通过物化视图进行查询优化

汇报人：朱道冰

DSC
分布式存储与计算实验室

2023.05.06

目录

01. 物化视图概述

02. 视图设计-识别候选子查询

03. 视图设计-选择子查询建立物化视图

04. 视图设计-淘汰物化视图

05. 视图维护-更新物化视图里面的数据

06. 视图利用-基于物化视图进行查询重写

07. 总结

01 物化视图概述

01 物化视图概述

分布式存储与计算实验室

1.1 背景

- 随着数据规模的日益增大，数据库查询面临着批量SQL查询效率低下的问题。
- 实验表明批量SQL查询中存在重复的子查询。
- 选择子查询建立物化视图并复用是降低批量查询中重复计算的重要手段。

user

Id	Name	Gender
1	Mike	M
2	Marry	F

item

Id	Name	Catalog
1	Cookie	c1
2	Cake	c2

ui

User_Id	Item_id
1	1
1	2
2	1
2	2

Create Materialized view:

```
Create Materialized view mv as
select gender, catalog, count(1) as c
from user, item, ui
where user.id = ui.user_id
    and item.id = ui.item_id
group by gender, catalog
```

Gender	Catalog	Count
M	c1	1
F	c1	1
M	c2	1
F	c2	1

Query:

```
select gender, count(1) as c
from user, item, ui
where user.id = ui.user_id
    and item.id = ui.item_id
group by gender
```

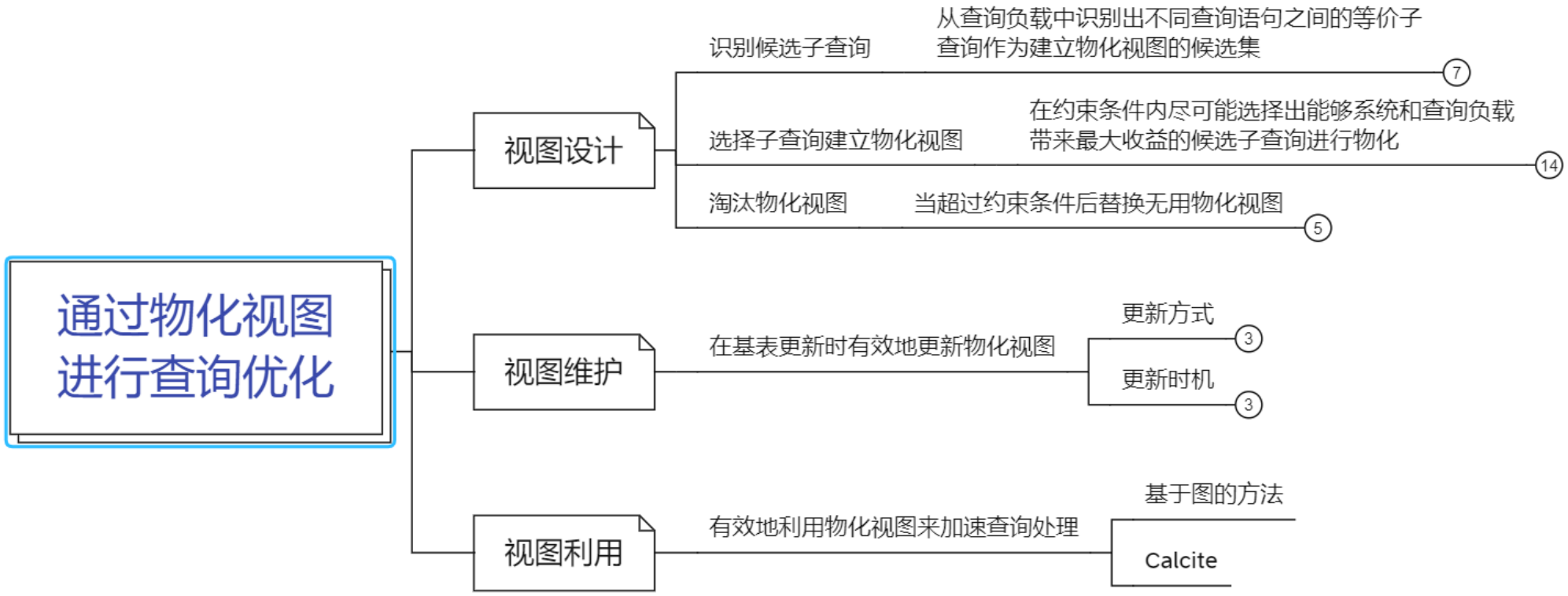
Plan Rewrite

New query:

```
select gender, count(1) as c
from mv
group by gender
```

01 物化视图概述

1.2 生命周期



02 / 视图设计 - 识别候选子查询

02 视图设计 - 识别候选子查询

2.1 背景与分类

识别候选子查询：从查询负载中识别出不同查询语句之间的等价子查询作为建立物化视图的候选集。

方法：

- **基于图的识别方法：**将查询负载转换为执行计划，不同执行计划构成一个图，用图的方法判断等价性。
- **识别等价子查询：**判断 SQL 语句的符号是否满足等价性。

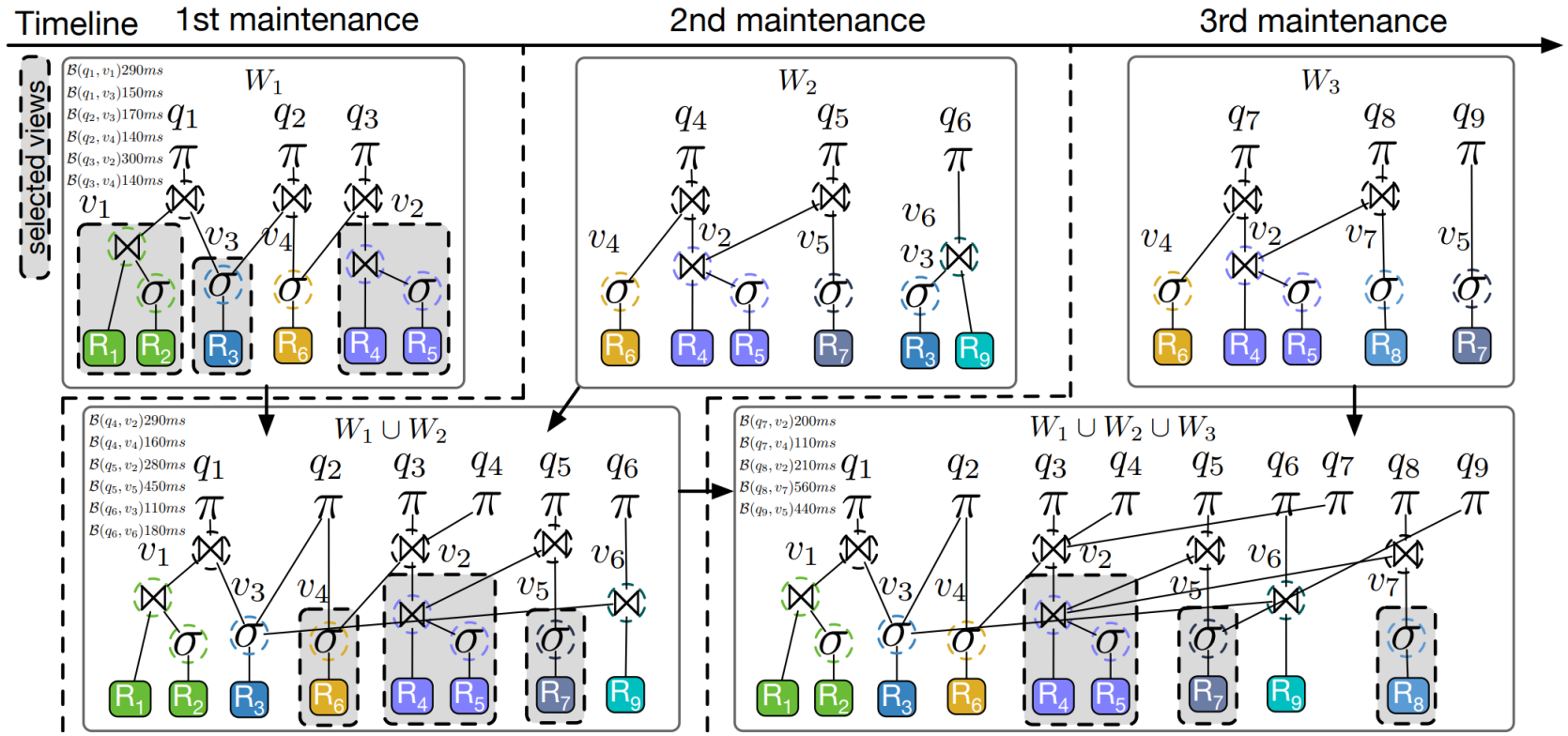
02 视图设计 - 识别候选子查询

分布式存储与计算实验室

2.2 基于图的识别方法

思想：

1．将查询负载中的查询语句解析成逻辑执行计划或者物理执行计划。
2．将上述执行计划聚合在一个图上，合并相同的节点，并记录频次。
3．根据频次等启发式规则选择候选子查询。



