

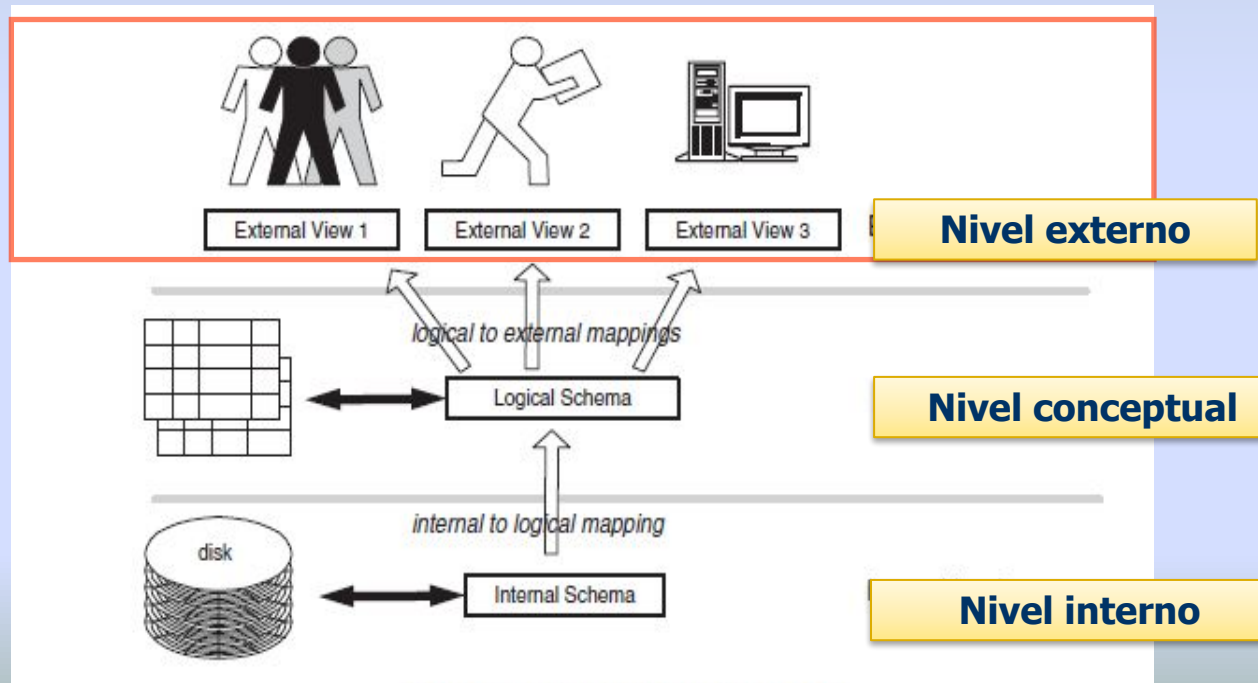
Bases de Datos I

Vistas



Vistas – Esquema Externo

- Las vistas forman parte del **esquema externo** de la BD
- Presentan parte/s de la BD de interés para grupos particulares de usuarios permitiendo ocultar el resto de la información (seguridad)



Creación de Vistas

- Una VISTA es una **relación derivada** de una o más tablas y/o vistas definidas previamente (da nombre a una expresión de consulta)

```
CREATE VIEW nom_vista [(n_col_1, ..., n_col_n)]  
AS expresión_consulta  
[WITH [opción] CHECK OPTION];
```

- **nom_vista**: nombre de la vista
- (**n_col_1**,..., **n_col_n**): nombres de columnas de la vista
 - Se puede especificar la lista completa de nombres columnas de la vista (opcional)
 - Si no, los nombres son los de las columnas de la/s tabla/s en la consulta (SELECT)
 - Se debe especificar con diferente nombre las columnas provenientes de distintas tablas pero con igual nombre
- **expresión_consulta**: consulta SQL que define la relación derivada
- Se considera una **tabla virtual**: las tuplas no se almacenan físicamente en la vista (salvo si está materializada), se generan al operar sobre ella

Creación de Vistas

Ejemplos

PROV_COMP con el identificador, nombre y ciudad de proveedores del rubro computadoras

```
CREATE VIEW PROV_COMP AS
  SELECT id_proveedor, nombre, ciudad
  FROM PROVEEDOR
  WHERE rubro = 'Computadoras';
```




