数据库系统原理

Database System Principle

邵蓥侠

Email：shaoyx@bupt.edu.cn

北京邮电大学计算机学院

计算机应用技术中心

PART 3

QUERY PROCESSING AND OPTIMIZATION

Chapter 15

Query Processing

Introduction

- Query
  - one or more operations on a database, or requests for DB access
  - a high-level database language (e.g. SQL, QUEL, declarative) statement, or a sequence of statements
- Query processing
  - (DBMS') activities involved in extracting data from a database, including *parsing and translation*, *query optimization*, and *evaluation*（执行）
- Programs compiling and executing vs. Query processing
  - refer to Fig. 15.0.1

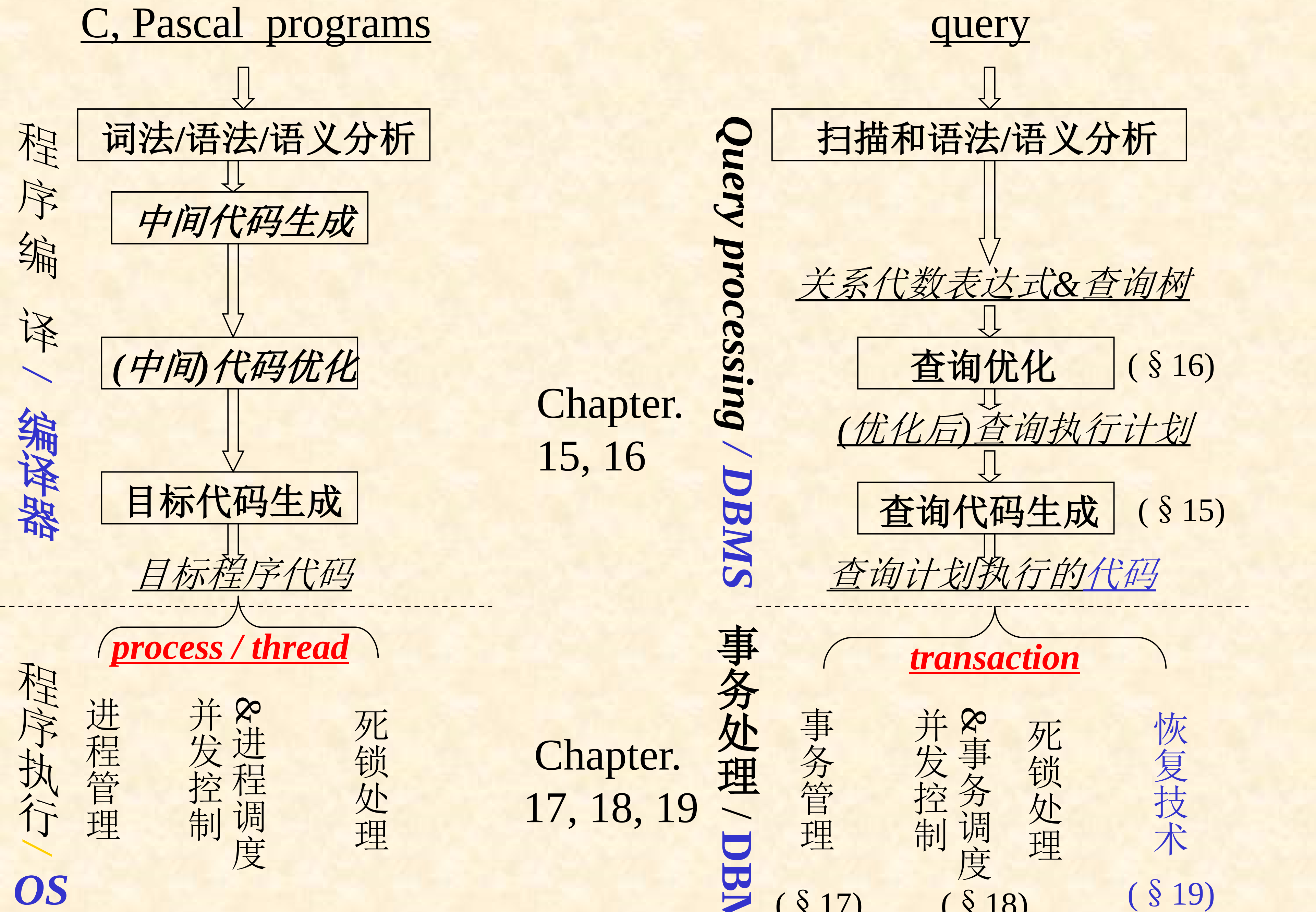

C, Pascal programs
词法/语法/语义分析
中间代码生成
(中间)代码优化
目标代码生成
目标程序代码
程序编译/编译器
process / thread
进程管理
并发控制&进程调度
死锁处理
程序执行/OS
Chapter. 15, 16
Chapter. 17, 18, 19
Query processing / DBMS
事务处理 / DBMS
query
扫描和语法/语义分析
关系代数表达式&查询树
查询优化
(§16)
(优化后)查询执行计划
查询代码生成
(§15)
查询计划执行的代码
transaction
事务管理
(§17)
并发控制&事务调度
(§18)
死锁处理
恢复技术
(§19)

Fig. 15.0.1

Chapter 15: Query Processing

- Overview
- Measures of Query Cost
- Selection Operation
- Sorting
- Join Operation
- Other Operations
- Evaluation of Expressions

Main Parts Chapter 15

- Basic steps in query processing, § 15.1 Overview
- Measures of query cost, § 15.2
- Evaluating of individual relational algebra operations,
  § 15.3- § 15.6
  - selection, sorting, join, project, set operations, …
- Evaluating of expression, i.e. a sequence of relational algebra operations, § 15.7

