第二部分　数据流

抽取

一旦开始了数据仓库项目，你就会很快意识到，集成整个企业中所有不同的系统来得到数据仓库可用的状态是一个真正的挑战。没有数据，数据仓库就是无用的。集成的第一步就是成功地从主要的源系统抽取数据。

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

本书的其它章节主要集中在转换和加载数据到数据仓库，本章的重点是为项目建立访问需要的源系统的接口。每一数据源都有不同的特点需要分别管理，以便为 ETL 过程有效的抽取数据。

随着企业的发展，需要建设或者继承多个不同的计算机系统来帮助公司运作他们的业务：销售系统、库存管理、生产控制、总帐系统－－这个列表还在不断增长。更糟糕的是，不但这些系统是分离的且产生于不同的时间，而且他们经常是在逻辑上和物理上不兼容。ETL 过程需要有效集成不同的系统：

- 数据管理系统
- 操作系统
- 硬件
- 通信协议

在开始创建抽取系统之前，需要一份逻辑数据映射，它描述了那些提交到前台的表中原始源字段和最终目标字段之间的关系。该文档将贯穿 ETL 系统的始终。我们将在本章第一部分介绍如何创建逻辑数据映射。本章第 2 部分列举了可能经常遇到的各种源系统。我们将深入研究每一种源系统，以便选择正确的处理方法。

本章结束部分介绍了变化数据和删除数据的捕获这一主题，15 年前我们还认为数据仓库是永恒不变的：它是一个一次性写成的数据大库。随着这些年的丰富经验的积累，我们现在知道数据仓库经常需要修改、纠错和更新。本章的变化数据捕获抽取技术只是这一复杂问题解决的第一步，在后续的提交和操作章节中，仍将继续这一变化数据捕获主题。

第 1 部分：逻辑数据映射

在物理实施之前如果没有仔细地规划结构将会是一场灾难。正如任何其它形式的建筑物一样，在敲击第一个钉子前必须拥有一个蓝图。在开始开发一个单一的 ETL 处理前，确保你拥有合适的文档，以便该处理与已有的 ETL 策略、过程和标准在逻辑上和物理上保持一致。

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

逻辑数据映射描述了 ETL 系统中起点和终点之间的关系。

物理之前设计逻辑

直接跳到物理数据映射会浪费宝贵的时间，并将无法文档跟踪。这一节描述如何开发逻辑的 ETL 过程并使用它来制订出物理 ETL 执行流程。在开始任何物理 ETL 开发之前，确定以下的步骤已经完成：

1. 有一个规划。这个 ETL 过程必须用逻辑的和文档化的形式表示出来。逻辑数据映射由数据仓库架构师提供，并且它是为 ETL 团队创建物理 ETL 作业而制定的。该文档有时会作为数据流报告。逻辑数据映射是元数据的基础，随后将提交给测试员来保证数据质量，并且最终将提交给最终用户来详细描述在源系统和数据仓库之间到底做了些什么。

2. 确定候选的数据源。从最高级别的业务对象出发，确定你认为将支持业务人员进行决策的可能的候选数据源。并且确定这些源中你认为对最终用户的数据很重要的特定的数据元素，这些数据元素将成为接下来的数据评估步骤的输入。

3. 使用数据评估工具分析源系统。源系统中的数据必须在数据质量，完整性和适合使用方面进行仔细检查。根据于你的组织结构，数据质量可能是或者不是由 ETL 小组负责，但是这个数据评估步骤必须由对最终使用数据仓库的决策者的需求有一定了解的人来负责。每一个源系统的数据都必须进行分析。任何监测到的异常数据都必须用文档记录下来，对任何进入数据仓库的数据都必须按照适当的业务规则进行修正是最好的选择。你必须要一直关注项目在这个阶段就停下来的可能性！如果数据不能支持业务目标，这个时候就要发现。在第四章中将更多的探讨数据评估。

4. 接收数据线和业务规则的遍历。一旦源数据完成数据评估步骤的质量保证并且理解了最终的目标数据模型，数据仓库架构工程师和业务分析师必须和 ETL 架构工程师及开发者一起完成为抽取，转换和加载的数据仓库的主题域的整个数据线和业务规则，并且最好他们理解这些规则。完全的理解这些数据线和业务规则是几乎不可能的，因为那样 ETL 小组必须遇到所有的数据实际问题，但这一步骤的目标是提交尽可能多的知识给 ETL 小组。数据评估步骤必须创建 ETL 特有的业务规则的两个子类：

4a. 在数据清洗步骤中需要进行改造的数据；

4b. 对分离的数据源的维度实体和可度量的数字事实强制一致性来获得标准的结构；

5. 充分理解数据仓库数据模型。ETL 小组必须充分理解数据仓库的物理数据模型。这种理解包括维度模型的概念，仅仅理解基本的表到表之间的映射是不够的，开发小组必须对如何使维度，事实及其他维度模型中的特定表一起发挥作用来实施一个成功的 ETL 解决方案有好的理解。切记 ETL 系统的主要目标是用最有效的方式将数据送给最终用户工具。

6. 验证计算和公式的有效性。与最终用户一起校验任何在数据链中任何指定的计算。这个规则来源于纽约市建筑行业的谚语“测两次，切一次”。正如你不想得到一座用错误尺寸的材料造成的摩天大楼，你同样不希望在数据仓库中部署不正确的度量指标。在 ETL 过程中花时间以错误的法法编码之前，确保你的计算公式都是正确的将是非常有用的。

逻辑数据映射内部

在深入你将遇到的不同的数据源细节之前，我们需要研究逻辑数据映射文档的实际设计。该文档包括整个企业针对数据仓库源系统的数据定义，目标数据仓库数据模型，以及从

原有格式到最终目的转换所需要的完全数据操作。

逻辑数据映射的组成

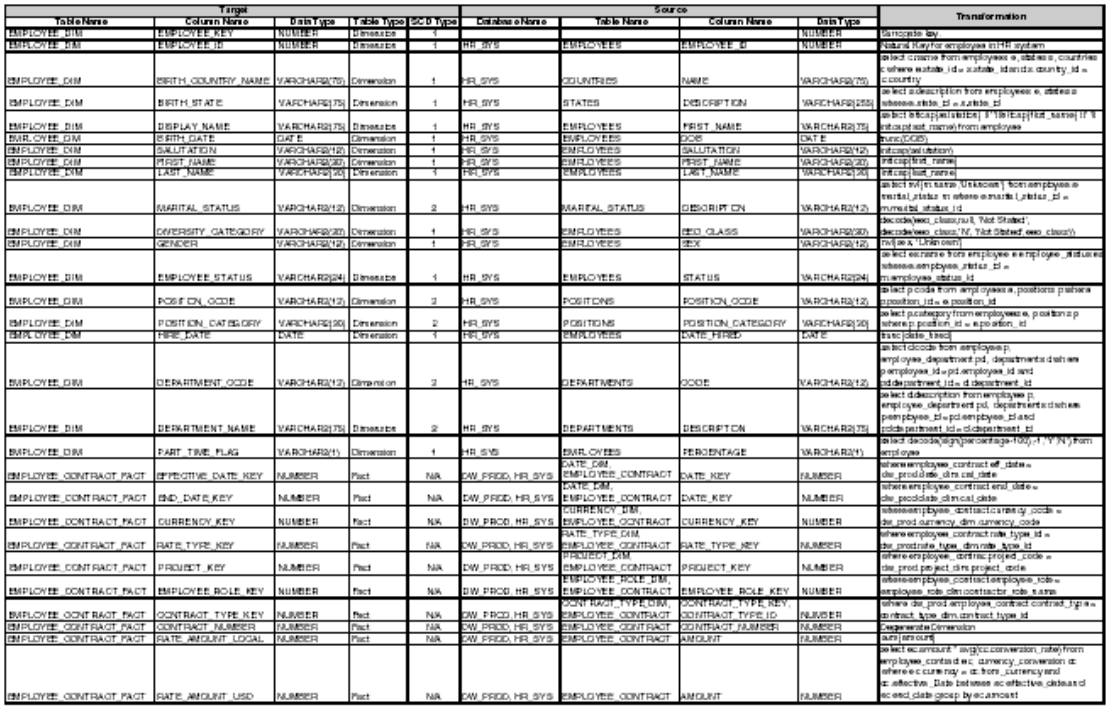
逻辑数据映射（见图 3.1）通常用一个表或者电子表格格式来表示，它包括以下特定的组成部分：

- 目标表名称：数据仓库中出现的物理表名称；目标列名称：数据仓库表中的列名称；
- 表类型：表示这个表是事实表，维表或者子维表（支节）
- SCD（缓慢变化维）类型：对维表，这个部分表示是类型 1，类型 2 或者类型 3 的缓慢变化维。这个指标对维表中的不同的列可以是不同的。
- 比如在客户维中，名字可能属于类型 2（保留历史信息），而姓可能属于类型 1（覆盖）。这些 SCD 类型将在第五章展开详细探讨。
- 源数据库：源数据所在的数据库实例的名称。这里通常是指连接数据库所需的连接字符串。如果出现在文件系统中，它也可以是一个文件的名称。这时，还需要包含这个文件的路径。
- 源表名称：源数据所在的源表的名称。很多时候需要多个表。这时，只需将生成目标数据仓库相关表的所有表简单列出即可。
- 源列名称：生成目标所需的相关列。简单的列出装载目标列需要的所有列。源列之间的关联在转换部分记录。
- 转换：源数据与期望的目标格式对应所需的详细操作。这个部分通常用 SQL 或者伪代码来编写。

逻辑数据映射中的列有时是组合的。比如，源数据库，表名称和列名称可能被组合在一个源列中。这个组合列的信息可能用原点来分隔信息，如 ORDERS.STATUS.STATUS_CODE。如果不考虑格式，这个逻辑数据映射文档的内容已经把进行有效的规划 ETL 过程的所有的关键的信息都提供了。

逻辑数据映射中的某些部分看起来很简单并且很直接。然而，当仔细研究的时候，该文档就会揭示许多 ETL 小组可能忽略的隐藏的需求。这个文档的主要的目标是为 ETL 开发者提供一个清晰的蓝图，它精确的说明可以从 ETL 过程获得什么。这个表必须清晰地描述在转换过程中包含的动作流程，不能有任何疑问的地方。

看一下图 3.1。



Target					Source				Transformation
Table Name	Column Name	Data Type	Table Type	SCD Type	Database Name	Table Name	Column Name	Data Type	
EMPLOYEE_DIM	EMPLOYEE_KEY	NUMBER	Dimension	1				NUMBER	Surrogate key
EMPLOYEE_DIM	EMPLOYEE_ID	NUMBER	Dimension	1	HR_SYS	EMPLOYEES	EMPLOYEE_ID	NUMBER	Natural Key for employee in HR system
EMPLOYEE_DIM	BIRTH_COUNTRY_NAME	VARCHAR2(75)	Dimension	1	HR_SYS	COUNTRIES	NAME	VARCHAR2(75)	select c.name from employees e, states s, countries c where e.state_id = s.state_id and s.country_id = c.country
EMPLOYEE_DIM	BIRTH_STATE	VARCHAR2(75)	Dimension	1	HR_SYS	STATES	DESCRIPTION	VARCHAR2(255)	select s.description from employees e, states s where e.state_id = s.state_id
EMPLOYEE_DIM	DISPLAY_NAME	VARCHAR2(75)	Dimension	1	HR_SYS	EMPLOYEES	FIRST_NAME	VARCHAR2(75)	select initcap(last_name)||', '||decode(initcap(first_name), '', '', initcap(first_name)||' ') from employee
EMPLOYEE_DIM	BIRTH_DATE	DATE	Dimension	1	HR_SYS	EMPLOYEES	DOB	DATE	trunc(DOB)
EMPLOYEE_DIM	SALUTATION	VARCHAR2(12)	Dimension	1	HR_SYS	EMPLOYEES	SALUTATION	VARCHAR2(12)	initcap(salutation)
EMPLOYEE_DIM	FIRST_NAME	VARCHAR2(20)	Dimension	1	HR_SYS	EMPLOYEES	FIRST_NAME	VARCHAR2(20)	initcap(first_name)
EMPLOYEE_DIM	LAST_NAME	VARCHAR2(25)	Dimension	1	HR_SYS	EMPLOYEES	LAST_NAME	VARCHAR2(25)	initcap(last_name)
EMPLOYEE_DIM	MARITAL_STATUS	VARCHAR2(12)	Dimension	2	HR_SYS	MARITAL_STATUS	DESCRIPTION	VARCHAR2(12)	select nvl(ltrim(rtrim('Unknown')),'Unknown') from employees e, marital_status ms where e.marital_status_id = ms.marital_status_id
EMPLOYEE_DIM	DIVERSITY_CATEGORY	VARCHAR2(30)	Dimension	1	HR_SYS	EMPLOYEES	RED_CLASS	VARCHAR2(30)	decode(exec_class,null,'Not Stated', decode(exec_class,'N','Not Stated',exec_class))
EMPLOYEE_DIM	GENDER	VARCHAR2(12)	Dimension	1	HR_SYS	EMPLOYEES	SEX	VARCHAR2(12)	nvl(sex,'Unknown')
EMPLOYEE_DIM	EMPLOYEE_STATUS	VARCHAR2(24)	Dimension	1	HR_SYS	EMPLOYEES	STATUS	VARCHAR2(24)	select ess.name from employee e, employee_status es where e.employee_status_id = es.employee_status_id
EMPLOYEE_DIM	POSITION_CODE	VARCHAR2(12)	Dimension	2	HR_SYS	POSITIONS	POSITION_CODE	VARCHAR2(12)	select p.code from employee e, positions p where e.position_id = p.position_id
EMPLOYEE_DIM	POSITION_CATEGORY	VARCHAR2(30)	Dimension	2	HR_SYS	POSITIONS	POSITION_CATEGORY	VARCHAR2(30)	select p.category from employee e, position p where p.position_id = e.position_id
EMPLOYEE_DIM	HIRE_DATE	DATE	Dimension	1	HR_SYS	EMPLOYEES	DATE_HIRED	DATE	trunc(date_hired)
EMPLOYEE_DIM	DEPARTMENT_CODE	VARCHAR2(12)	Dimension	2	HR_SYS	DEPARTMENTS	CODE	VARCHAR2(12)	select dd.code from employee p, employee_department pd, departments dd where p.employee_id = pd.employee_id and pd.department_id = dd.department_id
EMPLOYEE_DIM	DEPARTMENT_NAME	VARCHAR2(75)	Dimension	2	HR_SYS	DEPARTMENTS	DESCRIPTION	VARCHAR2(75)	select dd.description from employee p, employee_department pd, departments dd where p.employee_id = pd.employee_id and pd.department_id = dd.department_id
EMPLOYEE_DIM	PART_TIME_FLAG	VARCHAR2(1)	Dimension	1	HR_SYS	EMPLOYEES	PERCENTAGE	VARCHAR2(1)	select decode(sign(percentage-100),-1,'Y','N') from employee
EMPLOYEE_CONTRACT_FACT	EFFECTIVE_DATE_KEY	NUMBER	Fact	NA	DW_PROD, HR_SYS	DATE_DIM, EMPLOYEE_CONTRACT	DATE_KEY	NUMBER	where employee_contract.eff_date = dw_prod.date_dim.cal_date
EMPLOYEE_CONTRACT_FACT	END_DATE_KEY	NUMBER	Fact	NA	DW_PROD, HR_SYS	DATE_DIM, EMPLOYEE_CONTRACT	DATE_KEY	NUMBER	where employee_contract.end_date = dw_prod.date_dim.cal_date
EMPLOYEE_CONTRACT_FACT	CURRENCY_KEY	NUMBER	Fact	NA	DW_PROD, HR_SYS	CURRENCY_DIM, EMPLOYEE_CONTRACT	CURRENCY_KEY	NUMBER	where employee_contract.currency_code = dw_prod.currency_dim.currency_code
EMPLOYEE_CONTRACT_FACT	RATE_TYPE_KEY	NUMBER	Fact	NA	DW_PROD, HR_SYS	RATE_TYPE_DIM, EMPLOYEE_CONTRACT	RATE_TYPE_KEY	NUMBER	where employee_contract.rate_type_id = dw_prod.rate_type_dim.rate_type_id
EMPLOYEE_CONTRACT_FACT	PROJECT_KEY	NUMBER	Fact	NA	DW_PROD, HR_SYS	PROJECT_DIM, EMPLOYEE_CONTRACT	PROJECT_KEY	NUMBER	where employee_contrac.project_code = dw_prod.project_dim.project_code
EMPLOYEE_CONTRACT_FACT	EMPLOYEE_ROLE_KEY	NUMBER	Fact	NA	DW_PROD, HR_SYS	EMPLOYEE_ROLE_DIM, EMPLOYEE_CONTRACT	EMPLOYEE_ROLE_KEY	NUMBER	where employee_contract.employee_role = employee_role.role and contractor_role = 'Y'
EMPLOYEE_CONTRACT_FACT	CONTRACT_TYPE_KEY	NUMBER	Fact	NA	DW_PROD, HR_SYS	CONTRACT_TYPE_DIM, EMPLOYEE_CONTRACT	CONTRACT_TYPE_KEY, CONTRACT_TYPE_ID	NUMBER	where dw_prod.employee_contract.contract_type = contract_type_dim.contract_type_id
EMPLOYEE_CONTRACT_FACT	CONTRACT_NUMBER	NUMBER	Fact	NA	DW_PROD, HR_SYS	EMPLOYEE_CONTRACT	CONTRACT_NUMBER	NUMBER	Exagerate Conversion
EMPLOYEE_CONTRACT_FACT	RATE_AMOUNT_LOCAL	NUMBER	Fact	NA	DW_PROD, HR_SYS	EMPLOYEE_CONTRACT	AMOUNT	NUMBER	sum(amount)
EMPLOYEE_CONTRACT_FACT	RATE_AMOUNT_USD	NUMBER	Fact	NA	DW_PROD, HR_SYS	EMPLOYEE_CONTRACT	AMOUNT	NUMBER	select ec.amount * avg(cc.conversion_rate) from employee_contract ec, currency_conversion cc where ec.currency = cc.from_currency and cc.effective_date between ec.effective_date and ec.end_date group by ec.amount