02 视图设计 - 识别候选子查询

2.3 识别等价子查询

两个子查询被称为等价子查询：当且仅当他们对应的**关系代数表达式**是等价的，这表明两个等价子查询可以互相利用彼此的查询结果。

方法：

- **基于符号表示的等价判断：**判断 SQL 语句的符号是否满足等价性。

一种粗糙的办法：

1. 将 alias 还原为 relation
2. 排序，如对不同谓词，不同relation等
3. 采用hash等方式进行等价性判断

- **基于逻辑语义的等价判断：**将查询解析成符号关系上的约束，从而去验证等价性。

比如：where a !=0 和 where a>0 or a<0 是等价的。

02 视图设计 - 识别候选子查询

2.4 基于逻辑语义的等价判断 Cosette

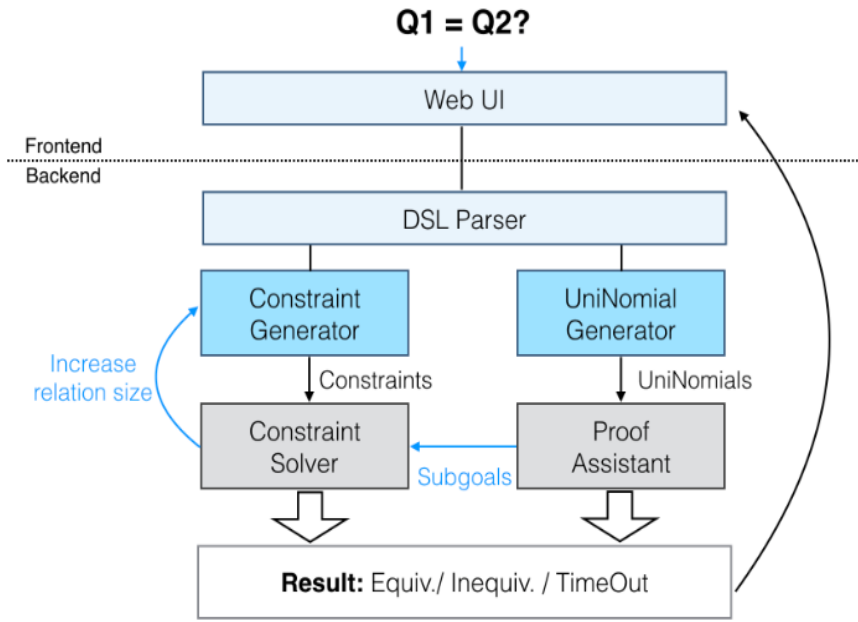
Cosette 是一个自动化证明SQL查询等价性的工具，它可以接受两个SQL查询作为输入，然后返回它们等价的反例或者形式化证明。、

组件：

- **反例：**一种找满足/反例的工具Rosette(SMT solver，可满足理论)
- **形式化证明：**一种形式化证明工具Coq(定理证明系统)

步骤：

1. 通过Rosette找反例，返回不等价(Inequiv)；没找到在算法允许的开销限制内increate relation size重新找。
2. 拆分为多个UniNomial表达式(subgoal)，使用Coq进行形式化证明，如果有不能证明的，则调用Rosette对subgoal找返例，最终根据所有subgoal的反例和形式化证明情况返回等价(Equiv)，不等价(Inequiv)和未知(TimeOut)。



Cosette System Architecture

02 视图设计 - 识别候选子查询

2.4 基于逻辑语义的等价判断 Cosette

Rosette 利用SMT solver将复杂的公式转化为布尔表达式，求解出满足条件的解或者证明不存在解。

针对：v1 && (v2 || v3) == (v1 && v2) || (v1 && v3)

满足的解：{v1:True, v2:True, v3:False} 前者True，后者True

针对：v1 && (v2 && v3) == (v1 && v2) || (v1 && v3)

不满足的解：{v1:True, v2:True, v3:False} 前者False，后者True

02 视图设计 - 识别候选子查询

2.4 基于逻辑语义的等价判断 Cosette

Cosette 能找到“嵌套优化中”的 **Count bug** 问题的反例。

PARTS

PNUM	QOH
3	6
10	1
8	0

SUPPLY

PNUM	QUAN	SHIPDATE
3	4	7-3-79
3	2	10-1-78
10	1	6-8-78
10	2	8-10-81
8	5	5-7-83

[KIE 84 2]

Kiessling defines Query Q2 as follows

Query Q2:

Find the part numbers of those parts whose quantities on hand equal the number of shipments of those parts before 1-1-80

找出那些库存量等于在1980年1月1日之前**购买次数**“COUNT(SHIPDATE)”的零件的零件编号。

```
SELECT   PNUM
FROM     PARTS
WHERE    QOH = (SELECT   COUNT(SHIPDATE)
                FROM     SUPPLY
                WHERE    SUPPLY.PNUM = PARTS.PNUM AND
                         SHIPDATE < 1-1-80)
```

[KIE 84 4]

Given the example tables PARTS and SUPPLY defined above, query Q2 will give the following result when evaluated using nested iteration

Result

PARTS.PNUM
10
8

[KIE 84 4]

PNUM为10的QOH是1，SUPPLY中购买日期早于1-1-80的购买次数有1次，所以符合要求。同理可以找出，PNUM为8的QOH是0，SUPPLY购买日期早于1-1-80的有0次，所以符合条件。

Application of Kim's algorithm NEST-JA to Query Q2 results in the following transformation

```
TEMP′ (SUPPNUM,CT) =
        (SELECT   PNUM, COUNT(SHIPDATE)
         FROM     SUPPLY
         WHERE    SHIPDATE < 1-1-80
         GROUP BY PNUM)

SELECT   PNUM
FROM     PARTS, TEMP′
WHERE    PARTS.QOH = TEMP′.CT AND
         PARTS.PNUM = TEMP′.SUPPNUM
```

[KIE 84 4]

TEMP′ evaluates to

TEMP′.SUPPNUM	CT
3	2
10	1

少了一个80

and the final result is

PARTS.PNUM
10

[KIE 84 5]

该语句属于NEST-JA优化，新建了一个GROUP BY的TEMP临时表，但是GROUP BY语句中不会出现COUNT(SHIPDATE)=0的row，导致最终的结果中少了 PNUM为8的情况。

02 视图设计 - 识别候选子查询

2.4 基于逻辑语义的等价判断 Cosette

Coq 是一种交互式的定理证明器，它可以用来构造数学对象和证明它们的性质。

Cosette中Coq步骤：

1. 将两个查询语句转换为 UniNomials关系代数。
2. 尝试多种 Coq script 证明他们的等价性。

Coq script是指关系代数的等价转换规则的组会，如右侧例子中的：

1. **加法交换律** commutativity of +
2. **乘法分配律** the distributivity of ×

```
-- Q1
SELECT * FROM (R UNION ALL S)
WHERE b
-- Q2
(SELECT * FROM S WHERE b)
UNION ALL
(SELECT * FROM R WHERE b)
```

$$Q_1(t) : b(t) \times (R(t) + S(t))$$
$$Q_2(t) : b(t) \times S(t) + b(t) \times R(t)$$

```
Lemma 1:
∀ t, Q1(t) = Q2(t).
Proof:
apply comm_plus.
apply dist_prod.
reflexivity.
Qed.
```

Figure 4: Sample queries, their compilation to UniNomials, and Coq script for their equivalence proof.

03 视图设计-选择子查询建立物化视图

03 视图设计-选择子查询建立物化视图

3.1 背景与分类

选择子查询建立物化视图：在约束条件内尽可能选择出能够系统和查询负载带来最大收益的候选子查询进行物化。

挑战：

- 在有限的存储空间下选择最优的视图进行物化

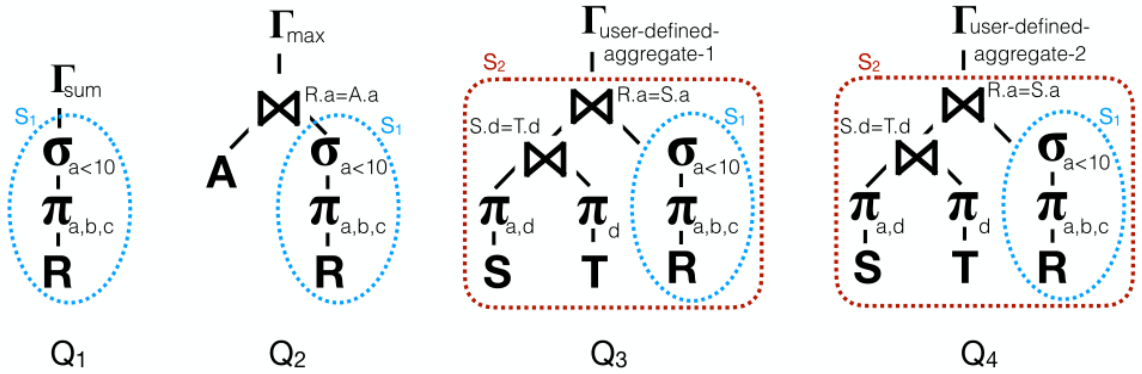


Figure 3: Illustrating the subexpression selection problem.

