Hive-ODCI - Users Guide

======================

Hive-ODCI is an Oracle Data Cartridge Interface for dynamically accessing Hadoop/Hive data-stores through an Oracle 12c database. In other words Hive-ODCI makes Hadoop/Hive tables accessible as first-class, native, objects directly using PL/SQL, SQL, VIEWS, DML, DDL, etc.... in an Oracle 12c database.

Author

Metasystem Technologies Inc. (MTI) www.mtihq.com

Nicholas Van Wyen nvanwyen@mtihq.com

License

Releases

All releases can be found on Github https://github.com/nvanwyen/hive-odci/releases, along with the latest release release.

The project home is publicly available on Github at https://github.com/nvanwyen/hive-odci

Installation and Removal

See INSTALL.md for instructions

Concepts

Hive-ODCI is a pass-through interface allowing SQL access from within an Oracle RDBMS to information retained in an external Hive/Hadoop data-store. Hive-ODCI provides PL/SQL interfaces using ODCI to accomplish this functionality, making the access viable as native object in Oracle.

Hive-ODCI is accessed via the `HIVE` schema, and controlled through RBAC permissions in Oracle. A client (user or application) is granted privileges through the `HIVE_USER` role or direct system privileges by the DBA.

The client uses the PL/SQL objects to query, or execute DML/DDL in the remote Hive datastore, using the `HIVE_T` object type or one of the PL/SQL packages.

The `HIVE` schema installs Java classes in the database, which perform the JDBC execution on the clients behalf, and streams `PIPLINED` results back through the same interface.

The client can dictate most levels of functionality at run-time, with predefined session data, bind-variables, etc... as customization for each call.

Take for example the following concepts. The Hive/Hadoop data-store is remotely accessible on a separate server. The Hive-ODCI Java classes access the remote data via the JDBC Driver loaded during installation.

The client accesses the Hive-ODCI interface using the PL/SQL objects provided and/or a first-class `VIEW`, controlled by RBAC, for a 2-way avenue for data.

```
................................................................................
.                                                                              .
.                              +--------+       +--------+                     .
.                              |  Hive  |------>| Hadoop |                     .
.                              +--------+       +--------+                     .
.       server                      |                                          .
.       ----------------------------|-----------------------------------       .
.       jvm                         |     jdbc                       --+       .
.                              +--------+                              |       .
.                              |  hive  |                              |       .
.                              +--------+                              |       .
.                                   |                                  |       .
.       ----------------------------|--------------------------------- | d     .
.       pl/sql                      |                                  | a     .
.                                   |                                  | t     .
.              +---------+     +--------+     +---------+              | a     .
.              | session | --- | hive_t | --- | binding |              | b     .
.              +---------+     +--------+     +---------+              | a     .
.                                  | |                                 | s     .
.       ---------------------------|-|-------------------------------  | e     .
.       sql                        | |                                 |       .
.             ...... HIVE ......   | |   ...... SCHEMA ......          |       .
.                              .   | |   .                             |       .
.                 +--------+   .   | |   .    +--------+               |       .
.                 | hive_q |-------+ +--------|  view  |               |       .
.                 +--------+   .   | |   .    +--------+               |       .
.                     |        .         .        |                  --+       .
.       --------------|---------------------------|------ SELECT -------       .
.       client        |        +---------+        |                            .
.                     +--------|   app   |--------+                            .
.                              +---------+                                     .
.                                                                              .
................................................................................
```

Tutorial

Because the majority of us *learn by example*, let's jump right into a tutorial of how to use Hive-ODCI. All information provided here, such as objects, packages, procedure, roles, etc... are detailed below so feel free to jump back-and forth for references when you encounter something you don't understand.

Scenario

Let's define a real-world scenario, where we have an Oracle database that has multiple Petabytes of data, some of which is *static*, meaning it changes very little if at all once data has been added. We'll call ours the USER_LOG which exists in the SCOTT schema. This table has billions upon billions of records consuming, multiple GB of storage space, which has been collected over the years, logging the activity metrics of our user community. Data over 30 days old is used in monthly reporting, but rarely changes once it's been added to the table.

Our tables looks like this ...

```
SQL> desc SCOTT.USER_LOG

 Name           Null?    Type
 -------------- -------- ---------------
 STAMP          NOT NULL DATE
 ACCOUNT                 VARCHAR2(30)
 MESSAGE                 VARCHAR2(4000)
```

Because we have tons of room available in our Hadoop cluster we decide that we want to move the data there, so it can be indexed and searched. But hold on, we have a problem, we still have an application that reads the table and creates reports for upper management and they are not going to change their application to read from 2 different places using 2 different methods (oh, what to do).

Hive-ODCI to the rescue

On top of inserting, updating and deleting capabilities to the table, the application also contains PL/SQL to create the reports and has a VIEW used in displaying the monthly metrics.

They look something like this ...

```
--
procedure user_log_report( p_report out xmltype ) is
begin

    for rec in ( select account,
                        message
                   from scott.user_log
                  order by account ) loop

        if ( rec.stamp > sysdate - 90 ) then

            p_report := ... -- do something special

        else

            if ( rec.account = user ) then

                p_report := ... -- if current user then ...

            end if;

        end if;

        if ( rec.message like '%ABC%' ) then

            p_report := ... -- write a particular format
        else

            p_report := ... -- write another format

        end if;

    end loop;

end user_log_report;

--
view user_log_monthly
as
select stamp,
       account,
       message
  from scott.user_log
 where stamp between sysdate - 30
                 and sysdate;
```

Assume we have already moved our data over to Haddop and created a Hive table, of the same name. Our remote table looks like this ...

```
$ beeline -u jdbc:hive2://hive.corp.com:10000 \
          -n oracle \
          -w welcome1.passwd

0: jdbc:hive2://localhost:10000> desc user_log;
+-------------------------+------------+----------+--+
|        col_name         | data_type  | comment  |
+-------------------------+------------+----------+--+
| stamp                   | date       |          |
| account                 | string     |          |
| message                 | string     |          |
+-------------------------+------------+----------+--+
3 rows selected (0.17 seconds)
```

Hive-ODCI configuration

Now that we have our table in Hadoop/Hive and we can connect via beeline, and see it there, let's setup Hive-ODCI as an access point to that table.

Because all of our clients will be accessing the same Hadoop/Hive data-store, we can setup a common connection strategy for Hive-ODCI via the parameters

```
dbms_hive.param( 'hive_jdbc_url',   'jdbc:hive2://hive.corp.com:10000' );
dbms_hive.param( 'hive_jdbc_url.1', 'user=oracle' );
dbms_hive.param( 'hive_jdbc_url.2', 'password=welcome1' );
```

If we have clients which need a different connection strategy or use different parameters, they can change these at the session level to meet their needs specifically, but for now we'll assume everyone is using the same thing.

Hive-ODCI object creation

As denoted above we have PL/SQL code and a VIEW which is now invalid because the `SCOTT.USER_LOG` table no longer exists.