图表 3.1　逻辑数据映射

仔细观察这个图，你可能会注意到一些注意事项，如果它们没有被关注，可能会引起许多故障诊断和调试的工作甚至最终延误这个项目。比如，你可能注意到 STATE 在源和目标中的数据类型已经从 255 字符转换为 75 字符。尽管数据分析文档可以支持数据范围的减小，但将来创建任何大于 75 字符的值时，就有可能丢失这些数据。而且，一些 ETL 工具实际上可能终止或者使整个包含这些数据溢出错误的过程失败。请注意 STATE 的转换说明没有明确定义这一数据转换—转换是隐含的。根据定义，不应该有一个明确定义的账户是隐含转换的。隐含转换在破坏你的处理流程方面是常见和臭名昭著的。为了避免不幸的发生，ETL 小组必须假定对这些类型的隐含数据转换都负责任地进行了明确的处理。

ETL 工具包通常持续跟踪这些隐含的数据转换并提供报告来标识任何的这种类型的转换。

表类型给了我们数据加载过程执行的次序—先是维表，然后是事实表。

与表类型一起，加载维表过程 SCD 类型显得至关重要。正如我们在这一章的前面所解释的，表结构的本身无法揭示缓慢变化维的策略是什么。错误地解释缓慢变化维策略将引起数周开发时间的浪费。在开始开发加载过程之前，需要清楚地了解哪些列需要保留历史信息以及如何获取历史信息所需的策略。这个列的值随着时间的推移可能发生改变。通常在单元测试的时候，当你第一次挑选用户来观察数据仓库中的数据时，他们看到了不想得到的结果。即使数据建模者极其努力的尝试，SCD 的概念还是很难向用户表达，并且一旦他们开始进行维度的加载时，他们总是想要与你的 SCD 策略背道而驰。这种需求是很常见的，这需要数据仓库项目经理来处理并改变管理流程。

映射中的转换是这个流程中的小阶段，这里是技术性好的开发者首先关注的。但你必须迫使自己不要只集中精力在编码上，在进入转换之前要仔细检查整个映射。这个转换可能包含了整个解决方案或者根本就什么都不是。最常见的，转换可能用 SQL 来表示。SQL 可能是也可能不是完整的语句。通常，它是那些不能在映射用其他元素表达的代码段，比如 SQL

中的 WHERE 子句。在其他时候，转换也可能是一种方法，没有特殊的 SQL，只有普通的英语说明，比如从一个平面文件进行预加载，或者基于数据库之外的标准进行加载转换，或者拒绝已知的异常数据到一个拒绝文件中的指导性说明。如果转换是空的，这说明映射是直接从源到目标的，没有任何转换的需求。

以上是逻辑数据映射的全部内容，在开始任何实际的编码开始之前和 ETL 开发者一起对文档做一个全面的走查。

使用工具设计逻辑数据映射

一些 ETL 和数据建模工具能直接抓取逻辑数据映射信息。人们很自然的想要在这些工具中直接表示数据映射。输入信息到工具中使得我们可以共享元数据，这是不错的选择。但在写信息的时候，并没有说明与逻辑数据映射相关的适当的数据元素的标准。在各种工具中可用的实际元素差别很大。正如数据仓库中的元数据标准是成熟的一样，对逻辑数据映射中的元素定义也应该建立一个标准。建立元数据标准将使得这些工具对这一目的变得更加一致和可用。你要调查你当前的工具包对存储逻辑数据映射来说是否可用并充分利用已有的任何功能。然而，如果你的工具没有实现所有你需要的元素，你将不得不把逻辑数据映射放在不同的地方，这使你的维护工作变成一件非常麻烦的事件。请密切关注各种产品在这方面所做的改进。

创建逻辑数据映射

数据仓库的成功主要来源于这样的情形，所有的数据放在一个逻辑位置，使用户可以执行交叉功能分析。在后台，ETL 小组无缝的整合和转换分散的无组织的数据并使它看起来好像从一开始就在一起的一样。数据仓库成功的主要标准之一是它存储的数据是干净的和一致的。统一的数据存储需要对每一个源系统有相当深入地了解。对数据源中的数据甚至源系统本身理解的重要性常常在 ETL 的项目规划阶段被忽略和低估。在源系统得到确认和分析之前完整的逻辑数据映射是不存在的。源系统分析通常分为两个主要阶段：

- 数据发现阶段；
- 异常检测阶段；

数据发现阶段

一旦理解了目标需要做成什么样子后，就要确认和检查数据源了。部分或者全部的源系统可能在数据建模期间发生很大变化，但并不需要过多地考虑这些。通常，只有主要的源系统在数据建模期间被确认出来。要知道数据建模工程师的主要目标是建立一个数据模型，这一点很重要。任何来自数据建模期间的逻辑数据映射都只是一个副产品----一个起点。况且，数据建模工程师花大部分的时间和最终用户在一起，所以在逻辑数据映射中定义的源系统可能不是真的初始的或者优化的源—记录系统。这需要 ETL 小组更深入到数据的需求中，确定每一个需要加载到数据仓库中的源系统、表和属性。为每一元素确定适当的源或者记录系统是一个挑战，必须仔细评估。完整的分析可以减少在 ETL 过程中由于使用错误的源而导致的数周时间的工期延误。

收集和文档化源系统

源系统通常由各种文档组成，包括会谈纪录，报表，以及数据建模人员的逻辑数据映射。ETL 小组通常需要更多的调研。和团队系统以及业务分析师一起工作来找到适当的源系统。在大型的组织架构中，你必须问“还有谁在使用这些数据？”这样的问题，并找出每一个用户组的数据源。一般的组织架构都拥有数不清的不同系统。ETL 小组有责任保持对这些系统的跟踪来发现和研究它们，并确定作为一个数据仓库的数据源是否有用处。

保持跟踪源系统

一旦源系统被确定，就要说明这些系统的意义以及谁对他们负责。图 3.2 就是为这个目的建立的图表。这个图表，也就是源系统跟踪报告，使得我们不用总是麻烦系统管理员或者业务管理者。如果走运的话，数据建模工程师将开始制作这个列表。不管谁定的初稿，这个列表的维护应该是 ETL 小组和数据建模小组共同努力来完成。如果在分析过程中认为一个源系统不适合作为数据仓库的源，保留它在这个列表中，并说明不使用它的原因；可能在后续阶段会用得到。

Subject Area	Interface Name	Business Name	Priority	Department/ Business Use	Business Owner	Technical Owner	DBMS	Platform	Daily User Count	Database Size (GB)	Transactions/ Day	Comments
Human Resources	PeopleSoft	HR Information	1	Human Resource Mgmt	Daniel Burris	Margaret Kerlin	Oracle	AIXon RS/6000	850	3	8,000	Everything revolves around HR
Finance	Oracle Financials	Oracle Financials	2	General Accounting	Mabel Kerr	Edmund Bykel	Oracle	AIX on S80	400	60	20,000	Driven sponsor: Department assessing & doing data
Materials Management	WMS	Warehouse Management System	3	Manufacturing Planning & Control	Annabel Giutu	Christina Hayim	Oracle	AIX on S80	350	200	5,000	Initiative to cut inventory of order movers. Need analytical support
Operations	BOMM	Bill of Materials Management	4	Production Planning	Lucy Kard	George Dimambro	Oracle	AIXon RS/6000	60	8	TBD	Need analysis of "kits". Want to build kits in-house
Marketing	IMDM	Interhemisphering & Campaign Management	5	Marketing	Elizabeth Impear	Brian Bridalung	SQL Server	W2K on Compaq	50	350	Varies	Needs Data Mining Capabilities
Human Resources	Positions Website	Positions Website	5	Human Resource Mgmt	Florence Ivzar	Andrew Acanthal	SQL Server	W2K on Compaq	50	1	10,000	Manager is near, eager for DW
Purchasing	PAS	Purchasing & Acquisition System	6	Purchasing Department	Guy Crowdil	Sybil Mal	SQL Server	NT on Compaq	75	4	1,500	
Customer Service	CSS	Customer Service System	7	Customer Care	Bernard Beals	Arthur Attoolo	Notes/ Domino	NT on Compaq	275	6	1,500	Plans to move application to Oracle next year.
Operations	QCS	Quality Control System	8	Quality Control	Thomas Ryar	Elias Caye	SQL Server	NT on Compaq	450	12	500	Needs failure rate analytics by product, by vendor, by year
Sales	SFA	Sales Force Automation	9	Sales	Anthony Aran	Emma Inulorm	DB2	AS/400	1200	10	2,500	Politically challenging. Interim reporting system in place

图表 3.2　源系统跟踪报告

源系统跟踪报告也可以作为数据仓库的后续阶段的概要。如果有 20 个源系统被确定在这个列表中，并且阶段 1 包括两个或者三个系统，则规划这个项目要很长很长的时间。

- 主题域：系统的数据集市常用的名称。
- 接口名称：源系统支持的交易应用系统的名称。
- 业务名称：业务用户通常使用的系统名称。
- 优先级：用于确定将来阶段的位置顺序，优先级通常在数据仓库总线矩阵完成后设置。
- 部门/业务用法：使用数据库的主要部门，比如：会计、人力资源等等。如果有许多部门使用这个应用，指出这个业务用法，比如：库存控制、客户跟踪等等。
- 业务所有者：与使用应用或者数据库的相关问题的联系人或者小组。一般来说该人或者该小组是这个主域域的数据负责人。
- 技术所有者：一般是负责维护数据库的 DBA 或者 IT 项目经理。
- DBMS: 源数据库管理系统的名称。大多数是关系型数据库，比如 ORACLE, DB2, 或者 SYBASE。也可能是非关系型数据存储如 LOTUS NOTES 或者 VSAM。
- 生产服务器/OS：这个列包含数据库运行的服务器的物理名称。也包括操作系统。当为 ETL 设计操作系统级的脚本时，就需要这个列。比如，不能在 NT 上使用 UNIX SHELL 脚本。
- #日常用户：让你知道组织中有多少操作型人员使用这些数据。这个数字并不是潜

在的数据仓库的最终用户数。

- DB 大小：DBA 可以提供这个信息。知道源数据的记录数有助于你确定 ETL 的优先级和将要付出的工作量。一般来说，越大的数据库的优先级别越高，因为当在交易系统中查寻大表或者几个表关联查询时性能常常变得很差。
- DB 复杂度：系统中的表和视图对象的数目。
- #每日交易数：估计出该数字使你对增量加载过程所需要的容量有个认识。
- 注解：通常在研究数据库的时候用来做一般观察报告。可能包括关于未来数据库版本的注释或者为什么某些实体是或不是一个记录系统的原因。

确定记录系统

如同数据仓库世界中的许多术语一样，记录系统有许多不同的定义—这些差别取决于你问谁。我们定义的记录系统相当简单：数据的发源地。记录系统的这个定义非常重要，因为大多数的企业数据是跨越许多不同的系统冗余存储的。企业这样做是为了让这些未集成的系统共享数据。这样相同的数据很容易在企业中复制，移动，操作，转换，修改，清洗，或者被破坏，就造成了同样的数据有多个不同版本。大多数情况下，数据链末端的数据和原始的数据—记录系统已经大不相同。我们曾经在一个项目中四次将原来确定的源数据从记录系统中删除。在系统 3 和 4 之间的处理过程中，数据通过一个试图清洗数据的算法来转换。这个算法有一个未知的缺陷，它将其他字段数据插进来从而破坏了数据。这个缺陷在项目的数据发现阶段被 ETL 小组发现了。

处理导出数据。

你可能对周围的导出数据感到有些混淆。是否 ETL 过程应接收源系统中的计算过的列作为记录系统？还是只有那些导出数据的基本元素才是合适的？这个问题的答案部分基于是否这个计算列是可加的。不可加的度量不能被用户在查询中组合，而可加度量却可以。因此你必须使用基本的元素来计算不可加度量。但要考虑清楚，是否你要在 ETL 过程中重建这些计算，如果这样做，你还要负责使这些计算和定义这些计算的业务规则保持同步。如果源系统中的业务逻辑改变了，ETL 过程将不得不重新修改和部署。因此，获取计算当作元数据以便让用户能够理解数据是如何得到的是很有必要的。

除非有确凿的证据表明原始的数据是不可靠的，否则我们建议你不要偏离我们的记录系统定义。紧记数据仓库的目标是在所有的主题域中可以共享一致的维度信息。如果你选择不用记录系统来装在你的数据仓库，那么一致的维度将变成不可能。如果你要为特定的需求用不同版本的数据来增加你的维度，则这些应该做为一致维度的额外的属性进行存储。

