Query Optimization

CSC 321/621 – 4/19/2012

Approaches to Query Optimization

- Will examine heuristics (general approaches/good ideas)
- If have time, will look at cost analysis and optimization
  - Need statistics to be used effectively

- General goal (should sound familiar):
  - Reduce disk accesses
  - Reduce tuple touches

A Motivating Example

"Find all managers who work at the London Branches"

```
SELECT * FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
(s.position='Manager' AND b.city='London')
```

Motivating Example

SELECT * FROM Staff s, Branch b

WHERE s.branchNo = b.branchNo AND (s.position='Manager' AND b.city='London')

Equivalent relational algebra expressions:

(1) $\sigma_{(\text{position}=\text{'Manager'}) \wedge (\text{city}=\text{'London'}) \wedge (\text{Staff.branchNo}=\text{Branch.branchNo})}(\text{Staff} \times \text{Branch})$

(2) $\sigma_{(\text{position}=\text{'Manager'}) \wedge (\text{city}=\text{'London'})}(\text{Staff} \bowtie_{\text{Staff.branchNo}=\text{Branch.branchNo}} \text{Branch})$

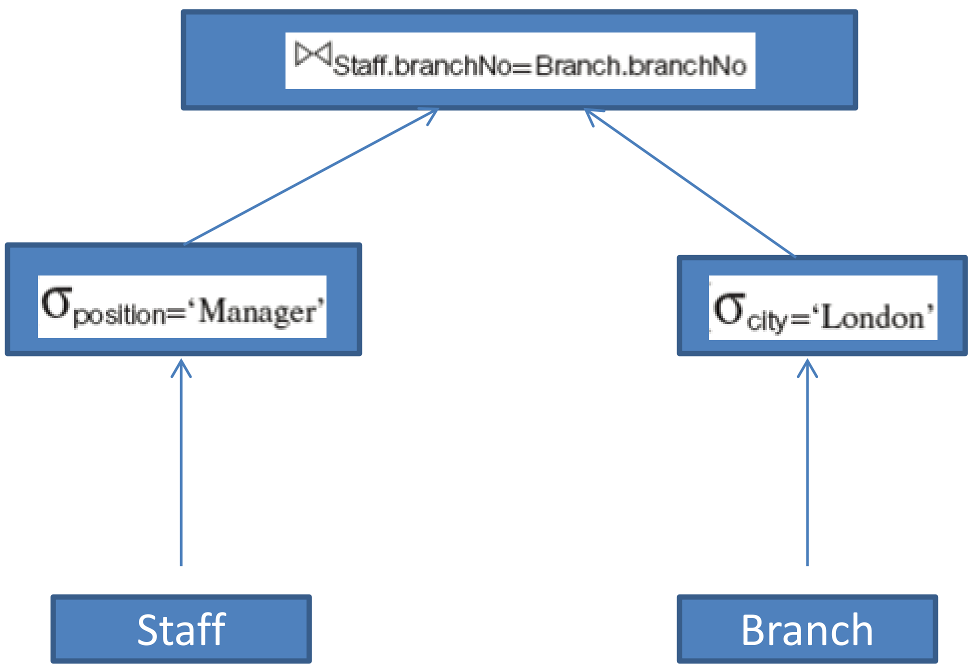
(3) $(\sigma_{\text{position}=\text{'Manager'}}(\text{Staff})) \bowtie_{\text{Staff.branchNo}=\text{Branch.branchNo}} (\sigma_{\text{city}=\text{'London'}}(\text{Branch}))$

Remember these required around 100,000 operations, 3000, and 1000 respectively.
The first is the natural translation from the SQL.

Query Optimization

- When a query is parsed, we will assume that it is turned into a tree representation

$$(\sigma_{\text{position}=\text{'Manager'}}(\text{Staff})) \bowtie_{\text{Staff.branchNo}=\text{Branch.branchNo}} (\sigma_{\text{city}=\text{'London'}}(\text{Branch}))$$

