Oracle 查询优化在三甲医院的应用研究

王山林 [1]
WANG Shanlin

摘 要 随着医疗信息化建设的持续推进，桂林医科大学第二附属医院电子病历系统（EMR）、影像归档与通信系统（PACS）、检验信息系统（LIS）等核心业务系统均依托 Oracle 数据库存储海量业务数据，数据库查询性能瓶颈问题日益突出。结合该院数据中心实际应用场景，文章阐述了 Oracle 数据库查询优化的基本原理，包括 SQL 执行流程与执行计划解析，并在此基础上提出合理设计索引、SQL 语句重构、数据分区及使用绑定变量四种优化策略，结合典型应用案例验证了优化效果。实践表明，经优化后数据库查询响应时间明显缩短，系统运行稳定性与资源利用率显著提升，可为医院信息系统的高效稳定运行提供技术支撑。

关键词 Oracle；查询优化；数据库

doi：10.3969/j.issn.1672-9528.2026.05.029

0 引言

信息化技术的不断发展，Oracle 数据库凭借其高可靠性、高安全性及强大的事务处理能力等优势，已被广泛应用于各类信息系统，作为底层核心数据存储平台。桂林医科大学第二附属医院的多项重要业务系统，均基于 Oracle 数据库开展日常业务运行与数据处理。

随着医院诊疗规模扩大、患者数量增多，业务数据量急剧增加，对 Oracle 数据库造成较大负担，在 SQL 的操作中，大部分为查询操作，而这些查询操作的速度快慢又直接影响整个数据库的应用效率以及临床工作的顺利进行，所以查询优化是提高 Oracle 数据库性能的关键。

本文根据医院实际情况以及在运维过程中遇到的问题，介绍查询优化的概念、方法和技术手段，提出可以实施的方法并用具体案例进行说明优化的效果。

1 Oracle 查询优化基础

理解查询优化基础，需从 Oracle 查询执行过程切入。

1.1 SQL 查询语句执行过程

SQL 查询语句的执行主要包含解析、优化、行源生成和执行四个阶段 [1]，如图 1 所示。

（1）解析阶段：用户提交 SQL 查询后，Oracle 首先进行语法分析，校验语句是否符合 SQL 语法规则；随后执行语义验证，检查语句涉及的表、列、函数等数据库对象是否存在，并验证用户是否具备对应的访问权限。完成语义验证后，数据库会进行共享池检索：通过 SQL 语句哈希值查询共享池中是否存在已缓存的解析结果（软解析）。若存在，则直接复用解析树与执行计划；若不存在，则触发硬解析流程，生成新的解析树并缓存至共享池 [2]。

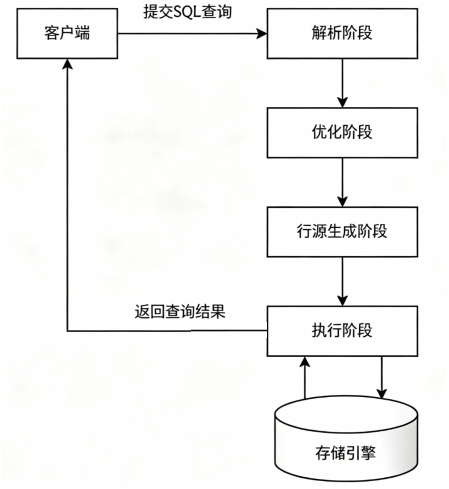

图 1 SQL 查询语句执行图

（2）优化阶段：基于成本的优化器（CBO）依据表数据量、索引分布、数据倾斜特征及表连接顺序等统计信息，对解析树进行逻辑优化。通过生成多套候选执行计划（如不同的索引策略、连接算法组合），优化器从数据访问路径（全表扫描 / 索引扫描）、表连接方式（嵌套循环 / 哈希连接 / 排序合并连接）、执行并行度等维度，计算各方案的 I/O 与 CPU 成本，最终选择总成本最低的执行计划 [3]。

