Performance Evaluation of an RDB and an ORDB: A Comparative Study using the BUCKY Benchmark

Abdelsalam M. Maatuk[1], M. Akhtar Ali[2], Raja A. Moftah[3], Mohammed R. Elkobaisi[4]
[1,3]Faculty of Information Technology, Benghazi University, Libya
[2]Faculty of Engineering and Environment, Northumbria University, UK
[4]Department of Computer Science, Omer AL-Mukhtar University, Libya

Address
[1]abdelsalam.maatuk@uob.edu.ly
[2]akhtar.ali@northumbria.ac.uk
[3]rajaMoftah@hotmail.com
[4]elkobaisi@gmail.com

***Abstract*— This paper highlights the functionality of object-based database systems by comparing the performance of relational database (RDB) and object-relational database (ORDB) systems. The study focuses on assessing the efficiency of database systems based on query processing and object complexity. We conducted an experiment that includes running the queries on the RDB and ORDB that were used in the BUCKY benchmark and implemented on Oracle 11*g*. The findings of this research show that the performance of both database systems depends on various factors, such as the size and type of databases, the schema and query structures, the number of tuples scanned in tables, indexes as well as the environment, in which the experiment was carried out.**

***Keywords*— Relational database, Object-relational database, benchmarks**

I. INTRODUCTION

Object-Relational databases (ORDBs) are starting to emerge in the market providing more functionality and flexibility [1]. The advantages provided by these technologies and the dominance of traditional relational databases (RDBs) and their weaknesses in handling data of a complex nature have motivated a growing trend of migrating RDBs into ORDBs instead of designing them from scratch. Database migration is concerned with the process of converting schema and data from a source RDB, as a one-time conversion, into a target database to be managed and handled in its new environment [2,14]. The target database may be accessed through the concepts of its data models with a reduced overhead in term of performance compared to an existing RDB. Furthermore, since a building information model size increases, query and performance issues become more interested [3].

This paper describes an experiment designed to explore the efficiency of query processing for an RDB and the equivalent ORDB created in Oracle 11*g* DBMS. We have been designed a query-based experiment based on the BUCKY benchmark [4]. The BUCKY benchmark and its queries is a published, fully released and freely available benchmark. The benchmark consists of an RDB and ORDB, including their semantically equivalent schema, data and sets of queries. The experiment has been designed to test many of the key features offered by ORDB systems in relation to RDB systems. The tested features include row types and inheritance, references and path expressions, sets of atomic values and references, and user-defined data types along with their methods. Most DBMS performance evaluations consider measurement of the query elapsed time, which is the amount of time query statements take to execute. The type of the evaluation is a comparison-based, in which we load the RDB and the ORDB into their systems to check and compare their performance.

Although not a direct issue for database migration, comparing the performance of input and output databases as results of the migration process may help the users to decide whether or not they should move into their chosen database if performance is a deciding factor. This study could assist in evaluating and choosing the most appropriate database to adopt for non-relational applications to be developed according to functionality, performance and suitability, and could help increase their acceptance among enterprises and practitioners.

This paper is structured as follows. A general overview of the related work is presented in Section II. In Section III, a detailed description of an experimental environment is introduced. Section IV describes the results obtained from applying the queries of the benchmark on Oracle 11*g* DBMS. Section V discusses the results and presents lessons learned from this study, and Section VI concludes this paper.

II. RELATED WORK

Several query-based benchmarks have been designed to test and measure different aspects of object-based systems' functionalities and performances [4, 5]. These benchmarks can be used to test the performance of databases based on a pre-defined criteria. It is very important evaluation issue for benchmarks that concern systems performance efficiency. A four primary criteria is defined by [6] to specify a good benchmark, including relevance, portability, simplicity and scalability. A perspective on the points where the benchmarks should focus, how they should be structured to test the performance of the databases satisfactorily is provided in [7]. We have noticed that a set of benchmarks are receiving more acceptance and interest.

