HZ BOOKS
华章IT

数据库
技术丛书

SQL优化最佳实践

构建高效率Oracle数据库的方法与技巧

韩锋 著

SQL Optimization Best Practice

- 15年DBA经验结晶，依托近百个真实案例，详细剖析SQL语句优化的原理、方法及技术要点，所讲内容均可直接落地
- 所有内容均来自一线真实项目，以一线开发工程师的语言和视角展开介绍，无论是初学者还是中级技术人员，都可轻松接受、吸收

机械工业出版社
China Machine Press

数据库
技术丛书

SQL优化最佳实践

构建高效率Oracle数据库的方法与技巧

SQL Optimization Best Practice

韩锋 著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

SQL 优化最佳实践：构建高效率 Oracle 数据库的方法与技巧 / 韩锋著 . —北京：机械工业出版社，2016.4
（数据库技术丛书）

ISBN 978-7-111-53454-9

I. S… II. 韩… III. 关系数据库系统 IV. TP311.138

中国版本图书馆 CIP 数据核字（2016）第 069845 号

SQL 优化最佳实践

构建高效率 Oracle 数据库的方法与技巧

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：孙海亮　　责任校对：殷　虹
印　　刷：三河市宏图印务有限公司　　版　　次：2016 年 5 月第 1 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：20.25
书　　号：ISBN 978-7-111-53454-9　　定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

Preface 前　言

为什么要写这本书

我早年毕业后，曾长期从事ERP、电子政务类软件的开发工作，作为一个数据库的使用者，我接触到了大量数据库，如FoxPro、SQL Server、Oracle、Informix……在不断的使用过程中，我对这一领域越来越感兴趣，并最终选择了数据库这条路。曾经长期担任DBA、数据库架构师等职，参与了大大小小很多项目的数据库设计、开发、优化工作，并在这一过程中积累了一些经验。在多年的工作中，我发现数据库领域存在一些现象。

现象一，开发人员将数据库视为“黑盒子”。开发人员不关心、不重视，也不了解SQL语句的执行情况、数据库的运行机理。甚至在很多O/R Mapping工具的辅助下，连基本的SQL语句也不需要手工编写。固然，通过引入这些工具可以大大加快研发速度，但其带来的弊端是，开发人员并不了解数据库是如何完成这些请求并获得数据的，优化更是无从谈起。

现象二，对SQL质量重视程度不足。我在长期的工作中发现，绝大多数公司对SQL质量的重视程度严重不足。往往在项目的前期设计、代码开发、测试等多个环节，都没有DBA的参与。直到项目上线，甚至到出现性能问题时，才会有DBA介入处理。这种救火员的模式，往往效果不好，即使有了解决方案，其代价、成本也必然是巨大的。

现象三，开发人员想提高却无从下手。有些开发人员认识到SQL语句质量的重要性，想要提高却无从下手。一方面，他们本身不具备数据库的专业知识；另一方面，SQL编程本身也有其特殊性，与其他常用开发语言有较大差异。正是这些因素，导致开发人员想要提高却困难重重。

现象四，重运行维护，轻开发优化。数据库的稳定运行、数据安全等是非常重要的，这也是DBA的核心职责之一。但对于开发优化，则往往存在重视程度不足的问题。我们经常会看到一个项目里，公司会花大笔费用购买昂贵的硬件、备份软件等，却不舍得购买与

数据库优化、SQL 审计相关的软件。此外，随着自动化运维的逐步推广，乃至数据库云服务的逐步成熟，传统意义的数据库运维工作必然会逐步萎缩，取而代之的则是数据库的设计、开发乃至整体架构工作逐步增多。这也是 DBA 未来发展的一个方向。

现象五，资料繁多，却无从选择。Oracle 数据库在国内流行多年，该领域的书籍也非常多，但涉及优化类的相对较少，特别是局限在 SQL 语句优化范畴的。近年来我也发现了几本不错的书籍，但普遍存在技术偏深、可操作性不强的问题。广大数据库开发的初学者或者有一定经验但急需提高的读者，不太适用。

正是因为存在上述种种现象，促使我有了将多年的经验汇集成册，编写出版的想法。一方面是能够帮助有相关需求的人，另一方面也是对自己多年工作的一个总结。最后，希望这本书能够引领开发人员、DBA 在 SQL 语句的编写优化上更进一步。倘若这本书能够帮助大家解决实际中遇到的问题，我将非常荣幸。

本书特色

本书从多角度阐述了 SQL 语句优化的方方面面，总体上可分为两大部分。第一部分主要讲解跟 SQL 优化相关的背景知识和基本原理；第二部分重点讲述了优化器的各种优化手段。本书整体具有以下几个特点：

