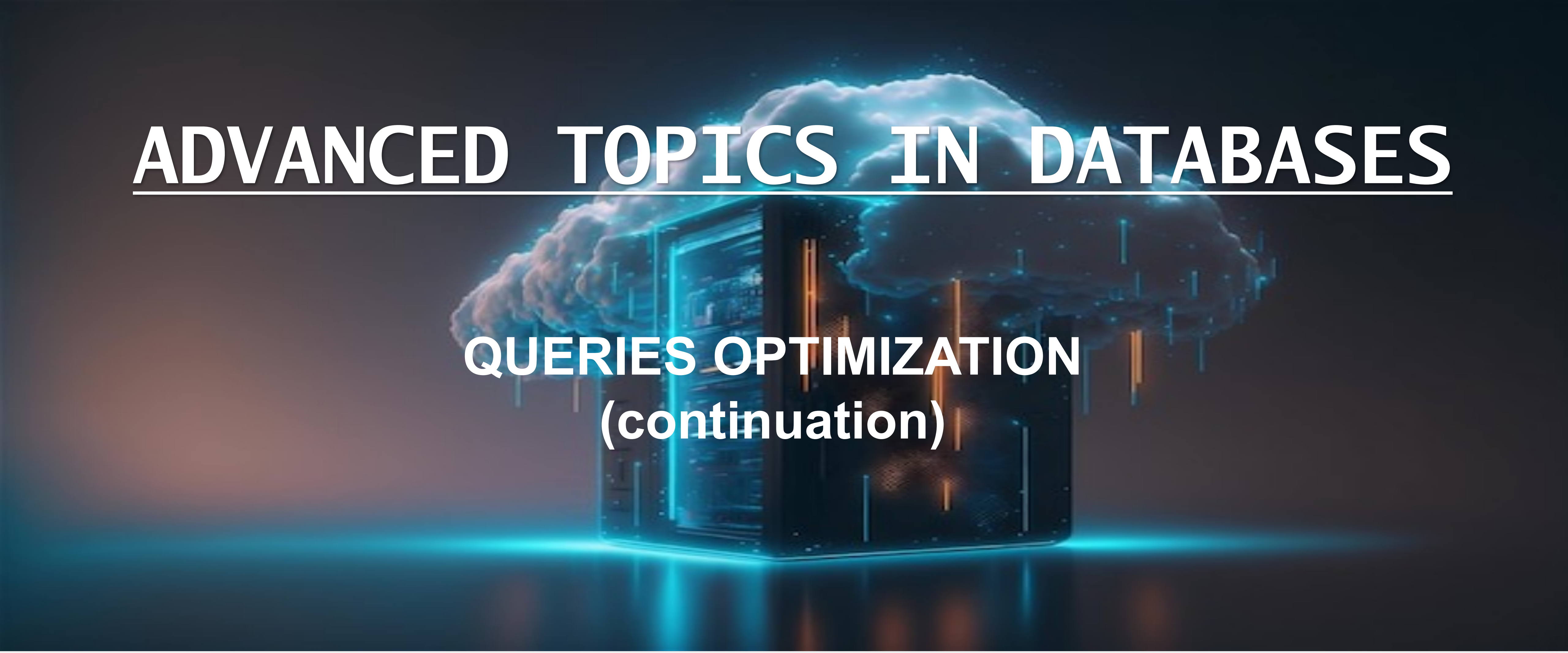


ADVANCED TOPICS IN DATABASES

QUERIES OPTIMIZATION
(continuation)

Master in Informatics Engineering
Data Engineering

Informatics Engineering Department

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Remarks - RBO

- Heurists query optimizer apply the Equivalence transformation rules to obtain an efficient plan;
- The Rules are:

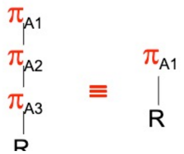
1. **Cascate of selection**

$$\sigma_{F1 \wedge F2}(E) \equiv \sigma_{F2}(\sigma_{F1}(E)) \equiv \sigma_{F1}(\sigma_{F2}(E))$$

