

ora_bench - Benchmark Framework for Oracle Database Drivers.

build failing license not identifiable by github release no releases or repo not found
release date no releases or repo not found github repo or version not found

Table of Contents

- 1. Introduction
 - 2. Framework Tools
 - 3. Coding Pattern
 - 4. Driver Specific Features
 - 5. Execution Variations
 - 6. Reporting
-

1. Introduction

OraBench can be used to determine the performance of different Oracle database drivers under identical conditions. The framework parameters for a benchmark run are stored in a central configuration file.

The currently supported database drivers are:

Driver	Programming Language(s)
godror	Go
Oracle cx_Oracle	Python 3
Oracle JDBC	Java & Kotlin
Oracle ODPI-C	C++ (gcc)
oranif	Elixir & Erlang

The following Oracle database versions are provided in a benchmark run via Docker container:

Shortcut	Oracle Database Version
db_18_4_xe	Oracle Database 18c 18.4 (Express Edition) - Linux x86-64
db_19_3_ee	Oracle Database 19c 19.3 - Linux x86-64

The results of the benchmark runs are collected in either csv (comma-separated values) or tsv (tab-separated values) files.

2. Framework Tools

2.1 Benchmark Configuration

The benchmark configuration file controls the execution and output of a benchmark run. The default name for the configuration file is `priv/properties/ora_bench.properties`. A detailed description of the configuration options can be found [here](#). For reasons of convenience the following files are generated:

- the configuration file `priv/ora_bench_c.properties` for C++ (gcc),
- the configuration file `priv/ora_bench_erlang.properties` with a corresponding map for Erlang, and
- the configuration file `priv/ora_bench_python.properties` for Python 3.

All the file names specified here are also part of the configuration file and can be changed if necessary.

2.2 Installation

The easiest way is to download a current release of **OraBench** from the GitHub repository. You can find the necessary link [here](#).

To download the repository [Git](#) is needed and for compilation the following software components are needed:

- [Elixir](#)
- [Erlang](#)
- [Go](#)
- [Gradle Build Tool](#)
- Java, e.g.: the [open-source JDK](#)
- [Kotlin](#)
- [Make for Windows](#)
- [Oracle Instant Client](#)
- [Python 3](#)
- [rebar3](#)

For changes to the **DBSeeder** repository it is best to use an editor (e.g. [Vim](#)) or a swuitable IDE. For using the Docker Image based databases in operational mode, [Docker Desktop](#) must also be installed. For the respective software versions, please consult the document [release notes](#).

2.3 Benchmark Execution

2.3.1 `run_bench_all_dbs_props_std`

This script executes the `run_properties_standard` script for each of the databases listed in chapter Introduction with standard properties. At the beginning of the script it is possible to exclude individual databases or drivers from the current benchmark. The run log is stored in the `run_bench_all_dbs_props_std.log` file.

2.3.2 `run_bench_all_dbs_props_var`

This script executes the `run_properties_variations` script for each of the databases listed in chapter Introduction with variations of properties. At the beginning of the script it is possible to exclude individual databases or drivers from the current benchmark. The run log is stored in the `run_bench_all_dbs_props_var.log` file.

2.3.3 `run_bench_<driver>_<programming language>`

The driver and programming language related scripts, such as `run_bench_jdbc` in the `src_java` directory, first execute the insert statements and then the select statements in each trial with the data from the bulk file. The time consumed is captured and recorded in result files.

2.4 Benchmark Results

In a file defined by the configuration parameters `file.result.delimiter`, `file.result.header` and `file.result.name`, the results of the benchmark run with the actions `benchmark`, `trial` and `query` are stored. If the result file does not yet exist, a new result file is created. Otherwise, the new current results are appended to existing results.

Column	Format	Content
release	alphanumeric	config param <code>benchmark.release</code>
benchmark id	alphanumeric	config param <code>benchmark.id</code>
benchmark comment	alphanumeric	config param <code>benchmark.comment</code>
host name	alphanumeric	config param <code>benchmark.host.name</code>
no. cores	integer	config param <code>benchmark.number.cores</code>
os	alphanumeric	config param <code>benchmark.os</code>
user name	alphanumeric	config param <code>benchmark.user.name</code>
database	alphanumeric	config param <code>benchmark.database</code>
language	alphanumeric	config param <code>benchmark.language</code>
driver	alphanumeric	config param <code>benchmark.driver</code>
trial no.	integer	0 if action equals <code>benchmark</code> , trial no. otherwise
SQL statement	alphanumeric	SQL statement if action equals <code>query</code> , empty otherwise
core multiplier	integer	config param <code>benchmark.core.multiplier</code>
fetch size	integer	config param <code>connection.fetch.size</code>
transaction size	integer	config param <code>benchmark.transaction.size</code>
bulk length	integer	config param <code>file.bulk.length</code>
bulk size	integer	config param <code>file.bulk.size</code>
batch size	integer	config param <code>benchmark.batch.size</code>
action	alphanumeric	one of <code>benchmark</code> , <code>query</code> or <code>trial</code>
start day time	yyyy-mm-dd hh24:mi:ss.fffffffff	current date and time at the start of the action
end day time	yyyy-mm-dd hh24:mi:ss.fffffffff	current date and time at the end of the action

