Introduction to Query Processing

Objectives:

- Learn basic structure/steps in a query execution environment
- An aside: there are *a lot* of things called join

some slides thanks to text authors

Problem Statement

- Given
  - a SQL query
  - relationally stored data

1. What has to happen to return a correct answer?
2. What has to happen that answers are returned as fast as possible?

Character of the Solution

- Executing statements in a computer program.

```
select *
from employees
where employees.salary = 0
```

--> The solution is organized as a compiler/interpreter

What are the steps?

Compiler:	SQL execution environment

1. Lex & Parse the input
  1. parse-trees
  2. symbol tables
	1. Lex & Parse the input

2. Compile parse-trees to abstract code	2. Create a *logical plan* • abstract code
3. [optimize abstract code]	3. Optimize, logical-plan
4. Compile abstract code to physical code.	4. Consider physical trade-offs (optimize), generate → *physical plan* (sequence of executable operators, I.e. code)
5. [optimize physical code]	

Difference: Creation of "physical code" is a critical optimization process

R	A	B	C
	a	1	10
	b	1	20
	c	2	10
	d	2	35
	e	3	45

S	C	D	E
	10	x	2
	20	y	2
	30	z	2
	40	x	1
	50	y	3

Example

Select B,D
From R,S
Where R.A = "c" $\wedge$ S.E = 2 $\wedge$ R.C = S.C

Where R.A = "c" $\wedge$ S.E = 2 $\wedge$ R.C = S.C

R	A	B	C
	a	1	10
	b	1	20
	c	2	10
	d	2	35
	e	3	45

S	C	D	E
	10	x	2
	20	y	2
	30	z	2
	40	x	1
	50	y	3

Answer

B	D
2	x

Some words about *join*

1. Logically, there are many kinds of joins.
2. Each logical join may be implemented by many different algorithms. (*physical operators*)
3. Formal definition of [inner]* join

$$R \bowtie_p S = \sigma_p (R \times S)$$

* Originally, and for a long time there were only *inner* joins

Today ⋈

"join" ≡ Any operator combining information from two tables.

- Inner Joins
  - natural join
  - equi join
  - theta join
- Outer Joins
- Anti-Joins
- Semi-joins

Inner Joins

- natural join, ⋈ , with no argument
  - Equality test on all pairs of columns with the same name.
- Theta join, $R \bowtie_p S$
  - Any relational predicate p (or theta)
- Equi join,
  - A join where the predicate contains only = tests
  - So, all natural joins are equi joins
  - all equi join joins are theta joins

SQL: All these mean the same thing

```
SELECT *
FROM Persons, Zips
INNER JOIN Persons ON Zips
Persons.zipcode = Zips.zipcode
```

```
SELECT *
FROM Persons
WHERE (zipcode, city, state) IN
        (SELECT *
         FROM Zips )
```

```
SELECT *
FROM Persons, Zips
Where Persons.zipcode = Zips.zipcode
```

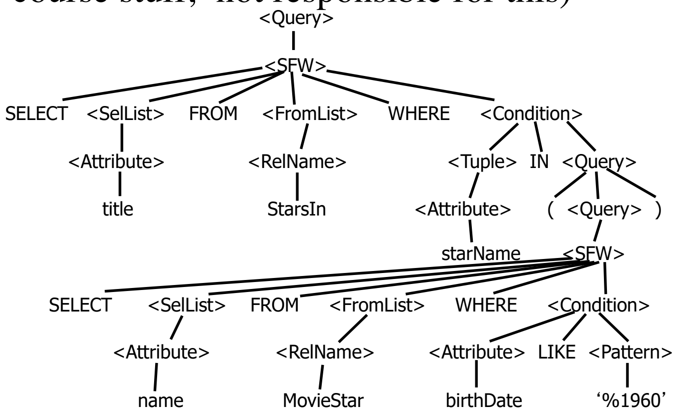
```
SELECT *
FROM Persons JOIN Zips
ON Persons.zipcode = Zips.zipcode
```