但是，每一种规则都会有例外。确定数据库名称或者文件名可能并不像想象中那样容易—尤其是当面对的是遗留的系统时。在一个项目中，我们曾经花费了数周的时间去获取订单数据库。每一个我们询问的人给这个数据库的说法都不同。后来我们发现每个地方都有一个本地的数据库版本。由于数据仓库的目标是跨越整个组织共享报表，我们开始记录每一个需要用 ETL 来迁移数据的本地数据库的名称。在我们研究这些数据库名称的时候，有一个开发员过来说，“你可以从这个数据复制程序中获得你需要的所有的数据库列表”。令我们惊奇的是，那儿竟然已经有了一个将本地数据库复制到一个中心资料库的程序。没有重建整个循环，我们选择使用这个程序来获得统一的数据库名称并从中心资料库加载数据到数据仓库。尽管真正的原始数据存在于每一个本地数据库中，然而使用中心资料库是最有效和最可靠的解决方案。

距离原始数据源越远，抽取到损坏数据的风险就越大。除非特殊情况，要保持实际的数据源只来自记录系统。

分析源系统：使用来自数据评估的结果

一旦确定了记录系统，下一步就是分析源系统以获得对其内容的更好的理解。如果是基于关系型技术，一般通过获得系统的 ER 图来对所选的记录系统加深理解。如果没有 ER 图（不要感到奇怪），则自己创建它们。ER 图可以通过对数据库进行反向工程来获得。反向工程是一种通过已知的数据库元数据来生成 ER 图的技术。许多数据评估工具可以很容易做到这点。几乎所有的标准的数据建模工具都有这个功能，主流 ETL 工具中有一些也有这个功能。

利用反向工程获得一个记录系统的 ER 图是很有用的。但他和利用正向工程从复杂的 ER 模型来建立简单维度模型结构不同。当在拥有上百张表的规范化环境中只见树木不森林时，数据建模工具实际上在这正向工程中往往会失败的很惨。

在你深入 ER 图之前，查看一下数据库中的表和列的更高级别的描述。如果可以找到，它可能以非结构化文本描述的形式存在，可能已经过了时，但从概述开始远比试图通过在巨量的细节来发现概述要好得多。当然，别忘了听取源系统的头头们的意见，他们可以告诉你谁了解这些源系统中所有复杂的逻辑和已经发生的不断增长的变化！

具有浏览 ER 图的能力是执行数据分析的基础。ETL 小组中的每一个成员都要能读懂 ER 图并能很快认出实体关系。图 3.3 图解一个简单的 ER 图。

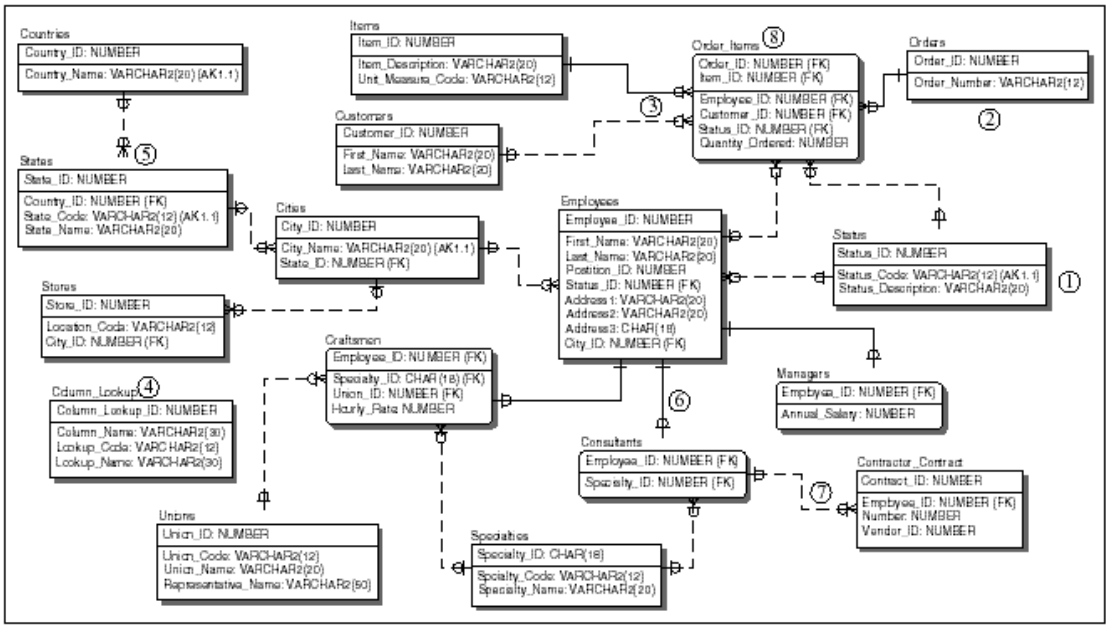


图 3.3　实体关系图

在以下数字标号的列表中，我们解释了在这个阶段你最需要搞清楚的最主要特性，包括唯一标识，是否可为空以及数据类型。这些是数据评估的主要输出结果。但更重要的是，我们还解释了当表彼此相关时如何来标识；以及哪些列在多个表中有依赖关系。图 3.3 中的 ER 图概要中的主要特性如下：

1．唯一标识和自然键。唯一标识指的是能唯一的确定表中的一行的列。这个定义可能被误解，所以我们需要对此进行深入的研究。从参考完整性的观点出发，唯一标识就是一个表的主键。大多数时候，主键是人造的，虽然从 ETL 的观点来说它是唯一的，但并不能提供足够的信息来确定是否这一行是唯一的。在每一个设计良好的事务性表中，除了主键，必须至少还有一个自然键。自然键用来唯一描述该行的业务作用。比如，一个状态表可能包含一个 status_id ,一个 status_code 和 status_description 。很明显 status_id 是主键，但取决于业务规则，为了 ETL 的目标，status_code 可能才是唯一标识的自然键。选择正确的自然键的时候要特别小心，尤其当加载缓慢变化维时。

2．数据类型。记住，作为 ETL 分析师，你没有得到任何授权。列名称不能推断出数据类型。仅仅因为列的名称为 Purchase Order Number，是否我们就可以肯定这个列里只有数字，没有字母？另外，如果只有数字，是否有前导零？这些零对最终用户重要吗？在一个具体项目中，当建立人力资源数据集市时，我们遇到一个名为 SCORE 的源列；它是一个 CHAR(2) 数据类型。在目标列，它是一个 NUMBER 数据类型。结果，我们发现大约 80％的数据是数字，其余的是字母（A-D 和 F）。ETL 程序需要转换任何的字母为它们对应的数字。DATE 和 TIME 元素存储为文本是常见的问题，当要将日期加载到数据仓库的时候，就由 ETL 程序来转换它们。

3．表之间的关系。理解表之间是如何关联的对获取数据时保证准确地连接是至关重要的。如果运气好，ER 图有关联表的连接线。鉴定表之间的关系包括分析关联线。不幸的是，数据处理者通常不得不仔细的察看 ER 图来确定表关联。当从不同的数据源往一个目标表上加载数据时，一种好的做法是将所有的数据源放在一个数据建模工具中并创建所有的关系。这种集成的 ER 图使得逻辑数据映射更加容易创建。

4．离散关系。在源系统的设计中，你不难见到用一个查找表来存储这个数据库中所有表的静态参考数据的情况。这个查找表中有一个列来表示哪一个表和列由由相应的行组来支持。这对于不熟悉的人来说需要花费时间去理解。需要详细的以文档记录每一个行组以及相应的表和列。这些信息在将来映射多个维度时会用到。

5．关联和列的基数。推算查询的结果需要知道关联的基数。用鱼尾纹表示法，一个单横线表示基数为 1，只有一个唯一的对应。一条线加一个圆圈表示有 0 或 1 个对应。线的下面加 3 条线像鱼尾纹形状表示可以有多个对应。在关系型数据库中，所有的表关联都是以下关联中的一种：

- 一对一：一对一关系只在超类/子类场景中以及垂直表分割的情况下出现。一对一关系可以通过查看是否每一个表中该列都是主键来确定。
- 一对多：对外键参考来说这是最常见的关系。它很容易确定，注意表中的非键属性要参考另外一个表的主键。我们称这种非键属性为外键，并且我们坚持所有的外键都是完备的，也就是说，它是其对应主键的一个实例。
- 多对多：这种关系通常涉及三个表，其中有两个一对多关系。更确切地说，两个表之间有一个关联表。中心表或者关联表有一个联合主键和两个外键，一个对应其中一个表的主键，另一个对应另一个表的主键。

通常情况下，源系统的数据字典中没有一致的外键或者参考完整性定义。这些问题也可能通过简单的列名匹配和更复杂的数据评估来发现。

确保仔细研究了在源、中间集结表以及最终要提交的表中的所有的数据类型。数据建模小组常常创建和源所用的并不完全匹配的数据元素。有时候，你发现数据类型是

故意设计成不匹配的。比如，即使当前系统只用数字并且是数字类型字段，但某些设计者还故意使所有的编码字段都支持字母字符。同时，准确评估每一个字段的长度。在某些情况中，目标数据仓库的数据长度或数字精度可以比源数据库小些。目标中的较短的数据长度会导致数据截断（数据丢失!）。当你发现不一致时，和数据建模小组一起检查以确定他们的意图。或者等待数据库纠正错误，或者获取转换和截断的业务规则。

数据内容分析

理解数据的内容是确定最好获取数据方法的关键。通常，在你开始和数据一起工作之前，你就已经意识到存在一些不规则数据。通常遇到的不规则数据列表如下：

- NULL 值：一个未处理的 NULL 值足以毁掉任何 ETL 程序。当出现在外键列中时，NULL 值是最危险的。以一个包含 NULL 值的列为基础来关联两个或更多表将导致数据丢失！切记，在关系型数据库中 NULL 并不等于 NULL。这就是为什么那些关联失败的原因。请检查源数据库中每一个外键是否有 NULL 值。如果存在 NULL 值，必须对表进行外关联。外关联返回所有的行，不管关联表的值是否匹配。如果 NULL 数据不是一个外键列而是一个数据仓库需要的列，你必须有一个处理 NULL 数据的业务规则。我们不希望在数据仓库中存储 NULL 值，除非它真的是一个未知的度量。只要可能，在往数据仓库中加载数据时一定要建立默认值来代替 NULL 值。
- 非日期字段中的日期：日期是非常奇特的元素，它们是唯一的有多种格式、字面上包含不同值但又意思相同的逻辑元素。幸运的是，大多数数据库系统支持多数的显示格式但只用一个标准格式来存储（针对此特定数据库）。然而有许多情形日期是以文本字段存储的，在遗留的旧系统中尤其常见。非日期字段中的可能的日期格式是多种多样的。以下是一些用文本来存储日期格式数据的例子：

  13-JAN-02

  January 13,2002

  01-13-2002

  13/01/2002

  01/13/2002 2:24 PM

  01/13/2002 14:24:49

  20020113

  200201

  012002

只要你能想得到，一个日期的各种格式可以写满数页纸。当源数据库系统对日期数据不做控制或规范时，你必须非常仔细地检查以保证确实得到你期望的东西。

不管如何仔细地分析，我们都建议在从关系型数据库系统中导出数据时使用外连接，因为通常远程系统的参考完整性很难保证的。

在 ETL 过程中收集业务规则

你可能以为在 ETL 过程的这个阶段所有的业务规则都必须收集完毕。没有清楚所有的业务规则之前数据建模者怎么能创建数据模型，不是吗？错。数据模型小组需要的业务规则和 ETL 小组需要的业务规则有很大的区别。比如，数据建模定义的状态维可能是这样的：

Status Code——状态是一个 4 位数的代码来唯一的标识产品的状态。这个代码有一个简短的描述，通常是一个词，还有一个完整的描述，通常是一句话。

相反的，ETL 定义的状态可能是这样：

Status Code——状态是一个 4 位数的代码。然而，还有些遗留的代码只有 3 位数在某些情况下仍然会用到。所有的 3 位数代码必须转换为相应的 4 位数代码。这个代码的名称中可能会包含有单词 OBSOLETE（已不用的）。OBSOLETE 要从代码名称中删除并且这些废弃的代码必须设置废弃标志为’Y’。当进入源系统时，无论如何使用，描述应该总是句子形式的。

在数据仓库项目中，业务规则对 ETL 处理来说更像技术性问题而不像任何的业务规则收集。无论它们技术上如何体现，这些规则仍然不是业务——ETL 小组不能投入到制定规则的业务中。ETL 设计者的任务是将用户需求转换成可用的 ETL 定义并且用一种业务用户能理解的方式清晰的说明这些技术定义。ETL 数据定义的过程是一个演化的过程。当你发现无法用事实证明的数据异常时，用文档记录下来并和业务人员（只有他们可以指示如何处理这些异常时）一起来讨论。任何从这些会议得到的转换都必须用文档记录，得到批准并签字。

集成异构数据源

这一章的前面部分陈述了许多当生成数据仓库时可能遇到的常见的数据系统。该部分讨论的是将面临的不同数据源集成的挑战。然而在可以整合数据之前，必须了解什么是数据集成。整合数据不仅仅是简单的将不同来源的数据收集并存储在一个资料库中。要更好的理解什么是集成，设想一下公司并购。在公司并购过程中，一个或者多个公司加入到其它类似（或不相似的）的公司中。当并购发生时，由业务决定哪一个公司存活以及哪一个公司被新的母公司合并。往往，当母公司发觉在某些它的附属的公司的实践和技术中的价值需要被合并到它的重建的组织中时就需要进行谈判了。一个成功的公司合并是形成一个具有单一业务利益的整体组织。这时需要英雄所见略同，以这种方式来统一业务术语（维度属性）和关键绩效指标（事实表中的事实）。如果你希望像公司合并一样来集成你的数据，那么你的数据仓库将是一个支持业务利益的有组织的单一信息源。

但对那些只完成了一半的合并，将允许他们的子公司做自己的业务，此时将会怎样呢？由于这些公司没有整合——他们只是联合，这种情况会引起一些问题。当建立一个数据仓库时，集成在几个地方发生。最直接的数据集成的形式是进行维度规范化。在数据仓库中，规范化的维度是统一整个企业分离的数据系统的一致性设计。

当一个维度是由来自几个独立系统的数据组成时，很重要的一点是把这些系统中的每一个唯一标识包含到数据仓库的目标维度中去。那些标识必须是用户可见的，确保他们能理解这个维度与其数据的对应关系，这样才可以回溯到他们的交易系统中。

当那些特殊的维度没法完全整合时会怎么样呢？不幸的是，这个问题更像一个政策问题

而不是技术问题。整合维度和事实对于数据仓库项目的成功是尤其关键的。如果你的项目最后提供的是没有跨业务主题域的、完全不同的维度，那你的目标就没有完成。第 5 章我们将详细讨论加载维度的问题，但我们在这里要提及一些特殊的技巧，用于从一个完全不同的源系统环境加载规范化的维度。

1．标识源系统。在数据评估的逻辑数据映射的设计阶段，数据仓库小组必须一起找出目标维度和事实所在的各种源系统。数据仓库结构设计师必须尽可能地发现数据仓库中每一个元素的潜在源并试着为每一个元素指定一个记录系统。要把这个记录系统当作将要加载的数据的最终的源。

