第一部分　需求、现状和架构

一切围绕需求

理想情况下，你必须从最困难的挑战之一开始设计你的 ETL 系统，那就是：一切围绕需求。这意味着要集中收集所有已知的需求、现状和影响 ETL 系统的约束关系，我们把这些简称为需求。

需求是我们必须要面对的，并且系统也必须适应这些需求。尽管在需求框架内，你还有很多地方可以作出你自己的决定，验证你的判断，发挥你的创造力，但是，需求正如其字面含义，是必须要满足的。该章的第一节将试图帮你勾画出全部的需求类型，并且让你明白在开发 ETL 系统过程中需求是何等的重要。

紧接着需求后面，我们确定一组在 ETL 项目开始阶段需要制定的架构方面的选择。这些选择是重要的前提，因为它们将在后面的实现过程中决定你所做的一切。该架构会影响软件、硬件、编码习惯、人员和操作。

最后一节描述了数据仓库的任务。我们还仔细定义了数据仓库的主要架构组成，包括后台、集结区、操作型数据存储和展现区。我们还对数据集市和企业数据仓库作了详细而明确的定义。请仔细阅读本章，这里描述的定义和边界确定了整本书的逻辑。如果理解了我们的思想，你会发现为什么我们的方法会比其它任何数据仓库设计方法论更合乎逻辑。最后，作为本章的结尾，我们简单陈述了 ETL 小组的工作任务。

流程检查

规划与设计：*需求/现状* -> 架构 -> 实现 -> 测试/发布
数据流：还没有到数据流步骤。

需求

在本书的介绍部分，罗列了我们认为非常重要的需求类型。虽然需求的每个部分都可以精彩纷繁，但业务方面的需求始终是更基础的和更重要的部分。

业务需求

业务需求是数据仓库最终用户的信息需求。我们使用业务需求这个术语来狭义地表示最终用户作出决策所需的信息内容。其他方面的需求会扩展业务需求的定义，当然这也意味着 ETL 小组必须引入更多的信息源。

在这里需要注意的是，业务需求直接决定了数据源的选择。ETL 小组的核心工作就是不断地理解和检验业务需求。

在《数据仓库生命周期工具箱》一书中，我们描述了调查最终用户和收集业务需求的过程。许多情况下，最初对数据源的调查不一定完全反映数据的复杂性和局限性。ETL 小组常常要做出重要的判断——原始数据源是否能解决用户的业务需求。当然，ETL 小组也常常会发现数据源的额外的能力，从而扩展最终用户的决策支持能力。从更广泛的意义上讲，

业务需求和数据源的内容都是不断变化的，需要不断地进行检验和讨论。

合规需求

近年来，尤其是 2002 年的 Sarbanes-Oxley 法案通过以后，企业上报的数据必须更加准确与完整，绝不允许篡改。当然，像电信这样的行业使用数据仓库来满足日常报表需求已经有很多年。但无论如何，对财务报表的要求已经变得越来越严格。

也许有些财务报表的问题跟数据仓库无关，但其他大部分的问题都能够在数据仓库范围内解决。对数据仓库典型的需求包括：

- 数据源的归档备份以及随后的数据存储
- 任何造成数据修改的交易记录的完整性证明
- 对分配和调整的规则进行完备的文档记录
- 数据备份的安全性证明，不论是在线还是离线进行

数据评估

正如 Jack Olson 在他的《数据质量：精度维度》一书中指出的那样：数据评估是设计任何数据使用系统的前提。他指出：“数据评估使用分析方法来检查数据，充分了解数据的内容、结构和质量。好的数据评估能够处理海量数据，使用分析方法找到需要解决的全部问题。”这方面对于 ETL 小组尤其重要，因为可能有些数据源的内容没有被很好的理解。例如，可能有某一数据源很好地满足了生产部门的需求，比如订单系统，但对数据仓库来说却可能是个灾难，因为数据仓库需要使用的字段并不以订单系统为中心，订单系统的信息对于数据仓库的分析来讲远远不够。

数据评估是一个系统的检测过程，对 ETL 需要使用的数据源的质量、范围和上下文进行检查。从极端意义上讲，一个清洁的数据源是一个维护良好的数据源，只需进行最少量的数据转换和人工干预就可以直接加载到最终的事实表和维表。但对于脏数据源可能需要如下的步骤进行处理：

- 完全清除某些输入字段
- 补入一些遗失的数据，产生特殊的代理键
- 自动替换掉某些错误数据值
- 在记录级别上进行人工干预
- 对数据进行完全规范化的表述

从更深的意义讲，如果数据评估指出某些数据源缺陷很大，不能支撑业务目标，其结果可能关系到数据仓库的成败。数据评估步骤为 ETL 小组不仅提供了关于数据清洁程度的指导，还能防止 ETL 小组漏掉一些重要的里程碑。一定要预先进行数据评估！使用数据评估的结果去说服业务主管，安排现实的开发进度，认清数据源的限制，追加投资用于在源系统中进行数据捕获。在第 4 章我们将详细研究数据评估和数据质量算法。

安全需求

近年来所有 IT 领域对安全关注的程度都有极大的提高，但对于大多数数据仓库小组而言，安全问题依然是一个事后的、额外的负担，对安全性的考虑充满争议。数据仓库的目的

是为决策者提供广泛的数据，而安全性则要求数据仓库只向用户提供应该访问的数据。

本书中我们推荐一种基于角色的安全定义方法，这种方法在应用一端控制能够访问的数据仓库数据。这意味着最终用户的安全授权不是在数据仓库的物理表一级，而是在被称为目录服务器的 LDAP 服务器中定义，各应用系统的用户安全定义都在各自的应用系统中进行授权。本书的后续部分将会安全问题进行详细的讨论。

这样的安全处理方式对 ETL 小组最直接的好处在于，不必过多考虑安全性的设计和管理，但是 ETL 小组仍需要考虑对物理表的读/写权限。ETL 小组的开发环境应当是在位于包过滤网关后面的子网内，如果 ETL 环境位于整个企业内网，那么内网内的任何恶意代码都可能在不知不觉中被安装，从而截获对数据库的管理员口令。

