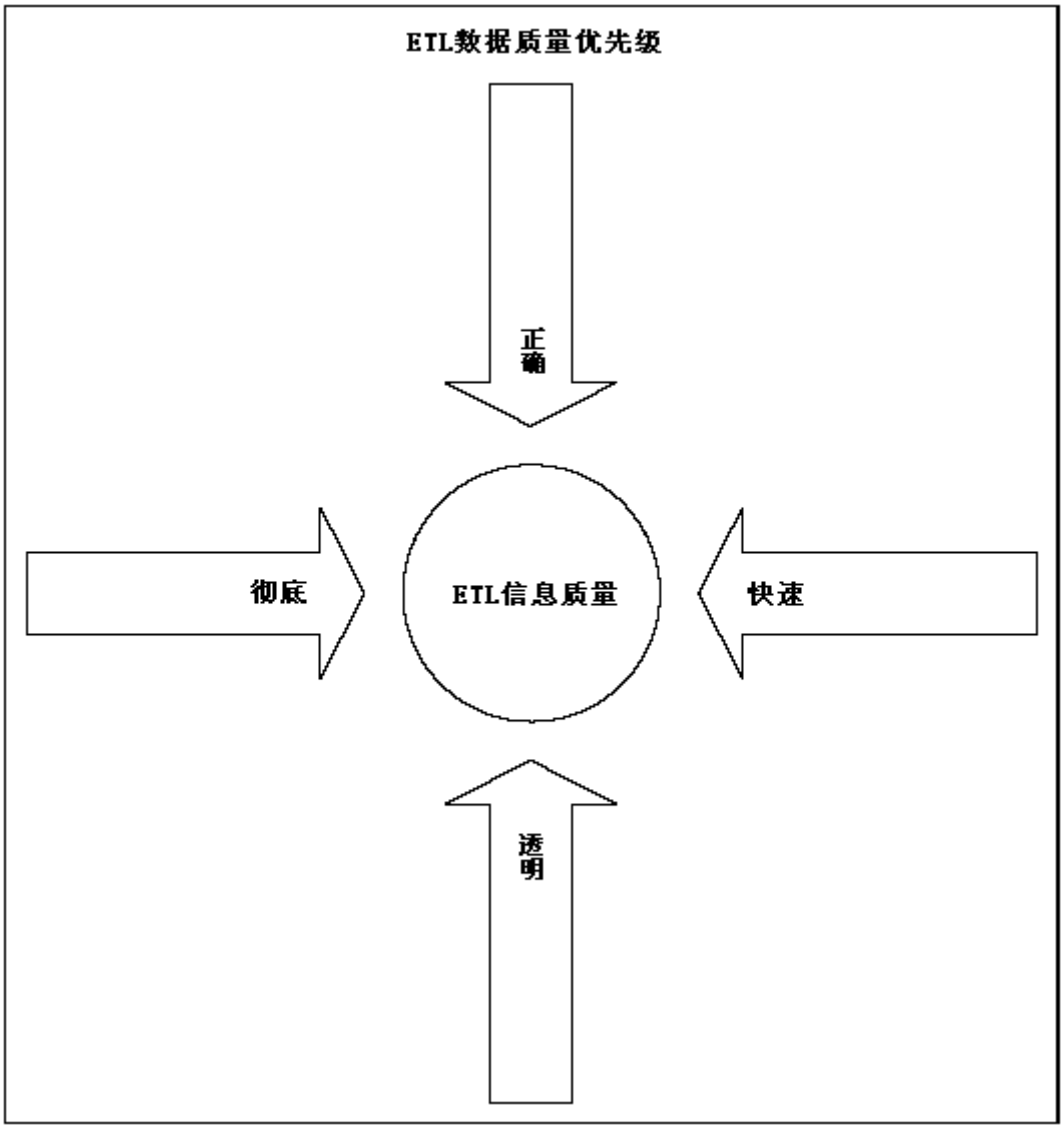

图 4.1　数据质量优先级

确保彻底

数据清洗子系统必须保证在检测、纠正和文档化发布给业务环境的信息质量方面的彻底性。最终用户希望能够把数据仓库作为一个可信赖的信息来源，在此之上来构建他们的管理度量指标、策略和规则。

确保快速

整个 ETL 过程流必须能够在越来越短的时间里处理不断增长的数据量。一些最新的也是最有趣的客户想要访问的信息非常的详细和个性化（比如 web 点击率）这会导致海量的数据被装载到数据仓库。

确保正确

事实上，在源系统或尽量接近源系统的位置改正数据质量问题当然是提升组织信息资产的最佳方式，这会降低高昂的成本，并降低劣质数据的产生机会。但是，现实情况是，大多数的组织并没有建立正规的数据质量环境和信息质量负责人制度，在这种情况下，数据仓库

小组是最有可能第一个发现困扰组织多年的质量问题的人，那么，理所当然的，数据仓库小组应该担当起矫正这些问题的责任。

确保透明

数据仓库必须暴露出错误，并对那些损害组织中数据质量的系统和业务实践加以注意。这些发现最终会导致业务流程的再造，而这又会提升源系统和数据项过程的水平。试图在源系统掩盖数据质量问题的“英勇行为”会造成这样一种结果，修复的代价远远大于问题本身。

平衡冲突的优先级

很显然，对于清洗系统来讲，完全同时达到这些目标是不可能的。必须要作适当的平衡，这反映了不同情况下的优先级。

完备性与快速性

数据质量 ETL 无法在快速性和完备性方面同时达到最优。相反，我们试图在图 4.2 的指数关系曲线上找到那个合适的点，来实现我们的平衡。

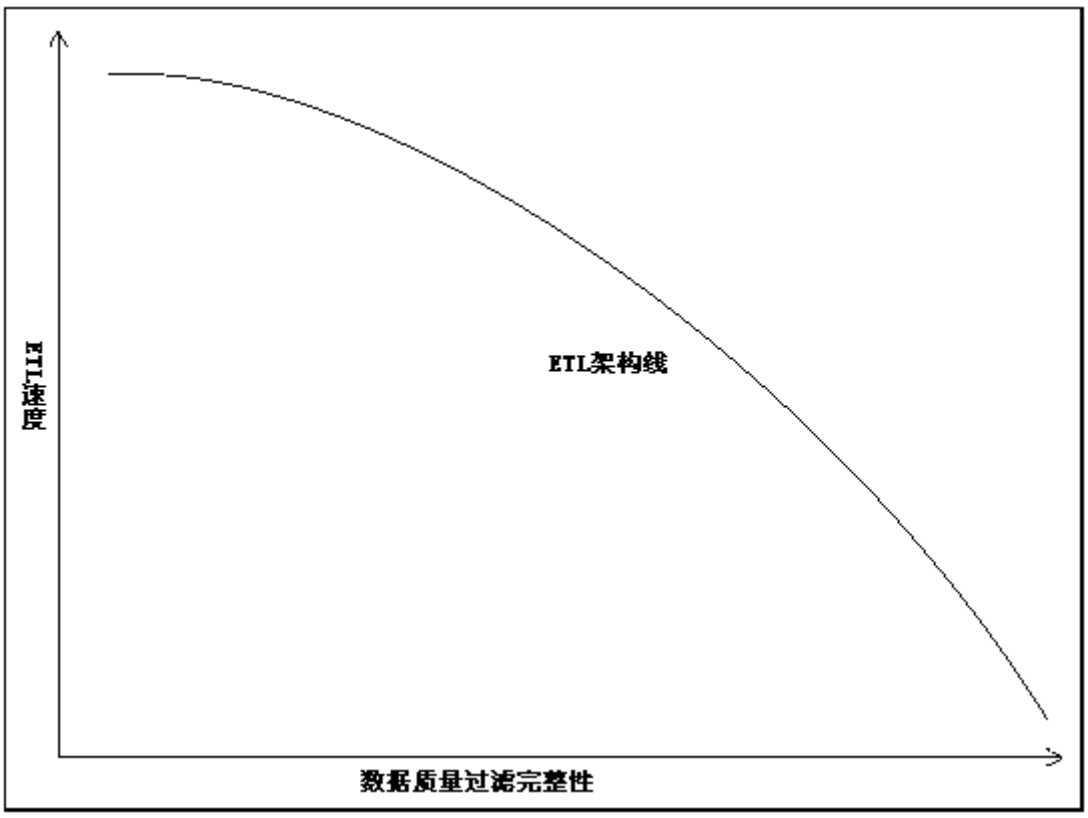

图 4.2　完整性对比速度

一种潜在的有启发意义的实现这种平衡的方法是通过粗略回答一些待建数据仓库中的数据质量方面的潜在问题来找到这个平衡点。比如：

- 在什么点数据变得过时？

以及

- 保证这些数据的正确性有多重要？

如果数据仓库小组必须要作出决定，比如，在数据质量方面更高程度的自信和发布周期

不能多于一天之间，到底该选择哪个？又比如，某个数据仓库每天发布数据，可能会选择画费一整天的时间的延迟来获得更高的数据质量，方法也许是通过更多的差异检验统计或者数据标准化和匹配，或者甚至采用手工的检查或审计。如果周一的交易数据因此而在周三而不是在周二发布，是否是可被接受的替换方式？回答这样的问题并不容易。

校正与透明

数据清洗过程通常用来改正脏数据，但同时会给组织提供一份不加掩饰的原始的视图。这里实现适当的平衡是非常必要的：完全透明的系统会产生一个功能不足的商务智能系统，使得无法深入分析；而过分纠正的系统会掩盖操作上的不足，从而减缓组织的进展。

解决的办法是对不同的错误建立一个可判断的原则界限，这些错误由清洗过程进行纠正或者标识，并生成一个容易使用的审计工具（审计维），用来如实记录那些修改、标准化以及错误检测和数据再造组件的基本规则和设想。

从制造业的质量管理理论中学习

制造业的质量管理已经有至少 30 年的发展历史，最基本的理论来源是 W. Edwards Deming 的全面质量管理(TQM)架构。在考虑数据质量问题时，Deming 的关于管理 TQM 的 14 个要点同样值得思考，虽然这已经超出了本书所论述的范围。但是也许 Deming 的主要观点是生产质量需要整个组织的所有部门都要为质量问题负责，绝不仅是装配线末端的单独一个检查员的任务。

数据质量管理可以从制造业的质量管理中借鉴很多东西，在这方面重要的一个步骤是是 IT 组织中出现了集中控制的数据质量小组。关心数据质量问题的数据仓库小组成员不能独立于数据质量小组单独采取行动。在这一章中我们定义的过滤器可以补充数据质量小组使用的其它过滤器以及评估能力。这将从所有规格数据质量整合结果到一个完整的数据库，而不仅是数据仓库。大多数来自 ETL 过滤器的问题结果都将是要求改善源系统，而不是要求做更多的清洗。而所有的这些改善源系统数据质量的需求都需要由数据质量小组来完成。

