2019
05
08-10
北京新云南皇冠假日酒店

数据风云十年变迁

DTCC 第十届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2019

IT168 ChinaUnix ITPUB

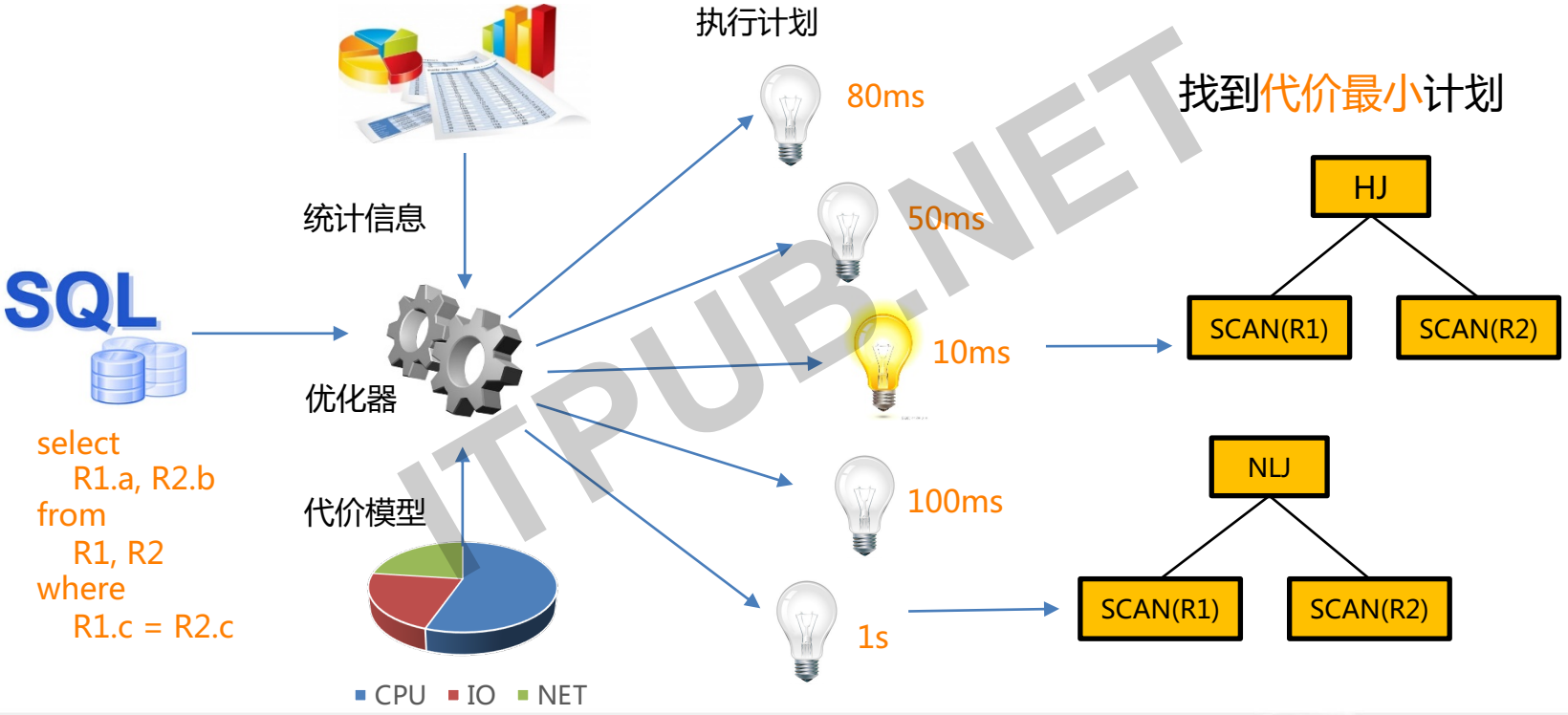
OceanBase查询优化器工程实践

王国平（蚂蚁金服）

议题大纲

- 查询优化器简介
- OceanBase查询优化器工程实践

查询优化器简介



查询优化器面临的挑战

精准的代价模型
和统计信息

海量计划空间

高效的计划管理机制

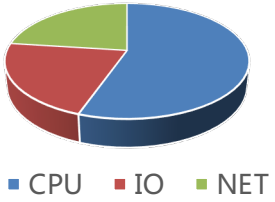
统计信息和代价模型

- 采样误差，滞后

统计信息

选择率计算
&
中间结果估计

- 静态模型: 比如buffer命中率，IO RT等



代价模型

- 单表谓词选择率: 比较准确(动态采样，多列直方图等技术)
- 多表连接谓词选择率: 基于均匀分布假设，错误率很高

查询优化器面临的挑战

精准的代价模型
和统计信息

海量计划空间

高效的计划管理机制

计划空间

表个数	等价逻辑计划个数
5	32
10	2304
15	114688
20	4980736
25	201326592
30	15569256448

星型查询等价逻辑计划个数

星型查询

- 物理算子实现 + 基于代价的改写 + 分布式优化
- 如何高效的枚举执行计划？

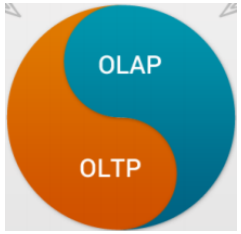
查询优化器面临的挑战

精准的代价模型
和统计信息

