THE UNIVERSITY OF MELBOURNE

INFO20003 Database Systems

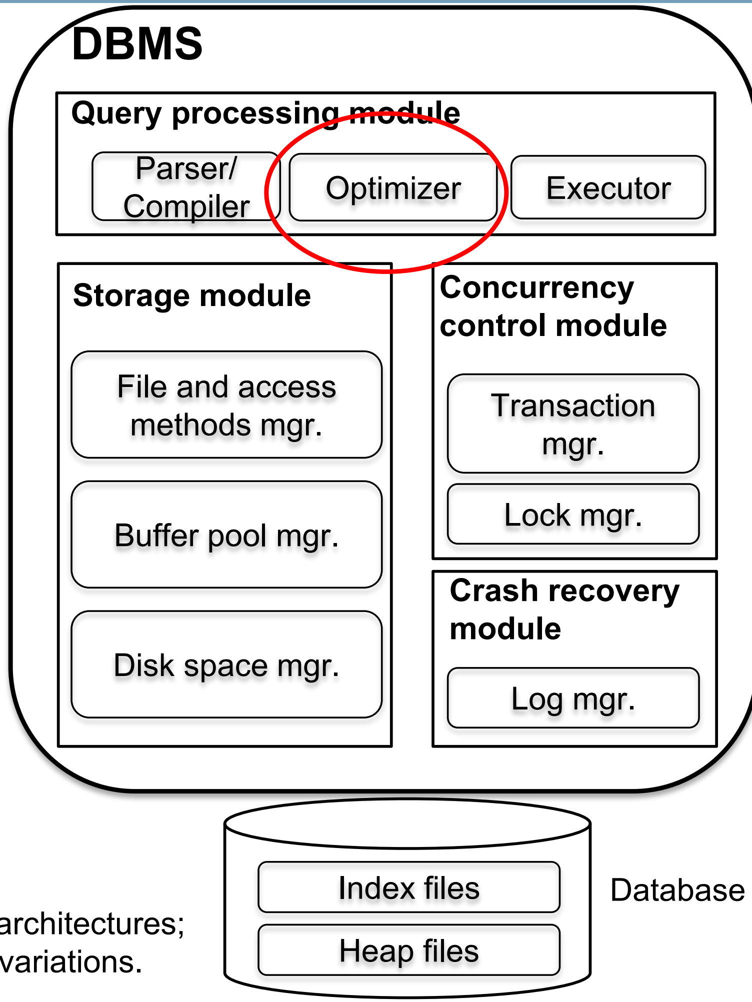
Dr Renata Borovica-Gajic

Lecture 13

Query Optimization Part I

Week 7

Remember this? Components of a DBMS



TODAY & Next time

This is one of several possible architectures; each system has its own slight variations.

