Benchmarking SQL

In this article, we're going to look into our recommended technique to benchmark SQL queries to find the fastest alternative. We also recommend this technique in our SQL Masterclass training.

Let's assume we're using the Sakila database. Which of the following logically equivalent queries do you think will be the fastest?

```
-- 1: "JOIN before WHERE"
SELECT
  first_name, last_name, count(*)
FROM actor a
JOIN film_actor fa
USING (actor_id)
WHERE last_name LIKE 'A%'
GROUP BY
  actor_id, a.first_name, a.last_name
ORDER BY count(*) DESC
```

```
-- 2: "WHERE before JOIN"
SELECT first_name, last_name, count(*)
FROM (
  SELECT *
  FROM actor
  WHERE last_name LIKE 'A%'
) a
JOIN film_actor USING (actor_id)
GROUP BY a.actor_id, a.first_name, a.last_name
ORDER BY count(*) DESC
```

```
-- 3: "GROUP BY in correlated subquery"
SELECT * FROM (
  SELECT first_name, last_name, (
    SELECT count(*) FROM film_actor fa
    WHERE a.actor_id = fa.actor_id
  ) AS c
  FROM actor a WHERE last_name LIKE 'A%'
) a
WHERE c > 0
ORDER BY c DESC
```

```
-- 4: "GROUP BY before JOIN"
SELECT first_name, last_name, c
FROM actor
JOIN (
  SELECT actor_id, count(*) c
  FROM film_actor
  GROUP BY actor_id
) fa USING (actor_id)
WHERE last_name LIKE 'A%'
ORDER BY c DESC
```

All of these queries count the number of films per actor for actors whose last name starts with the letter A. All of these queries produce this result:

```
+------------+-----------+-------+
| first_name | last_name | count |
+------------+-----------+-------+
| KIRSTEN    | AKROYD    | 34    |
| CHRISTIAN  | AKROYD    | 32    |
| ANGELINA   | ASTAIRE   | 31    |
| KIM        | ALLEN     | 28    |
| CUBA       | ALLEN     | 25    |
| DEBBIE     | AKROYD    | 24    |
| MERYL      | ALLEN     | 22    |
+------------+-----------+-------+
```

But which query is the fastest?

Database	1: JOIN before WHERE	2: WHERE before JOIN	3: Correl. subq.	4: GROUP BY before JOIN
Oracle 12c	Quite slow	Quite slow	Fast	Quite slow
PostgreSQL 9.5	Fast	Fast	Quite fast	Rather slow
SQL Server 2014	Fast	Fast	Rather slow	Rather slow
DB2 LUW 10.5	OK-ish	OK-ish	Fast	Rather slow
MySQL 8.0.2	OK-ish	OK-ish	Fast	Rather slow

We didn't publish actual numbers because we don't want you to draw any conclusions from these results (and because commercial RDBMS don't allow such publication).

How to interpret these results?

1. **The first thing** that can be seen is that all databases differ. There is no single fastest solution that works fast on all databases.
2. **The second thing** you might notice is that we didn't publish any execution times. It is really hard to compare executions between queries, let alone vendors. Benchmarks depend on many many things, including:
   - Database vendors
   - Database versions
   - Setups and configurations (some of these were run in docker)
   - Hardware (e.g. disks, cores, memory, etc.)
   - Data sets and cardinalities (the Sakila database is rather small. The correlated subquery was fast *in this case* but might be slower than the join solutions with larger data sets!)
   - Other activity on your operating system
   - ... and much more
3. **The third thing** is not to trust benchmarks. This cannot be said enough. In the above example, Oracle seemed to have quite slow execution speeds for statements #1, #2, and #4. But another way to interpret this is that statement #3 profited from a really cool optimisation "accidentally", and all the other versions are simply equally good.

How to get these results?

Nevertheless, the benchmark technique exposed here does help finding a *probably* faster query among several alternatives. Here's a piece of code that shows how we can find a good query in Oracle:

```
  v_ts TIMESTAMP WITH TIME ZONE;
  v_repeat CONSTANT NUMBER := 10000;
BEGIN

  -- Repeat the whole benchmark several times to avoid warmup penalty
  FOR r IN 1..5 LOOP
    v_ts := SYSTIMESTAMP;

    FOR i IN 1..v_repeat LOOP
      FOR rec IN (
        -- Paste statement 1 here
        SELECT 1 FROM dual
      ) LOOP
        NULL;
      END LOOP;
    END LOOP;

    dbms_output.put_line('Run ' || r ||', Statement 1 : ' || (SYSTIMESTAMP - v_ts));
    v_ts := SYSTIMESTAMP;

    FOR i IN 1..v_repeat LOOP
      FOR rec IN (
        -- Paste statement 2 here
        SELECT 1
        FROM dual
        CONNECT BY level < 100
      ) LOOP
        NULL;
      END LOOP;
    END LOOP;

    dbms_output.put_line('Run ' || r ||', Statement 2 : ' || (SYSTIMESTAMP - v_ts));
    dbms_output.put_line('');
  END LOOP;
END;
/
```

In the middle of all of this, you can see two trivial queries being compared, where one should obviously outpeform the other.

Query 1 is run a number of times (`v_repeat = 10000`) in a row, then query 2 is run equally often, times are measured for both, and then the whole benchmark is repeated 5 times, so there's no warmup penalty for either query.

That's it. A simple and primitive benchmark that can help find a better query among several alternatives (just copy paste one of the loops to compare 3 or 4 or more queries).

What can this benchmark do and what can it not do?

Again, it is very important to understand that this benchmark cannot ...

- ... reliably compare execution speeds among database vendors.
- ... display the difference in performance in real world production use-cases. It runs a single query 10000x in a row, thus profiting from caching effects much more than a production environment.
- ... help improve performance by low margins, i.e. this is not a good technique for micro optimisations.

But it most certainly can ...

- ... help find drastic differences between alternative queries
- ... prevent any side effects from running the benchmark from a client. It uses each database's procedural language capabilities

Where can I get it?

We've open sourced these benchmarks for DB2 LUW, MySQL, Oracle, PostgreSQL, SQL Server here on GitHub. Each script comes in two flavours:

- Displaying absolute execution times (as the above PL/SQL script)
- Displaying relative execution times (relative to the fastest execution)

To learn more about SQL performance, we recommend you visit our SQL Masterclass Training.

Download the benchmark scripts from GitHub