海量计划空间

高效的计划管理机制

计划管理



优化器

计划缓存

plan cache

计划演进

plan baseline

select * from R where R.a = 1 and R.b = 1

参数化

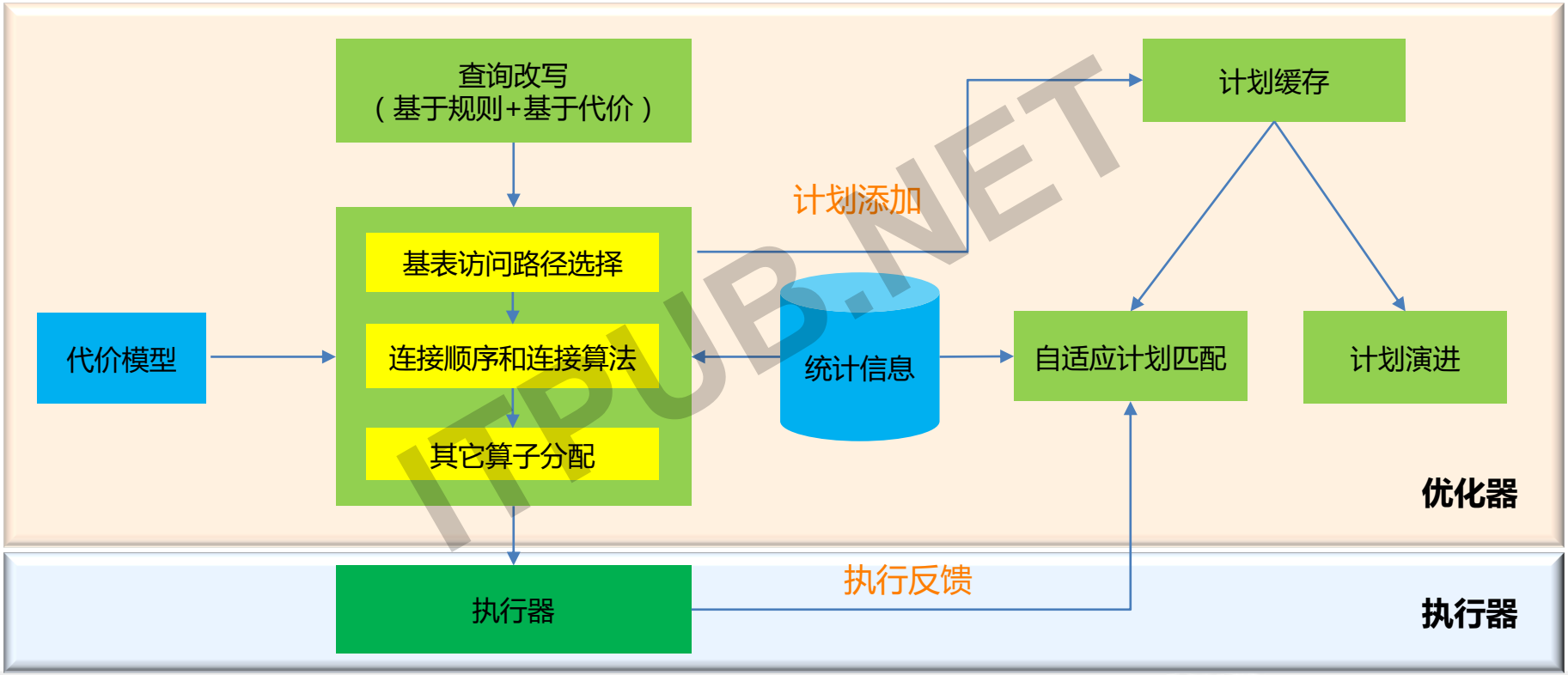
select * from R where R.a = ? and R.b = ?

是否参数化	优化一次	是否缓存	缺点
√	√	√	需要解决不同参数对应不同计划问题
×	√	√	计划过多
×	×	×	优化时间不可忽略

议题大纲

- 查询优化器简介
- OceanBase查询优化器工程实践

OceanBase优化器总体框架



议题大纲

- 查询优化器简介
- OceanBase查询优化器工程实践
  - ✓ 统计信息和代价模型
  - 计划空间
  - 计划管理

基表访问路径选择

- **如何评估一个索引的代价**
  - 扫描索引代价: 扫描行数，大部分场景是顺序扫描
  - 回表代价: 回表行数，随机扫描
- **如何获取行数信息(选择率计算)**
  - 统计信息(直方图)
  - 动态采样

索引代价计算例子

- 查询: select * from t1 where c2 > 200 and c2 < 800 and c3 < 200
- 索引: (c2,c3)
- 表行数: Card(t1) = 10w
- 谓词选择率: Sel(c2 > 200 and c2 < 800) = 0.1, Sel(c3 < 200) = 0.1
- 扫描索引代价: 1w行顺序扫描
- 回表代价: 1k行随机扫描

