16.4 Choice of Evaluation Plans

— Query optimization

- How to generate the ***evaluation plan*** for a relational algebra expression with lower or the lowest cost
  - generating of the optimized *query trees*
  - selecting
    - implementation algorithms for each operations in the tree
    - *pipeline* or *materialization* strategy for the expression

- E.g. Fig.16.2

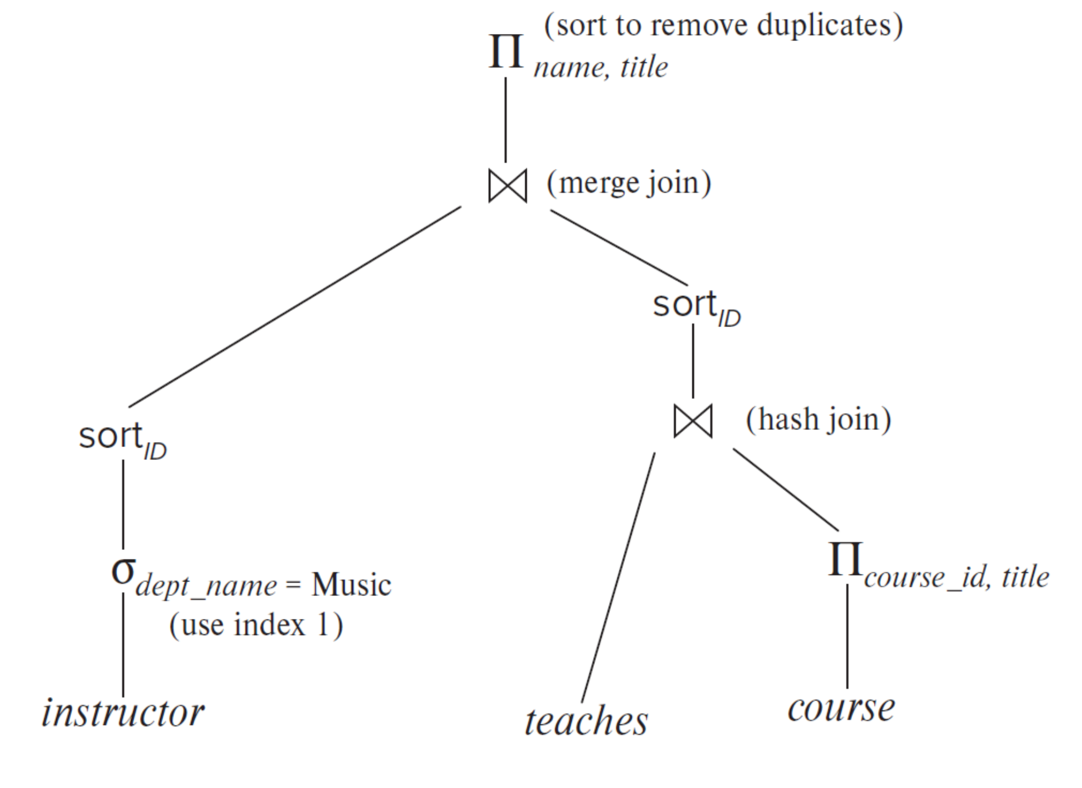


Fig.16.2 An evaluation plan

Choice of Evaluation Plans

- Must consider the interaction of evaluation techniques when choosing evaluation plans
  - choosing the cheapest algorithm for each operation independently may not yield best overall algorithm. E.g.
    - merge-join may be costlier than hash-join, but may provide a sorted output which reduces the cost for an outer level aggregation.
    - nested-loop join may provide opportunity for pipelining

Choice of Evaluation Plans

- **Query optimization**
  - choosing the evaluation plan with ***lowest*** or the ***lower cost***

- Practical query optimizers incorporate elements of the following two broad approaches:

  1. ***Cost-based*** (16.4.1, 16.4.2)

     Search all the plans and choose the best plan in a cost-based fashion.

  2. ***heuristic optimization*** (16.4.3)

     Uses heuristics to choose a plan

- Practical query optimizers incorporate elements of these two approaches

16.4.1 Cost-Based Join Order Selection

- The *join* operator is the basis for multiple table query, and its cost is expensive
- For a relational algebra expression with several *join* operation , the join order influences the query cost of the expression.
- Consider finding the best join-order for $r_1 \bowtie r_2 \bowtie . . . r_n$.
- There are $(2(n – 1))!/(n – 1)!$ different join orders for above expression. With $n = 7$, the number is 665280, with $n = 10$, the number is greater than 176 billion!
- No need to generate all the join orders. Using dynamic programming, the least-cost join order for any subset of $\{r_1, r_2, . . . r_n\}$ is computed only once and stored for future use.

Join Order Selection by Dynamic Programming

- No need to generate all the join orders. Using dynamic programming, the least-cost join order for any subset of $\{r_1, r_2, \ldots r_n\}$ is computed only once and stored for future use

- 方法：

  P768, Fig.16.7;

- 类似于《算法设计与分析》中的矩阵连乘积

$(A((BC)D))$ $(A(B(CD)))$ $((AB)(CD))$

$(((AB)C)D)$ $((A(BC))D)$

16.4.2 Cost-based Optimization with Equivalence Rules

- Cost-based optimization
  - search ***all??*** the possible plans and choose *the best* plan
  - a ***optimal*** plan with the minimum costs can be obtained
- Cost-based optimization is expensive

- **Physical equivalence rules** allow logical query plan to be converted to physical query plan specifying what algorithms are used for each operation.

Cost-based Optimization with Equivalence Rules

- Efficient optimizer based on equivalent rules depends on
  - A space efficient representation of expressions which avoids making multiple copies of subexpressions
  - Efficient techniques for detecting duplicate derivations of expressions
  - A form of dynamic programming based on **memoization**, which stores the best plan for a subexpression the first time it is optimized, and reuses in on repeated optimization calls on same subexpression
  - Cost-based pruning techniques that avoid generating all plans
- Pioneered by the Volcano project and implemented in the **SQL Server** optimizer

16.4.3 Heuristic Optimization

- Heuristic optimization
  - uses ***heuristics*** (or ***heuristic rules***) to choose *a better* plan
  - to reduce the number of choices that must be made in a cost-based fashion
  - a ***suboptimal*** plan with lower costs can be obtained

- Principles

  transforms the query-tree by heuristics to reduce cost

  - perform ***selection*** early to reduce the number of tuples
  - perform ***projection*** early to reduces the number of attributes
  - substitute ***Cartesian product*** and ***selection*** with ***join***

Heuristic Optimization (cont.)

- perform ***most restrictive selection*** and ***join*** operations before other similar operations
  - the *most restrictive* operations generate resulting relations with smallest size

- Heuristic optimization used in some versions of **Oracle**

Steps in Heuristic Optimization

- Step1 使用rule1(▶)，将conjunctive selection 分解为多个单独的*选择操作*，以使单个选择操作尽可能沿查询树下移（尽早执行选择操作，以减少中间计算结果）
- Step2 根据*选择操作*的交换率和分配率，利用rule2, rule7.a, rule7.b, rule11， 将查询树上的每个选择操作尽可能移向叶节点，以便尽早执行选择操作
- ***Step3*** 根据*连接操作*的结合律和交换率，使用rule6(▶)，重新安排查询树中的叶结点，使得具有***restrictive selection***特征的叶结点先执行
  - *restrictive selection*：执行此操作后，产生的结果关系最小（所含元组最少）

Steps in Heuristic Optimization (cont.)

