**Practical 7:**

1. **Querying Oracle metadata – the data dictionary**
2. **Indexes**
3. **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/

1. Oracle Database SQL Language Reference

https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlrf/sql-language-reference.pdf

1. **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.

1. **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;  CREATE OR REPLACE VIEW HIGH\_ORDER\_VIEW (CustomerNumber, OrderNumber,  OrderDate, OrderValue, TotalNoOfItems) AS  select A.customerNumber, A.orderNumber, orderDate,  sum(B.priceEach\*B.quantityOrdered) ,  sum(quantityOrdered)  from orders A, orderDetails B  where A.orderNumber = b.orderNumber  group by A.customerNumber, A.orderNumber, orderDate  having sum(B.priceEach\*B.quantityOrdered) > 50000; |

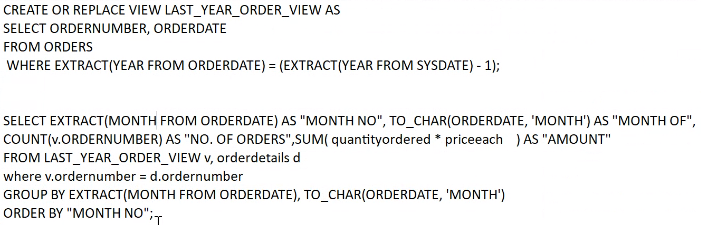
**Combining a view with a database table**

|  |
| --- |
| 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.



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

CREATE OR REPLACE VIEW TOP\_PRODUCT\_VIEW AS

SELECT PRODUCTCODE, SUM(QUANTITYORDERED) AS "TOTAL UNITS", SUM(QUANTITYORDERED \* PRICEEACH) AS "TOTAL SALES"

FROM ORDERDETAILS

GROUP BY PRODUCTCODE

ORDER BY "TOTAL SALES"

FETCH FIRST 10 ROWS ONLY;

SELECT A.PRODUCTCODE, B.PRODUCTNAME, A."TOTAL UNITS", A."TOTAL SALES"

FROM TOP\_PRODUCT\_VIEW A, PRODUCTS B

WHERE A.PRODUCTCODE = B.PRODUCTCODE;

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

CREATE OR REPLACE VIEW EMPLOYEE\_VIEW AS

SELECT OFFICECODE, COUNT(EMPLOYEENUMBER) AS "NO OF EMPLOYEE"

FROM EMPLOYEES

GROUP BY OFFICECODE;

SELECT O.COUNTRY, O.TERRITORY, E."NO OF EMPLOYEE"

FROM EMPLOYEE\_VIEW E, OFFICES O

WHERE E.OFFICECODE = O.OFFICECODE

ORDER BY O.OFFICECODE;