

OraBench - Benchmark Framework for Oracle Database Drivers.

build passing release v1.0.0 release date last tuesday commits since 1.0.0 2

Table of Contents

- 1. Introduction**
 - 2. Framework Tools**
 - 2.1 Benchmark Configuration**
 - 2.2 Installation**
 - 2.3 Benchmark Operation**
 - 2.4 Benchmark Results**
 - 2.5 Bulk File**
 - 3. Coding Pattern**
 - 3.1 Benchmark Function` (main function)**
 - 3.2 Trial Function**
 - 3.3 Insert Control Function**
 - 3.4 Insert Function**
 - 3.5 Select Control Function**
 - 3.6 Select Function**
 - 4. Driver Specific Features**
 - 4.1 Oracle cx_Oracle and Python**
 - 4.2 Oracle JDBC and Java**
 - 4.3 Oracle JDBC and Kotlin**
 - 4.4 Oracle ODPI-C and C++ (gcc)**
 - 4.5 oranif and Erlang**
-

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)
Oraclejl	Julia

Driver	Programming Language(s)
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](#).

OraBench is tested under [Ubuntu](#).

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

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

For changes to the **OraBench** repository it is best to use an editor (e.g. [Vim](#)) or a suitable 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](#).

The whole software environment for the operation and further development of OraBench can be created most easily by using a Docker container based on the Konnexions Development Image (version 2.0.4 from [here](#)).

Alternatively, in an Ubuntu 20.04 based environment, e.g.: in a virtual machine, the two following scripts can be used to install the necessary software:

- `scripts/kxn_dev/run_install_4-vm_wsl2_1.sh`
- `scripts/kxn_dev/run_install_4-vm_wsl2_2.sh`
 - run `sudo apt update`
 - run `sudo apt install git`
 - run `git clone https://github.com/KonnexionsGmbH/ora_bench` (cloning the **OraBench** repository)
 - run `cd ora_bench/scripts/kxn_dev`
 - run `./run_install_4_vm_wsl2_1.sh`
 - close the Ubuntu shell and reopen it again
 - run `cd ora_bench/scripts/kxn_dev`
 - run `./run_install_4_vm_wsl2_2.sh`

2.3 Benchmark Operation

2.3.1 Script `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 Script `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.

2.4 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. In the file directory `priv/statistics` reference statistics files are available per version of **OraBench**.

Excerpts from a sample file can be seen in the following image:

release	host name	no. cores	os	database	language	driver	trial no.	fetch size	transaction size	bulk length	bulk size	batch size	action	duration (sec)	duration (ms)
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	1	1024	512	1024	100000	256	query	2	1863205000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	1	1024	512	1024	100000	256	query	0	352902000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	1	1024	512	1024	100000	256	trial	3	2607945000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	2	1024	512	1024	100000	256	query	1	1379481000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	2	1024	512	1024	100000	256	query	0	327020000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	2	1024	512	1024	100000	256	trial	2	1975994000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	3	1024	512	1024	100000	256	query	1	1420224000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	3	1024	512	1024	100000	256	query	0	394371000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	3	1024	512	1024	100000	256	trial	2	2068830000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Python 3.10.0b2 (default, Jun 13 2021, 13:52:49) [GCC 10.2.0]	Oracle cx_Oracle (Version v8.2.1)	0	1024	512	1024	100000	256	benchmark	8	7597838000
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	1	1024	512	1024	100000	256	query	1	701127359
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	1	1024	512	1024	100000	256	query	0	99706717
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	1	1024	512	1024	100000	256	trial	1	800841756
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	2	1024	512	1024	100000	256	query	1	608365334
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	2	1024	512	1024	100000	256	query	0	103085171
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	2	1024	512	1024	100000	256	trial	1	711452835
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	1	1024	512	1024	100000	256	query	1	547438903
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	3	1024	512	1024	100000	256	query	0	107924057
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	3	1024	512	1024	100000	256	trial	1	655365120
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Go go1.16.5	godror v0.25.2	0	1024	512	1024	100000	256	benchmark	3	2828905672
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	1	1024	512	1024	100000	256	query	2	1590996377
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	1	1024	512	1024	100000	256	query	0	188452505
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	1	1024	512	1024	100000	256	trial	2	2317318877
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	2	1024	512	1024	100000	256	query	1	1445069036
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	2	1024	512	1024	100000	256	query	0	164810917
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	2	1024	512	1024	100000	256	trial	2	1857446448
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	3	1024	512	1024	100000	256	query	1	1447572133
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	3	1024	512	1024	100000	256	query	0	170519255
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	3	1024	512	1024	100000	256	trial	2	1928886239
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Kotlin 1.4.32	Oracle JDBC (Version 21.1.0.0.0)	0	1024	512	1024	100000	256	benchmark	7	7165612898
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	1	1024	512	1024	100000	256	query	2	1703377743
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	1	1024	512	1024	100000	256	query	0	289754072
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	1	1024	512	1024	100000	256	trial	2	2267529968
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	2	1024	512	1024	100000	256	query	1	1449423695
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	2	1024	512	1024	100000	256	query	0	179636373
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	2	1024	512	1024	100000	256	trial	2	1924773590
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	3	1024	512	1024	100000	256	query	1	1481805818
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	3	1024	512	1024	100000	256	query	0	182336324
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	3	1024	512	1024	100000	256	trial	2	1980840585
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	Java 16.0.1	Oracle JDBC (Version 21.1.0.0.0)	0	1024	512	1024	100000	256	benchmark	7	7345812121
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	gnu 10.2.0	OCPI-C (v4.2.1)	1	1024	512	1024	100000	256	query	0	462500155
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	gnu 10.2.0	OCPI-C (v4.2.1)	1	1024	512	1024	100000	256	query	0	96594704
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	gnu 10.2.0	OCPI-C (v4.2.1)	1	1024	512	1024	100000	256	trial	0	830490595
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	gnu 10.2.0	OCPI-C (v4.2.1)	2	1024	512	1024	100000	256	query	0	284975919
1.0.0	ubuntu	8	amd64 / Linux / 5.8.0-55-generic	db_18_4_xe	gnu 10.2.0	OCPI-C (v4.2.1)	2	1024	512	1024	100000	256	query	0	107636133

In detail, the following information is available in the result files:

Column	Format	Content
release	alphanumeric	config param benchmark.release
benchmark id	alphanumeric	config param benchmark.id
benchmark comment	alphanumeric	config param benchmark.comment
host name	alphanumeric	config param benchmark.host.name
no. cores	integer	config param benchmark.number.cores
os	alphanumeric	config param benchmark.os
user name	alphanumeric	config param benchmark.user.name
database	alphanumeric	config param benchmark.database
language	alphanumeric	config param benchmark.language
driver	alphanumeric	config param benchmark.driver
trial no.	integer	0 if action equals benchmark , trial no. otherwise
SQL statement	alphanumeric	SQL statement if action equals query , empty otherwise
core multiplier	integer	config param benchmark.core.multiplier
fetch size	integer	config param connection.fetch.size
transaction size	integer	config param benchmark.transaction.size
bulk length	integer	config param file.bulk.length
bulk size	integer	config param file.bulk.size
batch size	integer	config param benchmark.batch.size

Column	Format	Content
action	alphanumeric	one of benchmark , query or trial
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
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`)