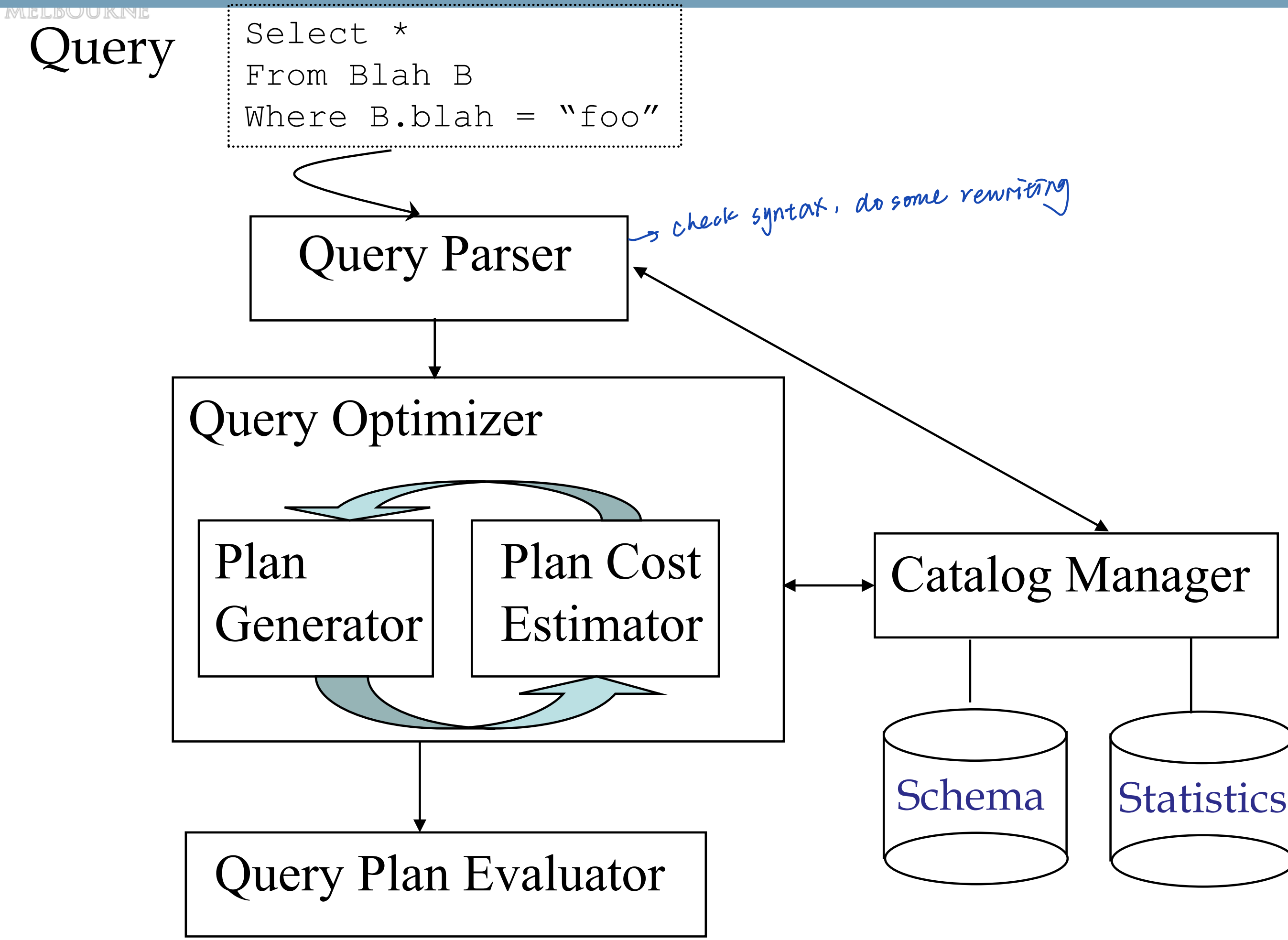
Coverage

MELBOURNE

- Overview
- Query optimization
- Cost estimation

Readings: Chapter 12 and 15, Ramakrishnan & Gehrke, Database Systems

Query Processing Workflow: Review



Query Optimization

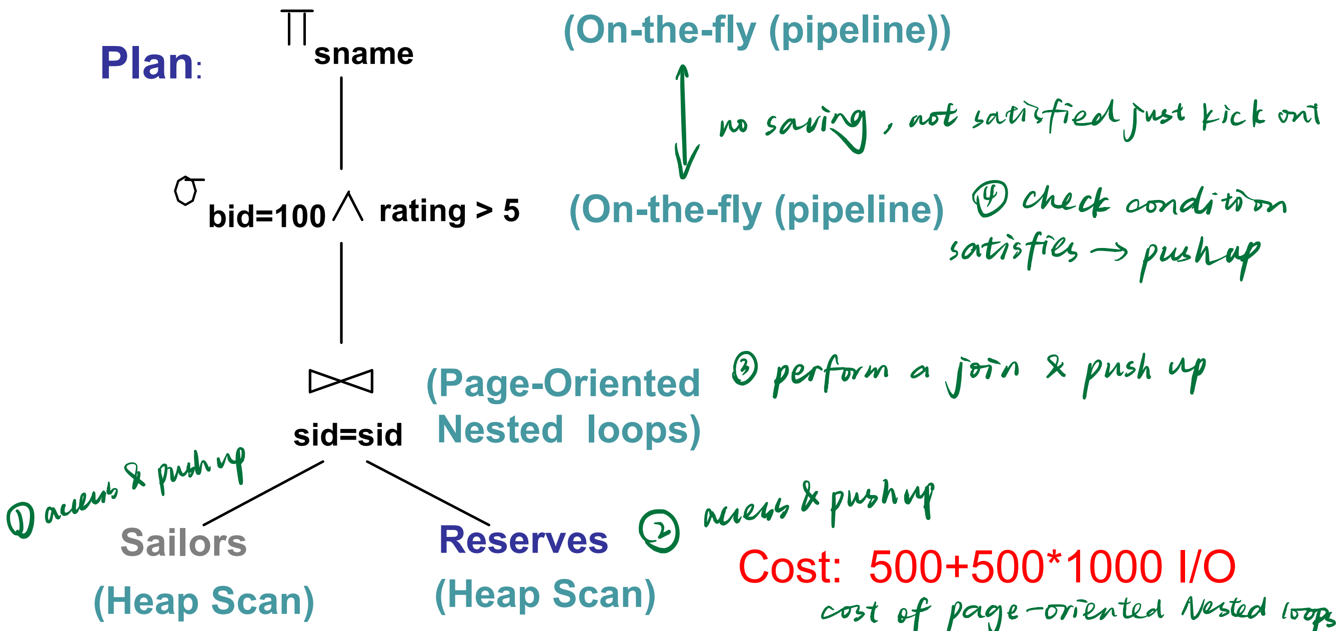
- Typically there are many ways of executing a given query, all giving the same answer
- Cost of alternative methods often **varies enormously**
- Query optimization aims to find the execution strategy with the lowest cost

- We will cover:
  - Relational algebra equivalences
  - Cost estimation
    - Result size estimation and reduction factors
  - Enumeration of alternative plans

Query Plan

- A tree, with relational algebra operators as nodes and access paths as leaves
- Each operator labeled with a choice of algorithm

SELECT sname from Sailors NATURAL JOIN Reserves
WHERE bid = 100 and rating > 5



Query Optimization

- Overview
- Query optimization
- Cost estimation

Readings: Chapter 15, Ramakrishnan & Gehrke, Database Systems

A Familiar Schema for Examples

Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)

Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

Boats (*bid*: integer, *bname*: string, *color*: string)

Query Optimization Overview

Example:

```
SELECT  S.sname
  FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

Query optimization steps:

1. Query first broken into “blocks” → every individual part start with SELECT is a block
2. Each block converted to relational algebra