So, let's put it back using Hive-ODCI. First let's create a new VIEW which replaces the table.

```
--
grant execute on hive_q to scott;

--
create or replace view scott.user_log
    (
        stamp,
        account,
        message
    )
as
select *
  from table( hive_q( q'[ select stamp,
                                 account,
                                 message
                            from user_log
                           order by stamp ]' ) )
/
```

Whew, that was easy. We now have a column-by-column replacement of the old table with a newly created remote table, that provides the same data types as before.

```
SQL> desc SCOTT.USER_LOG

 Name           Null?    Type
 -------------- -------- ---------------
 STAMP          NOT NULL DATE
 ACCOUNT                 VARCHAR2(4000)
 MESSAGE                 VARCHAR2(4000)
```

Let's see if it worked ...

```
SQL> alter procedure scott.user_log_report compile;

Procedure altered.
```

Excellent, so let's take a look at replacing the view. In this case, we only care about data from the last 30 days, no need to make Hadoop/Hive do more work than it has to. Let's use bindings to restrict the data at the Hadoop/Hive layer instead of at the Oracle layer.

```
--
```

```
grant execute on hive_bind  to scott;
grant execute on hive_binds to scott;

--
create or replace view scott.user_log_monthly
    (
        stamp,
        account,
        message
    )
as
select *
  from table( hive_q( q'[ select stamp,
                                 account,
                                 message
                            from user_log
                           where stamp between ? and ? ]',
              hive_binds( hive_bind( to_char( sysdate - 30,
                                              'yyyy-mm-dd' ),
                                     1 /* type_date */,
                                     1 /* ref_in */ ),
                          hive_bind( to_char( sysdate, ,
                                              'yyyy-mm-dd' ),
                                     1 /* type_date */,
                                     1 /* ref_in */ ) ) )
/
```

Let's break this one down, as it is a little more complex than the other. The hive_q function takes 3 parameters, 2 of which are defaulted to NULL. The first is a hive_binds object which is simply an array of hive_bind objects or individual bind data. The second parameter is the hive_session (e.g. URL, User, ...), but since we are using a common connection strategy we will ignore this argument, not pass it in, let it continue to be NULL

The hive_bind object also takes 3 arguments, the first is the data to be used as the bind. The second is the type of data to be bound (e.g. string, date, number, etc...). And the third is the scope of the bind, most useful for DML operations (e.g. IN, OUT, IN/OUT).

In our case we only care about scope references of IN, hence the 1 /* ref_in */. Since both bind operators are DATE variables they both have 1 /* type_date */.

The actual bound data is based on SYSDATE, but since we don't know up front the NLS_DATE_FORMAT setting for the session, we simply guarantee the format by wrapping it in a TO_CHAR() function and specify the format Hadoop/Hive is expecting.

The Oracle RBAC controls dictate who can SELECT from the VIEWS, just as before. So, if you have custom roles which need access to the VIEWS, they can be granted access in the same way.

```
grant select on scott.user_log to my_app_role;
grant select on scott.user_log_monthly to public;
```

Now wala! We have a VIEW that will retrieve the last 30 days worth of data from Hapdoop/Hive.

Almost there

We now have replacements for both the PL/SQL Reporting and the monthly VIEW, but we're just not yet at a 100%, so what have we forgot?

You guessed it; DML. Our application is still putting data in, modifying it and removing it so we need to support that too. No problem. We can use an INSTEAD OF trigger and send our INSEERT, UPDATE and DELETE commands to Hadopp/Hive using Hive-ODCI.

```
create or replace trigger scott.user_log_dml

    instead of delete or insert or update on scott.user_log
    for each row

declare

    cmd varchar2( 4000 );
    bnd hive_binds := hive_binds();

begin

    if ( inserting ) then

        cmd := q'[ insert into user_log
                       ( stamp, account, message )
                   values
                       ( ?, ?, ? ) ]';

        bnd.extend;
        bnd( bnd.count ) := hive_bind( to_char( :new.stamp,
                                                'yyyy-mm-dd' ),
                                       hive_binding.type_date,
                                       hive_binding.ref_in );

        bnd.extend;
        bnd( bnd.count ) := hive_bind( :new.account,
                                       hive_binding.type_string,
                                       hive_binding.ref_in );


        bnd.extend;
        bnd( bnd.count ) := hive_bind( :new.message,
                                       hive_binding.type_string,
                                       hive_binding.ref_in );

    elsif ( updating ) then

        cmd := q'[ update user_log
                      set account = ?,
                          message = ?
                    where stamp = ? ]';
```

```
            bnd.extend;
            bnd( bnd.count ) := hive_bind( :new.account,
                                           hive_binding.type_string,
                                           hive_binding.ref_in );


            bnd.extend;
            bnd( bnd.count ) := hive_bind( :new.message,
                                           hive_binding.type_string,
                                           hive_binding.ref_in );

            bnd.extend;
            bnd( bnd.count ) := hive_bind( to_char( :new.stamp,
                                                    'yyyy-mm-dd' ),
                                           hive_binding.type_date,
                                           hive_binding.ref_in );

        elsif ( deleting ) then

            cmd := q'[ delete from  user_log
                        where stamp = ? ]';

            bnd.extend;
            bnd( bnd.count ) := hive_bind( to_char( :new.stamp,
                                                    'yyyy-mm-dd' ),
                                           hive_binding.type_date,
                                           hive_binding.ref_in );

        else

            null; -- should never get here

        end if;

        if ( ( cmd is not null ) and ( bnd.count > 0 ) ) then

            -- execute the remote statement
            hive_remote.dml( cmd, bnd );

        end if;

    end user_log_dml;
    /
```

This may look more complex, but it's really not. We are simply creating a trigger to handle the DML events and passing them off to Hive-ODCI. The `HIVE_REMOTE.DML()` works in the same way as a query, accepting an array of bind objects and connection information.

The only real difference here is the example is creating and using local variables for `HIVE_BINDS` and the `HIVE_BINDING` types to show how it can be used in that manner.

Final thought

Note we have **not** changed any code in our application or in our PL/SQL procedure. Everything remains exactly as it was before, but our data exists only in Hadoop/Hive. We can view the data, use it with our PL/SQL and even manipulate it with DML. So, you can go to your meeting now and be the hero.

Guidelines

The following sections are guidelines based on practical real-world experience and are intended to help you in the decision making process only. They are not hard-and-fast rules of "thou shalt not" commandments that must be followed to use Hive-ODCI.

The sections are separated into focus areas for Developers and Administrators. In the real-world these are often blurry, completely undefined and are overlapping. While that is true, no matter what role you play, the distinction between them should always be upheld as much as possible.

For Developers

While Hive-ODCI is intended to help the transition of moving data into a Hadoop/Hive data-store, it can with planning and understanding make it transparent to your customers. But at some point it may require Developer intervention.