The OO1 [8] and OO7 [5] benchmarks were designed to evaluate the performance of OODBMS. BUCKY

[4] and BORD [9] benchmarks are for ORDBMSs. The OO7 benchmark represents a comprehensive test of the wide range of OO features of OODBMS performance [5]. The OO7 benchmark includes three clusters of operation: traversals, queries and structural modifications. The BUCKY benchmark is a query-oriented, which has been developed to test the maturity of an ORDB system's key features in relation to an RDB system [4]. It was implemented in an early ORDB system i.e., Illustra 97. The benchmark tests many of the key features of ORDBs, including row types and inheritance, references and path expressions, sets of atomic values and of references, methods and late binding and user-defined abstract data types along with their methods. Comparisons of the performance of ODBMSs and ORDBMSs using Db4o with a hybrid database solution on an artificial dataset are described in [10, 11]. However, the OO7 benchmark is re-implemented for performance evaluation in [11], whereas an object oriented application with focus on the complexity of objects is expressed in [10]. A performance evaluation of results for an RDB and ORDB-based IFC servers using the BUCKY benchmark are reported in [3, 12]. However, although the performance improvement ORDB server has been validated, many issues, e.g., data merge, subset model extraction, and large-model handling issues remain to be resolved. We conducted a benchmark applied on Oracle 10*g*, the outcomes of which have been discussed and evaluated against the results presented in the original BUCKY tests [13, 15].

III. EXPERIMENTAL ENVIRONMENT

This section explains how system has been setup, and a description of the experiment is described.

A. Database Descriptions

1) Relational Implementation

The RDB used in the experiment reported here is based on the university database (UniDB) used in the BUCKY benchmark [4]. Fig. 1 shows the logical UniDB schema, which includes the relations: Department, Person, Employee, Student, Staff, Instructor, TA, Professor, Course, CourseSection, Enrolled and Kids. The relationships are modelled using primary/foreign key pair. Once UinDB schema is are created, the data is bulk-loaded into it using SQL loader.

2) ORDB Implementation

The ORDB version of UniDB is generated by our system [2] in a folder, which contains a schema file, files for object definitions, files for relationship definitions, constraints files and a file contains a program, which runs these files in priorities in order to create the database. We proposed a conversion program for automatically migrating RDBs into ORDBs [2]. The program enacts the schema file firstly and then the files those contain object definitions. Files contains keys, indexes and other constraints are loaded into databases before relationship files. To speed up the response time in query processing, we created (after objects have been initialised) appropriate types in indexing.

DEPARTMENT (deptno, name, building, budget, *chair*, latitude, longitude) *chair* → PROFESSOR

COURSE (*deptno*, courseno, name, credits)
deptno → DEPARTMENT

COURSESECTION((*deptno*, *courseno*), sectionno, semester, *instructorid*, textbook, nostudents, building, roomno)
deptno, courseno → COURSE, *instructorid* → INSTRUCTOR

PERSON (id, name, street, city, state, zipcode, birthdate, picture, latitude, longitude)

EMPLOYEE (*id*, *dept*, datehired, status)
id → PERSON, *dept* → DEPARTMENT

INSTRUCTOR (*id*)
id → EMPLOYEE

STAFF (*id*, annualsalary)
id → EMPLOYEE

PROFESSOR (*id*, aysalary, monthsummer)
id → INSTRUCTOR

STUDENT (*id*, studentno, *majordept*, *advisor*)
id → PERSON, majordept → DEPARTMENT, *advisor* → PROFESSOR

TA (*id*, semestersalary, apptfraction)
id → INSTRUCTOR

KIDS (*id*, kidname)
id → EMPLOYEE

ENROLLED (*studentid*, (*deptno*, *courseno*, *sectionno*, *semester*), grade)
studentid → STUDENT,
deptno, courseno, sectionno, semester → COURSESECTION

