数据库设计学习相关笔记

数据库设计时需要考虑表之间的关联，而这种关联的实现途径就是“键”，数据库的“键”分为两种，**主键**和**外键**。主键用于唯一标识表中的一条记录，可以是一个单独的属性，也可以是多个属性的组合；外键则用于引用其他表中的主键，建立表之间的关联关系。

关于主键的选取，Fabian Pascal在《SQL and Relational Basics》中提到，主键的设计决策要参考三个原则：

* 最小化原则，即选取最少的列作为主键；
* 稳定性原则，即作为主键的列极少变动；
* 简单/熟悉原则，即作为主键的列为用户所熟知；

另外，尽量使用数值型属性作为主键，因相比文本型处理起来更容易，而且准确性高，不存在文本型属性如地址等可能发生的重复多对一情况。当没有合适的数值型属性时，往往会创建一个唯一键作为代理主键。但是唯一键通常不用于定义表之间的关系，这也是其区别于主键的一个特点。创建唯一键的关键字是UNIQUE，创建主键的关键字是primary key。

外键反映的实体关系有三种，一对一、一对多和多对多。其中多对多关系在数据库中不能被直接建模，它常常被设计为两个一对多关系——通过增加一个连接表。比如一位顾客可能买了多个保险公司的保险，而一个保险公司同时也拥有多个顾客，因此Cutomers\_TBL和Insurers\_TBL之间就是多对多的关系，此时通过建立一张连接表CustInsurance\_TBL将两张表变为两个一对多关系：一个顾客购买了0个，1个或多个保险；与此同时保险公司出售的保险也是这样一个关系。

三范式

范式的使用主要是为了在各个层面各个阶段尽量消除数据的冗余。满足第N范式的数据库一定先满足第N-1范式。但是有时候**适当的冗余**却可以提高查找效率。

1. **原子性**，所有列值必须是原子的，不可分解；方法是要求数据库的每个字段都只能存放单一值，而且每笔记录都要能利用一个惟一的主键来加以识别。
2. **主键依赖性**，每一个非键列都必须依赖整个主键，即完全依赖；通俗地说，第二范式要求主键能够描述表中每一条记录中的每一列。通过一个主键能够决定全部的其他列，而不是部分列。
3. **独立性**，所有非键列都是独立的；

三范式的应用往往可以使数据库设计更加规范化，节省物理空间，但同时也有可能导致查询时过多的表连接，从而降低性能。磁盘空间是廉价的，相比于I/O性能而言。

通常而言，第三范式以上的任何范式在商业上可能都是不切实际的，过分粒子化，过多表连接。

完整性规则

包括两点，即**实体完整性**和**引用完整性**。其中实体完整性是说主键不能为空值（NULL），引用完整性是指外键必须有所对应，即在其作为主键的表中必须存在。

这种完整性规则又称为业务规则，一定程度上避免了脏数据进入数据库中。

不推荐使用**视图**：视图本质上是查询定义，不包含任何数据，仅仅是对已有表的**逻辑覆盖**。针对视图的任何查询最终将执行包含在视图定义中的所有涉及底层表的查询。任何筛选都应该在视图定义过程中完成，因为对于视图自身的过滤筛选将在定义视图的查询执行结束之后再执行。视图可以加速开发过程，但在长期运行中可能完全影响数据库性能。

然而，还存在一种特殊的视图，**物化视图**。顾名思义，它不同于前面提到的视图那样停留在逻辑层面，而是建立的相应数据的物理层副本。这样就可以为其他用户释放底层的数据表，但是一个潜在的问题是，物化视图的内容可能与底层表数据并**不是同步**的，需要定期对其进行更新同步。

显式字段声明

是否空值（NULL/NOT NULL）、唯一性（UNIQUE）、有效性检验（CHECK）

存储函数、触发器（小心过度使用造成的触发器递归调用）

**固定长度字段的表更快**

如果表中的所有字段都是“固定长度”的，整个表会被认为是 “static” 或 “fixed-length”。 例如，表中没有如下类型的字段： VARCHAR，TEXT，BLOB。

固定长度的表会提高性能，因为MySQL搜寻得会更快一些，因为这些固定的长度是很容易计算下一个数据的偏移量的，所以读取的自然也会很快。而如果字段不是定长的，那么，每一次要找下一条的话，需要程序找到主键。

并且，固定长度的表也更容易被缓存和重建。不过，唯一的**副作用**是，固定长度的字段会浪费一些空间，因为定长的字段无论你用不用，他都是要分配那么多的空间。

**垂直分割**

“垂直分割”是一种把数据库中的表按列变成几张表的方法，这样可以降低表的复杂度和字段的数目，从而达到优化的目的。（以前，在银行做过项目，见过一张表有100多个字段，很恐怖）

示例一：在Users表中有一个字段是家庭地址，这个字段是可选字段，相比起，而且你在数据库操作的时候除了个人信息外，你并不需要经常读取或是改写这个字段。那么，为什么不把他放到另外一张表中呢？ 这样会让你的表有更好的性能，大家想想是不是，大量的时候，我对于用户表来说，只有用户ID，用户名，口令，用户角色等会被经常使用。小一点的表总是会有好的性能。

示例二： 你有一个叫 “last\_login” 的字段，它会在每次用户登录时被更新。但是，每次更新时会导致该表的查询缓存被清空。所以，你可以把这个字段放到另一个表中，这样就不会影响你对用户ID，用户名，用户角色的不停地读取了，因为查询缓存会帮你增加很多性能。

另外，你需要注意的是，这些被分出去的字段所形成的表，你不会经常性地去**Join**他们，不然的话，这样的性能会比不分割时还要差，而且，会是极数级的下降。

**拆分大的delete或者insert语句**

如果你需要在一个在线的网站上去执行一个大的 DELETE 或 INSERT 查询，你需要非常小心，要避免你的操作让你的整个网站停止相应。因为这两个操作是会锁表的，表一锁住了，别的操作都进不来了。

|  |
| --- |
| while (1) {  //每次只做1000条  mysql\_query("DELETE FROM logs WHERE log\_date <= '2009-11-01' LIMIT 1000");  if (mysql\_affected\_rows() == 0) {  // 没得可删了，退出！  break;  }  // 每次都要休息一会儿  usleep(50000);  } |

**越小的列越快**

对于大多数的数据库引擎来说，硬盘操作可能是最重大的瓶颈。所以，把你的数据变得紧凑会对这种情况非常有帮助，因为这减少了对硬盘的访问。当然，你也需要留够足够的扩展空间，不然，你日后来干这个事，你会死的很难看

好文参考：

1.<http://blog.csdn.net/u010665051/article/details/9261545>

2.鲍威尔, 洪波, and 恒. *数据库设计入门经典*. 清华大学出版社, 2007.

3.维基百科 <https://zh.wikipedia.org/wiki>

4.http://coolshell.cn/articles/1846.html