2.Cascading projections

$$\pi_X(E) \equiv \pi_X(\pi_{X,Y}(E))$$

$$\pi_{at1,\, at2, \ldots, atn}(R) \equiv \pi_{at1}(R)$$



3.Anticipation of selection with respect to join (pushing selection down)

$$\sigma_F(E_1 \bowtie E_2) \equiv E_1 \bowtie (\sigma_F(E_2))$$

F is a predicate on attributes in E_2 only

Remarks - RBO

4.Anticipation of projection with respect to join

$$\pi_L (E_1 \bowtie_p E_2) \equiv \pi_L ((\pi_{L1,J} (E_1)) \bowtie_p (\pi_{L2,J} (E_2)))$$

$L_1 = L$ -Schema(E_2)

$L_2 = L$ -Schema(E_1)

J = set of attributes needed to evaluate join predicate p

$$\pi_{at1,at2,at3} (R \bowtie_c S) \equiv \pi_{at1,at2,at3}((\pi_{at1,at3}(R)) \bowtie_c (\pi_{at2,at3}(S)))$$

5.Join derivation from Cartesian product

$$\sigma_F (E_1 \times E_2) \equiv E_1 \bowtie_F E_2$$

predicate F only relates attributes in E_1 and E_2

6.Distribution of selection with respect to UNION

$$\sigma_F (E_1 \cup E_2) \equiv (\sigma_F (E_1)) \cup (\sigma_F (E_2))$$

Remarks - RBO

7.Distribution of selection with respect to difference

$$\sigma_F\ (E_1 – E_2) \equiv (\ \sigma_F\ \ (E_1)\ ) – (\ \sigma_F\ \ (E_2)\ )$$

$$\equiv (\ \sigma_F\ \ (E_1)) – E_2$$

8. Distribution of projection with respect to union

$$\pi_X\ \ (E_1 \cup E_2) \equiv (\ \pi_X\ (E_1)) \cup (\ \pi_X\ \ (E_2))$$

- **Can projection be distributed with respect to difference?**

$$\pi_X\ \ (E_1 - E_2) \equiv (\ \pi_X\ (E_1))\ -(\ \pi_X\ \ (E2))$$

This equivalence **only** holds if X **includes the primary key** or a **set of attributes with the same properties** (unique and not null)

Remarks - RBO

9.Other properties

$$\sigma_{F1 \vee F2}(E) \equiv (\sigma_{F1}(E)) \cup (\sigma_{F2}(E))$$

$$\sigma_{F1 \wedge F2}(E) \equiv (\sigma_{F1}(E)) \cap (\sigma_{F2}(E))$$

10.Distribution of join with respect to union

$$E \bowtie (E_1 \cup E_2) \equiv (E \bowtie E_1) \cup (E \bowtie E2)$$

- All binary operators are commutative and associative **except for difference**

Example

Sailors(sid, sname,rating, age)

Reserves(sid, bid, day, rname)

SQL

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

Relational Algebra

$$\pi_{\text{sname}}\ (\sigma_{\text{reserves.sid= sailors.sid and bid=100}}\ (\text{ reserves x sailors }))$$

Example

π_{sname} ($\sigma_{reserves.sid= sailors.sid \text{ and } bid=100}$ (reserves x sailors))

Prop 1 – **Cascade of selection**

π_{sname} ($\sigma_{bid=100}$ ($\sigma_{reserves.sid= sailors.sid}$ (reserves x sailors))

Prop 5 - **Join derivation from Cartesian product**

π_{sname} ($\sigma_{bid=100}$ (reserves ⋈ sailors)

Example

$$\pi_{sname}\ (\sigma_{bid=100}\ (reserves \bowtie sailors\ )$$

Prop 3 . **Anticipation of selection with respect to join (pushing selection down)**