3. Then, for each block, several alternative query plans are considered
4. Plan with the lowest estimated cost is selected

per block

Step 1: Break query into query blocks

- Query block is any statement starting with select
- Query block = unit of optimization
- Typically inner most block is optimized first, then moving towards outers

Outer block

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.age IN
    (SELECT  MAX (S2.age)
     FROM  Sailors S2
     GROUP BY  S2.rating)
```

Nested block

Step 2: Convert query block into relational algebra expression

Query:

```
SELECT  S.sid
FROM  Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND B.color = “red”
```

Relational algebra:

$$\pi_{S.sid}(\sigma_{B.color = \text{“red”}}(\text{Sailors} \bowtie \text{Reserves} \bowtie \text{Boats}))$$

Step 3: Relational Algebra Equivalences

sailors $\sigma_{age>50 \wedge rating=10} \longleftrightarrow \sigma_{rating=10}\ \sigma_{age>50}$ flip also ok

传递

- **Selections:** $\sigma_{c_1 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}\Big(\dots\big(\sigma_{c_n}(R)\big)\Big)$ (Cascade)

  $\sigma_{c_1}\big(\sigma_{c_2}(R)\big) \equiv \sigma_{c_2}\big(\sigma_{c_1}(R)\big)$ (Commute)

- **Projections:** $\pi_{a_1}(R) \equiv \pi_{a_1}\Big(\dots\big(\pi_{a_n}(R)\big)\Big)$ *(Cascade)*

  a_i is a set of attributes of R and $a_i \subseteq a_{i+1}$ for $i = 1 \dots n-1$

  $\Pi_{ID}\big(\Pi_{ID,age,name}(S)\big)$

- These equivalences allow us to 'push' selections and projections ahead of joins.