- Step4 利用rule4.a(▶), 以*连接操作*代替相邻的选择和笛卡尔乘积操作

- Step5 利用rule3, 8.a, 8.b,12, 将查询树上的*投影操作*尽可能下移，以便尽早执行投影操作，减少中间计算结果

- Step6 将最后的查询树分解为多个子树，使子树中的各操作可以采用*流水线方式*执行，以减少对外设的访问次数
  - *e*.g. Fig.16.2 ▶

重点：前5步！！！

Example One

- Given
  - S(S#, sname, age, sex)
  - C(C#, cname, teacher)
  - SC(S#, C#, grade)
- Find all the ***students***, who are ***male***, and get ***grades*** more than 90 when they learn some one ***course***
- Step1. SQL statement

  -
    ```
    SELECT  DISTINCT   sname
    FROM                S,    SC
    WHERE   S.s# = SC.s#  AND  sex=M  AND  grade > 90
    ```

selection conditions

join condition

Example One (cont.)

- Principles

select *A1, A2, …, An*
from $r_1, r_2, \ldots, r_n$
where *P*

corresponds to

$$\prod_{A1,\ A2,\ \ldots,\ An} (\sigma_P (r_1 \times r_2 \times \ldots \times r_n))$$

Example One (cont.)

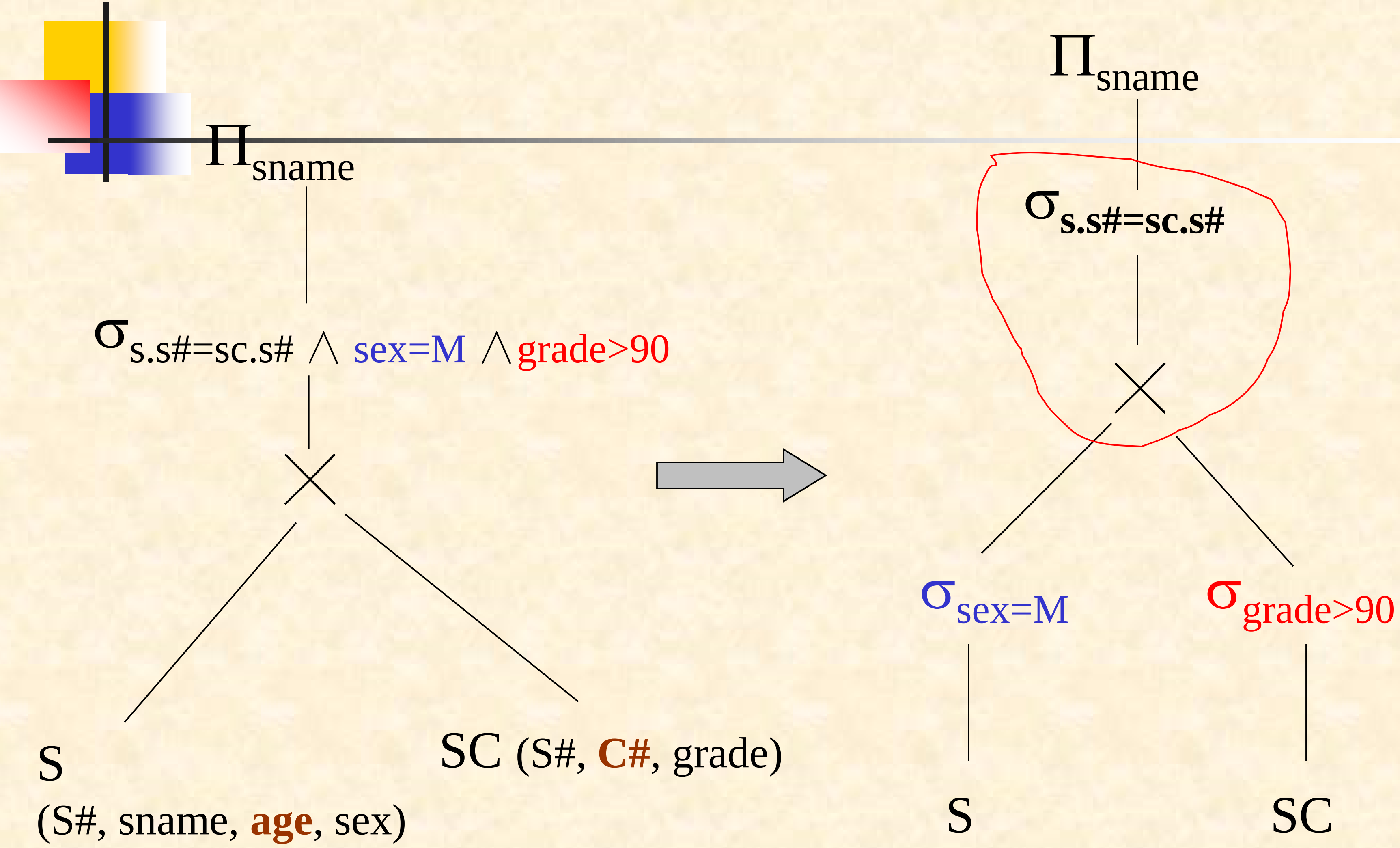
- Step2. Initial Relational algebra expression
  - E1 =

    $\Pi_{sname}(\sigma_{s.s\#=sc.s\# \land sex=M \land grade>90}(S \times SC))$

或：$\Pi_{sname}(\sigma_{sex=M \land grade>90}(S \bowtie SC))$

- Step3. Initial query tree
  - Fig. 16.0.5 (a), E1

- Step4. By means of *Rule1, Rule7b*, distribute σ operations over relation ***S*** and ***SC***
  - Fig. 16.0.5 (b), E2



Π_{sname}

$\sigma_{s.s\#=sc.s\# \land sex=M \land grade>90}$

$\times$

S

(S#, sname, **age**, sex)

SC (S#, **C#**, grade)

Π_{sname}

$\sigma_{s.s\#=sc.s\#}$

$\times$

$\sigma_{sex=M}$

$\sigma_{grade>90}$

S

SC

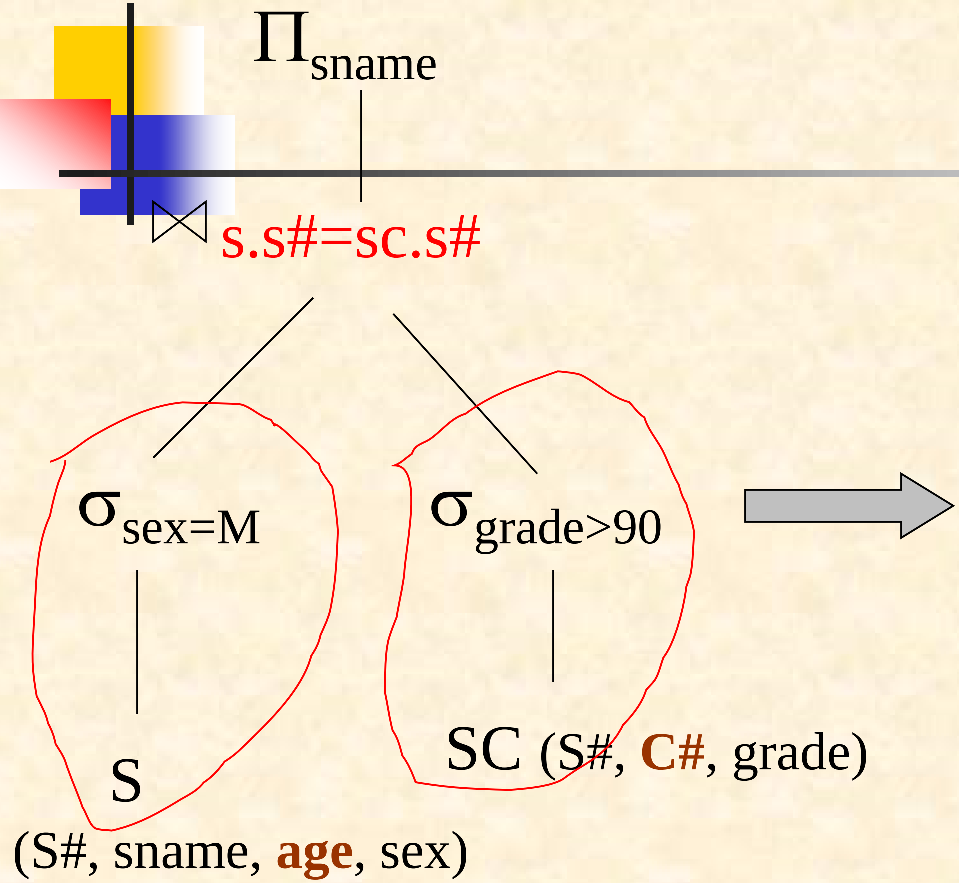
(a) Initial query tree
E1

（b）E2

Example One (cont.)

- Step5. By means of *Rule4.a*, replace *selection* and *Cartesian product* with *natural join*
    - Fig. 16.0.5 (c), E3

- Step6. !! By means of *Rule8b*, distribute Π operations over relation ***S*** and ***SC***
    - Fig. 16.0.5 (d),
- The final optimized expression E4 that is equivalent to initial expression E1 is
    - $\Pi_{\text{sname}} \{\Pi_{\text{sname, S\#}} (\sigma_{\text{sex=M}} (\text{S})) \quad \bowtie$
      $\sigma_{\text{grade>90}} (\Pi \text{grade, S\# (SC)}) \}$



Π_{sname}

$\bowtie$ s.s#=sc.s#

$\sigma_{sex=M}$

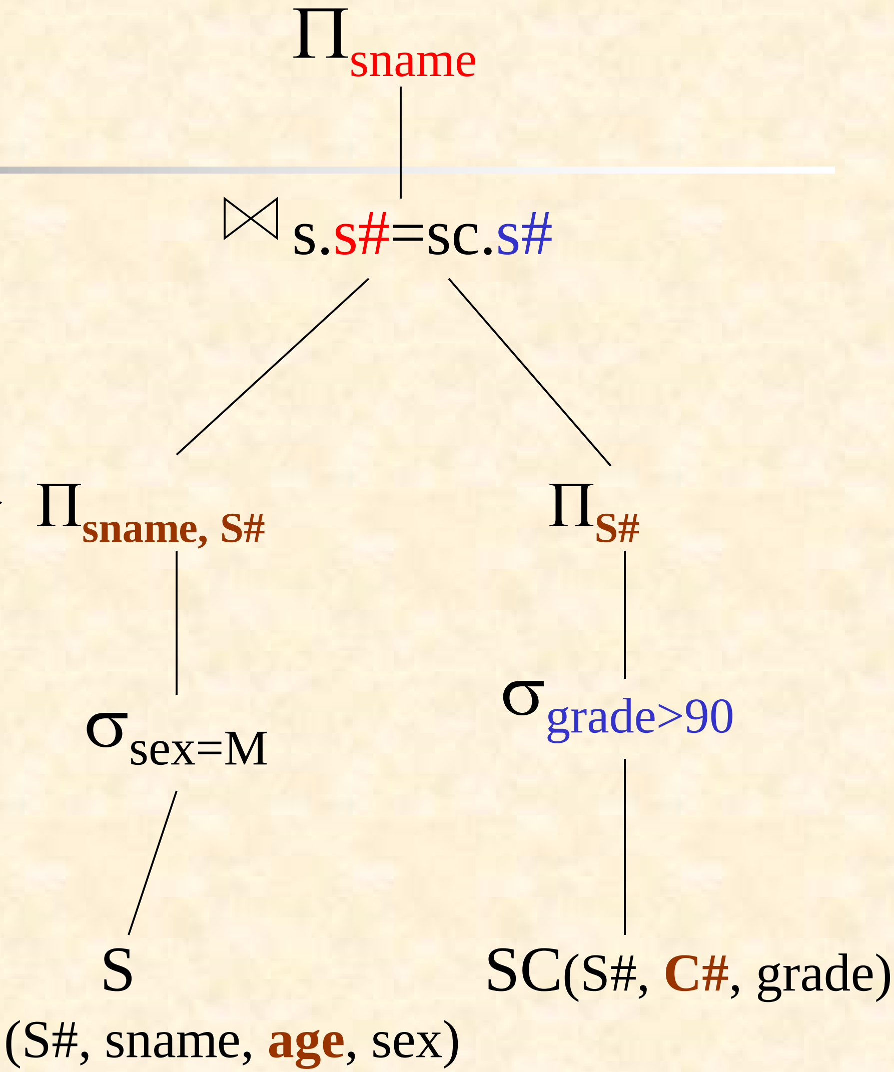
S

(S#, sname, **age**, sex)

$\sigma_{grade>90}$

SC (S#, **C#**, grade)

（c）E3



Π_{sname}

$\bowtie$ s.s#=sc.s#

$\Pi_{\textbf{sname, S\#}}$

$\sigma_{sex=M}$

S

(S#, sname, **age**, sex)

$\Pi_{\textbf{S\#}}$

$\sigma_{grade>90}$

SC(S#, **C#**, grade)

（d）E4

Example Two

- Consider the following insurance database, where the primary keys are underlined,
    - ***Person***(driver-id, name, address)
    - ***Car***(license, model, year)
    - ***Accident***(report-number, date, location)
    - ***Participated***(driver-id , license, report-number, damage-amount)

Example Two (cont.)

- Problems
  - Give a SQL statement to find out the ***driver name***, ***license***, ***report-number*** of the ***accidents*** which happened in *Beijing* and *before May 3, 2008*
  - Give the initial query tree for this query, and construct an optimized and equivalent *relational algebra expression* for it by means of heuristic optimization

Example Two (cont.)

- Step1. SQL statement
  - **Select** *name, license, Accident.report-number*

    From *Person, Participate, Accident*

    Where Person.driver_id = Participate.driver_id **AND**

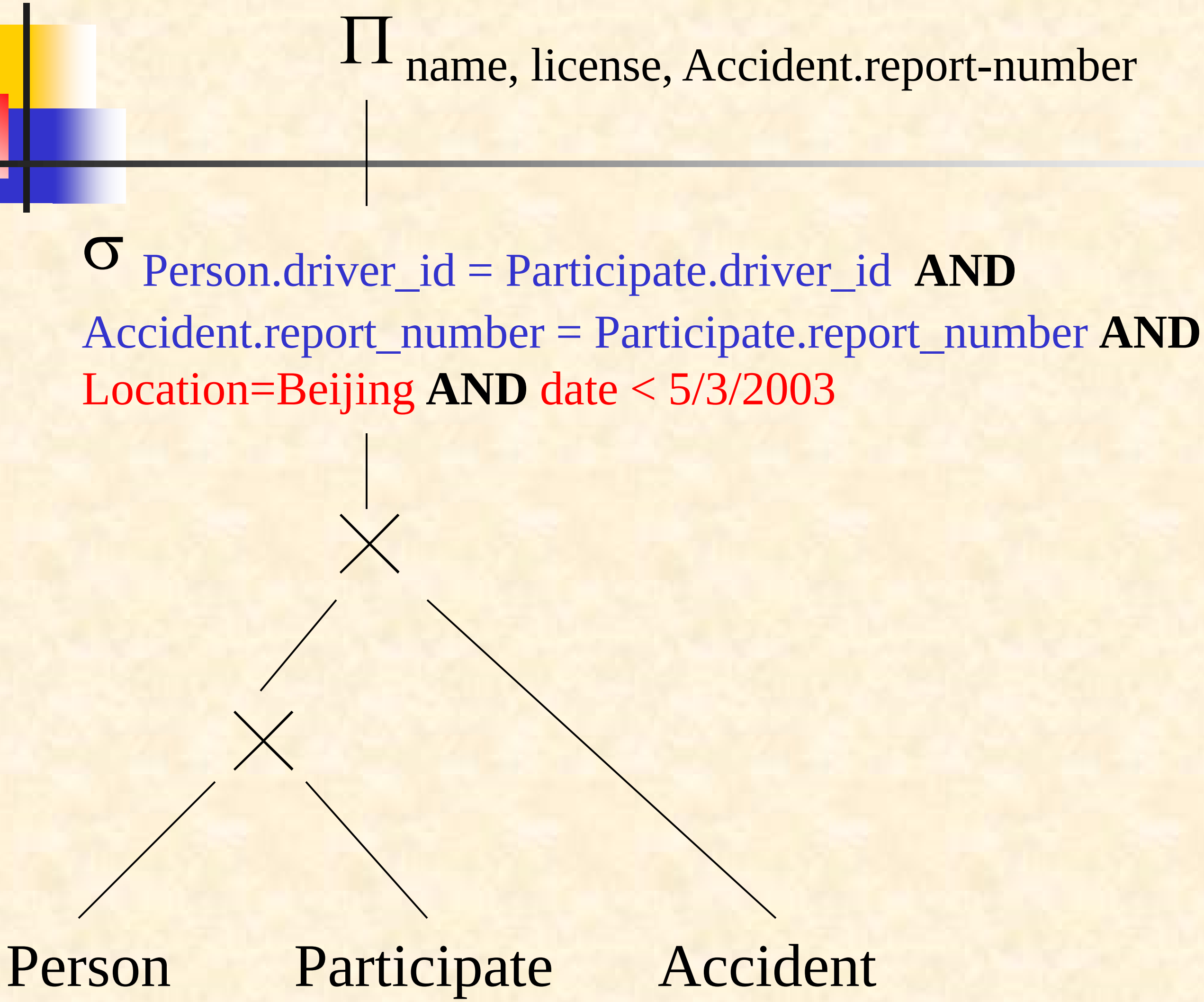
    Accident.report_number = Participate.report_number

    AND Location=Beijing AND date < 5/3/2008
- Step2. Initial expression

$\Pi_{\text{name, license, Accident.report-number}}$

$(\sigma_{\text{Person.driver\_id = Participate.driver\_id AND Accident.report\_number = Participate.report\_number AND Location=Beijing AND date < 5/3/2003}}$

$(\text{Person} \times \text{Participate} \times \text{Accident})$



Π name, license, Accident.report-number

σ Person.driver_id = Participate.driver_id **AND** Accident.report_number = Participate.report_number **AND** Location=Beijing **AND** date < 5/3/2003

×

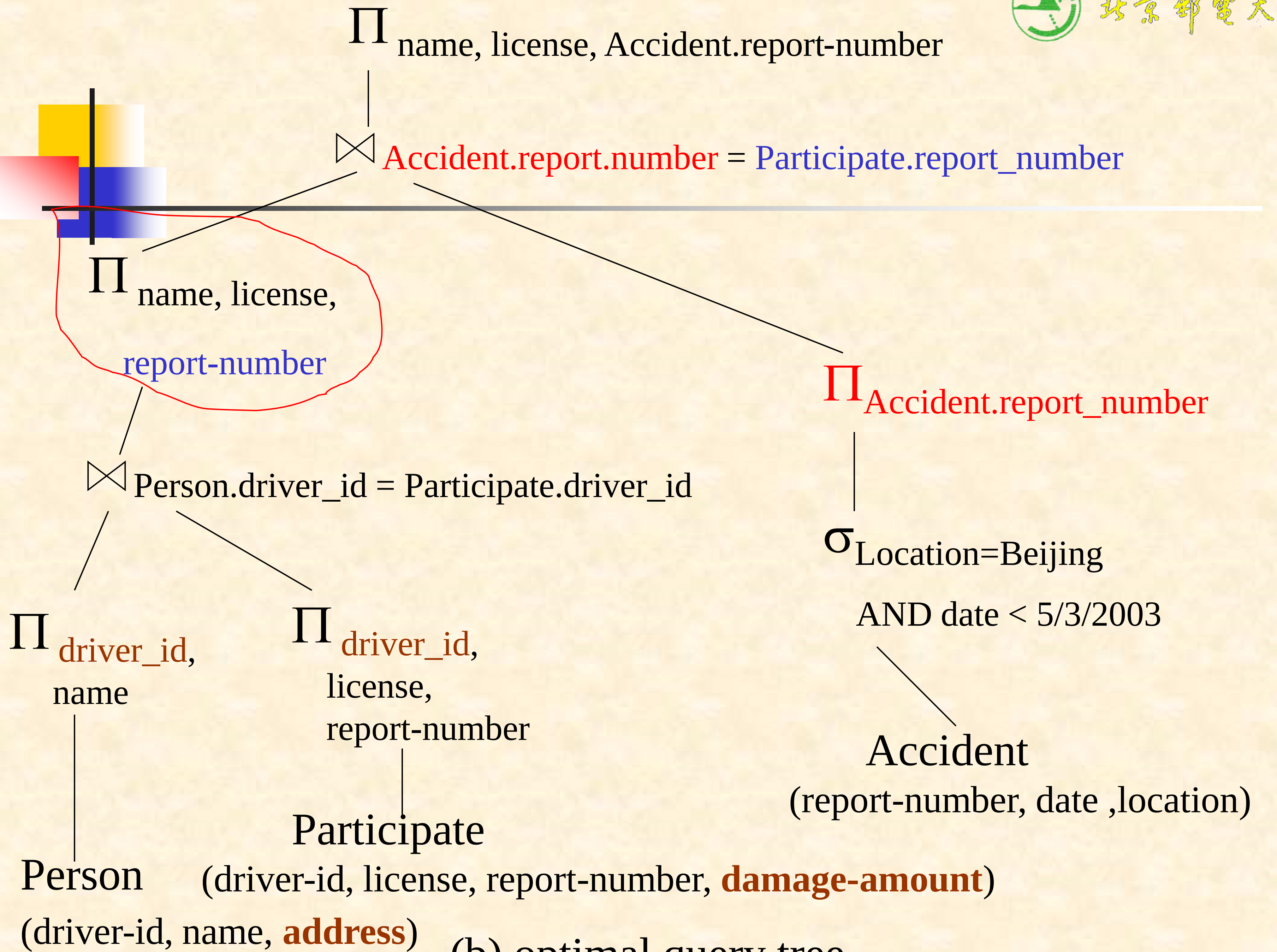
×

Person

Participate

Accident

(a) initial query tree



Π name, license, Accident.report-number

$\bowtie$ Accident.report.number = Participate.report_number

Π name, license, report-number

$\bowtie$ Person.driver_id = Participate.driver_id

Π driver_id, name

Person (driver-id, name, **address**)

Π driver_id, license, report-number

Participate (driver-id, license, report-number, **damage-amount**)

Π Accident.report_number

σ Location=Beijing AND date < 5/3/2003

Accident (report-number, date ,location)

(b) optimal query tree

Example Three

- Consider the following relations in banking enterprise database, where the primary keys are underlined
  - *branch (branch-name, branch-city, assets),*
  - *loan (loan-number, branch-name , amount)*
  - *borrower(customer-name, loan-number, borrow-date)*
  - *customer (customer-name, customer-street, customer-city)*
  - *account (account-number, branch-name, balance)*
  - *depositor (customer-name, account-number , deposit-date)*

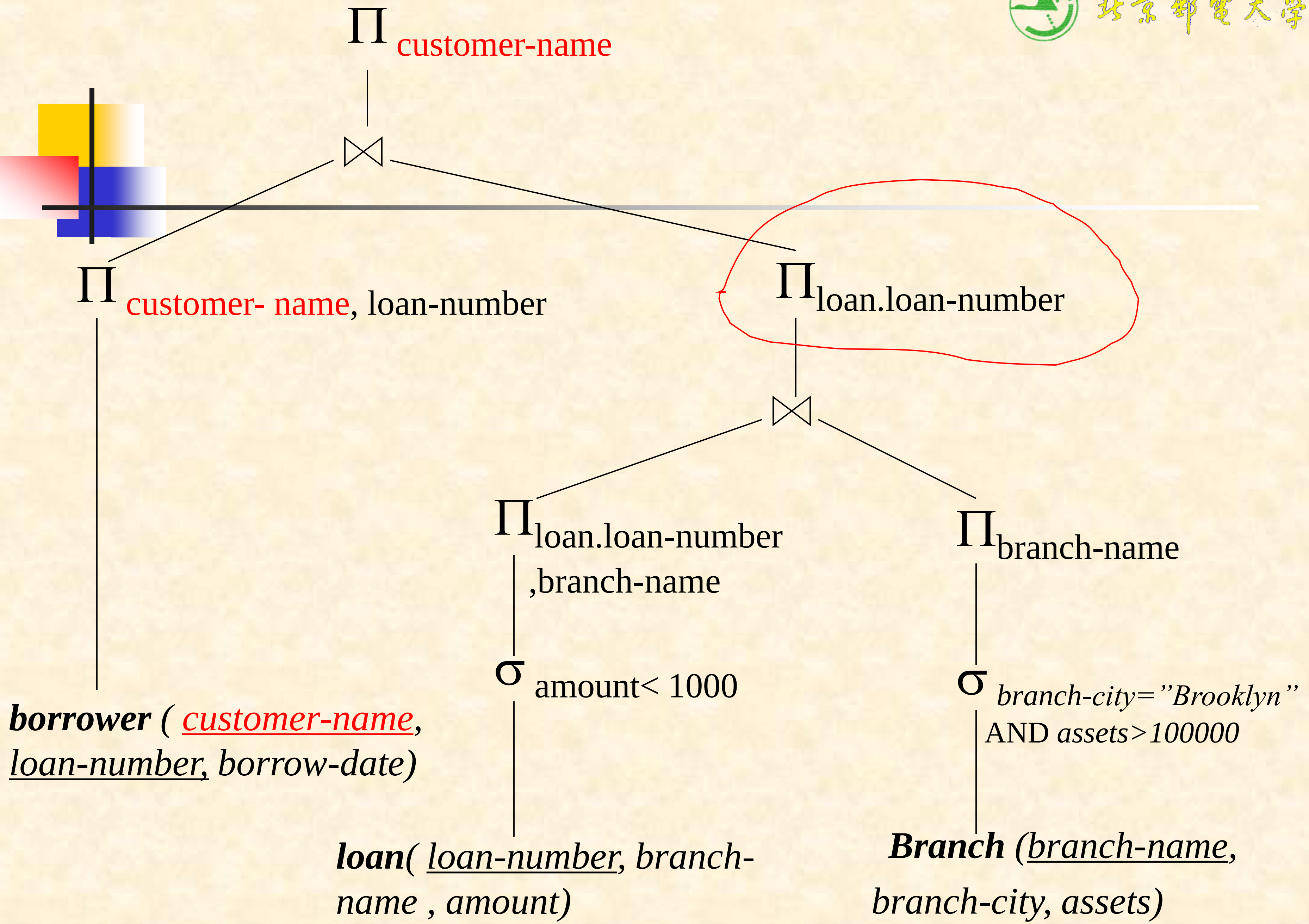
Example Three (cont.)

- For the query “ Find the ***names*** of all ***customers*** who have an ***loan*** at any ***branch*** that is located in ***Brooklyn*** and have ***assets*** more than $100,000, requiring that ***loan-amount*** is less than $1000”
  - give an SQL statement for this query
  - given a initial query tree for the query, and convert it into an optimized query tree by means of heuristic optimization

Example Three (cont.)

- SQL

