

Lenguajes de Bases de Datos Relacionales

Tema 9. SQL: Definición de Reglas de Integridad y otros elementos (DDL)

Tema 9. SQL: Definición de Reglas de Integridad y otros elementos

1

Objetivos

- Aprender la **sintaxis** del **SQL** con el fin de escribir sentencias **de definición de reglas de integridad, de vistas y de índices** en una base de datos relacional
 - ▣ *Data Definition Language (DDL)* del SQL
 - ▣ Diferenciar entre el ANSI SQL y los **dialectos SQL** implementados por sistemas de bases de datos comerciales
- Profundizar en el concepto de **Integridad** de Datos
- Entender el concepto de **vista relacional**, y la problemática asociada a la modificación de datos a través de vistas
- Tener un primer contacto con el concepto de **índice** y entender el propósito de su utilización

Tema 9. SQL: Definición de Reglas de Integridad y otros elementos

2

Bibliografía

- [CB 2015] Connolly, T.M.; Begg C.E.: ***Database Systems: A Practical Approach to Design, Implementation, and Management***, 6th Edition. Pearson. (Capítulos 6 y 7)
- [EN 2016] Elmasri, R.; Navathe, S.B.: ***Fundamentals of Database Systems***, 7th Edition. Pearson. (Caps. 6 y 7, 16 y 17)

Tema 9. SQL: Definición de Reglas de Integridad y otros elementos

3

Contenidos

□ 9.1 Reglas de Integridad

- ▣ Definición. Componentes

- ▣ Clasificación. Asertos

□ 9.2 Vistas relacionales

□ 9.3 Índices

□ 9.4 Esquemas y Catálogo

→ *Siempre con indicaciones acerca del lenguaje Oracle SQL*

□ Anexo

- ▣ “Implementación” de Vistas

- ▣ Metadatos: INFORMATION_SCHEMA del Catálogo

- Oracle: Diccionario de Datos
y Vistas de Rendimiento Dinámicas

9.1 Reglas de Integridad

4

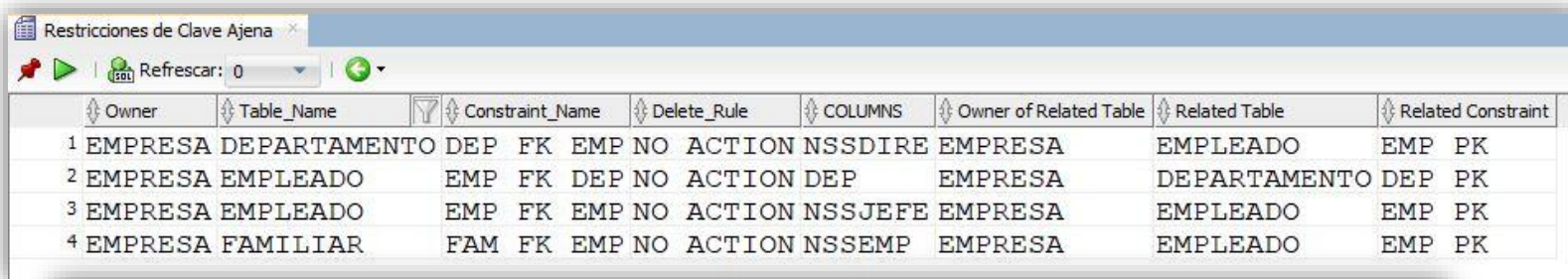
- Recordemos: **una BD es una representación** o modelo de una porción **del mundo real**
- Por ello, la **información almacenada** debe **cumplir** en todo momento las **restricciones, reglas o normas** existentes en esa realidad
 - ▣ *El DNI de los empleados debe ser único*
 - ▣ *El director de un departamento ha de ser uno de los empleados que trabajan en dicho departamento*
 - ▣ *Un empleado no puede ser director de más de un departamento*
 - ▣ ...



9.1 Reglas de Integridad

5

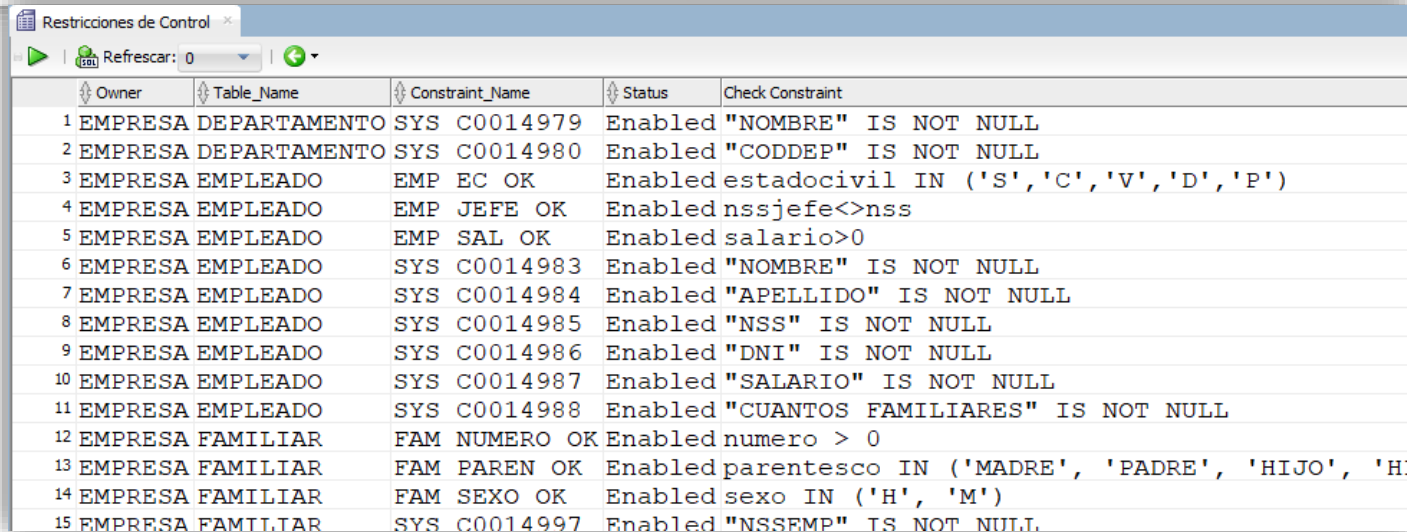
- Al definir/crear un esquema de BD Relacional, estas restricciones quedan **almacenadas junto con los datos** en forma de **Reglas de Integridad** (∈ *metadatos*)
 - ▣ Así el SGBD las conoce y las impone automáticamente



Restricciones de Clave Ajena

Refrescar: 0

	Owner	Table_Name	Constraint_Name	Delete_Rule	COLUMNS	Owner of Related Table	Related Table	Related Constraint
1	EMPRESA	DEPARTAMENTO	DEP FK EMP	NO ACTION	NSSDIRE	EMPRESA	EMPLEADO	EMP PK
2	EMPRESA	EMPLEADO	EMP FK DEP	NO ACTION	DEP	EMPRESA	DEPARTAMENTO	DEP PK
3	EMPRESA	EMPLEADO	EMP FK EMP	NO ACTION	NSSJEFE	EMPRESA	EMPLEADO	EMP PK
4	EMPRESA	FAMILIAR	FAM FK EMP	NO ACTION	NSSEMP	EMPRESA	EMPLEADO	EMP PK



Restricciones de Control

Refrescar: 0

	Owner	Table_Name	Constraint_Name	Status	Check Constraint
1	EMPRESA	DEPARTAMENTO	SYS C0014979	Enabled	"NOMBRE" IS NOT NULL
2	EMPRESA	DEPARTAMENTO	SYS C0014980	Enabled	"CODDEP" IS NOT NULL
3	EMPRESA	EMPLEADO	EMP EC OK	Enabled	estadocivil IN ('S','C','V','D','P')
4	EMPRESA	EMPLEADO	EMP JEFE OK	Enabled	nssjefe<>nss
5	EMPRESA	EMPLEADO	EMP SAL OK	Enabled	salario>0
6	EMPRESA	EMPLEADO	SYS C0014983	Enabled	"NOMBRE" IS NOT NULL
7	EMPRESA	EMPLEADO	SYS C0014984	Enabled	"APELLIDO" IS NOT NULL
8	EMPRESA	EMPLEADO	SYS C0014985	Enabled	"NSS" IS NOT NULL
9	EMPRESA	EMPLEADO	SYS C0014986	Enabled	"DNI" IS NOT NULL
10	EMPRESA	EMPLEADO	SYS C0014987	Enabled	"SALARIO" IS NOT NULL
11	EMPRESA	EMPLEADO	SYS C0014988	Enabled	"CUANTOS FAMILIARES" IS NOT NULL
12	EMPRESA	FAMILIAR	FAM NUMERO OK	Enabled	numero > 0
13	EMPRESA	FAMILIAR	FAM PAREN OK	Enabled	parentesco IN ('MADRE', 'PADRE', 'HIJO', 'HI
14	EMPRESA	FAMILIAR	FAM SEXO OK	Enabled	sexo IN ('H', 'M')
15	EMPRESA	FAMILIAR	SYS C0014997	Enabled	"NSSEMP" IS NOT NULL



Base
de Datos

Consideraciones generales

6

- La **integridad** es la **consistencia o corrección de los datos** almacenados en la base de datos
- Las **Reglas de Integridad** (RI) afectan a los datos almacenados y **se deben cumplir siempre**, independientemente de los usuarios o programas que accedan a los datos
- La integridad es **importante en ...**
 - ▣ Etapas de **diseño** (definir estructuras de datos junto con las reglas de integridad adecuadas)
 - ▣ **Ejecución** (asegurar la corrección de la información)

Componentes de una Regla de Integridad

7

□ Nombre

- Regla almacenada en el INFORMATION_SCHEMA del catálogo (metadatos) con ese nombre
- Aparecerá en los diagnósticos (mensajes de error) producidos por el sistema como respuesta a intentos de incumplimiento de la regla

□ Restricción

- Expresión booleana **Restricción de Integridad \subseteq Regla de Integridad**
- La regla ... se satisface \Leftrightarrow la restricción es TRUE
es incumplida \Leftrightarrow la restricción es FALSE



□ Respuesta a un intento de incumplimiento de la regla

- Indica al SGBD **qué hacer si se intenta una operación que incumple o rompe la RI**
- Por defecto es RECHAZAR, que implica...
 - No realizar la operación, o deshacer los posibles daños causados por ella
 - Y mostrar información de diagnóstico (mensaje)

Componentes de una Regla de Integridad. Ejemplos

- **Nombre** `dep_pk` -- en DEPARTAMENTO
Restricción `PRIMARY KEY(coddep)`
 □ Incluye `coddep IS NOT NULL` y que los valores de `coddep` no se pueden repetir en la tabla DEPARTAMENTO
Respuesta a un intento de incumplimiento
 Siempre RECHAZAR
- **Nombre** `emp_jefe_ok` -- en EMPLEADO
Restricción `CHECK (nssjefe <> nss)`
Respuesta a un intento de incumplimiento
 Siempre RECHAZAR
- **Nombre** `departamento_director_ok`
Restricción `NOT EXISTS (SELECT *`

