Towards the Maturity of Object-Relational Database Technology: Promises and Reality

Negin Keivani1, Abdelsalam M. Maatuk1, M. Akhtar Ali2, Shadi Aljawarneh3
1Faculty of Sciences, Omar Al-Mukhtar University, Libya
2School of Computing, Engineering & Information Sciences, Northumbria University, UK.
3Software Engineering Department, Al-Isra University, Jordan

Abstract: Object-relational technology provides a significant increase in scalability and flexibility over the traditional relational databases. The additional object-relational features are particularly satisfying for advanced database applications that relational database systems have experienced difficulties. The key factor to the success of object-relational database systems is their performance. To investigate the performance implications of using object-relational relative to relational technology, the query-oriented BUCKY benchmark has been previously applied to an early object-relational database system, i.e., Illustra 97. This paper presents the results obtained from implementing and running the BUCKY benchmark on Oracle 10g. The results acquired from the work described in this paper are compared with the results obtained in BUCKY benchmark. This study throws light on the functionality of object-relational databases, where object-relational technology has made improvements but some limitations are identified as well. In general, the performance of relational supersedes that of object-relational database system.

Keywords: Relational database, Object-relational database, benchmarks

1. Introduction

The database system is arguably the most important development in the field of software engineering, and the database systems are now the underlying framework of information systems [6]. Object-relational (O-R) database systems extend relational database systems (RDBMSs) by adding modelling primitives from object world such as support for user-defined data types, references, inheritance and polymorphism. There is significant demand for object-relational capabilities; otherwise, vendors would not have uniformly embraced the technology [13].

O-R database management systems (ORDBMSs) provide a huge increase in flexibility and support for rich data types compared to the RDBMSs. They are attractive, especially for non-traditional applications, such as geographic information systems, multimedia databases, etc, for which RDBMSs have experienced their shortcomings. Unlike object-oriented database systems, ORDBMSs support all features of RDBMSs including scalability and transaction processing performance. Moreover, language support for the new features of ORDBMSs has been added to recent releases of SQL standard, namely SQL:99 and SQL:2003.

So far, little is known about the performance implications of using these features [4]. BUCKY (Benchmark of Universal or Complex Query Ynterfaces) is a query-oriented benchmark that tests many of the key features offered by ORDBMSs and evaluates the maturity of O-R technology relative to relational technology in the same ORDBMSs [4]. It presents an O-R schema and a relational equivalent thereof. The outcome of running BUCKY on an early O-R product shows that the performance of the O-R version of the examined ORDBMSs was almost twice as slow as the relational version.

One key factor to the success of ORDBMSs is its performance. The concept of ORDBMSs, as a hybrid of relational and object-oriented database management system is very appealing, preserving the wealth of knowledge and experience that has been acquired by RDBMSs [13]. Users expect short time responses for their database operations, even though the complexity of data and queries. Stonebraker claims that ORDBMSs will be the next great wave in evolution journey of DBMSs [12].

Database migration is the process of converting a source database into a target database to be handled in its new environment. Comparing the performance of source and target databases may help the user to decide whether or not they should move into their chosen database if performance is a deciding factor [14].

The aim of this paper is to investigate the maturity of O-R technology relative to relational technology. This includes the evaluation of the performance of ORDBMSs relative to RDBMSs by applying an O-R query-oriented BUCKY benchmark on Oracle 10g database management system (DBMS). The results from this study and the original BUCKY results are compared and discussed.

The remainder of this paper is organized as follows. In Section 2, an overview of related work is given. The BUCKY benchmark is briefly described in Section 3. While Section 4 explains experimental set-up, the next section describes the first results obtained from applying the queries of the benchmark on Oracle 10g DBMS. In Section 6, we discuss the results as whole and present lessons we have learned from this study. Section 7 concludes this paper.

2. Related Work

A set of query-based benchmarks has been designed to test and measure different aspects of the functionality and performance of object-based systems. The Simple Database Operations benchmark [11], the Object
Operations Version 1 (OO1) [5] and OO7 [2, 3] benchmarks were created to evaluate the performance of OODBMSs. The OO7 represents a comprehensive test of the wide range of OO features of OODBMS performance [2]. OO7 includes three sets of operations: traversals, queries and structural modifications. BUCKY [4] and BORD [8] benchmarks are query-oriented benchmarks for ORDBMSs. They emphasize the features of O-R databases that are not covered by pure relational databases. BUCKY is a query-oriented benchmark, which has been developed to test the maturity of an ORDB system's key features in relation to an RDB system. BORD is a generic benchmark for ORDBMSs, not intended for RDBMSs or OODBMSs. It features scalability, use of a synthesized database only, and a query-oriented evaluation. In total, thirty-six test queries were designed under fifteen categories, which include exact matches, object identifiers, joins, class references, set-valued attributes, user-defined methods, built-in versus user-defined operators, flattening queries, generic abstract data types (ADTs), and spatial ADTs.

