Lab Sheet

NOTE: Go through all the concepts in sheet and whatever possible practices those commands.

You must also explore yourself SQL Date Format and Functions from SQL manuals

---

OBJECTIVES:

- Oracle Dates and Times
- Views
- SAVEPOINTS, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT
- Grant/Revoke Privileges

➢ Oracle Dates and Times (Go through the manuals of Oracle/SQL for further details and formats)

Overview: Oracle supports both date and time, albeit differently from the SQL2 standard. Rather than using two separate entities, date and time, Oracle only uses one, DATE. The DATE type is stored in a special internal format that includes not just the month, day, and year, but also the hour, minute, and second.

The DATE type is used in the same way as other built-in types such as INT. For example, the following SQL statement creates a relation with an attribute of type DATE:

create table dobs{name Varchar2(20), dob date};

DATE Format:

When a DATE value is displayed, Oracle must first convert that value from the special internal format to a printable string. The conversion is done by a function TO_CHAR, according to a DATE *format*. Oracle's default format for DATE is "DD-MON-YY". Therefore, when you issue the query

select dob from dobs;

Output will be something like:

```
DOB
---------
01-APR-98
```

Whenever a DATE value is displayed, Oracle will call **TO_CHAR** automatically with the **default DATE format**. However, you may override the default behaviour by calling TO_CHAR explicitly with your own DATE format. For example,

SELECT TO_CHAR(dob, 'YYYY/MM/DD') AS newdob FROM dobs;

Output result will be :

```
NEWDOB
----------
1998/04/01
```

Usage of TO_CHAR : **TO_CHAR(<date>,'<format>')**

here <format> string can be formed from over 40 options. Some of the more popular ones include:

MM	Numeric month (*e.g.*, 07)
MON	Abbreviated month name (*e.g.*, JUL)
MONTH	Full month name (*e.g.*, JULY)
DD	Day of month (*e.g.*, 24)
DY	Abbreviated name of day (*e.g.*, FRI)
YYYY	4-digit year (*e.g.*, 1998)
YY	Last 2 digits of the year (*e.g.*, 98)
RR	Like YY, but the two digits are ``rounded'' to a year in the range 1950 to 2049. Thus, 06 is considered 2006 instead of 1906
AM (or PM)	Meridian indicator
HH	Hour of day (1-12)
HH24	Hour of day (0-23)
MI	Minute (0-59)
SS	Second (0-59)

You have just learned how to output a DATE value using TO_CHAR. Now what about inputting a DATE value? This is done through a function called TO_DATE, which converts a string to a DATE value, again according to the DATE format. Normally, you do not have to call TO_DATE explicitly: Whenever Oracle expects a DATE value, it will automatically convert your input string using TO_DATE according to the default DATE format "DD-MON-YY". For example, to insert a tuple with a DATE attribute, you can simply type:

insert into dobs values(Shyam, '31-may-98');

Alternatively, you may use TO_DATE explicitly:

insert into dobs values(Ram, to_date('1998/05/31:12:00:00AM', 'yyyy/mm/dd:hh:mi:ssam'));

The general usage of TO_DATE is:

TO_DATE(<string>, '<format>') where the <format> string has the same options as in TO_CHAR.

Finally, you can change the default DATE format of Oracle from "DD-MON-YY" to something you like by issuing the following command in sqlplus:

alter session set NLS_DATE_FORMAT='<my_format>';

The change is only valid for the current sqlplus session.

Current Time

The built-in function SYSDATE returns a DATE value containing the current date and time on your system. For example,

select to_char(sysdate, 'Dy DD-Mon-YYYY HH24:MI:SS') as "Current Time" from dual;

Output will be:

```
Current Time
--------------------
Tue 2-Apr-2013 07:01:27
```

which is the time when I was preparing this document :-) Two interesting things to note here:

- You can use double quotes to make names case sensitive (by default, SQL is case insensitive), or to force spaces into names. Oracle will treat everything inside the double quotes literally as a single name. In this example, if "Current Time" is not quoted, it would have been interpreted as *two* case insensitive names CURRENT and TIME, which would actually cause a syntax error.
- DUAL is built-in relation in Oracle which serves as a dummy relation to put in the FROM clause when nothing else is appropriate. For example, try "select 1+2 from dual;".

Another name for the built-in function SYSDATE is CURRENT_DATE. Be aware of these special names to avoid name conflicts.

Operations on DATE

You can compare DATE values using the standard comparison operators such as =, !=, >, *etc.*
You can subtract two DATE values, and the result is a FLOAT which is the number of days between the two DATE values. In general, the result may contain a fraction because DATE also has a time component. For obvious reasons, adding, multiplying, and dividing two DATE values are not allowed. You can add and subtract constants to and from a DATE value, and these numbers will be interpreted as numbers of days. For example, SYSDATE+1 will be tomorrow. You cannot multiply or divide DATE values.
With the help of TO_CHAR, string operations can be used on DATE values as well. For example, to_char(<date>, 'DD-MON-YY') like '%JUN%' evaluates to true if <date> is in June.