Todo director de departamento debe ser uno de sus empleados

`FROM departamento`
`WHERE nssdire NOT IN (SELECT nss`
`FROM empleado`
`WHERE dep = coddep))`

Respuesta a un intento de incumplimiento
 Siempre RECHAZAR

Un INSERT o un UPDATE podría incumplir cualquiera de estas 2 primeras reglas

Un INSERT, un UPDATE o un DELETE podría incumplir esta 3ª regla

Componentes de una Regla de Integridad. Ejemplos

9

- **Nombre** `emp_fk_dep` -- en EMPLEADO
Restricción `FOREIGN KEY(dep) REFERENCES DEPARTAMENTO(coddep)`
 - ▣ Los valores de “dep” deben existir en la columna “coddep” de DEPARTAMENTO**Respuesta** a un intento de incumplimiento
En el INSERT: RECHAZAR; En el DELETE: RECHAZAR; En el UPDATE: CASCADE

- **Nombre** `emp_fk_emp` -- en EMPLEADO
Restricción `FOREIGN KEY(nssjefe) REFERENCES EMPLEADO(nss)`
 - ▣ Los valores de “nssjefe” deben existir en la columna “nss” de EMPLEADO**Respuesta** a un intento de incumplimiento
En el INSERT: RECHAZAR; En el DELETE: SET NULL; En el UPDATE: CASCADE

- **Nombre** `familiar_fk_emp` -- en FAMILIAR
Restricción `FOREIGN KEY(nssemp) REFERENCES EMPLEADO(nss)`
 - ▣ Los valores de “nssemp” deben existir en la columna “nss” de EMPLEADO**Respuesta** a un intento de incumplimiento
En el INSERT: RECHAZAR; En el DELETE: CASCADE; En el UPDATE: CASCADE

Un INSERT, un UPDATE o un DELETE podría incumplir cualquiera de estas reglas

Categorías de reglas de integridad

ANSI
SQL

No las veremos

10

1. Reglas de integridad de Dominio

- Asociadas a un *dominio de datos* específico
 - **No** hemos visto los dominios de ANSI SQL (*Oracle* no los implementa)
 - CREATE DOMAIN ... ;
- Es una expresión de complejidad arbitraria que define un dominio de datos

2. Reglas de integridad de Tabla

- RI de complejidad arbitraria incluidas en la definición de una tabla
- *Curiosidad: una tabla vacía cumple cualquier RI de tabla, aunque esa RI sea “esta tabla no puede estar vacía”*

3. Reglas de integridad Generales

- RI de complejidad arbitraria, **no** incluidas en la definición de ninguna tabla
- Son **otro elemento más de la BD**, al mismo nivel que una tabla o vista

Reglas de Integridad **de Tabla**

11

Visto ☒

- ❑ Restricción **asociada a una tabla específica**

- ▣ No existe si la tabla no existe y
- ▣ Eliminar la tabla (o columna) implica eliminar la RI

- ❑ RI **especificada dentro de CREATE TABLE**

CREATE TABLE <nombre tabla> (<lista de elemento de tabla>);

donde un elemento puede ser:

- ▣ Definición de columna, que puede incluir **Restricciones de columna**
- ▣ Definición de... (precedida o no de CONSTRAINT <nombre restricción>)
 - Restricción **de clave** candidata
 - Restricción **de clave ajena** (o externa)
 - Restricción **de comprobación** (CHECK)

- ❑ RI **añadida/eliminada con ALTER TABLE <nombre tabla>...**

- ❑ Toda RI de tabla es **comprobada inmediatamente**:

- ▣ Una operación de **modificación sobre la tabla incluye el chequeo de todas sus RI** (como paso final de la operación)+(una posible) **acción**



Reglas de Integridad **Generales** (**Asertos**)

12

- Es importante poder definir **restricciones específicas** de cada base de datos particular:
 - ▣ [BD de una biblioteca] *Un socio penalizado no puede tener ningún préstamo en curso.*
 - ▣ [BD de ventas de productos] *Un cliente puede conseguir un 15% de descuento en sus compras siempre que sea cliente desde hace más de 3 años y haya comprado más de 2 productos al año.*
 - ▣ [BD de una universidad] *Un alumno puede matricularse de un máximo de 9 créditos de libre elección.*
 - ▣ etc.
- Las Reglas de Integridad Generales o **Asertos** permiten especificar restricciones de integridad más allá de las de dominio y de tabla
- Los asertos suelen 'implementar' restricciones detectadas en el Diseño Conceptual y/o en el Diseño Lógico

Reglas de Integridad Generales (Asertos)

13


- Un aserto incluye un predicado que expresa una **condición** que la base de datos debe **satisfacer siempre**
- Puede involucrar cualquier número de columnas de cualquier cantidad de tablas

```
CREATE ASSERTION RI_libro_tiene_copias  
CHECK (NOT EXISTS (SELECT *  
                    FROM libro  
                    WHERE isbn NOT IN (SELECT isbn  
                                       FROM ejemplar))));
```

- Es un **elemento de BD independiente** de tablas y vistas existentes
- Tiene un **nombre** y consta de una **condición**

Reglas de Integridad Generales (Asertos)

14

- ❑ **Creación** de una RI general
CREATE ASSERTION *<nombre_aserto>* \Rightarrow nombre **obligatorio**
CHECK (*<condición>*);
- ❑ **Eliminación** de una RI general
DROP ASSERTION *<nombre_aserto>*;  Sin opción RESTRICT o CASCADE
 - ▣ Elimina el aserto del INFORMATION_SCHEMA del catálogo
- ❑ **Satisfacción e incumplimiento** de una RI general
 - ▣ Un **estado de la BD** **satisface un aserto** si ninguna (combinación de) fila(s) en dicho estado **incumple la condición** que incluye
 - ▣ Si **alguna fila** de la BD **hace falsa la condición**, el aserto es **incumplido**

Oracle SQL **NO** implementa directamente las sentencias
CREATE ASSERTION ni DROP ASSERTION.

Sí podremos **ejecutar la SELECT** que incluye **y comprobar**
que devuelve una **tabla vacía**

Redacción de asertos

15

- Normalmente, la *<condición>* se expresa **en negativo**:
 todo X satisface Y \equiv **ningún X satisface NO(Y)**

**Todo empleado del departamento 'D3' tendrá un salario de 900€ o más*

► **Ningún empleado del departamento 'D3' tendrá un salario menor de 900€**

```
CREATE ASSERTION RI1a
CHECK (NOT EXISTS (SELECT * FROM empleado
                    WHERE dep = 'D3'
                    AND salario < 900));
```

- ¿Y por qué no en “positivo”?

```
CREATE ASSERTION RI1b
CHECK (EXISTS (SELECT * FROM empleado
                WHERE dep = 'D3'
                AND salario >= 900));
```

- ¿Qué aserto (RI1a o RI1b) cumplen las filas de la tabla EMPLEADO?

¿Qué está mal? ¿?

EMPLEADO

nombre	...	dep	salario
JONÁS		D1	1100
RIGOBERTA		D3	900
EUSEBIO		D2	2100
MACARENO		D1	1100
CASIANA		D3	920
FILOMENA		D1	1100
ELEUTERIO		D3	800
CEFERINA		D4	4625
GUMERSINDA		NULL	850
PRUDENCIO		D4	2250
FERDINANDA		D4	1560

Redacción de asertos

16

**El director de un departamento ha de ser uno de los empleados del mismo*

► *Ningún departamento tiene un director que no trabaja en dicho departamento*

```
CREATE ASSERTION RI2_director_departamento
CHECK (NOT EXISTS (SELECT *
                    FROM departamento
                    WHERE nssdire NOT IN (SELECT nss
                                          FROM empleado
                                          WHERE dep = coddep))));
```

**Todo departamento puede tener un máximo de 100 empleados*

► *No existe un departamento con más de 100 empleados: si se cuentan los empleados por departamento, nunca es superior a 100.*

```
CREATE ASSERTION RI3_max_empleados_departamento
CHECK (NOT EXISTS (SELECT dep
                    FROM empleado
                    GROUP BY dep
                    HAVING COUNT(*) > 100));
```

Redacción de asertos

17

**Todo departamento debe tener al menos un empleado*

► No existe departamento cuyo código no esté referenciado desde un empleado

```
CREATE ASSERTION RI4_empleado_en_departamento  
CHECK (NOT EXISTS (SELECT * FROM departamento  
WHERE coddep NOT IN (SELECT dep  
FROM empleado))));
```

**Todo director de departamento cobra más que cualquier otro empleado del mismo departamento*

► **No existe un empleado no director cuyo salario sea superior o igual al del director del mismo departamento**

[illegible]

Creación, destrucción e independencia de las RI

18

- **Creación** de una RI (en cualquier momento)
 - El SGBD comprueba: *¿el estado actual de la BD satisface la RI?*
 - ▣ No \Rightarrow RI rechazada
 - ▣ Sí \Rightarrow RI aceptada, y...
 - Es almacenada en el INFORMATION_SCHEMA del catálogo del SGBD (metadatos)
 - La regla es activada (entra en vigor)
- **Destrucción** de una RI
 - ▣ El sistema elimina su definición del INFORMATION_SCHEMA
- **Una regla de integridad es independiente de cualquier aplicación** específica que acceda a la base de datos
 - ▣ No contiene parámetros
 - ▣ Tampoco *variables host*
 - Referencias a variables de los programas de aplicación

Características adicionales (**pseudo-RI**)

19

- El **SGBD rechaza todo intento de INSERT o UPDATE que infrinja una especificación de tipo de datos**
 - *Ejemplo: introducir un valor CHAR en una columna definida como NUMBER
 - ▣ Una especificación de tipo de datos puede verse como una “forma primitiva” de *restricción de integridad de dominio*
 - ▣ Un incumplimiento de una RI de dominio o de tipo de datos se detecta **en tiempo de ejecución** (... ¡compila!)
- El **SGBD rechaza todo intento de INSERT o UPDATE sobre una vista, si incumple la condición de definición de la vista**
 - ▣ Siempre que se haya especificado la “opción de verificación” en la definición de la vista (WITH CHECK OPTION)
 - Ver “Vista relacional” más adelante en este tema

Comprobación de restricciones

20

- En general, **el SGBD comprueba una RI de inmediato, como último paso de la ejecución de una sentencia SQL**
 - ▣ Si la RI es incumplida, *la sentencia es cancelada y no tiene efecto en la BD*

INSERT INTO empleado (nombre, apellido, nss, dni, dep, salario, nssjefe, ciudad, fechanacim, est_civil)

VALUES ('SALUSTIANO', 'SOMONTANO', 987, '98S', 'D3', 980, 333, 'YECLA', TO_DATE('31-MAY-90', 'dd-MON-yyyy'), 'C');

► **Comprobación de reglas de integridad:**

- ▣ Tipos de datos y valores (*CHECK nssjefe <> nss, est_civil, salario positivo*)
- ▣ Integridad referencial (claves ajenas):
 - *existe una fila en DEPARTAMENTO con coddep = D3*
 - *existe una fila en EMPLEADO con nss = 333*
- ▣ Aserto RI1a (*salario ≥ 900€ al ser del departamento D3*)
- ▣ Aserto RI5_... (*salario inferior al del director de su departamento*)
- ▣ ...

