Database Management Systems

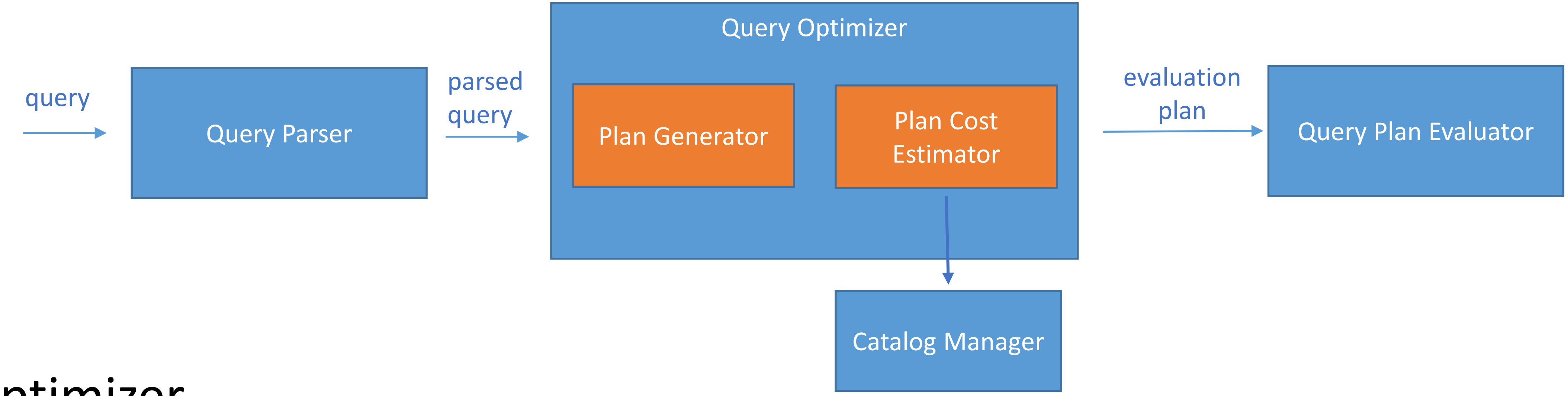
Lecture 9

Evaluating Relational Operators

Query Optimization

Sabina S. CS

Query Optimization



- optimizer
  - objective
    - given a query Q, find a good evaluation plan for Q
  - generates alternative plans for Q, estimates their costs, and chooses the one with the least estimated cost
  - uses information from the system catalogs

- running example - schema
  - Students (<u>SID: integer</u>, SName: string, Age: integer)
  - Courses (<u>CID: integer</u>, CName: string, Description: string)
  - Exams (<u>SID: integer, CID: integer, EDate: date</u>, Grade: integer)

  - Students
    - every record has 50 bytes
    - there are 80 records / page
    - 500 pages
  - Courses
    - every record has 40 bytes
    - there are 100 records / page
    - 1 page
  - Exams
    - every record has 40 bytes
    - there are 100 records / page
    - 1000 pages

