4. Query Processing Part 1

Slides adapted from Pearson Ed.

In this lecture – learning objectives

1

- Understand how a computer is able to translate natural-language SQL queries into a set of actions to perform on data.

2

- **Query decomposition**: Be familiar with both generic parsing techniques and ones specific to SQL queries / relational databases.

3

- **Query optimization**: Multiple ways exist to execute a single query.
  - Recognize equivalent relational algebra expressions.
  - Distinguish between slower and faster executions of the same query.

Query processing in RDBMSs

- SQL is a declarative (rather than procedural) language – specifies the desired information but not the strategy for retrieval.
- Job of optimizing performance of queries can be given to / centralized in a DBMS.
- Two components of query processing:
  - **Query decomposition**: conversion of SQL's natural language syntax into equivalent and valid relational algebra.
  - **Query optimization**: selection between equivalent relational algebra statements for the one that minimizes work (usually disk reads).

Query decomposition

Query decomposition

- Goal is to reject invalid SQL statements and convert valid SQL statements into equivalent relational algebra statement.
- Stages:
  - Parsing, normalization, semantic analysis, simplification.

Parsing
- Lexical analysis
- Syntactic analysis
- Generic semantic analysis

Normalization
- Standardize format of logical clauses (in selection conditions)

Semantic analysis
- Reject invalid queries

Simplification
- Final simplifications of query

Parsing

- SQL statement parsed using compiler techniques:
  - Lexical analysis to understand tokens (keywords, variable names, comma-separated lists, etc.).
    - Lexicon = dictionary. Make sure all parts of a query are a recognizable token.
  - Syntactic analysis to understand arrangement of tokens and check that they are grammatically valid.
    - Syntax = grammar / how words are properly arranged into sentences.
  - Generic semantic analysis to detect name mismatches.
    - Semantics = meaning. Is the query trying to do something sensible?

Parsing example

- Break query into tokens.
- Use a grammar to distinguish between acceptable and non-acceptable combinations of tokens.
- Flag issues and report them back to the user.

SELECT keyword | WILDCARD

SELECT *

FROM keyword | VARIABLE | VARIABLE ALIAS | COMMA | VARIABLE | VARIABLE ALIAS

FROM Staff s, Branch b

WHERE keyword | VARIABLE | DOT | ATTR | EQUALS | STRING LITERAL

WHERE s.position = "Manager"

