Introduction to Database Systems

Radim Bača

Department of Computer Science, FEECS

radim.baca@vsb.cz

dbedu.cs.vsb.cz

Content

- CASE
- Complex SQL tasks:
  - Conditional aggregation
  - Intersection
  - Greatest per group
  - Greatest aggregation
- SQL query processing

CTU Open

- CTU Open
- 18. - 19.10.2019

Test 1

- Zadání v roce 2016/17[1]
- Konzultace před testem - úterý 22.10. 12:30 na EB113
- Během testu nebude možné používat prohlížeč, pouze soubory, které si na test přinesete a otevřete jinak než v prohlížeči
- Budou se používat pouze pracovní stanice na učebně (vlastní PC jen v krajním případě nefunkční pracovní stanice)
- Data pro test budou vygenerována znova (je tam chyba v pieces)

[1] https://dbedu.cs.vsb.cz/course/50/Uvod-do-databazovych-systemu-2016-2017

CASE

- Allows us to write a procedural condition
- Used mainly in the SELECT statement

```
SELECT *, 'succeed'
FROM studies
WHERE gained_points > 50
   UNION ALL
SELECT *, 'fail'
FROM studies
WHERE gained_points <= 50
```

CASE

- Allows us to write a procedural condition
- Used mainly in the SELECT statement

```
SELECT *, 'succeed'
FROM studies
WHERE gained_points > 50
   UNION ALL
SELECT *, 'fail'
FROM studies
WHERE gained_points <= 50
```

?

$\Leftrightarrow$

```
SELECT *,
   CASE WHEN gained_points > 50
      THEN 'succeed'
      ELSE 'fail'
   END result
FROM studies
```

CASE

- Allows us to write a procedural condition
- Used mainly in the SELECT statement

```
SELECT *, 'succeed'
FROM studies
WHERE gained_points > 50
   UNION ALL
SELECT *, 'fail'
FROM studies
WHERE gained_points <= 50
   OR gained_points is null
```

$\Leftrightarrow$

```
SELECT *,
   CASE WHEN gained_points > 50
      THEN 'succeed'
      ELSE 'fail'
   END result
FROM studies
```

We have to consider NULL values!

Conditional aggregation

- By the term "conditional aggregation" we usually mean certain type of a solution using aggregation function + CASE
- *For each student find a number subjects studied in 2010 and 2011*

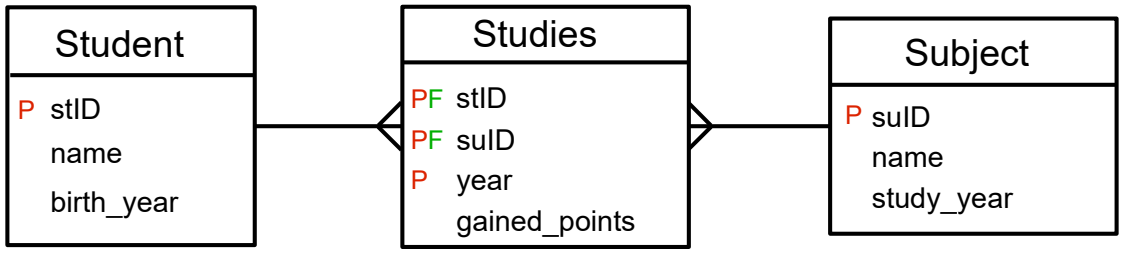
```
SELECT st.name,
   COUNT(CASE WHEN se.year = 2010 THEN 1 END),
   COUNT(CASE WHEN se.year = 2011 THEN 1 END)
FROM student st
LEFT JOIN studies se ON st.stID = se.stID
GROUP BY st.name
```

Conditional aggregation

- By the term "conditional aggregation" we usually mean certain type of a solution using aggregation function + CASE
- *For each student find a number subjects studied in 2010 and 2011*