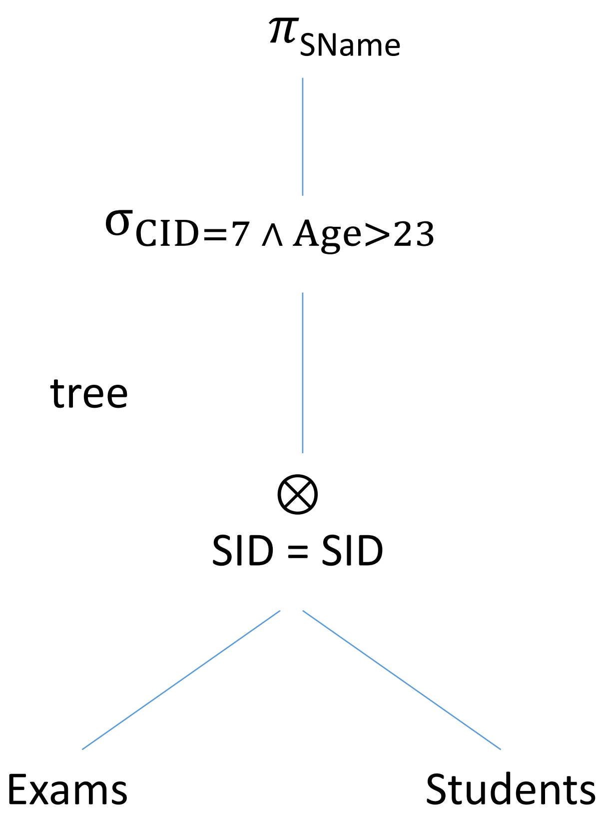
Query Evaluation Plans

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
     AND S.Age > 23
```

query

$$\pi_{SName}(\sigma_{CID=7 \wedge Age>23}(Exams \otimes_{SID=SID} Students))$$

relational algebra expression

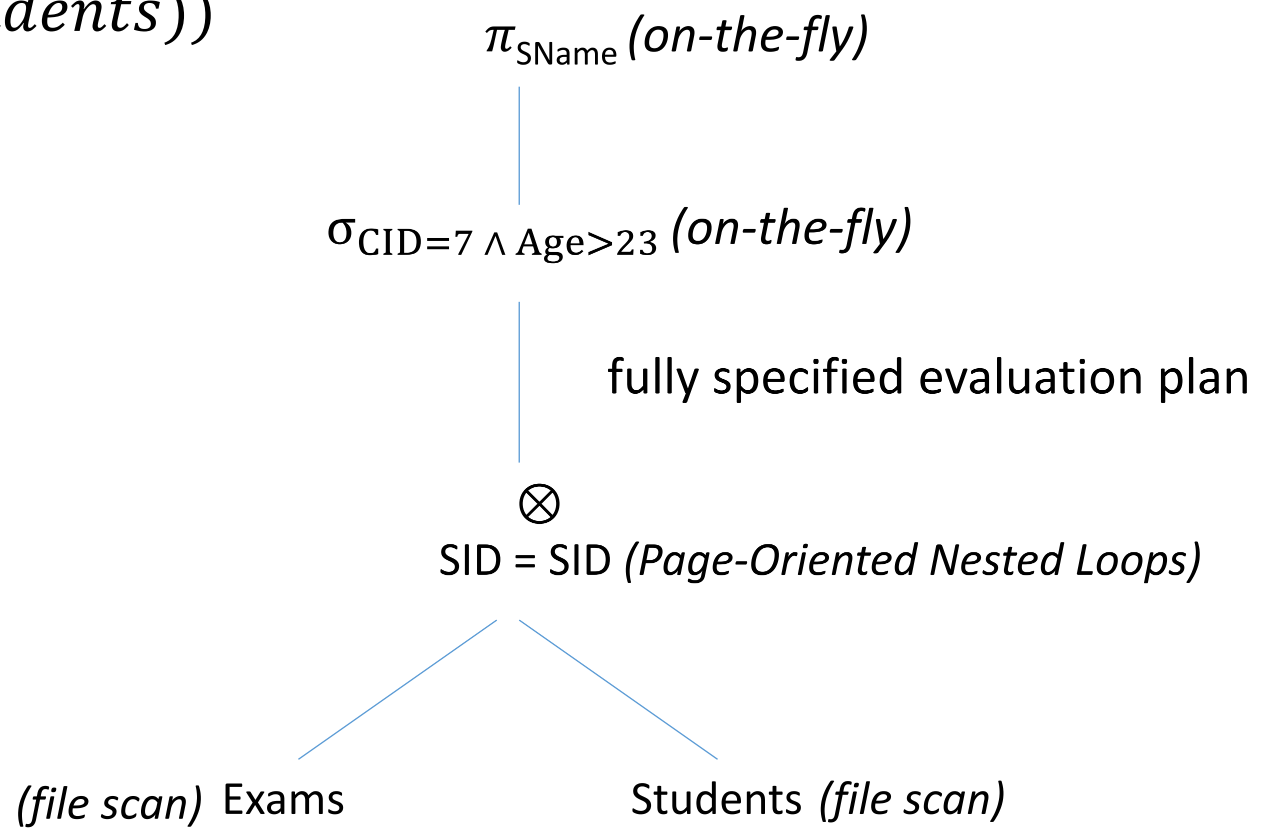


Query Evaluation Plans

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
     AND S.Age > 23
```

$$\pi_{SName}(\sigma_{CID=7 \land Age>23}(Exams \otimes_{SID=SID} Students))$$

- query evaluation plan
  - extended relational algebra tree
  - node – annotations
    - relation
      - access method
    - relational operator
      - implementation method



Query Evaluation Plans

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
     AND S.Age > 23
```

$$\pi_{SName}(\sigma_{CID=7 \wedge Age>23}(Exams \otimes_{SID=SID} Students))$$

- Page-Oriented Nested Loops Join
  - Exams – outer relation
- selection, projection applied on-the-fly
  to each tuple in the join result, i.e., the result of the join (before applying selection and projection) is not stored

π_{SName} *(on-the-fly)*

$\sigma_{\mathrm{CID=7 \wedge Age>23}}$ *(on-the-fly)*

fully specified evaluation plan

$\otimes$

SID = SID *(Page-Oriented Nested Loops)*

(file scan) Exams

Students *(file scan)*

Pipelined Evaluation

```
SELECT *
FROM Exams
WHERE EDate > '1-1-2020' AND Grade > 8
```

$\underbrace{\texttt{EDate > '1-1-2020'}}_{T1}$ AND $\underbrace{\texttt{Grade > 8}}_{T2}$

$$\sigma_{Grade>8}(\sigma_{EDate>'1-1-2020'}(Exams))$$

- index *I* matches *T1*
- v1 - *materialization*
    - evaluate *T1*
    - write out result tuples to temporary relation *R*, i.e., tuples are *materialized*
    - apply the 2nd selection to *R*
    - cost: read and write *R*

Pipelined Evaluation

```
SELECT *
FROM Exams
WHERE EDate > '1-1-2020' AND Grade > 8
```

$\underbrace{\texttt{EDate > '1-1-2020'}}_{T1}$ AND $\underbrace{\texttt{Grade > 8}}_{T2}$

- v2 – *pipelined evaluation*
  - apply the 2nd selection to each tuple in the result of the 1st selection as it is produced
  - i.e., 2nd selection operator is applied *on-the-fly*
  - saves the cost of writing out / reading in the temporary relation *R*

Query Blocks – Units of Optimization

- parse Q => collection of query *blocks* -> passed on to the optimizer
- optimizer:
    - optimize one block at a time
- query *block* - SQL query:
    - without nesting
    - with exactly: one SELECT clause, one FROM clause
    - with at most: one WHERE clause, one GROUP BY clause, one HAVING clause
        - WHERE condition - CNF

Query Blocks – Units of Optimization

- query Q:

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                  S.Age = (SELECT MAX(S2.Age)
                          FROM Students S2)
GROUP BY S.SID
HAVING COUNT(*) > 2
```

nested block

- decompose query into a collection of blocks without nesting

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                  S.Age = Reference to nested block
GROUP BY S.SID
HAVING COUNT(*) > 2
```

Query Blocks – Units of Optimization

* block optimization

- express query block as a relational algebra expression

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                  S.Age = Reference to nested block
GROUP BY S.SID
HAVING COUNT(*) > 2
```

$$\pi_{S.SID,\ MIN(E.EDate)}($$
$$HAVING_{COUNT(*) > 2}($$
$$GROUP\ BY_{S.SID}($$
$$\sigma_{S.SID = E.SID \wedge E.CID = C.CID \wedge C.Description = 'Elective' \wedge S.Age = value\_from\_nested\_block}($$
$$Students \times Exams \times Courses\ ))))$$

