基数估计的相关方法介绍

- 2021.1.8

基数估计的相关方法介绍

基数机器学习数据库技术

DB

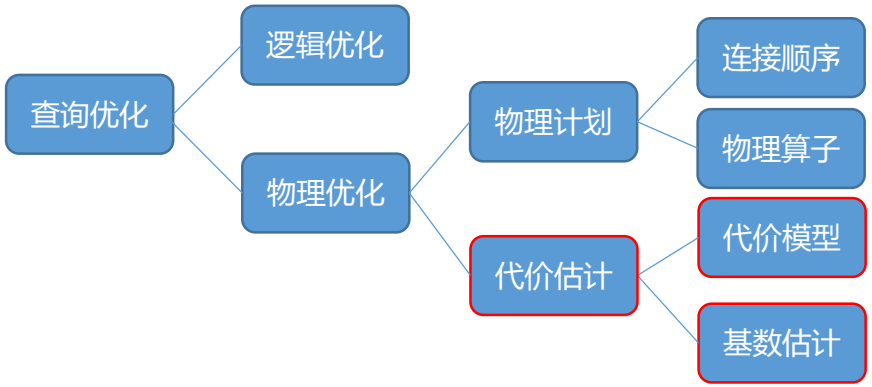
- 调参
  - 辅助调参工具
    - 基于规则，基于经验
  - 启发式自动调参
    - 划分分治采样+有界递归搜索
  - 传统机器学习调参
    - 拉丁超立方采样+高斯过程回归+FA、K均值特征降维
  - 深度强化学习调参
    - 深度网络逼近强化学习的Q-table
- 基数估计
  - 传统
    - 直方图
    - 数据画像
    - most common value
    - 采样
  - 机器学习 面向查询语句
    - 多集合卷积神经网络:MSCN
  - 机器学习 面向执行计划
    - LSTM两层树框架
    - Tpool
- 计划选择
  - 静态选择
    - 动态规划
    - 基因
    - 爬山
    - 贪心
  - 机器学习
    - DRL
    - LSTM
- 索引
  - 索引构建
    - 范围索引
    - Bloom 过滤器
  - 索引推荐
    - 在线
    - 离线
    - 半自动
    - ITLCS
- 物化视图
  - 识别等价子查询
    - 基于符号表示
    - 基于逻辑语义
  - 物化视图 静态选择
    - 基于DAG，启发式
  - 物化视图 动态选择
    - 视图打分机制
    - DRL

目录

目录

基数估计概念

- **什么是基数（Cardinality）？**
  - 数学意义上是一个集合中不重复的元素个数
  - 对列：就是不同值的个数（不可重复）。
  - 对关系/中间结果：关系的行（tuples）数（可重复）



- **什么是基数估计（Cardinality Estimation）？**
  - 代价估计 = 基数估计 + 代价模型
  - 代价模型
    总代价 = IO代价 + CPU代价
    cost = P * a_page_cpu_time + T * W

预计访问页面数　每个页面读取时间　访问元组数　权重因子

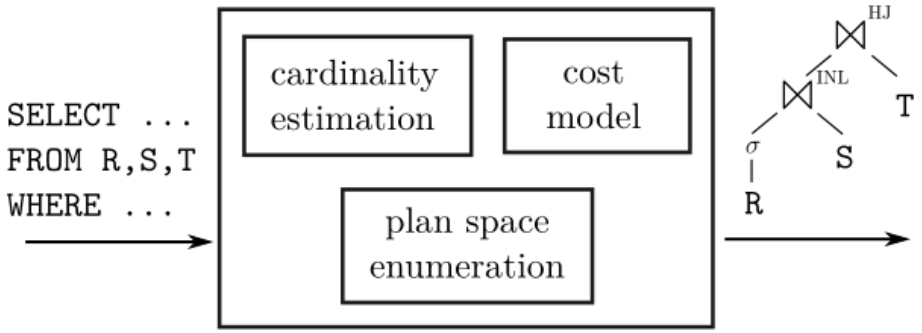


Figure 1: Traditional query optimizer architecture

  - 对物理执行计划或其子计划（中间结果）基数（行数）的估计。

基数估计概念

- **Cardinality In PostgreSQL-EXPLAN ANALYZE**

```
EXPLAIN ANALYZE SELECT *
FROM tenk1 t1, tenk2 t2
WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2 ORDER BY t1.fivethous;
```

代价估计　基数估计　实际执行时间　实际基数

```
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=717.34..717.59 rows=101 width=488) (actual time=7.761..7.774 rows=100 loops=1)
   Sort Key: t1.fivethous
   Sort Method: quicksort  Memory: 77kB
   ->  Hash Join  (cost=230.47..713.98 rows=101 width=488) (actual time=0.711..7.427 rows=100 loops=1)
         Hash Cond: (t2.unique2 = t1.unique2)
         ->  Seq Scan on tenk2 t2  (cost=0.00..445.00 rows=10000 width=244) (actual time=0.007..2.583 rows=10000 loops=1)
         ->  Hash  (cost=229.20..229.20 rows=101 width=244) (actual time=0.659..0.659 rows=100 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 28kB
               ->  Bitmap Heap Scan on tenk1 t1  (cost=5.07..229.20 rows=101 width=244) (actual time=0.080..0.526 rows=100 loops=1)
                     Recheck Cond: (unique1 < 100)
                     ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..5.04 rows=101 width=0) (actual time=0.049..0.049 rows=100 loops=1)
                           Index Cond: (unique1 < 100)
 Planning time: 0.194 ms
 Execution time: 8.008 ms
```

目录

基于统计信息的基数估计方法

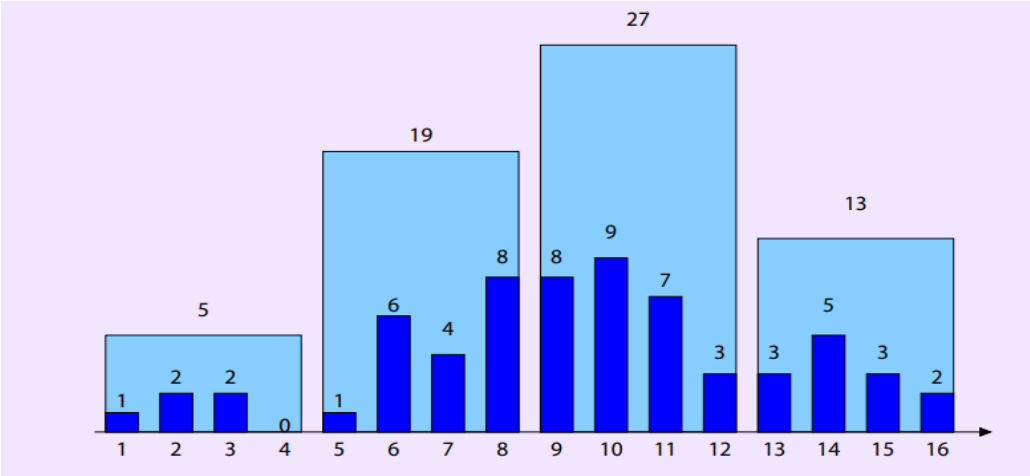
直方图（Histogram）

- **等宽直方图（Equi-width Histogram）**
- **等深直方图（Equi-depth Histogram）**
- **数据库场景下主要使用等深直方图**
  - 每列一个直方图
  - 直方图主要适用于范围查询
  - 误差来源于均匀分布（Uniform assumption）

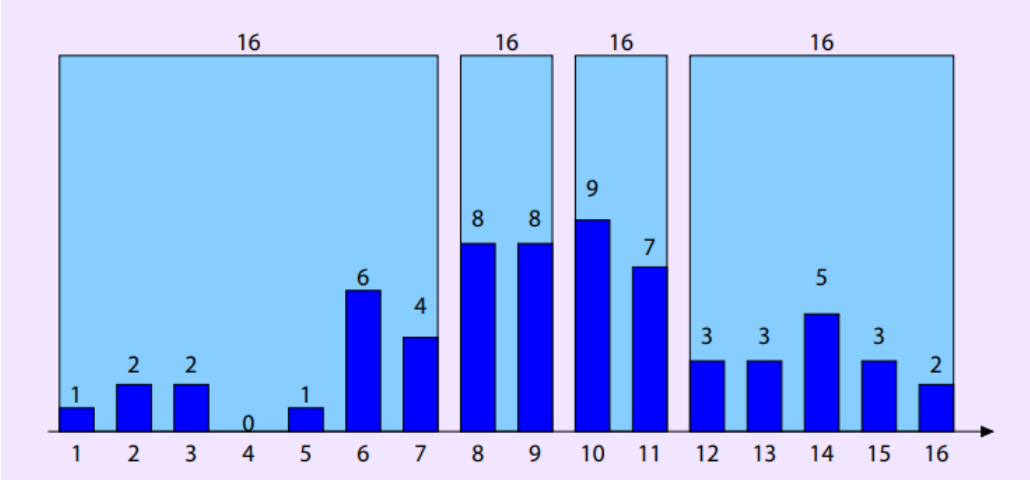
```
SELECT *
FROM s
WHERE s.f<=2;
```

实际上的值是3，估算的值16/7 * 2 ~= 4.5，

  - 能够突出高频数据（拥有更多的bucket）
  - 使用等深直方图的另一个原因是在最坏情况下其可以更好的保证误差[1]。



等宽直方图[2]



等深直方图[2]

[1]Piatetsky-Shapiro(1984). Accurate estimation of the number of tuples satisfying a condition. ACM Sigmod Record
[2]https://db.inf.uni-tuebingen.de/staticfiles/teaching/ws1011/db2/db2-selectivity.pdf

基于统计信息的基数估计方法

- **Histogram In PostgreSQL[1]-**

```
EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 1000;

                                 QUERY PLAN
-----------------------------------------------------------------------------
 Bitmap Heap Scan on tenk1  (cost=24.06..394.64 rows=1007 width=244)
   Recheck Cond: (unique1 < 1000)
   ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..23.80 rows=1007 width=0)
         Index Cond: (unique1 < 1000)
```

```
SELECT histogram_bounds FROM pg_stats
WHERE tablename='tenk1' AND attname='unique1';

        1000          histogram_bounds                  10个buckets
                                                        每个buckets占10%
--------+-----------------------------------------------
 {0, 993, 1997, 3050, 4040, 5036, 5957, 7057, 8029, 9016, 9995}
```

```
selectivity = (1 + (1000 - bucket[2].min)/(bucket[2].max - bucket[2].min))/num_buckets
            = (1 + (1000 - 993)/(1997 - 993))/10
            = 0.100697
```

- 等频率，每个bucket0.1，高=1/10*总行数

```
SELECT relpages, reltuples FROM pg_class WHERE relname = 'tenk1';

 relpages | reltuples
----------+-----------
      358 |     10000
```

```
rows = rel_cardinality * selectivity
     = 10000 * 0.100697
     = 1007  (rounding off)
```

[1]https://www.postgresql.org/docs/13/row-estimation-examples.html

基于统计信息的基数估计方法

数据画像(Sketch)

- Sketch算法是大数据统计的利器
- 常见Sketch算法
  - cout-min sketch[1]
  - FM sektch[2]
  - LinearCount [3]
  - LogLog [4]
  - HyperLogLog [5]
- Sketch算法主要使用于点查

[1] Cormode. (2005). An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms.
[2]P. Flajolet.(1985). Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences.
[3]K. Whang.(1990) A linear-time probabilistic counting algorithm for database applications. ACM Trans. Database Syst.
[4]M. Durand.(2003). Loglog counting of large cardinalities. In Algorithms - ESA.
[5]P. Flajolet. (2007). Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In AOFA.

