



CSE1505 : Information & Data Management
Q3 2021/22
Assignment #3

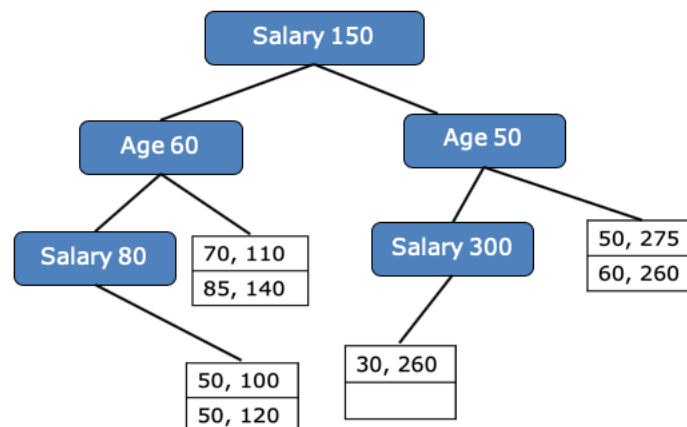
Cristoph Lofi, Asterios Katsifodimos & Christos Koutras

ReLaX: Relational Algebra Calculator

ReLaX is a web-based tool for calculating relational algebra queries on a database schema. You can find it on <https://dbis-uibk.github.io/relax/calc/local/uibk/local/0>. Take the short tour and play around a little (unless you are familiar with it since Assignment #2!), in order to familiarize yourself with the tool. For **Task 2**, you have to import the dataset referring to the given schema by browsing to this link: <https://dbis-uibk.github.io/relax/calc/gist/8218dddee54952ca3f254505cf78cda2>. If the data is loaded successfully, ReLaX will show the tables on the left.

Task 1: KD-Tree

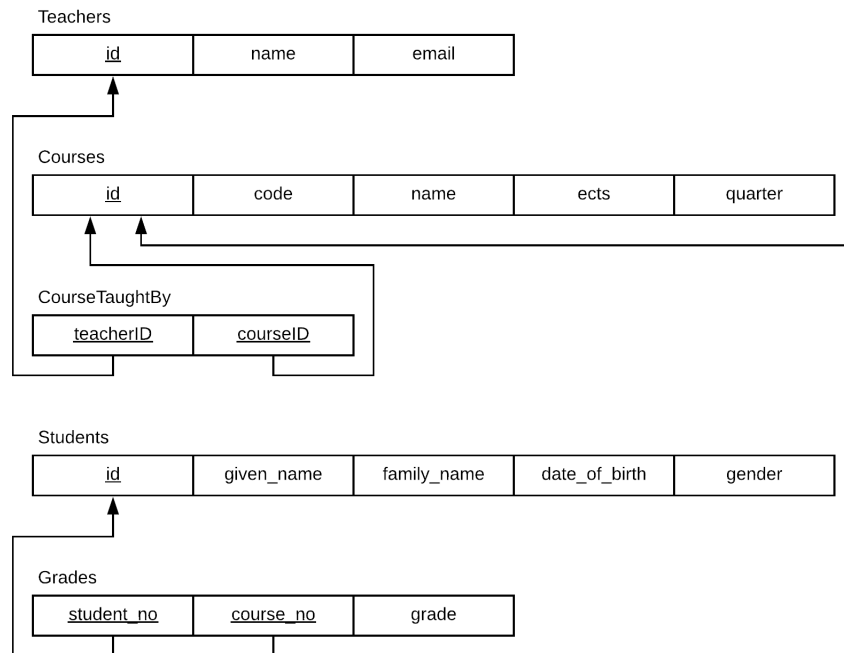
The following data structure below represents a KD-Tree with two dimensions, namely Salary and Age. The left of a given node in the tree have ' $<$ ' semantics, and the right of a key have ' \geq ' semantics. This exercise requires that you simply insert two values to the tree.