- GROUP BY, HAVING – operators in the extended algebra used for plans
- argument list of projection can include aggregate operations

Query Blocks – Units of Optimization

- query Q treated as a $\sigma\ \pi \times$ algebra expression
- the remaining operations in Q are performed on the result of the $\sigma\ \pi \times$ expression

```
SELECT S.SID, MIN(E.EDate)
FROM Students S, Exams E, Courses C
WHERE S.SID = E.SID AND E.CID = C.CID AND C.Description = 'Elective' AND
                  S.Age = Reference to nested block
GROUP BY S.SID
HAVING COUNT(*) > 2
```

$$\pi_{S.SID,\ E.EDate}(\sigma_{S.SID = E.SID \wedge E.CID = C.CID \wedge C.Description = 'Elective' \wedge S.Age = value\_from\_nested\_block}(Students \times Exams \times Courses))$$

- attributes in GROUP BY, HAVING are added to the argument list of projection
- aggregate expressions in the argument list of projection are replaced by their argument attributes

Query Blocks – Units of Optimization

* block optimization

- find best plan P for the $\sigma\ \pi \times$ expression
- evaluate P => result set RS
- sort/hash RS => groups
- apply HAVING to eliminate some groups
- compute aggregate expressions in SELECT for each remaining group

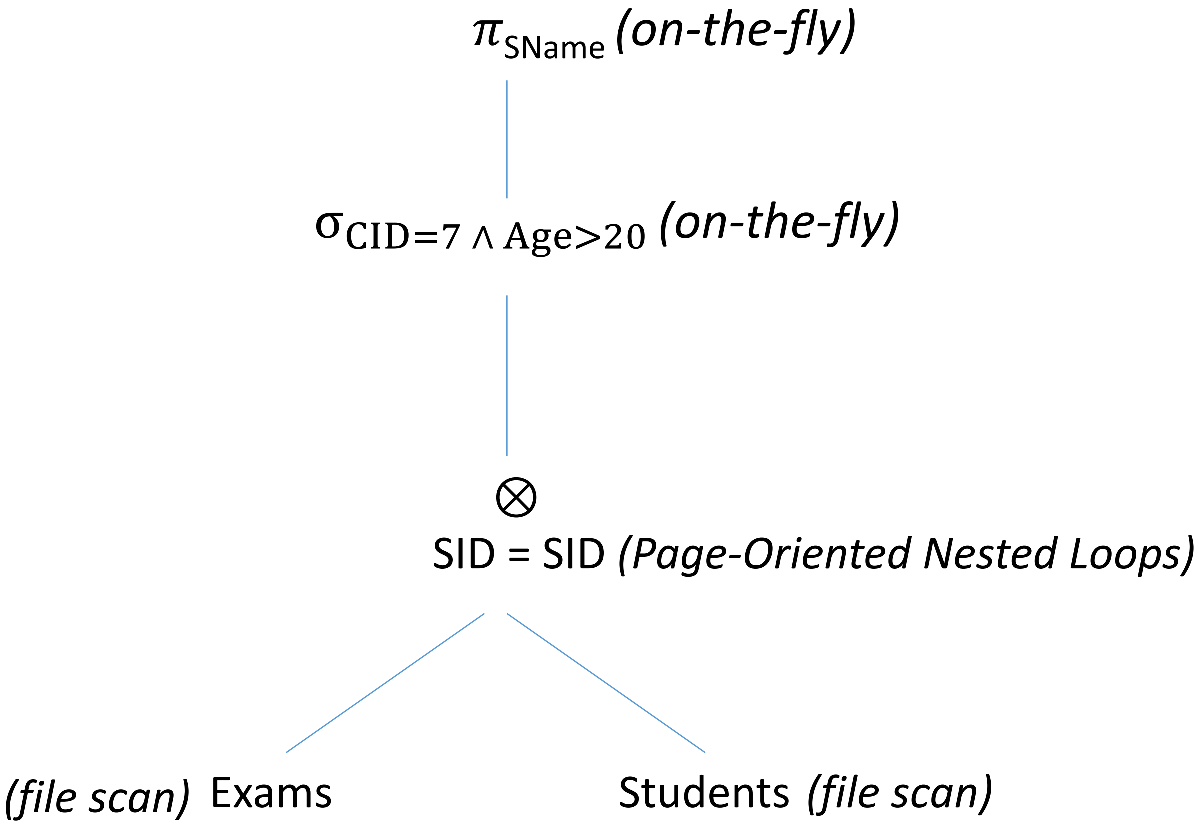
$$
\begin{aligned}
&\pi_{S.SID,\ MIN(E.EDate)}(\\
&HAVING_{COUNT(*) > 2}(\\
&GROUP\ BY_{S.SID}(\\
&\pi_{S.SID,\ E.EDate}(\\
&\sigma_{S.SID = E.SID \wedge E.CID = C.CID \wedge C.Description = 'Elective' \wedge S.Age = value\_from\_nested\_block}(\\
&\qquad Students \times Exams \times Courses)))))
\end{aligned}
$$

