README - SDB Demo Application & Monitoring tool

The following demo application is designed for a sharded database and simulates the workload of an online retail store. It can be used to validate the setup of any system managed (automatic sharding) database configuration. It is also provides a practical example of sharding concepts for administrators and developers new to database sharding.

The demo application assumes that a system managed sharded database environment has already been created along with CUSTOMER table-family. The environment may have any number of chunks and shards (database nodes). When run, the application will first populate the products table and then start a one-hour workload that may be paused at any time by the administrator. The workload includes four types of transactions: create a customer order, lookup the list of orders, create a new product, and multi-shard query with report generation. All aspects of a sharded database configuration are exercised.

The demo application includes the following features:

- Universal Connection Pool with sharded environment (Application is validated only with the ucp.jar file that is included in the zip file)
- Cross-shard queries for report generation in sharded environment.
- Executes the following workload periodically:

Read write workload:

```
INSERT INTO Customers (CustId, FirstName, LastName, CustProfile, Class,
Geo, Passwd) VALUES (:p1, :p2, :p3, :p4, :p5, :p6, PASSWCREATE(:p7))

INSERT INTO Orders(CustId, OrderId, OrderDate, Status, SumTotal) VALUES
(:p1, Orders_Seq.NEXTVAL, :p2, 'PRE', :p3) RETURNING ORDERID INTO :p4

INSERT INTO LineItems(CustId, OrderId, ProductId, Price, Qty) VALUES
(:p1, :p2, :p3, :p4, :p5)
```

Read only workload:

```
SELECT CustId, FirstName, LastName, CustProfile FROM Customers WHERE
CustId=:p1 AND PasswCheck(:p2, Passwd) = 0

SELECT ProductId, LastPrice FROM Products SAMPLE (1) ORDER BY
DBMS_RANDOM.VALUE

SELECT OrderDate, OrderId, Status FROM Orders WHERE CustId = :p1
```

Cross-shard queries:

```
SELECT ProductId Id, P.Name, P.DescrUri,
            sum(L.Qty) totalQty, sum(L.Qty * L.Price) totalSum
      FROM Products P natural join
      LineItems L join Orders R
```

```
            on (R.CustId = L.CustId AND R.OrderId = L.OrderId)
      WHERE R.Status = 'SENT' GROUP BY ProductId, P.Name, P.DescrUri
ORDER BY totalQty Desc


SELECT C.FirstName, C.LastName, CustId, count(R.OrderId),
            sum(R.SumTotal) totalSum
      FROM CUSTOMERS C NATURAL JOIN ORDERS R
      WHERE R.Status = 'SENT'
      GROUP BY CustId, C.FirstName, C.LastName ORDER BY totalSum Desc
```

Populating the Products Duplicated table:

```
INSERT INTO Products(Name, DescrUri, LastPrice) VALUES (:p1, :p2, :p3)
```

The demo application also includes a monitoring tool. On the monitor dashboard, you will see the sharded database configuration and the workload across the shards. At the bottom of the dashboard there are samples from the Orders table (the last inserted orders) with shard assignment. In addition, there is a chart showing the distribution of Order table's rows among chunks. The monitoring tool is run in a separate terminal window to validate that workload is balanced across all shards

Setup and configure the Sharding Demo Application

Demo application assumes that the CUSTOMER table family has been created (refer to the Sharding Schema section in the Oracle Administrators Doc)

We then create additional objects needed by the demo app executing the demo_app_ext.sql (as shown here) on the shard catalog:

```
$ . ./shardcat.sh
$ cd sdb_demo_app/sql
$ sqlplus / as sysdba
SQL>@demo_app_ext.sql
```

To setup and run Demo app:

```
$ cd $HOME/sdb_demo_app
```

Edit the demo.properties and modify the HOST and PORT of the shard-director1.
In this lab we are using shard0 as the HOST for shard-director1 and it is using 1571 as the PORT.

```
$ ./run.sh demo
```

Setup and configure the Monitoring Tool

To install the Monitor app:

On a separate terminal:

```
$ . ./shardcat.sh
$ cd $HOME/sdb_demo_app
$ ./run.sh monitor
```

Open a new terminal and launch Firefox browser

```
$ firefox &
```

Enter the address: **shard0:8081**

Appendix: Information about the characteristics of the Demo Application

Data Generation

In the data subdirectory there is a set of files with input data, used for generating random data.
All the input data are from the public domain and can be downloaded freely.

* data/first-f.txt -- The set of the popular female first names
* data/first-m.txt -- The set of the popular male first names
* data/last.txt -- The set of the popular last names
* data/parts.txt -- Input data for product generation: list of the Wikipedia automotive part articles.
* data/streets.txt -- Input data for the address generation: list for US street names.
* data/us-places.txt -- Input data for the address generation: list of US zip codes.

