



VISTAS

DEFINICIÓN DE VISTAS

- Una vista es una tabla “lógica” basada en una tabla o en otra vista.
- Una vista no contiene datos en sí misma , sino que se podría considerar como una tabla virtual través de la cual pueden verse determinados datos de las tablas reales.
- La vista se almacena como una sentencia SELECT en el Diccionario de Datos.

Representación de una vista

Tabla
B_PERSONAS

Vista
V_PERSONAS_FISICAS

ID	TIPO PERSONA	NOMBRE	APELLIDO
1	F	Jorge Amado	Pereira González
2	F	Roberto	Abente Gómez
5	F	Alfonso	Brizuela Cuevas
10	F	Reina	Rios
12	F	Roque	Talavera
14	F	Ramon	Gauto

ID	TIPO PERSONA	NOMBRE	APELLIDO
1	F	Jorge Amado	Pereira González
2	F	Roberto	Abente Gómez
3	J	Avente Comercial	
4	J	EARG S.A.	
5	F	Alfonso	Brizuela Cuevas
6	J	Cristhian	Achucarro
7	J	Amalgama S.A.	
8	J	Atlantic S.A.E.C.A.	
9	J	Cima S.R.L.	
10	F	Reina	Rios
11	J	Emac S.R.L.	
12	F	Roque	Talavera
13	J	Sanimax	
14	F	Ramon	Gauto
15	J	Servimex	
16	J	Libreria y Papeleria NOVA	

Ventajas de las Vistas

- Restringir el acceso a la Base de datos, puesto que permite desplegar sólo una porción de la misma
- Simplificar los queries, ya que las vistas permiten que los usuarios consulten información de diferentes tablas sin que requieran conocer la complejidad de la sentencia utilizada para el efecto.
- Permite lograr la independencia de los datos con respecto a las aplicaciones

Creando una Vista: Sintaxis

- Aplicación de un subquery con la sentencia CREATE VIEW.

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW vista
  [(alias[, alias]...)]
  AS subquery
  [WITH CHECK OPTION [CONSTRAINT constraint]]
  [WITH READ ONLY [CONSTRAINT constraint]];
```

- Los subqueries pueden contener una sentencia con compleja sintaxis de SELECT.

OPCIONES DE LA SENTENCIA

Opción	Descripción de la opción
FORCE	Crea una vista sin importar que las tablas base existan o no. Las tablas base deberán existir cuando se utilice una sentencia SELECT u otra DML sobre la vista
NOFORCE	Crea la vista solamente si la tabla existe
Vista	Es el nombre de la vista
Alias	Especifica el nombre que adoptarán los campos que conforman la vista y que deben coincidir con las columnas del SELECT
Subquery	Es la sentencia SELECT. Las columnas que conforman la sentencia SELECT serán las columnas de la vista
WITH CHECK OPTION	Especifica que la vista solamente admitirá la actualización o inserción de filas que cumplan con la restricción de la misma
Constraint	Es el nombre que se le da al constraint establecido con la cláusula WITH CHECK OPTION
WITH READ ONLY	Asegura que no se admitan sentencias de tipo DML sobre la tabla

Creando una Vista: Ejemplo

- Cree la vista Emp_contabilidad que contiene solamente los empleados del área de Contabilidad:

```
CREATE VIEW EMP_CONTABILIDAD AS
SELECT E.CEDULA, E.NOMBRE||' '||E.APELLIDO
"Nombre y Apellido",
A.NOMBRE_AREA
FROM   B_POSICION_ACTUAL P, B_AREAS A,
B_EMPLEADOS E
WHERE  A.ID = P.ID_AREA
AND    E.CEDULA = P.CEDULA
AND    P.FECHA_FIN IS NULL
AND    UPPER(A.NOMBRE_AREA) = 'CONTABILIDAD' ;
```

- Despliegue datos de la vista ejecutando SELECT contra la vista.
- Describa la estructura de la vista usando la orden DESCRIBE .

Para modificar eventualmente la vista Emp_contabilidad use CREATE OR REPLACE. También tiene la opción de usar alias para cada una de las columnas.

```
CREATE OR REPLACE VIEW EMP_CONTABILIDAD
(Cédula, Nombre_apellido, Area) AS
SELECT E.CEDULA, E.NOMBRE|| ' ' || E.APELLIDO,
A.NOMBRE_AREA
FROM   B_POSICION_ACTUAL P, B_AREAS A,
B_EMPLEADOS E
WHERE  A.ID = P.ID_AREA
AND    E.CEDULA = P.CEDULA
AND    P.FECHA_FIN IS NULL
AND    UPPER(A.NOMBRE_AREA) = 'CONTABILIDAD' ;
```

Los seudónimos de la columna en la orden CREATE VIEW se listan en el mismo orden en el que están colocadas las columnas en el subquery.

Creando una Vista Compleja: Ejemplo

- Cree una vista compleja que contiene funciones de grupo para desplegar valores de dos tablas.

