Table of Contents

1. Views

1.1. Reason for views

Suppose you have a fairly complex query dealing with customer orders that you need to run often. The query joins several tables and uses calculated columns etc. When you run this query, you filter for different order date ranges but the rest of the query is the same. You have asked the dba for help getting the query to be efficient and you do not want to keep writing the query over each time you need to run one of these reports. What you can do is create a database object called a view which encapsulates the complex query but still lets you add a filter for the order dates when you run that query. When you create your query, you can use the view in the From clause and not worry about the details of the join and calculated columns (not worrying about the details is a good thing!). You could also make the view available to programmers who may not have the experience to write complex queries.

A view is essentially a named, saved Select query. The view does not store any data; instead it stores the directions (the query) for getting the data. Once the view is created, you can use it as the table source for other queries. The definition of the view is stored on the server and is retrieved when the view is used. Each time you use the view, it goes to the actual base tables and retrieves the current data. The view metadata is stored in the data dictionary.

There are a few common reasons for using views

- to create a virtual table that presents a simplified set of column for queries. The view code might include joins and filters and calculated columns that the user of the view does not have to deal with directly
- to separate the logical database structure from the physical structure. The physical structure includes the objects create with the Create Table statement. The logical structure includes those tables and also the views. That way different users can have a different logical structure that reflects the user's needs.
- to improve security by letting a user see only certain columns and specified rows of a table.

1.2. Example of creating and using a view

Demo 01: We will use a fairly simple query for our view. When you work on a view, work out the details of the select statement first.

```
create or replace view  custReportGoodCredit_01 as
select customer_id as CustomerID
,  customer_name_first ||  ' ' ||  customer_name_last as CustomerName
from  cust_customers
where customer_credit_limit > 3000
with read only;
```

Demo 02: Using the view

```
select CustomerName, CustomerID
from custReportGoodCredit_01
order by CustomerID;
```

```
CUSTOMERNAME                                       CUSTOMERID
-------------------------------------------------- ----------
Arnold McGold                                          400300
Sally Williams                                         403000
Elisha Otis                                            403010
Alexis Hamilton                                        403050
James Stevenson                                        403100
. . .  rows omitted
```

Note that the person using this view cannot see the credit limit value because it is not exposed by the view.

Data displayed through a view will reflect the current data in the table at the time the query was run; not the data at the time the view was defined. A person using a view as the table expression might not be aware that this is a view and not a base table.

Demo 03: Creating a view that concatenates the customer names and calculates the linetotal

```
create or replace view  ordreport_01 as
select
   order_id      as orderid
,  order_date    as orderdate
,  customer_id     as customerid
,  customer_name_first || ' ' ||  customer_name_last as customername
,  prod_id     as itempurchased
,  prod_name   as itemdescription
,  quoted_price * quantity_ordered as linetotal
from cust_customers
join oe_orderHeaders  using (customer_id)
join oe_orderDetails using(order_id)
join prd_products using(prod_id)
where quoted_price > 0 and quantity_ordered > 0 with read only;
```

Demo 04: Describing the view. One thing to check is that the data types of the columns in the view are appropriate. Do not over format the columns you wish to filter; do not output a formatted number column if you might wish to filter on it. Remember that when you format a numeric column with To_Char you are returning a string- not a number.

```
Desc ordReport_01
```

```
Name                              Null?    Type
 --------------------------------- -------- --------------------
 ORDERID                           NOT NULL NUMBER(38)
 ORDERDATE                         NOT NULL DATE
 CUSTOMERID                        NOT NULL NUMBER(38)
 CUSTOMERNAME                               VARCHAR2(51)
 ITEMPURCHASED                     NOT NULL NUMBER(38)
 ITEMDESCRIPTION                   NOT NULL VARCHAR2(25)
 LINETOTAL                                  NUMBER
```

Demo 05: Using the view this is a lot easier than repeating that join for each query

```
select orderid, orderdate, itempurchased, linetotal
from ordreport_01;
```

```
   ORDERID ORDERDATE ITEMPURCHASED  LINETOTAL
---------- --------- ------------- ----------
       105 01-OCT-15          1020      155.4
```

```
       105 01-OCT-15          1030        300
       105 01-OCT-15          1010        750
       106 01-OCT-15          1060     255.95
       107 02-OCT-15          1110      49.99
. . .  rows omitted
```