Task: Draw how the below tree looks like after inserting the value: (65,110)

Task 2: Algebraic Hill Climbing - University

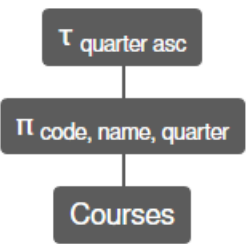
Consider the following schema:



Before starting, let's see how ReLaX translates a SQL query into relational algebra, draws an operator tree and visualizes the results. Open ReLaX, using the university schema and switch to the SQL tab. Then execute the following query:

```
SELECT DISTINCT code, name, quarter
FROM Courses
ORDER BY quarter
```

The output you get, should look like the following:



$\tau_{\text{quarter asc}}$
 $\pi_{\text{code, name, quarter}}$
 Courses

$\tau_{\text{quarter asc}}$	$\pi_{\text{code, name, quarter}}$	Courses
Courses.code	Courses.name	Courses.quarter
CSE1100	Object-oriented programming	1
CSE1300	Reasoning and Logic	1
CSE1400	Computer Organisation	1
CSE1200	Calculus	2
CSE1305	Algorithms and Data Structures	2
CSE1405	Computer Networks	2
CSE1505	Information and Data Management	2
CSE1105	OOP Project	3
CSE1205	Linear Algebra	3
CSE1500	Web and Database Technology	3
CSE1110	Software Quality and Testing	4
CSE1210	Probability Theory and Statistics	4

In this assignment, you will have to modify a query in its algebraic form. When you switch to the Relational Algebra tab and enter the following formula, you'll see the results are the same as before:

```

τ quarter asc
  (π code, name, quarter
    (Courses)
  )

```

This task requires you to apply the heuristics as described in the lecture (see Algebraic Hill-Climbing). This method consists of six steps. For the first five ones, use ReLaX to verify your algebraic formula (for all steps, the formula should be semantically equivalent, i.e., the results should be the same!) and visualize it to the operator tree. For the sixth step, you may need to use another tool in addition.

The following relational-algebraic formula needs to be optimized using Algebraic Hill-Climbing:

```
 $\pi$  Students.given_name, Students.family_name, Courses.code, Courses.name, Grades.grade
(σ Students.student_no = Grades.student_no ∧ Courses.quarter = 1 ∧
  Grades.course_no = Courses.id ∧ Grades.grade ≥ 9
  (Students ×
    (Grades × Courses)
  )
)
```

Your task is to execute each of the following 6 steps. After each one of them, include in your final report:

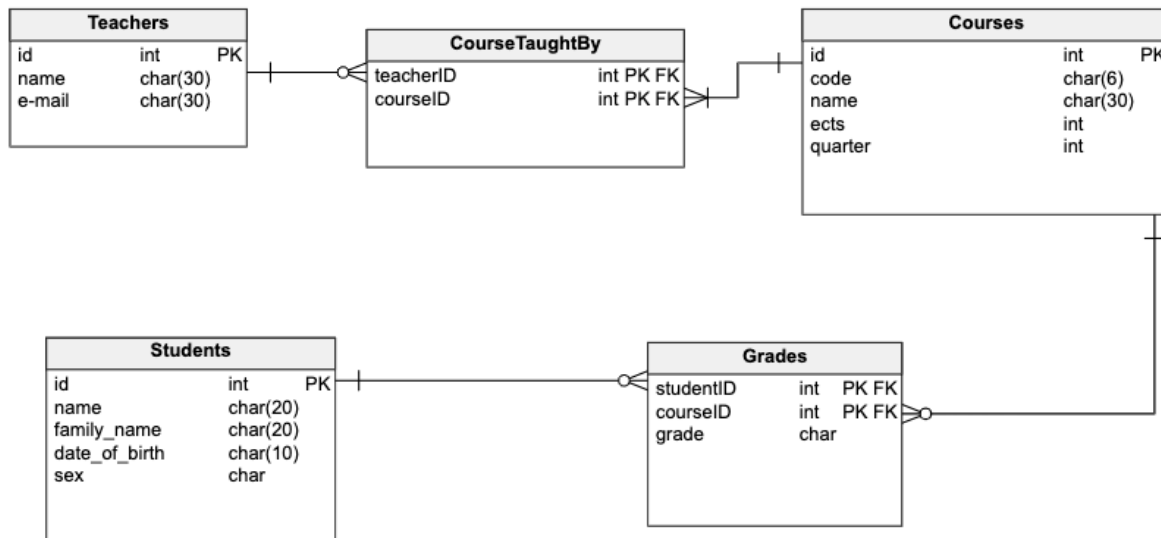
- The relational algebraic formula, except for step 6.
- A screenshot of the respective operator tree. For our convenience, please crop your screenshots, so that they only include the trees! In step 6 highlight the blocks for which you prepare pipelining.

