2ID70 Group 5, Richard Sinx, Tommie Kerssies, Thorn Jansen

Optimization process

The database structure we set up using the standard predefined way, with each table being created as specified in the assignment. First, primary and foreign keys were created during initialization. This would take hours to create, so foreign keys were deleted and the primary keys only added after all the data had been loaded. The queries were designed to work, disregarding runtime. After creating some queries we decided to check what impact indices would have on the runtime. As it turned out, primary keys and their automatically created indices were enough. Adding more indices to the existing tables made no difference on the runtime of the queries.
The step that took us most of the time and gave us substantial results was rewriting the queries to optimize their runtime. The biggest improvements were gained by limiting the amount of joins, and major overhauls of particularly inefficient queries.
The last step we took was creating materialized views. As we only have a limited amount of time to create materialized views and indexes we had to carefully evaluate which ones we wanted to create. We created multiple materialized views for query 1, as this query is executed 100 times. The first one was specifically for query 1 and could not be reused for any other query. It made query 1 run in 3ms, but took a lot of time to create. The second version only filters the CourseRegistrations table on the grades >= 5, plus an index on student registration IDs on this view. This view is reusable as it can speed up the creation of other views. As query 2, 3, and 7 all use (non)active students we decided to make a view with the active students, and if possible their GPAs added. Multiple versions were created, with the first ones containing bugs which were only found days after the creation of the view. The challenge with this view was getting it created within the 5 minutes given, as it is a relatively complex view made with several joins.

Chosen optimizations

The first optimization used a primary key on every table. This speeds up the joins significantly. The space taken by the primary keys is 1.9GB. The second optimization chosen is the optimization of the queries itself. We chose to spend time on rewriting the queries such that they would run more efficiently. This take no additional creation time or storage space. Then the last optimizations we did are materialized views and indexes on them. The first materialized view we have is a view on CourseRegistrations which only takes the rows with the grade value being >= 5, such that we only have all the CourseRegistrations with passing grades. There is an index on StudentRegistrationId created on this materialized view. The materialized view takes 1.6GB and the index takes 0.8GB. This view and index speed up one run of query 1 from 37.4 seconds without the view to 3.4 seconds with the view and index.
As we are not short on memory we have decided to split it up in multiple views such that we can make the first ones materialized and the later ones normal views. This means it at least partially speeds up the query as part of the view it needs is materialized.
The next materialized view created is AllStudentsDegrees which is a view containing the StudentId and DegreeId with how many ECTS there are required for that specific degree. This view is not directly used in queries, but it speeds up the creation of other views. It could be used for query 4, but this is not needed, as this query is already very quick.
The next two views we wanted to create materialized, but due to a shortage of time in creating materialized views but not in the query running time we have made them normal views. The first view is CompletedDegrees, which keeps a list of all students who have finished a degree with the respective degree. This view takes 0GB as we are unable to make it materialized. The second view is ActiveDegrees which keeps a list of students who are still active (within their degree) and their respective degree. CompletedDegrees is used to compute this. ActiveDegrees takes 0GB as we are unable to make it materialized. CompletedDegrees can be used for query 2 and ActiveDegrees for query 3 and 7. However as query 2 is not currently running on the server we are unable to determine the time difference the view provides. The difference in runtime on query 3 and 7 is minimal as it has to compute the views which could not be materialized in the query itself. If we would create the views materialized query 3 would run in 5:37 without the materialization of the views and in 0:25 with the materialization of the views. Query 7 has a difference we can not properly quantify as the query is impossible to run on the vm's due to a lack of RAM.

EXTRA: Queries created but not in Queries.sql

We have created the following query it should be very close to correct but we were unable to fix the last small issues, it is here as we did not want to add them to the Queries.sql file as they will not provide us with additional correct queries answers.

2ID70 Group 5, Richard Sinx, Tommie Kerssies, Thorn Jansen

Query 7:

```
SELECT adg.DegreeId AS degreeid, Students.BirthyearStudent AS birthyear, Students.Gender AS gender,
AVG(adg.gpa) AS avgGrade
FROM (
   SELECT ad.StudentId, ad.DegreeId, SUM(cr.Grade*Courses.ECTS)/SUM(Courses.ECTS) AS gpa
   FROM ActiveDegrees AS ad
   INNER JOIN grade5 AS cr ON ad.StudentRegistrationId = cr.StudentRegistrationId
   INNER JOIN CourseOffers as co ON cr.CourseOfferId = co.CourseOfferId
   INNER JOIN Courses ON co.CourseId = Courses.CourseId
   GROUP BY ad.StudentId, ad.DegreeId
) AS adg
INNER JOIN Students ON adg.StudentId = Students.StudentId
GROUP BY CUBE(adg.DegreeId, Students.BirthyearStudent, Students.Gender)
ORDER BY adg.DegreeId, Students.BirthyearStudent, Students.Gender ASC;
```