Demo 06: Using the view with a filter

```
Select
  orderid
, orderdate
, itempurchased
From ordreport_01
Where itempurchased in ( 1071, 1072)
order by orderdate;
```

```
   ORDERID ORDERDATE ITEMPURCHASED
---------- --------- -------------
       411 01-AUG-15          1071
       411 01-AUG-15          1072
       408 20-NOV-15          1071
      2122 23-JAN-16          1071
      2122 23-JAN-16          1072
      4510 01-FEB-16          1071
      4510 01-FEB-16          1072
```

Demo 07: Alternate syntax when creating a view- this defines the column names in the view header rather than in the select

```
create or replace view  ordreport_02(orderid, orderdate, customerid, linetotal)
as
select  order_id,  order_date,  customer_id
,  quoted_price * quantity_ordered
from oe_orderheaders
join oe_orderdetails using(order_id)
join prd_products using(prod_id)
where quoted_price > 0 and quantity_ordered > 0
order by order_id
with read only;
```

1.3. View versus CTE or subqueries

You may have noticed a similarity between CTE and views. A view is stored in the database as a persistent database object; a CTE exists for the lifetime of the query. So if you have a result set that you want recalculated each time you use it and you would use that result set often or you want other users to use that result set, then a View is a good idea. If you want to break a complex task into pieces to solve one step at a time and the task is unique and it is not sometime you want for other tasks, then a CTE is a good idea.

A production database will often have a large number of views set up for business purposes but you don't want to create a view that is used only once.

2. Some view syntax rules

A view is an object created in a specific database.

- You can use either the Create View syntax or the Create or Replace View syntax. If the name you select for the view is already used by another view, the Create view syntax will report an error; the Create or Replace View syntax will replace the current definition of the view.

- You cannot have a view with the same name as a table.
- You cannot create a view on a table that does not exist yet unless you include the word Force. You cannot use the view until the table is created, but this keyword will let you define the view.
- Since a view is a stored object, you can drop the view with a drop view command.

```
drop view my_view;
```

- Your user account has to have permissions to create views. Your CCSF Oracle account has that permission.
- You can assign new names for the columns when you define a view or you can use the names in the underlying tables.
- If you have a calculated column in the underlying select, you must provide an alias for that column.
- If you define the columns in the view header, the number of columns in the view definition must match the number of columns in the select used to define the view.
- When you create an alias within a view, the alias should be a single word and not enclosed in quotes
- Do not create a view using Select *; the view will be stored with the * expanded into the column names at the time the view was created and will not be updated if the underlying tables are changed.
- Oracle allows an Order By clause in the definition of a view

3. Updating with Views

Some restrictions for updating (any change) though a view. We will discuss some of these options in this document. If you think about these rules and the need for the dbms to understand exactly what data to change, these rules make more sense.

- If the view is based on all of the columns of a single table and the view was not created with a read-only option or a with check option, then you can update through the view.
- A read only option means that the view cannot be used for updates
- A With Check option means that you cannot do an update that would violate the check option.
- You cannot update a column in a view which is based on a calculated column. If the view has several columns and only some columns are calculated, you can update the other columns
- If you have any calculated columns, then you cannot insert via the view. However you might be able to delete via the view.
- You cannot update views that are defined with aggregate functions or Group By
- You cannot update views that are defined with Distinct or rownum
- You cannot update a view created with a set operation such as Union

- If the view is based on multiple tables, you cannot do an update on a column which maps to a non key-preserved table. (Essentially this means that if the view definition does not contain the pk – as a pk for the view set, then the view cannot be updated. So if we had a parent and child situation and the view definition includes the pk of the parent we would, in general, be able to update the parent. But we could not update the child even with the child table pk since it could occur more than once in the view rowset)

Demo 08: In order to not change the data in the regular table, insert the following rows for testing.