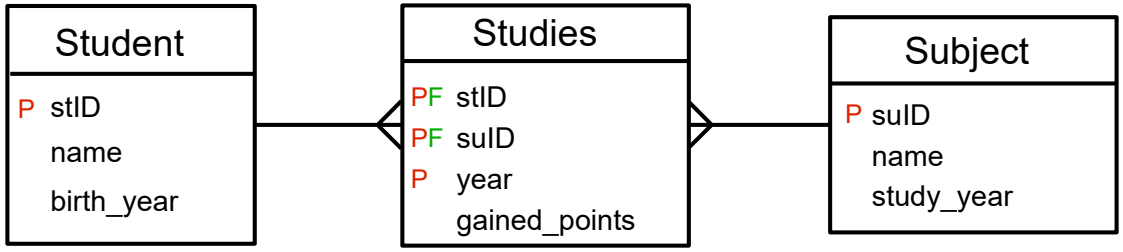
```
SELECT st.name,
   COUNT(CASE WHEN se.year = 2010 THEN 1 END),
   COUNT(CASE WHEN se.year = 2011 THEN 1 END)
FROM student st
LEFT JOIN studies se ON st.stID = se.stID
GROUP BY st.name
```

Intersection



- Intersection was already mentioned several times
- Let us show two alternative SQL syntax dealing with intersection
- *Find all students who study or studied both subjects with suIDs 1 and 5.*

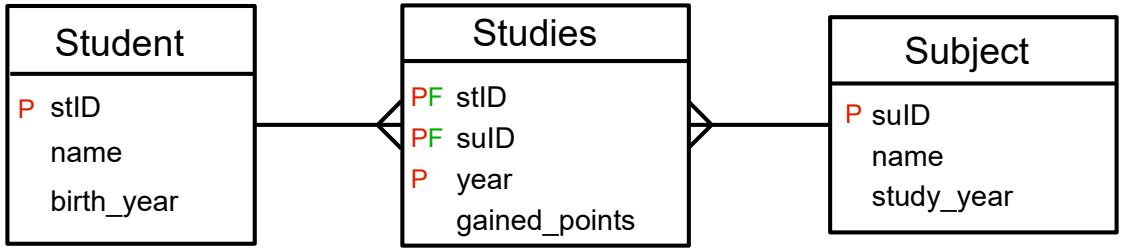
Intersection - HAVING COUNT DISTINCT



- *Find all students who study or studied both subjects with suIDs 1 and 5.*

```
SELECT st.name
FROM student st
JOIN studies se ON st.stID = se.stID
WHERE se.suID IN (1,5)
GROUP BY st.stID, st.name
HAVING COUNT(distinct se.suID) = 2
```

Intersection - HAVING COUNT DISTINCT

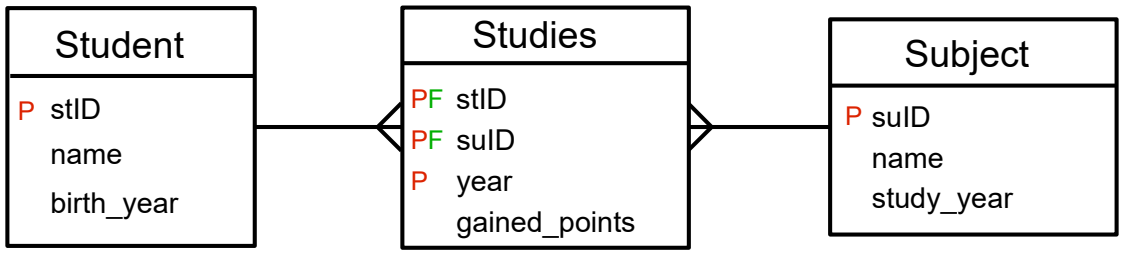


- *Find all students who study or studied both subjects with suIDs 1 and 5.*

```
SELECT st.name
FROM student st
JOIN studies se ON st.stID = se.stID
WHERE se.suID IN (1,5)
GROUP BY st.stID, st.name
HAVING COUNT(distinct se.suID) = 2
```

What kind of result we have if we omit HAVING clause?