Vista a partir
de una tabla

PROV_COMP_TANDIL con los proveedores de computadoras de Tandil (usando la vista previa):

```
CREATE VIEW PROV_COMP_TANDIL AS
  SELECT * FROM PROV_COMP
  WHERE ciudad = 'Tandil';
```

Vista a partir
de otra vista





PROVEEDOR

 id_proveedor	VARCHAR(10) NOT NULL
 nombre	VARCHAR(30) NOT NULL
 rubro	VARCHAR(15) NOT NULL
 ciudad	VARCHAR(30) NOT NULL

ENVIO

 id_proveedor (FK)	VARCHAR(10) NOT NULL
 id_articulo (FK)	VARCHAR(10) NOT NULL
 cantidad	NUMERIC(5,0) NOT NULL

ARTICULO

 id_articulo	VARCHAR(10) NOT NULL
 descrip	VARCHAR(30) NOT NULL
 peso	NUMERIC(5,2) NOT NULL
 ciudad	VARCHAR(30) NOT NULL

Creación de Vistas

Ejemplos

PROV_ENVIOS_TANDIL que contenga el identificador y nombre de los proveedores de computadoras de Tandil y los id. de artículos enviados por cada uno





```
CREATE VIEW PROV_ENVIOS_TANDIL AS  
SELECT P.id_proveedor, P.nombre, E.id_articulo  
FROM PROVEEDOR P JOIN ENVIO E  
ON P.id_proveedor = E.id_proveedor  
WHERE P.rubro = 'Computadoras'  
AND P.ciudad = 'Tandil';
```

Vista a partir de más de una tabla

o... **CREATE VIEW PROV_ENVIOS_TANDIL AS**
SELECT P.id_proveedor, nombre, E.id_articulo
FROM *PR_COMP_TANDIL* P **JOIN** ENVIO E
ON P.id_proveedor = E.id_proveedor;

Vista a partir de una tabla y una vista





PROVEEDOR

 id_proveedor	VARCHAR(10) NOT NULL
 nombre	VARCHAR(30) NOT NULL
 rubro	VARCHAR(15) NOT NULL
 ciudad	VARCHAR(30) NOT NULL

ENVIO

 id_proveedor (FK)	VARCHAR(10) NOT NULL
 id_articulo (FK)	VARCHAR(10) NOT NULL
 cantidad	NUMERIC(5,0) NOT NULL

ARTICULO

 id_articulo	VARCHAR(10) NOT NULL
 descrip	VARCHAR(30) NOT NULL
 peso	NUMERIC(5,2) NOT NULL
 ciudad	VARCHAR(30) NOT NULL

Vistas – Consultas/ Eliminación

- Una vista puede ser consultada como cualquier tabla: puede ser usada en cualquier SELECT y para generar otras vistas
- Para eliminar una vista del esquema de la BD:

```
DROP VIEW nom_vista [opción];
```

opción=

RESTRICT: se rechaza si hay objetos que hacen referencia a la vista
(*opción por defecto*)

CASCADE: procede siempre y se eliminan también los objetos dependientes

Actualizaciones Tabla Base → Vista

- Al actualizar las tuplas de una tabla → **los cambios se reflejan automáticamente sobre las vistas** definidas a partir de ella
- Las vistas por lo general no mantienen copias de los datos (salvo que sean *materializadas*)
- el SGBD asegura que las tuplas que conforman cada vista estén actualizadas al acceder a ellas

Actualizaciones Vista → Tabla Base

- Cada actualización sobre una vista se *debería propagar* a la/s tabla/s base (automáticamente)
- pero en ciertos casos la actualización podría traducirse a la/s tabla/s base de varias maneras (y lograr la modificación deseada en la vista) o se podrían generar **ambigüedades** o carecer de sentido o provocar efectos no deseados

Ejemplos

Cómo se propaga una actualización en caso de:

- un **campo derivado**?
- una **función de agregación (SUM, AVG, ...)**?
- una tupla que **no conserva la clave**?
- una selección con **distinct**?
- una vista resultante de un **ensamble**?



