2.6.1.- Concepto de vista.

- Una vista es una tabla derivada de otras tablas (básicas o virtuales).
- Una vista se caracteriza porque:
  - Se considera que forma parte del esquema externo.
  - Una vista es una tabla virtual (no tiene una correspondencia a nivel físico)
  - Se puede consultar como cualquier tabla básica.
  - Las actualizaciones se transfieren a la/s tabla/s original/es (con ciertas limitaciones).

2.6.2.- Aplicaciones de las vistas.

USOS:

- Para la especificación de tablas con información que se accede con frecuencia pero no posee existencia física:
  - Información derivada de la relación entre varias tablas.
  - Información derivada de la formación de grupos de tuplas (p.ej. para la obtención de estadísticas).
  - En general: información derivada de consultas complejas a la que se accede con frecuencia.

- Como mecanismo de seguridad: creación de vistas con, únicamente, los atributos de las tablas a los cuales se desea permitir acceder a determinados usuarios.

- Para la creación de esquemas externos.

2.6.3.- Vistas en SQL.

- La sintaxis para la creación de vistas en SQL es la siguiente:

CREATE | REPLACE VIEW *vista* [(*comalista_columna*)]

AS *expresión_tabla* [with check option]

en donde:

– CREATE VIEW es la orden que permite la creación de la vista.

– *vista* es el nombre de la tabla virtual que se va a crear.

– (*comalista_columna*) son los nombres de los atributos de la tabla y es opcional:

  - Si no se especifica, el nombre coincide con el nombre de los atributos resultantes en *expresión_tabla*.
  - Es obligatorio si algún atributo de *expresión_tabla* es el resultado de una función de agregación o una operación aritmética.

2.6.3.- Vistas en SQL.

- La sintaxis para la creación de vistas en SQL es la siguiente:

  CREATE | REPLACE VIEW *vista* [(*comalista_columna*)]

  AS *expresión_tabla* [with check option]

en donde:

– *expresión_tabla* es una consulta SQL cuyo resultado será el contenido de la vista.

– WITH CHECK OPTION es opcional y se debe incluir si se desea actualizar la vista.

– Para la eliminación de una vista se utiliza la instrucción:

  – DROP VIEW *vista* [restrict | cascade];

2.6.3.- Vistas en SQL (Ejemplos).

- Dada la siguiente relación de una base de datos:

  Cocinero(nombre:varchar, edad: number, país:varchar)

Obtén una vista con, únicamente, los cocineros franceses:

```
CREATE VIEW Franceses AS
    SELECT * FROM Cocinero W
    WITH CHECK OPTION
```

El Check Option impide que yo pueda añadir cocineros que no sean franceses

Obtén una vista con la edad media de los cocineros agrupados por país:

```
CREATE VIEW Estudio(país, edad_media) AS
    SELECT país, AVG(edad) FROM Cocinero GROUP BY país
```

2.6.3.- Vistas en SQL (Vistas Actualizables).

Motivos por los que una vista NO es actualizable:

- contiene operadores conjuntistas (UNION, INTERSECT,…).
- el operador DISTINCT
- funciones agregadas (SUM, AVG, ..)
- la cláusula GROUP BY

2.6.3.- Vistas en SQL (Vistas Actualizables).

Vista sobre una tabla básica:

- el sistema traducirá la actualización sobre la vista en una operación de actualización sobre la relación básica.

- siempre que no se viole ninguna restricción de integridad definida sobre dicha relación.

2.6.3.- Vistas en SQL (Vistas Actualizables).

Vista sobre una concatenación de relaciones:

- la actualización sólo puede modificar una de las tablas básicas
- actualización modificará la relación básica que cumpla la propiedad de *conservación de la clave* (aquella relación tal que su clave primaria podría ser también clave de la vista)
- la actualización no se realizará si viola alguna de las restricciones definidas sobre la relación básica que se va a actualizar

2.6.3.- Vistas en SQL (Vistas Actualizables).

