操作

“整个系统的速度取决于最慢的部件”。

——Gene Amdahl

开发装载数据仓库的 ETL 过程仅仅是 ETL 开发周期中的一部分，生命周期中的其它阶段主要是执行这些过程。在装载数据仓库过程中，不管作业是实时执行还是批处理方式执行，其调度时间、执行顺序和执行环境都是关键点。另外，当新作业创建后，新作业的执行必须与现有 ETL 过程无缝集成起来。本章假设已经创建了 ETL 作业，我们把重点放到 ETL 的操作策略上来。

在本章中，我们将描述如何创建一个 ETL 操作策略，来保证数据仓库的数据即时可用。本章前半部分，我们将描述在系统设计完成后支持 ETL 操作的关于 ETL 调度的一些技巧和技术。

本章后半部分描述了许多在作业或系统一级度量和控制 ETL 系统性的方法（我们在第 7 章中描述了数据库软件的性能问题）。可能会有数十种控制性的方法，我们将提供一个针对你的环境非常重要的平衡的方法。

本章最后，我们提供了一种简单而有效的在数据库、开发环境、QA 环境、生产环境和基础文件系统级别实现 ETL 系统安全性的方法。

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

本章描述了执行 ETL 操作的最佳实践方法。这些操作包括了初始化数据装载、执行和监控每天的数据流、容量计划、性能监控、元数据资料库维护和控制对后台数据库的访问。

调度和支持

ETL 执行策略主要分下面两种：

- 调度。ETL 调度是一个综合的应用，而不仅仅是安排作业在给定的时间点执行这么简单。在现实情况下，作业在哪天执行其实是无关紧要的。一个有效的调度包括 ETL 作业间的关联和依赖关系的指定，并作为一个可靠的机制来管理执行策略的物理实施。
- 支持。一旦部署了数据仓库，它便一直成为一个关键应用系统。用户，也包括一些下游应用系统，都会依靠数据仓库提供他们需要的信息。如果数据仓库没有持续装载数据，就被认为是失败了。因此，为了确保 ETL 过程运行并完成，数据仓库必须能够被监控活动状况，并且必须由专人来支持。

ETL 的可靠性、可用性和可管理性分析

数据仓库具有最好的维度数据模型，最优秀的商务智能工具，并为最高领导层服务。但是在它成为企业分析信息的可靠性来源之前，还不能证明它是真正完美的解决方案。

新数据仓库的目标是创建一个一致的、可靠的数据源的备份，以此来支持企业数据分析，从而提高业务能力。为了保证成功，ETL 和数据仓库团队必须完成以下三个关键标准：

- 可靠性。ETL 过程必须持续无误运行。数据必须在任何粒度级别都是可靠的。
- 可用性。就像数据仓库管理者在最初的启动会议上向发起人和用户承诺的那样，数据仓库必须稳定的运行。ETL 作业必须在分配的装载窗口中执行并完成。
- 可管理性。请牢记，数据仓库永远没有终点。因此它必须随着公司的成长具有修改和扩展的能力。ETL 过程也必须随着数据仓库的发展而发展。为了实现可扩展性，就需要尽可能地使过程变得简单，把复杂的流程切成更小的、更简单的单元。同时，也要避免作业启动过程的过于集中。另外，设计执行策略的很关键的一步是确保支持 ETL 的能力。ETL 团队必须要为所有的 ETL 部件和为每一次为失败准备的恢复程序文档提供元数据。如果你是手工编码方式建设系统，请确保具有相应的管理技能，并能够全面地控制一个长期的软件开发环境。

ETL 管理员必须为数据仓库每一个阶段进行评价，通过可靠性、可用性和可管理性（Reliability, Availability, Manageability，简称 RAM）标准来为项目打分。作业和调度方法必须通过这三个标准，获得好的得分，才能有资格去部署。如果没有元数据或恢复文档，则 points are deducted and the processes must be revisited and enhanced or coredted。对于过度复杂事实上几乎不能维护的作业，也必须改进到生命周期的下一个阶段。每一次数据仓库的部署在进行生产之前都必须取得好的 RAM 分数。

ETL 调度 101

调度 ETL 过程只需使它们运行即可，那么为什么这里要用一章的篇幅来介绍它呢？本章不仅仅讲述执行，更主要的是讲述执行策略。策略是一套精细的、系统的行动计划。任何人都可以执行一个程序，但是开发一个执行策略却需要一定技巧。

例如，在数据仓库和 ETL 设计回顾时，用户抱怨说数据仓库直到上午 11 点才可用。根据这一信息，我们立刻开始查看 ETL 作业，找到瓶颈所在，然后才可以提出补救措施。很快我们发现作业的效率很高，从开始执行到结束不应该要花整整三个小时时间(好像有问题)。“没错”，项目的 ETL 开发者说：“每天大约早上 8 点钟我来上班的时启动它们，在三个小时后，大约 11 点钟执行完成。”我们带着怀疑的态度问了一下开发者关于自动化问题，因为在他的实施中好像并没有使用。他声称从未参加过 ETL 工具调度方面的培训，因此不得不手工启动作业。

即使是执行你的程序，你也必须系统地来完成它。非常重要的是，ETL 团队需要理解环境中的工具，并且有能力合理地调度并自动化 ETL 过程，从而持续地装载数据仓库。

调度工具

任何企业数据仓库都必须拥有一个健壮的企业级 ETL 调度机制。主流的 ETL 供应商都提供其核心的 ETL 引擎提供打包的调度器。有些甚至根据一天中不同时段提供多种方式来执行 ETL 作业，有些提供了更复杂的 ETL 执行方案，可以基于变化的关键条件来触发作业。

如果你对与 ETL 产品绑定在一起的打包调度器不满意，或者你愿意冒险使用非产品的 ETL 方式，那么你仍会有多种选择。不管是否购买专门的 ETL 调度器，还是使用已存在的生产调度机制，或者手工编码来执行 ETL 作业，生产用 ETL 调度器都应该满足相应条件，成为一个可变的企业级解决方案。

ETL 调度器必须的功能

接下来的部分将介绍一些自动化 ETL 过程可用的选项。许多选项都可用，并且每一个都随着成本和易用性不同而不同。在特定生产 ETL 环境中需要相应的功能。当选择（或建设）ETL 调度解决方案时，请确定包括下面描述的功能。

■ 令牌识别

通常，数据仓库需要从外部数据源获取数据。外部数据源提供者非常广泛，因此 ETL 方案必须能够应对这些数据。外部数据源通常是文本文件或者 XML 格式。读取和处理这些数据一般都比较简单，挑战在于如何识别已经存在的数据。与数据库源不同，数据库源可以通过查看审计列来识别新记录，而外部数据源通常都通过 FTP 将数据文件装载到文件系统的目录中。由于每次的格式都是固定的，因此 ETL 过程可以处理这些数据。但是 ETL 系统如何知道外部源的数据文件已经到达并开始其处理过程？ETL 系统必须能够识别文件已经在文件系统中存在，并自动开始执行。这个过程叫做令牌识别。

令牌是一些由文件系统创建的文件，用来触发 ETL 事件。具有令牌识别功能的应用系统可以轮询一个目录（或数据库表），用来标识令牌文件（或记录行）的到达。当处理文本文件、Web 日志或者外部数据源时，必须要防止重复处理同一文件，还要确保当文件迟到时不会错失 ETL 过程执行。令牌文件被认为是一个令牌，因为它不是必须要被处理的实际文件，它可以是一个索引文件，仅仅通过其到达来告诉处理过程开始执行。

■ 日内执行

如今每日定时处理的方式已经越来越不被接受了，因为对实时性的要求越来越高。ETL 处理必须拥有在一天当中多次执行的能力，甚至可能是随需执行。过去按月或按天增量装载是完全能够满足需求的，在实时技术还不存在的情况下，12 小时、6 小时、4 小时增量，甚至是每小时更新已经变得越来越普遍。这些聚合需求意味着不但你的 ETL 作业必须是高效的，而且你的调度系统也必须能够承受每天大量的处理过程。

此外，你的处理过程必须能够跨越午夜，在其分配的时间窗口以外重新启动。以硬编码 SYSDATE-1 方式查找“昨天”的数据，使用这种操作方式启动并从源数据中选择数据是不恰当的。ETL 系统必须能够从数据源捕获新数据，不管这些数据是何时创建的，也不管在何时执行处理过程。

■ 实时能力

实时执行是数据仓库不能被忽略的实际需求。它是如此重要，以至我们单开一章来讨论它。第 11 章列举了多种技术来进行实时 ETL 执行。实时 ETL 已经成为众多企业的家常便饭，越来越多的用户现在期望数据仓库可以不间断地更新，而且对“陈旧”的数据越来越不耐烦，实时 ETL 不是奢想，而是很实实在在的需求。

此外，随着数据仓库的发展，其价值正被越来越多的未知用户所认识。因为它提供了整洁、一致、可信赖的数据，数据仓库本身正成为一种源系统。交易应用系统越来越依靠数据仓库提供的标准数据源作为参考源数据。为了实现这一所谓的“内循环”模式，数据仓库就必须实时更新来支持操作应用系统。

■ 命令行执行

ETL 产品大多把主要精力放在为它们的调度软件包提供友好的可视化用户界面（GUI），从而为缩短入门开发者的学习时间，也为资深的 ETL 专家节省开发时间。但是，大多数企业级系统操作环境都需要从命令行界面执行 ETL 作业的能力。因为日常维护人员往往也需要维护其它更多的应用系统，而很难针对每种应用系统学习另外不同的界面。因此，你的 ETL 应用系统必须能够从命令行执行数据仓库 ETL 过程，从而使你的系统操作成员可以胜任这项工作。注意，正是因为这种原因，主要的 ETL 工具产品都提供命令行执行选项。

■ 通知和呼叫

一旦 ETL 系统开发和部署完成，它的执行方法就变成了自动操作。就像时钟一样，无需人为参与，也不会发生错误。如果执行过程中出现了问题，则必须以电子邮件方式通知维护人员。ETL 解决方案必须能够针对不同的人员、不同类型的作业以及不同类型的错误通知相应的人。在我们写这本书的时候，无线 PDF 和智能电话正迅猛发展，这些设备将成为个人操作的标准设备。这些设备上可以显示复杂的文本和图形信息，操作者可以远程向 ETL 系统发出指令。另外请注意下面的警告！

E-mail 通知和呼叫必须完全自动化，因为可能没有时间等待维护人员被人工通知。自动通知可以用下列方式实现：

集成 ETL 工具：一些主要的 ETL 产品在他们的调度应用中集成了呼叫和通知功能。这些功能也许并不十分稳定，但已经比以前好多了。最起码，你需要区分成功的装载和失败的装载，并通知相应的维护人员。另外，对于失败的关键信息必须能自动发送消息（例如，作业名称、失败时间、装载的行数、失败的行数以及最后动态失败消息等）。

第三方消息应用系统：一些公司提供了即时消息产品，用于支持 24*7 系统运行，最小化宕机时间。另外，操作管理和监控工具一般都具有通知功能，如果你的操作支持团队使用了这样的工具，则可以使用该功能。

客户化脚本：你可以选择在操作系统上手工编制脚本 来提供 e-mail 通知的执行策略。这些脚本必须能与 ETL 作业交互，并在需要的时候触发。

当设计你的客户 e-mail 通知系统时，一定要小心嵌在脚本中的邮件地址。脚本在文件系统中是简单的文本文件。脚本非常容易被一些垃圾邮件发送者获取，并且用垃圾邮件攻击你。因此如果有可能，一定要使用来自安全产品的加密技术或解决方案。