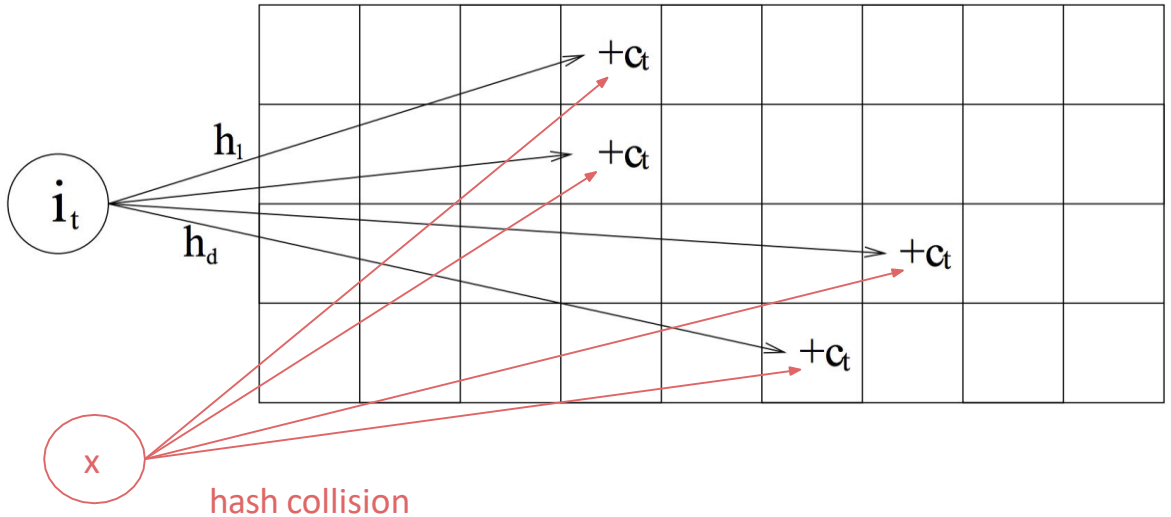
基于统计信息的基数估计方法

Count-Min sketch[1]

- 对每个item，采用d个hash函数映射到d个hash表对应的位置并累加次数
- 查询某个item出现次数时，返回d个hash表对应位置的频数的最小值

$$i_t \rightarrow T_d(h_d(i_t))\text{+=}c_t$$

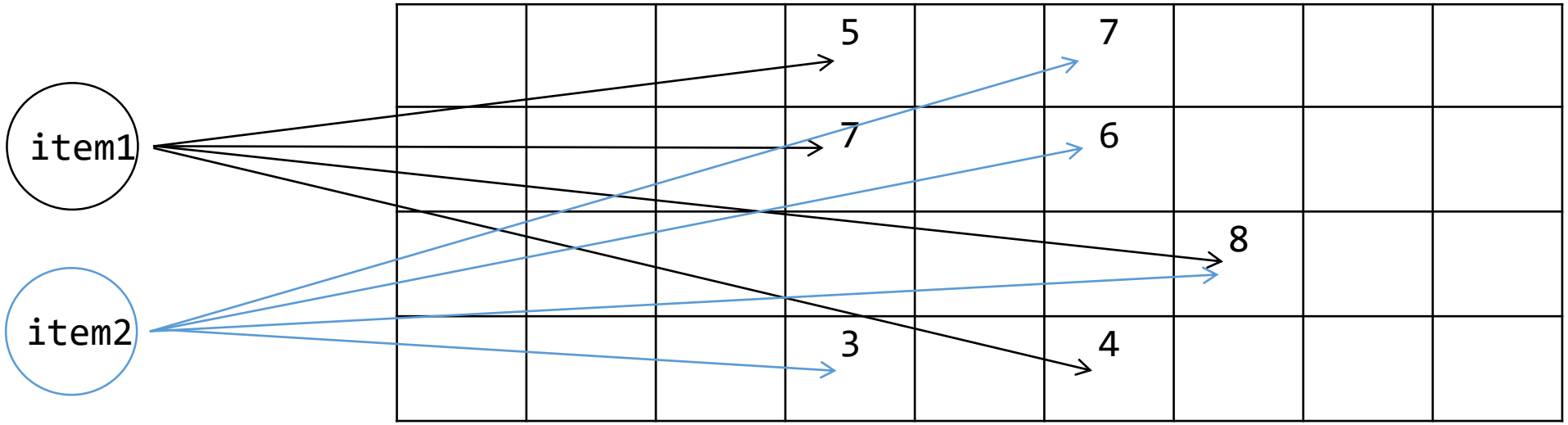
$$i_t \rightarrow min(T_d(h_d(i_t)))$$



[1] Cormode. (2005). An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms.

基于统计信息的基数估计方法

Count-Min sketch



$$i_t \rightarrow T_d(h_d(i_t))\text{+=}c_t$$

$$i_t \rightarrow min(T_d(h_d(i_t)))$$

- 可以理解成bloom过滤器在统计方面的变型
- 主要适用于点查
- TiDB中使用

基于统计信息的基数估计方法

Most Common values(MCVs)[1]

- **MCVs In PostgreSQL-点查**
  - constant in MCV list

```
EXPLAIN SELECT * FROM tenk1 WHERE stringu1 = 'CRAAAA';

                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=30 width=244)
   Filter: (stringu1 = 'CRAAAA'::name)
```

```
SELECT null_frac, n_distinct, most_common_vals, most_common_freqs FROM pg_stats
WHERE tablename='tenk1' AND attname='stringu1';

null_frac         | 0
n_distinct        | 676
most_common_vals  | {EJAAAA,BBAAAA,CRAAAA,FCAAAA,FEAAAA,GSAAAA,JOAAAA,MCAAAA,NAAAAA,WGAAAA}
most_common_freqs | {0.00333333,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003}
```

```
selectivity = mcf[3]
            = 0.003
```

```
rows = 10000 * 0.003
     = 30
```

  - constant not in MCV list

```
EXPLAIN SELECT * FROM tenk1 WHERE stringu1 = 'xxx';

                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on tenk1  (cost=0.00..483.00 rows=15 width=244)
   Filter: (stringu1 = 'xxx'::name)
```

```
selectivity = (1 - sum(mvf))/(num_distinct - num_mcv)
            = (1 - (0.00333333 + 0.003 + 0.003 + 0.003 + 0.003 + 0.003 +
                    0.003 + 0.003 + 0.003 + 0.003))/(676 - 10)
            = 0.0014559
```

```
rows = 10000 * 0.0014559
     = 15  (rounding off)
```

[1]https://www.postgresql.org/docs/13/row-estimation-examples.html

基于统计信息的基数估计方法

统计信息生成与更新

- 统计信息生成、更新时机
  - 手动（VACUUM, ANALYZE）
  - 触发(a few DDL cmds:CREATE INDEX)/定时
- 统计信息生成、更新方式
  - 随机采样生成统计信息（PostgreSQL）
  - 全表统计

基数估计的难点与数据集

统计信息小结

- 基于统计信息的基数估计方法
  - Equi-depth Histogram 适用于范围查询
  - Sketch:Count-min setch 适用于点查
  - Most common values 适用于点查
- 优点
  - 利用现有表格做统计，容易实现
  - 被广泛应用在现有商业数据库
- 挑战
  - 动态更新维护统计信息
  - 需要高效低误差的统计算法
  - 依赖于独立性假设（不能抓取列之间的关联性），

在不同列之间存在关联性时，准确率低

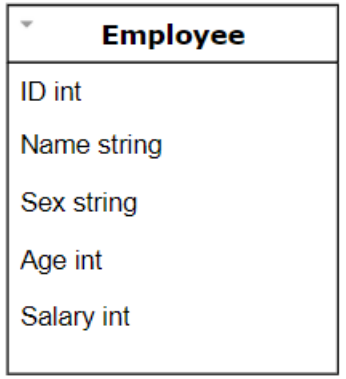
目录

基数估计的难点与数据集

基数估计的难点-列与列之间的关联性(corelation)

- 同一个表不同列之间可能存在关联性

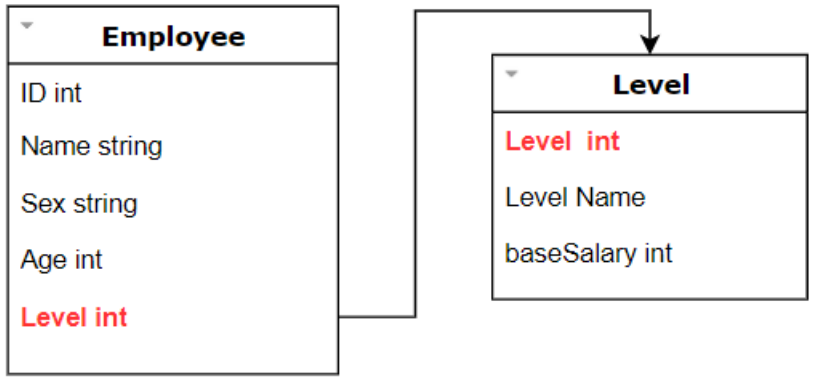
```
SELECT *
FROM Employee
WHERE Age < 25 and Salary < 15k
```



  - 如果基于独立性假设（Age < 25的选择率是40%,Salary<15k的选择率是60%)
  - Rows = 40% * 60% * Total rows =24% Total rows
  - RowsReal = 30% Total rows
  - 低估（under estimate)

- 不同表的列可能存在关联性

```
SELECT *
FROM Employee as e JOIN Level as l
ON e.level=l.level
WHERE e.Age < 25 and l.Salary < 15k
```



  - 跨表的列之间的关联性也会影响准确率

基于统计信息的基数估计方法

基数估计数据集与测试select语句-Join Order Benchmark(JOB)

- TPC-H[1], TPC-DS, Star Schema Benchmark (SSB)
  - 为了便于数据拓展，数据生成器假定了uniformity,independence
- IMDb数据集[2]
  - 真实世界的真实数据
  - 列之间有许多关联性(correlations)
  - 数据分布不均匀
- JOB[3]
  - 基于IMDb数据集
  - Join数量在3-16，平均在8个
  - 重要里程碑，后续论文都使用JOB来说明方法的有效性

```
SELECT cn.name, mi.info, miidx.info
FROM company_name cn, company_type ct,
     info_type it, info_type it2, title t,
     kind_type kt, movie_companies mc,
     movie_info mi, movie_info_idx miidx
WHERE cn.country_code ='[us]'
AND ct.kind = 'production companies'
AND it.info = 'rating'
AND it2.info = 'release dates'
AND kt.kind = 'movie'
AND ... -- (11 join predicates)
```

JOB-SQL-13d

[1] http://www.tpc.org/tpch/
[2] https://github.com/gregrahn/join-order-benchmark
[3] Leis. (2015). How Good Are Query Optimizers, Really?. Proceedings of the VLDB Endowment, 9(3).