AND keyword | VARIABLE | DOT | ATTR | EQUALS | STRING LITERAL

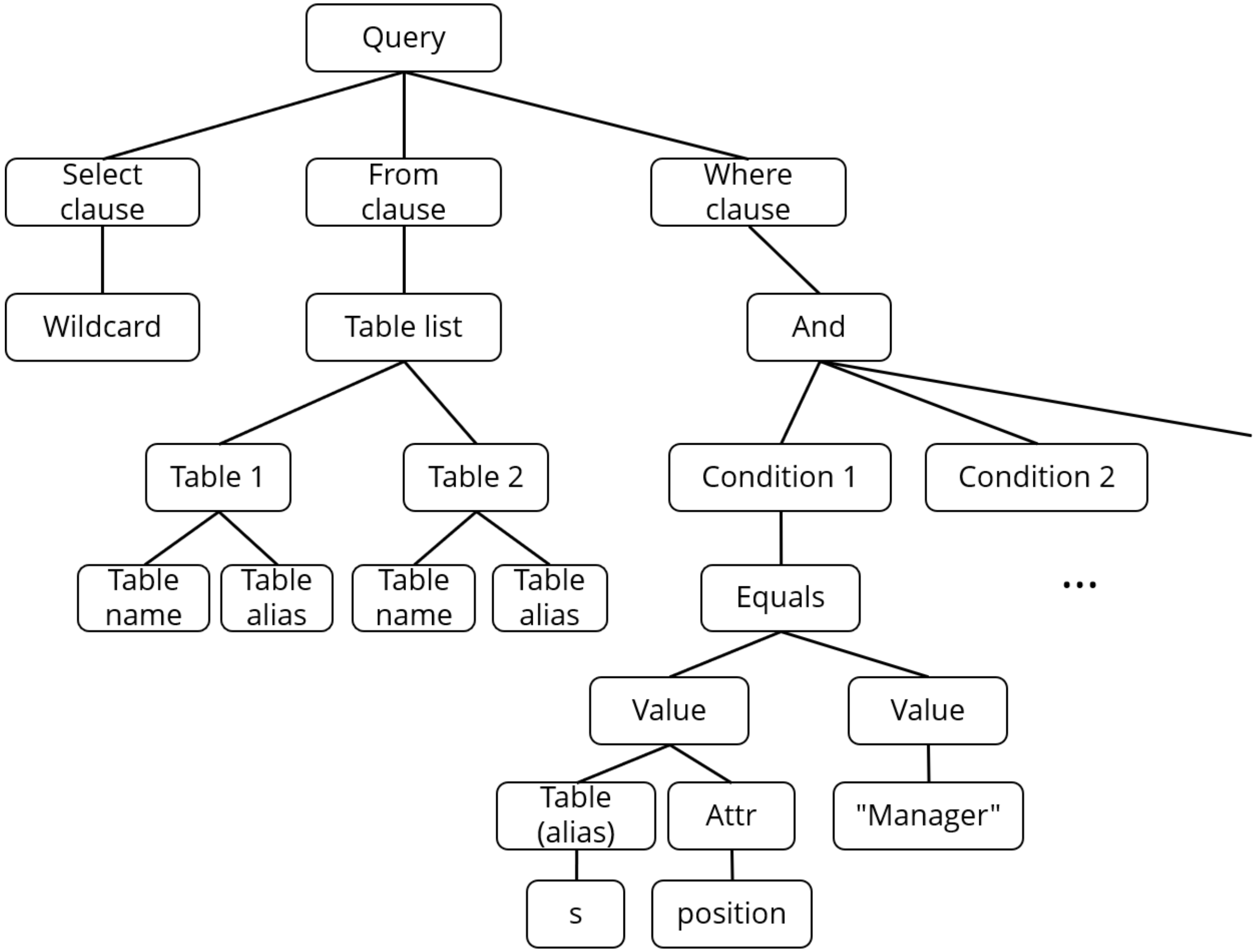
AND b.city = "London"

AND keyword | VARIABLE | DOT | ATTR | EQUALS | VARIABLE | DOT | ATTR | SEMI-COLON

AND s.branchNo = b.branchNo;

Result: abstract syntax tree

- Summarizes structure of query.

