数据库设计指南

如果把企业的数据比做生命所必需的血液，那么数据库的设计就是应用中最重要的一部分。有关数据库设计的材料汗牛充栋，大学学位课程里也有专门的讲述。不过，就如我们反复强调的那样，再好的老师也比不过经验的教诲。所以我们最近找了些对数据库设计颇有造诣的专业人士给大家传授一些设计数据库的技巧和经验。我们的编辑从收到的 130 个反馈中精选了其中的 60 个最佳技巧，并把这些技巧编写成了本文，为了方便索引其内容划分为 5 个部分：

第 1 部分—设计数据库之前

这一部分罗列了 12 个基本技巧，包括命名规范和明确业务需求等。

第 2 部分—设计数据库表

总共 24 个指南性技巧，涵盖表内字段设计以及应该避免的常见问题等。

第 3 部分—选择键

怎么选择键呢？这里有 10 个技巧专门涉及系统生成的主键的正确用法，还有何时以及如何索引字段以获得最佳性能等。

第 4 部分—保证数据完整性

讨论如何保持数据库的清晰和健壮，如何把有害数据降低到最小程度。

第 5 部分—各种小技巧

不包括在以上 4 个部分中的其他技巧，五花八门，有 了它们希望你的数据库开发工作会更轻松一些。

第 1 部分—设计数据库之前

1. 考察现有环境

在设计一个新数据库时，你不但应该仔细研究业务需求而且还要考察现有的系统。大多数数据库项目都不是从头开始建立的；通常，机构内总会存在用来满足特定需求的现有系统（可能没有实现自动计算）。显然，现有系统并不完美，否则你就不必再建立新系统了。但是对旧系统的研究可以让你发现一些可能会忽略的细微问题。一般来说，考察现有系统对你绝对有好处。

—Lamont Adams

我曾经接手过一个为地区运输公司开发的数据库项目，活不难，用的是 Access 数据库。我设置了一些项目设计参数，而且同客户一道对这些参数进行了评估，事先还查看了开发环境下所采取的工作模式，等到最后部署应用的时候，只见终端上出了几个提示符然后立马在我面前翘辫子了！抓耳挠腮的折腾了好几个小时，我才意识到，原来这家公司的网络上跑着两个数据库应用，而对网络的访问需要明确和严格的用户帐号及其访问权限。明白了这一点，问题迎刃而解：只需采用客户的系统即可。这个项目给我的教训就是：记住，假如你在诸如 Access 或者 Interbase 这类公共环境下开发应用程序，一定要从表面下手深入系统内部搞清楚你面临的环境到底是怎么回事。

—kg

2. 定义标准的对象命名规范

一定要定义数据库对象的命名规范 。对数据库表来说，从项目一开始就要确定表名是采用复数还是单数形式。此外还要给表的别名定义简单规则（比方说，如果表名是一个单词，别名就取单词的前 4 个字母；如果表名是两个单词，就各取两个单词的前两个字母组成 4 个字母长的别名；如果表的名字由 3 个单词组成，你不妨从头两个单词中各取一个然后从最后一个单词中再取出两个字母，结果还是组成 4 字母长的别名，其余依次类推）对工作用表来说，表名可以加上前缀 WORK_ 后面附上采用该表的应用程序的名字。表内的列要针对键采用一整套设计规则。比如，如果键是数字类型，你可以用_NO 作为后缀；如果是字符类型则可以采用 _CODE 后缀。对列名应该采用标准的前缀和后缀。再如，假如你的表里有好多“ money ”字段，你不妨给每个列增加一个_AMT 后缀。还有，日期列最好以 DATE_作为名字打头。

—richard

检查表名、报表名和查询名之间的命名规范。你可能会很快就被这些不同的数据库要素的名称搞糊涂了。假如你坚持统一地命名这些数据库的不同组成部分，至少你应该在这些对象名字的开头用 table、query 或者 report 等前缀加以区别。

—rrydenm

如果采用了 Microsoft Access，你可以用 *qry*、*rpt*、 *tbl* 和 *mod* 等符号来标识对象（比如 tbl_Employees）。我在和 SQL Server（或者 Oracle）打交道的时候还用过 *tbl* 来索引表，但我用 sp_company （现在用 sp_feft_）标识存储过程，因为在有的时候如果我发现了更好的处理办法往往会保存好几个拷贝。我在实现 SQL Server 2000 时用 udf_ （或者类似的标记）标识我编写的函数。