目录

基于采样的基数估计方法

Sampling-based-简单采样

- 优点
  - 不依赖于特定假设
  - 能发现列之间的关联
- 缺点
  - 采样消失（vanishing problem），连接表多之后，符合条件的元组数逐渐减少

```
SELECT *
FROM A JOIN B on A.id=B.id
JOIN C on B.id=C.id
WHERE A.id=1
```

  - 采样量大，在join表数量多时，开销比较大([1]uses 5% of the relation size)

$$n = m\frac{m}{|A|} \quad m = \sqrt{n|A|}$$

A表主键，B表外键，其中A为主键表，m为sample量，n为join结果的元组数，如果n=25/10000|A| ,m=5/100|A|

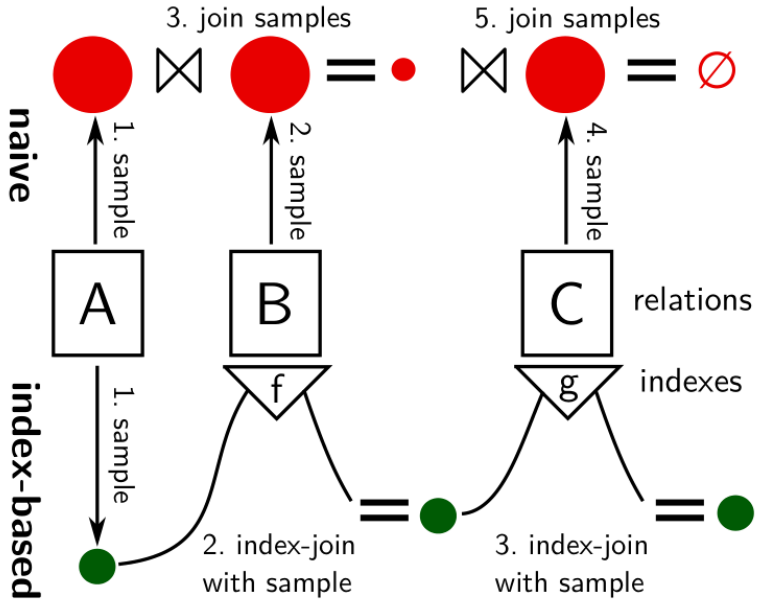


Figure 1: Naive (top) vs. index-based (bottom) join sampling

[1] W. Wu, J. F.(2016) Naughton, and H. Singh. Sampling-based query re-optimization. In SIGMOD.

基于采样的基数估计方法

Sampling-based-Index-based Sampling[1]

- 优点
  - 利用索引避免采样消失（vanishing problem）
  - 利用索引将采样样本量保持在一个常数级别
  - 减少采样开销

```
SELECT *
FROM A JOIN B on A.id=B.id
JOIN C on B.id=C.id
WHERE A.id=1
```

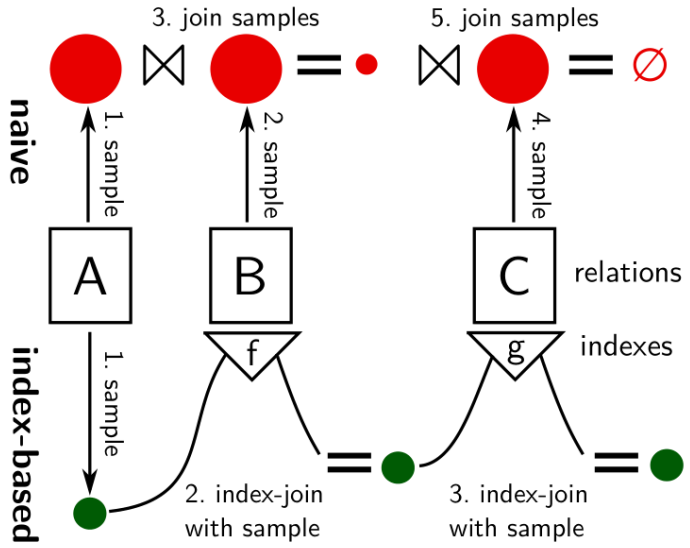


Figure 1: Naive (top) vs. index-based (bottom) join sampling

1. 在A上sample 100个records
2. 在100个records apply A.id=1得到4个records
3. 对于4个record，每个利用B.id上的index直接做等值查询（即join）的结果：
   - *record#1对应了100个B.id*
   - *record#2对应了200个B.id*
   - *record#3对应了300个B.id*
   - *record#4对应了400个B.id*

这样A和B join完得到了1000个record（文中称之为的intermediate result）

4. 遍历这1000个record，每个利用B.h上的index直接做等值查询：
   - *record#1对应了2个C.id*
   - *record#2对应了1个C.id*
   - *record#3对应了0个C.id ...*

这样得到了20个结果

5. **return** 20

[1] Leis. (2017, January). Cardinality Estimation Done Right: Index-Based Join Sampling. In Cidr.

基于采样的基数估计方法

Sampling-based-Index-based Sampling

1. 在A上sample 100个records
2. 在100个records apply A.id=1得到4个records
3. 对于4个record，每个利用B.id上的index直接做等值查询（即join）的结果：
   - record#1对应了100个B.id
   - record#2对应了200个B.id
   - record#3对应了300个B.id
   - record#4对应了400个B.id

   这样A和B join完得到了1000个record（文中称之为的intermediate result）
4. 遍历这1000个record，每个利用B.h上的index直接做等值查询：
   - record#1对应了2个C.id
   - record#2对应了1个C.id
   - record#3对应了0个C.id ...

   这样得到了20个结果
5. return 20

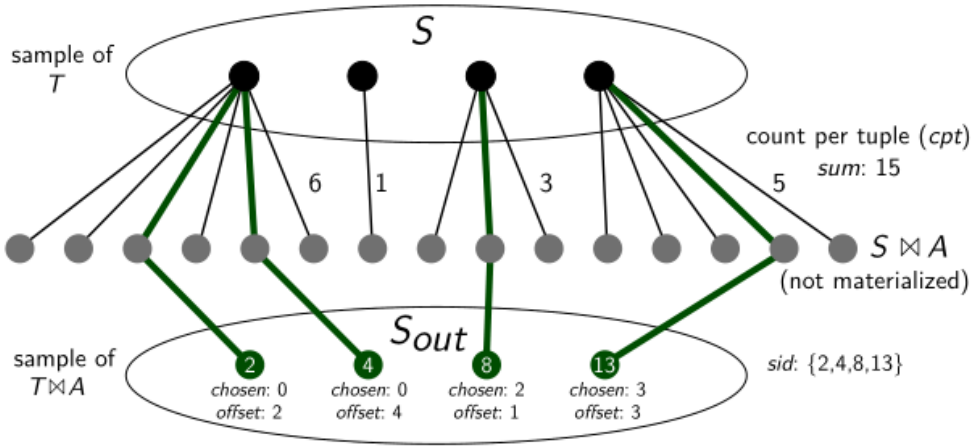


Figure 2: Illustration of Algorithm 1. Using a sample S, which consists of 4 tuples, and a suitable index, the algorithm first counts the number of join partners for each tuple in S (6, 1, 3, and 5). To compute the final sample, it then draws 4 random tuples (without replacement) from these 15 candidates

- 中间结果比较多，论文中在这一步还会进行一步采样
- 缺点
  - 强依赖于索引
  - 0-tuple problem（起始表的采样没有采样到符合条件的元素）

[1] Leis. (2017, January). Cardinality Estimation Done Right: Index-Based Join Sampling. In Cidr.

基于采样的基数估计方法

Sampling-based小结

- 采样方法的优点
  - 不依赖于特定假设
  - 能发现列之间的关联性
- 采样方法的缺点
  - 需要将一部分数据从磁盘读取到内存中代价大，不同的join顺序可能导致重新采样，查询开销大
  - 更加适用于内存数据库
  - 多表连接操作中会遇到0-tuple问题
  - Index-based sample 强依赖于索引

目录

基于机器学习的基数估计方法

可用于编码的对象

执行方向

- 面向SQL语句编码
- 面向Abstract Syntax Tree编码
- 面向Logical plan编码
- 面向Physical plan编码

难度

基于机器学习的基数估计方法

面向SQL语句编码-MSCN[1]

- 面向SQL查询语句编码
  - MSCN[1]
- 面向Physical plan编码
  - 使用强化学习来估计cardinality/cost[2]
  - Plan-Structured DNN[3]
  - Neo[4]
  - Learning-based cost estimator (Tpool) [5]

[1] Kipf, A.(2019)A. Learned Cardinalities: Estimating Correlated Joins with Deep Learning.In Cidr
[2] Jonschkowski.(2015). Learning state representations with robotic priors. Autonomous Robots, 39(3), 407-428.
[3] Marcus, R.(2019).Plan-Structured Deep Neural Network Models for Query Performance Prediction. Proceedings of the VLDB Endowment, 12(11).
[4] Marcus,R.(2019)N. Neo: A Learned Query Optimizer. Proceedings of the VLDB Endowment, 12(11).
[5] Sun, J., & Li, G.(2019) An End-to-End Learning-based Cost Estimator. Proceedings of the VLDB Endowment, 13(3).

