"El producto más valioso que conozco es la información"

07

TALLER DE BASE DE DATOS - INF 272 SEMESTRE 2/2022

VISTAS

Una vista es un objeto. Una vista es una alternativa para mostrar datos de varias tablas; es como una tabla virtual que almacena una consulta. Los datos accesibles a través de la vista no están almacenados en la base de datos, en la base de datos se guarda la definición de la vista y no el resultado de ella.

VISTAS

Entonces, una vista almacena una consulta como un objeto para utilizarse posteriormente. Las tablas consultadas en una vista se llaman tablas base. En general, se puede dar un nombre a cualquier consulta y almacenarla como una vista.

Una vista suele llamarse también tabla virtual porque los resultados que retorna y la manera de referenciarlas es la misma que para una tabla.

VISTAS

```
create view NOMBREVISTA as  SUBCONSULTA;
create view NOMBREVISTA as SUBCONSULTA with read only;
create or replace view NOMBREVISTA  as SUBCONSULTA;
create force view NOMBREVISTA as SUBCONSULTA;
```

El contenido de una vista se muestra con un "select":

```
select * from NOMBREVISTA;
```

Para ver sus campos

```
describe NOMBREVISTA;
```

Para eliminar una vista

```
drop view nombrevista;
```

VISTAS

```
select * from user_catalog
where table_type='VIEW';

select * from user_objects
where object_type='VIEW';

select * from user_views
where view_name like ‘poner nombre de vista';
```

VISTAS

```
create view NOMBREVISTA as
  SUBCONSULTA;
El contenido de una vista se muestra con un "select":
 select * from NOMBREVISTA;
create view vista_empleados as
  select nombre,salario,nom_dep as Departamento
   from empleados join departamento
   on departamento.depnro=empleados.depnro;

   select * from vista_empleados;
```

VISTAS

Para ver el codigo Fuente del SQL que lo genero:

```
SELECT text FROM user_views
WHERE lower(view_name) = 'emp_details_view';
```

Para ver desde el Usuario administrador

```
SELECT text FROM dba_views
WHERE lower(view_name) = 'emp_details_view'
AND OWNER = 'HR'
```

VISTAS MATERIALIZADAS

Una vista materializada se define como una vista común, pero en lugar de almacenar la definición de la vista, almacena el resultado de la consulta, es decir, la materializa, como un objeto persistente en la base de datos.

```
create materialized view
NOMBREVISTAMATERIALIZADA
REFRESH COMPLETE ON DEMAND
as SUBCONSULTA;
```

Para asignar permisos si no tiene→>>

```
GRANT CREATE MATERIALIZED VIEW TO HR
```

VISTAS MATERIALIZADAS

select * from user_objects where object_type='MATERIALIZED VIEW';

select * from user_mviews;

DROP MATERIALIZED VIEW NOMBRE_VM;

DROP MATERIALIZED VIEW NOMBRE_VM PRESERVE TABLE;

No se permite realizar "insert", "update" ni "delete" en las vistas materializadas.

Para actualizar una vista materializada

EXEC DBMS_MVIEW.REFRESH ('NOMBREVISTA',ATOMIC_REFRESH=>FALSE) ;

Secuencias

Una secuencia (sequence) se emplea para generar valores enteros secuenciales únicos y asignárselos a campos numéricos; se utilizan generalmente para las claves primarias de las tablas garantizando que sus valores no se repitan.

Una secuencia es una tabla con un campo numérico en el cual se almacena un valor y cada vez que se consulta, se incrementa tal valor para la próxima consulta.

Secuencias

Sintaxis general:

```
create sequence NOMBRESECUENCIA
 start with VALORENTERO
 increment by VALORENTERO
 maxvalue VALORENTERO
 minvalue VALORENTERO
 cache NRO|nocache cycle | nocycle;
```

si no se especifica cache asume 20

Si no se especifica ninguna cláusula, excepto el nombre de la secuencia, por defecto, comenzará en 1, se incrementará en 1, el mínimo valor será 1, el máximo será 999999999999999999999999999 y "nocycle".