（3）行源生成阶段：行源生成阶段是把优化器产生的逻辑执行计划转化为可以被计算机理解并执行的具体物理执行模型（行源树）。树形结构是由很多底层的操作节点构成，

1. 桂林医科大学第二附属医院 广西桂林 541199

在每一个节点上都有一个确定的数据读取及操作的方法，比如全表扫描（TABLE ACCESS FULL）、索引范围扫描（INDEX RANGE SCAN）、哈希连接（HASH JOIN）等。又以一定的顺序串联起来形成一条数据流动路径，供之后的执行使用。

（4）执行阶段：根据行源树进行表扫描、索引访问、连接等操作，在内存或者硬盘中读取数据，经过计算后通过 SQL*Net 发送给客户端结果集。

当 Oracle 数据库处理一个 SQL 查询时，是一个包括解析、优化、行源生成以及执行的过程，最主要的环节是优化这一步骤，可以把用户的业务逻辑 SQL 转为最优执行方案，而这个最优执行方案就是查询计划（Execution Plan）[4]。

1.2 执行计划

执行计划是 Oracle 优化器根据 SQL 语句语义、数据库对象结构（如表、索引）、统计信息（如数据量、数据分布情况）等信息，计算得出的最优执行步骤。作为分析与优化 Oracle SQL 查询性能的核心工具，执行计划可直观反映表的访问方式（如全表扫描、索引扫描）、表间连接顺序与方式（如嵌套循环连接、哈希连接、合并连接），以及数据过滤、排序、分组等操作的执行顺序与成本 [5]。

（1）执行计划的组成（如表 1 所示）

表 1 执行计划的组成

字段	含义
Id	执行步骤序号
Operation	当前步骤操作类型
Name	关联对象（表名、索引名、分区）
Rows	返回行数
Bytes	返回总字节数
Cost	成本开销
Time	执行时间
Predicate Information	条件过滤信息

（2）基于执行计划的查询优化思路

执行计划可以显示数据库查询的真实执行过程以及相关操作，有利于发现慢查询存在的问题，从而给出具体的改善建议以提高查询效率。

①定位高耗时、高成本节点 [6]：首先查看 Time 列，找出耗时最长节点；如果缺少真实耗时，则可以以 Cost 最大的节点为起点，但是要结合 Rows 进行判断（防止由于估算不准造成误判）。

②检查访问路径以及连接方式 [6]：根据 Operation 字段判断访问路径以及连接方式是否合理：

访问路径：

① TABLE ACCESS FULL（全表扫描）：一般由于缺乏合适索引或者查询条件的选择性较低引起的，需要考虑是否添加相应的索引；

② INDEX RANGE SCAN（索引范围扫描）：性能较好，在有范围条件情况下使用（如 >、<、BETWEEN）；

③ INDEX FULL SCAN / FAST FULL SCAN（全索引扫描）：虽然可以避免全表扫描，但是代价较大，要排除是否由于索引设计不合理造成的。

如果索引没有被使用应该首先考虑：查询条件是否可以利用该索引（比如是否对索引列进行函数操作等），以及是否因为统计数据不准确造成优化器选择了错误索引。

表连接方式：

① NESTED LOOPS（嵌套循环连接）：适合小型数据集连接，在大数据集上效率很低；

② HASH JOIN（哈希连接）：适用于大数据量连接但是需要大量内存空间；

③ MERGE JOIN（合并连接）：用于连接双方数据已经排好序的情况。

如果发现连接方式不合理，可以再次进行统计信息收集或者通过 Hint（例如：/*+ USE_HASH */）提示优化器选取更好的方法。

（3）关注排序以及临时空间使用 [6]：

如果操作中有“SORT”（如“SORT ORDER BY”“SORT AGGREGATE”），表示这个查询有排序的操作，而如果排序的数据量较大，则需要使用大量的临时表空间，容易成为性能瓶颈，此时可以通过增加合适的索引（比如包含排序字段的组合索引等）来减少不必要的排序。