基于机器学习的基数估计方法

面向SQL语句编码-MSCN[1]-编码举例

- MSCN 训练SQL语句train.csv

1	title t	movie_info_idx mi_idx#t.id=mi_idx.movie_id#t.kind_id	=	7	mi_idx.info_type_id	>	99#283812
2	title t##t.production_year	>	2004#1107925				
3	movie_info mi##mi.info_type_id	<	4#3624977				
4	title t	movie_companies mc#t.id=mc.movie_id#mc.company_id	<	27#134807			
5	movie_keyword mk##mk.keyword_id	<	55#54826				
6	title t	movie_companies mc	movie_info mi#t.id=mc.movie_id	t.id=mi.movie_id#t.production_year	>	1977	mc.company_id
7	movie_keyword mk##mk.keyword_id	<	35049#4007119				
8	title t	movie_companies mc#t.id=mc.movie_id##2609129					
9	movie_info_idx mi_idx##mi_idx.info_type_id	>	99#920110				
10	title t	movie_info mi	movie_info_idx mi_idx#t.id=mi.movie_id	t.id=mi_idx.movie_id#mi.info_type_id	>	16	mi_idx.info_type_id
11	title t	cast_info ci	movie_keyword mk#t.id=ci.movie_id	t.id=mk.movie_id#ci.person_id	=	890821	ci.role_id
12	title t	movie_info mi	movie_keyword mk#t.id=mi.movie_id	t.id=mk.movie_id#t.production_year	>	2010	mi.info_type_id
13	title t	movie_info_idx mi_idx#t.id=mi_idx.movie_id#t.production_year	<	2010	mi_idx.info_type_id	=	101#395802
14	cast_info ci##ci.role_id	=	10#4323018				
15	title t	cast_info ci#t.id=ci.movie_id#t.kind_id	>	1#25847549			
16	movie_info mi##mi.info_type_id	=	3#1533909				
17	title t	movie_info mi	movie_info_idx mi_idx#t.id=mi.movie_id	t.id=mi_idx.movie_id#t.production_year	>	2003	mi.info_type_id
18	movie_info mi##mi.info_type_id	>	16#2233850				
19	movie_info mi##mi.info_type_id	<	16#9565151				
20	movie_keyword mk##mk.keyword_id	=	18559#74				
21	title t	cast_info ci	movie_info_idx mi_idx#t.id=ci.movie_id	t.id=mi_idx.movie_id##40911741			
22	title t	movie_companies mc	movie_info mi#t.id=mc.movie_id	t.id=mi.movie_id#mi.info_type_id	<	4#7736878	
23	title t	movie_keyword mk#t.id=mk.movie_id#t.production_year	<	2007	mk.keyword_id	>	60992#183733
24	title t	cast_info ci	movie_info mi#t.id=ci.movie_id	t.id=mi.movie_id#t.kind_id	=	7	t.production_year
25	title t	cast_info ci	movie_info_idx mi_idx#t.id=ci.movie_id	t.id=mi_idx.movie_id#ci.person_id	>	761742	ci.role_id
26	movie_info_idx mi_idx##mi_idx.info_type_id	<	101#919850				
27	title t	movie_companies mc#t.id=mc.movie_id#t.kind_id	>	1	t.production_year	<	2010
28	title t	movie_info mi	movie_keyword mk#t.id=mi.movie_id	t.id=mk.movie_id#t.kind_id	<	7	t.production_year
29	title t	movie_keyword mk#t.id=mk.movie_id#mk.keyword_id	=	20450#123			
30	movie_info mi##mi.info_type_id	<	98#13963496				
31	title t	movie_companies mc	movie_info_idx mi_idx#t.id=mc.movie_id	t.id=mi_idx.movie_id#mc.company_type_id	=	2#1753403	
32	title t	movie_info mi#t.id=mi.movie_id#mi.info_type_id	>	98#543367			
33	title t	movie_info mi#t.id=mi.movie_id#t.production_year	<	1999#7212844			
34	title t	cast_info ci#t.id=ci.movie_id#t.kind_id	=	7	t.production_year	=	1992
35	movie_info_idx mi_idx##mi_idx.info_type_id	=	101#459925				
36	title t	movie_info mi#t.id=mi.movie_id#t.kind_id	=	1	t.production_year	=	1993
37	movie_companies mc##mc.company_type_id	<	2#1274246				
38	title t	movie_keyword mk#t.id=mk.movie_id#mk.keyword_id	>	11868#1408364			
39	movie_companies mc##mc.company_id	>	312	mc.company_type_id	=	2#1231696	
40	movie_keyword mk##mk.keyword_id	>	5071#2200397				
41	movie_info_idx mi_idx##mi_idx.info_type_id	<	100#459925				
42	movie_info_idx mi_idx##mi_idx.info_type_id	=	100#459925				
43	title t	movie_keyword mk#t.id=mk.movie_id#t.production_year	<	2001#2529592			
44	cast_info ci##ci.person_id	>	1423339#21791610				
45	title t	movie_info mi	movie_keyword mk#t.id=mi.movie_id	t.id=mk.movie_id##235420417			
46	movie_info mi##mi.info_type_id	=	7#1401902				
47	movie_keyword mk##mk.keyword_id	<	3639#2036196				
48	title t	movie_info mi#t.id=mi.movie_id#t.production_year	=	1913	mi.info_type_id	=	7#39491
49	title t##t.production_year	=	1993#26775				
50	title t##t.kind_id	>	3#1674098				
51	cast_info ci##ci.role_id	>	2#16121683				
52	title t	movie_info mi#t.id=mi.movie_id#mi.info_type_id	<	5#4923966			
53	title t	movie_keyword mk#t.id=mk.movie_id#t.kind_id	>	4	mk.keyword_id	=	5889#94
54	title t	movie_info_idx mi_idx#t.id=mi_idx.movie_id#t.kind_id	=	7#425718			
55	cast_info ci##ci.role_id	=	3#4008037				
56	cast_info ci##ci.person_id	<	170496#17216083				
57	title t	movie_companies mc#t.id=mc.movie_id#t.kind_id	>	1	t.production_year	<	2002#633430
58	title t	movie_keyword mk#t.id=mk.movie_id#t.kind_id	=	7	t.production_year	>	2011#39663
59	title t	movie_info_idx mi_idx#t.id=mi_idx.movie_id#t.kind_id	<	4	t.production_year	>	2001#354791
60	movie_keyword mk##mk.keyword_id	>	2758#2807307				
61	movie_keyword mk##mk.keyword_id	>	430#3931052				
62	movie_keyword mk##mk.keyword_id	<	15518#3330897				
63	title t##t.production_year	<	2012#2224569				
64	title t##t.kind_id	=	7	t.production_year	>	2011#150659	

[1] Kipf, A.(2019)A. Learned Cardinalities: Estimating Correlated Joins with Deep Learning.In Cidr

基于机器学习的基数估计方法

面向SQL语句编码-MSCN-总览

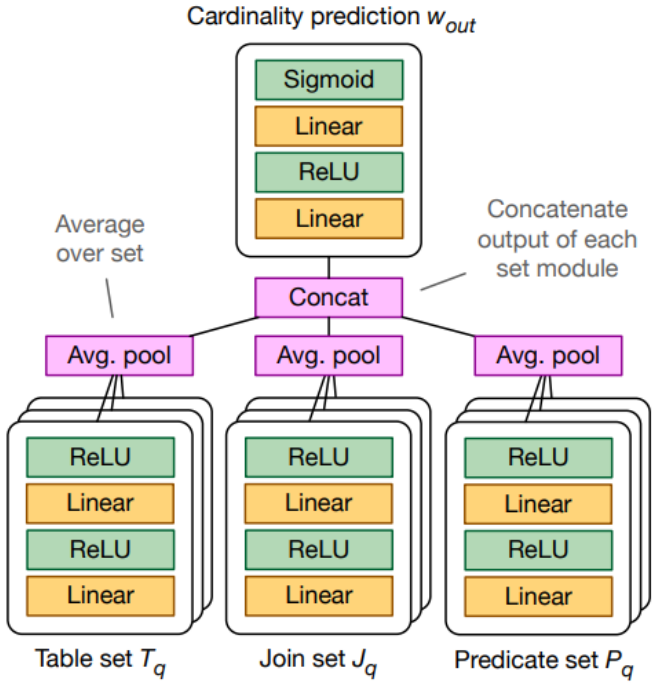
SELECT COUNT(*) FROM title t, movie_companies mc WHERE t.id = mc.movie_id AND t.production_year > 2010 AND mc.company_id = 160

Table set { [0 1 0 1 ... 0], [0 0 1 0 ... 1] } Join set { [0 0 1 0] } Predicate set { [1 0 0 0 0 1 0 0 0.72], [0 0 0 1 0 0 1 0 0.14] }

table id samples join id column id value operator id

Figure 2: Query featurization as sets of feature vectors.

- SQL语句哪些需要元素用于特征输入?
  - Tables（用于查询的表格）
  - Join（连接谓词）
  - Predicate（选择谓词）
  - Bitmaps(采样信息)
- 输出的label
  - 基数



MSCN:multi-set convolutional network

基于机器学习的基数估计方法

面向SQL语句编码-MSCN[1]-编码举例

- SQL查询语句编码

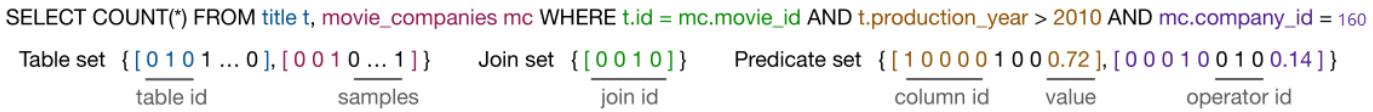


Figure 2: Query featurization as sets of feature vectors.

- table2vec:

```
{
‘cast_info ci’:           array([1., 0., 0., 0., 0., 0.]),
‘movie_companies mc‘:     array([0., 1., 0., 0., 0., 0.]),
‘movie_info mi‘:          array([0., 0., 1., 0., 0., 0.]),
'movie_info_idx mi_idx':  array([0., 0., 0., 1., 0., 0.]),
‘movie_keyword mk‘:       array([0., 0., 0., 0., 1., 0.]),
'title t':                array([0., 0., 0., 0., 0., 1.])
}
```

- join2vec:

```
{
'':                       array([1., 0., 0., 0., 0., 0.]),
't.id=ci.movie_id':       array([0., 1., 0., 0., 0., 0.]),
't.id=mc.movie_id':       array([0., 0., 1., 0., 0., 0.]),
't.id=mi.movie_id':       array([0., 0., 0., 1., 0., 0.]),
't.id=mi_idx.movie_id':   array([0., 0., 0., 0., 1., 0.]),
't.id=mk.movie_id':       array([0., 0., 0., 0., 0., 1.])
}
```

- Bitmaps(采样):每个Table一个1000bit的位图，其值为取1000行应用选择predicate，为真的行数的bit位置为1

基于机器学习的基数估计方法

面向SQL语句编码-MSCN[1]-编码举例

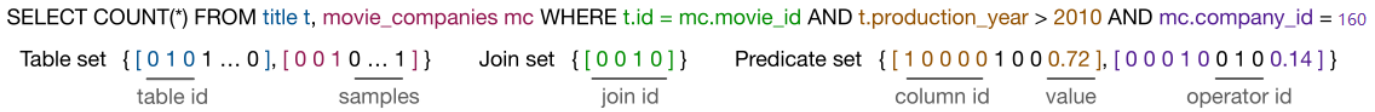


Figure 2: Query featurization as sets of feature vectors.

- SQL查询语句编码
  - column2vec:

```
{
'ci.person_id':         array([1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 'ci.role_id':          array([0., 1., 0., 0., 0., 0., 0., 0., 0.]),
'mc.company_id':        array([0., 0., 1., 0., 0., 0., 0., 0., 0.]),
'mc.company_type_id':   array([0., 0., 0., 1., 0., 0., 0., 0., 0.]),
'mi.info_type_id':      array([0., 0., 0., 0., 1., 0., 0., 0., 0.]),
'mi_idx.info_type_id': array([0., 0., 0., 0., 0., 1., 0., 0., 0.]),
 'mk.keyword_id':       array([0., 0., 0., 0., 0., 0., 1., 0., 0.]),
't.kind_id':            array([0., 0., 0., 0., 0., 0., 0., 1., 0.]),
't.production_year':    array([0., 0., 0., 0., 0., 0., 0., 0., 1.])
}
```

  - Op2vec

```
{
'<':  array([1., 0., 0.]),
'=':  array([0., 1., 0.]),
 '>': array([0., 0., 1.])
}
```

  - Value2vec，[0,1]之间的float
  - $\frac{value-\min}{max-min}$,其中min,max是这一列的min,max值

  - Lable:将真实cardinality normalize成[0,1]之间的float

    因为cardinality比较大（有些查询上亿），使用了ln函数来将数字缩放.