3. BUCKY Benchmark

BUCKY is a query-oriented benchmark that was also designed in Computer Sciences Department, University of Wisconsin-Madison, to test the performance of O-R technology relative to relational technology [4]. It tests many of the key features of ORDBMSs, including row types and inheritance, references and path expressions, sets of atomic values and of references, methods and late binding, as well as user-defined abstract data types along with their methods. The objective in designing BUCKY has been to test the key features that add the ‘object’ notion to O-R database systems.

The benchmark consists of an O-R schema, a data generation program and a set of queries over this schema and an additional relational schema that is semantically equivalent to the O-R schema. One of the goals of this benchmark has been to examine the performance tradeoffs between the different designs alternatives offered by O-R database systems as the O-R technology expands the space of alternatives available to a schema designer. For example, the use of inheritance hierarchy against a set of independent tables, the use of set-valued attributes versus foreign key linking between tables. By implementing both versions of the benchmarks on one system, which is possible since by definition an O-R database supports full relational DDL and DML, the designers of the benchmark compare and contrast the two approaches within a common framework. The result of this comparison provides insight into the relative maturity of O-R versus relational query optimisers and runtime systems. By comparing the list of features tested in BUCKY and the list presented by Stonebraker [12], both lists contain inheritance. References, object methods and set-valued attributes from BUCKY, can be considered part of Stonebraker’s complex object support. Set-valued attributes contribute to Stonebraker’s base type extension. BUCKY differs on triggers or rule system with Stonebraker’s list. BUCKY includes no trigger test. Designers of BUCKY believe triggers are orthogonal to whether or not a system is O-R.

BUCKY is designed to provide a benchmark that is as specific as possible yet still captures the essence of what is missed by other existing benchmarks. Some key features are as follows:

- BUCKY only tests the object features in ORDBMS.
- Client-side pointer traversal, often the focus for OODBMSs, has been placed outside the scope of BUCKY.
- Domain-specific testing is also outside the scope of BUCKY.

The BUCKY schema was modeled after a simple and conventional university database application. It was implemented in Illustra, an early O-R product. The authors conclude that:

- Most queries are more naturally and concisely expressible using O-R model and SQL extension;
- Loading the O-R database was far more challenging than loading an information equivalent relational database;
- Inverse relationship support would have helped loading the BUCKY database;
- New SQL language features that O-R systems offer provide new implementation challenges for implementers of DBMS engines;
- A number of BUCKY queries ran faster on the relational version, especially those involving sets.

4. Experimental Set-up

To analyze the promises made by O-R database systems and explore the maturity of O-R in relation to relational technology, BUCKY benchmark has been implemented on Oracle 10g. The benchmark consists of two semantically equivalent relational and O-R schemas, as well as a set of queries over these schemas. In this section, BUCKY’s design rationale and schema descriptions are presented, and the steps taken to implement the schemas and the test-bed configuration are discussed.

4.1. Design Rationale

BUCKY benchmark defines a number of queries, which test the key features offered by ORDBMSs. By appropriately mapping and implementing the semantically equivalent relational and O-R queries on one system, which is possible since by definition an ORDBMS supports full relational DDL and DML, we can compare and contrast the results of the queries in order to achieve insight into relative maturity of O-R runtime systems and query optimisers. The rationale for choosing these areas has been to focus on main primitive differences between O-R and relational DBMSs. The benchmark avoids testing the functionality that is shared between the two DBMSs.
The shared functionality is tested by existing relational benchmarks.

4.2. Database Description

BUCKY schema is based on a university database (UniDB) application, which has been designed to run on an O-R and relational systems. Figure 1 shows the conceptual schema of the UniDB in UML class diagram notation. The solid directed lines with a blank arrowhead represent generalization relationships from more specialized classes, e.g., Student and Employee to more generalized classes, e.g., Person. In other words, Student and Employee are sub-classes of the super-class Person; Staff and Instructor are sub-classes of Employee; Professor is a sub-class of Instructor, and TA (teaching assistant) is a sub-class of both Student and Instructor. These generalizations form an inheritance hierarchy. The other lines represent associations between objects of the classes and are labeled on each side with a role name by which the association is known at that end and with multiplicities at the other end. For example, an object of the Department class must (i.e., 1..1, one and only one) have a chair who is in turn an object of the Professor class; inversely, an object of the Professor class may (i.e., 0..1, none or only one) lead a department. A multiplicity of 1..* means at least one and possibly many (e.g., a department can have one or many employees because 1..* is at the Employee end of the line and an employee works in one and only one department because 1..1 is shown at the Department end of the line).