```
    select customer-name
from  borrower, loan, branch
where loan.loan-number=borrower.loan-number
    and branch.branch-name=loan.branch-name
    and branch-city=”Brooklyn”
    and assets>100000 and amount<1000
```



$\Pi_{\text{customer-name}}$

$\bowtie$

$\Pi_{\text{customer- name, loan-number}}$

$\Pi_{\text{loan.loan-number}}$

$\bowtie$

$\Pi_{\text{loan.loan-number ,branch-name}}$

$\Pi_{\text{branch-name}}$

$\sigma_{\text{amount< 1000}}$

$\sigma_{\textit{branch-city=''Brooklyn''} \text{ AND } \textit{assets>100000}}$

***borrower** (customer-name, loan-number, borrow-date)*

***loan**(loan-number, branch-name , amount)*

***Branch** (branch-name, branch-city, assets)*

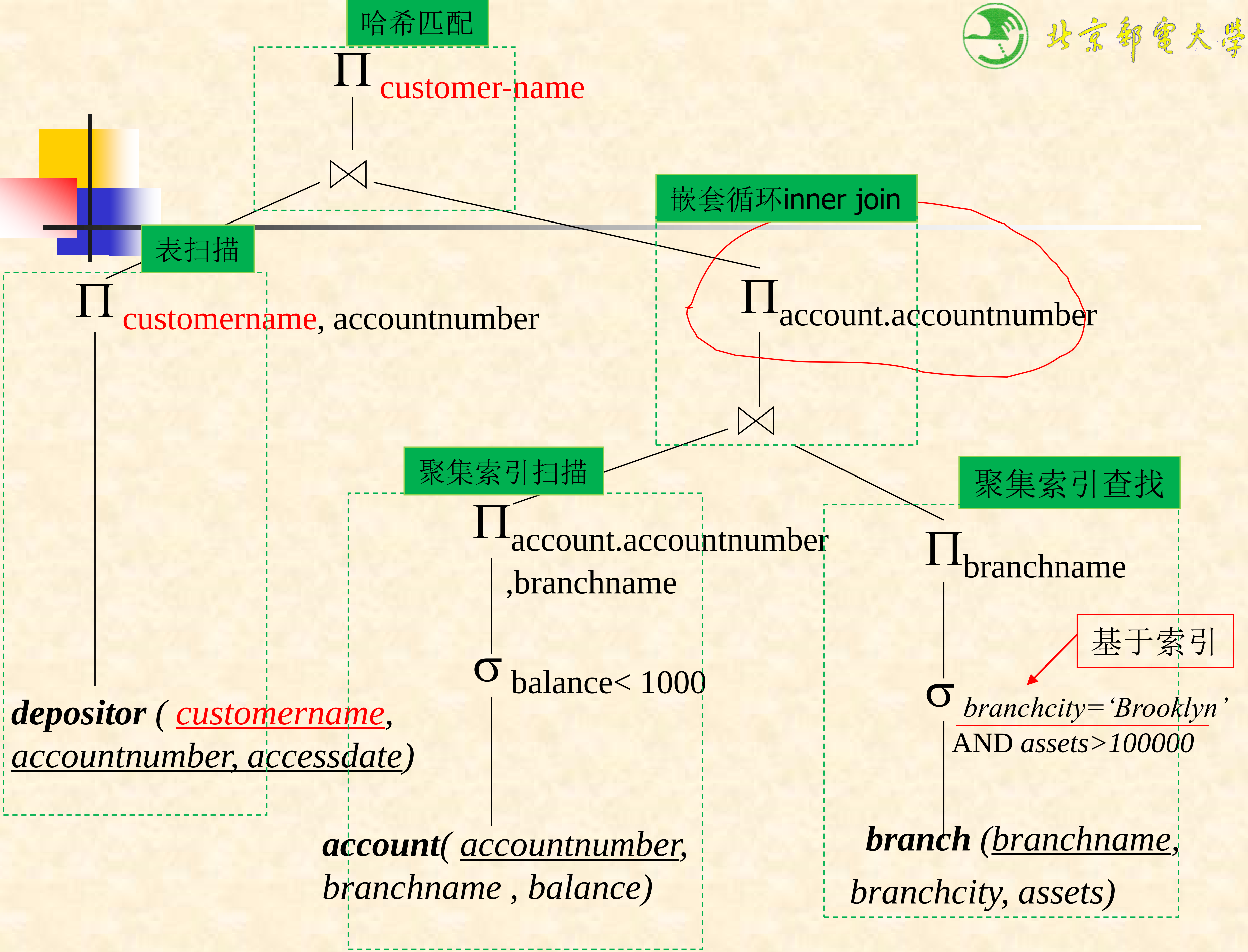
实际DBS中的查询优化

- 教科书上的选择、投影操作下移的实现方式

实际平台下（如SQL Server）与后面的连接操作结合在一起做，避免多次关系代数操作：

1. 采用pipeline，在对参与连接运算的关系扫描过程中完成符合条件的元组—选择

2. 再选出后续步骤需要的属性—投影



哈希匹配

$\Pi_{\text{customer-name}}$

$\bowtie$

嵌套循环inner join

表扫描

$\Pi_{\text{customername, accountnumber}}$

$\Pi_{\text{account.accountnumber}}$

$\bowtie$

聚集索引扫描

聚集索引查找

$\Pi_{\text{account.accountnumber ,branchname}}$

$\sigma_{\text{balance}< 1000}$

$\Pi_{\text{branchname}}$

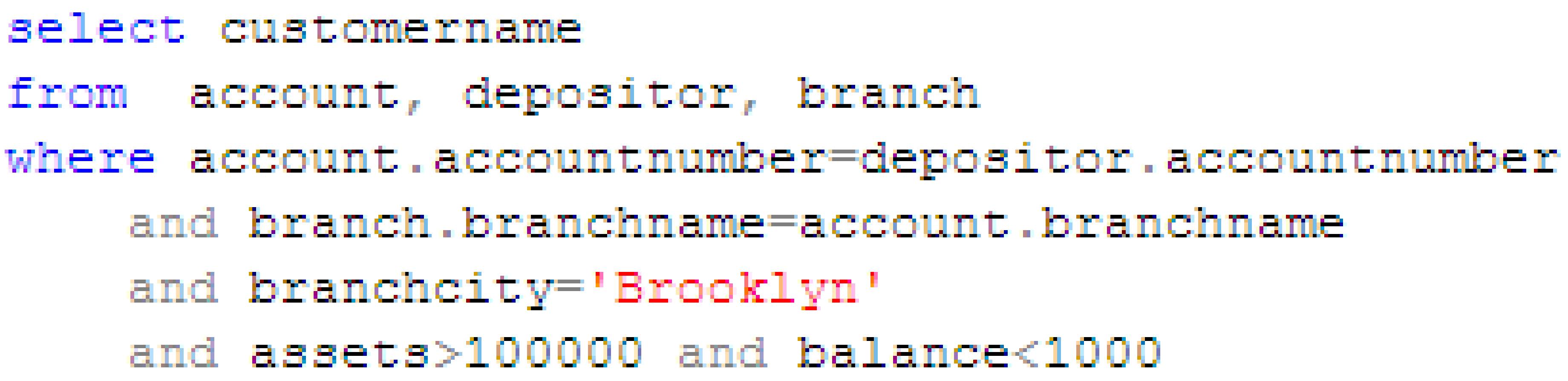
基于索引

σ *branchcity=‘Brooklyn’* AND *assets>100000*

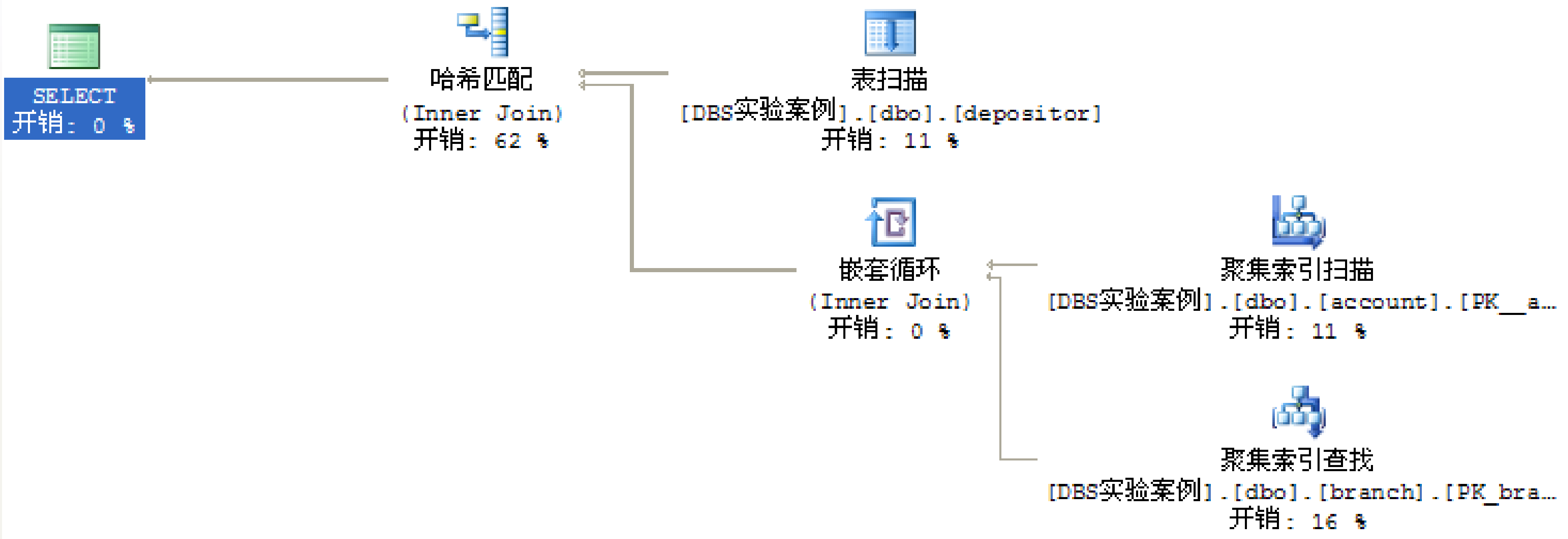
depositor (*customername, accountnumber, accessdate*)

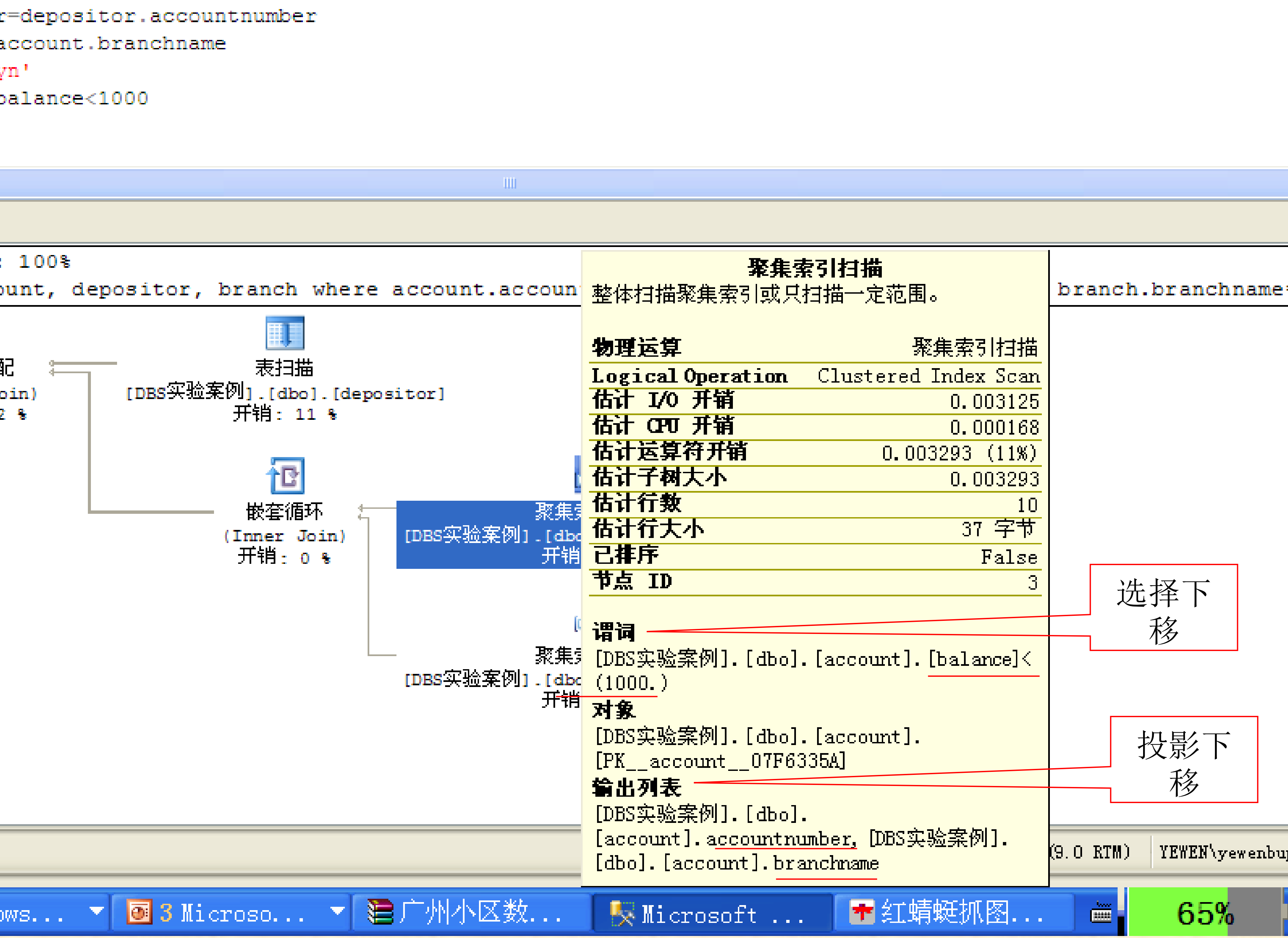
account(*accountnumber, branchname , balance)*

branch (*branchname, branchcity, assets)*