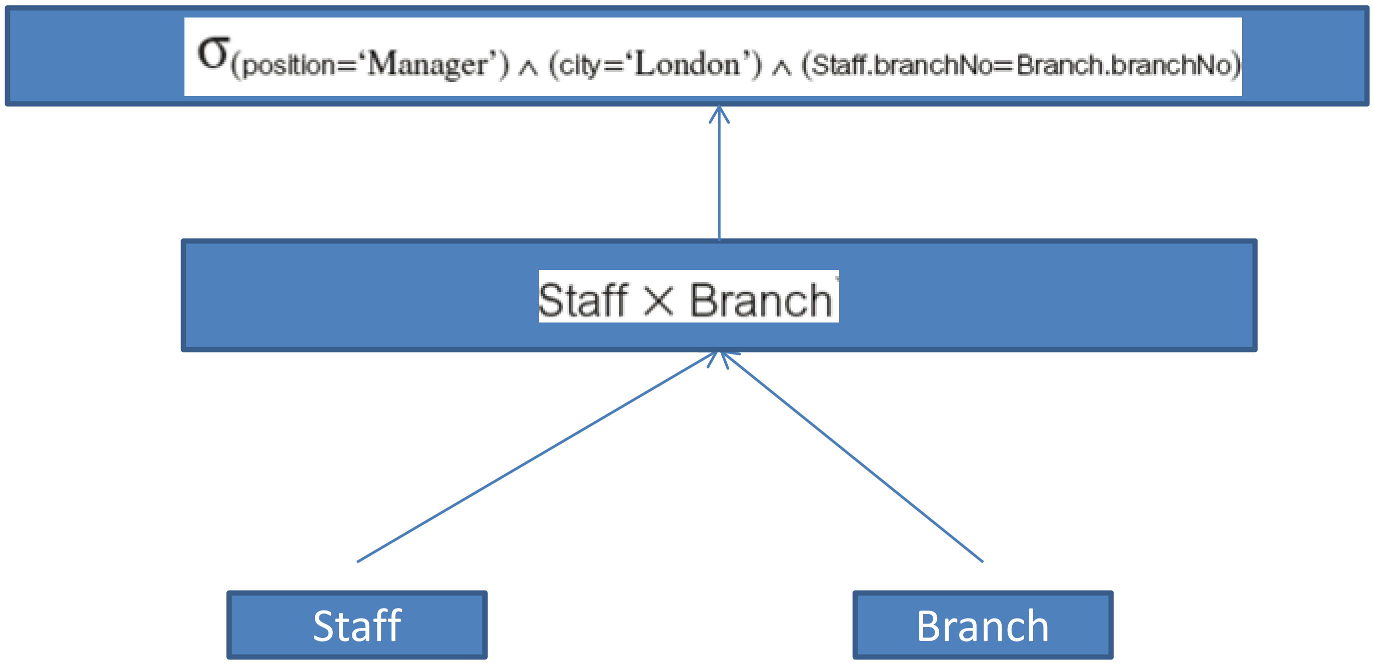


What is set of valid tree reorderings?

Query Optimization

- What is appropriate representation for:

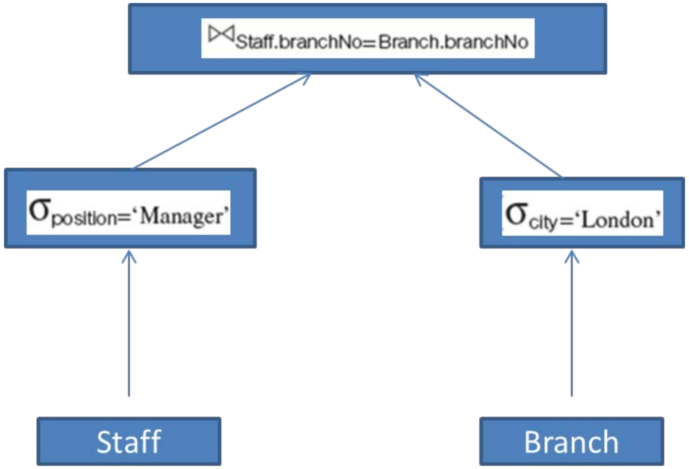
$$\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)}(Staff \times Branch)$$



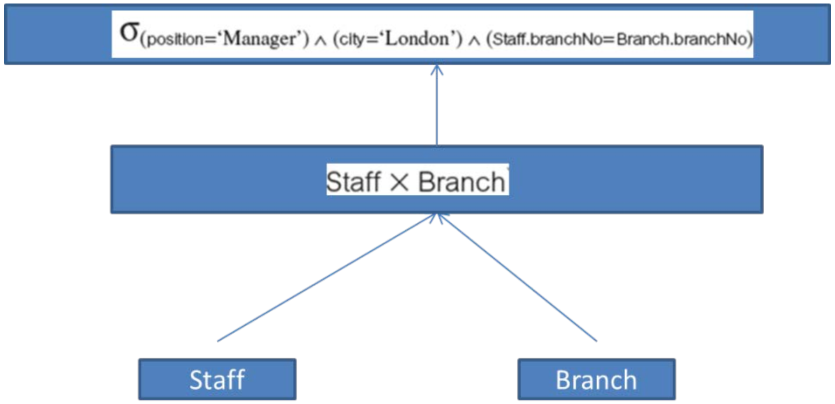
Query Optimization: General Heuristics

- Using a set of transformation rules (forthcoming on next few slides), want to attempt to reshape query as follows:
  - Filter early:
    - Perform selection as early as possible
    - Use associativity of binary operations (binary = joins, union, intersection, etc) where possible to attempt to perform most restrictive selection operations first
    - Perform projections as early as possible
  - Reduce number and size of intermediate forms
    - Convert Cartesian products followed by selection into Theta-joins
  - Compute common expressions once

Query Optimization: General Heuristics



Compare these two w/in context of previous slide



Query Optimization

- Transformation rules:
  - Conjunctive ("and-ed") selection of predicates can be cascaded into multiple nested selection statements

$$\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$$

  - Selection is commutative

$$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$$

Query Optimization

- Transformation rules:
  - If there are a series of projections, it can be reduced to just the last projection

$$\Pi_L \Pi_M \ldots \Pi_N(R) = \Pi_L(R)$$

  - If the selection predicate only involves attributes in the projection list, projection and selection are commutative

$$\Pi_{A_1, \ldots, A_m}(\sigma_p(R)) = \sigma_p(\Pi_{A_1, \ldots, A_m}(R)) \qquad \text{where } p \in \{A_1, A_2, \ldots, A_m\}$$

This one should feel familiar….

