ETL本质

ETL本质

做数据仓库系统，ETL是关键的一环。说大了，ETL是数据整合解决方案，说小了，就是倒数据的工具。回忆一下工作这么些年来，处理数据迁移、转换的工作倒还真的不少。但是那些工作基本上是一次性工作或者很小数据量，使用access、DTS或是自己编个小程序搞定。可是在数据仓库系统中，ETL上升到了一定的理论高度，和原来小打小闹的工具使用不同了。究竟什么不同，从名字上就可以看到，人家已经将倒数据的过程分成3个步骤，E、T、L分别代表抽取、转换和装载。

其实ETL过程就是数据流动的过程，从不同的数据源流向不同的目标数据。但在数据仓库中，ETL有几个特点，一是数据同步，它不是一次性倒完数据就拉到，它是经常性的活动，按照固定周期运行的，甚至现在还有人提出了实时ETL的概念。二是数据量，一般都是巨大的，值得你将数据流动的过程拆分成E、T和L。

现在有很多成熟的工具提供ETL功能，例如datastage、powermart等，且不说他们的好坏。从应用角度来说，ETL的过程其实不是非常复杂，这些工具给数据仓库工程带来和很大的便利性，特别是开发的便利和维护的便利。但另一方面，开发人员容易迷失在这些工具中。举个例子，VB是一种非常简单的语言并且也是非常易用的编程工具，上手特别快，但是真正VB的高手有多少？微软设计的产品通常有个原则是“将使用者当作傻瓜”，在这个原则下，微软的东西确实非常好用，但是对于开发者，如果你自己也将自己当作傻瓜，那就真的傻了。ETL工具也是一样，这些工具为我们提供图形化界面，让我们将主要的精力放在规则上，以期提高开发效率。从使用效果来说，确实使用这些工具能够非常快速地构建一个job来处理某个数据，不过从整体来看，并不见得他的整体效率会高多少。问题主要不是出在工具上，而是在设计、开发人员上。他们迷失在工具中，没有去探求ETL的本质。

可以说这些工具应用了这么长时间，在这么多项目、环境中应用，它必然有它成功之处，它必定体现了ETL的本质。如果我们不透过表面这些工具的简单使用去看它背后蕴涵的思想，最终我们作出来的东西也就是一个个独立的job，将他们整合起来仍然有巨大的工作量。大家都知道“理论与实践相结合”，如果在一个领域有所超越，必须要在理论水平上达到一定的高度

探求ETL本质之一

ETL的过程就是数据流动的过程，从不同异构数据源流向统一的目标数据。其间，数据的抽取、清洗、转换和装载形成串行或并行的过程。ETL的核心还是在于T这个过程，也就是转换，而抽取和装载一般可以作为转换的输入和输出，或者，它们作为一个单独的部件，其复杂度没有转换部件高。和OLTP系统中不同，那里充满这单条记录的insert、update和select等操作，ETL过程一般都是批量操作，例如它的装载多采用批量装载工具，一般都是DBMS系统自身附带的工具，例如Oracle SQLLoader和DB2的autoloader等。

ETL本身有一些特点，在一些工具中都有体现，下面以datastage和powermart举例来说。

1、静态的ETL单元和动态的ETL单元实例；一次转换指明了某种格式的数据如何格式化成另一种格式的数据，对于数据源的物理形式在设计时可以不用指定，它可以在运行时，当这个ETL单元创建一个实例时才指定。对于静态和动态的ETL单元，Datastage没有严格区分，它的一个Job就是实现这个功能，在早期版本，一个Job同时不能运行两次，所以一个Job相当于一个实例，在后期版本，它支持multiple instances，而且还不是默认选项。Powermart中将这两个概念加以区分，静态的叫做Mapping，动态运行时叫做Session。

2、ETL元数据；元数据是描述数据的数据，他的含义非常广泛，这里仅指ETL的元数据。主要包括每次转换前后的数据结构和转换的规则。ETL元数据还包括形式参数的管理，形式参数的ETL单元定义的参数，相对还有实参，它是运行时指定的参数，实参不在元数据管理范围之内。