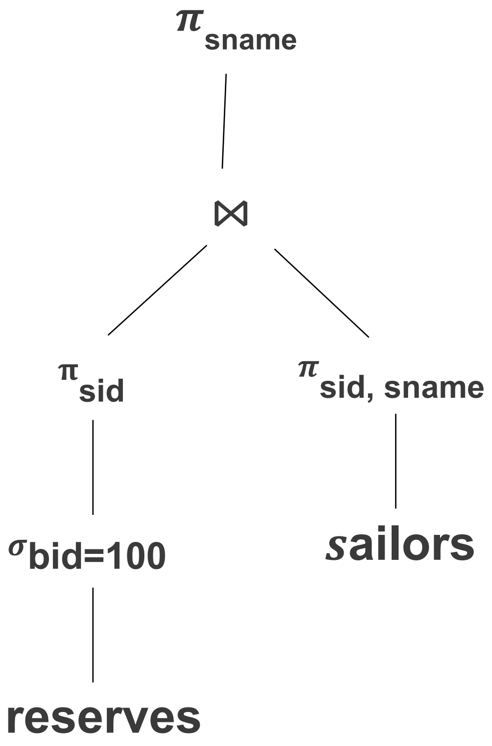
$$\pi_{sname}\ ((\sigma_{bid=100}\ (reserves)) \bowtie sailors\ )$$

Prop 2 - **Cascading projections** and Prop 4 - **Anticipation of projection with respect to join**

$$\pi_{sname}\ (\pi_{sid}(\sigma_{bid=100}\ (reserves)) \bowtie (\pi_{sid,\ sname}\ (Sailors\ )))$$

Example

$$\pi_{sname}\left(\pi_{sid}\left(\sigma_{bid=100}(reserves)\right) \bowtie \left(\pi_{sid,\ sname}(Sailors)\right)\right)$$



Hints

Some general steps follow:

- **Perform the SELECTION process foremost in the query.**
  - This should be **the first action** for any SQL table.
    - By doing so, we can decrease the number of records required in the query, rather than using all the tables during the query - **(push selections as far down as possible)**
- **Perform** all the projection as soon as achievable in the query.
  - Somewhat like a selection but this method helps in decreasing the number of columns in the query.
  - **Push projections as far down as possible**
- **Combine selections and Cartesian products to an appropriate join**

Hints

- **Perform the most restrictive joins and selection operations**.
  - What this means is that select only those sets of tables and/or views which will result in a relatively lesser number of records and are extremely necessary in the query.
  - Obviously, any query will execute better when tables with few records are joined.

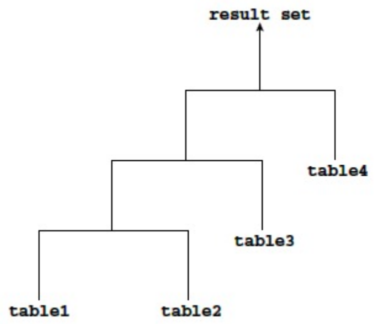
- **If the selection condition consists of several parts (AND or OR),**
  - **split into multiple selections and**
  - **push each one as far down the tree as possible**

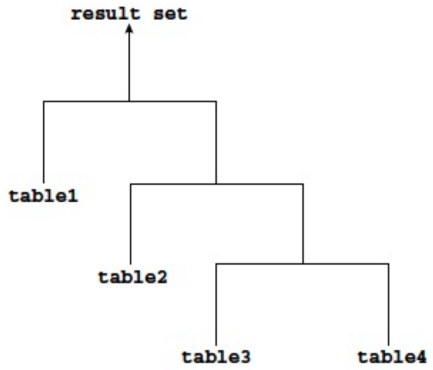
Attention: Don't ruin indexing: Pushing projection past a selection can ruin the use of indexes in the selection!