■ 嵌套批处理

批处理是一组作业或程序，作为一个简单的操作一起运行。通常，ETL 作业被分组或分批，来装载一个数据集市。而由数据集市组合而成的数据仓库，则由一批数据集市装载批处理来装载。装载批处理的批处理称为嵌套批处理。每个嵌套批处理可以包括 ETL 作业的若干层。例如，由于复杂的数据或业务规则，一个单一维表可能需要多个 ETL 作业来装载它。这些维表作业可被组合在一起作为一个批处理、这个批处理被包含在另一个批处理中，为数据集市装载维表的剩余部分。数据集市批处理又被组合到数据仓库批处理中，从而具有三个层次的批处理。理论上，嵌套批处理的层数没有限制。

ETL 作业一般都以嵌套批处理方式执行，几乎很少会在生产环境中运行单独的 ETL 作业。数据集市通常需要求每一个维表和事实表至少有一个作业，正如你所看到的，在装载数据仓库过程中，多层嵌套批处理非常普遍。因此，你的方案必须可以管理嵌套批处理。批处理管理包括下列内容：

图形界面：由于装载数据仓库时嵌套批处理的特性，ETL 批处理通常会变得非常复杂。选择批处理管理工具时，其必须具有像 Windows 资源管理器一样的目录结构导航能力。如果没有嵌套批处理的图形展示，管理就无从谈起。开发者应该可以通过 GUI 创建、删除、编辑和调度批处理，还可以通过拖拽方式在嵌套批处理和外部批处理移动作业。批处理管理最好通过图形化方式实现，并且还需要同图形化一致的逻辑命令标准。批处理之间依赖关系的可视化是维护的关键，可以清楚的理解哪个作业属于哪个批处理，另外还可以识别批处理之间的依赖关系。

依赖关系管理：当某个作业的执行取决于其它作业的成功完成时，就有了依赖关系。作业间的依赖规则由执行策略决定，并且必须由 ETL 调度系统在执行时加以约束。如果业务规则需要的话，批处理管理工具必须能够监控失败的作业并中止相应的批处理。例如，如果维表作业失败了，则必须停止装载事实表的过程。但并不是所有的情况都需要这样严格的批处理中止策略。例如，如果一个外关联验证失败，则通常仍将继续装载相应的维表。批处理管理工具应该足够健壮，按业务规则要求基于嵌套批处理来设置依赖关系。

参数共享：参数值需要在作业之间传递，或者在最外面批处理中一次性设置，然后在整个嵌套批处理中作为全局数据使用。批处理管理器必须包含参数管理功能。更多关于参数管理的内容在本章后面部分将要详细介绍。

平滑重启：如果一个作业在执行过程中发生错误该怎么办呢？你如何确切知道哪些作业已装载哪些还没有？如果重新启动的话，批处理管理工具必须能够系统地确定哪些记录已经处理并装载，从而只处理剩下的输入数据。必须要注意装载过程中的失败点。通常，如果只为支持重启这种情况，ETL 系统应该有一些策略点（数据被写到磁盘的处理步骤单元）。而且，如果某个 ETL 步骤需要人工干与来纠正数据，则更要特别注意。这些手工操作至少要保存在日志中，以便在 ETL 处理步骤必须要重新运行时可以再次调用。

顺序/并发执行：有些情况下需要顺序装载表。例如，当表间有依赖关系时，就必须在装载子表之前装载父表。与某维表的外关联是这种父子顺序的最好例子，同样也适用于普通维表和事实表。在装载完所有维表之前不能装载事实表。另外，有时顺序装载表而不是并发装载是为了平衡服务器的负载。如果你试图一次性装载数据集市中的所有维表，则可能会由于资源消耗过多而造成 ETL 服务器不能运行。相反，对于长时间运行的过程，你可以把作业拆分成多个小作业，让他们并发执行来提高装载效率。假设有足够的资源可用，则尽可能多的使用独立过程并发执行来最大化处理效率，并最小化装载窗口。更多关于并发和并行处理的内容在本章后面有详细地介绍。

预先/后续执行动作。在 ETL 过程之前或之后简单运行脚本并不属于执行管理范围。批处理管理必须在运行核心 ETL 作业之前知道已经成功执行了预处理脚本。而且，如果核心 ETL 作业成功执行完毕，必须要触发后处理脚本。最后，脚本除了能够在作业一级执行，同样也要能在批处理一级执行。这对于批处理的并发执行来说尤其重要，在每次批处理执行过程中最后完成的作业是不确定的。而且，在所有作业完成后还需要触发一个后处理脚本。

元数据捕获。批处理管理器控制下的所有元数据都必须可以被捕获、存储和发布。在最好的情况下，元数据应该存储在开放的资料库里，以便在其它应用系统间共享。每一个 ETL 作业都有一个调度执行时间，其执行频率、参数和恢复过程等所有元数据的形式必须能被展示，并且很容易被需要支持装载过程的人（也包括业务用户）获取。至少，元数据必须拥有报告能力，以便用户和开发者深入到数据仓库 ETL 操作的内部去观察。请参考第 9 章来了解更多关于 ETL 元数据库的内容。

ETL 工具在失败恢复方面已经做的很好，但平滑重启仍是一个苛刻的需求，还没有完全实现。很多情况下，对于处理过程中失败的情况，最安全的办法仍是删除已经部分装载的数据，然后重新开始这个失败的 ETL 过程。如果你使用工具包自动恢复丢失的数据，则强烈建议花些时间来审查已经完成的数据，从而确保数据质量和完整性。

■ 参数管理

ETL 系统在整个开发周期过程中在不同的环境间迁移。由于这个周期也包括 ETL 系统内的代码测试，因此不能在环境迁移间修改代码。在 ETL 系统中定义变量不能在代码中将参数写死。参数是在代码中替代常量的一种方式，一个健壮的 ETL 调度系统必须拥有为执行 ETL 作业管理和传递参数的能力。参数方式增加了 ETL 作业的灵活性，因此可以在不对应用重新编码的情况下平滑地改变环境或抽取规则。例如，一般情况下 ETL 作业的生命周期是在开发环境中开发，在测试环境中测试，然后最终迁移到生产环境中来支持产生数据仓库。ETL 生命周期中的每一个环境都有其特定的数据源、处理策略和目标数据库，以及文件系统和目录结构。通过参数方式来改变这些环境，ETL 系统就可以无需修改代码来指定相应的文件或数据库。因此，必须参数化环境变量，并在运行时让调度器传递应用数据到这些变量。

常用的参数项列表包括：

服务器名
数据库或实例名
结构描述文件名
数据库连接信息（不要把口令放到文本文件中！）
根目录或用来找到有用的控制文件的目录
元数据库连接信息

调度器必须能管理两种类型的参数：

全局参数。全局参数可以同时支持多个 ETL 作业。一般情况下，ETL 作业可以有很多个全局参数。例如，目标数据库名可以设为全局参数。否则，必须为装载数据库的每一个作业重复维护这个参数。

局部参数。局部参数只在一个 ETL 作业内有效。局部参数在某个作业内修改其值，而不会影响到批处理中的其它参数。局部参数的一个例子是设置用来从源表获取数据的最早日期。

本地 ETL 工具调度器是获取一个健壮的参数管理系统最好的证明，因为调度器通常都在本地与 ETL 引擎集成在一起。与 ETL 引擎的本地集成使得调度器可以在两个部件之间有效地通信。由第三方提供参数管理的架构可能效率不太好，但更灵活。不支持运行时设置参数的 ETL 解决方案会失去 RAM 的管理能力。

在企业环境中，为 ETL 作业中的参数生成元数据是非常重要的。如果没有健壮的参数管理系统，参数也可以通过文件系统中的文本形式进行管理维护。通过使用文本文件，操作团队可以简单地修改参数文件，而无需进入 ETL 系统。

ETL 调度器方案选择

在前面部分，我们描述了企业 ETL 调度系统应该具有的功能，另外还有一些实现同样功能的选择。在本节，我们提供了 5 种创建 ETL 调度器解决方案的选择：

1. 集成的 ETL 工具
2. 第三方调度器
3. 操作系统
4. 实时执行
5. 客户化应用

关键是要选择一种健壮的解决方案来满足你所有的需求，这些需求是基于你关于装载数据仓库的作业的需要，另外数据仓库建设要控制在预算范围内。最后要请教身边的专家来做决定。下一节将评估这五种选择的优劣。

■ 集成的 ETL 工具

一般的 ETL 工具都会协同调度系统来执行由其工具集创建的 ETL 作业。有些工具提供了最少的功能，而有些则拥有健壮的调度应用系统。如果不是强制使用操作支持团队已经建成的工具（作为标准的调度工具），而 ETL 工具又包括一个健壮的调度器，则最好使用集成的 ETL 调度器。集成的解决方案的优势如下：

ETL 厂商的产品支持。为两个应用使用同样的服务级别协议（SLA）。富有 IT 经验的人可能会对多厂商解决方案带来的混乱深有感触，需要说明的是，这从来不是某个厂商的问题，而是其它厂商与其兼容不好所带来的问题。使用同一厂商的产品包可以充分发挥厂商的支持能力，从而加快技术问题的解决。

调度器与 ETL 引擎的集成。集成包设计为在部件间传递参数，并且从本质上强调作业间的相关性。作业间的依赖关系指的是某个作业的执行可能依赖于另一个或一组作业的成功完成，这种关系对于合理地装载数据仓库和从 ETL 失败中恢复来说非常关键。

ETL 组中的工具集知识。由于 ETL 工具集是 ETL 团队的专用工具，他们可以在不学习其它应用的情况下直接建立执行策略。此外，一旦某个 ETL 作业充分测试过，则它在生产过程中极少会产生失败的情况。当作业确实失败时，ETL 团队通常需要某种级别上参与其中的能力。在 ETL 工具集控制范围内保留 ETL 调度，则团队可以很容易切换到技术支持角色来帮助失败的 ETL 过程进行恢复。

■ 第三方调度器

许多生产支持部门都制定了一个独立的调度系统标准，所有其它应用系统都必须适应它。在某些企业环境中，数据仓同其它应用系统一样，必须遵守生产支持团队制定的规则。在这种情况下，ETL 由支持整个企业环境所有应用的调度系统触发。操作企业级的调度系统是超出 ETL 团队知识范围的一个专项工作。对于失败恢复不能顺利完成的情况，ETL 团队需要紧密地与生产支持团队合作。

如果生产支持团队坚持要通过他们标准化的企业级调度应用来执行 ETL 作业，请确保它具有支持 ETL 执行策略所需的功能，包括作业间的依赖关系、参数管理和通知及告警。

■ 操作系统

通过本地操作系统调度系统（例如 Unix Crontab 或 Windows 调度器）来执行 ETL 过程是很常见的作法。即使你已经有了一个非常好的 ETL 产品，很多生产支持团队还是需要在生产过程中用脚本来执行 ETL 作业，因为这是整个企业中所有应用系统最常用的方法。通常，任何应用都可以在操作系统一级通过命令行或脚本来执行。在 Windows 平台下，批处理、.BAT、VBScript 或者 Jscript 文件可以用来管理 ETL 流程的执行。在 Unix 平台下，Crontab 用来运行作业。操作系统调度器可以直接执行 ETL 作业或者通过脚本来执行。正像大多数程序员所知道的，脚本的优势在于非常灵活。可以使用脚本语言来创建非常健壮的应用类型的逻辑。大多数的 RAM 标准都可以用脚本来满足。此外，Perl、VBScript 或者 JavaScript 可以运行在 Unix 或 Windows 平台下，在执行装载数据仓库的作业同时处理复杂的业务逻辑。事实上，脚本语言也可以在作业内提供逻辑功能。然而，我们仍然建议用健壮的 ETL 工具创建和维护 ETL 作业。使用脚本而不是 ETL 调度工具的劣势在于它没有元数据，任何关于 ETL 调度的有用的信息都在脚本中，因此必须由程序员来解释脚本中的信息。在执行脚本中维护元数据可以使用两种技术。