```
select customername
from  account, depositor, branch
where account.accountnumber=depositor.accountnumber
      and branch.branchname=account.branchname
      and branchcity='Brooklyn'
      and assets>100000 and balance<1000
```





```
r=depositor.accountnumber
account.branchname
yn'
balance<1000
```

聚集索引扫描

整体扫描聚集索引或只扫描一定范围。

物理运算	聚集索引扫描
Logical Operation	Clustered Index Scan
估计 I/O 开销	0.003125
估计 CPU 开销	0.000168
估计运算符开销	0.003293 (11%)
估计子树大小	0.003293
估计行数	10
估计行大小	37 字节
已排序	False
节点 ID	3

谓词

[DBS实验案例].[dbo].[account].[balance]<(1000.)

对象

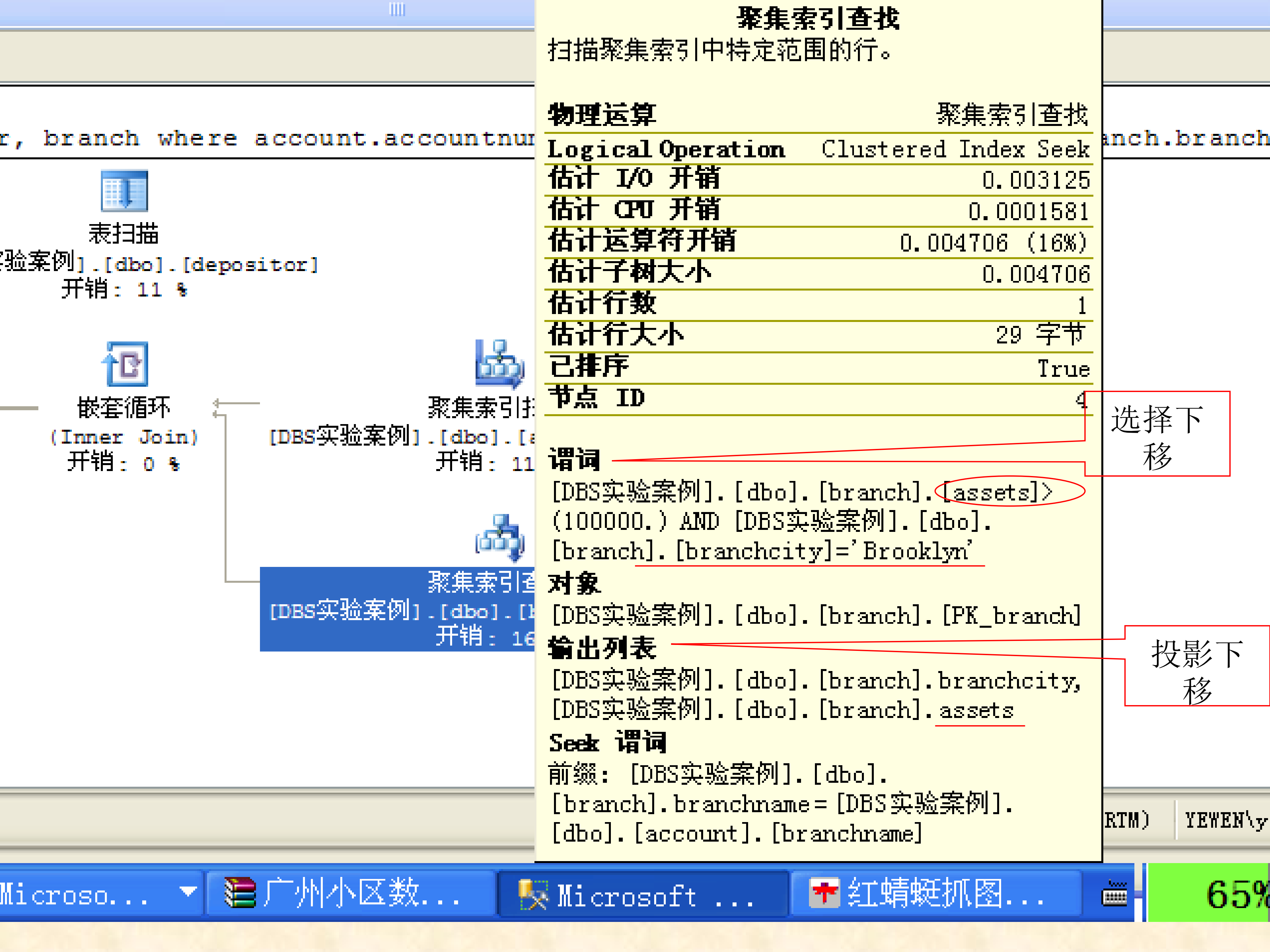
[DBS实验案例].[dbo].[account].[PK__account__07F6335A]

输出列表

[DBS实验案例].[dbo].[account].accountnumber, [DBS实验案例].[dbo].[account].branchname

选择下移

投影下移



聚集索引查找

扫描聚集索引中特定范围的行。

物理运算	聚集索引查找
Logical Operation	Clustered Index Seek
估计 I/O 开销	0.003125
估计 CPU 开销	0.0001581
估计运算符开销	0.004706 (16%)
估计子树大小	0.004706
估计行数	1
估计行大小	29 字节
已排序	True
节点 ID	4

谓词

[DBS实验案例].[dbo].[branch].[assets]>(100000.) AND [DBS实验案例].[dbo].[branch].[branchcity]='Brooklyn'

对象

[DBS实验案例].[dbo].[branch].[PK_branch]

输出列表

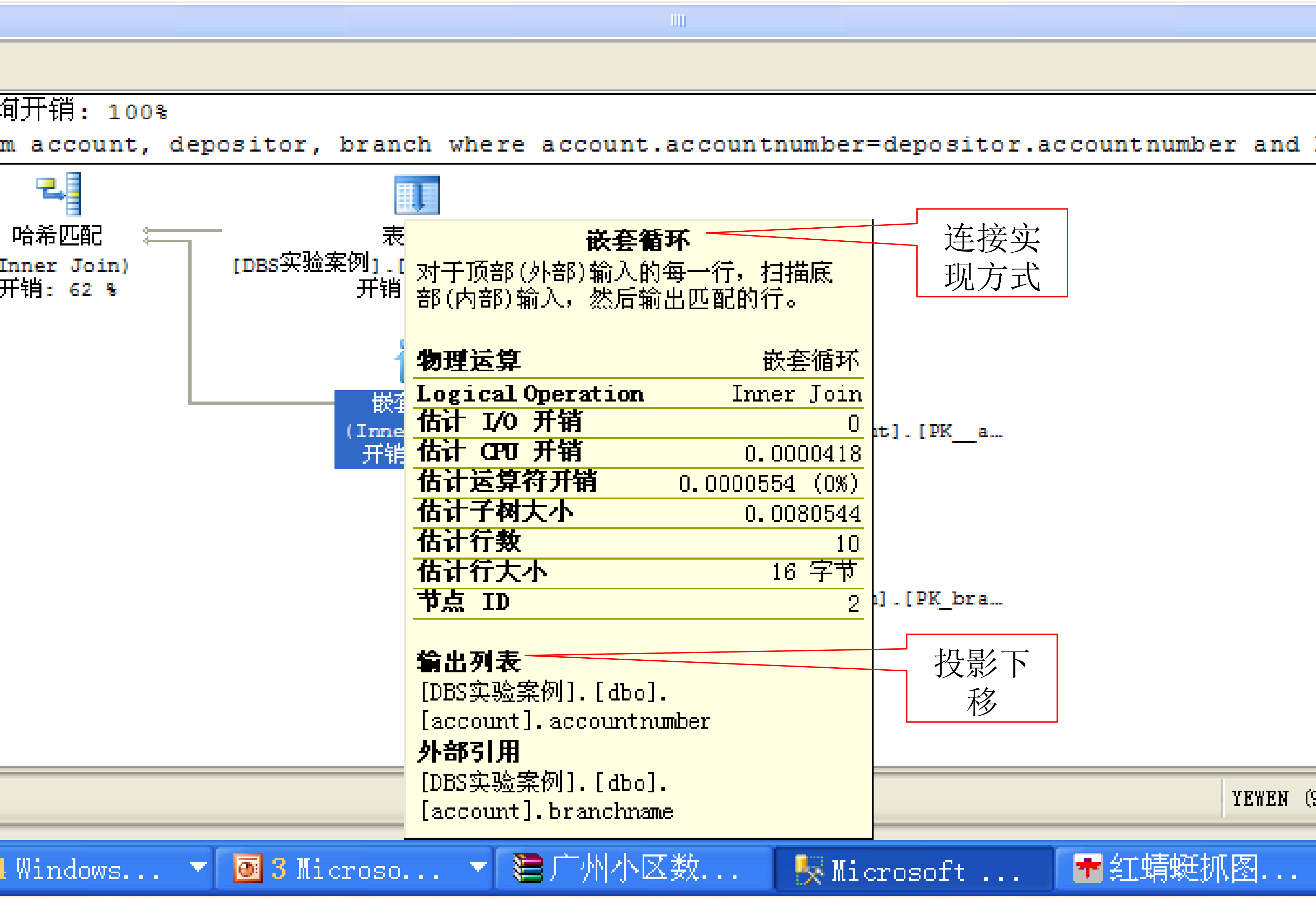
[DBS实验案例].[dbo].[branch].branchcity, [DBS实验案例].[dbo].[branch].assets

Seek 谓词

前缀: [DBS实验案例].[dbo].[branch].branchname=[DBS实验案例].[dbo].[account].[branchname]

选择下移

投影下移



询开销： 100%

m account, depositor, branch where account.accountnumber=depositor.accountnumber and

嵌套循环

对于顶部(外部)输入的每一行，扫描底部(内部)输入，然后输出匹配的行。

物理运算	嵌套循环
Logical Operation	Inner Join
估计 I/O 开销	0
估计 CPU 开销	0.0000418
估计运算符开销	0.0000554 (0%)
估计子树大小	0.0080544
估计行数	10
估计行大小	16 字节
节点 ID	2

输出列表

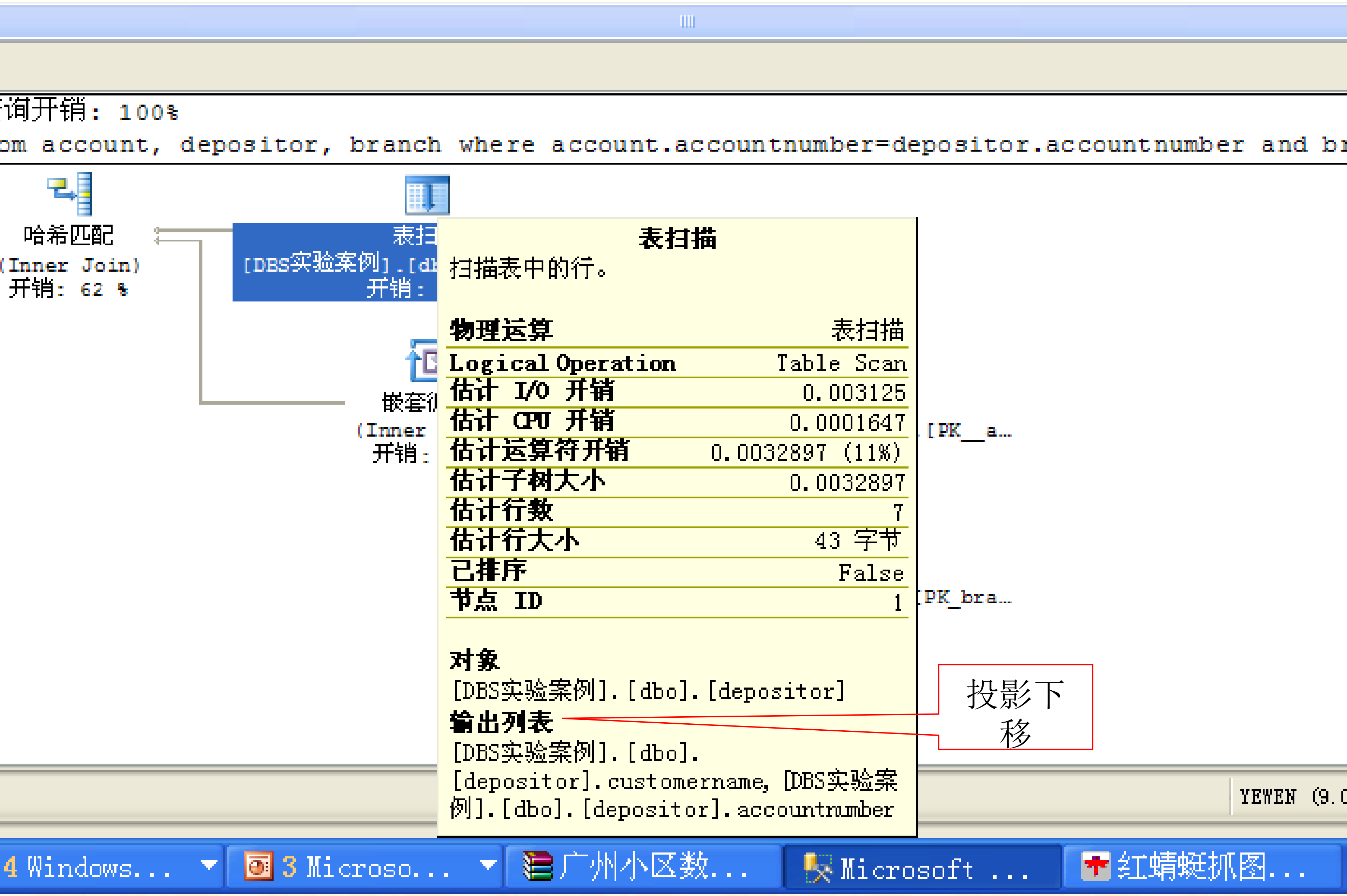
[DBS实验案例].[dbo].
[account].accountnumber

外部引用

[DBS实验案例].[dbo].
[account].branchname

连接实现方式

投影下移



询开销：100%