Query Optimization

- Transformation rules:
  - If the selection predicate involves only attributes of one of the relations being joined, the Selection and Join operations commute (also holds for CP)

$$\sigma_p(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r S$$

  - If the selection predicate involves a conjuction p^q and p are limited to attributes of relation R and q are limited to attributes of relation S, the Selection and Join operations commute

$$\sigma_{p \wedge q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$$

Query Optimization

- Transformation rules:
  - If the projection list is of the form A=A1 U A2, where A1 is limited to a set of attributes from R and A2 is a set of attributes from S, Projection and Join commute if the join condition only deals with attributes from A

$$\Pi_{L_1 \cup L_2}(R \bowtie_r S) = (\Pi_{L_1}(R)) \bowtie_r (\Pi_{L_2}(S))$$

  - If the Join condition deals with additional attributes not in A, and those also can be divided between relations (say sets B1, B2), then the following holds:

$$\Pi_{L_1 \cup L_2}(R \bowtie_r S) = \Pi_{L_1 \cup L_2}(\Pi_{L_1 \cup M_1}(R)) \bowtie_r (\Pi_{L_2 \cup M_2}(S))$$

Query Optimization

- We can reorder the following operations:
  - Theta join (any join), Cartesian product, Union, Intersection

$$R \bowtie_p S = S \bowtie_p R \qquad R \cup S = S \cup R$$

$$R \times S = S \times R \qquad R \cap S = S \cap R$$

- Selection and set operations are commutative

$$\sigma_p(R \cup S) = \sigma_p(S) \cup \sigma_p(R)$$

$$\sigma_p(R \cap S) = \sigma_p(S) \cap \sigma_p(R)$$

$$\sigma_p(R - S) = \sigma_p(S) - \sigma_p(R)$$

Query Optimization

- Projection and Union are commutative

$$\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$$

- Cartesian product and natural join are always associative

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$(R \times S) \times T = R \times (S \times T)$$

- Theta join sometimes is, given q that only is attributes from S and T

$$(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$$

Query Optimization

- Union and Intersection are associative

$$(R \cup S) \cup T = S \cup (R \cup T)$$

$$(R \cap S) \cap T = S \cap (R \cap T)$$

Query Optimization Question

- We saw "Projection and Union are commutative"

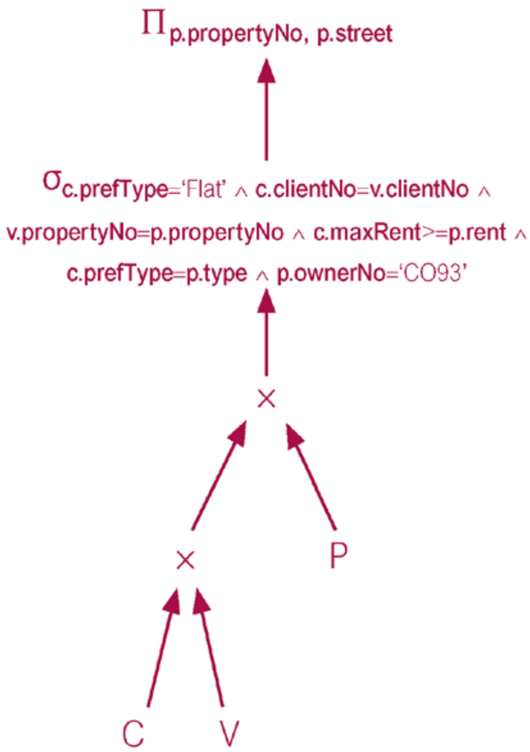
$$\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$$

- *Why is "Projection and Intersection are commutative" not true?*

Use of Transformation Rules

For prospective renters of flats, find properties that match requirements and owned by CO93.

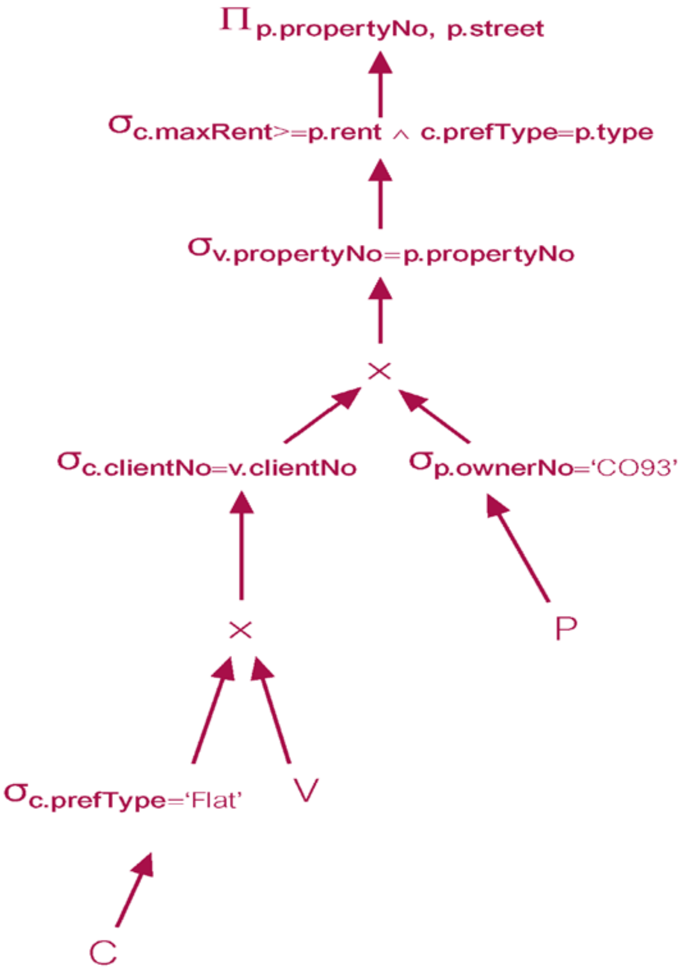
```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE   c.prefType = 'Flat' AND
            c.clientNo = v.clientNo AND
            v.propertyNo = p.propertyNo AND
            c.maxRent >= p.rent AND
            c.prefType = p.type AND
            p.ownerNo = 'CO93';
```