方法：

- 基于DAG图的启发式方法
- 基于ILP的方法
- 基于策略的动态选择方法

- S1比S2出现在更多的查询中，但它捕获的计算量比S2小。
- S2可能比S1大得多，因此会消耗更多的存储预算。
- 如果S2已经是物化的，那么S1对于评估Q3和Q4来说是多余的，因为它是S2的子树。

视图设计-选择子查询建立物化视图-DAG图

分布式存储与计算实验室

3.2 基于DAG图的启发式方法

DAG (Directed Acyclic Graph)

1. 基于AND/OR图的贪心算法
2. 基于MVPP (Multi-View Processing Plan) 图的遗传算法
3. 基于数据立方体格图的算法

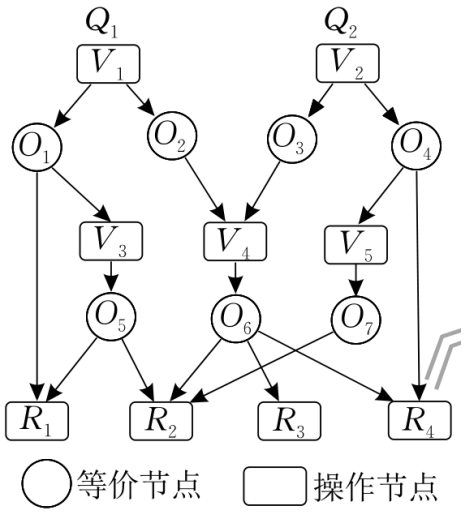


图 2 AND/OR 图示例

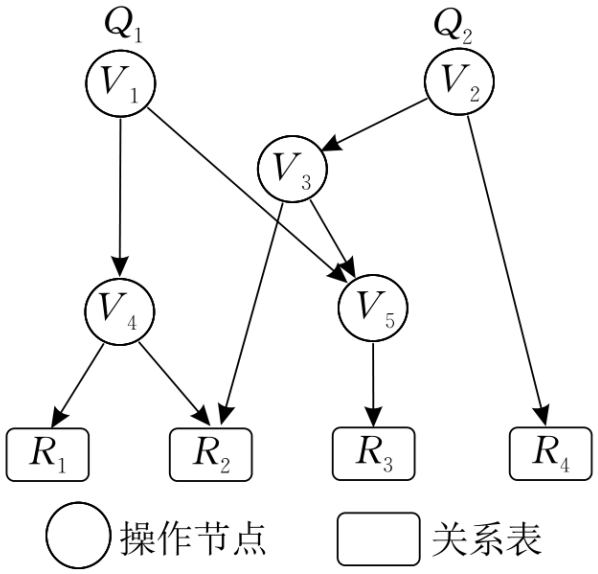


图 3 MVPP 图示例

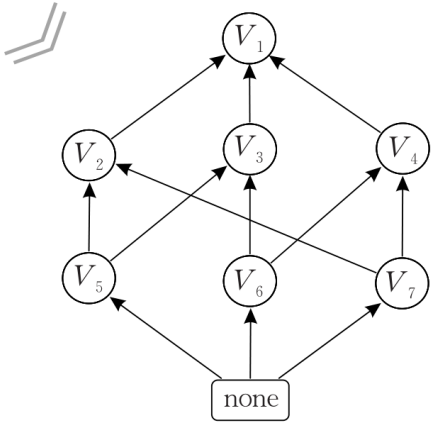


图 4 Data Cube Lattice 图示例

视图设计-选择子查询建立物化视图-ILP

分布式存储与计算实验室

3.3 ILP背景知识

- **最优化问题：**选择一组参数（变量），在满足一系列有关的限制条件（约束）下，使设计指标（目标）达到最优值。
- **线性规划问题（LP问题）：**指目标函数和约束条件皆为线性的最优化问题。
- **整数线性规划问题（Integer Linear Programming）：**上面最优解可能是分数或小数，但是对于某些具体问题要求解答是整数。我们称这样的线性规划问题为整数线性规划问题。
- **0/1 ILP：**所有的整数变量只能是0或者1

0/1整数线性规划问题（01ILP）

视图设计-选择子查询建立物化视图-ILP

分布式存储与计算实验室

3.3 0/1 ILP 举例

$$\begin{cases} \max z = \sum_{j=1}^{n} c_j x_j \\ s.t. \quad 0 < \sum_{j=1}^{n} b_j x_j \le B \\ x_j = 0 \text{或} 1 \end{cases}$$

$$\text{maximize} \sum_{i=1}^{n} \sum_{s_j \in R_i^{max}} u_{ij} \cdot z_j$$

$$\text{s.t.} \sum_{j=1}^{m} b_j \cdot z_j \le B_{max}$$

c_j ： 是利润

b_j ： 是投资金额

X_j ： 是是否投资

U_{ij} :收益

Z_j ： 是否选择

B_j ： 物化视图开销

03 视图设计-选择子查询建立物化视图-ILP

3.3 基于ILP的问题表述

子表达的效用：

$$u_{ij} = C_{\mathcal{D}}(s_j) - C_{acc}(s_j)$$

$C_D(s_j)$表示使用候选子表达 s_j 的效用，即： **“使用物化视图-不使用物化视图”**

$C_{acc}(s_j)$表示使用物化后的候选子表达 s_j 后的效用

子表达集的效用：

$$U_S(q_i) = \sum_{s_j \in R_i^{max}} u_{ij}$$

$s_j \in R_i^{max}$ 为使 q_i 的评估成本降低最多的子查询

视图设计-选择子查询建立物化视图-ILP

分布式存储与计算实验室

3.3 基于ILP的问题表述

潜在约束：

相互作用的子表达：

对于查询 q 的两个候选子表达s1、s2，如果其中一个的逻辑计划所对应的树是另一个的子树，那么这两个子表达式就是相互影响的。

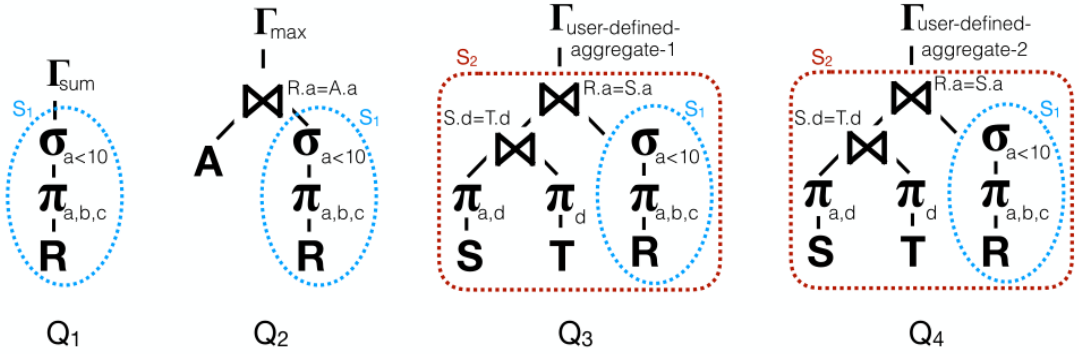


Figure 3: Illustrating the subexpression selection problem.

$$\text{maximize} \sum_{i=1}^{n} \sum_{s_j \in R_i^{max}} u_{ij} \cdot z_j$$

$$\text{s.t.} \sum_{j=1}^{m} b_j \cdot z_j \leq B_{max}$$

U_{ij} :收益

Z_j ： 是否选择

B_j ： 物化视图开销

视图设计-选择子查询建立物化视图-ILP

分布式存储与计算实验室

3.3 基于ILP的问题表述

$$\text{maximize} \sum_{i=1}^{n} \sum_{s_j \in R_i^{max}} u_{ij} \cdot z_j$$

$$\text{s.t.} \sum_{j=1}^{m} b_j \cdot z_j \leq B_{max}$$

U_{ij} :收益

Z_j ： 是否选择

B_j ： 物化视图开销

$$\text{maximize} \sum_{i=1}^{n} \sum_{j=1}^{m} u_{ij} \cdot y_{ij}$$

$$\text{s.t.} \sum_{j=1}^{m} b_j \cdot z_j \leq B_{max}$$

$$y_{ik} + \frac{1}{m} \sum_{\substack{j=1 \\ j \neq k}}^{m} y_{ij} \cdot x_{jk} \leq 1 \quad \forall i \in [1,n], k \in [1,m]$$

