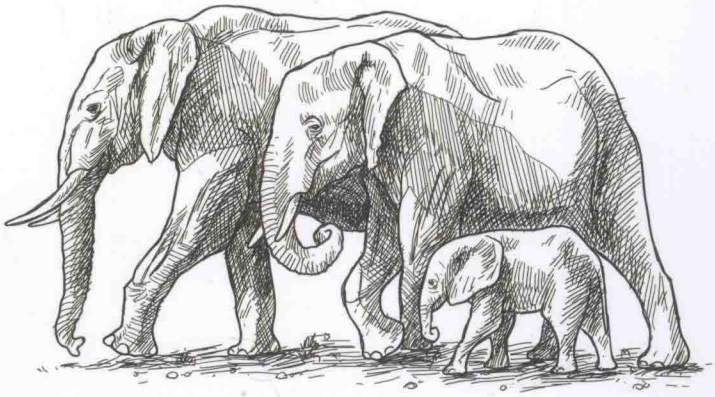
腾讯TDSQL数据库技术专家、MySQL技术专家力荐

Broadview
www.broadview.com.cn



PostgreSQL

查询引擎源码技术探析

以内核开发人员的角度抽丝剥茧，带您深入浅出PostgreSQL查询引擎内核技术内幕

李浩 编著

中国工信出版集团

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

PostgreSQL

查询引擎源码技术探析

李浩 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

PostgreSQL 作为当今最先进的开源关系型数据库，本书揭示 PostgreSQL 查询引擎的运行原理和实现技术细节，其中包括：基础数据结构；SQL 词法语法分析及查询语法树；查询分析及查询重写；子链接及子查询处理；查询访问路径创建；查询计划生成，等等。以深入浅出的方式讨论每个主题并结合基础数据结构、图表、源码等对所讨论的主题进行详细分析，以使读者对 PostgreSQL 查询引擎的运行机制及实现细节能有全面且深入的认识。

本书适合从事数据库领域相关研究人员、高等院校相关专业高年级本科生或研究生阅读，也可作为高等院校的数据库原理课程的有益补充，还可作为业界数据库相关人员的案头图书。本书有助于读者理解数据查询引擎内核的技术内幕。

图书在版编目（CIP）数据

PostgreSQL 查询引擎源码技术探析 / 李浩编著. —北京：电子工业出版社，2016.8
ISBN 978-7-121-29481-5

Ⅰ. ①P… Ⅱ. ①李… Ⅲ. ①关系数据库系统 Ⅳ. ①TP311.132.3

中国版本图书馆 CIP 数据核字（2016）第 173643 号

责任编辑：陈晓猛
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×980　1/16　印张：21.25　字数：408 千字
版　　次：2016 年 8 月第 1 版
印　　次：2016 年 8 月第 1 次印刷
印　　数：3000 册　　定价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819　faq@phei.com.cn。

前　言

随着移动互联的飞速发展，“数据”已成为当今最宝贵的资源；谁掌握了数据，谁就掌握了无尽的宝藏。而如何有效地管理这些海量数据则成为摆在人们面前的首要问题。从计算机出现以来，人们便孜孜不倦地追求着高效管理数据的办法，IBM 的 System R，U.C. Berkeley PostgreSQL 以及 Oracle MySQL 的诞生，无一不表明人们对于高效、快捷的数据管理的不懈追求。

虽然 Oracle、MySQL 广泛应用于国内外各大互联网公司的基础架构中，但作为另一款优秀的开源关系数据库，PostgreSQL 同样也得到了各大互联网公司的持续关注；另外随着大数据平台对 SQL 标准支持的日益丰富，SQL on Hadoop 的各种解决方案如雨后春笋般涌现，例如 Hadoop Hive、Facebook Presto、Cloudera Impala 等，这些 SQL on Hadoop 解决方案无一例外地需要一个表现优异的 SQL 查询引擎的支持。