4.3. Relational Implementation

The basic relations in the relational model of UniDB are Department, Student, Staff, TA, Professor, Course, CourseSection, Enrolled and Kids. Since there is no direct way to model inheritance in relational model, a separate relation is created for each non-abstract type in the hierarchy, i.e., Student, TA, Professor and Staff. The common attributes are repeated in each relation definition. There are several alternative ways to model inheritance in relational data model [7]. An alternative approach is to create a relation with all common attributes and build corresponding relations for each sub-type with attributes unique to that sub-type. The downsides are that the former alternative approach wastes space and the latter requires too many inter-entity joins. Because the relational model does not support relationships, the inter-entity relationships are modelled using primary and foreign keys. Also given that relational model lacks the multi-valued attributes, these are modelled by creating an additional relation called Kids, which has an id attribute corresponds to TA, Professor or Staff relations.

4.4. Object-Relational Implementation

In ORDB implementation, the UniDB is modelled in an inheritance hierarchy with a Person type as its root. The DDL description of the O-R schema is provided in Appendix A.

4.4.1. Extensible Base Data Types

Oracle supports ADT by providing ‘object types’. Object types are user-defined data types that make it possible to model real-world entities. For example instead of breaking up a location into unrelated attributes such as address, postcode, town etc in relational entities, the O-R approach defines a user defined type to represent a location. The following statements define location_adt, which represents a location data type:

```sql
create or replace type body location_adt as
  object (apptfrac varchar2(4), place varchar2(100),
           getlatitude member function return integer,
           getlongitude member function return integer,
           ndatediff member function return integer)
end
```

End

To model the one-to-many relationships, a number of collection data types are defined. For example, the O-R schema makes use of collection data type in form of nested table to maintain kid names along with their parent. The kidname_typ type and kidsnamesnt_typ nested table object type have been created to hold a list of kid names within each employee object.

4.4.2. Type Inheritance Hierarchy

The UniDB is modelled by having an inheritance hierarchy with Person as the root type, named person_typ, and contains all the common attributes of all university people.

```sql
create type person_typ as object (
  id integer, name varchar2(20), street varchar2(20), city varchar2(10), state varchar2(20), zipcode varchar2(6),
  birthdate date, kidnames kidnamesnt_typ, picture varchar2(100), place location_adt) not final
```
Since Oracle only supports single inheritance, a subtype can directly inherits only from a single super-type and not multiple super-types. The immediate children of Person object are student_typ and employee_typ. Staff and Instructor objects inherit from Employee objects. Instructor object has two subtypes, TA and Professor. Although in BUCKY schema, TA inherits from Instructor and Student objects. Since Oracle supports only single inheritance, TA type is defined to be a subtype of Instructor type.

4.4.3. Methods

Methods are functions that are declared in an object type definition to implement behavior of an object. Location object type has two methods that extract latitude and longitude. A third method calculates the distance between two locations. Person type and its subtypes have a Salary method. The Salary() method in each subtype overrides the corresponding method in their super-types to perform type-specific calculations in the subtype. The methods have been implemented in PL/SQL.

4.4.4. Other Object Types

BUCKY’s objects are mutually dependant, that is some types depend upon each other for their definitions. REFS and collections of REFS model associations among objects, particularly many-to-one relationships, thus reducing the need for foreign keys. Connections among mutually dependent types form a ring. For example, an employee works in a department. Therefore, Department object type has to be created before Employee type. However, Department type depends on Student type, which depends on Professor type. Incomplete declarations suggested by Oracle are used to serve as a placeholder for object types to break the REF rings. Collection types, as nested tables, are used to model multi-valued attributes.

4.4.5. Object Tables

In O-R schema, there are no instances of the non-leaf super-types. Only the leaf types actually have instances. For example, every instructor will either be a teaching assistant or a professor. In addition to these types, each of them has a corresponding table to hold its instances (i.e., Person_typ corresponds to Person table). These tables are contained in a table/sub-table hierarchy that mirrors the type/sub-type hierarchy. However, Oracle does not support the concept of table/sub-table hierarchy. Because of this restriction, it is necessary to alter the O-R schema. One approach is to create a corresponding table for each leaf type, i.e., to create Student, Staff, Professor and TA tables. An alternative approach is to create a corresponding table to the root type, i.e., only to create Person table.

4.5. Generating and Uploading the Data

BUCKY provides C++ programs to generate the data for both schemas. The programs generate two sets of information-equivalent data for RBD and ORDB, by adjusting a predefined variable in the code. After generating the data files, they can be bulk-loaded into the databases. Since originally BUCKY was implemented on a different platform, it is not possible to use the provided platform specific data sets. However, we used SQL loader and stored procedures to populate both schemas. Moreover, we proposed a solution for automatically migrating RDBs into ORDBs [10]. The paper that described this solution explains in further detail how to extract, transform and load an existing RDB into the formats suitable for Oracle ORDB systems. The process is performed in two passes. In the first pass, objects are defined in script files to instantiate ORDB tables with literal data. The second pass defines relationships among objects created in the first pass.

