Unidad Didáctica 2:
Uso de Bases de Datos Relacionales

Parte 2:
El Lenguaje SQL:
El lenguaje SQL: definición de datos (DDL)

U.D. 2.2

Bases de Datos y Sistemas de información
Departamento de Sistemas Informáticos y Computación / Universidad Politécnica de Valencia

Objetivos:

- Presentar la sintaxis del lenguaje de definición de datos de SQL.

- Ver algunos ejemplos sencillos para clarificar la semántica del SQL.

UD2.2 El lenguaje SQL: definición de datos (LDD)

1. Lenguaje de Definición de Datos (LDD)

El lenguaje de definición de datos es un subconjunto de instrucciones de SQL que permite

- crear,
- modificar y
- eliminar

componentes de las bases de datos

UD2.2 El lenguaje SQL: definición de datos (LDD)

2. Componentes de un esquema relacional

Los elementos de los que se van a presentar las instrucciones de definición son los siguientes:

- Relación o tabla.
- Vista
- Permiso

Toda la información, incluyendo:

- nombres de tablas,
- nombres de columnas y restricciones sobre columnas o tablas,
- nombres y definiciones de vistas,
- permisos, etc.

deben estar almacenados en tablas dentro de las bases de datos.

Las tablas que contienen tal información constituyen el Diccionario de datos (o catálogo).

UD2.2 El lenguaje SQL: definición de datos (LDD)

3. Definición de Relación o Tabla

Sintaxis de definición de tabla o relación:

CREATE TABLE *nom_tabla* (
elemento_tabla$_1$,
elemento_tabla$_2$,
...,
elemento_tabla$_n$
)

Donde un elemento_tabla es:

{ definición_atributo | restricción_tabla }

Definición de atributo:

nom_atributo tipo _dato
[DEFAULT {valor | NULL}]
[restricción_atributo$_1$ restricción_atributo$_2$ …
restricción_atributo$_n$]

Siendo los tipos de dato*:

{ VARCHAR [(n)] | VARCHAR2 [(n)] | CHAR [(n)]
| NUMBER [(n [,n])] | DATE }

* Dependen del sistema de gestión concreto

Ejemplo: CREATE TABLE equipo (
nomeq VARCHAR2(25),
director VARCHAR2(100));

Restricciones definidas sobre un solo atributo

[CONSTRAINT *nombre_restricción*]
{ NOT NULL
| UNIQUE
| PRIMARY KEY
| REFERENCES *nom_relación* [(*nom_atributo*)]
[MATCH { FULL | PARTIAL | SIMPLE }]
[directriz_borrado]
[directriz_actualización]
| CHECK (condición_búsqueda) }
[cuándo_comprobar]

Restricción_tabla (restricciones sobre más de un atributo)

[CONSTRAINT *nombre_restricción*]
{ UNIQUE (*nom_atributo*$_1$, *nom_atributo*$_2$, ..., *nom_atributo*$_n$)
| PRIMARY KEY (*nom_atributo*$_1$, *nom_atributo*$_2$, ..., *nom_atributo*$_n$)
| FOREIGN KEY (*nom_atributo*$_1$, *nom_atributo*$_2$, ..., *nom_atributo*$_n$)
REFERENCES *nom_tabla*
[(*nom_atributo*$_1$, *nom_atributo*$_2$, ..., *nom_atributo*$_n$)]
[MATCH {FULL | PARTIAL | SIMPLE}]
[directriz_borrado]
[directriz_actualización]
| CHECK (condición_búsqueda) }
[cuándo_comprobar]

Ejemplo: CREATE TABLE equipo (
nomeq VARCHAR(25),
director VARCHAR(100)) NOT NULL,
CONSTRAINT PK_equi PRIMARY KEY (nomeq));

La directriz_borrado es:

```
ON DELETE { CASCADE
          | SET NULL
          | SET DEFAULT
          | NO ACTION }
```

Y la directriz_actualización es:

```
ON UPDATE { CASCADE
          | SET NULL
          | SET DEFAULT
          | NO ACTION }
```

Ejemplo

Puerto (nompuerto: d_nom, altura: d_alt, categoria: d_cat,
pendiente: d_pen, netapa: d_no, dorsal: d_dor)