作为数据库的大脑，查询引擎的优劣直接决定了数据库管理系统的好坏。不同的查询引擎对相同查询语句的处理策略可能截然不同，而其导致的执行效率也千差万别，少则相差数倍，多则数百倍。因此，认真地分析设计优秀的查询引擎并学习其查询优化方法可以使我们能够以“巨人肩膀之上”的方式来提升我们自己数据库产品的查询效率。

PostgreSQL 作为一款优秀的开源关系型数据库管理系统，其源自 U.C. Berkeley，带有浓郁的学术气息，数据库相关理论贯穿于整个 PostgreSQL 的设计和实现中，尤以查询引擎更甚。当前，无论中文或是英文资料，对 PostgreSQL 查询引擎的介绍非常稀缺，仅有的相关资料要么是限于整体框架性的介绍，要么是粗浅的概述性描述。对 PostgreSQL 查询引擎中较多的实现及其相对应的理论基础并无较为深入的讲解，这使得很多相关内核开发人员初次阅读查询引擎源码时存在许多的学习难点和疑点。例如，查询引擎在子链接处理时的理论依据；Lateral Join 的处理方法；约束条件下推时需满足的条件；选择率对查询计划的影响等。而这也正是作者最初在阅读 PostgreSQL 查询引擎源码时所经历过的迷惑和不解。

本书的写作初衷正是为了加快数据库开发人员对 PostgreSQL 查询引擎的学习过程以及减少相关人员在源码学习中的迷惑，同时本书也为那些想一窥查询引擎究竟的 DBA 们提供

一个相互学习的机会和渠道，无论你是 MySQL DBA 或是 Oracle DBA。

读者定位

本书尝试以一种全新的角度给出对 PostgreSQL 查询引擎的分析，笔者作为一名数据库内核开发人员，了解内核开发人员关注的重点是什么。因此，本书以一位内核开发人员和架构师的角度来探讨如何设计并完成一个数据库查询引擎；数据库查询引擎在设计和实现过程中需要考虑哪些问题，又通过什么样的途径和方法来解决这些问题。例如，如何以数据结构来表示一条 SQL 查询语句？如何将 SQL 查询优化理论应用到查询引擎的实现中？相信读者在阅读本书后，能产生同样的思考。

本书主要面向的受众：首先是数据库内核开发人员。无论你是 MySQL 开发人员还是 PostgreSQL 开发人员，亦或是 Infomix 开发人员。一个优秀的查询引擎总是值得你花费一定的时间和精力进行研究并学习其设计和实现中的精华。其次，数据库 DBA 们同样也值得花费一定的时间和精力来阅读和研究查询引擎源码，所谓知彼知己，百战不殆。只有更好地了解内核中的理论基础和实现机制，我们才可能管理好数据库。最后，对于那些对数据库内核实现感兴趣的相关人员，无论您是经验丰富的老手还是初出茅庐的新手，我想本书也能够为想要了解 PostgreSQL 查询引擎的实现内幕的您提供一丝帮助。同样，对于高年级的本科生或是低年级的研究生，相信本书也可作为您学习数据库理论的有益补充。

当然，您在阅读本书之前还需要有一些必要的知识：对 SQL 标准有必要的认识和了解；了解数据库原理中的相关基本概念，例如查询计划、索引等；C/C++编程知识等。

本书组织

第 1 章以如何理解并执行查询语句为例，概括性地描述 PostgreSQL 查询引擎包含的相关模块以及各个模块的功能。

第 2 章从内核开发人员的角度出发重点介绍描述一条 SQL 查询语句需要的相关数据结构。

第 3 章主要讨论对一条 SQL 查询语句的识别以及相关知识，并以此为基础重点讨论将该字符串形式的 SQL 查询语句转为查询树的过程。

第 4 章将以第 3 章中所获得的查询树为基础，讨论如何对该查询树进行查询逻辑优化。例如，对 SubLinks 的优化处理，对 SubQueries 的处理，对表达式的优化，对约束条件的处

理，对 Lateral Join 的处理，等等。

第 5 章以查询中涉及的基表的物理参数为基础，依据查询代价来计算查询语句的最优查询访问路径的方法，并对物理优化中使用的相关技术和理论基础进行详细的讨论和分析。例如，所有可行查询访问路径的求解策略，多表连接的处理策略，索引创建和选择的策略，物理代价相关参数的分析，等等。