§ 15.1 Overview

- Steps in query processing
  - Fig.15.1

1. Parsing and translation
2. Optimization
3. Evaluation

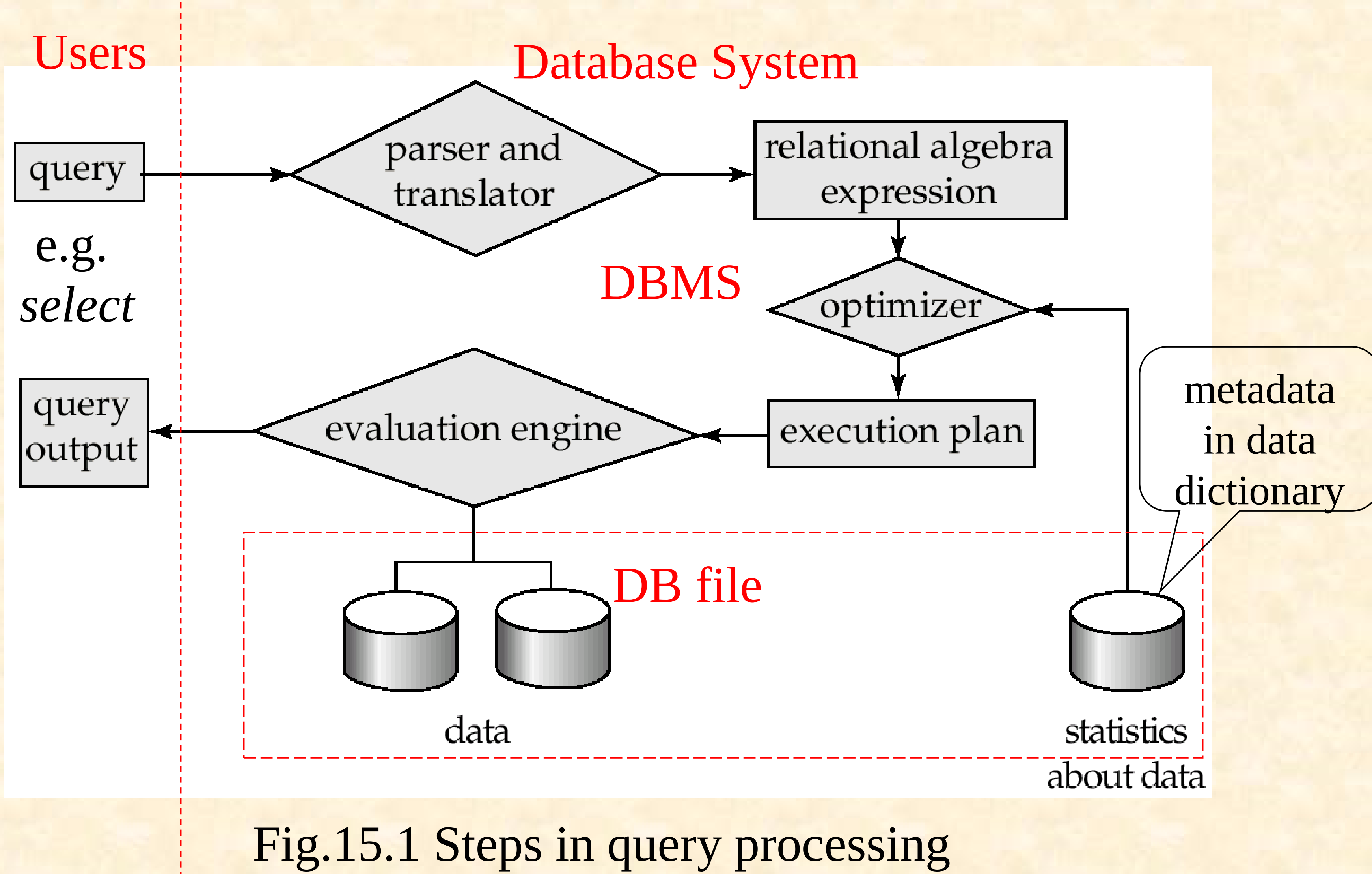


Fig.15.1 Steps in query processing

Overview (cont.)

- **Step1.Parsing and translation**
  - translate the query into a parser-tree representation
    - parser checks syntax, verifies the correctness of the relations in the query
    - parser also replaces the view in the query with the relations on which this view is built on
  - the parser-tree is then translated into relational algebra expression

Overview (cont.)

- **Step2. Query Optimization**
  - amongst all equivalent query ***evaluation plans***, choose the one with lowest cost
  - the cost is estimated using statistical information in data dictionary
    - e.g. the number of tuples in each relation, size of tuples, etc

```
select salary
from instructor
where salary < 75000;
```

Step2. Optimization

- A relational algebra expression may have many equivalent expressions
  - E.g., $\sigma_{salary<75000}(\Pi_{salary}(instructor))$ is equivalent to $\Pi_{salary}(\sigma_{salary<75000}(instructor))$
- Each relational algebra operation can be evaluated using one of several different algorithms
  - e.g. the ***select*** operation can be evaluated using one of A1, A2, …, A11 algorithms in §15.3
  - e.g. merge ***join***, hash ***join*** operation
- Correspondingly, a relational-algebra expression can be evaluated in many ways