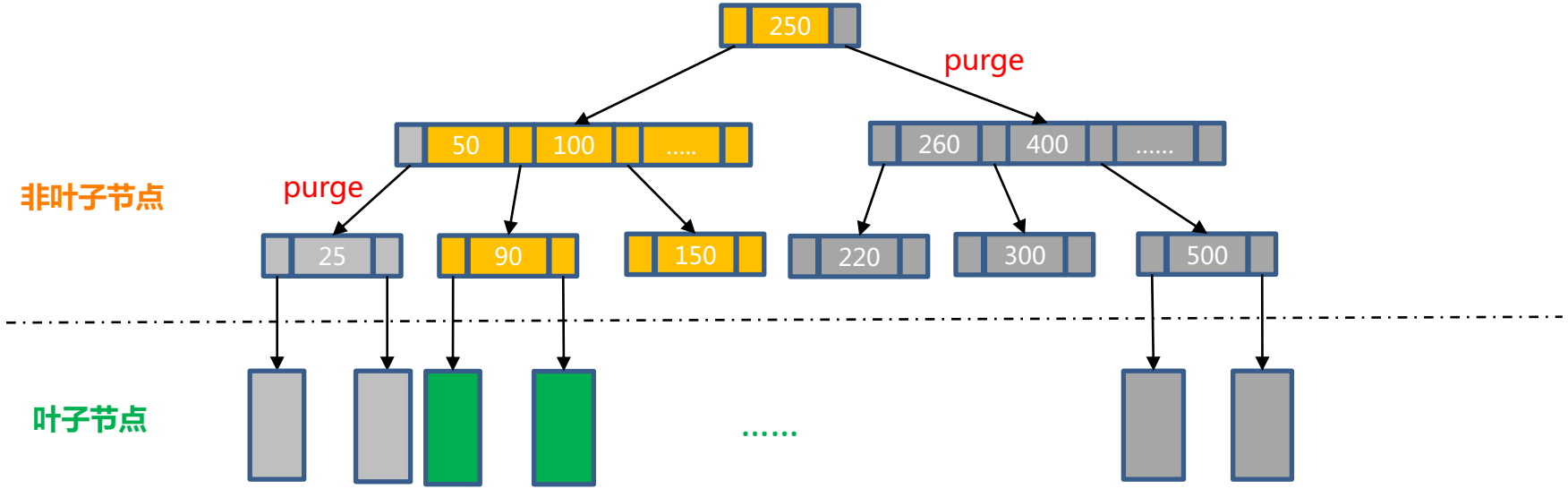
传统计算方式是否适合基于LSM-TREE的存储引擎？

基于LSM-TREE的存储引擎

- **天生适合写事务**
  - 无随机写盘
  - 不需要undo日志
- **对读有一定的性能损耗**
  - 需要合并基线数据和增量数据

OceanBase存储引擎的Purge逻辑

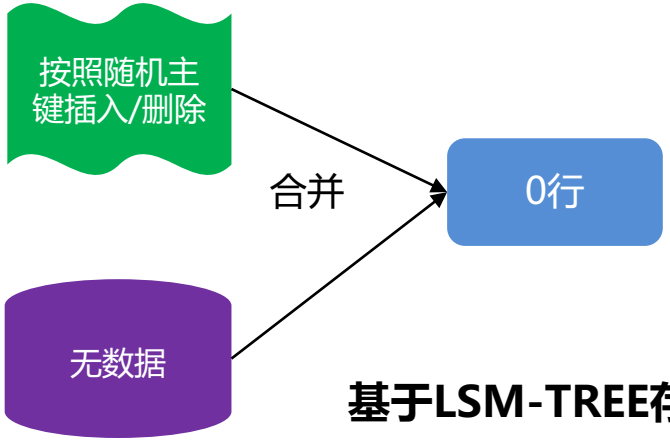
- 在特定场景下，增量数据中连续被删除的区间会被标识出来，后续查询可以跳过对这段区间的访问