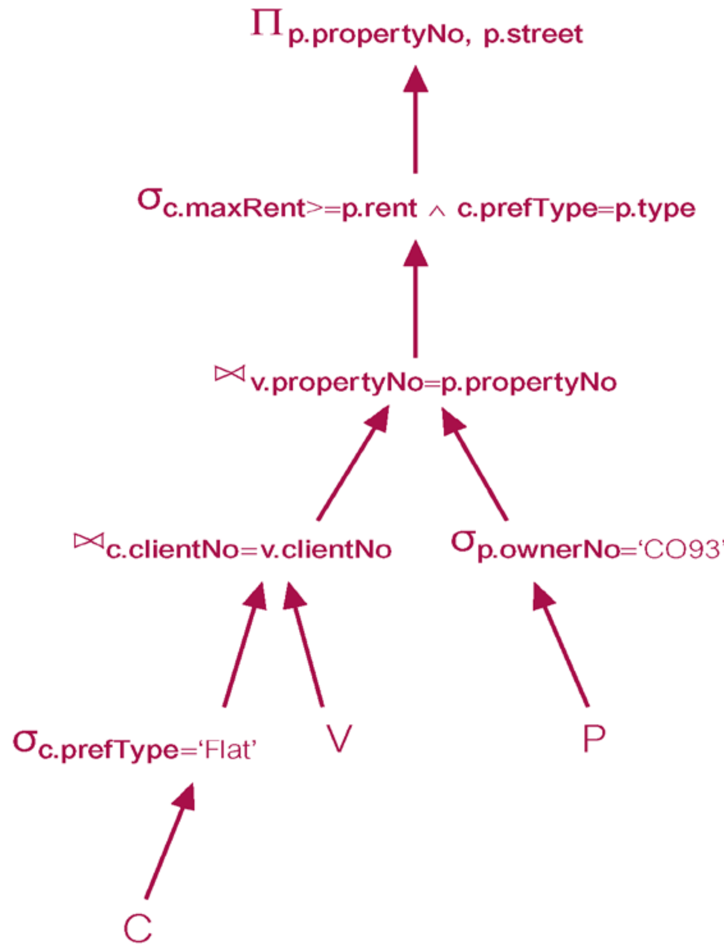
(a)

Is this the appropriate tree given the query?

Use of Transformation Rules



(b)

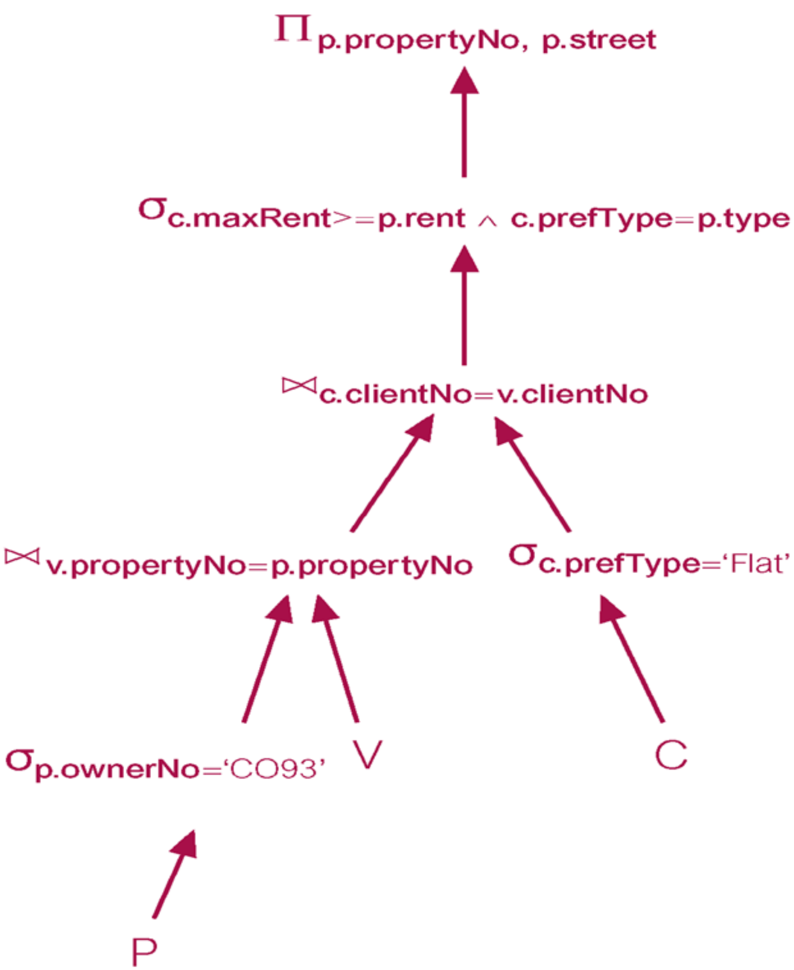


(c)