2．理解源系统（数据评估）。一旦确定了源系统，你必须对每一个系统进行全面的分析。这也是数据评估的一部分。源系统的数据分析将揭示不可预知的数据异常和数据质量问题。这个阶段将发布关于元素在源系统的可靠性报告。在项目的这个阶段，如果仍然存在数据质量问题或者由于任何原因数据的可靠性仍然是问题，记录系统的可能要重新分配。

3．创建记录匹配逻辑。一旦理解了所有考虑中的系统的所有实体的所有属性，则下一步目标就是设计匹配规则使得分离的系统中的实体可以关联起来。有时，匹配规则就像标识各种客户表的主键一样简单。但是更多时候分离的系统并没有共享的主键。因此，你必须基于模糊的逻辑来关联表。也许社会保险号码可以作为你的客户的唯一标识，或者可能还需要再联合姓名、e-mail 地址等电话号码等一起考虑。我们不是要提供一个匹配方案，而是为了获得对客户关联关系的理解。在最终的匹配逻辑中必须包含和体现各种业务领域。不要忘记要确保这个记录匹配逻辑与各种法律保密条款相一致，比如医疗卫生服务领域的 HIPAA。

4．建立生存规则。一旦记录系统已经标识出来并且匹配逻辑已经核准，则可以创建当 ETL 过程中发生数据冲突时的生存记录。这意味着如果一个客户表在应付帐款、产品控制和销售系统都有时，业务规则必须决定当属性重叠时哪一个系统占有压倒性的地位。

5．建立非键属性的业务规则。记住，维度（以及事实）一般源自某个系统中的许多表和列。此外，许多源系统可以（其实通常这样做）包含最终输入一个目标维度的不同属性。举例来说，部门列表可能最初在你的 HR 部门；然而，部门的会计科目可能来自你的财务系统。即使 HR 系统可能是记录系统，但某些属性可能来自其它系统更可靠些。当属性存在几个系统中，但是不在记录系统中时，为非键属性分配业务规则显得尤其重要。在这些情况下，用文档记录并发布数据线元数据是非常重要的，这可以在用户没有看到他们期望的内容时避免对数据仓库的完整性产生怀疑。

6．加载一致的维度。数据集成过程的最后一环是物理加载一致的维度。这个步骤要考虑缓慢变化维（SCD）类型并根据需要更新最新数据。加载一致的维度的详细信息请参考第 5 章。

数据仓库的吸引人之处就在于它具有真正集成的数据，同时也让用户从他们的角度查看维度。一致的维度和事实是企业数据仓库的基石。

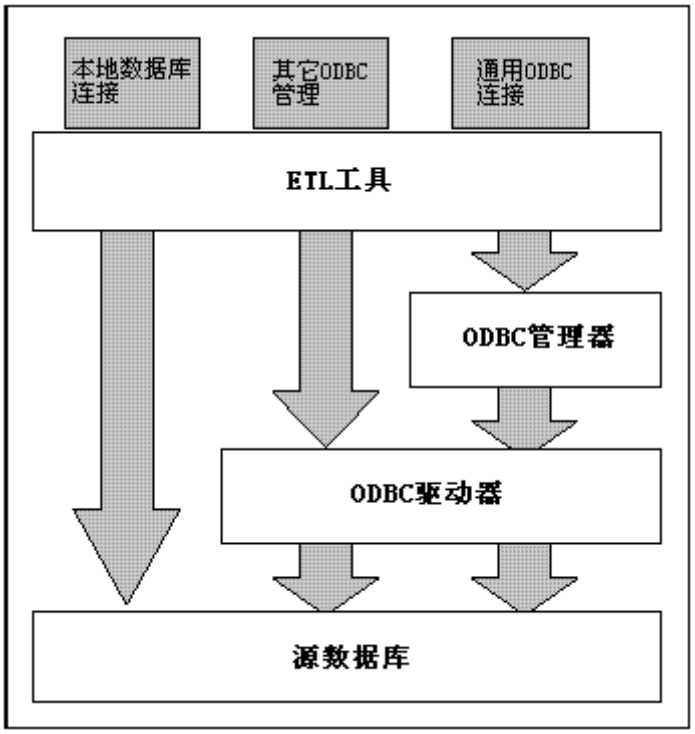
第 2 部分：从不同平台进行抽取的挑战

每一个数据源可能是在不同的 DBMS 中也可能在不同的平台上。尤其遗留的和专用的数据库和操作系统可能需要不同的过程语言来连接他们的数据。在企业级的数据仓库项目中，要准备连接到限定某种特殊语言的源系统。即使没有技术上的限制，部门或者子系统可能（并且通常是）有一种标准语言允许访问他们的数据。我们已经遇到的使用的标准包括 COBOL、FOCUS、EasyTrieve、PL/SQL、Transact-SQL 和 RPG。如果某种语言超出了你的 ETL 工具的使用范围或者根据经验变得无效，就需要向源系统的所有者求援把数据导出到平面文件格式中。

通过 ODBC 连接不同的源

开放数据库连接（ODBC）的建立使得用户可以从他们的 Windows 应用程序中去访问数据库。ODBC 建立的初衷是使得应用程序变成通用的，意思是说，如果一个应用程序的底层数据库改变了－－比如从 DB2 变成了 Oracle，这个应用程序不需要重新编码和编译就可以适应这个变化。只需简单的改变 ODBC 驱动程序即可，这对应用程序来说是透明的。可以获得实际任何平台上的每一种关系数据库的 ODBC 驱动程序。也可以使用 ODBC 来访问平面文件。

ODBC 的灵活性是以牺牲性能作为代价的。在将数据提交到数据操作过程前，ODBC 增加了几个层次的处理。对那些通过 ODBC 处理数据的 ETL 过程，在 ETL 系统和后台数据库之间增加了两层。图 3.4 说明了 ODBC 环境中包含的层次。



图表 3.4　ETL 流程中的 ODBC 结构图

- ODBC 管理器。ODBC 管理器是一个从 ETL 程序接收 SQL 并发送到相应的 ODBC 驱动程序的应用程序。它也维护应用程序和 ODBC 驱动程序之间的连接。
- ODBC 驱动程序。ODBC 驱动程序是 ODBC 环境中的实际处理核心。ODBC 驱动程序将 ODBC SQL 转换为到后台数据库的本地 SQL。

正如你所担心的，一旦使用 ODBC 你将失去许多关系数据库特殊的功能。尤其是非 ANSI 标准的 SQL 命令不会被 ODBC 管理器接受，因为它要保持开放性。ODBC，尤其是微软的 OLE DB 和.NET 组件，在近年来已经得到了显著的改善，但是对于性能和本地关系数据库功能要求非常高的情况则必须首先考虑本地数据库驱动程序。只是不要因噎废食。ODBC 对某些不容易导出的比较麻烦的数据源可以提供一个统一格式的接口。

主机数据源

主机在上世纪 60 年代中期出现，被广泛用在全球那些最大型的企业中。主机和其它计算机的唯一区别是硬件结构。非主机，包括小型机和微型计算机，使用他们的中央处理器(CPU)进行所有的实际处理，包括从硬盘和其它外围设备取数据。相反的，主机有特定的结构加强外围设备通道来处理输入/输出，CPU 可以专注处理数据，比如计算和均衡。

在许多大型企业，许多的日常业务数据由主机系统（某些小型机系统，比如 IBM AS/400）来处理和存储，从这些系统中集成数据到数据仓库时会面临一些特殊的挑战。以下是几种 ETL 小组必须熟悉并利用特定技术来处理的特殊的主机系统：

- COBOL copybooks
- EBCDIC 字符集
- 数字数据
- 重定义字段
- 压缩的十进制字段
- 多 OCCURS 字段
- 多记录类型
- 各种记录长度

接下来的部分将讨论这些主机特有的数据并提供特定的技术来进行处理。

处理 COBOL Copybook

COBOL 仍然是主机上的主要编程语言，数据的文件格式用 COBOL copybooks 来描述。一个 copybook 为主机数据文件定义字段名称和相关的数据类型。和其它在 ETL 过程中遇到平面文件一样，主机平面文件也只有两种数据类型：文本和数字。然而，数字值有许多存储方式，你必须充分理解来进行正确的处理。同样，日期仅简单的存储为数字（或文本）字符串，通常需要转换成为日期字段存储在数据仓库中。

		长度	位置		
01 EMP-RECORD.					
05 FIRST-NAME	PIC X(10).	10	1	-	10
05 MIDDLE-INITIAL	PIC X.	1	11	-	11
05 LAST-NAME	PIC X(15).	15	12	-	26
05 SSN	PIC X(9).	9	27	-	35
05 EMP-DOB.		8	36	-	43
10 DOB-YYYY	PIC 9(4).	4	36	-	39
10 DOB-MM	PIC 9(2).	2	40	-	41
10 DOB-DD	PIC 9(2).	2	42	-	43
05 EMP-ID	PIC X(9).	9	44	-	52
05 HIRE-DATE.		8	53	-	60
10 HIRE-YYYY	PIC 9(4).	4	53	-	56
10 HIRE-MM	PIC 9(2).	2	57	-	58
10 HIRE-DD	PIC 9(2).	2	59	-	60
05 TERM-DATE.		8	61	-	68
10 TERM-YYYY	PIC 9(4).	4	61	-	64
10 TERM-MM	PIC 9(2).	2	65	-	66
10 TERM-DD	PIC 9(2).	2	67	-	68
05 TERM-REASON_CODE	PIC X(2).	2	69	-	70

图 3.5　描述一个雇员记录的简单 copybook

图 3.5 说明了一个 70 字节的、固定长度的记录，用于描述一个简单的雇员记录。注意字段名称前有一个层级号。嵌套的层级号用来对相关的字段进行分组。COBOL 程序可以在定义的任何层次级访问字段名称。比如，一个程序可以访问 HIRE-DATE 来获得整个入职日期或者 HIRE-YYYY 来只获得年的部分。

文本和数字的数据类型用 PIC 语句来标志。PIC X 表示文本字段，而 PIC 9 代表这个字段是数字。字段长度是紧跟其后的数字。比如，语句 PIC 9(4)表示一个 4 字节的数字字段，而 PIC X(15)表示一个 15 字节的文本字段。PIC 语句也可以选择重复 X 或者 9 数据类型标志来表示，比如 PIC 9999 是一个 4 字节数字字段。

图 3.5 中的数据文件可以很容易通过 FTP 传输并加载到数据仓库中，因为所有的数据都包含显示格式。但是在试图从主机传输这个文件到数据仓库平台时，需要简单学习一下在 UNIX 和 Windows 平台上常见的 ASCII 字符集与使用在主机上的 EBCDIC 字符集之间的差别。

EBCDIC 字符集

大多数数据仓库所在的老的主机系统和基于 UNIX 和 Windows 的系统都采用二进制位和字节存储。每个字节由 8 个二进制位组成，每个二进制位表示一个二进制数。一个字节可以表示的最大数字是 255（就是 2 的 8 次方－1）。因此，可以用字节描述的唯一字符（比如，A-Z,a-z,0-9,标点符号和特殊字符）的数量是 256（包括字符 0）。

转换 EBCDIC 到 ASCII

你可能认为既然两种系统都使用二进制位和字节存储，数据从主机系统到你的 UNIX 或者 Windows 系统应该是直接可用的。但是 UNIX 和 Windows 系统使用美国标准信息交换码（ASCII）字符集，而主机使用不同的字符集，即著名的扩展二进制代码十进制互换码（EBCDIC）。EBCDIC 多少有些类似 ASCII 的字符集，但是用不同的 8 位组合来表示他们。

比如，小写字母 a，在 ASCII 中，字母 a 的字符编码是 97（01100001），但是在 EBCDIC，字符编码 97 是/（前向斜杠）。在 EBCDIC 中 a 是字符 129（10000001）。事实上，没有一个字符在 ASCII 和 EBCDIC 中的字符编码是一样的。要在 UNIX 或者 Windows 系统中使用来自主机系统的数据，首先必须将它从 EBCDIC 转换为 ASCII。

在平台间转换数据

幸运的是，从 EBCDIC 到 ASCII 的数据转换是相当简单的，并且实际上它是自动的。假设你使用文件传输协议（FTP）来从主机系统转移数据到数据仓库平台。一个 FTP 连接需要两个节点－－主机和客户端。当在两个系统之间创建 FTP 连接时，FTP 客户端会识别 FTP 主机的操作系统环境，由主机决定在两个系统之间传输数据时是否需要进行转换。因此，当在主机和 UNIX 或者 Windows 系统之间建立 FTP 连接时，FTP 主机将从 EBCDIC 到 ASCII 转换数据。另外，当数据在 UNIX 和 Windows 中时，FTP 增加特殊的换行和回车字符来表示一行（或记录）的结束。如果从 UNIX 或者 Windows 移动数据到主机上，FTP 也可以从 ASCII 转换到 EBCDIC。

如果收到的主机数据是在盒状磁带或者 CD-ROM 上而不是通过 FTP，则需要在数据仓

库系统中指定将数据从 EBCDIC 转换成 ASCII。这种转换可以用 UNIX 的 dd 命令带 conv=ascii 选项来执行。对 Windows 来说，可以在因特网上获得一个 dd 命令的端口——这也适用于许多其它有用的 UNIX 命令。另外，处理字符转换的商业产品也很多。所有的 ETL 工具套件都可以处理这种转换。大多数功能强大的专门为 ETL 设计的工具可以直接从 EBCDIC 转换成 ASCII。

如果源数据在主机系统上，则 ETL 工具具有从 EBCDIC 到 ASCII 的数据转换功能是很重要。如果可能，最好是在主机上进行转换以避免发生任何较小值的和压缩数字的损失。如果数据通过磁带或者其它介质获取，则这种转换必须由非主机环境中的 ETL 工具来实现。至少，这个 ETL 工具必须自动执行 FTP 以及在流中处理文件，并直接通过 ETL 过程将数据从主机提交到目标数据仓库中。

最后一点，虽然主机和 UNIX 或 Windows 系统使用不同的字符集，从一个系统向另一个系统转换数据也是一个相当简单的任务——简单，就是说，除非主机数据有些其它典型的主机环境特有的内容。后面的部分讨论主机数据可能具有的一些特征以及 ETL 过程中建议的处理策略。

处理主机数字类型数据

当你开始和量化的数据元素打交道时，比如美元总数、计数和余额，你会发现这些数字比眼睛看到的包含更多的信息。例如，通常在数字数据中找不到小数点，因为小数点是隐含的。比如，值 25,000.01 存储为 002500001。更糟糕的是，值 2,500,001 也是以同样的方式存储。那么主机 COBOL 程序是如何知道它是 25,000.01 而不是 2,500,001 呢？这在 PIC 语句中表示。下一部分会讨论 COBOL copybook 中的 PIC 语句的重要性和强大功能。

使用 PIC

你可以看到，在图 3.6 中 PIC 语句可以给同一个数据值不同的意思。要精确的处理来自一个旧的主机系统的数字值，你必须在传输它到数据仓库之前首先转换它为显示格式；否则，ETL 工具必须在 UNIX 或 Windows 平台上处理和解释这些主机值。要分解十进制数成为小数值，你可能认为简单的用这个数字值除以 10 的幂数(小数点位置数)就可以得到。如果所有的数字值只用隐含的小数点来存储，则这样做是正确的。然而，实际上不是那么简单，还必须考虑带符号的数字值问题。