EJEMPLO:

- Dadas las siguientes relaciones:

  Persona(nif: dom_nif, nombre: dom_nom, edad: dom_edad)
  CP: {nif}

  Vivienda(cod_viv: dom_cod, nif: dom_nif, dir: dom_dir, num_hab: dom_num)
  CP: {cod_viv}

- Dada la siguiente vista definida sobre dichas relaciones:

  CREATE VIEW Todo_Vivienda AS

  SELECT * FROM Persona NATURAL JOIN Vivienda

¿Podré modificar la dirección de una vivienda en Todo_Vivienda?

Sí, la CP de Vivienda podría funcionar como CP de Todo_Vivienda

¿Podré modificar el nombre del propietario de una vivienda?

No, la actualización real es ambigua

2.7.1.- Concepto de disparador.

Son reglas que especifican acciones que son activadas automáticamente por determinados eventos.

2.7.2.- Reglas Evento-Condición-Acción.

Forma de una regla de actividad:

Evento - Condición - Acción

acción que el sistema ejecuta cuando como respuesta a la ocurrencia de un *evento* cuando cierta *condición* se satisface. En Oracle:

- *evento*: operación de actualización
- *condición*: expresión lógica del SQL. Si esta condición existe, la acción sólo se ejecutará si es verdadera. Si no existe, se asume cierta.
- *acción*: procedimiento escrito en PL/SQL (incluye instrucción de manipulación de la BD)

2.7.3.- Aplicaciones de los disparadores.

Define el comportamiento activo del sistema. Aplicaciones:

- comprobación de restricciones de integridad (estáticas y dinámicas)
- restauración de la consistencia
- control de la seguridad
- definición de reglas de funcionamiento de la organización
- mantenimiento de información derivada

2.7.4.- Disparadores en SQL.

definición_regla::=
{CREATE | REPLACE} TRIGGER nombre_regla
{BEFORE | AFTER | INSTEAD OF} ***evento*** [*disyunción_eventos*]
ON {nombre_relación | nombre_vista}
[[REFERENCING OLD AS nombre_referencia
[NEW AS nombre_referencia]]
[FOR EACH {ROW | STATEMENT} [WHEN (***condición***)]]
bloque PL/SQL

disyunción_eventos ::= OR *evento* [*disyunción_eventos*]

evento ::= INSERT | DELETE | UPDATE [OF comalista_nombre_atributo]

2.7.4.- Disparadores en SQL.

EVENTOS

{BEFORE | AFTER | INSTEAD OF} ***evento*** [*disyunción_eventos*]
ON {nombre_relación | nombre_vista}

disyunción_eventos ::= OR *evento* [*disyunción_eventos*]

evento ::=
INSERT | DELETE | UPDATE [OF comalista_nombre_atributo]

2.7.4.- Disparadores en SQL.

EVENTOS

Parametrización de eventos:

- los eventos de las reglas FOR EACH ROW están parametrizados
- parametrización implícita:
  - evento INSERT o DELETE: *n* (*n* grado de la relación)
  - evento UPDATE: 2**n*
- nombre de parámetros:
  - evento INSERT: *NEW*
  - evento DELETE: *OLD*
  - evento UPDATE: *OLD* y *NEW*
- se pueden usar en la *condición de la regla*
- se pueden usar en el bloque PL/SQL

2.7.4.- Disparadores en SQL.

	FOR EACH STATEMENT	FOR EACH ROW
BEFORE	La regla se ejecuta una vez antes de la ejecución de la operación de actualización	La regla se ejecuta una vez antes de actualización de cada tupla afectada por la operación de actualización
AFTER	La regla se ejecuta una vez después de la ejecución de la operación de actualización	La regla se ejecuta una vez después de actualización de cada tupla afectada por la operación de actualización

2.7.4.- Disparadores en SQL.

CONDICIONES

WHEN (condición)

- expresión lógica de sintaxis similar a la condición de la cláusula WHERE de la instrucción SELECT
- no puede contener subconsultas ni funciones agregadas
- sólo se puede hacer referencia a los parámetros del evento

2.7.4.- Disparadores en SQL.