Motivating Example

* E - 1000 pages *

* S - 500 pages *

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
     AND S.Age > 20
```

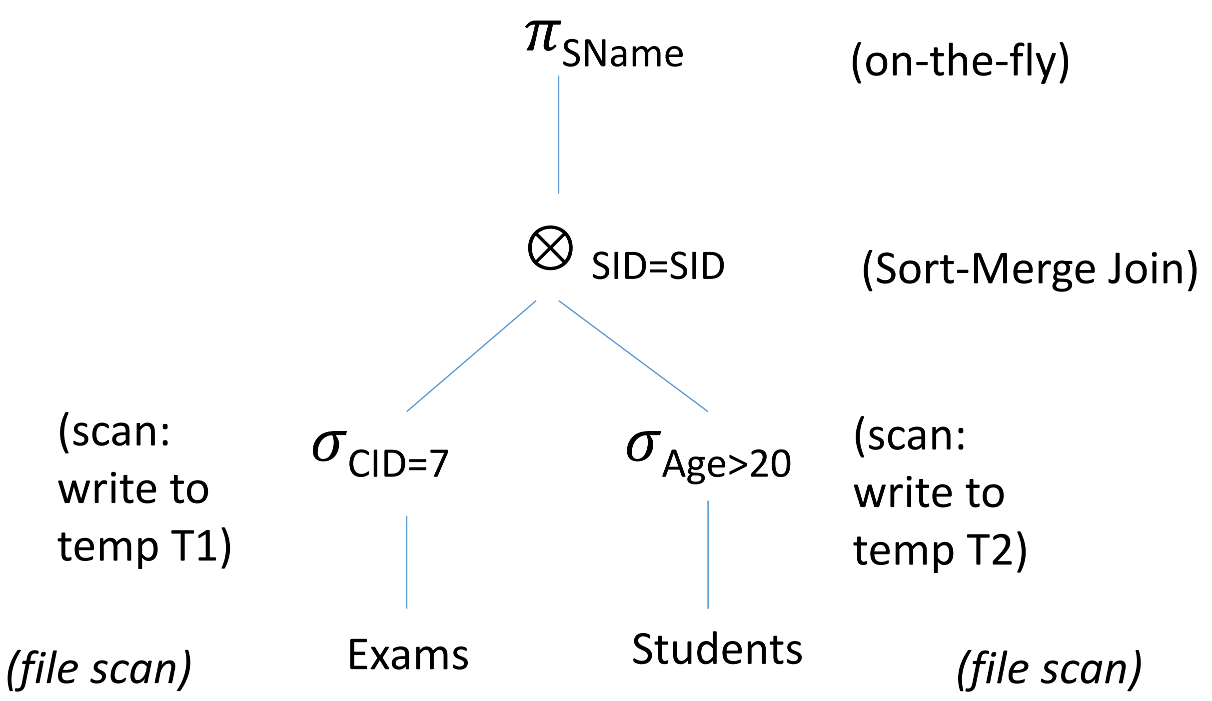


- σ, π – on-the-fly
- cost of plan – very high:
  - 1000 + 1000 * 500 = 501,000 I/Os

Motivating Example

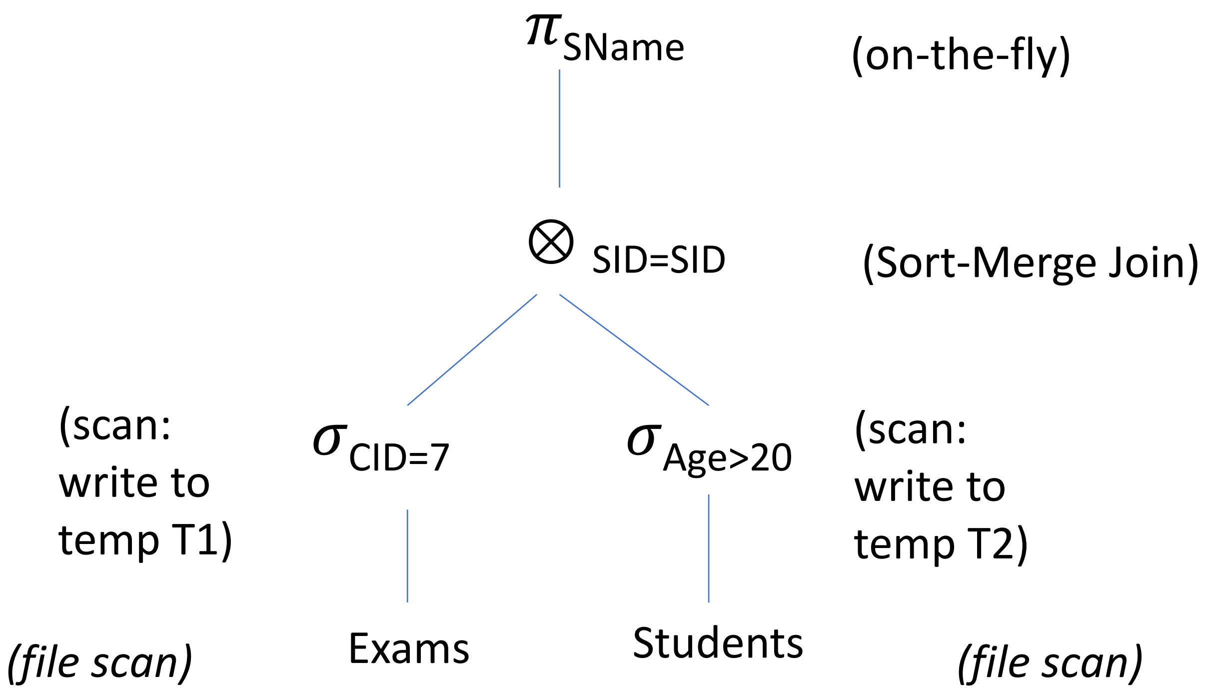
* optimizations

- reduce sizes of the relations to be joined
  - push selections, projections ahead of the join
- alternative plans
  - push selections ahead of joins
- selection
  - file scan
  - write the result to a temporary relation on disk
- join the temporary relations using Sort-Merge Join



Motivating Example

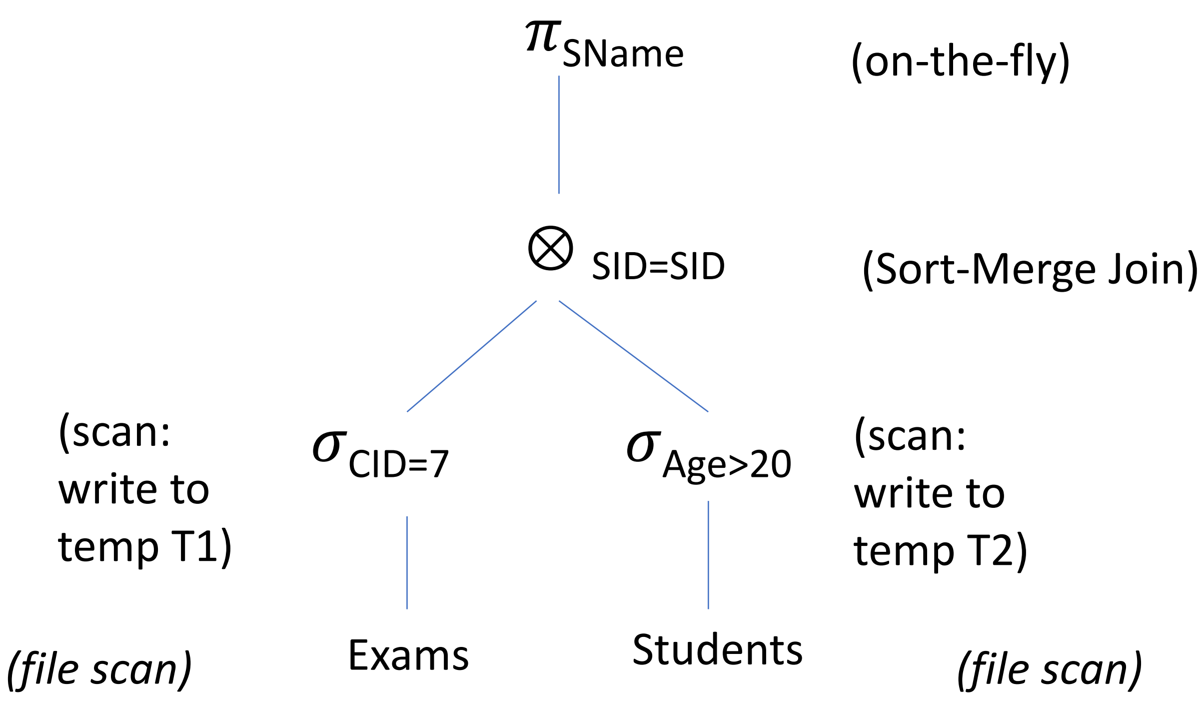
- 5 available buffer pages
- cost
  - $\sigma_{CID=7}$
    - scan Exams: 1000 I/Os