Secuencias

NEXTVAL (Siguiente valor de la secuencia)

CURRVAL (Valor actual de la secuencia)

EJEMPLO:

```
CREATE SEQUENCE s_coddep;

INSERT INTO EMPLEADOS
VALUES (s_coddep.nextval,'Depto PRUEBA','Ciudad Prueba');
```

Secuencias

Ver todas las secuencias que existen:

```
select * from all_sequences;
select * from all_objects;
select object_name from all_objects
where object_type='SEQUENCE';
```

Modificar los valores e una secuencia.

```
alter sequence NOMBRE_SECUENCIA;

ALTER SEQUENCE s_coddep MAXVALUE 1500;
```

Eliminar una secuencia:

```
DROP SEQUENCE nombre_secuencia;
```

TRIGGERS

Un "trigger" (disparador o desencadenador) es un bloque de código que se ejecuta automáticamente cuando ocurre algún evento (como inserción, actualización o borrado) sobre una determinada tabla (o vista); es decir, cuando se intenta modificar los datos de una tabla (o vista) asociada al disparador.

TRIGGERS

Se crean para conservar la integridad referencial y la coherencia entre los datos entre distintas tablas; para registrar los cambios que se efectúan sobre las tablas y la identidad de quien los realizó; para realizar cualquier acción cuando una tabla es modificada; etc.

Si se intenta modificar (agregar, actualizar o eliminar) datos de una tabla asociada a un disparador, el disparador se ejecuta (se dispara) en forma automática.

TRIGGERS

Hay varios tipos de trigger a partir de la versión 12C.

- Triggers tipo DML, que se activan cuando uno hace un INSERT, DELETE, UPDATE, que se activan cuando se quiere hacer alguna de estas operaciones.
- Triggers de tipo DDL, como CREATE, DROP.
- Triggers de tipo database que permiten controlar operaciones como LOGON, SHUTDOWN a la BD.

Tipos y Eventos

- Se disparan cuando se presenta algún evento.
- TIPOS → EVENTOS → FILAS AFECTADAS

TIPOS	EVENTOS	FILAS AFECTADAS
BEFORE	INSERT	STATEMENT – SE DISPARA UNA VEZ
AFTER	UPDATE	ROW - Se disparan por cada fila afectada
INSTEAD OF	DELETE	

TRIGGERS

Sintaxis general para crear un disparador:

```
create or replace trigger NOMBREDISPARADOR
 MOMENTO-- before, after
 EVENTO-- insert, update o delete
 of CAMPOS-- solo para update
 on NOMBRETABLA
 NIVEL --puede ser a nivel de sentencia (statement) o de fila
(for each row)
 when CONDICION--opcional
 begin
  CUERPO DEL DISPARADOR--sentencias
 end NOMBREDISPARADOR;
```

EJEMPLO TRIGGERS

Deseamos crear un Trigger para que cada vez que se inserte un registro en la tabla REGIONS, almacene en una de tabla de LOGS los datos de la operación que se hizo y el usuario que lo hizo:

```
CREATE OR REPLACE TRIGGERS INS_REGION
    AFTER INSERT ON REGIONS
BEGIN
    INSERT INTO TABLA_LOGS VALUES ( 'INSERCION TABLA REGIONS',USER,sysdate);
END;
```

Asumimos que nuestra tabla TABLA_LOGS tiene tres campos uno de tipo varchar2 TIPOLOG, otra NOMBRE_USR de tipo varchar2 y finalmente FECHA de tipo DATE.

TRIGGERS

COMPROBAR ESTADO DE LOS TRIGGERS:

```
DESC USER_TRIGGERS;

SELECT TRIGGER_NAME,ACTION_TYPE,TRIGGER_BODY FROM
USER_TRIGGERS;


SELECT OBJECT_NAME,OBJECT_TYPE,STATUS FROM USER_OBJECTS
WHERE OBJECT_TYPE='TRIGGER';
```