$$y_{ij} \leq z_j \quad \forall i \in [1,n], j \in [1,m]$$

y_{ij} :是否选择j这个物化视图 用于qi的重写

X_{jk} :jk两个物化视图是否有包含关系

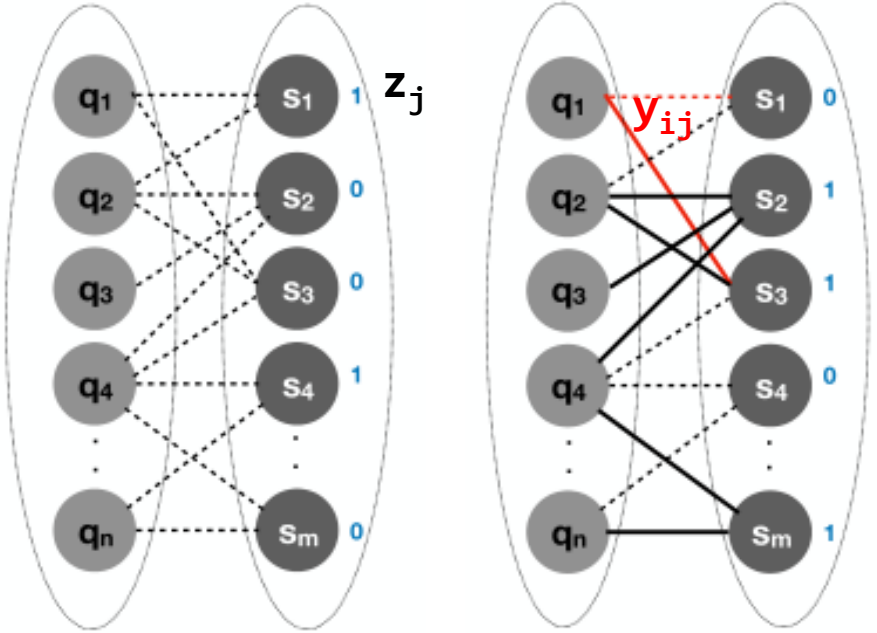
视图设计-选择子查询建立物化视图-ILP

分布式存储与计算实验室

3.3 二分图标记问题 BIGSUBS

1．建图

左边顶点为查询（query）q_i，右边顶点为子表达（subexpression）s_j，左右两边的顶点相连表示 s_j 是查询 q_i 的一个子表达。



2．问题转换

- 给右边的顶点都分配一个0-1标签 z_j，同时遵守预算成本的约束条件
- 给每条边都分配一个0-1标签 y_{ij}，同时遵守剩下的约束条件

3．期望得到的结果

被选择进行物化的子表达的标签为1，被选为进行查询重写的子表达与查询之间用实线相连

视图设计-选择子查询建立物化视图-ILP

分布式存储与计算实验室

3.3 子表达选择算法BIGSUBS

BIGSUBS算法采用了一种**迭代方法**。每次迭代包括两个步骤。

(i) 为子表达式顶点分配标签（翻转）。 <- 基于启发式策略的选择物化视图 z_j

(ii) 给出第一步确定的子表达顶点，为边分配标签。 <- 更小的ILP问题 y_{ij}

这个两步过程不断重复，直到顶点和边的标签没有变化，或者直到达到预定的迭代次数。

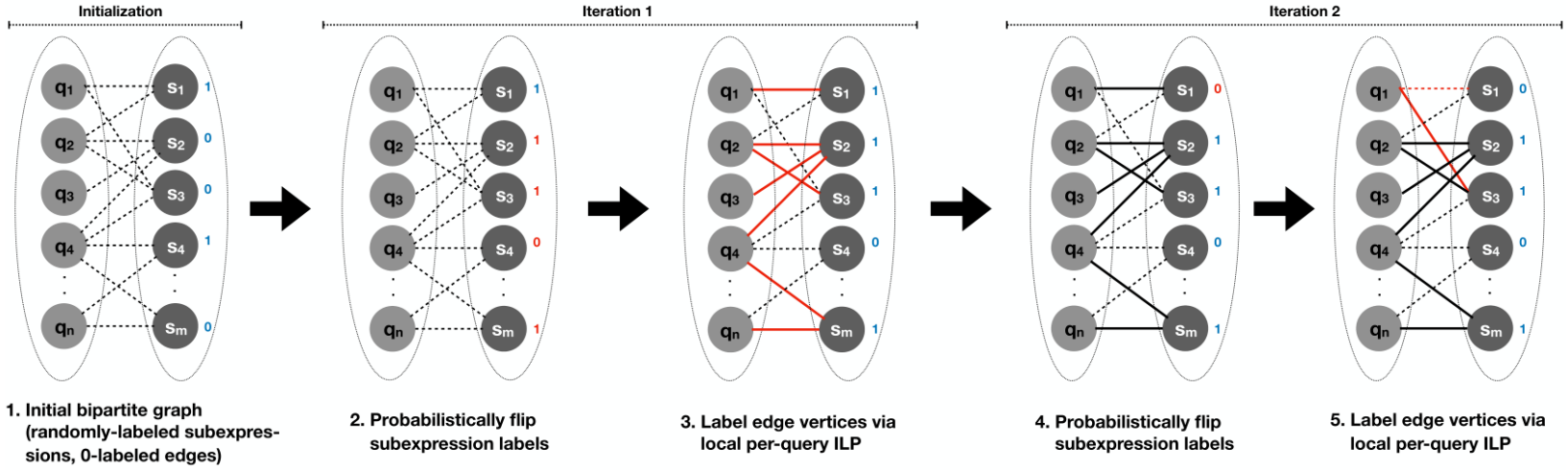


Figure 4: Illustrating first two iterations of subexpression selection via bipartite graph labeling. We assume each subexpression has storage footprint b_j=1 and the total budget is B_{max}=3. Subexpression labels are shown next to the vertices. For the edges, we use solid lines when label is 1 and dashed ones otherwise. At each iteration, we mark with red the labels whose value changed.

视图设计-选择子查询建立物化视图-ILP

3.3 子表达选择算法BIGSUBS

启发式规则标记子表达顶点：

- 离存储预算越远，翻转标签的概率就越高；
- 一个1标签的子表达的当前效用越高，它被取消选择的可能性就越低，而标记为0的子表达的潜在收益越高，它被选中的可能性就越高。

$$p^j_{capacity} = \begin{cases} 1 - B_{cur}/B_{max} & \text{if } B_{cur} < B_{max} \\ 1 - B_{max}/B_{cur} & \text{otherwise} \end{cases}$$

$$p^j_{utility} = \begin{cases} 1 - U^j_{cur}/U_{cur} & \text{if } z_j = 1 \\ \frac{U^j_{max}/b_j}{U_{max}/B_{max}} & \text{if } iter \leq p \text{ or } B_{cur} \leq B_{max} - b_j \\ 0 & \text{otherwise} \end{cases}$$

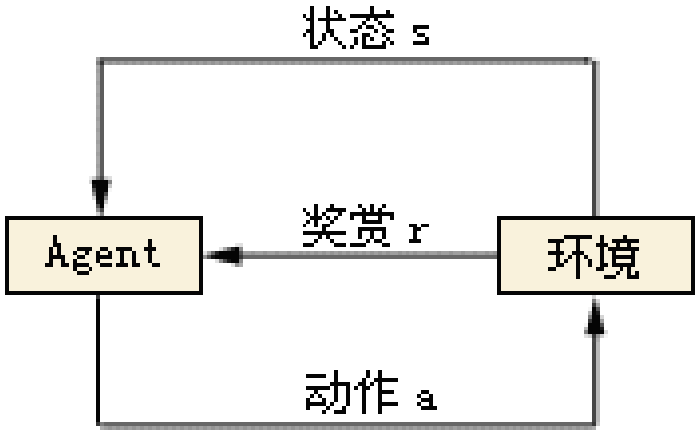
$$p^j_{flip} = p^j_{capacity} \cdot p^j_{utility}$$

视图设计-选择子查询建立物化视图-强化学习

3.4 背景知识

马尔科夫决策过程（MDP）：下一个状态的产生只和当前的状态有关

- 状态空间S（所有可能状态的集合）
- 行动空间A（所有可能的决定的集合）
- 初始状态p_0（系统如何开始）
- 状态转换概率P（在一个决定下，状态如何变化）
- 奖励函数R
- 折扣因子γ（对未来回报进行累积的权重）



视图设计-选择子查询建立物化视图-强化学习

3.4 背景知识

动作价值函数（state-action value function），也叫 **Q 函数**，即在某一状态 s 采取某一动作 a，假设一直使用同一个策略 π，得到的累计激励的期望值。

Q函数的增量更新方式：

真实　　估计

$$newQ_{S,A} = Q_{S,A} + \alpha\left(R_{S,A} + \gamma * \max Q'(s', a') - Q_{S,A}\right)$$

- $newQ_{S,A}$：基于状态和行动的新Q值
- $Q_{S,A}$：c当前Q值
- α：学习效率
- $R_{S,A}$：基于状态和行动的奖励
- γ：折扣因子
- $Q'(s', a')$：在给定新的状态和行动下未来最大的奖励

关键：$R_{S,A}$　创建物化视图的奖励

03 视图设计-选择子查询建立物化视图-强化学习

3.4 奖励函数分类

$R_{S,A}$ 创建物化视图的奖励

- 空闲时间异步收集奖励数据用于训练
- 训练学习型代价估计模型计算奖励
  - 2020 RLView Wide-Deep Model
  - 2021 AutoView Encoder-Reducer Model
  - 2022 GnnMV Gnn-Based Model

03 视图设计-选择子查询建立物化视图-强化学习

3.4 奖励函数-异步收集 DQM

State:	$M = \{0, 1\}$ view status, Q workload until q
Action:	$\{\emptyset, +\}$ create the view or do nothing
Reward:	$R(q, v)$ improvement minus amortized creation
Policy:	$\pi(Q, M) \mapsto \{\emptyset, +\}$ decision to create view

$$R(q, v) = \mathsf{Improvement}(q, v) - \mathsf{Cost}(v)$$

物化视图带来的提升

物化视图的开销

视图设计-选择子查询建立物化视图-强化学习

3.4 奖励函数-异步收集 DQM

评估Improvement()函数需要运行一个系统通常不会运行的查询，即不使用该视图的**反事实查询**。

$$\text{Improvement(q,v)} = \text{Query}(q, v) - \text{Query}(q_i, \emptyset).$$

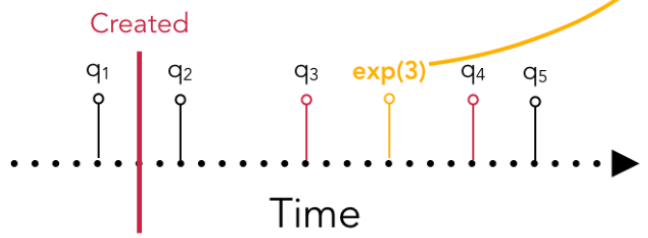
系统维护一个运行中的成对实验的缓冲区。在空闲时间，它执行两个查询（有视图和无视图）实验并存储每个查询的边际效益。

(A) Experience Buffer

State	View Used	Action	New State	Reward
No, 1	N/A	-	No, 2	0
Yes, 2	No	+	Yes, 3	0
Yes, 3	Yes	+	Yes, 4	36
Yes, 4	Yes	-	Yes, 5	?