    - write T1
      - assume exams are uniformly distributed across all courses, i.e., T1 has 10 pages (there are 100 courses)
  - $\sigma_{Age>20}$
    - scan Students: 500 I/Os
    - write T2
      - assume ages are uniformly distributed over the range 19 to 22, i.e., T2 has 250 pages



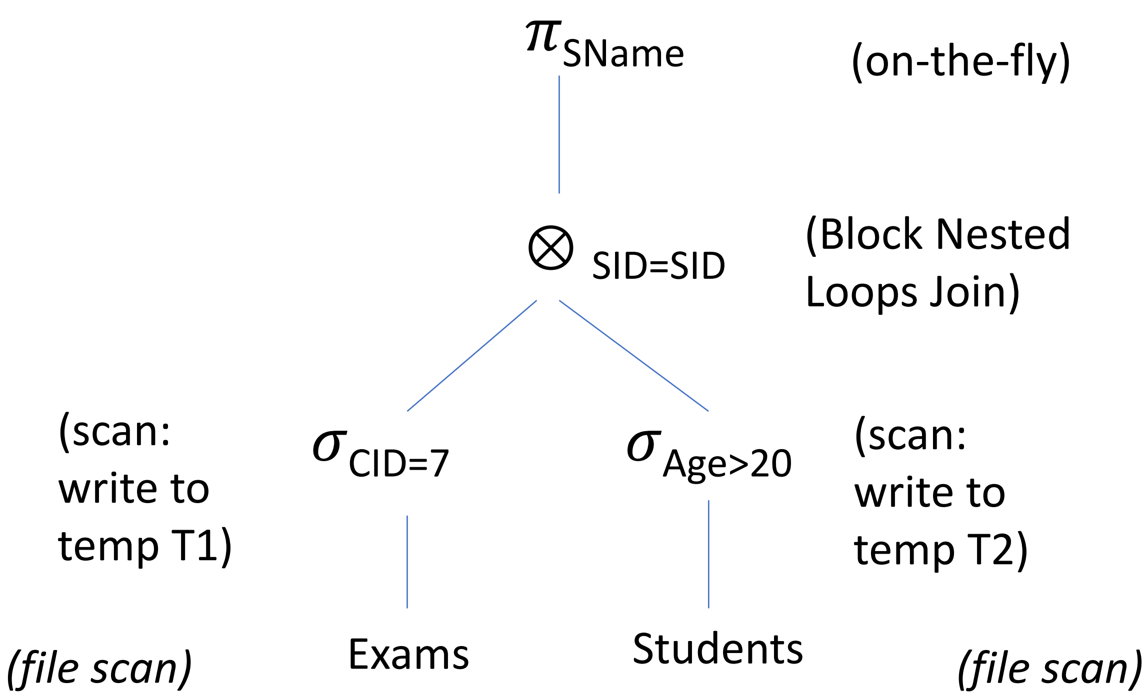
Motivating Example

- 5 available buffer pages
- cost
  - Sort-Merge Join
    - T1 - 10 pages
      - sort T1: 2 * 2 * 10 = 40 I/Os
    - T2 - 250 pages
      - sort T2: 2 * 4 * 250 = 2000 I/Os
    - merge sorted T1 and T2
      - 10 + 250 = 260 I/Os
  - π - on the fly

=> **total cost**: $\underbrace{1000 + 10 + 500 + 250}_{\text{selection}} + \underbrace{40 + 2000 + 260}_{\text{join}} = \mathbf{4060\ I/Os}$



Motivating Example



