可扩展数据库模型

# 表设计

## 一、行转列

以字符串存储各种数据类型，通过行转列实现实体属性读取。

以前提到的电子商务网站的商品实体为例，我们可以建立两个表“商品”和“商品属性”，商品表为普通的商品属性，可以将商品名称、价格等大部分商品的公共属性放到该表中。商品表与商品属性表形成一对多关系，商品属性表只需要定义商品“属性名”和“属性值”这两个属性用于保存一个商品的各个属性。

这样在每读取一个商品时，可以读取该商品的属性集合，然后将属性集合重新绑定到对象，将该对象暂时在页面上。

这种做法的优点是灵活，可以为商品创建无数个不同的属性，可以应对电商这种快速变化，快速上线的需求。缺点是后期做统计的时候会很慢，因为需要行转列，如果要涉及到各种Join查询之类的也会很麻烦。

## 二、冗余列

预定义大量的冗余列，根据用户对实体属性的类型设置匹配对应的列。

如果我们不希望行转列的话，那么可以预先定义好数据列，由于不确定是哪种数据类型，所以我们可以将表的列定义的特别多，每个不同的数据类型都定义几个或者十来个列，这些列都是允许为空的，如果没有使用已经预定义好的列，并不会占据多少数据空间。

这种数据库设计方法的优点是不会存在行转列的问题，所以在join或者出报表时性能较好，缺点就是使得一个表的列特别多，而且大部分列在大多数情况下是不使用的，而且扩展比较困难，比如我们要定义17个bit类型的列，但是系统默认只有16个，这种情况下，就需要在数据库中使用2行数据来表示1行列表数据。

## 三、XML存储

使用XML数据类型存储动态列数据。

XML数据类型是SQL的一个标准，目前主流的数据库都支持XML数据类型，数据库为XML提供专门的语法以快速检索和操作XML数据。在新版的SharePoint中，就使用XML来存储用户自定义列表的内容。

## 四、动态创建表

为用户定义的实体动态创建表。

还有一直动态方法是在程序中动态创建表，用户每在程序中定义一个实体的时候，就好根据用户定义创建一个对应的表。比如微软的Dynamic CRM就是这样实现的。用户可以在系统中创建大量的实体，并且还可以定义实体之间的关系，系统就会按照用户的定义创建对应的表，以及外键。

这种方法的优点是性能好，每个实体与其数据库表相对应，不存在大量的冗余列，也不会存在行转列的问题。缺点是开发难度大，对用户的要求高；而且在创建好实体并且存储了大量数据后，如果想要修改实体属性，那么将很困难。

## 五、复合字段

使用复杂字段。在某些特殊要求下还是很有用的， 比如，某个业务实体（某张表），有一些标记位，都是true/false 的标记，可以理解为这个实体的一些属性，经常需要添加，这种情 况，在生产中我们使用过**用一个数字**，按位来表示这些标记的，比如第三位表示他是不是付费用户，第四位表示他是合作方来的用户 还是自己注册的，等等。还有一种情况，需要更复杂的属性列表， 属性个数经常变，可以考虑使用**一个文本字段**，保存结构化的数据， 然后在使用时需要动态解析该字段。使用复杂字段的好处就在于比 较灵活，同一类型的数据可以放在一起（实际上相当于把应该是一 个关联表的数据放一个字段里了），操作的性能也不错，但是复杂 字段里面的内容查询比较困难，程序实现和修改起来也更复杂。

## 六、索引表

将数据的存储和索引（需要查询的内容）分开存放， 相当于主表就一个key value，把需要查询的字段放到其他单独的表里去，这种方法对数据的实时查询的效率有一定的影响，但是 针对实时性的要求并不是所有系统都有要求的，在有些地方是很好 的解决方案。

参考：

1. <http://www.cnblogs.com/studyzy/archive/2013/09/02/3297620.html>
2. <http://wenku.baidu.com/link?url=KEcBrUkZj6qaRzrhUlKfqkM8p-P-GhltcdescZcQrjwjJyjRBuAmdP-2IPp38aIRT40Nkg3tRz1svsCvuLYrTCIK_ZlECj8wM7dWGY8qPHq>