```
om account, depositor, branch where account.accountnumber=depositor.accountnumber and br
```

表扫描

扫描表中的行。

物理运算	表扫描
Logical Operation	Table Scan
估计 I/O 开销	0.003125
估计 CPU 开销	0.0001647
估计运算符开销	0.0032897 (11%)
估计子树大小	0.0032897
估计行数	7
估计行大小	43 字节
已排序	False
节点 ID	1

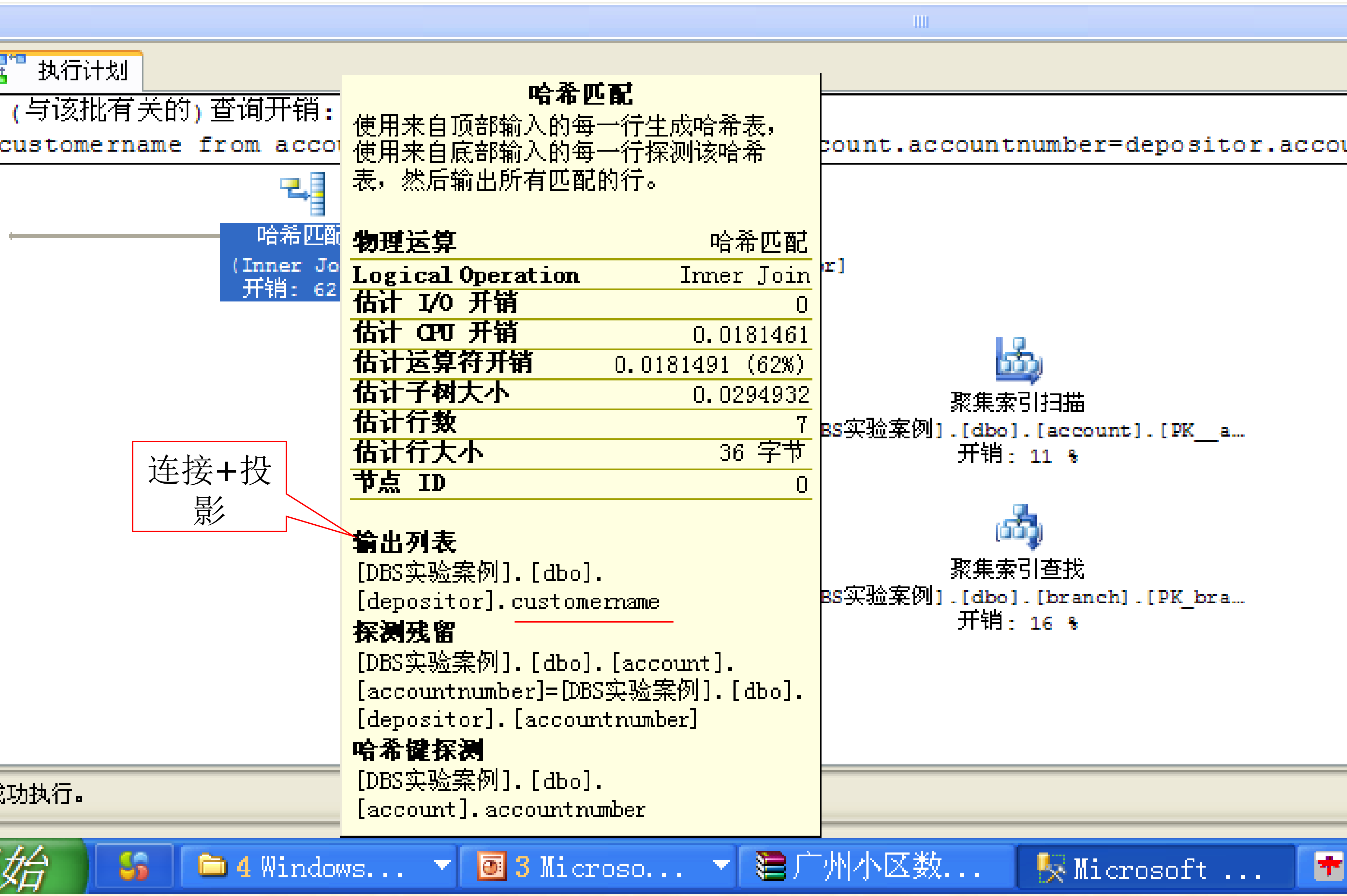
对象

[DBS实验案例].[dbo].[depositor]

输出列表

[DBS实验案例].[dbo].[depositor].customername, [DBS实验案例].[dbo].[depositor].accountnumber

投影下移



哈希匹配

使用来自顶部输入的每一行生成哈希表，使用来自底部输入的每一行探测该哈希表，然后输出所有匹配的行。

物理运算	哈希匹配
Logical Operation	Inner Join
估计 I/O 开销	0
估计 CPU 开销	0.0181461
估计运算符开销	0.0181491 (62%)
估计子树大小	0.0294932
估计行数	7
估计行大小	36 字节
节点 ID	0

输出列表

[DBS实验案例].[dbo].[depositor].customername

探测残留

[DBS实验案例].[dbo].[account].[accountnumber]=[DBS实验案例].[dbo].[depositor].[accountnumber]

哈希键探测

[DBS实验案例].[dbo].[account].accountnumber

连接+投影

Example Four (cont.)

给定如下关系数据库

Employee(Employee#, Name, Address, Super-E#, E-D#)

Department(D#, Dname, manager#, depart-location)

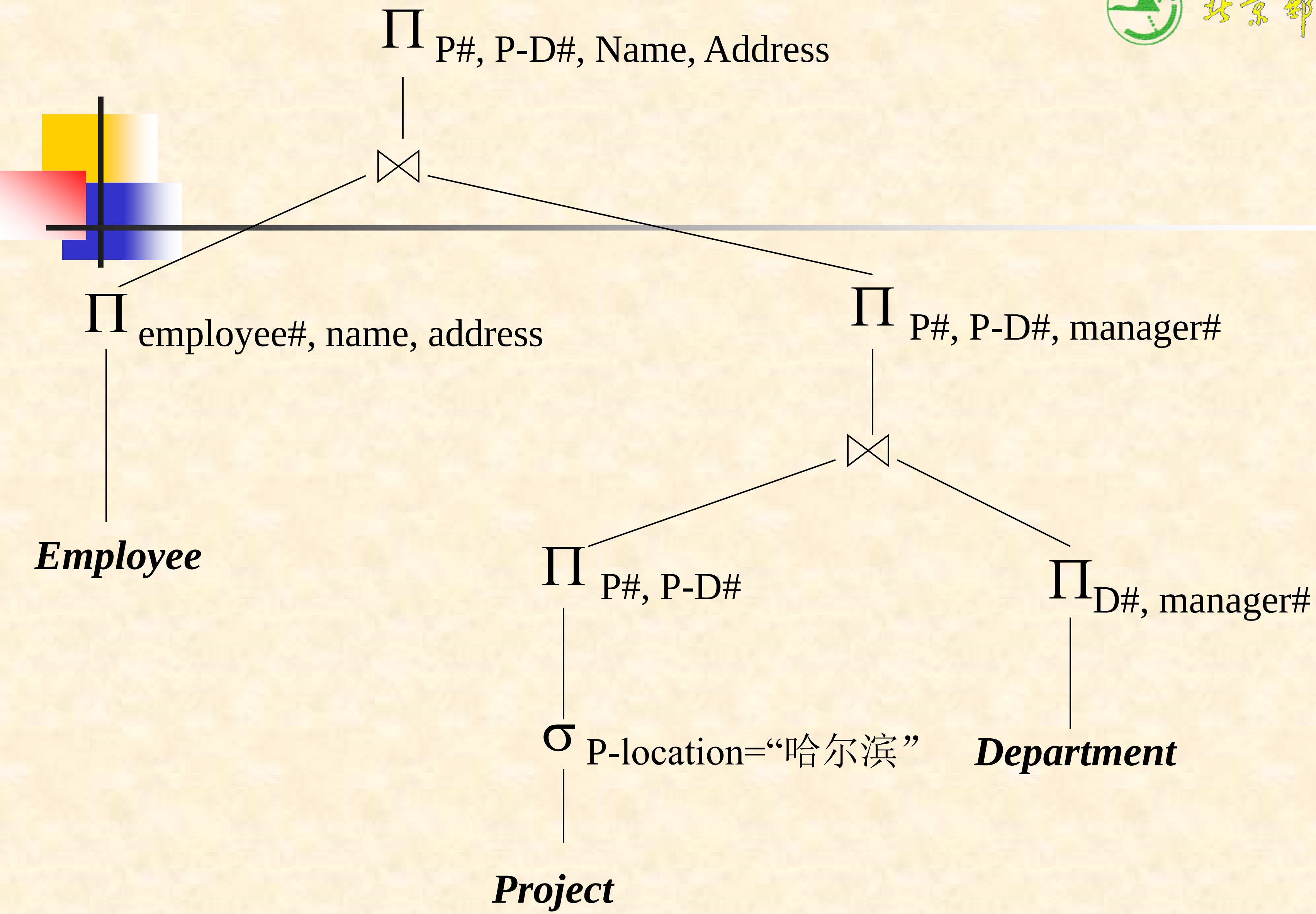
Project(P#, Pname, Plocation, P-D#)

- 查询：对于每个在“哈尔滨”进行的工程项目，列出其工程项目号P#， 工程所属部门号P-D#, 该部门领导的名字Name和地址Address
- 要求：写出该查询的SQL语句；转换为关系代数表达式并利用启发式方法进行查询优化；给出优化后的关系代数表达式。
- Select   P#, P-D#, Name, Address

  From   Project, Department, Employee

  Where   Plocation =哈尔滨 AND P-D#= D#

            AND manager# = Employee#



$\Pi_{P\#,\ P\text{-}D\#,\ Name,\ Address}$

$\bowtie$

$\Pi_{employee\#,\ name,\ address}$

$\Pi_{P\#,\ P\text{-}D\#,\ manager\#}$

Employee

$\bowtie$

$\Pi_{P\#,\ P\text{-}D\#}$

$\Pi_{D\#,\ manager\#}$

σ P-location=“哈尔滨”

Department

Project

作业1

- Consider the following schema, where the primary keys are underlined ,
- Suppliers (supplier-id, supplier-name, city, telephone, address)
- Parts (parts-id, parts-name, parts-color)
- Catalog (supplier-id, parts-id, price)
- .
- (1) Give an SQL statement to find out the *name* and *telephone* of the suppliers who supply a red *part* whose *price* is below $2000.
- (2) Translate this SQL statement into an initial query tree, and give an optimized query *tree* for it, by means of heuristic query optimization.

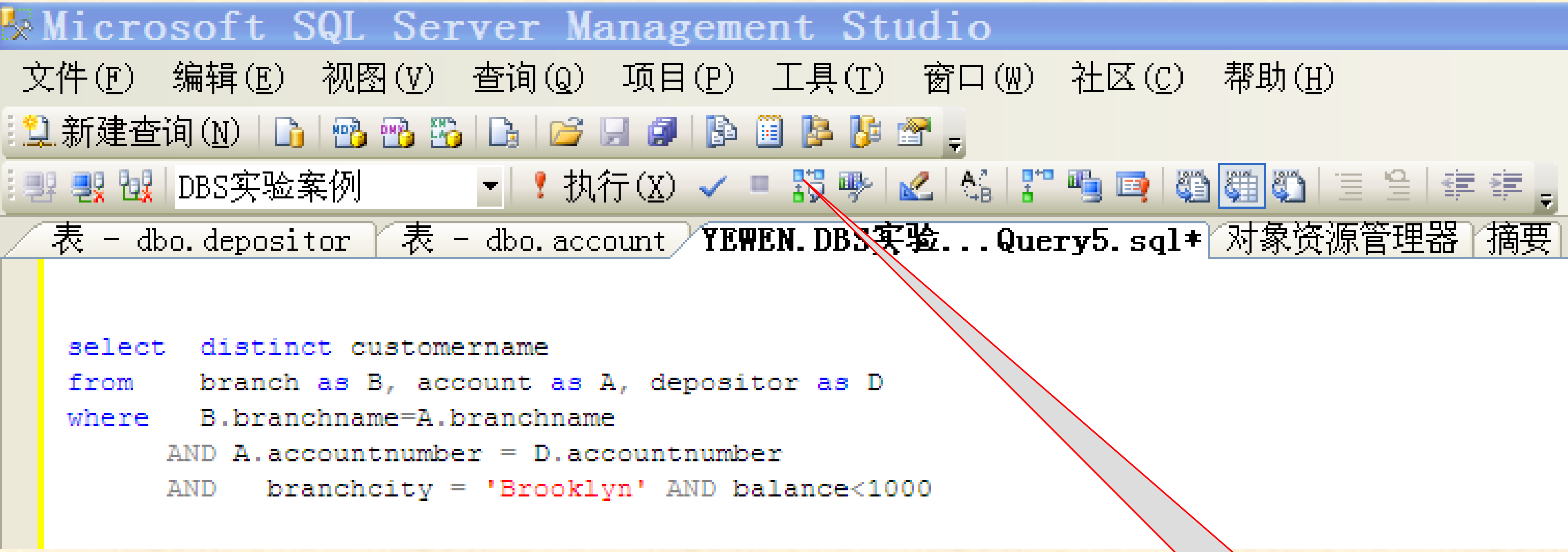
作业2

- Consider the database University given in the textbook,

- (1) Give a SQL statement to find some students and list their names and departments that they belong to. It is required that their total credits (presented by tot_cred) are more than 40, their departments are located in Building 3, and they take the course identified by ‘2016CSDBS’.

- (2) For the SQL statement in (1), give an optimized query tree.

Appendix A SQL Server平台下观察比较SQL语句查询执行计划


Microsoft SQL Server Management Studio
文件(F) 编辑(E) 视图(V) 查询(Q) 项目(P) 工具(T) 窗口(W) 社区(C) 帮助(H)
新建查询(N)
DBS实验案例
执行(X)
表 - dbo.depositor
表 - dbo.account
YEWEN.DBS实验...Query5.sql*
对象资源管理器
摘要

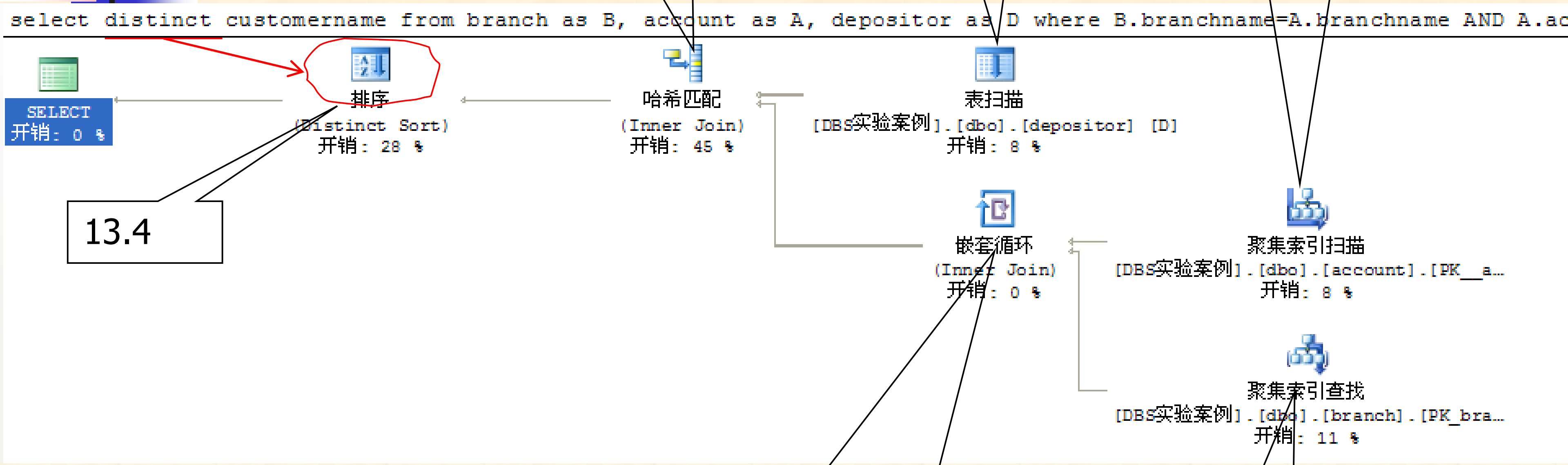
```
select   distinct customername
from     branch as B, account as A, depositor as D
where    B.branchname=A.branchname
      AND A.accountnumber = D.accountnumber
      AND   branchcity = 'Brooklyn' AND balance<1000
```

显示估计的查询执行计划

16.5.5 hash join，
Fig. 16.8

A1算法；
outer relation

A1算法；
outer relation

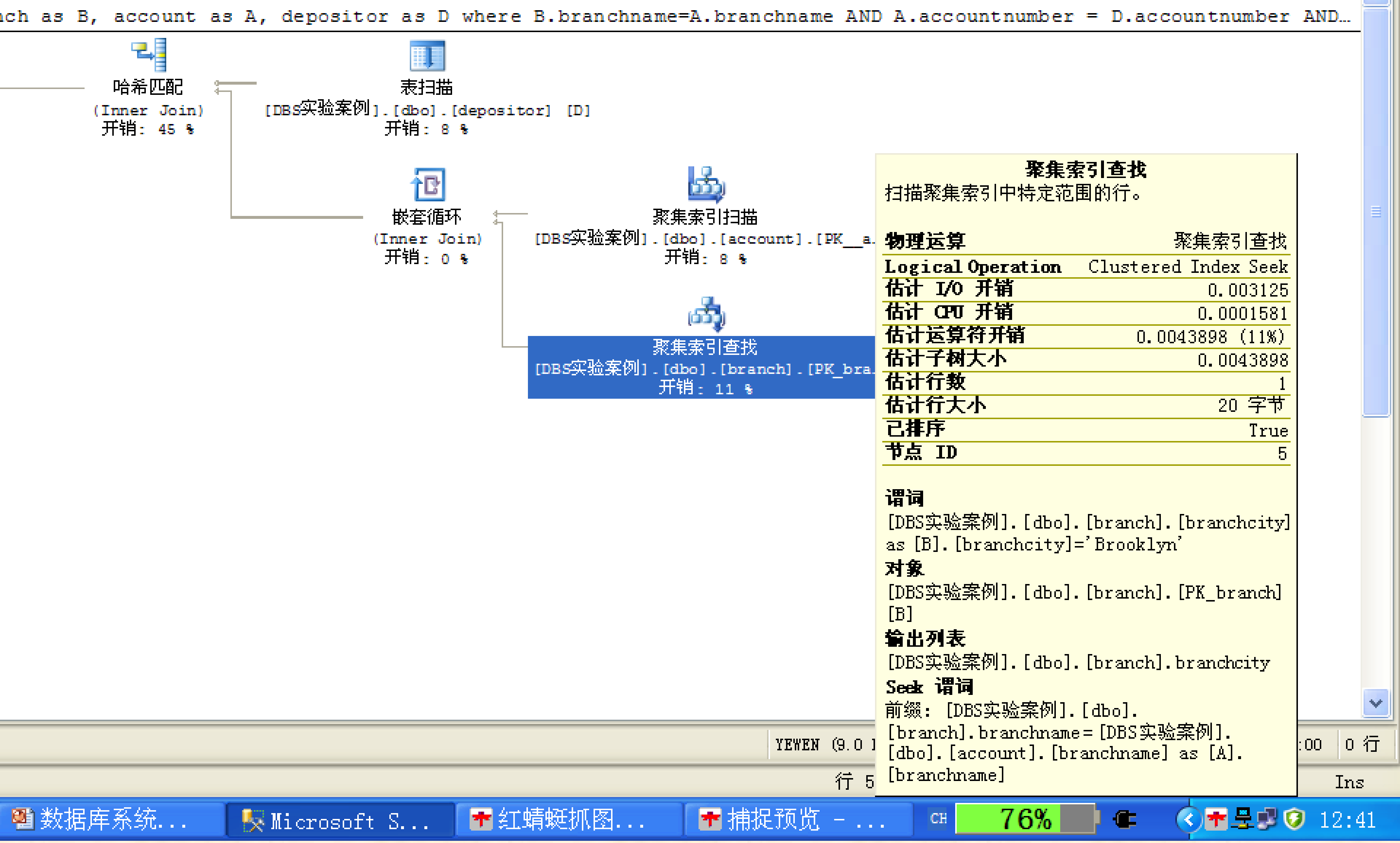