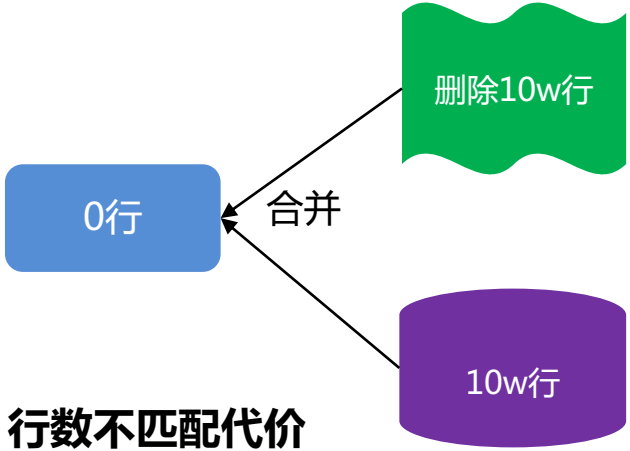
统计信息是否足够用来描述这种复杂的访问模式？

蚂蚁业务Buffer表问题

场景1: 基线无数据，增量数据需要合并



场景2: 基线数据和增量数据需要合并

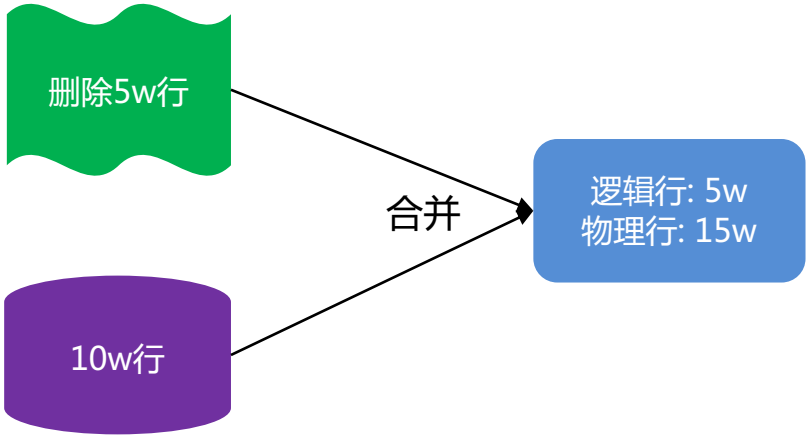


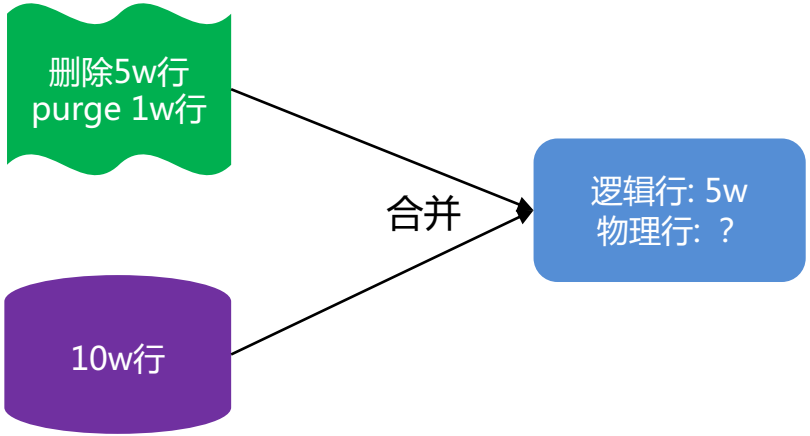
基于LSM-TREE存储引擎: 行数不匹配代价

问题：传统选择率计算出来的行数没办法描述真正计算代价所需要的信息

逻辑行和物理行

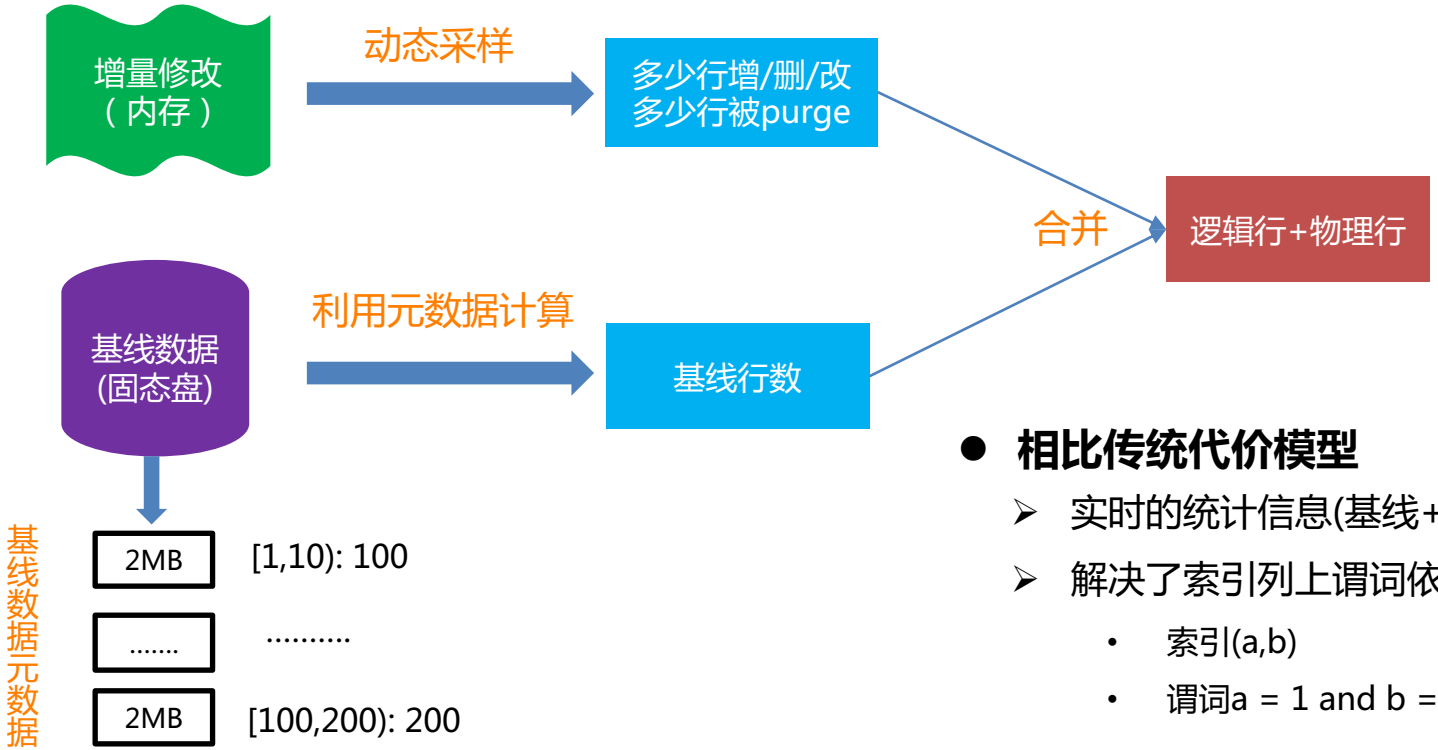
- **物理行: 基线和增量中需要访问的行数**
  - 用来计算索引顺序扫描代价
- **逻辑行: 合并之后的行数**
  - 用来计算回表代价