PIC子句	值	数据
PIC 9(9)	2,500,001	002500001
PIC 9(7)V99	25,000.01	002500001
PIC 9(6)V9(3)	2,500.001	002500001
PIC S9(W) DISPLAY SIGN LEADING SEPARATE	2,500.001	002500001
PIC S9(7)V99 DISPLAY SIGN LEADING SEPARATE	25,000.01	+002500001
PIC S9(7)V99 DISPLAY SIGN LEADING SEPARATE	(25,000.01)	-002500001
PIC S9(7)V99 DISPLAY SIGN TRAILING SEPARATE	25,000.01	002500001+
PIC S9(7)V99 DISPLAY SIGN TRAILING SEPARATE	(25,000.01)	002500001-
PIC S9(7)V99	25,000.01	00250000A
PIC S9(7)V99	(25,000.01)	00250000J

图 3.6　COBOL copybook 中的 PIC 语句表示一个数字值的十进制位置。

在主机数据中，正负符号可能在数字值的前面或者后面。甚至，正负符号可能嵌入到数字值中。

最常见的格式是分区数字，符号嵌入到数字的最后一位数中，如图 3.6 中的最后两行所示。因此，在最后位置的 A 如何意味着既表示数字 1 也表示符号＋？同样的，J 又如何既表示 1 也表示-？方法是最后一个字节被当作两个独立的半字节（每个包含 4 比特），每个半字节独立解释－－当然，是用 16 进制！

对正数，第一个半字节设位 C，16 进制值是 1100，而负数设位 D，16 进制值为 1101。第二个半字节设成相应的数字的 16 进制值。当将第一个半字节(1100 为正数，或 1101 为负数）和第二个半字节合并时，你就得到了相应的 EBCDIC 字符，如图 3.7。

到现在，可能为如何从主机系统导出数值型数据伤透了脑筋。那么好，在试图解决这个问题之前，这里仍然有一个更加困难的问题可能在大多数旧主机系统中会遇到。

BIN	HEX	EBCDIC
11000000	C0	不可打印
11000001	C1	A
11000010	C2	B
11000011	C3	C
11000100	C4	D
11000101	C5	E
11000110	C6	F
11000111	C7	G
11001000	C8	H
11001001	C9	I
11010000	D0	不可打印
11010001	D1	J
11010010	D2	K
11010011	D3	L
11010100	D4	M
11010101	D5	N
11010110	D6	O
11010111	D7	P
11011000	D8	Q
11011001	D9	R

图 3.7　十六进制到 EBCDIC

解压压缩的数字

虽然目前计算机硬盘相对便宜，但是在过去磁盘存储几乎是计算机系统中最昂贵的部件。为了节省磁盘空间，软件工程师设计了创新的格式来存储数值型数据，它只需要比数字中的数位更少的字节。这些格式中最普遍使用的是 COMP-3，又称之为压缩数字。

在许多主机系统中，大多数(不是全部)数字数据是以 COMP-3 格式存储的。COMP-3 格式是一种简单的磁盘空间节省技术，使用半字节－或四位字节而不是全字节来存储数字位。每个数字位可以用一个四位字节以二进制格式存储。一个 COMP-3 数字字段的最后一个半字节存储数字值的符号（正号/负号）。使用半字节存储数字位和使用显示格式存储相比几乎节省了一半的空间。但是这种简单的空间节省技术给 EBCDIC 到 ASCII 字符集转换制造了一个大麻烦。

这种转换难题的一个直接结果是，包含数字值并且使用数字存储格式如分区数字或 COMP-3（还没有提到 COMP、COMP-1 和 COMP-2）的主机数据不能简单的从 EBCDIC 转换到 ASCII，然后在 UNIX 或 Windows 数据仓库中进行处理。

以下这些技术在进行主机数字数据的集成过程中一定会用到的：

- 在将数据传输到数据仓库系统之前，使用一个用 COBOL、汇编语言或第四代语言

如 SAS、Easytrieve 或 FOCUS 写的简单程序来按照显示格式重新格式化主机的数据。一旦数据按照这种格式重新格式化，它就可以像本章前面所说的那样通过 FTP 转换到 ASCII。

- 以本地 EBCDIC 格式传输数据到数据仓库系统。这个选项只有当你的 ETL 工具或过程可以处理 EBCDIC 数据才有用。以下几种类型的工具可以执行这个任务。
  - 使用可以处理本地 EBCDIC 的成熟的 ETL 工具，包括精确的处理以任何主机类型的数字格式存储的数字数据。
  - 在数据仓库平台上使用一个能重新格式化数据到显示格式的应用程序。如果收到 EBCDIC 数据并且没有某个 ETL 工具的帮助来写 ETL 过程，我们强烈建议你购买一个可以执行数字格式转换和 EBCDIC 到 ASCII 转换的应用程序。有些可得到的相对便宜的商业程序可以很好的处理这种任务。

使用重定义字段

为了不浪费空间(过去通常存储是很昂贵的)，主机工程师设计了 REDEFINES，这种方式允许相互分割的数据元素占用相同的物理空间。图 3.8 包含了一个来自 COBOL Copybook 的摘要，可用来说明主机数据文件中的 REDEFINES 的概念。这个摘要描述了用来表示一个雇员的工资信息的数据字段。注意到 EMPLOYEE-TYPE，这是一个表示这个雇员是否是全职的的按小时计薪的单字节代码。另外，注意到两个独立序列的字段存储这个雇员的工资信息。这个字段集是否使用取决于是否这个雇员是全职的或是按小时计薪的。全职的雇员工资在三个字段（PAY-GRADE, SALARY, PAY-PERIOD）中表示，总共占了 8 个字节。按小计时计薪的雇员使用一个不同的字段集，需要占 7 个字节（PAY-RATE, JOB-CLASS）。

	长度	位置		
05 EMPLOYEE-TYPE PIC X.	1	71	-	71
88 EXEMPT VALUE 'E'.				
88 HOURLY VALUE 'H'.				
05 WAGES.	8	72	-	79
10 EXEMPT-WAGES.	8	72	-	79
15 PAY-GRADE PIC X(2).	2	72	-	73
15 SALARY PIC 9(6)V99 COMP3.	5	74	-	78
15 PAY-PERIOD PIC X.	1	79	-	79
88 BI-WEEKLY VALUE '1'.				
88 MONTHLY VALUE '2'.				
10 NON-EXEMPT-WAGES	8	72	-	79
REDEFINES EXEMPT-WAGES.				
15 PAY-RATE PIC 9(4)V99.	6	72	-	77
15 JOB-CLASS PIC X(1).	1	78	-	78
15 FILLER PIC X.	1	79	-	79

图表 3.8　COBOL copybook 中的 REDFINES 子句

因为一个雇员要么是全职的的要么是按时计薪的，不可能两者都是，这两个字段集同时只有一个用到。全职工资字段占据文件中从 72 到 79 的位置，而按时计薪工资字段占据从 72 到 78 的位置。此外，注意到全职和按时计薪工资即使在相同的位置也使用不同的数据类型。当读取雇员记录时，程序必须决定如何基于位置 71 的 EMPLOYEE-TYPE 的值来解释这些位置的信息。

相同的位置可能同时有多个 REDEFINES 与之相关，所以有多于两个用法的可能，相同的位置可以有两个、三个或更多可能的用法。REDEFINES 引入了一种更加复杂的状况，所

以仅仅有 EBCDIC 到 ASCII 字符集转换是不够的。

当遇到多 REDEFINES 源时，应该考虑对源数据定义每个独立传输的抽取逻辑，如果后续处理差别很大的话（如全职工资和按时计薪工资的例子）。这允许为每个抽取建立独立的代码而不是为两种情况设计一个包含许多种情况的复杂逻辑。

多重 OCCUR 子句

主机和 COBOL 先于关系数据库和 Edward Codd 的规格化规则。先于使用关系理论来设计数据库，主机 COBOL 程序使用 OCCURS 子句来处理重复分组，方法是在一个数据文件中定义重复的数据字段。比如，在图 3.9 你可以看到存储关于绩效等级信息的雇员记录的区域。这条记录设计用来对多达 5 个绩效等级进行跟踪。但并不是创建 5 次需要的字段(记住，这先于关系理论，因此这里没有一个带外关键字并指向雇员表的独立的绩效等级表)它们只在一个特殊 OCCURS 字段中命名一次。这个 OCCURS 子句表示这个字段重复的次数值。本质上，OCCURS 子句在文件中定义了一个数组。因此，在雇员记录中，第一个绩效等级的数据占据从 80 到 99 的位置，第二等级从 100 到 199，第三等级从 120 到 139，第四等级从 140 到 159，接着第五等级以及最后一个等级则从 160 到 179。

		长度	位置		
05 PERFORMANCE-RATING-AREA	PIC X(100).	100	80	-	179
05 PERFORMANCE-RATINGS		100	80	-	179
REDEFINES PERFORMANCE-RATINGS-AREA.					
07 PERFORMANCE-RATING	OCCURS 5 TIMES.	20	80	-	99
15 PERF-RATING	PIC X(3).	3	80	-	82
15 REVIEWER-ID	PIC X(9).	9	83	-	91
15 REVIEW-DATE.		8	92	-	99
20 REVIEW-DATE-YYYY	PIC 9(4).	4	92	-	95
20 REVIEW-DATE-MM	PIC 9(2).	2	96	-	97
20 REVIEW-DATE-DD	PIC 9(2).	2	98	-	99

图表 3.9　带有 OCCURS 子句的 COBOL copybook 在数据记录中定义重复组

在大多数情形下，ETL 处理必须对主机文件中的 OCCURS 部分的任何数据进行规格化。即使这可能可能用手工编码的 ETL 过程来管理重复数据，但在这里还是强烈建议你使用成熟的 ETL 工具，它们允许你使用 COBOL copybook 来定义输入或至少允许你以某种方式手动定义输入文件数组。如果你的工具不支持输入数组，你只能坚持费力地写代码来处理来自旧主机系统的源记录中的重复分组。

有时程序员使用 OCCURS 来存储不同的事实在一个数组中，而不是存储同一个事实 N 次。比如，假设 O-DATE 发生 4 次。第一个日期是 CREATE（创建），第二个是 SHIP（运输），第三个是 ACKNOWLEDGE（确认），第四个是 PAYMENT（付款）。因此，这时候你不必规格化这个 OCCURS 数据，只要为数组中的每个位置创建独立字段即可。

为确保数据集成，模型数据结果来自一个 COBOL OCCURS 子句的规格化方法(一个主表和一个子表)在数据仓库的集结区。在独立的表中集结这些数据是一个好办法，因为这个过程的结果很可能是加载数据到一个事实表和一个维表、两个独立维表或两个独立事实表中。我们发现在这种情况下，将数据集成到数据仓库之前把它安置下来是很有意义的。

管理多主机记录类型文件

多记录类型文件的概念在讨论 REDEFINES 的部分有简单涉及。前面讨论的 REDEFINES 和我们现在介绍的概念的主要区别是，多记录类型文件不是记录的一部分包含多个定义，而是整个记录有多个定义。多记录类型通常将一个单独的逻辑记录跨越存储在 2 个或更多的物理记录上。图 3.10 包含一个 COBOL copybook 的扩展，说明重定义整个记录的概念。

在图 3.10 中，REDEFINES 子句应用到整个记录。因此现在文件不仅仅包含一个雇员的基本信息，同时也包括一个雇员在这个公司的工作经历。在这个文件中，每一个雇员至少有两个记录：一个是 EMP-RECORD，另一个是 JOB-RECORD。当一个雇员换了新工作，文件增加一个新的 JOB-RECORD。因此一个雇员的全部工作经历包含在这个文件的两个或更多记录中：一个 EMP-RECORD 和一个或多个 JOB-RECORD。

		长度	位置		
01 EMP-RECORD.		70	1	-	70
05 REC-TYPE	PIC X(1).	1	1	-	1
05 FIRST-NAME	PIC X(10).	10	2	-	11
05 MIDDLE-INITIAL	PIC X.	1	12	-	12
05 LAST-NAME	PIC X(15).	15	13	-	27
05 SSN	PIC X(9).	9	28	-	36
05 EMP-DOB.		8	37	-	44
10 DOB-YYYY	PIC 9(4).	4	37	-	40
10 DOB-MM	PIC 9(2).	2	41	-	42
10 DOB-DD	PIC 9(2).	2	43	-	44
05 EMP-ID	PIC X(8).	8	45	-	52
05 HIRE-DATE.		8	53	-	60
10 HIRE-YYYY	PIC 9(4).	4	53	-	56
10 HIRE-MM	PIC 9(2).	2	57	-	58
10 HIRE-DD	PIC 9(2).	2	59	-	60
05 TERM-DATE.		8	61	-	68
10 TERM-YYYY	PIC 9(4).	4	61	-	64
10 TERM-MM	PIC 9(2).	2	65	-	66
10 TERM-DD	PIC 9(2).	2	67	-	68
05 TERM-REASON_CODE	PIC X(2).	2	69	-	70
01 JOB-RECORD REDEFINES EMP-RECORD.		70	1	-	70
05 REC-TYPE	PIC X(1).	1	1	-	1
05 DIVISION	PIC X(8).	8	2	-	9
05 DEPT-ID	PIC X(3).	3	10	-	12
05 DEPT-NAME	PIC X(10).	10	13	-	22
05 JOB-ID	PIC X(3).	3	23	-	25
05 JOB-TITLE	PIC X(10).	10	26	-	35
05 START-DATE.		8	36	-	43
10 START-YYYY	PIC 9(4).	4	36	-	39
10 START-MM	PIC 9(2).	2	40	-	41
10 START-DD	PIC 9(2).	2	42	-	43
05 END-DATE.		8	44	-	51
10 END-YYYY	PIC 9(4).	4	44	-	47
10 END-MM	PIC 9(2).	2	48	-	49
10 END-DD	PIC 9(2).	2	50	-	51
05 FILLER	PIC X(19).	19	52	-	70

图表 3.10　在同一文件中重新组织多个记录类型

在这个文件中，记录的物理顺序是至关重要的，因为 JOB-RECORD 没有任何信息来连接它们到相应的 EMP-RECORD 上。一个雇员的 JOB-RECORD 紧跟在其 EMP-RECORD 之

后。因此要精确的处理一个雇员的工作经历，你必须把两个或更多物理上相邻的记录当作一个逻辑记录。

使用多记录类型的好处是它再次节省了空间。另一个选择(没有使用关系理论)是用一个相当宽的、浪费空间的记录来存储所有的数据再同一个记录中。假设，例如想跟踪到前 5 个职位的工作经历，则必须向每个雇员记录增加 255 个字节（基本 EMP-RECORD，加上 5 次出现的 JOB-RECORD 字段（5×51 字节））。但工作经历字段的数量是不定的－－它取决于一个雇员保持多少个工作。

通过使用多记录类型，主机系统可以仅仅存储需要的工作经历记录，因此只有一个工作的雇员只有一个 JOB-RECORD(70 字节包括 FILLER)，在这个文件中节省了 185 字节。此外，也不再限制文件中的工作数量。每个雇员可以增加无限的 70 字节 JOB-RECORD 的个数。

