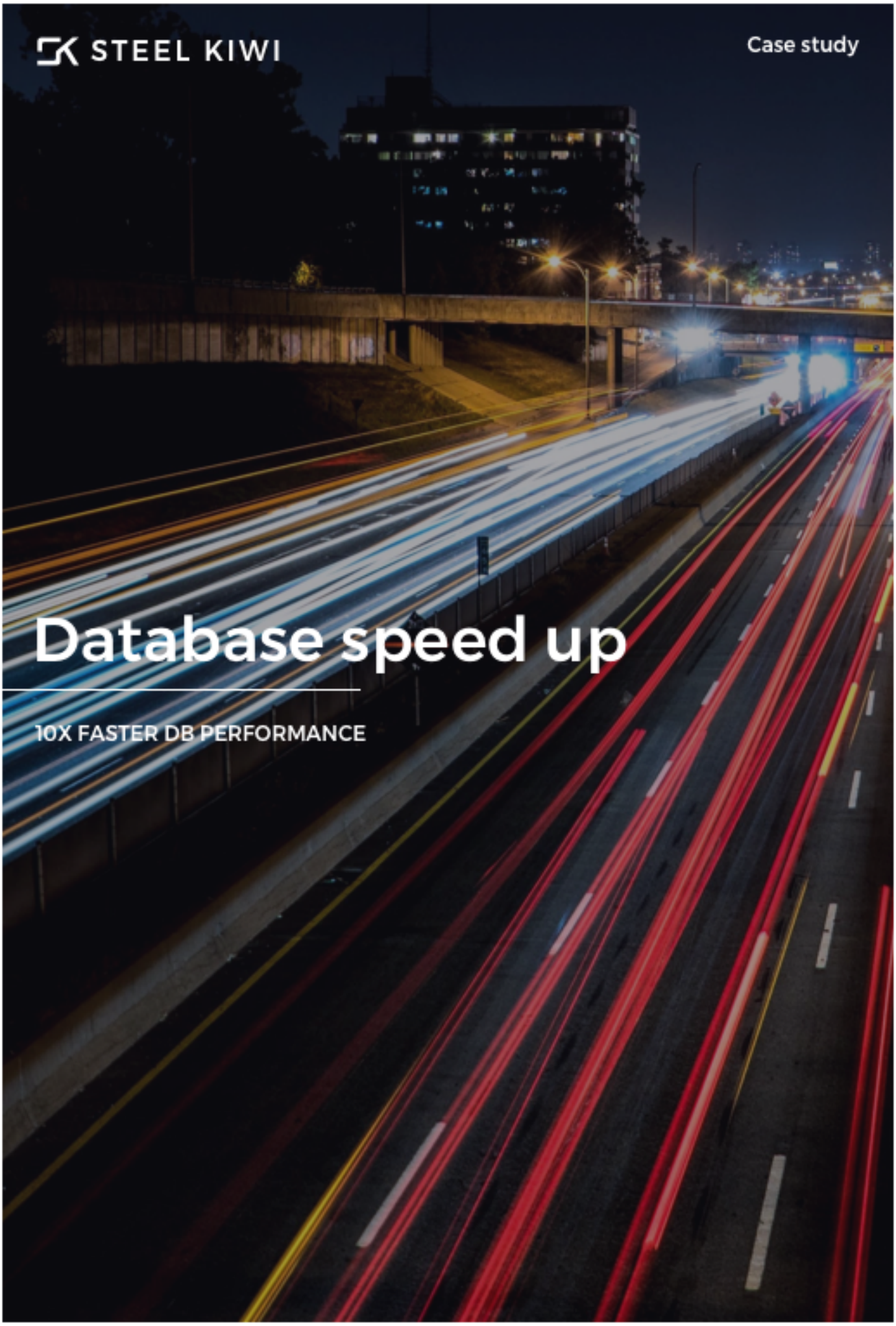
STEEL KIWI

Case study



Database speed up

10X FASTER DB PERFORMANCE

Customer location	Industry	Expertise
UK	**Civil services**	**Business process automation**

Summary

We developed a web application to manage information about people's certifications, allow them to upload certification details, and generate custom PDF reports showing tests, results, and certifications earned. This new web application replaces an existing system that suffered from performance problems. The main bottleneck we faced in our new system was uploading batches of records with up to 20,000 entries.