```
select distinct customername from branch as B, account as A, depositor as D where B.branchname=A.branchname AND A.ac
```

13.4

16.5.2 indexed nested loop join，
在内关系branch的连接属性branchname上建有索引

16.3.2 A3算法；
inner relation, **holded in *main memory***



```
ch as B, account as A, depositor as D where B.branchname=A.branchname AND A.accountnumber = D.accountnumber AND…
```

聚集索引查找

扫描聚集索引中特定范围的行。

物理运算	聚集索引查找
Logical Operation	Clustered Index Seek
估计 I/O 开销	0.003125
估计 CPU 开销	0.0001581
估计运算符开销	0.0043898 (11%)
估计子树大小	0.0043898
估计行数	1
估计行大小	20 字节
已排序	True
节点 ID	5

谓词

[DBS实验案例].[dbo].[branch].[branchcity] as [B].[branchcity]='Brooklyn'

对象

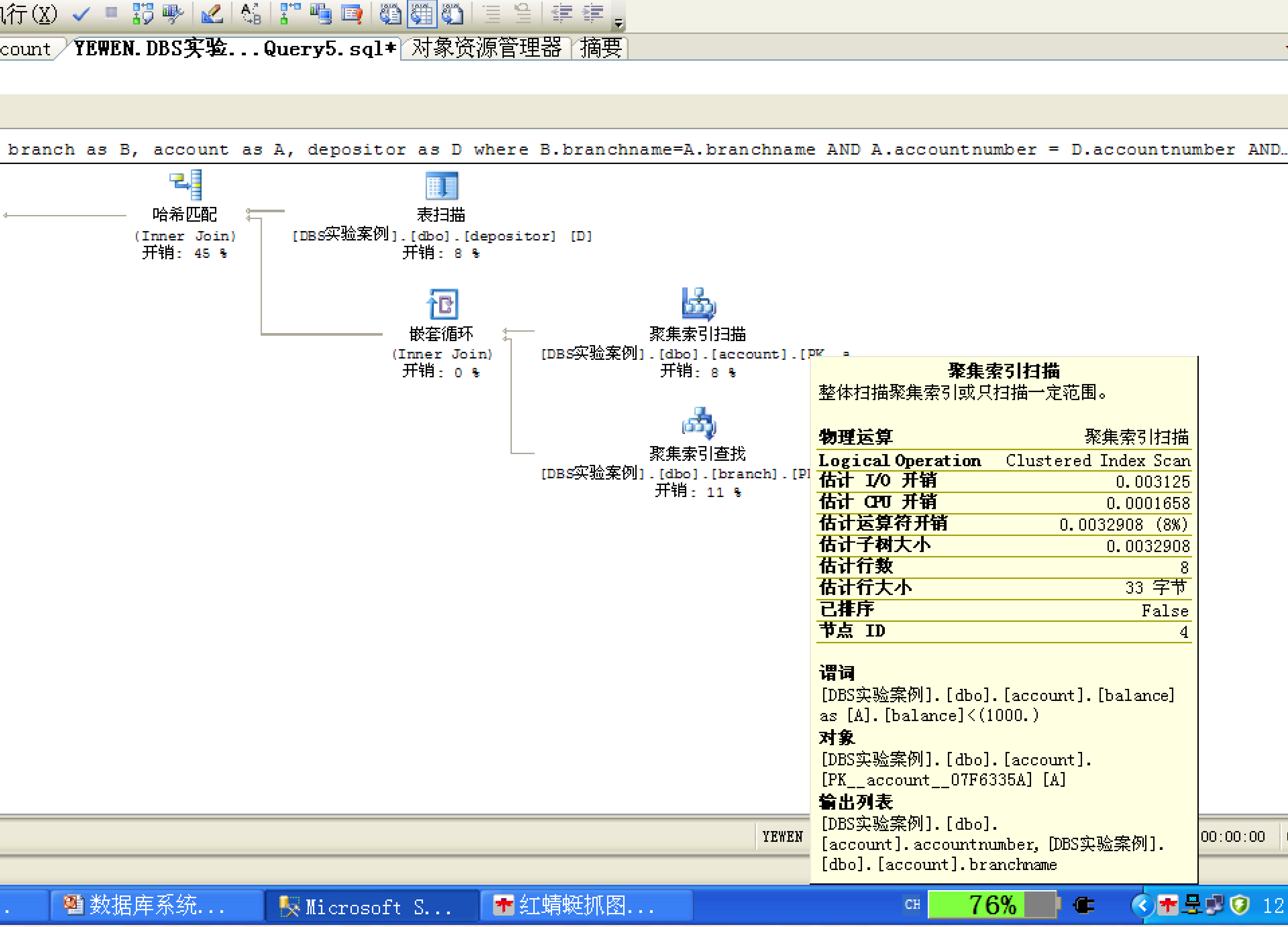
[DBS实验案例].[dbo].[branch].[PK_branch] [B]

输出列表

[DBS实验案例].[dbo].[branch].branchcity

Seek 谓词

前缀: [DBS实验案例].[dbo].[branch].branchname = [DBS实验案例].[dbo].[account].[branchname] as [A].[branchname]



```
branch as B, account as A, depositor as D where B.branchname=A.branchname AND A.accountnumber = D.accountnumber AND
```

聚集索引扫描

整体扫描聚集索引或只扫描一定范围。

物理运算	聚集索引扫描
Logical Operation	Clustered Index Scan
估计 I/O 开销	0.003125
估计 CPU 开销	0.0001658
估计运算符开销	0.0032908 (8%)
估计子树大小	0.0032908
估计行数	8
估计行大小	33 字节
已排序	False
节点 ID	4

谓词

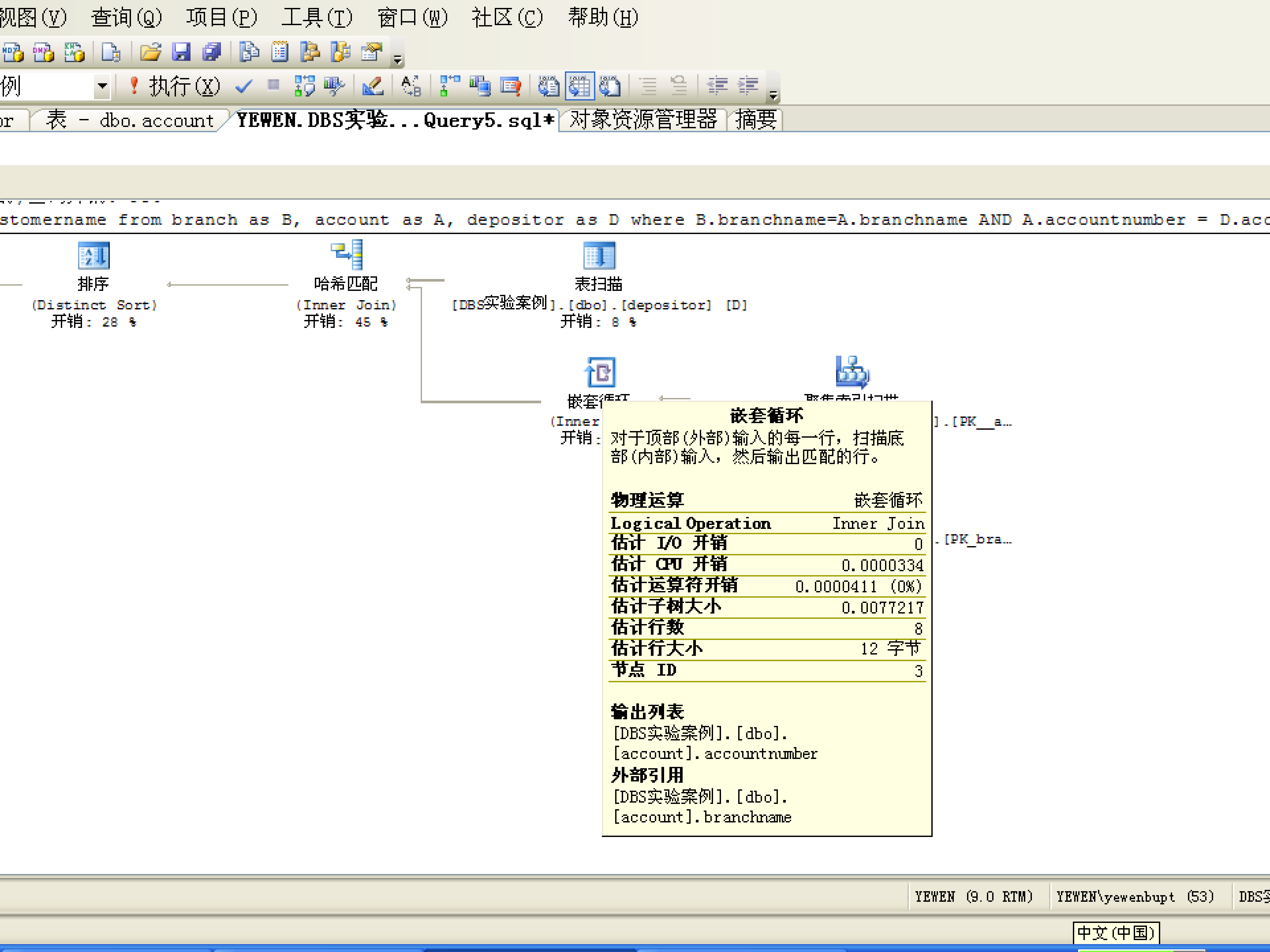
[DBS实验案例].[dbo].[account].[balance] as [A].[balance]<(1000.)

对象

[DBS实验案例].[dbo].[account].[PK__account__07F6335A] [A]

输出列表

[DBS实验案例].[dbo].[account].accountnumber, [DBS实验案例].[dbo].[account].branchname



