CS525: Advanced Database Organization

Notes 6: Query Processing
Convert Parse Tree into initial L.Q.P

Yousef M. Elmehdwi

Department of Computer Science

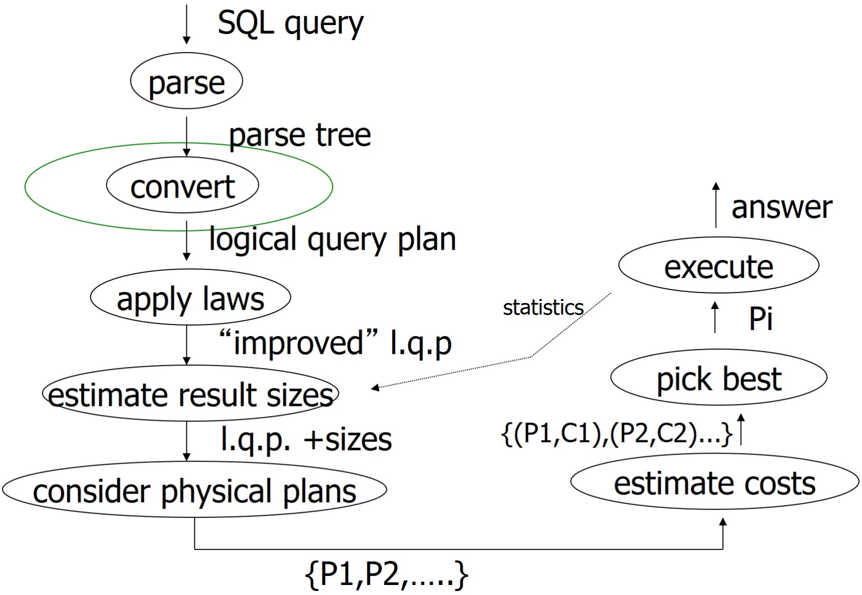
Illinois Institute of Technology

yelmehdwi@iit.edu

October 9, 2018

Slides: adapted from a course taught by Hector Garcia-Molina, Stanford

Where we are?



SQL query
parse
parse tree
convert
logical query plan
apply laws
"improved" l.q.p
estimate result sizes
l.q.p. +sizes
consider physical plans
{P1,P2,.....}
estimate costs
{(P1,C1),(P2,C2)...}
pick best
Pi
execute
answer
statistics

Parsing

- Goal is to convert a text string containing a query into a parse tree data structure:
  - leaves form the text string (broken into lexical elements)
  - internal nodes are syntactic categories
- Uses standard algorithmic techniques from compilers
  - given a grammar for the language (e.g., SQL), process the string and build the tree

The Pre-processor

- replaces each reference to a view with a parse (sub)-tree that describes the view (i.e., a query)
- does semantic checking:
  - are relations and views mentioned in the schema?
  - are attributes mentioned in the current scope?
  - are attribute types correct?

Today

- Convert the `parse tree` to an initial query plan, which is usually an algebraic representation of the query (relational algebra expression)

How Queries are Answered[1]

- A query is usually stated in a high-level declarative DB language (e.g., SQL)
  - For relational databases: DB language can be mapped to relational algebra for further processing
- To be evaluated it has to be translated into a low level execution plan

[1] Slide Credit: Wolf-Tilo Balke, Institut fuer Informationssysteme

Conversion

- Create an internal representation
  - Should be useful for analysis
  - Should be useful for optimization
- Internal representation
  - Relational algebra
  - Query tree/graph models

Relational Algebra

- Made popular by Edgar Frank "Ted" Codd 1970
- Theoretical foundation of relational databases
  - Describes how to retrieve interesting parts of available relations
  - Lead to the development of SQL
  - Relational algebra is mandatory to understand the query optimization process
- Set of operators that take relations as input and produce relations as output
  - closed: the output of evaluating an expression in relational algebra can be used as input to another relational algebra
- Relational algebra is based on a minimal set of operators that can be combined to write complex queries
- Databases implement relational algebra operators to execute `SQL` queries.
- Relations in `SQL` are really bags, or multisets; $\Rightarrow$ we shall introduce relational algebra as an algebra on bags.