- 书中内容由项目而生，以一线开发工程师的视角和言语展开。
- 注重实战。几乎所有的章节都配以代码，读者可在环境中直接编写代码并运行。大部分代码都附有详细的说明，便于读者理解内容。
- 涵盖了 SQL 语句的诸多方面，特别是第二部分，可作为工作手册供大家优化时查阅使用。

读者对象

本书适用于想要提高 SQL 语句运行效率乃至数据库整体性能的所有人，包括架构师、DBA、开发人员、测试人员等。书中讲解了 Oracle 数据库的 SQL 语句优化，但除了个别 Oracle 自有的优化特性外，其核心思想也适用于其他关系型数据库。书中没有讲解 Oracle 体系结构和 SQL 语言本身，这里假设大部分人已熟悉 Oracle 和 SQL 语言。具体来说，包括但不局限于下列人员：

- Oracle 数据库开发人员；
- 数据库架构师、数据库管理员；
- 其他关系型数据库的从业者；

- 对 SQL 语句优化感兴趣的人员；
- 大专院校计算机相关专业的学生。

如何阅读本书

本书分为四大部分：

第一部分为引入篇（第 0～1 章）。

引言部分我结合多年的工作经验，总结了进行 SQL 语句优化时可能会面临的一些问题。读者可以观察是否在自己的身边也存在类似的问题。后面还讲述了一些常见的关于 SQL 优化的误区，以方便读者正确看待 SQL 语句优化。

第 1 章讲述了我曾经处理过的几个案例。通过这些活生生的案例，可以让读者更直观感受到 SQL 语句优化的重要。同时在每个案例后面，我还针对案例出现的问题进行了总结。

第二部分为原理篇（第 2～9 章）。

第 2 章讲述了 SQL 语句优化的核心组件——优化器，以及优化的最基础概念——成本。这部分非常重要，建议初学者仔细阅读。

第 3～6 章介绍了和优化相关的几个重要概念：执行计划、统计信息、SQL 解析、游标、绑定变量。这部分都较为基础，建议初学者根据情况选择阅读。

第 7～8 章介绍了 SQL 语句的实体对象及物理上是如何存储的。这部分对于数据库结构设计有较大帮助。此外，在对 SQL 语句进行优化时，也需要考虑相关对象的情况，因为优化措施可能会影响该对象的其他语句，需要统筹考虑。

第 9 章介绍了 Oracle 专有的一些 SQL 语句。有时使用这些语句，可以达到意想不到的效果。如不考虑以后有数据库平台迁移的问题，可以充分利用这些语句。

原理篇是我们是迈入实战篇的基础，它几乎覆盖了 SQL 优化相关的所有原理知识。通过对这些内容的学习，可以为后面的优化部分打下良好的基础。如果你已经拥有相关知识，可以直接进入实战篇。

第三部分为实战篇（第 10～16 章）。它是本书的重点。

第 10 章介绍了一个重要的优化手段——查询转换。这部分相对来说比较难，相关资料说明较少，可作为重点来看。

第 11 章介绍了数据对象的访问方式。这部分也非常基础，应重点来看。

第 12～16 章介绍了多种操作及常见的优化手段，包括表关联、半 / 反连接、子查询、排序、并行等。这部分读者可根据实际需要进行有重点的阅读。

实战篇是本书的重点，这部分覆盖了常见优化的多个方面。读者可将这部分作为参考

资料，当需要时反复阅读。这部分还包含了大量示例代码，读者可以通过实践反复体会。

本书还提供了读者可能感兴趣的拓展知识，放在附录。

附录介绍了前面各章节提到的数据库参数、数据字典、等待事件、提示等内容。此外，还包括如何构造样例数据，方便读者进行实际操作。

以上是本书各个章节的安排情况和写作思路，希望有助于读者阅读。

勘误和支持

由于笔者水平有限，加之编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。大家可以通过邮箱 hanfeng7766@sohu.com 与我取得联系。你可以将书中的错误和问题反馈给我，我将尽量在线上为你提供最满意的解答。期待能够得到你的真挚反馈。

致谢

感谢每一位帮助过我的老师、同事和领导，是你们让我有了学习和总结的机会。感谢宜信公司的各级领导、同事对我的支持和鼓励，你们的支持充分体现了宜信开放、分享的企业文化。此外，也要感谢我的老东家——当当网，在那里我积累了丰富的经验，并坚定了完成本书的信心。