4.6. Test Bed Configuration

In this study, BUCKY is implemented in Oracle 10g on a standalone PC with 1.70 GHz processor and 512 MB of RAM under Windows XP. 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 are run with the buffer pool being empty. To flush the database buffer pool before running any of the queries, an enormous table that is not used in the benchmark queries is scanned. Both relational and O-R database schemas are created in two separate tablespaces 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. Since Oracle cannot index columns with data-type REF, indexes such as major on TA and Student tables have been omitted in both relational and O-R schemas.

5. Preliminary Experimental Results

This section presents the benchmark queries, what each query is intended to test, and results of running the benchmark on two different O-R versions and the relational version of the benchmark on the same DBMS, i.e., Oracle 10g. The first and second O-R approaches are referred to as O-R1 and O-R2 respectively. Because of lack of space; we offer 10 out of the 16 queries of BUCKY benchmark. Full description of all queries can be found in [9].

This query test provides a performance baseline that

- Exact Match over One Table (SINGLE_EXACT)
  - Query: Find the address of the staff member with id 6966.
  - O-R1: select name, street, city, state, zipcode from staff where id=6966;
  - O-R2: select s.name, s.street, s.city, s.state, s.zipcode from person s where value(s) is of (staff_typ) and s.id=6966;
  - RDB: select name, street, city, state, zipcode from staff where id=6966;
can be used to interpret the results of other queries. The relational and O-R1 results are similar as both tables that have the same number of tuples and are indexed on ID. The O-R2 is realistically slower. The cardinality of Person table with 125000 records is five times higher than that of Staff tables in the relational and the O-R1 schemas, which has only 25000 records.

**Exact Match on Table Hierarchy (HIER.EXACT)**
- **Query:** Find the address of the employee with id 6966.
  - **O-R1:**
    - select name, street, city, state, zipcode from staff where id=6966;
    - select name, street, city, state, zipcode from professor where id=6966;
    - select name, street, city, state, zipcode from ta where id=6966;
  - **O-R2:**
    - select s.name, s.street, s.city, s.state, s.zipcode from person s where value(s) is of (employee_typ) and s.id=6966;
    - select name, street, city, state, zipcode from ta where id=6966;
  - **RDB:**
    - select name, street, city, state, zipcode from staff where id=6966;
    - select name, street, city, state, zipcode from professor where id=6966;
    - select name, street, city, state, zipcode from ta where id=6966;

This query tests the efficiency of the system in handling queries over inheritance hierarchy. All queries perform very closely. Query O-R2 is more natural, concise and simple than the other two queries, and its schema has been very efficient. The cost of UNIONs in the O-R1 has slowed the query.

**Method Query over One Table (SINGLE-METH)**
- **Query:** Find all Professors who make more than 150000 per year.
  - **O-R1:**
    - select s.name, s.street, s.city, s.state, s.zipcode from professor s where s.salary()>150000;
  - **O-R2:**
    - select s.name, s.street, s.city, s.state, s.zipcode from person s where value(s) is of (employee_typ) and s.id=6966;
    - select name, street, city, state, zipcode from ta where id=6966;
  - **RDB:**
    - select name, street, city, state, zipcode from staff where id=6966;
    - select name, street, city, state, zipcode from professor where id=6966;
    - select name, street, city, state, zipcode from ta where id=6966;

This query tests the efficiency of the O-R system’s approach in indexing on function results as compared to indexing on stored relational attributes. To speed up the execution time, the nested table is indexed on object identifier and name attribute. Indexing improves the performance and elapsed time is dropped from 13.5 minutes to just 7.00s. At the first attempt, the cost of running the query in the O-R1 system is enormously high. The fact that the nested tables are scanned is confirmed by additional test queries. Creating indexes on the nested tables improves the performance considerably. Query O-R2 has done relatively well considering the size of Person table and that it consists of the overhead of overriding the salary method whilst the method is not indexed. Since Salary() is a method defined on subtypes of Person root type, it is not possible to index Person table on salary method. In theory the O-R system should be able to execute this query by doing an index lookup on the professor salary method. The test does not prove if there is anything to gain by indexing on methods.

**Relational Join Query (SINGLE JOIN)**
- **Query:** Find all Staff with the same birth date that live in an area with the same zip code.
  - **O-R1:**
    - select s1.zipcode, s1.birthdate, s1.id, s1.name, s1.street, s1.city, s1.state, s2.id, s2.name, s2.street, s2.city, s2.state from staff s1, staff s2 where s1.id<s2.id and s1.birthdate=s2.birthdate and s1.zipcode=s2.zipcode;
  - **O-R2:**
    - select s1.zipcode, s1.birthdate, s1.id, s1.name, s1.city, s2.id, s2.name, s2.city from person s1, person s2 where value(s1) is (staff_typ) and value(s2) is (staff_typ) and s1.id<s2.id and value(s1).birthdate=value(s2).birthdate and value(s1).zipcode=value(s2).zipcode;
  - **RDB:**
    - select s1.zipcode, s1.birthdate, s1.id, s1.name, s2.id, s2.name, s2.city, s2.state from staff s1, staff s2 where s1.id<s2.id and s1.birthdate=s2.birthdate and s1.zipcode=s2.zipcode;

This query is the baseline test for join processing. The relational and O-R1 schemas are very close, verifying the O-R query is just as efficient as the relational query for regular joins. O-R2 system has responded slower. The cost of the hash joins and Person table full scan is over three times the cost of the other two schemas.