Examples

MELBOURNE

Selection:

$$\sigma_{age<18 \wedge rating>5} (Sailors)$$

$$\leftrightarrow \sigma_{age<18} (\sigma_{rating>5} (Sailors))$$

$$\leftrightarrow \sigma_{rating>5} (\sigma_{age<18} (Sailors))$$

Projection:

not a subset

$$\cancel{\pi_{age,rating} (Sailors) \leftrightarrow \pi_{age} (\pi_{rating} (Sailors))} ?$$

↑ lost age

$$\pi_{age,rating} (Sailors) \leftrightarrow \pi_{age,rating} (\pi_{age,rating,sid} (Sailors))$$

Another Equivalence

- A projection commutes with a selection that only uses attributes retained by the projection

$$\pi_{age,\ rating,\ sid}\ (\sigma_{age<18\ \wedge\ rating>5}\ (Sailors))$$

$$\leftrightarrow\ \sigma_{age<18\ \wedge\ rating>5}\ (\pi_{age,\ rating,\ sid}\ (Sailors))$$

$$\pi_{age,\ sid}\ (\sigma_{age<18\ \wedge\ rating>5}\ (Sailors))$$

?

$$\cancel{\leftrightarrow\ \sigma_{age<18\ \wedge\ rating>5}\ (\pi_{age,\ sid}\ (Sailors))}$$

lost rating

Equivalences Involving Joins

$R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$ *(Associative)*

$(R \bowtie S) \equiv (S \bowtie R)$ *(Commutative)*

- These equivalences allow us to choose **different join orders**

Mixing Joins with Selections & Projections

- Converting selection + cross-product to join

expensive

$$\sigma_{S.sid = R.sid}\ (\text{Sailors x Reserves})$$

cross product + selection → natural join

$$\leftrightarrow \text{Sailors} \bowtie_{S.sid = R.sid} \text{Reserves}$$

- Selection on just attributes of S commutes with R $\bowtie$ S

$$\sigma_{S.age<18}\ (\text{Sailors} \bowtie_{S.sid = R.sid} \text{Reserves})$$

first selection → then join

$$\leftrightarrow (\sigma_{S.age<18}\ (\text{Sailors})) \bowtie_{S.sid = R.sid} \text{Reserves}$$

- We can also "push down" projection (*but be careful…*)

$$\pi_{S.sname}\ (\text{Sailors} \bowtie_{S.sid = R.sid} \text{Reserves})$$

careful about the attribute keep

$$\leftrightarrow \pi_{S.sname}(\pi_{sname,sid}(\text{Sailors}) \bowtie_{S.sid = R.sid} \pi_{sid}(\text{Reserves}))$$

Query Optimization

MELBOURNE

- Overview
- Query optimization
- Cost estimation

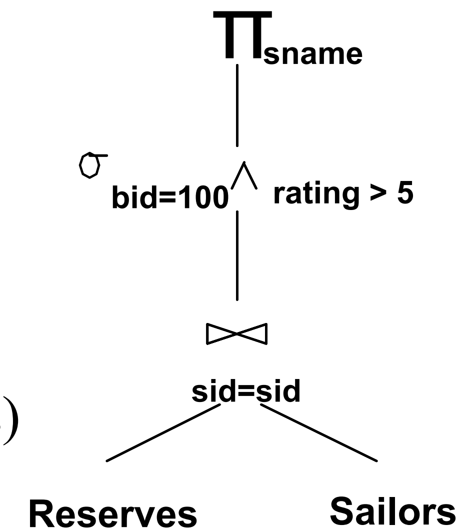
Readings: Chapter 15, Ramakrishnan & Gehrke, Database Systems