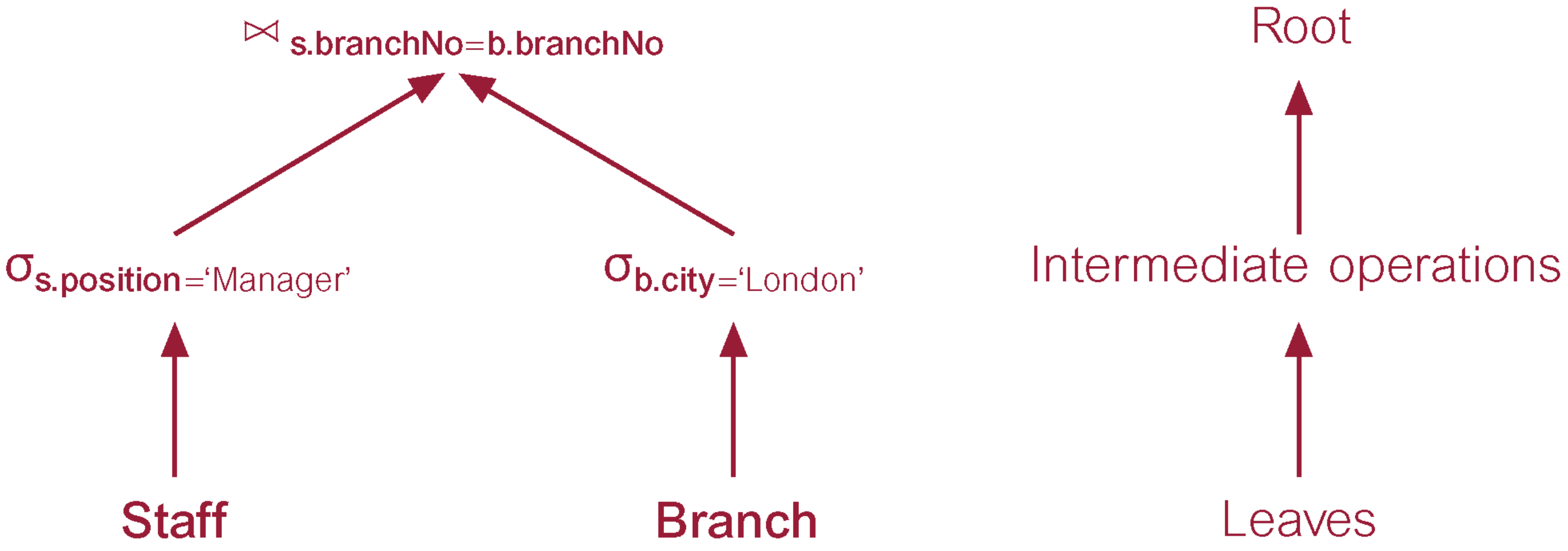


Parsing: checking for errors

- Generic semantic analysis:
  - Check that identifiers (table and attribute names) are valid.
  - Verify operations are appropriate to apply to types of objects.
- Example:
  SELECT staffNumber FROM Staff WHERE position > 10;
  - Issues:
    - "staffNumber" not defined – should be "staffNo".
    - Numerical comparison inappropriate for string data of "position".
- If everything looks okay, can convert from syntax tree to a relational algebra tree.

Relational algebra tree

- Leaf = relation; interior node = RA op; root = final op of query; arrows = connect args to op.



Normalization

- Can convert conditions into a standard form that makes checking them easier.
  - Conjunctive normal form: all conditions are made up of “OR” clauses “AND”ed together (e.g. (p v q) ^ r ^ (s v t v w)).
  - Disjunctive normal form: all conditions are made up of “AND” clauses “OR”ed together (e.g. (p ^ q) v r v (s ^ t ^ w)).
- Helpful for semantic analysis (next step).

Semantic analysis

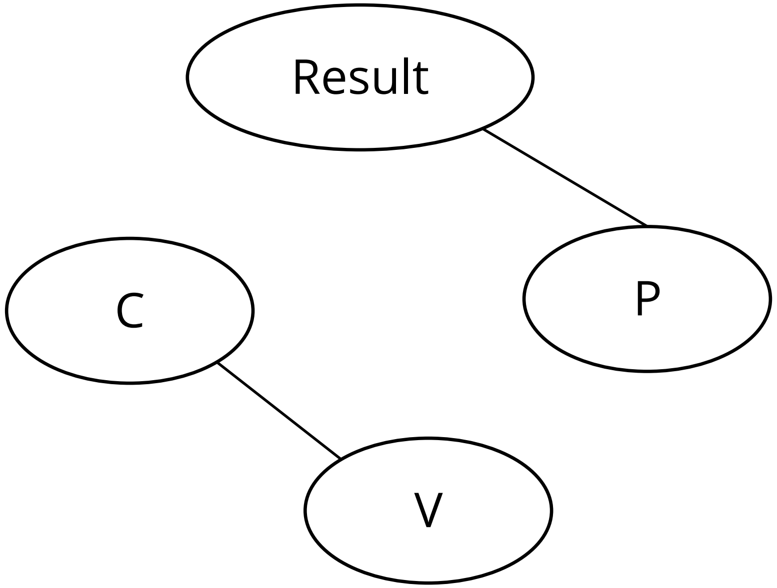
- Rejects normalized conditional formulas that are incorrectly formulated or contradictory.
  - Incorrectly formulated: components do not contribute to generation of result.
  - Contradictory: formula cannot be satisfied by any tuple.

Semantic analysis: Checking for incorrect formulations

- Draw *relation connection graph* with one node per relation in query and one node for the result.
- Add edges between relations if they're participating in a join condition together.
- Add edges between result node and relations that appear in final list of attributes.
- Unconnected graph = incorrect formulation.