Fig. 1: Logical Relational Schema for the UniDB

3) Database Sizes

We have worked with up to 27.5M of RDB data and up to 115M of corresponding data ORDB. The size difference comes from the update statements in the ORDB input files. Although the RDB version of UniDB is a relatively small data set, we have found that it is sufficient to evaluate the DBMS performance using it and its corresponding ORDB data. RDB data have been loaded to Oracle using SQL*Loader, which is a very efficient data loading tool. It was much faster than loading the script files generated by our program. As ORDB object definition and relationship files contain thousands of insert into and update statements, it was expected that loading these files would take much longer than using SQL*Loader, especially for object relationship files. We have loaded the RDB data and ORDB object definition files before creating any indexes since indexes increase the object loading time. Before loading ORDB object relationship files, we created indexes on user-defined object identifiers, which speed up the process of establishing relationships among objects.

B. Test Bed Configuration

In this study, BUCKY is implemented in Oracle 11*g* on a standalone PC with 3.2 GHz processor and 2GB of RAM under Windows 7. To ensure a secure and stable environment, the computer is isolated so that fluctuations in the network activity cannot affect the execution of the benchmark queries. All queries were run with the buffer pool empty as the Oracle system was shut down and restarted for each query. Both RDB and ORDB schemas are created in two separate table spaces under two different users so that running the queries in either schemas are completely isolated and have no impact on each other. The SQL*Plus TIMING command is used to collect and display elapsed time on the amount of computer resources used to run the queries. Necessary indexes are created after the data has been bulk-loaded, so as not to slow down the bulk loading process.

C. *Cost Metrics*

The query elapsed time is measured as performance metric. While we were obtaining elapsed times in repeating the query many times, it was found that apart from the first reading, all the subsequent elapsed times were somewhat similar. Thus the average was taken from the second to the fourth time readings.

D. *Queries*

The criteria we have used in the queries includes:

- Queries should be simple and basic operations are supported in both database systems.
- Queries should focus on the equivalence between both databases in data capacity, semantics preservation, efficiency and speed of retrieval of data from the system.
- The fundamental areas that should be covered by the queries include inheritance, object relationships, user-defined types and integrity constraints.

Followings are the essential query types selected to be used in our experiment.

- *Selection*: This type of query is selection including single and complex with relational operators.
- *Exact Match Lookup*: This type of query tests the database ability to handle simple string lookups as simple exact-match or over inheritance hierarchies.
- *Joins*: This query tests database's ability for join processing including single and inheritance joins.
- *Set Operations*: This type of query tests the computing of mathematical operations.
- *Set Membership*: This type of query tests for set membership, where the set is a collection of values extracted by selection statements.
- *Path-expressions*: This query tests the ability of handling references to persistence objects. A path expression, including a navigation path through a relationship in an ORDB is similar to outer join in an RDB.
- *UDT-based Data:* This query is for retrieving data stored as simple/composite multi-valued attributes or weak entities.

E. *Indexing*

To speed up the response time in query processing, we created other appropriate indexes, which are defined considering the queries and what data would be retrieved. Foreign keys are indexed, whereas primary keys have default indexes in Oracle. Nested tables have been indexed on NESTED_TABLE_ID. The `salary()` function, which is used to calculate employee salaries has also indexed.

IV. EXPERIMENTAL RESULTS

This section presents the experiment queries, what each query is intended to test, and results of running them on an RDB and ORDB versions of the benchmark on Oracle 11*g*. A set of queries and their results were presented, indicating the intended coverage for each query, regarding data retrieval performance. As Oracle 11*g* supports scoped references, the ORDB has been queried with and without index/scoped references. Table I shows the measured time (in seconds) as indexed and unindexed for RDB queries and indexed/scoped and unindexed/unscoped for ORDB queries. The times are shown as variant A/variant B for some queries. In addition to measuring elapsed times, the EXPLAIN PLAN statement was used to determine the execution plan that Oracle DBMS follows in performing each query. This table contains the necessary metrics, including the cost of executing the query, and CPU and I/O costs for any indexes defined in the table.