# 架构

## 集群

横向扩展，对程序端透明，多副本，负载均衡

### Share-Disk架构

　　Share-Disk架构是通过多个服务器节点共享一个存储来实现数据库集群。在此基础之上，Share-Disk架构又分为单活和双活，双活即为集群中的每一个节点都可以同时对外提供服务，而单活为集群中只有一个节点可对外提供服务，集群中的其他服务器作为冗余在活的节点出现故障时接替该服务器成为对外提供服务的节点。

　　这种方式的弊端也是显而易见的，如下：

* 硬件资源的严重浪费，同一时间集群中只有一台服务器活着，其他服务器只能作为冗余服务器。
* 集群无法提升性能，因为只有一台服务器可用。
* 存储方面存在单点故障，除非在存储层级保证高可用，通常需要昂贵的SAN存储。

　　因此该类方案仅仅可以做到服务器层面的高可用，无法带来性能的提升，也无法解决存储单点故障的问题。因此如果不搭配其他高可用或负载均衡的技术，存在的意义并不是很大。

　　另一类技术是Share-Disk中的双活的技术，与单活技术不同的是，双活的技术虽然也是共享磁盘，但集群中的所有节点都可以对外提供服务，典型的产品就是Oracle的RAC。RAC的技术性非常的高，因此需要水平比较高的人来运维系统。RAC设计的初衷并不是为了性能，而是为了高可用和可扩展性，如果应用程序不是针对RAC架构设计和开发的，则将应用程序迁移到RAC上会导致性能的急剧下降，并且节点越多性能下降越明显。

## Share-Nothing架构

　　Share-Nothing架构又分为两种，首先是**分布式架构**。将数据库中的数据按照某一标准分布到多台机器中，查询或插入时按照条件查询或插入对应的分区。另一种是每一个**节点完全独立**，节点之间通过网络连接，通常是通过光纤等专用网络。

在Share-Nothing架构中，每一个节点都拥有自己的内存和存储，都保留数据的完整副本。通常来说，又可以分为两种，**可以负载均衡和不可以负载均衡**。

　　首先谈谈不可负载均衡的集群，在不可负载均衡的技术中，集群中的节点会被分为主节点和辅助节点，主节点向外提供服务，辅助节点作为热备（二阶段事务提交）或暖备（不需要保证事务同步），同时有可能使得辅助节点提供只读的服务。

　　这种架构带来的好处包括：

* 辅助节点数据和主节点保持同步或准同步，当搭配第三方仲裁后，可以实现自动的故障转移，从而实现了高可用。
* 辅助节点由于和主节点完全独立且数据同步或准同步，因此主节点出现数据损坏后，可以从辅助节点恢复数据(自动或手动)。
* 由于Share-Nothing架构使用了本地存储（或SAN），相较于Share-Disk架构在慢速网络时有非常大的性能优势。

　　当然，弊端也显而易见，因为辅助节点无法对外提供服务或只能提供只读服务，因此该类集群的弊端包括：

* 扩展能力非常有限。
* 对性能没有提升，因为涉及到各节点的数据同步，甚至带来性能的下降。
* 辅助节点如果可读，虽然提升性能，但需要修改前端应用程序，对应用程序不透明。

另一类Share-Nothing架构中，是允许负载均衡的。所谓负载均衡就是将对数据库的负载分布到集群中的多个节点上，在集群中的每一个节点都可以对外提供服务，从而达到更高的吞吐量，更好的资源利用率和更低的响应时间。前端通过代理进行调度。可负载均衡的Share-Nothing架构的好处是每台服务器都能提供服务，能充分利用现有资源，达到更高的吞吐量。这类方案集群中的每个节点都会对外提供服务，因此有如下好处：

* 由于每一个节点都可以对外提供服务，因此可以提升性能。
* 扩展性得到提升，可以通过向集群添加节点直接进行Scale-Out扩充。
* 由于前端应用通过代理连接到集群，而集群中的每一个节点都保持完整的数据集，因此对应用程序端完全透明。

但该类方案的弊端也显而易见，因为每一个节点都需要完整的数据集，因此需要占用更多的存储空间。

参考

1. <http://server.yesky.com/datacenter/394/37513394.shtml>