- 5 available buffer pages
- cost
  - Block Nested Loops Join
    - T1 - 10 pages, T2 - 250 pages
    - T1 - outer relation

      => scan T1: 10 I/Os
    - $\lceil 10/3 \rceil = 4$ T1 blocks

      => T2 scanned 4 times: 4 * 250 = 1000 I/Os
    - BNLJ cost: 10 + 1000 = 1010 I/Os
  - π - on the fly

=> total cost: $\underbrace{1000 + 10 + 500 + 250}_{\text{selection}} + \underbrace{10 + 1000}_{\text{join}} = \mathbf{2770\ I/Os}$

Motivating Example

- push projections ahead of joins
  - drop unwanted columns while scanning Exams and Students to evaluate selections => T1[SID], T2[SID, SName]
- T1 fits within 3 buffer pages

  => T2 scanned only once

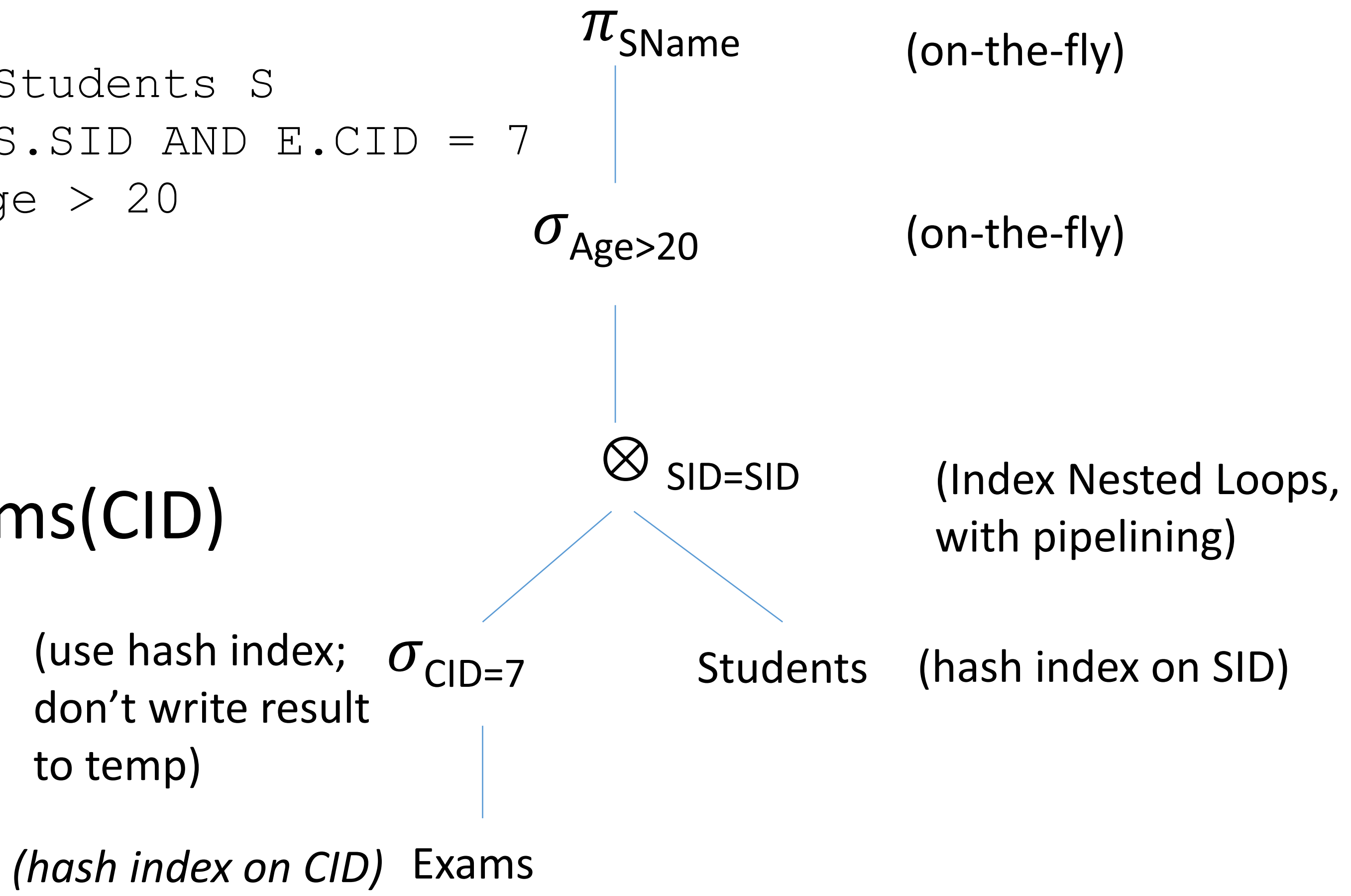
  => **total cost**: about **2000 I/Os**

Motivating Example