Comprobación de restricciones

21

UPDATE empleado **SET** dep = 'D3' WHERE nss = 543; -- Gumersinda Mimos

► **Comprobación de reglas de integridad:**

- ▣ Tipo de datos de dep: *el valor indicado es compatible*
- ▣ Integridad referencial (clave ajena):
 - *existe una fila en DEPARTAMENTO con coddep = 'D3'*
- ▣ Aserto RI1a (*entra al departamento D3: su salario debe ser $\geq 900\text{€}$*)
- ▣ Aserto RI5_... (*salario inferior al del director de su nuevo departamento*)

DELETE empleado WHERE nss = '543'; -- Gumersinda Mimos

► **Comprobación de reglas de integridad:**

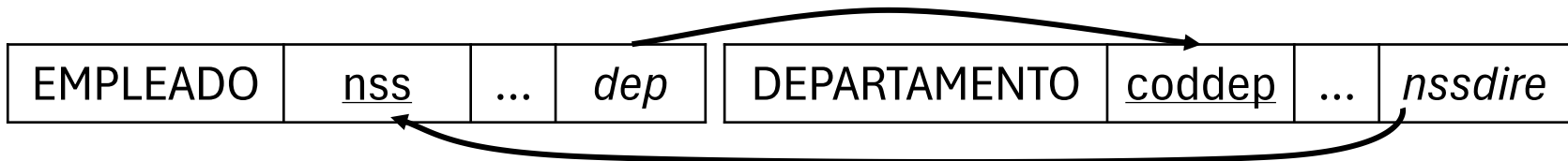
- ▣ Integridad referencial (clave ajena):
 - *¿existe alguna fila en EMPLEADO con nssjefe = '543'?*
 - *¿hay alguna fila en DEPARTAMENTO con nssdire = '543'?*
 - *¿existe alguna fila en FAMILIAR con nssemp = '543'?*
- ▣ Aserto RI4_... (*¿se queda su departamento sin empleados?*)

Comprobación de restricciones

22

- Pero a veces es necesario que ciertas **restricciones no sean comprobadas hasta pasado un tiempo**, pues si se hiciera de inmediato siempre fracasarían

► 2 tablas, inicialmente **vacías**, entre las que hay un **ciclo referencial**



```

CREATE TABLE EMPLEADO (
  nss NUMBER(12) PRIMARY KEY,
  dep CHAR(3) NOT NULL,
  ...,
  CONSTRAINT emp_fk_dep
  FOREIGN KEY (dep)
  REFERENCES DEPARTAMENTO(coddep)...
...);
  
```

```

CREATE TABLE DEPARTAMENTO (
  coddep CHAR(3) PRIMARY KEY,
  nombre VARCHAR(20) NOT NULL,
  nssdire NUMBER(12) NOT NULL,
  CONSTRAINT dep_fk_emp
  FOREIGN KEY (nssdire)
  REFERENCES EMPLEADO(nss)...
...);
  
```

Comprobación de restricciones

23

- Vamos a *poblar* las tablas (que aún están VACÍAS)...
INSERT INTO **empleado** (nss, nombre, ..., **dep**, ...)
VALUES(234, 'FILOMENA', ..., '**D1**', ...);
 - ▶ Comprobación de reglas de integridad:
 - ▣ Tipos de datos y valores ... OK
 - ▣ Integridad referencial:
 - ¿existe una fila en *DEPARTAMENTO* con *coddep* = 'D1'? ... **KO!!**
- Probemos a empezar introduciendo datos en la otra
INSERT INTO **departamento** (coddep, nombre, **nssdire**)
VALUES('D1', 'ADMINISTRACION', **111**);
 - ▶ Comprobación de reglas de integridad:
 - ▣ Tipos de datos y valores ...OK
 - ▣ Integridad referencial:
 - ¿existe una fila en *EMPLEADO* con *NSS* = 111? ... **KO!!**

Comprobación de restricciones

24

- Seguimos intentándolo sin dar valor a “dep” ni a “nssdire”...
INSERT INTO **empleado** (nss, nombre, ..., **dep**, ...)
VALUES(234, 'FILOMENA', ..., **NULL**, ...);
▶ Comprobación de reglas de integridad:
 - ▣ Tipos de datos y valores: **“dep” no puede ser nulo ...KO!!**INSERT INTO **departamento** (coddep, nombre, **nssdire**)
VALUES('D1', 'ADMINISTRACION', **NULL**);
▶ Comprobación de reglas de integridad:
 - ▣ Tipos de datos y valores: **“nssdire” no puede ser nulo ...KO!!**
- Vemos que, **con el chequeo inmediato de las RI de columna**, tanto de clave ajena (integridad referencial) como la de NOT NULL, **todo INSERT** de una fila en EMPLEADO o en DEPARTAMENTO **falla**: nunca encuentra la fila referenciada en la otra tabla...

¿Cuál sería la solución?

Comprobación de restricciones

25

- Solución: **alguna de las restricciones NO debe comprobarse de forma inmediata**
 - ▣ *En nuestro ejemplo:*
 - Conseguir que la **RI** de **clave ajena** “*emp_fk_dep*” **no se compruebe**
 - Que quede desactivada un tiempo
 - Insertar filas en *EMPLEADO*
 - Da igual los valores de su columna “*dep*”: no se comprueban
 - Insertar filas en *DEPARTAMENTO*
 - Los valores en “*nssdire*” sí deben existir en “*nss*” de *EMPLEADO*
 - **Reactivar** la comprobación de **la clave ajena** “*emp_fk_dep*”
 - Para cada fila de *EMPLEADO* el SGBD comprueba que el valor de “*dep*” existe en la columna “*coddep*” de *DEPARTAMENTO*
- Hay que usar la sentencia ALTER TABLE para **desactivar** (inhabilitar) **temporalmente una restricción**

Activación/Desactivación de RI

26

- **Solución** al problema de la inserción de filas en un ciclo referencial:

```
ALTER TABLE empleado
DISABLE CONSTRAINT emp_fk_dep;
INSERT INTO empleado(...)
VALUES (...);
...
INSERT INTO empleado(...)
VALUES (...);
INSERT INTO departamento(...)
VALUES (...);
...
INSERT INTO departamento(...)
VALUES (...);
ALTER TABLE empleado
ENABLE CONSTRAINT emp_fk_dep;
```

Desactivar una de las claves ajenas:
es necesario saber su nombre

Insertar filas en la tabla en la que se ha desactivado la clave ajena: las referencias a la otra tabla NO se comprueban (se han desactivado)

Insertar filas en la otra tabla: las referencias hacia la 1ª tabla sí se comprueban (no se han desactivado)

Reactivar la restricción de clave ajena: en este momento se **comprueban las referencias** (que se habían desactivado)

Para finalizar...

27

- Nos interesa el **soporte de RI declarativo**
No nos centraremos (en este tema) en la implementación mediante...
 - ▣ Procedimientos o funciones almacenados,
 - ▣ Disparadores (*triggers*)
- Recordatorio. Las **RI son parte de los metadatos**: son mantenidas en el INFORMATION_SCHEMA del catálogo
 - ▣ El SGBD (el módulo *Subsistema de Integridad*)
 - controla las operaciones de usuario (INSERT, UPDATE, DELETE...)
 - para asegurar que NO incumplen las reglas de integridad
- Una **BD en un estado de integridad es correcta** porque...
 - ▣ No incumple ninguna RI conocida por el SGBD, es decir,
 - ▣ Satisface el AND lógico de todas las RI definidas en su esquema



Vista relacional

28

- Es una “tabla” obtenida como **resultado de la ejecución de una consulta**
- Definida mediante una SELECT que extrae o deriva datos de ciertas **tablas base** (otras tablas o vistas)
 - ▣ Presenta datos contenidos en una o más tablas (o vistas)
 - ▣ Permite tratar el resultado de una consulta como una tabla
 - ▣ Por eso se le llama también *tabla derivada* o *tabla virtual*

EMPLEADO(nombre, apellido, nss, dni, fechanacim, ciudad, est_civil, salario, dep, nssjefe)

* Definición de la vista PERSONAL con base en la tabla EMPLEADO

```
CREATE VIEW personal
AS SELECT nss, nombre, nssjefe, dep
FROM empleado;
```

```
SELECT *
FROM personal;
```



Característica fundamental de las vistas

29

- ❑ **Actualización Permanente**
 - ▣ La información que deja ver está siempre actualizada
 - ▣ El responsable de esto es el SGBD
- ❑ El “secreto”:
 - ▣ La **vista** no se ‘**crea**’ cuando se define, sino **cada vez que se consulta**
 - ▣ **Una vista no contiene información, sino que “deja ver información” que está almacenada en sus tablas base**

```
CREATE VIEW personal_no_asignado
AS SELECT nombre, apellido
FROM empleado
WHERE dep IS NULL;
```

Si después se inserta en EMPLEADO dos **nuevas filas**:
el empleado **A** sin departamento y
el **B** asignado al departamento D1
entonces el resultado de
SELECT *
FROM personal_no_asignado;
incluirlá A y no incluirlá B

Usos de las vistas

30

- Dar nombre y **almacenar consultas complejas**
 - ▣ Y poder re-ejecutarlas sucesivamente con facilidad
 - ▣ *Ejemplo:* una vista puede realizar cálculos (aritméticos o de agregados) complejos con la información de las tablas base. Definiendo esta consulta como una vista...
 - Ejecutarla implica simplemente hacer `SELECT * FROM` la vista
 - Los cálculos se realizarán cada vez que la vista sea consultada
- Presentar los datos con una **perspectiva diferente** a la de las tablas base
 - ▣ Las columnas de una vista pueden ser renombradas sin afectar a las tablas base
 - ▣ Una vista puede incluir columnas “nuevas”, resultado de combinar columnas de las tablas base

Usos de las vistas

31

- **Ocultar la complejidad de los datos**
 - ▣ Ejemplo: una vista definida con JOIN oculta que la información que permite ver se deriva de combinar varias tablas

- Y, por tanto, simplificar las sentencias para el usuario**
 - ▣ Los usuarios redactan sentencias más sencillas
 - ▣ *Ejemplo:* una vista definida con JOIN y funciones agregadas, permite al usuario obtener información sin saber cómo realizar JOIN, ni cómo aplicar las funciones agregadas
 - Simplemente consultando la vista con SELECT ... FROM ...