TABLE I: MEASURED TIMES IN SECONDS FOR QUERIES

Query	Relational		Object-relational		rows selected
	IN	UI	IS	UU	
1-SINGLE-XACT	0.00	0.01	0.00	0.00	1
2- HIER-EXACT	0.00	0.01	0.00	0.01	1
3- SINGLE-METH	0.24	0.25	00.23	496.80	2014
4- HIER-METH	1.03	1.00	0.96	737.61	2788
5- SINGLE-JOIN	1.28	1.21	1.28	1.26	3044
6- HIER-JOIN	0.34	0.39	0.03	0.03	1
7- SET-ELEMENT	0.21	0.14	0.10	0.11	277
8- SET-AND	0.13	0.15	0.12	0.12	277
9- 1HOP-NONE	43.07	43.07	43.13	43.12	75000
10- 1HOP-ONE	0.00	0.03	0.00	0.46	1
11- 1HOP-MANY	0.04	0.07	0.04/0.03	0.07/0.48	318
12- 2HOP-ONE	0.07	0.07	7.22/0.06	61.8/0.31	530
Sum:	46.41	46.40	45.94	1279.90	
IN: Indexed	**UI: Unindexed**		**IS: indexed/scoped**	**UU: unindexed/unscoped**	

A. *Query 1: SINGLE-EXACT*

Find the name, building and budget of the department with number 1.

RDB:	select name, building, budget from department where deptno = 1;
ORDB:	select name, building, budget from department where deptno = 1;

This query tests exact match look up over a single table. As the RDB and ORDB tables have the same number of attributes, tuples and indexes, the result times were identical. The cost (0.00s) estimates were equal for both queries before and after indexing and scoping.

B. *Query 2: HIER-EXACT*

Find the name and annual salary of the staff with id 2

RDB:	select p.name, s.annualsalary from person p, staff s where s.id = p.id and s.id = 2;
ORDB:	select name, annualsalary from staff where id = 2;

This query assesses system efficiency in managing queries over inheritance hierarchies. Although indexing/scoping increases the time taken slightly (from 0.00s to 0.01s), all queries performed very similarly with respect to time. As the union operation was hidden in the query, the ORDB version was more natural and simple than the RDB query.

C. Query 3: SINGLE-METH

Find IDs of Professors who make 145000 or more per year.

RDB:	select id from professor p where (p.aysalary * (9 + p.monthsummer)/ 9.0) >= 145000;
ORDB:	select id from professor p where (p.aysalary * (9 + p.monthsummer)/9.0) >= 145000;

This query compares performance time for calculating data stored in attributes in the RDB with invoking functions in the ORDB. In the ORDB, we used the salary() function (shown in Fig. 2) to calculate the salaries of the professors in Variant B of the query. Without indexes/scopes, the ORDB query was painfully slow (496.80s). The bad performance, was because of the range scans that have been made by the optimizer to all nested tables in Professor table. To speed up the execution time, the nested tables are indexed, which improves the performance with the time dropping to 11.90s. Even this length of time shows that the ORDB query is still slow, compared to RDB time (0.25s). However, the performance was enhanced considerably when an index was created on the function. After indexing the function, the ORDB time (0.23s) shows that the system is more efficient, compared to the complex predicates of the RDB query.

```
create or replace type body Professor_t as
overriding member function salary return number is
  begin
     return (aysalary * (9 + monthsummer) / 9.0 );
  end;
end;
```

Fig. 2: The salary() function for Professor_t type

Variant B: select id, aysalary from professor p where p.salary() >= 145000;

D. Query 4: HIER-METH

Find names and addresses of all Employees who make 140000 or more per year.