Performance Considerations

Hive-ODCI is a `PIPLINE` type, so its performance is impacted almost entirely on the performance of Hadoop/Hive. So make sure that Hive is performing at optimal levels.

Ensure that when moving data from Oracle to Hadoop/Hive or creating new tables you have designed in performance from the beginning.

Analytics over in-line views

If you are using in-line views within the query, you may find that using the built-in `RANK()` and `OVER()` functions perform much better.

For example, take the following query

```
    select user_log.*
      from user_log join
           ( select account,
                    max( stamp ) as stamp
               from user_log
              group by account ) inline
       on user_log.account = inline.account
      and user_log.stamp = inline.stamp;
```

While there is nothing technically wrong with this query, based on the plan it could be much better. let's try again ...

```
    select *
     from ( select *,
                   rank() over
                   ( partition by account,
                     order by stamp ) as rank
             from user_log ) ranking
    where ranking.rank = 1;
```

Much better. We get the same result with almost a magnitude in increase on performance. If all of this looks familiar, then your right. This has been in Oracle RDBMS for years, so use the same common-sense techniques in your Hadoop/Hive queries as you would in your Oracle RDBMS queries.

The CBO

In recent additions of Hadoop/Hive a Cost-Based-Optimizer (CBO) was introduced, which like in Oracle RDBMS, performs optimizations based on cost, which can adjust the execution plan based on order joins, types of joins, degree of parallelism, etc... leading to increases in query performance.

If not enabled globally, you can still use the Hadoop/Hive CBO, by setting the following parameters at the beginning of the query.

```
    set hive.cbo.enable=true;
    set hive.compute.query.using.stats=true;
    set hive.stats.fetch.column.stats=true;
    set hive.stats.fetch.partition.stats=true;
```

Just like Oracle RDBMS, it's important to analyze the tables so the CBO has current cost information. For example, collect statistics at the table and column levels as necessary.

```
    analyze table my_table compute statistics;
    analyze table my_table compute statistics for columns;
    analyze table my_table compute statistics for columns col1, col2;
```

ORCFile

Using ORCFile for Hadoop/Hive table should already be a matter of practice because it is shown to be extremely beneficial in getting fast response times for queries.

If existing tables are not already ORC then it would be prudent to migrate them as soon as possible, even if you convert them by-hand. However, if possible the best case would be to modify the ingest process to use ORC up front.

Apache Tez

Whenever possible use the Apache Tez execution engine instead of a Map-reduce engine. If it is not enabled by default in your environment, then use the following setting at the beginning of the query.

```
    set hive.execution.engine=tez;
```

Vectorized queries

Like scans, aggregations, filters and joins vectorized query execution improves performance of operations by splitting them into batches of 1024 rows at a time instead of single row. If not enabled for your environment then use the following setting to at the beginning of the query.

```
    set hive.vectorized.execution.enabled = true;
    set hive.vectorized.execution.reduce.enabled = true;
```

New horizons

Putting Hive-ODCI into practice is not all that hard, but it is a new way of doing business. And with "new" things come hiccups and unforeseen circumstances that can put you behind or stop you dead in your tracks.

The biggest thing here, is to identify those risks upfront and mitigate them as soon as possible. Don't wait until the last minute, plan ahead and you won't regret it. I know this is "touchy-feely" advice that you can get at the local coffee shop, but we often forget about the small things in our rush.

Change your PL/SQL code

If your PL/SQL can be changed for the better when using Hive-ODCI, do it. In reviewing your PL/SQL you may find that leveraging the functionality provided by Hive-ODCI makes it more readable, faster, maintainable, etc... then go ahead and make arrangements to modify the existing code base or introduce new code.

Sometimes, that's not feasible or even possible but many times we (the big we) have a tendency to force-fit a solution when that isn't warranted based on the situation.

Keep signatures consistent

Even with the above statement being true, it's best to keep the underlying `TABLE` an `VIEW` objects with the same signature, column names, types, lengths, etc...

Familiarize yourself with the API

Finally, make sure you familiarize yourself and are comfortable with the Hive-ODCI API. I know reading documentation is boring (writing it even more so, trust me) but take the time and go through it as many times as needed to get a firm understanding. This is particularly important for Developers, as they are the "real" users of the system.

Write some test scripts, see how it behaves in particular situations and make darn sure you know what's coming when you start inserting this stuff into your process.

If you've read the manual, tried it out, found a bug, or simply do not understand, then see the Authors section for POC information and contact me. I'll be happy to help where I can or provide advice and moral support as needed.

For Administrators

Administrators are by enlarge the last line of defense against all manner of issues, they are the gatekeepers for security, performance, storage, new technologies, legacy systems, and a full onslaught of change management from every direction.

Whether the systems are large or small, clustered or single instance the administrator's toolbox is typically a plethora of knowledge, scripts and documentation. Hive-ODCI attempts add to that toolbox, without being just "something else to learn" by leveraging common facilities which already exist for the administrator. Most Hive-ODCI footprint objects will be first-class citizens in the Oracle ecosystem and will be managed, secured and monitored in the same way as other object types.

Wait events

An administrator knows which wait events to look for and which ones can be ignored or are simply part of a running system. Hive-ODCI does not change those presumptions. Mainly because the load Hive-ODCI incurs is on the remote Hadoop/Hive system and not the Oracle RDBMS.

Something to be aware of, however is that unless all your users are hitting Hive-ODCI object simultaneously a wait event for Hive-ODCI will not float to the top of the wait percentages.

A Hive-ODCI object having issue would still be in one of the more innocuous wait events or classes. Additionally, it won't be in a blocking state, so you have to recognize those events which may be indicating problems for the clients.

WAIT_TIME and SECONDS_IN_WAIT

In the GV$SESSION view the columns WAIT_TIME and SECONDS_IN_WAIT can easily be used to determine if Hive-ODCI is waiting on a remote call to respond or complete.

SQL*Net message to client

This wait event can be observed while the Hive-ODCI is sending or responding to the call. While this event itself is not an issue, one that is taking a longer time than normal may indicate that the query_limit parameter is set too high.

buffer latch and latch free

While these in normal circumstances, indicate query issues they do not necessarily mean the same thing for Hive-ODCI. You may get these events when the Hive-ODCI is materializing large amounts of data, for example in local sorting or aggregate processing of the returned data.

If this event is observed too high, or too often for a particular Hive-ODCI object then look into off-loading the sort, group or aggregate operations to Hadoop/Hive instead of locally.

Take for example the following

```
select *
  from table( hive_q( q'[ select cust_id,
                                 last_name,
                                 first_name
                            from cust ]',
                         null,
                         null ) )
 order by cust_id;
```

If this query is showing up in a wait event that is indicating issues materializing the cust_id column locally for sorting then considered making the following change to off-load that operation to Hadoop/Hive

```
select *
  from table( hive_q( q'[ select cust_id,
                                 last_name,
                                 first_name
                            from cust
                           order by cust_id ]',
                         null,
                         null ) );
```

This produces the same end result, but without local materialization and with less wait activity in the Oracle RDBMS.