另外，安全的范畴必须扩展到数据的物理备份。如果磁带或硬盘可从物理备份中轻易移走，那么安全性就存在一定的漏洞。

数据集成

数据集成对 IT 人员是一个比较大的题目，因为 IT 人员的最终目标是将全部系统无缝的集成到一起。“360 度的业务视图”就是对数据集成的业务解释。很多情况下，严格意义上的数据集成应当在数据进入数据仓库之前完成，但其实只有极少的时候才如此，除非企业只有一个单一的数据源，比如 ERP 系统，即使这样，通常情况下某些重要的交易系统也是位于 ERP 系统之外的。

本书中，数据集成采用规格化的维表和事实表。所谓规格化的维表是指在分离的数据库中建立公共维度实体（通常是文本标记和度量的标准单位），这样使用这些实体就能够构建横向钻取型的报表。在第 5 章和第六章中将会详细介绍这个过程。

规格化的事实表意味着在分离的数据库中建立公共业务度量，比如关键绩效指标（KPI）等，以便于通过差值和比率对这些数值进行比较。

ETL 系统中，数据集成是数据流程中一个独立的步骤，叫做规格化步骤。物理上讲，此步骤需要为一致性维度实体和事实设置通用名称。

数据延迟

数据延迟需求用于描述数据提交到最终用户的速度。数据延迟对系统的实施和架构有巨大的影响。从这一点上讲，本书中介绍的大多数传统的面向批处理数据流方式的系统都能通过更优化的处理算法、并行处理或者更有效的硬件来提高性能。但从另一个角度讲，如果数据延迟需求太紧迫，ETL 系统的架构就必须从面向批处理架构转化成面向流的架构。这种转换不是一个渐变的过程，而是在数据提交的每个主要步骤都要做大的改动，需要重新实现。我们将在第 11 章中对这种面向流的实时系统进行介绍。

数据归档和数据线性化

我们在前面的章节中提到了这一点。但即使没有法律法规对数据存储的要求，每个数据仓库也需要保留不同的数据备份，原因也许是为了与新数据进行比较以生成新的记录快照，也许是为了重新处理。

本书中，我们推荐在每个主要的转换发生时集结数据。这些集结点发生在全部的 4 个步

骤：抽取，清洗，转换与提交。那么，什么时候要将集结（将数据写入到磁盘）进行归档（将数据无限期地存储到永久的介质上）呢？

保守地讲，全部的集结数据都应当归档，除非已做出明确的决定，决不会恢复某个数据集。与之后要在 ETL 系统中重新处理数据相比，从永久性介质上读取数据几乎不算什么麻烦。当然，在经过一定时间后也不太可能仍按照旧的算法处理数据。

同时，每个集结/归档数据集都应当有与之相对的元数据，描述原始数据源和处理步骤。另外，对于合规需求而言，追踪记录所有的数据沿袭是严格必要的，应当成为每次归档流程的一部分。

最终用户提交界面

ETL 系统的最终步骤是将应用提交给最终用户。提交过程占据十分重要的位置。ETL 小组必须与建模小组紧密配合，对构成最终用户应用的数据结构和内容进行严格把关，务必使其简单快捷。将使用复杂、查寻缓慢的应用直接交给最终用户是不负责任的。最常犯的错误就是将完全规范化的数据模型直接交给最终用户就置之不理了。这也是为何在第 5 章和第 6 章会详细阐述建立最终提交给用户的维度模型的原因。

通常来讲，ETL 小组，数据建模人员以及最终用户应用开发人员需要紧密配合，开发针对详细需求的数据模型。而且，即使物理数据格式完全正确，不同的终端用户工具也有各自需要注意避免的用法，以及需要充分利用的优势。对 OLAP 立方体的数据准备也需要考虑相同的事情，我们将在第 6 章描述。

可用的技能

建立 ETL 系统时，重要的设计方面的决定必须由那些创建和管理系统的人做出。不要建立一个基于 C++处理模块的系统，即使这些编程技巧是公开的，也不必掌握这些技巧。如果你具有某些 ETL 工具的使用经验并且知道如何管理项目，那么在建立 ETL 系统时可能会有信心的多。

下一节中，我们将讨论到底是通过编码方式开发 ETL 系统还是直接使用 ETL 工具。在这里，我们的观点是抛开技术问题和许可证费用不提，也决不要使员工和管理人员使用此系统极不顺手。

已有的证可证

最后，很多情况下，主要的设计决策由上级管理部门做出，坚持使用已有的许可证。大部分情况下，这个要求是能够接受的，并且也是符合实际需要的。但有些情况下使用原有的许可证却是个错误。这是个很困难的处境，就算你对此很有信心，也可能是在需要拿工作做赌注了。如果执行上级管理部门的决定，继续使用已有的许可证，那就必须有充分的准备和坚定的信心去接受最后的结果，否则就可能需要另谋高就了。

架构

选择何种架构是 ETL 系统设计之初就必须作出的最基本的决定。架构的选择会影响一

切，架构的任何修改几乎都意味着整个系统要从头开始实施。架构选择一旦作出，使之最有效发挥作用的关键是坚持使用这种架构。你应该有目的地阅读下列子节中的每部分，其目的是首先确定特定的架构选择，然后应用在 ETL 系统中的任何地方。另外，虽然该节中每一部分都很重要，但早期架构选择中最重要的是确定使用供应商的 ETL 工具还是自己手工编码方式来实现系统。几乎所有设计 ETL 系统的细节都基于这次选择。

流程检查

规划与设计：需求/现状 -> *架构* -> 实现 -> 测试/发布
数据流：还没有到数据流步骤。

ETL 工具还是手工编码（购买工具集还是自己写程序？）

在《智能企业》杂志的一篇精彩的文章（2003.5.31，Ralph Kimball 编辑）中，Gary Nissen 对这个问题总结概括后，回答是：“看情况”。我们展开来分析一下 Gary 的观点。

