MySQL的JOIN

徐轶韬
MySQL Senior Solution Engineer
yitao.xu@oracle.com

ORACLE

MySQL

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

议程

表联接

Nested-Loop 联接算法

Block Nested-Loop 联接算法

执行计划中的联接类型

联接的优化

表联接

- （Inner Join）
  - 仅当两个表都满足ON子句中指定的条件时，从任何一个表返回出现在结果中的行
- Left Join（Outer Join）
  - 返回左表中的所有记录，以及右表中匹配的记录
- Right Join（Outer Join）
  - 返回右表中的所有记录，以及左表中匹配的记录

示例: Inner Join

- SELECT * FROM world_innodb.Country
  where Continent= ‘Europe’ order by Population desc;
- select * from Country;
- select * from Country, City;
- select * from Country, City where Continent="Africa";
- select * from Country, City where
  Country.Code=City.CountryCode;
- select * from Country join City on
  Country.Code=City.CountryCode;
- select * from Country inner join City on
  Country.Code=City.CountryCode;

示例: Outer Join

- select * from Country left join CountryLanguage on CountryLanguage.CountryCode=Country.Code;
- select * from Country left join CountryLanguage on CountryLanguage.CountryCode=Country.Code where CountryLanguage.CountryCode is null;
- select * from Country right join CountryLanguage on CountryLanguage.CountryCode=Country.Code;
- select * from Country right join CountryLanguage on CountryLanguage.CountryCode=Country.Code where CountryLanguage.CountryCode is null;

MySQL的JOIN

- MySQL的联接定义比较宽泛，并不只是两个以上表进行联接，可以将其考虑为每个查询既是一个联接。
- UNION
  - MySQL执行UNION时会将其视作一系列的单表查询，将每个结果放入一个临时表，最后再次读取该临时表。

Nested-Loop 联接算法

- 嵌套循环联接(NLJ)算法一次从循环中的第一个表读取一行，将每一行传递到一个嵌套循环，该循环处理联接中的下一个表。只要还有需要联接的表，这个过程就会重复多次。

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to client
    }
  }
}
```

Block Nested-Loop 联接算法

- 块嵌套循环(BNL)联接算法使用缓冲在外部循环中读取的行，以减少必须读取内部循环中的表的次数。
  - 例如，如果将10行读入一个缓冲区并将该缓冲区传递给下一个内部循环，则可以将内部循环中读取的每一行与缓冲区中的所有10行进行比较。这可以将必须读取内部表的次数减少一个数量级。

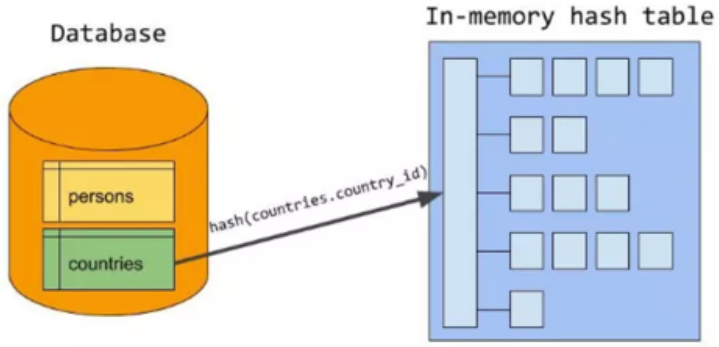
```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions, send to client
        }
      }
      empty join buffer
    }
  }
}