```
CREATE VIEW V_SALARIOS AS
SELECT MAX(S.ASIGNACION) MAXIMO, MIN(S.ASIGNACION)
MINIMO, AVG(S.ASIGNACION) PROMEDIO
FROM    B_POSICION_ACTUAL P, B_AREAS A, B_EMPLEADOS
E, B_CATEGORIAS_SALARIALES S
WHERE   A.ID = P.ID_AREA
AND     E.CEDULA = P.CEDULA
AND     S.COD_CATEGORIA = P.COD_CATEGORIA
AND     S.FECHA_FIN IS NULL
AND     P.FECHA_FIN IS NULL;
```

VISTAS SIMPLES vs VISTAS COMPLEJAS

CARACTERÍSTICA	VISTAS SIMPLES	VISTAS COMPLEJAS
Nº de Tablas	Una	Una o más
Funciones	No	Sí
Contiene grupos de datos	No	Sí
Admite sentencias DML	Sí	No siempre

No puede:	Si la vista contiene:
<ul style="list-style-type: none"> ◦ Borrar filas de una vista 	<ul style="list-style-type: none"> ◦ Funciones del grupo. ◦ Cláusula GROUP BY ◦ La cláusula DISTINCT ◦ La seudo columna ROWNUM
<ul style="list-style-type: none"> ◦ Modificar filas en una vista 	<ul style="list-style-type: none"> ◦ Funciones del grupo. ◦ Cláusula GROUP BY ◦ La cláusula DISTINCT ◦ La seudo columna ROWNUM ◦ Columnas definidas por expresiones
<ul style="list-style-type: none"> ◦ Agregar filas a una vista 	<ul style="list-style-type: none"> ◦ Funciones del grupo. ◦ Cláusula GROUP BY ◦ La cláusula DISTINCT ◦ La seudo columna ROWNUM ◦ Columnas definidas por expresiones ◦ Columnas NOT NULL en la tabla base que no están incluidas en la vista

Restricción de operaciones DML sobre la vista

```
CREATE VIEW V_DIARIO_CONTAB AS
SELECT ID, FECHA, CONCEPTO, USUARIO, FECHA_CIERRE,
ID_AREA
FROM B_DIARIO CABECERA
WHERE ID_AREA = 3
WITH READ ONLY;
```

- **READ ONLY** :Evita la utilización de sentencias DML sobre la vista. Si intenta modificar los datos a través de la vista, tendrá error.



Utilización de la restricción WITH CHECK OPTION

```
CREATE VIEW V_DIARIO_CONTAB AS
SELECT ID, FECHA, CONCEPTO, USUARIO, FECHA_CIERRE,
ID_AREA
FROM B_DIARIO CABECERA
WHERE ID_AREA = 3
WITH CHECK OPTION CONSTRAINT CHK_DIARIO_AREA;
```

- **CHECK OPTION:** Restringe los valores de la vista: Si usted intenta cambiar el id del área para cualquier fila en la vista, la declaración fallará porque viola el constraint CHECK.

Eliminando una Vista: Ejemplo

Al eliminar una vista, no se eliminan los datos, ya que como se indicó, ella no es sino una representación de otras tablas que sí existen en la BD

```
SQL> DROP VIEW V_DIARIO_CONTAB;  
View dropped.
```

CONSULTA DE LAS VISTAS CREADAS

- Las siguientes vistas pueden ser consultadas:
- **USER_VIEWS**
- **ALL_VIEWS**

Vistas Materializadas

Vistas MATERIALIZADAS

- Son objetos que contienen el resultado de un query
- El query de la vista puede aplicarse sobre tablas u otras vistas
- Las vistas materializadas permiten mejorar la velocidad de acceso de las consultas de grandes volúmenes de datos, ya que el query no se ejecuta cada vez que se consulta la vista.

Vistas MATERIALIZADAS

- o Las vistas materializadas son utilizadas para realizar resumen, cálculo, replicación y distribución de datos.
- o Se diferencian de las vistas convencionales por el hecho de que no son ‘virtuales’ como las vistas convencionales, sino que almacenan los datos resultantes del query, por tanto
 - **Los datos deben ser refrescados cuando la tabla maestra cambia**
 - **Mejoran la performance de las sentencias SQL**
 - **Consumen espacio de almacenamiento**

Vistas Materializadas- Sintaxis Simplificada