    $Lable = \frac{\ln(cardinality)-\ln(\min)}{Ln(max)-Ln(min)}$ 其中max,min为训练集中最大和最小的真实基数

基于机器学习的基数估计方法

面向SQL语句编码-MSCN[1]-编码举例

- SQL查询语句编码

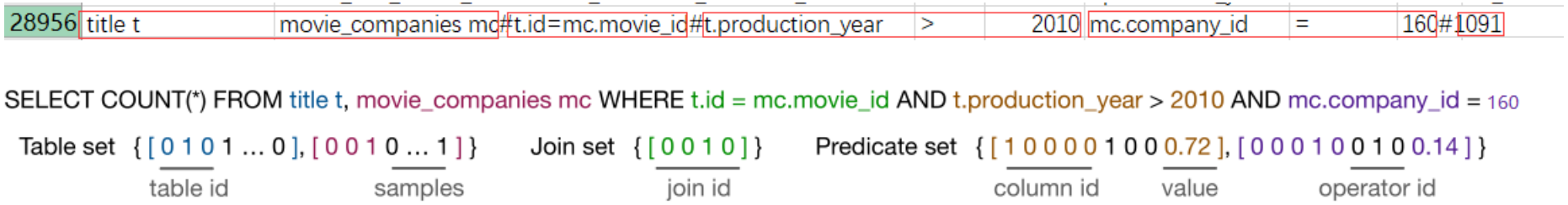


Figure 2: Query featurization as sets of feature vectors.

- Table set:{table1_one_hot|table1_bitmaps | table2_one_hot | table2_bitmaps…}

一个table: table + sample bitmaps = 1006个float

- Join set:{join1_one_hot | join2_one_hot…} 一个join predicate: 6个float
- Predicate set:{column1_one_hot|operator1_one_hot|vale1_float…}, 其中value也是用和label一样的标准化方法
- 一个predicate: 9 + 3 + 1 = 13个float
- Lable: $Lable = \frac{\ln(1091)-\ln(1)}{Ln(460456073)-Ln(1)}$=0.350658981266424 36

基于机器学习的基数估计方法

面向SQL语句编码-MSCN[1]-编码举例

- SQL查询语句编码

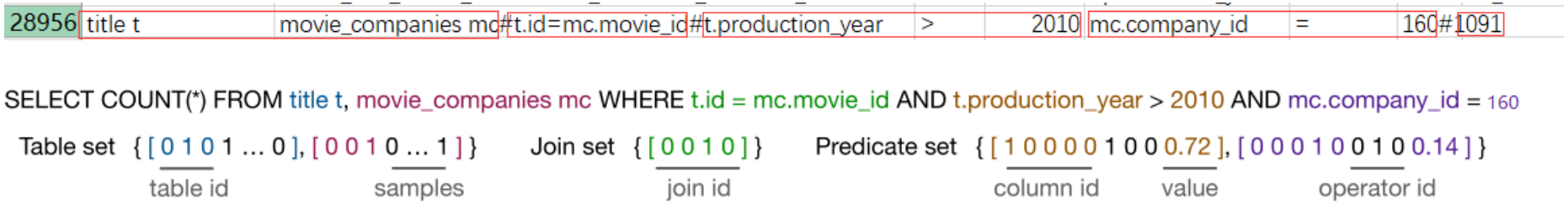
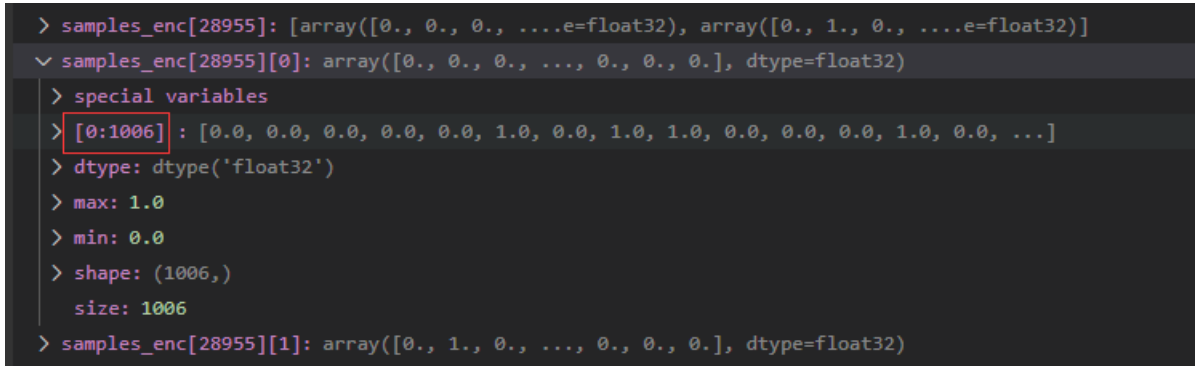
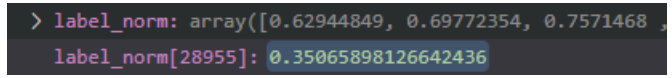
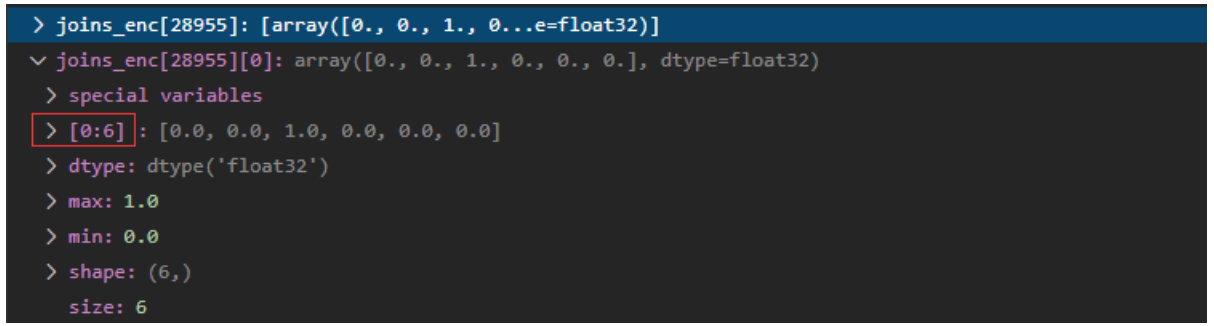
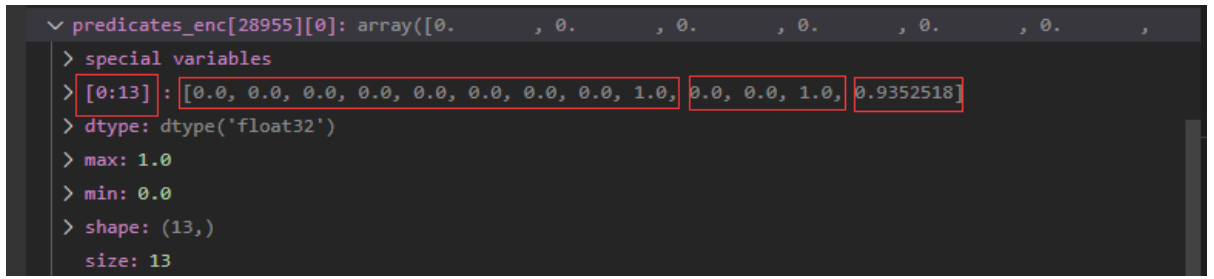


Figure 2: Query featurization as sets of feature vectors.









基于机器学习的基数估计方法

面向SQL语句编码-MSCN-模型

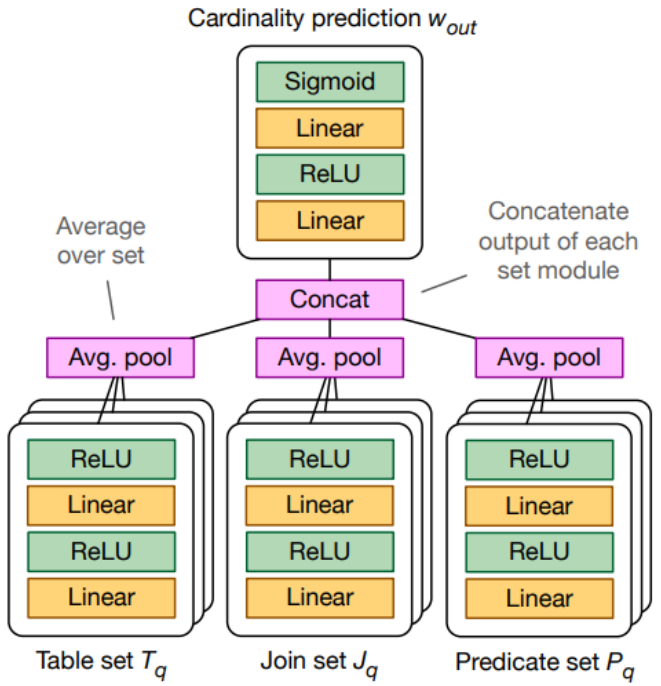
- MSCN: multi-set convolutional network
  - 不同set不同网络来学习参数
  - 将不同网络学习到的参数合并到一起

$$\text{Table module:} \quad w_T = \frac{1}{|T_q|} \sum_{t \in T_q} \text{MLP}_T(v_t)$$

$$\text{Join module:} \quad w_J = \frac{1}{|J_q|} \sum_{j \in J_q} \text{MLP}_J(v_j)$$

$$\text{Predicate module:} \quad w_P = \frac{1}{|P_q|} \sum_{p \in P_q} \text{MLP}_P(v_p)$$

$$\text{Merge \& predict:} \quad w_{\text{out}} = \text{MLP}_{\text{out}}([w_T, w_J, w_P])$$



MSCN:multi-set convolutional network

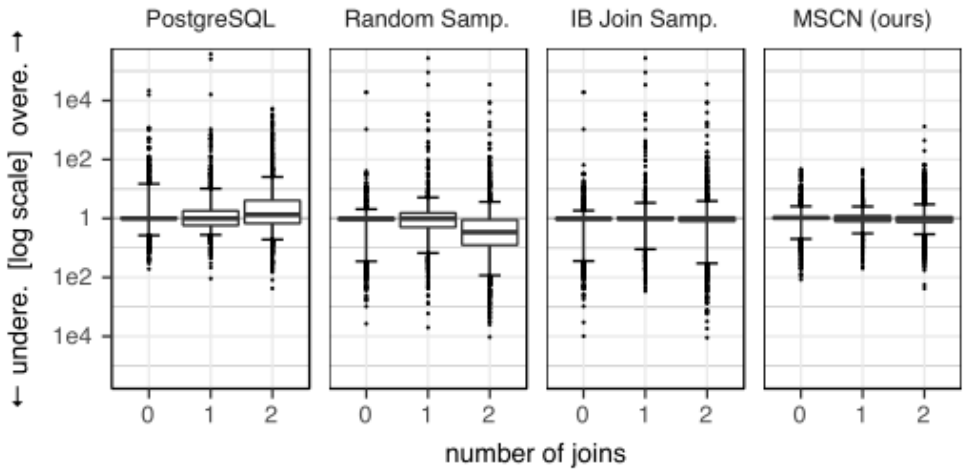
- 输出:[0,1]，通过标准化公式逆运算计算出估计的cardinality

$$Lable = \frac{\ln(cardinality) - \ln(\min)}{Ln(max) - Ln(min)}$$

基于机器学习的基数估计方法

面向SQL语句编码-MSCN-实验效果与小结

- 进步
  - 0-3个join时效果好
  - 相比Index-based smapling 估计消耗小
  - 改善了0-tuple问题
- 可以改进的地方
  - 面向SQL编码->面向物理执行计划编码
  - 当join数量上升之后效果不好，不能JOB测试上上work
  - 训练集与测试集比较局限，不支持like string等复杂语义



Q-error:真实行数1000，估计10，under q-error=100,over estimate q-error=100

$$\text{qerror} = \max\left(\frac{C}{\hat{C}}, \frac{\hat{C}}{C}\right)$$

	median	90th	95th	99th	max	mean
PostgreSQL	4.78	62.8	107	1141	21522	133
Random Samp.	9.13	80.1	173	993	19009	147
MSCN	**2.94**	**13.6**	**28.4**	**56.9**	**119**	**6.89**

Table 3: Estimation errors of 376 base table queries with empty samples in the synthetic workload.