Actualizaciones Vista → Tabla Base

- Cada actualización de una tupla en una vista debe poder **propagarse sin ambigüedades** a la/s tabla/s base^(*)
- **No toda vista se puede actualizar automáticamente**
→ cuando una actualización sobre la vista pueda generar distintas posibilidades de actualización a la/s tabla/s base^(*), se deberá recurrir a algún procedimiento específico para implementar la que sea requerida

(*) Puede ser otra vista, que debe ser actualizable

Si se tienen definidas vistas a partir de vistas : $T \rightarrow V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V_n$
 V_i será actualizable si V_{i-1} lo es y así sucesivamente

Propiedad de Preservación de Clave

- Establece la condición para que una vista sea actualizable
- La actualización en una vista debe traducirse en **una única actualización y del mismo tipo** en la tabla^(*) base cuya clave 'heredó' la vista
- La **clave de la vista** es:
 - la clave de la tabla^(*) de la cual procede (vistas de una tabla^(*))
 - la de alguna de las tablas^(*) de las cuales procede (v.ensamble)
- Se satisface si **cada fila en la tabla aparece como máximo una vez en la vista**
- La propiedad NO depende de los datos actuales en las tablas, sino que es una propiedad estructural de su esquema

(*) Puede ser otra vista, que debe ser actualizable

Nota: En una jerarquía de vistas, todas deben preservar la clave para que la propiedad se cumpla

Vistas Actualizables

(a partir de una tabla/vista)

- Según el estándar SQL, **una vista definida sobre una tabla base** (u otra vista actualizable) **es actualizable si**:
 - ✓ **conserva todas las columnas de la clave (primaria)**
 - ✓ **no contiene funciones de agregación o información derivada**
 - ✓ **no incluye la cláusula DISTINCT**
 - ✓ **no incluye subconsultas en el SELECT**
- **vistas σ - π** (a partir de **selección** de tuplas y **proyección** de columnas)
- El SGBD traduce una actualización de una vista definida a partir de una tabla/vista en una única operación de actualización del mismo tipo sobre la tabla/vista subyacente (siempre que no se viole ninguna restricción de integridad definida sobre la relación)

Nota: Algunos SGBD soportan otras posibilidades que las especificadas por el estándar SQL

Vistas Actualizables

(a partir de una tabla/vista)

Ejemplos

PROV_TANDIL que contenga el identificador y nombre de los proveedores de Tandil

```
CREATE VIEW PROV_TANDIL AS  
  SELECT id_proveedor, nombre  
  FROM PROVEEDOR WHERE ciudad = 'Tandil';
```



PROV_COMP con nombre y ciudad de proveedores de computadoras, ordenados por nombre

```
CREATE VIEW PROV_COMP  
AS SELECT nombre, ciudad  
FROM PROVEEDOR WHERE rubro = 'Computadoras'  
ORDER BY nombre;
```






TOTAL_ARTICULO que liste, para cada articulo, su identificador y la cantidad total enviada


```
CREATE VIEW TOTAL_ARTICULO (articulo, total) AS  
  SELECT id_articulo, sum(cantidad)  
  FROM ENVIO  
  GROUP BY id_articulo;
```







PROVEEDOR

 id_proveedor	VARCHAR(10)	NOT NULL
 nombre	VARCHAR(30)	NOT NULL
 rubro	VARCHAR(15)	NOT NULL
 ciudad	VARCHAR(30)	NOT NULL

ENVIO

 id_proveedor (FK)	VARCHAR(10)	NOT NULL
 id_articulo (FK)	VARCHAR(10)	NOT NULL
 cantidad	NUMERIC(5,0)	NOT NULL

ARTICULO

 id_articulo	VARCHAR(10)	NOT NULL
 descrip	VARCHAR(30)	NOT NULL
 peso	NUMERIC(5,2)	NOT NULL
 ciudad	VARCHAR(30)	NOT NULL

¿Migración de Tuplas?