基于 ETL 工具的优势

- 根据某个 ETL 工具厂商的说法：“一个有价值的工具目的不在于能够解决琐碎的问题，而是要将不可能的事情变得可能。”
- 开发过程简单、快速、低成本。如果项目足够大或足够复杂，工具的成本会显得微不足道。
- 了解业务知识但不精通编程的技术人员也能够有效使用 ETL 工具。
- 很多 ETL 工具内部集成了元数据资料库，这样，源系统、目标数据库以及其他 BI 工具的元数据既可以同步到这个元数据资料库中。
- 大多数 ETL 工具在每个处理流程会自动产生元数据，从而强制所有的开发人员必须遵循一致的元数据驱动的方法论。
- 大多数 ETL 工具都有内置的调度器，帮助生成文档、简化创建过程以及变更管理。如果系统发生故障，ETL 工具会负责处理全部复杂的依赖关系以及差错控制。
- ETL 工具的元数据资料库能够自动的产生数据沿袭（便于反查）和数据依赖分析（便于前查）。
- ETL 工具为大多数源和目标系统设有预置的连接器。在技术层面，ETL 工具应当处理能够处理全部种类的数据类型转换。
- ETL 工具通常提供内嵌的加密和压缩功能。.
- 大多数 ETL 工具处理大数据集时具有较高性能。如果数据量非常巨大，或者在未来几年可能会非常巨大，那么应该考虑采用 ETL 工具。
- ETL 工具可以跨服务器的复杂的负载均衡，避免服务器死锁。
- 当底层框架变更时，会对后续流程和应用造成影响，大多数 ETL 工具可以自动的进行这种“变更-冲击”分析。
- 在指定的处理模块中，ETL 工具可以使用编程语言进行扩展。比如，如果其自带的模块不能达到要求的功能和性能，就可以在 ETL 工具的数据流程中添加一个自定义的 CRC（循环冗余校验）算法；或者，将一个自定义的周期算法作为数据质量检查步骤的一部分加入进去，以判断某个观测值是否合理，等等。

手工编码方式的优势

- 在手工编码方式下可使用自动单元检测工具。比如，JUnit 库(www.junit.org)是一个

著名的、良好支持 Java 编写单元测试程序的工具，对其他语言也有一些类似的包。你可以使用某一个脚本语言，比如 TCL 或者 Python 来设置测试数据，在 ETL 过程中运行，然后验证结果。采用这些自动化测试流程能够极大的提高 QA 人员的工作效率和最终提交版本的质量。

- 面向对象的编程技术可以帮助你保持差错报告、有效性和元数据更新的转换一致性。
- 如果建立了自己的元数据访问接口，手工编码方式可以更直接的管理元数据。
- 对 ETL 系统的简要需求分析可以指导你快速使用基于文件的流程，而非数据库的存储过程。基于文件的处理流程要更加直接，编码更简单，测试更容易，也更便于理解。
- 可以不必考虑用户许可证问题。
- 可充分利用内部的编程人员。
- 基于某个工具的开发使你只局限于某个工具厂商的工具及其特殊的脚本语言。手工编码方式的系统可使用通用的编程语言。(公平的讲，所有 ETL 工具在各自的模块中都与标准的编程语言隔离开。)
- 手工编码方式可以提供无限的灵活性，几乎可以完成任何想做的事情，当然前提是你确实需要这样。很多情况下，一种独特的方法或一种特定的语言可以提供非常的优势。

这里我们还可以为采用 ETL 工具提出新的理由：ETL 工具自动完成文档的能力更强，可维护性更高，尤其是 IT 人员流失较严重的情况下。而反方的观点是，如果 ETL 开发小组有很强的软件开发能力，并且管理良好，文档和维护性不应当是一个大问题。

使用已证实的技术

建立数据仓库时许多初始成本是无法避免的。比如必须购买一些专用的服务器：至少一台数据库服务器，一台 BI 服务器，通常还有一台专用的 ETL 服务器。同时还需要数据库的许可证，BI 工具的用户许可证。在开始一个新的项目时可能还需要购买一些咨询服务以及其他的一些成本。建立数据仓库的这些成本是必须要花费的。但是，有一种成本通常被认为是可有可无，并经常作为成本消减的目标——这就是购买一个专门的 ETL 工具。不用 ETL 工具实现数据仓库是可能的，本书中也没有假定用户必须使用或不使用 ETL 工具。但是，从长远上看，购买 ETL 工具对于建立和维护数据仓库来讲是降低成本的。使用已经证明的 ETL 技术有一些特别的优势：

- 一次定义，多次使用。通过共享并重用业务规则和结构化程序，保持数据仓库中数据的一致性。
- 冲击分析。判断哪些表、列、过程会由于某些修改而受到影响。
- 元数据资料库。轻松地创建、维护和发布数据沿袭，继承数据建模工具中的业务定义，在 BI 工具中展现元数据。
- 增量聚合。动态更新聚合表，只将最新的和修改过的数据加入到聚合表中，而无需在每次数据加载时重建聚合表。
- 管理批处理加载。减少 Shell 脚本，执行条件加载，加载统计，自动 Email 通知等等。
- 轻松连接到多种复杂数据源，如 SAP 和主机系统等。
- 并发执行多线程操作。
- 供应商经验，包括针对维度模型的成功经验和数据仓库支持方面的记录跟踪。

与能够用到 ETL 工具的很多高级特性相比，购买一个 ETL 工具更重要的理由是可以帮

助避免走太多的弯路。这些 ETL 工具的设计基于一个目的：准确执行用户想要完成的工作——加载数据仓库。大多数工具已经发展成为能够支持从不同的异构数据源抽取数据、完成复杂的数据转换、加载维度数据仓库的稳定、强大的 ETL 引擎，

 **在 ETL 配置中不要加入新的、未经测试的产品。像月度仪表盘这样的功能尽管很吸引最终用户，但若放在后台就太不计后果了。保守一些，等待 ETL 产品的成熟。与那些具有良好记录的厂商配合，确保在将来的 5 年中会不断的支持和完善这些产品。**

