10a. Exercises with useful tips&patterns

Exercise

```
SELECT E.StudentCode, Name, AVG(Score)
FROM EXAM E, STUDENT S
WHERE E.StudentCode = S.StudentCode
and Score ≥ 18
GROUP BY E.StudentCode, Name
HAVING AVG(Score) ≥ 26
ORDER BY E.StudentCode;
```

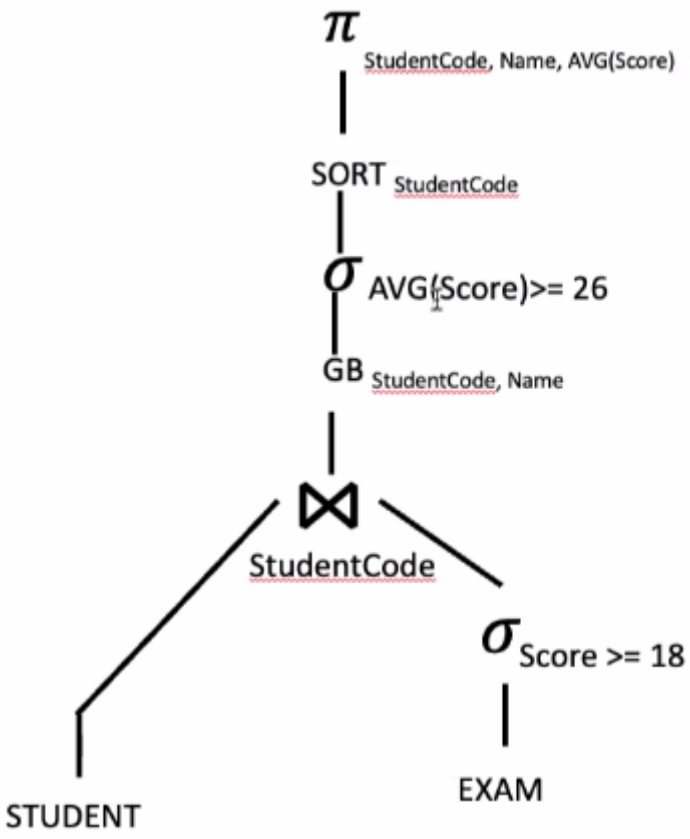
The following relations are given (primary keys are underlined):

- STUDENT(<u>StudentCode</u>, Name, Surname, Birthdate)
- EXAM(<u>StudentCode</u>, <u>CourseExam</u>, <u>Date</u>, Score)

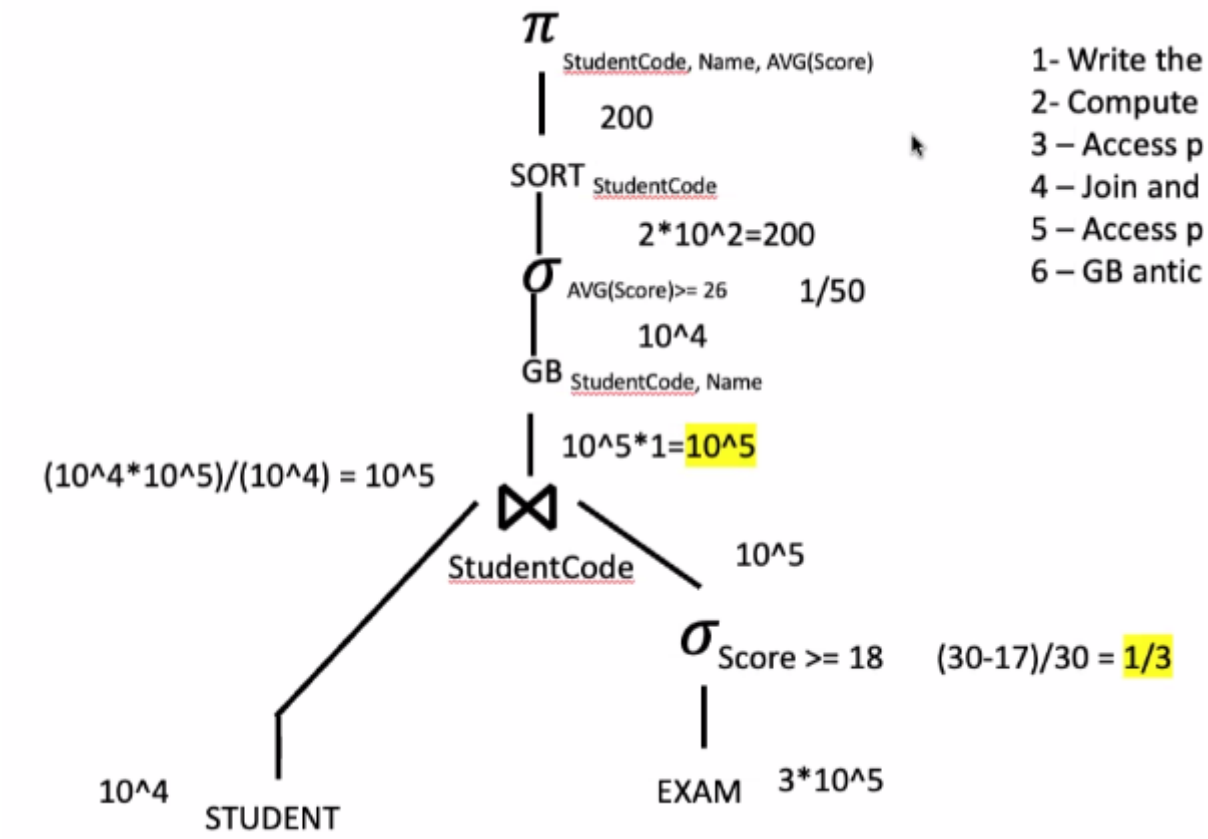
Assume the following cardinalities:

- card(STUDENT) ≈ 10^4 tuple
- card(EXAM) ≈ 3 * 10^5 tuple
  - MIN(EXAM.Score) = 1
  - MAX(EXAM.Score) = 30
- Selectivity HAVING AVG(Score) ≥ 26 equal to $\frac{1}{50}$

1. SQL To Algebra
    1. First the table access
    2. Then the filtering(WHERE clauses)
    3. Then the Join
    4. Then the Group By
    5. Then the Having condition
    6. Then the Order By
    7. Finally the Projection(SELECT)



2. Compute Cardinalities



Group By NEVER increases cardinality. It has high boundary of the previous cardinality.

3. Access path without indexes
   - Table access full on student and table access full + filter on exam
4. Join and Group By types
   1. **Join:**
      - If a table has $< 10^3$ it's considered small, otherwise it's considered as big
      - If one of the table is **small**, we go for a **Nested Loop**. The inner table in **NL join** is always the small one.
      - **Merge join** is almost never used. Only in case one of the table is already sorted(due to using certain **indexes**).
      - In this case, the result table is big and we go for a **Hash Join**
   2. **Group By**:
      - If a table has $< 10^3$,we go for a **GB Sort**, otherwise we go for a **GB HASH**
      - **GB NO HASH**: if we use **HASH JOIN** before, and both HASH and GB share same attribute, we go for **GB NO HASH**.
      - **GB NO SORT:** if we use **MERGE JOIN** before, we go for **GB NO SORT**.
      - No sort and no hash? We apply the rule based on the number of rows before **the group by**.
      - In this case, we go for **GB NO HASH**(since we used **hash join**).
5. Access path with indexes

- We decide whether to create an index on the table or not.
- **Primary** indexes are used on attributes that represents natural order of the data. E.g. StudentCode on table Student(but not on Exam!). Not necessarily needed on **small** tables.
- If we dont have any filtering conditions on the table, we dont use any **index**.
  - If we have a filtering condition, we have to consider if we have a good reduction factor.
  - If reduction factor is $\leq \frac{1}{10}$, we have a good reduction factor.
  - On index with multiple attributes, the first one must be always the one with the highest reduction factor.
    - If $condition_1$ has a red. factor of $\frac{1}{100}$ and $condition_2$ has a red. factor of $\frac{1}{2}$, the resulting composite index will have a reduction factor of $\frac{1}{200}$
- **B+Tree** is always used for attributes that can be ordered(date,numbers)
- **Hash** is **always** used for text attributes
- In this case we create a **primary b+tree index** on **StudentCode**(it's used for join)
- We dont create an index on **exam** since the reduction factor is $\frac{1}{3} > \frac{1}{10}$

6. GB Anticipation
   - Whether to push down GB or not
   - In this case, we can push down **GB+Having** down to the Exam table(after the filtering). Also, now the **GB** becomes a **GB HASH**.
   - Cardinalities should be recomputed in theory, but **not in the exam**

Nota IN/NOTIN EXIST/NOT EXIST: IN/EXIST is converted into a **semi-join** where the inner table corresponds to the **subquery**
Cardinality is computed as follows: $\#TuplesOuter \times \frac{\#TuplesInner_{\text{after query execution}}}{\#TuplesInner_{\text{before query execution}}}$
NOT IN/NOT EXIST, is converted into a **semijoin**.
Cardinality is computed as follows:
$\#TuplesOuter \times (1 - \frac{\#TuplesInner_{\text{after query execution}}}{\#TuplesInner_{\text{before query execution}}})$