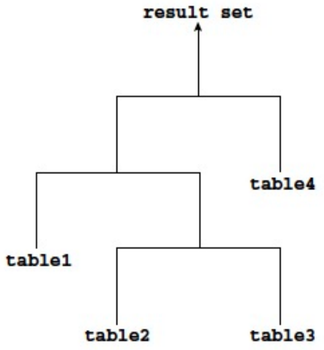
Joins order can have an impact on query performance

Joins

- The optimizer processes the join from left to right.
- The input of a join can be the result set from a previous join.
  - If the right child of every internal node of a join tree is a table, then the tree is **a left deep join tree.**
  - If the left child of every internal node of a join tree is a table, then the tree is called a **right deep join tree**,
  - If the left or the right child of an internal node of a join tree can be a join node, then the tree is called **a bushy join tree**.







Join Order

- An important task in query evaluation **is to determine a good join order**
- Many systems (including Oracle) construct only QEPs which start with one table, join it with a second, join the result with a third, and so on:

$$\mathbf{(R1 \bowtie R2) \bowtie R3) \bowtie R4}$$

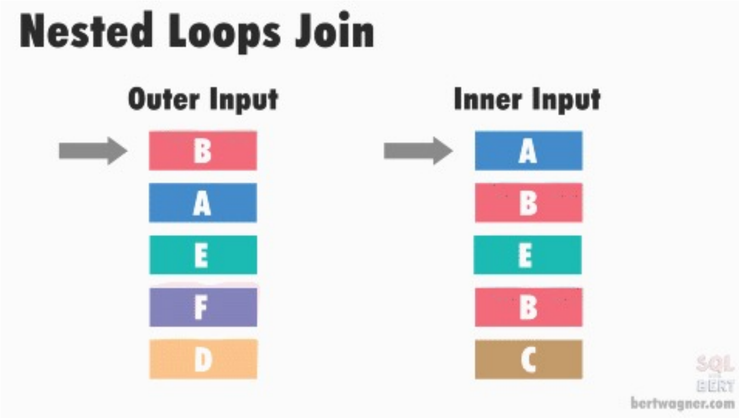
- These joins give normally good results.
- In rare situations, a "bushy" join would be better: $\mathbf{(R1 \bowtie R2) \bowtie (R3 \bowtie R4)}$

  E.g. R1 and R4 are very small and R2 and R3 are large. The join with R1 and R4 might reduce the size of R2 and R3 (like σ).

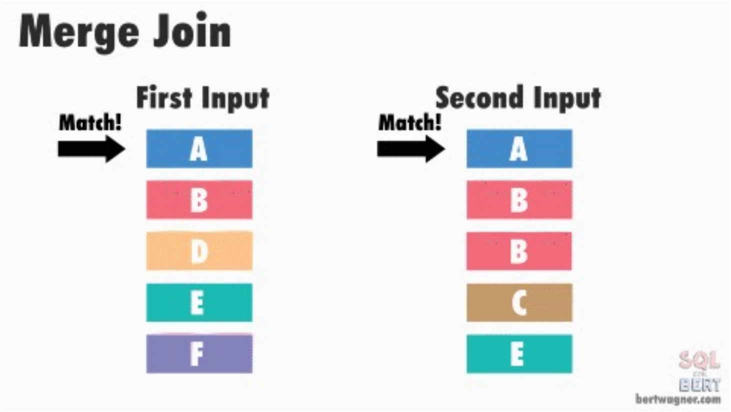
The **optimizer explores different permutations of joins** to find the most efficient arrangement, considering factors like **table sizes, indices, and estimated costs.**