Purge逻辑让逻辑行和物理行的计算变得复杂

逻辑行和物理行计算



- **相比传统代价模型**
  - 实时的统计信息(基线+增量)
  - 解决了索引列上谓词依赖关系
    - 索引(a,b)
    - 谓词a = 1 and b = 1

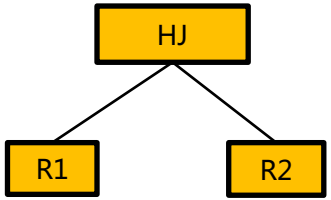
议题大纲

- 查询优化器简介
- OceanBase查询优化器工程实践
  - 统计信息和代价模型
  - ✓ 计划空间
  - 计划管理

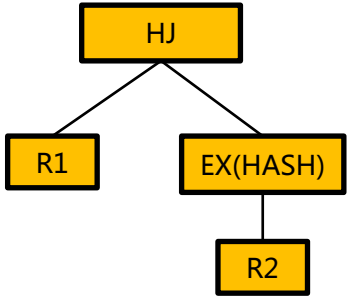
分布式计划优化

- **分布式计划优化会增大计划的搜索空间**
  - 算子分布式算法: 比非分布式算法空间要大
  - 物理属性: 序和分区信息(如何分区以及分区的物理信息)
  - 其它方面: 分区裁剪/并行度优化/分区内(间)并行/……

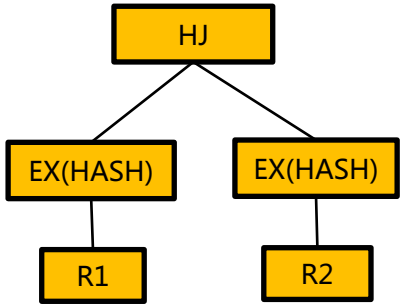
Partition Wise Join



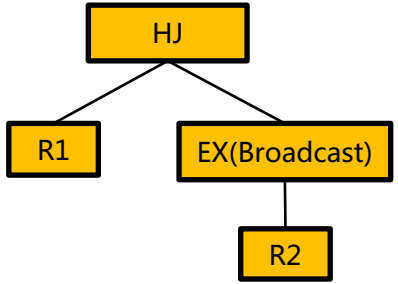
Partial Partition Wise Join



Hash-Hash Distribution Join



Broadcast Distribution Join

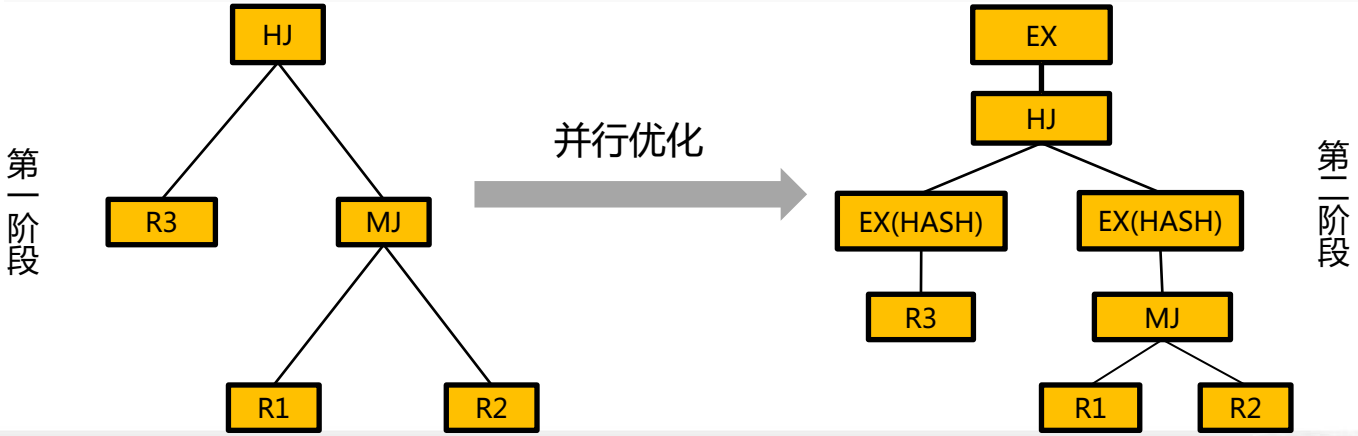


分布式连接算法

二阶段分布式计划优化

- **二阶段分布式计划优化**
  - 第一阶段: 基于所有表都是本地的假设生成一个最优计划
  - 第二阶段: 并行优化, 用启发式规则来选择算子的分布式算法

```
create table R1(a int primary key, b int, c int, d int) partition by hash(a) partitions 4
create table R2(a int primary key, b int, c int, d int) partition by hash(a) partitions 4
create table R3(a int primary key, b int, c int, d int) partition by hash(a) partitions 5
select * from R1, R2, R3 where R1.a = R2.a and R2.b = t3.b;
```