Recall: Query Optimization Overview

1. Query first broken into "blocks"
2. Each block converted to relational algebra
3. Then, for each block, several alternative query plans are considered
4. Plan with lowest estimated cost is selected

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```

$$\pi_{(sname)}\sigma_{(bid=100 \wedge rating > 5)} (Reserves \bowtie Sailors)$$



Cost-based Query Sub-System

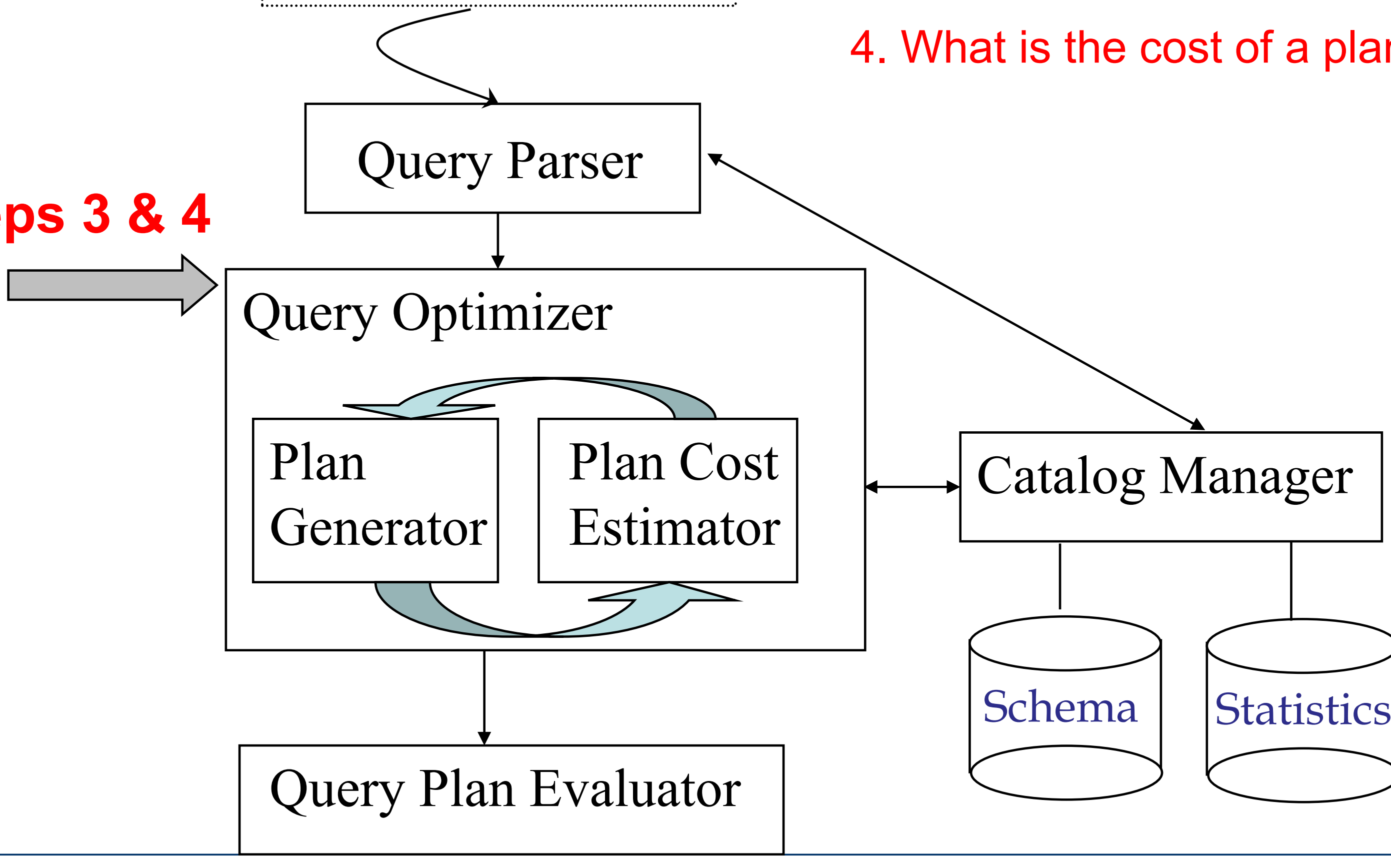
Queries

```
Select *
From Blah B
Where B.blah = “foo”
```