```
SELECT S.SName
FROM Exams E, Students S
WHERE E.SID = S.SID AND E.CID = 7
      AND S.Age > 20
```

* optimizations

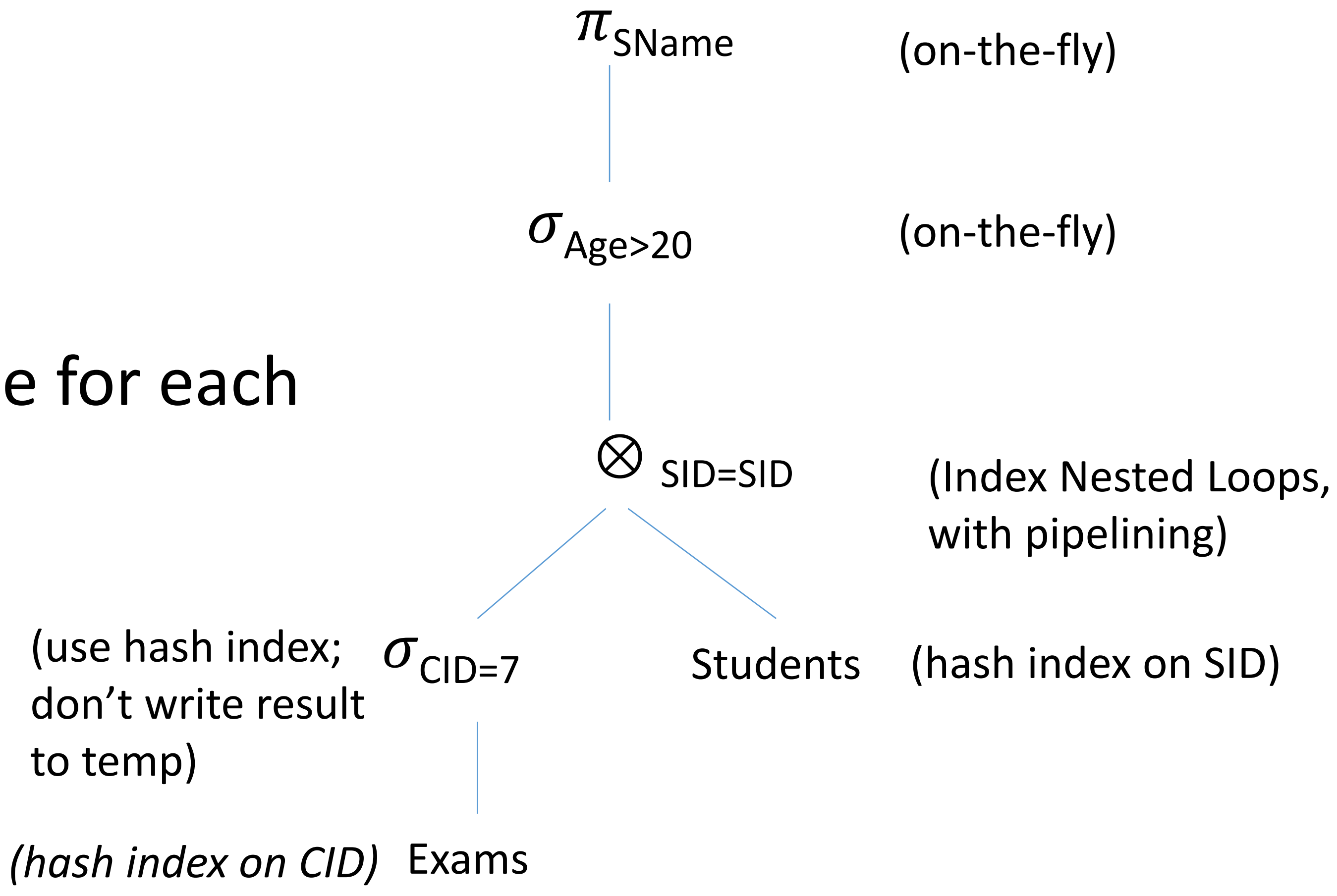
- investigate the use of indexes



- clustered static hash index on Exams(CID)
- hash index on Students(SID)
- cost
  - $\sigma_{CID=7}$
    - assume exams are uniformly distributed across all courses => 100,000 exams / 100 courses => 1,000 exams / course
    - clustered index on CID => 1,000 tuples for course with CID=7 appear consecutively within the same bucket => cost: 10 I/Os
    - the result of the selection is not materialized, the join is pipelined

Motivating Example

- cost
  - Index Nested Loops
    - find matching Students tuple for each selected exam
    - use hash index on SID
      - assume the index uses a1 => cost of 1.2 I/Os (on avg.) per exam



  - σ, π – performed on-the-fly on each tuple in the result of the join

=> **total cost** = $\underbrace{\underline{10}}_{\sigma \text{ on Exams}}$ + $\underbrace{\underline{1000}}_{\text{num. of Exams tuples}}$ * $\underbrace{\underline{1.2}}_{\text{find matching Students tuple (on avg.)}}$ = **1210** I/Os

* can we push the selection *Age>20* ahead of the join?

- running example - schema
  - Students (SID: integer, SName: string, Age: integer, RoundedGPA: integer)
  - Courses (CID: integer, CName: string, Description: string)
  - Exams (SID: integer, CID: integer, EDate: date, Grade: integer)

  - Students
    - every record has 50 bytes
    - there are 80 records / page
    - 500 pages
  - Courses
    - every record has 40 bytes
    - there are 100 records / page
    - 1 page
  - Exams
    - every record has 40 bytes
    - there are 100 records / page
    - 1000 pages

IBM's System R Optimizer

- tremendous influence on subsequent relational optimizers
- design choices:
    - use statistics to estimate the costs of query evaluation plans
    - consider only plans with binary joins in which the inner relation is a base relation
    - focus optimization on SQL queries without nesting
    - don't eliminate duplicates when performing projections (unless DISTINCT is used)

Estimating the Cost of a Plan

- estimating the cost of an evaluation plan for a query block
  - for each node N in the tree:
    - estimate the cost of the corresponding operation (pipelining versus temporary relations)
    - estimate the size of N's result and whether it is sorted
      - N's result is the input of N's parent node
      - these estimates affect the estimation of cost, size, and sort order for N's parent

Estimating the Cost of a Plan

- estimating costs
    - use data about the input relations (such statistics are stored in the DBMS's system catalogs)
        - number of pages, existing indexes, etc.
- obtained estimates are at best approximations to actual sizes and costs

=> one shouldn't expect the optimizer to find the best possible plan

- optimizer - goals:
    - avoid the worst plans
    - find a good plan

Statistics Maintained by the DBMS

- updated periodically, not every time the data is changed
  - relation R
    - cardinality - *NTuples(R)*
      - the number of tuples in R
    - size - *NPages(R)*
      - the number of pages in R
  - index I
    - cardinality - *NKeys(I)*
      - the number of distinct key values for I
    - size - *INPages(I)*
      - the number of pages for I
      - B+ tree index
        - number of leaf pages

Statistics Maintained by the DBMS

- index I
  - height - *IHeight(I)*
    - maintained for tree indexes
    - the number of nonleaf levels in I
  - range - *ILow(I), IHigh(I)*
    - the minimum / maximum key value in I

Estimating Result Sizes

- query Q