Storage

Hive-ODCI is a zero storage object solution, like a VIEW, for the majority of how its used. However certain aspects of Hive-ODCI will consume space, such as the Hive-ODCI Log and the Saved Filters (Binding).

It is unlikely that your user community, no matter how large, or how much Hive-ODCI is used will generate Filters that impact your storage, necessitating TS extents beyond what was allocated during the installation.

The Hive-ODCI Log data is another story. Depending on the log_level set for the System, large amounts of data can be generated. The log_level value is a bitmask, which turns types on and off. The types are detailed below but also here

```
create or replace package impl as

    --
    none  constant number := 0;
    error constant number := 1;
    warn  constant number := 2;
    info  constant number := 4;
    trace constant number := 8;

    ...

end impl;
```

Each level is progressively verbose with what is written. The types should be obvious, but needless to say error writes only critical exceptions, while trace writes all operations. So a value of 3 would be error + warn and a value of 31 would be error + warn + info + trace.

If you find that the Hive Log is filling up faster than expected, review the column VALUE in the DBA_HIVE_PARAMS to determine the current value.

```
select value
  from dba_hive_params
 where name = 'log_level';
```

If you find that the log_level is set appropriately, or as you expected, then this means that the client has set the session_log_level() to something too high, which may need to be changed or justified.

The account which created the log data is stored in the column NAME found in the DBA_HIVE_LOG view. Use the following to get counts of the name generating the most logging information.

```
select name,
       count(0) total
  from dba_hive_log
 group by name
 order by 2 desc;
```

Log information can be purged by any trusted account granted the HIVE_ADMIN role using the following statement.

```
exec dbms_hive.purge_log;
```

Be aware that the statement removes **ALL** log data.

Move to a different TS

By default Hive-ODCI installs to the same tablespace assigned to the SYSTEM schema. This for most installations will be sufficient. If you disagree with the installation choice, or you have a larger storage consumption than anticipated you can move Hive-ODCI to another tablespace using the MOVE_TS procedure in the DBMS_HIVE package.

```
create or replace package dbms_hive as

    ...

    --
    procedure move_ts( ts  in varchar2,
                       obj in varchar2 default null );

end dbms_hive;
```

Purge log and filters

If the Hive-ODCI log data or filter data becomes too large, it can easily be purged using the PURGE_LOG and PURGE_FILTER procedures respectively in the DBMS_HIVE package.

```
create or replace package dbms_hive as

    ...

    --
    procedure purge_log;
    procedure purge_filter( key in varchar2 default null );

    ...

end dbms_hive;
```

These procedures do exactly what they indicate. And because they are exposed through the Hive-ODCI management package they can be called by a trusted user who has been granted the HIVE_ADMIN role.

Role Based Access Control

As indicated multiple times now, Hive-ODCI objects are primarily first-class and are managed with RBAC just like any other object.

When Hive-ODCI is installed, it creates 2 helper roles for you setting up the appropriate permissions for their use.

The HIVE_USER role has access to the common Hive-ODCI objects making it possible to Query, use DML or DDL, manage Bind variables, etc...

The second role, is HIVE_ADMIN which provides access to the management of Hive-ODCI itself, such as Logs, Parameters, etc... This role is also granted to the DBA role by default so be cognizant that a user with HIVE_ADMIN is considered a trusted and responsible party, similar to a user with DBA but having less permissions in the database.

Bindings

Filters, or saved HIVE_BINDS work more like Network ACL (see DBMS_NETWORK_ACl_ADMIN) and Java Policies (see DBMS_JAVA) than they do first-class object permissions.

When created a Filter will by default assign ownership of the HIVE_BINDS object key to the account which created it.

That account, or a HIVE_ADMIN assigned account can grant and revoke permissions to both users and roles using the HIVE_BINDING type methods.

```
create or replace package binding as

    ...

    --
    priv_read       constant guard     :=  1;
    priv_write      constant guard     :=  2;
    priv_readwrite  constant guard     :=  3;

    --
    procedure allow( key in varchar2,
                     act in varchar2,
                     lvl in guard default priv_readwrite );

    --
    procedure deny( key in varchar2,
                    act in varchar2 );


    ...

end binding;
```

The ALLOW() procedure works like a GRANT and the DENY() works like a REVOKE

When a HIVE_BIND is accessed from the stored Filter the permission is checked for the operation (e.g. read, write, etc...). If a PUBLIC grant is made, then all accounts may access the Filter at the level granted.

Java Policies

Hive-ODCI will in no way automatically grant or revoke Java Policy permissions, therefore is is necessary for the administrator to take on this action as needed.

When Hive-ODCI attempts to set a Java Property at run-time, which has not been granted the appropriate policy an exception is thrown. As part of that exception, the necessary DBMS_JAVA.GRANT_PERMISSION() is provided to the caller as part of the exception text. It is likely that the text generated by the database will be sent to the administrator requesting it be executed.

Before doing so, make sure there is a clear understanding of what that permission provides as well as what the property enables or achieves.

Failure to understand the property and/or permission may result in reducing the security posture of the database or at worst make the database unstable. For the most part this will not happen, but it is good practice to have full situational awareness before proceeding.

API

The Application programming Interface (API) for Hive-ODCI is accessible through the PL/SQL objects created during installation. The objects include Views, Packages, Procedure, Types and Functions each providing a unique set of functionality based on need.

Views

Views are the insight into Hive-ODCI, providing invaluable information for both Administrators and Users.

dba_hive_params

Lists both the system and session level values.

This view is accessible only by the `HIVE_ADMIN role

- Columns

```
name            - Parameter name
session_value   - Current session value
system_value    - Default system value
```

dba_hive_filters

Lists the saved Filters.

This view is accessible only by the `HIVE_ADMIN role

- Columns

```
key     -   Filter key name
seq     -   The ordinal bind sequence
type    -   The bind type
scope   -   The bind reference scope
value   -   The value of the bind data
```

dba_hive_filter_privs

Lists the privileges of the saved Filters.

This view is accessible only by the `HIVE_ADMIN role

- Columns

```
key     -   Filter key name
grantee -   The grantee name, user or role
read    -   Contains read access
write   -   Contains write access
```

dba_hive_log

Lists the log data

This view is accessible only by the `HIVE_ADMIN role

- Columns

```
stamp   -   The time stamp the log record
name    -   Account name who create the log record
type    -   The type of log record
text    -   The text of the log record
```

user_hive_params

Lists current parameter values used by the session

This view is accessible by both the HIVE_ADMIN`` andHIVE_USER``` roles

- Columns

```
name    -   Parameter name
value   -   Parameter value
```

user_hive_filters

Lists the saved Filters which are accessible by the session

This view is accessible by both the HIVE_ADMIN`` andHIVE_USER``` roles

- Columns

```
key     -   Filter key name
seq     -   The ordinal bind sequence
type    -   The bind type
scope   -   The bind reference scope
value   -   The value of the bind data
```

user_hive_filter_privs

Lists the privileges of the saved Filters which are accessible by the session

This view is accessible by both the HIVE_ADMIN`` andHIVE_USER``` roles

- Columns

```
key     -   Filter key name
grantee -   The grantee name, user or role
read    -   Contains read access
write   -   Contains write access
```

Packages

binding

An interface for creating, manipulating and saving HIVE_BINDS arrays of HIVE_BIND objects. These objects are used in the marshaling and interpretation of arguments to and from the JDBC Driver providing both data type and direction.