Step2. Optimization

- Annotated expression specifying detailed evaluation strategy is called an **evaluation-plan**, such as *index*, *evaluation algorithms*, etc.
- e.g., can use an index on *salary* to find ***instructors*** with ***salary*** < 75000,
- or can perform complete relation scan and discard instructors with ***salary*** ≥ 75000

$$\Pi_{salary}\text{: A00}$$

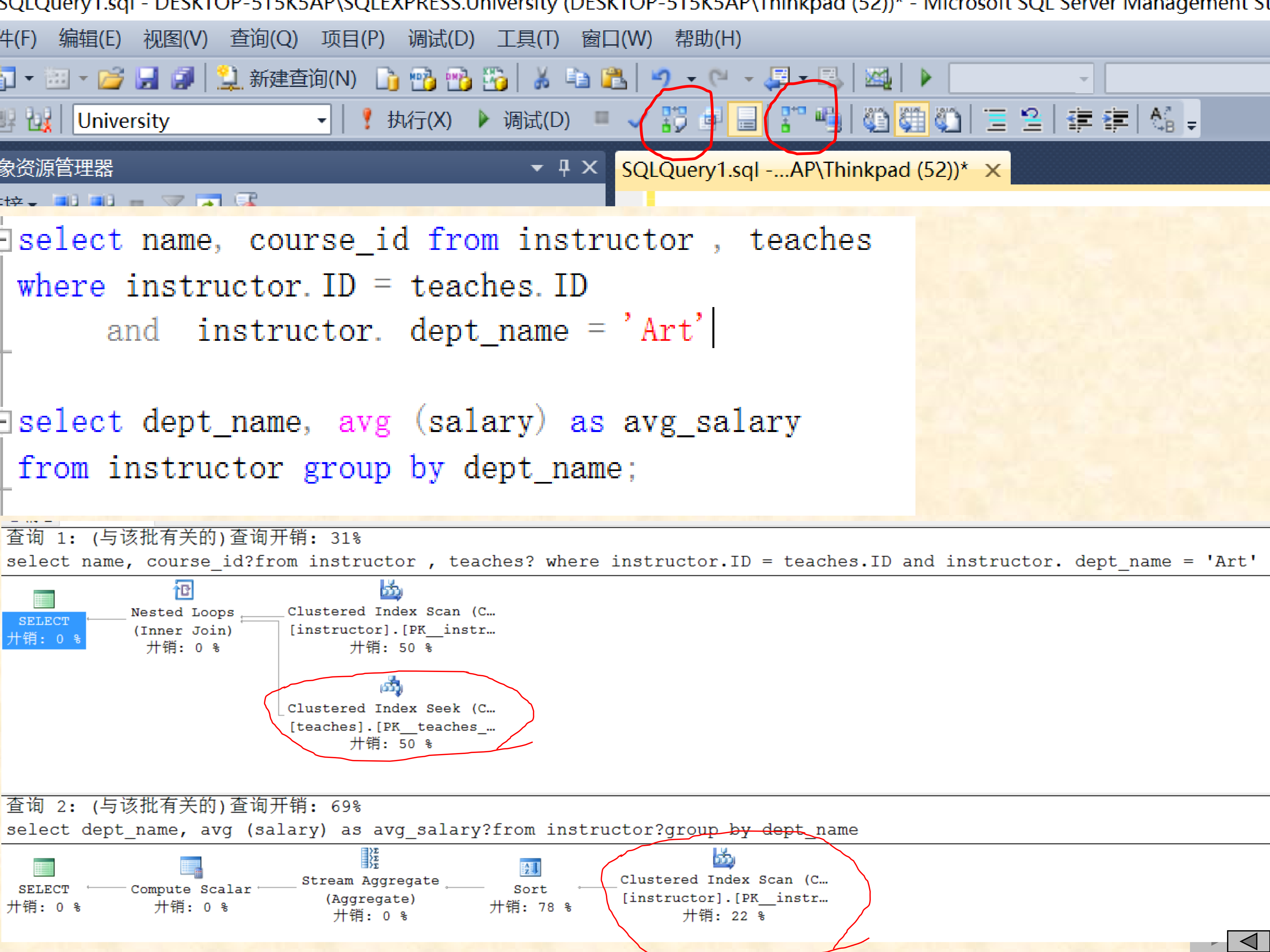
$$\sigma_{salary<75000}\text{ ; use index1 A5}$$

$$instructor$$

Fig. 15.2 A query-evaluation plan

Overview (cont.)

- **Step3. Evaluation plan execution**
  - the query-execution engine takes a *optimized* query-evaluation plan, executes that plan, and returns the answers to the query


SQLQuery1.sql - DESKTOP-5T5K5AP\SQLEXPRESS.University (DESKTOP-5T5K5AP\Thinkpad (52))* - Microsoft SQL Server Management S
件(F) 编辑(E) 视图(V) 查询(Q) 项目(P) 调试(D) 工具(T) 窗口(W) 帮助(H)
新建查询(N)
University
执行(X)
调试(D)
象资源管理器
SQLQuery1.sql -...AP\Thinkpad (52))*
select name, course_id from instructor , teaches
where instructor.ID = teaches.ID
and instructor. dept_name = 'Art'
select dept_name, avg (salary) as avg_salary
from instructor group by dept_name;
查询 1: (与该批有关的)查询开销: 31%
select name, course_id?from instructor , teaches? where instructor.ID = teaches.ID and instructor. dept_name = 'Art'
SELECT
开销: 0 %
Nested Loops
(Inner Join)
开销: 0 %
Clustered Index Scan (C...
[instructor].[PK__instr...
开销: 50 %
Clustered Index Seek (C...
[teaches].[PK__teaches_...
开销: 50 %
查询 2: (与该批有关的)查询开销: 69%
select dept_name, avg (salary) as avg_salary?from instructor?group by dept_name
SELECT
开销: 0 %
Compute Scalar
开销: 0 %
Stream Aggregate
(Aggregate)
开销: 0 %
Sort
开销: 78 %
Clustered Index Scan (C...
[instructor].[PK__instr...
开销: 22 %