Al efectuar actualizaciones en una vista, podría ocurrir que las tuplas afectadas dejen de pertenecer a la vista

Ejemplo

```
CREATE VIEW PROV_TANDIL AS  
SELECT * FROM PROVEEDOR  
WHERE ciudad = 'Tandil';
```

¿Cuál sería el resultado de la siguiente operación?

```
UPDATE PROV_TANDIL SET ciudad= 'Azul';
```

- En este caso todos los registros de la vista serían actualizados con un valor de ciudad diferente a Tandil y entonces **dejarían de pertenecer a la vista** (migración)

Vistas con Opción de Chequeo

Solución a la migración de tuplas

→ incluir la cláusula **WITH CHECK OPTION (WCO)** al crear la vista (controla el comportamiento de vistas automáticamente actualizables)

```
CREATE VIEW nom_vista [(n_col_1, ..., n_col_n)]  
AS expresión_consulta  
[WITH [opción] CHECK OPTION];
```

- Si se especifica WCO: una actualización sobre la vista **procede si satisface la condición de consulta** que la define, si no se rechaza
- **opción=**
 - CASCADED**: se chequea contra las condiciones de la vista y de las vistas subyacentes (*opción por defecto*)
 - LOCAL**: se chequea sólo contra las condiciones definidas en la misma vista (y no las de las vistas subyacentes, a menos que también tengan especificado CHECK OPTION)

Vistas con Opción de Chequeo

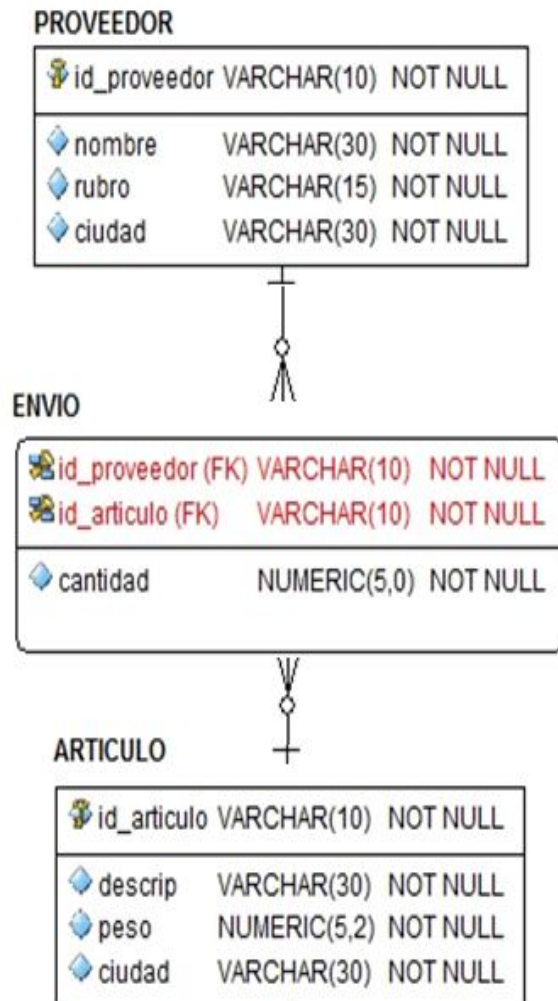
Ejemplo

Vista *ENVIOS500* con los envíos de 500 o más unidades de algún artículo (a partir de ENVIO):

```
CREATE VIEW Envios500 AS  
SELECT * FROM ENVIO  
WHERE cantidad >= 500;
```

Cuál es el efecto de las operaciones, si la vista

- se define sin WCO
- se define con WCO
- `INSERT INTO Envios500 VALUES ('P1', 'A1', 500);`
- `INSERT INTO Envios500 VALUES ('P2', 'A2', 300);`
- `UPDATE Envios500 SET cantidad= 100 WHERE id_proveedor= 'P1';`



Vistas con Opción de Chequeo

Ejemplo

```
CREATE VIEW Envios500 AS  
  SELECT * FROM ENVIO  
  WHERE cantidad >= 500;
```