Intersection - Self Join



- *Find all students who study or studied both subjects with suIDs 1 and 5.*

```
SELECT DISTINCT st.name
FROM student st
JOIN studies se1 ON st.stID = se1.stID
JOIN studies se2 ON st.stID = se2.stID
WHERE se1.suID = 1 and se2.suID = 5
```

Greatest Per Group

Studies

sID	pID	year	points
1	35	2010	23
8	35	2010	89
7	21	2010	89
2	46	2011	59
3	1	2011	69
21	28	2011	91
5	46	2012	2
3	1	2012	99

- For example find students with a maximum number of points per year
- **Group** - what exactly is a group?
- **Greatest** - the aggregation does not have to be always max!

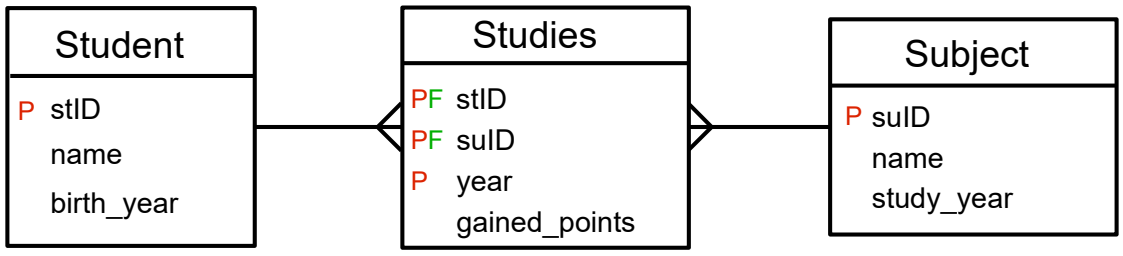
Greatest Per Group

Studies

sID	pID	year	points
1	35	2010	23
8	**35**	**2010**	**89**
7	**21**	**2010**	**89**
2	46	2011	59
3	1	2011	69
21	**28**	**2011**	**91**
5	46	2012	2
3	**1**	**2012**	**99**

- For example find students with a maximum number of points per year
- **Group** - what exactly is a group?
- **Greatest** - the aggregation does not have to be always max!
- We are interested about the whole rows not only aggregates

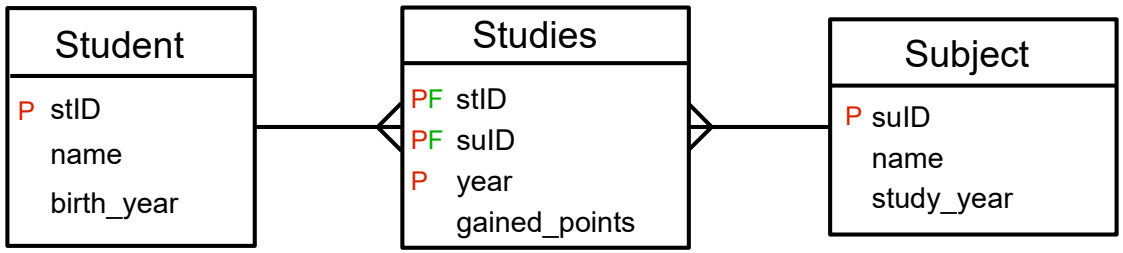
Greatest Per Group - Subquery + Join



- *Find students with a maximum number of points per year*

```
SELECT se.*
FROM studies se
JOIN (
    SELECT se.year, MAX(gained_points) max_gp
    FROM studies se
    GROUP BY se.year
) t on se.year = t.year and
        se.gained_points = t.max_gp
```

Greatest Per Group - Not Exists



- *Find students with a maximum number of points per year*

```
SELECT *
FROM studies se1
WHERE NOT EXISTS(
    SELECT 1
    FROM studies se2
    WHERE se1.year = se2.year
      and se1.gained_points < se2.gained_points
)
```