Execute the following queries : ***SELECT LEAST("23-JAN-2013","24-DEC-2006") FROM DUAL;***

There is an apparent error in this query. Try to resolve it and use the ***GREATEST*** function.

MONTHS_BETWEEN returns the number of months between two dates.

SELECT MONTHS_BETWEEN(TO_DATE('23-JAN-2006',"dd-mon-yyyy"),SYSDATE) FROM DUAL ;

Use the ***ROUND*** and ***TRUNC*** function to round or truncate the results.

ADD_MONTHS allows you to offset the months by a particular numbers

SELECT ADD_MONTHS(sysdate,3) FROM Dual;

LAST_DAY returns the last day of the month. Try the following query:

SELECT LAST_DAY(SYSDATE)+1 FROM DUAL;

For Further details on the date formats, you can refer to your Lab sheet 1.

VIEWS:

- After a table is created and populated with data, it may become necessary to prevent all users from accessing all columns of a table, for data security reasons. This would mean creating several tables having the appropriate number of columns and assigning specific users to each table, as required.

To reduce **redundant data** to the minimum possible, Oracle allows the creation of an object called a **View.** A View is mapped, to a SELECT sentence. The table on which the View is based is described in the FROM clause of the SELECT statement. The SELECT clause consists of a sub-set of the columns of the table. Thus a View, which is mapped to table, will in effect have a sub-set of the actual columns of the table from which it is built. This technique offers a simple, effective way of hiding columns of a table.

The reasons why views are created are:

- When Data Security is required
- When Data redundancy is to be kept to the minimum while maintaining data security

Creating View:

SYNTAX: create view <view name> as

select <columnname1>, <columnname2>…….. from <tablename> where <columnname> = <expression> group by <grouping criteria> having <predicate>

Note: order by clause cannot be used while creating a view)

You have created the student table in labs. So now we are applying view on the student table.

Example: **create view vw_student as select * from student;**

Renaming the columns of a View: column of the view can take different names from the table columns, ifrequired

Example: we change the idno to identity no and name column to student name in the View.

```
create view vw1_student as
        select idno "identityno", name "studentname" from student;
```

Check the contents of this view.

```
DESC vw1_student;
```

Selecting a data from a View: After a view has been created, it can be queried exactly like a base table.

select <columnname1>, <columnname2> …. from viewname;

Example: select idno, name from vw_student;

NOTE: Instead of a **table name** in the FROM clause, a **view name** is used. The SELECT statement can have the all clauses like WHERE, ORDER BY etc.

Updateable Views: View can be also used for data manipulation (i.e. user can perform insert, update, and delete operations). Views on which data manipulation can be done are called **Updateable Views.** When an **updateable view name** is given in an Insert Update, or Delete SQL statement, modification to data in the view will be immediately passed to the underlying table.

For a view to be updateable, it should meet the following criteria:

Views defined from a single table.

If the user wants to insert records with the help of view then the Primary key column(s) and all the not null columns must be included in the view.

The user can update, delete records with the help of a view even if the Primary key column and Not Null column(s) are excluded from the view definition

When an INSERT option is performed using the View:

```
insert into vw_student values('2004C6PS123','rohit','IS','BD',123,8);
```

When a MODIFY operation is performed using the View:

```
update vw_student   set name='mohit'    where name='rohit';
```

When a DELETE operation is performed using the view:

```
delete from vw_student   where name='mohit';
```

View Defined From Multiple Tables:

If view is created from multiple tables, which were created using **a Referencing Clause** (i.e. a logical linkage exists between the tables), then through primary key column(s) as well as the not null column, are included in the view definition, the view 's behavior will be as follows:

An **INSERT** operation is not allowed

The **DELETE** or **MODIFY** operations **do not** affect the master table

The view can be used to **MODIFY** the columns of the detail table included in the view.

If a DELETE operation is executed on the view, the corresponding records from the detail table will be deleted.

EX:

Table name: **DEPT** Column name	Data type	Size	Attributes
deptname	Varchar2	15	
deptno	Number	2	Primary key
addr	Varchar2	20	
city	Varchar2	15	

Table name: **EMP** Column name	Data type	Size	Attributes
eid	Varchar2	10	Primary key
ename	Varchar2	20	
salary	Number	5	
deptno	Number	2	Foreign key references deptno of the DEPT

SYNTAX for creating a Master View :

```
create view vw_emp as  select eid, ename, salary, emp.deptno, deptname, addr, city
                from EMP, DEPT
                        where EMP.deptno= DEPT.deptno;
```

Note: In select statement we are using emp.deptno because of deptno column is exist in both the tables. If you simply give deptno oracle shows an error: column ambiguously defined. Try it yourself.