```
CREATE MATERIALIZED VIEW
[esquema.]vista_materializada
ON PREBUILT TABLE [{WITH | WITHOUT}
REDUCED PRECISION]
[USING INDEX | USING NO INDEX
[opciones de refresh]
[FOR UPDATE] [{DISABLE | ENABLE} QUERY
REWRITE] AS subquery;
```

Descripción de la sintaxis

- **Vista_materializada:** Es el nombre de la vista
- La cláusula **ON PREBUILT** permite registrar una tabla ya existente en el esquema como vista materializada (asumiendo que la tabla refleja la materialización de un subquery)
- **WITH REDUCED PRECISION:** permite pérdida de precisión
- **FOR UPDATE:** Especifica que la vista será actualizada
- **Subquery:** Es el query definido para la vista materializada

OPCIONES PARA REFRESCAR LA VISTA

```
{ REFRESH
  { { FAST | COMPLETE | FORCE }
  | ON { DEMAND | COMMIT }
  | { START WITH | NEXT } fecha
  | WITH { PRIMARY KEY | ROWID }
  | USING
    { DEFAULT [ MASTER | LOCAL ] ROLLBACK SEGMENT
    | [ MASTER | LOCAL ] ROLLBACK SEGMENT
segmento_rollback
    }
    [ DEFAULT [ MASTER | LOCAL ] ROLLBACK SEGMENT
    | [ MASTER | LOCAL ] ROLLBACK SEGMENT
segmento_rollback
  ]... | [NEVER REFRESH] }
```

Vistas MATERIALIZADAS

Opciones de refresh

- **FAST:** Aplica cambios incrementales a medida que cambia la tabla ‘master’. Los cambios son almacenados en la tabla de LOG.
- **COMPLETE:** Refresca la vista ejecutando de nuevo el query, aún cuando existan condiciones de hacer un FAST refresh.
- **FORCE:** Aplica Fast Refresh si fuera posible, o de lo contrario un refresco completo. Si no se especifica ninguna opción (FAST, COMPLETE, FORCE), entonces se aplica FORCE por default.
- **NEVER:** Indica que la vista no será refrescada

Vistas MATERIALIZADAS

Opciones de refresco

- **REFRESH ON COMMIT:** se debe producir un refresh cuando se da COMMIT a una transacción de la tabla ‘master’. Tiene la desventaja de que las operaciones DML tarden más, por lo que no se recomienda para ambientes de mucha concurrencia.
- **REFRESH ON DEMAND.** En el método ON DEMAND se deberá utilizar explícitamente uno de los procedimientos disponibles en el paquete DBMS_MVIEW. Es la opción por defecto.
- **START WITH** y **WHEN** representan la fecha inicial y el intervalo para el refresh (cuando no es ON DEMAND)

Ejemplos: Creación de una vista con LOGSs

```
CREATE MATERIALIZED VIEW LOG ON b_diario_cabecera  
WITH SEQUENCE, ROWID  
(ID, FECHA, CONCEPTO , USUARIO, FECHA_CIERRE,  
ID_AREA ) INCLUDING NEW VALUES;
```

```
CREATE MATERIALIZED VIEW LOG ON b_diario_detalle  
WITH SEQUENCE, ROWID  
(ID, NRO_LINEA, CODIGO_CTA, DEBE_HABER, IMPORTE )  
INCLUDING NEW VALUES;
```

Ejemplos: Creación de una vista con LOGSs (continuación)

```
CREATE MATERIALIZED VIEW v_diario FAST
BUILD IMMEDIATE
REFRESH FAST ON COMMIT
AS
  SELECT c.id, c.fecha, d.nro_linea,
         DECODE(debe_haber, 'D', importe, 0) DEBITO,
         DECODE(debe_haber,'C',importe, 0) CREDIT
    FROM b_diario_detalle d JOIN b_diario_cabecera c
      ON   c.id = d.id;
```

Ejemplos (sin log)

```
CREATE      MATERIALIZED VIEW v_diariosONDEMAND
BUILD DEFERRED
REFRESH COMPLETE ON DEMAND
AS
SELECT c.id, c.fecha, d.nro_linea,
       DECODE(debe_haber, 'D', importe, 0) DEBITO,
       DECODE(debe_haber,'C',importe, 0) CREDITO
  FROM b_diario_detalle d JOIN b_diario_cabecera c
    ON      c.id = d.id;
```

Para refrescar:

```
EXEC DBMS_MVIEW.REFRESH('v_diariosONDEMAND') ;
```

Refresco programado (START WITH y NEXT)

```
CREATE      MATERIALIZED VIEW v_diariosProgramado
BUILD IMMEDIATE
REFRESH START WITH TRUNC(SYSDATE) + 11/24
          NEXT NEXT_DAY(TRUNC(SYSDATE), 'LUNES') + 15/24
AS
SELECT c.id, c.fecha, d.nro_linea,
       DECODE(debe_haber, 'D', importe, 0) DEBITO,
       DECODE(debe_haber,'C',importe, 0) CREDITO
  FROM b_diario_detalle d JOIN b_diario_cabecera c
ON      c.id = d.id;
```

La vista se refrescará iniciando hoy a las 11 am, y luego cada lunes a las 15:00