SELECT
开销: 0 %

(Inner Join)
开销: 0 %

[instructor].[PK__instr…
开销: 50 %

Clustered Index Seek (C…
[teaches].[PK__teaches_…
开销: 50 %

```
select name, course_id from instr
where instructor.ID = teaches.ID
    and instructor.dept_name =

select dept_name, avg (salary) as
from instructor group by dept_nam
```

查询 2: (与该批有关的)查询开销: 69%
select dept_name, avg (salary) as avg_salary?from instructor?group b

SELECT
开销: 0 %

Compute Scalar
开销: 0 %

Stream Aggregate
(Aggregate)
开销: 0 %

Sort
开销: 78 %

Clustered Inde
[instructor].
开销: 2

Clustered Index Scan (Clustered)

整体扫描聚集索引或只扫描一定范围。

物理运算	Clustered Index Scan
Logical Operation	Clustered Index Scan
估计的执行模式	Row
存储	RowStore
Estimated Operator Cost	0.0032831 (22%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001581
Estimated Subtree Cost	0.0032831
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Row Size	26 字节
Ordered	False
节点 ID	3

对象
[University].[dbo].[instructor].
[PK__instruct__3214EC277D1BCB90]

Output List
[University].[dbo].[instructor].dept_name, [University].
[dbo].[instructor].salary

就绪

§ 15.2 Measures of Query Costs

- Cost is generally measured as total elapsed time for answering query
  - many factors contribute to time cost
    - *disk accesses*, *CPU*, or even network *communication*
- Typically ***disk access*** is the predominant cost, and is also relatively easy to estimate. Measured by taking into account
  - number of seeks * average-seek-cost
  - number of blocks read * average-block-read-cost
  - number of blocks written * average-block-write-cost
    - cost to write a block is greater than cost to read a block
      - data is read back after being written to ensure that the write was successful

Measures of Query Cost (cont.)

- For simplicity we just use the **number of block transfers** *from disk and the* **number of seeks** as the cost measures
  - t_T – time to transfer one block
  - t_S – time for one seek
  - cost for b block transfers plus S seeks

    $$b * t_T + S * t_S$$
- We ignore CPU costs for simplicity
  - real systems do take CPU cost into account
- We do not include cost to writing output to disk in our cost formulae

Measures of Query Cost (Cont.)

- Several algorithms can reduce disk IO by using extra buffer space
  - Amount of real memory available to buffer depends on other concurrent queries and OS processes, known only during execution
    - We often use worst case estimates, assuming only the minimum amount of memory needed for the operation is available
- Required data may be buffer resident already, avoiding disk I/O
  - But hard to take into account for cost estimation

§ 15.3 Selection Operation

- Selection operation on a relation ***r*** by **file scan**
  - locating and scanning the file in which ***r*** is stored to retrieving the *file records satisfying the selection conditions*

- E.g. $\Pi_{salary}(\sigma_{salary<2500}(instructor))$

Selection Operation

- Types of query conditions (查询条件类型)
  - equality(等值), e.g.*balance* = 100
  - range (范围), e.g. *balance* between 50 and 400
  - comparison (比较), e.g. *balance* >300
- Several file scan algorithms
  - linear search/scan – A1
  - selections using indices – A2, A3, A4
  - selections involving comparisons – A5, A6
  - complex selections – A7, A8, A9, A10

A1: File scan by **linear search**

- Algorithm **A1** (**linear search**).
- Scan each file block and test all records to see whether they satisfy the selection condition.
  - cost estimate = b_r block transfers + 1 seek
    - b_r denotes number of blocks containing records from relation r
  - if selection is on a key attribute, can stop on finding record
    - cost = (b_r /2) block transfers + 1 seek

—必须扫描全部blocks，方能找到全部满足查询条件的数据

A1: File scan by **linear search**

- Linear search can be applied regardless of
    - selection condition or
    - ordering of records in the file, or
    - availability of indices

- Note: binary search generally does not make sense since data is not stored consecutively
    - except when there is an index available,
    - and binary search requires more seeks than index search

Selections Using Indices

- **Index scan** – search algorithms that use an index

—— e.g. in SQL Server, ***index seek***

  - selection condition must be on search-key of index,

    i.e. ***instructor***(***<u>ID</u>***, name, ***dept-name***, salary)

- **A2** (**primary index, equality on key such as *ID***).

  Retrieve a single record that satisfies the corresponding equality condition, using a B^+-tree as the clustering/primary index

  - $Cost = (h_i + 1) * (t_T + t_S)$
  - h_i : height of the tree
  - t_T : time to transfer one block
  - t_S : *time for one seek*