Column	Format	Content
duration (sec)	integer	time difference in seconds between start time and end time of the action
duration (ns)	integer	time difference in nanoseconds between start time and end time of the action

2.5 Bulk File

The bulk file in `csv` or `tsv` format is created in the `run_create_bulk_file` script if it does not already exist. The following configuration parameters are taken into account:

- `file.bulk.delimiter`
- `file.bulk.header`
- `file.bulk.length`
- `file.bulk.name`
- `file.bulk.size`

The data column in the bulk file is randomly generated with a unique key column (MD5 hash code).

3. Coding Patterns

3.1 Benchmark Function (main function)

```
run_benchmark()

    save the current time as the start of the 'benchmark' action

    READ the configuration parameters into the memory (config params
`file.configuration.name ...`)
    READ the bulk file data into the partitioned collection
bulk_data_partitions (config param 'file.bulk.name')
        partition key = modulo (ASCII value of 1st byte of key * 256 + ASCII
value of 2nd byte of key,
                                number partitions (config param
'benchmark.number.partitions'))
    Create a separate database connection (without auto commit behaviour) for
each partition

    trial_no = 0
    WHILE trial_no < config_param 'benchmark.trials'
        DO run_benchmark_trial(database connections, trial_no,
bulk_data_partitions)
    ENDWHILE

    partition_no = 0
    WHILE partition_no < config_param 'benchmark.number.partitions'
        close the database connection
    ENDWHILE
```

```
WRITE an entry for the action 'benchmark' in the result file (config param
'file.result.name')
```

3.2 Trial Function

```
run_trial(database connections, trial_no, bulk_data_partitions)
INPUT: the database connections
      the current trial number
      the partitioned bulk data

save the current time as the start of the 'trial' action

create the database table (config param 'sql.create')

IF error
    drop the database table (config param 'sql.drop')
    create the database table (config param 'sql.create')
ENDIF

DO run_benchmark_insert(database connections, trial_no,
bulk_data_partitions)
DO run_benchmark_select(database connections, trial_no,
bulk_data_partitions)

drop the database table (config param 'sql.drop')

WRITE an entry for the action 'trial' in the result file (config param
'file.result.name')
```

3.3 Insert Control Function

```
run_insert(database connections, trial_no, bulk_data_partitions)
INPUT: the database connections
      the current trial number
      the partitioned bulk data

save the current time as the start of the 'query' action

partition_no = 0
WHILE partition_no < config_param 'benchmark.number.partitions'
    IF config_param 'benchmark.core.multiplier' = 0
        DO Insert(database connections(partition_no),
bulk_data_partitions(partition_no))
    ELSE
        DO Insert(database connections(partition_no),
bulk_data_partitions(partition_no)) as a thread
    ENDWHILE
```

```
WRITE an entry for the action 'query' in the result file (config param
'file.result.name')
```

3.4 Insert Function

```
insert(database connection, bulk_data_partition)
INPUT: the database connection
       the bulk data partition

count = 0
collection batch_collection = empty

WHILE iterating through the collection bulk_data_partition
  count + 1

  add the SQL statement in config param 'sql.insert' with the current
bulk_data entry to the collection batch_collection
  IF config_param 'benchmark.batch.size' > 0
    IF count modulo config_param 'benchmark.batch.size' = 0
      execute the SQL statements in the collection batch_collection
      batch_collection = empty
    ENDIF
  END IF

  IF config_param 'benchmark.transaction.size' > 0 AND count modulo
config_param 'benchmark.transaction.size' = 0
    commit
  ENDIF
ENDWHILE

IF collection batch_collection is not empty
  execute the SQL statements in the collection batch_collection
ENDIF

commit
```

3.5 Select Control Function

```
run_select(database connections, trial_no, bulk_data_partitions)
INPUT: the database connections
       the current trial number
       the partitioned bulk data

save the current time as the start of the 'query' action

partition_no = 0
WHILE partition_no < config_param 'benchmark.number.partitions'
  IF config_param 'benchmark.core.multiplier' = 0
    DO Select(database connections(partition_no),
```

```

bulk_data_partitions(partition_no, partition_no)
    ELSE
        DO Select(database connections(partition_no),
bulk_data_partitions(partition_no, partition_no) as a thread
    ENDWHILE

    WRITE an entry for the action 'query' in the result file (config param
'file.result.name')

```

3.6 Select Function

```

run_select(database connection, bulk_data_partition, partition_no)
INPUT: the database connection
       the bulk data partition
       the current partition number

save the current time as the start of the 'query' action

count = 0

execute the SQL statement in config param 'sql.select'

WHILE iterating through the result set
    count + 1
ENDWHILE

IF NOT count = size(bulk_data_partition)
    display an error message
ENDIF

