ORACLE
NOSQL DATABASE

Oracle NoSQL Database – Parent-Child Joins and Aggregation

ORACLE WHITE PAPER | MAY, 2018

ORACLE

Table of Contents

Introduction 1

Parent Table – Child Table Joins 2

- Comparison to RDBMS LEFT OUTER JOIN 3
- Example 3

Aggregates 7

- count(*) and count(any) 7
- sum(any) 7
- avg(any) 7
- min (any) or max (any) 7
- Examples 7
- Queries 7

Conclusion 8

Introduction

Oracle NoSQL Database tables can be organized in a parent/child hierarchy. There is no limit to how many child tables can be created, nor is there a limit to how deep the child table nesting can go. When retrieving data by default, child tables are not retrieved when querying a parent table, nor is the parent retrieved when you retrieve a child table. Each of these tables is treated individually and provides the application a method to maintain data relationships as part of the same hierarchy.

The parent/child table relationship in Oracle NoSQL Database is very similar to the parent/child table relationship in a classic relational database. The difference being that no join statements are required to query the data.. Because of the way Oracle NoSQL Database stores the parent/child table data, it is very efficient to traverse data maintained in this relationship. The entire child data set associated to its parent is guaranteed to reside on the same shard thus allowing the retrieval in a single network call.

Prior to Oracle NoSQL Database, Version 18.1, when retrieving from parent/child tables, the data in child tables was not retrieved when querying a parent table and the same holds true in reverse. Each of these tables was treated individually and required the application to handle the join(s) between them separately.

Aggregations enable basic numerical operations to be performed on column values. This allows the data to be used to conduct analysis. Prior to Oracle NoSQL Database, Version 18.1 such operations would have to done by the client application or via a batch map-reduce job.

Oracle NoSQL Database, Version 18.1 has introduced two very useful features for addressing these shortcomings.

- Taking the parent-child table relation to the next level, a special kind of join among tables that belong to the same table hierarchy (parent and children) was added. These joins can be executed efficiently, because only co-located rows may match with each other.
- Run aggregate operations such as sum, average, max etc. on the data.

Each of these features is detailed below. To help understand and relate to these two features, the use case in the example below depicts a Customer and his shopping activities on a shopping cart kind of application.

Parent Table – Child Table Joins

To query from tables that belong to the same hierarchy the NESTED TABLES clause must be used. The syntax is as follows:

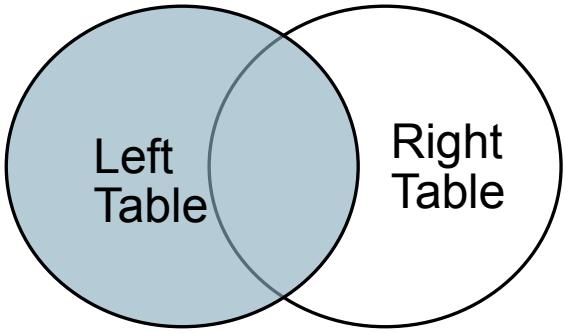
The NESTED TABLES clause specifies the participating tables and should be mentioned as 3 separate groups. The first group specifies the target table and is included within the parentheses after NESTED TABLES. The second group is the ANCESTORS, if present and is included within parentheses after the target table. The third group is the DESCENDANTS, if present and is included within parentheses after the ANCESTORS. There can be any number of ANCESTORS and DESCENDANTS and are comma separated. The ANCESTORS and DESCENDANTS are tables from which you would like to extract rows related to the target table.

The syntax is as follows:

```
Nested Tables: from
      NESTED TABLES
      ( target_table
        ANCESTORS (ancestor1, ancestor2…)
          DESCENDANTS (descendant1, descendant2,…)
       )
```

Comparison to RDBMS LEFT OUTER JOIN

The NESTED TABLES clause is similar to the RDBMS style LEFT OUTER JOIN using the sample syntax “Select columns from LeftTable LEFT JOIN RightTable on LeftTable.col1 = RightTable.col1". In the query that uses LEFT OUTER JOIN , the table to the left of the “LEFT JOIN” keyword is called the left table while to the right of the keyword is called as right table. The result of the query will include rows that are part of the left table matching with the rows in the right table. The rows which don’t have a corresponding row in the right table would return a NULL.



The above Venn diagram depicts the LEFT OUTER JOIN, with the shaded part showing what rows are returned. The intersection between two circles are rows that match in both tables, and the remaining part of the left circle are rows in the table that do not have any matching row in the right table. Hence, all rows in the left table are included in the result.

It should also be understood that the results are part of the WHERE clause and any other optional ON expressions used.