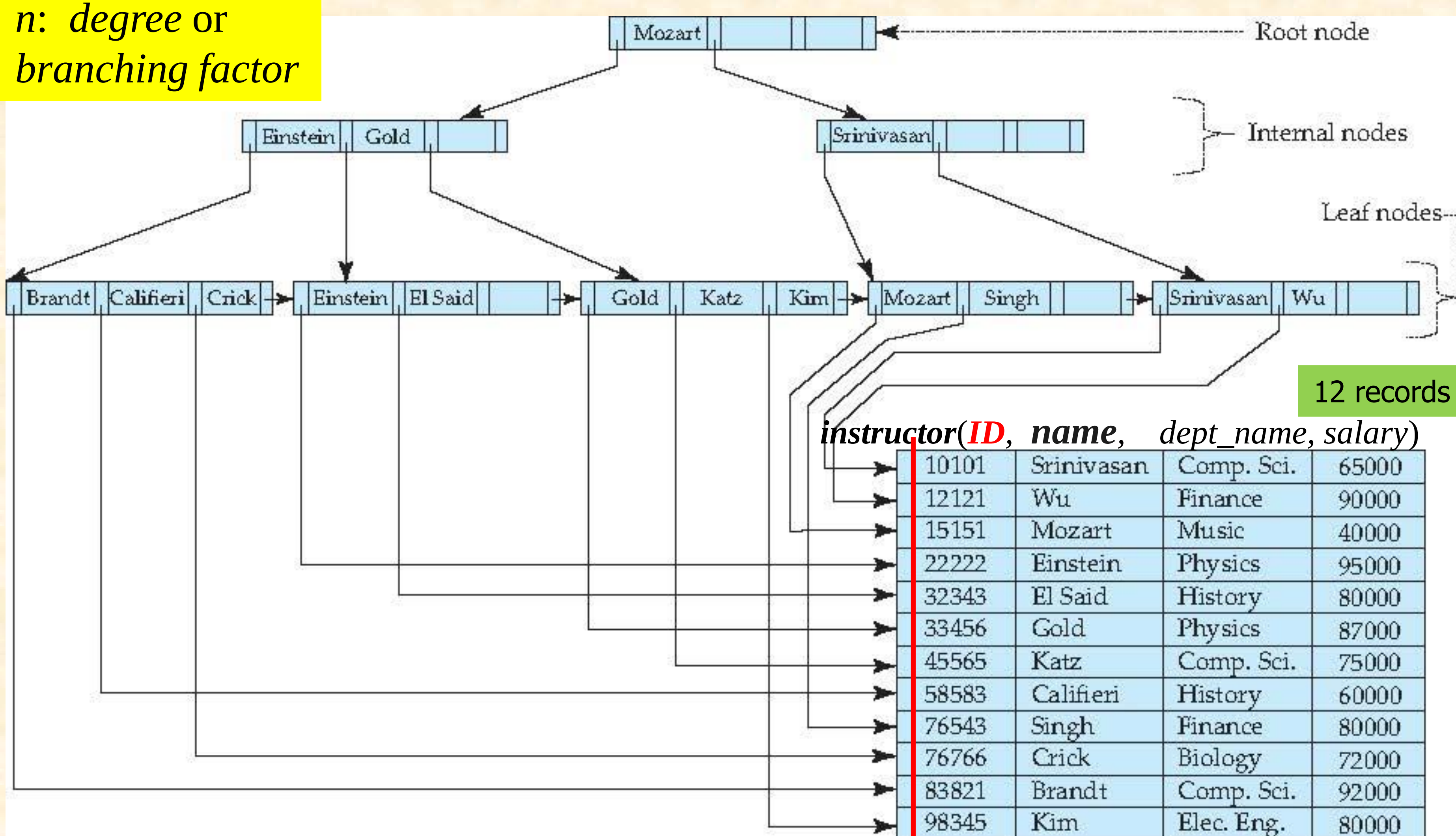
Example of B^+-Tree (n=4)

Not a primary index!!

非根、非叶结点的儿子结点个数*k*:

$2=\lceil n/2 \rceil \le k \le n=4$

n: *degree* or *branching factor*



12 records

instructor(*ID*, *name*, *dept_name*, *salary*)

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Selections Using Indices

- **A3** (**primary index, equality on nonkey**)

  Retrieve multiple records.

  - Records will be on consecutive blocks
    - Let b = number of blocks containing matching records
  - $Cost = h_i * (t_T + t_S) + t_S + t_T * \mathrm{b}$

Selections Using Indices

- **A4** (**secondary index, equality on nonkey**).
  - retrieve a single record if the search-key is a candidate key, e.g. ***name*** in ***instructor***
    - *cost* = $(h_i + 1) * (t_T + t_S)$
  - retrieve multiple records if search-key is not a candidate key
    - each of *n* matching records may be on a different block, e.g. ***salary*** in ***instructor*** - *next slide*
    - cost = $(h_i + n) * (t_T + t_S)$
      - can be very expensive!