阐述策略

图 4.3 所示的是一种方法，用于把数据仓库项目中面对的数据质量挑战进行分类，以及把由 ETL 数据质量子系统处理的内容分隔开来。

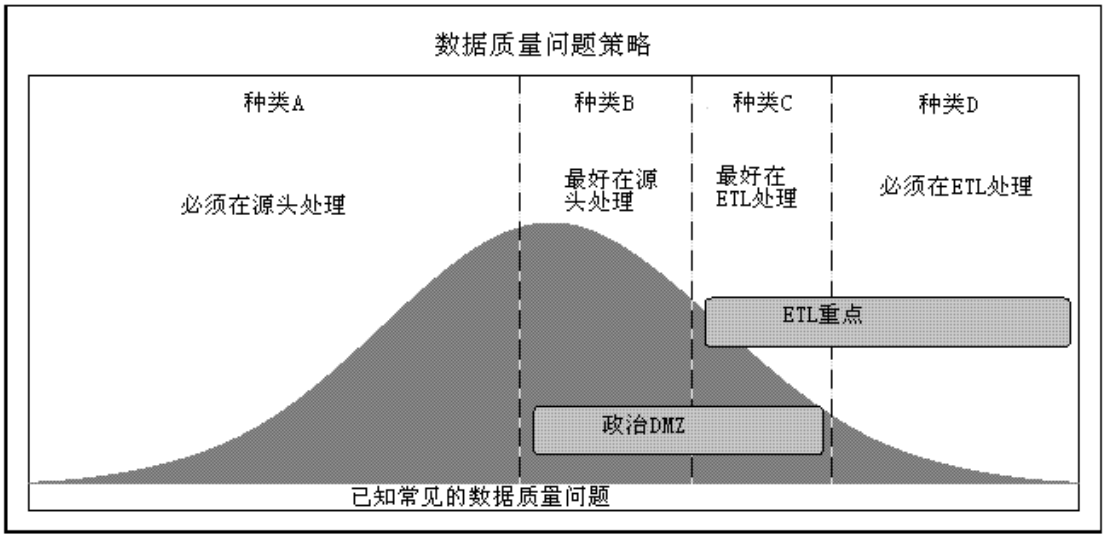

图 4.3　数据质量问题策略

- 不论甚么原因导致的 A 类问题,都必须简单地在数据源进行处理。例如缺失的客户投诉主题或者录入到字段中的主观取得的客户对销售电话接受能力的假信息。还没有什么技术方法来获取或者再造这些信息。它们要么从数据源被正确地获取，要么被遗失。当处理这些 A 类数据质量问题时，清洗子系统应该把他们看作是数据源的缺陷,把所有的明显的假信息从原始报表和分析维和事实中删除，并清楚地标明这些信息是缺失或者伪造，从而使管理工作直接集中在源系统的问题上面。在大多数的数据仓库项目中，大部分数据问题都属于这类问题，所以必须要检测数据质量问题，并与最终用户群体做清晰地沟通。
- D 类(我们先跳过来) 数据质量问题只能在 ETL 系统独自解决。例如来自独立的第三方的数据提供者的缺失的和不完整的信息，这可以通过集成完全修正，另外还有来自固定的操作型源系统的坏数据的修正。D 类问题在大多数数据仓库项目中相对很罕见，ETL 系统被授权创建这些值来来修正数据缺陷，但是必须要保证它的策略和行为能够通过描述的和完善的元数据对用户可见。
- B 类问题应该在数据源处理，即使有一些创造性的方法能推断或再造那些丢弃的信息。A 类问题和 B 类问题的分界线是技术方面的，而不是策略方面的。如果一个给定的数据问题能很有把握地在技术方面解决，那它很明显地是属于 A 类问题范畴。
- 因为种种原因，C 类问题最好被定位于数据质量 ETL 过程处理，而不是在于数据源。同样，C 类和 D 类的界限也是技术上的并非策略上的。如果一个给定的数据质量问题能被合理地数据源在处理，它很明显也是属于 D 类范畴。

经过划分和解决我们的数据质量问题，我们发现真正模糊的唯一界限是在 B 类和 C 类之间：从技术立场上来看，问题既可以在数据源可以在 ETL 系统进行处理。这叫做策略上的 DMZ(非军事区)。

第 2 部份：清洗提交报告

改善数据质量的一项重要任务必须以严格的测量为基础。这应该包括保证你发现的各种数据质量问题类型的记录的精确性，何时发现，看到了什么，以及结果是什么。此外，你需要能够回答来自数据仓库管理员，信息驾驶员和数据质量负责人的关于你做的处理以及发现的数据质量内部的相关问题，比如：

- 数据质量正在逐渐好转或是变得更糟？
- 哪一个数据源系统会产生最多的/最少的数据质量问题？
- 在整个细查数据质量问题的过程宫，有没有有趣的模式或者发现一些趋势？
- 数据质量水平和作为整体组织的效率之间有没有任何明显的相关性？

也许数据仓库管理员会问：

- 在我的 ETL 过程中，哪个数据质量过滤器消耗了最多/最少的时间?
- 对于那些不再数据中出现的已发现的问题类型，是否可以去掉它们的数据质量过滤器？

数据清洗子系统在整个 ETL 过程流中跟随在抽取步骤之后。下面将要讨论的三个主要提交报告包括：

- 数据评估结果
- 错误事件表

- 审计维

数据评估提交报告

数据清洗在构造 ETL 系统的第一个步骤之前就要真正开始。我们已经强烈地建议在开始计划和设计阶段之前执行充分的数据源的数据评估分析。好的数据评估采用一种特定的元数据资料库的描述方式：

- 结构定义
- 业务对象
- 域
- 数据源
- 表定义
- 别名
- 数据规则
- 数据值规则
- 需要处理的问题

数据评估不仅仅是对你的原始数据源的一种好的定量评估方式，这种输出还应该对下面描述的两种可提交的内容有很大的影响。Jack Olson 的书（数据质量：精确维）中附录 B 中有一个完事的列表，内容是前述列表的扩展子类，他们应该提前通过数据评估分析来创建，以此来组成元数据资料库的基础。

清洗提交报告#1：错误事件表

第一个主要的数据清洗的提交报告是一个叫错误事件表的事实表和一组维表。这一提交报告组织成一个维度数据模型,也就是一个维度的星状模型(见图 4.4)。

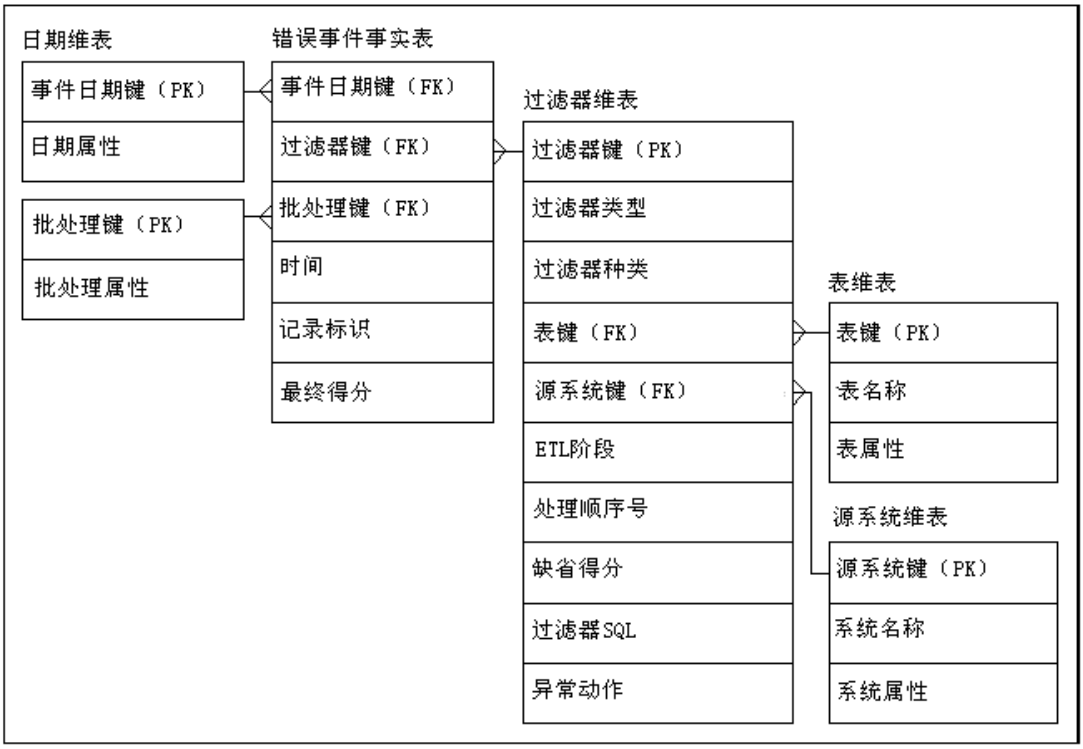

图 4.4　错误事件表结构

每一个被数据清洗子系统发现的数据质量错误或问题都被记录在错误事件事实表中。换句话说，这一个事实表的粒度是每个数据质量检验的每一个错误实例。记住一个质量检验是一个过滤器。所以，如果你运行 10 个不同的针对一些数据集的过滤器，且每个过滤器发现 10 个缺陷记录，总共就有 100 个记录会被写到错误事件事实表。

事件日期是一个标准的维，表示为日历日期。日期时间被从午夜以后按秒数记录在事实表里，以一个整数表示。

批处理维为整个批处理过程的每一次执行创建一个记录，并且通常都包含有意义的时间戳,以及处理的记录数量。

过滤器维包含描述每个已应用的数据质量检查或过滤信息的内容。它不是一次特定运行的描述(那是事实表记录的东西)，而是过滤可以做什么以及应用到哪里。它的属性之一 ,缺省的严重程度得分，为可能遇到的每个错误类型都定义一个严重程度值。 这些对严重程度评分很多作为最后严重程度得分错误事件事实表的基础。例如当海量数据累积起来后，最后的严重程度得分可能是比其缺省值得分高。

过滤器维的属性如下：

- ETL 阶段描述在整个 ETL 过程中数据质量过滤器被应用的阶段。
- 处理顺序编号是一个基本的调度/依赖设计,用于通知整个 ETL 主过程该过滤器的顺序号。在同一 ETL 阶段，拥有相同的处理顺序编号的数据质量过滤器可以并行运行。
- 缺省的严重程度评分即为应用到每个异常的错误严重程度得分，这些异常在压力处理规则之前由过滤器发现。这一缺省评分将增加或减少在事实表中的最终的严重程度得分。
- 异常动作属性将告诉整个 ETL 过程在发现这种类型的错误时是否通过该条记录，或者拒绝该记录，或者停止整个 ETL 过程。
- 过滤器类型和种类名称用来按主题把相关的数据质量过滤器分组，例如完整或者验

证或者越界。

- 最后， SQL 语句捕获用于执行数据质量检验的实际 SQL 片断或者过滤 SQL。如果可行，这 SQL 应该返回数据质量过滤器捕获的所有行的唯一标识符集，这样就能在错误事件事实表中插入新的记录。

为了生成报告,把每个过滤器与其细查的表或一组列相关联是很非常有用的，这样信息质量负责人就能运行确定数据质量问题发生处的报表，并长期跟踪这些问题。这就是在过滤器维中定义表外键的目的。

源系统维用来识别有缺陷数据的数据源。因为数据质量过滤即可以运行在属于单一数据源的集结数据上，也可以运行在那些从很多数据源提取出来的数据，因此错误事件可以与特定的（虚拟的）集成的数据源系统相关联。

那些有问题记录具有唯一标识符，从而可以让错误事件直接地被追踪到问题记录。这些唯一标识符在事实表中表示为退化维，它是由 ROWID 或是其他指向可疑记录的指针构成的。注意这个设计还有一个隐含的功能就是维护过滤器维表的标识和真实记录间的参考完整性。如果你删除了真实的记录,过滤器记录将会变成一个“孤儿”。 过滤器种类字段只是被用于分类那些被过滤器发现的错误类型。可能的数值包括： 不正确的，模棱两可的，不一致的和不完整的,这使得分析师可以对错误事件进行有意义的类。

在 ETL 系统中，错误事件事实表是捕获、分析并且控制数据质量的核心表。 所有来自 ETL 过程的所有错误事件都应该被写到该表中。过滤器维当然就是这一个表的主要驱动器。这一架构是 ETL 系统的主控制面板的基础。

清洗提交报表#2： 审计维

在前面章节描述的错误事件事实表获取数据清洗事件，事件粒度是在 ETL 系统中的任何表中的单条记录。显然，这些事件不可能在最终提交到前端的表的个别记录的粒度上发生。为了把数据质量指示器和最终用户的事实表进行关联,我们需要在这些表的粒度上单独赋值建立一个维，我们把它叫做审计维。审计维描述了被递交到前端的事实表记录的完整数据质量上下文。