SteelKiwi engineers applied their expertise working with complex databases. As this new product is supposed to interact with the existing reporting system and is embedded in business processes, we added strict requirements for the database structure and development tools.

Technical challenges

1. While uploading batches with up to 20,000 records, we needed to validate data from newly uploaded records with other records in the same file as well as with those already in the database. We also needed to generate a file with any errors.

2. Due to requirements, we couldn't use any asynchronous tools like Celery for file processing. At the same time, we had strict rules on the database structure, and even custom indexes were a problem.

3. We had definite requirements for how much time it could take to process a file of a particular length in a database of a particular size:

   - Small file uploads of fewer than 500 records containing 80% duplicates had to be processed within 5 seconds and support five concurrent uploads.
   - File uploads of up to 20,000 records with 95% duplicates had to be processed within 60 seconds with one upload at a time.
   - Searches of up to 100 records had to be processed within 2 seconds with 10 concurrent searches.

We measured these targets against a database populated with two years' worth of data (approximately 300,000 training records). All measures in the table below show the system response time and exclude data transfer time and response time on the user's end.

APPROXIMATE FIGURES FOR DATABASE SIZES AFTER SEVERAL YEARS OF USE ARE AS FOLLOWS:

Training Providers ~ 250	Delegates ~ 100,000
Training Records ~ 1,000,000	Searches ~ 750,000
Courses ~ 20	Countries ~ 300

Technical solutions

At the beginning, we implemented functional requirements without any special optimization. We populated the database and checked how much time it took to process files. We had a situation like this:

FILE LENGTH, RECORDS	ACTUAL TIME, SECONDS	TIME LIMIT, SECONDS
300	3.47	5
500	3.36	5
5,000	32.82	60
20,000	120.00	60

1. To speed things up, we first got rid of all unnecessary requests that could easily be avoided using techniques provided by Django ORM. This wasn't enough, however, as uploads were still taking too long.

2. We added Django's select_related and prefetch_related methods, which are in fact SQL joins, as well as a utility that fetches related data with one request from the database to ORM calls. Still, this was not enough.

3. We thought about using raw SQL queries, but since Django treats this like it's the last thing to do when nothing else helps (since it's harder to maintain and may cause potential security issues), we continued to optimize with more ORM features. With Django, objects are generally saved to the database one by one, so it takes a lot of time to perform each request. Django does offer the option for bulk creation of objects, though, and we took it. We used batches of about 200 records. This helped a lot, but we still were clocking in at over 60 seconds per upload.

4. We needed to check items in the database with a unique key from the file, since some of them might be duplicated. We implemented a local in-memory cache as a Python dictionary object so we didn't fetch an object twice. This in-memory cache helped when there were a lot of duplicates, but didn't help for files that contained a lot of unique IDs as it was taking too much time to populate the cache. In short, this helped a bit with our main task, but not as much as we needed.

5. We decided to implement one more idea: reading the whole file at once and loading all records to a python object. But as each file can have as many as 20,000 to 30,000 records, the data was too long to be stored in an object. As result, we decided to read files by chunks, process possible validations, and then get data from the database in bulk. After that, we go through a chunk once again, do final validations, and update the database.

Results

Our solution doesn't look optimal at first glance: read files by chunks and validate twice. Even more, to perform this for a whole file at the same time, we needed additional calculations and duplicated cycles in python. Still, we managed to make everything much faster.

FILE LENGTH, RECORDS	ACTUAL TIME, SECONDS	TIME LIMIT, SECONDS
300	0.297	5
500	0.523	5
5,000	3.74	60
20,000	14.61	60

Contact us

SKYPE: STEELKIWISALES

EMAIL: HELLO@STEELKIWI.COM

PHONE: +1 415-449-8696