Automatic Physical DB Design

数据库物理模型自动设计

概要

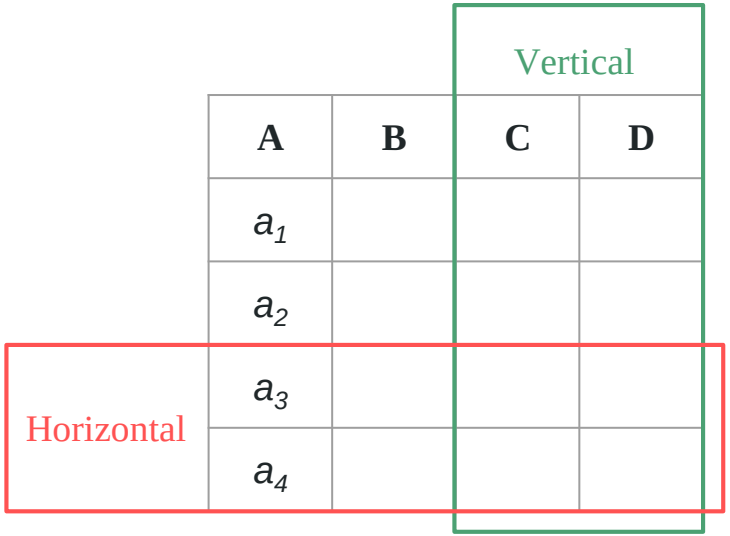
- 物理模型设计简介
- 垂直分表
- 水平分表
- 总结

物理模型设计简介

物理模型设计特性

- 分表：逻辑表 → 物理子表
  - 垂直分表：切分属性集
  - 水平分表：切分记录集
- 复制/冗余
- 顺序/主索引
- 编码/压缩

- 二级索引
  - 垂直切分+冗余+顺序
- 物化视图



垂直分表

- 给定一张逻辑表 **R(A, B, C, D)** 和一批查询:
  - Q_1: `SELECT A, B FROM R`
  - Q_2: `SELECT C, D FROM R`
- 如何物理存储表 **R** 的各列?
  - **#1** 使用一张物理表存储所有列：**P(A, B, C, D)**
    - 查询需要访问多余列，Q_1：**C, D**　Q_2：**A, B**
  - **#2** 使用多张物理表分别存储各列：**P_1(A), P_2(B), P_3(C), P_4(D)**
    - Q_1: `SELECT A, B FROM` P_1 **`JOIN`** P_2 `USING rowid`
    - Q_2: `SELECT C, D FROM` P_3 **`JOIN`** P_4 `USING rowid`
  - **#3 P_1(A, B), P_2(C, D)**
    - Q_1: `SELECT * FROM` P_1　　Q_2: `SELECT * FROM` P_2
    - Q_3: *`SELECT B, C FROM R ?`*

行存储

列存储

混合存储

水平分表

- 例：将销售记录表按年份和月份水平切分
  - **Sales → Sales-2016-01, Sales-2016-02, ...**

- 作用 #1：数据聚集（Data Clustering）
  - 将具备某些特征的数据相邻存储，以降低I/O开销
    - 特征：频繁共同访问、在某些维度上有相同值

- 作用 #2：数据跳过（Data Skipping）
  - 在查询处理时跳过不必要的数据

水平分表——分布键

- 环境：分布式**并行数据库**
  - 数据被切分存储到各个节点以支持并行处理
    - 要求为各表选取一个分布键（包含一列或多列）
    - 在分布键上执行hash分区或范围分区
- 查询处理: 避免通信开销
  - 假设 $\mathbf{R} \bowtie_{\mathbf{R.J = S.J}} \mathbf{S}$
    - ***local join***: **R** 和 **S** 都以 **J** 为分布键
    - ***broadcast join***: **R** 以 **J** 为分布键, **S** 被广播至各节点
    - ***directed join***: **R** 以 **J** 为分布键, **S** 被重新分发至各节点
    - ***repartitioned join***: **R** 和 **S** 都被重新分发至各节点
  - 优先选择: *local join* > *broadcast join* 或 *directed join* > *repartitioned join*
  - 此外：local group-by, local window functions