审计维按照字面上与数据仓库中的每个事实记录相关联，它将保存 ETL 过程中里程碑的时间戳和输出结果,以及明显的错误和他们在记录上发生的频率，另外还有整个数据质量评分。审计维记录在清洗和规范化事实表记录处理的最后一步创建，并且必须包括应用到该记录的修复描述。

审计维取得每一事实表记录的特定数据质量上下文。 但这通常不会引起审计大小记录的巨量增加，因为审计维的目的是描述遇到的每种数据质量类型。在理想情况下，对于一个将新数据的载入到事实表完整干净的过程，只有一条审计记录产生。如果该过程除了一些因为异常值而触发了越界检查的输入记录外都是干净的，则将生成两条审计记录：一条是为正常的记录和一条是为越界记录。绝大多数事实记录会使用代理键作为正常的审计记录，极少数异常的事实记录会使用代理键作为越界审计记录。

图 4.5 给出了有代表性的审计维设计。

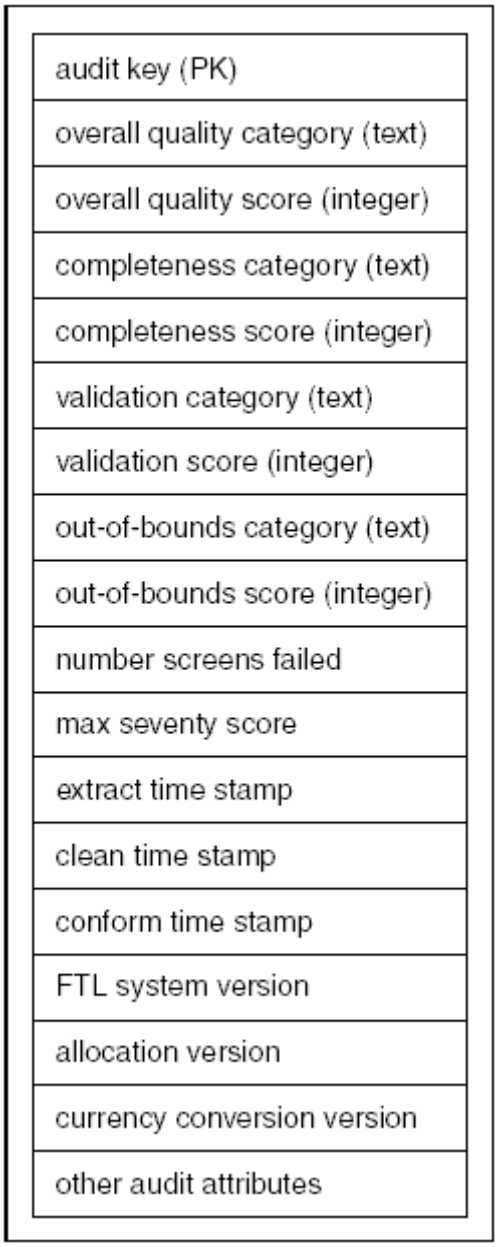

图 4.5　审计维

数据质量属性和其评分是通过为整合的记录以及它的相关数据源系统的记录进行所有错误事件事实的检查来计算。审计维包含一些错误事件事实表的属性，它们是通过累计事实记录的错误评分和与其相关的规范化维实例的评分，以及每条创建了集成维和事实的源记录而得到的。如果你为每个过滤器进行分类，则对于每一个分类的累计数据质量得分都能以本文的和数字的形式在审计维中记录描述属性。文本表格对于错误条件的定性描述的标签报表非常有用。数据质量的完整性，检验和越界，以及图 4.4 中展示的审计维属性等就是这一技术的例子。

同样地，你还可以计算错误事件的总数和针对每一事件的最大严重程度评分，作为审计维中有意义的属性记录。最后，审计维是一个完美的占位符，它记录了所有的时间戳和你已经收集的 ETL 编码的系统属性，用于支持数据仓库经理的分析和解决问题。

也许最大的回报是来自把审计维开放给最终用户群体。最终用户的查询和报表现在既可以在正常模式下执行，也可以在工具模式下运行。只需简单地把审计维的属性之一拖曳进查询和报表中，原始结果就会按照不同的数据质量条件分为多个部分。例如，只需把越界分类

的属性拖曳到销售总额报表中，大量的店铺就会被划分成三行(正常的销售店,异常高的店铺，和异常低的店铺)。

注意我们已经偷偷地将一些全局的元数据语境放进审计维！在表的下面部分是一些全局发布版本和版本号。因此，如果你在报表期间改变了你的收入分配结构,你可以拖曳分配逻辑版本号属性到报表中，它将会把每个结果集的行分开进入到用旧结构和新结构进行计算的两部份。我们在此添加了数据状态的元数据。

审计维的最佳点

目前还没一个能被广泛接受的、比较成熟的计算事实记录全部的数据质量得分的方法。定义一种表示实际验证过的数据质量水平的方法是一种挑战,可以以某种形式执行这样方法，使其能随着时间的推移，对一组数据质量过滤器预先进行调整。毕竟，每次当信息质量负责人要调整过滤器的时候，你不想要在数据仓库中为所有的事实再次调整数据质量评分。例如，如果执行很少的过滤器，实际验证过的数据质量水平要比在之后加入到 ETL 流中更多更全面的过滤器要低。

一种计算所有有效事实数据评分的技术是对与事实相关的所有错误事件记录的错误事件严重程度评分进行加和。当然，这里要假设数据源到目标的 ID 映射是作为 ETL 匹配数据集成功能的一个副产品产生的(稍后在这个章节中将会介绍)。观测事件得分的总和是从一个最坏情况错误得分中减去决定在审计维被用的全部有效数据质量的得分后得到的。最坏情况错误得分所有用于抽取的，清洗的和规范化的数据过滤器的最严重错误得分之和。因此，如果十个不同的过滤器在一个事实表记录和九个维表记录上执行——-每个过滤器都可以生成一个最坏情况，即数据质量严重性得分的 10 分——则所有的最坏情况得分总数是 100。再次声明： 如果每个过滤器在应用的过滤器中都发现缺陷那么总共的数据质量严重程度得分会是 100 。了解了这些，你可能会选择给出完全有缺陷的事实一个零得分，而给没有任何错误的事实得分 100。因此，这一技术提供了针对实际应用的一系列过滤器的数据质量全面的度量方法。如果组织选择在 ETL 过程中增加更多的过滤器，则验证的数据质量得分的就很有可能增加。这似乎很合理,因为组织现在认为它的数据处于一个较高的数据质量水平上。

针对每个事实表能定制唯一的审计维结构。换句话说，你可能选择创建一个审计维设计家族，而不是强制所有审计维包含相同的信息。这将使得对单独事实质量的单独诊断记录到一个单独的审计维记录中，这里的关键是要保存这一个表的维度信息。

该部分已经讨论了审计维的设计，其中描述了属于事实表记录的数据质量诊断和所采取的动作相关的内容。这样，它为所有的事实表建立了一个清晰的维度模型。但是是否可能为维表设置一个审计维呢？我们的回答是“不”；你也不需要这样做。我们更喜欢在维表中直接嵌入数据质量诊断和动作。针对整个数据可靠性的数据质量诊断应该作为附加字段包含在维表中。类型 1 对维信息的修改（覆盖）也可用这种方式描述。类型 2 的修改(在一个特定的时间点对属性进行变更) 则使用面更广，包括时间戳和原因代码,这能完成个别的审计维的很多目的。如果需要针对数据仓库结构的所有变化的完整审计轨迹来生成报告,则你需要设计特定的结构来分别记录这些变化。

第 3 部份： 过滤器及其度量

我们现在准备好做一些详细的设计。该部分是在大多数数据清洗引擎的核心描述了一组基本的检验和测试。它描述了这些功能的作用,如何做，以及它们如何互相作用以提交清洗好的数据到维度数据仓库中。我们非常感谢 Jack Olsen 创造了在后面部分介绍的架构和词汇表，这些描述在他的著作中：<<数据质量，精确维度>>。

流程检查

规划与设计：需求/现状 -> 架构 -> *实现* -> 测试/发布
数据流：抽取 -> *清洗* -> 规格化 -> 提交

异常检测阶段

数据异常指的是那些不适合和其它数据放到一起的数据。还记得当你还是小孩子的时候，有人会给你一张图片，然后问你“这图片有什么问题吗？”。你会指出车轮是方的或者停止标志挂颠倒了之类。数据异常就是数据库中是方形车轮。发现这些异常需要有特定的技术而且需要仔细地分析研讨。在该部分我们将介绍已经在我们的数据仓库项目中被证明是成功的异常检测技术。

期待的时候该期待什么

在 ETL 过程创建时暴露数据异常是导致 ETL 部署延时的主要原因。发现数据异常会花很多的时间和分析。提前做这样的分析,可以节省时间而且减少犯错。较好的方式是花时间一次又一次重建同一 ETL 作业，此时可以尝试去纠正由于没有被发现的数据异常导致的错误映射。

可以通过一些发生在数据仓库以外的数据质量问题来发现数据异常。但是发现数据异常通常是 ETL 团队的职责,除非你的项目已经制定了成熟的数据质量分析子系统的预算。

数据采样

最简单的检测异常的方法是对有问题的列进行分组时计算该表的行数。这个简单的查询，结果展示在图 4.6 中，揭示了值的分布和可能损坏的数据。

```
Select state, count(*)

From order_detail

Group by state
```

如图 4.6 所示,数据异常立即暴露了出来。在结果集中的异常就是数据异常，强烈建议通知给业务负责人，以便在源系统中进行整理。

State	Count(*)
Rhode Island	1
Missisippi	2
New Yourk	5
Connecticut	7
New Mexico	43,844
Vermont	64,547
Mississippi	78,198
Utah	128,956
Wyoming	137,630
Missouri	148,953
Rhode Island	182,067
Minnesota	195,197
North Dakota	203,286
Michigan	241,245
Washington	274,528
Pennsylvania	287,289
Montana	337,128
Louisiana	341,972
Virginia	346,691
West Virginia	359,848
Delaware	422,091
Iowa	456,302
Massachusetts	480,958
Tennessee	483,688
New York	494,332
Nevada	506,235
South Dakota	514,485
Indiana	553,267
Connecticut	562,436

图 4.6　值分布查询结果

技术点：数据评估工具正好就是用于执行这种类型数据采样的。

分析源数据听起来很容易，是吧？但当你的源表有数亿行，并有 250,000 个不同的取值时会怎么样呢？分析海量数据源的最好方法是的数据采样。我们已经为采样数据使用过许多不同的技术,从简单地选择开始的 1,000 行到使用最精细的算法，没有哪一个是效果很明显的。我们发现在如下的查询中，只是简单计算表的行数，而且均匀地把表分成几个特定的部分，就可以准确地采样，而不用考虑表中数据的值：

```
select a.*

from employee a,

(select rownum counter, a.*

from employee a) B

where a.emp_id = b.emp_id and

mod(b.counter, trunc((select count(*)

from employee)/1000,0)) = 0
```

为了检验更多或更少的数据，只需改变想要采样返回的行的数目 1,000 即可。

另一种方式是把一个列的随机数目加入到数据中,这可以用来排序选择整个表中任意需要的部分。

一旦你得到这些采样数据，你可以进行常用的值分布分析。通过任意其它方式选择数据，包括选择全部数据，可能会歪曲你的测试结果。

我们已经注意到一个常用的错误，那就是选择一个日期区间来缩小结果集。数据异常通常发生于应用程序的缺陷中或者由于未经训练的操作员的误操作。我们已经碰到的大多数的异常都是临时发生的；则纠正应用程序或者更换操作人员，都会造成异常消失。因此在一定日期区间中选择数据可以很容易的错过这些异常。