3、数据流程的控制；要有可视化的流程编辑工具，提供流程定义和流程监控功能。流程调度的最小单位是ETL 单元实例，ETL单元是不能在细分的ETL过程，当然这由开发者来控制，例如可以将抽取、转换放在一个ETL单元中，那样这个抽取和转换只能同时运行，而如果将他们分作两个单元，可以分别运行，这有利于错误恢复操作。当然，ETL单元究竟应该细分到什么程度应该依据具体应用来看，目前还没有找到很好的细分策略。比如，我们可以规定将装载一个表的功能作为一个ETL单元，但是不可否认，这样的ETL单元之间会有很多共同的操作，例如两个单元共用一个Hash 表，要将这个Hash表装入内存两次。 4、转换规则的定义方法；提供函数集提供常用规则方法，提供规则定义语言描述规则。

5、对数据的快速索引；一般都是利用Hash技术，将参照关系表提前装入内存，在转换时查找这个hash表。Datastage中有Hash文件技术，Powermart也有类似的Lookup功能。

探求ETL本质之二（分类）

昨在IT-Director上阅读一篇报告，关于ETL产品分类的。一般来说，我们眼中的ETL工具都是价格昂贵，能够处理海量数据的家伙，但是这是其中的一种。它可以分成4种，针对不同的需求，主要是从转换规则的复杂度和数据量大小来看。它们包括

1、交互式运行环境，你可以指定数据源、目标数据，指定规则，立马ETL。这种交互式的操作无疑非常方便，但是只能适合小数据量和复杂度不高的ETL过程，因为一旦规则复杂了，可能需要语言级的描述，不能简简单单拖拖拽拽就可以的。还有数据量的问题，这种交互式必然建立在解释型语言基础上，另外他的灵活性必然要牺牲一定的性能为代价。所以如果要处理海量数据的话，每次读取一条记录，每次对规则进行解释执行，每次在写入一条记录，这对性能影响是非常大的。

2、专门编码型的，它提供了一个基于某种语言的程序框架，你可以不必将编程精力放在一些周边的功能上，例如读文件功能、写数据库的功能，而将精力主要放在规则的实现上面。这种近似手工代码的性能肯定是没话说，除非你的编程技巧不过关（这也是不可忽视的因素之一）。对于处理大数据量，处理复杂转换逻辑，这种方式的ETL实现是非常直观的。

3、代码生成器型的，它就像是一个ETL代码生成器，提供简单的图形化界面操作，让你拖拖拽拽将转换规则都设定好，其实他的后台都是生成基于某种语言的程序，要运行这个ETL过程，必须要编译才行。Datastage就是类似这样的产品，设计好的job必须要编译，这避免了每次转换的解释执行，但是不知道它生成的中间语言是什么。以前我设计的ETL工具大挪移其实也是归属于这一类，它提供了界面让用户编写规则，最后生成C++语言，编译后即可运行。这类工具的特点就是要在界面上下狠功夫，必须让用户轻松定义一个ETL过程，提供丰富的插件来完成读、写和转换函数。大挪移在这方面就太弱了，规则必须手写，而且要写成标准c++语法，这未免还是有点难为最终用户了，还不如做成一个专业编码型的产品呢。另外一点，这类工具必须提供面向专家应用的功能，因为它不可能考虑到所有的转换规则和所有的读写，一方面提供插件接口来让第三方编写特定的插件，另一方面还有提供特定语言来实现高级功能。例如Datastage提供一种类Basic的语言，不过他的Job的脚本化实现好像就做的不太好，只能手工绘制job，而不能编程实现Job。

4、最后还有一种类型叫做数据集线器，顾名思义，他就是像Hub一样地工作。将这种类型分出来和上面几种分类在标准上有所差异，上面三种更多指ETL实现的方法，此类主要从数据处理角度。目前有一些产品属于EAI（Enterprise Application Integration），它的数据集成主要是一种准实时性。所以这类产品就像Hub一样，不断接收各种异构数据源来的数据，经过处理，在实施发送到不同的目标数据中去。