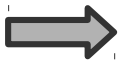
Greatest Per Group

- The previous solutions are not completely equivalent
- There is another popular solution using `row_number()` window function for this problem

Greatest aggregation

Studies

sID	pID	year	points
1	35	2010	23
8	35	2010	89
7	21	2010	89
2	46	2011	59
3	1	2011	69
21	28	2011	91
5	46	2012	2
3	1	2012	99



SUM(points) per year

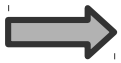
year	SUM(points)
2010	201
2011	219
2012	101

- We perform an aggreagtion per group

Greatest aggregation

Studies

sID	pID	year	points
1	35	2010	23
8	35	2010	89
7	21	2010	89
2	46	2011	59
3	1	2011	69
21	28	2011	91
5	46	2012	2
3	1	2012	99

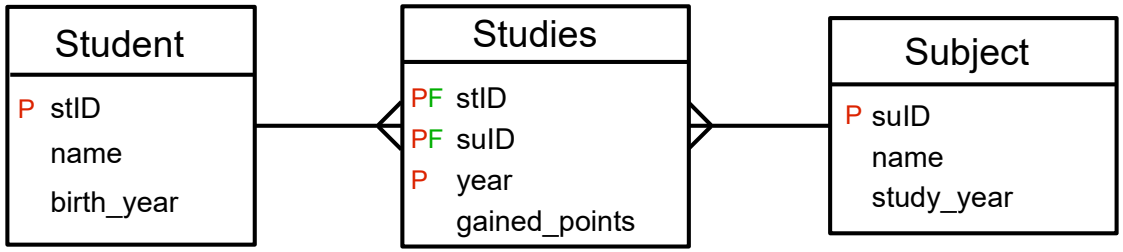


SUM(points) per year

year	SUM(points)
2010	201
2011	**219**
2012	101

- We perform an aggreagtion per group
- and we are looking for the groups with highest aggregation result

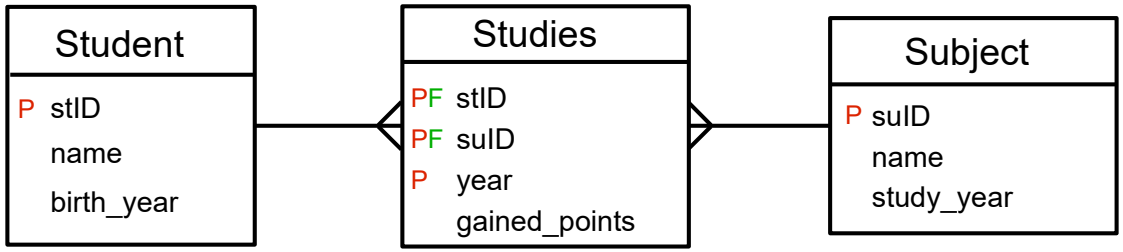
Greatest aggregation - Having



- *Find the students with highest number of subjects*

```
SELECT st.stID, COUNT(se.stID)
FROM student st
JOIN studies se ON st.stID = se.stID
GROUP BY st.stID
HAVING COUNT(se.stID) >= all(
    SELECT COUNT(se.stID)
    FROM student st
    JOIN studies se ON st.stID = se.stID
    GROUP BY st.stID
)
```