约束的类型

把各种不同类型的数据质量检查分为四个大的类型是有用的：

- 列属性约束
- 结构约束
- 数据约束
- 值约束

列属性约束

列属性约束保证由源系统输入的数据包含系统期望的数值。较有用的列属性约束检查包括如下过滤器：

- 检查列的空（Null）值
- 超出期望的最高和最低范围的数字值
- 长度超长和超短的列
- 包含有效值列表之外的数值
- 匹配所需的格式或一组格式
- 在已知的错误值列表中命中数，之所以用这个列表是由于可接受值的列表太长了
- 拼写检查器

一些特定的过滤器技术将在这一个章节的后面部分介绍，这些技术用于执行精确地有效性的检查和异常捕获。基于这些过滤器的检查结果，ETL 作业流可以选择：

1. 让没有错误的记录通过
2. 让记录通过，同时标记有错误的列值
3. 拒绝记录
4. 停止 ETL 作业流

一般的情况会选择第 2 项,即让记录通过 ETL 过程流并记录任何遇到的已验证的错误到错误事件事实表中，使得这些错误对最终用户可见，从而避免由于虚假的完整性而造成数据仓库的可信度的损害。有很严重缺陷的数据记录是不可能出现在数据仓库中，或者会严重损害数据仓库可信度，所以可以被完全的忽略，当然也要把错误事件适时的记录到错误事件事实表中。最后,影响整个 ETL 批处理过程的数据完整性的数据验证错误应该完全停止批处理过程，这样数据仓库经理能更进一步地进行调查。过滤器维包含一个异常动作列，它与每种

过滤器的这三种可能动作相关联。

结构约束

列属性约束重点是在单独每个列上,但是结构约束却是把重心集中在列之间的关系上。我们通过确认每个表有合适的主键和外键，并且遵循参照完整性来约束结构。我们检查每组字段里显性的的和隐性的层次和关联关系，比如构建一个合法的邮件地址。结构约束还检查层次父子关系，来确定每个子节点有一个父节点或者在一个家族中是根节点。

数据和值规则约束

对于数据和值规则域的范围，从简单的业务规则，比如如果客户处于优先状态,则透支上限至少是$1000，到更复杂的逻辑检验,比如一个商业客户不能够同时是一个有限责任合作伙伴和类型 C 公司。值规则是这些数据上合理性检验的扩展，并且可以用聚合值业务规则的形式，例如一个临床医师正在报告需要 MRI（核磁共振成像）的挫伤手肘者的不可能的数目。值规则也能提供一个数据可能不正确的概率警告。确有叫做 Sue 的男孩，至少在 Johney Cash 的歌里，但是也许这个记录应该被标记以等待检查。如果这个记录是不正确的,你不会知道是名字还是性别应该被改正。

这些发现都很难包含在错误事件事件表中，因为这些错误包括许多记录。要确定哪个记录不正确是几乎不可能的。这样就只剩下两个选择：把所有这些记录（例如需要 MRI 的挫伤手肘者）标记为有问题，或者创建一个虚拟的聚合表，在这个表里错误可以用发生的次数来表示。

驱动过滤器设计的度量

流程检查

规划与设计：需求/现状 -> 架构 -> *实现* -> 测试/发布
数据流：抽取 -> *清洗* -> 规格化 -> 提交

这一个部分讨论进入数据仓库的数据清洗的基础都需要做些什么，包括简单的探测、捕获、定位普通的数据质量问题的方法，以及提供给企业整个数据流的可视化能力和随着时间的进展数据质量逐步得到改善的各个方面。

整个处理流

一系列的数据质量过滤器或错误检验程序按照元数据中定义的规则排队等待执行。每个过滤器都描述在创建的过滤器维中，这同时也作为本章前面部分错误事件结构的一部分。当过滤器执行的时候,每个错误的发生都记录为一个错误事件记录。 每一错误检查的元数据也会记录了错误事件的严重程度。最严重的数据质量错误被归类为致命错误，它将会引起整个 ETL 过程中止。 举一个发生致命错误事件的例子，比如来自多个商店的日销售数据完全丢

失，或者在一个重要列中出现了一个不可能的非法数值，而针对这个值没有相应的转换规则。

当每一个数据质量检验已经执行后，就会在错误事件事实表中查询在整个数据质量处理过程中遇到的致命事件。 如果没有发现，继续正常的 ETL 过程；否则向整个 ETL 处理返回暂停指令,然后将按次序执行 ETL 停机过程，并且通知数据仓库系统管理员或信息质量驾驶员。这个过程描述在图 4.7 中。

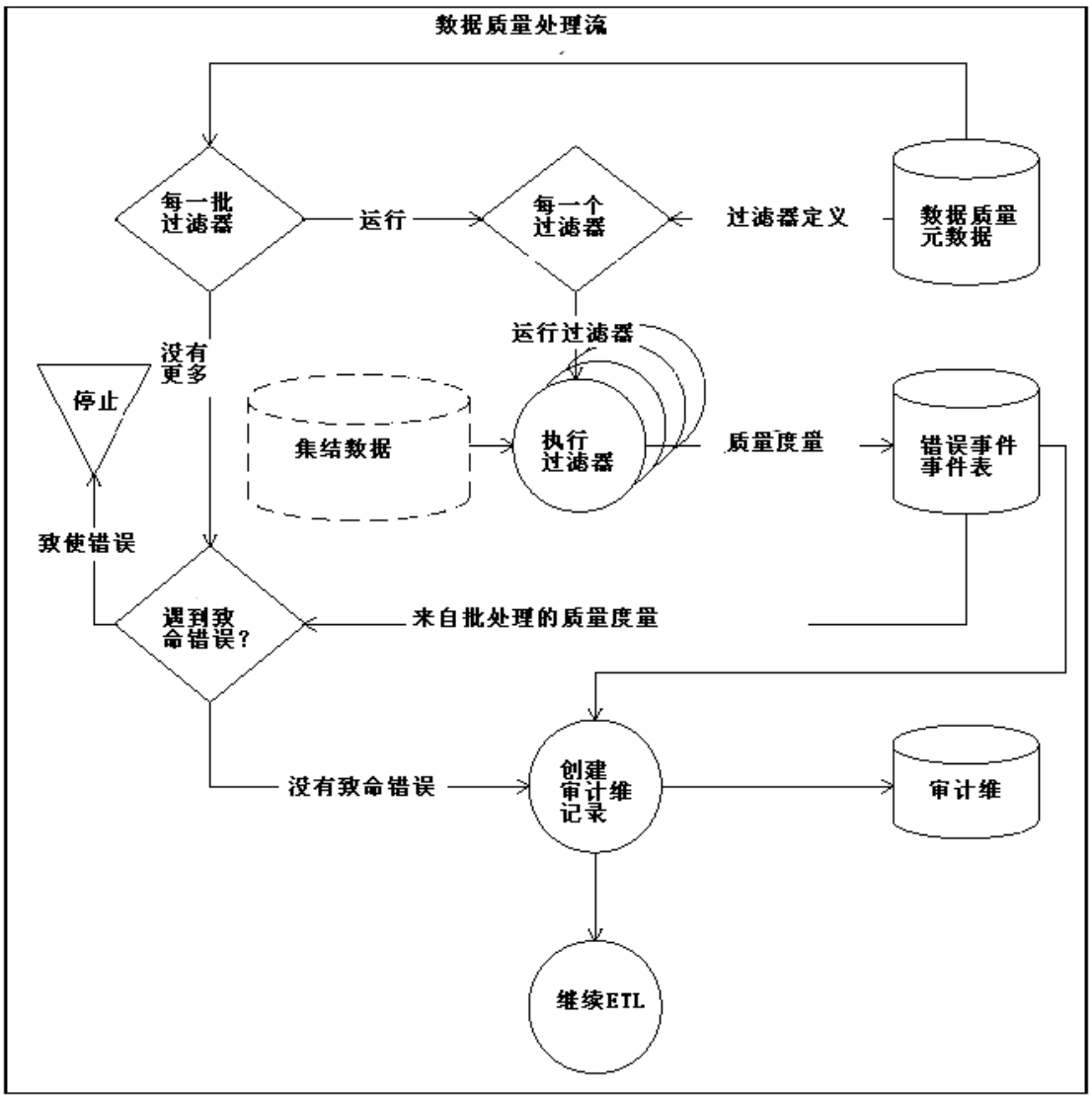

图 4.7　整个处理流

为了更好的性能，数据清洗子系统的处理流程的目标是触发可以并行运行的批过滤器。这些过滤器识别出数据质量问题，并在错误事件事实表中插入记录。为了最大程度减少数据库的冲突问题，应该在错误事件事实表上避免不需要的索引或者约束，以便当从多个过滤器处理同时写记录时不会引起问题。调用过程在触发下一个批过滤器前要等待每一个批过滤器执行完成，直到没有剩下的批过滤器要执行。正如在本章前面所述的，过滤器元数据表中的处理顺序号是用于调度过滤器的。具有相同的处理顺序号的过滤器能并行执行。标准的数据仓库作业调度工具也可以用来做更复杂的过滤器调度并管理他们之间的依赖关系。

当清洗子系统完成清洗处理并规范化记录后，还需要做一些额外的工作，为审计维给出整个数据质量得分。这通过把在流程中清洗和规范化过的记录的错误事件事实和那些相关的源记录（如果这种关系是可用的)进行聚合来实现，并作为 ETL 集成/匹配过程的附属品来保存。有意思是，过滤器也能被应用到错误事件事实表本身,可以创建一些特定的过滤器，来

度量在整个数据清洗作业流中任何阶段积累起来的数据质量错误的数目和类型。这项技术将在下一部分中进行更进一步的描述。

建议的运行过滤器的方法是建立一个通用的软件模型，它可以执行任何过滤器，只需要设置批处理 ID 和过滤器代理键作为参数即可。这一模型为过滤器抽取元数据并构造一个动态的 INSERT 语句，它将为每一个过滤器返回的有问题记录组装错误事件事实表。动态 INSERT 语句的通用形式如下：

```
INSERT INTO data_quality_error_event_fact

(etl_batch_surrogate_key, day and time of day surrogate keys,

list of values from the Screen Meta Data record,

offending_record_surrogate_key)

SELECT offending_record_surrogate_keys provided by the Screen's SQL

Statement
```

持续运行

数据清洗子系统的指导原则是发现并且记录存在的数据质量错误,而不是跳过记录或者是停止 ETL 过程。数据质量问题在数据仓库生命周期中是一个不幸并且存在的事实，业务经理不得不面对不完善和不准确的数据每天作出艰难的决定。这种情形不是一个晚上就能解决的。相反，应该为整个组织提供一些工具来精确度量正在使用的数据的质量，并在整个提高数据质量过程中了解其过程。

也就是说，数据清洗子系统也必须提供一些处理意想不到情况的机制，这些情况既包括那些由于太严重的缺陷而不被允许进入数据仓库的记录，也包括那些很严重的系统级的缺陷，能导致整个 ETL 过程停止的记录。由于实际的和策略的原因，触发这些异常的应对措施的阈值必须进行平衡，既要使数据仓库保持对业务来说是一个有用的可行的工具，又要提供足够的数据质量保证，从而对于最终用户来说保持系统可信性。这是一个困难的平衡，而且要随着时间变化不断的调整。因此 ETL 数据质量子系统应该支持调节这些阈值的能力，并当遇到数据质量错误时修改相应的动作。