感谢机械工业出版社华章公司的编辑孙海亮，在这一年多的时间中始终支持我的写作。第一次著书写作，过程漫长而艰辛，正是你的鼓励和帮助引导我顺利完成全部书稿。

感谢远在哈尔滨的爸爸、妈妈和姐姐，是你们在艰苦环境下将我培养成人，并时时刻刻为我灌输爱的力量！感谢我的岳父母，是你们承担了琐碎的家务，让我能安心写作。最后，也是最重要的，一如既往地感谢陪伴我左右的妻子和孩子，你们的爱和支持是本书得以完成的最大动力。

谨以本书献给我最亲爱的家人和朋友，以及正在为自我实现而奋斗的、充满朝气的 IT 工程师们！

韩　锋

Contents 目　录

第一篇 Part 1

引 入 篇

Chapter 0 第 0 章

引　言

笔者早年间从事了多年开发工作，后因个人兴趣转做数据库。在长期的工作实践中，看到了数据库工作（特别是 SQL 优化）面临的种种问题，同时也发现人们在对数据库优化的认识上存在一些误区。

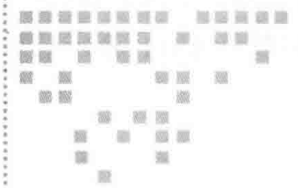
1. 面临的问题

- **没有专职人员**：在很多公司或者说绝大多数公司，没有独立的数据库团队。往往由开发人员完成部分 DBA 的职责，包括结构设计、SQL 优化甚至部分运维工作。受限于自身的精力，开发人员很难做到专业化。
- **“赶工期”现象**：在项目驱动的公司，经常出现赶工期的现象，而且往往牺牲的就是数据库的设计、评测、优化的时间。常常只是开发完毕后就匆忙上线，直到在线上运行出现问题后才会回头进行处理。但这时往往已经造成了很大的损失。
- **话语权不大**：数据库团队在公司中或者在项目中，往往话语权不高。在很多产品、项目决策过程中，常常会忽略 DBA 的声音。
- **需求不明**：很多项目在设计初期，往往对业务描述很详尽，但对数据库却只字未提。相关数据库的存储量、访问特征、高峰时间的 TPS 及 QPS 等往往只有到上线后才有比较清晰的认识。其后果就是往往需要大量优化工作，甚至导致需要对底层架构进行修改，这样最终会导致成本大大提高，有时增加的成本甚至是不可接受的。
- **重运维、不重架构设计**：有些公司认识到数据库的重要性，但往往只重视运维而忽视了前期的架构设计、开发优化等问题。系统上线后暴露出问题后，只能采取事后补救措施，但这往往会带来高昂的成本。

- **盲目优化**：有些公司确实很重视 SQL 优化工作，但又缺乏必要的技术投入。经常见到这样的开发规范——所有 WHERE 条件字段都必须加上索引。其结果就是数据库被“过分”优化，适得其反。

2. 常见误区

- **关系数据库已死**：近些年来，随着 NoSQL 的蓬勃发展，有一种观点也逐渐盛行——关系数据库必将死亡，NoSQL 将取而代之！随之而来的就是 SQL 优化没有必要，不必在其上再花费很大力气。NoSQL 作为一种新兴的技术，的确有其鲜明的特点，也适用于一些场合。但我们要看到，很多需要 ACID 的场景下，传统数据库仍然是不二选择，不可取代。
- **“SQL 优化”很简单**：有些人认为，SQL 优化很简单，甚至碰到过这种观点——SQL 优化不就是加几个索引嘛，有啥难的！其带来的直接后果就是，不重视这部分工作。笔者也确实在某业务系统 (OLTP) 中，观察到单表存在 30 多个索引的情况。也遇到过，因为索引过多导致执行性能出现问题的情况。这种情况，往往只有在血淋淋的事故后，才能引起领导的重视。
- **硬件技术发展很快，不用再计较 SQL 优化**：确实，硬件技术近些年来发展迅速，特别是以多核 CPU 为代表的并行处理技术和以 SSD 为代表的存储技术。这些新技术的使用，使得服务器的处理能力有了极大的提升。但我们清醒地看到，SQL 优化才是问题的根本解决之道。我们后面可以看到一条 SQL 语句，可以轻易跑死一个数据库。这不是简单地通过硬件升级就可以解决的问题。
- **SQL 优化只是 DBA 的事情**：在很多设计、开发、测试人员的眼中，SQL 优化只是 DBA 的事情，他们不需要去关心。落实到具体工作中，相关人员就缺乏相应的优化意识，只注重自身功能的实现而忽略了相应的执行成本。最终的结果往往就是代码质量不高，上线后问题过多。
- **数据仓库都使用 Hadoop，不用传统关系型数据库了**：Hadoop 作为一种新兴技术，被越来越多地用在数据分析领域。很多国内外的大型公司，都采用了这个解决方案。但我们清醒地看到，它的定位更倾向于是一种“离线数据分析平台”，而不是“分布式数据库”。其时效性、准确性等难以满足特性需求。现在有很多公司在 Hadoop 上面做了类似“SQL 引擎”的东西，就是仿照关系数据库的处理方式处理 Hadoop 中的数据，但要想达到发展了数十年的数据库水平，还有很长的路要走。笔者对这两者的认识是：各有所长，互为补充。