Relation connection graph example

- SELECT p.propertyNo, p.street
  FROM Client c, Viewing v,
  PropertyForRent p
  WHERE c.clientNo = v.clientNo AND
  c.maxRent >= 500 AND
  c.prefType = ‘Flat’ AND
  p.ownerNo = ‘CO93’;

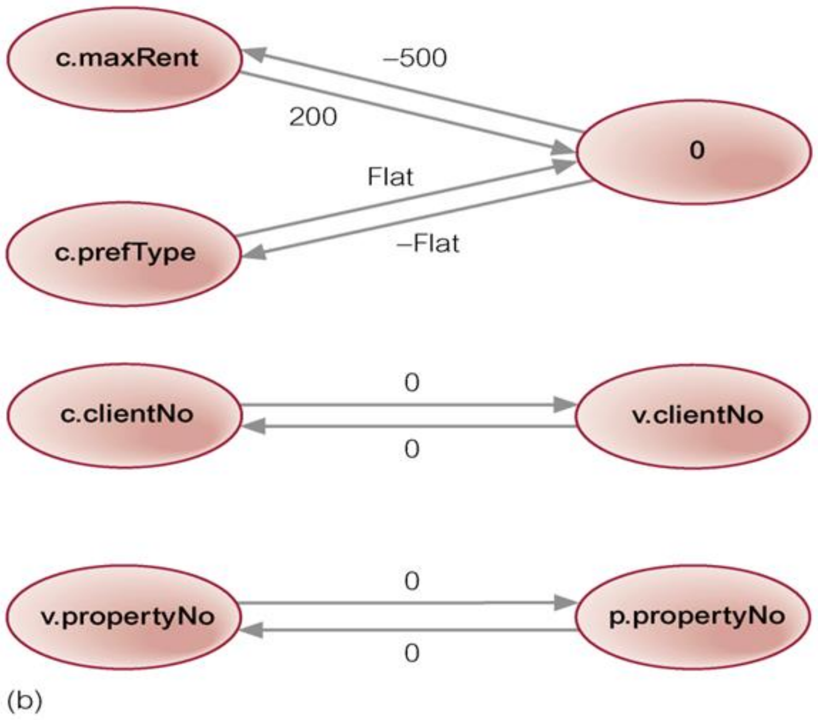


Semantic analysis: Checking for contradictions

- Draw a *normalized attribute connection graph* with one node for each attribute and a ‘0’ constant node.
- Add edges from attribute a to attribute b if they’re involved in an ‘a ≤ b’ comparison, and between an attribute and the ‘0’ node if the attribute is compared to a constant.
  - Equality is represented with two arrows.
  - Use weights on arrows to adjust the constant appropriately.
- Cycle with negative total weight = unsatisfiable condition.

Normalized attribute connection graph

- SELECT p.propertyNo, p.street
  FROM Client c, Viewing v,
  PropertyForRent p
  WHERE c.clientNo = v.clientNo AND
  c.maxRent >= 500 AND
  v.propertyNo = p.propertyNo AND
  c.prefType = 'Flat' AND
  c.maxRent < 200;



Simplification

- Query seems okay.
- Check if user has permissions to access everything it touches.
  - Reject operation otherwise.
- Simplify conditions using Boolean algebra laws and knowledge of integrity constraints.

Query optimization

Example of importance of query optimization

- Find all Managers who work at a London branch:
  SELECT *
  FROM Staff s, Branch b
  WHERE s.branchNo = b.branchNo AND (s.position = ‘Manager’
  AND b.city = ‘London’);

Different strategies

- Three equivalent RA queries are:

1. $\sigma_{(position=‘Manager’) \wedge (city=‘London’) \wedge (Staff.branchNo=Branch.branchNo)}$ (Staff X Branch)
2. $\sigma_{(position=‘Manager’) \wedge (city=‘London’)}$ (Staff $\bowtie_{Staff.branchNo=Branch.branchNo}$ Branch)
3. ($\sigma_{position=‘Manager’}$(Staff)) $\bowtie_{Staff.branchNo=Branch.branchNo}$ ($\sigma_{city=‘London’}$(Branch))