An Insert for the employees table

```
delete from emp_employees where  emp_id = 1995;
insert into emp_employees  values
   (1995,'Zahn', 'Joe', '111111111', 100, 10, date'2009-09-09', 500, 2);
```

An Insert for the products table

```
delete from  prd_products where  prod_id = 1995;
insert into prd_products values
(1995, 'book','Train your cat book',29.95, 'PET');
```

3.1. Determining which columns in a view can be changed

Demo 09: What is the structure of emp_employees?

```
desc emp_employees
```

```
Name                      Null?    Type
 ------------------------ -------- -------------
 EMP_ID                    NOT NULL NUMBER(38)
 NAME_LAST                 NOT NULL VARCHAR2(25)
 NAME_FIRST                         VARCHAR2(25)
 SSN                       NOT NULL CHAR(9)
 EMP_MNG                            NUMBER(38)
 DEPT_ID                   NOT NULL NUMBER(38)
 HIRE_DATE                 NOT NULL DATE
 SALARY                             NUMBER(8,2)
 JOB_ID                    NOT NULL NUMBER(38)
```

Demo 10: Simple, single table view. This view can be used for updates of the employee name and the id

```
CREATE OR REPLACE VIEW vw_Emp AS
    select emp_id, name_last, name_first
    from emp_employees;
```

Demo 11: Which columns can be changed? Use the data dictionary view user_updatable_columns

```
select column_name, updatable
from user_updatable_columns
where table_name = 'VW_EMP';
```

```
COLUMN_NAME                    UPDATABLE
------------------------------ ---------
EMP_ID                         YES
NAME_LAST                      YES
NAME_FIRST                     YES
```

Discussion: This update works.

```
Update vw_emp set name_first = 'Susan' where emp_id = 1995;
```

This one does not work; the attribute salary is not accessible through the view.

```
Update vw_emp set salary = 12500 where emp_id = 1995;
```

```
ORA-00904: "SALARY": invalid identifier
```

This update does not work; the attribute name_last is accessible through the view, but that attribute is not nullable in the base table. The table constraints must always be met.

```
Update vw_emp set name_last = null where emp_id = 1995;
```

```
ORA-01407: cannot update ("ROSE151A"."EMP_EMPLOYEES"."NAME_LAST") to NULL
```

Can you do an insert via this view? No- there are several attributes in the table which are not nullable and not accessible via the view.

Can you do a delete via this view? Yes, unless there is another constraint that would disallow the delete.

```
Delete from vw_emp where emp_id = 1995;
```

If you have removed or changed that row, reinsert it.

Demo 12: This is a view that shows summary data

```
CREATE OR REPLACE VIEW vw_HighPriceByCategory AS
    select Catg_id, MAX(prod_list_price) As HighPrice
    from prd_products
    group by Catg_id;
```

Which columns can be changed? You cannot change data in a view with a group by.

```
select column_name, updatable
from user_updatable_columns
where table_name = 'VW_HIGHPRICEBYCATEGORY';
```

```
COLUMN_NAME                    UPDATABLE
------------------------------ ---------
CATG_ID                        NO
HIGHPRICE                      NO
```

If we try to change data through the view , we get an error because the attribute HighPrice does not tie back to a specific row in the underlying table

```
update  vw_HighPriceByCategory
set     HighPrice = 250
where   Catg_id = 'PET';
```

```
ORA-01732: data manipulation operation not legal on this view
```

Demo 13: This is a view that includes a calculated column

```
CREATE OR REPLACE VIEW vw_NewPrice AS
    select Catg_id, prod_name, Round(prod_list_price * 1.05,2) As NewPrice
    from prd_products;
```

Some of these columns can be updated but we cannot update the calculated column.

```
select column_name, updatable
from user_updatable_columns
where table_name = 'VW_NEWPRICE';
```

```
COLUMN_NAME                    UPDATABLE
------------------------------ ---------
CATG_ID                        YES
PROD_NAME                      YES
NEWPRICE                       NO
```

Trying to update the calculated column is blocked.

```
Update vw_NewPrice
set NewPrice = 45
where prod_name = 'Train your cat book';
```

```
ORA-01733: virtual column not allowed here
```