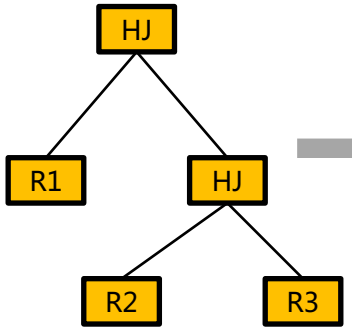


二阶段分布式计划优化

- **二阶段分布式计划优化**
  - 优点: 计划优化复杂性降低
  - 缺点: 搜索空间变小，计划次优

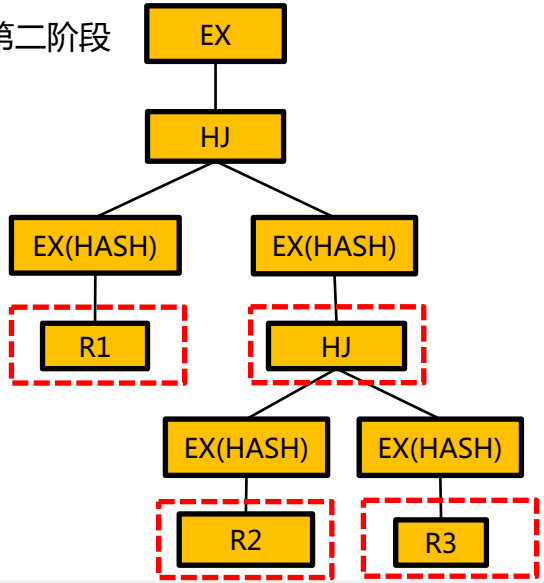
```
create table R1(a int primary key, b int, c int, d int) partition by hash(a) partitions 4
create table R2(a int primary key, b int, c int, d int) partition by hash(a) partitions 4
create table R3(a int primary key, b int, c int, d int) partition by hash(a) partitions 5
select * from R1, R2, R3 where R1.a = R2.a and R2.b = t3.b;
```

第一阶段

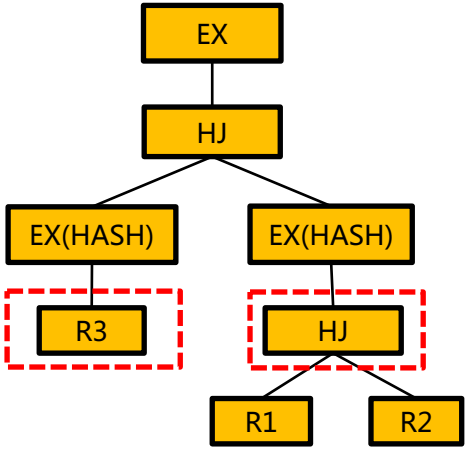


并行优化

第二阶段



最优的计划可能会是这个



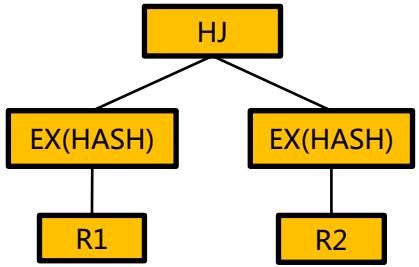
决定连接顺序时没有考虑分区信息

一阶段分布式计划优化(on-going)

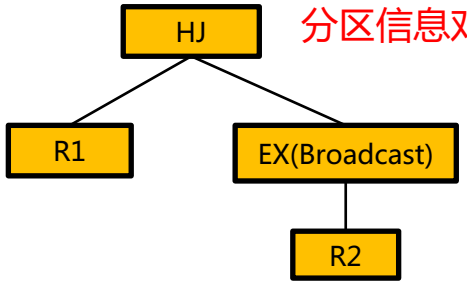
- **一阶段分布式计划优化**
  - 实现方式: 考虑算子的所有分布式实现，维护物理属性
  - 保留如下计划: 代价最小/有Interesting order/有Interesting分区信息

```
create table R1(a int primary key, b int, c int, d int) partition by hash(a) partitions 4
create table R2(a int primary key, b int, c int, d int) partition by hash(a) partitions 5
select R1.a, sum(R2.c) from R1, R2 where t1.b = t2.b group by R1.a
```