  ```
  SELECT attribute list
  FROM relation list
  WHERE term_1 ∧ ... ∧ term_k
  ```

- the maximum number of tuples in Q's result:
  - $\prod |R_i|$

  where $R_i \in$ `relation list`
- each $term_j$ in the WHERE clause eliminates some candidate tuples
  - associate a reduction factor RF_j with each term $term_j$
  - RF_j models the impact $term_j$ has on the result size
- estimate the actual size of the result:
  - $\prod |R_i| * \prod RF_j$
  - i.e., the maximum result size times the product of the reduction factors for the terms in the WHERE clause

Estimating Result Sizes

- query Q

```
SELECT attribute list
FROM relation list
WHERE term₁ ∧ ... ∧ termₖ
```

- assumption
  - the conditions tested by the terms in the WHERE clause are statistically independent

Estimating Result Sizes

- compute reduction factors for terms in the WHERE clause
- assumptions:
  - uniform distribution of values
  - independent distribution of values in different columns

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```

- *column = value*
  - index I on *column*

  => RF approximated by 1/NKeys(I)
  - no index on *column*

  => RF: 1/10
    - maintain statistics on *column* (e.g., number of distinct values in *column*) to obtain a better value

Estimating Result Sizes

- *column1 = column2*
  - indexes *I1* on *column1*, *I2* on *column2*

  => RF: 1/MAX(NKeys(I1), NKeys(I2))
  - only one index I (on one of the 2 columns)

  => RF: 1/NKeys(I)
  - no indexes

  => RF: 1/10
- *column > value*
  - index I on *column*

  => RF: (IHigh(I) - *value*) / (IHigh(I) - ILow(I))
  - no index on *column* or *column* not of an arithmetic type

  => a value less than 0.5 is arbitrarily chosen
  - similar formulas can be obtained for other range selections

Estimating Result Sizes

- *column IN (list of values)*
  => RF: (RF for *column = value*) * number of items in list (but at most 0.5)
- *NOT condition*
  => RF: 1 - RF for *condition*

- obtain better estimates
  - use more detailed statistics (e.g., histograms of the values in a column)

Relational Algebra Equivalences

- central role in generating alternative plans
- different join orders can be considered
- selections, projections can be pushed ahead of joins
- cross-products can be converted to joins

- selections
  - *cascading selections*
    - $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$
  - *commutativity*
    - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$
- projections
  - *cascading projections*
    - $\pi_{a1}(R) \equiv \pi_{a1}(\pi_{a2}(\ldots(\pi_{an}(R))\ldots))$
    - a_i – set of attributes in R
    - $a_i \subseteq a_{i+1}$, for i = 1..n-1

Relational Algebra Equivalences

- joins and cross-products
  - assumption
    - fields are identified by their name, not by their position
  - *associativity*
    - $R \times (S \times T) \equiv (R \times S) \times T$
    - $R * (S * T) \equiv (R * S) * T$
  - *commutativity*
    - $R \times S \equiv S \times R$
    - $R * S \equiv S * R$
    - can choose the inner / outer relation in a join

Relational Algebra Equivalences

- joins and cross-products
  - e.g., check that R * (S * T) ≡ (T * R) * S
    - commutativity
      - R * (S * T) ≡ R * (T * S)
    - associativity
      - R * (T * S) ≡ (R * T) * S
    - commutativity
      - (R * T) * S ≡ (T * R) * S

Relational Algebra Equivalences

- can commute σ with π if σ uses only attributes retained by π
  - $\pi_a(\sigma_c(R)) \equiv \sigma_c(\pi_a(R))$
- can combine σ with $\times$ to form a join
  - $R \bigotimes_c S \equiv \sigma_c(R \times S)$
- can commute σ with $\times$ or a join when the selection condition includes only fields of one of the arguments (to the cross-product or join)
  - for instance:
    - $\sigma_c(R * S) \equiv \sigma_c(R) * S$
    - $\sigma_c(R \times S) \equiv \sigma_c(R) \times S$
      - condition c must include only fields from R
- in general: $\sigma_c(R \times S) \equiv \sigma_{c1}(\sigma_{c2}(R) \times \sigma_{c3}(S))$
    - c1 – attributes of both R and S
    - c2 – only attributes of R
    - c3 – only attributes of S

Relational Algebra Equivalences

- can commute π with $\times$
  - $\pi_a(R \times S) \equiv \pi_{a1}(R) \times \pi_{a2}(S)$
  - a1 – attributes in *a* that appear in R
  - a2 – attributes in *a* that appear in S
- can commute π with join
  - $\pi_a(R \otimes_c S) \equiv \pi_{a1}(R) \otimes_c \pi_{a2}(S)$
    - every attribute in *c* must appear in *a*
    - a1 – attributes in *a* that appear in R
    - a2 – attributes in *a* that appear in S
  - *a* doesn't contain all the attributes in *c* – generalization
    - eliminate unwanted fields, compute join, eliminate fields not in *a*
      - $\pi_a(R \otimes_c S) \equiv \pi_a(\pi_{a1}(R) \otimes_c \pi_{a2}(S))$
      - a1 – attributes of R that appear in either *a* or *c*
      - a2 – attributes of S that appear in either *a* or *c*

References

- [Ra02] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems (3rd Edition), McGraw-Hill, 2002
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Ga09] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book (2nd Edition), Pearson Education, 2009
- [Ra02S] RAMAKRISHNAN, R., GEHRKE, J., Database Management Systems, Slides for the 3rd Edition, http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- [Si19] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts (7th Edition), McGraw-Hill, 2019
- [Si19S] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts, Slides for the 7th Edition, http://codex.cs.yale.edu/avi/db-book/
- [Ul11] ULLMAN, J., WIDOM, J., A First Course in Database Systems, http://infolab.stanford.edu/~ullman/fcdb.html