This object is accessible through the HIVE_USER role and is referenced as the SYNONYM name HIVE_BINDING

Subtypes

- reference -- number for bind scope values
- typeof -- number for bind value types
- guard -- number for bind access privileges

Constants

Helper constants for abstracting type information

Generic

```
Name              Value
----------------- --------
none                     0
```

Privilege

```
Name              Value
----------------- --------
priv_read                1
priv_write               2
priv_readwrite           3
```

Scope

```
Name              Value
----------------- --------
scope_in                 1
scope_out                2
scope_inout              3
```

Type

```
Name              Value
----------------- --------
type_bool                1
type_date                2
type_float               3
type_int                 4
type_long                5
type_null                6
type_rowid               7
type_short               8
type_string              9
type_time               10
type_timestamp          11
type_url                12
```

Interface

get()

Overloaded FUNCTION to retrieve a saved HIVE_BINDS filter based on Key name or returns a single HIVE_BIND from the session array.

- Prototype

```
function get( key in varchar2 ) return binds;

function get( idx in number, lst in binds ) return bind;
```

- Parameter

```
key     -   Saved key name
idx     -   Ordinal number of the item.
```

count()

Overloaded FUNCTION to count the length of a saved HIVE_BINDS filter based on Key name or an existing one provided.

- Prototype

```
function count( key in varchar2 ) return number;
function count( lst in binds ) return number;
```

- Parameter

```
key     -   Saved key name
lst     -   Existing HIVE_BINDS array
```

new()

A FUNCTION to create a new, single, HIVE_BIND object.

- Prototype

```
function new( value in varchar2,
              type  in typeof    default type_string,
              scope in reference default scope_in ) return bind;
```

- Parameter

```
value   -   Value of the HIVE_BIND
type    -   Interpretation type of the HIVE_BIND
scope   -   Reference scope, direction, for using the HIVE_BIND
```

append()

Overloaded PROCEDURE to append a HIVE_BIND object to an existing HIVE_BINDS array, saved or passed in.

- Prototype

```
procedure append( key   in varchar2,
                  value in varchar2,
                  type  in typeof    default type_string,
                  scope in reference default scope_in );

procedure append( value in     varchar2,
                  type  in     typeof    default type_string,
                  scope in     reference default scope_in,
                  lst   in out binds );

procedure append( key in varchar2, val in bind );

procedure append( val in bind, lst in out binds );

procedure append( key in varchar2, val in binds );

procedure append( val in binds, lst in out binds );
```

- Parameter

```
key     -   Saved key name
value   -   Value of the HIVE_BIND
type    -   Interpretation type of the HIVE_BIND
scope   -   Reference scope, direction, for using the HIVE_BIND
lst     -   Existing HIVE_BINDS array
val     -   A HIVE_BINDS array to be appended to the existing array
```

change()

A PROCEDURE to modify an exiting element of a saved HIVE_BIND object.

- Prototype

```
procedure change( key   in varchar2,
                  idx   in number,
                  value in varchar2,
                  type  in typeof    default type_string,
                  scope in reference default scope_in );
```

- Parameter

```
key     -   Saved key name of the array
idx     -   Ordinal number of the item to modify
value   -   Value of the HIVE_BIND
type    -   Interpretation type of the HIVE_BIND
scope   -   Reference scope, direction, for using the HIVE_BIND
```

remove()

A PROCEDURE to delete an exiting element of a saved HIVE_BIND object.

- Prototype

```
procedure remove( key in varchar2,
                  idx in number );
```

- Parameter

```
key     -   Saved key name of the array
idx     -   Ordinal number of the item to modify
```

replace()

A PROCEDURE to replace one HIVE_BINDS array with another.

- Prototype

```
procedure replace( val in binds, lst in out binds );
```

- Parameter

```
val     -   A HIVE_BINDS array to be appended to the existing array
```

```
lst     -   HIVE_BINDS array to overwrite
```

clear()

An overloaded PROCEDURE to remove all elements in a saved Filter or a passed in array.

- Prototype

```
procedure clear( key in varchar2 );
procedure clear( lst in out binds );
```

- Parameter

```
key     -   Saved key name of the array
lst     -   HIVE_BINDS array to overwrite
```

allow()

A PROCEDURE to grant access rights to saved Filter. If an existing grant was already made, then it is replaced by the change

- Prototype

```
procedure allow( key in varchar2,
                 act in varchar2,
                 lvl in guard default priv_readwrite );
```

- Parameter

```
key     -   Saved key name of the array
act     -   Name of the account to allow access, which
            can be an Oracle username or role
lvl     -   The level of access to grant
```

deny()

A PROCEDURE to revoke access rights from a saved Filter. If no existing grant exists an exception if thrown.

- Prototype

```
procedure deny( key in varchar2,
                act in varchar2 );
```

- Parameter

```
key     -   Saved key name of the array
act     -   Name of the account to allow access, which
            can be an Oracle username or role
```

save()

A PROCEDURE to save a passed in HIVE_BINDS array. If a key of the same name already exists, an exception is thrown.

- Prototype

```
procedure save( key in varchar2, lst in binds );
```

- Parameter

```
key     -   Saved key name of the array
lst     -   HIVE_BINDS array to save
```

Exception

ex_unknown

This exception is thrown when an unknown error has been encountered.

```
ex_unknown  exception;
es_unknown  constant varchar2( 256 ) := 'Unknown error encountered';
ec_unknown  constant number := -20001;
pragma      exception_init( ex_unknown, -20001 );
```

ex_denied

This exception is thrown when a saved Filter access attempt has failed because of insufficient privileges.

```
ex_denied   exception;
es_denied   constant varchar2( 256 ) := 'Request denied,
                                         insufficient privileges';
ec_denied   constant number := -20002;
pragma      exception_init( ex_denied, -20002 );
```

ex_no_grant

This is exception is thrown when a revoke attempt is made on a saved Filter that does not contain a previous grant for the account specified

```
ex_no_grant exception;
es_no_grant constant varchar2( 256 ) := 'Privileges not granted';
ec_no_grant constant number := -20003;
pragma      exception_init( ex_no_grant, -20003 );
```

dbms_hive

A PACKAGE provideing the interfaces for management of Hive-ODCI configurations.

