● 软件技术

基于ORM的数据库设计与实现

张党进[1] 张建行[2]

（1. 电信科学技术第十研究所 陕西 西安 710061；2. 西安通信学院基础部 陕西 西安 710106）

摘 要： ORM是一种理论上比较完善的数据访问层解决方案。因为ORM核心是把数据对象与数据库二维表进行相互转换，所以，在设计数据库时要求考虑这两种不同数据结构的异同。结合Normalization和Denormalization两种数据库设计规范，从实体映射和关系映射角度出发，克服ORM自身的缺点，在项目开发中更好地使用ORM。

关键词： ORM Normalization规范 Denormalization规范 实体映射 关系映射

从ODBC到JDBC/ADO、Java、C/C++和C#等开发语言都对数据库操作进行了二次封装，提供数据库连接和标准SQL语句执行等接口。但是，随着软件开发的复杂化和企业级系统框架的出现，把数据库访问操作从业务层分离出来，独立成一层或者组件化，一是符合多层体系结构的设计要求，体现了分层给系统带来的可维护性和可扩展性等优点；二是减轻了应用开发人员的负荷，使其把更多的精力投入到视图层设计和系统业务逻辑功能实现等上来。

把对数据库操作从业务层分离出来，即数据访问层，有许多相关理论和成熟解决方案，其中ORM（Object Relation Mapping，对象关系映射）是一种比较完善的理论，并且与之相关的产品也比较成熟。ORM是把业务层中的数据对象和数据库表中的数据进行转换的一个“桥梁”。它基于数据对象和表之间的相似性，而数据对象和表之间的区别给这两种不同数据之间转换带来难度。虽然ORM理论上有许多解决方案，但是在项目开发中，因为不同的数据库设计规范，会带来不同的映射关系，所以，与数据库设计有很大关系。为了更好利用ORM来实现数据访问层设计，需要对数据库设计作相应的调整。

一、数据库设计规范

在数据库设计中，一般有两种规范方式，即：Normalization和Denormalization设计。

1. Normalization设计

关系模式Normalization设计的基本思想：通过对关系模式进行分解，用一组等价的关系子模块来代替原有的关系模式，消除数据依赖（包含函数依赖和多值依赖）中不合理的部分，使得一个关系仅描述一个实体或者实体之间的一种联系。这一过程必须在保证无损连接性、保持函数依赖性的前提下进行。即确保不破坏原来的数据，并可将分解的关系通过自然连接回复原来的关系。Normalization设计过程就是按照不同的范式，将一个二维表不断地分解成多个二维表并建立之间的关联，最终达到一个表只描述一个实体或者实体间的一种关联的目标。

Normalization设计的优点：包括有效的消除数据冗余，理顺数据的从属关系，保持数据库的完整性，增强数据库的稳定性、伸缩性、适应性等。其缺点：增加了查询时的连接表运算，导致了计算机在时间、空间、系统及其运行效率等方面的损失。

在一定程度上，按照Normalization设计数据库，逐步优化过程是把大数据表进行分解，然后通过关系关联起来，达到数据一致和减少冗余（这一点跟面向对象设计相同，对象设计就是细化对象颗粒，通过组合和继承等面向对象机制把对象有机关联起来）。然而，一个关系分解成多个关系，要使得分解有意义，其起码要求是分解之后不丢失原来的信息，这些信息不仅包括数据的本身，而且还包括由函数依赖所表示的数据之间的相互制约；分解的同时，必须考虑无损连接性和保持函数依赖这两个问题，有时不可能

做到既有无损连接性，又完全保持函数依赖，因而需要对其进行权衡[1]。进行分解的目的是达到更高的、一致的规范化程序。

2. Denormalization 设计

Denormalization 设计的基本思想：虽然关系型数据库是基于严紧的代数运算而来的，但是，数据库设计并不总是依从完美的数学化关系模式来进行。强制性地对事务进行规范化设计，形式上显得简单化，内容上趋于复杂化，更重要的是导致数据库运行效率的降低。非规范化要求适当地降低甚至抛弃关系模式的范式，不再要求一个表只描述一个实体或者实体间的一种联系。其主要目的在于提高数据库的运行效率。

Denormalization 处理的主要技术：包含增加冗余或者派生列，对表进行合并、分割或者增加重复表等。一般认为，在下列情况下可以考虑进行非规范化列：

(1)大量频繁的查询过程所涉及的表都需要进行连接；

(2)主要的应用程序执行要将表连接起来进行查询；

(3)对数据的计算需要临时表或者进行复杂的查询。

Denormalization 设计的优点：减少查询操作所需要的连接，减少了外部和检索的数量，可以预先进行统计计算，提高了查询时的相应速度。其缺点：增加了数据冗余，影响数据库的完整性，降低了数据库更新速度，增加了存储表所占有的物理空间。其中，最重要的是数据库的完整性问题，这一问题一般可以通过建立触发器、应用事务逻辑、在适当的时间间隔进行批命令或存储过程等方法得到解决。