批处理式数据流还是流式数据流

ETL 系统的标准架构是从数据源中周期性的以批处理方式抽取数据，流经整个系统，最后以批处理的方式对最终用户表进行批量更新。本书也是主要基于此种架构组织。但正如我们在第 11 章中描述的那样，如果数据仓库加载的实时性要求变得很急迫，批处理的方式就会被打破。替代的方法就是流式数据流，记录级数据从源系统不停顿地流向最终用户数据库和屏幕。

批处理方式到流方式的改变会改变所有的一切。尽管我们仍然必须支持基本的抽取、清洗、转换和提交步骤，但这些步骤都必须进行修改，以适应实时性记录处理需求。尤其是对于快速流方式而言，很多关于数据到达甚至参照完整性方面的常规假设都必须进行修改。比如，某个新客户的销售交易记录可能在这个客户的描述信息到达之前就已到达。甚至，即使客户身份得到识别后，增强/清洗/复制过的客户记录也可能在最初事件的数个小时或数天后才到达。所有这些都需要对处理逻辑和数据库进行更新，而这种更新对于面向批处理方式的数据流可能是要尽量避免的。

本节的开始部分，我们建议对整个数据仓库只采用某一种架构模式。很明显，无论是选择批处理方式还是流方式，这种选择都应当是基于不同的应用需求的。在第 11 章中，我们将讨论两种方法的共同点，以及在流方式下应用批处理方法的效果。

水平方式还是垂直方式的任务依赖

水平方式组织任务流是指每个最终的数据库加载相互独立运行。因此，如果有订货和配送两项任务，这两项数据库加载任务会相互独立运行。这通常意味着抽取、清洗、转换和提交的步骤在两个工作流之间是非同步的。

垂直方式任务流会对多个离散的作业进行同步，这样最终的数据库加载会同步进行。尤其是多个系统使用共同的维表时，比如客户或供应商等，之前的步骤一定要同步。这样，如果之前的步骤没有全部执行完，后面的步骤，比如转换或提交等就不会往下运行。

调度自动化

另一个架构方面的选择是，在多大程度上使用自动化调度技术控制整个 ETL 系统。一种极端的方式是，每一个作业都是由输入一个命令行或者点击某个图标发起；另一种极端的方式是，有一个主调度工具管理全部的作业，判断作业是否成功运行，查看各种系统状态是否满足要求，处理人机交互，比如输出警报和作业流状态报告。

异常处理

异常处理不是在文件中放置一些随机的少量的警告和注释，而应当是系统地、统一地将整个 ETL 过程中抛出的全部异常事件放置在数据库中，包括进程的名称，发生异常的时间，诊断的严重性，应当采取的动作，以及异常的最终状态等等。因此，每个作业都需要向数据

库写入异常报告的记录。

质量控制

类似的，用户应当对数据处理过程中产生的质量问题制定一种常规的应对机制。除了触发异常报告机制外，所有的质量问题还应当产生相应的审计记录附加到最终的维表或事实表的数据中。错误的或可疑的数据需要按照一定的方式进行处理，比如在缺失的文本数据中填入问号，或者，在提交到数据仓库之前使用最小偏差估计量的数值来替代错误数据等等。这些主题在第 4 章中会深入讨论。

恢复与重启

从一开始建立 ETL 系统的时候，就应当考虑如何保证系统从非正常结束状态下的恢复和重启能力。比如有一个 ETL 作业是从全部的产品种类中抽取某个品牌产品的销售业绩，这样的任务不允许执行两次。在设计每一个 ETL 作业时都需要这样来考虑问题，因为每个作业迟早都会出现非正常终止或者错误地执行多次的情况。无论如何，必须想办法防止发生这种情况。

元数据

来自于关系型数据库表和数据模型设计工具的元数据比较容易获取，但这些元数据可能只占系统全部元数据的 25%。还有 25%的元数据会在数据清洗过程中产生。对于 ETL 小组而言，最大的元数据难题是在哪里以及以何种方式存储流程信息。ETL 工具的一个重要的优势在于它们能够自动维护流程元数据。如果是使用手写编写 ETL 系统，用户则必须构建流程元数据的中央资料库，具体情况参看第 9 章。

安全性

本章在前面描述了推荐的安全架构，即针对最终用户的基于角色的安全模式。ETL 环境中的安全问题虽然并不比针对最终用户的安全性还要细化，但是，为了满足安全需求，要求针对 ETL 环境中的每个表和备份磁带都要有物理上和管理上的安全设置。最敏感和最重要的数据集需要操作系统打印出针对这些数据集的每一次访问和每个指令的详细报告。打印日志应当锁定在房间里的特定打印机，不对普通的 IT 人员开放。存档的数据集存储时应当带有校验，以表明它们没有被修改。