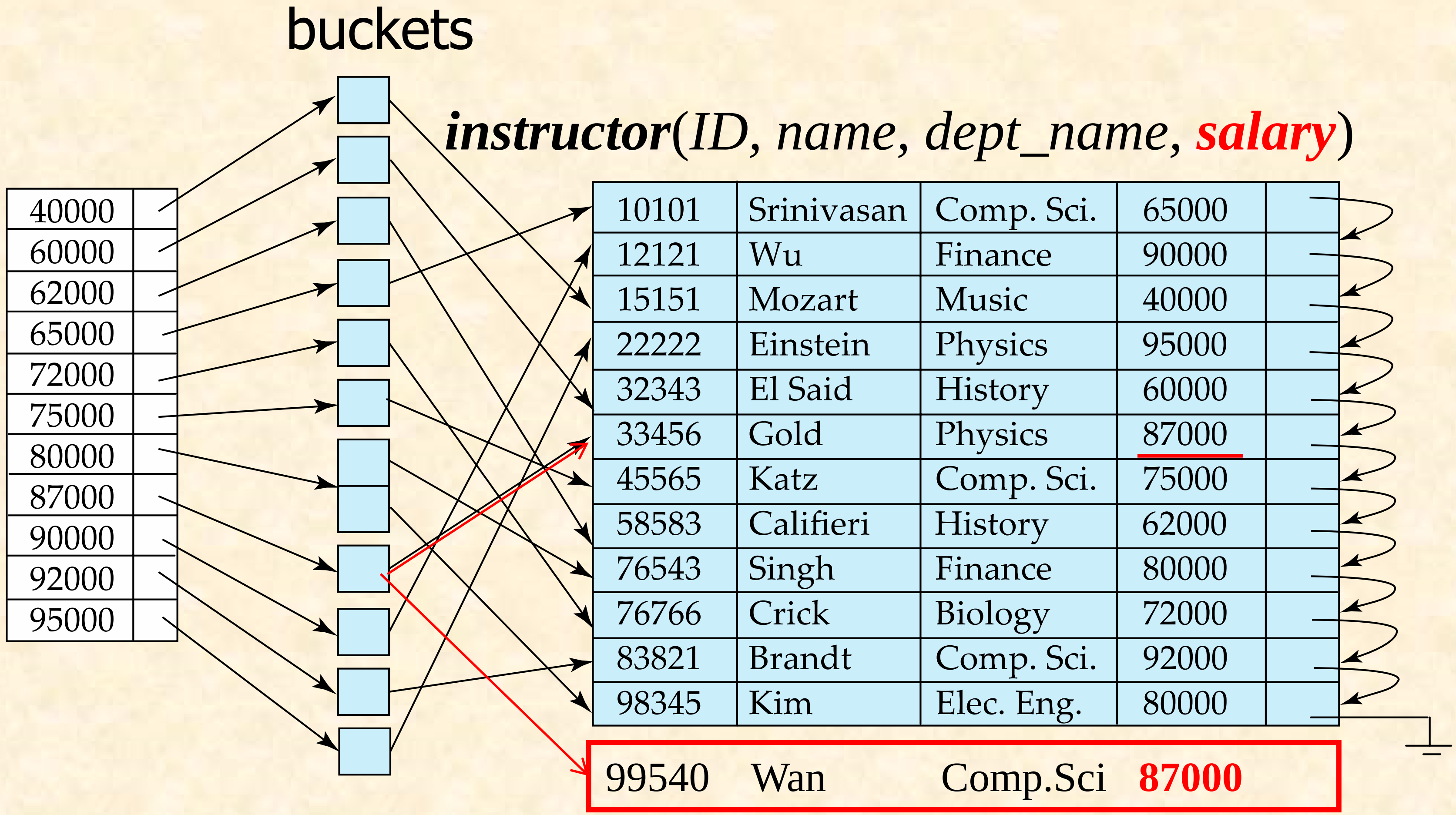
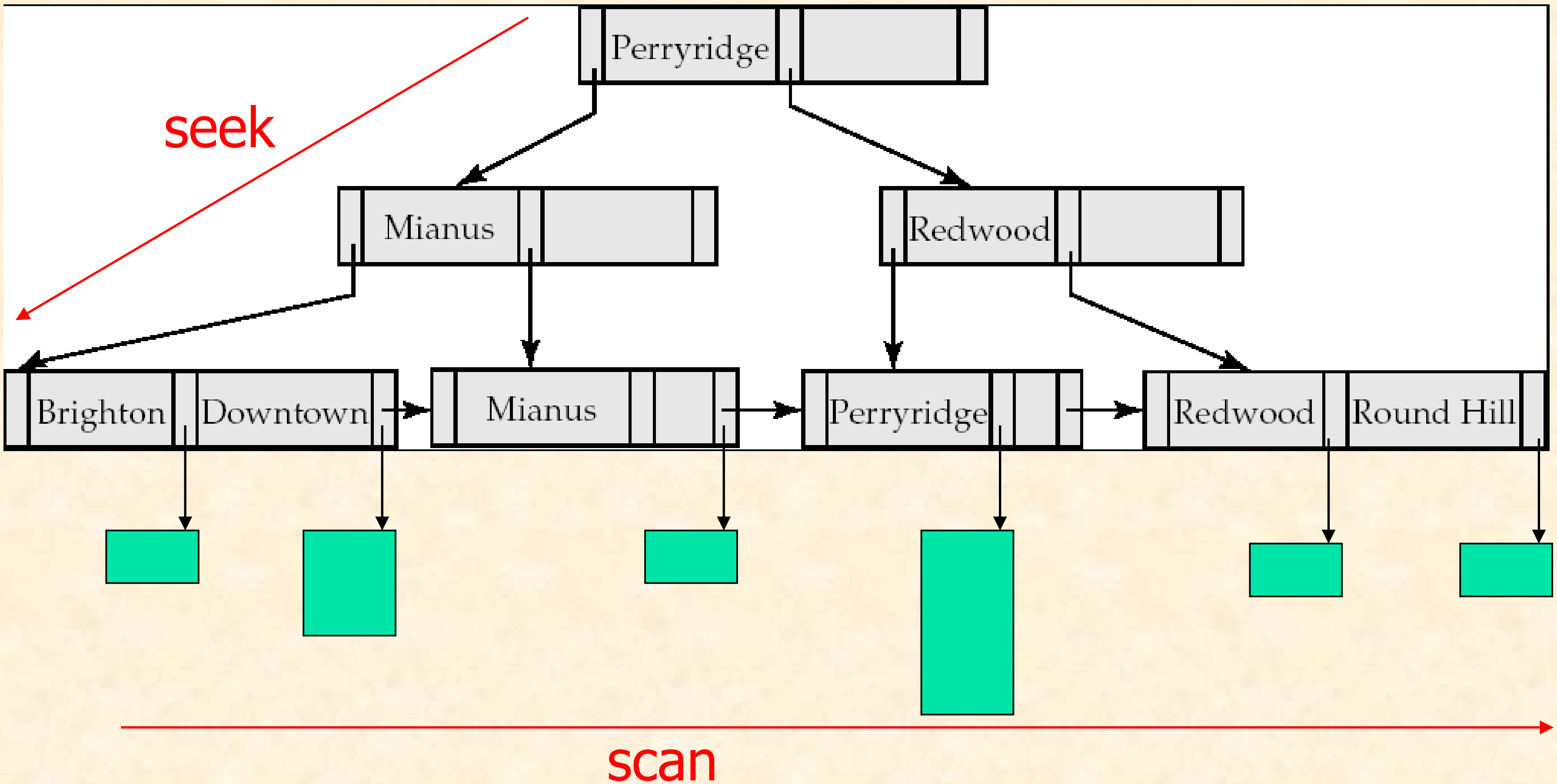