Chapter 1 第 1 章

与 SQL 优化相关的几个案例

案例 1　一条 SQL 引发的血案

1. 案例说明

某大型电商公司数据仓库系统，正常情况下每天凌晨 0~9 点会执行大量作业生成前一天的业务报表，供管理层分析使用。但某天早晨 6 点开始，监控人员就频繁收到业务报警，大批业务报表突然出现大面积延迟。原本 8 点前就应跑出的报表，一直持续到 10 点仍然没有结果。公司领导非常重视，严令在 11 点前必须解决问题。

DBA 紧急介入处理，通过 TOP 命令查看到某个进程占用了大量资源，杀掉后不久还会再次出现。经跟开发人员沟通，这是由于调度机制所致，非正常结束的作业会反复执行。暂时设置该作业无效，并从脚本中排查可疑 SQL。同时对比从线上收集的 ASH/AWR 报告，最终定位到某条 SQL 比较可疑，经跟开发人员确认系一新增功能，因上线紧急，只做了简单的功能测试。正是因为这一条 SQL，导致整个系统运行缓慢，大量作业受到影响，修改 SQL 后系统恢复正常。

具体分析：

```
SELECT /*+ INDEX (A1 xxxxx) */ SUM(A2.CRKSL),  SUM(A2.CRKSL*A2.DJ) ...
FROM xxxx A2, xxxx A1
WHERE A2.CRKFLAG=xxx AND A2.CDATE>=xxx AND A2.CDATE<xxx;
```

这是一个很典型的两表关联语句，两张表的数据量都较大。下面来看看执行计划，如图 1-1 所示。

执行计划触目惊心，优化器评估返回的数据量为 3505T 条记录，计划返回量 127P 字

节，总成本 9890G，返回时间 999:59:59。

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				9890G(100)			
1	SORT AGGREGATE		1	41				
2	MERGE JOIN CARTESIAN		3505T	127P	9890G (1)	999:59:59		
3	PARTITION RANGE ITERATOR		25M	1010M	170K (1)	00:34:12	153	243
4	TABLE ACCESS FULL	[redacted]	25M	1010M	170K (1)	00:34:12	153	243
5	BUFFER SORT		135M		9890G (1)	999:59:59		
6	INDEX FULL SCAN	[redacted]	135M		382K (1)	01:16:34		

图 1-1　执行计划

分析结论：从执行计划中可见，两表关联使用了笛卡儿积的关联方式。我们知道笛卡儿连接是指在两表连接没有任何连接条件的情况。一般情况下应尽量避免笛卡儿积，除非某些特殊场合。否则再强大的数据库，也无法处理。这是一个典型的多表关联缺乏连接条件，导致笛卡儿积，引发性能问题的案例。

2. 给我们的启示

从案例本身来讲，并没有什么特别之处，不过是开发人员疏忽，导致了一条质量很差的 SQL。但从更深层次来讲，这个案例可以给我们带来如下启示：

- 开发人员的一个疏忽，造成了严重的后果，原来数据库竟是如此的脆弱。需要对数据库保持一种“敬畏”之心。
- 电脑不是人脑，它不知道你的需求是什么，只能根据写好的逻辑进行处理。
- 不要去责怪开发人员，谁都会犯错误，关键是如何从制度上保证不再发生类似的问题。

3. 解决之道

（1）SQL 开发规范

加强对数据库开发人员的培训工作，提高其对数据库的理解能力和 SQL 开发水平。将部分 SQL 运行检查的职责前置，在开发阶段就能规避很多问题。要向开发人员灌输 SQL 优化的思想，在工作中逐步积累，这样才能提高公司整体开发质量，也可以避免很多低级错误。

（2）SQL Review 制度

对于 SQL Review，怎么强调都不过分。从业内来看，很多公司也都在自己的开发流程中纳入了这个环节，甚至列为考评范围，对其重视程度可见一斑。其常见典型做法是利用 SQL 分析引擎（商用或自研）进行分析或采取半人工的方式进行审核。对于审核后的结果，可作为持续改进的依据。SQL Review 的中间结果可以保留，作为系统上线后的对比分析依据，进而可将 SQL 的审核、优化、管理等功能集成起来，完成对 SQL 整个生命周