(B) Experiment Buffer

State	Improvement
3	36
4	?



视图设计-选择子查询建立物化视图-强化学习

分布式存储与计算实验室

3.4 奖励函数-学习型代价估计模型

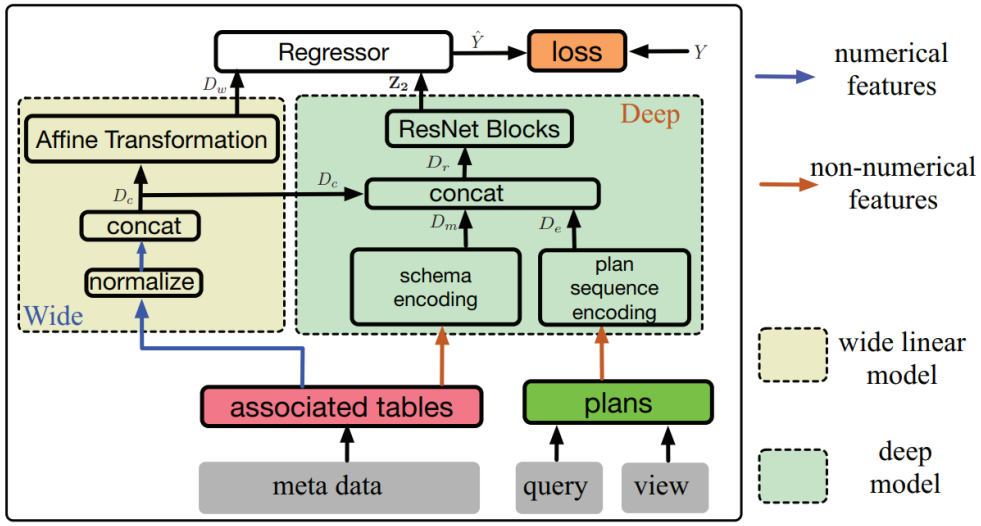


Fig. 5. Wide-Deep Model

2020年 RLView Wide-Deep Model

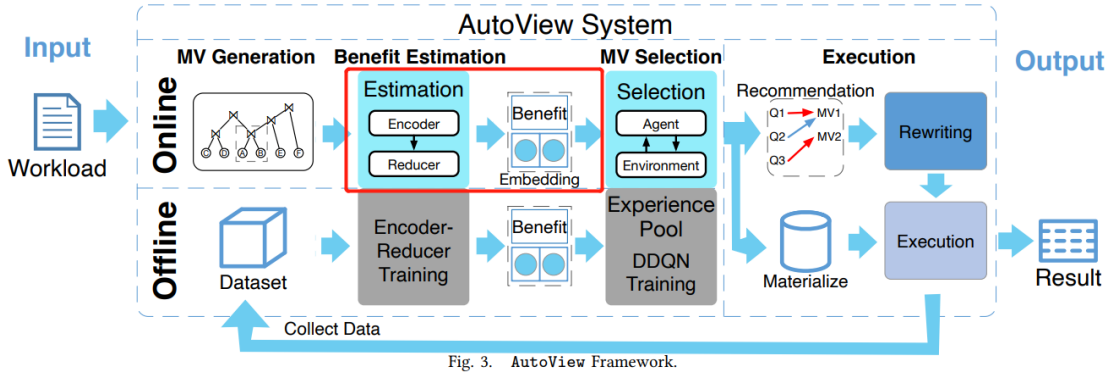
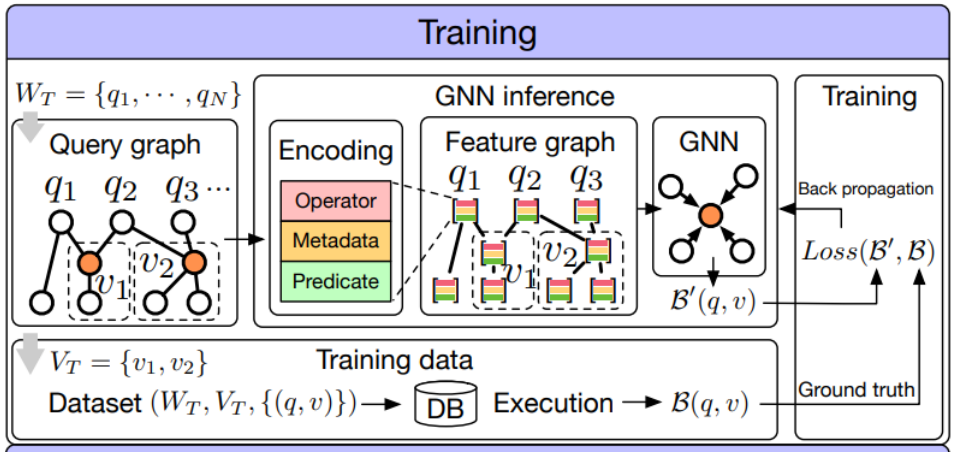


Fig. 3. AutoView Framework.

2021年 AutoView Encoder-Reducer RNN Model



2022年 GnnMV Gnn-Based Model

04 视图设计-淘汰物化视图

04 视图设计-淘汰物化视图

分布式存储与计算实验室

4.1 视图淘汰背景

视图淘汰：当超过约束条件后替换无用物化视图

物化视图淘汰的原因：

- 查询负载的变化
- 物化视图存储占用空间有限

淘汰策略分类：

- Static： 根据一个workload生成物化视图后保持不变
- DynaMat：依据频率、执行时间、大小等对物化视图进行打分
- DQM：异步真实运行 + 指标打分
- HAWC：Cost Model + ILP(执行代价最小)
- GnnMV：Learned Cost Model + ILP(收益最大)

Methods	Start	1st maintenance	2nd maintenance	3rd maintenance	# of MV usages	Save time (ms)
Static	$V_0^*(S)=\emptyset$	$V_1^*(S)=\{v_1,\underline{v_2},\underline{v_3}\}$	$V_2^*(S)=\{v_1,\underline{v_2},v_3\}$	$V_3^*(S)=\{v_1,v_2,v_3\}$	5 times: $\{(q_4,v_2),(q_5,v_2),(q_6,v_3),(q_7,v_2),(q_8,v_2)\}$	1090
Dynamic	$V_0^*(D)=\emptyset$	$V_1^*(D)=\{v_1,\underline{v_2},\underline{v_3}\}$	$V_2^*(D)=\{\underline{v_2},\underline{v_4},\underline{v_5}\}$	$V_3^*(D)=\{v_2,v_5,v_7\}$	7 times: $\{(q_4,v_2),(q_5,v_2),(q_6,v_3),(q_7,v_2),(q_7,v_4),(q_8,v_2),(q_9,v_5)\}$	1640
		$+v_1,+v_2,+v_3$	$-v_1,-v_3,+v_4,+v_5$	$-v_4,+v_7$		

Fig. 2. An Example of Dynamic MVs Management.

04 视图设计-淘汰物化视图

4.2 DynaMat

依据频率、执行时间、大小等对物化视图进行打分

- **LRU:** 淘汰掉最久没有被访问的物化视图
- **LFU:** 淘汰掉使用频率最少的物化视图
- **SFF:** Smaller-Fragment-First, 淘汰掉占用空间小的物化视图。
- **SPF:** Smaller Penalty First (SPF),