Relational Algebra Recap

- Formal query language
- Consists of operators
  - Input(s): relation
  - Output: relation
  - $\Rightarrow$ Composable
    - The operators take one or two relations as inputs and produce a new relation as a result.
- Set and Bag semantics version

Relation Schema, Relation Instance, Tuple

- Relation Schema
  - Schema for a relation defined the names of the attributes and the domain for the attributes
- Relation (instance)
  - A (multi-)set of tuples with the same schema
- Tuple
  - List of attribute value pairs (or function from attribute name to value)

Set- vs. Bag semantics

- **Sets**: $\{a, b, c\}$, $\{a, d, e, f\}$, ...
- **Bags**: $\{a, a, b, c\}$, $\{b, b, b, b, b\}$, ...
- `Set semantics`
  - Relations are Sets
  - Used in most theoretical work
- `Bag semantics`
  - Relations are `Multi-Sets`: Each element (tuple) can appear more than once
  - `SQL` uses bag semantics

Set- vs. Bag semantics

Set

Name	Purchase
Peter	Guitar
Joe	Drum
Alice	Bass

Bag

Name	Purchase
Peter	Guitar
Peter	Guitar
Joe	Drum
Alice	Bass
Alice	Bass

Bag semantics notation

- We use $\mathtt{t}^{m}$ to denote tuple $\mathtt{t}$ appears with multiplicity $\mathtt{m}$

Operators

- `Selection`
- `Renaming`
- `Projection`
- `Joins`
  - Theta, natural, cross-product, outer, anti
- `Aggregation`
- `Duplicate removal`
- `Set operations`

Select

- Pick certain tuples/rows
- Syntax: σ_c(R)
  - R is input
  - c is a condition (called the selection predicate)
- Semantics:
  - Return all tuples that match condition c
  - Set: { t | t $\in$ R AND t fulfills c}
  - Bag: { t^n | t^n $\in$ R AND t fulfills c}

Select: Example

- $\sigma_{a>5}$(R)

R

a	b
1	13
3	12
6	14

Result

a	b
6	14

Project

- Pick certain columns
- Syntax: π_A(R)
  - R is input
  - A is list of projection expressions
- Semantics:
  - Project all inputs on projection expressions
  - Set: $\{$ t.A | t$\in$R$\}$
  - Bag: $\{$ (t.A)^n | t^n $\in$R$\}$

Project: Example

- $\pi_b(\mathtt{R})$

R

a	b
1	13
3	12
6	14

Result

b
13
12
14

Compose: Select and Project

- to pick both columns and rows, we can compose operators
  - Example: $\pi_{A_1, A_2, \ldots, A_n}(\sigma_{condition}(\texttt{Expression}))$

Renaming

- To unify schemas for set operators
- For disambiguation in "self-joins"
- Syntax: ρ_A(R)
  - R is input
  - A is list of attribute renaming b←a
- Semantics:
  - Applies renaming from A to inputs
  - Set: { t.A | t∈R}
  - Bag: { $(t.A)^n$ | t^n ∈R}

Renaming: Example

- $\rho_{c \leftarrow a}(\mathtt{R})$

R

a	b
1	13
3	12
6	14

Result

c	b
1	13
3	12
6	14

Cross Product

- Combine two relations (a.k.a Cartesian product)
- Syntax: R $\times$ S
  - R and S are inputs
- Semantics:
  - All combinations of tuples from R and S
  - = mathematical definition of cross product
  - Set: { (t,s) | t$\in$R AND s$\in$S}
  - Bag: { $(t,s)^{n \times m}$ | $t^{n} \in$S AND $s^{m} \in$S}

Cross Product: Example

- $R \times S$

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
1	13	a	5
1	13	b	3
1	13	c	4
3	12	a	5
3	12	b	3
3	12	c	4

Join

- Syntax: $\texttt{R} \bowtie_c \texttt{S}$
  - R and S are inputs
  - c is a condition
- Semantics:
  - All combinations of tuples from R and S that match c
  - Set: { (t,s) | t∈R AND s∈S AND (t,s) matches c}
  - Bag: { $(t,s)^{n \times m}$ | t^n ∈R AND s^m∈S AND (t,s) matches c}