虽然，这些类看似各又千秋，特别在BI项目中，面对海量数据的ETL时，中间两种的选择就开始了，在选择过程中，必须要考虑到开发效率、维护方面、性能、学习曲线、人员技能等各方面因素，当然还有最重要也是最现实的因素就是客户的意象。

探求ETL本质之三（转换）

ETL探求之一中提到，ETL过程最复杂的部分就是T，这个转换过程，T过程究竟有哪些类型呢？

一、宏观输入输出

从对数据源的整个宏观处理分，看看一个ETL过程的输入输出，可以分成下面几类： 1、大小交，这种处理在数据清洗过程是常见了，例如从数据源到ODS阶段，如果数据仓库采用维度建模，而且维度基本采用代理键的话，必然存在代码到此键值的转换。如果用SQL实现，必然需要将一个大表和一堆小表都Join起来，当然如果使用ETL工具的话，一般都是先将小表读入内存中再处理。这种情况，输出数据的粒度和大表一样。

2、大大交，大表和大表之间关联也是一个重要的课题，当然其中要有一个主表，在逻辑上，应当是主表Left Join辅表。大表之间的关联存在最大的问题就是性能和稳定性，对于海量数据来说，必须有优化的方法来处理他们的关联，另外，对于大数据的处理无疑会占用太多的系统资源，出错的几率非常大，如何做到有效错误恢复也是个问题。对于这种情况，我们建议还是尽量将大表拆分成适度的稍小一点的表，形成大小交的类型。这类情况的输出数据粒度和主表一样。

3、站着进来，躺着出去。事务系统中为了提高系统灵活性和扩展性，很多信息放在代码表中维护，所以它的“事实表”就是一种窄表，而在数据仓库中，通常要进行宽化，从行变成列，所以称这种处理情况叫做“站着进来，躺着出去”。大家对Decode肯定不陌生，这是进行宽表化常见的手段之一。窄表变宽表的过程主要体现在对窄表中那个代码字段的操作。这种情况，窄表是输入，宽表是输出，宽表的粒度必定要比窄表粗一些，就粗在那个代码字段上。

4、聚集。数据仓库中重要的任务就是沉淀数据，聚集是必不可少的操作，它是粗化数据粒度的过程。聚集本身其实很简单，就是类似SQL中Group by的操作，选取特定字段（维度），对度量字段再使用某种聚集函数。但是对于大数据量情况下，聚集算法的优化仍是探究的一个课题。例如是直接使用SQL的 Group by，还是先排序，在处理。

二、微观规则

从数据的转换的微观细节分，可以分成下面的几个基本类型，当然还有一些复杂的组合情况，例如先运算，在参照转换的规则，这种基于基本类型组合的情况就不在此列了。ETL的规则是依赖目标数据的，目标数据有多少字段，就有多少条规则。

1、直接映射，原来是什么就是什么，原封不动照搬过来，对这样的规则，如果数据源字段和目标字段长度或精度不符，需要特别注意看是否真的可以直接映射还是需要做一些简单运算。 2、字段运算，数据源的一个或多个字段进行数学运算得到的目标字段，这种规则一般对数值型字段而言。 3、参照转换，在转换中通常要用数据源的一个或多个字段作为Key，去一个关联数组中去搜索特定值，而且应该只能得到唯一值。这个关联数组使用Hash算法实现是比较合适也是最常见的，在整个ETL开始之前，它就装入内存，对性能提高的帮助非常大。

4、字符串处理，从数据源某个字符串字段中经常可以获取特定信息，例如身份证号。而且，经常会有数值型值以字符串形式体现。对字符串的操作通常有类型转换、字符串截取等。但是由于字符类型字段的随意性也造成了脏数据的隐患，所以在处理这种规则的时候，一定要加上异常处理。 5、空值判断，对于空值的处理是数据仓库中一个常见问题，是将它作为脏数据还是作为特定一种维成员？这恐怕还要看应用的情况，也是需要进一步探求的。但是无论怎样，对于可能有NULL值的字段，不要采用“直接映射”的规则类型，必须对空值进行判断，目前我们的建议是将它转换成特定的值。