执行计划是优化器“决策结果说明书”，即了解优化器内部优化机制窗口。Oracle 优化器主要原则（例如成本计算方法、访问路径选取条件、连接方式倾向等），都可以从执行计划的操作类型、预计总共字节数、成本组成等方面体现出来。根据这些可以还原出优化器是如何进行选择，从而找到影响 SQL 性能问题并对其进行改进措施 [7]。

2 Oracle 查询优化方法

通过对计划分析了解优化器规则之后，可以修改 SQL 或者数据库对象，影响优化器决策过程，让其自主选择最优执行方案，从而提高查询效率。

2.1 合理设计索引

在数据库性能优化体系中，索引是数据检索的核心辅助结构，通过对表内一列或多列值进行有序组织，构建类似书籍目录的快速定位机制，进而显著提升数据查询效率。其中，B 树索引是最常用的类型，其采用多路平衡树结构（区别于二叉树），通过分层存储键值与行地址（ROWID）的映射关系，实现对目标数据行或行集的高效访问，尤其适用于高频出现在查询谓词中的高选择性列 [8]。

索引创建的核心策略与适用场景

（1）聚焦高价值查询列

优先在以下场景的列上创建索引：

①高频作为 WHERE 子句查询条件的过滤列（如患者信

息表的 patient_id、就诊记录表的 visit_date）；

②表连接操作中的关联列（如 JOIN 语句中用于匹配的外键列）；

③ ORDER BY/GROUP BY/DISTINCT 等排序、聚合操作涉及的列。

这类列一般有较高查询选择性（即过滤后返回的行数占表总的行数的比例较小），索引可以帮助减少扫描的数据量。

（2）规避低选择性与高更新场景

①对于数据重复性强的列（如性别列 gender、状态码列 status），索引的性能增益有限。实践表明，当目标数据量占表总量的比例低于 5%~10% 时，索引检索的 I/O 成本显著低于全表扫描；若超过该阈值，全表扫描可能因避免索引回表开销而更高效。

②谨慎在高频更新（如 INSERT/UPDATE 频繁）的列上创建过多索引。B 树索引的结构特性决定了每次数据变更需维护索引树平衡，过度索引会增加数据写入延迟。

③避免对索引列施加函数或计算操作（如 TO_CHAR(create_time, 'YYYY') 或 price×0.8）。B 树索引存储的是原始键值与行地址（ROWID）的映射，对索引列的任何计算都会导致查询条件与索引键值不匹配，进而强制触发全表扫描。

SQL 示例及效果验证：

医院的患者信息表（pats_index）中，需频繁根据身份证号（id_card）查询患者信息。原表未创建索引时，查询语句如下：

```
SELECT * FROM  pats_index
WHERE id_card = '110101199001011234';
```

执行该查询时，数据库会触发全表扫描，执行计划中 Access Type 显示为 TABLE ACCESS FULL，成本值较高。为 id_card 列创建 B 树索引后：

```
CREATE INDEX idx_patient_id_card
ON pats_index(id_card);
```

再次执行上述查询，执行计划中 Access Type 变为 INDEX RANGE SCAN，成本值显著降低，查询响应时间从原 5 s 缩短至 0.2 s，查询效率大幅提升。

2.2 SQL 语句重构

不合理的 SQL 语句结构会导致查询效率低下。通过重构 SQL 语句（如将子查询转换为连接查询以减少不必要的嵌套、用 EXISTS 替代 IN、合理使用聚合函数与分组查询等），可提升查询效率。优化原理为：重构后的 SQL 语句能减少不必要的数据处理与计算量，使数据库系统更高效地执行查询操作 [9]。

需注意：子查询分为相关子查询与非相关子查询。其中，相关子查询会根据外部查询返回的每一行数据循环执行，执行次数等同于外部查询返回的行数；而连接查询可减少此类循环执行次数，进而提升效率。