Usos de las vistas

32

- **Proporcionar seguridad**
 - ▣ Permiten restringir el acceso a un conjunto de filas y/o de columnas predeterminadas de una tabla
 - La cláusula WHERE permite seleccionar sólo algunas filas
 - Las columnas de la vista pueden no incluir algunas de las columnas de una tabla base

- **Aislar a las aplicaciones de los cambios en la definición de las tablas base**
 - ▣ *Ejemplo:* si la vista PERSONAL hace referencia a 4 columnas de la tabla EMPLEADO y se añade otra columna “puesto” a EMPLEADO, no se ven afectadas ni la definición de PERSONAL ni las aplicaciones que usan (acceden a) PERSONAL

Nombres de columna

33

- Por defecto, la vista **'hereda'** los nombres de las columnas seleccionadas desde las tablas base

```
CREATE VIEW familiar_empleado
AS SELECT e.nombre, f.nombre, f.parentesco
FROM empleado e JOIN familiar f ON e.nss = f.nssemp;
```

nombres de columna
que hereda la vista

- Definición de **nuevos nombres** para columnas de la vista

```
CREATE VIEW familiar_empleado (empleado, familiar, parentesco)
AS SELECT e.nombre, f.nombre, f.parentesco
FROM empleado e JOIN familiar f ON e.nss = f.nssemp;
```

- **Obligatorio** cuando alguna columna es el **resultado de una operación aritmética, de concatenación (||) o una función de agregados**

```
CREATE VIEW info_depto (nombre_dep, num_emps, sal_total)
AS SELECT d.nombre, COUNT(*), SUM(salario)
FROM departamento d JOIN empleado e
ON d.coddep=e.dep
GROUP BY d.nombre;
```

Dos tablas más del esquema de BD “Empresa”

34

EMPLEADO

nombre	apellido	<u>nss</u>	dni	fechanacim	ciudad	est_civil	salario	<u>nssjefe</u>	dep
JONÁS	SOLANO	123	11A	10/10/1945	MURCIA	P	1100	111	D1
RIGOBERTA	CALAVERA	321	21C	12/11/1974	YECLA	C	900	333	D3
EUSEBIO	MULETAS	222	22B	01/01/1969	TOTANA	D	2100	123	D2
MACARENO	SOSO	111	23D	06/04/1944	JUMILLA	S	1100	NULL	D1
CASIANA	FABERGÉ	333	33B	15/06/1943	MURCIA	V	920	123	D3
FILOMENA	RASCAS	234	34E	18/07/1970	MURCIA	C	1100	111	D1
GUMERSINDA	MIMOS	543	45F	10/02/1980	PINOSO	P	850	NULL	NULL

PROYECTO

titulo	<u>codproy</u>	lugar	dep
PROYECTO X	X	MURCIA	D2
PROYECTO Y	Y	YECLA	D3
PROYECTO Z	Z	TOTANA	D4

DEDICACION

<u>empleado</u>	<u>proyecto</u>	horas
123	X	32.5
543	Z	40.0
321	Y	7.5
111	X	20.0
333	Y	10.0
222	Z	10.0
543	Y	20.0

Selección de todas las columnas

35

- Uso de **SELECT *** en la consulta de definición de la vista
 - ▣ Las columnas de la vista son **todas** las del resultado

```
CREATE VIEW departamento_oficina
AS SELECT *
FROM departamento
NATURAL JOIN oficina;
```

Todas las columnas de
DEPARTAMENTO y OFICINA

```
CREATE VIEW veterano
AS SELECT *
FROM empleado
WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy');
```

Todas las columnas
de EMPLEADO

```
CREATE VIEW empleado_preferente
AS SELECT *
FROM empleado
WHERE nss IN (SELECT nssemp
              FROM familiar)
AND salario < 1000;
```

Todas las columnas
de EMPLEADO

Nuevas columnas

36

- Es posible generar **nuevas columnas** en la vista, aplicando operaciones sobre las extraídas

CREATE VIEW empleado_proyecto (empleado, proyecto, horas)
AS SELECT e.nombre || ' ' || e.apellido, p.titulo, d.horas
FROM (empleado e **JOIN** dedicacion d **ON** e.nss = d.empleado)
JOIN proyecto p **ON** d.proyecto = p.codproy ;

SELECT *
FROM empleado_proyecto;

La vista muestra el nombre y apellido de cada empleado (**en una sola columna**), el título de cada proyecto al que está asignado y cuántas horas dedica al mismo

empleado	proyecto	horas
JONÁS SOLANO	PROYECTO X	32.5
GUMERSINDA MIMOS	PROYECTO Z	40.0
RIGOBERTA CALAVERA	PROYECTO Y	7.5
MACARENO SOSO	PROYECTO X	20.0
CASIANA FABERGÉ	PROYECTO Y	10.0
EUSEBIO MULETAS	PROYECTO Z	10.0
GUMERSINDA MIMOS	PROYECTO Y	20.0

Vista como “tabla base” de otra

37

- Una **vista** puede aparecer en el FROM de la SELECT de definición de **otra vista**

```
CREATE VIEW directivo (nss, nombre, coddep, departamento)
AS SELECT p.nss, p.nombre, d.coddep, d.nombre
   FROM personal p JOIN departamento d
        ON p.dep = d.coddep
   WHERE p.nss IN (SELECT nssjefe
                  FROM empleado);
```

PERSONAL es una **vista** creada anteriormente

Vemos que una vista puede ser “tabla base” de otra vista

```
SELECT nombre, departamento
FROM directivo
ORDER BY departamento;
```

nombre	departamento
MACARENO	ADMINISTRACIÓN
JONÁS	ADMINISTRACIÓN
CEFERINA	FORMACION
CASIANA	PERSONAL

```
❶ CREATE VIEW personal
AS SELECT nss, nombre, nssjefe, dep
   FROM empleado;
```

Oracle. Creación de vistas

38

```
CREATE [ OR REPLACE ] [ [ NO ] FORCE ] VIEW nombre_vista  
  [ (lista_nombres_columnas) ]  
  AS consulta_definición  
  [ WITH { READ ONLY  
        | CHECK OPTION } [ CONSTRAINT nombre_restricción ] ]
```

❗ La CHECK OPTION la veremos después

- ❑ **OR REPLACE** permite reemplazar la definición de la vista en el diccionario de datos (y *recompila* la vista), evitando eliminarla y volver a crearla
- ❑ Si la vista se definió con SELECT * y se añade columnas a las tablas base, la vista no incluirá esas nuevas columnas hasta ejecutar CREATE OR REPLACE
- ❑ **FORCE** permite crear la vista siempre, aunque no existan las tablas base o el propietario de la vista no tenga permisos sobre ellas
- ❑ **WITH READ ONLY** prohíbe hacer UPDATE, INSERT y DELETE sobre las tablas base a través de la vista

Oracle. Modificación de vistas

39

```
CREATE VIEW empleado_familia_numerosa
AS SELECT * FROM empleado
WHERE nss IN (SELECT nssemp FROM familiar
              WHERE parentesco LIKE 'HIJ_'
              GROUP BY nssemp
              HAVING COUNT(*) >= 4);
```

Si se desea **modificar la sentencia de definición** de una vista ya existente, hay que usar **CREATE OR REPLACE**

```
CREATE OR REPLACE VIEW empleado_familia_numerosa
AS SELECT nss, nombre, apellido, dep
FROM empleado
WHERE nss IN (SELECT nssemp FROM familiar
              WHERE parentesco LIKE 'HIJ_'
              GROUP BY nssemp
              HAVING COUNT(*) >= 3);
```

Elegir algunas **columnas** y cambiar el **nº** de familiares

```
CREATE OR REPLACE VIEW empleado_familia_numerosa
AS SELECT nss, nombre, dep FROM empleado
WHERE nss IN (SELECT nssemp FROM familiar
              WHERE parentesco LIKE 'HIJ_'
              GROUP BY nssemp
              HAVING COUNT(*) >= 3);
```

“**Eliminar**” una **columna** de la vista (apellido)

Consulta de datos a través de vistas

40

- ❑ Las vistas **no tienen limitación en operaciones de consulta**
 - ❑ El usuario no distingue si el elemento al que accede es una tabla o una vista
- * *Nombres de los empleados y de sus hijos/as*
SELECT empleado, familiar
FROM **familiar_empleado**
WHERE parentesco LIKE 'HIJ_';
 - * *Datos del departamento 'Investigación'*
SELECT * FROM **info_depto**
WHERE nombre_dep = 'INVESTIGACION';
 - * *Nombres y apellidos de empleados que trabajan en el proyecto 'PROYECTO X'*
SELECT empleado
FROM **empleado_proyecto**
WHERE proyecto = 'PROYECTO X';

Consulta de datos a través de vistas

41

- El SGBD traduce cualquier sentencia SELECT sobre la vista a **una expresión equivalente sobre sus tablas base**: reemplaza el nombre de la vista por su consulta de definición, la combina con la sentencia original y la ejecuta

CREATE OR REPLACE

VIEW **veterano**

```
AS SELECT nombre, dni, nss,
      fechanacim, dep
FROM empleado
WHERE fechanacim <
      TO_DATE('01/01/1970',
      'dd/mm/yyyy');
```

Sentencia de usuario	Traducción (lo que realmente se ejecuta)
SELECT * FROM veterano WHERE dep = 'D1';	SELECT nombre, dni, nss, fechanacim, dep FROM empleado WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy') AND dep = 'D1';
SELECT v.nombre, d.nssdire FROM veterano v JOIN departamento d ON v.dep = d.coddep;	SELECT v.nombre, d.nssdire FROM empleado v JOIN departamento ON v.dep=d.coddep WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy');
SELECT nombre FROM veterano WHERE nss IN (SELECT nssemp FROM familiar WHERE parentesco <> 'HIJO');	SELECT nombre FROM empleado WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy') AND nss IN (SELECT nssemp FROM familiar WHERE parentesco <> 'HIJO');



Manipulación de datos a través de vistas

42

- El SGBD traduce (si puede) cualquier sentencia INSERT, UPDATE y DELETE sobre la vista a una **expresión equivalente sobre sus tablas base**
- Una vista **no contiene datos**, así que un INSERT, UPDATE o DELETE sobre una vista **no** puede modificar “el contenido de la vista”...
 - ▶ Hay que **modificar los datos almacenados en las tablas base**, para que la siguiente vez que se consulte la vista, los datos se vean así cambiados

Sentencia de usuario	Traducción (lo que realmente se ejecuta)
INSERT INTO veterano (nombre, apellido, nss, dni, fechanacim, salario, dep) VALUES ('EVA','EME',789,'78E', TO_DATE('09/07/1967', 'dd/mm/yyyy'), 980,'D4');	INSERT INTO empleado (nombre, apellido, nss, dni, fechanacim, salario, dep) VALUES ('EVA', 'EME', 789, '78E', TO_DATE('09/07/1967', 'dd/mm/yyyy'), 980, 'D4');
UPDATE veterano SET dep = 'D1' WHERE dep = 'D2';	UPDATE empleado SET dep = 'D1' WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy') AND dep = 'D2';
DELETE FROM veterano WHERE dep IS NULL;	DELETE FROM empleado WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy') AND dep IS NULL;