if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions, send to client
    }
  }
}
```

Hash Join 联接算法

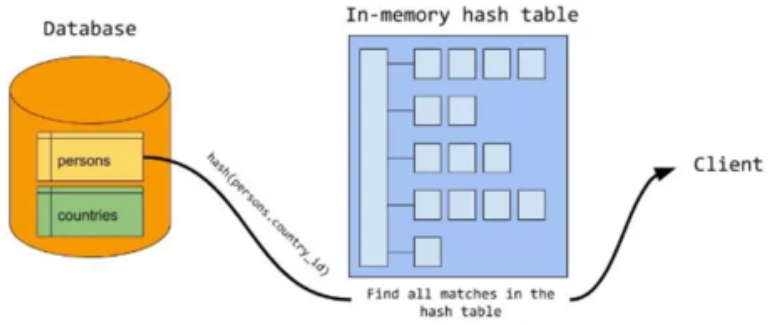
- 将外部表加载到内存生成哈希表，遍历一次内部表即可进行匹配输出结果。
- MySQL采用了经典的In memory哈希算法和 GRACE哈希算法。

In memory哈希算法

- 构建阶段：选择一个表用于联接的字段计算哈希值并将其加载到内存。通常会选择较小的表。



- 探测阶段：另外一个表作每次读取一行，计算其哈希值，并在哈希表中进行匹配，输出结果。



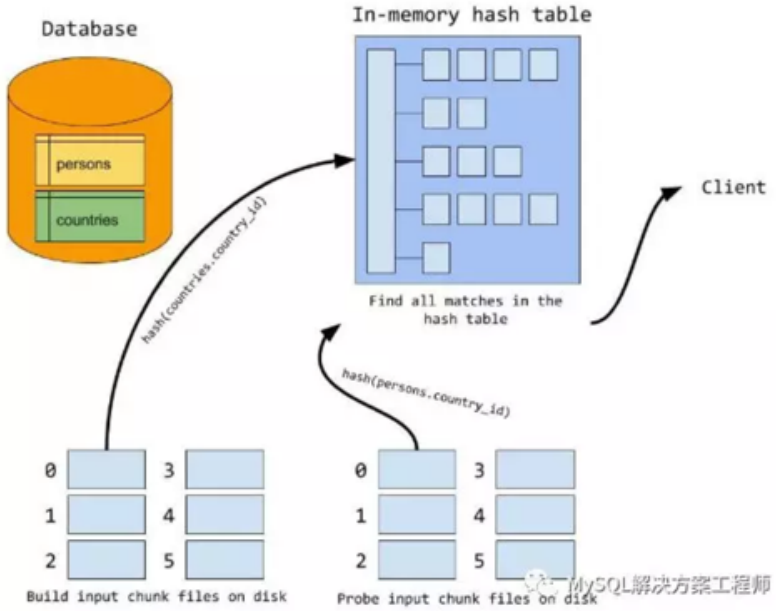
GRACE哈希算法

- 当内存中无法容纳哈希表时，使用磁盘执行。

1.计算构建表和探测表中数据行的哈希值以及使用多个分区文件的哈希值，多个分区文件用于将它们存储在磁盘上。

2.将构建表的第一个分区加载到内存，使用与In memory相同的算法将探测表的第一分区内容与其进行匹配。

3.清空内存，继续处理其它分区。



执行计划中的联接类型

system

const

eq_ref

ref

fulltext

ref_or_null

index_merge

unique_subquery

index_subquery

range

index

ALL

EXPLAIN

理解查询计划

- 使用 EXPLAIN 输出查询计划:

EXPLAIN SELECT * FROM t1 JOIN t2 ON t1.a = t2.a WHERE b > 10 AND c > 10;

id	select_type	table	partitions	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	t1	NULL	range	PRIMARY, idx1	idx1	4	NULL	12	33.33	Using index condition
2	SIMPLE	t2	NULL	ref	idx2	idx2	4	t1.a	1	100.00	NULL

- Explain 正在运行的查询(MySQL 5.7 追加的功能):

  EXPLAIN FOR CONNECTION *connection_id*;

结构化EXPLAIN

- JSON 格式:

```
EXPLAIN FORMAT=JSON SELECT …
```

- 包含更多的信息:
  - Used index parts
  - Pushed index conditions
  - Cost estimates
  - Data estimates

MySQL 5.7
追加