基于机器学习的基数估计方法

面向Physical plan编码-Tpool[1]-框架

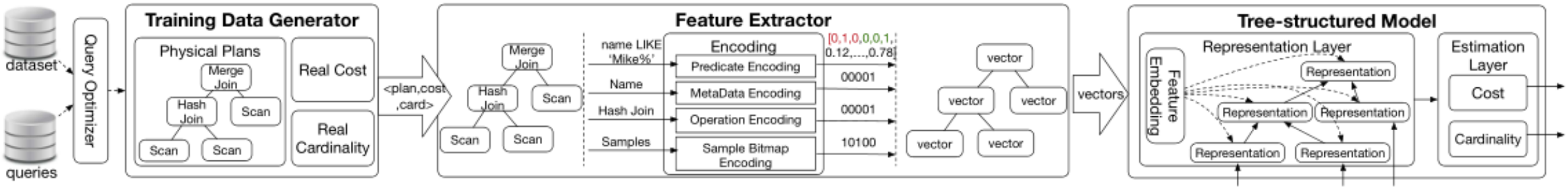


Figure 2: Architecture of learning-based cost estimator

- Training Data Generator
  - 根据数据集（IMDB，JOB），建立join graph,节点是table,边是两表之间的PK-FK。选几个表通过一定的规律生成连接predicate以及选择predicate。
  - 对于每一个query，用真实的optimizer获取其physical plan，real cost,real cardinality
  - a triple: **<a physical plan,the real cost of the plan, the real cardinality of the plan>**
- Feature Extractor:将physical plan编码
  - **One-hot encoding** :table ,column ,operator ,operation
  - **连接/选择 predicate**: <column, operator, operand>
- Tree-structured Model
  - 使用树型模型更好地学习和预测cost & cardinality

[1]Sun, J., & Li, G. An End-to-End Learning-based Cost Estimator. Proceedings of the VLDB Endowment, 13(3).

基于机器学习的基数估计方法

面向Physical plan编码-Tpool-编码举例

- Excution Plan: Physical Plan
  - 1-9个（DFS遍历）物理算子节点
- One-hot Encoding
  - Operator:选择/连接predicate

    可能用到的全部操作符
  - Operation：全部物理算子（scan,join)
  - Table Name:全部表
  - Column:全部列
- Dictionary
  - 词向量，

    再把规则转换成tensor
- Sample Datasets：同MSCN

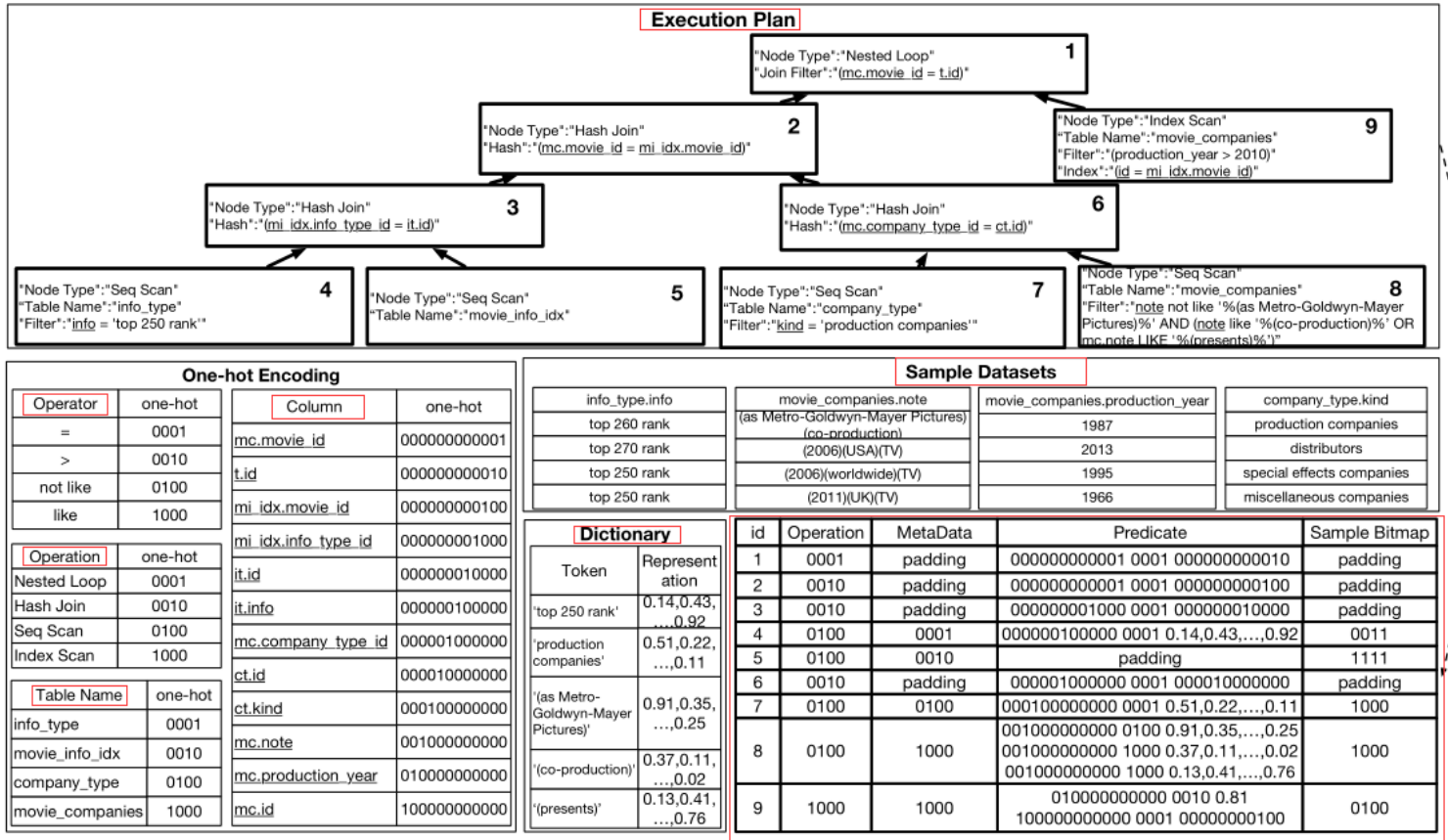
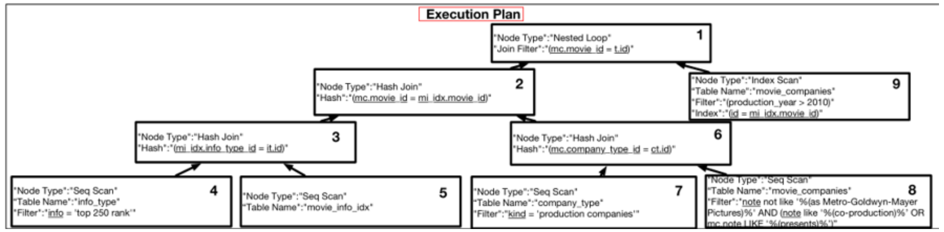


Figure 3: Running Example of query plan encoding (padding means filling up the corresponding blocks with zeros)

基于机器学习的基数估计方法

面向Physical plan编码-Tpool-总览

- Embedding Layer 处理compuand predicate特征
- Representation Layer 使用LSTM学习执行计划的特征
- Estimation Layer 使用multi learning估计cost和cardinality



