1. Splitting tables

We split the *courseregistrations* table into four separate tables:

- *courseregistrations_4*, including only course registrations with grade = 4
- *courseregistrations_failed*, including only course registrations with grade < 5
- *courseregistrations_null*, including only course registrations with grade null
- *courseregistrations_passed*, including only course registrations with grade >= 5

2. Composite primary key

We created a composite primary key in the different *courseregistrations* tables on attributes *studentregistrationid* and *courseofferid*.

3. Indices

- *student_degree*, on the *studentid* from *studentregistrationstodegrees*
- *course_id*, on the *courseid* from *courseoffers*

4. Normal and materialized views

Note that we used materialized views for computing GPA/completed degrees/number of students a course had throughout its lifetime. However, we could not manage to have correct queries so those were not included in our optimization, despite them being quite effective in optimizing our (wrong) queries.

Concise descriptions of the normal and materialized views can be found in the overview tables below.

Optimizations

type	name	remarks	space	create
index	student_degree	on sr_to_deg(studentid)	0.2 GB	8 s
index	course_id	on courseoffers(courseid)	~0.0 GB	0 s
Mview	inactive	srid's for students who completed degree	~0.0 GB	80 s
Mview	GPA	weighted average of passing course offers	0.2 GB	100 s
Mview	non_fail_stu	students who haven't failed a course	~0.0 GB	46 s
Mview	good_student	excellent students (as per Q6)	~0.0 GB	26 s

Query runtimes

	optimizations	before	after
Q1	index on sr_to_deg	157 s	2 s
Q2	non_fail_stu, GPA, inactive	900 s	190 s
Q3	inactive	191 s	20 s
Q4		11 s	11 s
Q5		150 s	150 s
Q6	good_student	111 s	5 s
Q7	GPA	353 s	231 s
Q8		23 s	23 s