—Timothy J. Bruce

3. 预先计划

上个世纪 80 年代初，我还在使用资产帐目系统和 System 38 平台，那时我负责设计所有的日期字段，这样在不费什么力气的情况下将来就可以轻松处理 2000 年问题了。许多人给我说就别去解决这一问题了，因为要处理起来太麻烦了（这在世人皆知的 Y2K 问题之前很久了）。我回击说只要预先计划今后就不会遇到大麻烦。结果我只用了两周的时间就把程序全部改完了。因为预先计划的好，后来 Y2K 问题对该系统的危害降到了最低程度（最近听说该程序甚至到了 1995 年都还运行在 AS/400 系统上，唯一出现的小问题是从代码中删除注释费了点工夫）。

—generalist

4. 获取数据模式资源手册

正在寻求示例模式的人可以阅读《*数据模式资源手册*》一书，该书由 Len Silverston、W. H. Inmon 和 Kent Graziano 编写，是一本值得拥有的最佳数据建模图书。该书包括的章节涵盖多种数据领域，比如人员、机构和工作效能等。

—minstrelmike

5. 畅想未来，但不可忘了过去的教训

我发现询问用户如何看待未来需求变化非常有用。这样做可以达到两个目的：首先，你可以清楚地了解应用设计在哪个地方应该更具灵活性以及如何避免性能瓶颈；其次，你知道发生事先没有确定的需求变更时用户将和你一样感到吃惊。

—chrisdk

一定要记住过去的经验教训！我们开发人员还应该通过分享自己的体会和经验互相帮助。即使用户认为他们再也不需要什么支持了，我们也应该对他们进行这方面的教育，我们都曾经面临过这样的时刻“当初要是这么做了该多好……”。

—dhattrem

6. 在物理实践之前进行逻辑设计

在深入物理设计之前要先进行逻辑设计。随着大量的 CASE 工具不断涌现出来，你的设计也可以达到相当高的逻辑水准，你通常可以从整体上更好地了解数据库设计所需要的方方面面。

—chardove

7. 了解你的业务

在你百分百地确定系统从客户角度满足其需求之前不要在你的 ER（实体关系）模式中加入哪怕一个数据表（怎么，你还没有模式？那请你参看技巧 9）。了解你的企业业务可以在以后的开发阶段节约大量的时间。一旦你明确了业务需求，你就可以自己做出许多决策 了。

—rangel

一旦你认为你已经明确 了业务内容，你最好同客户进行一次系统的交流。采用客户的术语并且向他们解释你所想到的和你所听到的。同时还应该用可能、将会和必须等词汇表达出系统的关系基数。这样你就可以让你的客户纠正你自己的理解然后做好下一步的 ER 设计。

—teburlew

8. 创建数据字典和 ER 图表

一定要花点时间创建 ER 图表和数据字典。其中至少应该包含每个字段的数据类型和在每个表内的主外键。创建 ER 图表和数据字典确实有点费时但对其他开发人员要了解整个设计却是完全必要的。越早创建越能有助于避免今后面临的可能混乱，从而可以让任何了解数据库的人都明确如何从数据库中获得数据。

—bgumbert

有一份诸如 ER 图表等最新文档其重要性如何强调都不过分，这对表明表之间关系很有用，而数据字典则说明了每个字段的用途以及任何可能存在的别名。对 SQL 表达式的文档化来说这是完全必要的。

—vanduin.chris.cj

9. 创建模式

一张图表胜过千言万语：开发人员不仅要阅读和实现它，而且还要用它来帮助自己和用户对话。模式有助于提高协作效能，这样在先期的数据库设计中几乎不可能出现大的问题。模式不必弄的很复杂；甚至可以简单到手写在一张纸上就可以了。只是要保证其上的逻辑关系今后能产生效益。

—Dana Daigle

10. 从输入输出下手

在定义数据库表和字段需求（输入）时，首先应检查现有的或者已经设计出的报表、查询和视图（输出）以决定为了支持这些输出哪些是必要的表和字段。举个简单的例子：假如客户需要一个报表按照邮政编码排序、分段和求和，你要保证其中包括了单独的邮政编码字段而不要把邮政编码糅进地址字段里。

—peter.marshall

11. 报表技巧

要了解用户通常是如何报告数据的：批处理还是在线提交报表？时间间隔是每天、每周、每月、每个季度还是每年？如果需要的话还可以考虑创建总结表。系统生成的主键在报表中很难管理。用户在具有系统生成主键的表内用副键进行检索往往会返回许多重复数据。这样的检索性能比较低而且容易引起混乱。