```
EXPLAIN FORMAT=JSON
SELECT * FROM t1 WHERE b > 10 AND c > 10;
EXPLAIN
{
 "query_block": {
  "select_id": 1,
  "cost_info": {
   "query_cost": "17.81"
  },
  "table": {
   "table_name": "t1",
   "access_type": "range",
   "possible_keys": [
    "idx1"
   ],
   "key": "idx1",
   "used_key_parts": [
    "b"
   ],
   "key_length": "4",
   "rows_examined_per_scan": 12,
   "rows_produced_per_join": 3,
   "filtered": "33.33",
   "index_condition": "(`test`.`t1`.`b` > 10)",
   "cost_info": {
    "read_cost": "17.01",
    "eval_cost": "0.80",
    "prefix_cost": "17.81",
    "data_read_per_join": "63"
   },
   ……..
   "attached_condition": "(`test`.`t1`.`c` > 10)"
  }
 }
}
```

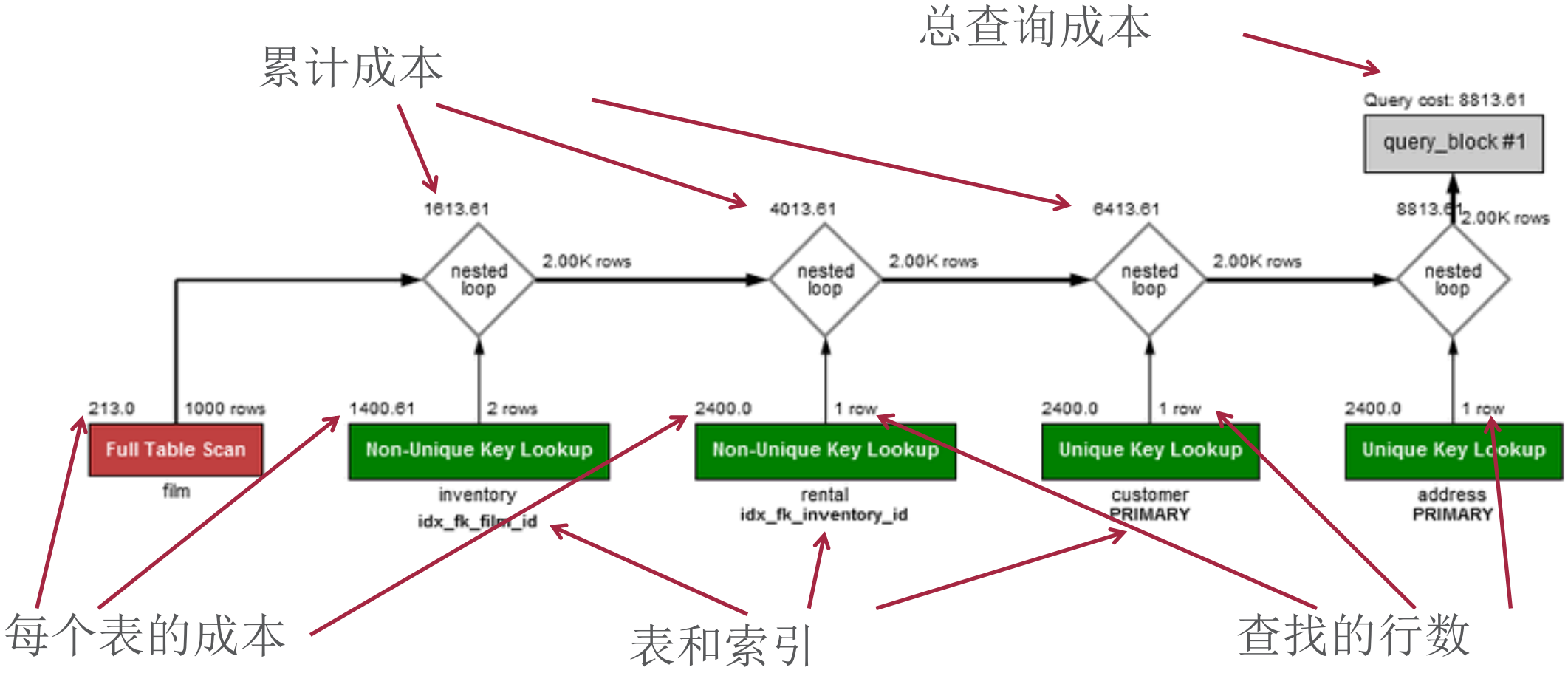
结构化EXPLAIN
为表分配条件

EXPLAIN FORMAT=JSON SELECT * FROM t1, t2
WHERE t1.a=t2.a AND t2.a=9 AND (NOT (t1.a > 10 OR t2.b >3) OR (t1.b=t2.b+7 AND t2.b = 5));

EXPLAIN