Demo 14: This is another view that includes a calculated column

```
CREATE OR REPLACE VIEW vw_NewPrice2 AS
    select Catg_id, prod_name, prod_list_price * 1 As NewPrice
    from prd_products;
```

If you think about this mathematically, prod_list_price * 1 does not really change the value, but this is still a calculated column and cannot be updated via the view.

```
Update vw_NewPrice2
set NewPrice = 45
where prod_name = 'Train your cat book';
```

```
ORA-01733: virtual column not allowed here
```

3.2. Updatable view using a join

Create a view that joins the department table (the parent) and the employee table (the child). Can we update columns in the parent? in the child? There are several different ways to do the join. The next three views use different join techniques

Demo 15: Creating the join with the USING clause.

```
CREATE OR REPLACE VIEW vw_em_dept_1 AS
    select  emp_id, name_last
    ,       dept_id
    ,       dept_name
    from    emp_employees join emp_departments using(dept_id);
```

Demo 16: Creating the join with the ON clause including the dept_id column from the child table.

```
CREATE OR REPLACE VIEW vw_em_dept_2 AS
    select  emp_id, name_last
    ,       emp_employees.dept_id
    ,       dept_name
    from    emp_employees
    join    emp_departments
    on      emp_employees.dept_id = emp_departments.dept_id;
```

Demo 17: Creating the join with the ON clause including the dept_id column from the parent table.

```
CREATE OR REPLACE VIEW vw_em_dept_3 AS
    select   emp_id, name_last
    ,        emp_departments.dept_id
    ,        dept_name
    from     emp_employees
    join     emp_departments
    on       emp_employees.dept_id = emp_departments.dept_id;
```

If you do a Select * for any of these three views you get the same result set.

Demo 18: But there is a difference. Which columns are updatable ? (The Break command is used to make the output easier to read)

```
BREAK ON table_name SKIP 1

select table_name, column_name, updatable
from user_updatable_columns
where table_name LIKE 'VW_EM_DEPT_%' ;
```

```
TABLE_NAME          COLUMN_NAME         UPDATABLE
------------------- ------------------- ---------
VW_EM_DEPT_1        EMP_ID              YES
                    NAME_LAST           YES
                    DEPT_ID             NO
                    DEPT_NAME           NO

VW_EM_DEPT_2        EMP_ID              YES
                    NAME_LAST           YES
                    DEPT_ID             YES
                    DEPT_NAME           NO

VW_EM_DEPT_3        EMP_ID              YES
                    NAME_LAST           YES
                    DEPT_ID             NO
                    DEPT_NAME           NO
```

(This is rather complex- I mostly want you to understand that if a view is based on a join of tables, updating gets more complex.)
In all of these we can update the Emp_id and name_last which come from the child table. For each Emp_id or Emp_name, that value ties back to a single definable row in the employees table. There is only one row in the employees table for each row in the view results table. (the term 'key-preserved" is used for a base table that

has a 1-1 relationship with the rows in the views- therefore a row in the view can be unambiguously tied to a row in that base table.)We cannot update the dept_name which comes from the parent table; there are multiple rows in the result set for a row in the departments table.

The dept_id can be updated in the second view- where that attribute in the view comes uniquely from the child table. The dept_id cannot be updated in the third view- where that attribute in the view comes from the parent table and is not unique. The dept_id cannot be updated in the first view- where that attribute in the view comes from the new generated column and not from either table.

Demo 19: An attempt to update the dept_id using the third view. This fails.

```
update vw_em_dept_3
set dept_id = 80
where emp_id = 1995;
```

```
set     dept_id = 80
         *
ERROR at line 2:
ORA-01779: cannot modify a column which maps to a non key-preserved table
```

Discussion: We can do the update via the second view. In that case we are changing the dept id for child table- (the employee table). So this changes a single row. If we try to do the change via the third view, we would be changing the dept_id in the department table- so presumably we would be changing dept_id 10 to 80 in the department table- that would result in two rows in the department table with ID 80. Allowing this type of change would create problems. (Again, if you did the update, do a rollback.)

4. Views that are more limiting

There are some options that can be used in the creation of a view that provide protection of the data in the base table.