```
stomername from branch as B, account as A, depositor as D where B.branchname=A.branchname AND A.accountnumber = D.acc
```

嵌套循环

对于顶部(外部)输入的每一行，扫描底部(内部)输入，然后输出匹配的行。

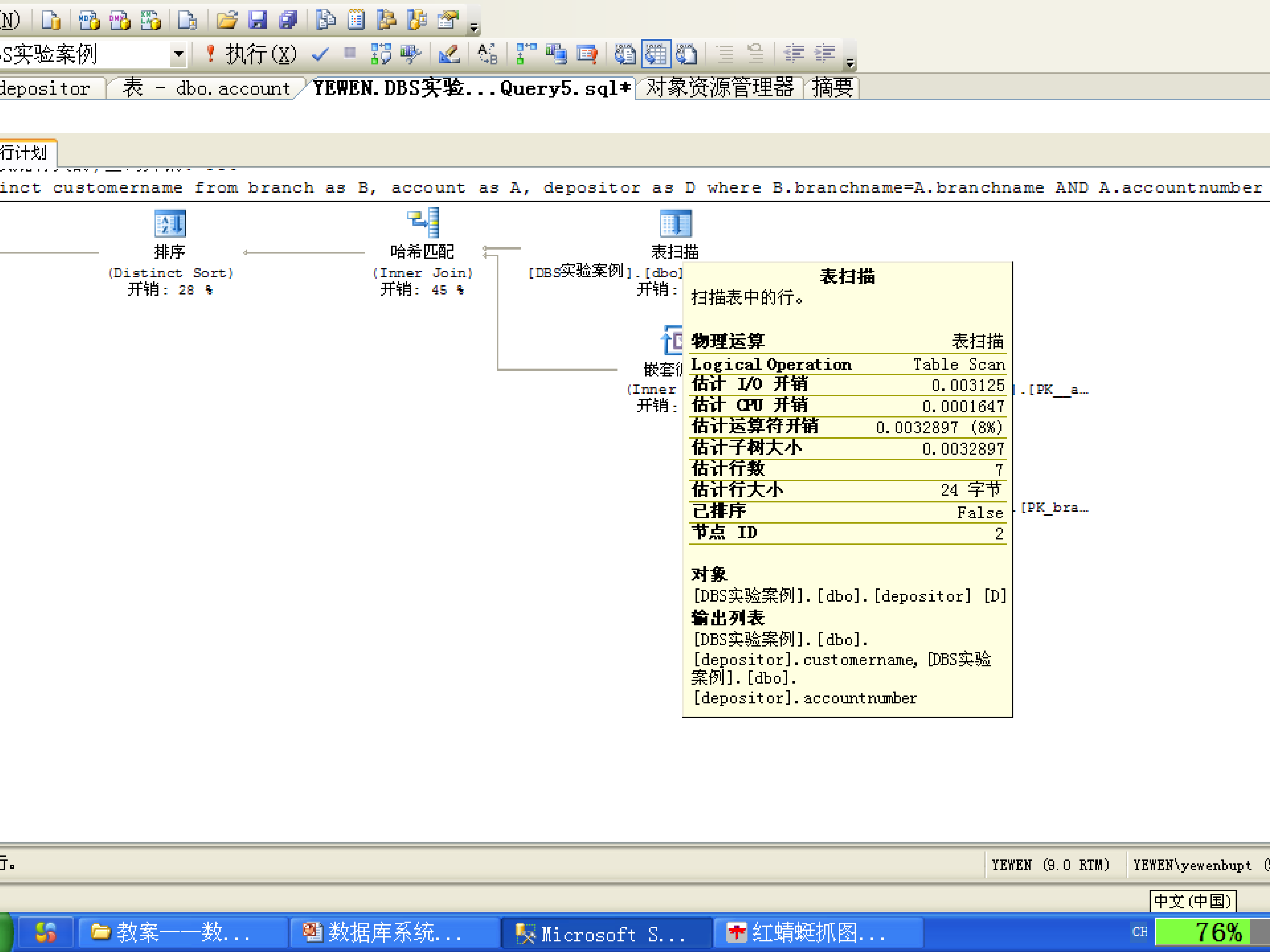
物理运算	嵌套循环
Logical Operation	Inner Join
估计 I/O 开销	0
估计 CPU 开销	0.0000334
估计运算符开销	0.0000411 (0%)
估计子树大小	0.0077217
估计行数	8
估计行大小	12 字节
节点 ID	3

输出列表

[DBS实验案例].[dbo].[account].accountnumber

外部引用

[DBS实验案例].[dbo].[account].branchname



```
inct customername from branch as B, account as A, depositor as D where B.branchname=A.branchname AND A.accountnumber
```

表扫描

扫描表中的行。

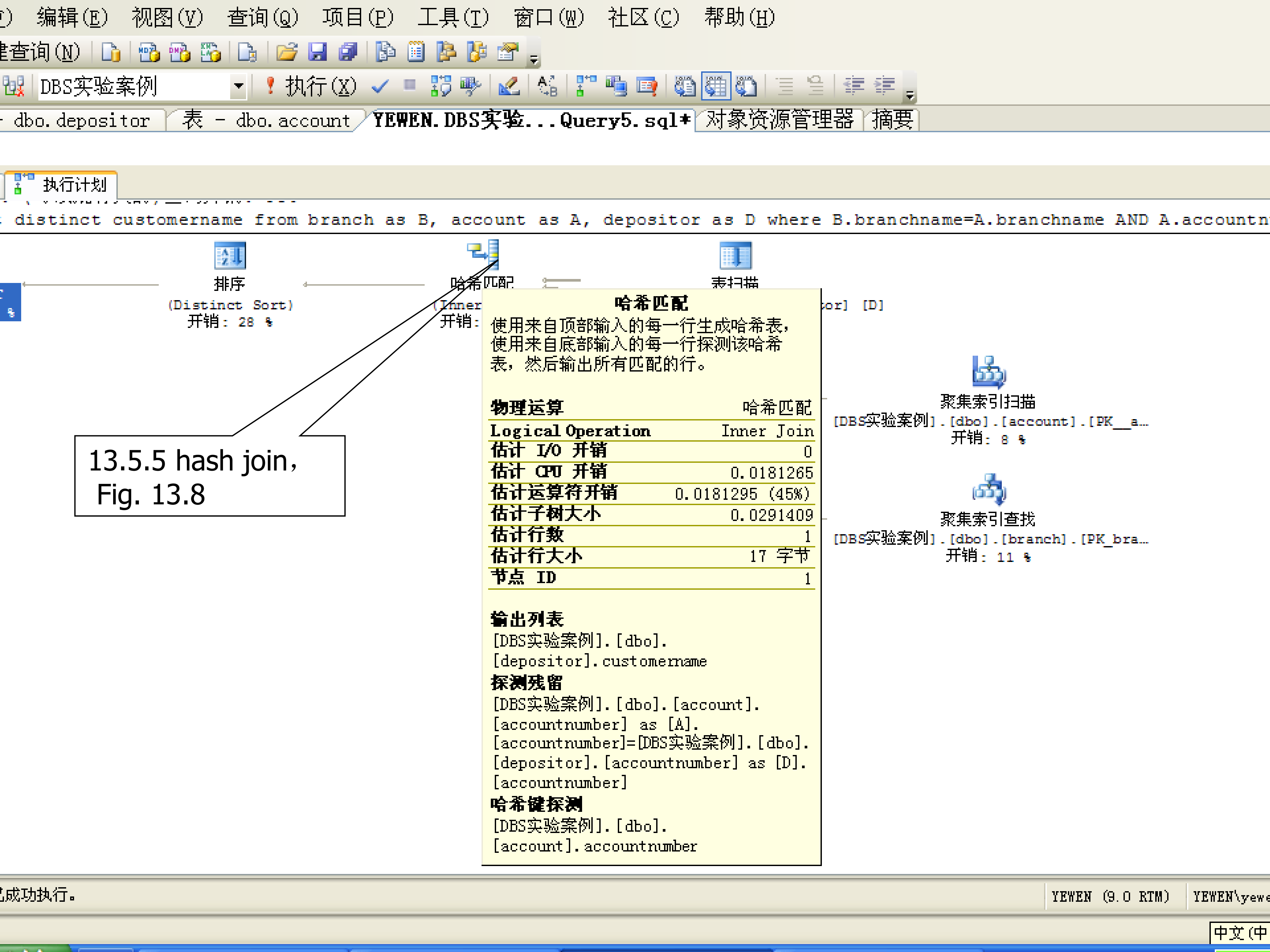
物理运算	表扫描
Logical Operation	Table Scan
估计 I/O 开销	0.003125
估计 CPU 开销	0.0001647
估计运算符开销	0.0032897 (8%)
估计子树大小	0.0032897
估计行数	7
估计行大小	24 字节
已排序	False
节点 ID	2

对象

[DBS实验案例].[dbo].[depositor] [D]

输出列表

[DBS实验案例].[dbo].[depositor].customername, [DBS实验案例].[dbo].[depositor].accountnumber



```
distinct customername from branch as B, account as A, depositor as D where B.branchname=A.branchname AND A.accountn
```

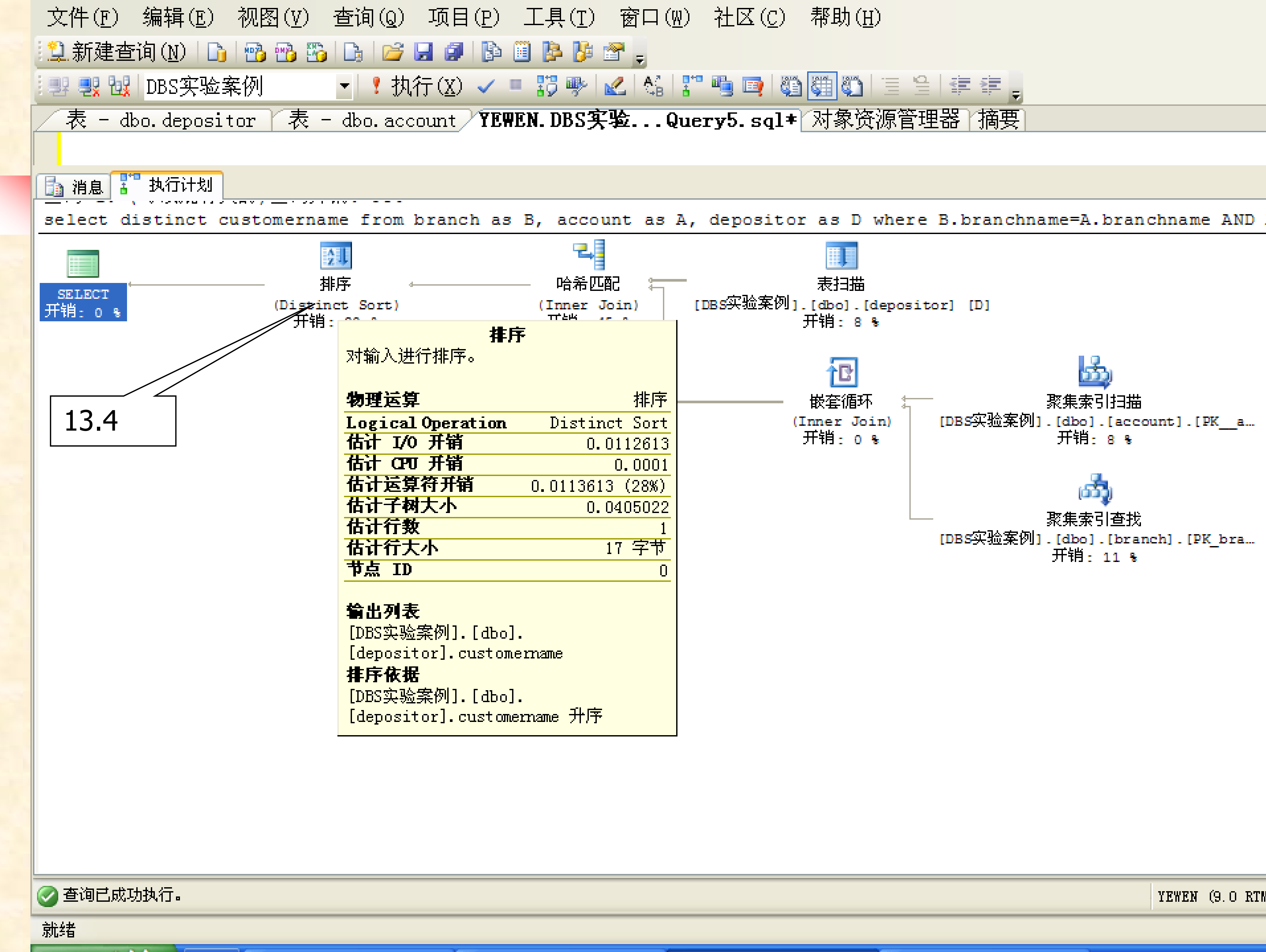
13.5.5 hash join,
Fig. 13.8

哈希匹配

使用来自顶部输入的每一行生成哈希表，使用来自底部输入的每一行探测该哈希表，然后输出所有匹配的行。

物理运算	哈希匹配
Logical Operation	Inner Join
估计 I/O 开销	0
估计 CPU 开销	0.0181265
估计运算符开销	0.0181295 (45%)
估计子树大小	0.0291409
估计行数	1
估计行大小	17 字节
节点 ID	1

输出列表
[DBS实验案例].[dbo].[depositor].customername
探测残留
[DBS实验案例].[dbo].[account].[accountnumber] as [A].[accountnumber]=[DBS实验案例].[dbo].[depositor].[accountnumber] as [D].[accountnumber]
哈希键探测
[DBS实验案例].[dbo].[account].accountnumber