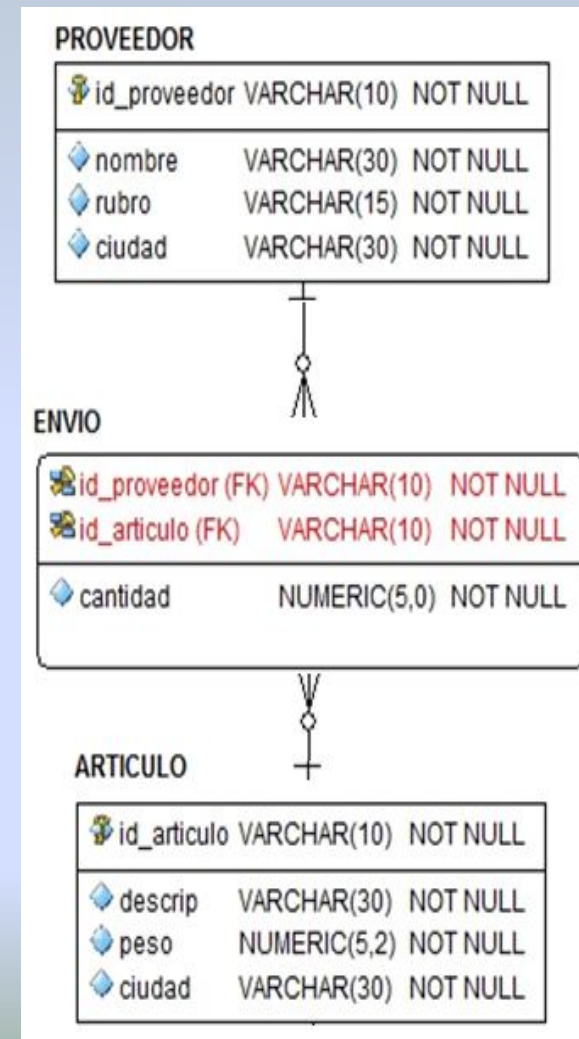
*ENVIOS500-999*_con los envíos de entre 500 y 999 unidades (a partir de *ENVIOS500*)

```
CREATE VIEW Envios500-999 AS  
  SELECT * FROM ENVIO500  
  WHERE cantidad < 1000;
```

- Si *Envios500* no tiene WCO, ¿cuál es la respuesta ante la operación:

```
UPDATE Envios500-999 SET cantidad= 300 where  
(id_proveedor = 'P1' and id_articulo = 'A1');
```

- si *Envios500-999* posee CASCADDED Check Option?
- si *Envios500-999* posee LOCAL Check Option?
- ¿y si *Envios500* posee WCO?



Vistas Actualizables

(a partir de 2 o más tablas/vistas)

Según el estándar SQL:1999, la actualización sobre una **vista definida a partir de más de una tabla/vista**^(*)

- **sólo puede modificar una de las tablas base**: la que cumpla la propiedad de **preservación de la clave** (la que tiene la misma clave de la vista, y entonces cada tupla aparece a lo sumo una vez en la vista)
- No debe definirse en base a Unión, Intersección o Diferencia
(Manifiestos para SQL:1999 y versiones posteriores → sugieren incluir operaciones, como **Intersección**)
- Se denominan **vistas σ - π - \bowtie** : se obtienen mediante **condiciones de ensamble sobre los pares FK \rightarrow PK especificados en las RIRs**
- Nota: La actualización no se realizará si se viola alguna restricción definida sobre la relación base a actualizar

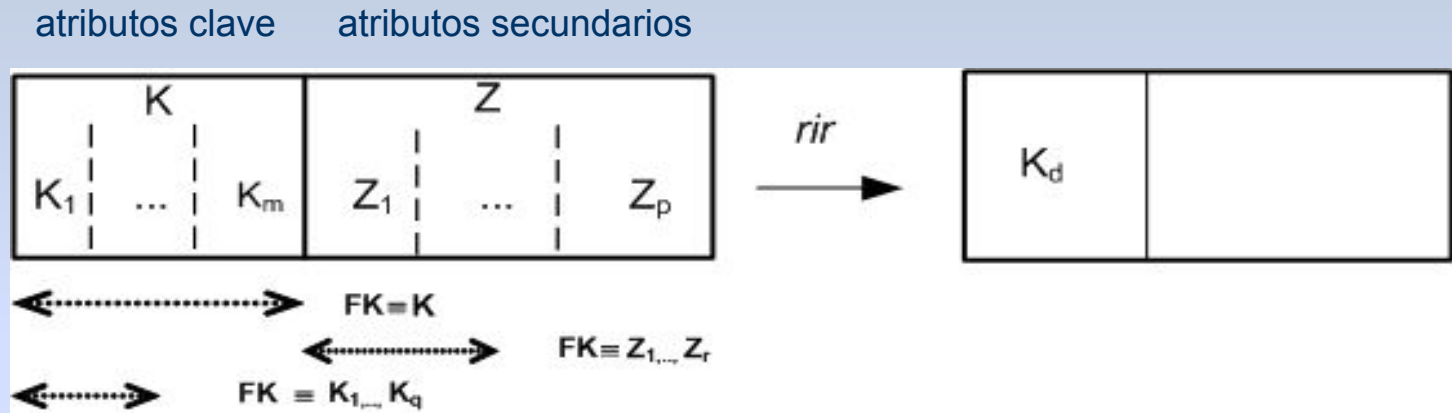
**Soportado en Oracle,
no en PostgreSQL**