在一些情况下，如果检测到过多低级错误，就要采取异常动作。举例来说，一个客户地址记录中无效的美国州的邮编通常会有一个错误事件写到数据质量主题域,但不会终止整个的 ETL 过程。但如果一批的记录中都有非法的美国州邮编,则可能表示在上一级的处理过程过程中有严重问题，严重到影响 ETL 处理流中所有数据的完整性。针对这种情况，建议的处理方式是创建额外的数据质量过滤器，让它直接地针对错误事件事实表运行，计算在整个数据质量批处理中捕获的数据质量错误事件记录的数目，并且触发异常处理程序。

注意这些特别的过滤器，它们是把发现的错误写回到错误事件事实表中。它们读写同一张表中，这也容易造成数据库冲突。与其它大多数数据质量过滤器处理方式不一样，它并不是针对每个有问题的记录都写到事实表中，而是统计来自特定表的某种特定类型的错误数，并只有当统计值超过了允许的上限时才写一条记录到错误事件事实表。这样将可以避免大多数冲突问题。

过滤器

在运行过滤器之前,应该要先建立一个全局的数据评估基本规则，其中包括定义列的一些属性，如无效值，数字列的范围,字符列长度限制以及表大小。对于一个又一个数据源,没有比深入研究数据本身来确定高质量样本数据的特征更合适的方法了。这一研究过程应该包括数据提供者的技术文档研究和源数据本身逐列查看。对于每个被载入数据仓库的数据源，一个数据评估的检查列表应该包括：

- 为抽取的表提供每天记录数的历史
- 提供每天的关键业务度量的总数历史
- 确定需要的列
- 确定需要唯一的列集合
- 确定允许(以及不允许)为空的列
- 确定数字字段的可接受数值范围
- 确定字符列的可接受的字符长度
- 为所有列定义明确的有效数值集
- 确定列中无效值(不在有效值集合中)出现的频繁程度

如果没有专用的数据评估工具，数据评估的好处通过手写 SQL 也可以实现部分，只是这需要一支专家队伍，以及大量的时间和辛苦的努力。这和在本书一开始就讨论的手工方式和工具方式的区别以及为何要选择完整的 ETL 工具相对应。换句话说，厂商提供的工具正在逐渐地抬高这一门槛，这使得自己开发方式越来越不能满足需求，除非的需求相对非常简单。数据评估实验的结果应该由信息质量负责人来维护，因为他接下来可以直接地把它们运用到数据质量过滤器的元数据定义中，而这可以驱动 ETL 的数据质量进程。

已知表行数

在某些情况下,信息质量负责人通过业务规则完全可以知道来自一个给定数据提供者某种数据类型的记录数。举个例子来说，比如一个仓库每周的库存清单，其中每个部分都会被记录，即使是零。在其他的情况下，信息质量负责人可以基于历史信息指定来自某个数据提供者的可接受的记录数范围，并创建过滤器来检测过高或过低的记录数。已知表的记录数工作能用简单的过滤器 SQL 来处理处理，如下所示：

```
SELECT COUNT(*)

FROM work_in_queue_table

WHERE source_system_name = 'Source System Name"

HAVING COUNT(*) <> 'Known_Correct_Count'
```

因为这是一个表级的过滤器,错误事件事实表中清洗过或者规范化过的识别符应该置为空。

列空值

决定数据记录中那些列是非空的（与允许为空相比）是非常重要的，同时通常随着数据

源系统改变而变化。举例来说,一个销售网点的应用系统可能允许客户地址属性为空，但是一个物流系统客户地址属性就必须是非空的了。元数据结构的目的是基于每个数据源捕获空值规则。在维度模型中，完整的记录往往比源数据有更严格的空值规则。因为几乎所有维的属性列都要求非空，即使仅是“未知”、“不可用”或“不能用”之类的字符串也可以。

系统自动给空的文本字段加上一个实际的值可以消除二义性，不管这个字段值是丢失了还是真的为空。这一技术也简化很多 SQL 查找表；不幸地是，关系数据库处理空白文本字段的方法和空值文本字段的方法是不同的。 即使没有给空文本字段赋值，我们也建议至少要转换所有的空文本字段为空白文本字段。

检测空值的建议方法是建立一个指定源的空值 SQL 语句库,它返回有问题的记录的唯一标识符，如下所示：

```
SELECT unique_identifier_of_offending_records

FROM work_in_queue_table

WHERE source_system_name = 'Source System Name'

AND column IS NULL
```

对于来自完整记录的过滤器错误，应该稍微调整 SQL，在其中使用特定的源系统名称，如下：

```
SELECT unique_identifier_of_offending_records

FROM work_in_queue_table

WHERE source_system_name = 'Integrated'

AND column IS NULL
```

过滤器返回的每个有问题的记录都会被插入到错误事件事实表中，而且那些有问题的行的唯一标识符也被作为退化维写入到事实表中。

列的数字和日期范围

虽然许多数值和日期在关系数据库表中都可以接受很大范围的值，不过从数据质量的角度来看,它们应该加以更严格的有效值范围限制。如果单一某个客户交易了一百万个单位，这可信吗？也许是的，如果我们的生意是一个全球的 B2B 交易，但是如果这是消费者零售点交易则就不可能了。你一定想要你的数据清洗子系统能够发现并且记录包含在信息质量负责人定义的合法数值范围之外的实例。在某些情况下，尤其是那些第三的 ETL 计算使用到的数字列，这些数值范围将会由信息驾驶员定义。在这里再一次提醒，整合数据的列可能有一些和其他任何数据源都不相同的合法的数值范围，因此，你需要用不同的过滤器来验证它们。一个过滤这些潜在的错误的 SQL SELECT 语句的例子如下所示：

```
SELECT unique_identifier_of_offending_records

FROM work_in_queue_table

WHERE source_system_name = 'Source System Name'
```

AND numeric_column IS NOT BETWEEN min AND max

假如我们有一个事实表，它跟踪 **600** 个商店的每日销售额，其中每个商店有 **30** 个部门。 这样每天收到 **18,000** 个销售额记录。使用一个以标准偏差为基础的快速统计检测，可以让我们判断这 **18,000** 个销售额数目的合理性。 这一技术也让我们很快地更新统计基值，从而为明天数据的载入做好准备。

记住：标准偏差是差值的平方根。差值是每一个历史数据点和数据点的平均数之差的平方的总和，再除以 **N-1**,这里 **N** 是数据的天数。不幸的是，这一个公式需要我们遍历销售历史的全部时间，虽然可能，但是在快速运行的 **ETL** 环境中，这样的计算毫无吸引力。但是如果我们已经掌握 **SUM SALES** 和 **SUM SQUARE SALES**，我们能把方差的计算公式写出这样：**(1/(N-1))*(SUM SQUARE SALES - (1/N)*SUM SALES*SUM SALES)**，检查公式吧**!**

因此如果用 **VAR** 缩写我们的偏差公式，则数据有效性检查就如下所示：

```
SELECT s.storename, p.departmentname, sum(f.sales)

FROM fact f, store s, product p, time t, accumulatingdept a

WHERE

(first, joins between tables... )

f.storekey = s.storekey and f.productkey = p.productkey and

f.timekey = t.timekey and s.storename = a.storename and

p.departmentname = a.departmentname and

(then, constrain the time to today to get the newly loaded data... )

t.full_date = #October 13, 2004# and

(finally, invoke the standard deviation constraint... )

HAVING ABS(sum(f.sales) - (1/a.N)*a.SUM_SALES) > 3*SQRT(a.VAR)
```

我们扩展先前介绍中的 **VAR** 并使用 **a.**做 **N**，**SUM SALES** 和 **SUM SQUARE SALES** 的前缀。并且已经假定部门按产品分组，因此可以在产品维中使用聚合值。

在这一个模型之上包括两个以下两个查询的运行：高于平均值三个标准偏差以上的销售额和低于平均值三个标准偏差以下的销售额。也许对于这二种情形有不同的解释。如果在 **SQL** 中不喜欢写这样的 **HAVING** 子句，还可以不使用 **ABS** 函数。如果通常情况下每天得到波动的销售额(比如，周一和周二比起周六来低很多)，则可以加入一个 **DAY OF WEEK** 字段到聚合的部门表中，并且约束到合适的日范围。这样就不会把正常的每天波动状况混杂在标准偏差测试中了。

当用前面的 **SELECT** 语句完成输入数据检查的时候，就可以更新 **SUM SALES** 和 **SUM SQUARE SALES**，只要把今天的销售额和今天的销售额平方分别加到聚合的部门表中即可。

列长度限制

文本列的字符串长度的过滤对于 Staged 的和完整的记录错误都是很有帮助的。例如，可以检查客户的名字，过滤掉确定太长或者太短的值。如下的 SQL SELECT 语句就是执行这样过滤的例子：

```
SELECT unique_identifier_of_offending_records

FROM work_in_queue_table

WHERE source_system_name = 'Source System Name'

AND LENGTH(numeric_column) IS NOT BETWEEN min AND max.
```

列的明显的有效值

对于一个给定的由它的源系统定义的离散的有效值集合的列，可以在处理的列中查找未知的值来过滤异常。同样，也可以针对来自任何数据提供者的列使用有效值的通用列有效性参考表来，从而把它作为 staging 过滤器来处理。因此,代表性的 SQL 语句如下所示：

```
SELECT unique_identifier_of_offending_records

FROM work_in_queue_table Q

WHERE source_system_name = 'Source System Name'

AND column NOT EXISTS

( SELECT anything

FROM column_validity_reference_table

WHERE column_name = "column_name"

AND source_system_name = 'Source System Name'

AND valid_column_value = Q.column_value

)
```

列的明显的无效值

对于一个给定的列，经常含有一些已知不正确的值，并且这里并没有离散的有效值集合，那么信息质量负责人可能选择使用这些无效值的过滤器。例如在一个客户名称字段中，“未知”这样的字符串会经常出现，但这里并没有定义全部的有效客户名称集合。则这些明显的无效值过滤器显然并不是试图完全过滤出所有的无效数值，而只是选出那些经常出现的错误。像名字和地址的标准化及匹配这样的其他数据清洗技术，应该更适合这些工作。下而是一个含有问题字符串的过滤器 SQL 语句。

```
SELECT unique_identifier_of_offending_records
```

```
FROM work_in_queue_table

WHERE source_system_name = 'Source System Name'

AND UPPER(column) IN ("UNKNOWN", "?", list of other

frequent offenders... )
```

更好的方法是把数值与记录了频繁出现的无效值的表进行比较，如下所示：

```
SELECT unique_identifier_of_offending_records

FROM work_in_queue_table Q

WHERE source_system_name = 'Source System Name'

AND EXISTS

( SELECT 'Got One' FROM Table_Of_Frequent_Offenders WHERE column_name =

Q.column_name)
```

如果一个列的有效数值的集合太大而无法明确定义或者是未知的，那么这类过滤器只拥有有限的值集合，但是在某些有用的情况下可以使用最近发现的错误规则；并不断重复这些错误规则。

检查表行数的合理性

这一类的过滤器功能强大但是实现也更复杂。它试图确定来自某个数据源的行的数目是合理的，也就是说行数落在基于先前历史有效的记录数的有限范围内。为了测试表行数的合理性，可以选择使用一些简单的统计测试方法，比如根据前一个类似值的平均值计算标准偏差值，或者是选择更高级更专业的预测值，例如 X.12 标准，或者甚至像 ARIMA(自回归的整合移动平均)这样的技术。如果你对这样功能强大的统计工具感兴趣，则你将需要一个优秀统计师几个星期的咨询。一个正好完成这样的统计的地方是在市场研究部门，如果有如此一个部门的话。