—kol

12. 理解客户需求

看起来这应该是显而易见的事，但需求就是来自客户（这里要从内部和外部客户的角度考虑）。不要依赖用户写下来的需求，真正的需求在客户的脑袋里。你要让客户解释其需求，而且随着开发的继续，还要经常询问客户保证其需求仍然在开发的目的之中。一个不变的真理是：“ 只有我看见了我才知道我想要的是什么 ” 必然会导致大量的返工，因为数据库没有达到客户从来没有写下来的需求标准。而更糟的是你对他们需求的解释只属于你自己，而且可能是完全错误的。

—kgilson

第 2 部分—设计表和字段

1. 检查各种变化

我在设计数据库的时候会考虑到哪些数据字段将来可能会发生变更。比方说，姓氏就是如此（注意是西方人的姓氏，比如女性结婚后从夫姓等）。所以，在建立系统存储客户信息时，我倾向于在单独的一个数据表里存储姓氏字段，而且还附加起始日和终止日等字段，这样就可以跟踪这一数据条目的变化。

—Shropshire Lad

2. 采用有意义的字段名

有一回我参加开发过一个项目，其中有从其他程序员那里继承的程序，那个程序员喜欢用屏幕上显示数据指示用语命名字段，这也不赖，但不幸的是，她还喜欢用一些奇怪的命名法，其命名采用了匈牙利命名和控制序号的组合形式，比如 cbo1、 txt2、txt2_b 等等。

除非你在使用只面向你的缩写字段名的系统，否则请尽可能地把字段描述的清楚些。当然，也别做过头了，比如 *Customer_Shipping_Address_Street_Line_1* l 虽然很富有说明性，但没人愿意键入这么长的名字，具体尺度就在你的把握中。

—Lamont Adams

3. 采用前缀命名

如果多个表里有好多同一类型的字段（比如 FirstName），你不妨用特定表的前缀（比如 CusLastName）来帮助你标识字段。

—notoriousDOG

时效性数据应包括“最近更新日期/时间”字段。时间标记对查找数据问题的原因、按日期重新处理/重载数据和清除旧数据特别有用。

—kol

5. 标准化和数据驱动

数据的标准化不仅方便了自己而且也方便了其他人。比方说，假如你的用户界面要访问外部数据源（文件、XML 文档、其他数据库等），你不妨把相应的连接和路径信息存储在用户界面支持表里。还有，如果用户界面执行工作流之类的任务（发送邮件、打印信笺、修改记录状态等），那么产生工作流的数据也可以存放在数据库里。预先安排总需要付出努力，但如果这些过程采用数据驱动而非硬编码的方式，那么策略变更和维护都会方便得多。事实上，如果过程是数据驱动的，你就可以把相当大的责任推给用户，由用户来维护自己的工作流过程。

—tduvall

6. 标准化不能过头

对那些不熟悉标准化一词（*normalization* ）的人而言，标准化可以保证表内的字段都是最基础的要素，而这一措施有助于消除数据库中的数据冗余。标准化有好几种形式，但 Third Normal Form（3NF）通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。简单来说，3NF 规定：

- 表内的每一个值都只能被表达一次。
- 表内的每一行都应该被唯一的标识（有唯一键）。
- 表内不应该存储依赖于其他键的非键信息。

遵守 3NF 标准的数据库具有以下特点：有一组表专门存放通过键连接起来的关联数据。比方说，某个存放客户及其有关定单的 3NF 数据库就可能有两个表：Customer 和 Order。Order 表不包含定单关联客户的任何信息，但表内会存放一个键值，该键指向 Customer 表里包含该客户信息的那一行。

更高层次的标准化也有，但更标准是否就一定更好呢？答案是不一定。事实上，对某些项目来说，甚至就连 3NF 都可能给数据库引入太高的复杂性。

—Lamont Adams

为了效率的缘故，对表不进行标准化有时也是必要的，这样的例子很多。曾经有个开发财务分析软件的活就是用非标准化表把查询时间从平均 40 秒降低到了两秒左右。虽然我不得不这么做，但我绝不把数据表的非标准化当作当然的设计理念。而具体的操作不过是一种派生。所以如果表出了问题重新产生非标准化的表是完全可能的。

—epepke

7. Microsoft Access 报表技巧

如果你正在使用 Microsoft Access，你可以用对用户友好的字段名来代替编号的名称：比如用 Customer Name 代替 txtCNaM。这样，当你用向导程序创建表单和报表时，其名字会让那些不是程序员的人更容易阅读。