Cascaded and pushed selection down
(via CP, selection commutativity) as far as possible…

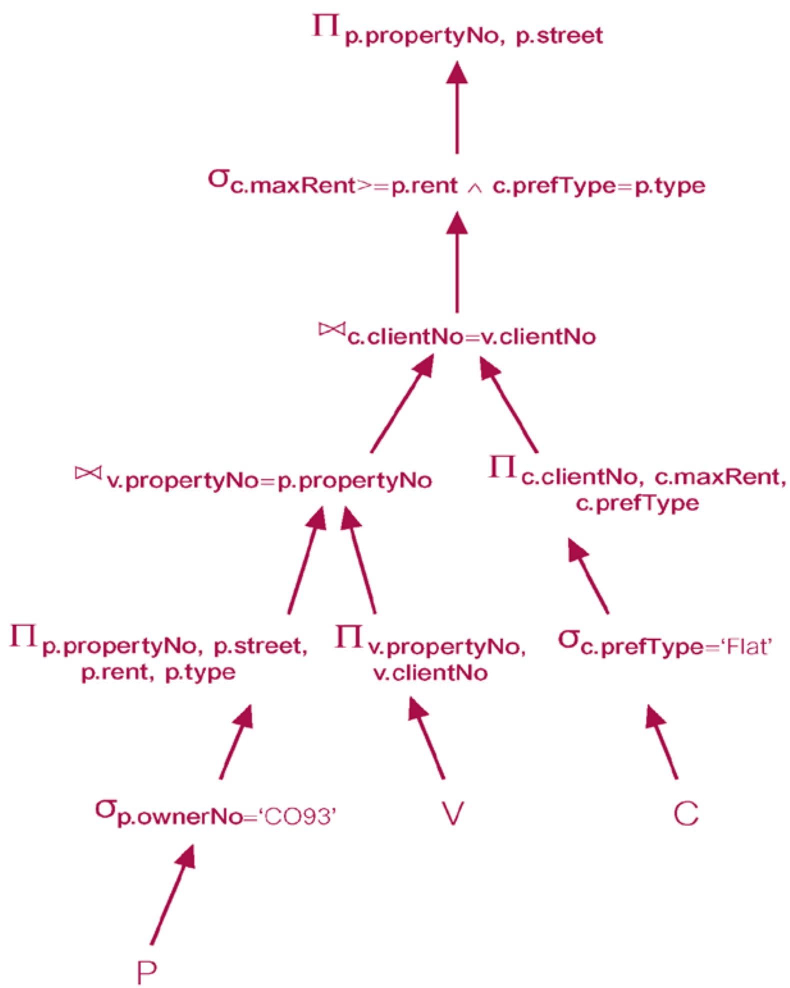
What did we do here?

Use of Transformation Rules



(d)

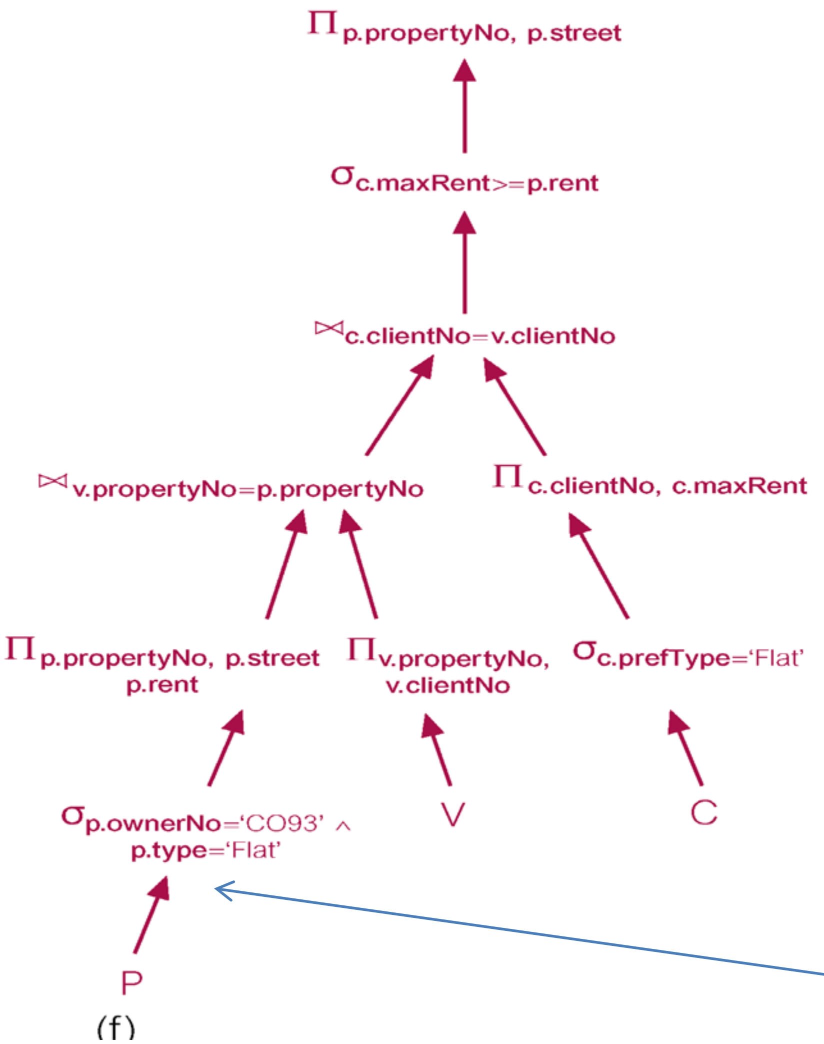
Associativity of theta-join if certain constraints met