$$goodness(f) = t_{last\_access}(f)$$

$$goodness(f) = freq(f)$$

$$goodness(f) = size(f)$$

$$goodness(f) = \frac{freq(f) * c(f)}{size(f)}$$

频率*计算视图的开销

视图占用空间

淘汰掉“不物化带来的惩罚”小的物化视图。

04 视图设计-淘汰物化视图

4.3 DQM

异步真实运行 + 指标打分

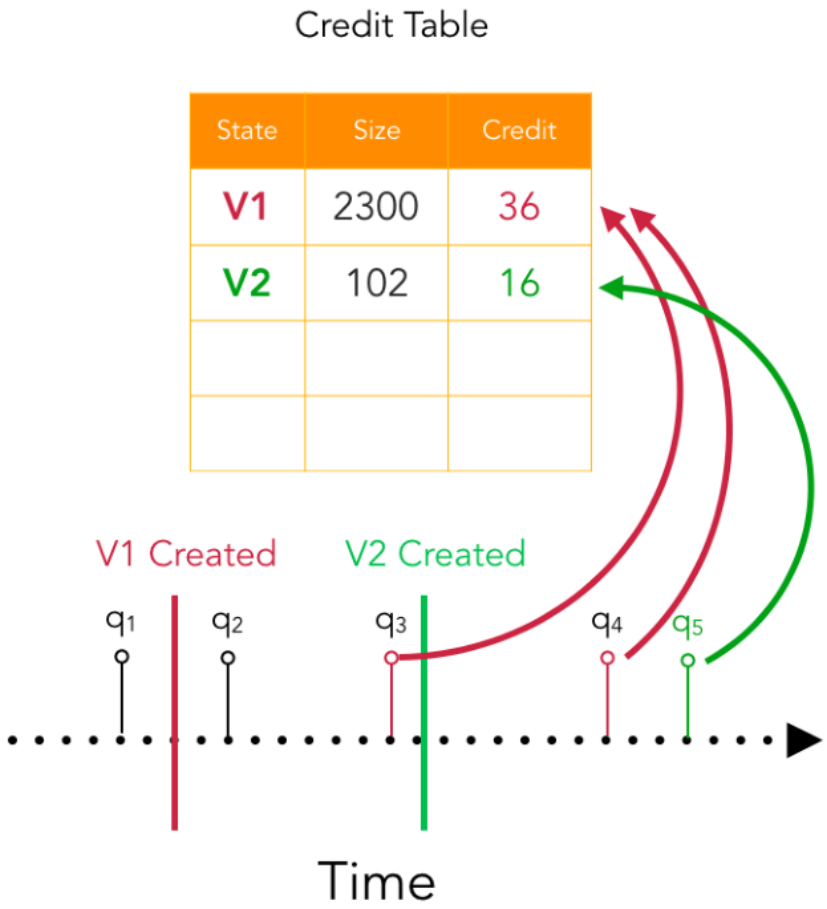
对于每个创建的视图，保持其物化的信用是：

$$R^{-1}(v) = \sum_{q \in Q} \text{Improvement}(q, v) + \text{Cost(v)},$$

$$C_{T+1}(v) = C_T(v) + R^{-1}(v).$$

$$C_{T+1}(v) = \mu \cdot C_T(v) + R^{-1}(v).$$

删除策略：驱逐信用最低的视图，直到为新的视图腾出足够的空间。



04 视图设计-淘汰物化视图

分布式存储与计算实验室

4.4 HAWC

Cost Model + ILP(执行代价最小)

$$\sum_{h \in \mathcal{H}^*} h.w \times \hat{g}_{\mathrm{q}}(h)$$

$$\hat{g}_{\mathrm{q}}(h) = \sum_{p \in h.P} \pi_p \times \hat{g}_{\mathrm{p}}(p)$$

$$\hat{g}_{\mathrm{p}}(p) = \mathbf{r} \cdot p.\mathbf{c} + \sum_{(v,p_i) \in V_p} \mu_v \times (v.\mathbf{c} - p_i.\mathbf{c})$$

$$\sum_{p \in h.P} \pi_p = 1$$

$$\pi_p \times (\mu_v + \mu_{v'}) \leq 1$$

$$\sum_{v \in \mathcal{V}} \mu_v \times v.\Delta.s \leq \mathcal{S}$$

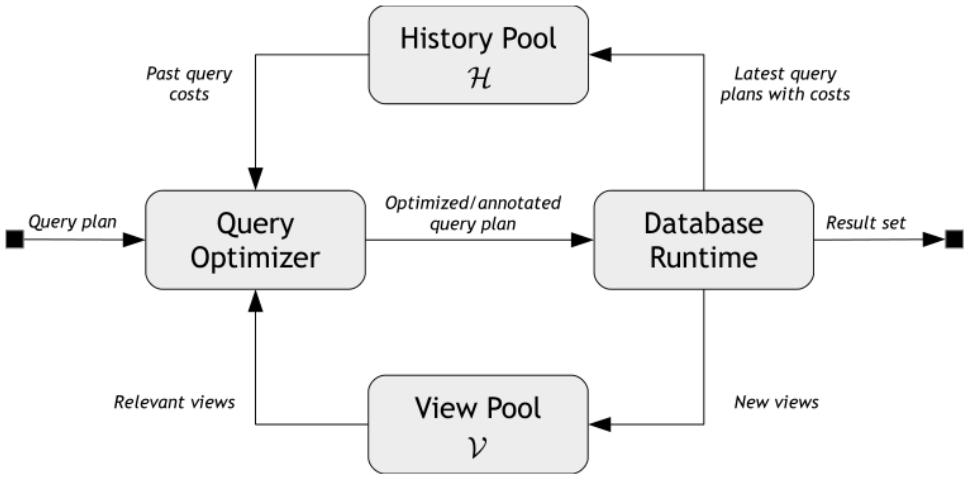


Fig. 1: Architecture of the Hawc query optimization framework.

- History Pool 多个历史查询query。
- 同一个query保留的多个不同的执行计划p。
- 使用Cost Model 计算ILP中的uij。

删除策略：删除不在ILP结果中的物化视图

04 视图设计-淘汰物化视图

4.5 GnnMV

Learned Cost Model + ILP(收益最大)

步骤:

1. 监控当前MV的累计收益（sectionV-A），当累计收益低于一定的阈值时出发 mv set maintainence。
2. queries,上次mainintainence到现在为止的query workload加入到新的图上进行训练。
3. 重新用Gnn模型估计v和q之间的收益。
4. 将问题转换为二步图通过ILP解出最终的该物化和淘汰物化哪些视图。

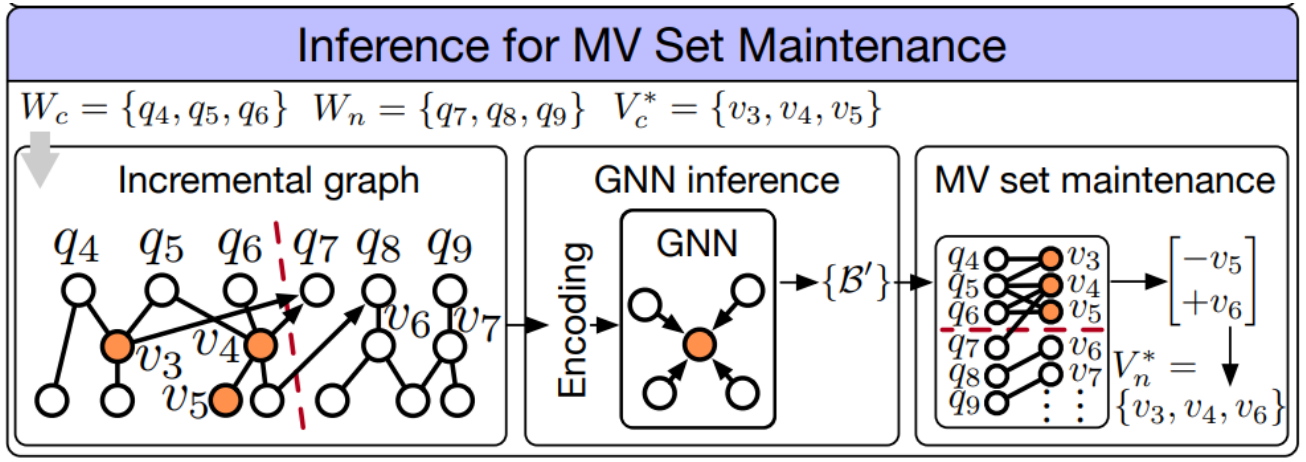
学习型Cost Model 计算uij。

删除策略：删除不在ILP结果中的物化视图

$$\arg\max_{\{x_{ik}\}} \sum_{q_k \in \mathcal{W}_c, v_i \in \mathcal{V}} \mathcal{B}(q_k, v_i) x_{ik},$$

$$s.t., \sum_{v_i \in \mathcal{V}} |v_i| \max\{x_{ik} | \forall k \in [1, |\mathcal{W}_c|]\} \leq \tau, \quad (8)$$

$$T_{ijk} + x_{ik} + x_{jk} < 3$$



05 视图维护-更新物化视图里面的数据

05 视图维护-更新物化视图里面的数据

5.1 视图维护

视图维护：在基表更新时有效地更新物化视图。

视图维护主要是数据新鲜度(一致性)和数据操作开销(延迟)之间的权衡。

更新方式：

1. **增量更新(FAST)：**添加上次刷新到当前时间段内，base表变化过的数据。
2. **全量更新(COMPLETE)：**相当于重新执行一次创建视图的查询语句。
3. **智能(FORCE)：**能增量的时候增量，不能的时候全量更新，如有的数据库不支持update的更新。

更新时机：

1. **立即更新(ON COMMIT)：**物化视图和基表同步更新，保持较高的数据新鲜度。
2. **手动更新(ON DEMAND)：**提供接口让用户按需主动触发更新。
3. **定时更新(START WITH&NEXT)：**按照一定的时间间隔更新，避免物化视图的频繁更新。

视图维护-更新物化视图里面的数据

分布式存储与计算实验室

5.2 常见DB视图维护实现情况

数据库	物化视图	刷新方式	刷新时机	查询重写
Oracle	支持	全量/增量 **默认FORCE**	1. 立即更新 **2. 手动更新** 3. 定时更新	支持
PostgreSQL	支持	全量	**1. 手动更新** 2. 定时更新	不支持
MaxCompute	支持	全量	定时	支持
Doris	支持	增量	**立即更新**	支持
MySql/TiDB	不支持	/	/	/