P1:100ms



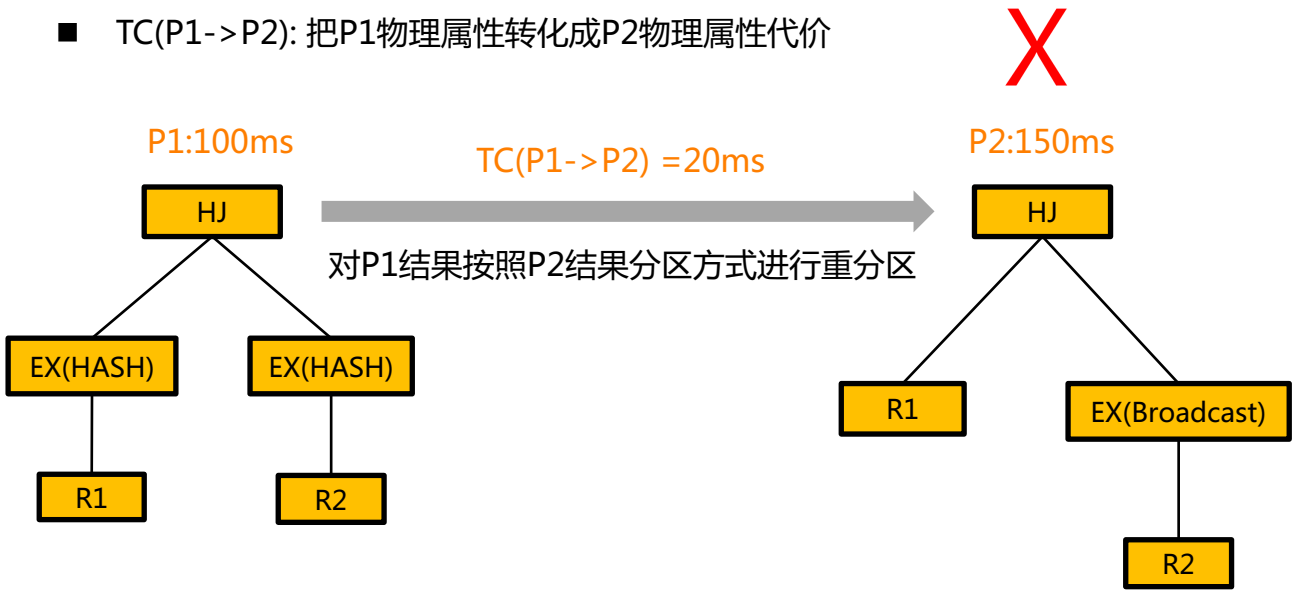
P2:150ms



分区信息对后续Gby算子有用

一阶段分布式计划优化(on-going)

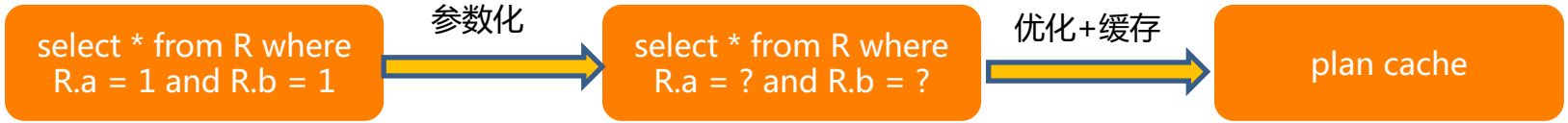
- **一阶段分布式计划优化**
  - 计划空间很大: 需要一些pruning规则来减少计划空间
  - P2可以被裁剪: C(P1) + TC(P1->P2) <= C(P2)
    - TC(P1->P2): 把P1物理属性转化成P2物理属性代价



议题大纲

- 查询优化器简介
- OceanBase查询优化器工程实践
  - 统计信息和代价模型
  - 计划空间
  - ✓ 计划管理

参数化计划缓存



- **为什么参数化？**
  - 为每一个参数缓存一个计划是不切实际的
  - 大部分查询一个计划能满足所有参数需求

- **为什么缓存？**
  - 性能考虑

是否命中计划	性能
✓	百us级别
×	几个ms级别

参数化计划缓存带来的问题

- 数据存在倾斜，不同参数需要缓存不同计划

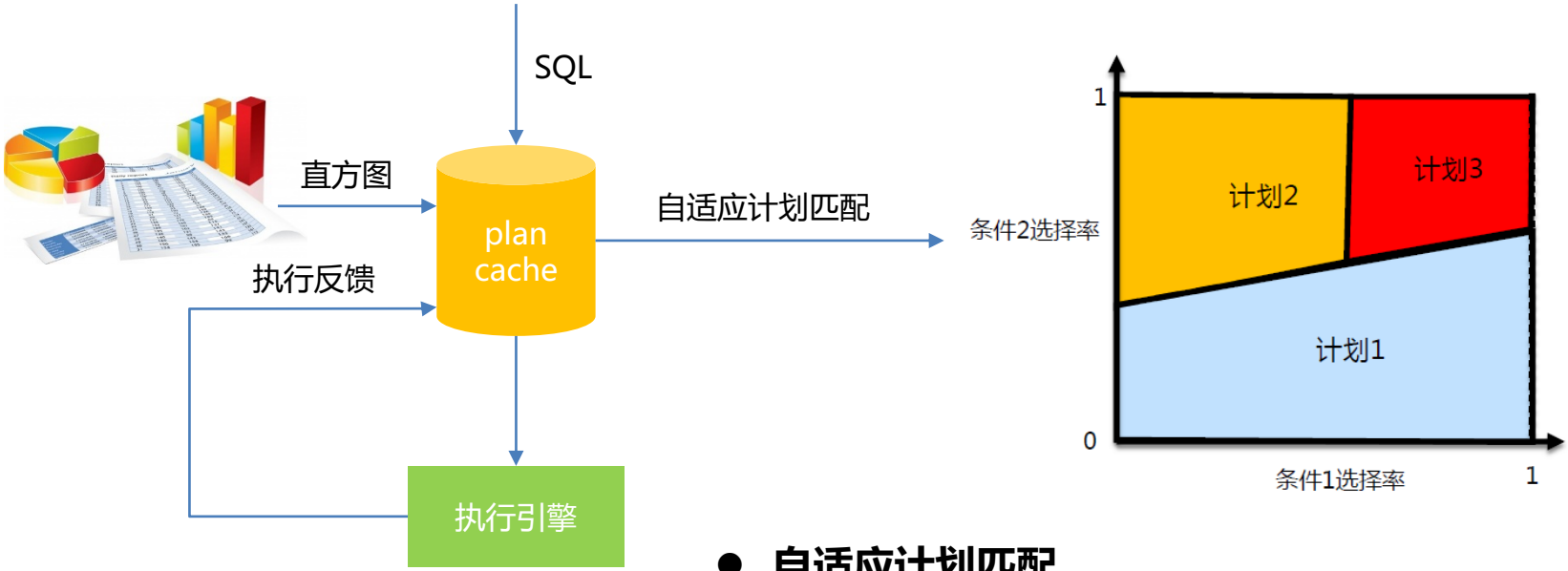
商户	占比
淘宝	50%
美团	10%
LV	0.1
.....	.....