Normalization 和 Denormalization 设计规则，各有利弊，并不是相互对立、非此即彼的关系。具体的选择和应用，要根据实际情况而定。例如：有时候保留部分冗余可能更方便数据查询，尤其是对于那些更新频率不高、查询频率极高的数据库系统更是如此(例如日志系统)。

目前，Normalization 遵循的主要范式包括 1NF、2NF、3NF、BCNF、4NF 和 5NF 等；而对于 Denormalization，一般分为下列几种方法：

(1)水平分割：主要是将一个表分成几张表，每张表都属于原始表的行的一个子集，它们都与原始表一样包含相同的列；

(2)垂直分割：主要是将一个表分成几张表，每张表都包含原始表的列的一个子集，它们都与原始表一样，包含相同的主键和相同的行数；

(3)打碎表关系：通过将几张表合并为一张表，从而减少表之间的连接提高查询效率[2]。

二、ORM 理论与实现

1. ORM 的理论基础

数据访问层从业务层分离出来，其核心是把数据对象与数据库二维表这两种不同数据结构进行转换。在业务层中，数据是存放在临时对象中，即以数据对象为最小单元；而数据库中，数据是永久性存放在二维关系的表中(这里我们讨论的数据库，均是目前比较成熟和被广泛应用的关系型数据库)。这两种数据类型各有特点，在数据对象设计中，要考虑的是对象的组合、继承和多态等；而在数据库表设计中，考虑的是表之间关联定义和关系描述。其相同点是：一个类有对象名，表也有表名；对象有属性，其中包括名称和类型，表有字段，其中包括字段名称和类型。这些相同点使得一个实例化对象和一张表的一行记录，通过对象的 Identity 和表的主键关联起来成为理论基础。其中涉及到表名与类名、表字段名称与类属性名称、表字段类型和类属性类型(这个跟具体开发语言还有很大关系，因为开发语言的基本类型和数据库表字段类型都存在部分类型差异)的对应关系，(即实体映射)以及表之间的关联关系对应到对象之间的组合关系的映射描述(即关系映射)，这就是 ORM 的理论基础所在。

2. ORM 的实现方案

目前，在 J2EE 架构中，有产品化的 Hibernate、JDO 和实体 Bean；在. NET 架构中，有 NHibernate

等。这些产品都是以ORM为理论基础，实现每一个实例化对象与数据库表的一行记录对应，由此，在业务层通过对数据对象的创建、读取、更新和删除操作(CRUD，Create、Read、Update、Delete)，对应数据库的插入、查询、修改和删除操作(ISUD，Insert、Select、Update、Delete)。

一般，实体映射和关系映射，有两种模式。其一，从现有数据库映射到数据对象(即数据驱动模型)，其中包括每张表到每个数据对象的映射信息；这里把数据库表之间的关联关系One－Many、One－One、Many－Many，映射为对象之间的继承、组合关系。其二，从现有数据对象映射到数据表(即对象驱动模型)，其中包括每个数据对象的映射信息到每张表的映射信息，这里把对象之间的继承、组合等关系映射为表之间的One－Many、One－One、Many－Many关系。通常前者较为常用。

三、ORM对数据库设计的要求

因为采用基于ORM理论来实现数据访问层设计，与一般直接根据业务编写SQL并执行SQL的方式有许多不同。前者把数据库的访问操作转变为对数据对象的操作，使业务层能更好地利用面向对象相关机制，更好地实现了系统在设计上强类聚、松耦合，也更能符合面向对象设计要求。由于前者把数据对象读取、编写和执行SQL语句等操作封装起来，其中涉及查询上述映射信息(不过前者的效率问题，被快速发展的硬件和建立缓存机制所抵消)等，因此，其缺点是执行效率没有后者高。N＋1次查询问题就是一个典型范例，这个问题在后者可以通过表之间的join来实现，但是在前者中，要求多查询一次。打一个比方，前者实现途径是沿着边走到对角位置处，后者实现途径是沿着其对角线走到对角位置处，走了捷径。原因在于前者是以数据对象为最小单元，而后者是以单个数据为单元，显然后者比较灵活。所以，针对他们的差异，基于以下原因，在采用ORM理论实现时，应在设计数据库上作相应的调整，才可以避免ORM自身的一些缺点，更好地实现ORM。

原因之一，ORM关于数据库表复杂关系映射处于理论阶段，在项目中很难得到应用。一般在项目设计中，数据库表之间更多采用One－Many和Many－Many(两张表之间的One－One关系一般被合并为一原张表方式取代)映射到对象之间的关系，以对象之间组合的方式最为常用，对象的继承等机制很少得到应用。另外，涉及到表复杂的关联关系映射，有许多种映射的可能。例如，对于多层的父子关系表，有四种映射方案[3]，不同情况使用不同方案，这给关系映射时带来不便，理论上的几种解决方案有时候在项目中却没有最好的应用。

原因之二，ORM对于复杂数据查询比较麻烦。例如，在Hibernate中，设计了一种HQL检索方式，其实质与SQL大同小异。它通过实体映射转化成标准SQL，那么在业务层中可以编写与业务相关的HQL语句，这一点跟数据访问设计层原则相违背。这是由于在业务层中不能出现SQL之类的数据库操作语句，编写SQL语句操作必须封装在数据访问层中，并且不能与具体业务数据访问数据库操作相关；而一般业务层开发人员编写的SQL都跟具体业务相关。