电子表格。ETL 管理员或程序员必须维护一个电子表格，其中包含了重要的元数据，包括参数、批处理中的作业、执行时间等等。

数据表：动态脚本方案是元数据驱动的。所有相关的元数据都存储在数据表中（数据库或者文件），它们在运行时传递给脚本。当集成的 ETL 调度器不可行时，元数据驱动脚本是个可行的方式，应该创建并使用。

- 实时执行

如果数据仓库的一部分是实时的，则需要选择一种机制（在 11 章详细介绍）来支持实时的需求。整个数据仓库实时加载是很少见的，通常，数据仓库的一部分可能会实时装载，而其它部分则周期性的批处理装载。必需要注意两种 ETL 技术的结合，以此来确保一个无缝的、一致的解决方案。

- 客户化应用

我们总会选择创建一个客户化的调度解决方案。然而，还没有一种合理评判客户化调度应用的方法，但这并不影响我们创建它。如果选择用脚本执行所有的作业，则有必要创建一个应用来管理它们，但是创建一个客户化的 GUI 则显得有些过分。通常情况是使用脚本编程，再加上元数据表，是客户化 ETL 调度较为可行的方案。

装载依赖

在作业间定义依赖关系也许是批处理 ETL 作业最重要的内容。如果一个子维表装载作业失败，也许你还可以继续装载这个维表，但是如果维表装载失败，你还能继续装载事实表吗？这肯定是不赞成的。作业间依赖关系属于元数据，装载过程必须要了解这一点。ETL 功能需要操作元数据来进行恰当地操作。如果一个事实表 ETL 过程在所有其相关维表成功装载之前执行，它将不会发生装载错误，因为没有数据约束。另外，如果事实表没有设计成可以执行更新操作，则在重新执行处理过程前，所有错误数据必须手工恢复或者删除掉。手工干预是纠正失败 ETL 装载成本最高的方式，在 ETL 作业间强制依赖关系可以节约大部分这种成本。依赖关系保证了桥接表和维表间，层系映射表和维表间的逻辑一致。使用前面所述的列表作为作业依赖关系定义的参考。具体说来：

- 在子维表成功完成之前不要装载维表
- 在维表成功完成之前不要装载桥接表
- 在父表（包括桥接表和维表）装载完成之前不要装载事实表

然而，头脑中始终要记住一个明智的数据仓库格言：针对每一个拇指规则，都会有相应的其它四个手指副规则。例如，如果一个维表设计为更新用来关联自己到子维表的外键，则当子维表装载失败，没有必要停止装载整个数据集市。因此，任何时候作业没有按期望的完成，调度器都要产生一个告警。

元数据

想像一下如果火车在没有公布其运行时间表的情况下运行，其它人如何能知道这列火车的运行情况？在没有公布元数据的情况下开始执行策略对于用户来说是非常危险的。在本章前面部分，我们讲到调度器必须能捕获元数据，包括内容和批处理、嵌套批处理的时间表，元数据必须像对数据仓库团队一样对业务用户可用。批处理元数据就像数据仓库的火车运行表，它应该事先告之何时数据将到达，何时对用户可用。

调度系统还应该在数据迟到时告诉用户，这与前面部分讲述的失败通知不一样。数据可

用性元数据是交互的关键，同时也是响应用户期望的关键机制。用来通知用户数据到达的元数据属于过程元数据的类型。过程元数据捕获 ETL 过程中的操作统计。典型的数据包括装载成功的行数、拒绝行数、消耗时间、每秒处理行数，以及已完成每行所用的时间。这些都是非常重要的过程元数据，因为它可以帮助用户了解情况，就像火车站的广播一样。

在清理和提交步骤中收集的元数据有多种操作角色。比如给 ETL 团队提供建议数据是否已经适合提交给最终用户。审计元数据中的数据意味着可与普通数据组合生成特定方式的数据质量报告，它既可以用于报表结果的可信度，还可以用于保证报表的一致性。因此，清理和提交元数据为提高源始数据的数据质量所采取动作的导向。

所有由批处理管理器控制的元数据都必须能被捕获、存储和发布。最好的情况是，元数据应该存储在开放的资料库中，从而可与其它应用共享。元数据至少应该有生成报告能力，因此用户和开发者可以深入了解数据仓库 ETL 操作的具体细节。

迁移到生产系统

迁移过程根据因素不同有许多种情况，这些因素包括指导思想、技术架构以及 ETL 工具集。通常，ETL 团队属于开发团队的一部分，应该避免由于提供第一级的针对数据仓库的生产支持而过于分心，除非 ETL 组足够强大，可以保证拥有一支专门的 ETL 生产支持队伍。

本章的目标是假设 ETL 团队仅作开发工作，当其生产环境的各种任务准备完毕时，就把工作交给生产支持团队。然而，这一过程可能由于组织架构和环境中所用的工具变化而有不同。该节内容应该作为结束 ETL 生命周期实现的参考。

数据仓库的操作支持

有意思的是，大多数书籍和文章都讨论数据仓库团队需要如何维护数据仓库。事实上，至少是基于我们的经验，数据仓库团队（包括 ETL 团队）只是开发者和分析员。他们收集所有业务需求，分析它们并且创建数据仓库。一旦创建完成，他们通常把它交给另一支队伍，由这支队伍监控和维护生产环境。

数据仓库架构师和模型师负责维度建模，ETL 管理者负责组装按维度设计的数据仓库，ETL 发开团队创建装载数据仓库的过程，质量保证团队写测试计划来测试它们。数据仓库需要移交给组织内部的团队，以便支持每天的操作。如果用户只是一个小公司，或者数据仓库仍旧处于初级阶段，开发团队可能会支持 ETL 生产阶段的操作。但是随着数据仓库的成长（添加更多的数据集市），开发团队需要从生产环境的支持和操作这些日常的工作中解脱出来。

一旦 ETL 过程经过了开发和测试，则第一级的数据仓库操作支持和 ETL 工作应该由一支专门作生产监控管理的特定队伍提供，而不是由数据仓库开发团队来完成。只有当操作支持团队被问题困扰已久而不能解决时才需要求助于数据仓库团队。

版本发布

一旦 ETL 团队克服了 ETL 开发过程中各种难题的挑战，并完成了管理装载数据仓库或

数据集市需要的作业的创建，则这些作业必须打包并迁移到下一环境，与数据仓库管理团队已经实现的生命周期一致。

与 ETL 工具提供商仔细讨论一下 ETL 部署的问题。工具是否支持所有版本增量的脚本，以便可以用简单的命令行实现从测试系统到开发系统的迁移。或者是否支持所有文件一次性的打开、检查、关闭和转换吗。

针对每一个数据仓库版本，开发团队应该提供如图 8.1 所示的发布过程文档。

环境：质量管理环境（QA）
服务器名：从na-dwqasvr-01迁移到na-dwprdsvr-01
ETL程序：从DWQA迁移到DWPRD
数据库名：从DWQA迁移到DWPROD

文档文件名：**DW_Release_5.0_SALES_PROD.doc**
最后修改日期：16-Feb-04 7:10:00 AM
修改类型：Major Release
发布号：**5.0**

系统修改请求的发布如下：

修改请求	项目领域	修改类型	描述	沿袭报告名称	PVCS脚本名称	PVCS版本
10845	零售	数据模式/ETL模式	新的星型计划 增加到零售分析	sales_mart_data_lineage.xls	sales_by_month_release_5_build1.zip	

系统需要如下手续来移动数据集市到另外一个环境：

☑ 标准发布手续

☐ 手续

☐ 项目编码

- DDL、ETL脚本为 **Sales_By_Month_Release_5_Build1.zip.**

☐ 表/视图

- 在DW_PROD数据库中顺序执行下面的脚本

从 PVCSProject:/DWQA/sql 取得DDL脚本
DW_SALES_MART_STEP1_CreateTablesIndexes.sql Version 1.0
DW_SALES_MART_STEP2_CreateStagingTables.sql Version 1.0
DW_SALES_MART_STEP3_CreatePubSynonyms.sql Version 1.0

☐ 安全视图修改

无应用

☐ SQL包

- 如下的包需要移动到DWPROD

从PVCS取得脚本
DWQA/packages/PKG_SALES.sql Version 1.0

☐ 配置修改

- 包括在ETL服务器上登陆 **intnsnames.ora**

无应用

☐ ETL作业

- 迁移所有的PVCS或者ZIP文件

从下面的文件中导入所有的任务到DWROD中
DW_SALES_MART_Release_5.0_ETL_Jobs.zip Version 1.0

☐ 复制SHELL脚本

- 从 **/dw/sales/qa/shellto /dw/sales/prod/shell** 复制文件

无应用

☐ 复制SQL脚本

- 从 **/dsprojects/DWDEV /sql** 复制文件到 **/dsprojects/DWQA /sql**

无应用

☐ 复制平面文件

无应用

☐ 复制/编辑参数文件

- 在 **/dsprojects/DWQA/paramfile/param.ini.** 修改参数

如果如下参数在.ini文件里不存在，那么把值设成如下：
TgtStageUserName：设成dwetluser
TgtStagePassword: 设成DB用户的密码

☐ 附加注释

☐ 执行步骤

- 如下步骤来完成当前的发布

1、做所有的上面提及的步骤
2、运行BatchSalesLoad.sh

图 8.1　数据集市发布文档

数据集市发布文档介绍发布以及提供用于迁移和支持发布所需的技术细节。文档包括下列内容：

- 环境：包括源和目标环境。环境迁移通常是指从开发环境到测试环境或从测试环境到生产环境。如果还有其它更多的专用环境，如用户体验环境或质量管理（QA）环境，这些环境也包括在内，这依赖于你的项目处于生命周期的哪个阶段。
- 服务器名：迁移过程中环境中服务器的物理名称。列出了 ETL 和 DW 服务器列表。
- ETL 程序：列出了程序所在的目录。如果在使用某种 ETL 工具，这可以为发布使用组件来标识正确的程序或作业。
- 数据库名：迁移源或目的数据库。通常是开发环境到 QA 环境或从 QA 环境到生产环境。
- 文档文件名：包括迁移信息的文件名，其中包括恢复过程的操作步骤。
- 最后修改日期：发布文档修改的最近时间。
- 修改类型：关于发布类型的描述，类型包括主体发布、小版本发布或打包。请参考第 2 章关于发布类型更详细的解释。
- 发布号：数据仓库成为最后发布结果的版本号。
- 修改请求号：对应的是部署过程中作为范围管理过程规定并包含的请求号。
- 过程：迁移作业的操作步骤。通常在发布文档中提供以下标准信息内容：
  - 项目代码：版本管理工具中的字段，用来查找用于创建数据仓库的版本的代码。
  - 表/视图：为新的数据集市创建新结构和索引所用的数据定义语言（DDL）。
  - 安全：支持发布的任何新安全策略或需求。
  - SQL 包：ETL 使用的数据库存储过程。
  - 配置修改：针对发布所需的全局设置和配置项，例如 TNSNames（Oracle 远程数据库名称的别名）或目标数据库连接名（ODBC）。
  - ETL 作业：发布所需的 ETL 作业，通常指版本控制管理器或 ETL 工具中某个域。
  - Shell 脚本：ETL 依赖的操作系统级前置/后置执行的 shell 脚本。
  - SQL 脚本：ETL 依赖的操作系统级前置/后置执行的 SQL 脚本。
  - 文本文件：文本文件列表，也包括新数据源或 staging 文件的路径。
  - 编辑参数文件：用于带有可管理参数的环境，在这里列出针对 ETL 过程所用的新的源数据系统数据库或任何新的修改了的参数。
  - 附加注释：任何其它关于发布的指示或注释，可用于帮助系统操作团队完成迁移工作。
  - 执行步骤：执行作业直接的指导。对于新的数据集市或独立的 ETL 过程，你可以指定运行的调度计划或频率。