Fig. 11.6 Secondary index on *salary* field of *instructor*

聚集索引树：索引+数据

访问方式：

1. 聚集索引查找seek，e.g. branch-name, 二分查找
2. 聚集索引扫描sacn, e.g. account-number，线性扫描



未建立索引，堆文件
访问方式：线性扫描、表扫描

Account(***account_number***, ***branch-name***, ***balance***)

A-217	Brighton	750	
A-101	Downtown	500	
A-110	Downtown	600	
A-215	Mianus	700	
A-102	Perryridge	400	
A-201	Perryridge	900	
A-218	Perryridge	700	
A-222	Redwood	700	
A-305	Round Hill	350	

instructor(*ID*, *name*, *dept_name*, *salary*)

Biology	
Comp. Sci.	
Elec. Eng.	
Finance	
History	
Music	
Physics	

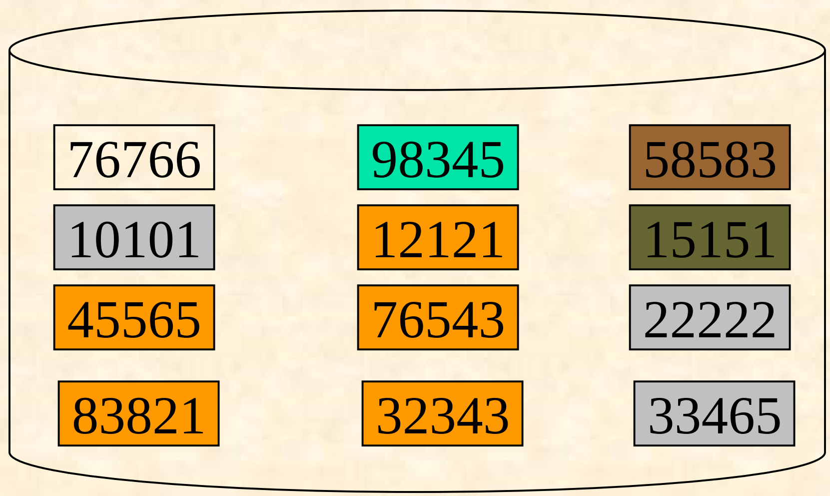
index file

76766	Crick	Biology	72000	
10101	Srinivasan	Comp. Sci.	65000	
45565	Katz	Comp. Sci.	75000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
12121	Wu	Finance	90000	
76543	Singh	Finance	80000	
32343	El Said	History	60000	
58583	Califieri	History	62000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
33465	Gold	Physics	87000	

logical file ***instructor***

indexed file

Note: the file ***instructor*** is logically a sequential file, but its records may be stored *non-contiguously or non-ordered* on the disk



physical file ***instructor***

Fig. 11.1 DB *indexed* file **instructor** and its *index* file

Selections Involving Comparisons - Range Search

- Can implement selections of the form $\sigma_{A \le V}(r)$ or $\sigma_{A \ge V}(r)$ by using
  - a linear file scan,
  - or by using indices in the following ways:

- **A5** (**primary index, comparison**). (Relation is sorted on A)
  - For $\sigma_{A \ge V}(r)$ use index to find first tuple $\ge v$ and scan relation sequentially from there
  - For $\sigma_{A \le V}(r)$ just scan relation sequentially till first tuple $> v$; do not use index

- e.g. $\sigma_{ID \ge 22222}$ *(instructor)* in

Selections Involving Comparisons

- **A6** (**secondary index, comparison**).
  - For $\sigma_{A \geq V}(r)$ use index to find first ***index entry*** $\geq v$ and scan index sequentially from there, to ***find pointers*** to records.
  - For $\sigma_{A \leq V}(r)$ just scan leaf pages of index finding pointers to records, till first entry $> v$
  - In either case, retrieve records that are pointed to
    - requires an I/O for each record
    - Linear file scan may be cheaper

Implementation of Complex Selections - **Conjunction**

- **Conjunction:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$
- **A7** (**conjunctive selection using one index**).
  - Select a combination of θ_i and algorithms A1 through A7 that results in the least cost for $\sigma_{\theta i}(r)$.
  - Test other conditions on tuple after fetching it into memory buffer.
- **A8** (**conjunctive selection using composite index**).
  - Use appropriate composite (multiple-key) index if available.

Implementation of Complex Selections - **Conjunction**

- **A9** (**conjunctive selection by intersection of identifiers**).
  - Requires indices with record pointers.
  - Use corresponding index for each condition, and take intersection of all the obtained sets of record pointers.
  - Then fetch records from file
  - If some conditions do not have appropriate indices, apply test in memory.

Algorithms for Complex Selections

- **Disjunction:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$.
- **A10** (**disjunctive selection by union of identifiers**).
  - Applicable if *all* conditions have available indices.
    - Otherwise use linear scan.
  - Use corresponding index for each condition, and take union of all the obtained sets of record pointers.
  - Then fetch records from file
- **Negation:** $\sigma_{\neg\theta}(r)$
  - Use linear scan on file
  - If very few records satisfy $\neg\theta$, and an index is applicable to θ
    - Find satisfying records using index and fetch from file

Other Operations

- *Sorting*, *join*, *projection*, *set operations* (e.g. union, intersection, set-difference), *aggregation*, and *duplicate elimination* are implemented on the basis of the operation *select*
  - sorting,

    e.g. merge sort
  - join,

    (block/indexed) nested-loop join, merge join, hash join
  - *project*, *distinct*, *order by*, *outer join*, *aggregation*

§ 15.7 Evaluation of Expressions

- How to evaluate an expression containing multiple evaluation primitives
  - key: how to process intermediate computing results
  - e.g. $\Pi_{sarlay}(\sigma_{salary<2500}(\mathit{instructor}))$

- Two strategies, that is ***materialization***(实体化/物化) and ***pipeline*** are used for expression evaluation

Materialization

π_{salary}

$\sigma_{salary < 75000;\ \text{use index 1}}$ Temp数据库

instructor

Fig. 12.0.2 Pictorial representation of an expression

Materialization (cont.)