When an INSERT option is performed using the View :

```
insert  into  vw_emp  values('2008CH1234','ramesh',10000,10,'CSIS','BITS','PILANI');
```

Oracle gives the following error message:

ERROR at line 1:

ORA-01776: cannot modify more than one base table through a join view

When a MODIFY operation is performed using the view

```
update vw_emp set ename ='mohit' where deptno=10;
```

It will causes the row to be updated successfully.

When a DELETE operation is performed using the View

```
delete from vw_emp where deptno=10;
```

Destroying a View: The DROP VIEW command is used to remove a view from the business.

Syntax: drop view <view name>

Example: drop view vw1_student;

SAVEPOINTS, ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT

The SAVEPOINT statement sets a named transaction savepoint with a name of ***identifier***. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

ROLLBACK TO SAVEPOINT:

statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does *not* release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the ROLLBACK TO SAVEPOINT: statement returns the following error, it means that no savepoint with the specified name exists:

ERROR 1305 (42000): SAVEPOINT ***identifier*** does not exist

RELEASE SAVEPOINT : statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist. All savepoints of the current transaction are deleted if you execute a COMMIT, or a ROLLBACK that does not name a savepoint.

A new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

Example:

```
insert into emp (empno,ename,sal) values (109,'Sami',3000);
savepoint a;

insert into emp (empno,ename,sal) values (110,'Mounika',3001);
savepoint b;
insert into emp (empno,ename,sal) values (111,'Kashyap', 3002);
```

Now if you give

rollback to a;

The second row will not be present in the table.

RELEASE SAVEPOINTa; *it will release the savepoint a.*

If you give

commit; Then the whole transaction is committed and all savepoints are removed.

Note: **For more suitable examples must refer link@**
https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/sqloperations.htm#BABGAAIG

Oracle/PLSQL: Grant/Revoke Privileges

Grant Privileges on Tables:

You can grant users various privileges to tables. These privileges can be any combination of select, insert, update, delete, references, alter, and index. Below is an explanation of what each privilege means.

Privilege	Description
Select	Ability to query the table with a select statement.
Insert	Ability to add new rows to the table with the insert statement.
Update	Ability to update rows in the table with the update statement.
Delete	Ability to delete rows from the table with the delete statement.
References	Ability to create a constraint that refers to the table.
Alter	Ability to change the table definition with the alter table statement.
Index	Ability to create an index on the table with the create index statement.

Syntax for granting privileges on a table is:

grant privileges on object to user;

Example: if you wanted to grant select, insert, update, and delete privileges on a table called suppliers to a user name smithj, you would execute the following statement:

grant select, insert, update, delete on suppliers to smithj;

You can also use the all keyword to indicate that you wish all permissions to be granted. For example:

grant all on suppliers to smithj;

If you wanted to grant select access on your table to all users, you could grant the privileges to the public keyword. For example:

grant select on suppliers to public;

Revoke Privileges on Tables:

Once you have granted privileges, you may need to revoke some or all of these privileges. To do this, you can execute a revoke command. You can revoke any combination of select, insert, update, delete, references, alter, and index.

syntax for revoking privileges on a table is:

revoke privileges on object from user;

Example: if you wanted to revoke delete privileges on a table called suppliers from a user named anderson, you would execute the following statement:

revoke delete on suppliers from anderson;

If you wanted to revoke all privileges on a table, you could use the all keyword. For example:

revoke all on suppliers from anderson;

If you had granted privileges to public (all users) and you wanted to revoke these privileges, you could execute the following statement:

revoke all on suppliers from public;

Grant Privileges on Functions/Procedures: When dealing with functions and procedures, you can grant users the ability to execute these functions and procedures. Execute privilege is explained below:

Privilege	Description
Execute	Ability to compile the function/procedure. Ability to execute the function/procedure directly.

syntax for granting execute privileges on a function/procedure is:

grant execute on object to user;

Example: if you had a function called Find_Value and you wanted to grant execute access to the user named smithj, you would execute the following statement:

grant execute on Find_Value to smithj;

If you wanted to grant all users the ability to execute this function, you would execute the following:

grant execute on Find_Value to public;

Revoke Privileges on Functions/Procedures:

Once you have granted execute privileges on a function or procedure, you may need to revoke these privileges from a user. To do this, you can execute a revoke command.

Syntax for the revoking privileges on a function or procedure is:

revoke execute on object from user;

If you wanted to revoke execute privileges on a function called Find_Value from a user named anderson, you would execute the following statement:

revoke execute on Find_Value from anderson;

If you had granted privileges to public (all users) and you wanted to revoke these privileges, you could execute the following statement:

revoke execute on Find_Value from public;