雇员例子只有两个记录类型，但多 REDEFINES 可以用来创建任何数量的记录类型，这些可以组合在一个单一逻辑记录中。如果我们展开这个雇员例子，你可能想到第三个记录类型用来存储雇员的边际利益和第四个类型存储雇员的家庭成员数。

ETL 过程必须通过在内存变量中保持来自第一个物理记录的值在一个集中(它只是逻辑记录的第一部分)来管理多记录类型，这样就可以和后续记录的其它数据进行关联。

处理主机变化记录长度

在前面部分，我们讨论了如何使用多记录类型将一个单一实体跨越存储到两个或更多记录的方法。一个可变的记录长度是另一种用在主机文件存储位置信息的方法。它不是用独立的 JOB-RECORD 来存储一个雇员的工作经历，而是每个工作都存储在一个 OCCURS 工作经历段中。此外，如图 3.11 所示，记录的这些段没有固定数量的段数，段的数量在 0 到 20 之间变化，这基于在 DEPENDING ON JOB-HISTORY-COUNT 子句中的数字值。每次增加雇员工作(直到最大值 20)，记录长度都会增加 50 字节。

		长度	逻辑位置		
01 EMP-RECORD.		70	1	-	70
05 REC-TYPE	PIC X(1).	1	1	-	1
05 FIRST-NAME	PIC X(10).	10	2	-	11
05 MIDDLE-INITIAL	PIC X.	1	12	-	12
05 LAST-NAME	PIC X(15).	15	13	-	27
05 SSN	PIC X(9).	9	28	-	36
05 EMP-DOB.		8	37	-	44
10 DOB-YYYY	PIC 9(4).	4	37	-	40
10 DOB-MM	PIC 9(2).	2	41	-	42
10 DOB-DD	PIC 9(2).	2	43	-	44
05 EMP-ID	PIC X(8).	8	45	-	52
05 HIRE-DATE.		8	53	-	60
10 HIRE-YYYY	PIC 9(4).	4	53	-	56
10 HIRE-MM	PIC 9(2).	2	57	-	58
10 HIRE-DD	PIC 9(2).	2	59	-	60
05 TERM-DATE.		8	61	-	68
10 TERM-YYYY	PIC 9(4).	4	61	-	64
10 TERM-MM	PIC 9(2).	2	65	-	66
10 TERM-DD	PIC 9(2).	2	67	-	68
05 TERM-REASON_CODE	PIC X(2).	2	69	-	70
05 JOB-HISTORY-COUNT	PIC 9(2).	2	71	-	72
05 JOB-HISTORY OCCURS 20 TIMES DEPENDING ON JOB-HISTORY-COUNT.		varies	73	-	varies
05 DIVISION	PIC X(8).	8	73	-	80
05 DEPT-ID	PIC X(3).	3	81	-	83
05 DEPT-NAME	PIC X(10).	10	84	-	93
05 JOB-ID	PIC X(3).	3	94	-	96
05 JOB-TITLE	PIC X(10).	10	97	-	106
05 START-DATE.		8	107	-	106
10 START-YYYY	PIC 9(4).	4	107	-	110
10 START-MM	PIC 9(2).	2	111	-	112
10 START-DD	PIC 9(2).	2	113	-	114
05 END-DATE.		8	115	-	114
10 END-YYYY	PIC 9(4).	4	115	-	118
10 END-MM	PIC 9(2).	2	119	-	120
10 END-DD	PIC 9(2).	2	121	-	122

图表 3.11　使用 DEPENDING ON 表示的 COBOL copybook 中的变长记录

在 copybook 中使用 DEPENDING ON 子句的可变长记录使直接的 EBCDIC 到 ASCII 字符集转换变得无效。以下列出一些可以减少在 ETL 过程由于可变长记录引起的数据破坏的风险的方法。

- 在主机上转换所有数据到显示格式并转成固定长度记录，为所有不用可变段的每个记录的末尾增加空格。
- 以二进制格式传输文件到数据仓库平台。这种技术需要有工具可以解释本章所讨论的所有的主机数据格式的细微差别。功能强大的专用 ETL 工具可以处理大多数或所有这些情况。如果你的数据仓库项目没有这样的工具，各种价格范围的第三方的应用程序都可以用来解释和转换主机数据到 UNIX 或 Windows 平台。
- 最后一个选择是开发自己的代码来处理所有已知的、在处理旧数据时可能遇到数据格式问题。然而，这个选择一定要慎重，处理所有可能的数据场景的时间和精力的付出可能超过在主机上开发重新格式化程序或购买一个实用程序来处理主机数据的成本。

从 IMS, IDMS, Adabase 和 Model 204 抽取

如果使用任何以上这些系统，则需要特殊的抽取器。对于初学者，这些中的每一个都可

以使用 ODBC 通道。详细讨论从这些旧数据库系统抽取的技术超出了本书的范围，但是它们可能对某些人是很重要的。

平面文件

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

平面文件广泛应用于任何数据集结应用系统。在大多数数据仓库环境中，你不可能避免平面文件。ETL 过程利用平面文件至少有三个原因：

- 传输源数据。当数据源在主机系统或外部系统中时，以平面文件 FTP 到数据集结区是很平常的。来自个人数据库或电子表格的数据常常也通过平面文件传输。
- 工作表/集结表。工作表是 ETL 过程为自己使用专门创建的。最常见的情况是，使用平面文件是因为直接从文件系统读写的 I/O 速度远远快于插入和查询 DBMS 系统。
- 块加载准备。如果 ETL 工具不支持数据流程中的块加载，或是想为安全或归档原因装载一个文件，在所有的数据转换完成后，你需要在文件系统上创建一个平面文件。一旦创建了这个平面文件，则块加载过程就可以读取这个文件并加载它到数据仓库中。

不是所有的平面文件都是相同的。平面文件基本上分为两种：

- 固定长度的
- 分隔符分隔的

处理固定长度平面文件

有时，不能从原始系统得到数据仓库需要的物理数据。在这种情况下，需要一个由支持源系统的程序员创建的平面文件。通常，这些文件是固定长度的——也称之为基于位置的平面文件。有时我们身处数据仓库项目中，它需要来自遗留源系统并且需要非常复杂计算的数据。计算的名称已经给了数据仓库架构师，但数据源及其计算规则却相当不清晰。在一段耗费时间的调查和研究工作之后，在 1977 年用 COBOL 写出了一份已发现的字段的报告。自然而然的，负责写这份报告的工程师离开了公司。此外，再也找不到源代码。这样这些计算就不再复杂，因为它们根本就不存在。

不幸的是，不能告诉业务用户说这些数据不能提供，并且在数据仓库中是不可用的。小组的解决方案是重定向这个包含所需数据的为输出平面文件，并且把来自报告的预生成的数据作为数据源。由于这个报告的自然属性——固定宽度列——而可以简单的把它当作一个基于位置的平面文件并在后续的 ETL 过程中处理。

处理固定长度平面文件需要布局文件来准确描述文件中的字段，如图 3.12 所示。固定长度布局文件应包含文件名称、字段开始位置、字段长度、字段数据类型（一般为文本或数字）。有时，结束位置也要提供，因为如果没有的话，就必须基于开始位置和长度来计算每个字段的结束位置，而这在 ETL 工具中是必须的。(如 Pervasive 的 Structured Schema Designer)。

字段名称	长度	开始	结束	类型	注释
记录类型	2	1	2	AlphaNumeric	记录类型，如用“H”表示标题，或者用“D”表示详细记录
SSN	9	3	11	Numeric	雇员的社会保险号
名字	20	12	31	AlphaNumeric	雇员的名字
中间名	1	32	32	AlphaNumeric	雇员的中间名
姓	20	33	52	AlphaNumeric	雇员的姓
名字前缀	5	53	57	AlphaNumeric	Jr. Sr. 等等
生日	8	58	65	Numeric	雇员的生日，用“YYYYMMDD”表示
状态编码	6	66	71	Numeric	雇员的状态（“A”，“R”，“T”等）
分公司编码	2	72	73	Numeric	雇员分公司编码
部门编码	2	74	75	Numeric	雇员在分公司的部门编码
位置编码	2	76	77	Numeric	雇员在组织结构中的位置编码
筛选	1	78	78	AlphaNumeric	筛选空格
添加日期	8	79	86	Numeric	添加记录到系统的日期
修改日期	8	87	94	Numeric	记录最后一次修改日期

图表 3.12　固定长度平面文件显示

大多数 ETL 工具很可能必须手工一次性输入平面文件的布局文件。输入布局之后，工具会记住这个布局并在每次访问实际的平面文件时使用相同的布局。如果文件布局改变了或数据超出了原来分配的位置，ETL 处理程序会失败。不幸的是，不像 XML，当你处理固定长度平面文件时不会发生隐含的布局文件有效性检验——在数据处理之前明确的预处理测试必须成功通过。

当处理固定长度平面文件时，尝试检验文件中数据的位置是否是正确的。一种快速检验位置有效性的办法是测试任意日期（或时间）字段以确认其是一个有效的日期。如果产生了移位，日期字段很可能包含希腊字母或不合逻辑的数字。其它具有非常特定含义的字段也可以用同样的方法来测试。XML 提供更加具体的有效性检验功能。如果数据校验或一致性是一个问题，试着说服数据提供者以 XML 格式提交数据。

基于位置的平面文件一般在文件系统中以.TXT 作为后缀。然而，基于位置的平面文件实际可以用任何文件扩展名——或根本没有——都可以以同样的方式处理。

处理有分隔符的平面文件

平面文件一般用一套分隔符分割文件中的数据字段。分隔符代替了使用位置来说明字段的开始和结束位置。分隔符文件可以用任意符号或符号组来分割平面文件的字段。最常用的分隔符是逗号。逗号分隔符文件常常以.CSV 作为文件扩展名。然而很显然，其它特定应用程序的分隔符平面文件可能简单的有个.TXT 扩展名或根本没有扩展名。

大部分 ETL 工具都有分隔符文件向导，一旦开发者指定了实际的分隔符号，就会扫描平面文件或它的样本，来检验文件中的分隔符和指定布局文件。通常，分隔符文件的第一行包含列名称。ETL 工具应该可以识别第一行中的列名称并在元数据层分配逻辑列名称，在接下来的后续数据处理中则忽略这一行。

和基于位置的平面文件一样，分隔符文件也没有隐含的校验方法。需要 ETL 小组进行明确的校验测试并将其加入数据处理例行测试中。

XML 数据源

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

扩展标志语言（XML）慢慢的但的确正在成为共享数据的标准。这个以规则格式文档存储数据的新规范已经产生了许多令人振奋的消息。在它的大力推广之后，我们现在认为本质上所有的数据仓库源都应该包含 XML。但至今为止，我们已经看到共享因特网数据的方法并没有改变多少。另一方面，共享外部数据的方法在过去几年已经完全成熟或开始几乎完全是 XML。

XML 已经显示出成为企业之间交换数据的通用语言。如果你的数据仓库包括来自外部源的数据－－那些来自企业之外的－－可能那些源都将以 XML 提供。

要处理 XML，首先必须清楚它的工作原理。XML 包含 2 个重要的元素：它的元数据和数据本身。XML 元数据可以用多种方式提供。以下的部分说明不同格式的 XML 元数据以及这些对 ETL 意味着什么。

字符集

字符集是用来显示和打印计算机输出的唯一符号组。大多数关系型数据库管理系统的默认字符集是 ISO8859-15 (Latin 9)。这个字符集通过启用欧元符号：€ 代替了 ISO8859-1 (Latin 1) ，这个拉丁字符集用在西方国家来支持基于英语字母表的语言。然而，由于 XML 是主要用于互联网的语言，它必须支持全世界的语言和字母表，而不仅仅是西方国家的。因此，XML 支持 UTF-8 字符集。UTF-8 是一种保留基本 ASCII 编码方法并且也支持统一字符编码（ISO10646）的通用字符集(UCS)。UTF-8 支持来自全世界的大多数语言和字母表。

许多问题的产生是由于源 XML 文档和目标数据仓库基于不同的字符集造成的。当然，这种不同步的字符集在集成独立的系统(不仅仅是 XML 数据集)时总是存在危险。但在绝大多数情况下，这种风险是极小的，因为数据库系统几乎毫无例外的使用拉丁字符集。不使用基于拉丁字符集而是使用统一字符编码来支持特殊字母表或字符的组织，应该采用一个企业范围的标准字符集来避免集成的难度。无论何时遇到集成数据的需求，尤其使用 XML 集成来自外部源的数据，就必须做好处理不同的字符集的准备。比较好的一点是 XML 至少可以通过适当的元数据来标记文档，以此来指定使用的字符集。例如，标签<?xml version=”1.0” encoding=”UTF-8” ?> 表示这个 XML 文档编码采用 UTF-8 字符集。

XML 元数据

我们常常听到有人说 XML 从存放数据角度来讲没有比平面文件强什么。在我们看来，确实是这样的。。唯一使得 XML 和平面文件有点类似的地方是它们都存储在文件系统而不是数据库中。并且事实上，许多数据库系统增加了读取、创建和存储 XML 的能力，称之为内部嵌入 XML。

XML 是一个特殊的实体，因为它存储数据但又被看作一种语言。它不是一个应用程序，因此，它依赖于其它应用程序来使它工作。然而它不仅仅是数据因为它内嵌了标签。XML

文档中的标签是为什么 XML 如此功能强大的原因。但是自定义数据也带来了一定的成本。标签，包含它的元数据，可能占据 XML 文件大小的 90%，只留下大约 10%的 XML 文件是实际的数据。如果你当前的文件是 XML，它们可能 10 倍于它们初始的大小——即正好存储同样行数的数据。

具有讽刺意味的是，数据仓库的主要目标是保持数据尽可能的收缩以使得处理的尽可能的快，虽然 XML 标签会引起背离这个目标的大量数据处理，但是许多人还是坚持使它成为一个数据交换的标准。标签不仅仅增加数据文件的大小，而且它们还增加了相当的复杂度。由于这些固有的复杂性，绝对不要试图写自己的 XML 文档界面来解析 XML 文档。XML 文档的结构相当棘手，并且 XML 解析器的构造是它本身要解决的问题——而不是由数据仓库小组来设计。市场上有许多 XML 解析器（或者处理器），并且现在大多数 ETL 厂商在他们的产品中也会包含它们。

不要试图手工解析 XML 文件。XML 文档需要一个 XML 处理器引擎来处理。许多主流的 ETL 工具现在都在他们的产品中包含 XML 处理器。确认所用的 ETL 工具确实具有它们声称的 XML 的处理能力。

要处理一个 XML 文档，必须首先知道这个文档的结构。一般 XML 文档的结构放在一个独立的文件中。下面这些部分将讨论可能伴随 XML 以及提供 XML 文档结构的每个可能的元数据文件。

DTD（文档类型定义）

正如那些将 XML 看作数据源而不是编程语言的人一样，我们将 DTD 等同于 COBOL 的布局文件。它是一个描述 XML 文档或文件的数据结构的文件。在 XML 文档中可以嵌入定义信息，但是为了有效性检验，应该让元数据和实际数据文件相互独立。DTD 可能相当复杂，包含以下这些允许的 XML 数据结构：