—jwoodruf

8. 不活跃或者不采用的指示符

增加一个字段表示所在记录是否在业务中不再活跃挺有用的。不管是客户、员工还是其他什么人，这样做都能有助于再运行查询的时候过滤活跃或者不活跃状态。同时还消除了新用户在采用数据时所面临的一些问题，比如，某些记录可能不再为他们所用，再删除的时候可以起到一定 的防范作用。

—theoden

9. 使用角色实体定义属于某类别的列

在需要对属于特定类别或者具有特定角色的事物做定义时，可以用角色实体来创建特定的时间关联关系，从而可以实现自我文档化。

这里的含义不是让 PERSON 实体带有 Title 字段，而是说，为什么不用 PERSON 实体和 PERSON_TYPE 实体来描述人员呢？然后，比方说，当 John Smith, Engineer 提升为 John Smith, Director 乃至最后爬到 John Smith, CIO 的高位，而所有你要做的不过是改变两个表 PERSON 和 PERSON_TYPE 之间关系的键值，同时增加一个日期/时间字段来知道变化是何时发生的。这样，你的 PERSON_TYPE 表就包含了所有 PERSON 的可能类型，比如 Associate、Engineer、Director、CIO 或者 CEO 等。

还有个替代办法就是改变 PERSON 记录来反映新头衔的变化，不过这样一来在时间上无法跟踪个人所处位置的具体时间。

—teburlew

10．采用常用实体命名机构数据

组织数据的最简单办法就是采用常用名字，比如：PERSON、ORGANIZATION、ADDRESS 和 PHONE 等等。当你把这些常用的一般名字组合起来或者创建特定的相应副实体时，你就得到了自己用的特殊版本。开始的时候采用一般术语的主要原因在于所有的具体用户都能对抽象事物具体化。

有了这些抽象表示，你就可以在第 2 级标识中采用自己的特殊名称，比如，PERSON 可能是 Employee、Spouse、Patient、Client、Customer、Vendor 或者 Teacher 等。同样的，ORGANIZATION 也可能是 MyCompany、MyDepartment、Competitor、Hospital、Warehouse、Government 等。最后 ADDRESS 可以具体为 Site、Location、Home、Work、Client、Vendor、Corporate 和 FieldOffice 等。

采用一般抽象术语来标识“事物”的类别可以让你在关联数据以满足业务要求方面获得巨大的灵活性，同时这样做还可以显著降低数据存储所需的冗余量。

—teburlew

11．用户来自世界各地

在设计用到网络或者具有其他国际特性的数据库时，一定要记住大多数国家都有不同的字段格式，比如邮政编码等，有些国家，比如新西兰就没有邮政编码一说。

—billh

12．数据重复需要采用分立的数据表

如果你发现自己在重复输入数据，请创建新表和新的关系。

—Alan Rash

13．每个表中都应该添加的 3 个有用的字段

- dRecordCreationDate，在 VB 下默认是 Now()，而在 SQL Server 下默认为 GETDATE()
- sRecordCreator，在 SQL Server 下默认为 NOT NULL DEFAULT USER
- nRecordVersion，记录的版本标记；有助于准确说明记录中出现 null 数据或者丢失数据的原因

—Peter Ritchie

14．对地址和电话采用多个字段

描述街道地址就短短一行记录是不够的。Address_Line1、Address_Line2 和 Address_Line3 可以提供更大的灵活性。还有，电话号码和邮件地址最好拥有自己的数据表，其间具有自身的类型和标记类别。

—dwnerd

过分标准化可要小心，这样做可能会导致性能上出现问题。虽然地址和电话表分离通常可以达到最佳状态，但是如果需要经常访问这类信息，或许在其父表中存放“首选”信息（比如 Customer 等）更为妥当些。非标准化和加速访问之间的妥协是有一定意义的。

—dhattrem

15．使用多个名称字段

我觉得很吃惊，许多人在数据库里就给 *name* 留一个字段。我觉得只有刚入门的开发人员才会这么做，但实际上网上这种做法非常普遍。我建议应该把姓氏和名字当作两个字段来处理，然后在查询的时候再把他们组合起来。

—klempan

Klempan 不是唯一一个注意到使用单个 *name* 字段的人，要把这种情况变得对用户更为友好有好些方法。我最常用的是在同一表中创建一个计算列，通过它可以自动地连接标准化后的字段，这样数据变动的时候它也跟着变。不过，这样做在采用建模软件时得很机灵才行。总之，采用连接字段的方式可以有效的隔离用户应用和开发人员界面。