如图 4.8 所示的数据 staging 表记录数表每天从每个数据源的每张表中获取记录数，每天每个数据源对应一行。

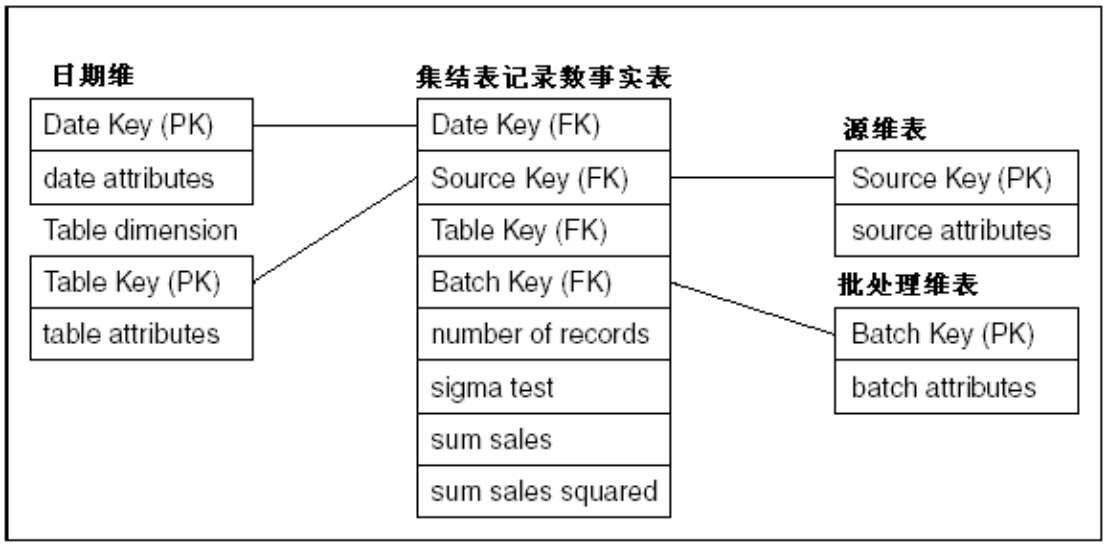

图 4.8　表级实用元数据

图 4.8 用维度结构展示了这些表。一些 ETL 工具在正常操作过程中作为副产品生成这样的表。使用数据 staging 表记录数表，这一个过滤器可能需要两路 SQL 来处理，如下所示：

```
SELECT AVERAGE(Number_of_Records). 3 * STDDEV(Number_of_Records),

AVERAGE(Number_of_Records) + 3 * STDDEV(Number_of_Records)

INTO Min_Reasonable_Records,

Max_Reasonable_Records

FROM data_staging_table_record_count

WHERE source_system_name = 'Source System Name"

;

SELECT COUNT(*)

FROM work_in_queue_table

WHERE source_system_name = 'Source System Name"

HAVING COUNT(*) NOT BETWEEN

Min_Reasonable_Records AND Max_Reasonable_Records

;
```

聪明的 SQL 高手可以实现前面提到的过滤器，既可以使用多路 SQL(如前面 SQL 所示)，其中每个 SQL 都针对一个数据源，或者使用单个过滤器根据特定的数据质量需求和需要的严重程度得分灵活性来验证所有数据源的表记录数合理性。信息质量负责人也可以根据应用的不同的标准偏差容许度和不同的严重程度得分来为相同的表和源系统定义多个过滤器，举例来说，记录偏移平均值两个标准偏差的严重性错误为低，评定偏移平均值三个标准偏差为高严重性错误，并且如果偏离四个标准偏差，需要彻底地停止整个 ETL 过程。

表行数过滤器能很容易被扩展到数据仓库的任何额外的度量上，以支持合理性检测。举

例来说，添加一个总销售额度量到图 4.8 的表，当销售额严重偏离时，过滤器就能记录下这一情况：

```
SELECT AVERAGE(Total_Sales_Dollars). 3

* STDDEV(Total_Sales_Dollars),

AVERAGE(Total_Sales_Dollars) + 3

* STDDEV(Total_Sales_Dollars)

INTO Min_Reasonable_Sales_Dollars,

Max_Reasonable_Sales_Dollars

FROM staging_table_record_count

WHERE source_system_name = 'Source System Name"

;

SELECT SUM(Total_Sales_Dollars)

FROM work_in_queue_table

WHERE source_system_name = 'Source System Name"

HAVING SUM(Total_Sales_Dollars) NOT BETWEEN

Min_Reasonable_Sales Dollars AND

Max_Reasonable_Sales_Dollars
```

检查列分布合理性

具有发现数据维属性偏离了常态分布能力的过滤器是另外一种功能强大的过滤器。 当一个拥有一组离散的有效值集合的列由异常偏离的数据分布组成时，这个过滤器就能够发现并且捕获这一情况。举例来说，被过滤的列可能是在来自自动售货系统(SFA)的一个销售电话事实表的产品列表。假设历史经验证明大多数的销售电话都集中在产品 A(这一商品有较高的退货率) 而只有少许电话集中在产品 B(退货率低的产品)。你需要设计一个过滤器来提醒信息质量负责人，以针对那些突然产品 A 的销售电话减少或者产品 B 的销售电话增多时的情况。

可以使用和前面所述的表行数合理性技术来创建这一过滤器。另外，还需要一个 staging 表来保存历史记录数和随时间变化的列的有效记录数，由此还可以为过滤器计算平均值和标准偏差。因为对一个给定的列和许多有离散分布有效值集合的列，可能会出现很多值，因此需要偏离你的元数据标准和为表及列集合设定的计划 staging 表，并由过滤器进行审查。当然，这样就增加了数据 staging 表的数目，但是它使得 ETL 架构在物理实现这些有潜在的大表的时候显得更加灵活。某些情况下，这种非通用的数据 staging 方法创建的表会过大而不适用于高效的 ETL 过程，因此就可以使用前面提到的判断数据的平均值和标准偏差的统计技术。

注意前面提到的统计方法同样可以支持多列过滤，即测试多个列组合的有效值的合理

性。在本章前面前面，我们把它叫做值规则约束。例如，按产品和商店来审查每天的销售额，或按产品、商店和星期审查，找到偏离历史标准的不合理结果。

修改图 4.8 中的表，加入产品作为一个维，使得我们可以得到按产品的每天销售电话次数。使用这个表，过滤器能按历史的产品编码和源销售电话总数的平均值和当前的 ETL 批处理的值相比较。 对于那些平均超过已经建立的标准偏差 (正如在过滤器定义的 SQL 语句中中的定义) 的阈值的那些产品应该记录错误事件到事实表中。

处理这类使用提及过的技术的过滤器，需要由主流的过程性 SQL 语言扩展很好的支持的级别上进行过程性编程。这一过程性 SQL 可以能被包含在过滤器 SQL 语句定义中或在它之外处理。重要的是，ETL 构架在为所有过滤器维护过滤器元数据实例以及为所有过滤器发现的所有错误事件生成错误事件事实表方面需要具有连续性。

无论选择用什么方法实现，由过滤器创建的错误事件事实都被认为是表级的过滤器，因此错误事件事实表的清洗/规范化记录标识符应该为 NULL。

通用的数据和值规则合理性

数据和值规则正如本章前面所定义那样，是与主题密切相关的，因此我们不能够给出一个实现的特定检测清单。但是清晰的合理性查询与本节中例子中给出的简单数据列和结构检验很相似。

第 4 部分：规范化报表

数据集成意味着创建规范化的维，以及通过组合来自多个数据源的最有效信息为一个综合的视图来创建的事实实例。为了实现这些，导入的数据需要结构统一，并过滤掉无效纪录，将内容术语标准化，删除重复记录，从而产生新的规范化后的映像。在这部分，我们将按三个构建规范化维和事实的步骤来描述：

- 标准化
- 匹配和删除重复记录
- 生存

当我们规范化数据时，我们可以把来自三个数据提供者的性别代码(M，F)，(M,W),或(Man，Woman)转换为标准的性别维属性(Male，Female)。同样我们也可以用专门的工具来规范化姓名、地址等信息。

跨越多个数据源、多个数据集市，以及访问分布式的数据仓库的多个远程客户端来规范化描述属性，对于数据仓库架构师和 ETL 团队来讲是一个关键的开发步骤。在其它一些工具箱书籍中，也有许多关于这一主题的技术性和管理性的描述。ETL 团队所关心的是捕获重复的和不一致输入信息，并支持维表管理员和事实表提供者的需要。

规范化维

不管硬件构架如何，数据仓库从某种程度来说都是分布的，因为每种类型的度量总是必须存在于各自的事实表中。在 ER 模型中相同的语句是真实的。因此，对于从多个分离的事实表组合数据的最终用户应用来说，我们必须为这些事实表提供统一的界面，这样数据才可

以被整合。我们称这类一致的界面叫做规范化维和规范化事实。

规范化维对于每一个可以被关联的事实表来说都是相同的。通常，规范化维对每一事实表都是相同的。规范化维的一个更准确的定义是：

如果可以共享一个或更多的取自同一个域的值的属性，那么这两个维是规范化的。当使用规范化维在不同的事实表间钻取时，应用请求必须使用这一相同的属性来作为约束和分组的基础。

图 4.9 举例说明了支持规范化产品维的三个事实表间的钻取操作。

来自规范化维的属性 ↓

	事实表 1	事实表 2	事实表 3	计算列
Product	Manufacturing Shipments	Warehouse Inventory	Retail Sales	Turns
Framis	2940	1887	761	21
Toggle	13338	9376	2448	14
Widget	7566	5748	2559	23

过程：1）打开到每一源的单独连接
2）分析所有3个答案集
3）合并答案集到规范化的行标题（product）

图 4.9　在三个事实表间钻取

经常需要规范化的维的例子包括：客户维、产品维、地理维、促销维和日期维（时间）。中心数据仓库设计团队的主要职责是创建、发布、维护和约束规范化维。

创建规范化维对于一个组织来说是非常重要的一个步骤，我们在《数据仓库生命周期工具箱》一书里描述了组织决策以及实现规范化维定义的整个过程。一个规范化的客户维是一个客户的主表，含有干净的代理客户键和许多维护的很好的描述每一客户的属性信息。规范化的客户维是一个从多个旧系统以及外部数据源汲取融合了许多数据的结果。例如，客户维的地址字段应该包含企业中任何位置的每个客户的可邮寄的和已知的地址信息。这通常是中心数据仓库团队的职责，创建规范化的客户维并作为企业其它部分的的数据源，既提供给数据仓库也提供给旧系统使用。

规范化的产品维是企业达成一致的主要产品列表，包括所有的产品的分类和产品属性。一个好的产品维，就像一个好的客户维，应该拥有至少 50 个单独的文本属性。

规范化的日期维几乎总是一张粒度到日期的表，一般有十年或更多。每一个日期都对应许多有用的属性，这些属性来自与企业合作的许多国家和地区的法定日历，另外还有只为内部管理所用的特定的财务日历周期和市场季节日历。

规范化维对于数据仓库来讲非常重要。如果不严格的坚持规范化维，数据仓库将就不能作为一个整体发挥作用。如果一个像客户或产品这样的维是不规范化的，则多个事实表将不能被简单地在一起使用，或者更糟的是，尝试使之结合起来的将会导致错误的结果。更肯定地说，规范化维使得在同一数据仓库空间里的多个事实表上使用单一的维表成为可能，并且面向统一的用户界面和统一的数据内容，以及统一的属性描述，这样就可以在不同的事实表间进行聚合。

设计规范化维

确定并设计规范化维需要花数周的时间。大多数规范化维很自然的被定义成最小的粒度（原子级）级别。客户维和产品维的粒度自然的将在最小粒度，以便与原系统的条目相对应。日期维的粒度通常为天。

应用

一旦中心数据仓库团队成功的为企业定义并提供了主要的规范化维，则对于那些事实表的拥有者来使用这些维就显得非常重要。对使用规范化维达成一致意见并不是一个技术决定，而是一个业务策略，并且企业数据仓库发挥作用的关键。规范化维的使用应该获得最高级别的领导的支持，这对企业的 CIO 来说可是件头痛的事。

