TE 31/01/2012

```
Student(<Matricola>, FirstName, LastName, BirthDate)
Exam(<Matricola>, <CourseId>, Date, Grade)
Course(<CourseId>, Title)
```

Table	Tuples	Blocks	Organization	Indeces
Student	150K	10K	primary hash (Matricola)	
Exam	2M	8K	entry sequenced	B+ tree on Matricola, height = 3

Query:

```
SELECT *
FROM Student S
  JOIN Exam E ON S.Matricola = E.Matricola
```

No collisions and no caching.

1. **Hash join** in which exam is re-structured in a hash storage

   - Read Exam and rewrite it
   - Do the hash join

   $$\underbrace{8K}_{\text{Read Exam}} + 2 \cdot \underbrace{2M}_{\text{tuples of exam}} + \underbrace{10K}_{\text{Read Student}} + \underbrace{10K}_{\text{Read restructured Exam}} \approx 4M$$

   Restructured Exam must have the same size as Student to do the Hash-join.

   We need to build a hash structure (in order to ba able to use it!). This requires scanning the whole table, extracting each tuple and inserting it into the right «bucket». The number of buckets is of course the same, as the hash funcion is the same (e.g., Matricola mod 10.000): these buckets however will be rather «empty» w.r.t. the blocks in the previous representation (the storage grows overall from 8K to 10K blocks). We therefore need to perform 8K i/o ops in order to read and as much as 4M i/o ops to update, as the right bucket must be found for each exam, and the corresponding block needs to be read (moved to main mamory), updated with the new exam, and then re-written to disk.

2. **Nested-loop** (`Student` as external)

   $$\underbrace{10K}_{\text{Read Student}} + \underbrace{150K}_{\text{Tuples of Student}} \left(\underbrace{3}_{\text{Tree access}} + \underbrace{13}_{\text{Exams per student}} \right) = 2.41M$$

1. **Nested-loop** (Exam as external)

   $$\underbrace{8K}_{\text{Read Exam}} + \underbrace{2M}_{\text{Tuples of Exam}} \cdot \underbrace{1}_{\text{hash}} \approx 2M$$

TE 21/02/2012

`Student(<Matricola>, FirstName, LastName)`

Table	Tuples	Blocks	Organization	Indeces
Student	200K	10K	B+(Matricola), height=3	Hash(LastName), 2K blocks

Note that in this case the *blocks* are actually the *leaves* of the tree. We also know that there are about 50K unique last names.

Ignore collisions and caching.

```sql
-- Query A
SELECT * FROM Student WHERE Matricola = '623883'
```

- Load all students, i.e., read all blocks: $10K$
- Use the tree, i.e., 3 (no disk access here)
- Use the hash? Ofc not.

```sql
-- Query B
SELECT *
FROM Student
WHERE LastName IN ('Braga', 'Campi', 'Comai', 'Paraboschi')
  AND Matricola > '575478'
```

Use the hash

$$\underbrace{4}_{\text{Surnames}} \cdot \left(\underbrace{1}_{\text{Hash access}} + \underbrace{\frac{200K}{50K}}_{\text{Number of students per surname}} \right) = 20$$

This is clearly the best.

```sql
-- Query C
SELECT *
FROM Student
WHERE LastName < 'B'
```

Here the hash is not very useful, neither is the tree. It's just better to read every block, so $10K$.