RDB:	select p.name, p.street, p.city, p.zipcode from person p, staff s where p.id = s.id and s.annualsalary >= 140000 union select p.name, p.street, p.city, p.zipcode from person p, professor f where p.id = f.id and (f.aysalary * (9 + f.monthsummer) / 9.0) >= 140000 union select p.name, p.street, p.city, p.zipcode from person p, ta t where p.id = t.id and apptfraction * (2 * t.semestersalary) >= 140000;
ORDB: Variant A	select s.name, s.street, s.city, s.zipcode from staff s where s.annualsalary >= 140000 union select p.name, p.street, p.city, p.zipcode from professor p where (p.aysalary * (9 + p.monthsummer) / 9.0) >= 140000 union select t.name, t.street, t.city, t.zipcode from ta t where apptfraction * (2 * t.semestersalary) >= 140000;

This query tests the system efficiency in invoking indexed functions over inheritance. Without indexes/scopes and unindexed function, the ORDB query was very slow (737.61s). Similar to SINGLE-METH, the performance of Oracle improved significantly, with a response time of 0.96s for the ORDB, after the function was indexed, and was then faster than the relational time of 1.03s.

Variant B: select s.name, s.street, s.city, s.zipcode from staff s where s.salary()>=140000 union select p.name, p.street, p.city, p.zipcode from professor p where p.salary() >= 140000 union select t.name, t.street, t.city, t.zipcode from ta t where t.salary() >= 140000;

E. Query 5: SINGLE-JOIN

Find names, buildings and budgets of departments with the same budget.

RDB:	select d1.name, d1.building, d1.budget, d2.name, d2.building, d2.budget from department d1, department d2 where d1.budget = d2.budget and d1.deptno < d2.deptno;
ORDB:	select d1.name, d1.building, d1.budget, d2.name, d2.building, d2.budget from department d1, department d2 where d1.budget = d2.budget and d1.deptno < d2.deptno;

This query is the baseline test for RDB join operations. As the structures of both queries were the same, the query times and the execution plans were similar. Although the system seems slower with indexes (1.28s), the results show that Oracle is efficient in handing join operations in both RDB and ORDB.

F. Query 6: HIER-JOIN

Find all TAs with the same hired date as those live in the same zip code area.

RDB:	select p1.id, p1.name, p2.id, p2.name from person p1, person p2, employee e1, employee e2, ta t1, ta t2 where e1.datehired = e2.datehired and p1.zipcode = p2.zipcode and p1.id < p2.id and p1.id = e1.id and p2.id = e2.id and p1.id = t1.id and p2.id = t2.id;
ORDB:	select t1.id, t1.name, t2.id, t2.name from ta t1, ta t2 where t1.datehired = t2.datehired and t1.zipcode = t2.zipcode and t1.id < t2.id;

This query tests the efficiency of the system in handling joins among inheritance hierarchies. Executing this query, Oracle was almost 10 times faster with ORDB compared to the RDB query, with similar performance before and after indexing and scoping with times of 0.03s. The relational times were slower at 0.34s and 0.39s before and after indexing, respectively.

G. Query 7: SET-ELEMENT

Find ids, names and addresses of all staff who have a child named boy90.

RDB:	select p.id, p.name, p.street, p.city, p.state, p.zipcode from person p, staff s, kids k where p.id = k.id and s.id = k.id and k.kidname = `boy90';
ORDB:	select s.id, s.name, s.street, s.city, s.state, s.zipcode from staff s, table (s.kidnames) k where k.kidname = 'boy90';

This query tests the system's ability to handle collection data types. The RDB query includes joins among Person, Staff and Kids tables, which make it slower than the ORDB query. The ORDB query performed better than the RDB query, which proves that Oracle is powerful in managing nested tables. An index was created on the object identifier for the kidnames_staff_nt nested table and the *kidname* attribute. However, it seems that indexing does not improve the performance and the elapsed time was still similar, although the nested table is accessed by the index range scan.

H. Query 8: SET-AND - Anded Set Membership

Find ids, names and addresses of all Staff who have children named girl90 and boy90.