注：加粗为默认方式

TiDB可以通过 TiCDC + Flink 的方式实现强一致的物化视图。

06

视图利用-基于物化视图进行查询重写

06 视图利用-基于物化视图进行查询重写

分布式存储与计算实验室

6.1 视图利用

视图利用：有效地利用物化视图来加速查询处理。

关键技术：基于物化视图的查询匹配与重写。

步骤：

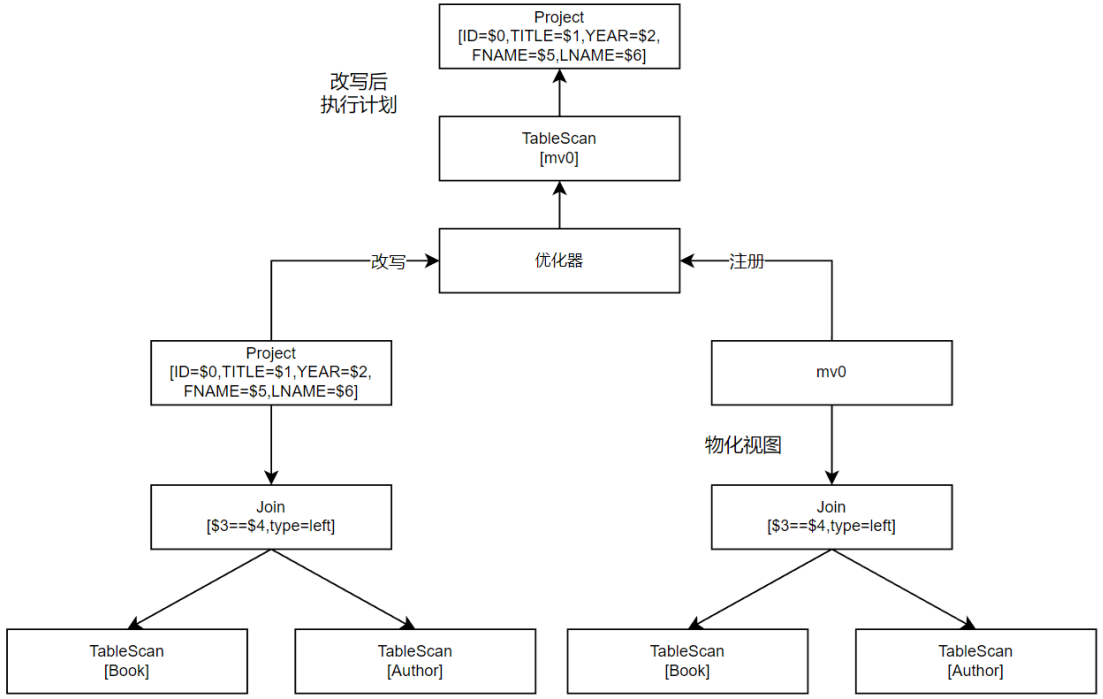
1. 将创建好的物化视图注册到优化器中。

2. 优化器将物化视图作为普通表参与查询优化过程。

方法：

1. Calcite

2.基于图的物化匹配



视图利用-基于物化视图进行查询重写

分布式存储与计算实验室

6.2 Calcite

物化匹配重写先决条件：

1．物化视图包含的行，需要覆盖用户查询所需要的行；
2．用户查询需要的数据，都能够在视图中查询得到；
3．查询语句中的表达式都能过被视图计算出来;如列不能少。
4．有相同的重复语义，比如 distinct。

物化匹配测试：

1．等价包含测试（Equijoin subsumption test);
2．范围谓词包含测试（Range subsumption test);
3．剩余谓词包含测试（Residual subsumption test)。

单表物化匹配重写步骤：

1．计算查询和视图的**等价类**（join语句）；
2．检查每一个视图等价类都是查询等价类的一个子集。查询等价类的子集。如 果不是，则拒绝该视图;

3．计算查询和视图的范围区间;
4．检查每个视图范围是否包含相应的查询范围。如果没有，拒绝该视图;
5．检查视图的残余谓词中是否都包含在用户查询语句中，如果没有，拒绝该视图。

视图利用-基于物化视图进行查询重写

6.3 Calcite

步骤 1: 计算等价类

View 等价类:{*l_orderkey*, *o_orderkey*}, {*l_partkey*, *p_partkey*}, {*o_orderdate*}, {*l_shipdate*}

Query 等价类:{*l_orderkey*, *o_orderkey*}, {*l_partkey*, *p_partkey*}, {*o_orderdate*, *l_shipdate*}

步骤 2: 检查视图等价类包含

对比发现 View 的等价类相比于 Query 的等价类少了一个，如果要用视图进行查询改写，需要创建补偿性谓词 $o\_orderdate = l\_shipdate$。

步骤 3：计算范围谓词

View 范围谓词: $\{l\_partkey, p\_partkey\} \in (150, +), \{o\_custkey\} \in (50, 500)$

Query 范围谓词: $\{l\_partkey, p\_partkey\} \in (150, 160), \{o\_custkey\} \in (123, 123)$

步骤 4：检查查询谓词范围包含关系

Query 中 $\{l\_partkey, p\_partkey\}$ 的范围 (150, 160) 在相应的 View 相关谓词的范围内，但上限不匹配，因此我们必须在 View 中添加补偿谓词 ($\{l\_partkey, p\_partkey\} <= 160$)。$\{o\_custkey\}$ 上的范围 (123, 123) 在 View 也在相应的视图谓词的范围内，但边界不匹配，因此我们必须添加补偿谓词 ($o\_custkey >= 123$) 和 ($o\_custkey <= 123$)，可以简化为 ($o\_custkey = 123$)。

步骤 5：计算剩余谓词

View 剩余谓词: $p\_name\ like\ \%abc\%$

Query 剩余谓词: $p\_name\ like\ \%abc\%, l\_quantity * l\_extendedprice > 100$

该 View 只有一个剩余谓词 ($p\_name\ like\ \%abc\%$)，它也存在于 Query 中。必须添加补偿额外的剩余谓词 ($l\_quantity * l_extendedprice > 100$)。

View:

```
Create view V2 with schemabinding as
Select l_orderkey, o_custkey, l_partkey,
       l_shipdate, o_orderdate,
       l_quantity*l_extendedprice as gross_revenue
From dbo.lineitem, dbo.orders, dbo.part
Where l_orderkey = o_orderkey
  And l_partkey  = p_partkey
  And p_partkey >= 150
  And o_custkey >= 50 and o_custkey <= 500
  And p_name like '%abc%'
```

Query:

```
Select l_orderkey, o_custkey, l_partkey,
l_quantity*l_extendedprice
From lineitem, orders, part
Where l_orderkey = o_orderkey
  And l_partkey  = p_partkey
  And l_partkey >= 150 and l_partkey <= 160
  And o_custkey = 123
  And o_orderdate = l_shipdate
  And p_name like '%abc%'
  And l_quantity*l_extendedprice > 100
```

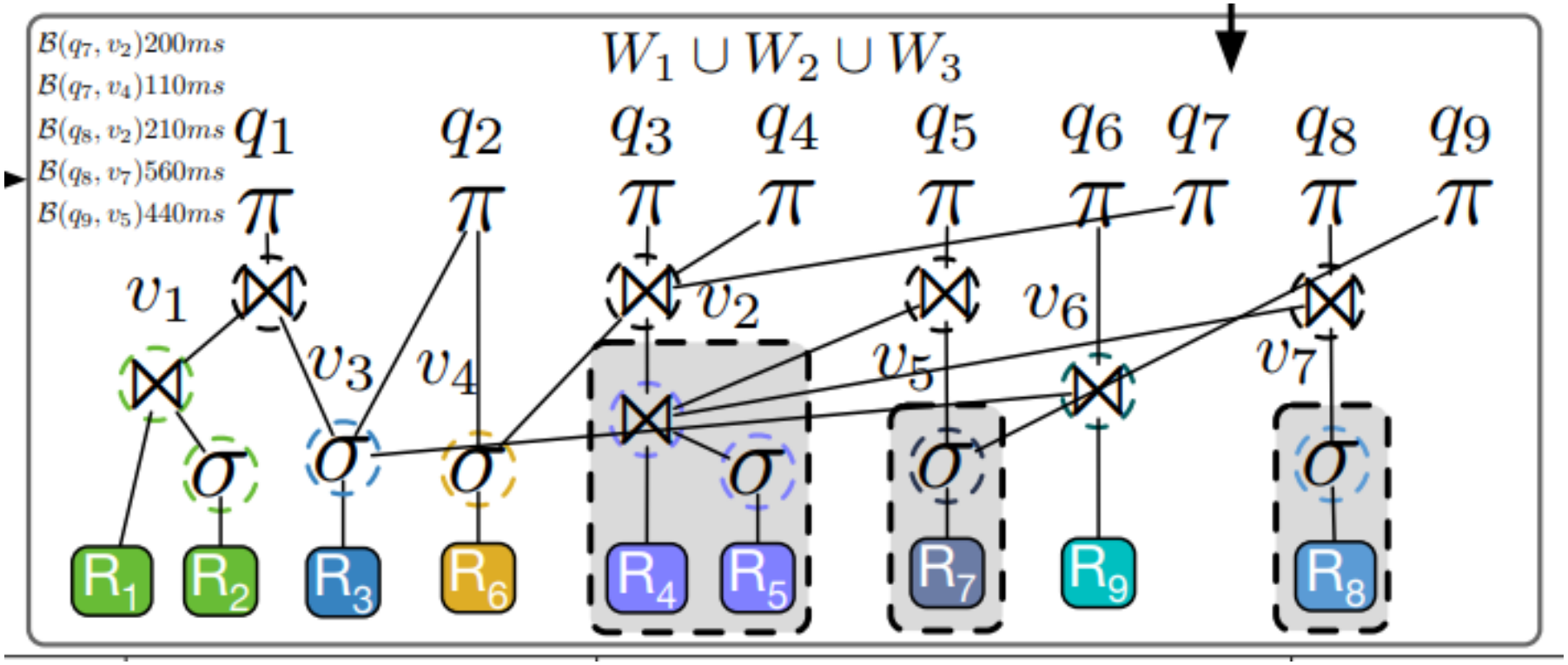
该视图通过了所有测试，它包含查询语句所有必需的行，可以基于该视图对查询语句进行查询改写。为了达到结果上的一致性，必须添加的补偿谓词是 ($o\_orderdate = l\_shipdate$)、($\{l\_partkey, p\_partkey\} <= 160$)、($o\_custkey = 123$) 和 ($l\_quantity * l\_extendedprice > 100$)。