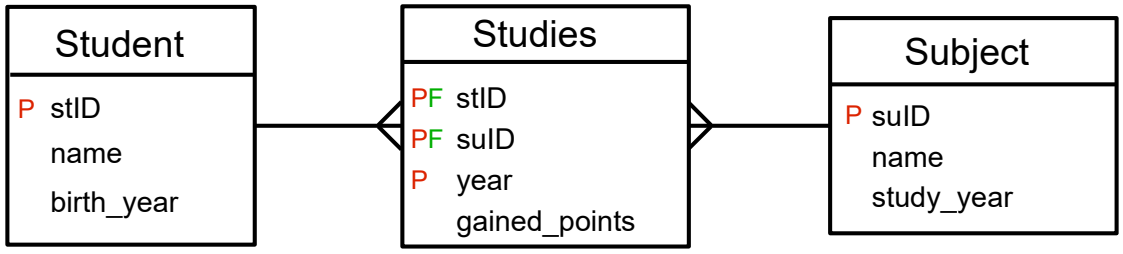
Greatest aggregation - Having



- *Find the students with highest number of subjects*

```
SELECT st.stID, COUNT(se.stID)
FROM student st
JOIN studies se ON st.stID = se.stID
GROUP BY st.stID
HAVING COUNT(se.stID) >= all(
    SELECT COUNT(se.stID)
    FROM student st
    JOIN studies se ON st.stID = se.stID
    GROUP BY st.stID
)
```

Greatest aggregation - Having



- *Find the students with highest number of subjects*

```
SELECT st.stID, COUNT(se.stID)
FROM student st
JOIN studies se ON st.stID = se.stID
GROUP BY st.stID
HAVING COUNT(se.stID) >= all(
    SELECT COUNT(se.stID)
    FROM student st
    JOIN studies se ON st.stID = se.stID
    GROUP BY st.stID
)
```

- We have the same subquery twice in the SQL!

With

- SQL enables to use so-called common table expressions (CTE)
- It reduces redundancy in SQL code

```
SELECT st.stID, COUNT(se.stID)
FROM student st
JOIN studies se
    ON st.stID = se.stID
GROUP BY st.stID
HAVING COUNT(se.stID) >= all(
    SELECT COUNT(se.stID)
    FROM student st
    JOIN studies se
    ON st.stID = se.stID
    GROUP BY st.stID
)
```

→

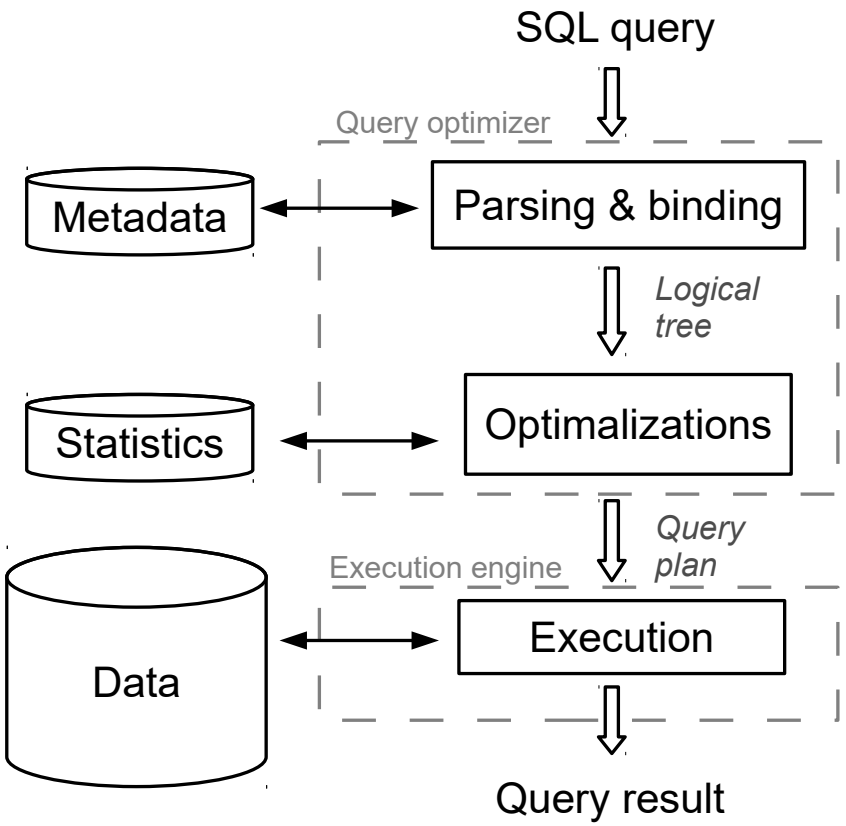
```
WITH studentCounts as (
    SELECT st.stID,
       COUNT(se.stID) counts
    FROM student st
    JOIN studies se
       ON st.stID = se.stID
    GROUP BY st.stID
)
SELECT *
FROM studentCounts
WHERE counts >= all(
    SELECT counts
    FROM studentCounts
)
```