**Set Membership (SET-ELEMENT)**
- **Query:** Find all Staff who have a child named ‘girl16’.
  - **O-R1:**
    - select s.id, s.name, s.street, s.city, s.state from staff s, table(s.kidnames) k where ‘girl16’ in k.kidname;
  - **O-R2:**
    - select s.id, s.name, s.street, s.city, s.state from person s, table(s.kidnames) k where value(s) is (staff_typ) and ‘girl16’ in k.kidname;
  - **RDB:**
    - select s.id, s.name, s.street, s.city, s.state from staff s, kids k where s.id=k.id and k.kidname=’girl16’;

This query tests the O-R system’s handling on nested tables/sets. The relational query the SQL involves a join with Kids table. O-R1 has outperformed the relational query, which verifies that the O-R system is capable in handling of nested tables.

**Single-Hop Path, No Selection (1HOP.NONE)**
- **Query:** Find all student/major pairs.
  - **O-R1:**
    - select s.id, s.name, s.city, s.state, s.major.dno, s.major.name from student s, major.dno, major.name from person s where value(s) is (student_typ) and s.major.dno, s.major.name from ta where ta.major.dno, ta.major.name from person s where value(s) is (ta_typ);
  - **O-R2:**
    - select s.id, s.name, s.city, s.state, treat(value(s) as student_typ).major.name, treat(value(s) as student_typ).major.dno, treat(value(s) as ta_typ).major.name, treat(value(s) as ta_typ).major.dno from person s where value(s) is (ta_typ);
  - **RDB:**
    - select s1.id, s1.name, s1.city, s1.state, d.deptno, d.name dname from student s, department d where s.majordept=d.deptno and d.deptname in value(s) is (student_typ) and s.majordept=d.deptno and d.deptname in value(s) is (ta_typ);

This query tests the efficiency of the O-R system at processing queries that involve path expressions. Since Oracle does not support multiple inheritance and given that TA is both a student and an employee, the O-R
queries take a relational format using UNION operator. However, as Oracle supports scoped references, e.g., the system knows that student.major.dno points to an object of Department type. O-R queries have done well. Similar to relational query join, the O-R system with scoped reference would write the query path as an explicit Object-Identifier (OID) join between two tables, e.g., Student.major=Department.OID. By comparing the syntax of query variants, it is clear the structure of the SELECT statement affects the performance. It is evident that O-R path expressions measure up to relational joins processing.

- **Two-Hop Path, Many-Side Selection (2HOP-ONE)**

  Query:  
  `Find the semester, enrolment limit, department number, and department name for all sections of courses taught in room 69.`

  **O-R1:**  
  Variant A: select s.semester, s.nostudents, s.course.dept.dno, s.course.dept.name from coursesection s where s.roomno=69;
  Variant B: select sec.coursesection_ref.semester, sec.coursesection_ref.nostudents, d.dno, d.name from department d, table(d.courseoffered) c, sec.coursesection_ref.nostudents, d.dno, d.name where sec.coursesection_ref.roomno=69;

  **O-R2:**  
  Variant A: select s.semester, s.nostudents, s.course.dept.dno, s.course.dept.name from coursesection s where s.roomno=69;
  Variant B: select sec.coursesection_ref.semester, sec.coursesection_ref.nostudents, d.dno, d.name from department d, table(d.courseoffered) c, table(c.courses_ref_sections) sec where sec.coursesection_ref.roomno=69;

  **RDB:**
  select sec.coursesection_ref.semester, sec.coursesection_ref.nostudents, d.dno, d.name, s.roomno, s.courseno from coursesection s, department d where ((s.deptno = d.deptno) and (s.roomno = 69));

This query tests the O-R system’s handling of path queries with longer paths. Object tables involved in this query are very much the same in the two O-R schemas. Consequently, the O-R times are very close. Variant A query is as efficient as the relational query. The performance of the O-R variant B query with selection on the target of a two-hop chain of set-valued references is very poor.