Join Methods

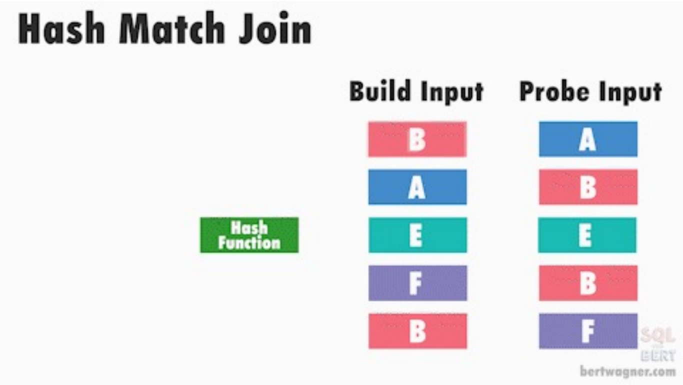
➢ The optimizer selects the most suitable algorithm for each join.



takes the first value from our first table and compares it to every value in our second "inner" table



require that data from both inputs is sorted



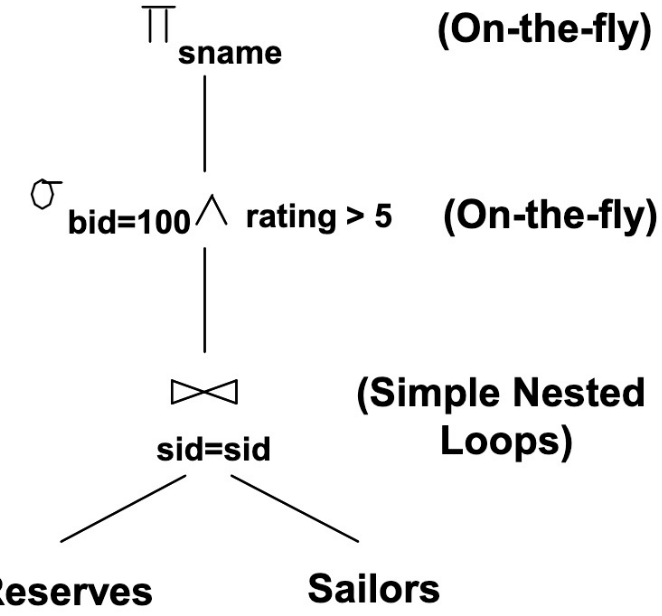
Hash join is used to find the matching in two tables with a hash table

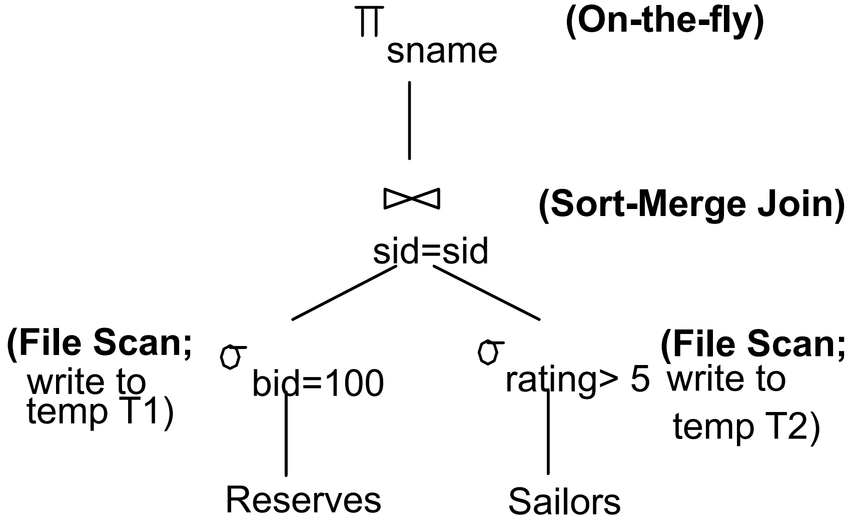
The query optimizer **considers** various components such as **join ordering, algorithms, conditions, indices, and costs to create an efficient execution plan**

Join Methods

Example

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





Plans for Joins

How the Optimizer Chooses Execution

generates a set of execution plans

- according to possible
- **join orders,**
- **join methods, and**
- **available access paths**.