VNN: {netapa}

CP: {nompuerto}

CAj: {netapa} $\rightarrow$ ETAPA

CAj: {dorsal} $\rightarrow$ CICLISTA

Ejemplo

```
CREATE TABLE Puerto (
 nompuerto VARCHAR2 (35)
          CONSTRAINT PK_puerto PRIMARY KEY,
 altura NUMBER(4),
 categoria CHAR(1),
 pendiente NUMBER(3,2),
 netapa NUMBER(2) NOT NULL
          CONSTRAINT FK_puerto_eta  REFERENCES etapa (netapa),
 dorsal NUMBER(3)
          CONSTRAINT FK_puerto_cicli   REFERENCES ciclista (dorsal)
);
```

Ejemplo

Llevar (dorsal: d_dor, netapa: d_no, codigo: d_cod)

CP: {netapa, codigo}

VNN: {dorsal}

CAj: {netapa} $\rightarrow$ ETAPA

CAj: {dorsal} $\rightarrow$ CICLISTA

CAj: {codigo} $\rightarrow$ MAILLOT

Ejemplo

```
CREATE TABLE Llevar  (
 dorsal NUMBER(3) NOT NULL
        CONSTRAINT FK_llevar_cicli
        REFERENCES ciclista (dorsal),
 netapa NUMBER(2)
        CONSTRAINT FK_llevar_etapa REFERENCES etapa (netapa),
 codigo CHAR(3)
        CONSTRAINT FK_llevar_mai REFERENCES maillot (codigo),
 CONSTRAINT PK_lle PRIMARY KEY (netapa, codigo)
);
```

Cuándo_comprobar

[[NOT] DEFERRABLE] [INITIALLY {IMMEDIATE | DEFERRED}]

propiedad **estado**

- Por defecto es *NOT DEFERRABLE INITIALLY INMMEDIATE*
- La combinación *NOT DEFERRABLE INITIALLY DEFERRED* no está permitida.

Evaluación:

- Inmediata (INMEDIATE): después de cada operación de actualización.
- Diferida (DEFERRED): Cuando se finalice la transacción.

Para cambiar dinámicamente dentro de una transacción el estado de las restricciones diferibles se usa la siguiente instrucción:

SET CONSTRAINT

{ nombre_restricción$_1$, nombre_restricción$_2$, …, nombre_restricción$_n$ | ALL }

{ IMMEDIATE | DEFERRED }

4. Modificación de la definición de relación o tabla

La modificación del esquema de una relación

```
ALTER TABLE nombre_tabla
      { ADD (definición_atributo)
      | MODIFY [COLUMN ] (nombre_atributo)
                { DROP DEFAULT |
                 SET DEFAULT {literal | funcion_sistema | NULL} |
                 ADD definicion_restriccion |
                 DROP nombre_restriccion }
       | DROP [COLUMN ]  nombre_atributo
                 {RESTRICT | CASCADE}}
```

Ejemplo:

ALTER TABLE ciclista ADD (estatura NUMBER(3))

5. Eliminación de una relación o tabla

La eliminación del esquema de una relación tiene la sintaxis siguiente:

DROP TABLE *nom_relación* { RESTRICT | CASCADE }

6. Definición de vistas

CREATE VIEW *nombre_vista*

[(*nombre_atributo*$_1$, *nombre_atributo*$_2$, *nombre_atributo*$_n$)]

AS sentencia_SELECT

[WITH CHECK OPTION]

Impide actualización sobre la vista si viola su definición

Ejemplo:

Se van a hacer consultas frecuentes sobre las etapas que tienen puertos de montaña.

```
CREATE VIEW Etapas_con_puertos AS
        SELECT *
        FROM Etapa
        WHERE netapa IN (SELECT netapa FROM Puerto);
```

Se puede hacer una consulta utilizando la vista:
Obtener la longitud máxima de las etapas que tienen puertos de montaña

```
SELECT MAX(km)
FROM Etapas_con_puertos;
```

Se usa la vista

7. Borrado de vistas

DROP VIEW *nombre_vista* {RESTRICT | CASCADE}

8. Operaciones sobre vistas

Se pueden aplicar las operaciones de inserción, borrado y modificación a las vistas.

Cualquier operación sobre vistas debe cumplir las restricciones que estén definidas sobre las relaciones básicas que intervienen en la definición.

En los sistemas de gestión comerciales están limitadas, permitiéndose sólo modificaciones o inserciones cuando en la definición de la vista no intervienen funciones agregadas, ni operadores conjuntistas, ni la cláusula DISTINCT.