RDB:	select p.id, p.name, p.street, p.city, p.state, p.zipcode from person p, staff e, kids k1, kids k2 where e.id = p.id and e.id = k1.id and e.id=k2.id and k1.kidname = 'girl90' and k2.kidname = 'boy90';
ORDB:	select s.id, s.name, s.street, s.city, s.state, s.zipcode from staff s, table (s.kidnames) k1, table (s.kidnames) k2 where k1.kidname = `girl90' and k2.kidname = 'boy90';

This query is similar to the SET-ELEMENT with a more complex structure to test the effectiveness of Oracle in handling more complex value-based collections. Although the response times of both queries were close (i.e., 0.13s and 0.15s for the RDB query and 0.12s for the ORDB query) the results show that the system is still efficient in handling value-based collection/sets of data stored in nested tables.

I. Query 9: 1HOP-NONE

Find the details of all student/major pairs.

RDB:	select p.id, p.name, p.state, d.deptno, d.name from person p, department d, student s where p.id = s.id and s.majordept = d.deptno;
ORDB:	select s.id, s.name, s.state, s.major.deptno, s.major.name from student s;

This query tests the system efficiency at processing one-hop path expressions. In the query, the entire Student table was scanned. The two versions of queries are very close in elapsed time. Although in the ORDB query, path expressions and scoped references were used, Oracle was slightly faster in the RDB query (43.07s) compared to the ORDB query (43.13s). Using scoped references, the system uses the knowledge that the ref-based attribute points to an object of a particular type (i.e., Department_t). However, indexes and scoped references do not increase performance in the ORDB query.

J. Query 10: 1HOP-ONE

Find the major of the student named studentName75001.

RDB:	select p.id, p.name, d.deptno, d.name, d.building from person p,student s, department d where p.id = s.id and s.majordept = d.deptno and p.name= `studentName75001';
ORDB:	select s.id, s.name, s.major.deptno, s.major.name, s.major.building from student s where name= 'studentName75001';

This query tests how Oracle handles a short path expression. The elapsed times of both RDB and ORDB queries with indexes were similar, whereas with unindexd/unscoped settings, the ORDB query was 15 times slower than the RDB query without an index. As bi-directional relationships are offered in the ORDB, this query can have another variant, in which the system efficiency at handling queries involving a collection of references can be tested. However, intuitively, as the data required are for a particular student where its related object contains a reference pointing to the department object, it would be better to avoid this variant.

Variant B: select s.column_value.id, s.column_value.name, d.deptno, d.name, d.building from department d, table (d.students) s where s.column_value.name= 'studentName75001';

K. Query 11: 1HOP-MANY

Find ids and names of all students majoring in Department1.

RDB:	select p.id, p.name from person p, student s, department d where p.id = s.id and s.majordept = d.deptno and d.name = 'deptname1';
ORDB:	select st.column_value.id, st.column_value.name from department d, table(d.students) st where d.name = 'deptname1';

This query tests the efficiency of Oracle at handling collections of references. The ORDB query Variant A with column_value performed well in the cases indexed/scoped (0.04s) or unindexed/unscoped (0.07s). However, Variant B with unindexed/unscoped references was slower than the RDB and the ORDB Variant A queries. The query response time was 0.46s compared to just 0.03s and 0.04s in the other equivalent queries. In other words, it was 16 times slower than the equivalent ORDB query Variant B with an index and scoped references, and 12 times slower than the equivalent RDB query with an index.

Variant B: select s.id, s.name from student s where s.major.name = `deptname1';

L. Query 12: 2HOP-ONE

Find the semester, enrolment limit, department number, and department name for sections of courses taught in room 50.

RDB:	select x.semester, x.nostudents, d.deptno, d.name from coursesection x, course c, department d where x.deptno = c.deptno and x.courseno = c.courseno and c.deptno = d.deptno and x.roomno = 50;
ORDB:	select se.column value.semester, se.column value.nostudents, d.deptno, d.name from department d, table(d.offers) co, table(co.column_value.sections) se where se.column_value.roomno = 50;

This query examines Oracle ability in handling longer path expressions. The performance of ORDB Variant A was very poor before indexing (61.8s) compared to the RDB and the ORDB Variant B. Thus, the performance of Variant A with the selection of two-hop chain set-valued references was very poor. Although the time improved (7.22s) when the references were scoped and indexes created for nested tables, we could not find a way to increase the performance of the Variant A. However, Variant B using the inverse side of the relationship performed pretty well (0.31s) compared to ORDB Variant A (61.8s). In addition, Variant B with indexes/scoped references did even better (0.06s) than RDB version of the query (0.07s).