```
{
  "query_block": {
    "select_id": 1,
    "nested_loop": [
      {
        "table": {
          "table_name": "t1",
          "access_type": "ALL",
          "rows": 10,
          "filtered": 100,
          "attached_condition": "(t1.a = 9)"
        } /* table */
      },
      {
        "table": {
          "table_name": "t2",
          "access_type": "ALL",
          "rows": 10,
          "filtered": 100,
          "using_join_buffer": "Block Nested Loop",
          "attached_condition": "((t2.a = 9) and ((t2.b <= 3) or ((t2.b = 5) and (t1.b = 12))))"
        } /* table */
      }
    ] /* nested_loop */
  } /* query_block */
}
```

可视化EXPLAIN (MySQL Workbench)



const

- 表最多有一个匹配行，在查询开始时读取。因为只有一行，所以优化器的其余部分可以将这一行中的列的值视为常量。
- 将主键或惟一索引的所有部分与常量值进行比较时，将使用const。

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
  WHERE primary_key_part1=1 AND primary_key_part2=2;
```

system

- 表只有一行(=系统表)。这是const联接类型的一种特殊情况。

eq_ref

- 当联接使用索引的所有部分，并且索引是主键或唯一的非空索引时，将使用它。
- eq_ref可用于使用“=”进行比较的索引列。比较值可以是一个常量或一个表达式。在下面的例子中，MySQL可以使用eq_ref联接来处理ref_table:

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

ref

- 如果联接仅使用键最左边的前缀，或者键不是主键或唯一索引(联接不能根据键值选择单行)，则使用ref。
- ref可用于使用=或<=>操作符进行比较的索引列。在下面的例子中，MySQL可以使用ref来处理ref_table:

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

ref_or_null

- 这个联接类型类似于ref，但是MySQL会对包含空值的行进行额外的搜索。这种联接类型优化最常用于解决子查询。在下面的例子中，MySQL可以使用ref_or_null联接来处理ref_table:

```
SELECT * FROM ref_table
  WHERE key_column=expr OR key_column IS NULL;
```

index_merge

- 该联接类型表示使用索引合并优化。

unique_subquery

- 此类型替换了以下形式的子查询中的eq_ref:

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

- unique_subquery只是一个索引查找功能，它完全替换子查询，以提高效率。

index_subquery

- 此联接类型类似于unique_subquery。它代替了子查询，但它对以下形式的子查询中的非唯一索引有效:

```
value IN (SELECT primary_key FROM single_table WHERE some_expr)
```

range

- 使用索引检索给定范围内的行。
- 当键与常量比较时，可以使用=、<>、>、>=、<、<=、is NULL、<=>、BETWEEN、LIKE或IN()操作符:

```
SELECT * FROM tbl_name
  WHERE key_column = 10;

SELECT * FROM tbl_name
  WHERE key_column BETWEEN 10 and 20;

SELECT * FROM tbl_name
  WHERE key_column IN (10,20,30);

SELECT * FROM tbl_name
  WHERE key_part1 = 10 AND key_part2 IN (10,20,30);
```

ALL

- 对前一个表中的每个行组合进行全表扫描。

index

- 除了要扫描索引树之外，index联接类型与ALL类型相同。包括两种方式:
  - 如果索引是查询的覆盖索引，并且可以用于满足表中所需的所有数据，则只扫描索引树。索引扫描通常比全表扫描更快，因为索引的大小通常小于表数据。
  - 使用索引读取数据来执行全表扫描，按照索引顺序查找数据行。
- 当查询只使用属于单个索引的列时，MySQL可以使用此联接类型。

hash join

- 8.0.20之后， 如果联接不使用索引 ， 可以使用哈希联接。
- 通过优化器提示BNL和NO_BNL进行控制

```
SELECT *
    FROM t1
    JOIN t2
        ON t1.c1=t2.c1;
```

```
           id: 1
  select_type: SIMPLE
        table: t2
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using where; Using join buffer (hash join)
```

semijoin

- 半联接是一种转换，它支持多种执行策略，如表拉出、删除重复、首次匹配、松散扫描和物化。优化器使用半联接策略来改进子查询的执行。

```
SELECT class.class_num, class.class_name
    FROM class
    INNER JOIN roster
    WHERE class.class_num = roster.class_num;
```

```
SELECT class_num, class_name
    FROM class
    WHERE class_num IN
        (SELECT class_num FROM roster);
```