```
select distinct customername from branch as B, account as A, depositor as D where B.branchname=A.branchname AND
```

排序

对输入进行排序。

物理运算	排序
Logical Operation	Distinct Sort
估计 I/O 开销	0.0112613
估计 CPU 开销	0.0001
估计运算符开销	0.0113613 (28%)
估计子树大小	0.0405022
估计行数	1
估计行大小	17 字节
节点 ID	0

输出列表

[DBS实验案例].[dbo].
[depositor].customername

排序依据

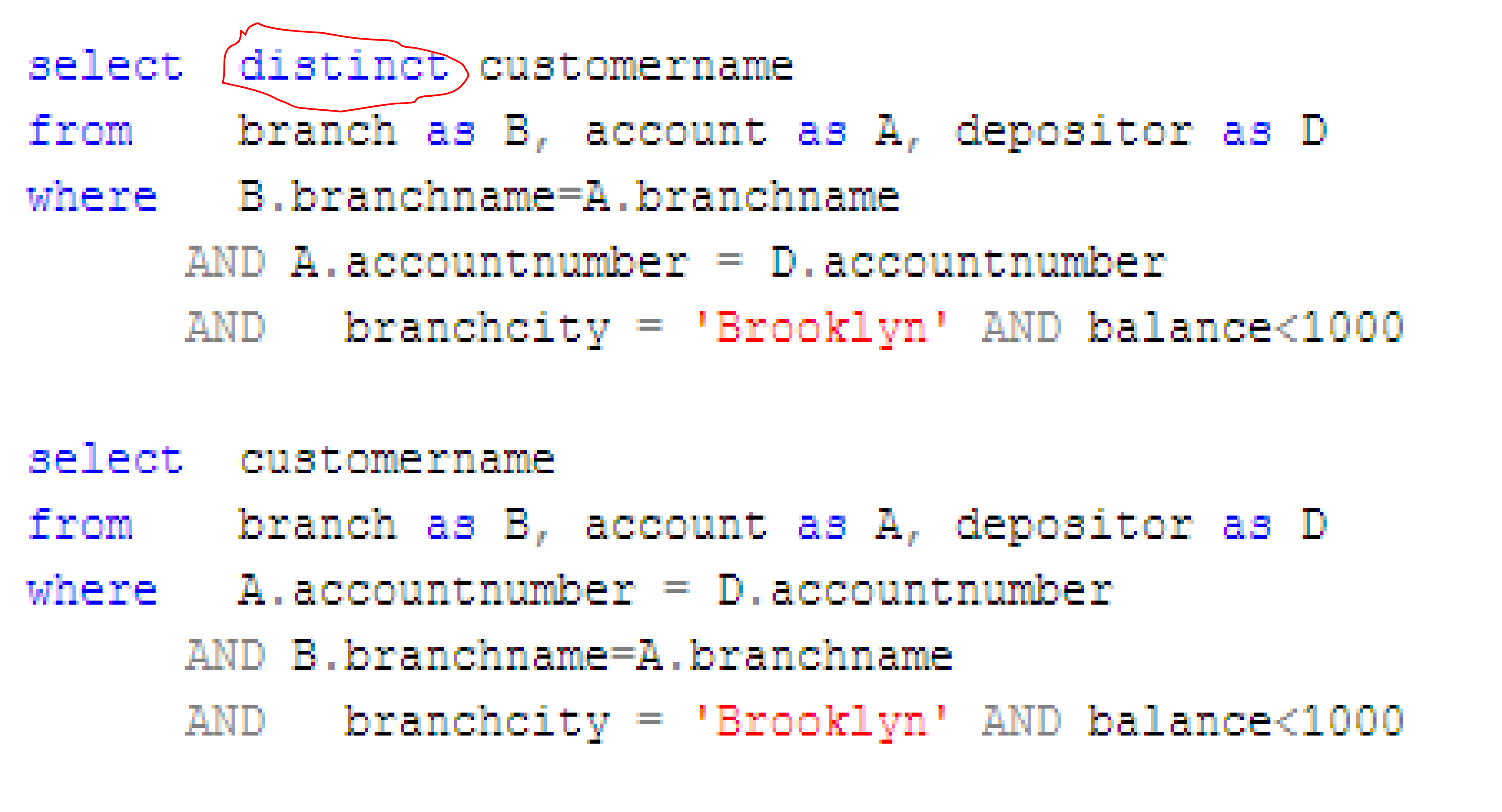
[DBS实验案例].[dbo].
[depositor].customername 升序

13.4

查询已成功执行。

比较2条SQL语句查询成本

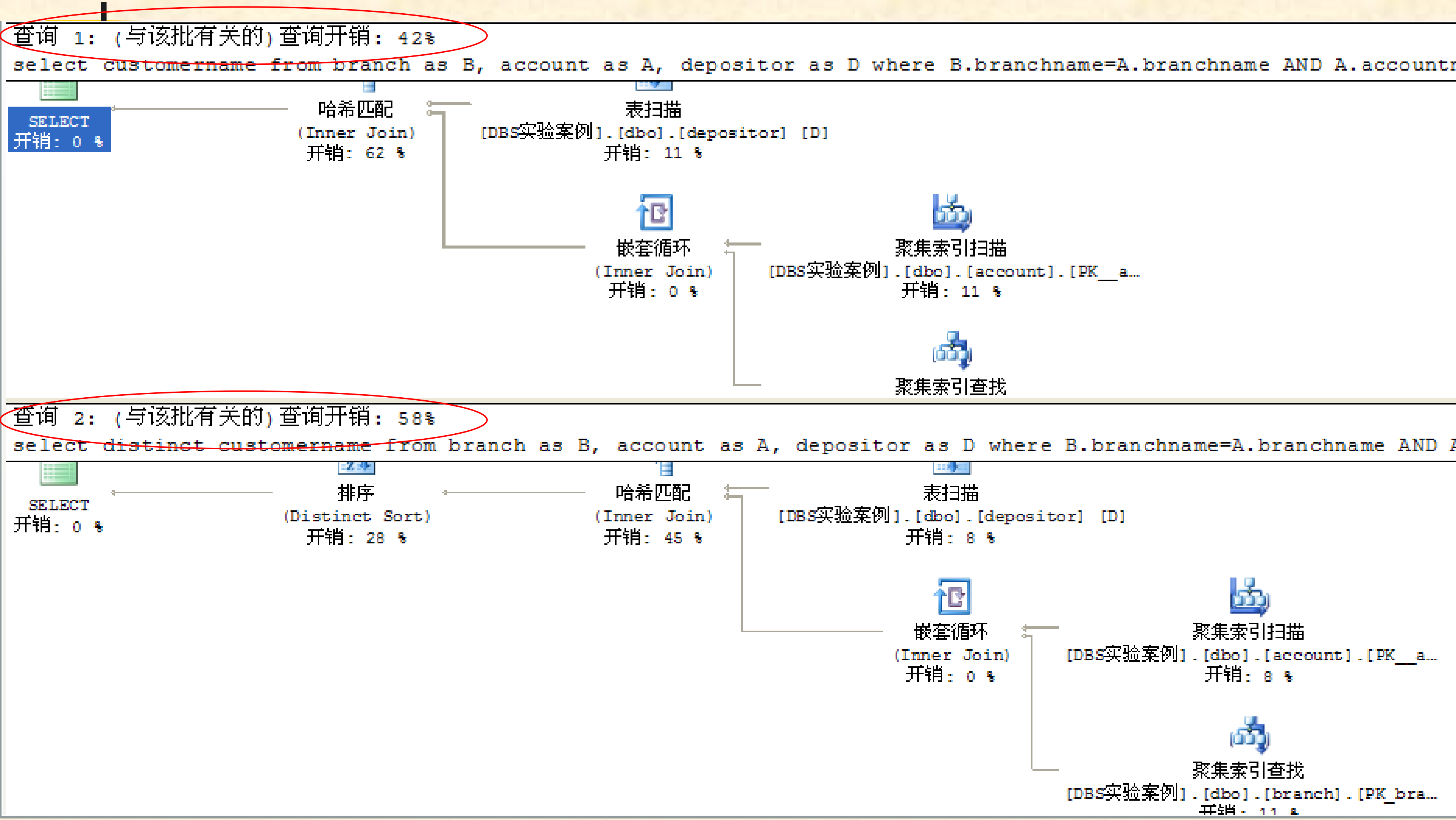
同时提交1批（2条）查询语句：



```
select  distinct customername
from    branch as B, account as A, depositor as D
where   B.branchname=A.branchname
      AND A.accountnumber = D.accountnumber
      AND   branchcity = 'Brooklyn' AND balance<1000

select  customername
from    branch as B, account as A, depositor as D
where   A.accountnumber = D.accountnumber
      AND B.branchname=A.branchname
      AND   branchcity = 'Brooklyn' AND balance<1000
```

该批次中2条查询语句的成本之比： 42% vs. 58%

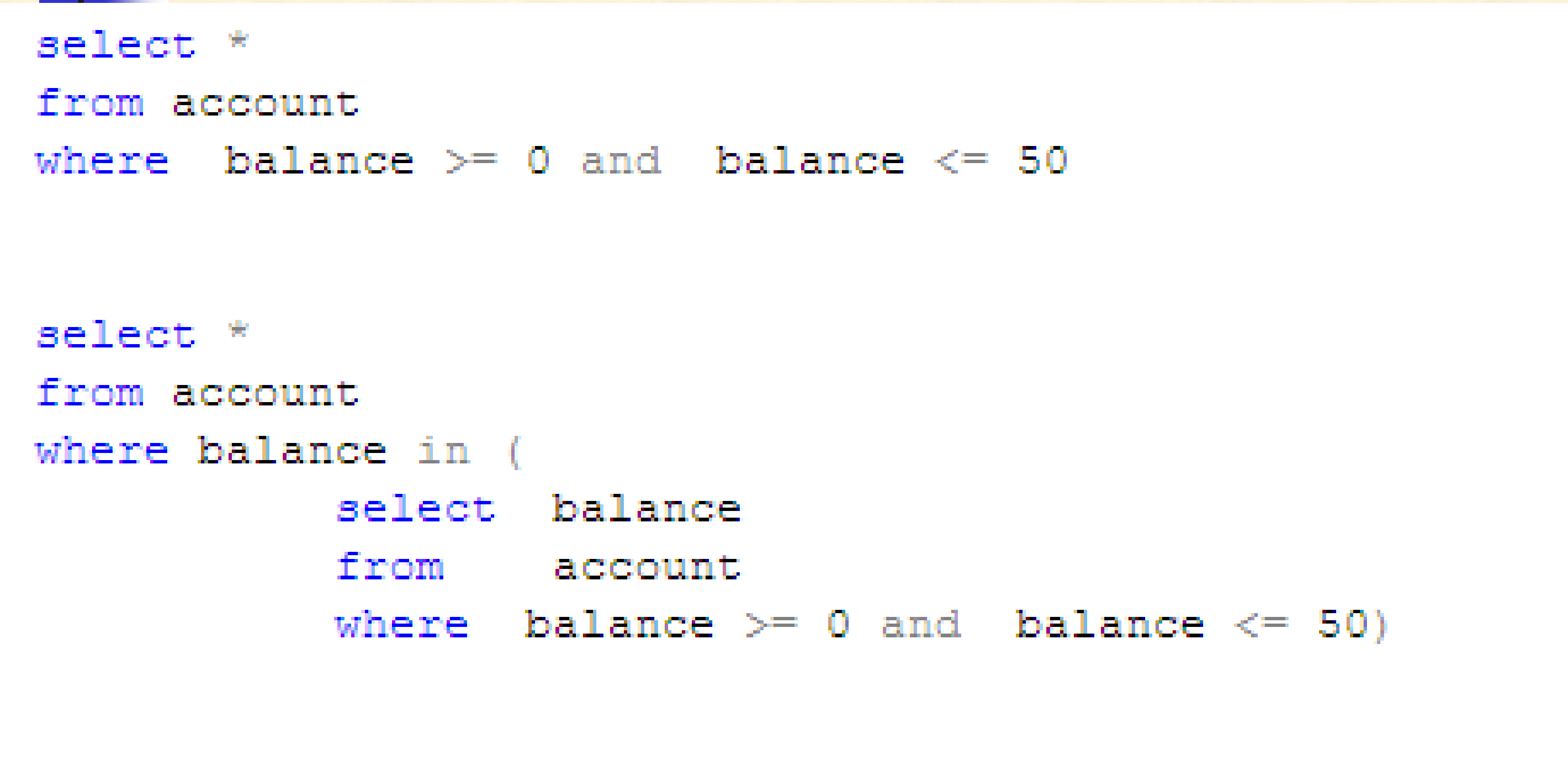


查询 1: (与该批有关的)查询开销: 42%

```
select customername from branch as B, account as A, depositor as D where B.branchname=A.branchname AND A.accountr
```

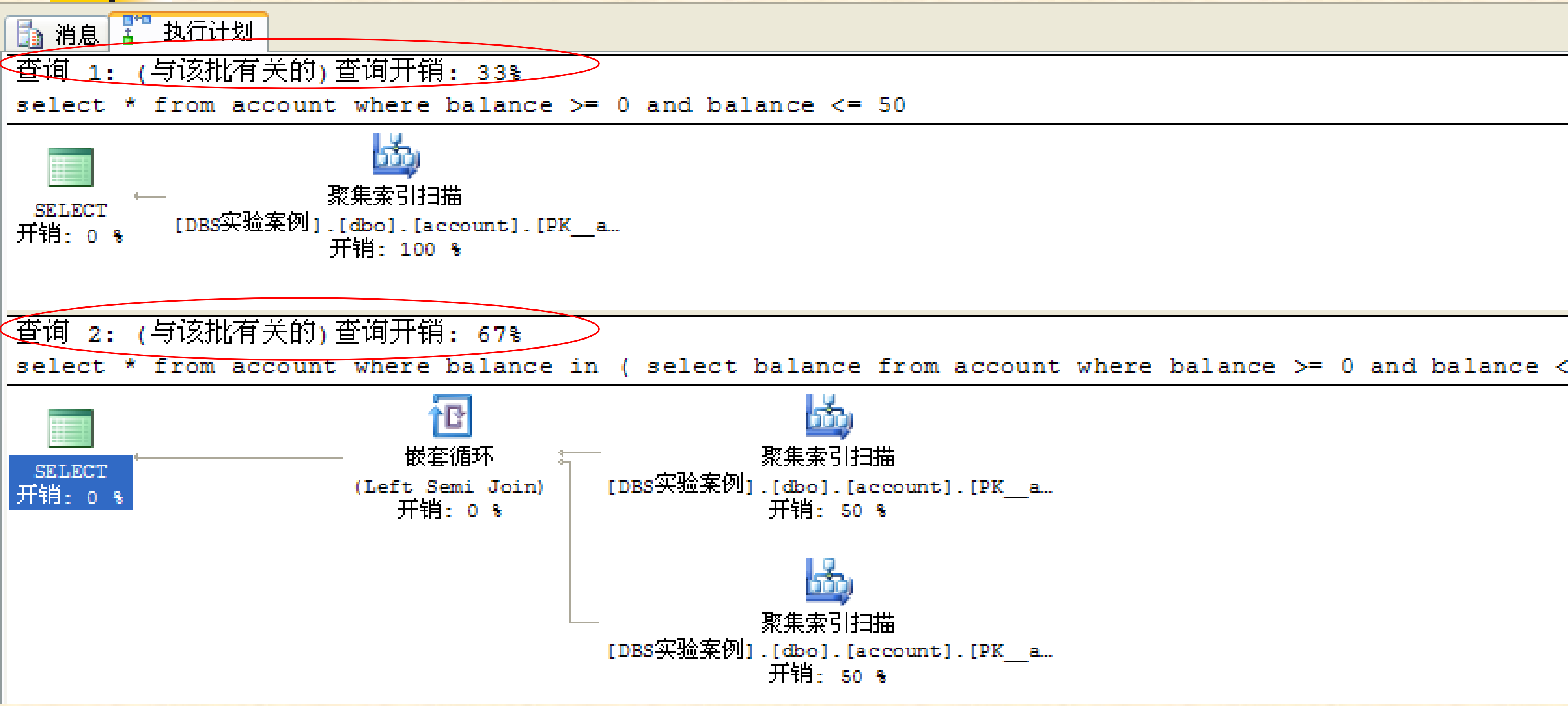
查询 2: (与该批有关的)查询开销: 58%

```
select distinct customername from branch as B, account as A, depositor as D where B.branchname=A.branchname AND A
```



```
select *
from account
where  balance >= 0 and  balance <= 50


select *
from account
where balance in (
           select  balance
           from    account
           where  balance >= 0 and  balance <= 50)
```



查询 1: (与该批有关的)查询开销: 33%

```
select * from account where balance >= 0 and balance <= 50
```

查询 2: (与该批有关的)查询开销: 67%

```
select * from account where balance in ( select balance from account where balance >= 0 and balance <
```