—damon

16．提防大小写混用的对象名和特殊字符

过去最令我恼火的事情之一就是数据库里有大小写混用的对象名，比如 CustomerData。这一问题从 Access 到 Oracle 数据库都存在。我不喜欢采用这种大小写混用的对象命名方法，结果还不得不手工修改名字。想想看，这种数据库/应用程序能混到采用更强大数据库的那一天吗？采用全部大写而且包含下划符的名字具有更好的可读性（CUSTOMER_DATA），绝对不要在对象名的字符之间留空格。

—bfren

17．小心保留词

要保证你的字段名没有和保留词、数据库系统或者常用访问方法冲突，比如，最近我编写的一个 ODBC 连接程序里有个表，其中就用了 DESC 作为说明字段名。后果可想而知！DESC 是 DESCENDING 缩写后的保留词。表里的一个 SELECT *语句倒是能用，但我得到的却是一大堆毫无用处的信息。

—Daniel Jordan

18．保持字段名和类型的一致性

在命名字段并为其指定数据类型的时候一定要保证一致性。假如字段在某个表中叫做“ agreement_number ”，你就别在另一个表里把名字改成“ ref1 ”。假如数据类型在一个表里是整数，那在另一个表里可就别变成字符型了。记住，你干完自己的活了，其他人还要用你的数据库呢。

—setanta

19．仔细选择数字类型

在 SQL 中使用 smallint 和 tinyint 类型要特别小心，比如，假如你想看看月销售总额，你的总额字段类型是 smallint，那么，如果总额超过了$32,767 你就不能进行计算操作了。

—egermain

20．删除标记

在表中包含一个“ 删除标记 ”字段，这样就可以把行标记为删除。在关系数据库里不要单独删除某一行；最好采用清除数据程序而且要仔细维护索引整体性。

—kol

21．避免使用触发器

触发器的功能通常可以用其他方式实现。在调试程序时触发器可能成为干扰。假如你确实需要采用触发器，你最好集中对它文档化。

—kol

22．包含版本机制

建议你在数据库中引入版本控制机制来确定使用中的数据库的版本。无论如何你都要实现这一要求。时间一长，用户的需求总是会改变的。最终可能会要求修改数据库结构。虽然你可以通过检查新字段或者索引来确定数据库结构的版本，但我发现把版本信息直接存放到数据库中不更为方便吗？。

—Richard Foster

23．给文本字段留足余量

ID 类型的文本字段，比如客户 ID 或定单号等等都应该设置得比一般想象更大，因为时间不长你多半就会因为要添加额外的字符而难堪不已。比方说，假设你的客户 ID 为 10 位数长。那你应该把数据库表字段的长度设为 12 或者 13 个字符长。这算浪费空间吗？是有一点，但也没你想象的那么多：一个字段加长 3 个字符在有 1 百万条记录，再加上一点索引的情况下才不过让整个数据库多占据 3MB 的空间。但这额外占据的空间却无需将来重构整个数据库就可以实现数据库规模的增长了。

—tlundin

24．列命名技巧

我们发现，假如你给每个表的列名都采用统一的前缀，那么在编写 SQL 表达式的时候会得到大大的简化。这样做也确实有缺点，比如破坏了自动表连接工具的作用，后者把公共列名同某些数据库联系起来，不过就连这些工具有时不也连接错误嘛。举个简单的例子，假设有两个表：Customer 和 Order。Customer 表的前缀是 cu_，所以该表内的子段名如下：cu_name_id、cu_surname、cu_initials 和 cu_address 等。Order 表的前缀是 or_，所以子段名是：or_order_id、or_cust_name_id、or_quantity 和 or_description 等。

这样从数据库中选出全部数据的 SQL 语句可以写成如下所示：

```
Select * from Customer, Order
Where cu_surname = "MYNAME"
and cu_name_id = or_cust_name_id
and or_quantity = 1;
```

在没有这些前缀的情况下则写成这个样子：

```
Select * from Customer, Order
Where Customer.surname = "MYNAME"
and Customer.name_id = Order.cust_name_id
and Order.quantity = 1
```

第 1 个 SQL 语句没少键入多少字符。但如果查询涉及到 5 个表乃至更多的列你就知道这个技巧多有用了。

—Bryce Stenberg

第 3 部分—选择键和索引

1. 数据采掘要预先计划

我所在的市场部门一度要处理 8 万多份联系方式，同时填写每个客户的必要数据（这绝对不是小活）。我从中还要确定出一组客户作为市场目标。当我从最开始设计表和字段的时候，我试图不在主索引里增加太多的字段以便加快数据库的运行速度。然后我意识到特定的组查询和信息采掘既不准确速度也不快。结果只好在主索引中重建而且合并了数据字段。我发现有一个指示计划相当关键——当我想创建系统类型查找时为什么要采用号码作为主索引字段呢？我可以用传真号码进行检索，但是它几乎就象系统类型一样对我来说并不重要。采用后者作为主字段，数据库更新后重新索引和检索就快多了。

