Practical 7:

A. Querying Oracle metadata – the data dictionary
B. Indexes
C. Views

Learning objectives:

1. Using the Data Dictionary Views to obtain user information
2. Knowing when to apply indexes to speed up processing
3. Using VIEWS to enhance access control

References

1. Oracle Database 2-Days Developer Guide
https://docs.oracle.com/en/database/oracle/oracle-database/19/tdddg/

2. Oracle Database SQL Language Reference
https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlrf/sql-language-reference.pdf

A. Querying Oracle metadata – the data dictionary

The Oracle data dictionary maintains a lot of information on activities that are performed on the database.

Data Dictionary that can be queried by a user:

user_sys_privs	role_tab_privs	user_indexes
user_tab_privs_made	user_tables	user_ind_columns
user_tab_privs_recd	user_tab_columns	user_views
user_col_privs_recd	user_constraints	user_source
user_role_privs	user_cons_columns	user_sequences
role_sys_privs		

Task

1. Copy to C:\ drive **privis.sql.**

2. View some metadata information:

```
SQL> spool c:\metadata.txt
SQL> start c:\privis.sql
SQL> spool off
```

Open the metadata.txt file and observe some of the information in relation to the current state of your database.

B. Indexes

Reference: SQL Language Reference
References for further reading:
http://www.interspire.com/content/2006/02/15/introduction-to-database-indexes
http://www.db2go.com/articles/an-introduction-to-database-indexing.html

Due to the small data set in our sample database, there will not be any noticeable speed difference in accessing the data even if we create indexes for the various columns in the various tables. However, we will practice using the CREATE INDEX commands.

By default, ORACLE will automatically create an index for the primary key of each table.
Query the data dictionary views to see the tables and the various indexes already defined for the primary key.

```
Select A.INDEX_NAME, A.INDEX_TYPE, A.TABLE_OWNER, A.TABLE_NAME,
       A.TABLE_TYPE,A.UNIQUENESS, B.COLUMN_NAME
   from user_indexes A, user_ind_columns B
   where A.index_name = B.index_name
   order by table_owner, table_name;
```

When to create indexes

If your query uses columns other than the PK in its search condition (and if you run this query often) consider creating an index on the column to speed up processing.
[NB: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So indexes should only be created on columns (and tables) that will be frequently searched against.]

For example:

```
Select ProductCode, ProductName, MSRP, quantityInStock
from PRODUCTS
where ProductName LIKE 'A%';
```

This query uses the `ProductName` as a search criteria. Therefore `ProductName` should be indexed if the range of product is large and this query is used often.

```
CREATE INDEX prod_name_idx ON PRODUCTS(ProductName);
```

Other criterion for creating indexes on non key attributes:

- Non key attributes used for joining tables
- Non key attributes used for sorting records
- Non key attributes used to qualify a filter

You can enforce uniqueness of values in a column using a unique index.

```
CREATE UNIQUE INDEX cust_postcode_idx ON CUSTOMERS(PostalCode);
```

Are you able to create the unique index? Why?

Creating a Function-Based Index

In the previous section, you have created the index `prod_name_idx`
Let's say you issue the following query:

```
Select ProductCode, ProductName, MSRP, quantityInStock
from PRODUCTS
where UPPER(ProductName) = LIKE 'CARS%';
```

Because this query uses a function-UPPER( ), the `prod_name_idx` index isn't used. If you want an index to be based on the results of a function you must create a function-based index.

For example:

```
CREATE INDEX prod_name_idx_func_idx ON PRODUCTS (UPPER(Productname));
```

Index on Multiple Columns

If you know a group of multiple columns will be always used together as search criteria, you should create a single index for that group of columns with the "ON table_name(col1, col2, ...)"

```
CREATE INDEX customer_names ON
customers(contactlastname,contactfirstname);



select customernumber
from customers
where contactfirstname like 'Ma%' and
      contactlastname like 'De%';
```

C. Views

Creating and Using a View

There are two basic types of views;

1 Simple views, which contain a subquery that retrieves from one base table
2 Complex views, which contain a subquery that;
   - Retrieves from multiple base tables
   - Groups rows using a GROUP BY or DISTINCT clause
   - Contains a function call

Creating and Using Simple Views

Let's say that you only allow employee Steven to view detail about customers from London.

```
   CREATE VIEW London_customers_view AS
       SELECT *
       FROM customers
       WHERE UPPER(city) = 'LONDON';
```

Query the view:

```
 SELECT CustomerNumber, customerName, ContactLastName, Phone
       FROM London_customers_view;
```

Check the structure of the view: DESC `London_customers_view`

Performing an INSERT Using a View

Note: You can only perform DML operations with simple views. Complex views don't support DML.

Try to insert some records into the `London_customers_view` created above.

Create a view that show products that are low in stock (you decide what is "low stock")

Insert a record into Products that has quantity lower than your "low stock" threshold.

Insert a record into Products that has quantity above your "low stock" threshold.

View the data you've just inserted into the "low stock" view. Explain why there is no output.

Creating a View with a CHECK OPTION Constraint

You can specify that DML operations on a view must satisfy the subquery by adding a CHECK OPTION constraint to the view.

```
CREATE VIEW low_stock2 AS
     SELECT *
     FROM products
     WHERE quantityInStock < 50
     WITH CHECK OPTION CONSTRAINT low_stock2_50qty;
```

Insert a row into low_stock2 with a `QuantityInStock` of 139 (more than 50).

```
 insert into low_stock2 values('TEST1234','Pont Yacht', 'Ships',
     '1:72', 'Unimax Art Galleries', 'NULL',139,33.3,54.6);
```

What error message did you see?

Creating a View with a READ ONLY Constraint

You can make a view read only by adding a READ ONLY constraint.

```
CREATE VIEW low_stock3 AS
SELECT *
FROM products
WHERE quantityInStock<50
WITH READ ONLY CONSTRAINT low_stock3_readOnly;
```

Insert a row into low_stock3 with a `QuantityInStock` of 10 (less than 50)
What error massage did you get?

Getting information on View Constraints

```
Select OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE, TABLE_NAME, STATUS
    from user_constraints
    order by owner, table_name;
```

Creating and Using Complex Views

```
CREATE OR REPLACE VIEW HIGH_ORDER_VIEW AS
   select A.customerNumber, A.orderNumber, orderDate,
   sum(B.priceEach*B.quantityOrdered) AS "Order Value",
   count(B.productCode) as "No. of Products"
   from orders A, orderDetails B
   where  A.orderNumber = b.orderNumber
   group by A.customerNumber, A.orderNumber, orderDate
   having sum(B.priceEach*B.quantityOrdered)>5000;
```

Combining a view with a database table

```
select customerName, country, B.*
   from customers A, high_order_view B
   where A.customerNumber = b.customerNumber;
```

Exercises

Create views for:

1. All orders for last year (choose relevant details). Query this view to show total orders by month.

2. Total values of each product sold to-date. Query this view to list the Top 10 least popular product.

3. Number of employees in each territory. Query this view to show the distribution of employees across territories.