- **Simple ADT Function (ADT-SIMPLE)**

  Query:  
  `Find the latitudes of all staff members.`

  **O-R1:**
  select s.place.getlatitude() from staff s;
  **O-R2:**
  select treat(ref(s) as ref staff_typ).place.getlatitude() from person s, staff s where value(s) is of (staff_typ);  
  **RDB:**
  select s.place.getlatitude() = 34 and s.place.getlatitude() = 35;

This query tests the efficiency of the O-R system’s function dispatch mechanism (versus the efficiency of retrieving stored data). The method body is very simple. The results are comparable with a slight overhead for method invocation in O-R schemas.

- **Complex ADT Function (ADT-COMPLEX)**

  Query:  
  `For each Staff member, find the distance between him and the staff member with id 6966.`

  **O-R1:**
  select s.place.distance( s2.place ) from staff s, staff s2 where s.id=6966;
  **O-R2:**
  select treat(ref(s) as ref staff_typ).place.distance (treat(ref(s2) as ref staff_typ).place) from person s, person s2 where value(s) is of (staff_typ) and value(s2) is of (staff_typ) and s.id=6966;
  **RDB:**
  select sqrt(power(s1.latitude - s2.latitude, 2) + power(s1.longitude - s2.longitude, 2)) from staff s1, staff s2 where s1.id=6966;

This query tests the O-R system’s function dispatch mechanism, but this time it does so versus a case where the relational case’s expression is quite complex. The O-R and relational times are essentially the same. Again, there is slight overhead caused by method invocation in O-R schemas.

- **Exact-Match on Simple ADT Function (ADT-SIMPLE-EXACT)**

  Query:  
  `Find the ids of the Staff who live at latitude of 34 and a longitude of 35.`

  **O-R1:**
  select id from staff s where s.place.getlatitude() = 34 and s.place.getlongitude() = 35;
  **O-R2:**
  select id from person s where value(s) is of (staff_typ) and s.place.getlatitude() = 34 and s.place.getlongitude() = 35;
  **RDB:**
  select id from staff where latitude=34 and longitude=35;

This query tests the O-R system’s efficiency at handling an exact match query involving an ADT, which requires ADT indexing support. Although comparable, the slight slowness of O-R1 in relation to relational case can be due to the overhead caused by method invocation. As expected, O-R2 query is slower. The estimated cost is higher than the other two queries.

6. **Discussions and Lessons**

This section describes the results of running the BUCKY benchmark on two O-R implementations and a relational of the benchmark in Oracle and discussed the lessons learned. The elapsed times from running the benchmark on the schemas are shown in Table 1.

By comparing the results, it can be concluded that:

- In the single and hierarchical match queries, relational and O-R systems perform almost in the same way;
- In the single and hierarchical method queries the results are comparable. However, if the nested tables are not indexed, the O-R performance is rather poor. It seems that invoking the object methods scans the nested-table. As it is very unlikely that users are aware of the need for such indexes, they might not index nested-tables and experience a very poor performance;
- The cost of self joins on big tables like Person table in O-R2 is very high;
- O-R set-value attributes win over relational joins;
- In general, O-R system has proved to be efficient in queries with path expressions and it benefits from readable and concise SELECT statements;
O-R1’s method dispatch mechanism is close to retrieving of stored data in relational case. The cost of ADT method invocation has made O-R queries slightly slower. An issue that is worth pointing out is the flexibility of the DBMSs in applying changes to the O-R schemas. There is a tremendous value in being able to make amendments or answer questions at run-time that were not anticipated or overlooked at design time. On a number of occasions, applying changes to the user-defined types was unfeasible and consequently had to start again by rebuilding the types and re-populating the tables. Since it is fairly effortless to make changes to a relational schema, type evolution or changing the user defined types is a challenge in O-R schemas. It is clear that Oracle 10g has taken a huge step forward in supporting O-R features. However, despite supporting inheritance from super-type to subtype, Oracle 10g does not support table inheritance hierarchy. Multiple inheritance, which is a feature of a good ORDBMS, is not supported in Oracle 10g either. These shortcomings reduce the flexibility of Oracle in design and implementation of the benchmark’s O-R schema. BUCKY defines two bottom-line metrics. The O-R efficiency index and the O-R power rating. The metrics measure the relative performance of the system’s O-R against relational functionality and the absolute performance of the system’s O-R functionality respectively. The O-R power rating is useful only when comparing two O-R systems. Because each query addresses a different aspect of an O-R system, the idea of boiling down at entire benchmark down to a single number is not decisive. Consequently, to measure the relative performance of the system’s O-R and relational functionality, the ratio of each O-R query elapsed time to the corresponding relational query elapsed time has been calculated individually. If O-R RELATIONAL ratio is 1, it means both systems perform evenly. If the ratio is less than 1, it means O-R performs better. For ratios greater than 1, the relational outperforms O-R system.

Table 1 presents the results that were published in the BUCKY paper [4] and those of the tests carried out in this study. The O-R first and second approaches are referred to as O-R1 and O-R2 respectively.