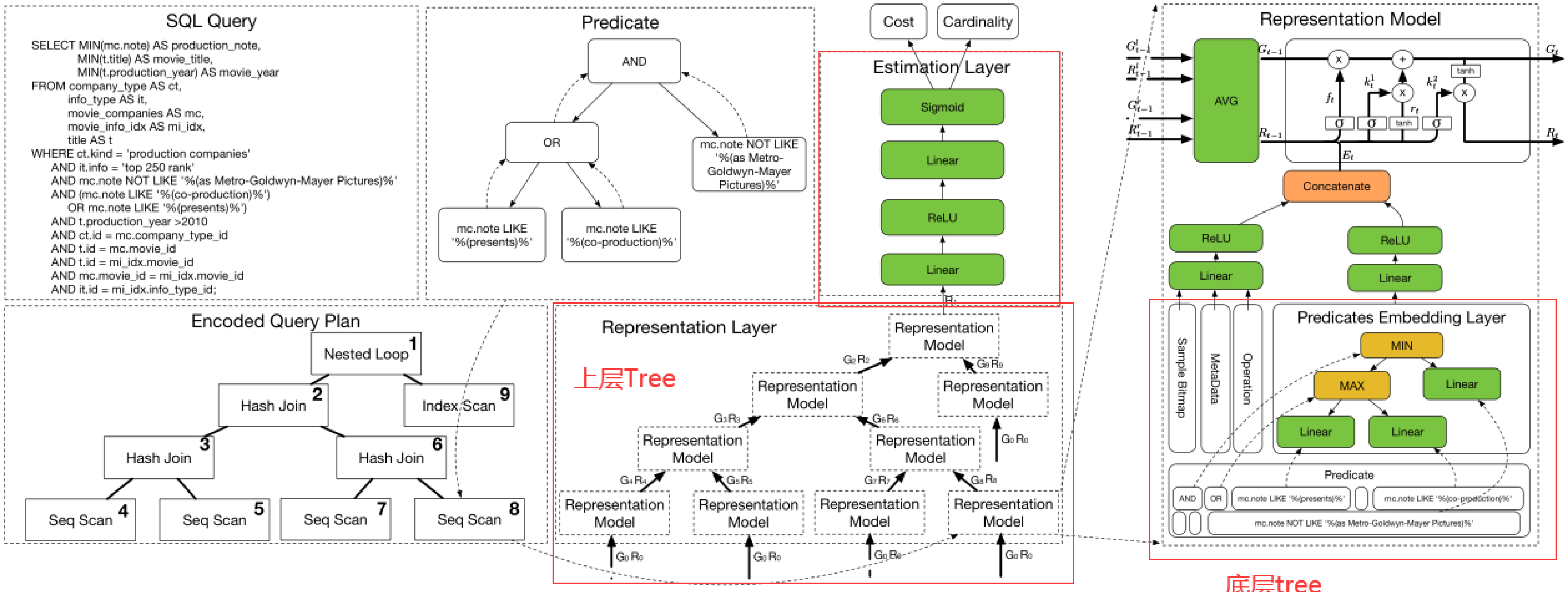


Figure 5: Two Level Tree Model

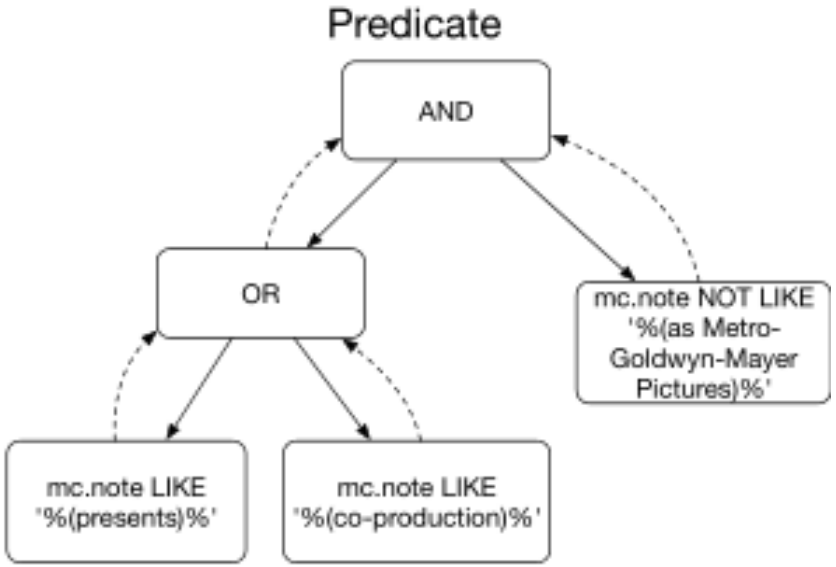
基于机器学习的基数估计方法

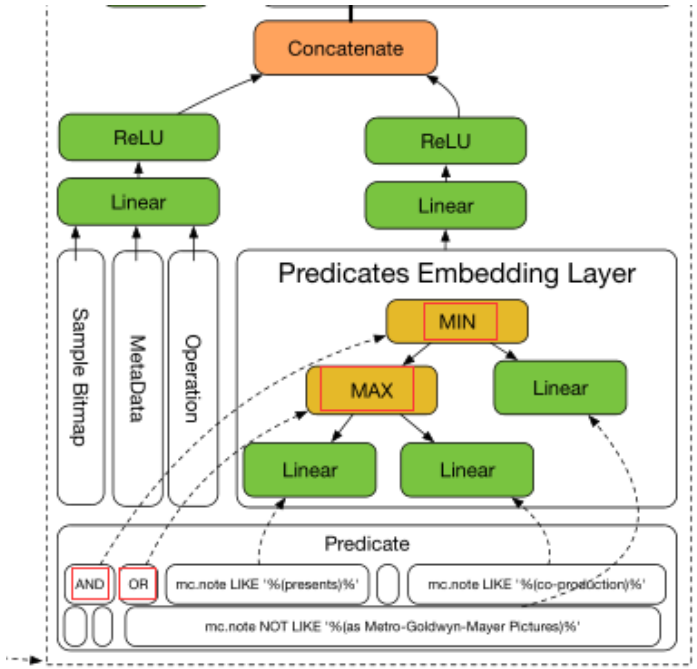
面向Physical plan编码-Learning-based cost estimator-模型

- Embedding Layer
  - 将大且稀疏的输入压缩成高维度的特征（features）
  - 使用深度优先算法（DFS，也可以用广度优先）遍历join/选择 Predicates操作
  - 对于compound predicate：

    AND使用MIN pool

    OR使用MAX pool





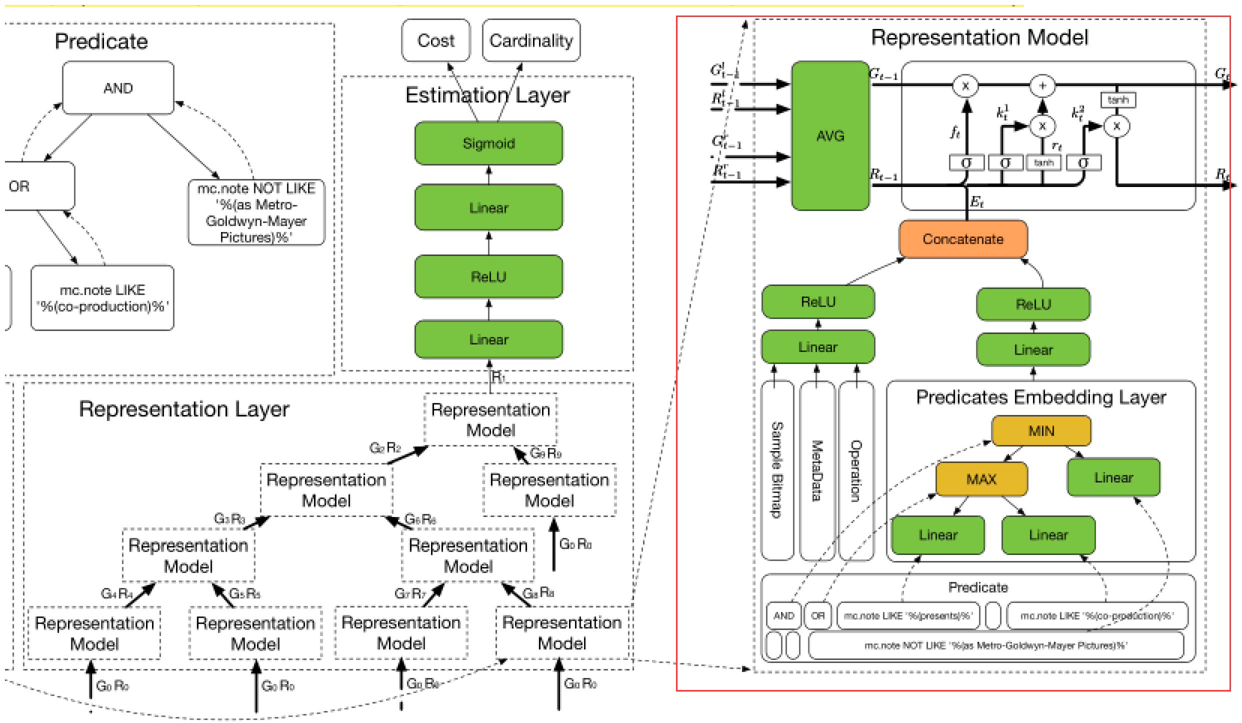
$$E = [embed(O_t), embed(M_t), embed(B_t), embed(P_t)]$$
$$embed(O_t) = ReLU(W_o \cdot O_t + b_o)$$
$$embed(M_t) = ReLU(W_m \cdot M_t + b_m)$$
$$embed(B_t) = ReLU(W_b \cdot B_t + b_b)$$
$$embed(P_t) = \begin{cases} min(embed(P_t^l), embed(P_t^r)) & P_t = and, \\ max(embed(P_t^l), embed(P_t^r)) & P_t = or, \\ W_p \cdot P_t + b_p & P_t = expr. \end{cases}$$

where $type(P_t)$ is the type of a node, which includes AND, OR, and a predicate expression.

基于机器学习的基数估计方法

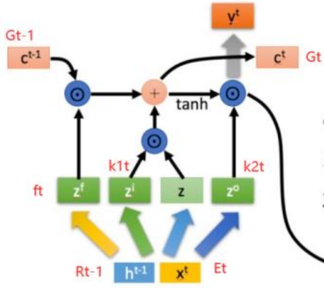
面向Physical plan编码-Learning-based cost estimator-模型

- Representation Layer
  - LSTM网络（变种RNN）
- Input
  - Xt:E
  - Gt-1(ct): 上一层的 cell state
  - Rt-1(ht):上一层的 hidden state
  - Ft(zf):控制哪些信息需要被忘记
  - K1t(zi):哪些信息应该被加入长期记忆
  - Rt(z):输入数据
  - K2t(zo):哪些数据作为hidden state 输出
- Output
  - Gt(ct):当前层输出的 cell state
  - Rt(ht):当前层输出的 hidden state



$$x_t = E$$
$$G_{t-1} = (G^l_{t-1} + G^r_{t-1})/2$$
$$R_{t-1} = (R^r_{t-1} + R^r_{t-1})/2$$
$$f_t = Sigmoid(W_f \cdot [R_{t-1}, x_t] + b_f)$$
$$k^1_t = Sigmoid(W_{k_1} \cdot [R_{t-1}, x_t] + b_{k_1})$$
$$r_t = tanh(W_r \cdot [R_{t-1}, x_t] + b_r)$$
$$k^2_t = Sigmoid(W_{k_2} \cdot [R_{t-1}, x_t] + b_{k_2})$$
$$G_t = f_t \times G_{t-1} + k^1_t \times r_t$$
$$R_t = k^2_t \times tanh(G_t)$$

where E is the vector, W is a weight and b is a bias.



$$c^t = z^f \odot c^{t-1} + z^i \odot z$$
$$h^t = z^o \odot tanh(c^t)$$
$$y^t = \sigma(W' h^t)$$

基于机器学习的基数估计方法

面向Physical plan编码-Learning-based cost estimator-模型

- Estimation Layer
  - 同时估计Cost和Cardinality,使用了multitask learning的方法。

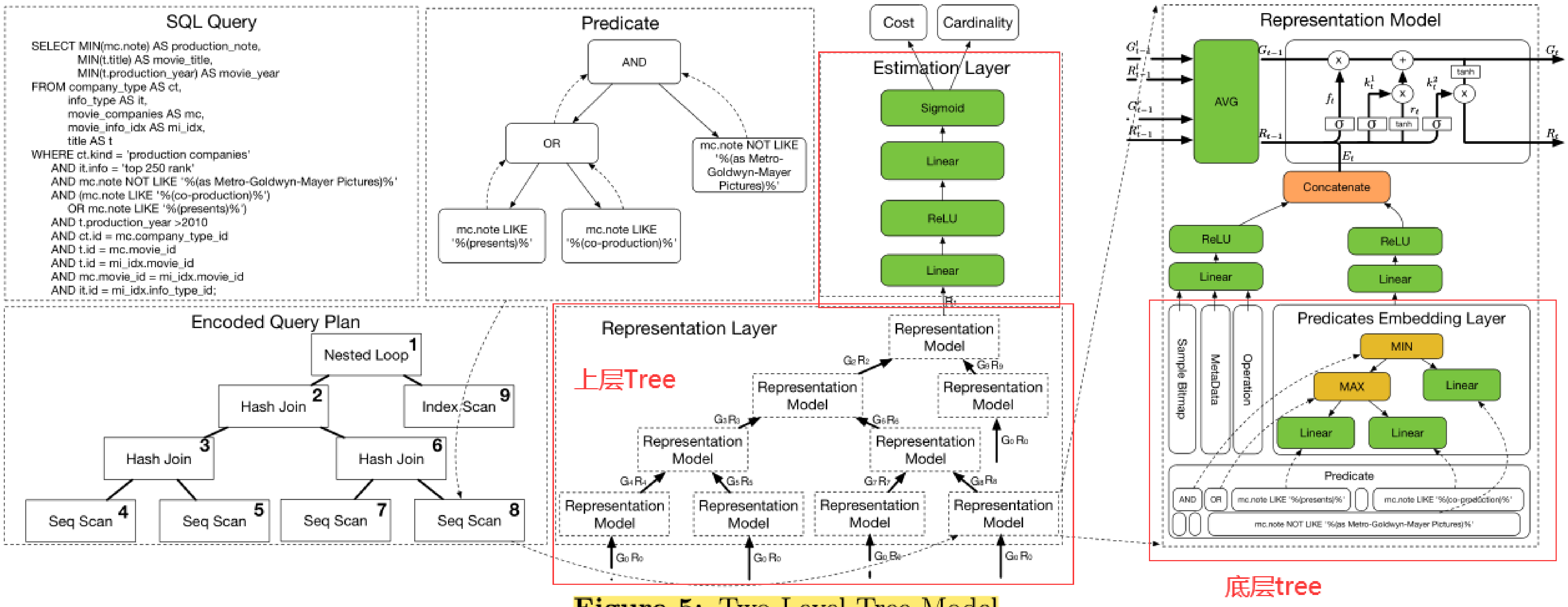


Figure 5: Two Level Tree Model

基于机器学习的基数估计方法

面向Physical plan编码-Learning-based cost estimator-实验效果

- 对比结论
  - Sample > Non-Sample
  - TLSTM > CNN(MSCN) 完全不同同两种模型
  - TLSTM > TNN
  - Tpool > TLSTM
  - TLSM-Multi > TLSM
- 小结
  - 面向physical plan编码
  - 使用two level tree model及LSTM
  - 提出string的一种解决方案，在JOB上更好的k
  - 工作量比较大，做了很多对比
  - 实验效果比较好
  - 当具有大量predicate,选择率较低时效果不好 [1]