一旦数据集市发布文档完成，ETL 团队会将文档交给实施团队。在迁移过程中，ETL 团队应该随时待命，以防可能发生的影响生产系统的紧急问题。对于第一次发布的数据仓库版本，可能 ETL 团队的快速响应还不是一个关键需求，但随着数据仓库逐渐成为关键业务应用，任何关于系统的问题都将对整个企业组织影响重大，必须着力避免。一旦迁移完成，ETL 团队可以再回到常规数据仓库的下一阶段开发工作中去。生产环境的日常支持应该由专门的生产支持团队提供。下一节将介绍数据仓库所需的不同的支持级别。

在生产环境中支持 ETL 系统

本章的开始部分介绍了如何使 ETL 过程在获取针对调度、批处理和迁移 ETL 作业方面多个技术方面的运行和共享的建议。本节重点介绍处于生产环境的 ETL 的支持。通常来讲，任何软件实施的支持都包括三个级别：

1.第一级支持。第一级支持通常指针对各种情况的帮助桌面。如果发生失败或者用户收到数据错误，则会通知第一级支持。第一级支持团队利用由 ETL 团队提供的专门的过程来做各种尝试，如果实在不能解决问题则进入下一级支持。

2.第二级支持。如果帮助桌面不能解决支持问题，则通常会通知系统管理员或 DBA。支持的第二级支持通常精通技术，可以提供常规架构类型失败的支持。

3.第三级支持。如果生产操作技术人员仍不能解决问题，下一个通知的是 ETL 管理者。ETL 管理者拥有解决大多数生产环境问题的知识。有时 ETL 管理者会针对特定的情况与开发者或外部供应商交流来获得建议。

4.第四级支持。当所有这些都失败时，则直接找源头去吧。第四级支持需要实际的 ETL 作业的开发者分析代码来发现 bug 或解决问题。如果问题是由供应商产品 bug 引起的，则去找相应的供应商来提供对产品的支持。

在较小的环境中，将支持级别三和级别四进行合并是可以接受的，但这种合并通常会给第二级支持团队带来很大的负担。我们不建议每次发生作业失败都去找 ETL 开发者。第一级支持不能仅是电话服务，在进入一级支持前他们必须要做最大的努力来解决生产环境中的问题。

获得最优的 ETL 性能

现在，你已经通过本书完成了一个优秀的 ETL 方案。但是不要高兴太早，工作还没有完成！随着数据仓库的扩展，必须要确保 ETL 方案能伴随着一起成长。一个可扩展的 ETL 方案意味着设计的过程能够在不需要重新设计的情况下处理更大量的数据。设计必须能有效的执行，以此获得处理装载比现有数据量大得多的数据时所需的性能。可扩展性和性能是设计数据仓库 ETL 时不可忽视的两个因素。

ETL 开发者、DBA 和数据仓库管理团队将从本章获益非浅，因为它描述了监控和提高现存 ETL 执行效率的策略。本章假设你已经完成了分析工作，设计好了逻辑数据流程以及实现了物理 ETL 过程，从而直接进入到应用新的或现有的 ETL 作业来获取更优技术性能的细节和技术。

在本章末尾，你将得到在 data-staging area 如何处理安全问题的一些技巧。我们尤其阐述了 FTP 协议的不足，并提供了加密和解密数据方式来保证有效的安全的方法。读完本章，你将能够对数据仓库团队提供专家级的 ETL 调优技术。本章所涉及的技术都是针对 data-staging 环境，而不能应用在数据仓库展示层调优。

如果你需目标数据仓库的优化信息，我们建议你阅读 Gary Dodge 和 Tim Gorman 写的<<Essential Oracle8i Data Warehousing:Designing,Building,and Maintaining Oracle Data Warehuses>>一书。

估计装载时间

评估把从交易系统的所有历史数据装载到数据仓库的初始化时间是非常困难的，尤其是当这个时间会持续几周甚至几个月时。在整本书中，我们根据 ETL 整体目标，往往都把抽取、转换、装载过程作为一个整体来看待。然而，当估计一个很长的初始化装载时，就很有必要将其 ETL 系统分成三个独立的过程：

- 从源系统抽取数据
- 将数据转换成维度模型
- 把数据装载到数据仓库

估计抽取过程时间

令人惊讶地是，从源系统抽取数据是 ETL 过程最耗时的部分。装载数据仓库的历史数据过程从一个查询抽取了大量的数据，而且联机交易处理（OLTP）系统并未设计成可以返回如此大的数据集。然而，一次性历史数据库装载与每天的增量装载是大不一样的。

但是无论如何，交易系统没有设计成按生成事实表所需的方式提供数据。ETL 抽取过程经常需要其它的方法，例如视图、光标、存储过程和相关子查询等。在抽取开始之前估计一下它将执行多长时间是非常有必要的。但以下不利因素为直接估计这一过程带来了不便：

- 测试环境的硬件通常比生产环境要差很多。基于测试环境估计的 ETL 过程执行时间可能是完全失真的，因为测试服务器和生产服务器之间存在着巨大的差别。
- 由于抽取历史数据的 ETL 作业可能持续几天，因此在完整数据集上执行测试是不可行的。我们都遇到过在项目中抽取作业连续运行很长时间后最终失败了，而再次执行后最终还是以失败告终，这样反复几次，还没有产生任何效果的情况下数天或数周已经过去了。

为了克服处理海量数据的困难，应该把抽取过程分成两个小的部分：

- 执行响应时间：从查询开始执行到开始返回数据为止的时间。
- 数据集获取时间：第一条返回的记录到最后一条返回的记录之间的时间。

由于初始化抽取装载数据量都是非常大的，因此建议取一部分数据进行估计。大多数情况下，装载中的事实表被分区，我们建议针对整个事实表的一个样本分区获取足够的数据来生成它。使用数据分区作为样本是因为数据库分区是把数据（或多或少）分隔成大小相等的部分，这样可以提供一个有效的基准。一旦知道了抽取一个分区的时间，则乘以事实表的分区数目就可以估计出整个抽取时间。该种方法的不足是把查询响应时间和数据获取时间合在一起，这将会造成估计失真。

- 计算查询响应时间

把两个过程分隔开的最好方法是使用查询监控工具，大多数 ETL 工具在它们的应用中都有监控工具。如果正在使用 ETL 工具，请记住涉及的数据表应该在主抽取过程开始前将装载到内存中去。因此，需要将缓存处理从原始抽取过程中分离出去。

- 计算数据获取时间

一旦抽取查询开始返回数据，则开始精确记录装载一部分数据所用的时间，其中要选择对你的估计是有意义的一部分数据。比如，如果要获得 2 亿条记录数据装载，也许 100 万是一个不错的测试选择。当数据装载刚刚达到 100 万条记录时停止作业，查看所用的时间，然后乘以 200（2 亿条记录/100 万测试记录）来估计整个历史数据装载的时间。

一旦测试了整个抽取作业，则将估计的样本作业放到生产环境中去执行，以防

止在较小的环境中运行可能发生的结果失真情况。

估计转换过程时间

一般人都认为操作数据可能是非常耗时的过程，但令人意外的是实际的数据转换都是在内存里以惊人的速度来完成的。相对于它的兄弟过程，抽取和装载，物理转换数据所用的时间是非常不成比例的。如果你使用了带有光标的存储过程，则考虑重新设计系统吧。根据实际情况，最好是尽可能利用 ETL 工具而不用数据库存储过程来完成转换过程。大多数 ETL 处理都是在 I/O（物理磁盘的读和写）上耗时。如果转换耗时没有比抽取和装载过程少很多，则一定在转换逻辑中有严重的瓶颈。估计转换时间的最简单方法是得到抽取时间和装载时间估计后再运行整个过程。一旦你有了这些统计值，减去抽取和装载的时间就是转换所用的时间。

估计装载过程时间

当计算装载时间时，需要确保数据转换不会造成延迟。即使数据是以流水线操作的形式从转换到装载的，为了估计准确，最好是将转换的数据放到文本文件中。

很多因素影响装载时间。需要考虑的最重要的两个因素是索引和日志。请确保在测试阶段的环境与现有生产环境的物理条件相一致。像数据获取一样，数据装载也是按比例处理的。这意味着可以只一次装载一个样本集数据（2 亿条中的 100 万条），然后再乘以 200 来得到整个装载估计时间。

长时间运行 ETL 过程的不足

ETL 过程的目标是从源交易系统选出数据、转换它并把它装载到数据仓库。ETL 团队的目的是在面对这些复杂任务的时候设计有效的过程来应对意外情况。

水平 ETL 系统流和垂直 ETL 系统流

ETL 系统本质上按水平或垂直方式组织。在水平方式组织中，一个特定的“抽取-清洗-规范化-提交”作业从头开始运行直到完成，与其它主要的数据流没有依赖关系或很少的关系。这样，一个客户定单 ETL 作业可以完成运行，而另一个库存跟踪 ETL 作业则可能会失败，这就留给组织中的决策制定者一个不一致的和不能接受的结果。

在垂直 ETL 系统组织中，多个 ETL 作业关联在一起，每一个作业中的并行步骤各自完成运行，但要等待其它作业以到达同一执行点。还是用上面的例子，客户定单和库存跟踪就需要都完成抽取步骤后才能进行清洗、规范化和提交步骤。

在 ETL 系统中使用水平方式还是垂直方式来组织，依赖于环境中的两个主要因素：

1. 需要多个 ETL 作业并行处理的详细的数据依赖关系。（即，在库存跟踪作业失败时，也许因为客户定单作业没有定义好的产品编码。）
2. 最终用户修改部分数据的必要性（例如，定单已经修改，但运输却要延迟一天）。

分析失败类型

不幸的是，成功执行并完成的 ETL 过程依赖于多种因素。一旦 ETL 过程用于生产，失败通常的原因都超越了过程控制本身。生产用 ETL 失败的原因主要包括：

- 网络失败
- 数据库失败

- 磁盘失败
- 内存失败
- 数据质量失败
- 未公告的系统升级

为了熟悉环境中可能对 ETL 过程带来威胁的这些因素，下面详细介绍每一个因素并提供最小化风险的建议。

网络失败

网络通常是连接数据仓库各部件的最底层结构。无论是数据库服务器还是应用服务器，都通过内部网络连接在一起。随着电缆、路由器和节点的增多，网络故障的风险是不可避免的。网络失败永远不能完全避免，但是 ETL 可以采取措施最小化网络失败带来的危害。

减少这一风险的预防措施之一是把 ETL 引擎和目标数据仓库数据库放在同一台服务器。很明显，这种方式会使 ETL 作业和最终用户查询之间的资源争夺加剧，但是就最小化网络失败可能性来看，实际上它可以减少 50%的网络传输，因为数据可以从 ETL 引擎在服务器总线内部传递到目标数据仓库。很多情况下，如果数据仓库查询一般在白天发生，而 ETL 过程主要在夜间占用大多数系统资源，这种共存就显得更有意义。