原因之三，对于调用存储过程，ORM没有一个比较完善的处理方式。一般项目中，比较复杂的业务经常放在存储过程中处理，而对于存储过程在ORM中的实现，存储过程名称可以和对象名称对应起来，参数可以和数据对象属性对应起来。但是，存储过程的处理结果，通过映射来处理比较复杂。因为一些存储过程要求返回查询结果，所以对于ORM的映射理论来说，这是一个“未知数”(因为在生成实体映射和关系映射信息时，ORM无法从存储过程定义中获取其返回结果信息)。

针对ORM自身的特点，在项目应用中要求在实体映射、关系映射和缓存机制中做一些相应的调整。

1. 实体的唯一性映射

ORM的基石是每一个实例化对象跟数据库表的一行记录一一对应，那么就要通过对象的ID和数据库表ID来进行关联，确定数据的唯一性。对数据对象而言，Identity是可以通过GetHashCode()来确定唯一性，也可以override这个函数；对数据库而言，要求每张表通过主键来识别唯一性；但是对于子表，一般是有两个键，即主键和外键来组成联合键，表示唯一性。在ORM中，要求数据对象ID和表ID一一对应，

如果表 ID 是一个组合键，映射比较复杂。因此，一般在设计表时，增加一个跟业务没有关系的字段 ID，设置为递增字段(或者制定编号规则)，这样在配置文件中描述表字段 ID 和数据对象的 ID 对应，简单实用。

2. 类型映射

在数据对象和表映射中，包含类型映射。一般要求数据对象中的属性类型跟对应数据库表字段类型相同；不过数据库系统所定义的字段基本类型，与具体开发语言定义的基本类型有所不同。例如：在 SQL Server 系列数据库中，float 类型是 8 个字节，而在 C#语言中，float 类型是四个字节[4]，double 才是 8 个字节；在这个映射中 float 对应的是 double 类型。因此，在数据库表字段设计中，应尽量采用常用的数据库字段类型(把数据对象的每个属性类型映射为 object，这样可以把从数据库读取的数据装在到对象中，也可以从数据对象中读取数据存放到数据库中。其缺点是：在业务层中，从数据对象中提取的数据，需要进行强制转换，要进行业务数据处理比较麻烦，也容易出现转换异常)。

3. 字段类型长度规范定义

在设计数据库表时要求定义字段长度。例如：字段类型为 char 或者 vchar 等，要求定义长度。ORM 在映射中要求描述字段长度，然后在执行 SQL 时自动截取数值长度。因此，在给数据对象赋值时，要注意该属性对应的表字段长度。这个不像直接编写 SQL 语句，如果参数太长，将会直接引发数据库处理异常。

4. 有效使用 ORM 缓存

ORM 采用缓存机制来管理数据对象，以减少访问数据库次数、提高速度。把操作的对象放在缓存中，ORM 采用定时机制跟数据库同步。如果是批量查询，ORM 首先到缓存里找数据对象，如果没有找到，再到数据库中去查询；然后把查找的数据以对象的形式放在缓存中，这样会减少其他操作的命中率。故在使用 ORM 产品时，要有效使用缓存；对于批量操作的情况，在配置中取消缓存配置，直接访问数据库。

5. 简化表关系复杂度

虽然，从理论上 ORM 可以处理 N 层表之间的关联关系，但是，可操作性很差。一是复杂表关系在关系映射时比较复杂，在生成对应的数据对象关联比较多；二是维护性很差，一旦关系局部变动，整个关系就得重新生成映射信息，数据对象要求重新生成，那么项目就要重新打包发布等。所以一般采用 Denormalization 设计规范来简化表关系，ORM 的映射也就简单而实用。

四、小结

一般数据库设计常采用 Normalization 和 Denormalization 两种规范，根据实际需求，把两者相互结合起来是一种最好的规范。在实现数据库访问时，是把数据库操作从业务层分离出来，即数据访问层，提高了系统的可维护性和可扩展性，从而也出现了以 ORM 为理论基础的许多成熟解决方案。但是，在项目设计开发中，不但要考虑到一般性数据库的设计，而且还要考虑到 ORM 在实现过程的一些细节，以克服缺点，达到简单实用的目的。

本文从 ORM 实体映射和关系映射，以及缓存机制等角度出发，分析了 ORM 实现特点，对数据库设计作了相应的调整，并在项目开发中得到很好的应用。

参考文献

[1] 数据库管理系统原理与设计. Raghu Ramhkrishnan 等著/周立柱等译 清华大学出版社 2004 年 3 月第一版

[2] PowerDesign 数据库系统分析设计与应用 姜江等编著. 电子工业出版社. 2004 年 8 月第一版

[3] 孙卫琴著. 精通 Hibernate Java 对象持久化技术详解，电子工业出版社，2005，5

[4] 精通 DotNet 核心技术原理与构架，刘小华编著. 电子工业出版社，2002 年 8 月第一版■