- 基本数据。如果一个元素必须只包含数据，它用标签#PCDATA 声明。
- 元素结构。在 DTD 中一个元素的结构是在一个元素中用元素名称列表来指定。比如，<!ELEMENT OrderLineItem (ProductID, QuantityOrdered,Price)>表示一个订单项（order line item）由产品编号（product ID）、订货量（quantity ordered）和订货时的产品价格组成。
- 混合的内容。当允许数据或元素中任意一个时，用 PCDATA 声明来表示允许基本数据及元素名来表示允许嵌套元素。
- 是否可为空。这不是一个类型！在 XML 中，你用’nill=’或’nillable=’标签来表示如果一个字段提否可以是 NULL。在 DTD 中，会发现用一个问号（？）来表示子元素是可选的。比如，代码<!ELEMENT Customer (FirstName, LastName,ZipCode?, Status)>表示名、姓以及状态是必须的，但邮政编码是可选的。
- 对应关系。用加号（＋）表示一对多。比如，<!ELEMENT Customer (FirstName, LastName,ZipCode+, Status)>表示客户可以有多个邮政编码。
- 允许的值。类似于约束性检查，XML 通过列出用竖线（|）分隔的可接受的值来指定允许的值。比如，<!ELEMENT State (Alabama|Louisiana|Mississippi)>表示州必须包含 Alabama、Louisiana 或 Mississippi。

在写这本书时候，XML 规范仍然在修订和改变中。今天，DTD 多数被当作过时的技术。因为 XML 正在发展更多的数据中心角色，类似于关系型数据库，大多数的 DTD 将被当作

标准元数据的 XML 结构所取代。下一部分将讨论 XML 结构以及它们与 DTD 之间的不同。

XML 结构

XML 结构是 DTD 的继任者。XML 结构比 DTD 更加丰富和实用，因为它们是设计用来扩展 DTD 的。XML 结构允许直接定义一个 SQL CREATE TABLE 语句。这在简单的 DTD 中是绝不可能的，因为在 DTD 中没法指定详细的数据类型和字段长度。XML 结构的一些特征包括：

- 出现在 XML 文档中的元素
- 出现在 XML 文档中的属性
- 子元素的数量和顺序
- 元素和属性的数据类型
- 元素和属性的默认值和固定值
- 添加的可扩展性
- 支持命名空间

命名空间

XML 越来越来普遍的原因是它强制独立的数据源产生一致的和可预期的数据文件。但实际是不同的系统中同一个实体的元素在意义和用法上总是有些微小的差别。比如，如果得到一个同时来自人力资源系统和交易系统的客户文件，它们可以对这个客户有不同的定义。一个部门可能有些交易与组织进行，同时又与个人进行其它交易。虽然两者都是客户，但组织和个人的属性差别还是很大的。为了减少这种情况，XML 文档可以引用命名空间。命名空间指示到哪里取得元素或属性的定义。同一个实体可以基于它声明的命名空间有不同的意思。同一个客户实体，带有标签<Customer xmlns=http://www.website.com/xml/HRns>命名空间与引用<Customer xmlns=http://www.website.com/xml/OPSns>的同一个实体具有不同的含义。

因为 XML 对基于 Web 的应用程序是新兴的数据源，这将很可能对数据仓库数据源产生越来越多的影响。当选择 ETL 工具时，确认它可以天然处理 XML 和 XML 结构。

Web 日志数据源

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

实际上世界上几乎每个公司都有自己的网站。每个网站之下也都有日志－－网络日志－－记录每个对象从 Web 服务器上传或下载的服务。网络日志非常重要，因为它们揭示了用户在网站上的交互情况。

这里的网络日志不是 weblog 或 blog！我们的网络日志指的是每一个 Web 服务器自动产生的控制文档。blog（博客）是一种由个人发布和维护的日记，主要是青少年，所有人都可以阅读它们。

理解的网站上的客户行为就像跟着一个客户逛商店并记录他或她的一举一动一样有价值。想象一下，如果知道客户在逛你的商店时的一举一动，则将可以更好的组织你的商店，并知道有多少机会可以销售他们更多的商品。网络日志提供了类似的信息。分析网络日志的活动并存储结果在数据集市来分析客户行为被称之为点击流数据仓库。

交叉参考：点击流数据仓库的更多信息请参考一本优秀的图书《Clickstream Data Warehousing》，作者是 Mark Sweiger, Mark R. Madsen, Jimmy Langston, and Howard Lombard，Wiley 出版社 2002 年出版。

从数据建模的角度看，点击流数据集市可能相对数据仓库中的其它主题没有什么挑战。然而，ETL 过程与其它可能遇到的数据源有很大的不同。差别在于点击流的源是基于文本日志的，这些日志又必须和其它源系统集成。还好，基于文本日志的格式是标准化的。这个标准由世界网络协会（W3C，World Wide Web Consortium）维护。

W3C 通用和扩展格式

即使网络日志的格式是标准化的，它的格式和内容仍然有很多不同。Web 服务器的操作系统和控制日志内容的参数设置会影响写入日志的内容。如果不考虑操作系统，网络日志有一些通用的字段，一般包括：

- 日期。这个字段有一个通用的日期格式－－一般为 dd/mm/yyyy。如果在 ETL 过程中时区被调整了，则必须合并日期和时间并一起做调整；否则，可能会有一天的误差。可以在加载时再把它们分开。
- 时间。这是网络点击的时间。格式是 HH:MM:SS 并且一般设置为格林威治标准时间（GMT）。然而，时区也可以更改。确定知道 Web 服务器设置的时区，这通常也包括为数据集市把时间转换成本地时间。
- c-ip。这是用户的互联网服务提供商（ISP）的 IP 地址。它是一个标准的 IP 地址，可以用来为域名系统（DNS）查找并推算用户来自哪里。使用 c-ip 几乎没有什么可信度，因为众所周知它是没有规则的，比如 AOL（拥有最大量用户的 ISP）给上百万的用户同一个 IP 地址并显示所有他们的用户都来自美国的同一个州，即使他们可能实际上来自全世界各个角落。
- 服务名。这涉及到运行在客户计算机上的互联网服务，比如，w3svc1, w3svc2, w3svc3 等等。这个字段识别来自多网站或域环境的站点和日志。这个字段对单一站点环境通常设为关闭。
- s-ip。这是服务器的 IP 地址。它是标准的 IP 地址格式。在一个多网络服务器的环境中用来识别某个 Web 服务器是很有用的。它也可以用来做负载平衡分析。
- cs-method。这个字段中只有两个可能值：POST 或 GET。一般只有 GET 的记录存储在点击流数据集市中。
- cs-uri-stem。这是资源可访问性（即 HTML 或 ASP 页面请求）。
- cs-uri-query。这是客户发来的查询。这个字段包含高度客户化的且非常有价值的数据。我们把这个和 cookie（稍后将讨论）称为网络日志的重要信息。这个字段用

label=value 表示并用符号（&）作分隔符，当然也可以用其它的符号。Cs-uri-query 的更多分析在本章的稍后的”名称值对”部分讨论。

- sc-status。这是 HTTP 状态，比如，302（重定向），402（错误），及 200（ok）。HTTP 状态代码的完全列表可以在网络上搜索 HTTP 状态代码来查找。我们建议预先加载所有可能的代码和描述到 HTTP 状态维。
- sc-bytes。这是服务器发送的字节数，一般当作点击的一个事实。
- cs（用户代理）。这是客户使用的浏览器类型和版本。用户代理以及日期和时间可以用来唯一的确定一名访问者。

更多关于标识网站唯一用户的信息，可以参考《数据仓库工具箱：建立基于 web 的数据仓库》，作者 Ralph Kimball and Richard Merz ，Wiley 出版社 2000 年出版。

- cs（Cookie）。这是 cookie 发送或接收的任意内容。和 cs-uri-query 一起，这个字段是 web 日志中的另一种重要信息。Cookie 是高度可自定义的且非常有价值。它可以清楚的识别用户以及关于用户会话的许多其它特征。
- cs（参考）。这是指引用户访问当前站点的网站的 URL 说明。

以下的字段出现在 W3C 扩展格式中，但这不是全部的列表。如果想要了解环境中所有字段的列表，可以参考 web 服务器文档或 W3C 网站（www.w3c.org）。

- 服务器名称。这是日志产生的服务器的名称。这与 web 服务器的 IP 地址 s-ip 应该是一一对应的。
- Cs-用户名称。这是用户名称并且只有当方法为 POST 时才有值。
- 服务器端口。这是客户端连接的端口号。
- 接收字节。这是用户接收的字节数目。
- 消耗时间。这是这个动作消耗的时间。
- 协议版本。这是客户端使用的协议版本，比如 HTTP 1.0, HTTP 1.1。

Web 日志中的名称值对

Web 日志由各种不同内容的标准字段组成。对于其中最主要的部分，无需太多的转换逻辑就可以导出 web 日志的内容。尤其对于日期、时间、c-ip、服务名称、s-ip、cs-method、cs-uri-stem、sc-status 和 sc-bytes 来说更是这样。然而，例如 cs-uri-query 和 cs（cookie）这样的字段根本就不是标准的。事实上，两个不相关的网站在这些字段中几乎不可能有相同的内容。cs-uri-query 和 cs（cookie）包含自定义的名称值对来获取一个交互的特定属性，这些交互对业务来说很重要。

cs-uri-query 一般包含关于交互的详细信息，比如页面上提供的产品。请看下面这个 cs-uri-query 例子：

/product/product.asp?p=27717&c=163&s=dress+shirt

看一看这个查询字符串并注意以下的部分：

- /product/－－查询字符串的开始部分表示要执行的程序所在的目录。目录总是在前面和后面加一个斜杠’/’。在这个例子中，product.asp 程序在产品目录下。如果程序在根目录下，在程序名称前加一个单斜杠。
- Product.asp－－这是产生 web 页面的可执行程序文件。通用的可执行程序包含扩展名：.asp(动态服务页面)或.jsp（java 服务页面）。程序文件紧跟在目录后面。

- ？——问号表示后面的参数要传递给程序文件。在这个例子中，有三个参数：p，c 和 s。

在问号之前的查询字符串是标准的，问号后面是程序的自定义参数的位置。网站开发者可以为每个程序文件定义不同的参数集合。在 web 日志的名称值对中可以获得这些参数。在这个例子中，你可以看到三个参数，之间用符号（&）分隔。

- p 表示产品号码
- c 表示产品类别号码
- s 表示用户输入用来查找产品的搜索字符串

符号（&）是 web 日志中最通用的参数分隔符，但并不是唯一的。确认在分析阶段中扫描日志确实是可视化的，保证可以识别参数分隔符。

注意到在 s=参数 中搜索字符串写着 dress＋shirt。实际上，用户输入的是 dress shirt。由于 HTTP 不能处理空格，因此这个+是 web 浏览器自动插入的。当处理查询字符串中的文本型描述信息时必须非常小心。在用它们来查找或存储到数据仓库中之前，ETL 过程必须将文本型描述中的任何＋都用空格来代替。

这一部分的目标是揭示 web 日志的信息，给点击流数据集市项目开个头。再次强调，如果要深入了解或规划在不久的将来实施一个点击流项目，则应该重点关注这个部分前面提到的两本书中的任何一本。

ERP 系统数据源

流程检查

规划与设计：*需求/现状* -> *架构* -> 实现 -> 测试/发布

正如本节所述，ERP 系统的存在给 ETL 系统的规划和设计带来很大的影响。包括通过各种方式将 ERP 系统当作一个简单的固定长度的数据源，到把 ERP 系统当作包含所有组件(包括 ERP 系统)的数据仓库。我们在这一部分给出相应的建议。

数据流：抽取 -> 清洗 -> 规格化（也许） -> 提交

企业资源计划（ERP）系统的在建设时也面临着数据仓库正在面临的问题之，即集成异构的数据。ERP 系统设计为企业集成的解决方案，它使得企业的每个主要实体，比如销售、会计、人力资源、库存和产品控制都在同一个平台、数据库和应用框架上来完成。

和想象的一样，ERP 系统是非常复杂并且难以实施，往往需要数月或数年时间来定制开发，因此它包含了详细的功能来满足运行一个特殊业务的所有需求。但正如不可能提供一个最完整的解决方案一样，虽然进行过很多努力，仍极少看到整个企业只用一个 ERP 系统来运作整个公司。

ERP 系统的复杂性人人皆知，而且由于它实际上是一个框架而不是一个应用程序，因此其数据模型往往很丰富，通常包含数以千计的表。此外，由于它们的灵活性，支持 ERP 处理的数据模型非常难以确定。市场上主流的 ERP 系统包括 SAP、PeopleSoft、Oracle、Baan 和 J.D.Edwards 等。

由于由无数的表和属性组成，以及通常 ERP 实施的复杂度，因此把这些系统和任何其它交易源系统一样进行利用是错误的。随便进行系统和数据分析是费时费力的，并且为以后出错带来了更多的可能。如果数据仓库的数据源是一个已存在的 ERP 系统，最好是获得对这个特定 ERP 系统的底层数据库结构非常了解且以及对应用的业务对象具有丰富经验的人

的帮助。

作为帮助，许多主流 ETL 厂商现在都提供 ERP 适配器来连接主流 ERP 系统。如果以 ERP 系统为数据源，那么就应该充分利用这些适配器。它们可以帮助你探索这些系统的元数据并更好地理解这个应用。

对 SAP 的特殊考虑

因为 SAP 在 ERP 市场的支配地位，经常有人问起在数据仓库中关于 SAP 角色的问题。以下我们的最原始的建议。

你可能已经听过这样的观点，从决策支持的角度来看，SAP ERP 像一个黑洞：数据流进去了，但没法将信息取出来。为什么？

一个 SAP ERP 实施好像是有一个包含上万张物理表的数据基础，几乎没有任何 DBMS 定义的表间关系，并且实体和属性名称采用的是德文缩写的！因此，SAP RDBMS 对实际的实施来说，是专用的和不可理解的。SAP ERP 产生一个广泛的操作型报表库，但这些一般难以满足大多数业务部门全面的决策支持的需求。这并不是设计的缺陷。SAP 的 OLTP 数据结构对基本业务报表需求缺乏支持，比如历史交易和主要数据情况的查询，可理解的和简单的数据结构浏览，以及灵活的查询执行功能。

一些早期实施了 SAP 的企业试图在它们的数据仓库中创建 ERP 主题域来让它们的 ERP 中的操作型数据保持独立。实事证明，几乎没有特定的工具可以帮助它们完成这个难以实现的任务，许多努力并没有获得显著的效果。

为了满足这一不断增长的需求，SAP 设计了一个 ERP 自身的决策支持的扩展，叫做业务信息仓库（SAP BW）。SAP BW 的早期版本相当简单，并且主要由 SAP 特定的 ETL 提供的专用 OLAP 资料库组成，因此缺少许多同期的数据仓库所拥有的基本结构元素。最新发布的 SAP BW 已经有了显著的增强，现在已经包含了许多同期数据仓库的核心原则和结构：更好的支持非 SAP 数据源以及主流数据资料库（集结区，ODS，数据仓库，维度数据集市和 OLAP 立方体）。这些资料库中的一部分支持向第三方报表工具的开放访问。

SAP BW 提供了有吸引力的价格和，以此来吸引 CIO 们的注意力。与此同时，同期的数据仓库架构师可能被要求考虑在企业数据仓库视图中 SAP BW 的角色。

在下表中，我们介绍了在整个企业 DW 战略中针对不同的 SAP BW 角色的 pros，cons 和建议。我们认识到这是一个快速发展的领域，会有许多变化，这其中几乎没有完全满足所有需求的解决方案。BW 和 ETL 工具的功能将逐步改变，但我们还是希望我们的评估过程对你有用并且可延伸到特定的问题中。