—hscovell

可操作数据仓库（ODS）和数据仓库（DW）这两种环境下的数据索引是有差别的。在 DW 环境下，你要考虑销售部门是如何组织销售活动的。他们并不是数据库管理员，但是他们确定表内的键信息。这里设计人员或者数据库工作人员应该分析数据库结构从而确定出性能和正确输出之间的最佳条件。

—teburlew

2. 使用系统生成的主键

这一天类同技巧 1，但我觉得有必要在这里重复提醒大家。假如你总是在设计数据库的时候采用系统生成的键作为主键，那么你实际控制了数据库的索引完整性。这样，数据库和非人工机制就有效地控制了对存储数据中每一行的访问。

采用系统生成键作为主键还有一个优点：当你拥有一致的键结构时，找到逻辑缺陷很容易。

—teburlew

3. 分解字段用于索引

为了分离命名字段和包含字段以支持用户定义的报表，请考虑分解其他字段（甚至主键）为其组成要素以便用户可以对其进行索引。索引将加快 SQL 和报表生成器脚本的执行速度。比方说，我通常在必须使用 SQL LIKE 表达式的情况下创建报表，因为 case number 字段无法分解为 year、serial number、case type 和 defendant code 等要素。性能也会变坏。假如年度和类型字段可以分解为索引字段那么这些报表运行起来就会快多了。

—rdelval

4. 键设计 4 原则

- 为关联字段创建外键。
- 所有的键都必须唯一。
- 避免使用复合键。
- 外键总是关联唯一的键字段。

—Peter Ritchie

5. 别忘了索引

索引是从数据库中获取数据的最高效方式之一。95%的数据库性能问题都可以采用索引技术得到解决。作为一条规则，我通常对逻辑主键使用唯一的成组索引，对系统键（作为存储过程）采用唯一的非成组索引，对任何外键列采用非成组索引。不过，索引就象是盐，太多了菜就馍了。你得考虑数据库的空间有多大，表如何进行访问，还有这些访问是否主要用作读写。

—tduvall

大多数数据库都索引自动创建的主键字段，但是可别忘了索引外键，它们也是经常使用的键，比如运行查询显示主表和所有关联表的某条记录就用得上。还有，不要索引 memo/note 字段，不要索引大型字段（有很多字符），这样作会让索引占用太多的存储空间。

—gbrayton

6. 不要索引常用的小型表

不要为小型数据表设置任何键，假如它们经常有插入和删除操作就更别这样作了。对这些插入和删除操作的索引维护可能比扫描表空间消耗更多的时间。

—kbpatel

7. 不要把社会保障号码（SSN）选作键

永远都不要使用 SSN 作为数据库的键。除了隐私原因以外，须知政府越来越趋向于不准许把 SSN 用作除收入相关以外的其他目的，SSN 需要手工输入。永远不要使用手工输入的键作为主键，因为一旦你输入错误，你唯一能做的就是删除整个记录然后从头开始。

—teburlew

上个世纪 70 年代我还在读大学的时候，我记得那时 SSN 还曾被用做学号，当然尽管这么做是非法的。而且人们也都知道这是非法的，但他们已经习惯了。后来，随着盗取身份犯罪案件的增加，我现在的大学校园正痛苦地从一大摊子数据中把 SSN 删除。

—generalist

8. 不要用用户的键

在确定采用什么字段作为表的键的时候，可一定要小心用户将要编辑的字段。通常的情况下不要选择用户可编辑的字段作为键。这样做会迫使你采取以下两个措施：

- 在创建记录之后对用户编辑字段的行为施加限制。假如你这么做了，你可能会发现你的应用程序在商务需求突然发生变化，而用户需要编辑那些不可编辑的字段时缺乏足够的灵活性。当用户在输入数据之后直到保存记录才发现系统出了问题他们该怎么想？删除重建？假如记录不可重建是否让用户走开？
- 提出一些检测和纠正键冲突的方法。通常，费点精力也就搞定了，但是从性能上来看这样做的代价就比较大了。还有，键的纠正可能会迫使你突破你的数据和商业/用户界面层之间的隔离。