第 6 章以第 5 章中所获得的最优查询访问路径为基础，重点讨论如何依据该查询访问路径构建执行引擎所需的查询计划。例如，顺序扫描查询计划的构建，连接关系查询计划构建等。

第 7 章主要分析我们在源码阅读过程中遇到的一些重要辅助函数。

错误说明

由于笔者水平有限，本书中会存在一定的错误，例如笔误、理解错误等，对于本书中出现的错误，读者可以在 www.leehao.org 中查询该错误的勘误信息，或者发送邮件至 hom.lee@hotmail.com 与作者联系。笔者非常希望能够与读者共同进步，共同推动国内数据库内核开发人员对 PostgreSQL 查询引擎的认识。相比 MySQL 而言，PostgreSQL 的相关资料非常少，而专门介绍 PostgreSQL 查询引擎之类的资料则更加少，而这也是本书成书的原因。

学习资料

源代码作为最好的学习资料，应该永远值得我们认真对待和重视，本书所有分析均基于 PostgreSQL 9.4.1 版本。读者可以在 https://github.com/postgres/postgres 中下载最新源码。当然，最新版本的源码可能与本书讨论中给出的源码有所不同，但这并不影响我们对主题问题的讨论。

为了了解最新特性或想参与 PostgreSQL 内核开发，读者可以订阅 PostgreSQL 邮件列表，其中包括开发人员邮件列表、本地化相关邮件列表、相关 Bugs 邮件列表等。

- http://www.postgresql.org/list/pgsql-cluster-hackers/ 集群相关内核开发人员邮件列表；
- http://www.postgresql.org/list/pgsql-committers/ 内核 commiters 邮件列表，为内核日常技术问题讨论，读者可从中了解内核 commiters 的最新动态；
- http://www.postgresql.org/list/pgsql-hackers/ 内核开发人员邮件列表；

- http://www.postgresql.org/list/pgsql-interfaces/ 相关接口讨论邮件列表，例如 odbc、jdbc 等；
- http://www.postgresql.org/list/pgsql-docs/ 相关文档邮件列表；
- http://www.postgresql.org/list/pgsql-bugs/ 相关 bugs 邮件列表。

相关邮件列表并不限于上述几类，具体内容还请读者参考 http://www.postgresql.org/list/ 中给出的详细信息。

致谢

本书在编写过程中得到了许多朋友的关心和帮助。首先，感谢北大方正信息产业集团基础软件中心（上海）的小伙伴们：王博、王鑫、蒋灿、彭川、罗正海、黄诚一、刘慧娟、何奇、刘钰，等等。正是他们的鼓励和帮助，我才有完成本书的勇气和动力。同时，还要感谢基础软件中心（上海）关健和陈敏敏两位领导。

腾讯 TDSQL 技术专家赵伟、Oracle MySQL 技术专家赖铮阅读了本书的书稿并给出了许多具有洞察力的建议和意见，使得本书增色不少。同样，两位数据库内核专家也为本书撰写了精彩的评论，两位对本书的谬赞让我诚惶诚恐，唯恐书中的错误和不足让两位专家的鼓励蒙羞。两位专家作为我的好友，其毋庸置疑的技术能力和为人、做事一直是我前进路上的榜样。在此，对二位的鼓励表示真挚的感谢。

由书稿到铅字出版，离不开本书的责任编辑博文视点陈晓猛编辑的辛勤工作。无论从本书主题、版式风格，到稿件的修改等诸多方面都体现了晓猛编辑出色的业务能力和辛勤的劳动。本书能够顺利出版离不开他的辛劳，在此表示衷心感谢。