后台——准备数据

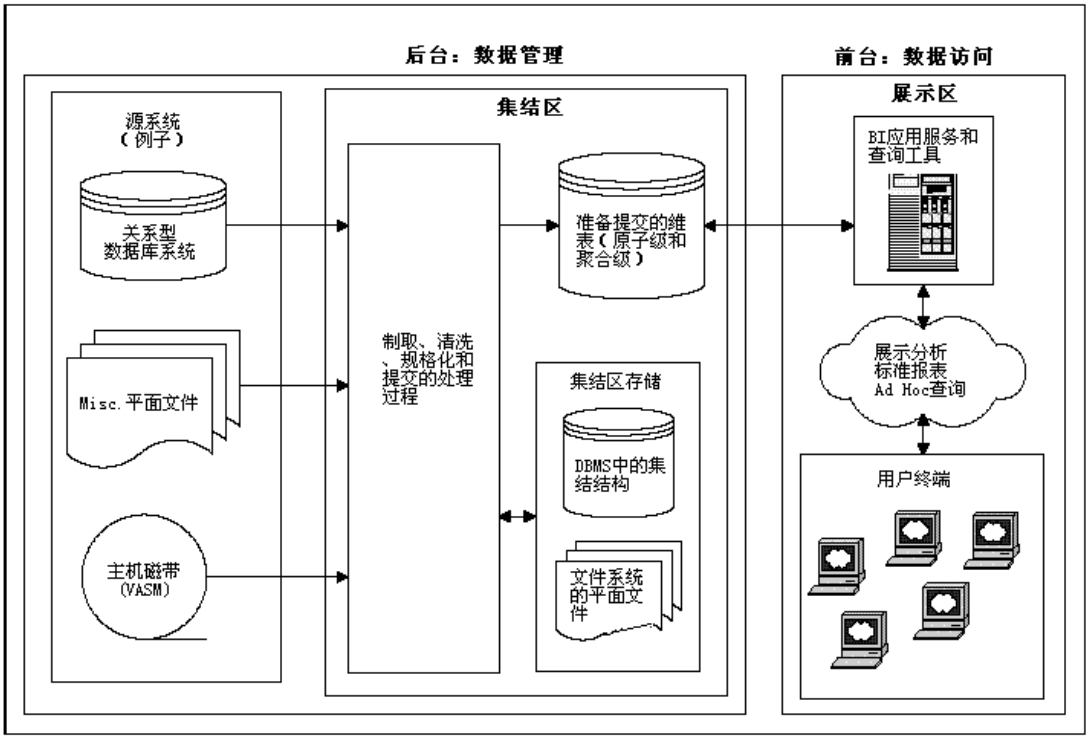
流程检查

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

数据仓库的后台和前台从物理上、逻辑上以及管理上都是分开的。换句话说，大多数情况下，后台和前台都基于不同的机器，依赖于不同的数据结构，由不同的 IT 人员管理。

图 1.1 显示了典型数据仓库中两个独立的组件。

准备数据，通常也叫做数据管理，是指获取数据并将数据转化成信息，最终将这些信息提交到前端的查询界面。后台不提供查询服务，请反复理解这句话。我们的数据仓库方法论假设在后台数据访问是被严格禁止的，这是前台的唯一目的。



图表 1.1　数据仓库的后台和前台

假设有一个餐厅。餐厅的客人是最终用户，提供的食物是数据。提供给餐厅里客人的食物完全符合客人的要求：干净，组织良好，每一片都能轻易辨认与食用。但是，在食物进入餐厅之前，是在厨房中由经验丰富的厨师仔细准备。食物要进行挑选、清洗、切片、烹饪以及摆放。厨房只是工作间，不允许客人访问。在好的餐厅，厨房是完全与客人隔离的——一旦暴露厨房，那里食物还处于半加工状态，是会严重影响客人的胃口的。如果客人需要了解食品准备的信息，厨师必须从厨房出来到餐厅中面对客人，在一个安全、干净、客人觉得比较舒服的环境中解释食物的准备过程。

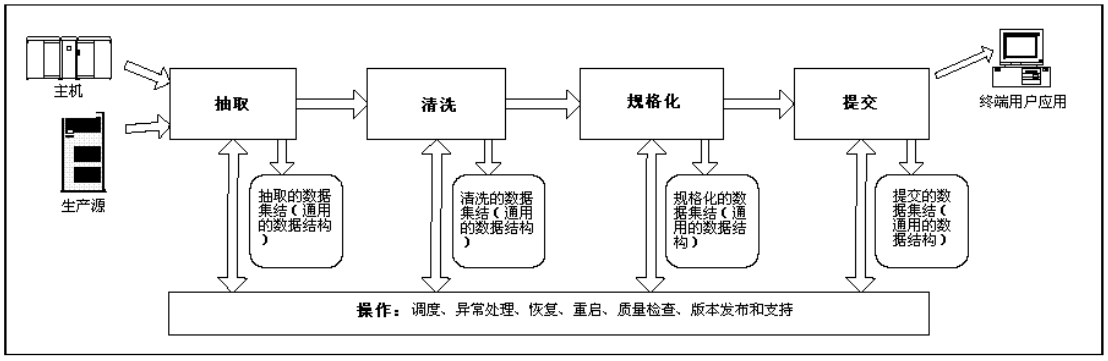
集结区就是数据仓库的厨房。它只对经验丰富的数据集成专家开放。这是一个后台的处理工具，不允许最终用户访问。在这里，从源系统抽取来的数据进行清洗、处理和准备，然后加载到数据仓库的展示层。由 ETL 过程产生的对最终用户有用的元数据都来自于后台，并在数据仓库的展示层进行展示。

后台系统禁止数据访问使得 ETL 小组不用关心下列事情：

- 在行、列和应用一级提供详细的安全性
- 建立提高查询性能的索引和聚合
- 提供不间断的服务运行时间
- 确保全部数据集的一致性

当然我们最终还是需要完成上述的要求，但只不过是在前台，而非后台而已。实际上，是否支持数据访问正是后台和前台的关键区别。如果有例外的情况，允许最终用户直接访问后台系统结构，那么以我们的观点来看，这将严重损害数据仓库。

回到后台系统，我们经常使用集结（Staging）这个词来描述后台的各个独立步骤。集结意味着临时的或永久的物理数据快照。在几乎每个数据仓库中都能发现 4 个集结步骤，如图 1.2 所示。这与本书前面介绍的四个步骤的数据流程一致，只不过使用集结步骤更加明显。在本书中我们假设每个 ETL 系统都是按照这四个步骤组织，每个步骤的数据在传递给下一个步骤的同时也并行写入到磁盘中。



图表 1.2　数据仓库的四个集结步骤

本书中间部分的章节会重点介绍这四个步骤，这四个步骤是：

1. 抽取。源系统的原始数据在进行大的转换之前通常直接写入到磁盘。来自于结构化源系统的数据（比如 IMS 数据库，或者 XML 数据集）在这一步中经常写入到文本文件或者关系型数据库表中。这使得最初的抽取尽可能简单和快速，并且如果发生中断的话，重新抽取会有更大的灵活性。最初获取的数据在以后的步骤中可能会被多次读取，有些时候，在清洗步骤完成后，最初的数据会被丢弃，有些时候则可能归档以长期保存。也有可能最初的数据要保存至少一个抽取周期，以便于计算连续的抽取之间的不同之处。

我们可以在清洗和转换步骤进行重要的内容转换，但是解决遗留数据格式问题的最佳时机还是在抽取阶段。这些格式问题包括计算重复的组、重定义、重载列、执行低级别数据转化，比如，将位编码转化为字符，将 EBCDIC 转化为 ASCII，将小数转化为整数。具体细节我们将在第 3 章讨论。