Ejercicio

COMPAÑIA (*comp_id:* texto, *nombre*: texto, *teléfono*: texto, *franquicia*: entero)

CP: {*comp_id*}

VNN: {*nombre*}

DOCTOR (*dni*: entero, *nombre*: texto, *apellidos*: texto, *teléfono*: texto, *especialidad*: texto)

CP: {*dni*}

PACIENTE (*dni*: entero, *nombre*: texto, *apellidos*: texto, *año_nacimiento*: entero, *dirección*: texto, *teléfono*: texto, *compa*: texto)

CP: {*dni*}

VNN: {*apellidos*}

CAj: {*compa*} □ COMPAÑÍA f(*compa*)= *comp_id*

OPERACION (*op_id*: entero, *doctor*: entero, *paciente*: entero, *descripción*: texto, *fecha*: fecha, *quirófano*: texto, *coste*: entero)

CP: {*op_id*}

CAj: {*doctor*} □ DOCTOR f(*doctor*)= *dni*

CAj: {*paciente*} □ PACIENTE f(*paciente*)= *dni*

```
CREATE TABLE Compañia (
        comp_id VARCHAR(10) PRIMARY KEY,
        nombre VARCHAR(30)  NOT NULL,
        telefono VARCHAR(8),
        franquicia NUMBER(2)
);

CREATE TABLE Doctor (
        dni NUMBER(8) PRIMARY KEY,
        nombre VARCHAR(30),
        apellidos VARCHAR(50) NOT NULL,
        telefono VARCHAR(8),
        especialidad VARCHAR(10)
);
```

```
CREATE TABLE Paciente (
        dni NUMBER(8) PRIMARY KEY,
        nombre VARCHAR(30),
        apellidos VARCHAR(50) NOT NULL,
        año_nacimiento NUMBER(4),
        direccion VARCHAR(100) ,
        telefono VARCHAR(8),
        compa VARCHAR(10),
        CONSTRAINT fk_compa
        FOREIGN KEY (compa) REFERENCES Compañia (comp_id)
 );
```

```
CREATE TABLE Operacion (
        op_id NUMBER(8) PRIMARY KEY,
        doctor NUMBER(8),
        paciente NUMBER(8),
        descripción VARCHAR(100),
        fecha DATE,
        quirofano VARCHAR(10),
        coste NUMBER(6),
        CONSTRAINT fk_esdoctor
                FOREIGN KEY (doctor) REFERENCES Doctor (dni),
        CONSTRAINT fk_espaciente
                FOREIGN KEY (paciente) REFERENCES Paciente (dni)
```

9. Gestión de autorizaciones

definición_privilegio::= GRANT
{ ALL |
SELECT |
INSERT [(*nom_atr*$_1$,…, *nom_atr*$_n$)] |
DELETE |
UPDATE [(*nom_atr*1,…, *nom_atr*$_m$)]
}
ON *nom_relación*
TO {*usuario*1,…, *usuario*$_p$ | PUBLIC}
[WITH GRANT OPTION]

Permite otorgar los privilegios a otros usuarios

Para eliminar los privilegios se usa la sentencia **REVOKE**

Disparadores (triggers)

Los **triggers** (disparadores) introducen el concepto de reactividad, y tienen muchas otras aplicaciones aparte de la integridad:

- Mantenimiento de información derivada.
- Implementación de reglas de la organización.
- Administración de bases de datos (backups, avisos, etc.) y aspectos relacionados con seguridad (trazabilidad, registros, …)

Disparadores (triggers)

Generalmente, las restricciones generales se implementan con:

- CREATE TRIGGER:
  Al indicar qué operaciones hay que controlar, se pueden definir muchos triggers en el sistema con una sobrecarga sobre el mismo menor

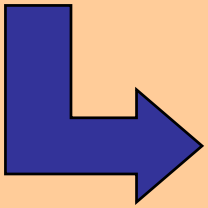
Y no con

- CREATE ASSERTION:
  Si el SGBD lo permite, puede tener un uso puntual, porque si se generaliza el sistema se enlentece muy significativamente al tener que ejecutar cada assertion para cualquier actualización de la base de datos.

Disparadores (triggers)

Los disparadores permiten modelar un comportamiento activo (autónomo) del sistema (SGBD) como respuesta a la ocurrencia de ciertos sucesos o condiciones

Disparador ≡ Regla de actividad



Evento – Condición – Acción

Disparadores (triggers)

Regla de actividad

Evento – Condición – Acción

- Evento: Especifica el suceso a la ocurrencia del cual ha de responder el sistema
- Condición: Especifica el contexto en el cual la regla, l'evento de la cual se ha producidp, ha de ejecutarse
- Acción: Especifica las acciones a ejecutar por el sistema como respuesta a la ocurrencia del evento cuando la condición es cierta