6、日期转换，在数据仓库中日期值一般都会有特定的，不同于日期类型值的表示方法，例如使用8位整型20040801表示日期。而在数据源中，这种字段基本都是日期类型的，所以对于这样的规则，需要一些共通函数来处理将日期转换为8位日期值、6位月份值等。

7、日期运算，基于日期，我们通常会计算日差、月差、时长等。一般数据库提供的日期运算函数都是基于日期型的，而在数据仓库中采用特定类型来表示日期的话，必须有一套自己的日期运算函数集。

8、聚集运算，对于事实表中的度量字段，他们通常是通过数据源一个或多个字段运用聚集函数得来的，这些聚集函数为SQL标准中，包括sum,count,avg,min,max。

9、既定取值，这种规则和以上各种类型规则的差别就在于它不依赖于数据源字段，对目标字段取一个固定的或是依赖系统的值。

探求ETL本质之四（数据质量）

“不要绝对的数据准确，但要知道为什么不准确。”

这是我们在构建BI系统是对数据准确性的要求。确实，对绝对的数据准确谁也没有把握，不仅是系统集成商，包括客户也是无法确定。准确的东西需要一个标准，但首先要保证这个标准是准确的，至少现在还没有这样一个标准。客户会提出一个相对标准，例如将你的OLAP数据结果和报表结果对比。虽然这是一种不太公平的比较，你也只好认了吧。

首先在数据源那里，已经很难保证数据质量了，这一点也是事实。在这一层有哪些可能原因导致数据质量问题？可以分为下面几类：

1、数据格式错误，例如缺失数据、数据值超出范围或是数据格式非法等。要知道对于同样处理大数据量的数据源系统，他们通常会舍弃一些数据库自身的检查机制，例如字段约束等。他们尽可能将数据检查在入库前保证，但是这一点是很难确保的。这类情况诸如身份证号码、手机号、非日期类型的日期字段等。

2、数据一致性，同样，数据源系统为了性能的考虑，会在一定程度上舍弃外键约束，这通常会导致数据不一致。例如在帐务表中会出现一个用户表中没有的用户ID，在例如有些代码在代码表中找不到等。

3、业务逻辑的合理性，这一点很难说对与错。通常，数据源系统的设计并不是非常严谨，例如让用户开户日期晚于用户销户日期都是有可能发生的，一个用户表中存在多个用户ID也是有可能发生的。对这种情况，有什么办法吗？

构建一个BI系统，要做到完全理解数据源系统根本就是不可能的。特别是数据源系统在交付后，有更多维护人员的即兴发挥，那更是要花大量的时间去寻找原因。以前曾经争辩过设计人员对规则描述的问题，有人提出要在ETL开始之前务必将所有的规则弄得一清二楚。我并不同意这样的意见，倒是认为在ETL过程要有处理这些质量有问题数据的保证。一定要正面这些脏数据，是丢弃还是处理，无法逃避。如果没有质量保证，那么在这个过程中，错误会逐渐放大，抛开数据源质量问题，我们再来看看ETL过程中哪些因素对数据准确性产生重大影响。

1、规则描述错误。上面提到对设计人员对数据源系统理解的不充分，导致规则理解错误，这是一方面。另一方面，是规则的描述，如果无二义性地描述规则也是要探求的一个课题。规则是依附于目标字段的，在探求之三中，提到规则的分类。但是规则总不能总是用文字描述，必须有严格的数学表达方式。我甚至想过，如果设计人员能够使用某种规则语言来描述，那么我们的ETL单元就可以自动生成、同步，省去很多手工操作了。

2、ETL开发错误。即时规则很明确，ETL开发的过程中也会发生一些错误，例如逻辑错误、书写错误等。例如对于一个分段值，开区间闭区间是需要指定的，但是常常开发人员没注意，一个大于等于号写成大于号就导致数据错误。

3、人为处理错误。在整体ETL流程没有完成之前，为了图省事，通常会手工运行ETL过程，这其中一个重大的问题就是你不会按照正常流程去运行了，而是按照自己的理解去运行，发生的错误可能是误删了数据、重复装载数据等。