Chooses the one with the lowest cost;

- first determines whether joining two or more tables results in a row source containing at most one row. (**recognizes based on unique e primary key**)

➢ The optimizer estimates the cost of a query plan by computing **the estimated I/Os and CPU**

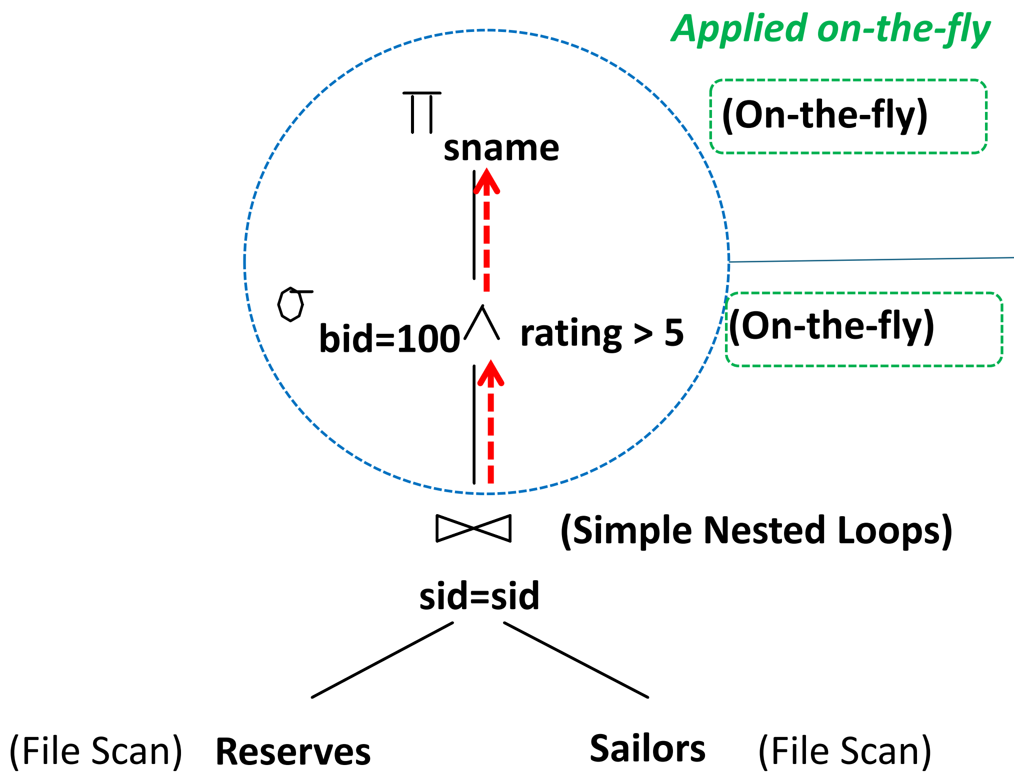
Plans for Joins

How the Optimizer Chooses Execution

- **The cost of a nested loops join** depends on the cost of reading each selected row of the outer table and each of its matching rows of the inner table into memory.
  - The optimizer estimates these costs using statistics in the data dictionary.
- **The cost of a sort merge join** depends largely on the cost of reading all the sources into memory and sorting them.
- **The cost of a hash join** largely depends on the cost of building a hash table on one of the input sides to the join and using the rows from the other side of the join to probe it.

Pipelining vs. Materializing

➢ When a query is composed of several operators, the result of one operator can sometimes be *pipelined* to another operator



Pipeline the output of the join into the selection and projection that follow

≠

In contrast, a temporary table can be *materialized* to hold the *intermediate result* of the join and *read back* by the selection operation!

Query Otimization

Query Otimization – Exercise

- Consider the following SQL query that finds all applicants who want to major in CSE, live in Seatle and go to a school ranked less than 10 (i.e., rank < 10).

