>>

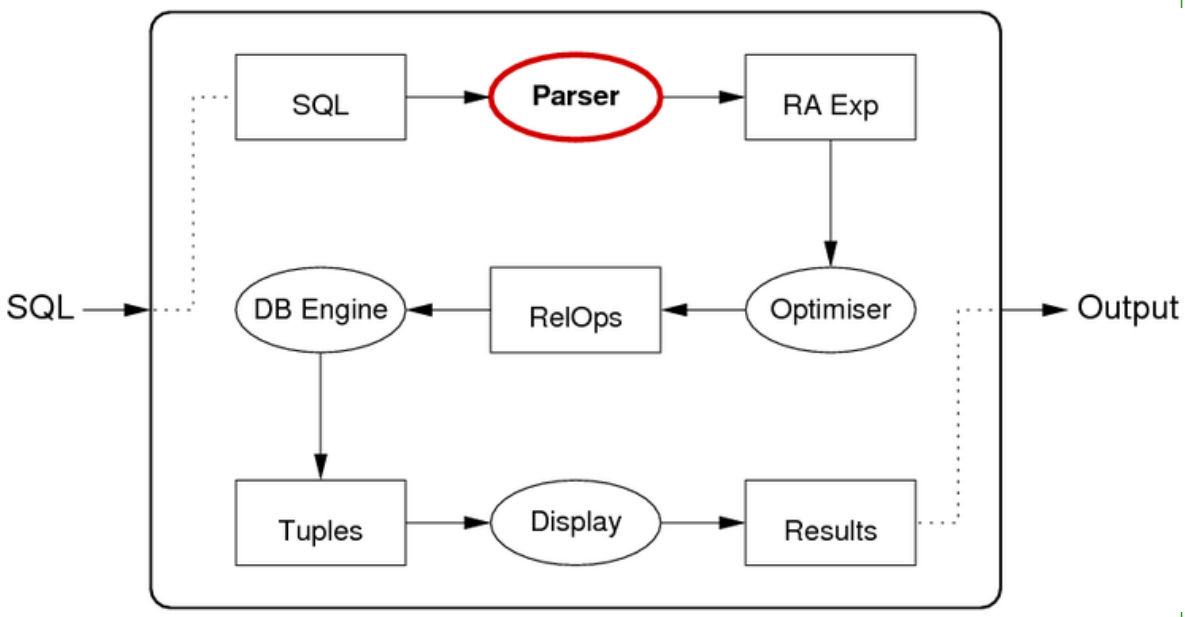
Query Translation

- Query Translation
- Parsing SQL
- Expression Rewriting Rules
- Relational Algebra Laws
- Query Rewriting

^ >>

❖ Query Translation

Query translation: SQL statement text → RA expression



<< ^ >>

❖ Query Translation (cont)

Translation step: SQL text → RA expression

Example:

```
SQL: select name from Students where id=7654321;
-- is translated to
RA:  Proj[name](Sel[id=7654321]Students)
```

Processes: lexer/parser, mapping rules, rewriting rules.

Mapping from SQL to RA may include some optimisations, e.g.

```
select * from Students where id = 54321 and age > 50;
-- is translated to
Sel[age>50](Sel[id=54321]Students)
-- rather than ... because of index on id
Sel[id=54321&age>50](Students)
```

<< ^ >>

❖ Parsing SQL

Parsing task is similar to that for programming languages.

Language elements:

- keywords: **`create`**, **`select`**, **`from`**, **`where`**, ...
- identifiers: **`Students`**, **`name`**, **`id`**, **`CourseCode`**, ...
- operators: **`+`**, **`-`**, **`=`**, **`<`**, **`>`**, **`AND`**, **`OR`**, **`NOT`**, **`IN`**, ...
- constants: **`'abc'`**, **`123`**, **`3.1`**, **`'01-jan-1970'`**, ...

PostgreSQL parser ...