数据库失败

切记，初始化装载并不只是发生在数据仓库实施的开始阶段。如果是使用数据仓库总线架构实现的企业级数据仓库，则数据仓库的每一阶段都需要初始化装载。每一数据集市都需要在增量更新之前完成历史数据的一次性装载。物理数据库失败是常见的不可预期的宕机情况。当今可用的技术（事实上把一切都冗余了）使得不可预期的宕机可以被避免，但必须要确保拥有复杂的服务水平协议（SLA）来指定可被接受的不可预期的宕机率。

此外，数据库并不是只有宕机才被认为是出了问题。ETL 团队的目标之一是管理所需的过程来装载数据仓库，而整个过程对用户透明。如果一个 ETL 过程中某个表锁住了，或者临时空间不足了，按用户的经验这也属于失败。因此这种类型的失败和数据库物理失败同样有害，会损害数据仓库的声誉。

磁盘失败

数据仓库中的数据存储也许是整个数据仓库所有潜在失败点中最薄弱的环节。通常，在 ETL 过程中有三个磁盘组：

- 源系统磁盘：通常情况下，ETL 过程仅是从源系统磁盘读数据，失败风险很小。然而，当从系统中抽取原始历史数据时一定要注意。多年历史数据的抽取可能是一个非常大的数据集。如果运行复杂的带有多个连接和 order by/group by 子句的抽取查询，也许将超出为这种类型的操作所分配的磁盘空间。为了减少这种风险，最好与源系统的 DBA 团队一起工作，在执行历史数据装载测试过程时监控临时空间使用率。并确保在运行整个过程之前分配了足够的空间。
- 集结区：集结区是 ETL 过程的工作台。通常，它包括一组集结的文件，描述了从源到最终维度目标的数据流中不同的步骤。数据一般在每一个主要的步骤（抽取、清洗、规范化和提交准备）后都要立即被集结。对该域的读和写主要有多种原因，其中包括为了数据连续性和数据保护，以及在转换过程中保持数据独立性。数据集结域可能有源系统和数据仓库组合的那么大，然而这通常很罕见，但是切记这是有可能的，并且数据集结数据库经常在数据仓库和 DBA 团队管辖范围之外。作为预防措施，经常性地检查数据集结环境的可用空间来确保运行不至于太慢。

- 数据仓库存储：数据仓库可能成长的比初始预期要快得多。尤其是索引和临时空间总是被低估，总是超过所分配的空间。当 ETL 过程试图写入未分配的磁盘时，该过程就可能崩溃。从磁盘空间这样的错误进行恢复往往是非常困难的。为了防止超过空间，你需要向 DBA 团队撒个小慌，夸大一下估计的初始化装载容量和数据仓库估计的三个月的大小。我们过去总是把估计加倍，但是现在，经历一些事情后，我们认为三倍才是比较安全的。我们建议把估计的初始化装载乘 3 以防止潜在的灾难。请相信我们：空间总是不会浪费的。

仅仅估算一下容量和利用的存储大小是远远不够的，ETL 团队应该经常监视这些量，并在达到某一告警值时采取措施。我们见到过告警值设在容量的 90%，但是这一告警被忽视了，直到 6 个星期后，砰!!!

内存失败

在下面三种环境中内存都可能失败：

- 源系统
- Staging area
- 数据仓库

这些环境在分配的内存超载时都同样脆弱，会造成 ETL 过程的失败。内存超载并不一定会使过程崩溃，但当开始使用虚拟内存（操作系统让数据在 RAM 与磁盘间交互）时会严重降低运行速度。请确保咨询 ETL 应用供应商来获得关于针对特定历史数据装载的缓存设置方面的建议。

如果硬件坏了，则需要纠正这一问题并重新启动过程（除非 ETL 工具可以平滑地恢复）。

临时空间

临时空间是数据库中的域，在进行排序或连接数据时使用。当在一个针对交易处理的数据库环境中执行数据仓库类型的查询时，临时空间经常被耗尽。由于这一主要原因，为了避免临时空间的耗尽，历史数据应该以简单格式从源系统抽取到 staging area，然后再在专门的 staging 环境中完成转换。如果填满了所有数据库环境中的临时空间，处理过程都将停止。

当发生了临时空间失败的情况时，DBA 团队需要分配更多的空间，然后重启处理过程。根据临时空间失败发生的位置和时间不同，总是需要清洗数据。由于数据仓库设计用来进行查询，因此临时空间应该足够大。这种问题一般都发生在源系统环境中，因为那里 ETL 团队并没做有效的控制。

数据空间

在数据仓库中，数据应该与索引分开存储，以此减少冲突并减轻管理空间的负担。估计历史数据装载大小的一种有效的方法是只装载一小部分样本数据。装载一个分区表的一个简单分区是一个不错的基准。然后就可以把已装载数据所用的空间和表的分区数做乘积，就可以得到估计的大小。第 4 章阐述了提供关于估计数据空间的更多信息。

在大量装载之前，一定要保证 data-staging 数据库和数据仓库数据库拥有足够的空间。磁盘空间失败是一种严重的失败，需要手工进行数据清洗和重启装载过程。

索引空间

索引空间的估计是非常科学的一件事，因为索引并不像数据表那样成比例地增长。我们不会太详细讲解索引空间的问题，它的计算非常复杂，数据仓库架构、数据建模者和 DBA 应该在装载开始之前合理创建索引空间。通常分配的索引空间至少应该与基础表大小差不多。

当装载历史数据时，装载之前一定要把目标表的索引删除掉，并在装载完成后重建索引。通过删除和重建索引，不仅可以提高性能，还可以将索引空间不足的情况与装载失败隔离开来。一旦完成了表装载，总是可以为索引分配更多的空间，并在不影响数据表的情况下重建索引。

文本文件空间

由于数据库中为数据分配的空间可能不够，因此文件系统中的空间也必须进行监视以防止超过文本文件分配空间的限制。幸运的是，staging area 中需要的空间是按 ETL 团队的需求分配的，因此如果按照第 4 章中对文件系统需求估计的建议，对于处理历史数据装载将是安全的。一些功能较强的 ETL 工具包括检查点功能，可以保证过程中到达某一特点检查点的任何一条记录都被安全地写到磁盘中。但是，这些检查点文件写到文件系统，可能正好成为增加文件空间的罪魁祸首。检查点、缓存、哈希表、平面文件、临时空间或者任何其它 data －staging 文件都会填满磁盘空间。如果一旦由于任何一种文件超过了分配的空间造成过程崩溃，需要增加空空间，都建议从头开始重新处理，而不是试图去挽救已经处理的数据。

数据质量失败

生产环境中的数据质量失败是一种很容易被检测到的灾难性管理失败，例如字段丢失或者违反参考完整性，或者是在长时间运行过程中逐渐达到的数据质量告警的阈值。生产环境中数据质量失败应该是异常事件，往往需要专家参与。也许作业可以在已知异常数据的情况下完成运行，或者作业需要停止并修复源数据。

末声明的系统升级

也许末声明的系统升级唯一的好处是它造成的错误通常是彻底显而易见的。在这个过程中，ETL 作业停止了，一般在这种混乱状态下不会进行任何简单的修复。如果是这样，则必须要恢复到系统升级之前。这种情况对于很多其它的软件修改变化都是一样的。对于关键系统，在安装升级版本到生产环境 ETL 系统之前，必须要在测试系统环境中进行充分的升级测试。

恢复问题

当过程失败时，ETL 团队下意识的反应是试图挽救失败点之前所处理的内容。如果足够幸运并使用了具有检查点功能的 ETL 工具，则可以重新启动处理过程，并将失败时丢失的数据找回来。虽然有产品供应商的保证，我们还是对检查点技术的可靠性表示怀疑。如果处理在中间过程中失败，最好的方法是清除数据并从头开始处理。可以把处理过程分成若干个子过程来使恢复工作变得简单和有效，这样就可以不用重新处理整个历史数据装载，而只处理数据的某个子集就可以了。

最小化装载失败风险

以下是处理历史数据的几个基本规则：

- 分隔过程：使用日期或范围或自然键来把过程分隔成更小的可管理的单元。当失败发生时，只有这部分数据需要重新装载。
- 利用恢复点：为了安全，在每个主要的中间处理（例如整体抽取，关键的转换，或者分配了代理键等）之后将数据写入文本文件。
- 并行装载：不仅是数据装载，也包括 ETL 的每个组件都应该并行运行来减少处理数据所消耗的时间。
- 维护元数据：操作元数据（例如最后装载日期或者装载记录数）是检测失败发生时 ETL 每个部件状态的关键。

清除历史数据

当设计任何数据库应用系统时，都会创建一个矩阵表格来跟踪插入、修改、删除和选择数据的处理过程。这个矩阵表通常指的是 CRUD 矩阵表（创建、读取、修改和删除）。CRUD 矩阵表保证了每一个实体都有一个以上四种方式之一的数据操作过程。当开发应用软件时，矩阵表中的 D 是非常常见的，它的意思是这里的数据可以被修改和读取，但不能执行常规的处理来删除它。当没有开发常规的处理来清除历史数据时，通常会导致两件事情：会在系统上运行后台脚本来删除历史数据，或者记录不确定地留在系统中。可以想像，两种方案都不适合于数据仓库。

随着每个主题域计划完成，必须也有相应的清除处理过程。如果数据量相对很小，并且未来 10 年或更长时间的数据量不太会影响性能，则 ETL 并不需要马上开发。然而，清除策略元数据仍必须和初始化实施一起被收集和发布。

应该由 DBA 而不是 ETL 团队来完成数据仓库数据的存档工作。然而，对于从数据仓库中永久性地删除数据则必须由 ETL 团队来完成。关于所删除数据的业务规则必须由经过充分测试和质量保证的 ETL 过程来执行。

监控 ETL 系统

依赖于数据仓库的业务数据按照经过事先商定的频率（或持续不断的）来刷新。完成这个任务过程中的失败会导致数据仓库的可靠性和可信赖性被怀疑。因此，数据仓库如果没有有效的和持续的数据反馈就不能成功。ETL 团队必须监控和评估 ETL 作业来确保它们有效率地运行，以及数据仓库是否以有效的方式装载。ETL 监控使得处理过程的多个方面需要考虑。ETL 系统范围之外的资源如硬件和基础部件的管理和使用，以及源和目标环境，都是 ETL 系统的整体效率中的关键。在这里我们介绍几个 ETL 性能指标，来告诉你过程执行的好还是不好。这些指标是操作型元数据的一部分，应该存储在资料库中，以便被整个 ETL 团队随时分析。

度量 ETL 特定性能指标

对于那些负责系统或数据库管理的人来说，他们比较了解那些在环境中捕获的特定的度量来保证适当的性能。就像所期望的那样，ETL 系统也有自己的性能指标集。ETL 指标是专门用来描述实际数据的真实移动和管理情况的，它们是典型性能指标之下的指标，这里“之下”指的是不在操作系统一级或硬件资源一级来度量，而是在 ETL 处理本身过程中。

ETL 效率的度量大多数都用来表示处理数据所用的实际时间。请记住：ETL 系统的目标除了创建高质量的信息外，还有在分配的装载窗口内装载数据仓库。但是如果一个作业需要 20 分钟时间来完成，这是好还是不好呢？确实没办法知道，除非知道在这段时间里有多少条记录被处理。例如，如果在 20 分钟时间里处理了 5000 万条记录，则这是相当惊人的效率。而如果只处理了 100 条记录，显然效率就太差了。下面是 ETL 相关的一些度量，在研究装载性能方面非常有用。