同样要对我成长路上的诸多师长和同学及友人表达最衷心的感谢，正是他们的谆谆教诲和陪伴，我才可以自由地追逐梦想。

李浩

目　录

第 1 章　PostgreSQL 概述

1.1　概述

PostgreSQL 作为关系数据库中学院派的代表，在 U.C. Berkeley 完成了初始版本，其后 U.C. Berkeley 将其源码交于开源社区，PostgreSQL 现由开源社区对其进行维护。PostgreSQL 代码具有简洁、结构清晰、浓重的学院派气息等特性。虽然，其在国内并未像 MySQL 一样广泛在互联网公司内部使用，但是随着国内对 PostgreSQL 的认识加深，越来越多的公司逐渐采用 PostgreSQL 作为其解决方案中数据的基础架构部件；更有许多公司在 PostgreSQL 的基础上进行二次开发来满足自己的需求，例如 TeraData 公司的 AsterData 产品，EMC 公司的 GreenPlum 产品等产品均在 PostgreSQL 基础之上进行架构来实现 MPP（Massively Parallel Processing，MPP）应用，PostgreSQL-XL/XC 同样也是一款基于 PostgreSQL 的 MPP 产品；Alibaba 云计算平台也已采用 PostgreSQL 作为 RDB（Relational Database，RDB）的基础构件。同样，Fujitsu、EnterpriseDB 以及 2ndQuadrant 等均提供对 PostgreSQL 的技术解决方案。

同时，随着数据仓库（Dataware house）及 Bussinesss Intelligence（BI）等对 PostgreSQL 处理能力要求的提高，众多开源界内核开发人员以单机 PostgreSQL 为基础，构建基于 PostgreSQL 的大规模分布式应用 PostgreSQL-XL 及 PostgreSQL-XC。上述所有案例无一不表明虽然在 MySQL 大行其道的情况下，PostgreSQL 仍然在开源关系型数据库市场中占有一席之地并值得我们给予其足够的重视。

作为数据库内核中的重要一环，查询引擎在整个数据库管理系统中起到了“大脑”的作用。查询语句是否以最优的方式来执行等均与查询引擎有着密不可分的联系；不同的数据库对同一条查询语句的执行时间各不相同，有快有慢。究其原因，除了存储引擎之间的差别，查询引擎生成的查询计划和执行计划的优劣直接影响数据库在查询时的性能表现。不同的查询引擎产生的查询计划千差万别，表现在查询效率上也千差万别。例如，查询语

句中的连接操作（Join Operation），不同的查询引擎产生的优化策略会导致执行时间存在着数倍甚至数百倍的差距。

作为学院派代表的 PostgreSQL 有着一套复杂的查询优化策略，例如对子链接的处理，基于代价的优化策略，基于规则的查询优化策略等。对于这些优化策略，PostgreSQL 并非墨守成规，而是也将这些优化策略的实现接口开放给第三方的内核开发者，使得用户可以灵活地使用适用于特定应用场景的自有优化策略。例如，pg_rewrite 中描述的基于查询语法树的改写（Rewrite）规则 pg_rules，等等。

作为连接服务器层（Server Framework）与存储引擎层（Storage Engine）的中间层，查询引擎将用户发送来的 SQL 语句按照 scan.l 和 gram.y 中预先定义的 SQL 词法（Lexcial Rules）及语法规则（Grammatic Rules）生成查询引擎系统内部使用的查询语法树形式（Abstract Syntax Tree，AST），查询引擎会将该查询语法树进行预处理：将其转换为查询引擎可处理的形式——查询树 Query。

在由语法树到查询树的转换过程中，查询引擎会将查询语句中的某些部分进行转换。例如，“*”会被为被扩展为相对应关系表的所有列，并在后续转换的过程中，根据语法树所标示的类型进行分类处理，如 SELECT 类型语句、UPDATE 类型语句、CREATE 类型语句等。

在查询引擎语法树到查询树转换后，PostgreSQL 查询引擎会使用 pg_rewrite 中设定的转换规则进行所谓的基于规则的转换，例如，PostgreSQL 查询引擎会将 VIEW 进行转换，为后续的优化提供可能。

在完成了基于规则的优化后，PostgreSQL 查询引擎进入到我们称之为逻辑优化的阶段。在该阶段中，PostgreSQL 查询引擎将完成对公共表达式的优化，子链接的上提，对 JOIN/IN/NOT IN 的优化处理（进行 Semi-Join、Anti-Semi-Join 处理等），Lateral Join 的优化等优化操作。