(*) Puede ser otra vista, que debe ser actualizable.

Si $T \rightarrow V_1 \rightarrow V_2 \rightarrow \dots \rightarrow V_n$: V_i será actualizable si V_{i-1} lo es y así sucesivamente

Comentarios sobre RIRs

Posibles ubicaciones de la FK en relación al esquema de la relación:



- (1) **$FK \equiv Z_1, \dots, Z_r$** (atrib. secundarios); con $FK \cap K = \emptyset \rightarrow$ proviene de la abstracción de relaciones 1:1, N:1 o n -arias con al menos una cardinalidad 1
- (2) **$FK \equiv K_1, \dots, K_q$; con $FK \subset K$** \rightarrow resulta de las relaciones N:N, n -arias en general y las correspondientes a los vínculos entidad fuerte-entidad débil (*rel. identificatorias*)
- (3) **$FK \equiv K$** \rightarrow procede de la representación de relaciones de tipo-subtipo

Vistas Actualizables

(a partir de 2 o más tablas/vistas)

Caso (1): más intuitivo → Ensamble mediante RIR, FK= atrib. secundarios

Ejemplo

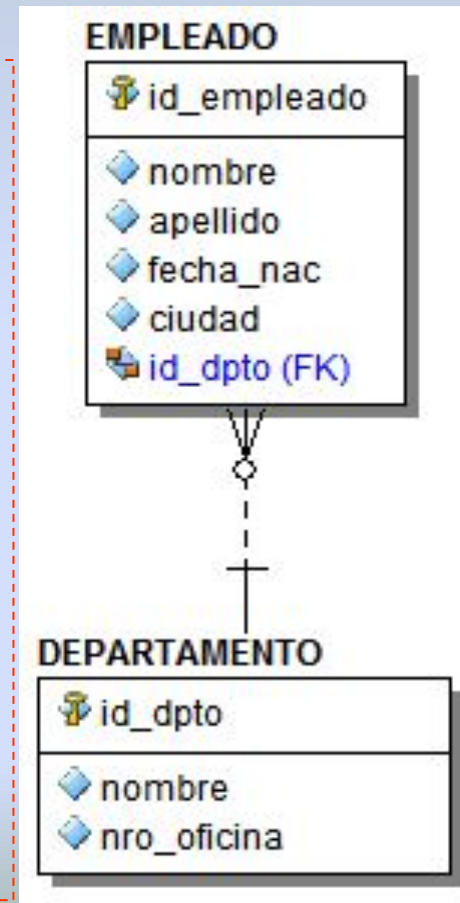
~~Vista ensamble proveniente de relación N:1~~

```
CREATE VIEW EMPL_SISTEMAS AS
SELECT E.id_empleado, E.apellido, E.id_dpto, D.nombre
FROM EMPLEADO E, DEPARTAMENTO D
WHERE E.id_dpto = D.id_dpto
      AND D.nombre= 'Sistemas';
```

La clave de *EMPL_SISTEMAS* es *E.id_empleado*
(análisis de dependencias funcionales)

→ se pueden hacer actualizaciones sobre columnas
provenientes de EMPLEADO

Nota: El análisis no depende de los datos sino de la estructura del esquema
(aunque en la vista hubiera sólo un empleado en cada departamento – y
entonces id_dpto fuera único– la tabla Departamento igualmente NO es la
que preserva la clave a la Vista)



Vistas Actualizables

(a partir de 2 o más tablas/vistas)

