Slide - 1
Hey and welcome everybody to our project pitch for our project "SQL test case generation using Multi-objective optimization".

Slide - 2
Before we start let's lookt at our table of contents. First we will introduce the paper we use as a base for our research. Followed by that we explain to you in detail what challenge we derive from the paper. Finally, we formulate our goal, how we are going to reach it and our expected outcome of our experiment.

Slide - 3
In data-centric programs SQL queries play an important role and therefore need to be tested thoroughly. For instance such programs include applications in accounting, customer management, medical information storage/processing, etc.
As the underlying application heavily use SQL queries, it is tedious to generate test cases for each small query by hand. On the other side is time-consuming to write a test case by hand for a complex query. Unfortunately this is still considered "state-of-the-art" for most cases.

Slide - 4
For example look at this query. I hope you are all somewhat familiar with SQL if not, you can easily guess what is happening here. But how should one write a test table that covers all possible branches of this query.

Slide - 5
Castelein explores in the paper from the reading list "Search-based Test Data generation for SQL queries" how to overcome this problem. As this not a paper presentation we will go only briefly through the important parts of the paper to understand our problem.

Slide - 6
As the title suggest the authors model the test data generation for SQL-queries as a search-based problem. For this problem they apply three differen approaches, random search, biased random search and genetic algorithms. We will focus on the genetic algorithms in the following discussion, as the authors showed that it outperforms the other approaches.
To test their algorithms the authors apply their framework on 2.135 queries taken from 4 different applications.

Slide - 7
We will now dive deep into how the authors modeled their problem. In order to understand the fitness function we briefly need to look at a SQL query and it's query plan.
For each query executed in a SQL databases such a query plan is created. For example the plan for the shown query consits of two steps.
If the first step is not executed succesfully the plan will stop pre-maturely and will not continue to be executed.
If it is successful we will therefore continue with the second step.

Slide - 8
So on this slide you can see such a query plan. This plan gets executed from left to right. Already covered steps are highlited in green. While the remaining steps are shown in red.
On the bottom you can see how the fitness function is constructed, it consists of two parts. The step level, which is the amount of uncovered steps in the query plan. The second term is the current step distance, basically how far away we are from covering the current step. The step distance is calculated by summing up all operation distances in one step. Therefore the fitness function gives us a value when a query (plan) is executed on a database filled with values. If the query plan gets executed until the end, the fitness is obviously zero and the query can be seen as solved.

Slide - 9
One problem that is mentioned by the authors in the paper is the inefficieny of a single-target strategy. A target or a coverage target, is a single branch of one query. So a single query generates multiple coverage targes. In the next slides we explain this concept more.
Regarding our problems: In the paper the authors view each query as a single-target and therefore process the GA as one single target. From this we derive two possible problems. First, the order in which the coverage targes are optimized is not fixed. Therefore, we might redo unnessary steps in each single-target optimization. Second, the authors mention that they had revisited all queries manually to remove infeasible coverage targes.

Slide - 10
As a solution to this we propose to rephrase the problem as a Multi-Objective Optimization.

Slide - 11
To understand it further let me quickly explain the coverage targets for you. Assume we have this simple toy query which selects everything from a product table which has a certain category. For this query there are two possible branches aka coverage targets.

Slide - 12
First, we need to test the normal query as one possible coverage target. For instance let's assume we have this table on the right side. If we run the first coverage target on it you can see that the query will

First, we need to test the normal query as one possible coverage target. For instance let's assume we have this table on the right side. If we run the first coverage target on it you can see that the query will return the first row and therefore the fitness value for this coverage target is 0.

Slide - 13
Second, another branch is the negation of each operator. In this it is just one. If we assume the same table and execute this coverage target on it we will return the second and third row. Again the whole plan gets executed fully and therefore the fitness is zero.

Slide - 14
AS previously hinted the current implementation of the authors view each coverage target as a seperate problem to solve with a GA. As you can guess, it might not be the most efficient one. While the authors reuse successful coverage targets of one query for seeding there is no explicit connection for solving each coverage target.

Slide - 15
We therefore want to search for a solution that optimizes all targets of a query simulatensiously. One simple, possible solution to sum up the fitness of all targets and provide this as the overall fitness for the GA. A second possible solution would be to phrase it as a multi-objective optimization problem.

Slide - 16
Let me explain to you in detail how could phrase this as a multi-objective optimization problem. So we still have the same query as before. Our two coverage targets are indicated on the right side of this slide with c_1 and c_2. We shorten the queries to only capture the import parts. Our fitness function is the same as before, we insert coverage target, generate the query plan and evaluate it on the given table it.
From the given query we can also derive the schema of our database, so we know there is one only a single table, with one column named category and the obligatory id entry.
Each coverage target is one objective of our multi-objective optimization problem. As we have two coverage targets our optimization space is two-dimensional. The first dimension is the fitness value for coverage target one, the second for target two. Now one individual can be described as a set of rows in our database tables. The amount of rows is not fixed and is something the GA will mutate. In our case we have three individuals in our population the first individual in the top left, covers the second target as car is not toy. But not the first one therefore the fitness value for the first target is still high. Vice versa for the individual in the bottom right. Only a combination of those two individuals cover both targets perfectly. In this trivial case this might seem obvious to you but in the paper there are queries with more than 20 coverage targets.

Slide - 17
A second challenge is the inefficient allocation of the search budget. So assume you have something like this A > bigger than 10 and A smaller than 10. This is impossible to solve. Therefore the authors needed to go through each coverage target manually and remove these from their set of all queries, as the GA couldn't find a solution. With the multi-objective approach we expect the GA will not waste as much as ressources as previously on these targets but rather focus on the other targers which are feasible.

Slide - 18
Finally this what we expect as an outcome of our results.
First, our multi-objective optimization implementation can solve more complex queries aka more coverage targets simulatensiously compared to a single-objective coverage target.
Second, each solution is guided towards a faster convergence by similar solutions from other coverage targets.
Third, the multi-objective optimization doesn't waste as much as computionally power on infeasible targets as the single-target optimization. In the single-target optimization the budget is split equally across all targets, therefore the infeasible ones are getting optimizid for an unnessary amount of time. The provided solution of the authors by manually removing them does not scale well to very complex and a lot of targets.

Slide - 19
Thank you for your attetention, do you have any questions?