Manipulación de datos a través de vistas

43

- Sin embargo, la **actualización de datos a través de vistas sí tiene limitaciones**
- Por un lado, **algunas actualizaciones no tienen sentido**

- *Ejemplo:* columnas definidas mediante funciones de agregados

```
UPDATE info_depto  
  SET sal_total = 10800  
  WHERE nombre_dep = 'INVESTIGACION' ;
```

► sal_total se calcula: para cada departamento contiene la suma de salarios individuales de sus empleados

¿qué sentido tiene modificar su valor “a mano”?

```
CREATE VIEW info_depto (nombre_dep,  
                        num_emps, sal_total)  
AS SELECT d.nombre, COUNT(*), SUM(salario)  
   FROM departamento d JOIN empleado e  
                        ON d.coddep=e.dep  
   GROUP BY d.nombre;
```

Manipulación de datos a través de vistas

44

- Por otro lado, actualizar a través de una **vista definida sobre varias tablas base** suele dar problemas, pues puede haber ambigüedad
 - ▣ Una modificación puede traducirse a **dos o más actualizaciones distintas de las tablas base** de la vista

- ▣ Ejemplo: recordemos la definición de esta vista

```
CREATE VIEW empleado_proyecto (empleado, proyecto, horas)
AS SELECT nombre || ' ' || apellido, titulo, horas
FROM (empleado e
      JOIN dedicacion d ON e.nss = d.empleado)
      JOIN proyecto p ON d.proyecto = p.codproy ;
```

- ▣ Se desea realizar esta modificación:

```
UPDATE empleado_proyecto
SET proyecto = 'PROYECTO X'
WHERE empleado = 'GUMERSINDA MIMOS'
AND proyecto = 'PROYECTO Z';
```

Se desea cambiar a
GUMERSINDA MIMOS
del PROYECTO Z
al PROYECTO X

Manipulación de datos a través de vistas

45

- ¿Qué muestra la vista, antes de modificarla?

```
SELECT * FROM empleado_proyecto;
```

empleado	proyecto	horas
JONÁS SOLANO	PROYECTO X	32.5
GUMERSINDA MIMOS	PROYECTO Z	40.0
RIGOBERTA CALAVERA	PROYECTO Y	7.5
MACARENO SOSO	PROYECTO X	20.0
CASIANA FABERGÉ	PROYECTO Y	10.0
EUSEBIO MULETAS	PROYECTO Z	10.0
GUMERSINDA MIMOS	PROYECTO Y	20.0

GUMERSINDA MIMOS está asignada al PROYECTO Z y al PROYECTO Y

Obsérvese que EUSEBIO MULETAS también está asignado al PROYECTO Z

- Con el UPDATE se desea desvincular a GUMERSINDA MIMOS del PROYECTO Z y asignarla al PROYECTO X
- Veamos **DOS actualizaciones** de las tablas base **como traducción** del UPDATE...

Manipulación de datos a través de vistas

46

❑ Actualización de las tablas base ①



UPDATE **dedicacion**

SET proyecto = (SELECT codproy FROM proyecto
WHERE titulo = 'PROYECTO X')

WHERE empleado = (SELECT nss FROM empleado
WHERE nombre = 'GUMERSINDA'
AND apellido = 'MIMOS')

AND proyecto = (SELECT codproy FROM proyecto
WHERE titulo = 'PROYECTO Z');

3º Obtiene el identificador del PROYECTO X: **X**

1º Obtiene el identificador de la empleada: **543**

2º Obtiene el identificador del PROYECTO Z: **Z**

DEDICACION **ANTES**

empleado	proyecto	horas
123	X	32.5
543	Z	40.0
321	Y	7.5
111	X	20.0
333	Y	10.0
222	Z	10.0
543	Y	20.0

❖ El UPDATE modifica los vínculos en DEDICACION: Cada fila que relacionaba el nss de 'GUMERSINDA MIMOS' con el código del 'PROYECTO Z', ahora lo relaciona con el código del 'PROYECTO X'

DEDICACION **DESPUÉS**

empleado	proyecto	horas
123	X	32.5
543	X	40.0
321	Y	7.5
111	X	20.0
333	Y	10.0
222	Z	10.0
543	Y	20.0

Manipulación de datos a través de vistas

47

❑ Actualización de las tablas base ②



```
UPDATE proyecto SET titulo = 'PROYECTO X'
WHERE titulo = 'PROYECTO Z';
```

PROYECTO ANTES

titulo	<u>codproy</u>	lugar	dep
PROYECTO X	X	MURCIA	D2
PROYECTO Y	Y	YECLA	D3
PROYECTO Z	Z	TOTANA	D4

Dos proyectos con igual título: ¡CAOS!

PROYECTO DESPUÉS

titulo	<u>codproy</u>	lugar	dep
PROYECTO X	X	MURCIA	D2
PROYECTO Y	Y	YECLA	D3
PROYECTO X	Z	TOTANA	D4

❖ Produce igual efecto que ① porque cambia título en PROYECTO: al visualizar la vista, mostrará 'PROYECTO X' para todo empleado que antes aparecía junto con 'PROYECTO Z'


Consigue que GUMERSINDA MIMOS aparezca asignada al PROYECTO X, pero **también lo está EUSEBIO MULETAS** ¡más CAOS!

SELECT * FROM empleado_proyecto;

empleado	proyecto	horas
JONÁS SOLANO	PROYECTO X	32.5
GUMERSINDA MIMOS	PROYECTO X	40.0
RIGOBERTA CALAVERA	PROYECTO Y	7.5
MACARENO SOSO	PROYECTO X	20.0
CASIANA FABERGÉ	PROYECTO Y	10.0
EUSEBIO MULETAS	PROYECTO X	10.0
GUMERSINDA MIMOS	PROYECTO Y	20.0

Manipulación de datos a través de vistas

48

- *Mala noticia:* el SGBD **no puede saber qué actualización es correcta y cuál no**, y si hubiera varias correctas, no sabría cuál elegir como la más adecuada
- Así que **no se garantiza que “toda vista sea actualizable”**
 - ▣ Es decir, que permita cambios a través de ella
- Una **vista** sería **actualizable** si implicara **una única actualización posible de las tablas base** 
- Por tanto, en general podemos afirmar que...
 - ▣ Una **vista con una sola tabla base**
 - **SÍ es actualizable** si sus columnas **contienen una clave** (primaria o alternativa) de la tabla base
 - Así se establece una correspondencia entre cada fila de la vista y una única fila de la tabla base
 - ▣ Una **vista definida sobre varias tablas mediante JOIN (reunión)**
 - **NO** es actualizable
 - ▣ Una **vista definida con agrupación y funciones agregadas**
 - **NO** es actualizable

Manipulación de datos a través de vistas

49

□ Vistas actualizables. Ejemplos.

```
CREATE VIEW personal
AS SELECT nss, nombre, nssjefe, dep
FROM empleado;
```

```
CREATE VIEW veterano
AS SELECT *
FROM empleado
WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy');
```

```
CREATE VIEW empleado_familia_numerosa
AS SELECT nss, nombre, dep
FROM empleado
WHERE nss IN (SELECT nssemp
              FROM familiar
              WHERE parentesco LIKE 'HIJ_'
              GROUP BY nssemp
              HAVING COUNT(*) >= 3);
```

Estas vistas sólo tienen una tabla base (EMPLEADO) e incluyen el atributo “nss”, clave primaria

Son vistas que SÍ permiten modificar la tabla base a través de ellas

Manipulación de datos a través de vistas

50

□ Vistas NO actualizables. Ejemplos.

```
CREATE VIEW familiar_empleado (empleado,familiar,parentesco)
AS SELECT e.nombre, f.nombre, f.parentesco
FROM empleado e JOIN familiar f ON e.nss = f.nssemp;
```

```
CREATE VIEW info_depto (nombre_dep, num_emps, sal_total)
AS SELECT D.nombre, COUNT(*), SUM(salario)
FROM departamento d
JOIN empleado e ON d.coddep = e.dep
GROUP BY d.nombre;
```

```
CREATE VIEW departamento_oficina
AS SELECT *
FROM departamento
NATURAL JOIN oficina
```

Estas vistas están definidas mediante JOIN, y/o contienen funciones de agregados para calcular los valores de las columnas

Son vistas que NO permiten modificar la tabla base a través de ellas

Manipulación de datos a través de vistas

51

□ Opción de verificación de vistas

- ▣ Veamos qué es mediante un ejemplo

```
CREATE VIEW precario
AS SELECT nombre, apellido, nss, dni, salario, dep
FROM empleado
WHERE salario < 550 ;
```

```
SELECT * FROM precario;
```

nombre	apellido	...	dni	salario	...
ROSAURO	GUZ	...	11G	350	...
ARGIMIRA	BOL	...	22A	480	...
TASIO	TOR	...	33T	500	...
...

Esta **vista** **SÍ** es **actualizable**:
Sí permite modificar la tabla
base a través de ella

Manipulación de datos a través de vistas

52

□ Opción de verificación de vistas

**¿Qué pasaría al ejecutar estas sentencias?*

INSERT INTO precario (nombre, apellido, nss, dni, salario, dep)
VALUES ('DIMAS', 'PI', 555, '55D', **1025**, 'D1');

```
CREATE VIEW precario
AS SELECT nombre, apellido, nss,
           dni, salario, dep
FROM empleado
WHERE salario < 550 ;
```

El salario es superior a 550

nombre	apellido	...	dni	salario	...
ROSAURO	GUZ	...	11G	350	...
ARGIMIRA	BOL	...	22A	480	...
TASIO	TOR	...	33T	500	...
...

El INSERT se ejecuta sin errores. Pero
un SELECT * FROM PRECARIO
NO muestra la nueva fila

UPDATE precario SET salario = **585**
WHERE dni = '**33T**'; -- antes cobraba 500€

El nuevo salario es superior a 550

nombre	apellido	...	dni	salario	...
ROSAURO	GUZ	...	11G	350	...
ARGIMIRA	BOL	...	22A	480	...
...