所以还是重提一句老话：你的设计要适应用户而不是让用户来适应你的设计。

—Lamont Adams

不让主键具有可更新性的原因是在关系模式下，主键实现了不同表之间的关联。比如，Customer 表有一个主键 CustomerID，而客户的定单则存放在另一个表里。Order 表的主键可能

是 OrderNo 或者 OrderNo、CustomerID 和日期的组合。不管你选择哪种键设置，你都需要在 Order 表中存放 CustomerID 来保证你可以给下定单的用户找到其定单记录。

假如你在 Customer 表里修改了 CustomerID，那么你必须找出 Order 表中的所有相关记录对其进行修改。否则，有些定单就会不属于任何客户——数据库的完整性就算完蛋了。

如果索引完整性规则施加到表一级，那么在不编写大量代码和附加删除记录的情况下几乎不可能改变某一条记录的键和数据库内所有关联的记录。而这一过程往往错误丛生所以应该尽量避免。

—ljboast

9. 可选键有时可做主键

记住，查询数据的不是机器而是人。

假如你有可选键，你可能进一步把它用做主键。那样的话，你就拥有了建立强大索引的能力。这样可以阻止使用数据库的人不得不连接数据库从而恰当的过滤数据。在严格控制域表的数据库上，这种负载是比较醒目的。如果可选键真正有用，那就是达到了主键的水准。

我的看法是，假如你有可选键，比如国家表内的 state_code，你不要在现有不能变动的唯一键上创建后续的键。你要做的无非是创建毫无价值的数据。比如以下的例子：

```
Select count(*)
from address, state_ref
where
address.state_id = state_ref.state_id
and state_ref.state_code = 'TN'
```

我的做法是这样的：

```
Select count(*)
from address
where
and state_code = 'TN'
```

如你因为过度使用表的后续键建立这种表的关联，操作负载真得需要考虑一下了。

—Stocker

10. 别忘了外键

大多数数据库索引自动创建的主键字段。但别忘了索引外键字段，它们在你想查询主表中的记录及其关联记录时每次都会用到。还有，不要索引 memo/notes 字段而且不要索引大型文本字段（许多字符），这样做会让你的索引占据大量的数据库空间。。

—gbrayton

第 4 部分—保证数据的完整性

1. 用约束而非商务规则强制数据完整性

如果你按照商务规则来处理需求，那么你应当检查商务层次/用户界面：如果商务规则以后发生变化，那么只需要进行更新即可。

假如需求源于维护数据完整性的需要，那么在数据库层面上需要施加限制条件。

如果你在数据层确实采用了约束，你要保证有办法把更新不能通过约束检查的原因采用用户理解的语言通知用户界面。除非你 的字段命名很冗长，否则字段名本身还不够。

—Lamont Adams

只要有可能，请采用数据库系统实现数据的完整性。这不但包括通过标准化实现的完整性而且还包括数据的功能性。在写数据的时候还可以增加触发器来保证数据的正确性。不要依赖于商务层保证数据完整性；它不能保证表之间（外键）的完整性所以不能强加于其他完整性规则之上。

—Peter Ritchie

2. 分布式数据系统

对分布式系统而言，在你决定是否在各个站点复制所有数据还是把数据保存在一个地方之前应该估计一下未来 5 年或者 10 年的数据量。当你把数据传送到其他站点的时候，最好在数据库字段中设置一些标记。在目的站点收到你的数据之后更新你的标记。为了进行这种数据传输，请写下你自己的批处理或者调度程序以特定时间间隔运行而不要让用户在每天的工作后传输数据。本地拷贝你的维护数据，比如计算常数和利息率等，设置版本号保证数据在每个站点都完全一致。

— Suhair TechRepublic

3. 强制指示完整性

没有好办法能在有害数据进入数据库之后消除它，所以你应该在它进入数据库之前将其剔除。激活数据库系统的指示完整性特性。这样可以保持数据的清洁而能迫使开发人员投入更多的时间处理错误条件。

—kol

4. 关系

如果两个实体之间存在多对一关系，而且还有可能转化为多对多关系，那么你最好一开始就设置成多对多关系。从现有的多对一关系转变为多对多关系比一开始就是多对多关系要难得多。

—CS Data Architect

5. 采用视图

为了在你的数据库和你的应用程序代码之间提供另一层抽象，你可以为你的应用程序建立专门的视图而不必非要应用程序直接访问数据表。这样做还等于在处理数据库变更时给你提供了更多的自由。