BW 作为企业数据仓库	利用 SAP BW 作为企业数据仓库策略的基本核心
	我们不推荐使用 BW 作这个角色。使用 BW 作为一个企业 BI 结构的中央元件在目前结构是是不稳固的。正如这里所写的，BW 没有为非 SAP 数据的清洗/集成或非 SAP 分析的发布提供相应的能力。打包 BI 方案也趋向于减少通过分析能力比较不同的机会。尽管如此，那些没有把 BI 当作一个比较不同策略的领域并且他们的报表需要以 SAP 为中心并对其评价很高的组织仍可能考虑这样使用 SAP BW。
BW 在维度数据仓库总线	利用 SAP BW 作为一组数据集市的集合，这些数据集市在广泛分布的维度数据仓库总线结构中进行相应的操作。

	虽然这可能有一天会成为利用 BW 的一种合适的方式，我们发现（正如这里所写的）BW 并不完全适合在一个广泛的维度数据仓库总线结构中当作 ERP 中心数据集市角色。当我们写这本书的时候，SAP 宣布了一个叫做 Master Data Management 的功能，声称可以处理产品说明和客户的交叉描述问题功能。这种独特的功能能够创建企业级一致的维度还是合并外部一致的维度到 ETL 过程并且应用到相应的事实表，还不好判断。利用跨越主题域的一致维度和事实是维度数据仓库总线结构的核心宗旨。尽管如此，已经拥有了维度数据仓库和 BW 的 IT 组织可能会选择扩展 BW 来利用外部一致维度这一任务，这样就允许它在总线结构即插即用。警告：通过 SAP 升级维护这些扩展可能不是一个简单的任务。
ETL 和集结区	利用 SAP BW 作为将 ERP 数据导入下游维度数据仓库的通道和集结区。
	这个选项刚出现的时候相当引人注目，因为它的目的是简化和减少工作量来将 ERP 数据导入维度数据仓库，并且同时也为 SAP 专用报表提供标准 BW 报表能力。但是当将这个选项与利用带有很好的 SAP 适配器（见下一节“放弃 BW”）的专用第三方 ETL 工具这个选择相比，它可能就不再是最佳方案了。用 SAP 作为数据源，通过 ETL 工具为数据仓库导出大量附加结果到表集合，一般可以提供更好的可控性、灵活性和元数据的一致性。我们建议这个在 SAP 中的 ETL 系统的实施选项只有当组织有不成熟的 ETL 小组并且时间很紧近且需要出 ERP 报表时才选择，谁能在容量狭小的 SAP BW 报表中发现价值很高的信息。不如认真寻找带有专用 SAP 适配器的成熟的 ETL 工具。见下一节“放弃 BW”。
放弃 BW	利用大多数优秀的 ETL 工具提供的 SAP 适配器来执行完全集成 ERP 主题域到独立的企业维度数据仓库总线结构中。
	我们绝对相信购买胜过自己做，这是非常合适的。但是考虑到 SAP 在创建或发布一致维度或利用外部一致维度方面跟踪记录的缺陷，这是我们建议的默认 BW 结构形式。

第 3 部分：抽取变化数据

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

在初始化加载时，捕获源数据中的数据内容的变化不是很重要，因为很可能会导出整个数据源或其中的一部分。然而一旦初始加载完成，捕获源系统中的数据变化立即变成非常重要的任务。如果等到初始加载完成才开始规划数据变化的捕获技术，则将遇到很大的麻烦，捕获数据变化绝不是一个简单的任务，必须制定相应的策略来在项目中捕获源数据中不断增长的变化。

ETL 小组在后续的加载过程中负责捕获数据内容的变化。需求由用户提出，而这些需求的实际可行性由源系统的 DBA 小组决定（如果运气好的话）。更多的时候并不是这样，需要针对特定的情形做一些研究来决定最有可能的增量加载策略。在这一节，我们提供几种选项并讨论每一种的优缺点。当然，不必为每一种情形选择所有这些技术。选择在整个项目过程中最能满足每个 ETL 需求的方法。

为识别源系统中变化的数据制定适当的策略可能要进行一些分析工作。当分析源系统时，绝对不要假设所见即所得。在许多案例中，还有许多未加使用的甚至是禁止使用的审计列，或更糟糕一些的是完全不一致的列用法。确保为增量加载过程分配了足够的分析时间来调查和决定捕获数据内容变化的最佳方法。

监测变化

当管理者谈论数据仓库维护时，他们最经常讨论的是保持数据最新，也就是数据真正反映了公司业务系统的状态。捕获源系统内容的变化对一个数据仓库的成功至关重要，数据内容的维护则依赖于增量加载过程。有几种捕获源数据变化的方法，每一种在它们相应的环境中都是有效的。

使用审计列

在大多数情形下，源系统都包含审计列。审计列附在每个表的最后用来存储记录增加或修改的日期和时间。审计列一般通过数据库触发器产生，当插入或更新记录时自动生成。有时，为了提高性能，这些列由前端应用程序而不是由数据库触发器产生。当这些字段不是由数据库触发器以外的任意其它方法加载时，必须要特别注意它们的完整性。必须分析并测试每个列以确保其是表示变化数据的可靠的数据源。如果发现任何 NULL 值，则必须另寻替代办法来监测数据变化。

阻止 ETL 过程使用审计列的最常见情形是当这些字段是由前端应用程序产生的并且 DBA 小组允许后台脚本修改数据时。如果是这种情形，则将在增量加载过程中面临很高的错过变化数据的风险。最小化这一风险防范措施之一是规定所有的后台脚本由一个质量保证小组来验证，在验收之前坚持并测试由脚本产生的审计字段。

一旦确信审计列是可靠的，则需要制定一个策略来利用它们。有各种各样利用审计列来捕获变化数据的方法，并且所有的方法都拥有同样的逻辑目标：比较每个记录最后修改的日期和时间与上次加载后的最后的日期和时间，取它们中较晚的值。

我们已经发现一个利用源系统中审计列的有效方法。实际上，这个过程就是从创建日期和最后修改日期列中选择最后日期和时间。有些最后修改列是随着插入或更新记录的变化而修改的，其它最后修改列在记录插入时是 NULL，只有当这个记录更新时才插入相应日期值。当最后修改日期没有填充时，为了不丢失新记录必须给定任意一个非常早的日期作为默认值。以下代码可以帮助解决 NULL 修改日期：

```
select max(greatest(nvl(create_date,'01-JAN-0001'),

nvl(last_mod_date,'01-JAN-0001')))
```

对事实表中的行只插入不更新的情形，可以简单的从源系统选择创建日期和时间大于上次加载的最后日期和时间来实现，从而忽略最后修改日期列。

由于事实表和维表可以来源于许多不同的表和系统，并且由于事实表只包含外键和度量，因此不能直接存储审计日期列到事实表中。在每次导出的时候，需要建立一个 ETL 的最后修改表来捕获每个源表和在源系统审计列找到的最大日期。如果事实表需要对行数进行审计统计，考虑使用第 4 章所讲述的建立一个审计维的方法。

数据库日志的获取和提取

日志获取是在调度点（一般在深夜）对数据库重做日志进行有效的快照并把它作为对

ETL 加载过程中所关心的表产生影响的数据源。提取包含重做日志的登记，并从中捕获相应事务数据。抓取日志可能是所有技术中最笨的。事务日志经常会溢出，这意味日志被写满并阻止新事务的产生。当这种溢出发生在一个生产环境时，负责的 DBA 最快的反应会是清空日志的内容以使业务操作重新恢复。但是当日志清空时，里面所有的事务也都丢失了。如果已经用尽了所有的其它技术，然后发现日志抓取是为寻找新的或变化的记录的最后求助的手段，那么就试图说服 DBA 创建一个特殊的日志来满足这一特殊需要。大概只需要源数据库成百张表中很少的某些交易表，当插入或更新这些表时触发动作写入专用日志中。

如果想继续日志提取，我们建议调查市场上可用的 ETL 工具来寻找验证有效的方案而不是试图从抓取开始手工写这个处理过程。许多实时 ETL 方案提供商都利用了日志提取技术。

按时抽取

选择所有创建或修改的日期等于 SYSDATE-1 的行，即获得昨天的所有记录。看起来很不错，对吗？其实是错误的。纯粹的按照时间来加载记录是大多数刚开始进行 ETL 开发的人常犯的错误，这种处理方式极其不可靠。

当从中间处理失败后重启时，基于时间的数据选择会加载重复的行。这意味着无论什么原因使处理失败都需要人工干预和数据清洗。其间，如果某天的加载处理没有运行并错过了一天，存在的风险是错过那天的数据将永远不会进入数据仓库中。因此除非 ETL 处理非常简单并且数据量特别小，否则不要纯粹基于时间加载数据。

排除处理

排除处理在集结区保存了每个上次导出数据副本为将来所用。在下一次运行过程中，整个源表被拿到集结区与最后一次处理留下的数据副本进行比较。只有不同（增量）部分会送到数据仓库。虽然这不是最有效的技术，但排除处理是所有捕获变化数据的增量加载技术中最可靠的。因为这个处理进行了逐行的比较并查找出变化的行，它原则上不可能遗漏任何数据。这种技术对找到那些从源数据中删除的行也有帮助，而这些删除的行往往被其它的技术遗漏掉。

这种技术可以在数据库管理系统的内部或外部完成。如果选择使用 DBMS，则为了提高效率必须成批的将数据加载到集结数据库来处理。

初始和增量加载

创建两个表：previous_load 和 current_load。

初始化处理批量加载到 current_load 表。因为初始加载过程中变化检测是无关的，因此数据连续被转换并加载到最终目标事实表中。

当这个处理完成后，就删除这个 previous_load 表，并把 current_load 表重命名为 previous_load 表，然后创建一个空的 current_load 表。由于这些任务都没有涉及数据库日志，因此速度会非常快！

下一次加载过程运行时，这个 current_load 表就填充了数据。

选择 current_load 表 MINUS（减去）previous_load 表，转换并加载结果集到数据仓库中。

完成后删除 previous_load 表并再次重命名 current_load 表为 previous_load。最后，创建一个空的 current_load 表。

由于在数据库管理系统中 MINUS 是非常慢的技术，因此最好使用 ETL 工具或第三方应用程序来执行例行的排除处理。

抽取的技巧

当进行抽取处理时考虑以下几点：

- 强制列索引。和 DBA 一起工作来确保源系统中所有在 WHERE 语句中的列都有索引，否则将可能引起对整个生产数据库的全表扫描。
- 获取需要的数据。优化的查询会正确返回需要的数据。不应该获取整个表，然后在 ETL 工具中再过滤掉不想要的数据。有一种情况可能不遵守这个规则，就是交易系统的 DBA 拒绝对你的查询需要限制返回行的列索引时。另一个例外是当必须下载整个源数据表来查询增量时。
- 谨慎使用 DISTINCT。DISTINCT 语句非常慢。在抽取查询中执行 DISTINCT 和在 ETL 工具中对结果进行聚合或分组两种方式如何平衡是一个挑战，一般会随着数据源中重复百分比的不同而不同。因为有许多其它因素可以影响这种决定，所有我们所能建议的就是谨慎的测试每个策略，从而找出最有效的结果。
- 谨慎使用 SET 操作符。UNION，MINUS 和 INTERSECT 都是 SET 操作符。这些和 DISTINCT 一样都是非常慢的。可以理解的是有时这些操作符是不可避免的。一个技巧是使用 UNION ALL 而不要使用 UNION。UNION 执行和 DISTINCT 相当，处理缓慢，而 UNION ALL 的缺点是会返回重复行，因此都要谨慎处理。
- 根据需要使用 HINT。大多数数据库支持 HINT 关键字。可以使用 HINT 做许多事情，但最重要的是强制查询使用常规的索引，尤其当使用 IN 或 OR 操作符时这个功能尤其重要，因为这时往往会进行全表扫描而不是使用索引，即使有可用的索引存在。
- 避免 NOT。如果有可能，尽量避免不等于的限制和连接。无论使用关键字 NOT 还是操作符‘<>’，数据库此时也会选择扫描整个表而不是利用索引。
- 避免在 where 子句中使用函数。这是很难避免的一种，尤其是当包含日期和类似日期的处理时。在犯下在 WHERE 语句中使用函数的错误前，先要用不同的技术进行试验。无论如何尽量尝试使用比较关键字而不是函数。比如：

  LIKE 'J%' 而不是 SUBSTR('LAST_NAME',1,1) = 'J'

  EFF_DATE BETWEEN '01-JAN-2002' AND '31-JAN-2002' 而不是

  TO_CHAR(EFF_DATE, 'YYY-MON') = '2002-JAN'

抽取查询的目的是获得所有相关的自然键和度量。它可能和从一个表选择多个列那么简单，也可能和实际创建不存在的数据那样复杂，范围可能从很少几个表的关联到不同数据源的许多表的关联。在一个特定项目中，我们必须建立一个周期性的快照事实表来表示库存中每个产品的销量，即使在这个期间没有销售任何产品。我们不得不产生一个产品列表，获得所有产品的销售，并执行一个产品列表和产品销售之间的外关联，没有销售的产品的销售量默认为 0。

监测数据源中删除或覆盖的事实记录

如果没有删除或覆盖事件发生的通知机制，对于源系统中删除或覆盖了的度量（事实）记录如何发现可能会给数据仓库带来非常大的挑战。由于一般不可能重复抽取旧的交易记录，关于如何查找这些被删除的和修改的数据，我们能提供最好的方法如下：

- 和源系统的所有者商讨，如果可能，明确的通知所有删除或覆盖的度量。
- 周期性的检查来自源系统的度量的历史汇总并通知 ETL 人员有些内容变化了。当发现变化时，尽可能的钻取下去找到这个变化。

当识别出一个删除或修改的度量记录时，就可以使用前面所述的最后返回数据技术了。对于删除或修改事实记录的情形，不仅在数据仓库中执行删除或更新，我们还推荐插入一条新的记录来实现在事实表中消除或抵消初始放的值。在许多应用中，这将汇总报表事实得到正确的数量（如果它是添加的）并提供一种纠错发生时的审计跟踪功能。在这些情形下，它可能也方便的携带额外的管理时间戳来识别这些数据库行为发生的时间。

总结

在本章，我们深入探讨了 ETL 数据流中的抽取这一步骤。我们建议再回到刚开始的地方来确认抽取是非常有价值的部分！可以通过一个数据评估工具来决定做或不做这一工作，工具将告诉你是否数据质量满足了业务目标。

下一个大的步骤是准备连接到初始源数据到最终数据的逻辑数据映射。也许逻辑数据映射最重要的部分是应用在输入和输出之间的转换规则描述。当实际执行 ETL 系统时将会发生一些变化，因此应该经常检查并定期更新逻辑数据映射。如果维护的好，它可能是 ETL 系统中最有价值的描述。有时，如果由一个新人来了解做过的东西，他或她应该从逻辑数据映射开始。

本章的中心是带你了解将来可能遇到的各种各样的源系统。其中一些抽取复杂度是相当高的，当然，没有什么比实际经验更有价值了。

本章的最后部分介绍了只导出新数据、变化的数据甚至是已经删除的数据的方法。在后续的章节，我们将指出在这些情况下必须采取的特殊处理措施。