El UPDATE se ejecuta sin errores. Pero
un SELECT * FROM PRECARIO, muestra
que "TASIO TOR" **¡ha desaparecido!**

Esto ocurre porque **el INSERT y el UPDATE NO cumplen la condición de definición de la vista** salario < 550

Manipulación de datos a través de vistas

53

- Opción de verificación de vistas

Cláusula **WITH CHECK OPTION**

- ▣ Se debe incluir en la definición (CREATE VIEW) de toda **vista actualizable que se vaya a utilizar para la modificación de datos**
- ▣ Indica al SGBD que **debe comprobar cada INSERT y UPDATE sobre la vista**, y rechazarlo si su realización implicara que la fila nueva o modificada no cumpliera la condición de definición de la vista

CREATE VIEW precario

AS SELECT nombre, apellido, nss, dni, salario, dep

FROM empleado

WHERE salario < 550

WITH CHECK OPTION ;



Así, tanto el INSERT como el UPDATE anteriores fallarían

Oracle. Check Option

54

```
CREATE OR REPLACE NOFORCE VIEW plantilla  
AS SELECT nombre, apellido, nss, dni, salario, dep  
FROM empleado  
WHERE salario < 3000 -- se ocultan datos de "jefazos y jefazas"
```

WITH CHECK OPTION CONSTRAINT plantilla_const;

- Con la cláusula **WITH CHECK OPTION** se crea la vista de manera que se garantice que los INSERT y UPDATE no pueden resultar en filas que la vista no pueda seleccionar
 - ▣ Cumplen las condiciones que hay en el WHERE de la consulta de definición
- Se le puede dar un **nombre** CONSTRAINT *nombre*, que el SGBD usará en los mensajes de error ante un intento de incumplimiento

```
INSERT INTO plantilla  
VALUES('BENITA', 'FA', '758', '11F', 3500, 'D4');
```

- ▶ Error: INSERT no permitido: **rompe la condición de definición.**

Un SELECT * FROM plantilla **no mostraría** la fila de 'BENITA'

Almacenamiento de vistas

55

- Una vista es una tabla virtual o lógica
- **No contiene datos**, por lo que **no ocupa espacio** de almacenamiento
- **Sólo se almacena la definición de la vista** en los metadatos
 - ▣ El texto de la consulta que define la vista, *stored query*
 - ▣ Dentro del INFORMATION_SCHEMA del Catálogo

Oracle. Eliminación de vistas

56

- Sentencia para eliminar una vista

DROP VIEW *vista*;

DROP VIEW plantilla;

DROP VIEW veterano;

- Si la vista eliminada ha sido usada como base para otras vistas, éstas no se eliminan, pero se marcan como INVALID

- La vista DIRECTIVO tiene 2 tablas base: la **vista PERSONAL** y la tabla DEPARTAMENTO

CREATE VIEW

```
directivo (nss, nombre, coddep, departamento)
AS SELECT p.nss, p.nombre, d.coddep, d.nombre
   FROM personal p JOIN departamento d
                        ON p.dep = d.coddep
  WHERE p.nss IN (SELECT nssjefe
                  FROM empleado);
```

- Si se ejecuta **DROP VIEW** **personal**;

... la vista DIRECTIVO sigue existiendo, pero queda marcada como **INVALID**: **no se puede ejecutar**

- SELECT * FROM directivo; -- da **error**

Índices

57

- El concepto de índice no pertenece al nivel conceptual ni lógico, sino al **nivel físico**
 - ▣ Por eso no está incluido en el ANSI SQL
- Pero es importante que, dentro de una asignatura de fundamentos de Bases de Datos, se explique **qué es un índice y para qué se usa**
- Además, comprender **qué es un índice y cómo utilizarlo** es muy valioso para futuros/as ingenieros/as en informática
- Hablemos primero, brevemente, de los **datos almacenados** en la base de datos en disco y de **cómo se accede** a ellos...

Datos almacenados

58

- En el modelo relacional se indica que **las filas dentro de las tablas no están ordenadas**
 - ▣ Quedan en el orden de inserción
- Se puede asumir que **cada tabla** se almacenará **en un fichero** del sistema operativo, en el disco duro
 - ▣ Cada **fila** en un **registro**
 - ▣ Cada valor de **columna** como valor de un **campo**
- Los registros dentro del fichero se almacenan ocupando **bloques o páginas** del disco

Fichero **EMPLEADO**

nss	nombre	...	fechanacim	dep
123	JONÁS		10/10/1945	D1
321	RIGOBERTA		12/11/1974	D3
222	EUSEBIO		01/01/1969	D2
111	MACARENO		06/04/1944	D1
333	CASIANA		15/06/1943	D3
234	FILOMENA		18/07/1970	D1
543	GUMERSINDA		10/02/1980	NULL
444	CEFERINA		30/12/1992	D4
456	PRUDENCIO		22/01/1995	D4

Fichero con **5 bloques**
Cada bloque contiene **2 registros**

Búsqueda de datos almacenados

59

- La ejecución de una consulta SQL implica la búsqueda de ciertas filas con base en ciertos datos y la recuperación de (parte de) su contenido
- **Ejemplos** de búsquedas de datos
 - ▣ *¿Cuál es el departamento de la empleada con nombre 'CASIANA'?*
 - ▣ *¿Cuándo nació el empleado con nss 333?*
 - ▣ *¿Qué empleados pertenecen al departamento 'D1'?*
- ¿Cómo podemos **encontrar una fila concreta** (o varias) **en una tabla**, es decir, un registro concreto en un fichero?
- Opción: **búsqueda secuencial**

```
SELECT dep  
FROM empleado  
WHERE nombre = 'CASIANA';
```

```
SELECT fechanacim  
FROM empleado  
WHERE nss = 333;
```

```
SELECT nss  
FROM empleado  
WHERE dep = 'D1';
```

Búsqueda de datos almacenados

60

□ * *¿Cuándo nació el empleado con nss = 333?*

□ * *¿Qué empleados pertenecen al departamento 'D1'?*

□ Ojo: los datos se almacenan en 5 **bloques** (*páginas*) de disco

Fichero **EMPLEADO**

bloque	nss	nombre	...	fechanacim
B1	123	JONÁS		10/10/1945
	321	RIGOBERTA		12/11/1974
B2	222	EUSEBIO		01/01/1969
	111	MACARENO		06/04/1944
B3	333	CASIANA		15/06/1943
	234	FILOMENA		18/07/1970
B4	543	GUMERSINDA		10/02/1980
	444	CEFERINA		30/12/1992
B5	456	PRUDENCIO		22/01/1995

Búsqueda de datos almacenados

61

- ❑ La **búsqueda secuencial** implica un *gran número de lecturas de bloque* (página)
 - ▣ Copias de bloques de disco a la memoria principal
 - ▶ **No es eficiente**
- ❑ ¿Cómo podemos entonces **buscar de manera eficiente** filas en una tabla (registros en un fichero)?
 - ▣ ‘Eficiente’ significa leer el menor número posible de bloques (páginas) de disco
- ❑ Para esto se utilizan los **índices**

Uso de **índices**

62

- Un índice es una **estructura de datos auxiliar** que permite **acelerar el acceso** a las filas de una tabla **con base en los valores de una o más columnas**
- *Ejemplo: la consulta...*
 - * *¿Cuándo nació el empleado con nss = 333?*
 - Busca datos usando la **columna “nss”**
 - Se podría **acelerar el acceso a las filas de EMPLEADO** creando un **índice con base en los valores de nss**
 - Así, **nss** es la **columna de indexación**

```
SELECT fechanacim  
FROM empleado  
WHERE nss = 333;
```

Construcción de un índice

63

Índice

IDX_NSS_EMPLEADO

nss	bloque
111	B2
123	B1
222	B2
234	B3
321	B1
333	B3
444	B4
456	B5
543	B4

1º Coger todos los valores del campo de indexación

2º Ordenarlos

3º Para cada uno de los valores, crear una **entrada de índice**.
El campo "nss" contendrá el valor.
El campo "bloque" contendrá un puntero al bloque de EMPLEADO en el que está el registro con tal valor en "nss"

Fichero **EMPLEADO**

bloque	nss	nombre	...
B1	123	JONÁS	
	321	RIGOBERTA	
B2	222	EUSEBIO	
	111	MACARENO	
B3	333	CASIANA	
	234	FILOMENA	
B4	543	GUMERSINDA	
	444	CEFERINA	
B5	456	PRUDENCIO	

Obsérvese que, por ejemplo, no hay un valor de nss = 555 en el índice... Porque **no existe** tal valor en EMPLEADO.nss

Estructura y contenido de un índice

64

- Un índice contiene **entradas** (registros) con **dos campos**, para almacenar...
 - ▣ Cada **valor** existente en la **columna de indexación**
 - Extraídos de la tabla: valores de dicha columna en las filas
 - Es decir, valores de dicho campo en los registros del fichero
 - ▣ Un **puntero al bloque** dentro del fichero de datos que contiene el registro con dicho valor del campo de indexación
- Es posible tener **varios índices**, sobre campos distintos, en una misma tabla (en un mismo fichero)
- El índice ocupa poco espacio: cabe en un único bloque de disco

Búsqueda mediante índice

65

- Las **entradas** están **ordenadas** según el **valor del campo de indexación**

- Es similar a un *glosario o índice final de un libro de texto*

- De esta forma es posible realizar **búsquedas binarias** sobre el índice

Índice
IDX_NSS_EMPLEADO

nss	bloque
111	B2
123	B1
222	B2
234	B3
321	B1
333	B3
444	B4
456	B5
543	B4

Ordenado
por "nss"

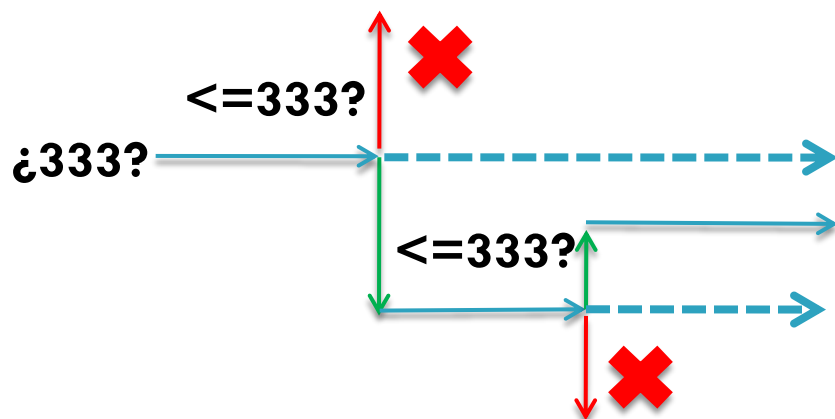


Búsqueda mediante índice

66

* ¿Cuándo nació
el empleado con *nss* = 333?

**Búsqueda
binaria**



Ordenado por "nss"

nss	bloque
111	B2
123	B1
222	B2
234	B3
321	B1
333	B3
444	B4
456	B5
543	B4

Fichero **EMPLEADO**

bloque	nss	fechanacim
B1	123	10/10/1945
	321	12/11/1974
B2	222	01/01/1969
	111	06/04/1944
B3	333	15/06/1943
	234	18/07/1970
B4	543	10/02/1980
	444	30/12/1992
B5	456	22/01/1995

Conviene crear índices sobre...