Example Goal

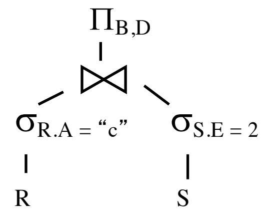
```
Select B,D
From R,S
Where
   R.A = "c"  ∧
   S.E = 2    ∧
   R.C=S.C
```

```
output(
   project((B,D),
      nestedLoopJoin("C=C",
   select("A = "c"", tableScan(R))
   select("E = 2", tableScan(S))))
```

First step: Parse (standard compiler course stuff; not responsible for this)

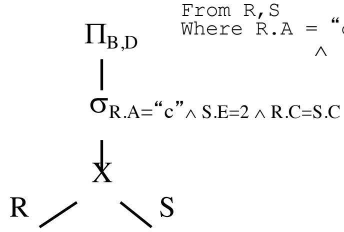


Logical Plan for the Above:



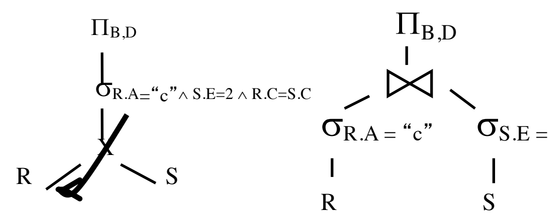
Probably the Real Starting Point

```
Select B,D
From R,S
Where R.A = “c”  ∧  S.E = 2
          ∧  R.C=S.C
```



<u>OR:</u> $\Pi_{B,D} [\, \sigma_{R.A=\text{“c”} \wedge S.E=2 \wedge R.C = S.C} (RXS)]$

We say <u>*“we pushed the selects down”*</u>

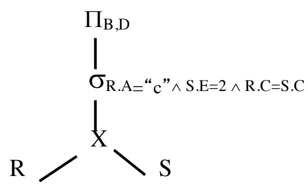


Generate a logical plan

People version:

$\Pi_{B,D} [\ \sigma_{R.A=\text{“c”} \wedge S.E=2 \wedge R.C = S.C}\ (RXS)]$

Machine version:



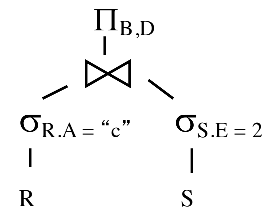
Logical Optimization

Given
- axioms and
- identities

of relational algebra

E.g.
- $(R \bowtie S) = (S \bowtie R)$
- pushing selects
  - If all columns in p are in R

$\sigma_p (R \bowtie S) \rightarrow [\sigma_p (R)] \bowtie S$



Manipulate logical plan

Big, but not too big, library of transformations, choice is important.

- $R \bowtie S \rightarrow S \bowtie R$ (logically equiv. but not physically)
- $\sigma_{p1 \wedge p2} (R) \rightarrow \sigma_{p1} [\sigma_{p2} (R)]$
- $\sigma_p (R \bowtie S) \rightarrow [\sigma_p (R)] \bowtie S$
- $\pi_x [\sigma_p\ (R)] \rightarrow \pi_x \{\sigma_p [\pi_{xz}\ (R)]\}$

Physical Plan Generation

Replace logical operators with physical operators, while considering the cost of alternatives

- Hard to separate optimization from generation process.

Suppose there is an index on R.A?

```
Select B,D
From R,S
Where R.A = "c" ∧ S.E = 2 ∧ R.C=S.C
```

If it's the primary index, no thinking

```
output(
  project((B,D),
    nestedLoopJoin("C=C",
  ~~select("A = "c"", tableScan(R))~~
  primaryIndexScan(c,R)
  select("E = 2", tableScan(S))))
```

If it's a secondary index (remember city ="Austin")

select("A = "c"", tableScan(R)) *or* secondaryIndexScan(c,R) ?

Better compute the cost of the alternatives.

Other components of cost

- size of the relation
  - base relation : a table stored in the database
  - relation: any table, stored or computed
- Given an operator, a predicate a relation
  - (e.g. R.city = "austin")

  How big is the computed relation?

In this next section we will cover

- details of implementing individual database operators.
- how to estimate the cost of the operators
  - cost(operator, predicates, input parameters)
  - returns
    - cost values, I/O, CPU
    - input parameters of the relation for the next cost
- search methods for assigning the final choices.