Caso (2): Ensamble mediante RIR, $FK \subset K$
(subconjunto de los atrib. clave)

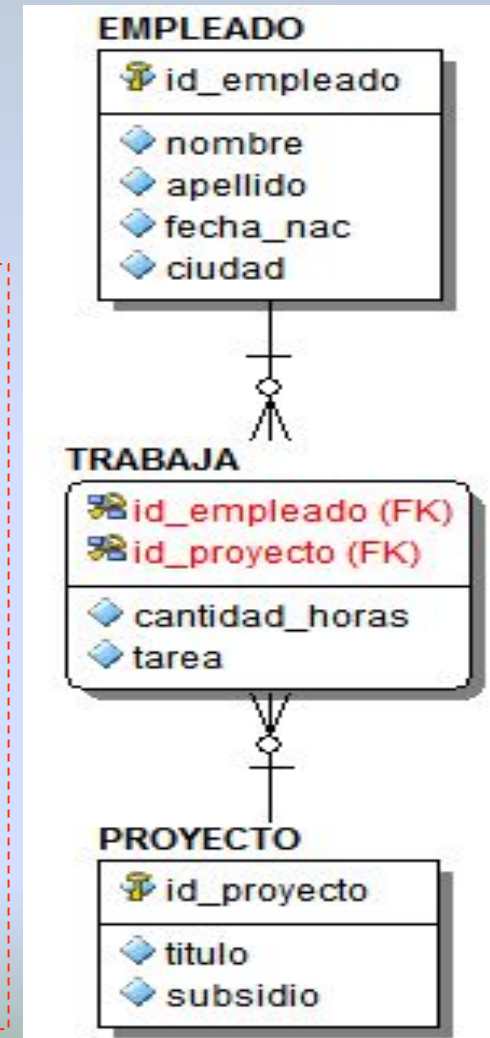
Ejemplo

~~Vista ensamble proveniente de relación N:N~~

```
CREATE VIEW EMPL_PROY AS
SELECT T.id_empleado, T.id_proyecto, T.tarea, E.apellido
FROM TRABAJA T, EMPLEADO E
WHERE T.id_empleado = E.id_empleado
AND E.cantidad_horas < 3000;
```

La clave de *EMPL_PROY* es *T.id_empleado*,
T.id_proyecto
(análisis de dependencias funcionales)

→ se pueden hacer actualizaciones sobre columnas
provenientes de TRABAJA



Vistas Actualizables

(a partir de 2 o más tablas/vistas)

Caso (3): Ensamble mediante RIR, $FK \equiv K$ (coincide con la clave)

Ejemplo

Vista ensamble correspondiente a relación tipo-subtipo

```
CREATE VIEW ING_TANDIL AS
SELECT I.id_empleado, I.titulo, E.apellido, E.nombre
FROM INGENIERO I, EMPLEADO E
WHERE I.id_empleado = E.id_empleado
AND E.ciudad = 'Tandil';
```

La clave de *ING_TANDIL* es *I.id_empleado* (tabla subtipo)

Nota: En esta vista, Empleado e Ingeniero se corresponden con sólo una persona, o sea que *id_empleado* es único en la vista (resultado del ensamble), pero Empleado no sería la tabla de la cual se preserva la clave (porque es esperable que su dominio de definición sea más extenso que el de Ingeniero)



Vistas – Actualización vía Triggers

- Cuando **una vista** no resulta automáticamente actualizable
 - Se pueden definir **triggers INSTEAD OF** (opción especial para vistas) para las distintas operaciones de actualización requeridas
 - Entonces se “interceptan” las operaciones de actualización: el trigger se dispara automáticamente **en lugar de** la sentencia disparadora, en forma “invisible” para el usuario
 - Por defecto, los triggers INSTEAD OF son *for each row*
- Importante: Queda en manos del usuario la responsabilidad de implementar las actualizaciones necesarias y de manera “adecuada”



Vistas – Actualización vía Triggers

Ejemplo

```
CREATE VIEW info_tutores (nro_al, nom_al, id_tut, nom_tut)
AS SELECT A.nro_al, A.nombre, A.id_tutor, P.nombre
   FROM Alumno A JOIN Profesor P
   ON A.Id_tutor = P.id_prof;
```