例：分布键选择

- 给定一个 TPC-H 数据库和一批查询：
  - Q_1: ... `lineitem JOIN orders ON l_orderkey = o_orderkey`
  - Q_2: ... `lineitem JOIN supplier ON l_suppkey = s_suppkey`

- 如何选择表 **lineitem**, **orders** 和 **supplier** 的分布键?
  - **#1: lineitem(l_orderkey), orders(o_orderkey), supplier(s_suppkey)**
    - 具有相同 *orderkey* 值的记录被划分至同一节点
    - Q_1: local join Q_2: redirected join
  - **#2: lineitem(l_suppkey), orders(o_orderkey), supplier(s_suppkey)**
    - 具有相同 *suppkey* 值的记录被划分至同一节点
    - Q_1: redirected join Q_2: local join

其它物理模型设计特性

- 复制/冗余
  - 权衡：存储开销 ↔ 性能和安全
- 顺序：为查询优化提供更多选择
  - ***ORDER BY, sort merge join, pipelined group-by***
- 编码/压缩
  - 权衡：存储开销和 I/O 开销 ↔ CPU 开销
  - 在压缩数据上执行查询

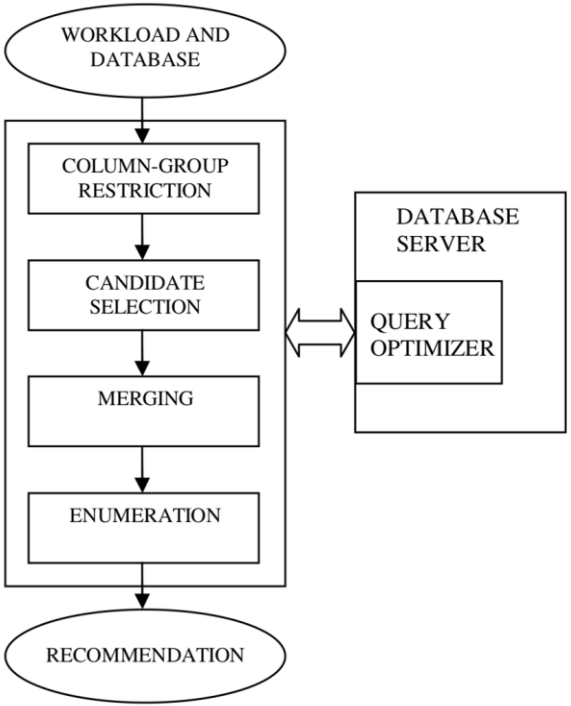
物理模型设计算法——高层抽象

- 生成一系列候选的物理模型设计方案
  - 通常使用启发式的剪枝策略缩小搜索空间
- 使用某种代价模型评估各个设计方案的收益
  - 给定一批查询，收益 = 该设计方案下的代价 – 基准代价
- 选择收益最高（代价最小）的设计方案

代价模型

- 重新设计代价模型？
  - 建模难度与查询优化器能力相关
  - 重复劳动：查询优化器内部的代价模型

- 集成查询优化器 [Agrawal'06]
  - 充分利用查询优化器的代价模型
  - 确保物理模型设计与优化器决策相匹配
  - 原理：查询优化器依赖统计信息评估执行计划的代价
    - 构建近似的统计信息，注入到元数据目录



垂直分表

- 问题描述
- 算法
- 列存储

问题描述

- 给定:
  - 原表的集合 $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$
  - 一批查询 W
  - 存储开销上界 B
- 输出一个分表的集合 $\mathcal{P} = \{P_1, P_2, \ldots, P_N\}$ 使得:
  - 每张分表 $P_k \in \mathcal{P}$ 存储 $\mathcal{R}$ 中的部分列以及附加的 id 列
  - 每张原表 $R_i \in \mathcal{R}$ 的各列至少被一张分表存储
  - $\mathcal{P}$ 的存储开销不超过 B
  - W 的执行开销最小化
- 其他可选的约束
  - 禁止冗余：每张原表 $R_i \in \mathcal{R}$ 的各列仅被一张分表 P_k 存储