An important note for NoSQL:

- The query is executed as parallel scan operations on each shard, which means that the query performance is high.
- If the target table has indexed columns which are used as part of the predicate, then the index would be used in the query, resulting in very fast retrieval.

These are 2 different types of databases and cannot be used for data management for the same situations and requirements interchangeably.

A child table is represented with a “.” notation in Oracle NoSQL Database.

It is assumed that the reader knows to create the table in the respective databases.

Example

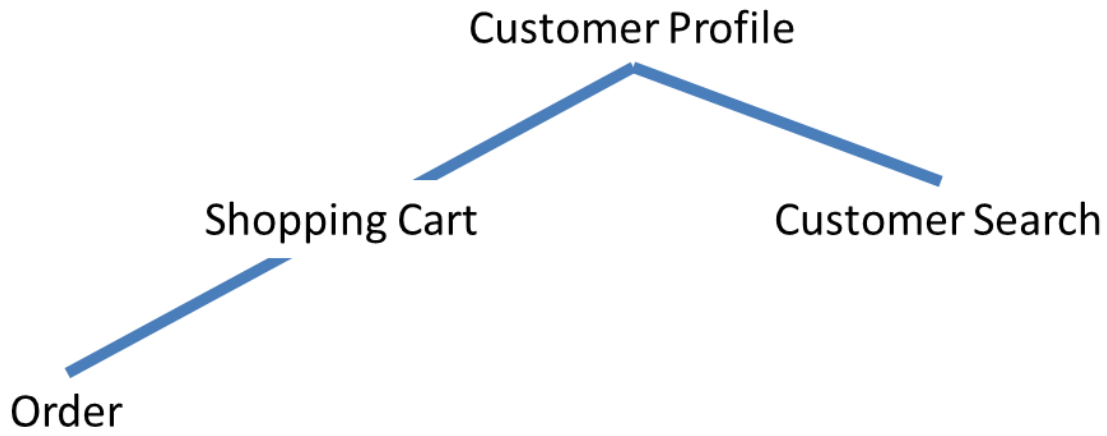
As part of the example we show the results from similar tables in Oracle NoSQL Database vs MySQL (RDBMS style) which demonstrate the power of this new feature.

Tables

MySQL – CustomerProfile, Cart, Order, Searched

Oracle NoSQL Database – customerProfile, customerProfile.cart, customerProfile.cart.order, customerProfile.searched

A graphical representation of the table hierarchy in Oracle NoSQL Database would look like



Based on this, the query for a simple to extract from the above tables would be:

-----All products that Mike has searched and placed an order for, along with the status of each order on the site-----

The table scripts, insert statements and queries for both MySQL and Oracle NoSQL Database are available as a zip file.

MySQL - The Query in MySQL database would look like:

```
select * from customer left outer join cart on
customer.id = cart.customer_id left outer join searched
on customer.id = searched.customer_id left outer join
sys.order on cart.cart_id=sys.order.cart_id;
```

The output looks like – for sake of brevity only a few are shown:

{"id" : 1,"name" : "Mike","lastname" : "Brey","phone_no" : "8888888888","cart_id" : 1001, “customer_id" : 1,"product_Info" : "{\"Date\": {\"ExpiryDate\": \"May 2019\", \"PurchaseDate\": \"May 2017\"}, \"Price\": 30, \"Quantity\": 1, \"ProductID\": 1101, \"Description\": \"Table\", \"ProductName\": \"Table\"}", "searched_id" : 10,”customer_id" : 1 ,"product_Info" : "{\"Date\": {\"ExpiryDate\": \"May 2019\", \"PurchaseDate\": \"May 2017\"}, \"Price\": 30, \"Quantity\": 1, \"ProductID\": 1101, \"Description\": \"Table\", \"ProductName\": \"Table\"}"," order_id " : 1101,"cart_id":1001,"status" : "paid"}

{"id" : 1,"name" : "Mike","lastname" : "Brey","phone_no" : "8888888888","cart_id" : 1002, "customer_id" : 1, "product_Info" : "{\"Date\": {\"ExpiryDate\": \"May 2019\", \"PurchaseDate\": \"June 2017\"}, \"Price\": 110, \"Quantity\": 1, \"ProductID\": 1102, \"Description\": \"Teak Table\", \"ProductName\": \"Teak Table\"}","searched_id ": 11,”customer_id" : 1 ,"product_Info" : "{\"Date\": {\"ExpiryDate\": \"Dec 2019\", \"PurchaseDate\": \"Jan 2017\"}, \"Price\": 1200, \"Quantity\": 1, \"ProductID\": 1201, \"Description\": \"TV\", \"ProductName\": \"TV\"}","order_id" : 1101,"cart_id":1001,"status" : "cod"}}