- 使用EXISTS和IN的效果一样（8.0.16之后）

antijoin

- 如果查询语句没有使用IN (SELECT ... FROM ...) 或 EXISTS (SELECT ... FROM ...)会将其转换为反联接。
- 反联接是一种只返回没有匹配的行的操作。

```
SELECT class_num, class_name
    FROM class
    WHERE class_num NOT IN
        (SELECT class_num FROM roster);
```

- 查询会被重写为SELECT class_num, class_name FROM class ANTIJOIN roster ON class_num;
- 这意味着，对于class中的每一行，只要在roster中找到匹配，就可以丢弃class中的行。

semijoin / antijoin的策略

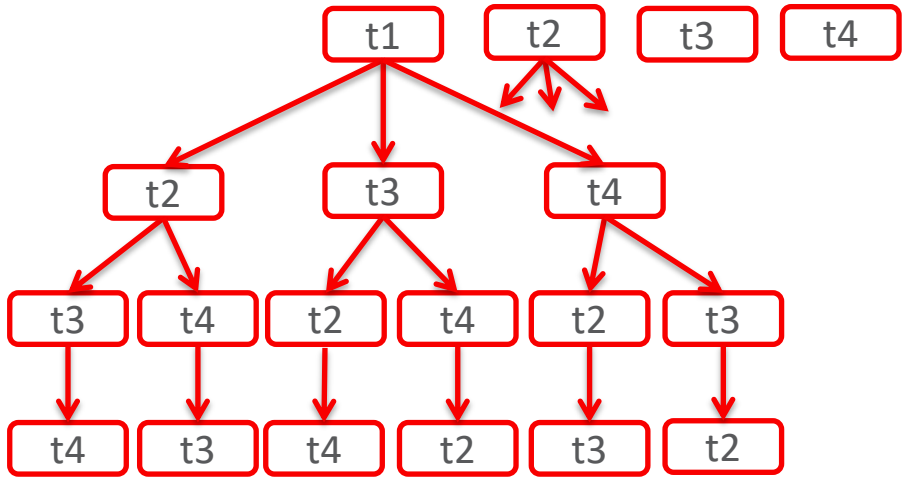
- 将子查询转换为联接，子查询表和外部表进行内联接。表拉出将表从子查询中取出到外部查询。
- 删除重复:与使用联接一样，使用临时表删除重复记录。
- 首次匹配:当扫描内部表的行组合时，如果多个给定值，选择一个而不是全部返回。
- 松散扫描:使用索引扫描子查询表，该索引允许从每个子查询的值组中选择单个值。
- 将子查询物化为用于执行联接的临时表，该临时表具有索引，用于删除重复项。当将临时表与外部表联接时，索引也可用于查找。

联接的优化

”贪婪的搜索策略”