- Principles
  - start from the lowest-level, i.e. at the bottom of the tree, evaluate one operation at a time
  - the results of each evaluation (i.e. intermediate computing results) are stored in ***temporal relations*** on the disk for subsequent evaluation
    - serial evaluating
- E.g., in Fig.12.0.2, compute and store in a ***temporal relation***

$$\sigma_{salary<2500}(instructor)$$

at first;

Materialization (cont.)

- Cost of *writing results to disk and reading them back* can be quite high
- Approximately,

overall cost =

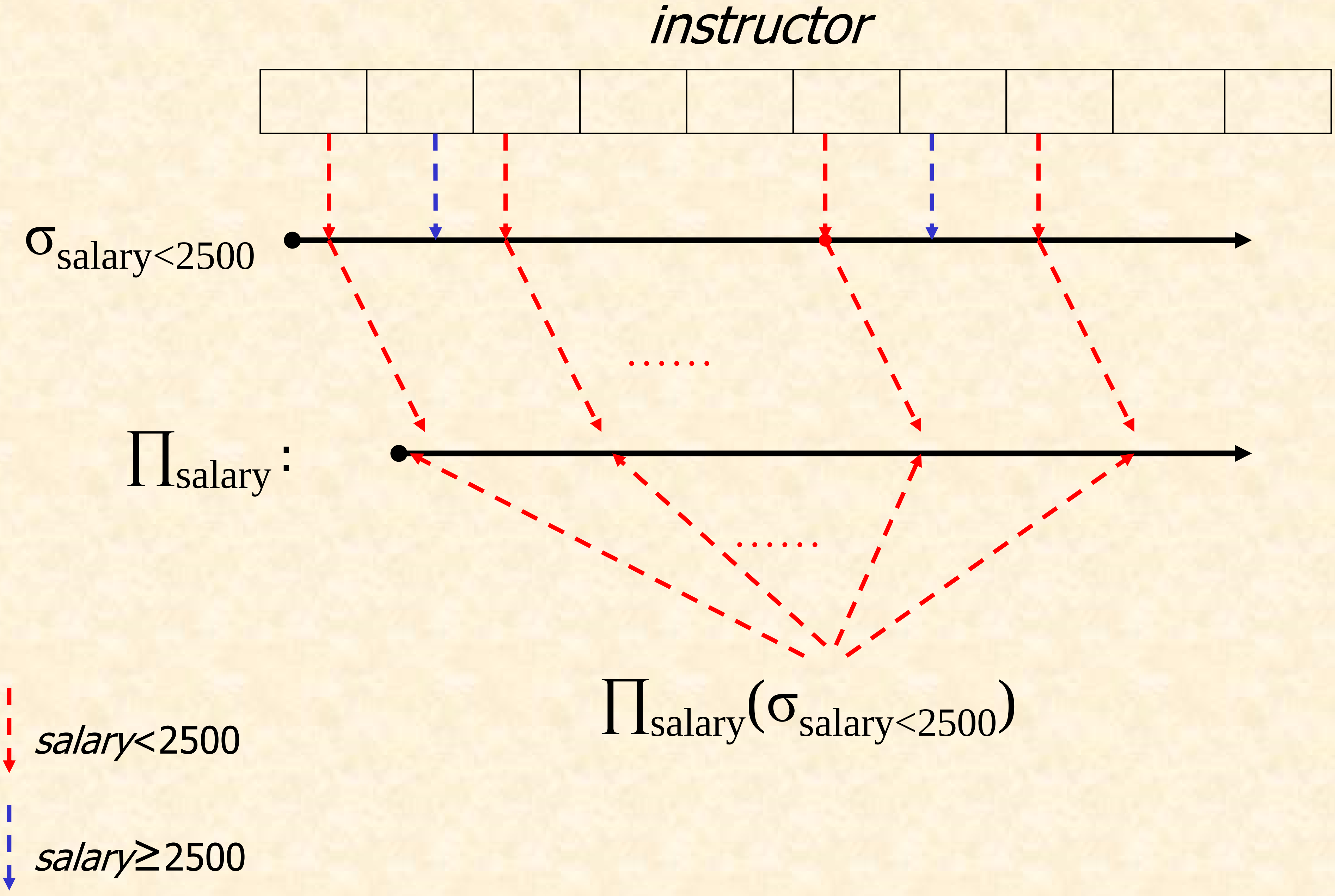
sum of costs of individual operations +
cost of writing intermediate results to disk

Pipelined（流水线）

- Principles of pipelined evaluation
  - evaluate several operations simultaneously in a pipeline, with the results of one operation passed to the next, without the need to store temporary relations in disk
    - parallel evaluating
- Much cheaper than materialization:
    - no need to store a temporary relation to disk
- E.g., in the expression tree in Fig.12.0.2, don't store result of

$$\sigma_{salary<2500}(instructor)$$

  - don't store result of selection, pass tuples directly to projection.



instructor
$\sigma_{salary<2500}$
Π_{salary} :
$\Pi_{salary}(\sigma_{salary<2500})$
salary<2500
salary≥2500