- implemented via lex/yacc (**`src/backend/parser`**)
- maps all identifiers to lower-case (A-Z → a-z)
- needs to handle user-extendable operator set
- makes extensive use of catalog (**`src/backend/catalog`**)

<< ^ >>

❖ Expression Rewriting Rules

Since RA is a well-defined formal system

- there exist many algebraic laws on RA expressions
- which can be used as a basis for expression rewriting
- in order to produce equivalent (more-efficient?) expressions

Expression transformation based on such rules can be used

- to simplify/improve SQL →RA mapping results
- to generate new plan variations to check in query optimisation

<< ^ >>

❖ Relational Algebra Laws

Commutative and Associative Laws:

- $R \bowtie S \leftrightarrow S \bowtie R$, $(R \bowtie S) \bowtie T \leftrightarrow R \bowtie (S \bowtie T)$ (natural join)
- $R \cup S \leftrightarrow S \cup R$, $(R \cup S) \cup T \leftrightarrow R \cup (S \cup T)$
- $R \bowtie_{Cond} S \leftrightarrow S \bowtie_{Cond} R$ (theta join)
- $\sigma_c(\sigma_d(R)) \leftrightarrow \sigma_d(\sigma_c(R))$

Selection splitting (where c and d are conditions):

- $\sigma_{c \wedge d}(R) \leftrightarrow \sigma_c(\sigma_d(R))$
- $\sigma_{c \vee d}(R) \leftrightarrow \sigma_c(R) \cup \sigma_d(R)$

<< ^ >>

❖ Relational Algebra Laws (cont)

Selection pushing ($\sigma_c(R \cup S)$ and $\sigma_c(R \cup S)$):

- $\sigma_c(R \cup S) \leftrightarrow \sigma_c R \cup \sigma_c S, \quad \sigma_c(R \cap S) \leftrightarrow \sigma_c R \cap \sigma_c S$

Selection pushing with join ...

- $\sigma_c (R \bowtie S) \leftrightarrow \sigma_c(R) \bowtie S$ (if c refers only to attributes from R)
- $\sigma_c (R \bowtie S) \leftrightarrow R \bowtie \sigma_c(S)$ (if c refers only to attributes from S)

If *condition* contains attributes from both R and S:

- $\sigma_{c' \wedge c''} (R \bowtie S) \leftrightarrow \sigma_{c'}(R) \bowtie \sigma_{c''}(S)$
- c' contains only R attributes, c'' contains only S attributes

❖ Relational Algebra Laws (cont)

Rewrite rules for projection ...

All but last projection can be ignored:

- $\pi_{L1}(\pi_{L2}(...\pi_{Ln}(R))) \rightarrow \pi_{L1}(R)$

Projections can be pushed into joins:

- $\pi_L(R \bowtie_c S) \leftrightarrow \pi_L(\pi_M(R) \bowtie_c \pi_N(S))$

where

- M and N must contain all attributes needed for c
- M and N must contain all attributes used in L $(L \subset M \cup N)$

<< ^ >>

❖ Query Rewriting

Subqueries ⇒ convert to a join

Example: (on schema Courses(id,code,...), Enrolments(cid,sid,...), Students(id,name,...)

```
select c.code, count(*)
from   Courses c
where  c.id in (select cid from Enrolments)
group  by c.code
```

becomes

```
select c.code, count(*)
from   Courses c join Enrolments e on c.id = e.cid
group  by c.code
```

<< ^ >>

❖ Query Rewriting (cont)

But not all subqueries can be converted to join, e.g.

```
select e.sid as student_id, e.cid as course_id
from   Enrolments e
where  e.sid = (select max(id) from Students)
```

has to be evaluated as

Val = max[id]Students

Res = $\pi_{(sid,cid)}(\sigma_{sid=Val}$Enrolments)

<< ^

❖ Query Rewriting (cont)

In PostgreSQL, views are implemented via rewrite rules

- a reference to view in SQL expands to its definition in RA

Example:

```
create view COMP9315studes as
select stu,mark from Enrolments where course='COMP9315';
-- students who passed
select stu from COMP9315studes where mark >= 50;
```

is represented in RA by

```
COMP9315studes
  = Proj[stu,mark](Sel[course=COMP9315](Enrolments))
-- with query ...
Proj[stu](Sel[mark>=50](COMP9315studes))
-- becomes ...
Proj[stu](Sel[mark>=50](
  Proj[stu,mark](Sel[course=COMP9315](Enrolments))))
-- which could be rewritten as ...
Proj[stu](Sel[mark>=50 & course=COMP9315]Enrolments)
```

Produced: 5 Apr 2021