2. 清洗。大多数情况下，源系统可接受的数据质量程度依据数据仓库要求的质量而不同。数据质量的处理可能包括几个独立的步骤，包括有效值检测（如是否是已有的邮政编码？是否在有效值范围内？）、一致性检测（如邮政编码与城市代码是否一致？）、删除重复记录（如是否有同一个客户出现两次而相关的属性略有不同？）、检测是否有复杂的业务规则和过程需要增强（如白金客户是否有相关的信用状态？）等等。数据的清洗转换可能需要人为的干预和判断。数据清洗步骤的结果往往半永久保存，因为需要的转换往往难度非常大，并且是不可逆的。另外，清洗过的数据是否需要返回到源系统以提高数据质量，从而减少抽取时可能发生的问题呢？这是个很有趣的问题。即使清洗过的数据不能物理返回到源系统，也应当具备数据异常报告机制以提高源系统的质量。这些数据的问题在最终的商务智能应用中也是非常重要的。

3. 规格化。 当多个数据源合并到数据仓库时就需要数据规格化了。除非每个数据源的文本标记完全相同，并且数值度量已经经过合理的算术计算，使得不同度量之间的差值和比率有意义，否则不能同时对多个分离的数据源进行联合查询。数据规格化过程比简单的数据清洗要重要的多。数据规格化需要整个企业范围内使用标准统一的口径和度量。本书中将在第 5 章和第 6 章深入讨论此步骤。

4. 提交。 后台任务的终点就是准备好数据以方便查询。这一步骤中至关重要的是将数据物理地组织成简单、对称的框架模型，我们称之为维度模型，或者星型模型。这种框架大大地降低了查询时间，简化了开发过程。许多查询工具都需要维度框架，也是构建 OLAP 立方体的必要的基础。本书中我们强烈地建议将维度模型作为每个数据仓库后台的最终目标。在第 5 章中，我们将详细描述维度模型的结构，并为围绕着这样的结构建立数据仓库提

供充分的理由。关于维度模型的更多介绍，请参看其它的工具箱系列书籍，特别是《数据仓库工具箱（第二版）》（Wiley，2002）。

图 1.2 好像是说我们必须顺序执行抽取、清洗、转换、提交四个步骤，每个步骤之间都有定义良好的清晰界限。但实际上，ETL 系统中可能会有多个同时执行的数据流，数据清洗的步骤经常嵌入到抽取的过程中。

ODS 已经被数据仓库吸收。

十年前，操作型数据存储（ODS）还是一个独立的系统，位于源交易系统和数据仓库之间。它主要是为了回答一些业务问题而建立的紧急抽取，比如“订单是否已配送？”或者“支付是否已完成？”等等。 当数据仓库 ETL 过程的数据有一定延迟或者数据已经被聚合处理时，ODS 是有价值的。很多情况下，ODS 是从一个数据源的紧急查询抽取，它自身也可以作为数据仓库的一个数据源，因为 ODS 也是从源交易系统抽取而来。有些时候 ODS 只提供此功能而不用于查询。这就是为什么 ODS 往往有两个方面：一方面是为了查询，一方面是作为数据仓库的数据源。

现在，ODS 作为独立于数据仓库外的一个离散系统已经不再必要了。现在的数据仓库常常是以日为单位进行抽取，并且很多实时技术允许数据仓库的数据差不多达到当前的时点。与过去相比，现在的数据仓库变得更加面向运营。数据仓库和 ODS 的功能完全重合，看不到在两个系统之间有明显的界限。

最后，不论是早期的 ODS 还是现在的数据仓库常常包含有接口，允许最终用户直接修改产生的数据。

前台——数据访问

在数据仓库展示层访问数据必须与 ETL 系统的建立和管理紧密配合。ETL 系统的最终目的就是为了在展示层中提供维度模型，便于查询工具、报表工具、仪表盘和 OLAP 立方体的访问。前台的数据就是最终用户实际看到的数据。

数据集市是前台一个重要的组成部分。它是为支撑某个业务流程而建立的维表集合。有些作者也把业务流程叫做主题区。主题区是一个很模糊的概念，具有多重含义。比如，我们常常听见人们把主题区分为产品、客户以及订单等。但实际上，产品和客户实体与真正的以度量为中心的流程——比如订单有很大的差异。按照我们的观点，数据集市是以度量为中心的主题（比如订单），周围围绕着描述性实体，比如产品和客户。

虽然本书并不是专门介绍数据集市，但我们还是需要对它有很好的了解。

1. 数据集市基于数据来源，而非部门的视图。换句话说，在面向产品的企业只有一个订单数据集市。不同部门最终用户的查询工具访问数据集市时只有一个统一的、一致的订单状态。

2. 数据集市包含全部的原子级粒度数据，支持向下钻取到最低的数据级别。数据集市只包含聚合数据的观点是数据仓库设计人员最容易犯下的错误之一。聚合数据不包含最低层次的原子级数据意味着不能回答明细业务问题，不能钻取到详单。数据集市应当是一个金字塔状的维表结构，其基石就是原子级数据。

3. 数据集市既可以集中管理也可以分布式管理。换句话说，企业级数据仓库可能物理

上集中在一台机器上，而数据集市则可能等到 ETL 集结区的数据整合达到某个层次后才进行部署，或者数据集市进行独立的开发，与数据仓库的维表事实表并不同步进行。我们认为完全集中式的数据仓库只是理想情况，并不现实。更加现实的情况是增量开发和部分分布的数据仓库。毕竟，企业处于不断的变化中，需要新的数据源和新的视角。因此在真实环境中，我们关注的是数据仓库增量的、可接受的开发策略，而非理想状况下的在数据仓库完成之前对全部信息进行管理。

前台还有很多的任务和责任，超出了本书的范畴。为了避免混乱，本书不讨论下列议题：

- 为提高查询性能对展示层数据进行索引
- 选择前台工具，包括查询工具，报表工具，以及仪表盘工具
- 编写 SQL 语句解决最终用户查询
- 数据挖掘技术
- 预测、行为打分和计算分配
- 最终用户能够访问的库表及应用安全性
- 支撑最终用户工具的元数据
- 最终用户培训和文档