在执行上述优化操作中，我们将遵循一条“简单”法则：先做选择运行（δ Operation），后做投影运算（π Operation）。经过此阶段的优化操作后，所得到的查询树为一棵遵循了先选择后投影规则的最优查询树，并以此为基础构建最优查询访问路径（Cheapest Access Path）。

在完成了对查询树的优化处理并获得最优查询访问路径后，PostgreSQL 查询引擎接下

来要做另外一件非常重要的事情是查询计划的生成（Plans Generating）。PostgreSQL 查询引擎会依据最优查询访问路径，通过遍历该查询访问路径，来构建最优查询访问路径对应的查询计划（Query Plans or Plans）。

在查询计划的生成过程中，PostgresQL 查询引擎会在所有可行的查询访问路径中选择一条最优的查询访问路径来构建查询计划。不同方式所构建的查询访问路径的代价不尽相同，例如，执行多表 JOIN 操作时，不同的 JOIN 顺序产生的查询访问路径不同，而这直接导致了查询访问路径中的中间元组规模的不同；同时，关系表上索引的有无也将影响查询访问路径的代价，不同的表扫描方式将会极大地影响执行效率。

通常，我们依据 COST = CPU_cost + IO_cost 公式来选择一条最优的执行路径，其中，CPU_cost 表示执行该条执行计划需要的 CPU 代价，IO_cost 则为相应的 I/O 代价（启动代价，这里我们将其计入到 IO_cost 中）。

综上所述，一个查询引擎应该包括：查询语句接收模块、词法解析模块、语法解析模块、查询树改写模块（规则优化模块）、查询优化模块（包括逻辑优化和物理优化两部分）、查询计划生成模块、元数据管理模块、访问控制模块等基本模块。当然不同的查询引擎在实现时，这些模块的划分可能不同，但是一个普通的查询查询都应含有上述模块，图 1-1 为一个常规的查询引擎架构图。

1.2　查询语句优化

当查询引擎接收到一条用户查询请求后，查询引擎会依据该查询语句的类型进行分类处理；但在处理查询语句之前，考虑到复杂查询语句求解最优访问路径时的代价，有些查询引擎会使用查询计划缓存机制（Query Plans Caching 或 Query Paths Caching）：数据库管理系统提供原生的最优查询访问路径代价缓存机制或使用第三方的查询计划缓存解决方案。但在使用此缓存机制时需要注意：查询语句需满足一定条件，例如满足不含有易失函数（Volatile Function），语句中涉及的基表定义发生变化后的正确处理等条件后，才能对其使用缓存机制，否则可能导致查询结果不正确。

查询引擎对不同类型的查询语句有着不同的处理机制，对于工具类查询语句以及非工具类查询语句，PostgreSQL 有着截然不同的处理流程。

图 1-1　查询引擎架构图

1.2.1　工具类语句

当查询语句为工具类查询（Utility Statements）语句时，查询引擎将经过词法分析和语法分析后获得的查询语句作为其执行计划。工具类查询语句由 ProcessUtility 函数调用 standard_ProcessUtility 依据该语句的类型进行分类处理。例如，对于 CreateTableSpace、Truncate、PrePare、Execute、Grant 等命令，查询引擎将分别使用 CreateTableSpace、ExecuteTruncate、PrepareQuery、ExecuteQuery、ExecuteGrantStmt 等函数对这些命令进行分类处理。那么哪些语句可归为工具类语句呢？

PostgreSQL 将如下语句归为工具类型语句并将其交由 standard_ProcessUtility 函数处理。

工具类语句中包含：事务（Transaction）类语句，例如，开始事务、提交事务、回滚

事务、创建 SavePoint 等；游标（Cursor）类语句，例如，打开游标、遍历游标、关闭游标等；内联过程语句类语句（Inline Procedural-Langauge）；表空间（TableSpace）操作类型语句，例如，创建表空间、删除表空间、修改表空间参数等；Truncate 类语句；注释类语句；数据库对象安全标签类语句（Security Label to a Database Object）；SQL Copy 类语句；Prepare 类型语句；权限或角色操作相关类语句；数据库操作类语句，例如，创建数据库、删除数据库等；索引维护类语句；Explain 语句；Vacuum 语句。