In SINGLE-EXACT and SINGLE-JOIN queries Illustra and Oracle O-R1 perform very much the same. It was expected that SINGLE-METH query in Oracle to be as efficient as in Illustra, however, the actual results are different. In Illustra the O-R schema is able to run the query by doing an index lookup on employee salary method, whereas the complexity of the query predicate in the relational case forces a query plan that involves a sequential scan. Oracle scans the nested-tables, which results in poor performance.

The reason that in method invocation queries, nested tables are being scanned is not clear. Indexing nested-tables improves the performance significantly. However, developers unaware of query optimisers’ internal processes may not create such indexes, experiencing poor performance or query failures. O-R2 has not been successful in SINGLE-METH query. The optimiser performs a costly full scan on Person table.

Although O-R HEIR-EXACT and HEIR-METH queries are slower than the corresponding relational times in Oracle but they are more efficient than equivalent queries in Illustra. O-R1 is rather slow due...
to the union of the object tables while O-R2 provides a more natural way of expressing the queries. In Oracle, the structure of SELECT statements and the indexing of nested tables affect the query execution time. It is clear that the database design and implementation methodology have a direct affect on the system performance.

Oracle’s performance in HEIR-JOIN is rather poor. O-R times are slower than corresponding relational times. The structure of the SELECT statement in O-R2 is simpler, more concise and readable.

In handling SET-ELEMENT and SET-AND queries, O-R1 is faster than relational schema. Considering the size of Person table, the second O-R system has done quite well. This time Oracle has been significantly more efficient than Illustra.

In 1HOP-NONE the O-R times are more or less the same as relational times. Although Oracle supports scoped references but since it does not support multiple inheritance, in this study the queries were written using the UNION operator. By comparing the relative performance of O-R systems to relational, it is evident that the path queries being more concise are as efficient as joins in relational queries.

By executing 1HOP-ONE query, O-R systems’ scoped short path expression query beats relational joins. Comparing the relative performance of the two systems show that path query execution has been improved.

In general, Oracle shows progress in handling 1HOP-MANY query. In O-R2, the schema design and the size of Person table where various O-R predicates and functions, such as REF and TREAT, slow down this query.

In 2HOP-ONE, the last query of the path expression group, Illustra performs better. In Oracle, the selection on the source of a two-hop-chain of scalar-valued references is as efficient as relational joins. However, Oracle is not efficient in handling queries with selection on the target of a two-hop chain of set-valued references.

In the last group of queries, which test the use of ADTs and their methods, O-R times are almost comparable or slightly higher than relational times. The O-R queries, taking advantage of ADTs, are more simple and concise. In this group Illustra outperforms Oracle.

Wrapping up we can see that Oracle has been more efficient in handling queries over table hierarchy and set element queries. Oracle suffered from failures in one occurrence, questioning the reliability of the ORDBMSs. The object tables in O-R1 are treated as relational tables in various queries where the scanning of their nested tables results in additional costs. The schema design in O-R2 is more inclined toward object-oriented design with structural features such as inheritance and polymorphism. Unlike Illustra, Oracle does not support data or table inheritance or in another words, the use of UNDER keyword in CREATE TABLE statement. The single big Person table in O-R2 holds the inheritance hierarchy, benefiting from more concise and naturally defined queries. In O-R2 a number of functions and predicates, such as REF, VALUE and TREAT, are used to treat a super-type instance as a subtype instance. These functions increase the cost of queries in O-R2. Also, as examined in 1HOP-MANY query, how and where the O-R functions are being invoked affects the performance.

At the end, the O-R efficiency index is calculated for both DBMSs. The calculations are based on the reported scoped times in Illustra and the path expression times in Oracle. Compared with the corresponding relational versions, O-R1 proves to be more efficient in Oracle than the O-R schema in Illustra. O-R2, which is based on Oracle’s proposed O-R design methodology, is not efficient as the other two O-R schemas.

7. Conclusion

In this paper, BUCKY benchmark was applied on Oracle 10g. The outcome was discussed and evaluated against the results presented in the original BUCKY tests. This study has thrown light on the functionality of ORDBs, which made improvements but some limitations were also identified. In general, the performance of relational supersedes that of ORDBMSs. In most cases relational queries outperform their corresponding O-R queries. However, O-R version is more efficient in inheritance hierarchy and set-element queries. There is no doubt that ORDBMSs extend the functionality of RDBMSs. The O-R queries are expressed more naturally and concisely than their relational counterparts. The additional O-O features such as inheritance and polymorphism open up a new field of advanced applications for O-R database systems.

The flexibility of a database system in applying amendments is another factor in measuring the maturity of the system and is probably more valuable than performance over the long run [1]. Oracle is not entirely flexible when it comes to altering user-defined types and methods because of type dependency.