(e)

What did we do here?

Use of Transformation Rules



 Exploited that there was a fixed prefType choice to push a predicate down tree

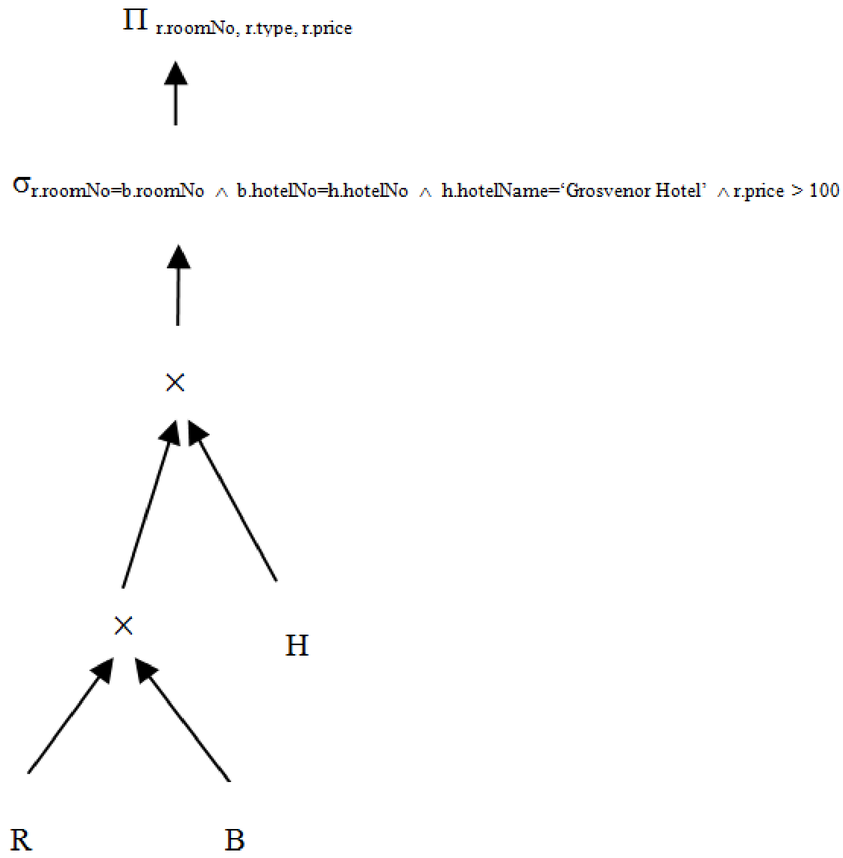
Transformation Example 2

Given this query, suggest the initial Relational Algebra tree, then a tree improved by heuristics optimizations.

SELECT r.roomNo, r.type, r.price FROM Room r, Booking b, Hotel h WHERE r.roomNo=b.roomNo AND b.hotelNo=h.hotelNo AND h.hotelName='Grosvenor Hotel' AND r.price > 100;

Transformation Example 2

SELECT r.roomNo, r.type, r.price FROM Room r, Booking b, Hotel h WHERE r.roomNo=b.roomNo AND b.hotelNo=h.hotelNo AND h.hotelName=‘Grosvenor Hotel’ AND r.price > 100;