Cost comparison

- Assuming:
  - 1000 staff members, 50 branches, 50 managers, 5 London branches.
  - No search aids such as indices.
  - Tuples accessed one at a time.
- Cost in disk accesses (reads/writes):

1. Read Staff and Branch, write full Cartesian product, comb through result to select = 101,050 disk accesses.
2. Read Staff and Branch, write results of joining Staff to matching Branch info, comb through result set = 3,050 disk accesses.
3. Read/select Staff, write manager set, read/select Branch, write London set, read manager/London sets to join = 1,160 disk accesses.

Requirements for query optimization

- An understanding of the space of equivalent relational algebra statements to create alternatives.
- Strategy for selecting more efficient options.
- Aided by **statistics** to more accurately approximate the cost of operations.

Two strategies for query optimization: Dynamic vs. static processing

- Possible to process query each time the query is run (*dynamic query optimization*).
  - Uses most up-to-date information/statistics.
  - Time-consuming and less responsive to the user.
- Alternatively, possible to pre-process query and use pre-decided strategy when query is run (*static query optimization*).
  - Doesn’t take into account current statistics.
  - More optimization possible because response time isn’t an issue.
- Hybrid approaches: re-optimization triggered by significant database changes, or compute once per session and cache.

Relational algebra transformation rules

Equivalences

- Intersection: $R \cap S = R - (R - S)$
- Join: $\mathrm{R} \bowtie S = \sigma_{predicate}(R \times S)$
- Natural join: $\Pi_{reduced\ list}(\sigma_{R.c1=S.c1,etc.}(R \times S))$
- Etc.

Conjunctions in selection conditionals

- Conjunctions in a selection condition can be broken into one selection per clause:
  - $\sigma_{p \wedge q}(R) = \sigma_p(\sigma_q(R))$
- And reordered (commutativity):
  - $\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$

Projection simplification

- A series of valid projections can be reduced to just the last (outer) projection.
  - Valid = assumes that it does not try to project onto missing / non-existent attributes.
  - $\Pi_{L_1}\Pi_{L_2} \dots \Pi_{L_n}(R) = \Pi_{L_1}(R)$

Distributing selection across a projection

- Assuming a selection's condition involves a subset of a projection's attributes, can freely flip the order of the two operations:
  - $\Pi_A(\sigma_p(R)) = \sigma_p(\Pi_A(R))$ where attr(p) ⊆ A
- Cannot do this if selection involves attributes not in the projection, as performing the selection after the projection is invalid.

Commutation properties of binary operations

- The following operations commute (are symmetric):
  - Cartesian product: R x S = S x R
  - Joins (including equijoin, natural join): $R \bowtie_p S = S \bowtie_p R$
  - Set union: $R \cup S = S \cup R$
  - Set intersection: $R \cap S = S \cap R$
  - But not set difference.
- Switching order of products/joins changes order of attributes, but does not change relation overall.

Distributing selection across binary operations

- Can distribute a selection across:
  - Any set operation including set difference:

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

  - Cartesian products and joins as long as selection condition clauses can be separated into ones that only apply to one relation or the other

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

    (p only involves R's attrs; q only involves S's attrs)

Distributing projection across binary operations

- Can distribute a projection across:
  - Any set operation including set difference:

$$\Pi_A(R \cup S) = \Pi_A(R) \cup \Pi_A(S)$$
$$\Pi_A(R \cap S) = \Pi_A(R) \cap \Pi_A(S)$$
$$\Pi_A(R - S) = \Pi_A(R) - \Pi_A(S)$$

  - Cartesian products: $\Pi_A(R \times S) = \Pi_A(R) \times \Pi_A(S)$
  - Joins as long as projection doesn't mask out attributes needed for the join condition clauses (if any): $\Pi_{L_1 \cup L_2}\left(R_1 \bowtie_p R_2\right) = \Pi_{L_1 \cup L_2}\left(\Pi_{L_1 \cup A}(R_1\right) \bowtie_p \Pi_{L_2 \cup A}(R_2))$ (L1= attrs from R1 in final projection; L2 = attrs from R2 in final projection; A = join condition attrs)
    - If no conditional, can simplify to: $\Pi_{L_1 \cup L_2}\left(R_1 \bowtie_p R_2\right) = \Pi_{L_1}(R_1) \bowtie_p \Pi_{L_2}(R_2)$