SQL 示例及效果验证：

原查询语句通过子查询获取确诊为糖尿病且年龄大于 60 岁的患者信息：

```
SELECT * FROM pats_index WHERE patient_id IN (SELECT patient_id FROM diagnosis WHERE diagnosis_desc like '% 糖尿病 %' ) AND age > 60;
```

该查询执行时间较长，执行计划显示子查询被多次执行，总成本值较高。将其重构为连接查询：

```
SELECT p.* FROM pats_index p JOIN diagnosis d ON p.patient_id = d.patient_id WHERE d.diagnosis_desc like '% 糖尿病 %'  AND p.age > 60;
```

重构后，执行计划中无子查询的多次执行记录，成本值降低，查询响应时间从原 8 s 缩短至 1.5 s。

2.3 数据分区

当表数据量一般超过 2 GB 后，普通索引优化作用大打折扣，这时就需要分区来提高查询效率。它是把一个大表或者索引分成若干个小部分管理，每个小部分存放在不同的表空间中，在外表上看似一个整体，在内部被分为许多部分，对大量数据读取起到良好作用—应用程序完全不知道存在分区，进行查询时数据库会自动找到相应分区，不需要遍历整个表 [10]。

（1）分区键：数据分布的核心规则

创建分区表的首要步骤是定义分区键，其决定数据的物理存储路径：

①优先考虑经常出现在 WHERE 条件、JOIN 关联或排序聚合中的列（如时间字段、分类标识）；

②执行数据操作（DML）时，数据库根据分区键自动把记录导向相应的分区，不需要人为地进行；

③根据数据特点选取合适的分区键：对于时间序列型数据采用日期字段作为分区键（如就诊日期），对于枚举值类型的数据采用状态码字段进行分区（如科室编号）。

（2）主流分区方法及典型应用

①范围分区（时间 / 数值场景首选）：根据一定范围对数据进行分块（例如年份、季度或者数值段等），每一块存放该范围内数据，在时间序列中使用较多。

②列表分区（枚举值场景最优）：当分区键是有限几个可能取值的情况时（例如科室代码、性别标记等），可以明确指定每个分区所包含的具体值，“Default”分区用来处理没有被分配到任何一个具体分区中的数据。

③哈希分区（数据均衡分布方案）：用哈希算法把数据平均分到各个分区中去，一般情况下分区数目设置成 2 的幂次方（例如：4、8、16 等），防止数据倾斜。

④组合分区（复杂场景分层管理）：“主分区 + 子分区”，如：“以年度区间为主分区 + 以科室列表为子分区”，在满足时间和业务两个方面的同时也保证良好的过滤速度以及便于操作。

SQL 示例及效果验证：

医院的诊疗记录表（pat_visits）的数据量很大，要按每年划分一个区，原来的查询语句用来查询 2023 年的诊疗记录：

```
SELECT * FROM pat_visits WHERE visit_date BETWEEN '2023-01-01' AND '2023-12-31';
```

对表按年份进行范围分区的语句如下：

```
CREATE TABLE pat_visits (
    patient_id VARCHAR2(50),
    visit_id NUMBER,
    visit_date DATE,
    -- 其他字段 ... ...
) PARTITION BY RANGE (visit_date) (
 PARTITION p2021 VALUES LESS THAN
     (TO_DATE('2022-01-01', 'YYYY-MM-DD')),
 PARTITION p2022 VALUES LESS THAN
      (TO_DATE('2023-01-01', 'YYYY-MM-DD')),
 PARTITION p2023 VALUES LESS THAN
     (TO_DATE('2024-01-01', 'YYYY-MM-DD')),
 PARTITION p2024 VALUES LESS THAN
     (TO_DATE('2025-01-01', 'YYYY-MM-DD')),
 PARTITION p_overflow VALUES LESS THAN
      (MAXVALUE)
);
```