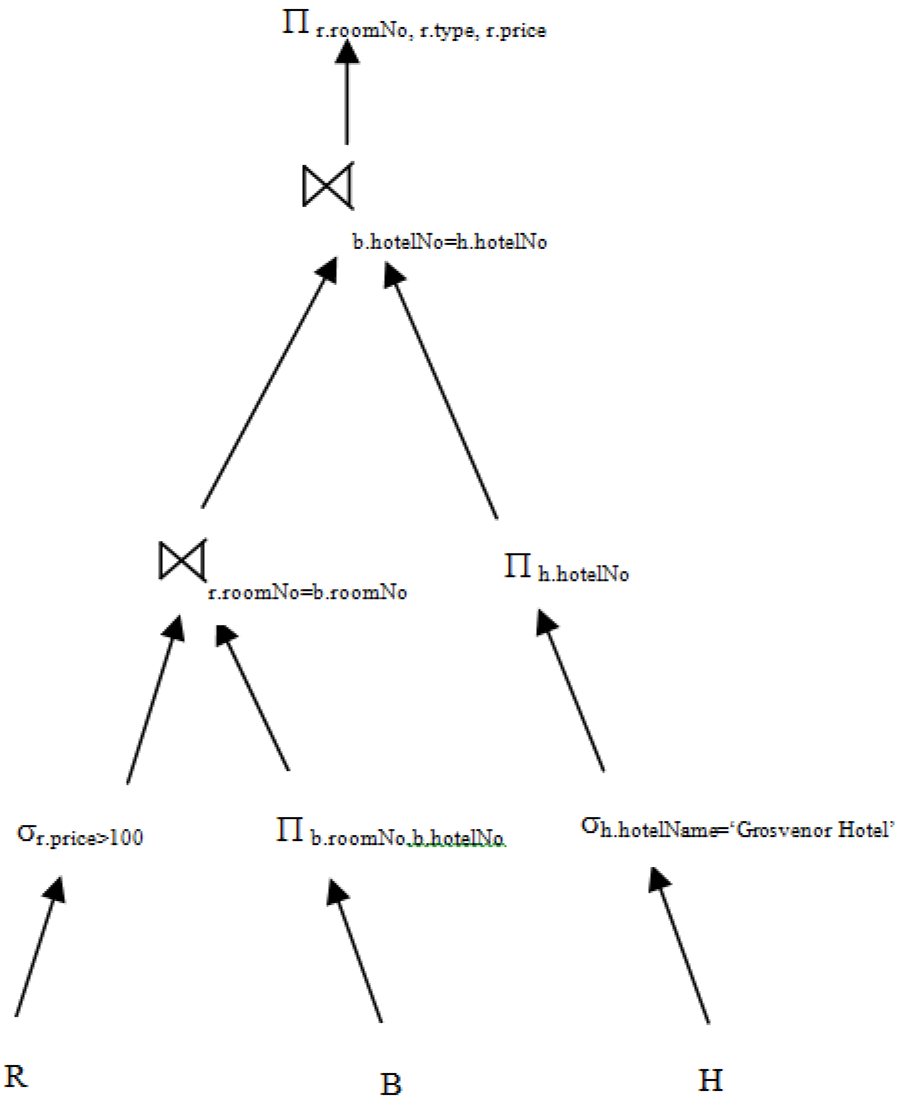


Initial

Transformation Example 2

SELECT r.roomNo, r.type, r.price FROM Room r, Booking b, Hotel h WHERE r.roomNo=b.roomNo AND b.hotelNo=h.hotelNo AND h.hotelName='Grosvenor Hotel' AND r.price > 100;

Optimized



Cost Estimation

- A DBMS may be able to implement a given RA operator in multiple ways – wants to pick the most efficient
  - So, we are diving down another level of concreteness

- Costs are based again on disk accesses

Database Statistics

- Assume that the DBMS maintains the following statistics:
  - For relation R,
    - nTuples(R) –> cardinality (# of tuples)
    - bFactor(R) –> blocking factor (# of tuples that fit in one page)
    - nBlocks(R) –> nTuples(R)/bFactor(R)

  - For multilevel index I on attribute A of R,
    - $nLevels_A(I)$ –> number of levels in index
    - $nLeafBlocks_A(I)$ –> number of leaf blocks in index

Database Statistics

- For attribute A of relation R,
  - $nDistinct_A(R)$ –> number of distinct values that appear for attribute
  - $min_A(R)$, $max_A(R)$ –> min and max possible values for the attribute
  - $SC_A(R)$ – selection cardinality of A – average number of tuples that satisfy an equality condition on attribute A
    - 1 if A is a primary key field
    - $nTuples(R)/nDistinct_A(R)$, assuming uniform distribution
    - We can estimate this for non equality tests as well (inequality, in set, AND, OR)…

Database Statistics

- Statistics maintenance requires too much overhead to do dynamically as inserts/updates/deletes occur, so often computed during periods of low activity in the DBMS

- Costs directly depend on:
  - Physical representation of relation (including presence of indices)
  - The algorithm we employ to work with that physical representation (ideally the best one)
- Want to choose a strategy with minimal costs

Cost Analysis of Selection Operation

- Selection:
  - Works on a single relation
  - May involve comparison of one attribute value to a constant or another attributes value
  - May employ a composite predicate (using AND, OR, NOT)

- Costs here also affected by the predicate (equality, inequality)

Cost Analysis of Selection Operation

- The book lists 9 different strategies for implementing selection!

- We will look at a few, starting with single comparisons (equality or inequality).
- As a warm-up, consider:
  - Equality condition on a KEY value, unordered file, no index
    - On average, **nBlocks / 2** – why?