Thus, you should have 5 formulas and 6 operator trees in your final report for this task.

1. Break up selections
2. Push selections as far as possible
3. Break, Eliminate, Push and Introduce Projection. Try to project intermediate result sets as greedily as possible.
4. Collect selections and projections such that between other operators there is only a single block of selections followed by a single block of projections (and no projections followed by selections).
5. Combine selections and Cartesian products to joins.
6. Prepare pipelining for blocks of unary operators.

Task 3: Cost-Based Access Path

Suppose you are being given the same university schema as in the previous task, only that now you also have some information about the attributes, as shown below:



Each *int* field uses 4 Bytes, while a *char* 2 bytes per length. Records are stored row-wise using fixed length without spanning between blocks, i.e., for the most block there is free space left. Records are sorted on disk ascending with respect to the primary key. A *block* stores 496 Bytes. In addition, the following table and column statistics are available:

Table Name	#records	#blocks	bytes/record	record/block
Teachers	1000	250	124	4
Courses	2000	400	84	5
Students	5000	1250	106	4
CourseTaughtBy	5000	81	8	62
Grades	10000	205	10	49

Table Name	Column Name	#unique values
Teachers	e-mail	1000
Courses	ects	5
Courses	quarter	4
Students	sex	2
CourseTaughtBy	teacherID	1000
CourseTaughtBy	courseID	2000
Grades	studentID	5000
Grades	courseID	2000

Based on the above schema, perform the following tasks for the two operator trees shown below:

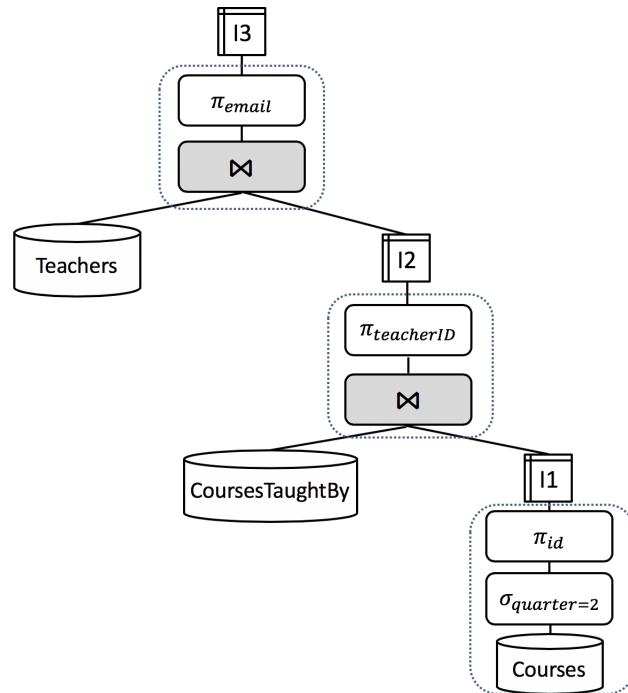


Figure 1: Operator Tree 1

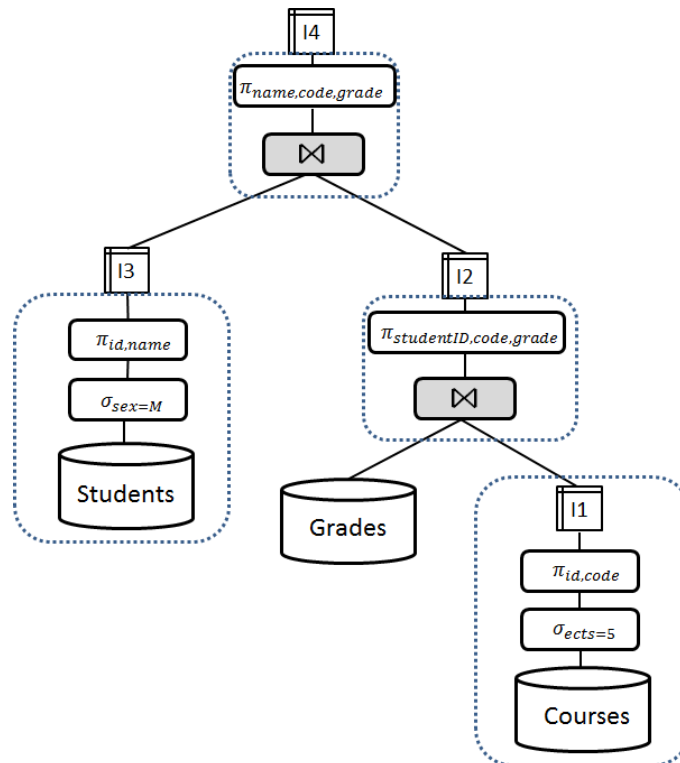


Figure 2: Operator Tree 2

1. Estimate the **size of the intermediate result** sets (labeled l1,l2, ...) in both **number of records** (rounded to integer values), **number of bytes** and **number of blocks** by using suitable heuristics. Show clearly how you obtained the result!
2. Decide which **algorithm** implementing the highlighted operations (using dashed boxes) is optimal by **computing the costs** (in combined block accesses for both reading and writing blocks) of all applicable **alternatives**, and select the cheapest one. You may skip any alternative if you can convincingly argue that it is guaranteed to be bad. However, if you decided in the previous question that an attribute should be indexed, show that using the index is indeed cheaper with respect to block accesses than not using it. It must be clear how you obtained the result! The available algorithms that you can use are:

- *Scans*: Relation Scan (RS), Index Scan (IS)
- *Joins*: Block-Nested-Loop Join (BNL), Index-Nested-Loop Join (INL), Merge Join (MJ)

You can assume that all attributes are indexed with a B+ tree of height 2, and (only) the root node is always in main memory.

3. What is the **final cost in combined block access** for executing the whole tree? Show clearly how you obtain the result!

Task 4: Transaction Management

Consider the following schedules of three and four transactions, respectively:

T1	R(X)		R(Z)			W(X)					R(X)
T2		R(Z)						R(Y)	W(Y)	W(Z)	
T3				R(X)	R(Y)		W(Y)				

T1			W(Y)	R(Y)							
T2					R(Y)			R(X)			
T3	R(X)					W(Z)	W(X)		R(Z)		
T4		W(X)								W(Z)	R(Z)

For each of the above schedules, draw the respective conflict graph and use it to determine (explain very briefly) whether the schedule is conflict-serializable or not.

Task 5: Recovery

Task 5.1: UNDO Logging and Checkpointing

After a system crash, the **undo log** using non-blocking checkpointing contains the following data:

1. <START T1>
2. <T1, X, 5>
3. <START T2>
4. <T1, Y, 7>
5. <T2, X, 9>
6. <START T3>
7. <T3, Z, 11>
8. <COMMIT T1>
9. <START CKPT(T2,T3)>
10. <T2, X, 13>
11. <T3, Y, 15>

Show the actions of the recovery manager during recovery and briefly explain why.

Task 5.2: UNDO/REDO Logging

A DBMS runs the following transaction T whose actions are listed on the first column. The column marked as ‘t’ contains the current value of the local variable ‘t’ throughout the execution of transaction T, the ‘**Mem A**’ column contains the value of the memory for resource A, while the ‘**Disk A**’ column contains the value of resource A on disk (the same for ‘**Mem B**’ and ‘**Disk B**’).

Task 5.2.1: Add the log messages (<COMMIT T>, <START T>, <T, VARIABLE, OLD, NEW>) in the last column (‘**Log**’), assuming that we are using **UNDO/REDO** Logging. Also, fill in the missing values whenever you see a ‘?’.

Action	t	Mem A	Mem B	Disk A	Disk B	Log
READ(A,t)	5	5		5	5	
$t := t \cdot 3$	15	5		5	5	
WRITE(B,t)	15	5	15	5	5	
WRITE(A,t)	15	15	15	5	5	
READ(B,t)	?	15	15	5	5	
$t := t + 5$?	15	15	5	5	
WRITE(B,t)	?	15	?	5	5	
FLUSH LOG						
OUTPUT(A)	?	15	?	15	5	
OUTPUT(B)	?	15	?	15	?	
FLUSH LOG						

Task 5.2.2: In short, what will the recovery procedure do (i.e., restore old values, rewrite new values, which ones?), if we pull the plug right after the first ‘FLUSH LOG’?

Task 6: Join Order Optimization

Based on the schema of Task 3, consider the join:

$CourseTaughtBy \bowtie (\sigma_{quarter=2} Courses) \bowtie Grades \bowtie Students$.

Use **Dynamic Programming** to obtain your result, considering only Deep-Left-Join Trees and using the expected size (in number of rows) of intermediate results as costs.