Associative properties of binary operations

- The following operations are associative:
  - Set union and intersection (but not difference):
$$(R \cup S) \cup T = R \cup (S \cup T)$$
$$(R \cap S) \cap T = R \cap (S \cap T)$$
$$(R - S) - T \neq R - (S - T)$$
  - Cartesian product: (R x S) x T = R x (S x T)
  - Natural joins: $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
  - Other joins as long as join conditions are handled correctly to reflect that different pairs of relations are being joined
$$\left(R \bowtie_p S\right) \bowtie_q T = R \bowtie_{p'} \left(S \bowtie_{q'} T\right)$$

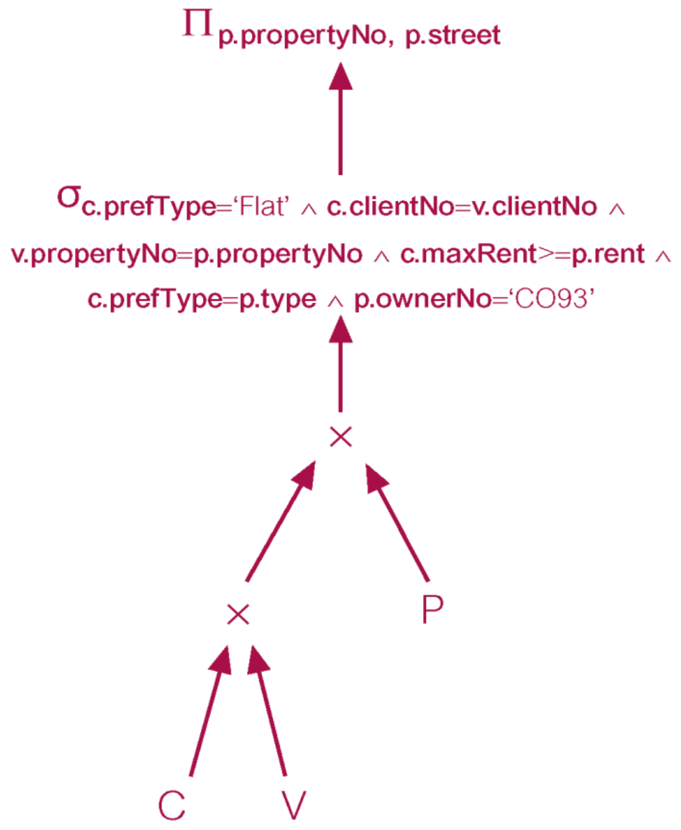
Using RA rules for query optimization

Example: Using transformation rules

- Task: For prospective renters of flats, find properties that match requirements and owned by owner 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';