CTE

- CTE may simplify notation and avoid redundancy
- However, some database systems evaluate the CTE first and store the result [2]
- This may cause problems for certain queries

[2] https://www.postgresql.org/docs/10/queries-with.html

SQL Query Compilation



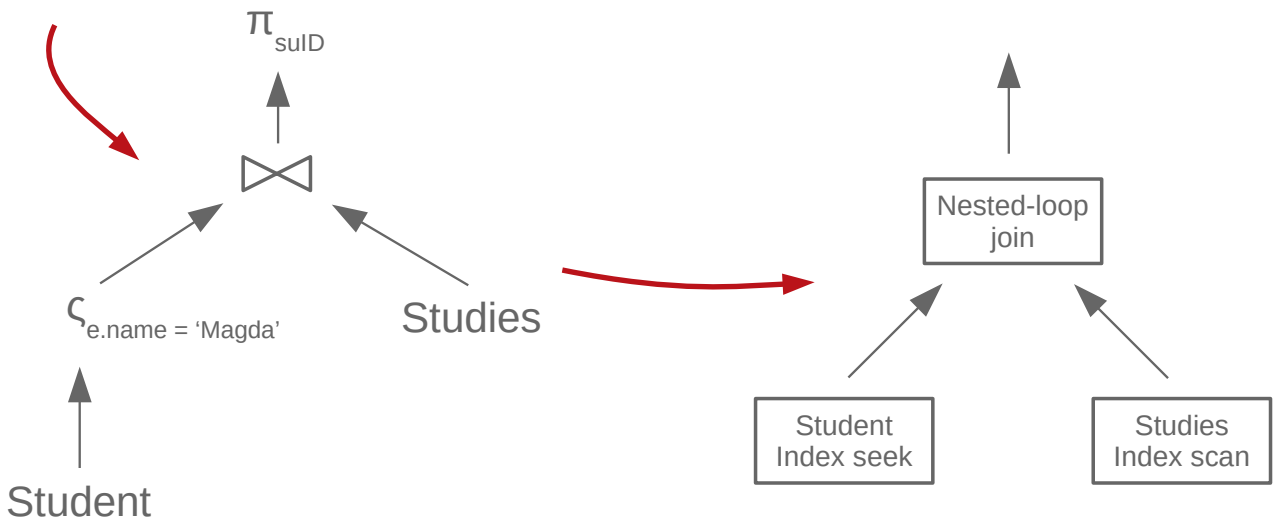
- SQL query optimization is a process which takes a SQL and output an query plan
- The query plan is stored in a plan cache
- The process of plan creation should be deterministic