- 查询: 找出特定商户过去一年总销售额

主表扫描 VS 索引扫描

蚂蚁商户域的账单场景

自适应计划匹配

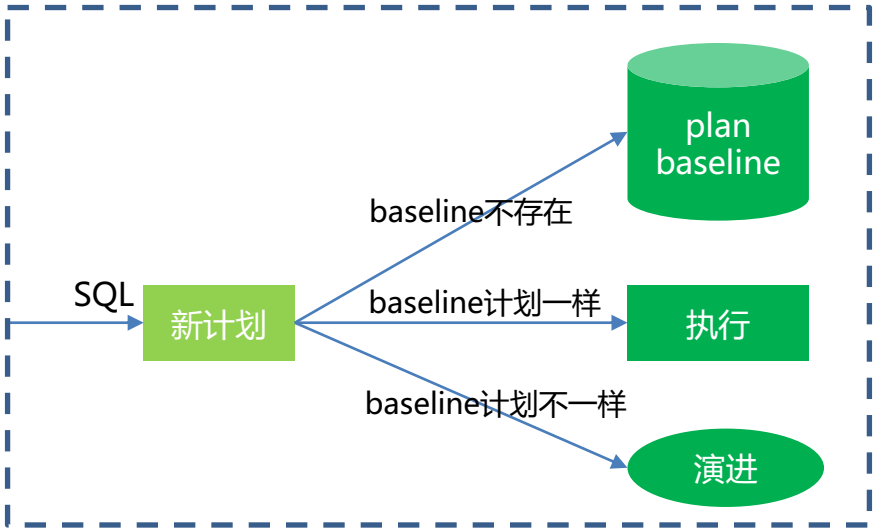


- **自适应计划匹配**
  - 只有必要才会缓存多个计划
  - 通过渐进式的合并选择率空间来达到划分的目的

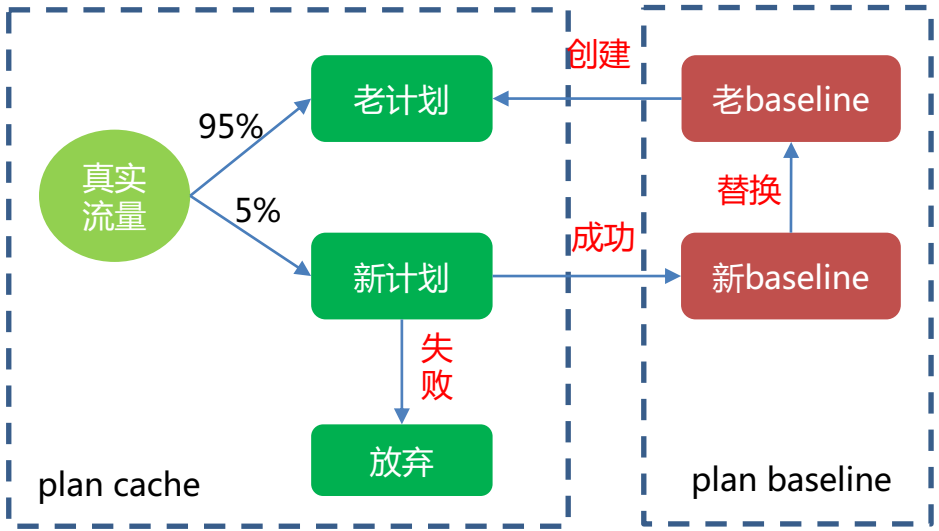
计划演进

- **计划演进: 对新生成计划进行验证，保证新计划不会造成性能回退**
  - 统计信息更新/SCHEMA变更/优化器参数变更/……

计划演进时机



计划演进过程



小结

- **查询优化器立足于自身架构和业务场景特点**
  - 基于LSM-TREE的存储引擎
  - Share-nothing的分布式架构
  - 大规模的运维稳定性
- **OceanBase致力于打造OLTP和OLAP融合的查询优化器**
  - 立足于蚂蚁/阿里真实业务场景，完美承载业务需求
  - 对标商业数据库，进一步打磨HTAP的优化器能力

欢迎关注OceanBase公众号
了解更多OB最佳技术实践内容

扫码加入OceanBase微信群与
蚂蚁金服技术专家一对一沟通

THANKS