```

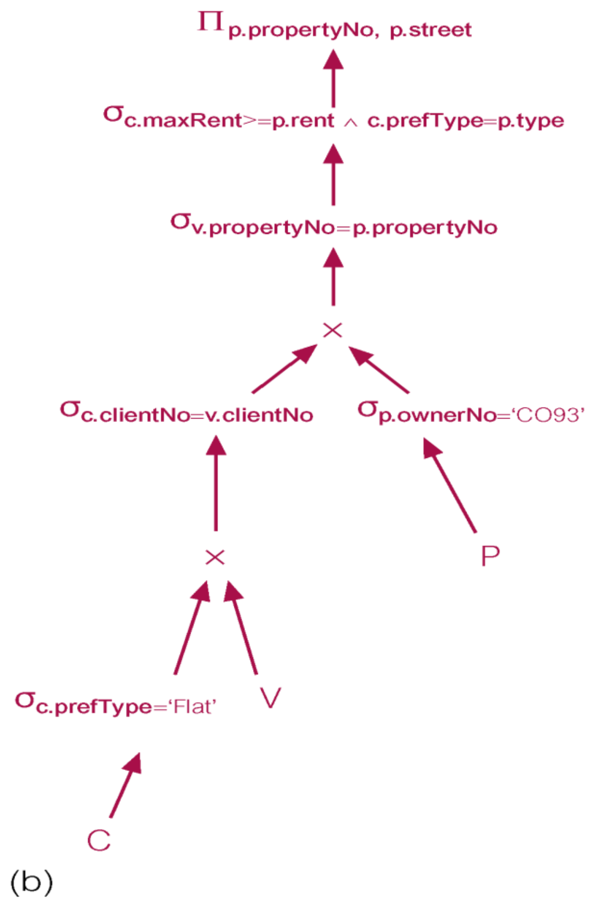
Original RA tree from SQL query



(a)

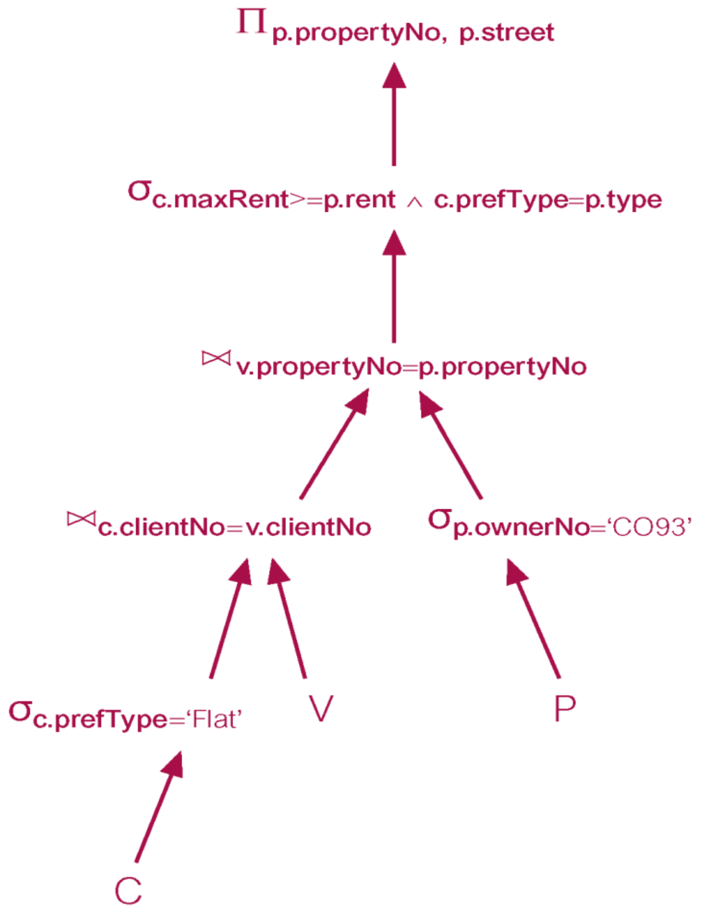
Distributing selections inward

- Can separate conjunctions of a selection.
- Can distribute selections across Cartesian products as long as they only use an individual relation's attrs in their condition.



Join equivalences

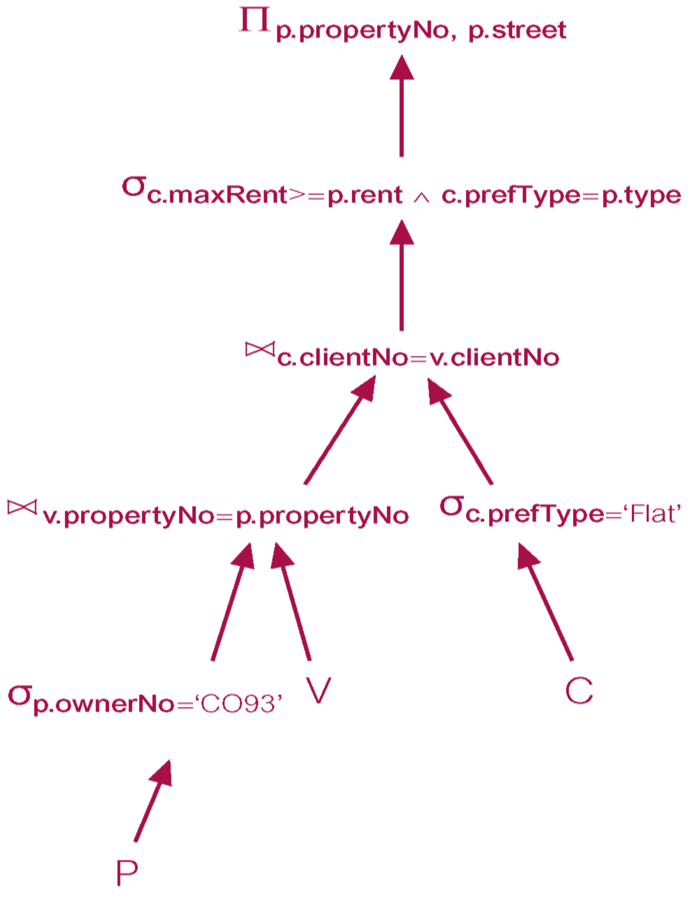
- Can convert product + selection into equivalent equijoin



(c)

Associativity of joins

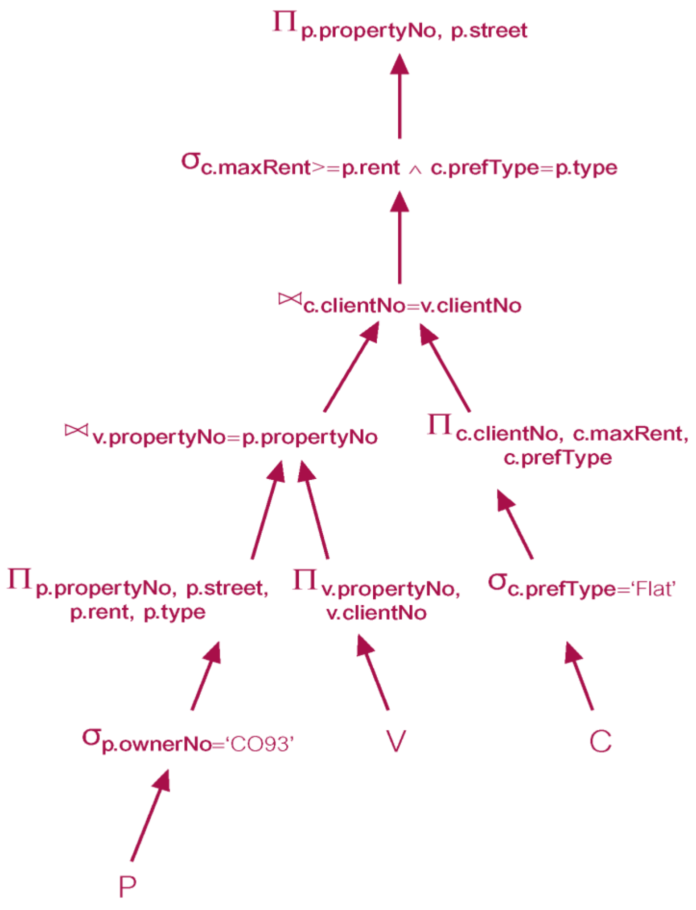
- Can use associativity rules to reorder which join is performed first (innermost)
- Choose the most selective to go first (the one that selects all flats owned by one owner)



(d)

Distributing projections

- Move projections inward.



(e)

Heuristics for speeding up queries

- Perform selection and projection ops as early as possible because they reduce the size of datasets.
  - Distribute ops inward using rules.
- Combine Cartesian products followed by selections with joins to combine two ops into one.
- Perform most restrictive selections first.
  - Rearrange leaves using associativity rules for binary operations.
- Compute common expressions once and store result (if not too big).

Summary

- Parsing techniques help a computer translate natural-language SQL queries into *relational algebra trees* (RATs) corresponding to an RA expression.
- Different RATs can have drastically different execution time.
- Use RA equivalences to transform an RAT into one that is more efficient but *does the same work*.