67

- Toda **clave primaria** (PRIMARY KEY) y toda clave **alternativa** (UNIQUE)
 - ▣ **Oracle** ya lo hace automáticamente
- Un campo (**columna**) **no clave** que se usa muy frecuentemente...
 - ▣ en condiciones de selección (condiciones en el WHERE)
 - ▣ en operaciones de reunión (JOIN en el FROM)
- Pero siempre que la tabla...
 - ▣ Sea **grande**: contiene muchos datos
 - ▣ Sus **consultas más frecuentes** recuperan un bajo porcentaje de los datos (**menos del 20% de las filas**)

Índice sobre campo no clave

68

* ¿Cuáles son los empleados del departamento 'D1'?

Varios punteros
por entrada

dep	bloque		
D1	B1	B2	B3
D2	B2		
D3	B1	B3	
D4	B4	B5	



Ordenado por "dep"

El índice aún cabe en
un **único** bloque

No se indexan las filas con NULL
en la columna de indexación

Fichero **EMPLEADO**

bloque	nss	nombre	...	fechanacim	dep
B1	123	JONÁS		10/10/1945	D1
	321	RIGOBERTA		12/11/1974	D3
B2	222	EUSEBIO		01/01/1969	D2
	111	MACARENO		06/04/1944	D1
B3	333	CASIANA		15/06/1943	D3
	234	FILOMENA		18/07/1970	D1
B4	543	GUMERSINDA		10/02/1980	NULL
	444	CEFERINA		30/12/1992	D4
B5	456	PRUDENCIO		22/01/1995	D4

Oracle. Creación de índices

69

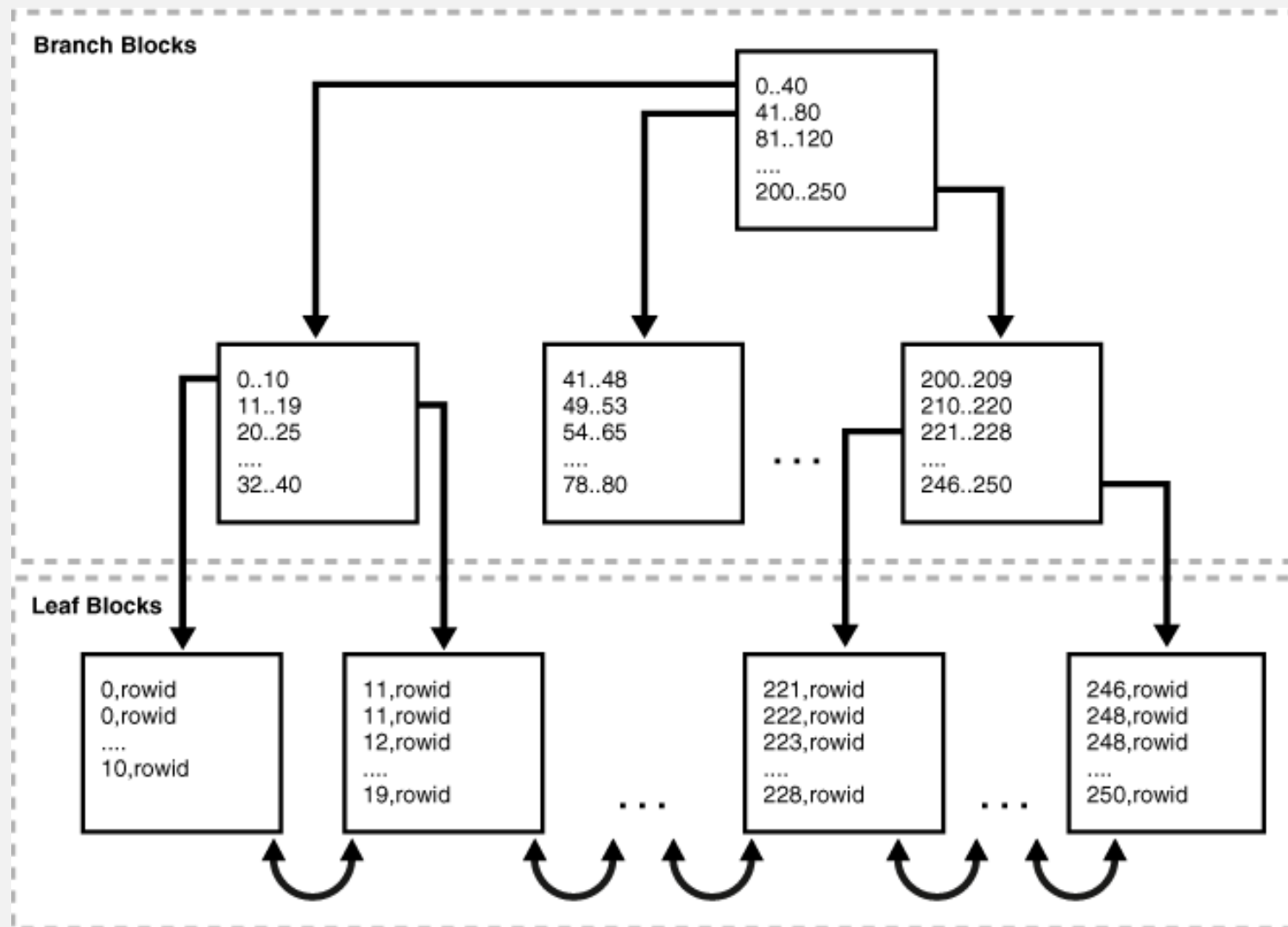
```
CREATE INDEX nombre_índice  
  ON nombre_tabla (columna [ASC | DESC]  
    [, columna [ASC | DESC]...])  
  ...;
```

- ❑ Estructuras de acceso opcionales asociadas con tablas (o con otros elementos Oracle) y mantenidas por el SGBD
- ❑ Son lógica y físicamente **independientes** de los datos de la tabla asociada
- ❑ Los más comunes son los índices en árbol B (*B-tree index*)

Oracle. Índices en árbol B (*B-tree index*)

70

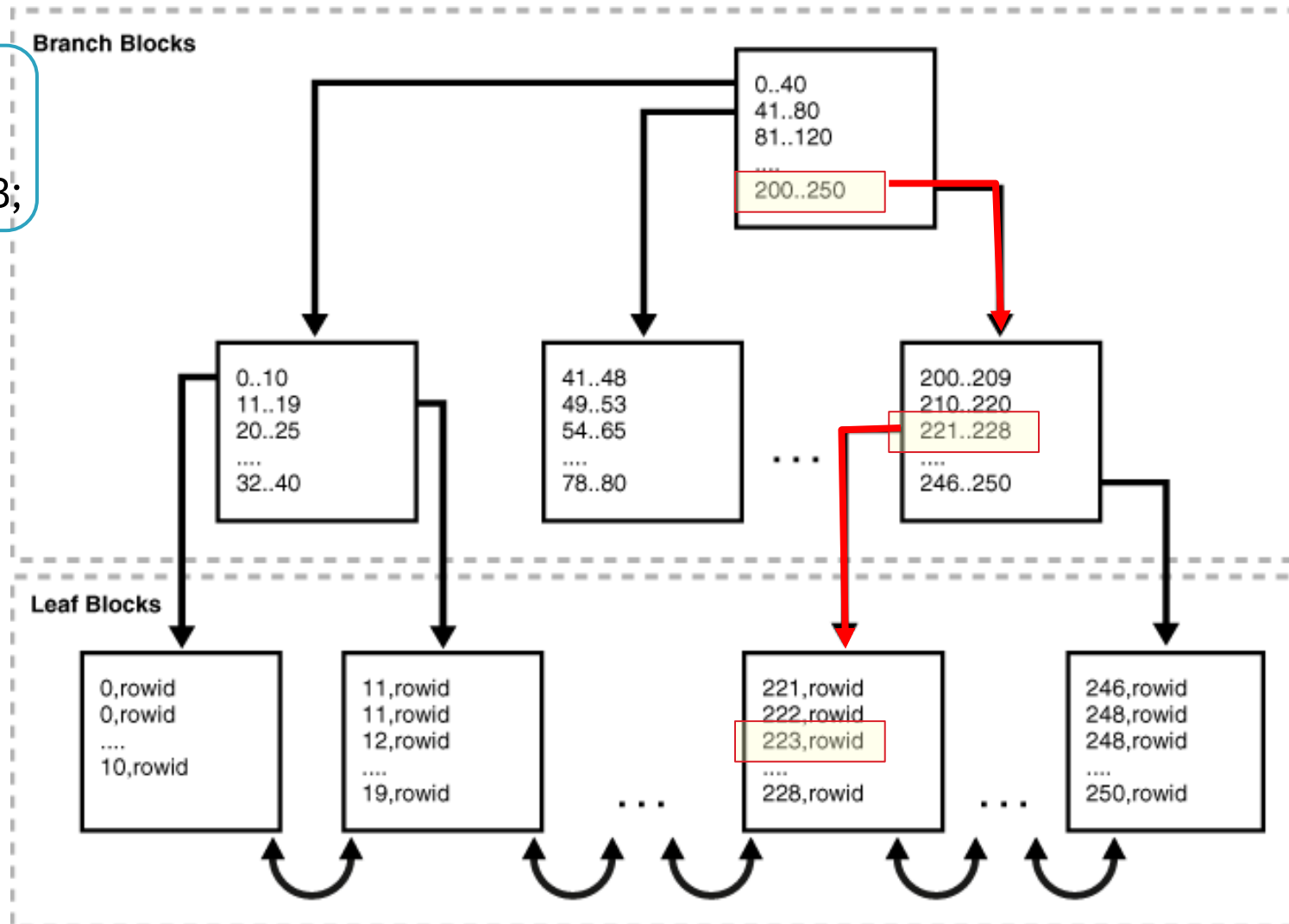
Ejemplo:
índice
sobre la
columna
dep en
EMPLEADO



Oracle. Índices en árbol B (*B-tree index*)

71

SELECT nombre
FROM empleado
WHERE dep = 223;



Oracle. Índices en árbol B (*B-tree index*)

72

- ¿Qué es el **rowid** en ORACLE?
 - ▣ Identificador único que Oracle asigna a cada fila de una tabla
 - ▣ Permite localizar la fila directamente en disco, sin búsquedas
- ¿Qué contiene el rowid?
 - ▣ Fichero de datos
 - ▣ Bloque dentro del fichero
 - ▣ Posición de la fila dentro del bloque
 - ▣ Ejemplo: AAAPzSAAEAAAABzAAA
 - Formato codificado (Base64) con info de ubicación física