This object is accessible through the HIVE_ADMIN role and is referenced as the SYNONYM name DBMS_HIVE

Interface

exist()

A FUNCTION returning a BOOLEAN value (true or false) if the parameter name exists

- Prototype

```
function exist( name in varchar2 ) return boolean;
```

- Parameter

```
name    -   Case sensitive name of the parameter to check
```

param()

An overloaded FUNCTION and PROCEDURE to both set and get a parameter. When setting, if the parameter exists then it's value is overwritten

- Prototype

```
-- get
function param( name in varchar2 ) return varchar2;

-- set
procedure param( name in varchar2, value in varchar2 );
```

- Parameter

```
name    -   Case sensitive name of the parameter to get
            or set
value   -   Value of the parameter when setting. The value
            is the return for the FUNCTION
```

If the a parameter does not exist, then the function returns NULL

remove()

A PROCEDURE for removing a parameter. If the parameter does not exist, no error is thrown.

- Prototype

```
procedure remove( name in varchar2 );
```

- Parameter

```
name    -   Case sensitive name of the parameter to
            delete
```

purge_log()

A PROCEDURE which purges all data in the `DBA_HIVE_LOG

- Prototype

```
procedure purge_log;
```

purge_filter()

A PROCEDURE for removing all saved Filters by name. If no name value or a NULL is provided as the argument then **ALL** Filters are removed.

- Prototype

```
procedure purge_filter( key in varchar2 default null );
```

- Parameter

```
key     -   Key name of the array to remove, this defaults
            to NULL which indicates ALL values are removed
```

move_ts()

A PROCEDURE moving the Hive-ODCI objects to different tablespace. If the current tablespace name is the same as the one specified then no errors are thrown. If the tablespace specified does not exists then an exception is thrown.

An OBJECT name is alternatively provided which will move only a single object. If no value or NULL is used then **ALL** objects are moved.

Only the values "NULL", "param", "filter", "priv" or "log" can be used, all other values will be ignored.

- Prototype

```
procedure move_ts( ts  in varchar2,
                   obj in varchar2 default null );
```

- Parameter

```
ts      -   Destination tablespace name.
obj     -   Optional object name, can be NULL, "param", "filter",
            "priv" or "log". Any other value will be ignored
```

remote

This PACKAGE is the interface for managing the remote connectivity of the current session. This is also the interface to execute remote commands, such as queries, DDL and DML operations.

This object is accessible through the HIVE_USER role and is referenced as the SYNONYM name HIVE_REMOTE

Interface

session_param()

Overloaded FUNCTION and PROCEDURE to get and set parameters only for the current session. Parameters set using this PROCEDURE are valid only for the duration of the Oracle session and are discarded when the session ends.

A session level parameter takes precedence over the system level parameter of the same name.

- Prototype

```
-- get
function session_param( name  in varchar2 ) return varchar2;

-- set
procedure session_param( name  in varchar2,
                         value in varchar2 );
```

- Parameter

```
name    -   Case sensitive name of the parameter to get
            or set
value   -   Value of the parameter when setting. The value
            is the return for the FUNCTION
```

If the session level parameter is not set, or does not exist then the function returns NULL

session_log_level()

A PROCEDURE which allows the logging level to be set for the current session. This setting is valid only for the duration of the Oracle session and are discarded when the session ends.

- Prototype

```
procedure session_log_level( typ in number );
```

- Parameter

```
typ     -   The log level type
```

Log level values are bit masks, to allow for multiple values to be specified in a single numeric value. They are defined as part of the IMPL package specification.

```
none  constant number := 0;
error constant number := 1;
warn  constant number := 2;
info  constant number := 4;
trace constant number := 8;
```

session_clear()

This PROCEDURE clears the session CONNECTION type.

- Prototype

```
procedure session_clear;
```

After being cleared, consecutive attempts will rebuild the CONNECTION type from the parameter information.

session()

An overloaded FUNCTION and PROCEDURE which creates, gets, or modifies the current session CONNECTION type.

- Prototype

```
-- get
function session return connection;

-- set
procedure session( url in varchar2 );

procedure session( usr in varchar2,
                   pwd in varchar2 );

procedure session( url in varchar2,
                   usr in varchar2,
                   pwd in varchar2 );

procedure session( url in varchar2,
                   usr in varchar2,
                   pwd in varchar2,
                   ath in varchar2 );

procedure session( con in connection );
```

- Parameter

```
url     -   The JDBC URL of the remote connection
usr     -   User name to be applied for the connection
pwd     -   The user password to be applied
ath     -   The authentication type of the connection
```

When getting the current CONNECTION type, the password value is redacted by the function, no matter what its value or the authentication type specified.

query()

A PIPELINED FUNCTION which returns an ANYDATASET value. This value contains the description information of a given row type along with the set of data instances of that row.

- Prototype

```
function query( stm in varchar2,
                bnd in binds      default null,
                con in connection default null )
         return anydataset pipelined using hive_t;
```

- Parameter

```
stm     -   The SQL query statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the SQL statement
con     -   Optionally, the remote CONNECTION information
```

This function can be used in a casting statement to retrieve the remote records, but is does not contain a PACKAGE BODY definition, rather it uses the HIVE_T ODCI object type making the following 2 examples equivalent

```
val := hive_q( 'select cust_id,
                       last_name,
                       first_name
                  from cust', null, null );
```

vs.

```
val := hive_remote.query( 'select cust_id,
                                  last_name,
                                  first_name
                             from cust', null, null );
```

dml()

This PROCEDURE allows remote DML to be executed using the provided SQL statement, bind variables and connection type.

- Prototype

```
procedure dml( stm in varchar2,
               bnd in binds      default null,
               con in connection default null );
```

- Parameter

```
stm     -   The DML statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the DML statement
con     -   Optionally, the remote CONNECTION information
```

ddl()

This PROCEDURE allows remote DDL to be executed using the provided command statement and connection type.

- Prototype

```
procedure ddl( stm in varchar2,
               con in connection default null );
```

- Parameter

```
stm     -   The DML statement to be executed remotely
con     -   Optionally, the remote CONNECTION information
```

impl

This PACKAGE is the primary implementation of the ODCI layer and the interface called by Oracle ODCI through the HIVE_T object type.

This interface is used internally only, and while it is available for execution with the HIVE_ADMIN role it should never be called directly. It is included here only for completeness of the documentation.

There is no SYNONYM for this object and both its specification and body are wrapped during the installation process.

Constants

Helper constants for abstracting type information

Log level

```
none  constant number := 0;
error constant number := 1;
warn  constant number := 2;
info  constant number := 4;
trace constant number := 8;
```

Log level values are bit masks, to allow for multiple values to be specified in a single numeric value.

Interface

log()

This procedure writes a logging record of the type specified, if log_level at the systems or session set set to support it.