Variant B: select s.semester, s.nostudents, s.course.dept.deptno, s.course.dept.name from coursesection s where s.roomno = 50;

V. DISCUSSIONS

In this experiment, we ran the first 12 queries used in BUCKY benchmark on the RDB UniDB and the corresponding ORDB. We loaded the entire RDB and ORDB into Oracle 11*g* to measure the performance for both versions of the queries. All the queries were run with and without indexing, and with and without scoped references for the ORDB. After analyzing the results, we can draw the following conclusions:

- The relational and object-relational elapsed times are virtually identical for all queries on a single table. Indexing and reference scoping do not improve performance in these kinds of queries.
- In single/hierarchical function queries, the elapsed times are very close. The system performance with ORDB queries improved when the functions were indexed. However, when not indexed, the ORDB query performance was very poor. That is because all nested tables, embedded in accessed object tables, are scanned while invoking the functions.
- The system with the ORDB version of HIER-JOIN query was faster than in the RDB query, verifying that the ORDB outperforms the RDB in handling inheritance and traditional join operations.
- In handling SET-ELEMENT and SET-AND queries, the system was slightly faster with ORDB than with the RDB queries. The results verify that Oracle is more efficient in handling value-based collection data type stored in nested tables. The ORDB with set value-based attributes succeeds over relational joins. Indexing/scoped references make no difference to performance in both versions of the queries.
- By looking at path expression queries, it can be noticed that the elapsed times for RDB and ORDB queries were almost identical. The 1HOP-NONE times were more or less the same in both of the query versions. This is for indexed/unindexed RDB queries and only indexed/scoped ORDB queries. In addition, using column_value for de-referencing objects was effective for the 1HOP-MANY ORDB query. However, the time taken for the 2HOP-MANY query with unindexed/unscoped references was obviously slow. Oracle was inefficient in managing queries of two-hop chain of ref-based collections. As relationships in the ORDB schema are defined bi-directionally, we used the opposite direction in this query, i.e., the M side of the relationship. For this option with indexing nested tables and scoped references, the query performance much improved. Hence, for ORDB queries with index and reference scoping, Oracle was faster in handling path expressions than in the RDB queries.
- The performance of the system is directly affected by the number of tables and attributes, and also by the structure of the query and the number of rows in each table. The query structure in ORDB queries is more simple and concise than in relational ones.
- After having the summation of the elapsed times of each set of queries, the ORDB efficiency with indexed/scoped data was slightly better than the RDB queries. However, the ORDB query with unindexes/unscoped references was painfully slow. The overall time of RDB queries with indexes was 46.41s and without indexes was 46.40s. The overall time of ORDB queries with indexes/scoped references was 45.94s and with unindexes/unscoped references was 1279.90s.

VI. CONCLUSIONS

This paper evaluates the efficiency of RDB and ORDB systems in terms query processing. An experiment has been conducted, which includes running the queries used in the BUCKY benchmark. The queries are implemented on Oracle 11*g*. In the experiment, we have measured the elapsed time as query processing metric. Comparing the RDB queries with their equivalents in an ORDB, it was found that the system is more efficient in handling ORDB queries over inheritance hierarchies, indexed functions, path expressions and set element queries. In addition, the structure of ORDB queries is more simple and concise than the RDB ones. The ORDB queries with indexed/scoped data was slightly efficient than that of the RDB, whereas the ORDB queries with unindexes/unscoped references was painfully slow. The system performance with the RDB queries is not improved when data were indexed. The ORDB queries with indexes/scoped references are slightly more efficient compared to the RDB queries. In addition, the performance of the system is directly affected by the number of tables and attributes in each query, and the query structure as well as the number of rows in each table.