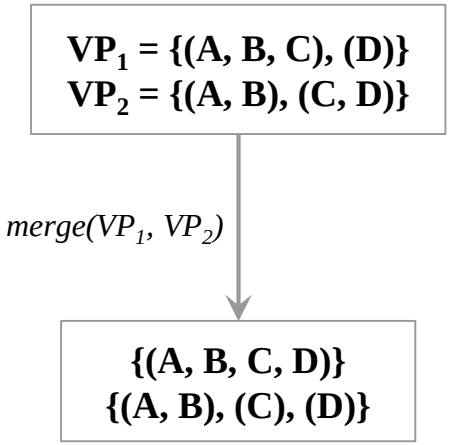
问题简化

- $\mathcal{R} = \{R_1\}$, 即垂直切分一张原表 $R_1(A_1, A_2, \ldots, A_m)$
- 要求 R_1 的每列 A_j 仅被一张分表 P_k 存储（即禁止冗余）
- 穷举法搜索空间的大小: 贝尔数 B_m
  - 划分一个集合（大小为m）的不同方法的数目, $O(e^{m\ln(m)})$

- 多种启发式算法 [Jindal'13]
  - 基本思想 #1: 利用查询中各列的“亲和度”
  - 基本思想 #2: 利用某种代价模型评估划分方法的收益
  - 搜索策略：自顶向下或自底向上
    - 自顶向下： [(A, B, C, D)] → [(A, B), (C, D)] → [(A), (B), (C, D)]
    - 自底向上： [(A), (B), (C), (D)] → [(A), (B), (C, D)] → [(A, B), (C, D)]

自底向上方法

- **Hill-Climb** 算法 [Hankins'03] (***considered the best*** [Jindal'13])
  - 起点：每一列作为一张分表
  - 每次迭代选择两张分表合并（选择合并后收益最大的两张分表）
  - 没有收益提升时停止迭代
- SQL Server **AutoAdmin** 使用的算法 [Agrawal'04]
  - 起点：感兴趣的各表划分方法
    - 由查询产生: [(频繁项集), (剩余的列)]
    - 支持度剪枝 + 兴趣度剪枝
  - 生成两两合并的划分方法
    - 通过分表的交和并生成新分表



垂直分表与列存储

- [Jindal'13] 的结论：**列存储已经足够，甚至更优**
  - 当缓冲区小于100MB时垂直分表相较于列存储有轻微优势
  - 垂直分表的优势
    - TPC-H: 3.7%, Star Schema Benchmark: 5.3%
    - 内存数据库: 0%
    - 商用的列存储数据库: -xx%（重量级压缩的开销）

- 跨表的情形：materialized pre-join view
  - 可作为物化视图选择的子问题

水平分表

- 分布键选择
- 数据聚集

水平分表——分布键选择

- 给定：
  - 数据库 $\mathcal{D} = \{R_1, R_2, \ldots, R_n\}$，其中表 R_i 包含 N_i 列 $(A_{i1}, A_{i2}, \ldots, A_{iN_i})$
  - 一批查询 **W**
  - 存储开销上界 **B**
- 输出 $\mathcal{D}$ 中所有表的划分方案 $P(p_1, p_2, \ldots, p_n)$ 使得：
  - 表 R_i 的分布键 $p_i \in \{A_{i1}, A_{i2}, \ldots, A_{iN_i}, \text{复制}\}$
    - 即可选择各表的任意列作为其分布键，或复制该表
  - 冗余表的总体存储开销不超过 **B**
  - **W** 的执行开销最小化
- 穷举： $(N_1 + 1) * (N_2 + 1) * \ldots * (N_n + 1)$

基本思想/简单方法 [Zilio'96]