探求ETL本质之五（质量保证）

上回提到ETL数据质量问题，这是无法根治的，只能采取特定的手段去尽量避免，而且必须要定义出度量方法来衡量数据的质量是好还是坏。对于数据源的质量，客户对此应该更加关心，如果在这个源头不能保证比较干净的数据，那么后面的分析功能的可信度也都成问题。数据源系统也在不断进化过程中，客户的操作也在逐渐规范中，BI系统也同样如此。本文探讨一下对数据源质量和ETL处理质量的应对方法。

如何应对数据源的质量问题？记得在onteldatastage列表中也讨论过一个话题 - "-1的处理"，在数据仓库模型维表中，通常有一条-1记录，表示“未知”，这个未知含义可广了，任何可能出错的数据，NULL数据甚至是规则没有涵盖到的数据，都转成-1。这是一种处理脏数据的方法，但这也是一种掩盖事实的方法。就好像写一个函数FileOpen(filename)，返回一个错误码，当然，你可以只返回一种错误码，如-1，但这是一种不好的设计，对于调用者来说，他需要依据这个错误码进行某些判断，例如是文件不存在，还是读取权限不够，都有相应的处理逻辑。数据仓库中也是一样，所以，建议将不同的数据质量类型处理结果分别转换成不同的值，譬如，在转换后，-1表示参照不上，-2表示NULL数据等。不过这仅仅对付了上回提到的第一类错误，数据格式错误。对于数据一致性和业务逻辑合理性问题，这仍有待探求。但这里有一个原则就是“必须在数据仓库中反应数据源的质量”。

对于ETL过程中产生的质量问题，必须有保障手段。从以往的经验看，没有保障手段给实施人员带来麻烦重重。实施人员对于反复装载数据一定不会陌生，甚至是最后数据留到最后的Cube，才发现了第一步ETL其实已经错了。这个保障手段就是数据验证机制，当然，它的目的是能够在ETL过程中监控数据质量，产生报警。这个模块要将实施人员当作是最终用户，可以说他们是数据验证机制的直接收益者。首先，必须有一个对质量的度量方法，什么是高质什么是低质，不能靠感官感觉，但这却是在没有度量方法条件下通常的做法。那经营分析系统来说，联通总部曾提出测试规范，这其实就是一种度量方法，例如指标的误差范围不能高于5%等，对系统本身来说其实必须要有这样的度量方法，先不要说这个度量方法是否科学。对于ETL数据处理质量，他的度量方法应该比联通总部测试规范定义的方法更要严格，因为他更多将BI系统看作一个黑盒子，从数据源到展现的数据误差允许一定的误差。而ETL数据处理质量度量是一种白盒的度量，要注重每一步过程。因此理论上，要求输入输出的指标应该完全一致。但是我们必须正面完全一致只是理想，对于有误差的数据，必须找到原因。

在质量度量方法的前提下，就可以建立一个数据验证框架。此框架依据总量、分量数据稽核方法，该方法在高的《数据仓库中的数据稽核技术》一文中已经指出。作为补充，下面提出几点功能上的建议：

1、提供前端。将开发实施人员当作用户，同样也要为之提供友好的用户界面。《稽核技术》一文中指出测试报告的形式，这种形式还是要依赖人为判断，在一堆数据中去找规律。到不如用OLAP的方式提供界面，不光是加上测试统计出来的指标结果，并且配合度量方法的计算。例如误差率，对于误差率为大于0的指标，就要好好查一下原因了。

2、提供框架。数据验证不是一次性工作，而是每次ETL过程中都必须做的。因此，必须有一个框架，自动化验证过程，并提供扩展手段，让实施人员能够增加验证范围。有了这样一个框架，其实它起到规范化操作的作用，开发实施人员可以将主要精力放在验证脚本的编写上，而不必过多关注验证如何融合到流程中，如何展现等工作。为此，要设计一套表，类似于DM表，每次验证结果数据都记录其中，并且自动触发多维分析的数据装载、发布等。这样，实施人员可以在每次装载，甚至在流程过程中就可以观察数据的误差率。特别是，如果数据仓库的模型能够统一起来，甚至数据验证脚本都可以确定下来，剩下的就是规范流程了。