本书关注的是 ETL 系统如何从源系统获取数据并将其提交到前台。

数据仓库的任务

数据仓库的任务是发布企业的数据资产，用于支持更加有效的决策制定。该任务描述中的关键词是发布。正如传统的杂志发行，其成功的起点和终点是读者，数据仓库成功的起点和终点是其最终用户。由于数据仓库是一个决策支持系统，因此主要的成功标准是数据仓库是否为企业的最重要的决策制定过程提供了帮助。虽然必须仔细管理硬件、软件、劳动力、咨询服务和维护的成本，但是其隐含的成本——支持企业重要决策方面的失败——可能会更大。数据仓库的由 IT 管理的可见成本是可能战术上的，但是更为重要的决策支持成本和收益却是战略上的。

交易数据库在企业中应用已经超过 30 年。虽然十几年来我们一直把各种数据输入到特定的交易应用系统中，但很清楚的是，从这些系统获取数据来进行分析是非常困难的。在数据库应用系统上已经花费了数十亿美元的投资，但它们的数据却像犯人一样被关在系统里面。为了从交易系统中把数据取出来已经花费了不可估算的时间，但就像在迷宫中游走一样，大部分的努力都以死胡同结束。ETL 系统必须完成这个任务，那就是以可用的方式把数据交给最终用户应用系统。

建设一个无所不包的、可靠的数据仓库是一件很有意义的任务，必须有一系列的标准组件来支撑。数据仓库最重要且基础的组件就是后台和前台。该书是关于后台的。

数据仓库是什么

数据仓库是将数据从原有交易系统数据库中提取出来，经过转换后形成有组织的信息的过程，它帮助进行数据分析，支持决策的制定。此过程包括从原始数据格式到目标数据仓库格式的转换，大多数数据仓库项目中这至少要占去全部时间、人力和开销的 70%。经过多个数据仓库的实践，我们可以得到数据仓库的如下定义：

数据仓库是一个将源系统数据抽取、清洗、规格化、提交到维度数据存储的系统，为

决策的制定提供查询和分析功能的支撑与实现。

此定义减少了关于数据仓库实施成本的误解。从历史上看，数据仓库项目最可见的部分是数据访问部分——通常是以产品的形式提供，还有一部分注意力放在了维度模型上。但如果只关注这些，会在数据仓库生命周期中留下巨大的隐患。就算建立了维度模型，数据访问工具也准备就绪，距离数据仓库的真正使用仍然有一段距离，因为需要等待 ETL 过程的完成。多花些时间和精力在后台的数据管理组件上，数据仓库建设者才能够更好地把握数据仓库的真正价值——帮助最终用户的决策——同时为建立数据仓库分配切合实际的预算。

意外的延误可能会使数据仓库项目失败的风险提高，但建立 ETL 过程决不是意外的延误。数据仓库小组都知道 ETL 工程花费会占据数据仓库的绝大多数时间。但如果数据仓库建设者知道数据仓库的部署依赖于 ETL 过程的实现程度的话就不会有这种感觉。ETL 系统在时间上面临的最大风险来自于未知的数据质量问题，我们会在第 4 章中讨论如何降低此种风险。

数据仓库不是什么

构成数据仓库的核心要素到底是什么？这个问题经常被误解。时至今日，就算你询问十位专家关于数据仓库的定义，你可能会得到十个不同的答案。这些答案最大的不同可能都会落到构成数据仓库项目的组件上。要澄清这些误解，任何想要加入数据仓库小组，尤其是 ETL 小组的人员必须要知道他的边界。

数据仓库的环境包括几个组件，每个组件都有自己的技术、工具和产品。最重要的是要记住每个单一的组件都不能构成数据仓库的全部。ETL 系统是数据仓库一个主要的组件，但也需要许多其他的组件才能构成完整的实施。根据我们的经验，我们可以看到数据仓库小组也不断的挣扎在错误观念中。关于数据仓库最常犯的 5 个错误是：

1. 某个产品。与很多产品提供商所声称的相反，你不能直接买到一个数据仓库。数据仓库包含了系统分析，数据处理与清洗，数据转移，以及最后的维度模型和数据访问。没有一个单一的产品能完成数据仓库的全部过程。

2. 某种语言。 数据仓库不是一门语言，我们不必以学习一门编程语言的方式，诸如 XML，SQL,VB 等，去学习如何编码实现数据仓库。数据仓库由多个组件组成，每一部分或多或少都需要些编程。

3. 一个项目。一个数据仓库包括多个项目，将数据仓库简单的部署为一个项目的尝试几乎都失败了。成功的企业级数据仓库通常是以可管理的数据集市开始的，每个数据集市都可看成是单独的项目，带有自己的项目周期和预算。关键因素在于每个数据集市带有一致的维度和标准的事实表，这样便于将单个的数据集市集成到一个紧密的单元——企业级数据仓库中。随着各个数据集市项目的完成，企业级数据仓库将最终发展起来。因此，思考数据仓库更好的方法是将它看成一个过程，而非一个项目。

4. 一个数据模型。数据模型本身并不能构成数据仓库。通过定义我们可以知道数据仓库是一个全面的过程，必须包括 ETL 过程。毕竟，没有数据，再好的数据模型也没用。

5. 交易系统的一套备份。 另一个常见的错误是认为将交易系统复制成独立的报表系统就建立了数据仓库。就像数据模型本身不能构成数据仓库一样，只完成数据迁移过程而不重构数据存储也不能构成数据仓库。

业界术语使用不一致

本节中，我们归纳一些行业的术语，这些术语可能其它的作者有不同的称呼。可能现实中完全没有统一的行业术语，但至少在本书中使用这些术语我们能有一个清楚的立场。

数据集市

许多作者常常将数据集市定义为回答某个部门特殊的业务问题而建立的数据聚合。当然，这个定义有它自己的准则！但在本书中，我们将数据集市统一定义为，基于企业的原子级粒度的数据，面向业务流程的数据集合，只依赖于数据度量事件的本身，而非预期的用户问题。请注意下列关于数据集市不同的定义：