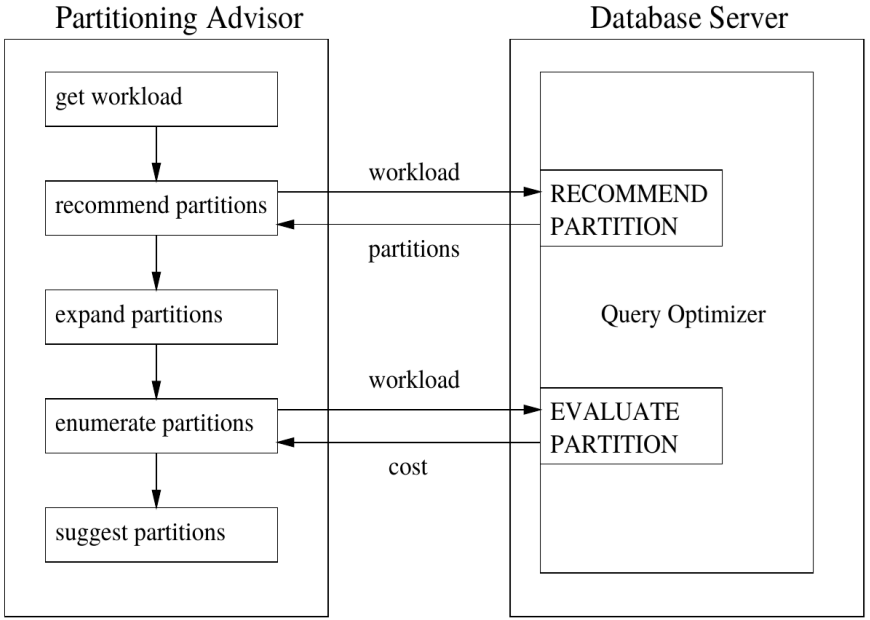
- 避免考虑不必要的列
  - 数据高度不平衡的列、取值范围小的列（如性别）
- 避免考虑非常小的表
  - 小表可以复制到每个节点
- 次优的划分方案：为每张表选择收益最高的分布键
  - 启发式的代价/收益模型：根据每条查询中不同的操作符为其涉及的列加分
    - Join: +1.0, group-by: +0.1
    - 去重: +0.08, 常量选择: -0.05, 变量选择: +0.05
  - 最后在每张表中选取收益最高的列
- 改进：为每张表保留多个收益最高的候选列
  - 穷举搜索所有表的候选列组合
  - 使用查询优化器估计代价和收益

压缩搜索空间

浏览搜索空间

集成查询优化器 [Rao'02]

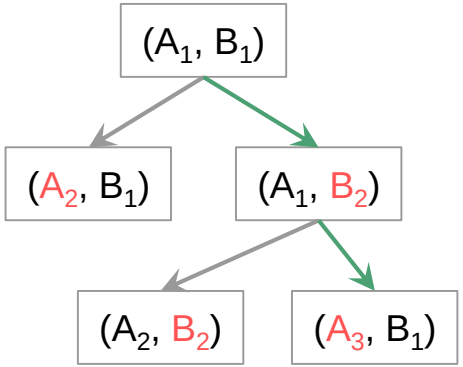
- RECOMMEND 模式
  - 对于每条查询：
    - 找出每张表中有意义的分布键（equi-join，group-by）
    - 评估选择各个分布键的收益
    - 为每张表各推荐一个分布键
- EVALUATE 模式
  - 评估在给定的划分方案下所有查询的执行代价



搜索算法：Rank-based [Rao’02]

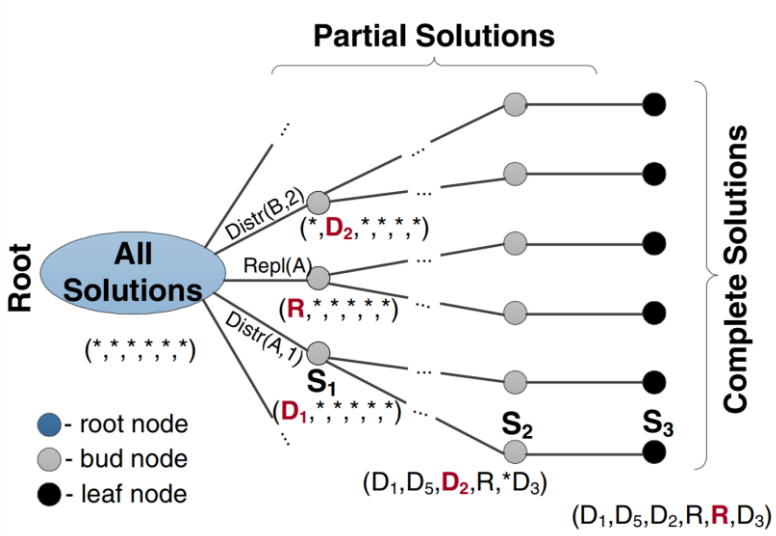
- 搜索树根节点 $\mathbf{P_0}$：为各表选择局部最优的分布键
- 在节点 **P** 处生长：
  - 考虑所有与 **P** 在一张表上相异的子节点
  - 为相异的表选择具有下一个最高收益值的分布键
- 为新节点评分，将其存放于优先队列中
- 选择队列中分值最高的节点作为下一个搜索点