3. What plans are considered?

4. What is the cost of a plan?

Steps 3 & 4



Cost Estimation

- For each plan considered, must estimate cost:
  - Must *estimate size of result* for each operation in tree

    output for this query will be an input for next step
    - Use information about input relations (from the system catalogs), and apply rules (discussed next)
  - Must estimate *cost* of each operation in plan tree
    - Depends on input cardinalities
    - We've already discussed how to estimate the cost of operations (sequential scan, index scan, joins)
    - Next time we will calculate the cost of entire plans…

Statistics and Catalogs

- To decide on the cost, the optimizer needs information about the relations and indexes involved. This information is stored in the **system catalogs**.

keep metadata (description of data)

- **Catalogs** typically contain at least:
  - # tuples (**NTuples**) and # pages (**NPages**) per relation
  - # distinct key values (**NKeys**) for each index (or relation attribute)
  - low/high key values (**Low/High**) for each index (or relation attribute)
  - Index height (**Height(I)**) for each tree index
  - # index pages (**NPages(I)**) for each index

eg sailor ranking 1---10 Nkeys 10 (distinct key value) domain of value

- Statistics in catalogs are updated periodically

not necessary 100% accurate

Result size estimation

- Consider a query block:

```
SELECT  attribute list
  FROM  relation list
WHERE  predicate1 AND ... AND predicate_k
```

- Maximum number of tuples in the result is the product of the cardinalities of relations in the FROM clause

- Reduction factor (RF) associated with each predicate reflects the impact of the predicate in reducing the result size. RF is also called selectivity.

Result size estimation calculations

- Single table selection:

$$\textbf{ResultSize} = \mathrm{N}Tuples(R) \prod_{i=1..n} RF_i$$

- Joins (over k tables):

$$\textbf{ResultSize} = \prod_{j=1..k} NTuples(R_j) \prod_{i=1..n} RF_i$$

- If there are no selections (no predicates), reduction factors are simply ignored, i.e. they are ==1

Calculating Reduction Factors(RF)

- Depend on the type of the predicate:

1. Col = value

   RF = 1/NKeys(Col)

   # distinct value

2. Col > value

   RF = (High(Col) – value) / (High(Col) – Low(Col))

3. Col < value

   RF = (val – Low(Col)) / (High(Col) – Low(Col))

4. Col_A = Col_B (for joins)

   RF = 1/ (Max (NKeys(Col_A), NKeys(Col_B)))

   max of distinct value

5. In no information about Nkeys or interval, use a “magic number” 1/10

   RF = 1/10

ranking

1 10

① ranking : 5

$RF = \frac{1}{10}$

Nkey(ranking)

low val high

② col > value

$\frac{\text{high-value}}{\text{high-low}}$

③ col < value

$\frac{\text{value-low}}{\text{high-low}}$

Let's try it..

MELBOURNE

Sailors (S): NTuples(S) =1000, Nkeys(rating) = 10 interval [1-10], age interval [0-100], Nkeys(sid)=1000

$RF = \frac{1}{10}$ $\quad$ $RF = \frac{50}{100} = \frac{1}{2}.$

*SELECT * FROM Sailors WHERE rating = 3 AND age > 50;*

Calculate result size:

NTuple · RF(rating=3) · RF(age>50)

$= 1000 \times \frac{1}{10} \times \frac{1}{2} = 50$ tuple.

NTuples(S) = 1000

RF(rating) = 1/10 = 0.1

RF(age) = (100-50)/(100-0) = 0.5

ResultSize = NTuples(S)*RF(rating)*RF(age) ✓

= 1000*0.1*0.5= 50 tuples

What’s examinable

MELBOURNE

- What is query optimization/describe steps?
- Equivalence classes
- Result size estimation

- Important for Assignment 3 as well

Next Lecture

- Query optimization Part II
  - Plan enumeration