3、规范流程。上回提到有一种ETL数据质量问题是由于人工处理导致的，其中最主要原因还是流程不规范。开发实施人员运行单独一个ETL单元是很方便的，虽然以前曾建议一个ETL单元必须是“可重入”的，这能够解决误删数据，重复装载数据问题。但要记住数据验证也是在流程当中，要让数据验证能够日常运作，就不要让实施者感觉到他的存在。总的来说，规范流程是提高实施效率的关键工作，这也是以后要继续探求的。

探求ETL本质之六（元数据漫谈）

对于元数据（Metadata）的定义到目前为止没有什么特别精彩的，这个概念非常广，一般都是这样定义，“元数据是描述数据的数据（Data about Data）”，这造成一种递归定义，就像问小强住在哪里，答，在旺财隔壁。按照这样的定义，元数据所描述的数据是什么呢？还是元数据。这样就可能有元元元...元数据。

我还听说过一种对元数据，如果说数据是一抽屉档案，那么元数据就是分类标签。那它和索引有什么区别？元数据体现是一种抽象，哲学家从古至今都在抽象这个世界，力图找到世界的本质。抽象不是一层关系，它是一种逐步由具体到一般的过程。例如我->男人->人->哺乳动物->生物这就是一个抽象过程，你要是在软件业混会发现这个例子很常见，面向对象方法就是这样一种抽象过程。它对世界中的事物、过程进行抽象，使用面向对象方法，构建一套对象模型。同样在面向对象方法中，类是对象的抽象，接口又是对类的抽象。因此，我认为可以将“元”和“抽象”换一下，叫抽象数据是不是好理解一些。 常听到这样的话，“xx领导的讲话高屋建瓴，给我们后面的工作指引了清晰的方向”，这个成语“高屋建瓴”，站在10楼往下到水，居高临下，能砸死人，这是指站在一定的高度看待事物，这个一定的高度就是指他有够“元”。在设计模式中，强调要对接口编程，就是说你不要处理这类对象和那类对象的交互，而要处理这个接口和那个接口的交互，先别管他们内部是怎么干的。

元数据存在的意义也在于此，虽然上面说了一通都撤到哲学上去，但这个词必须还是要结合软件设计中看，我不知道在别的领域是不是存在Metadata这样的叫法，虽然我相信别的领域必然有类似的东东。元数据的存在就是要做到在更高抽象一层设计软件。这肯定有好处，什么灵活性啊，扩展性啊，可维护性啊，都能得到提高，而且架构清晰，只是弯弯太多，要是从下往上看，太复杂了。很早以前，我曾看过backorifice的代码，我靠，一个简单的功能，从这个类转到父类，又转到父类，很不理解，为什么一个简单的功能不在一个类的方法中实现就拉到了呢？现在想想，还真不能这样，这虽然使代码容易看懂了，但是结构确实混乱的，那他只能干现在的事，如果有什么功能扩展，这些代码就废了。

我从98年刚工作时就开始接触元数据的概念，当时叫做元数据驱动的系统架构，后来在QiDSS中也用到这个概念构建QiNavigator，但是现在觉得元数据也没啥，不就是建一堆表描述界面的元素，再利用这些数据自动生成界面吗。到了数据仓库系统中，这个概念更强了，是数据仓库中一个重要的部分。但是至今，我还是认为这个概念过于玄乎，看不到实际的东西，市面上有一些元数据管理的东西，但是从应用情况就得知，用的不多。之所以玄乎，就是因为抽象层次没有分清楚，关键就是对于元数据的分类（这种分类就是一种抽象过程）和元数据的使用。你可以将元数据抽象成0和1，但是那样对你的业务有用吗？必须还得抽象到适合的程度，最后问题还是“度”。

数据仓库系统的元数据作用如何？还不就是使系统自动运转，易于管理吗？要做到这一步，可没必要将系统抽象到太极、两仪、八卦之类的，业界也曾定义过一些元数据规范，向CWM、XMI等等，可以借鉴，不过俺对此也是不精通的说，以后再说