Table 7: Cardinality on numeric workloads(test errors)

Synthetic	median	90th	95th	99th	max	mean
PostgreSQL	1.69	9.57	23.9	465	373901	154
MySQL	2.07	22.6	50.6	625	458835	353
Oracle	1.97	12.4	40.1	473	545912	378
MSCN-NoSamp	2.14	6.72	11.5	114	1870	23.6
TLSTM-NoSamp	1.97	5.53	9.13	81.5	988	10.3
MSCN	1.19	3.32	6.84	30.51	1322	2.89
TNN	1.40	5.51	10.7	43.1	441	3.57
TLSTM	1.20	3.21	6.12	25.2	357	2.87
TPool	**1.18**	**3.19**	**6.05**	**24.5**	**323**	**2.81**
Scale	median	90th	95th	99th	max	mean
PostgreSQL	2.59	200	540	1816	233863	568
MySQL	3.08	90.1	329	7534	54527	426
Oracle	2.43	114	482	3412	102833	397
MSCN-NoSamp	2.33	96.1	257	1110	4013	131
TLSTM-NoSamp	2.06	69	176	931	3295	78.2
MSCN	**1.42**	37.4	140	793	3666	35.1
TNN	1.59	58.7	141	573	2238	31.3
TLSTM	1.43	38.8	139	469	1892	28.1
TPool	**1.42**	**37.3**	**125**	**345**	**1813**	**26.3**
JOB-light	median	90th	95th	99th	max	mean
PostgreSQL	7.93	164	1104	2912	3477	174
MySQL	9.55	303	685	2256	2578	149
Oracle	8.32	374	976	2761	3331	157
MSCN-NoSamp	5.43	126	978	1310	2020	100
TLSTM-NoSamp	5.18	97.3	613	864	1541	72.3
MSCN	3.82	78.4	362	927	1110	57.9
TNN	**2.95**	76.8	275	799	902	49.8
TLSTM	3.73	50.8	157	256	289	24.9
TPool	3.51	**48.6**	**139**	**244**	**272**	**24.3**

Table 8: Cost on numeric workloads (test errors)

Synthetic	median	90th	95th	99th	max	mean
PostgreSQL	15.1	65.1	173	1200	8040	62.7
MySQL	4.51	39.7	94.7	449	7203	32.4
Oracle	6.72	41.1	124	796	6674	56.1
MSCN-NoSamp	10.3	24.7	234	569	2110	31.6
TLSTM-NoSamp	5.34	21.2	153	328	1345	19.8
MSCN	3.14	7.43	18.1	65.8	739	10.3
TNN	1.49	4.50	10.6	61.5	718	4.35
TLSTM	1.56	4.47	10.7	57.7	689	4.45
TLSTM-Multi	1.49	4.33	10.2	55.8	624	4.16
TPool	**1.48**	**4.12**	**10.1**	**47.6**	**532**	**3.99**
Scale	median	90th	95th	99th	max	mean
PostgreSQL	13.3	38.9	81.1	718	1473	35.7
MySQL	4.25	37.4	131	577	5157	40.7
Oracle	6.49	27.7	61.4	623	3612	31.5
MSCN-NoSamp	3.32	20.9	30.5	274	1173	21.2
TLSTM-NoSamp	2.19	13.4	21.7	228	1162	14.9
MSCN	1.79	10.6	27.1	88.8	1027	8.22
TNN	1.61	5.37	13.5	72.7	714	5.53
TLSTM	1.58	5.51	14.4	70.1	611	5.21
TLSTM-Multi	1.56	5.56	12.2	68.6	**254**	4.41
TPool	**1.54**	**5.29**	**11.9**	**67.6**	**254**	**4.39**
JOB-light	median	90th	95th	99th	max	mean
PostgreSQL	26.8	332	696	2740	3020	173
MySQL	9.47	102	342	1293	2228	84.5
Oracle	12.3	157	278	1366	1825	102.1
MSCN-NoSamp	12.4	152	231	1071	1553	62.7
TLSTM-NoSamp	10.4	103	217	986	1271	38.3
MSCN	4.75	11.3	40.1	563	987	27.4
TNN	2.06	25.5	134	293	401	19.1
TLSTM	3.66	32.1	80.3	445	583	17
TLSTM-Multi	**1.85**	13.2	22.9	**95**	**123**	5.81
TPool	**1.85**	**11.1**	**20.3**	101	125	**5.76**

[1] Hasan. (2020). Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. SIGMOD/PODS '20

基于机器学习的基数估计方法

面向Physical plan编码-Learning-based cost estimator-String Embedding

- Rule Generation
  - String的vector 稀疏和不连续不容易学习
  - 论文定义了一套规则来替代like Din%, %06%,

  对like的编码转换成了对新规则的编码
- String Indexing
  - 大量的string,用字典存消耗比较大,

  使用了trie index (前缀树)

Table 4: Candidate rules for "Dinos in Kas" → "Din%"

	Rules
"Dinos" → "Din"	$\langle Prefix, P_t(\text{“D”})P_l, 3\rangle$ $\langle Prefix, P_C P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{“i”})P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{“in”})P_l, 3\rangle$ $\langle Prefix, P_t(\text{“Din”})P_l, 3\rangle$
"Dinos in" → "Din"	$\langle Prefix, P_t(\text{“D”})P_l P_s P_l, 3\rangle$ $\langle Prefix, P_C P_l P_s P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{“i”})P_l P_s P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{“in”})P_l P_s P_l, 3\rangle$ $\langle Prefix, P_t(\text{“Din”})P_l P_s P_l, 3\rangle$
"Dinos in Kas" → "Din"	$\langle Prefix, P_t(\text{“D”})P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_C P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{“i”})P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_C P_t(\text{“in”})P_l P_s P_l P_s P_C P_l, 3\rangle$ $\langle Prefix, P_t(\text{“Din”})P_l P_s P_l P_s P_C P_l, 3\rangle$

基于机器学习的基数估计方法

机器学习方法小结

- 机器学习方法的优点
  - 不依赖于特定假设
  - 通过机器学习的方法抓取不同列之间的关联性
  - 可以提前训练好模型，估计耗时比较小

- 机器学习方法的挑战
  - 仍在研究阶段，并没有商用的例子，实用性有待考证
  - 数据集不同时，需要重新训练
- 研究的方向
  - 面向physical plan编码
  - 更加高效地利用SQL语句、表格统计信息、采样数据
    列之间的关系等特征
  - 尝试各种可能的网络（MLP，CNN，LSTM，GCN,GAN)
  - 对查询语句的选择率(selectivity)进行估计：Naru[1]，[2]

[1]Yang, Z.(2019) Deep Unsupervised Cardinality Estimation. Proceedings of the VLDB Endowment
[2] Hasan(2020). Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. SIGMOD

目录

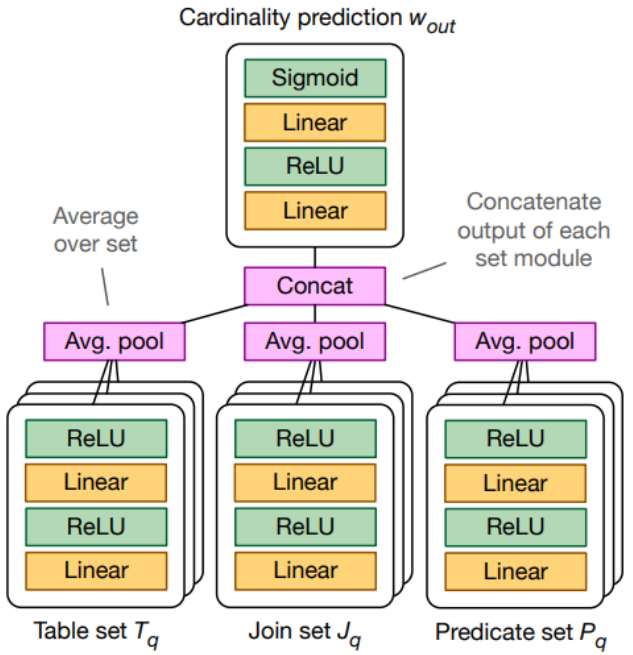
后续实验研究方向

结合现有工作和图卷积神经网络

- 改进MSCN
  - 将用于标准化的max替换成应用选择条件之后每个表符合条件的行数（通过采样bitmaps获取）的乘积

- 结合MSCN和Learning-based cost estimator（Tpool）
  - MSCN是重要里程碑，代码开源，可以学习代码实现
  - Tpool的面向 physical plan的编码
  - Tpool的two level tree model

- 图神经网络
  - 利用SQL查询语句workload建立不同列之间的关系图（类似于Tpool生成数据的方法PK-FK做边）
  - 使用图神经网络来提取利用不同列之间的关联性
- 对查询语句的选择率(selectivity)进行估计



[1]Yang, Z.(2019) Deep Unsupervised Cardinality Estimation. Proceedings of the VLDB Endowment
[2] Hasan(2020). Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. SIGMOD

目录

基数估计的相关方法介绍

总结

- 基于统计信息的基数估计方法
  - 广泛应用在商业数据库
  - 利用现有表格做统计，单表查询准确率高
  - 依赖于独立性假设，在不同列之间存在相关性时，准确率低
- 基于采样的基数估计方法
  - 采样思想和方法广泛应用在商业数据库
  - 不依赖于特定假设，能发现不同列之间的关联性
  - 存在vanish problem & 0-tuple问题，采样消耗影响估计时间
- 基于深度学习的基数估计方法
  - 仍在研究阶段，应用少
  - 不依赖于特定假设，设计巧妙地模型能够抓取不同列之间地关联性，估计耗时比较小
  - 数据集不同时，可能需要重新训练

谢谢