未分区时，该查询需扫描整个表，执行时间为 12 s，执行计划显示全表扫描；分区后再次执行，执行计划显示仅扫描 2023 年对应的分区，执行时间缩短至 2 s。

2.4 绑定变量

Oracle 执行 SQL 语句时，解析方式分为硬解析与软解析两类：

（1）硬解析：若某条 SQL 语句首次执行，且在共享池中无法找到可复用的解析树与执行计划，数据库会依次进行语法解析、语义识别，并根据统计信息生成最优执行计划，最终执行 SQL——这一过程即为硬解析，耗时较长。

（2）软解析：若 SQL 语句可在共享池中找到匹配的解析树与执行计划，无需重复生成计划，直接调用执行——这一过程即为软解析，效率更高 [11]。

绑定变量的核心原理是将硬解析转化为软解析，减少 Oracle 在 SQL 解析环节的时间与资源消耗，具体做法是用：variable_name 的形式替代固定谓词值。

SQL 示例及效果验证

原两条 SQL 语句因谓词值不同，需分别执行硬解析：

```
SELECT patient_id FROM pats_index
WHERE patient_id = 'M100001';
SELECT patient_id FROM pats_index
WHERE patient_id = 'M100002';
```

改为绑定变量后，两条查询统一为同一 SQL 查询模板：

```
SELECT patient_id FROM pats_index
WHERE patient_id = :id;
```

优化后查询语句只需要进行一次硬解析即可，在类似查询请求情况下可以直接复用该执行计划而不需要再次进行硬解析，而在笔者多次院内测试中发现一般情况下可以节省大约 40% 的时间。

3 结语

本文以 Oracle 查询优化为切入点进行研究工作，在介绍 SQL 执行过程以及执行计划基础上，给出索引设计、SQL 重写、数据分区及绑定变量四种优化方式。通过实例证明优化前后数据库查询响应时间有较大降低，系统运行速度、稳定性也有很大提高，很好地缓解了大量医疗信息带来的查询问题，在智慧医院的发展过程中，可以继续引入智能化的技术对本课题进行更深层次的研究，更好地服务医院的信息系统。

参考文献：

[1] 吴洁明 , 周锦 . 基于 Oracle 数据库 SQL 查询语句优化规则的研究 [J]. 陕西理工学院学报 (自然科学版), 2013, 29(4):34-38.

[2] 郭珉 .ORACLE 数据库 SQL 优化原则 [J]. 计算机系统应用 , 2010, 19(4):170-173.

[3] 黄建军 , 龚玮玮 , 肖英剑 . 基于 Oracle 数据库查询优化策略的研究 [J]. 电脑知识与技术 ,2019,15(13):10-11.

[4] 魏玉芬 , 王玥 . 基于 ORACLE 成本优化器的 SQL 查询优化分析与应用 [J]. 内蒙古农业大学学报 (自然科学版), 2018, 39(2):88-93.

[5] 李智敏 . 面向性能的 ORACLE 数据库 SQL 调优的研究与实践 [D]. 武汉 : 武汉轻工大学 , 2018.

[6] 赵新民 , 崔海艳 . Oracle 数据库 SQL 语句优化要点分析 [J]. 信息安全与技术 , 2014, 5(6): 65-66.

[7] 孟元 , 朱林超 , 郝井勋 , 等 . 基于执行计划的 Oracle SQL 优化策略分析 [J]. 电工技术 ,2024(16):217-219.

[8] 夏忠球 , 俞国红 . Oracle 索引在数据查询中的应用 [J]. 无线互联科技 , 2017(20): 134-135.

[9] 樊新学 . 基于 Oracle 数据库的 SQL 语句优化 [J]. 办公自动化 , 2017, 22(2): 44-45.

[10] 陆家俊 , 顾梅 . Oracle 查询优化的研究与应用 [J]. 信息技术与信息化 , 2022(1): 57-60.

[11] 孟津平 . Oracle 数据库下的系统性能调整与优化的研究 [D]. 长春 : 长春理工大学 , 2018.

【作者简介】

王山林（1986—），男，广西桂林人，本科，研究方向：数据库、Web 应用开发。

（收稿日期：2025-09-18 修回日期：2026-04-17）