Customer generation

We use email as a customer ID in our demo application. The random customer email generation is a key point of
our data insertion process. The process is implemented by `oracle.demo.CustomerGenerator` class.
First, we randomly select a customer gender. Then, the email is generated as:
[random first name].[random last name]@x.bogus.

At first, there are no customers with the given names. But the total set of possible emails is limited and at
some point generated names start to collide with each other. We treat that as the returning customer.

Workload

Demo application executes an open workload with 4 kinds of transactions.

1. **Create a customer order**. First, random customer name is generated. We connect to the shard, providing this

customer name as a key. Is the customer does not exist, we insert a new record into the `CUSTOMERS` table,
creating a new customer. Then, we add an order for that customer, randomly selecting a number of products from
the `PRODUCTS` duplicated table. Please check the
`oracle.demo.actions.CreateOrder` class for the implementation.

2. **Lookup the list of orders**. Again, we connect with a randomly generated customer name.
If the customer exists, we acquire the full history of his orders. If the customer does not exist, we
select random products from the `PRODUCTS` table. Please check the
`oracle.demo.actions.OrderLookup`
class for the implementation.

3. Once in while, we **create a new product**. Generate a random product name and insert a new product into
the `PRODUCTS` table. This query is executed on the catalog database. Please check `oracle.demo.actions.AddProducts`
class for implementation.

4. Execute **cross-shard query with report generation**. The report is the aggregation of sales (`LINE_ITEMS` table)
by products. This is executed on the catalog side. The report is then put into the `/tmp` directory.
Please see `oracle.demo.actions.GenerateReport` class.

Monitoring Tool

The monitoring tool provides an example of simple DIY-style sharding management tools. It uses the `dbms_global_views` package.

The purpose of this demo is to show that it is possible to monitor the status
of shards using only the catalog connection. In fact, the monitoring application does not make any connections to shards directly.

The `dbms_global_views` package creates a public `shard_dblinks_view` and a
public dblink to each shard.

To run the monitoring demo, execute:

```
./run.sh monitor.sh
```

Note, it is normal to see some errors about missing files, like `favicon`.

Screen Description

When the monitor is running it starts up the web server, which is by default
on `localhost:8081`. Currently it is only tested to work in the Firefox browser.
On the web page, you can find several demo widgets.

Shard Widgets

Each shard widget shows the shard name, its status, and the list of chunks,
which belong to that shard. It also shows the status of the chunk, by using
different colors: yellow-ish chunk means read/write state, while red chunk means
read-only state. The color transition can be noticed during chunk move.

Service Workload Widget

User Calls per Second (UCPS) value from the `gv$service_metric` view on each shard. The color of each line corresponds to the color, which is shown
in the corresponding Shard widget (at the top).

This is queried as:
```
select PUBLIC_DBNAME, 0, 0, sum(CALLSPERSEC)
  from SGV$SERVICEMETRIC
    where service_name like 'oltp%' and group_id=10 group by
PUBLIC_DBNAME
```

where `SGV$SERVICEMETRIC` is created by the `dbms_global_views` package as:

```
exec dbms_global_views.create_gv('SERVICEMETRIC');
```

where `create_gv` is a function which creates a catalog view, based on shards' `gv$` view.

Active Sessions

This chart shows the number of open **user** sessions to each of the databases.
Usually, even with the demo app running, this number is not very big due to
low number of default threads (4) in the demo app.

Order Row Distribution

This chart shows the distribution of orders across partitions, which corresponds to particular chunks. This is gathered querying the view `GLOBAL_DBA_TAB_PARTITIONS`, which is defined as an aggregation view:

```
exec dbms_global_views.create_dba_view('DBA_TAB_PARTITIONS');
```

Note, that this is an aggregation across all the partitions, so the total
number of rows from this char is twice as the actual number of rows (in

presence of
one standby per primary).

Order Table Sampling

This widget shows a random sample from recently inserted Order table rows.
The query is the aggregation of the `SAMPLE_ORDERS` view, which is defined
on __each shard__ as follows:

```
CREATE OR REPLACE VIEW SAMPLE_ORDERS AS
  SELECT OrderId, CustId, OrderDate, SumTotal FROM
    (SELECT * FROM ORDERS ORDER BY OrderId DESC)
      WHERE ROWNUM < 10;
```

Appendix: Troubleshooting of the Demo Application

After incremental deploy new shards are not represented in the monitor or monitor logs an exception

To manually update the monitored shard list,
execute the following line from sys as sysdba:

```
dbms_global_views.create_all_database_links();
```

Monitoring tool only shows four or less shards

The problem may be in `open_links` parameter on the catalog database.
You can check that by running `sqlplus / as sysdba` in the catalog environment
and executing:

```
show parameter open_links
```

The value of this parameter determines the maximum number of shards
we can connect to. Set it to some higher number:

```
alter system set open_links=64 scope=SPFILE;
```

Resetting this parameter (SPFILE only) requires catalog restart.