Oracle. Eliminación de índices

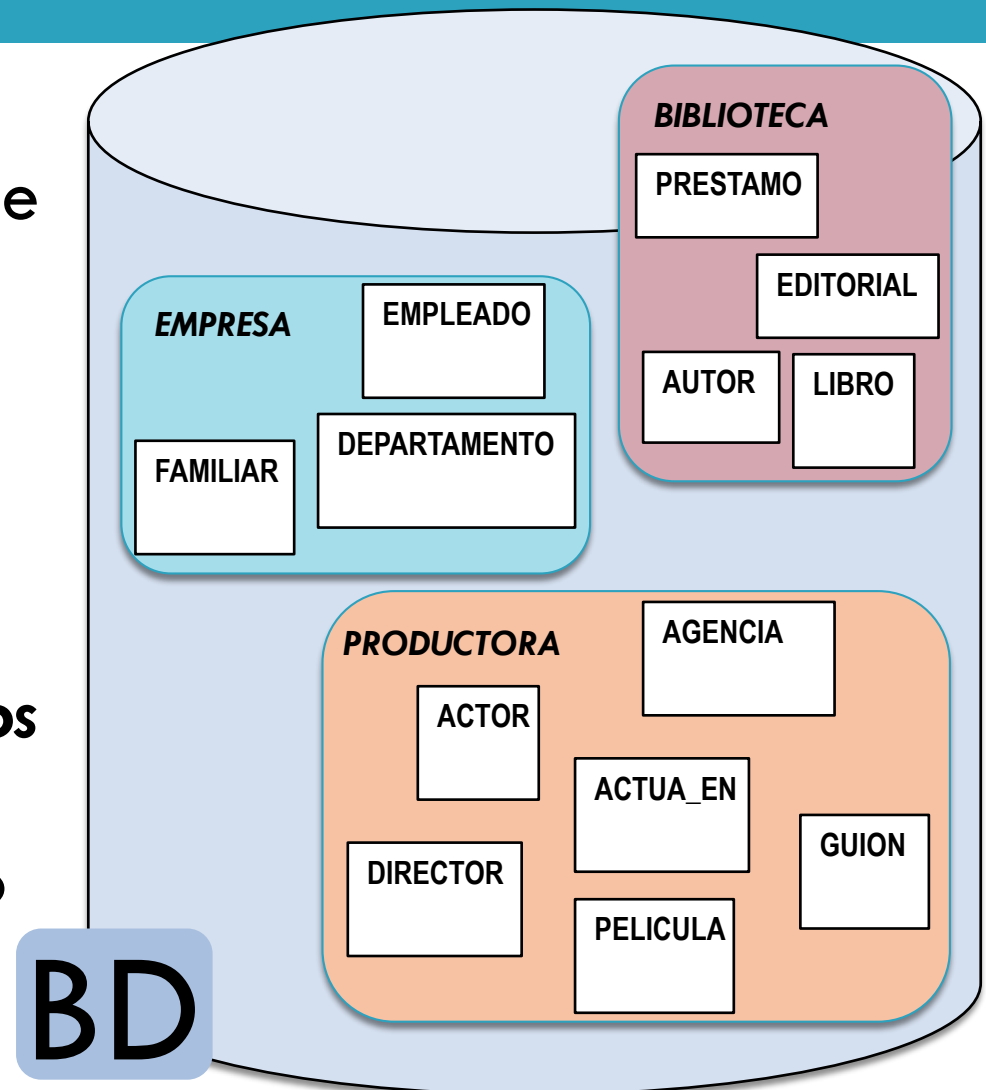
73

- Sentencia **DROP INDEX** *nombre_índice*;
- Razones para eliminar un índice
 - ▣ Ya no se espera realizar consultas basadas en su campo de indexación
 - ▣ No se usa o no acelera las consultas, porque la tabla...
 - es muy pequeña, o
 - contiene muchas filas, pero hay muy pocas entradas en el índice
- El espacio que ocupaba queda disponible y desaparece el coste de su mantenimiento (actualización permanente)
- Para eliminar un índice asociado a UNIQUE o a PRIMARY KEY hay que desactivar (DISABLE) o eliminar (DROP) la restricción

Esquema de Bases de Datos Relacional

74

- En general, una **base de datos** (física) puede contener **varios esquemas**
 - ▣ Ya vimos algo de esto en el Tema 1
- Un **esquema** *agrupa tablas y otros elementos relacionados de algún modo*: los de un mismo usuario, los de cierta aplicación, etc.



Definición o creación de esquemas

75

- Orden **CREATE SCHEMA**

CREATE SCHEMA *nombre_de_esquema*

AUTHORIZATION *identificador_de_autorización*

- ▣ *identificador_de_autorización* es el usuario/cuenta **propietario del esquema**

- *Ejemplo: crear el esquema Empresa*

CREATE SCHEMA empresa AUTHORIZATION eusebio;

- ▣ El usuario cuya cuenta es eusebio es el propietario del esquema EMPRESA

- Una vez creado el esquema, a continuación se puede especificar las definiciones de los *elementos* contenidos en él
- *Elementos* de un esquema:
 - ▣ Tablas, Vistas, Dominios, Permisos o Privilegios, Asertos, etc.

Creación de elementos dentro de un esquema

76

- Se puede indicar el esquema donde crear una tabla
 - ▣ Esquema **Explícito**
 - Un usuario puede crear una tabla en el esquema de otro usuario (si tiene permiso para ello)
CREATE TABLE **empresa**.empleado (
...);
 - ▣ Esquema **Implícito** en el contexto
 - Un usuario crea una tabla en el esquema activo en su cuenta
CREATE TABLE empleado (
...);
- En SQL de **Oracle**, la orden cambia un poco
CREATE SCHEMA AUTHORIZATION *cuenta*;
 - **No** se le puede dar **nombre** al esquema
 - **Sólo UN esquema por cuenta**: *cuenta* debe coincidir con el usuario conectado
 - Tiene poco o nulo efecto en la BD

Eliminación de esquemas

77

□ Orden **DROP SCHEMA**

- ▣ Destruye un esquema de BD, junto con su definición en el INFORMATION_SCHEMA del catálogo

DROP SCHEMA *nombre_esquema* *opción*;

opción puede ser...

- ▣ RESTRICT: Destruye el esquema sólo si no contiene elementos
- ▣ CASCADE: Elimina el esquema y además las tablas, dominios y demás elementos contenidos en dicho esquema

□ En SQL de **Oracle** no existe esta orden de destrucción de un esquema

- ▣ Se elimina un esquema cuando se elimina la cuenta de usuario

Catálogo de base de datos relacional

78

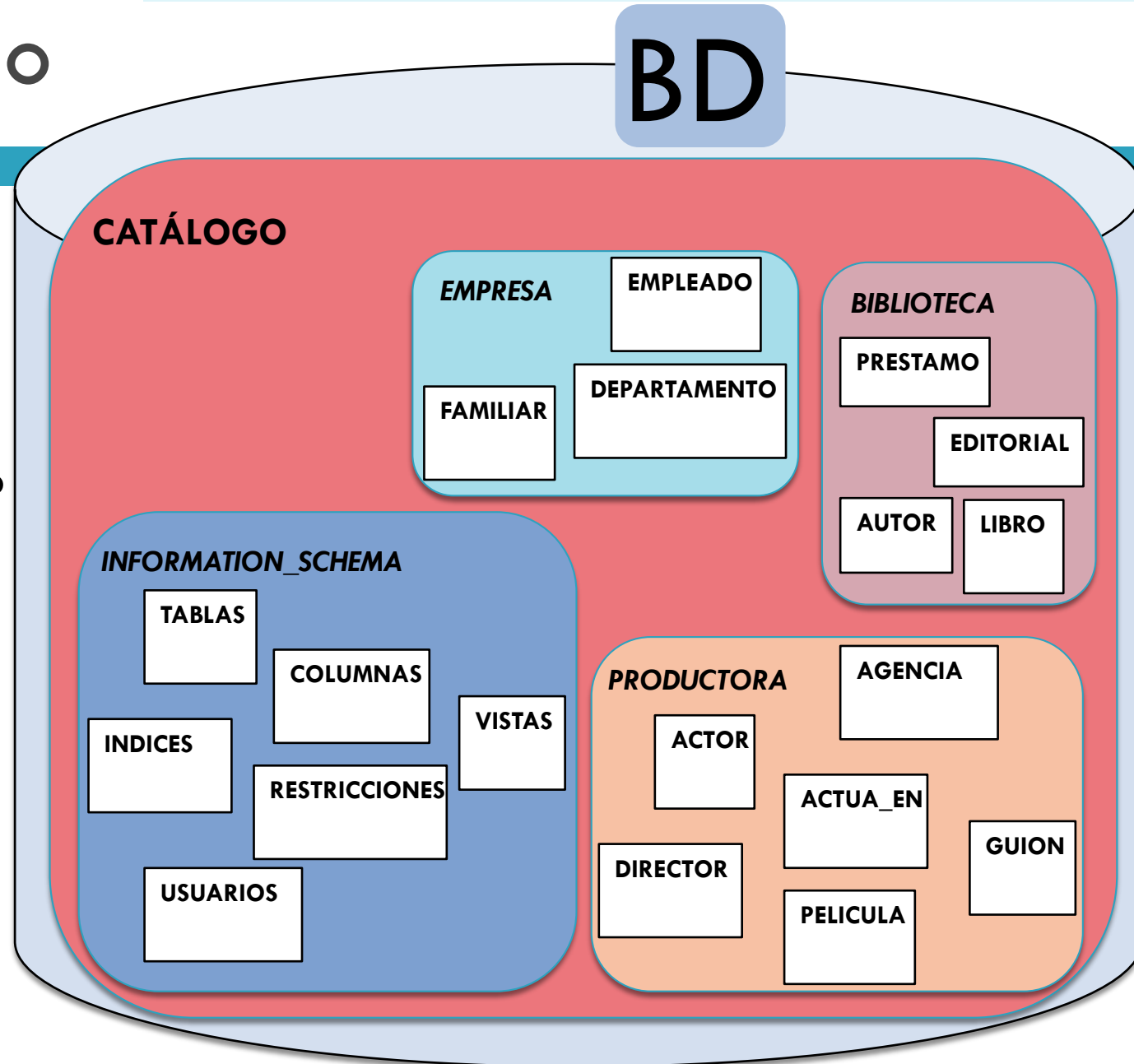
- ❑ Conjunto nombrado de **todos los esquemas** de BD existentes en un mismo entorno SQL
- ❑ Contiene un esquema especial, **INFORMATION_SCHEMA**, que almacena **metadatos: la definición de todos los elementos de todos los esquemas existentes** en el catálogo
 - ▣ Lo vimos en el tema 1
- ❑ En **Oracle**, el Diccionario de Datos (*Data Dictionary*) se corresponde con el INFORMATION_SCHEMA del estándar ANSI SQL

Catálogo

BD

79

- Puede definirse **restricciones de integridad referencial entre tablas** que existan en esquemas dentro **del mismo catálogo**
- Ejemplo. La tabla PRODUCTORA.GUION podría contener una clave ajena que hiciera referencia a la tabla BIBLIOTECA.LIBRO, para poder indicar la novela en la que están basados los guiones de películas



Catálogo de base de datos relacional

80

Terminamos recordando una gran ventaja de usar un SGBD

- La existencia de los metadatos proporciona a la base de datos una ***naturaleza autodescriptiva***

$$\text{BD} = \text{DATOS} + \text{METADATOS}$$

- Permite que el SGBD “sepa” acceder a datos de **cualquier** usuario y aplicación
 - ▣ SGBD = **Sistema software de propósito general**