Query plan

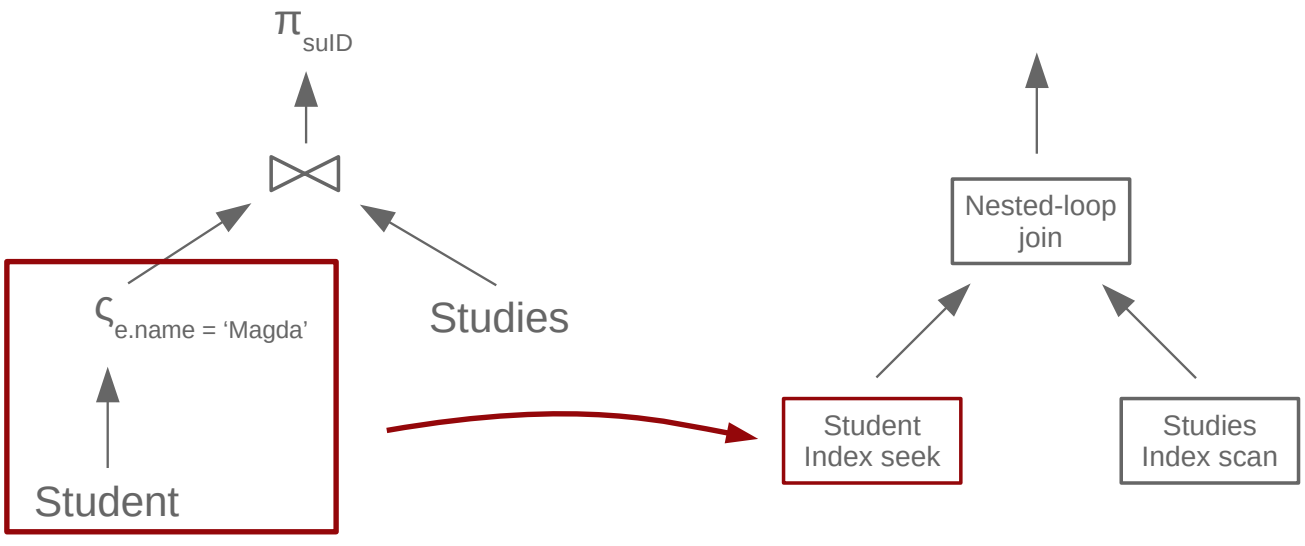
- **Query plan** is a tree where nodes
- Node is an operator and edge represents a fact that output of one node is input of another node
- We recognize two major types of plans:
  - Logical tree - typically a relational algebra
  - Physical (query plan) - specific algorithms

Query plan

- ```
  SELECT * FROM Student st
  JOIN Studies se on st.stID = se.stID
  WHERE name = 'Petr'
  ```

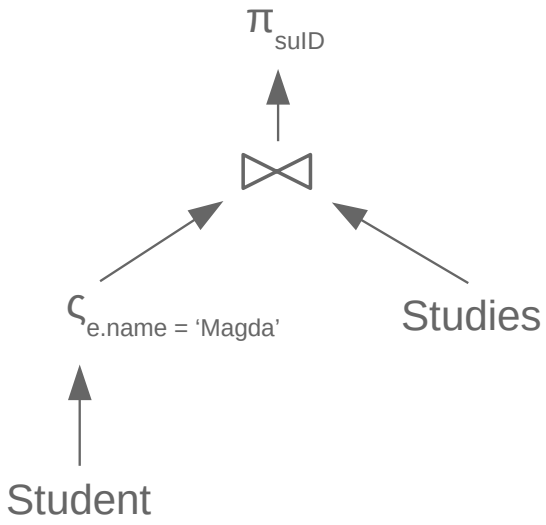


a) Logical tree

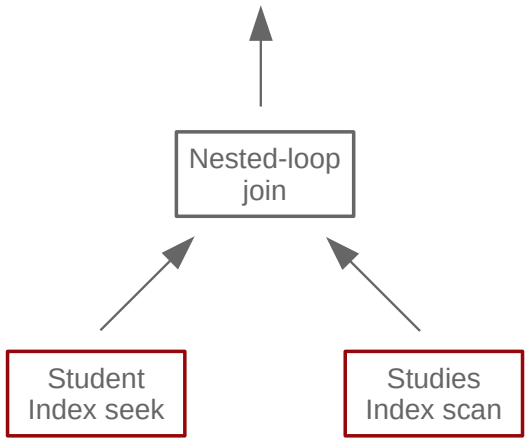
b) Query plan
  ```

Query plan

- ```
  SELECT * FROM Student st
  JOIN Studies se on st.stID = se.stID
  WHERE name = 'Petr'
  ```



a) Logical tree

b) Query plan
  ```

Query plan

- The important aspect of every plan are data access operators
- Two major types of data access op. are *Scan* and *Index Seek*
- Rule of thumb: *We should avoid scan in large tables*



a) Logical tree



b) Query plan

References

- Course home pages `http://dbedu.cs.vsb.cz`