ACCIONES

bloque PL/SQL

- bloque escrito en el lenguaje de programación de Oracle PL/SQL
- sentencias de manipulación de la BD: INSERT, DELETE, UPDATE, SELECT ... INTO ...
- sentencias de programa: asignación, selección, iteración
- sentencias de manejo de errores
- sentencias de entrada/salida

2.7.4.- Disparadores en SQL.

Lenguaje de reglas:

- Creación: CREATE TRIGGER nombre_regla ...
- Eliminación: DROP TRIGGER nombre_reglas
- Modificación: REPLACE TRIGGER nombre_regla ...
- Recompilación: ALTER TRIGGER nombre_regla COMPILE
- Des/Habilitar regla: ALTER TRIGGER nombre_regla [ENABLE | DISABLE]
- Des/Habilitar todas las reglas sobre una relación:

  ALTER TABLE nombre_relación [{ENABLE | DISABLE} ALL TRIGGERS]

2.7.4.- Disparadores en SQL (Ejemplo).

La restricción R2 :

R2) ∀Px: Pieza (∃Sx:Suministro (Sx.código=Px.código))

se define mediante una restricción general:

```
create assertion R2 check
not exists(select * from Pieza P
                where not exists(select *
                                 from Suministro S
                                 where P.código=S.código));
```

¿Cómo controlar la restricción mediante reglas de actividad?

2.7.4.- Disparadores en SQL (Ejemplo).

Detectar qué eventos pueden afectar a la R.I.:

TABLA,	OPERACIÓN,	ATRIBUTO
Suministro,	Borrado,	-
Suministro,	Modificación,	código
Pieza,	Inserción,	-

Construir Triggers para controlar estos eventos.

2.7.4.- Disparadores en SQL (Ejemplo).

```
CREATE TRIGGER T1
AFTER DELETE ON Suministro OR UPDATE OF codigo ON Suministro
FOR EACH ROW
DECLARE
    N: NUMBER;
BEGIN
    SELECT COUNT(*) INTO N
        FROM Suministro S
        WHERE :old.codigo = S.codigo;
    IF N=0 THEN
            RAISE_APPLICATION_ERROR(-20000, 'No se puede
               borrar el suministro, porque la pieza se quedaría sin suministros.');
    END IF;
END;
```

2.7.4.- Disparadores en SQL (Ejemplo).

```
CREATE TRIGGER T2
AFTER INSERT ON Pieza
FOR EACH ROW
DECLARE    N: NUMBER;
BEGIN
     SELECT COUNT(*) INTO N
          FROM Suministro S  WHERE :new.codigo = S.codigo;
     IF N=0 THEN
          RAISE_APPLICATION_ERROR(-20000, 'No se puede
               insertar una pieza, porque la pieza no tiene suministros.
               Cree las dos tuplas (la de pieza y la de suministro)
              dentro de una transacción deshabilitando este trigger.');
     END IF;
END;
```

2.8.- Limitaciones del modelo relacional.

- Los modelos de datos tradicionales (relacional, jerárquico y red) han tenido éxito en aplicaciones tradicionales de negocios.

- Los modelos tradicionales presentan deficiencias en aplicaciones:
  - de diseño y fabricación en ingeniería (CAD/CAM/CIM),
  - experimentos científicos,
  - telecomunicaciones,
  - sistemas de información geográfica y
  - multimedia.

2.8.- Limitaciones del modelo relacional.

- Requisitos y características de las nuevas aplicaciones:
  - estructuras más complejas para los objetos,
  - transacciones de mayor duración,
  - nuevos tipos de datos para almacenar imágenes o elementos de texto grandes y
  - la necesidad de definir operaciones no estándar específicas para las aplicaciones.

- Evolución de las bases de datos relacionales:
  - bases de datos deductivas,
  - bases de datos activas,
  - bases de datos orientadas a objetos y
  - bases de datos objeto-relacionales (SQL3)