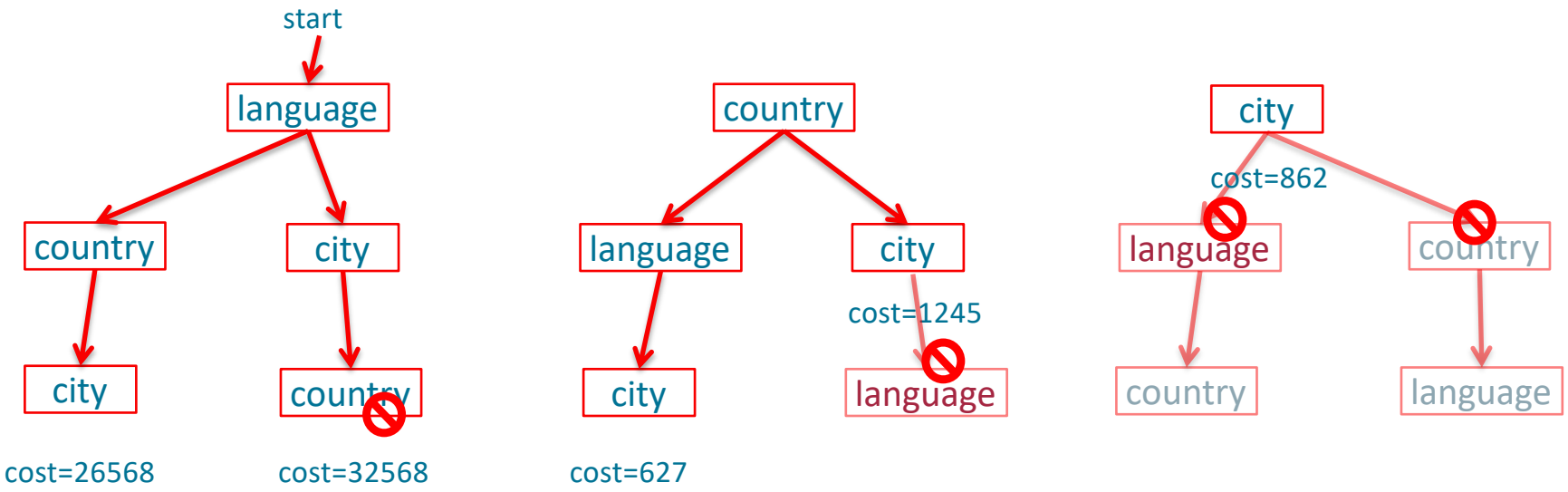
- 目标：给定N个表的联接，找到最佳联接排序：
  - 从所有单表计划开始（根据大小和键依赖性排序）
  - 用剩余的表扩展每个计划
    - 深度优先
  - 如果 “部分计划成本” > “最佳计划成本”：
    - “删去” 计划
  - 探索式删去：
    - 删去不太有希望的部分计划
    - 可能会在罕见的情况下错过最佳计划
      （关闭set optimizer_prune_level = 0）

N! 可行计划



联接优化图解

```
SELECT city.name AS capital, language.name
FROM city
  JOIN country ON city.country_id = country.country_id
  JOIN language ON country.country_id = language.country_id
WHERE city.city_id = country.capital
```



联接的优化

例

EXPLAIN SELECT *
FROM customers JOIN orders ON c_custkey = o_custkey
WHERE c_acctbal < -1000 AND o_orderdate < '1993-01-01';

id	select type	table	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	orders	ALL	i_o_orderdate, i_o_custkey	NULL	NULL	NULL	15000000	31.19	Using where
1	SIMPLE	customer	eq_ref	PRIMARY	PRIMARY	4	dbt3.orders.o_custkey	1	33.33	Using where

联接的优化

使用STRAIGHT_JOIN改变联接顺序

EXPLAIN SELECT STRAIGHT_JOIN *
FROM customer JOIN orders ON c_custkey = o_custkey
WHERE c_acctbal < -1000 AND o_orderdate < '1993-01-01';

id	select type	table	type	possible keys	key	key len	ref	rows	filtered	Extra
1	SIMPLE	customer	ALL	PRIMARY	NULL	NULL	NULL	1500000	33.33	Using where
1	SIMPLE	orders	ref	i_o_orderdate, i_o_custkey	i_o_custkey	5	dbt3. customer. c_custkey	15	31.19	Using where

MySQL 8.0:

SELECT /*+ JOIN_ORDER(customer, orders) */ ...