- 耗时（秒）：这是其它计算的最直接的基础数据。耗时指的是 ETL 过程从开始到结束所用的时间（秒）。例如，如果一个过程从 4:00 a.m.开始并到 4:15 a.m.结束，则用时是 900 秒。
- 每秒处理的记录数：这相当于每秒装载的记录数，除非当源数据比目标数据要大的时候（聚合装载），这时就等于每秒读取的记录数。一个每秒记录数的例子是在 15 分钟时间里处理了 1000000 行记录，则为(1000000/(15*60))=1111.11 条记录/秒。
- 每秒读取的记录数：从源系统用 SQL 读取数据结果的记录数，再除以所用时间（秒）。数据再转给 ETL 流程中下游转换过程，这里根据处理过程的不同记录条数可能增加也可能减少。
- 每秒写入记录数：转换之后提交到目标表的记录数，再除以所用时间（秒）。对于有多个目标表的情况，指的是插入到所有表的记录数总合再除以所用时间（秒）。写入的记录数可能多于也可能少于读取的记录数。
- 吞吐量：吞吐量指每秒处理的记录数乘以每条记录的字节数。吞吐量和其它所有性能指标一起，可以用来作为提高性能的依据。

大多数主要的 ETL 工具都提高了必需的指标来度量 ETL 性能。你可以利用 ETL 系统来为消耗了比经验值更多或更少的时间的 ETL 作业触发告警。

在一个处理流程中，当一个处理部件的吞吐量不能处理前一个部件的输出时，就会产生瓶颈。例如，一个批装载可以按每秒 1000 条记录的速度向磁盘写数据，而磁盘只能写入每秒 800 条记录，因此在磁盘这个部件上就发生了吞吐量瓶颈。结果是，整个处理过程只能以最慢的部件的速度来处理。

度量基础性能指标

下一个要检查的部件是 data-staging area 架构。不同的监控软件包都提供了多个可用的处理度量，或者以手工编码 ETL 方式写入日志。只有少数供应商方案本身就提供了直接的影响 ETL 性能的指标。关于 ETL 性能重要的度量指标只要在过程运行的监控过程中才能获取。处理过程中提供的直接指示可能是瓶颈的度量包括：

- CPU 利用率
- 内存分析

- 服务冲突

实质上，网络流量和其它已知的基础部件性能指标可能影响 ETL 的性能。不幸的是，这些指标不是很稳定和具有一致性，因此可靠性较差。此外，网络流量本身很难确定或在测试环境中复现。如果你怀疑有网络问题，请联系网络管理员寻求帮助。

CPU 利用率

ETL 过程运行在其服务器上的中央处理单元（CPU）上。CPU 是处理器或芯片，实际中用于操作计算机所需的计算功能，使软件运转，以及完成 ETL 目标。大多数 ETL 服务器包含多个处理器来处理抽取、转换和装载数据到数据仓库所需的大量计算功能。在 ETL 工具中一般是不能得到 CPU 利用率报告的，因为这超出了 ETL 系统的范围。然而，如果在 UNIX 上运行处理过程，可以使用 SAR －u 命令来系统中每个处理器的使用列表。

在基于 Windows 的操作系统中，可以使用图形化界面性能监视器，在其中可以添加新的计数器到目前可用的性能日志中。在 Windows XP 中，在控制面板的管理工具中可以找到性能监视器。为了添加新的计数器，打开性能监视器，右键点击系统监视细节面板，然后点击添加计数器，在这里可以选择处理器作为性能对象，然后再选择相应的计数器。性能监视器将创建一个日志文件来捕获关于 CPU 利用率的统计以便进行分析。

如果发现在 ETL 过程中 CPU 经常达到能力上限，则需要添加处理器。这些 CPU 监视工具还帮助检查是否 ETL 过程在所有可用的处理器上平均分配负载。

内存分配

RAM 可以在多种不同的位置分配给 ETL 过程，但总的来说，服务器上的内存必须物理可用才行。如果购买了 ETL 工具，供应商应该提供硬件配置说明和工具建议和处理现有装载数据量所需的 RAM 数。一旦在服务器上安装了 RAM，内存必须分配给处理过程。对于大多数 ETL 工具，内存使用可以在作业级或者批处理级设置。

为了实现最好的效率，对每一 ETL 过程必须分析适当的内存。如果 ETL 过程不断地读和写磁盘，而不是在内存处理数据，则这种方案将会非常低效。ETL 过程分配的适合的内存比其它设置更能影响转换性能，因此一定要确保设置是正确的。

ETL 工具可以告诉你每个过程分配了多少内存，过程实际使用了多少，以及使用了多少虚拟内存（磁盘缓存），在 POC 测试过程中要确保由 ETL 工具提供了这些操作元数据。如果应该在 RAM 中的大量数据正在写入磁盘，你应该为过程分配更多的内存或者给你的服务器添加更多的物理内存。

有些 ETL 工具使得内存管理对开发团队完全透明，而其它的则可能需要手工配置特定项。主流 ETL 工具中的内存设置包括：

- 共享内存：当 ETL 引擎读和写数据时，它们使用了内存中一块专用区域叫做共享内存。共享内存是数据在进入或退出 ETL 过程的物理转换步骤排队等待的位置。如果作业没有足够的共享内存，就会产生很多的磁盘缓存。相反地，如果分配了过多的共享内存，则会保留大量不必要的 RAM，其它过程也不能使用。ETL 供应商应该根据处理的数据量大小提供如何计算最优的共享内存设置的指导。有些 ETL 引擎可能会试图动态管理共享内存，寻找一种工具使你可以针对特定情况不考虑系统分配的设置，这种情况下引擎也许不能进行估算。
- 缓存块大小：缓存块设置是分配性能相关设置时需要考虑的关键因素。缓存块大小适合设置依赖于转换流程中数据的行大小。如果工具需要或者允许自定义调整缓存块大小，ETL 供应商可以提供优化设置的计算建议。

当程序需要比物理可用内存更多的内存时，操作系统（或应用系统）把不能放到内存的数据写到磁盘。由于需要处理大量的数据，程序必须能够读/写磁盘而不是使用 RAM。虚拟内存通常是指页面交换，因为内存是按页面存储的，当需要更多的页面时，就会在磁盘和 RAM 间交换。页面交换是性能杀手，应该在 ETL 过程中尽量避免。如果检测到连续不断的页面交换，就应该为 ETL 服务器和 ETL 过程添加更多的 RAM 了。

如果选择手工编码方式实现 ETL，则可以用 vmstat 命令行手工进行内存使用监视。Vmstat 命令报告了虚拟和实际内存使用情况，以及页面活动和磁盘操作情况。

服务器争夺

另一个潜在的性能杀手是服务器争夺。当多个处理过程试图使用同一资源时就发生了冲突。可能会遇到内存争夺、磁盘访问争夺或者数据访问争夺。最常见的冲突是两个 ETL 过程试图访问同一数据。ETL 过程可能会引起死锁。当过程 A 试图把过程 B 关在外面，而过程 B 也试图把过程 A 关在外面时就发生了死锁，系统宕机。通常，DBMS 在管理数据访问争夺方面做的很好，但在 ETL 开发者开发过程中的某些点上还是会发生数据访问争夺。当 ETL 过程并发而不是并行运行时，ETL 系统非常容易发生服务争夺。当这种争夺发生时，ETL 过程不断地争抢资源，从而产生冲突。最好的防卫方式是避免并发过程，除非每一过程都有专门的过程流并有指定的数据区。

内存争夺

当多个应用或过程在同一服务器上运行时，它们都需要物理 RAM 来进行操作。不幸的是，RAM 是有限的资源，每个过程一定会争夺它。如果并发执行 ETL 过程，则可能会产生内存争夺问题。当并发执行过程时，作业级的内存分配设置就成为关键。每一 ETL 产品为了减少内存争夺都有其自己的建议。基本规则是，为小作业分配最小的内存，把空间留给大作业，例如第 2 类缓慢变化维、桥接表或者事实表。ETL 工具应该能在其工具内部管理内存来避免内存争夺。任何情况下，工具应该提供操作型元数据来展示内存争夺情况。如果工具不能实现，UNIX 的 SAR 命令可以用来帮助检测内存争夺情况。SAR 命令对于检测那些和 ETL 工具一起运行并竞争同一 RAM 的过程的内存使用情况非常有用。如果可能的话（预算允许），一定要确保 ETL 引擎是数据仓库装载过程中服务器上唯一运行的进程。如果 ETL 进程没有在规定的装载窗口内完成，则其效率就值得怀疑了。有效的监视经常可以发现大多数装载延迟并不是 ETL 无效率的结果，而是由于外部进程与 ETL 运行在同一时间，并且一起竞争服务器资源。

磁盘争夺

大多数磁盘都有在给定时间内读写的数据量和访问连接数的限制。当达到限制值时，ETL 过程将排队等待对磁盘的访问。如果在同一磁盘的同一数据文件进行多个表的数据并发装载，则会发生忙点现象。忙点指的是在磁盘上某个区域重复读写访问。如果使用 Oracle，可以使用下面的 SQL 来检测源数据库、staging 数据库或目标数据库的忙点：

```
select d.name datafile_name, f.phyrds reads_count, f.phywrts

writes_count

from v$datafile d, v$filestat f

where f.file# = d.file#
```

```
order by greatest(f.phyrds, f.phywrts) desc
```

该查询按照读和写的数目降序排列查询的结果，读写最多的数据文件排在最上面。如果有一些数据文件与其它部分不成比例，则需要重新配置 staging 表的物理属性，以便分布更均匀一些。

磁盘争夺也发生在数据库外部。ETL 引擎使用临时区域和隐含创建的文件来保持过渡过程数据。此外，开发者也公开的在文件系统创建 staging 表、配置和参数文件。如果要从操作系统角度获取磁盘活动信息，可以使用 UNIX 命令行 IOSTAT。IOSTAT 命令列出了每一磁盘及其相关的信息：

- 磁盘名
- 每秒读次数
- 每秒写次数
- 每秒读字节数（KB）
- 每秒写字节数（KB）
- 服务的交易等待平均数（队列长度）
- 正在服务的活动交易平均数（从队列中删除但还没有完成）
- 平均服务时间（毫秒）
- 有等待服务的交易的时间百分比（非空队列）
- 磁盘忙时百分比（处理交易）

关于如何解决磁盘争夺的信息在本章后面提供。

数据库争夺

如果 ETL 过程试图在同一时间更新同一表的记录，就会经常发生数据库争夺问题。本来，管理数据库争夺是 DBMS 的作业，但是当进程争夺同一资源时就会发生死锁，使得进程永久等待下去。请参考专门的 DBMS 参考手册或者联系本地 DBA 获得检测数据库争夺的最佳步骤。

处理器争夺

有时，如果硬件没有配置成并行运行，当试图在软件一级并行处理时就会发生问题。当有多个进程（超过可用的进程数）试图在同一时间运行时，就会使 CPU 满负荷并引起严重的性能问题。可以使用 UNIX 的 SAR 命令或者 Windows 的 PerfMon 命令来获取 CPU 使用率的统计。

度量数据仓库利用率来帮助管理 ETL 过程

在本章前面提到的供应链例子中，你会注意我们确定了四个关键部件，用来帮助把原始数据转换成为用户使用的有用的格式的结果。至此，我们已经描述了如何在 ETL 系统范围内以及 ETL 环境的硬件和基础架构内监控活动。现在我们来看一看数据仓库展示层的重要度量指标。

这里的度量并不直接影响 ETL 系统，但是捕获和分析它们仍很重要，因为它们对装载过程有影响。我们希望后面列表中的度量已经被数据仓库团队收集好，用来帮助管理他们的用户经验以及删剪掉数据仓库中存储的不再使用的数据。