- 分值函数: $rank(P) = -(cost(P.parent) - p_i.benefit * sqrt(Ri.size / max\_size))$



搜索算法：Branch and Bound [Nehme’11]

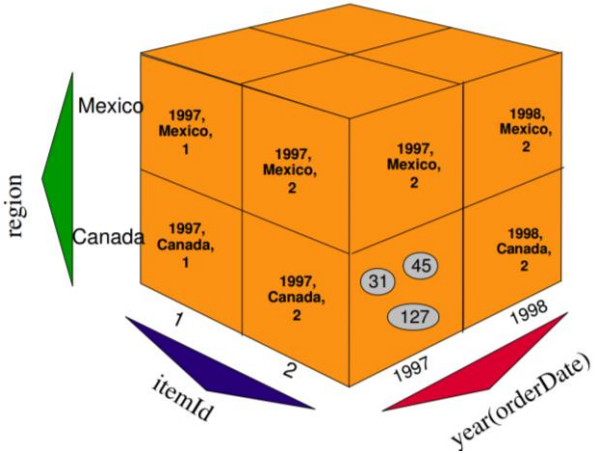
- ***-partitioned table***
  - The optimizer can pick any **concrete** partition scheme for query plans referencing this table
  - Used by the **bounding** function
- Branching policy
  - Node selection: depth-first
    - The first **incumbent** is reached quickly
  - Table/column selection: rank-based
    - Try replication before any partitioning
  - Pruning
    - Storage bound
    - No descendant will be optimal



水平分表——数据聚集

- IBM DB2 MDC（Multi-Dimensional Clustering）

```
CREATE TABLE Sales(
  storeId   int,
  orderDate date,
  region    int,
  itemId    int,
  price     int,
  yearOd    int generated always as year(orderDate)
)
ORGANIZE BY DIMENSIONS (region, yearOd, itemId)
```



- 聚集键自动选择 [Lightstone’04]

总结

总结

- 使用列存储的情况下，垂直分表可作为物化视图选择的子问题

- 水平分表：优先考虑分布键选择
  - 数据聚集、排列顺序？
    - 需考虑查询优化器能力：物理模型特性能否被查询优化器充分利用

- 代价模型
  - 选项 #1：利用查询优化器的代价模型
    - 需要查询优化器提供接口
  - 选项 #2：重新设计代价模型
    - 查询优化能力越强 → 越难建模

参考文献

- Varadarajan R, Bharathan V, Cary A, Dave J, Bodagala S. *DBDesigner: A customizable physical design tool for Vertica Analytic Database.* ICDE 2014
- Jindal A, Palatinus E, Pavlov V, Dittrich J. *A Comparison of Knives for Bread Slicing.* PVLDB 2013
- Hankins RA, Patel JM. *Data Morphing: An Adaptive, Cache-Conscious Storage Technique.* PVLDB 2003
- Papadomanolakis S, Ailamaki A. *AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning.* SSDBM 2004
- Agrawal S, Bruno N, Chaudhuri S, Narasayya VR. *AutoAdmin: Self-Tuning Database Systems Technology.* IEEE Data Eng. Bull. 2006

参考文献

- Zilio DC, Jhingran A, Padmanabhan S. *Partitioning Key Selection for a Shared-Nothing Parallel Database System.* IBM Research Report 1996
- Rao J, Zhang C, Lohman G, Megiddo N. *Automating Physical Database Design in a Parallel Database.* SIGMOD 2002
- Nehme R, Bruno N. *Automated Partitioning Design in Parallel Database Systems.* SIGMOD 2011
- Garcia-Alvarado C, Raghavan V, Narayanan S, Waas FM. *Automatic Data Placement in MPP Databases.* ICDEW 2012
- Lightstone S S, Bhattacharjee B. *Automated design of multidimensional clustering tables for relational databases*. VLDB 2004