Join: Example

- R $\bowtie_{a=d}$ S

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
3	12	b	3

Natural Join

- Enforce equality on all attributes with same name
- Eliminate one copy of duplicate attributes
- Syntax: $\texttt{R} \bowtie \texttt{S}$
  - R and S are inputs
- Semantics:
  - All combinations of tuples from R and S that match on common attributes
  - A = common attributes of R and S
  - C = exclusive attributes of S
  - Set: $\{(\texttt{t,s.C}) \mid \texttt{t} \in \texttt{R AND s} \in \texttt{S AND t.A=s.A}\}$
  - Bag: $\{(\texttt{t,s.C})^{n \times m} \mid \texttt{t}^{n} \in \texttt{R AND s}^{m} \in \texttt{S AND t.A=s.A}\}$

Natural Join: Example

- $R \bowtie S$

R

a	b
1	13
3	12

S

c	a
a	5
b	3
c	4

Result

a	b	c
3	12	b

Left-outer Join

- Syntax: $\texttt{R} ⟕_c \texttt{S}$
  - R and S are inputs
  - c is condition
- Semantics:
  - R join S
  - t∈R without match, fill S attributes with NULL

    {(t,s) | t∈R AND s∈S matches c}
    union
    {(t,NULL(S))| t∈R AND NOT exists s∈S: (t,s) matches c}

Left-outer Join: Example

- $R \,⟕_{a=d}\, S$

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
1	13	NULL	NULL
3	12	b	3

Right-outer Join

- Syntax: R $\rtimes_c$ S
  - R and S are inputs
  - c is condition
- Semantics:
  - R join S
  - s∈S without match, fill R attributes with NULL

    {(t,s) | t∈R AND s∈S matches c}

    union

    {(NULL(R),s)| s∈S AND NOT exists t∈R: (t,s) matches c}

Right-outer Join: Example

- R $\rtimes_{a=d}$ S

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
NULL	NULL	a	5
3	12	b	3
NULL	NULL	c	4

Full-outer Join

- Syntax: $R ⟗_c S$
  - R and S are inputs
  - c is condition
- Semantics:

```
{(t,s) | t∈R AND s∈S AND (t,s) matches c}
union
{(NULL(R),s)| s∈S AND NOT exists t∈R: (t,s) matches c}
union
{(t,NULL(S))| t∈R AND NOT exists s∈S: (t,s) matches c}
```

Full-outer Join: Example

- R $⟗_{a=d}$ S

R

a	b
1	13
3	12

S

c	d
a	5
b	3
c	4

Result

a	b	c	d
1	13	NULL	NULL
NULL	NULL	a	5
3	12	b	3
NULL	NULL	c	4

Aggregation

- Grouping and aggregation generally need to be implemented and optimized together
- Syntax: $_{G}\gamma_{A}$(R)
  - A is list of aggregation functions
  - G is list of group by attributes
- Semantics:
  - Build groups of tuples according G and compute the aggregation functions from each group
  - {t.G, agg(G(t)) | t∈R}
  - G(t) = {t' | t'∈R AND t'.G = t.G}

Aggregation: Example

- ${}_{b}\gamma_{sum(a)}$ (R)

R

a	b
1	1
3	1
6	2
3	2

Result

sum(a)	b
4	1
9	2

Duplicate Removal

- `Syntax:` δ`(R)`
  - `R` is input
- Semantics:
  - Remove duplicates from input
  - `Set:` N/A
  - `Bag:` $\{\texttt{t}^1 \mid \texttt{t}^n \in \texttt{R}\}$

Duplicate Removal

- δ(R)

R

a	b
1	13
1	13
6	14

Result

a	b
1	13
6	14

Union, Intersection, and Difference

- Input: `R` and `S`
  - Have to have the same schema
    - Union compatible: two relations have the same schema: exactly same attributes drawn from the same domain