规范化维的可变更性

对于那些已知的特定事实表的数据范围仅仅只是子集的性况，有可能为这些事实表创建规范化维表的子集。举个例子：主产品维表可以被限制为只有部分产品，这些产品只在某个特殊地区生产，并且数据集市也只适用于这一地区。由于缩小了的维表保留原始维表的所有属性并且和初始粒度一样，我们把它称为简单的数据子集。

聚合的数据子集系统地从初始的维表删除行和列。例如，把日期维表从天聚合到月非常普遍。在这个例子中，我们可能仅需要保存每个月的第一天的记录，同时我们也必须删除所有那些仅仅在日粒度出现的像“星期几”和“节假日标记”这样的属性。如图 4.10 所示。

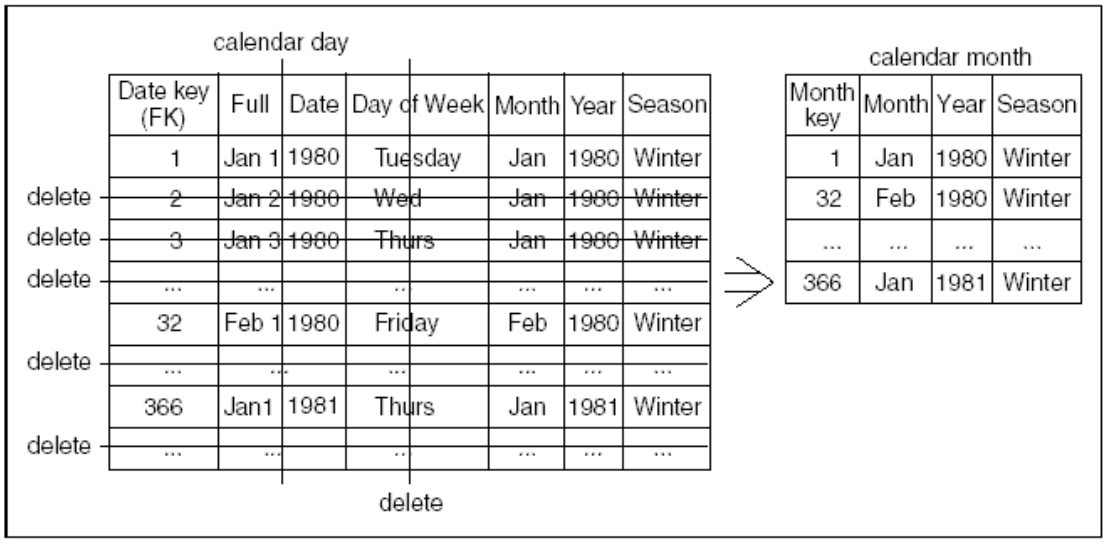

图 4.10　创建规范化的日历月份表

也许你会对如何在一个被子集化了的规范化维环境中创建查询感到困惑？哪一张维表可以用？以及用在哪里？事实上，它比听起来要简单的多。每一个维表和其关联的事实表相对应，任何在事实表间钻取的应用都不可避免的要用多路 SQL 来分别并按顺序查询每一个数据集市。通常情况是每一个钻取报表的每个列都生成一个独立的 SQL 查询。如果不使用规范化维，则只有当报表中使用的维属性在每个维表中找到时才能完成报表执行。由于维被

规范化了，业务问题答案才得以保证一致。如果我们创建了规范化的事实定义，这个数量就可以比较了。

规范化事实

我们已经花了很长的篇幅来讲述关于如何创建规范化维来把数据集市绑定在一起这一主要任务。但这也只完成了上层构架的 80%，剩下的 20%是就建立标准的事实定义。

幸运地是，在确定规范化维的同时也确定了标准的事实定义。当我们用相同的术语贯穿所有事实表以及创建单个报表在前面部分描述过的事实表间钻取的时候，我们都需要标准的事实定义。

建立规范化维是一个协作的过程，因为每一张事实表的管理员都必须就使用规范化维达成一致。在规范化会议中，管理员也需要确定在每一张事实表里呈现的类似的事实。例如，许多事实表可以报告收入信息。如果最终用户应用期望添加或比较这些来自多个事实表收入度量，则定义这些收入度量的业务规则必须是相同的。也许有的表是在月末记录收入度量，但是另外的表则可能从帐单期的聚合得到收入信息。或许有的表是销售的总量，但是另外的表则仅仅是可接受的通用会计原则（GAAP）范围内一部分销售量。

规范化的事实可以直接进行比较，甚至可以直接进行数学计算，如求和、计算比率等。如果事实表的管理员达成一致意见，则对于一部分或者所有的事实表的数据准备步骤就可能是对事实的转换，以满足通用定义的需要。

事实表提供者

尽管这一部分超过了操作讨论的范围，但是我们仍要完成本章描述的规范化视图。在下一部分，我们将定义维度管理员这一角色，他是准备并发布规范化维到整个企业的中心权威人士。事实表提供者是维度管理员的接受客户端。事实表提供者拥有一张或者更多的事实表，并且负责最终用户如何访问他们。如果事实表参与了整个企业范围内的钻取操作，则他们必须使用由维度管理员提供的规范化维，并且必须仔细准备数字化的事实，这些事实被组织确认为规范化（标准）事实。

维表管理员：发布规范化维来影响事实表

规范化维必须是集中管理的对象。组织必须要认命一个主要的维度管理员来管理和发布任何规范化维。

当维表管理员者发布了维度的新版本时，事实表提供者有义务立刻更新本地的维表拷贝。在理想情况下，发布的维度在每一个纪录都包含一个版本号，当钻取应用在准备报表的最后一步把分隔的答案绑定在一起的时候，则必须要使版本号相等。如果事实表提供者滞后更新维，则由于版本号不匹配，钻取应用将失败。尽管这听起来有些苛刻，但是强制执行这一规定是非常重要的；同一维度的不同版本将导致隐藏的，不可预知的错误。

每一个规范化维都必须在每一条记录上都有一个类型 1 的版本号字段（如果不了解，可以详见第五章有关于类型 1、2 和 3 缓慢变化维的讨论）。一旦维表管理员针对各个事实表发布了维，则在每一条记录中的版本号就会被覆盖。对于任何从两个及以上的事实表使用两个及以上的维拷贝进行关联数据的钻取查询都必须要确保维度版本号的严格匹配。这就需要维

表管理员同步复制任何修订过的维度到所有事实表。在支持事实表间钻取查询的环境中，强制维度版本的失败是非常严重的错误，因为虽然请求可能会非常好的完成，但是汇总和分组则可能完全错误，并且也没有在最终报表上检测不一致的有效方法。

对于单一机器、单一 DBMS 上的单一表空间来说，由于只需要有一份维表的拷贝，规范化维管理就显得非常简单。这一拷贝在查询时关联到表空间中已存在的所有事实表。然而，这一好处只体现在最小最简单的数据仓库中。一旦事实表出现在是多个表空间，多个 DBMS，或者多个远程设备上时，则维表管理员必须要负责上一段描述过的所有职责，以便完成跨越多个数据集的钻取操作。

必须再次强调，事实表提供者和维度管理员的角色不但对在物理上分布的数据仓库环境中有效，也适用于高度集中的由一组 DBA 管理的单机数据仓库。一旦事实表分散在各自的表空间中，则所有这些问题都有相关性，因为他们必须拥有多个物理的维表拷贝。

规范化维的详细提交步骤

创建规范化维不仅仅只是在某一标准描述的维度属性上达成一致。在下面的步骤中，涉及类型 1、2 和 3 的缓慢变化维的内容将在第五章中解释。维度管理员必须要做：

1、增加新的记录到规范化维，产生新的代理键。

2、为类型 2 的变化增加新纪录到现有维度条目中（在某个时间点及时的真实的物理变化），并生成新的代理键。

3、为类型 1（覆盖）和类型 3（交替变化）的变化修改记录，但不改变代理键。一旦完成了任何类型 1 或 3 的变化修改，则更新维度的版本号。

4.为所有的事实表提供者同步复制修订后的维度。

接收的事实表提供者需要完成更复杂的工作。他必须要：

1、接收或者下载更新的维。

2.处理那些标记为新并且当前在用的维度记录，修改映射到代理键的当前键。

3、处理那些标记为新并且已经过时的维度记录，这将触发一个复杂的过程来进行通常的代理键处理（在第 5、6 章中描述）。

4、用正确的代理键替换自然键后，把所有的新记录添加到事实表中。

5、为纠正的错误、聚合的快照和过期的维更新来修改所有的事实表，最好是使用按分区的方式来完成。详见第 9 步的注释。

6. 删除无效的聚合。只有当类型 1 或者类型 3 的变化发生在聚合的目标属性上，或者历史事实记录在第五步中被修改的时候，当前的历史聚合才会变得无效。其他属性的改变不会使聚合无效。例如，某产品的口味属性的改变就不会使基于类别属性的聚合变得无效。

7.重新计算受影响的聚合。如果维度的新版本没有更新版本号，则只有当加载了新的事实数据时才需要重新处理。如果维度的版本号改变了，则在第 6 步中的删除聚合表后，整个的历史聚合将重新计算。OLAP 系统会自动处理这些步骤。

8.确保所有的基础表和聚合表的数据质量，确保正确计算聚合表。

9.把更新的事实表和维表上线。有可能使事实表（或者更确切的说事实表的一部分）短期下线的的详细策略在《生命周期工具箱》一书中有描述。

10.通知最终用户数据仓库已被更新。告诉用户是否发生了主要的变化，包括更新维度版本号，添加过时的记录以及修改历史聚合。

实现规范化模块

为了实现规范化维和事实，规范化子系统需要参考元数据，这些元数据捕获了来自源系统的有效值集和规范化的维度属性值以及规范化的事实值之间的关系。

许多 ETL 工具支持这些类型的映射，可能用预先建立的元数据属性，或者让 ETL 小组针对源表对象使用扩展元数据属性。图 4.11 显示了一个支持数据规范化的元数据表的例子。表和列的实体分别捕获了每个表和它相关列的元数据。事实表记录了来自每一源系统的正式的规范化值定义。整个源系统的标识符在源系统表里得到。列维度包含了被映射到正式的规范化值的源值。因此,正如在前面简单的例子中引用到的，如果 Male 和 Female 是性别的目标规范化值，则事实表将会在源系统 A 中用 M 关联 Male，用 F 关联 Female；而在源系统 B 中，用 M 关联 Male，但是用 W 关联 Female；在源系统 C 中，用 Man 关联 Male，用 Woman 关联 Female。

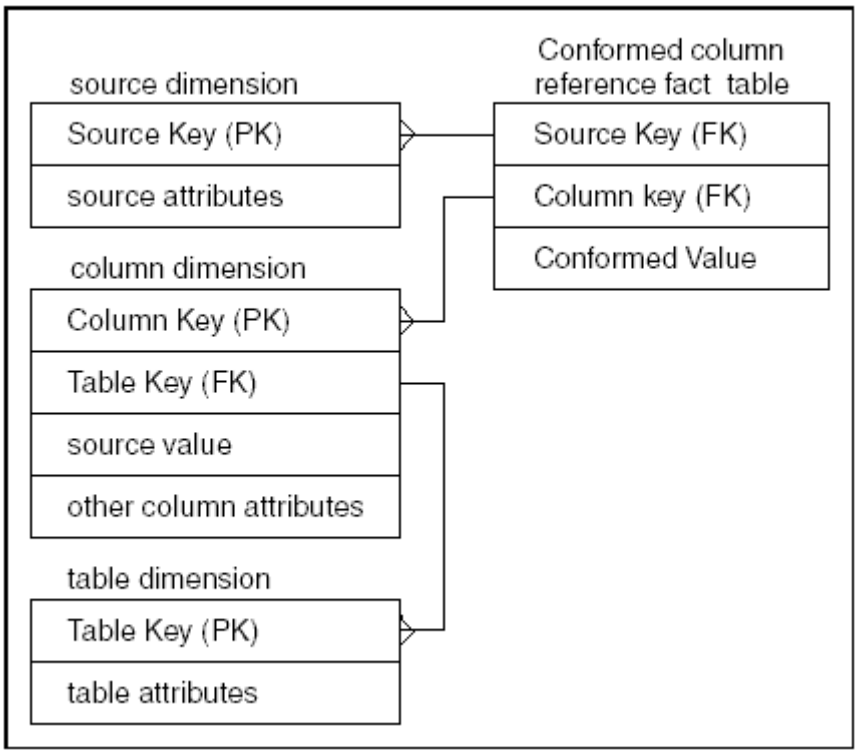

图 4.11　规范化的列支持结构

对于记录中含有无效值的列，即值不在列的维表的有效值范围内，应该用一个来自标准化值参考表的预定义的数值（如 Unknown）来替换；这一替换操作应该记录到错误事件事实表中。