ETL 团队应该充分利用数据仓库使用报告，从中寻找重新安排装载调度、修改装载频率或者清除装载睡眠表的维护作业等方面的机会。例如，如果一个表只在一个月的第一天被

访问，它就不应该天天更新。另一个有功效的方法是分析索引使用率。ETL 过程的很大部分工作是在每次装载后重建数据仓库中的索引。如果使用率分析结果显示有些索引从不使用，则它的重建过程就应该从 ETL 流程中删除。支持 ETL 作业管理的利用率度量包括：

- 表利用率：表利用率报告的内容可能不尽相同，但是一个有用的报告应该包括以下内容：数据表列表、第一次和最近访问数据表的时间、和该数据表相关的查询数以及查询该表的不同用户数。一个月才使用一次的表应该从按天装载进程中删除并转向按月频率装载进程中。对于除了刷新以外不断被利用的表，应该使用一些高可用性技术。高使用率表一般在后台都装载了一个复本，一旦装载完成，两个相同结构的表就改变名称。该技术使得在数据刷新过程使表仍在线可用。
- 索引利用率：索引是数据仓库表性能提高的关键，但会给 ETL 带来负担。因为很多情况下，每一次数据装载都会删除并重建索引。当数据仓库架构为展现层创建维度结构时，一般都倾向于尽可能多的对列建索引，以防止用户较低的执行效率。但事实情况是很多索引列从没有约束关系，索引也从未被使用。索引利用率报告能发现那些睡眠索引，数据仓库团队可以合理地处理它们。
- 聚合表利用率：聚合表通常和索引在同一时期创建，当需要时就创建它。但和索引的情况一下，有些聚合表创建了但从没被使用过。或者随着时间的迁移，它们越来越没有用处，最终进入睡眠状态。聚合表利用率报告可以发现那些不再有用而应该被删除的表。
- 睡眠数据：睡眠数据报告总是会受关注，因为数据仓库是按照用户交流的结果创建的，这些交流用来找到那些执行完成作业所需的分析的数据元素。然而由 ETL 每天刷新的表处于无用状态是不可避免的。即使表有用，有些列也可能从没被选择使用过。我们总是很关心事实表列利用率，因为它对于发现最复杂的导出度量的利用率（由于它们的定义没有满足用户的需求）非常有用。数据睡眠报告可以帮助 ETL 团队确定那些从未使用的度量和维度属性。

有多种方法收集数据仓库利用率的统计值。有些数据库管理系统本身提供使用率信息。然而，一定要测试打开和关闭使用率报告功能对性能的影响，它可能会影响查询响应时间或者 ETL 装载时间。一种不影响性能的跟踪利用率统计的方法是使用像 Teleran Technologies（www.teleran.com）这样的中间件。这些数据仓库监控工具在数据库之外的网络包一级捕获 SQL 语句和数据。我们确信还有其它工具提供数据库利用率统计，可以在 www.google.com 中试着用 data warehouse usage tracking 来查找这一领域的供应商列表。

ETL 过程调优

为了更好的理解如何优化 ETL 进程，必须要熟悉数据库是如何工作的。大多数数据库管理系统可用的功能不可以在数据仓库环境中使用，而有些几乎从不在交易系统中使用的特性在数据仓库中不仅有用，而且是功能首选。很多设计决策是基于开发的进程所迁移的数据量。例如，如果有一个非常小且数据稳定的维表，则使用 update 语句增量更新维表现有数据是可行的。但是如果维表有 2000 万行记录之大，且数据变化频繁，则删除并批量装载该表也许更有效一些。可以使用数据仓库架构师或项目经理创建的报告来了解初始化阶段有多少数据将被装载，以及实施 6 个月之后计划的增长情况，从而计划可能的容量，以及确定 ETL 系统的扩展性，并在之后一直跟踪实际的增长情况。

后面部分重点介绍在可控的 ETL 环境中不需要而应该避免的功能，并提供实施过程中更快且同样效果的解决方案。

了解数据库

在创建关系型数据库之前，数据存储在平面文件中。这些文件往往数据质量较差，主要表现在重复的分组和元素，没有主键，以及表间没有强制关系等等。整个数据库中的一切都是重复的，此外，在数据库一级不存在数据验证。当时，数据库只是一个没有关联的文件集合。简单的说，当时的情况简直一团糟。

在 1970 年，E.F.Codd 发明了关系代数，这成为关系型数据库系统的设计基础。关系型数据库提供了以下功能：强制参考完整性、数据唯一、主键、约束检查、外键等等。结果是数据清晰可靠，但是降低了操作速度。关系型数据库的每一特性对于交易处理都意义重大，不再需要程序额外的处理来执行后台错误检查和控制。在本节，我们讨论作为 ETL 团队中一员经常会遇到的不同的数据库特性，并提供克服数据库问题的建议。但是，这并不是什么 DBA 培训。事实上，本节的内容并不能使你成为数据库管理员。本章的目标是帮助 ETL 团队以及 DBA 来了解优化 ETL 过程所需要的特定考虑。优化 ETL 相关的大多数工作都在数据库之外。本章的内容用来深入了解如何让数据库处理大量的数据，以及提供关于提高处理速度的建议和技巧。

插入、更新、删除

数据操作语言（DML）有四个主要的动作：select, insert, update 和 delete。四个 DML 操作都可以用来在不同的数据库操作数据。切记，DBMS 主要设计用来避免交易失败。因此，作为预防措施，理论上每种 DBMS 都维护一个回滚日志。回滚日志记录了 DML 操作并提供了 DML 提交后结果的取消机制。对于中间交易失败的情况，DBMS 自动回滚完成了一半的交易结果，使数据回到交易开始前的状态。

日志的作用

每一种类型的 DML 都会以不同方式影响回滚日志。Select 语句不会写入日志，因为它们不会改变现有数据。在大多数数据库中，insert 语句都会在随机输入数据或者交易中间失败时写入日志，DBMS 可以简单的回滚记录，而不是删除或清除它。Udpate 和 delete 都需要写入回滚日志。Delete 需要较大的成本，因为它要在删除发生之前保存旧记录，update 需要所有 DML 语句中最大的成本，因此处理起来非常慢。

索引的作用

使用数据仓库是因为它比交易系统更快更可靠。数据仓库提供的速度优势是由于以下特性带来的：

- 维度模型有意地建立维表和事实表索引
- 良好的索引策略
- 物理存储聚合记录
- 并行查询机制

设计技巧和维度数据模型的优势在整本书都有涉及。在本节，我们来讨论一下索引。

索引是数据仓库中查询响应时间的基石。数据库中每一次查询都至少要使用一个索引。不幸的是，由于索引帮助数据仓库用户查询，因此 ETL 团队就有很多工作量是在 ETL 进程中管理现有索引。索引管理是大多数 ETL 进程中的必要部分。

在进入数据仓库装载阶段索引管理的技巧之前，首先回顾一下大多数数据库可用的不同的索引类型。一般有两种主要类型的索引，理解这两种类型之间的区别非常重要：

- B 树索引：B 树或者平衡树索引在反向的树结构中保存键值以及指针。B 树索引对于高 cardinality 的列比较理想，其中高 cardinality 指的是不同值的数量。反向树结构利用一种非常高效的分隔筛选技术来找到某个特定的值（或值范围）。B 树索引对于常规的查询非常有用，但是对于像数据仓库这样的 ad-hoc 环境支持起来就不太灵活了。B 树索引不灵活是由于你不能临时动态地组合索引列来创建复合索引，以此来解决新的、不可预期的查询。所有索引必须在之前创建同，因此 DBA 必须要猜测哪些列将包含在查询中。此外，列放置的顺序位置也决定了它们能否被使用。结果是 DBA 团队必须创建很多很多的复合 B 树索引以及很多不同顺序的同一批列的组合。
- 位图索引：位图索引与 B 树索引功能完全不同。位图索引更适于低 cardinality 的列。多个单列位图索引可以动态关联在一起，以此创建所需的组合索引，从而支持 ad-hoc 查询。由于它们的灵活性，在数据仓库中事实表上每一个代理键上创建单列位图索引就是很通用的方法。

现在我们回头看看索引如何影响 DML 操作。B 树索引的每一个条目都包含了一个且仅有一个行号，它指向回到基础表的相应记录。相反，位图索引在索引中为每个值包括了一个行号的范围。如果值被修改，则该行号范围对应的每一条记录就被锁住。最终，每当记录修改时，为了管理所有被锁住的记录，就给了数据库相当大的压力。不幸的是，事实表通常都有很多位图索引，大量的数据更新会带来非常大的性能下降。

因此，我们建议在对事实表进行数据操作前删除所有的位图索引。另外最好把事实表分区，这样就可以只删除当前分区的本地索引（假设表是按照日期键分区的）。B 树索引在 ETL 过程执行前也需要删除。统计显示，在大多数情况下，删除数据仓库的索引后，装载表并创建索引比带着索引装载表要快得多。

约束关系和外键

关系型 DBMS 中的外键强制了表间数据的完整性。例如，只有在定单状态表中存在有效值的情况下才可以输入定单状态。但是在数据仓库中，交易已经发生并且已经在源系统中验证过，因此所有的外键和约束关系都应该禁止掉，尤其是在 ETL 过程中。对于 DBA 团队来说，删除外键可能是很大的挑战。因此必须在整个 ETL 过程中与 DBA 团队一起工作，并且要解释清楚，事实表记录如果没有来自相关维表的代理键就不能简单存在。而且，必须指出的是，维表的原始键用来确保记录不会重复插入。一旦 DBA 团队明白了 ETL 是真正可控制可管理的环境，他们就会意识到数据库约束和外键是多余的，这样只会降低 ETL 处理的效率，而不会带来任何好处。

把数据装载到数据仓库最快的方式是用数据库的批量装载，下面四个步骤用于数据库准备：

- 尽可能消除 DML 语句
- 禁止回滚日志功能
- 删除现有索引
- 消除数据库外键和约束

一旦完成了上述四个步骤，就做好了使用数据库批量装载工具的准备。在第 7 章里有关于使用批量装载的向导。

ETL 系统安全

在数据集结区中的数据库安全比在数据仓库展现数据库里要容易实现得多。一般来说，

除了 ETL 引擎和程序，没人能够读写 data-staging area。大多数数据库使用角色和用户在数据库一级强制安全性。角色机制是这样的，它使安全管理员可以把许多用户组成一组，而这组用户拥有相同的数据库访问权限。每一用户都有其自己的用户 ID 和验证机制。当用户通过系统验证后，他就被授予其相关的角色。

如果没有角色，安全管理员就必须对每个用户分别授予适当的权限。用户是使用数据库的个体。

通常，需要创建一个具有下列权限的数据仓库管理角色：

- Select, Insert, Update 和 Delete 所有对象的权限
- TRUNCATE TABLE 权限
- 使用批装载权限
- 删除和创建索引权限

对于高度敏感数据，例如赔偿率或销售潜在客户，这些数据应该在 ETL 团队抽取之前由源系统加密。通常，列一级的安全通过使用数据库视图来控制。视图是基于表之上的对象，可以隐藏敏感列数据。然而，ETL 团队必须能够从源系统选择数据，以及能够在数据仓库上执行任意 DML 语句。所以，视图就不是隐藏敏感数据的有效机制了。此外，ETL 团队不应该负责为敏感数据加密，这个工作应该是源系统安全管理员的责任。

开发环境安全

在开发环境中，ETL 团队中每个人都授予了 DWETL 角色权限（所有对象的 DML 和 TRUNCATE 权限），也正是在这里创建所有的 staging 表。虽然 data-staging area 被 ETL 团队拥有，但有时表创建权限却被数据仓库架构师或在 BA 控制。有些时候，ETL 架构师在没有授权情况下也具有在 data-staging area 创建表的权限。