联接顺序

Performance

MySQL Database Service on Oracle Cloud

100% 由MySQL团队开发、管理并提供支持

容易

- 完整管理的数据库服务
- 即时部署
- 最新功能

安全

- 数据保护
- 高级别安全性
- 最新安全性升级

企业就绪

- MySQL Enterprise Edition
- 兼容本地部署
- 基于Gen 2 Cloud Infrastructure

MySQL Database Service vs. RDS

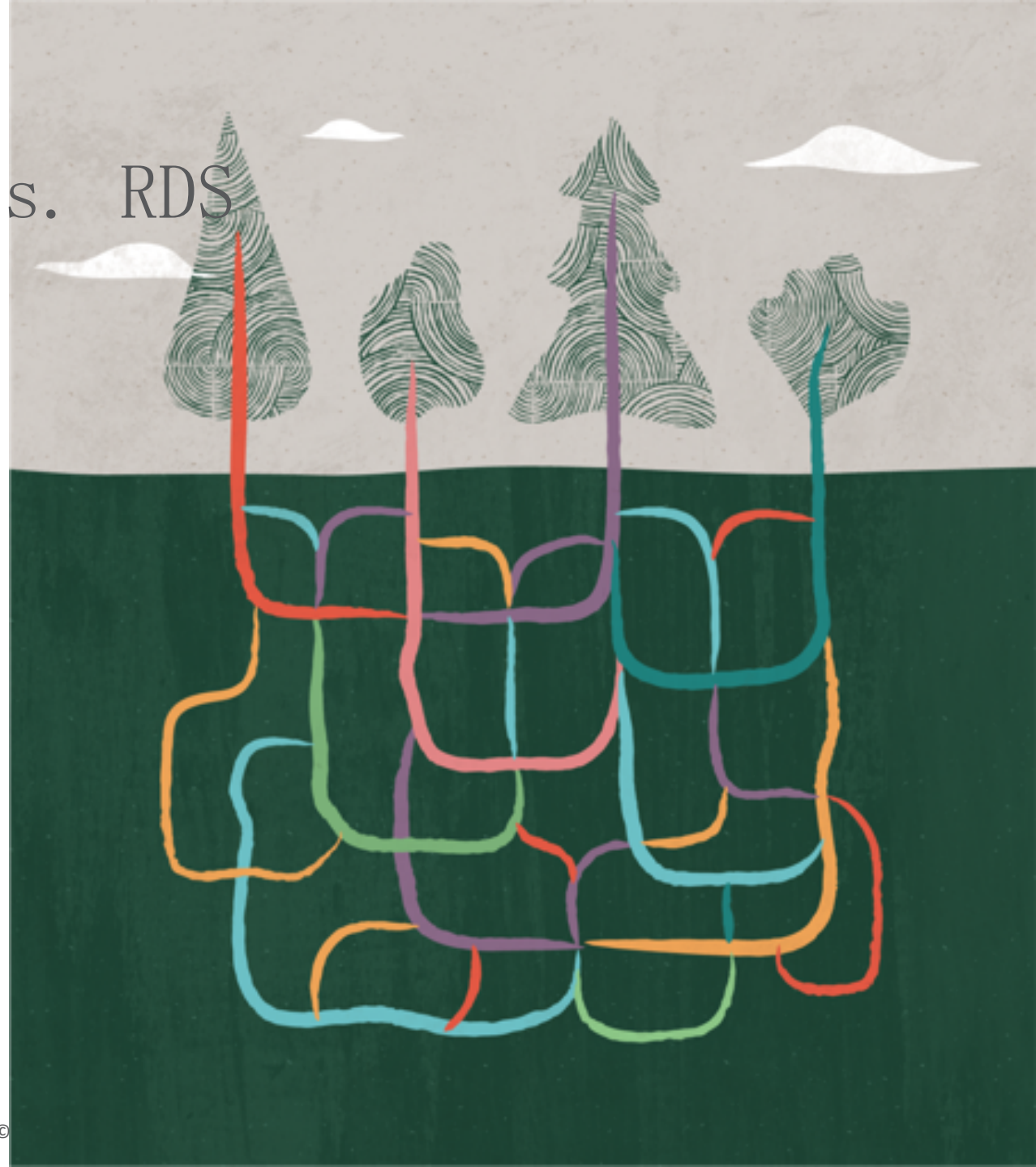
MySQL Enterprise Edition
- 24x7 MySQL 团队支持

100% 兼容本地部署MySQL
- 混合云部署
- 不会云锁定

最新的安全性更新
- 最新的MySQL安全性修复

最新功能
- X Dev API, MySQL Shell, Document Store

基于Gen 2 Cloud Infrastructure
- 针对企业工作负载的安全性



Q&A