很重要的是，对于不能被标准化的假的或者非法的数据应该让其在下游 ETL 流程（如匹配）中消失，并且对最终用户也不可见。

更复杂形式的标准化通常被用在处理名字和地址的数据清洗方面。支持这一过程的专用软件工具的功能对于 ETL 团队来说想要进行复制是非常困难的。一定要参看一下第 7 章列出的这个竞技场中的主要运动员。在某些情况下，标准化的值使用某种技术按概率获得，这种技术使用统计方法根据常用的已知名称或者地址来排列那些不准确的数据。要知道一些概

率标准化工具也有自调整的集成引擎，它们可以随时间累计学习更多的数据分布情况，特别是针对特殊的应用，然后合理地调整它们的处理算法。这是一个强大的功能，同时也向 ETL 构架师的能力提出了挑战，他们要测试出哪一种数据集成引擎会变得越来越聪明。大多数的标准化工具对其再造数据过程及处理数据过程中遇到的异常和错误进行反馈。捕获,保存和挖掘数据集成的线性过程非常重要，并用之后留下的日志表创建错误事件事实表。

匹配驱动去重复

匹配，或叫做去重复，包括标准化记录重复值的删除。在某些情况下，重复纪录可以通过在一些关键列中特定值的出现来很容易地探测到，比如社会保险号码、电话号码或交易卡号码。不幸地是这样轻松的情况是很少见的。在其他很多时候，并不能发现这样明确的匹配，删除重复值的可用的唯一线索是几乎能匹配的多个列的相似性。还有更困难的情况,确定找到了多个定义匹配的列，但是它们又互相矛盾。

专用数据集成匹配工具现在已经非常成熟了，并且在广泛的使用，就是用来处理这些非常特殊的数据清洗问题的。这些工具通常和数据标准化工具有关系密切，并且经常打包一起出售。

匹配软件必须把数据流中的记录集与规范化维记录相比较并且返回如下内容：

- 匹配相似性的量化得分值
- 在标准化记录范围内把输入记录与规范化维实例链接起来匹配键的集

这样，一个通过匹配处理过程运行的输入记录可以是匹配空或者是一个规范化维的记录，或者匹配 0 个，1 个或者是更多的在批处理队列中的其他输入记录。在任何情况下，匹配软件的作业将把这些详细描述了导出匹配关系的输入记录和匹配键进行联合。这些匹配键被在下一部分描述的 survivorship 模型使用，它指出了哪一个记录匹配到了另一个记录，然后把它作为整合到单一集成记录的候选。

许多数据匹配工具也包括匹配得分，或叫匹配置信度量，那描述了获得匹配的相似度。通常，这些匹配得分通过创建多个匹配方法或多路来获得，每一路的匹配相似度得分的结果应用到推荐的一组匹配键和一个全局权重得分。

需要有稳定的去重复能力的组织可以选择维护一个之前匹配数据的永久性库，其中每个数据项仍然和单一的数据提供者相关联,使用这个统一的的库来改进匹配的结果。这样，匹配引擎不但可以在规范化维记录上应用匹配过程，而且还可以在之前匹配的来自所有源系统的维记录的完整集上应用。这一方式可以得到更好的匹配结果，因为匹配内容更丰富了，而且能更好的面对处理匹配规则的变化，因为现在不需要通过整个数据集成过程运行所有源系统的数据就可以满足了。但是它也使匹配过程变得复杂，因为匹配可能发生在数据源内部和外部，甚至是完全规范化的数据集。

正如本文所写的，匹配工具并非是那种能插入到 ETL 过程中而且自动知道如何去做的简单的拧钥匙实现方式。相反，它们需要基于组织的数据做大量的评估和训练工作，决定合并哪些属性 (匹配过程或者角度) 的匹配策略的建立很可能是重复的先兆，把不同的角度集成到一个匹配策略的调整，以及基于组织的承受能力设置积极的或保守的匹配和不匹配阈值。ETL 工具包厂商正在试图完成这一应用目标，然而，应该检查一下这些产品的内嵌的匹配转换器。

共存：规范化的最后一步

共存性是指整合一组匹配（去重复）记录到统一的映像，它把每一条匹配的记录组合成最高质量的列值来建立规范维记录。这就建立了业务规则，它为来自所有可能的数据源的列值定义了一个架构，并捕获了生成共存记录（规范化记录）时应用的源数据到目标数据的映射。

此外，共存性必须能够针对列的组合，而不仅是单个的列。这是为了某些情况的需要，这时单个可共存列的组合可能导致毫无意义的混乱结果，例如，把来自三个不同的源系统的 1，2，3 地址行组合在一起生成一个组合地址，则这个地址记录的可信度还没有这三条任何一个高。对于这种情况，最好是创建规则来要求某些列的联合（共存块）能够在一起共存：所有的或者没有。图 4.12 中所示的元数据表支持最常用的共存需求。

- 共存源到目标映射表捕获了在源数据列（已清洗但没有规范化的输入数据）与目标数据列（规范化维表的列）之间的数据集成映射关系。为了兼顾灵活性与简便性，允许任何列的组合可用做到任何目标数据列的组合的源，这也使得 ETL 架构为了合理地生成它带来了的负担（不仅要考虑参数完整性，还包括更加复杂的结构）。
- 共存块表把映射的源到目标的分组成块，这些块必须能够在一起共存（合理解决前面提到 1，2，3 类地址的问题）。共存块允许只有一个源和一个目标，因此通过强制所有共存性按块执行，可以简化元数据模型和共存的处理过程。该表包含一个排序，它使得可以使用动态 SQL 来确定源系统字段块的优先级，该 SQL 按照共存块源排序优先级按顺序在每一块中查找非空值，并根据匹配键是（UPDATE）否（INSERT）已经存在规范化记录的代理键来设置合适的 INSERT 或 UPDATE 语句。

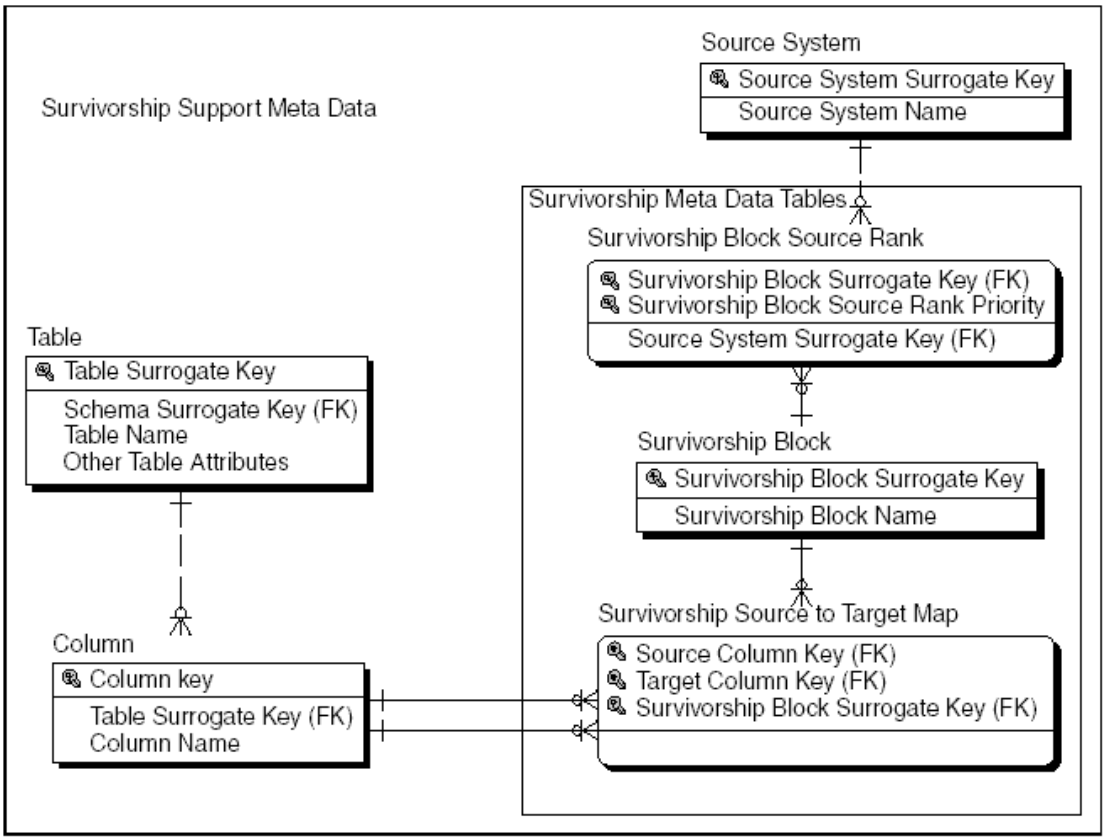

图 4.12　生存支持元数据

对于去重复过程成功的把分离的源实体联合成为一个单一的实体（如客户实体）的情况，如果这些源实体在各自的源系统中已经指定了主键，则需要有一个表对这些过时的主键进行维护，以提高对源系统数据进行去重复处理的速度。

提交

提交是 ETL 最后的重要步骤。在此步骤中，已经清洗并规范化过的数据被实际写入到维结构中供最终用户和应用系统访问。在只由一个表空间组成供最终用户访问的最小经数据仓库中，维表只是简单地写入这个表空间。但对于大型的数据仓库，从含有多个表空间到更广泛的分布式数据集市自治网络，维表必须以一致的方式仔细地发布。提交非常重要，我们将在第 5 章和第 6 章中详述。

小结

回顾以上细节，本章主要覆盖了四个大的主题：目标、技巧、元数据和度量。

数据清洗和规范化的目标在于减少数据中的错误，提高数据的质量和可用性，并标准化整个企业共享的关键描述属性和量化度量。

数据质量技巧涵盖了从数据库级别校验单个字段的定义检查（列属性约束），到校查字段到字段之间的一致性（结构约束），再到数据的特定业务规则的检查（数据和值规则约束）。数据质量处理的最后阶段（规范化和去重复）也就实现了，在这一阶段解决跨越多个数据源的数据差异问题。

数据质量元数据包括所有技术的定义和业务规则。我们描述了构建一组数据过滤器的方法，每个过滤器都表示一个数据质量 investigation。一些过滤器作为每一个 ETL 过程的一部分定期执行，还有一些是偶尔运行来进行检查或特定的 investigation。常规过滤器提供了诊断指标和度量，并存储在详细的错误事件事实表和与事实表关联的审计维表中。这些审计维非常有意义，因为在某种意义上它们把元数据提升为真实数据。数据质量指标可以参与到最终用户的查询中，就像它们是普通数据一样。

最后，我们建议的数据质量度量是 ETL 团队需要的度量的起始集，它是为了创建一个综合的数据质量处理流程。

当数据经过了数据质量处理流程后，就已经为最后的提交步骤做好了准备，这将在第 5、6 章中详细描述。