  - Costs are disk-access based (so block-based)
  - Only looking for one item
  - By chance, sometimes find it as first tuple, sometimes as last, so on average over time, visit half the tuples (so half the blocks)

Cost Analysis of Selection Operation

- Inequality condition or equality on non-key, unordered file, no index
  - **nBlocks(R)** (can't stop early)

- Binary search, ordered file, no index
  - Equality on KEY attribute: **log2(nBlocks(R))**
  - Equality on non-key: **log2(nBlocks(R)) + $\mathbf{SC_A}$(R)/bFactor(R) – 1**
    - Find the first instance, then the rest are nearby (in order); expect there to be **$\mathbf{SC_A}$(R)** instances, which live in some number of blocks, already found one block in initial binary search
  - Inequality: **log2(nBlocks(R)) + nBlocks(R)/2**

Cost Analysis of Selection Operation

- Equality, using B+-index on primary key (primary index)
  - $nLevels_A(I) + 1$

- Inequality (A.value > x), using B+-index on primary key
  - $nLevels_A(I)$ to get to x, then using uniform distribution, on average expect ½ items to meet inequality so

    $$nLevels_A(I) + nBlocks(R)/2$$

Cost Analysis of Selection Operation

- Equality, using B+-secondary index
  - $nLevels_A(I) + SC_A(R)$ (get to tuple, then scan over items; since not primary key, might be multiple instances of same value for attribute, items may be in different blocks)
- Inequality (A.value > x), using B+-secondary index
  - $nLevels_A(I) + nLeafBlocks_A(I)/2 + nTuples(R)/2$

Why this value when index on primary key was just $nLevels_A(I) + nBlocks(R)/2$?

Previous slide could exploit sorted nature of primary index to just deal with data once found first tuple instead of having to deal with scanning index

Cost Analysis of Selection: Composite Attributes

- Conjunctive – AND only: Collecting tuples that satisfy all attribute tests
- Disjunctive – At least one OR, collecting tuples that satisfy any of the attributes
  - Worst cases:
    - AND: No indices at all
      - If an index exists on at least one attribute, use it to retrieve tuples and then spot check match other attribute values
    - OR: No index on one
      - Prevents indexing each and unioning
  - Both devolve into linear search on one attribute in each case, with spot-checking tuples as visit → nBlocks(R)

Cost Analysis of Selection Example

Using the Hotel schema, assume the following indexes exist:

- a hash index with no overflow on the primary key attributes, roomNo/hotelNo in Room;
- a clustering index on the foreign key attribute hotelNo in Room;
- a B^+-tree index on the price attribute in Room;
- a secondary index on the attribute type in Room.

nTuples(Room)	= 10,000	bFactor(Room)	= 200
nTuples(Hotel)	= 50	bFactor(Hotel)	= 40
nTuples(Booking)	= 100,000	bFactor(Booking)	= 60
$nDistinct_{hotelNo}$(Room)	= 50		
$nDistinct_{type}$(Room)	= 10		
$nDistinct_{price}$(Room)	= 500		
min_{price}(Room)	= 200	max_{price}(Room)	= 50
$nLevels_{hotelNo}$(I)	= 2		
$nLevels_{price}$(I)	= 2	$nLfBlocks_{price}$(I)	= 50

Cost Analysis of Selection Example

- What are costs for reasonable (optimal) strategies for following operations and worst-case (linear search) strategies?

$$\sigma_{\text{roomNo}=1 \wedge \text{hotelNo}=\text{'H001'}}(\text{Room})$$

$$\sigma_{\text{hotelNo}=\text{'H002'}}(\text{Room})$$

$$\sigma_{\text{price}>100}(\text{Room})$$

Cost Analysis of Selection Example

- What are costs for reasonable (optimal?) strategies for following operations and worst-case (linear search) strategies?

$$\sigma_{roomNo=1 \wedge hotelNo='H001'}(Room)$$

Optimal: There is hash index on those two attributes, so 1 step to get to tuple.
Linear search: Number of rooms/block size / 2 (on average) → 10,000/200/2 →25

Cost Analysis of Selection Example

- What are costs for reasonable (optimal?) strategies for following operations and worst-case (linear search) strategies?

$$\sigma_{hotelNo=\text{'H002'}}(\text{Room})$$

Optimal: 2 levels of index, it is a clustering index, so need to iterate over items
Assume 10,000 rooms / 50 hotels = 200 rooms/ hotel; bFactor for rooms is 200, so requires 1 page accessed → 3 total pages accessed
Linear search: There are 10,000/200 pages total of rooms → 50 pages – have to traverse all

Cost Analysis of Selection Example

- What are costs for reasonable (optimal?) strategies for following operations and worst-case (linear search) strategies?

$$\sigma_{price>100}(\text{Room})$$

Optimal?: There is a B+ index for this, on a non-key attribute (a secondary index).
This is inequality analysis: $nLevels_A(I) + nLeafBlocks_A(I)/2 + nTuples(R)/2$ ➔
 $2 + 50/2 + 10000/2$ ➔ 5027

Linear search: Actually better in this case! 10,000 pages / 200 rooms per page == 50

Cost Analysis of Projections

- Implementation of projections is essentially:
  - Removing non-desired columns
  - Removing duplicates

Which do you think is the more complex component?

Removing duplicates (but only if no key attribute in the projection)

Cost Analysis of Projections

- To drop unwanted columns from R,
  - Read all tuples of R, make new relation with only desired columns → nBlocks(R) to read from disk
- Duplicates are typically eliminated by *sorting* or *hashing*
  - *Sorting:* Sort set of tuples, using combination of fields to sort on → nBlocks(R) * [log2(nBlocks(R))]
    - That sort key is nontrivial in itself (up to |A| comparisons)
  - *Hashing:* 2*nBlocks(R)
    - Hash tuples (by combination of attributes) into blocks (writing)
    - Only possible equal items are in same block
    - For each hash-target-block independently, read and rehash items with different hash function; if have a collision, check for equality and remove one if equal
    - Ultimately, dealing with each block 2x here (read & write)