4.1. Declaring the view to be read-only

Demo 20: Declaring that the view is read only. None of the attributes are updatable.

```
CREATE OR REPLACE VIEW ProductList AS
    select prod_id, prod_list_price, prod_name, prod_desc
    from prd_products
WITH READ ONLY;
```

4.2. Making some columns not updatable

Demo 21: The list price and the description can be updated but not the id or name. because these are calculated columns

```
CREATE OR REPLACE VIEW ProductList2 AS
    select   prod_id + 0 as prod_id
           , prod_list_price as prod_list_price
           , prod_name || '' as prod_name
           , prod_desc
    from     prd_products;

select column_name,updatable
from user_updatable_columns
where table_name = 'PRODUCTLIST2';
```

```
COLUMN_NAME                    UPD
------------------------------ ---
PROD_ID                        NO
PROD_LIST_PRICE                YES
```

```
PROD_NAME                      NO
PROD_DESC                      YES
```

4.3. Updates with check constraints

The next few SQL statements show the purpose of the With Check Option clause. Suppose I create a view that has a WHERE clause. I can use the view to update the data in such a way that the data no longer satisfies that WHERE clause. The With Check Option clause restricts this behavior.

Demo 22: Creating a view with a check constraint

```
CREATE OR REPLACE VIEW EmpDept80 AS
    select   emp_id, name_last
    ,        dept_id
    from     emp_employees
    where    dept_id = 80
WITH CHECK OPTION;
```

Which columns can be changed?

```
COLUMN_NAME                    UPD
------------------------------ ---
EMP_ID                         YES
NAME_LAST                      YES
DEPT_ID                        YES
```

Demo 23: It is OK to change the last name value through this view. (Be certain that you have employee 1995 in dept 80 before you run this one.

```
update   EmpDept80
set      name_last = 'Hiller'
where    emp_id = 1995;
```

Demo 24: It is not OK to change the department to a value that does not meet the check constraint.

```
update   EmpDept80
set      dept_id = 20
where    emp_id = 1995;
```

```
ORA-01402: view WITH CHECK OPTION WHERE  -clause violation
```

Demo 25: Creating a view with a check constraint

```
CREATE OR REPLACE VIEW EmpHighSalary AS
    select   emp_id, name_last, salary
    ,        dept_id
    from     emp_employees
    where    salary > 70000
WITH CHECK OPTION;

-- the record set
select *
from EmpHighSalary;
```

```
    EMP_ID NAME_LAST                     SALARY    DEPT_ID
---------- ------------------------- ---------- ----------
       100 King                          100000         10
       101 Koch                           98005         30
       205 Higgs                          75000         30
       162 Holme                          98000         35
       146 Partne                         88954        215
       161 Dewal                         120000        215
```

```
       206 Geitz                          88954         30
       204 King                           99090         30

8 rows selected.
```

Demo 26: We can update employee 1995 salary via the employees table and then that employee will appear via the view.

```
update emp_employees
set name_last = 'Prince',
    salary = 75000
where emp_id = 1995;

select *
from EmpHighSalary ;
```

```
    EMP_ID NAME_LAST                     SALARY    DEPT_ID
---------- ------------------------- ---------- ----------
       100 King                          100000         10
       101 Koch                           98005         30
       205 Higgs                          75000         30
       162 Holme                          98000         35
       146 Partne                         88954        215
       161 Dewal                         120000        215
       206 Geitz                          88954         30
       204 King                           99090         30
      1995 Prince                         75000         10

9 rows selected.
```

Demo 27: Updating through the view. This will do the update.

```
update EmpHighSalary
set name_last = 'Prince',
    salary = 85000
where emp_id = 1995;
```

```
1 row updated.
```

Demo 28: Updating through the view. This update will fail due to the check constraint

```
update EmpHighSalary
set name_last = 'Koch',
    salary = 10000
where emp_id = 1995;
```

```
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Similarly you cannot do an Insert via a view with a check constraint if that insert would not be displayed with the view- i.e. if the insert data does not meet the Where clause criterion.

It would be a good idea to reload the Altgeld tables at this time.