Relation	Cardinality	Number of pages	Primary key
Applicants (id, name, city, sid)	2,000	100	id
Schools (sid, sname, srank)	100	10	sid
Major (id, major)	3,000	200	(id,major)

SELECT A.name

FROM Applicants A, Schools S, Major M

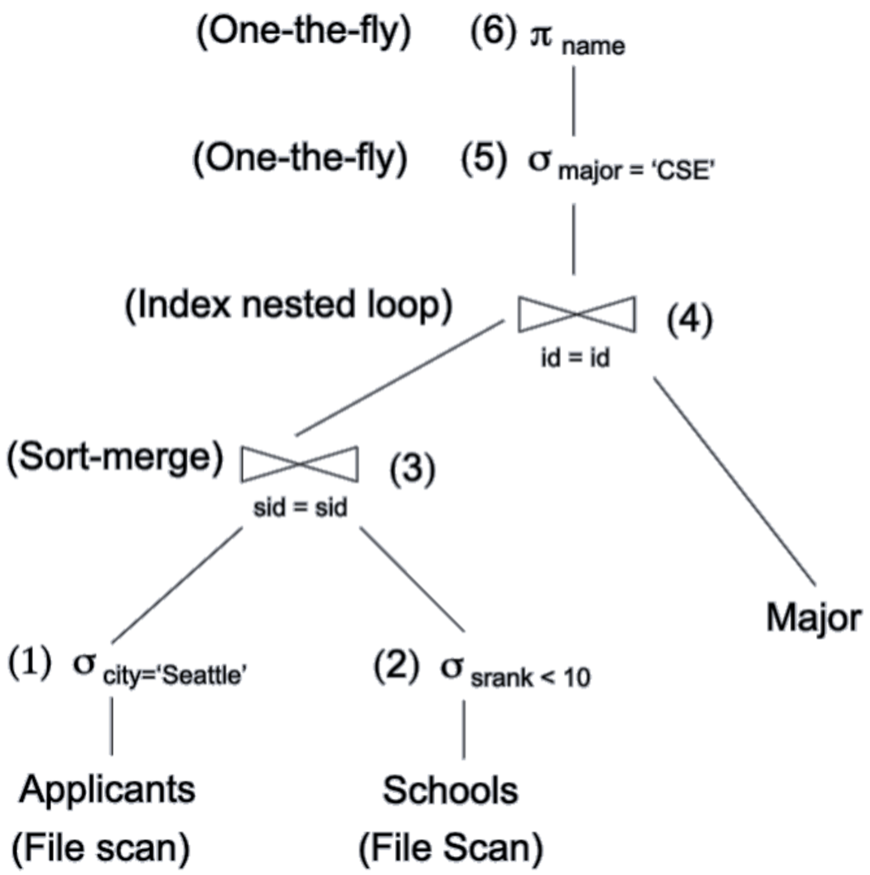
WHERE A.sid = S.sid AND A.id = M.id

AND A.city = ’Seattle’ AND S.rank < 10 AND M.major = ’CSE’

Create the query execution plan

Query Otimization – Exercise

Resolution



Query Otimization – Exercise

Let the database of a bookstore be represented by the following relational schema:

Editora(CodEditora,NomeEditora)

Livro(CodLivro,Titulo,Autor,Assunto,AnoPub,CodEditora)

Instituicao(CodInst,NomeInst,Sigla,Local)

Adotado-por(CodLivro,CodInst,AnoAdocao)

```
SELECT NomeInst FROM Instituicao
WHERE CodInst IN
        (SELECT CodInst FROM Adotado-por
        WHERE AnoAdocao = ‘2007’ AND CodLivro IN
                (SELECT CodLivro FROM Livro
                 WHERE Assunto = ‘Portugues’ AND CodEditora IN
                                (SELECT CodEditora FROM Editora
                                WHERE NomeEditora = ‘Editora Campus’)))
```

References

Slides based on

- Database Systems A Practical Approach to Design, implementation, Management, Thomas Connolly, Carolyn Begg;

- The Oracle Optimizer Explain the Explain Plan