期的管理。

（3）限流 / 资源控制

有些数据库提供了丰富的资源限制功能，可以从多个维度限制会话对资源（CPU、MEMORY、IO）的使用。可避免发生单个会话影响整个数据库的运行状态。对于一些开源数据库，部分技术实力较强的公司，还通过对内核的修改实现了限流功能，控制资源消耗较多的 SQL 运行数量，从而避免拖慢数据库的整体运行。

案例 2　糟糕的结构设计带来的问题

1. 案例说明

这是某公司后台的 ERP 系统，系统已经上线运行了 10 多年。随着时间的推移，累积的数据量越来越大。随着公司业务量的不断增加，数据库系统运行缓慢的问题日益凸显。为提高运行效率，公司计划有针对性地对部分大表进行数据清理。在 DBA 对某个大表进行清理时出现了问题。这个表本身有数百 GB，按照指定的清理规则只需要根据主键字段范围（运算符为 >=）选择出一定比例（不超过 10%）的数据进行清理即可。但在实际使用中发现，该 SQL 是全表扫描，执行时间大大超出预期时间。DBA 尝试使用强制指定索引方式清理数据，依然无效，整个 SQL 语句的执行效率达不到要求。为了避免影响正常业务运行，不得不将此次清理工作放在半夜进行，还需要协调库房等诸多单位进行配合，严重影响正常业务运行。

为了尽量减少对业务的影响，DBA 求助笔者帮助协同分析。这套 ERP 系统是由第三方公司开发的，历史很久远，相关的数据字典等信息都已经找不到了，只能从纯数据库的角度进行分析。这是一个普通表（非分区表），按照主键字段的范围查询一批记录并进行清理。按照正常理解，执行索引范围扫描应该是效率较高的一种处理方式，但实际情况都是全表扫描。进一步分析发现，该表的主键是没有业务含义的，仅仅是自增长的数据，其来源是一个序列。但奇怪的是，这个主键字段的类型是变长文本类型，而不是通常的数字类型。当初定义该字段类型的依据，现在已经无从考证，但实验表明正是这个字段的类型“异常”，导致了错误的执行路径。

下面通过一个实验重现这个问题。

（1）数据准备

两个表的数据类型相似（只是 ID 字段类型不同），各插入了 320 万数据，ID 字段范围为 1~3200000。

```
create table t1 as select * from dba_objects where 1=0;
alter table t1 add id int primary key;
```

```
create table t2 as select * from dba_objects where 1=0;
alter table t2 add id varchar2(10) primary key;

insert into t1
select 'test','test','test',rownum,rownum,'test',sysdate,sysdate,'test','test',
    '','','',rownum
from dual
connect by rownum<=3200000;
insert into t2
select 'test','test','test',rownum,rownum,'test',sysdate,sysdate,'test','test',
    '','','',rownum
from dual
connect by rownum<=3200000;
commit;
execdbms_stats.gather_table_stats(ownname => 'hf',tabname => 't1',cascade
    =>true,estimate_percent => 100);
execdbms_stats.gather_table_stats(ownname => 'hf',tabname => 't2',cascade
    =>true,estimate_percent => 100);
```

（2）模拟场景

相关代码如下：

```
select * from t1 where id>= 3199990;
11 rows selected.
-------------------------------------------------------------------------------
| Id | Operation                    | Name        |Rows |Bytes|Cost (%CPU)|  Time    |
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |             | 11  | 693 |   4  (0) | 00:00:01 |
|  1 | TABLE ACCESS BY INDEX ROWID| T1          | 11  | 693 |   4  (0) | 00:00:01 |
|* 2 | INDEX RANGE SCAN           |SYS_C0025294| 11  |     |   3  (0) | 00:00:01 |
-------------------------------------------------------------------------------
Statistics
----------------------------------------------------------
1  recursive calls
0  db block gets
6  consistent gets
0  physical reads
```

对于普通的采用数值类型的字段，范围查询就是正常的索引范围扫描，执行效率很高。

```
select * from t2 where id>= '3199990';
755565 rows selected.
-------------------------------------------------------------------------
| Id  | Operation          | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |      |  2417K|   149M|  8927   (2)| 00:01:48 |
|*  1 |  TABLE ACCESS FULL| T2   |  2417K|   149M|  8927   (2)| 00:01:48 |
-------------------------------------------------------------------------
Statistics
----------------------------------------------------------
```