- Prototype

```
procedure log( typ in number, txt in varchar2 );
```

Additionally, there are pass-through procedures which call this procedure using the predefined type.

```
procedure log_error( txt in varchar2 );
procedure log_warn( txt in varchar2 );
procedure log_info( txt in varchar2 );
procedure log_trace( txt in varchar2 );
```

- Parameter

```
typ     -   The type of log to write, ignored if the log_level
```

```
                is not set to support that type
    txt     -   The text of the message
```

session_param()

Overloaded FUNCTION and PROCEDURE to get and set parameters only for the current session. Parameters set using this PROCEDURE are valid only for the duration of the Oracle session and are discarded when the session ends.

A session level parameter takes precedence over the system level parameter of the same name.

- Prototype

```
    -- get
    function session_param( name  in varchar2 ) return varchar2;

    -- set
    procedure session_param( name  in varchar2,
                             value in varchar2 );
```

- Parameter

```
    name    -   Case sensitive name of the parameter to get
                or set
    value   -   Value of the parameter when setting. The value
                is the return for the FUNCTION
```

If the session level parameter is not set, or does not exist then the function returns NULL

session_log_level()

A PROCEDURE which allows the logging level to be set for the current session. This setting is valid only for the duration of the Oracle session and are discarded when the session ends.

- Prototype

```
    procedure session_log_level( typ in number );
```

- Parameter

```
    typ     -   The log level type
```

session_clear()

This PROCEDURE clears the session CONNECTION type.

- Prototype

```
    procedure session_clear;
```

After being cleared, consecutive attempts will rebuild the CONNECTION type from the parameter information.

session()

An overloaded FUNCTION and PROCEDURE which creates, gets, or modifies the current session CONNECTION type.

- Prototype

```
    -- get
    function session return connection;

    -- set
    procedure session( url in varchar2 );

    procedure session( usr in varchar2,
                       pwd in varchar2 );

    procedure session( url in varchar2,
                       usr in varchar2,
                       pwd in varchar2 );

    procedure session( url in varchar2,
                       usr in varchar2,
                       pwd in varchar2,
                       ath in varchar2 );

    procedure session( con in connection );
```

- Parameter

```
    url     -   The JDBC URL of the remote connection
    usr     -   User name to be applied for the connection
    pwd     -   The user password to be applied
    ath     -   The authentication type of the connection
```

When getting the current CONNECTION type, the password value is **not** redacted by this function, unlike the function of the same name used in the REMOTE package

sql_describe()

This overloaded FUNCTION is called by ODCI to describe information for a table whose return type is ANYDATASET. This function is called when HIVE_T ODCITableDescribe is encountered.

- Prototype

```
    function sql_describe( stm in varchar2,
                           bnd in binds      default null,
                           con in connection default null ) return anytype;

    function sql_describe( typ out anytype,
                           stm in  varchar2,
                           bnd in  binds      default null,
                           con in  connection default null ) return number;

    function sql_describe( key in  number,
                           typ out anytype ) return number;
```

- Parameter

```
stm     -   The SQL query statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the SQL statement
con     -   Optionally, the remote CONNECTION information
typ     -   The ANYTYPE value that describes the returned rows
            from the table function
key     -   The transient key value used in the concurrent
            calling chain, identifying the record of the context
```

If successful the return value for each function will be ODCICONST.SUCCESS otherwise it will return ODCICONST.ERROR

sql_open()

This FUNCTION is called by ODCI to initialize the scan of a table function. This function is called when HIVE_T ODCITableStart is encountered.

- Prototype

```
function sql_open( key out number,
                   stm in  varchar2,
                   bnd in  binds      default null,
                   con in  connection default null ) return number;
```

- Parameter

```
key     -   The transient key value used in the concurrent
            calling chain, identifying the record of the context
stm     -   The SQL query statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the SQL statement
con     -   Optionally, the remote CONNECTION information
```

If successful the return value for each function will be ODCICONST.SUCCESS otherwise it will return ODCICONST.ERROR

sql_fetch()

This FUNCTION is called by ODCI to next batch of rows from a table function. This function is called when HIVE_T ODCITableFetch is encountered.

- Prototype

```
function sql_fetch( key in  number,
                    num in  number,
                    rws out records ) return number;
```

- Parameter

```
key     -   The transient key value used in the concurrent
            calling chain, identifying the record of the context
num     -   The number of rows the system expects in the current
            fetch cycle.
rws     -   The RECORDS array for the rows requested in the
            current fetch cycle.
```

If successful the return value for each function will be ODCICONST.SUCCESS otherwise it will return ODCICONST.ERROR

sql_close()

This FUNCTION is called by ODCI to performs cleanup operations after scanning a table function cycle is complete. This function is called when HIVE_T ODCITableClose is encountered.

- Prototype

```
function sql_close( key in number ) return number;
```

- Parameter

```
key     -   The transient key value used in the concurrent
            calling chain, identifying the record of the context
```

If successful the return value for each function will be ODCICONST.SUCCESS otherwise it will return ODCICONST.ERROR

sql_dml()

This PROCEDURE allows remote DML to be executed using the provided SQL statement, bind variables and connection type.

- Prototype

```
procedure sql_dml( stm in  varchar2,
                   bnd in  binds      default null,
                   con in  connection default null );
```

- Parameter

```
stm     -   The DML statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the DML statement
con     -   Optionally, the remote CONNECTION information
```

sql_ddl()

This PROCEDURE allows remote DDL to be executed using the provided command statement and connection type.

- Prototype

```
procedure sql_ddl( stm in  varchar2,
                   con in  connection default null );
```

- Parameter

```
stm     -   The DML statement to be executed remotely
con     -   Optionally, the remote CONNECTION information
```

Exception

ex_not_eligible

This exception is thrown when a request to change a parameter at the session level is ineligible for the operation.

```
ex_not_eligible exception;
es_not_eligible constant varchar2( 256 ) := 'Parameter is not
                                            eligible for
                                            change at the
                                            session level';
ec_not_eligible constant number          := -20103;
```

Functions

Only the HIVE_Q function is available for execution, while all other functions are for internal use only.

hive_q

A PIPELINED FUNCTION which returns an ANYDATASET value. This value contains the description information of a given row type along with the set of data instances of that row.

This function is accessible through the HIVE_USER role and is referenced as the SYNONYM name HIVE_Q

- Prototype

```
function hive_q( stm in varchar2,
                 bnd in binds      default null,
                 con in connection default null )
         return anydataset pipelined using hive_t;
```

- Parameter

```
stm     -   The SQL query statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the SQL statement
con     -   Optionally, the remote CONNECTION information
```

This function can be used in a casting statement to retrieve the remote records, but is does not contain a PACKAGE BODY definition, rather it uses the HIVE_T ODCI object type making the following 2 examples equivalent

```
val := hive_q( 'select cust_id,
                       last_name,
                       first_name
                  from cust', null, null );
```

vs.

```
val := hive_remote.query( 'select cust_id,
                                  last_name,
                                  first_name
                             from cust', null, null );
```

bitor

This FUNCTION performs a logical inclusive bit-wise OR operation on each pair of corresponding bits provided.