PostgreSQL 调用相应的命令处理函数对上述工具类语句进行分类处理，因此，对于 standard_ProcessUtility 函数的实现，读者可轻松地猜到如下的实现方式，对应其中的某类具体实现，在这里就不再详细给出，还请读者自行分析。函数原型如程序片段 1-1 所示。

程序片段 1-1　standard_ProcessUtility 函数的原型

```
void
standard_ProcessUtility(Node *parsetree,
          const char *queryString,
          ProcessUtilityContext context,
          ...
          )
{
  switch (nodeTag(parsetree))
  {
    case T_TransactionStmt: {...} break;
    case T_PlannedStmt: {...} break;
    ...
    case T_DropTableSpaceStmt:{...}break;
    ...
    default:{...} break;
  }
}
```

1.2.2　查询类语句的处理

对于非工具类查询语句，即普通查询类语句，除了经历与工具类查询语句一样的语法分析过程和词法分析过程，还需完成：将原始语法树转换为查询语法树；以查询语法树为基础对其进行逻辑优化；对查询语句进行物理优化；查询计划创建等过程。

经过词法分析（Lexical Processing）和语法分析（Grammatical Processing）后，PostgreSQL

需要将原始语法树转换为查询语法树并在转换过程中进行语义方面的合法性检查。例如，基表（Base Relation）的有效性检查，目标列（Target List）的有效性检查及展开，基表的 Namespace 冲突检查等。

transformStmt 函数依据查询语句的类型进行相应语法树到查询树的转换工作，例如，由 transformSelectStmt 函数完成对 SELECT 类型查询语句的转换操作，由函数 transformInsertStmt 完成对 INSERT 类型语句的语法树的转换。

查询引擎将对 SELECT 类型查询语句中不同的语法部分进行分类处理。由 transformTargetList 函数对目标列子句进行转换处理；transformWhereClause 函数完成 WHERE、HAVING 子句的语法树转换处理；transformLimitClause 函数完成 Limit 和 Offset 语句的转换工作；transformSortClause 函数和 transformGroupClause 分别完成对 ORDER BY 语句及 GROUP BY 语句的转换。经过上述转换后，我们将获得一棵（或数棵）由原始语法树转换而得到的 Query 类型查询树，并以此为基础进入到查询优化的下一阶段：基于规则的查询改写。

原始语法树经过上述转换操作后，查询引擎获得 Query 类型的查询树，接下来，查询将依据系统中定义的规则，对该查询树进行依据规则的改写操作，例如，视图的改写等。元数据表 pg_rules 中描述了当前系统中具有的规则说明。除了使用 CREATE RULE、ALTER RULE、DROP RULE 命令来维护该规则系统，我们还可以通过“暴力”手段，直接修改 pg_rules 元数据表来“维护”规则系统。在完成基于规则的改写后，查询引擎将进入下一阶段的优化：查询逻辑优化（Logical Optimization）。

逻辑优化阶段中，会对所有导致查询变慢的语句进行等价变换，依据数据库理论中给出的经典优化策略：选择下推，从而尽可能减少中间结果的产生。即所谓的先做选择操作，后做投影操作。优化原则如图 1-2 所示。

首先，查询引擎由函数 pull_up_sublinks 分别对 IN 和 EXISTS 类型子链接（SubLink）进行优化处理：将子链接转为 SEMI-JOIN，使得子链接中的子查询有机会与父查询语句进行合并优化。函数 pull_up_sublinks 中，PostgreSQL 在确定子链接满足 SEMI-JOIN 转换的条件后，分别由 convert_ANY_sublink_to_join 函数及 convert_EXISTS_sublink_to_join 函数将 IN 和 EXISTS 类型的子链接转换为 SEMI-JOIN 类型的 JOIN 连接。经过转换后，查询效率较低的 IN/EXISTS 子链接操作转换为查询效率较高的 JOIN 操作。