It appears that performance is affected by the design methodologies and features offered by various ORDBMSs. The lack of a single O-R model affects the design and organization of data in various ORDBMSs. Schema designs could become DBMS specific. One design might not suit or work in another ORDBMS. Due to the lack of support for table inheritance in Oracle, two O-R schemas were implemented. The performance of the second approach that is based on Oracle’s design methodology was poor with the highest O-R efficiency index. There is a need for a single O-R design framework and development methodology.
The database used in BUCKY benchmark was rather simple and conventional; therefore the ORDBMS suitability for advanced database applications in terms of functionality and performance needs to be expressed.

References


Appendix A: DDL of O-R Schema

- **User-Defined Types**

  ```sql
  create or replace type location_adt as object (lat integer, lon integer, member function getlatitude return integer deterministic, member function getlongitude return integer deterministic, member function distance(loc in location_adt) return number);

  create type kidname_typ as object (name varchar2(10));
  create type kidnames_nt_typ as table of kidname_typ;

  create type person_typ as object (id integer, name varchar2(20), street varchar2(20), city varchar2(10), state varchar2(20), zipcode varchar2(6), birthdate date, kidnames kidnames_nt_typ, picture varchar2(100), place location_adt) not final;

  create or replace type employee_typ under person_typ (datehired date, status integer, worksin ref department_typ, member function salary return number deterministic) not final;

  create or replace type staff_typ under employee_typ (annualsalary integer, overriding member function salary return number deterministic) not final;

  create or replace type employee_typ under person_typ (datehired date, status integer, worksin ref department_typ, member function salary return number deterministic) not final;

  create or replace type instructor_typ under employee_typ (teaches coursesection typ, overriding member function salary return number deterministic) not final;

  create or replace type professor Typ under instructor Typ (advises student Typ, overriding member function salary return number deterministic) not final;

  create or replace type student Typ under person Typ (studentid integer, major ref department Typ, advisor ref professor Typ, hastaken enrolled Typ not final);

  create or replace type department Typ as object (dno integer, name varchar2(20), building varchar2(10), budget integer, coursesoffered course nt Typ, chair ref professor Typ, employees employee nt Typ, majors student nt Typ, place location adt);

  create or replace type course Typ as object (cno integer, name varchar2(20), dept ref department Typ, credit integer, sections coursesection nt Typ);

  create or replace type course Typ as object (cno integer, name varchar2(20), dept ref department Typ, credit integer, sections coursesection nt Typ);

  create or replace type professor Typ under instructor Typ (advises student Typ, overriding member function salary return number deterministic) not final;

  create or replace type department Typ as object (dno integer, name varchar2(20), building varchar2(10), budget integer, coursesoffered course nt Typ, chair ref professor Typ, employees employee nt Typ, majors student nt Typ, place location adt);

  create or replace type course Typ as object (cno integer, name varchar2(20), dept ref department Typ, credit integer, sections coursesection nt Typ, grade char(2));

  create or replace type department Typ as object(}
```
dno integer, name varchar2(20), building varchar2(10), budget
courseoffered course_nt_typ,
chair ref professor_typ, employees employee_nt_typ,
majors student_nt_typ, place location_adt);

create or replace type student_typ under person_typ (studentid integer, major ref department_typ, advisor ref professor_typ, hastaken enrolled_nt_typ);

create or replace type coursesection_typ as object (course ref course_typ, semester integer, textbook varchar2(20), nostudents integer, building varchar2(10), roomno integer, teacher ref instructor_typ, students enrolled_nt_typ);

- **O-R1 Tables**

  create table courses of course_typ
  nested table sections store as sections_ntab;

  create table coursesection of coursesection_typ
  nested table students store as section_students_ntab;

  create table department of department_typ
  nested table courseoffered store as courseoffered_ntab,
  nested table employees store as employees_ntab,
  nested table majors store as majors_ntab;

  create table enrolled of enrolled_typ;

  create table professor of professor_typ
  nested table kidnames store as professor_kidnames_nt,
  nested table teaches store as prof_teaches_nt,
  nested table advisees store as prof_advisees_nt;

  create table staff of staff_typ
  nested table kidnames store as staff_kidnames_nt;

  create table student of student_typ
  nested table kidnames store as student_kidnames_nt,
  nested table hastaken store as student_hastaken_nt;

  create table ta of ta_typ
  nested table kidnames store as ta_kidnames_nt,
  nested table teaches store as ta_teaches_nt,
  nested table hastaken store as ta_hastaken_nt;

- **O-R2 Tables**

  create table courses of course_typ
  nested table sections store as sections_ntab;

  create table coursesection of coursesection_typ
  nested table students store as section_students_ntab;

  create table department of department_typ
  nested table courseoffered store as courseoffered_ntab,
  nested table employees store as employees_ntab,
  nested table majors store as majors_ntab;

  create table enrolled of enrolled_typ;

  create table person of person_typ
  nested table kidnames store as kidnames_nt;