视图利用-基于物化视图进行查询重写

分布式存储与计算实验室

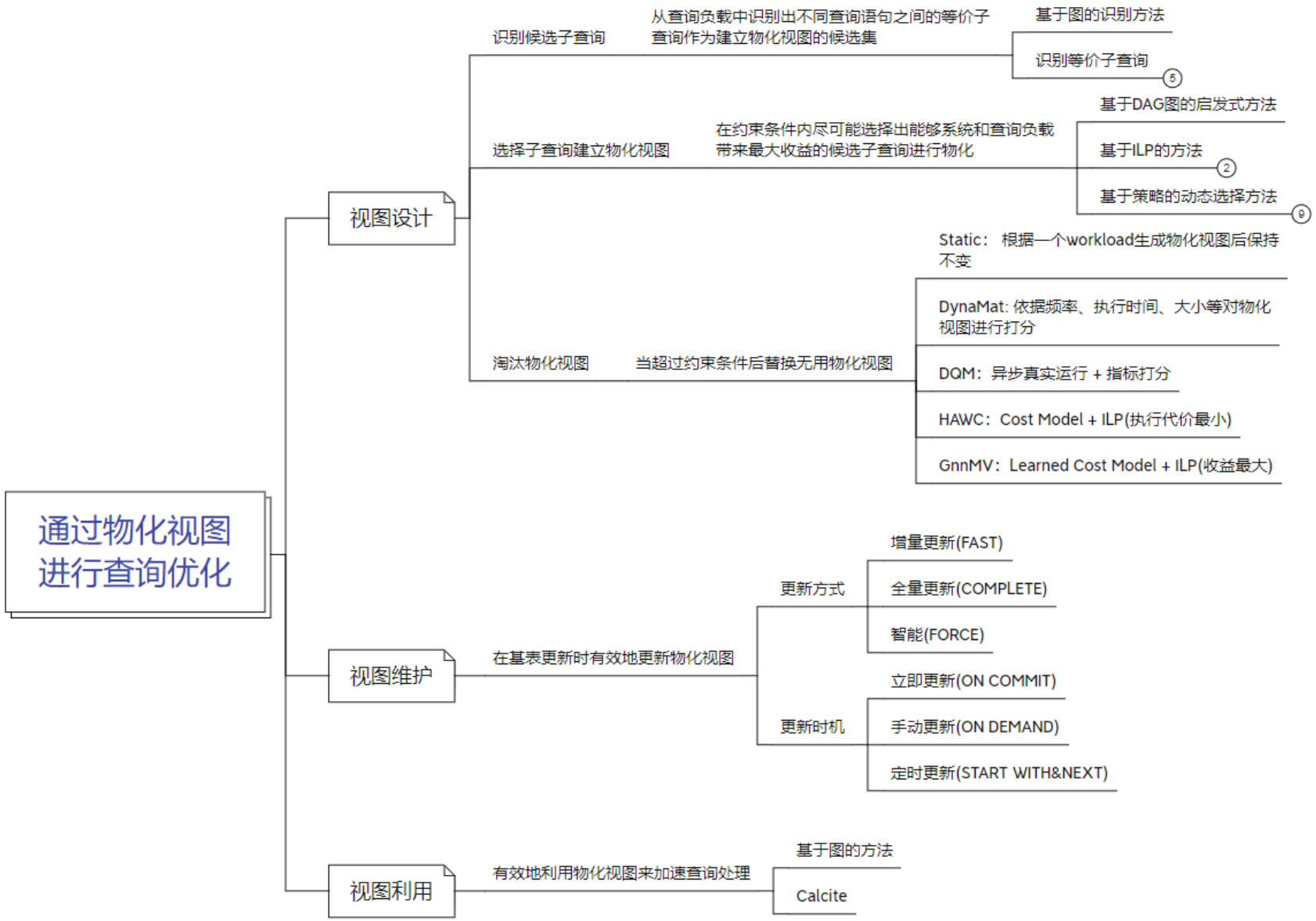
6.2 基于图的物化匹配

原理：
在图上查询语句和物化视图存在：
包含关系
重叠关系。
构建补偿谓词。



07 总结

07 总结


通过物化视图进行查询优化
视图设计
识别候选子查询
从查询负载中识别出不同查询语句之间的等价子查询作为建立物化视图的候选集
基于图的识别方法
识别等价子查询
5
选择子查询建立物化视图
在约束条件内尽可能选择出能够系统和查询负载带来最大收益的候选子查询进行物化
基于DAG图的启发式方法
基于ILP的方法
2
基于策略的动态选择方法
9
淘汰物化视图
当超过约束条件后替换无用物化视图
Static： 根据一个workload生成物化视图后保持不变
DynaMat: 依据频率、执行时间、大小等对物化视图进行打分
DQM：异步真实运行 + 指标打分
HAWC：Cost Model + ILP(执行代价最小)
GnnMV：Learned Cost Model + ILP(收益最大)
视图维护
在基表更新时有效地更新物化视图
更新方式
增量更新(FAST)
全量更新(COMPLETE)
智能(FORCE)
更新时机
立即更新(ON COMMIT)
手动更新(ON DEMAND)
定时更新(START WITH&NEXT)
视图利用
有效地利用物化视图来加速查询处理
基于图的方法
Calcite

07 文献

分布式存储与计算实验室

[01] – 综述- 李国良，周煊赫，孙佶，等. 基于机器学习的数据库技术综述[J]. 计算机学报，2020，43(11): 2019-2049.
[02] - Cosette - Cosette 官网：http://cosette.cs.washington.edu/，demo： https://demo.cosette.cs.washington.edu/ Paper：Cosette: An Automated Prover for SQL
[03] - Count Bug - 嵌套查询优化中的 COUNT bug：https://developer.aliyun.com/article/69133，Ganski，Paper: R. A.，& Wong，H. K. (1987). Optimization of nested SQL queries revisited. ACM SIGMOD Record，16(3)，23-33.
[04] - MVPP - Horng J T，Chang Y J，Liu B J. Applying evolutionary algorithms to materialized view selection in a data warehouse[J]. Soft Computing，2003，7(8): 574-581.
[05] - BigSubs - Jindal A，Karanasos K，Rao S，et al. Selecting subexpressions to materialize at datacenter scale[J]. Proceedings of the VLDB Endowment，2018，11(7): 800-812.
[06] - DQM - Liang X，Elmore A J，Krishnan S. Opportunistic view materialization with deep reinforcement learning[J]. arXiv preprint arXiv:1903.01363，2019.
[07] - RLView - Yuan，H.，Li，G.，Feng，L.，Sun，J.，& Han，Y. (2020，April). Automatic view generation with deep learning and reinforcement learning. In 2020 IEEE 36th International Conference on Data Engineering (ICDE) (pp. 1501-1512). IEEE.
[08] - AutoView - Han，Y.，Li，G.，Yuan，H.，& Sun，J. (2021，April). An autonomous materialized view management system with deep reinforcement learning. In 2021 IEEE 37th International Conference on Data Engineering (ICDE) (pp. 2159-2164). IEEE.
[09] - GnnMV - Han，Y.，Chai，C.，Liu，J.，Li，G.，Wei，C.，& Zhan，C. Dynamic Materialized View Management using Graph Neural Network.
[10] - DynaMat - Kotidis，Y.，& Roussopoulos，N. (1999). Dynamat: A dynamic view management system for data warehouses. ACM Sigmod record，28(2)，371-382.
[11] - HAWC - Perez，L. L.，& Jermaine，C. M. (2014，March). History-aware query optimization with materialized intermediate views. In 2014 IEEE 30th International Conference on Data Engineering (pp. 520-531). IEEE.
[12] - View Matching used by Calcite - dstein J，Larson P.-Å. Optimizing queries using materialized views: a practical，scalable solution[J]. ACM SIGMOD Record，2001，30(2): 331-342.

谢谢~