- Prototype

```
function bitor( x in number, y in number ) return number;
```

- Parameter

```
x       -   First bit mask
y       -   Second bit mask
```

bitxor

This FUNCTION performs the logical exclusive OR operation on each pair of corresponding bits provided.

- Prototype

```
function bitxor( x in number, y in number ) return number;
```

- Parameter

```
x       -   First bit mask
y       -   Second bit mask
```

bitnot

This FUNCTION performs logical negation on each bit, forming the complement of the given binary value provided.

- Prototype

```
function bitnot( x in number ) return number;
```

- Parameter

```
x       -   Bit mask
```

oid

This FUNCTION returns the database object identifier for the name provided, which is expected to be a USERNAME or ROLE

The return value is used as the join key for the saved Filter privileges

- Prototype

```
function oid( o in varchar2 ) return number;
```

- Parameter

```
o       -   Object name
```

oname

This FUNCTION returns the database object name for the identifier provided, which is expected to be a USER# representing the USERNAME or ROLE

The return value is used as the join key displaying names for the saved Filter privileges.

- Prototype

```
function oname( o in number ) return varchar2;
```

- Parameter

```
o       -   Object identifier
```

Types

Hive-ODCI types are both objects and table (arrays) and are used in the transparent conversions from the JDBC Driver to the PL/SQL calls.

attribute

This OBJECT type is used to describe a column value. it is used internally only to define rows in a record set and a table description.

- Members

```
name    - Name of the column
code    - Data type code of the column
prec    - The precision of the column
scale   - The scale of the column
len     - The length of the column
csid    - The locale identifier
csfrm   - The locale format
```

attributes

A TABLE array of ATTRIBUTE objects.

- Members

```
table of attribute
```

data

This OBJECT type contains a single column of information.

- Members

```
code          - The data type code, pointing to the member
                which actually contains the data
val_varchar2  - String data
val_number    - Numeric data
val_date      - Date data
val_timestamp - Timestamp data
val_clob      - CLOB data
val_blob      - BLOB data
```

records

A TABLE array of DATA objects, representing a single row

- Members

```
table of data
```

connection

This OBJECT type contains the remote connection information

- Members

```
url     -   The JDBC URL of the remote connection
name    -   User name to be applied for the connection
pass    -   The user password to be applied
auth    -   The authentication type of the connection
```

bind

This OBJECT defines the bind variable used for the remote command

- Members

```
value   - The value of the data to be bound
type    - The data type code, for how to bind the data
```

```
scope   - The reference code direction of the bind
```

binds

A TABLE array of BIND objects

- Members

```
table of bind
```

hive_t

An OBJECT type for the data cartridge abstraction later. This object is the compliant interface for ODCI callback functionality.

- Members

```
key     - The transient key value used in the concurrent
          calling chain, identifying the record of the context
ref     - The persistent ANYTYPE reference being populated
          in the current calling cycle
```

ODCITableDescribe

Retrieves describe information for a table function whose return type is ANYDATASET.

- Prototype

```
function ODCITableDescribe( typ out anytype,
                            stm in  varchar2,
                            bnd in  binds      := null,
                            con in  connection := null ) return number
```

- Parameter

```
typ     -   The AnyType value that describes the returned rows from
            the table function
stm     -   The command statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the statement
con     -   Optionally, the remote CONNECTION information
```

ODCITablePrepare

This FUNCTION prepares the scan context and command information upon a compile request.

- Prototype

```
function ODCITablePrepare( ctx out hive_t,
                           inf in  sys.ODCITabFuncInfo,
                           stm in  varchar2,
                           bnd in  binds      := null,
                           con in  connection := null ) return number
```

- Parameter

```
ctx     -   The scan context created by this routine is the value
            passed in as a parameter to the later routines
            in the command cycle
inf     -   The projection information and the return descriptor
            object
stm     -   The command statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the statement
con     -   Optionally, the remote CONNECTION information
```

ODCITableStart

This FUNCTION initializes the scan of a table function to start the command cycle.

- Prototype

```
function ODCITableStart( ctx in out hive_t,
                         stm in     varchar2,
                         bnd in     binds      := null,
                         con in     connection := null ) return number
```

- Parameter

```
ctx     -   The scan context modified by this routine is the value
            passed in as a parameter to the later routines
            in the command cycle
stm     -   The command statement to be executed remotely
bnd     -   Optionally, the HIVE_BINDS array containing the
            values to be bound to the statement
con     -   Optionally, the remote CONNECTION information
```

ODCITableFetch

This FUNCTION returns the next batch of rows in the command cycle.

- Prototype

```
function ODCITableFetch( cts in out hive_t,
                         num in     number,
                         rws out    anydataset ) return number
```

- Parameter

```
ctx     -   The scan context modified by this routine is the value
```

```
                passed in as a parameter to the later routines
                in the command cycle
    num     -   The number of rows the system expects in the current
                fetch cycle.
    rws     -   The RECORDS array for the rows requested in the
                current fetch cycle.
```

ODCITableClose

This FUNCTION performs cleanup operations after command cycle is complete

- Prototype

```
    function ODCITableClose( ctx in hive_t ) return number
```

- Parameter

```
    ctx     -   The scan context created by this routine is the value
                passed in as a parameter to the later routines
                in the command cycle
```

Parameters

Parameters provide the customization of Hive-ODCI.

application

The application name

version

The installed or patched version

license

The BSD licensing agreement

log_level

The current log level

hive_jdbc_driver

The fully qualified Java class name of the driver to load

hive_jdbc_url

The JDBC URL for the remote system

hive_jdbc_url.X

These paraemters are addtional URL key value pairs in consecutive order. A gap in the numbering sequence will cause Hive-ODCI to stop reading the parameters assuming that it has reached the end of the list.

java_property.X

These paraemters are the Java Syetms properties to set in consecutive order. A gap in the numbering sequence will cause Hive-ODCI to stop reading the parameters assuming that it has reached the end of the list.

hive_user

The `Driver,.getConnection()` call in Java is overloaded to accept a URL only or optionally with a User/Password. When set this parameter is passed through to the call as the user value.

hive_pass

The `Driver,.getConnection()` call in Java is overloaded to accept a URL only or optionally with a User/Password. When set this parameter is passed through to the call as the password value.

hive_auth

The authentication type for the JDBC Driver.

query_limit

The ceiling limitation for any given query,. When set no query will return more rows than the parameter specified

Roles

hive_user

This role provides the access privileges necessary for common Hive-ODCI functionality.

hive_admin

This role provides the escalated access privileges necessary used by administrators of Hive-ODCI.

Synonyms

Hive-ODCI synonyms are the public alternative names for the interface objects providing location transparency.

- hive_q
- hive_t
- hive_remote
- hive_bind
- hive_binds
- hive_binding
- hive_attribute
- hive_attributes
- hive_data
- hive_records
- hive_connection
- dbms_hive
- dba_hive_params
- dba_hive_filters
- dba_hive_filter_privs
- dba_hive_log
- user_hive_params
- user_hive_filters
- user_hive_filter_privs