{ "id" : 1,"name" : "Mike","lastname" : "Brey","phone_no" : "8888888888","cart_id" : 1003, "customer_id" : 1,"product_Info" : "{\"Date\": {\"ExpiryDate\": \"Dec 2022\", \"PurchaseDate\": \"Jan 2017\"}, \"Price\": 350, \"Quantity\": 1, \"ProductID\": 1103, \"Description\": \"Dining Table\", \"ProductName\": \"Dining Table\"}"," searched_id": 12,"customer_id" : 1 ,"product_Info" : "{\"Date\": {\"ExpiryDate\": \"Jan 2020\", \"PurchaseDate\": \"Jan 2017\"}, \"Price\": 3400, \"Quantity\": 1, \"ProductID\": 1301, \"Description\": \"Refrigirator\", \"ProductName\": \"Refrigirator\"}"," order_id " : 1101,"cart_id":1001,"status" : "paid"}}

{"id" : 1,"name" : "Mike","lastname" : "Brey","phone_no" : "8888888888","cart_id" : 1004, "customer_id" : 1, "product_Info" : "{\"Date\": {\"ExpiryDate\": \"May 2019\", \"PurchaseDate\": \"June 2017\"}, \"Price\": 35, \"Quantity\": 6, \"ProductID\": 1103, \"Description\": \"Chairs\", \"ProductName\": \"Chairs\"}","searched_id ": 13,"customer_id" : 1 ,"product_Info" : "{\"Date\": {\"ExpiryDate\": \"Dec 2020\", \"PurchaseDate\": \"Sep 2017\"}, \"Price\": 1800, \"Quantity\": 1, \"ProductID\": 1401, \"Description\": \"Dish Washer\", \"ProductName\": \"Dish Washer\"}"," order_id " : 1101,"cart_id":1001,"status" : "paid"}}}

……

Oracle NoSQL Database - Similar query in Oracle NoSQL Database would look like –

```
select * from NESTED TABLES (customerProfile cp
DESCENDANTS (customerProfile.cart cc,
customerProfile.cart.order,customerProfile.searched))
```

The output would be – There are in all 13 rows returned, but for sake of brevity only a few are shown.

{"cp":{"id":1,"name":"Mike","lastname":"Brey","phone_no":"8888888888"},"cc":{"id":1,"cart_id":1001,"customer_id":1,"product_Info":{"Date":{"ExpiryDate":"May 2019","PurchaseDate":"May 2017"},"Description":"Table","Price":30,"ProductID":1101,"ProductName":"Table","Quantity":1}},"customerProfile_cart_order":{"id":1,"cart_id":1001,"co_id":1101,"status":"paid"},"customerProfile_searched":null}

{"cp":{"id":1,"name":"Mike","lastname":"Brey","phone_no":"8888888888"},"cc":{"id":1,"cart_id":1002,"customer_id":1,"product_Info":{"Date":{"ExpiryDate":"May 2019","PurchaseDate":"June 2017"},"Description":"Teak Table","Price":110,"ProductID":1102,"ProductName":"Teak Table","Quantity":1}},"customerProfile_cart_order":{"id":1,"cart_id":1002,"co_id":1102,"status":"cod"},"customerProfile_searched":null}

{"cp":{"id":1,"name":"Mike","lastname":"Brey","phone_no":"8888888888"},"cc":{"id":1,"cart_id":1003,"customer_id":1,"product_Info":{"Date":{"ExpiryDate":"Dec 2022","PurchaseDate":"Jan 2017"},"Description":"Dining Table","Price":350,"ProductID":1103,"ProductName":"Dining Table","Quantity":1}},"customerProfile_cart_order":{"id":1,"cart_id":1003,"co_id":1103,"status":"paid"},"customerProfile_searched":null}

{"cp":{"id":1,"name":"Mike","lastname":"Brey","phone_no":"8888888888"},"cc":{"id":1,"cart_id":1004,"customer_id":1,"product_Info":{"Date":{"ExpiryDate":"May 2019","PurchaseDate":"June 2017"},"Description":"Chairs","Price":35,"ProductID":1103,"ProductName":"Chairs","Quantity":6}},"customerProfile_cart_order":{"id":1,"cart_id":1004,"co_id":1104,"status":"cod"},"customerProfile_searched":null}