正确的定义	错误的定义
基于流程	基于部门
原子级数据基础	只是聚合级数据
基于数据度量	基于用户的业务问题

数据集市（也称为维度数据集市）对所有的访问者是相同的，都能访问底层的度量事件。而且，由于维度数据集市基于最基础的原子数据，因此在应用级很少发生改变；通过定义可以知道，数据集市包含了能够从数据源获取的全部明细数据。根据错误的定义建立的数据集市不能处理变化的业务需求，因为它们的数据已经被聚合了。

企业级数据仓库 (EDW)

有时候使用 EDW 这个名称来做为一种专门的实现方法（与未作为企业资产的企业数据仓库相对，EDW 通常泛指大规模的数据仓库资产）。很多人也将 EDW 称之为 CIF（Corporate Information Factory）。EDW 实现方法与本书中介绍的 DW 总线架构（Data Warehouse Bus Architecture）方法有本质的不同。EDW 中的很多主题，需要跟 DW 总线的方法对照着来解释。同时，如果将逻辑问题和物理实现问题分开来看会更有帮助。

从逻辑来看，这两者都提倡对分散在整个企业内部的不同数据源进行统一的定义。DW 总线架构采用规格化维表和规格化事实表的方式。而 EDW 的方法看起来非常的难以归类，你必须坚信如果企业信息有了一个统一的、高度规范化的 ER 模型，就能知道如何管理企业内成百上千张数据表。当然如果不考虑细节，我们也可以同意这两种方法在某一点上是一致的，那就是两种方法都试图对各个分布式数据源应用统一的标准。

即使这两种方法在创建企业数据的一致性表述这一点上达到了稍许的一致，一旦开始进入到物理设计和部署的阶段，EDW 和 DW 总线之间的差异又立刻显现出来了。

DW 总线架构采用一致的维表和一致的事实表，一致的维表具有公共的字段，这些字段的相关值域都是相同的。这一点保证了用户能够对事实表通过其关联的维表进行查询，并且也能够将列合并到最终的结果中。当然也包括横向钻取。更广泛的讲，我们还能在分布式数据仓库中管理规格化维表和规格化事实表。我们从没有在 EDW 的方法论中看到过有类似的建议。有意思的是，即使是在物理集中存储的 EDW 中，也不得不将数据存储在相互独立的表空间中，这就有必要通过相同的逻辑复制规格化维表。但我们也从来没看见过 EDW 所描述的系统过程提倡这样做。哪些表需要做同步，何时同步？DW 总线架构详细地描述了这一点。

DW 总线方法中维表的非规范化特征允许管理随时间变化的维度（SCD 类型 1，2，3）。而在高度规范化的 EDW 环境中，我们没有看到如何建立和管理缓慢变化维的相关描述。各个实体都需要大量的时间戳，与维度方法相比需要更多的管理键。另外，这里我们描述的代

理键方法实际上与维度模型无关。在 EDW 中，如果希望像 SW 总线架构一样能跟踪到缓慢变化维的维度信息变化状况，规范化的雪花型维表的根表就必须对相同数量的记录采取相同的键管理策略（使用代理键或者使用自然键加日期）。

DW 总线维度的非规范化特征使得我们能够用系统的方法去定义聚合表，这一最强大和成本最低的提高大型数据仓库性能的手段。维度聚合技术与规格化维表的使用密切相关。聚合事实表的收缩维度完全是 DW 总线中基础维表的一个子集。而在 EDW 中，没有系统的文档介绍处理聚合表的方法，对查询工具和报表工具如何使用这些聚合表也没有指导。这个问题牵扯到下面要讨论的“向下钻取”。

最重要的，大多数 EDW 架构有一个关键性假设，就是数据集市是由集中式数据仓库发布的。这些数据集市都是基于回答某个业务问题而建，这在前面的章节中已有介绍。如果用户问的问题需要原子级数据才能回答，这种假设就无法成立，又不得不离开汇总的数据集市，降级到后台第三范式的原子级数据上。以我们观点看来，这种方式的每一步都是错误的。两种架构相比，我们提倡的数据仓库总线架构在各个层面上都占优：通过规格化维表向下钻取到原子数据；缓慢变化维的统一编码；使用优化性能的聚合表；以及与查询服务隔离，保持后台集结数据的干净。

解决架构冲突：混合总线方法

有没有可能将两种方法融合呢？我们也在这样想。本书全书，我们都支持对数据清洗使用规范化数据。通过规范化过程，强制整理脏数据源的多对一关系，数据源会因此而受益。然后我们要求 ETL 小组将规范化的数据结构转换为简单的维度结构，为其后的规格化和提交步骤做准备。这包括原子级的数据。在这一点上，企业对规范化物理结构的投资可以得到利用。我们把它称为混合总线方法。

数据仓库如何改变

正如我们书中写到的，数据仓库正经历着巨大的变化，这也许是自数据仓库诞生以来最大的变化。本书中提到的数据仓库的目标是支持决策制定无疑是正确的，但很多情况下数据仓库新的发展更关注于运营和实时性。尽管对于实时性应用，前台和后台组件仍然是必要的，但传统基于批处理方式的 ETL 过程正让位于基于流方式的 ETL 过程，用户驱动型查询和报表工具正让位于数据驱动和事件驱动型的仪表盘应用。我们将在第 11 章介绍这些新的进展和它们如何扩展数据仓库的概念。

ETL 小组的任务

最后我们来简要描述 ETL 小组的任务。从最高层面讲，ETL 小组的任务是建立数据仓库的后台。详细一些讲，ETL 系统必须：

- 将数据更有效的提交到最终用户工具
- 通过清洗和转换步骤增加数据的价值
- 保护数据沿袭并进行文档化

我们可以看到在几乎每个数据仓库中后台必须支持如下关键步骤：

- 从原始数据源抽取数据

- 保证数据质量，清洗数据
- 保证来自于各个数据源的数据一致性
- 为查询工具，报表和仪表盘提交物理格式的数据。

本书中将详细介绍这些步骤。