Alumno (nro_al, nombre, id_tutor)
Profesor (id_prof, nombre)
Alumno(id_tutor) << Profesor(id_prof)

Una **posible semántica para el insert** sobre *info_tutores* podría ser:
(no significa que sea la única, dependerá de los requerimientos)

```
CREATE TRIGGER insert_info_tutores
INSTEAD OF INSERT ON info_tutores
FOR EACH ROW
BEGIN
  IF NOT EXISTS (SELECT * FROM Profesor P WHERE P.id_prof= :new.id_tutor)
    THEN INSERT INTO Profesor VALUES (:new.id_tutor, :new.nom_tut);
    ELSE UPDATE Profesor SET nombre= :new.nom_tut WHERE id_prof= :new.id_tutor;
  END IF;
  IF NOT EXISTS (SELECT * FROM Alumno A WHERE A.nro_al= :new.nro_al)
    THEN INSERT INTO Alumno VALUES(:new.nro_al, :new.nom_al, :new.id_tutor);
    ELSE UPDATE Alumno SET id_tutor= :new.id_tutor WHERE nro_al= :new.nro_al;
  END IF;
END;
```

(sintaxis SQL estándar)

Deberían plantearse triggers similares para las operaciones de UPDATE y DELETE
(en PostgreSQL la función podría implementar el comportamiento para todos los eventos)

Vistas Materializadas

- Algunas aplicaciones pueden requerir alto grado de respuesta por parte de la BD (no siendo suficiente la optimización de la/s consulta/s)
- Una vista puede ser **materializada**
 - el SGBD pre-calcula y almacena su contenido (físicamente, como sucede con una tabla)
 - Puede ser usada como si fuera una relación común (incluso crear índices para mejorar la performance de consultas)
 - Se genera duplicación de datos y necesidad de **mantener la consistencia** respecto del contenido de las tablas base
- Cuestiones a evaluar:
 - ¿Qué vistas deberían materializarse? (decisión compleja)
 - ¿Cómo será la sincronización entre vistas materializadas y tablas base?
 - Aplicar actualización por regeneración de la vista? o incremental?
 - En forma inmediata (al darse algún cambio)? periódicamente? forzado?

Vistas - Ventajas

- **Simplifican la percepción** que los usuarios tienen de la BD, presentando la información necesaria y ocultando el resto
- **Presentan diferentes datos** a distinto tipo de usuarios, aún cuando los estén compartiendo (sobre la misma BD)
- **Permiten definir consultas complejas/frecuentes** para no tener que especificarlas cada vez que se utilizan
- **Facilitan la independencia de los datos** (ocultando a los usuarios cambios en la estructura en las tablas base)
- **Permiten aplicar políticas de seguridad:** dando privilegios selectivamente sobre distintas vistas (control de acceso)
 - Vistas sobre algunas columnas, ocultando otras reservadas para usuarios específicos (ej. *antecedentes penales*)
 - Vistas sobre determinadas filas, ocultando otras reservadas para usuarios específicos (ej. *películas no aptas para público infantil*)



Vistas - Desventajas

- **Actualizaciones VISTA → Tabla BASE restringidas:** hay varias limitaciones sobre la estructura de las vistas para asegurar que resulten automáticamente actualizables (debido a posibles anomalías)
- **Cuestiones de Rendimiento:** el proceso de resolución de la vista puede exigir el acceso a múltiples tablas cada vez que se accede a ella → evaluar si podría justificarse su materialización (técnicas de mantenimiento de vistas)
- **Necesidad de sincronización** en caso de vistas materializadas (y duplicación de datos)
- Modificaciones a la estructura de tablas base (ej. agregado de columnas) no serán advertidos por la vista, salvo que sea re-creada



BIBLIOGRAFÍA

Date, C., "An Introduction to Database Systems". 8º ed., Addison Wesley, 2004

Elmasri, R., Navathe, S., "Fundamentals of Database Systems", Addison Wesley, 2011 (Cap. 5)

Ramakrishnan R., Gehrke J., "Database Management Systems", 3º ed. , McGraw-Hill, 2003 (Cap. 3 y 25)

Silberschatz, A., Korth, H, Sudarshan, S., "Database System Concepts", McGraw Hill, 2001 (Cap. 4)