Union

- Syntax: $\texttt{R} \cup \texttt{S}$
  - R and S are union-compatible inputs
- Semantics:
  - Set: $\{\texttt{t} \mid \texttt{t} \in \texttt{R OR t} \in \texttt{S}\}$
  - Bag:
    - An element appears in the union of two bags the sum of the number of times it appears in each bag.
    - $\{(\texttt{t,s})^{n+m} \mid \texttt{t}^{n} \in \texttt{R AND s}^{m} \in \texttt{S}\}$
    - e.g., $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

Union: Example

- R ∪ S

R

a
1
3

S

b
1
2
3

Result

a
1
2
3
1
3

Intersection

- Syntax: `R ∩ S`
  - `R` and `S` are `union-compatible` inputs
- Semantics:
  - Set: $\{t \mid t \in R \text{ AND } t \in S\}$
  - Bag: $\{(t,s)^{min(n,m)} \mid t^{n} \in R \text{ AND } s^{m} \in S\}$
    - An element appears in the intersection of two bags the minimum of the number of times it appears in either
    - $\{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}$

Intersection: Example

- $R \cap S$

R

a
1
3

S

b
1
2
3

Result

a
1
3

Set Difference

- Syntax: `R − S`
  - `R` and `S` are `union-compatible` inputs
- Semantics:
  - Set: $\{$`t | t`$\in$`R AND NOT t`$\in$`S`$\}$
  - Bag: $\{$`(t,s)`$^{n-m}$ `| t`^{n} $\in$`R AND s`^{m} $\in$`S`$\}$
    - An element appears in the difference `R − S` of bags as many times as it appears in `R`, minus the number of times it appears in `S`.
    - But never less than `0` times.
    - $\{$`1,2,1,1`$\}$ `−` $\{$`1,2,3`$\}$ `=` $\{$`1,1`$\}$

Set Difference: Example

- R − S

R

a
1
5

S

b
1
2
3

Result

a
5

Canonical Translation to Relational Algebra

- Given an `SQL` query
- Return an equivalent relational algebra expression

Canonical Translation to Relational Algebra

- **FROM** clause into joins and crossproducts
- **WHERE** clause into selection
- **SELECT** clause into projection and renaming
  - If it has aggregation functions use aggregation
  - **DISTINCT** into duplicate removal
- **GROUP BY** clause into aggregation
- **HAVING** clause into selection
- **ORDER BY** - no counter-part

Set Operations

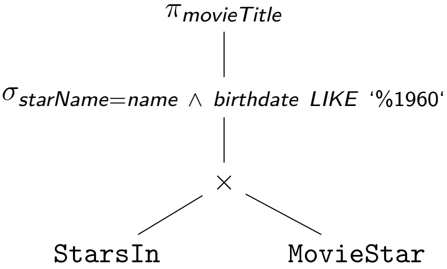
- **UNION ALL** into union
- **UNION** duplicate removal over union
- **INTERSECT ALL** into intersection
- **INTERSECT** add duplicate removal
- **EXCEPT ALL** into set difference
- **EXCEPT** apply duplicate removal to inputs and then apply set difference

Expression Trees

- Leaves are operands
- Interior nodes are operators, applied to their child or children.

Example: Relational Algebra Translation

```sql
SELECT movieTitle
FROM StarsIn, MovieStar
WHERE starName = name AND birthdate LIKE '%1960';
```



Example: Relational Algebra Translation

```sql
SELECT sum(a)
FROM R
GROUP BY b
```

$$\pi_{sum(a)}$$

$$|$$

$${}_{b}\gamma_{sum(a)}$$

$$|$$

$$\mathtt{R}$$

Example: Relational Algebra Translation

```sql
SELECT dep, headcnt
FROM (SELECT count(*) AS headcnt, dep
      FROM Employee
      GROUP BY dep)
WHERE headcnt > 100
```

$$\pi_{dep,headcnt}$$
$$|$$
$$\sigma_{headcnt>100}$$
$$|$$
$$\rho_{headcnt \leftarrow count(*)}$$
$$|$$
$${}_{dep}\gamma_{count(*)}$$
$$|$$
$$\texttt{Employee}$$