{"cp":{"id":1,"name":"Mike","lastname":"Brey","phone_no":"8888888888"},"cc":null,"customerProfile_cart_order":null,"customerProfile_searched":{"id":1,"cs_id":10,"product_info":{"Date":{"ExpiryDate":"May 2019","PurchaseDate":"May 2017"},"Description":"Table","Price":30,"ProductID":1101,"ProductName":"Table","Quantity":1}}}

{"cp":{"id":1,"name":"Mike","lastname":"Brey","phone_no":"8888888888"},"cc":null,"customerProfile_cart_order":null,"customerProfile_searched":{"id":1,"cs_id":11,"product_info":{"Date":{"ExpiryDate":"Jan 2020","PurchaseDate":"Jan 2017"},"Description":"Refrigirator","Price":3400,"ProductID":1201,"ProductName":"Refrigirator","Quantity":1}}}

{"cp":{"id":1,"name":"Mike","lastname":"Brey","phone_no":"8888888888"},"cc":null,"customerProfile_cart_order":null,"customerProfile_searched":{"id":1,"cs_id":12,"product_info":{"Date":{"ExpiryDate":"Dec 2019","PurchaseDate":"June 2017"},"Description":"Microwave Owen","Price":800,"ProductID":1301,"ProductName":"Microwave Owen","Quantity":1}}}

……

Aggregates

Aggregate functions in Oracle NoSQL Database evaluate an expression for each row, and aggregate the returned values into a single value. Syntactically, aggregate functions appear in the SELECT clause. If the SELECT clause contains any aggregate functions, the entire set of rows produced by the FROM or the WHERE clauses is considered and the aggregate functions are evaluated over this single group. The currently available aggregate functions are count, sum, avg, min, and max.

count(*) and count(any)

Returns the number of rows in a group or computes the input expression on each row in the group and returns a count of all the non-NULL values.

sum(any)

Computes the input expression on each row in the group and sums all the numeric values in the sequence. The resulting value type is based on the input values. If there is one input item of type integer then the result would be an integer, or if there is one input type of double or float then the return would be a double.

Note: There is a known Java error when trying to sum 2 floating point numbers if the return type is double.

avg(any)

Computes the input expression on each row in a group; sums all the numeric values in the sequence and counts all the numeric values. The resulting value is the average (division of sum by its count) and the type is based on the input values. If there is one input item of type integer then the result would be an integer, or if there one input type of double or float then the return value would be a double.

min (any) or max (any)

Computes the input expression on each row in a group and returns either the minimum or maximum value of the expression for the group.

Examples

We continue with the above shopping cart example and look for:

1. Total count of products that a particular customer has searched for.
2. Sum of the orders that a customer has done till date.
3. Highest amount purchase that a customer has done till date.
4. Lowest amount purchase that a customer has done till date.
5. Average spend for a customer

Queries

1. In the customerProfile table – Mike has id 1 and is referenced with the id in customerProfile.searched table. To query for the count of how many searches are done by Mike the query would be

```
sql -> select count(*) from customerProfile.searched where
id=1;
{"column_1":4}

1 row returned
```

2. In the customerProfile table – Mike has id 1 and is referenced with the id in the customerProfile.cart table. To query for the total monies spent by Mike for his purchases the query would be:

```
sql -> Select sum(cc.product_info.Price) from
customerProfile.cart cc where id=1;
{"column_1":525}

1 row returned
```

3. In the customerProfile table – Mike has id 1 and is referenced with the id in the customerProfile.cart table. To query for the costliest purchase done by Mike the query would be:

```
sql -> Select max(cc.product_info.Price) from
customerProfile.cart cc where id=1;
{"column_1":350 }

1 row returned
```

4. In the customerProfile table – Mike has id 1 and is referenced with the id in the customerProfile.cart table. To query for the least expensive purchase done by Mike the query would be:

```
sql -> Select min(cc.product_info.Price) from
customerProfile.cart cc where id=1;
{"column_1":30}

1 row returned
```

5. In the customerProfile table – Mike has id 1 and is referenced with the id in the customerProfile.cart table. To query for the average spend by Mike the query would be:

```
sql -> Select avg(cc.product_info.Price) from
customerProfile.cart cc where id=1;
{"column_1":131.25}

1 row returned
```

Conclusion

Oracle NoSQL Database with these enhanced query features make it easy to retrieve related data stored in the same hierarchy of tables represented as Parent-Child tables. With the inclusion of aggregates to the query capabilities, retrieving meaningful data from your customer's stored data has also become very easy.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Integrated Cloud Applications & Platform Services

Oracle NoSQL Database 18.1 new Features – Parent Child and Aggregation
May, 2018

Author: Vishal Settipalli
Contributing Authors: Michael Schulman, Michael Brey, Tim Goh