```

4. Driver Specific Features

4.1 Oracle cx_Oracle and Python 3

- all configuration parameters are managed by the program OraBench.java and made available in a suitable file (`file.configuration.name.python`)
- Python 3 uses for batch operations the `executemany` method of the `cursor` class for the operation `INSERT`
- the value fetch size (`connection.fetch.size`) is not used because the operation `SELECT` uses the operation `Cursor.fetchall()`

4.2 Oracle JDBC and Java

- the Java source code is compiled with the help of Gradle
- Java uses the `PreparedStatement` class for the operations `INSERT` and `SELECT`
- Java uses for batch operations the `executeBatch` method of the `PreparedStatement` class for the operation `INSERT`

4.3 Oracle JDBC and Kotlin

- the Kotlin source code is compiled with the help of Gradle
- Kotlin uses the `PreparedStatement` class for the operations `INSERT` and `SELECT`
- Kotlin uses for batch operations the `executeBatch` method of the `PreparedStatement` class for the operation `INSERT`

4.4 Oracle ODPI-C and C++ (gcc)

- all configuration parameters are managed by the program `OraBench.java` and made available in a suitable file (`file.configuration.name.c`)

4.5 oranif and Erlang

- all configuration parameters are managed by the program `OraBench.java` and made available in a suitable file (`file.configuration.name.erlang`)

5. Reporting

[see here](#)

6. Operating Instructions

5.1 Ubuntu 20.04 LTS (including VMware)

- **Requirements:**
 - Ubuntu 20.04 installed directly or via VMware
 - run `sudo apt update`
 - run `sudo apt install dos2unix git`
 - add the following lines to `.bash_profile`:

```
if [ -f ~/.bashrc ]; then
    source ~/.bashrc
fi
```

- run `export DOCKER_USERNAME=\<user name\>`
- run `export DOCKER_PASSWORD=\<password\>`
- run `git clone https://github.com/KonnexionsGmbH/ora_bench` (cloning the **OraBench** repository)
- run `cd ora_bench`
- run `./scripts/run_prep_bash_scripts.sh` (preparing the shell scripts)

- run `./scripts/run_install_4_ubuntu_20.04_vm_wsl2.sh` (setting up the WSL2 environment)
- close the Ubuntu shell and reopen it again
- run `cd ora_bench`
- **Execution:** run `./run_bench_all_dbs_props_[std|var].sh`

5.2 Ubuntu 20.04 LTS and `kxn_dev` Image

- **Requirements:**
 - pull the `kxn_dev` image from DockerHub: `docker pull konnexionsgmbh/kxn_dev:latest`
 - create an appropriate container: `docker run -it --name kxn_dev -v /var/run/docker.sock:/var/run/docker.sock konnexionsgmbh/kxn_dev:latest bash`
 - run `export DOCKER_USERNAME=<user name>`
 - run `export DOCKER_PASSWORD=<password>`
 - run `git clone https://github.com/KonnexionsGmbH/ora_bench` (cloning the **OraBench** repository)
 - run `cd ora_bench`
 - run `./scripts/run_prep_bash_scripts.sh` (preparing the shell scripts)
 - run `gradle copyJarToLib`
- **Execution:** `./run_ora_bench.sh`
- **Issues:**
 - Trino Distributed Query Engine and Microsoft SQL Connector

5.3 Ubuntu 20.04 LTS and Windows Subsystem Linux 2

- **Requirements:**
 - install Ubuntu 20.04 from Microsoft Marketplace
 - run `sudo apt update`
 - run `sudo apt install dos2unix`
 - add the following lines to `.bash_profile`:

```
if [ -f ~/.bashrc ]; then
    source ~/.bashrc
fi
```

- activate the **WSL INTEGRATION** for Ubuntu 20.04 in Docker
- **Requirements (continued):**
 - run `export DOCKER_USERNAME=<user name>`

- run `export DOCKER_PASSWORD=\<password\>`
- run `git clone https://github.com/KonnexionsGmbH/ora_bench` (cloning the **OraBench** repository)
- run `cd ora_bench`
- run `./scripts/run_prep_bash_scripts.sh` (preparing the shell scripts)
- run `./scripts/run_install_4_ubuntu_20.04_vm_wsl2.sh` (setting up the WSL2 environment)
- close the Ubuntu shell and reopen it again
- run `cd ora_bench`
- run `gradle copyJarToLib`
- **Execution:** run `./run_ora_bench.sh`
- **Issues:**
 - Trino Distributed Query Engine and Microsoft SQL Connector
 - YugabyteDB and Docker image

5.4 Windows 10 Pro

- **Requirements:**
 - run `set DOCKER_USERNAME=\<user name\>`
 - run `set DOCKER_PASSWORD=\<password\>`
 - run `git clone https://github.com/KonnexionsGmbH/ora_bench` (cloning the **OraBench** repository)
 - run `cd ora_bench`
- **Execution:** run `run_ora_bench.bat`
- **Issues:**
 - Trino Distributed Query Engine and Microsoft SQL Connector
 - YugabyteDB and Docker image