REFERENCES

[1] J. M. Stonebraker, P. Brown and D. Moore. (1999). *Object-Relational DBMSs: The Next Great Wave and Object-Relational DBMSs: Tracking the Next Great Wave.* Morgan Publishers.

[2] A. Maatuk, M. A Ali and B. N. Rossiter. (2010). Converting relational databases into object relational databases. In *Journal of Object Technology,* vol. 9(2), pp. 145-161.

[3] G. Lee, J. Jeong, J. Won, C. Cho, S. You, S. Ham and H. Kang. (2014). Query Performance of the IFC Model Server Using an Object-Relational Database Approach and a Traditional Relational Database Approach. In *Jour. Comput. Civ. Eng.,* vol. 28, pp. 210-222.

[4] M. Carey, D. DeWitt, J. Naughton, M. Asgarian, P. Brown, J. Gehrke and D. Shah. (1997). The BUCKY object-relational benchmark. In *SIGMOD Rec.*, vol. 26(2), pp. 135–146.

[5] M. Carey, D. DeWitt and J. Naughton. (1993). The OO7 benchmark. In *SIGMOD Rec.,* vol. 22(2), pp. 12–21.

[6] J. Gray. (1993). *The Benchmark Handbook for Database and Transaction Systems* (2nd Edition). Morgan Kaufmann.

[7] V. Geetha and N. Sreenath. (2012). Augmenting the Performance of Existing OODBMS Benchmarks. In *Int. Jour. of Comp. Appl,* vol. 40(5).

[8] R. G. Cattell and J. Skeen. (1992). Object operations benchmark. In *ACM Trans. Database Syst.,* vol. 17(1), pp. 1–31.

[9] S. H. Lee, S. Kim and W. Kim. (2000). The BORD benchmark for object-relational databases. In *Proc. of DEXA '00*, pp. 6–20.

[10] R. Kalantari and C. Bryant. (2010). Comparing the Performance of Object and Object Relational Database Systems on Objects of Varying Complexity. In *BNCOD'10*, Springer-Verlag, Berlin.

[11] P. Van zyl, G. Derrick and A. Boake. (2006). Comparing the Performance of Object Databases and ORM tools. In *Proc of South African Institute for Comp. Sci. and Info. Technologists*, pp. 1-11.

[12] J. Jeong, G. Lee and H. Kang. (2010). Preliminary Performance Evaluation of an ORDB-based IFC Server and an RDB-based IFC Server by Using the BUCKY Benchmark Method. In *Proc of CIB World Congress*. Salford, UK, pp. 192 -201.

[13] N. Keivani, A. M. Maatuk, S. Aljawarneh and M. Akhtar. (2015). Towards the Maturity of Object-Relational Database Technology: Promises and Reality. *The Int. Jour. of Technology Diffusion (IJTD),* vol. 6(4) , pp. 1-19, doi:10.4018/IJTD.2015100101.

[14] A. M. Maatuk, M. Akhtar and N. Rossiter, N. (2011). Re-Engineering Relational Database: The Way Forward. In *ISWSA* 2011, Jordan, ACM, pp. 18.

[15] A. M. Maatuk, M. Akhtar and S. Aljawarneh. (2015): Translating Relational Database Schemas into Object-based Schemas: University Case Study. In *Recent Patents on Computer Science* . Innovations in Educ. Tech. and E-learning Social Networking, vol. 8(2), pp. 11.

[16] A. M. Maatuk, M. A. Akhtar and S. Aljawarneh. 2015: An algorithm for constructing XML Schema documents from relational databases. In *Proceeding of ACM International Conference on Engineering & MIS* (ICEMIS '15). ACM, New York, NY, USA, Article 12, 6 pages. DOI=http://dx.doi.org/10.1145/2832987.2833007