—Gay Howe

6. 给数据保有和恢复制定计划

考虑数据保有策略并包含在设计过程中，预先设计你的数据恢复过程。采用可以发布给用户/开发人员的数据字典实现方便的数据识别同时保证对数据源文档化。编写在线更新来“更新查询”供以后万一数据丢失可以重新处理更新。

—kol

7. 用存储过程让系统做重活

解决了许多麻烦来产生一个具有高度完整性的数据库解决方案之后，我所在的团队决定封装一些关联表的功能组，提供一整套常规的存储过程来访问各组以便加快速度和简化客户程序代码的开发。在此期间，我们发现 3GL 编码器设置了所有可能的错误条件，比如以下所示：

```
SELECT Cnt = COUNT (*)
FROM [<Table>]
WHERE [<primary key column>] = <new value>
IF Cnt = 0
BEGIN
INSERT INTO [<Table>]
( [< primary key column>] )
VALUES ( <New value> )
END
ELSE
BEGIN
<indicate duplication error>
END
```

而一个非 3GL 编码器是这样做的：

```
INSERT INTO [<Table>]
( [< primary key column>] )
VALUES
( <New value> )
IF @@ERROR = 2627 -- Literal error code for Primary Key Constraint
BEGIN
<indicate duplication error>
END
```

第 2 个程序简单多了，而且事实上，利用了我们给数据库的功能。虽然我个人不喜欢使用嵌入文字（2627）。但是那样可以很方便地用一点预先处理来代替。数据库不只是一个存放数据的地方，它也是简化编码之地。

—a-smith

8. 使用查找

控制数据完整性的最佳方式就是限制用户的选择。只要有可能都应该提供给用户一个清晰的价值列表供其选择。这样将减少键入代码的错误和误解同时提供数据的一致性。某些公共数据特别适合查找：国家代码、状态代码等。

—CS Data Architect

第 5 部分—各种小技巧

1. 文档、文档、文档

对所有的快捷方式、命名规范、限制和函数都要编制文档。

—nickypendragon

采用给表、列、触发器等加注释的数据库工具。是的，这有点费事，但从长远来看，这样做对开发、支持和跟踪修改非常有用。

—chardove

取决于你使用的数据库系统，可能有一些软件会给你一些供你很快上手的文档。你可能希望先开始在说，然后获得越来越多的细节。或者你可能希望周期性的预排，在输入新数据同时随着你的进展对每一部分细节化。不管你选择哪种方式，总要对你的数据库文档化，或者在数据库自身的内部或者单独建立文档。这样，当你过了一年多时间后再回过头来做第 2 个版本，你犯错的机会将大大减少。

—mrs_helm

2. 使用常用英语（或者其他任何语言）而不要使用编码

为什么我们经常采用编码（比如 9935A 可能是墨水笔的供应代码，4XF788-Q 可能是帐目编码）？理由很多。但是用户通常都用英语进行思考而不是编码。工作 5 年的会计或许知道 4XF788-Q 是什么东西，但新来的可就不一定了。在创建下拉菜单、列表、报表时最好按照英语名排序。假如你需要编码，那你可以在编码旁附上用户知道的英语。

—amasa

3. 保存常用信息

让一个表专门存放一般数据库信息非常有用。我常在这个表里存放数据库当前版本、最近检查/修复（对 Access）、关联设计文档的名称、客户等信息。这样可以实现一种简单机制跟踪数据库，当客户抱怨他们的数据库没有达到希望的要求而与你联系时，这样做对非客户机/服务器环境特别有用。

—Richard Foster

4. 测试、测试、反复测试

建立或者修订数据库之后，必须用用户新输入的数据测试数据字段。最重要的是，让用户进行测试并且同用户一道保证你选择的数据类型满足商业要求。测试需要在把新数据库投入实际服务之前完成。

—juneebug

5. 检查设计

在开发期间检查数据库设计的常用技术是通过其所支持的应用程序原型检查数据库。换句话说，针对每一种最终表达数据的原型应用，保证你检查了数据模型并且查看如何取出数据。

—jgootee

6. Access 设计技巧

对复杂的 Microsoft Access 数据库应用程序而言，可以把所有的主表放在一个数据库文件里，然后增加其他数据库文件和装载同原有数据库有关的特殊函数。根据需要用这些函数连接到主文件中的主表。比如数据输入、数据 QC、统计分析、向管理层或者政府部门提供报表以及各类只读查询等。这一措施简化了用户和组权限的分配，而且有利于应用程序函数的分组和划分，从而在程序必须修改的时候易于管理。

—Dennis Walden