此外，任何 ETL 团队成员可以添加、删除或修改文件系统中指定目录中的文件。ETL 团队之外没有人具有访问 data-staging 环境的权限。

生产环境安全

ETL 团队通常只具有生产环境的读权限。有时，在安全度要求非常高的环境中，例如银行，可能就根本没有访问权限。生产环境存在 DWETL 角色，但是只有 ETL 引擎使用的用户 ID 和口令在生产环境中可以被创建。如果为了某种非常特殊的原因，ETL 团队成员必须要有生产环境的写权限，例如修改一个只有在生产环境中存在的错误，则应该只临时授予权限，在修复完成后马上回收权限。

FTP 的不足

必须把所有人挡在 FTP 之外，只有 FTP 进程可以在指定目录中进行写入或删除操作。此外，只有 ETL 引擎可以读这个目录。如果处在某种紧急情况下，则应该临时授予权限给预先确定的管理员，并在问题解决后马上回收。

加密/解密

最大的性能障碍是磁盘的读写（I/O）。然而，为了安全必须对流转的数据加密。通常，流转的数据安全包括下面几个步骤：

- 加密数据并存储在磁盘上

- 读取并在网络上传输加密数据
- 存储加密数据在 data-staging 服务器上
- 解密数据并存储在磁盘上
- 转换并加载解密数据到数据仓库

也可以把上面步骤减少为 3 步，当源数据读入内存时进行加密，然后传输加密数据，最后在进入转换和装载进程时解密数据。可以看到，第二种方案从数据读取直到装载入数据仓库都没有涉及磁盘。这种过程叫做流中解密。有些 ETL 工具支持流中解密功能。如果需要在流过程中加密或解密来提高性能，一定要选择支持该功能的工具；否则你就需要在 Java 中写自己的小程序了。对于 Java 小程序的情况，请确保 ETL 工具至少可以嵌入外部进程（例如流中解密小程序），从而不需要过多的磁盘 I/O。

短期归档和恢复

有多个原因需要在 ETL 系统中保存多个 data-staging 结果。在本书中，我们确定了需要短期重启容量，当没有合适变更数据捕获系统时把按天抽取数据进行对比来检测差别，以及法律和财务审计方面需求。所有这些归档和恢复情景都应该是 IT 人员熟悉的挑战。备份数据到当前介质并确保它能恢复，确保拥有可信赖的审计轨迹来标明所有对数据的访问和修改，确保数据物理上是安全的，并按保护在线数据的方式保护存档数据。

但是如果数据要保存多年该是什么样子呢？

长期归档和恢复

数据仓库管理员的重要目标之一是保存历史数据。我们以多种方式保存企业历史信息，成为公司的档案保管人。我们并不经常许诺使所有历史数据保持在线可用，但我们经常声明为了安全要将其保存在某处。当然，为了安全保存它意味着当有人有兴趣查看这些数据时能够再次恢复这些数据。

大多数数据仓库管理员都忙于以下事情：建设数据仓库，避免烟道型数据集市，配合新的数据库技术，以及满足对 Web 的大量需求。这样我们的任务就变成了存档，把数据备份到磁带上，然后就忘记了这些磁带的存在。或者也许我们还把数据追加到最初的事实表上，但我们并没有真正想过这些旧数据能做什么。

但是计算机业界有一种普遍的观点是，保存数字信息并不算完，这就带来了一个严重和难以解决的问题：

数据仓库还需要保持旧数据吗？

大多数数据仓库管理员都是被市场这样的部门的紧急需求驱动的，都只是为了战术的考虑。很少有市场部门会关心超过 3 年以上的旧数据，因为我们的产品和市场变化的如此之快。因此，就只需思考市场客户，而把不再满足需求的数据放弃掉。

但是稍微思考一下我们就会意识到，我们仍然需要大量的数据仓库中的一些数据，这些数据绝对必须要保存。这些数据包括：

- 针对法律、财务和税收而用的详细销售记录
- 长期跟踪具有战略意义的趋势调查数据
- 政府调整或与跟踪相一致所需的所有记录
- 必须保存 100 年以上的医疗记录
- 支持专利声明的临床测试和试验结果

- 有毒废弃物处理文档、燃料传输和安全检查
- 对任何人、任何时候可能拥有历史值的所有其它数据

面对这个列表，我们必须承认需要一个计划来获取所有这些类型的 5 年、10 年甚至 50 年的数据。我们都清楚这可能将是一个很大的挑战。磁带以什么方式持续多久来保存？CD-ROM 或 DVD 能解决问题吗？将来还能读取这些格式吗？几年前我们还有一些 8 寸软盘，但现在绝对不可恢复也没有用了。突然，这听起来像是一个非常艰难的项目。

介质、格式、软件和硬件

当我们开始真正考虑长期保存数字数据时，我们发现出了大麻烦了。先让我们从存储介质开始。关于物理介质实际的寿命有很多的不同意见，像磁带和 CD-ROM 光盘，寿命估计在 5 年到数十年之间。但是，当然，我们的介质可能没有档案质量，它们可能没有以一种最优的方式存储和处理。我们必须平衡供应商的乐观许诺与相关专家的实际评价之间的差异，专家认为今天我们拥有的大多数磁带或物理介质上的数据在超过 10 年以后都会面临可用性问题。然而，所有关于物理介质保存期的争论与关于格式、软件和硬件的争论比较起来显得就不足为重了。所有的数据对象在物理介质上都是按现在的格式来编码。所有的一切，从介质的密度，到目录的编排，再到最终更高级别的特定应用的数据编码，都会像堆叠起来的纸牌一样最终倒掉。拿 8 寸软盘举例，靠什么来读取嵌入的数据呢？可能需要支持 8 寸软盘驱动器的硬件配置，还需要 8 寸驱动器的软件驱动程序，以及最初写数据到文件的应用程序。

过时的和废弃的格式

在数字保护主义者的词典里，过时的格式就不再支持了，但是仍有现存的硬件和软件可以读取和显示这些原始格式的数据内容。陈旧的格式传递给下层领域。我们关心的 8 寸软盘是一种陈旧的格式，再也不会恢复它上面的数据。以 A 类线形文字著名的腓尼基人文字系统也是一种陈旧的格式，很明显它永远地消失了。我们的软盘或许要比 A 类线形文字的解晰要简单一些。

硬拷贝、标准和博物馆

针对恢复不同格式的旧数据这项困难工作，我们设计了多种方案。一种简单的方案是通过硬拷贝还原一切，也就是说，把所有数据打印到纸上。当然，这样做将屏蔽所有数据格式、软件和硬件的问题。如果只有非常少量的数据，这是一种不错的选择，总比丢失数据要好多了，但也存在多个致命的缺点。对于现在来说，拷贝到纸上并没有可扩展性。打印一个 GB 的 ASCII 字符，如果每页打印 4000 个字符，则需要 250000 页，一 TB 数据将需要 250000000 页！请记住我们不能采取欺骗手段，将纸张放到 CD-ROM 或磁带上，因为这将再次引入数字格式问题。最后，我们也许将认真折衷考虑数据结构、用户界面和系统最初认为的行为来呈现和解释数据。大多数情况下，纸质备份将破坏数据的可用性。

第二种方案是建立数据展现和存储的标准，这将保证一切数据都能以永远可读的格式来表示。在数据仓库中，接近这一标准的唯一数据是以 ANSI 标准格式存储的关系数据。但是绝大多数关系型数据库的实现都对数据类型、SQL 语法和相关元数据做了很大的扩展，以此来提供所需的功能。当我们把带有应用系统和元数据的整个数据库卸载到磁带上时，不管

它是来自 Oracle 还是 DB2，我们并不能确信 30 年或 50 年之后仍能使用这些数据。狭义 ANSI 标准 RDBMS 定义之外的其它数据更是纷繁多样，市场上还没有任何一种机制，能够把所有可能的 OLAP 数据存储机制集成到单一的物理标准，以此来保证到标准格式间的无损转换。

最后一种经典的方案是支持“博物馆”，这里保留了所有硬件、操作系统和应用软件的旧版本，这样旧数据就可以被读取了。该方案至少理解了问题的核心，为了解释旧数据就必须真正拥有旧软件。但是博物馆思想也不具有扩展性，也不能深入解决问题。我们如何让 Digital Data Whack 9000 工作 50 年？当最后一个人死了时会怎样？当一个使用旧数据的人迁移数据到流行的 DVD ROM 介质时，Digital Data Whack 9000 界面如何面对 DVD？有人会为过时的旧机器写新的驱动吗？也许它还是个 8 位总线。

刷新、迁移、模拟和封装

有些专家建议说，IT 机构应该周期性地从旧介质物理移动数据到新介质，以此刷新数据的存储。关于刷新的更进一步是迁移，这里数据不仅是物理转换，而且还重新格式化，以便被现有的应用系统读取。刷新和迁移确实解决了一些短期保存危机问题，因为如果你成功完成了刷新和迁移，就可以从旧介质和旧格式的问题中解脱出来。但是从长远来看，这种方法至少有两个严重的问题。首先，迁移是需要消耗大量人力，自定义工作对于作业几乎没有任何杠杆作用，反而可能会带来原始功能的损失。此外，更严重的是，迁移不能处理主要的机制转换。我们都希望从版本 8 的 RDBMS 迁移到版本 9，但是当异构数据库系统（HDS）占据一切时会怎样？没有人（包括我们）知道什么是 HDS 这一事实就表达了我们的观点。毕竟，当从网络数据库到关系型数据库发生机制转换时，我们不能够迁移很多数据库，是这样吧？

上面我们描述了很多的困难，如果这样，专家们还有什么希望长期保存数字信息呢？如果你对一主题非常感兴趣，并想在 50 年后仍能访问存储在数字数据仓库中的数据，请阅读 Jeff Rothenberg 的论文 Avoiding Technological Quicksand, Finding a Viable Technical Foundation for Digital Preservation，这是 Council on Library and Information Resources (CLIR) 的报告。这个 41 页的报告可以在 www.clir.org/pubs/reports/rothenberg 的链接上获得 PDF 版本文件。该报告写的非常好，有较高的推荐率。

Jeff 对于这一问题，主要思想是建议开发一个模拟系统，虽然它是运行在现代硬件和软件上，但却完全模拟旧硬件。模拟选择在硬件级是因为硬件模拟被证明是重新创建旧系统的技术。他还描述了把旧数据集与解释该数据集所需的元数据封装在一起的必要性，以及针对模拟本身的所有规范。通过把所有内容保留在一个封装包里，数据就可以历经岁月，50 年后仍可以根据需要恢复一切内容。你需要做的是说明现有硬件上的模拟规范。图书馆已经完全解决了数字保存问题。在 Google 的搜索引擎里查找 embrittled documents，需要学习它们的技术并适应我们的仓库需求。

总结

在本章中，我们提供了典型 ETL 系统的操作框架整体介绍。本章前半部分重点介绍主要的调度方法。后半部分细述了随着系统成长和变得日益复杂的工作而带来的管理性能问题。最后，还为 ETL 系统安全性提出了一个简单框架。

北京易事通慧科技有限公司（简称“易事科技”，ETH）是国内领先的专注于商业智能领域的技术服务公司。凭借着多年来在商业智能领域与国内高端客户的持续合作，易事科技在商务智能与数据挖掘咨询服务、数据仓库及商业智能系统实施、分析型客户关系管理、人力资源分析、财务决策支持等多个专业方向积累了居于国内领先的专业经验和技能。

作为Solvento集团旗下的联盟公司，易事科技获得授权为客户和合作伙伴提供MicroStrategy产品，SPSS产品，Pervasive产品和i2产品的销售及技术服务。更多的信息请访问公司的官方网址 http://www.ETHTech.com，或拨打电话+8610 68008008。